Automatic GUI Code Generation with Deep Learning

Xulu Yao

A thesis submitted in partial fulfilment of the requirements of
Manchester Metropolitan University
for the degree of Doctor of Philosophy

Department of Computing and Mathematics

2021

Abstract

Manually converting the design of a graphical user interface (GUI) into code is a time-consuming and error-prone process. A feasible solution is to automatically generate code by designing images or text through a GUI. Recently, deep learning technology has shown promising results in detecting GUI elements for this automation.

This project develops new approaches to GUI development and evaluation. First, a code semantic metric (CSM) is developed. It uses n-gram sequence features and cosine similarity to judge the accuracy of translated code. The results show that this metric has better performance than bilingual evaluation understudy (BLEU). Second, a modified framework is proposed to solve the problem of feature vector losses in a pix2code model, which generates the specific GUI code with a screenshot. The results of the empirical study outperform the state-of-the-art methods based on BLEU. Third, a UIGAN model that performs better than the traditional generative adversarial network (GAN) is proposed, and a new data augmentation method is introduced to overcome data deficiency in GUI generation. Fourthly, to address the problem of existing text-to-image generation models, a scene graph-to-UI (SG2UI) model is proposed for GUI generation. In this approach, a graph convolutional network (GCN) is used as the feature extraction network of the input scene graph. The Fréchet inception distance (FID) and perceptual loss are used to calculate the difference between the generated GUI and the real GUI. The experimental results demonstrate that the object details of the final GUI generated are more apparent and the model improves the quality and creativity of the generated GUI.

Future research is needed to improve the model to directly generate complex scene layouts with the hypertext nature of GUI design.

Acknowledgements

I am greatly indebted to my supervisors, Dr. Yanlong Zhang and Prof. Moi Hoon Yap. I wish to gratefully acknowledge their contributions for the completion of my PhD thesis. They have devoted invaluable guidance, advice and time to the project.

I wish to thankfully acknowledge help and support from all colleagues in the Department of Computing and Mathematics, Manchester Metropolitan University, in providing respectively support for my study. Furthermore, I will thank all the colleagues and students who carefully completed my questionnaires and attended interviews.

Lastly, my family, my mother and my father are sources of encouragement, and part of my success.

Contents

List of Figures

Chapter 1

Introduction and Overview

1.1 Background

A graphical user interface (GUI) is an interface through which users interact with electronic devices such as computers, handheld devices and other devices [171]. Text-based interfaces display data and commands in text format, whereas GUIs use icons, menus, and other visual graphics to represent information and related user controls to be displayed. It is widely used in online websites, modern desktop software and mobile applications. As a necessary part of using electronic information products, a GUI realises the information interaction between people and software. This human–computer interaction makes the user's operation more convenient [82].

For the ever-changing electronic products, the GUI is becoming increasingly important [186]. A beautiful and friendly interface design is often more appealing to customers and has become the key for enterprises to obtain a competitive advantage. For example, many computer users believe that Apple's Macintosh system has a better GUI than Microsoft's Windows system, which improves their brand loyalty to Apple [186].

A successful GUI design may require complex and time-consuming processes. The design process must follow many design principles and rules, including smooth interaction and applicability [48]. Clear readability and aesthetics are also key elements in the design. Therefore, GUI designers must innovate through design and adapt to the latest applications/software to keep up with fashion trends and meet the changing market needs [28].

This kind of work is usually left to a few designers, so software developers also need to undertake design tasks to fill this gap most of the time [69]. A survey of more than 5,700 developers found that 51% of developers said they performed application GUI design tasks more frequently than other tasks [62]. Software developers often

lack UI/UX design training and have limited artistic sensitivity. Developers strive to create GUIs from scratch. Instead, developers usually search the internet for existing GUI designs and then modify them to meet their needs [148]. This is usually done in open-source software projects or small start-ups without professional UI/UX designers.

Although the existing studies help retrieve GUI and analyse its code structure, some substantive problems have not been solved [15, 12]. First, there is no connection between the developer's intention and the problem of outputting text. There is also a gap between visual GUI design and text queries, which may lead to the retrieved GUI not meeting the developers' needs. Second, GUI design can be copied by other developers, which can negatively affect the application's uniqueness and originality. Using other GUIs directly may also cause potential intellectual property problems. Third, some GUI design styles may be outdated, and developers may be unable to keep up with current trends. Hence, an automated method must be built for creative GUI design to reduce the workload of novice developers and designers [23, 14, 122]. Developers can use generated GUI design to automatically generate GUI code and simplify the entire GUI development process.

1.2 Aims and Objectives

The aim of this project is to develop a system that can automatically generate a specific platform code for a given GUI screenshot or GUI elements as inputs. The extended version of this method might reduce the need to manually program GUIs.

The objectives of this thesis are:

- Design a novel metric to evaluate the accuracy for GUI layouts.
- Propose an improved framework to solve the problem of feature vector losses in a pix2code framework to a certain extent.
- Synthesise realistic GUI images using generative adversarial networks.
- Create a scene graph model to generate additional GUI examples such as actual texts, images, and buttons.

1.3 Main Contributions

The main contributions of this thesis are as follow:

- This study has designed a code semantic metric (CSM) using n-gram sequence features and cosine similarity to measure the accuracy of translation code.
- An improved framework based on pix2code is developed to automatically generate a specific platform code for a given GUI screenshot as an input.
- The UIGAN is developed for GUI data augmentation.
- A scene graph-to-UI (SG2UI) generation model based on a graph attention network is proposed to generate higher quality GUI layouts.

1.4 Organisation of Thesis

1.4.1 Chapter 2: Literature Review

Deep learning techniques are being widely adopted to classify and recognise GUI components. This chapter studies the relevant literature for deep learning techniques used to recognize GUI components. It provides a review of the techniques and approaches used in deep learning and computer vision to classify components.

1.4.2 Chapter 3: Evaluating Semantic Similarity for Source Code Translation

Statistical machine translation (SMT) is a research hotspot in machine translation and natural language processing. Recently, source code translation tasks based on the SMT model have been applied to software engineering. Unfortunately, there is no automated metric that can effectively detect the accuracy of code translation. Considering the similarity between code similarity detection and the machine translation scoring process, this chapter proposes CSM based on traditional code plagiarism detection metrics to verify its applicability to code translation tasks. Our empirical research shows that the results of different code plagiarism detection methods are quite different. After a specific parameter adjustment, the CSM can reflect the correctness of translation code semantics to a certain extent. We confirm that the CSM has a high correlation with human judgment in the semantic accuracy of translated code, and surpasses the scores of MOSS and JPlag, the mainstream traditional code plagiarism detection methods.

1.4.3 Chapter 4: Automatic GUI Generation with Domain-Specific Language (DSL) Model

Over the past few years, various studies have been conducted to solve the problems of automatically converting image models into source code. Since 2018, pix2code has inspired and promoted research in this domain. This chapter presents a new model architecture to improve the framework of pix2code. We designed a framework that can automatically generate a specific platform code for a given GUI screenshot as an input. Although bilingual evaluation understudy (BLEU) is natural language processing metric, it has been adopted for source code evaluation. To overcome the limitations of BLEU in DSL tokens evaluation, we introduced a modified BLEU (MBLEU) score. Our results show our proposed frameworks outperform the state-of-the-art methods in BLEU and MBLEU. The MBLEU is suitable for DSL similarity evaluation, but further research is necessary to establish this new metric.

1.4.4 Chapter 5: Data Augmentation on Graphical User Interface Generation

As a branch of artificial neural networks, deep learning is widely used in the field of image recognition but the lack of its datasets leads to imperfect model learning. By analysing the data scale requirements of deep learning and aiming at its application in GUI generation, it is found that the collection of GUI datasets is a time-consuming and laborious project, and it is difficult to meet the needs of existing deep learning networks. To solve this problem, this chapter proposes a semi-supervised deep learning model that relies on the original small-scale datasets to produce a large number of reliable data sets. By combining the cyclic neural network with the generated countermeasure network, the cyclic neural network can learn the sequence relationship and characteristics of the data, make the generated countermeasure network produce reasonable data, and then expand the Rico dataset. Relying on the network structure, the characteristics of collected data can be thoroughly analysed, and a large quantity of reasonable data can be generated according to these characteristics. After data processing, a reliable dataset for model training can be formed, which resolves the problem of dataset shortages in deep learning.

1.4.5 Chapter 6: Scene Graph-to-UI (SG2UI) Model for Graphical User Interface Layout Generation

Presently, the text-to-image generation model only performs well on scene image datasets with a single object. When a scene image involves multiple objects and relationships, the generated layout becomes chaotic. The best existing solution is to convert the text description into a scene graph structure that can better represent the scene relationship in the layout, and then use the scene graph to generate the GUI design. However, the final GUI generated by the existing scene graph to the GUI layout generation model is not clear enough, and the object details are insufficient. Therefore, a scene graph-to-UI generation model based on a graph attention network is proposed to generate higher quality GUI layouts.

The model consists of a graph attention network for extracting scene graph features and an object layout network for synthesising scene layout. The graph attention network transfers the output object feature vector with stronger expression ability to the improved object layout network to synthesise the scene layout closer to the real label. At the same time, a feature-matching method is proposed to calculate the layout loss, which makes the final generated layout more similar to the real GUI in semantics. By training the model to generate 64 *times* 64 pixel images in the Rico dataset containing multiple objects, the model in this chapter can generate complex scene layouts containing multiple objects and relationships. The Frechet Inception Distance (FID) of the generated GUI is about 8.8, which is 0.5 higher than the original scene image-to-image generation model. The SG2UI generation model based on the graph attention network proposed in this chapter can not only generate complex GUI layouts containing multiple objects and relationships, but also produce GUIs with higher quality and clearer details.

1.4.6 Chapter 7: Conclusion

This chapter concludes the thesis with a summary of the findings and a discussion of possibilities for further research.

Chapter 2

Literature Review

2.1 Introduction

Deep learning techniques are being widely adopted in classifying and recognizing Graphical user interface components. This chapter studies the relevant literature and extracts the context for deep learning techniques to recognize GUI components. It provides a review of techniques and approaches used by deep learning and computer vision to classify components. This chapter is the literature review of the topic and provides data from valid and relevant sources for the comprehensive study of the subject.

2.2 Software Automation

Software automation mainly involves software development, software specification, automatic generation and automatic verification [17]. Since the beginning of the 21st century, the global influence of the information industry has been increasing, and software systems have gradually penetrated into various industrial fields and promoted the continuous development and progress of those fields. At the same time, the scale and complexity of software systems have increased dramatically. However, human errors inevitably exist in the process of software development, resulting in defects in software design or implementation, making the behaviour of software systems more and more difficult to predict and control during operation [145]. Once the software is run with specific input parameters or execution flow, defects can be activated, resulting in software failure. In addition to causing a lot of property damage, this can also result in the loss of life. To improve software quality and minimise the risk of software failure, it is necessary to take appropriate control measures during the software development process. Moreover, there are a large number of feedback

mechanisms in the process of software design, development, testing, operation and maintenance that provide sufficient space for the use of control theory to solve software engineering problems [104]. Thus, software automation uses the corresponding theory to rely on software to achieve specific functions.

With the development of the modern social information network, software testing has become a valuable part of software engineering and it occupies an increasingly important position in the software development process [141]. Software testing is used not only to find errors in the system, but to also test the software system by applying various testing techniques and methods, which can effectively improve the quality of the software system and enhance the testers' perception of confidence in the product's quality. Although the errors existing in the software cannot be fully predicted, after software testing, the possibility of software failures and the severity of the consequences of the failures can be accurately determined [113]. Through software testing, the probability of software errors in the system can be limited to an acceptable range, thus greatly improving the reliability of software quality.

The traditional software testing method primarily uses manual testing, which requires a significant allocation of human resources, a long testing cycle and low testing efficiency; moreover, it is very dependent on the tester's personal experience, which is easily affected by personal thinking habits, resulting in poor testing work, omissions and errors [65]. Due to the gradual reduction of software development time and the gradual expansion of software development scale, there is an increasing number of problems in software testing, and the use of software testing automation technology has become an inevitable trend in software development. Software test automation technology can quickly and thoroughly test software systems and eliminate the test omissions and errors caused by the testers' personal thinking habits, thereby effectively improving software quality, saving a significant amount of money related to human resources and development costs and reducing software development time [6].

Currently, software automation testing focuses on managing the automation of the software testing process and the automation of dynamic testing, such as performance testing automation and functional testing automation [42]. Test automation consists of the following components: the automated process of testing and the automated analysis of test results [191]. In the automated process of testing, the tester does not need to use the test cases one-by-one to test the software. The automatic analysis of test results means that the tester does not need to analyse and record the test data process and the intermediate results of the test. If there is an error in the test

process, the automated test tool automatically reports the error and provides some important clues, so the problems that occur during the test can be quickly identified.

Automated testing mainly evaluates the application set to be tested through test scripts, which are provided by automated testing tools [4]. Test scripts are codes written and implemented in a specified language (such as C#) in a specific environment. The software testing system method is different; it can be regarded as a text that uses specific language to parse the test function, or it can be regarded as a simple 'batch' command. It can also be regarded as a powerful function with more complex scripting language program fragments.

2.3 Human–computer interaction (HCI)

Human–computer interaction (HCI) is a multidisciplinary field of study that focuses on interaction modes between humans and machines and is sometimes also used in manufacturing or process control systems [161]. The more general term 'human–machine interface' (HMI) refers to the interface's manufacturing or process control systems. In other words, the HCI discipline is concerned with all issues related to the design and implementation of interfaces between humans and computers. Due to its nature and goals, HCI will naturally involve multiple disciplines of computer science (image processing, computer vision, programming languages, etc.) and multiple disciplines of humanities (ergonomics, human factors, cognitive psychology, etc.) [121]. HCI research primarily deals with the design, implementation, and evaluation of novel interfaces that can improve HCI [121]. The improvements here involve multiple aspects, including intuitive use and interface robustness.

An intuitive, natural, efficient, robust, and customizable interface can significantly bridge the gap between human mental models and computers, machines, or robots accomplishing a given task [121]. While HCI research dates back to 1975, recent advances in consumer electronics have opened exciting new horizons. Designing affordable, natural user interfaces (NUIs), gestures, hand and body gestures, speech, and gaze are just a few examples of the many natural interaction modes.

In the early days of computer science, designers and developers paid little attention to the usability or 'user friendliness' of hardware and software products [114]. However, as more users expect devices to be easy to use, researchers are finally focusing on usability.

The International Organisation for Standardisation (ISO) defines usability as the degree to which a product can be used effectively, efficiently, and satisfactorily by a

user to achieve a specific goal [16]. That is, usability defines a set of criteria, including efficiency, safety, and practicality, primarily related to computer systems.

In the mid-1990s, another important concept related to usability emerged. User experience (UX) focuses on user-related factors, such as satisfaction, liking, emotional satisfaction, and aesthetic appeal [5]. In some areas, the UX concept has expanded and become more explicit. For example, web interface designers often use the UX honeycomb to determine priorities during the design phase. The honeycomb-shaped seven hexagons represent the parameters that must be carefully balanced for a satisfactory quality of experience (QoE) for users: useful, usable, needed, visible, convenient, reliable, and valuable [159].

Understanding human mental models is another important HCI issue. Different users learn and maintain knowledge and skills in unique ways, often influenced by age as well as cultural and social backgrounds. Therefore, the purpose of HCI research is to bridge the gap between users and new technologies (now changing faster than ever) [64]. Effective, efficient, and natural forms of human–machine interaction can lessen the required skill level to operate complex equipment, thereby potentially reducing inequalities between people, which in turn helps to address the 'digital divide'—the ability to access the gap between those who are able to use ICT technologies and those who are unexposed and lack relevant skills [121].

For years, humans have been giving orders to machines through a keyboard and mouse, also known as WIMP (windows, icons, menus, pointing devices) [67]. In addition to the point-and-click devices normally associated with computers, we use a variety of keyboards, such as dialling phone numbers, interacting with a TV, and selecting the intricate functions on a car's dashboard. In most cases, the device feeds back its output to the user through a significant means, such as a monitor.

A few affordable sensors are also starting to shake up the way people interact with devices. Touch and multi touch screens have driven the transition from regular cell phones to smartphones, and gestures have become the primary interaction mode for activating functions on personal devices [73]. At the same time, speech recognition technology and the increasing computing power of CPUs also allow users to type effectively when gestures are inconvenient.

Personal devices are the clearest example of new HCI forms that can reduce the gap between human mental models and technology. One market leading this huge innovation in HCI is entertainment. As users look to game and device makers for new ways to control characters, console developers have devised a variety of new controllers to free players from keyboards and mice [34]. The new interface offers haptic feedback

and some form of tactile interface (controllers become steering wheels, guns, tennis racquets, etc.).

Sensors—such as the Microsoft Kinect—are a major step towards an all-natural interface, where the human body itself is the controller [45]. The device allows users to send instructions to the machine through gestures and body postures, and the embedded background hardware processes the raw data gathered by the depth camera in real time, thereby obtaining the skeleton's basic shape composed of human bones and joints. Recognizing the position and orientation of the bones, the hardware can recognize the posture and gestures and then map them into machine instructions.

Researchers are also working on sensors that can track a user's hands [79]. For example, Leap Motion interactively tracks a user's hands by identifying fingertip positions and palm centres and then utilises inverse kinematics to solve for knuckle positions. Some automakers have developed interactive solutions based on hand tracking as an alternative to the traditional model of managing infotainment functions via touchscreens. Likewise, some smart TVs can replace traditional remote controls by allowing users to employ gestures to make selections.

The above scenarios only existed in science fiction movies a few years ago but have become an HCI reality. On the other hand, new and more interesting scenarios will develop soon; for example, brain interfaces seem to be poised to reverse the relationship between humans and machines [118]. The success of new interaction models depends on future technological advancements that aim to transform interface devices into wearable embedded objects. Interfaces based on augmented reality (AR) technology are clear examples of this shift. Travel, entertainment, repair, shopping, and social networking applications for personal devices are emerging, and new wearable sensors may soon change our habits [95]. Google Glass will hit the market shortly, with new application areas emerging every day. The concepts of HCI and human–computer integration are destined to become one. In fact, solutions like Google Glass may soon be replaced by contact lenses to truly achieve a natural, eye-mounted interface.

New HCI forms will dramatically change our lives. Novel interaction paradigms will improve the quality of life for those who have difficulty enjoying today's interfaces, such as those with physical disabilities [51]. Conversely, new issues will arise—especially regarding privacy, security, and ethics—that could slow the rollout of wearable hardware and software. While some researchers are already working on interface design and legal and privacy-related issues, countries' diverse legal systems are not ready for future HCI advancements.

2.3.1 Graphical User Interface (GUI)

A graphical user interface (GUI) is a type of user interface through which users interact with electronic devices via visual indicator representations [82]. It is an amalgamation of technologies and gadgets that provides users with a set of rules by which to navigate a computing environment and enables the interrelationship between elements of an electronic device. A GUI is an interaction program with photographic constituents for computer software, wherein different recipients who transmit statistical data and the actions taken by users are presented [151]. Apart from a GUI, there are three other computer interfaces, namely, a command line interface, a menu-driven interface and a touchscreen interface. GUIs and touchscreen interfaces are similar, but the latter are an advancement that is extensively used in today's world.

The advantages of a GUI are: 1) It renders computer operation more intuitive and consequently eases usage and learning [157]; 2) It normally provides a user with rapid visual feedback about the result of each action and enables multiple programs or activities to be simultaneously displayed; and 3) Interplay through intelligibility and control, officiousness and seamless user navigation. The best GUI is designed in such a way that anticipates users' needs. It captures and maintains awareness. The most significant and perhaps the most frequently used advantage of a GUI is multi-tasking, which enables computer users to interrelate with a computer and allows for the revision of a standard layout—a major quality of any software for user accessibility. As can be seen, a GUI is very advantageous and significant for the technological advancement of the world.

GUIs are quiet and easy to use upon startup because they can simply shuffle information between programs when different functions are used, as is the case with the 'copy' and 'paste' shortcuts and the 'drag-and-drop' function. The use of a GUI entails increased storage and can also be a preparatory ability. For modern users, the functions of a GUI can go beyond the advantages of injection lines that use the interface. The four main GUI attributes that were introduced are windows, icons, menus and the pointer, which constitute the WIMP (windows, icons, menus, pointer) system [134]. The WIMP system is the avenue through which a user and a visual input device interact: It enables users to manipulate a pointing device, most often the mouse, arranges information in windows and exhibits icons that are available for executing commands—all these functions are combined in the menu and the gesture that is accomplished for enabling signalling with a pointing device. A window manager expedites the interconnection among windows, applications and the

windowing system. The windowing system advances hardware-related actions, such as personalising a device, the graphics hardware and the positioning of the pointer. Advancements in GUIs and the development of the touchscreen interface eliminated the need to use pointing devices.

2.3.2 Component of Graphical User Interface

The GUI is underlain by two major languages: presentation language, which paves the way for a computer's depiction of a human's concerns, and action language that distinguishes between human and computer segments [50]. Together, these components transform the appearance of the GUI and the satisfaction with its expression (i.e. the interface). The key components of the GUI are the pointer, icons, windows, menus, scroll bar and intuitive input. Such components, which signify the assemblage of computer programs, should correspond with a computer's graphics for the easy creation and use of a computer program [158]. Familiar GUIs are Microsoft Windows, Mac OSX, Chrome OS, GNOME and Android.

Gosling developed a programming language that was initially called 'oak', after the oak sapling that stood outside his office [170]. This development was then succeeded by the green project, which was eventually rechristened 'Java', from a type of coffee produced in Indonesia. Gosling's aim was to build apparatus from essential machines and a programming language that includes C, which is similar to annotation but whose development involved more consideration of consistency and intelligibility than that devoted to C/C++. The term 'GUI' is used not only for Java but also in all programming languages that are the building blocks of such interfaces. As previously stated, a GUI assembles the graphical components through which a user can interrelate with pages and applications. Java's GUI incorporates components such as labels, text fields, text areas and buttons, among others. Its conceptual windowing toolkit can also incorporate receptacles that include the aforementioned components. A Java program should accommodate two principal issues that clear the way for manipulating the sub-components. The programs must be catalogued to incidents on the components, and the programs can serve as an instrument in an event, that is, the instructor that can be called when an occasion occurs.

Python, which Guido van Rossum invented, is the expound, the recipient of a device located, the soaring flush in the programming language with dynamic semantics. Python was developed in 1980 at the Centrum Wiskunde and Informatics (CWI) in the Netherlands as the descendant of the ABC programming language, which was created on the basis of SETL. SETL can manipulate anomalies and can be used in

conjunction with the Amoeba intervene complex. Python was launched in December 1989. It is very easy to learn, and its syntax emphasises explicitness, thereby reducing the cost of program preservation. Python is an extensive programming language applicable to many different projects, but it is used primarily for web development, artificial intelligence, machine learning, operating systems development, mobile application development and video game development. The language also features unique GUI frameworks, namely, PYQT, PYGUI and TKINTER, among others. The frameworks afford a Python user easy fabrication of GUI elements using gizmo, which is found in the TK toolkit. TK widgets can be used to establish buttons, menus, data fields and other elements that constitute Python gadgets. Python can provide numerous options for the creation of a GUI, and an excellent Python interface can be developed using the TK GUI toolkit that comes with the language. Developing a GUI using applications is an easy task in ML.

MATLAB was originated by mathematician and computer programmer Cleve Moler in consultation during the conducive algebraic programming era of 1967. Its GUI is made up of apps that enable point-and-click actions for controlling software and eliminate the need for users to master different languages and different types of orders to scurry the solicitation. MATLAB can also be used with different applications independent of a desktop or the websites of these applications. Users are allowed control over motif and creation in MATLAB, as these elements explain the formation and behaviours of different apps.

2.4 Computer Vision

Computer vision is the field of study to develop techniques to aid the computers in seeing and understanding the context of the images and videos [178]. It ensures that computers are compatible and work efficiently with images and videos. Computer vision is the most common and popular application of deep learning. It is an amalgamation of different disciplines, including computer science, physics, mathematics and psychology. Given the broadness of these disciplines, computer vision is considered related to artificial intelligence [178]. Some of the most effective and extensively used computer vision or deep learning techniques are 'image classification, object detection, object tracking, semantic segmentation, instance segmentation, and image reconstruction'.

2.4.1 Image Classification

Image classification is considered the primary domain of deep learning or computer vision. It is aimed at recognising and identifying the features of images as objects. It is also defined as the process of differentiating and categorising groups of pixels and vectors on an image or identifying classes of images. This process takes two forms: supervised classification and unsupervised classification. In machine learning or computer vision, image classification is a 'supervised learning problem' resolved by defining a set of target classes on an image and training a model through labelled examples of images to enable it to predict image classes or sets [147]. Playing a critical role in image classification are neural networks, among which the best-suited deep learning algorithms are convolutional neural networks (CNNs).

2.4.2 Object Detection

Object detection is the process of defining objects on an image, labelling them and creating boundaries for better recognition [190]. It is a methodology that enables a computer to locate and identify an object, image or video. This identification method is used to comprehensively label an object on an image and gather data about its location and other factors. A CNN is also an applicable and exceptionally suitable algorithm for object detection. Object detection differs from image classification in that it creates inbounding boxes over the objects detected on an image. For example, image classification labels only images containing a cat; even if there are two cats in a photo, the image will still be labelled 'cat'.

Conversely, object detection creates boxes and labels those individual boxes on an image [190]. There can be numerous boxes on an image or video. Object detection is very useful for applications involving webcams, security cameras, crowd counting, anomaly scanning and self-driving cars, among other uses. Deep learning approaches come with state-of-the-art methodologies for the recognition of objects. Because of these methodologies, deep learning techniques are commonly used to recognise GUI components. Object tracking is interlinked with other computer vision and deep learning approaches. The basic structure of object detection approaches underlain by deep learning consists of encoders and decoders [190]. An encoder takes an image and assesses it through layers and networks to analyse its features and differentiate it from other images. The image classified by the encoder is taken by a decoder, which then binds and labels the image. The output port of the encoder is attached to a regressor, which serves as the decoder. This basic model structure has its limitations. On a

multi-object image, for instance, a module requires more regressors. The extension of regressors for such an image is called a 'region proposal network', which is a more efficient module. The proposed locations of an object on an image show that the pixels of proposed predictions are analysed through classification networks [207].

2.4.3 Object Tracking

Object tracking can be referred to as the second stage of object detection. It involves detecting, labelling and tracking similar objects on an image or video [190]. This approach creates a unique and distinct ID for each individual object detected and is widely used in video tracking. It also helps in object counting and locating. Problem object tracking approaches use object detection once during the initial stages of detection. It is accurate and efficient enough to locate an object when it moves outside boundaries. An object-tracking algorithm is robust between occlusions and efficient enough to track objects during rapid motions and movements.

2.4.4 Semantic Segmentation

Semantic segmentation, also called image segmentation, is the process of segmenting objects on an image through the clustering of objects that belong to the same class [71]. In simple terms, it involves the partitioning of digital images into segments. The aim or purpose of semantic segmentation is to alter an image into a detectable and readable structure. Segmentation is achieved by using different colours to represent different classes of objects. Semantic segmentation can also be defined as the method of labelling every pixel of an image and classifying similar characteristics of image pixels in one class. Semantic segmentation is used in computer vision, deep learning and artificial intelligence. It is an advanced method of classification and detection. On an image, similar objects are classified as one segmentation: For example, the people in a picture make up one segment, the animals constitute another and the picture's background is a separate segment. The segmentation approach uses the K-means algorithm for the clustering of images. Segmentation is a highly complex task that gives rise to more understandable and comprehensive images. It is an essential part of computer vision and deep learning. For liberation, a dense pixel prediction by models is required.

2.4.5 Instance Segmentation

An expansion of semantic segmentation is instance segmentation, in which every model and object on an image is classified, segmented and labelled. In segmentation, for instance, every car on an image is a different object or belongs to different segments. Semantic segmentation groups cars on an image into a single category, but instance segmentation breaks down each car into individual categories represented with different colours. Instance segmentation uses more examples, approaches and algorithms than semantic segmentation.

2.4.6 Image Reconstruction

The restoration of old or destroyed images is termed image reconstruction, wherein models use current datasets to protect eroded or degraded parts of an image. It is a very advanced field that continuously evolves, with researchers studying this discipline and its applications. Image reconstruction is used in deep learning and artificial intelligence [181]. It also helps remove substantial noise from images and develops high-resolution images via CNNs in deep learning.

2.5 Deep Neural Networks

Deep learning is part of an inclusive ancestry of the machine learning procedure root on the artificial neural network with the depiction of learning [101]. It is also termed deep structured learning. Learning can be conducted, also be semi-conducted, or can be conducted. Deep learning builds deep neural systems, deep belief systems, deep reinforcement learning, recurrent neural systems, and convolution neural networks. They have been trying to pasture counting the computer vision, speech recognition, machine translation, bioinformatics, medical image analysis, drug design, a medium that survey and board game programs, where they have manufactured the answer similar to and, in some instances, outstanding skillful human production. Deep learning is a type of machine learning and artificial intelligence (AI) that emulates the path in which humans earn a definite type of information. Virtual assistants are cloud-based entreaties that realize natural language voice order and finish the work for the user [101]. It is widely being used in Chatbots, healthcare, engineering, music, etc. Chatbots are generated for the use of machine learning algorithms. The deep learning Chatbot can gain aggregate from the data and the human-to-human talk. Deep

learning is also used to create music based on the waving net and can be fructuous on the replica for the raw audio, and it was Invented by google deep mind.

Python conducts the group, with 57% of fact researchers and machine learning designers utilizing it and 33% categorizing it for evolution. Little admiration is given to all the development in the deep learning that Python chassis over the two years past, counting on the declaration of the Tensor flow and extensive selection of the other libraries [54]. The skills of deep learning to operate a large number of functions make deep learning very strong when dealing with amorphous data. Although, deep learning algorithms can be a belly full for less complex problems because they need access to a huge amount of the data to be effective. Deep learning has become an integral and vital part of machine learning. The most beneficial or significant advantage of deep learning is that deep learning is the machine language technique that guides the computer to do what comes consistently to humans [101].

The history of deep learning was from 1943 when Walter Pitts and Warren McCulloch produced a computer model based on the visual matrix of the human mind. They used a union of algorithms and the mathematics they knew as 'threshold logic' to mimic the thought process. From that time, Deep learning has been advanced regularly, with only the two-notable smash in its evolution. They both were tied to the infamous Artificial Intelligence developments. The first important Deep learning advancement came in the 1960s when the Soviet Mathematician Alexei lvakhnenko (helped by his assistant VG Lap) produced a small but practical sensual matrix. Deep learning is called deep because of the formation of those ANNs [60]. Four decagons back, sensual matrices were only two layers deep as it was not arithmetically practicable to build the biggest matrices. It is common to have sensual matrices with 10+ layers, and even 100+ layer ANNs are being tried upon. Using different sensual matrices in deep learning, computers now have the volume to see, learn, and react to compound circumstances as well or better than humans. The crash of deep learning in the industry started in the early 2000s, when things were already handled as approximately 10% to 20% of all the inspects written in the US, according to Yann LeCun [102]. Commercial applications of deep learning to large-scale speech recollection started around 2010. Deep learning is obtaining much popularity due to its ascendancy in terms of precision when trained with a large amount of data. In a normal way, Machine Learning is the set of algorithms that intrinsically data, learn from them, and use what they have learned to make brilliant resolutions. The father of deep learning is Geoffrey Everest Hinton. Geoffrey Hinton was born on 6 December 1947 in Wimbledon, London. He was an Alma master of the University of

Cambridge (BA) and University of Edinburgh (Ph.D.) and is known for Applications of Backpropagation Boltzmann machine Deep learning Capsule neural network. The abstraction of Deep learning is much tranquil; it can be created by using the (GUI) graphic user interface, which is more useful for the beginner who is impassioned to learn and for the apparatus of the deep learning algorithms. Deep learning is easier and more comfortable for the apprentice. The deep tools are the entourage of the python tools, which are exceptionally created for the methodological inspection of the high executes that sequencing the information [102]. The applications of deep learning are the virtual subordinate.

2.5.1 Convolutional neural network (CNN)

The convolutional neural network (CNN) is a deep learning approach to computer vision, and recognition of Graphical user interface components is very effective and highly used in today's times. It is a double shot or two-stage detector. It is a multi-layer neural network with a special architecture that enables it to recognize and detect the complexities in the data and images or information. It is highly used in image classification, deep learning techniques, robotics, Artificial intelligence, and autonomous vehicles [91]. The 'R-CNN', 'Faster R-CNN' and 'Mask R-CNN' are the types of double shot detectors.

Digital images compressor pixels and each pixel is represented by numbers between 0 and 255, which means every pixel in the image has a digital representation or identity, which enables computers to work efficiently with the images. Instead of a fully connected network to brief each pixel, the CNN weights to look at a patch of the object's image. Sources have comprehended the convolution neural network approaches by the example of reading a book using magnification glass. An individual will read the whole page but will look at the smaller detail for a single time. Due to using overabundant measures of information, deep learning networks can foresee the model of the information with high precision from the user accommodation and classification or detection of issues [91]. Deep learning is the new period of AI, and it is spreading massively in the public eye. It is being utilized in the web looking, internet business locales, independent vehicles, man-made consciousness, and every one of the uses of computer vision. The picture classification highlight of deep learning is the most progressive application on the planet with straightforward and complex pictures. The layers of the neural organization of deep learning engineering work better with pictures. Deep learning utilized CNNs for its approaches and algorithms [91].

A CNN works by following five convolution steps: rectified linear unit, pooling, flattening, and full connection. The first step, convolution, is the combined integration of two functions. It consists of three factors that are; Input image, feature detector, and feature map. A feature detector is also termed a kernel or filter. In the second step, after convolution, the rectifier function is applied to increase the non-linearity of the image in the convolution neural network. The deep convolutional neural network has been in the core position of the deep learning domain. However, CNNs were used as an image classification technique for simple digits or character recognition tasks in earlier times. However, due to the success of recent work, using a deep convolutional neural network to beat state-of-art in the ImageNet challenge, the deep convolutional neural network has become a common and useful tool for image classification problems. The CNN is similar to the traditional neural network; it also has weights and biases in each neuron. However, the CNN is more focused on complicated inputs, such as images. CNNs are structured by three essential ideas, local receptive fields, shared weights, and subsampling [3]. A typical CNN contains multiple hidden layers, such as a convolutional layer, pooling layer, and fully connected layer. A neural network is based on affine transformations. The input vector is multiplied by the weights vector to produce an output. When the input is an image, depending on the type of image, it can be seen as a single signal channel multi-dimensional array or a three-channel multi-dimensional array (a color image consisting of three channels, red, green, and blue). A convolutional layer performs a linear transformation. In this process, only a couple of input information contribute to an output unit. The shared weights are applied to different locations in the input. In a conventional neural network, pooling operations decrease the size of feature maps by utilizing some functions. Generally, there are two types of pooling, max pooling and average pooling. Max pooling means taking the max pixel value of one specific region.

On the other hand, average pooling means taking the average of one particular region. After the pooling, the transformation of the entire map is achieved through the fishing process. The neural map is passed through a network made up of the input layer, connection layer, and output layer. The fully connected layer is also called the hidden layer.

The CNN is the pennant of today's deep learning and machine learning approaches, and it can be used for clarifying the seniority of the complication [3]. Numerous software or tools utilize convolution neural network methods and approaches to classify and detect problems and objects. The Tensor is the category of the data and the formation which is used in linear algebra. The cognitive toolkit is used for

faster instruction and the effectiveness of the deep learning models. The cognitive toolkit can be taught to the cavalcade and corroborate with the different quintessence for the neural networks. The PyTorch is the initial which is started as a source of deep learning for the skeleton, invented and financed by the social app known as Facebook.

2.5.2 Graph Convolutional Network (GCN)

Representation of data in a graph could be helpful in several ways to give an insight into underlying information. Graphs have many real-world applications including fraud detection, computer vision, and social analysis. Graph structure data is normally non-Euclidean in nature that leads to complexity. However, this problem can be overcome by representing graphs in low-dimension Euclidean space with the help of the embedding technique. These include the traditional graph embedding method and network method. Feature transformation, aggregation of neighborhood nodes, and non-linear activation are propagation rules of GCN to refine the embedding. A large fraction of machine learning (ML) problems would be effective if modeled by the graph. A convolution graph is similar to a social graph representation that helps to understand the friend's connection in a social network. In GCN, the input is graphically provided to the neural network (NN). This way the graph did not predict a single value rather calculate a value for each node in the graph [203].

The GCN is a modern technique that attracts the attention of researchers. Its applications reside in different fields from anomaly detection to computer vision and analyzing the demand and supply. An overview of GCN is illustrated by Zhang et al [203]. The applications of GCN in some of the fields have been mentioned but it does not end here. The list of the applications of GCN is too long as it dramatically improved the state of art in different fields. The objective of the graphical convolution network was to leverage deep learning. However, certain modes are still suffering from the shallow structure. The deeper the architecture of GCN is, the smoother will be the representation of nodes on a graph. Although the researchers attempt to address the problem by skipping the connection-based model, the problem is still an open challenge [203].

Various deep learning architecture based on CNN has been employed for computer vision. However, some shortcomings are associated with CNN as it becomes difficult to encode the graph structure for specific learning. GCN performs better results to encode images, videos, point clouds, and meshes.

Visual reasoning is an emergent hot topic in computer vision. Classification of images is necessary for many real-world applications. By employing the graph construction method, the unstructured image could be converted into structured graph data and can be fed to GCN. Visual question-answering explores the answers of a particular question on images. Images are usually complex as they contain more than one object and understanding the link between those objects becomes necessary in visual questioning to characterize the interaction among them. Narashimen et al. proposed a GCN-based model for answering the questions by taking multiple facts of an image [127]. Cui et al proposed a GCN-based model that takes both graphs of words and scenes into account [33]. Yang et al. proposed a Graph Convolution (GC) model that considers the reliable edges while overlooking the effect of unreliable edges from the graph [194]. The model Graph R-CNN is efficient at detecting the objects in a picture and their relationship with each other. The results of the study showed that the proposed model outperforms existing methods for scene graph generation . In contrast to this model, Johnson et al proposed a GCN-based model that takes input in the form of a graph and generates an image out of it [86]. This way GCN helps in computer vision for extracting information from a picture as well as generates an image out of graphical data.

Action recognition helps in understanding a video but it is a complex task because there is a huge volume of data and high computational cost. The idea to develop GCN architecture was to bridge the gap between spatial-based approaches and spectral-based approaches by using automatic learning potential that also helps to tackle problems with arbitrarily structured graphs [31]. Felipe F. Costa classifies video actions using GCNs.

The proposed approach provides better flexibility and accuracy. Comparison of different video action classification models and their accuracy reveal that presented approach has highest accuracy among all models. However, it is a time-consuming task to generate the graphs that is the major drawback.

Point clouds are a convenient way to represent 3D data. 3D sensors have wide applications in robotics and self-driving cars. However, due to computational strains, 3D sensors produce noisy point clouds. Moreover, point clouds facilitate by providing a geometrical representation of different applications in graphics.

GCN is considered a powerful tool to process non-Euclidean data. Research has shown the power of GCN-based models in encoding locally as well as global information. Guocheng Qian et al. proposed NodeShuffle and Inception DenseGCN for

upsampling point clouds. The proposed GCN-UP pipeline performs better as it is more efficient in inference [144].

Another study conducted by brings forth an architecture named as Graph-Convolutional Point Denoising Network (GPDNet) to denoise the geometry of point clouds. The objective of the study was to investigate the potential GCN in dealing with permutation invariance problems during the processing of point clouds [139].

The experimental results of the study successfully learn the hierarchies of features. It proved that the proposed method would improve the state-of-art technique. Wang et al. proposed the GCN for both point cloud classification and segmentation [180]. Similarly, Velsesia et al. propose a localized generative model to generate 3D cloud models using GCN [173]. A study conducted by Nitika Verma presents a deep neural network –FeaStNet – to dynamically determine the link between nodes and filter weights in a localized graph [176]. The results of the study show that FeaStNet learns the properties of shapes as well as part labeling more efficiently and precisely.

CNN is not extended to data that is not represented in 3D meshes or another graphical form on which local convolutional operators are not applied. GCN finds correspondence between a collection of shapes thus has an edge over a convolution neural network. Litany et al. proposed a combination of auto-encoder and GCN-based models for image completion tasks [108].

Text classification is a classical problem in natural language processing (NLP). Application of text classification includes the organization of the document, spam detection, opinion mining, and news filtering. Traditional methods represent text as bags of words and n-grams. However, recent research in graph embedding has attracted the attention of the world. Information extraction is a cornerstone in applications related to natural language processing. The graph convolution has been applied in this field.

A study conducted by Yao et al. proposed a graph neural network-based method for the classification of text. The authors build heterogeneous text graphs that contain word nodes and document nodes [195].

The result of simple two-layer text GCN demonstrates that the model can achieve text classification results and also could learn predictive documents. Since the structure of a sentence contains rich linguistic knowledge it is useful for language understanding. The main issue in natural language processing is effectively integrating the syntactic structure information in neural language models. GCN successfully explores grammar. Recent studies have exploit GCN for improving the representation

learning in text mining tasks. These tasks include semantic role labeling and relation extraction.

For language processing tasks, the graph convolution gathers non-consecutive words based on syntactic constraints. Word dependency gate mechanism model the syntactic dependency, analyze the valuable dependency, as well as discard irrelevant dependencies [41]. This way GCN-WDG model effectively obtains valuable word dependency paths and encode relative syntax-related words.

GCN can leverage the structure of the network. Tobais et al. introduced a novel GCN-based model for road networks [83]. Traffic forecasting has become a challenge due to varying traffic patterns at different times of the day. Graphical Convolution can effectively extract the data of local traffic patterns in graphical form. GCN assists to learn the interaction between roadways and forecast the network-wide state of traffic. Researchers have to exploit GCN based model in traffic forecasting and traffic flow prediction by taking external factors, spatial, and temp factors into account. This traffic forecasting in return assists in highway traffic management. Traffic flow prediction is necessary to avoid traffic congestion as well as help the government in planning and developing roads.

Zhang et al. proposed and employed a hybrid GCN to achieve better results of traffic prediction at highways [205]. HGCN is a combination of GCN and Feedforward Neural Network (FNN). The network effectively obtained the non-Euclidean features of the highway network, weather type, and data type of toll stations to predict the subject effectively. The results of the study showed that the GCN-based model has high prediction accuracy as compared to other models such as LSTM, GBRT, and KNN.

Besides this, Guo et al. uses a spatial-temporal-based GCN for traffic forecasting [56]. The model – Attention-based spatial-temporal graph convolution model (AST-GCN) – capture the dynamic characteristics of traffic data. The accuracy of the results shows the potential of the proposed model in efficient traffic forecasting.

The demand for bike-sharing is increasing globally due to a shift in a sustainable lifestyle. For monitoring of supply-demand relation of bikes, the GCN data has been employed to get real-time accurate forecasting of demand. Since bike demand is affected by many complex factors such as weather patterns, holidays, Current time, and the number of bikes present at the station, the deep learning mechanism would be helpful to get meaningful information from the raw dataset. Frade and Ribeiro analyze the features of weather patterns and other temporal patterns for bike demand.

Similarly, Kim et al. proposed a GCN-based model to predict the hourly bike sharing demand [92].

The study considers the influence of global variables on the demand for bikes. The proposed framework reflects the spatial dependencies among stations and temporal patterns at different times. The predictive power of the model employed for this purpose was robust for the sudden change in the cycling environment. The results give a clear picture of the resilience of using GCN models such as GCN-UP and GCN-IDW in predicting the demand-supply curve. The study fairly contributed to minimizing the imbalance between demand and supply of cycling. Moreover, the same model could be employed to analyze the D-S of other transportation units.

GCN is a building block for neural networks on graph-structured data. Graph convolution has also gracefully found its applications for anomaly detection. Anomaly detection is quite important in artificial intelligence systems to encounter abnormal events that disturb the functionality of an organization. Ding et al proposed an unsupervised anomaly detection method using a GCN-based auto encoder [40]. The approach leverage topical structure and nodal attributes to reconstruct normal instances as small errors while encoding anomalies as large errors. However, the proposed method is prone to outliers that affect the credibility of the results obtained. Kumagai et al proposed two GCN-based anomaly detection models that are semi-supervised [99]. The first model labels only normal incidents while the later one labels both normal as well as anomalous incidents. However, both models suffer from hypersphere collapse problems. Another study conducted by Mesgaran et al. proposed a graph fairing convolutional network (GFCN) model for anomaly detection [116]. The model gets information from distant nodes by skipping connections between layers and detects anomalies using graph structure.

The GCN has its applications in particle physics. PartcleNet is built on edge convolution. It is a neural network that operates on particle clouds using jet tagging. Furthermore, GCN has its applications in IceCube signal classification. Another application of a graphical network is to predict the structural-based dynamics of a cube such as how a cube deforms as it hit the surface of a particular thing. Mrowca et al. proposed a hierarchical graph-based representation of objects that decomposes the particles of an object and could assemble the particles in the same group [125]. Then, they propose a GCN to predict the physics of particles.

GCNs have a wide range of applications in physics, chemistry, biology, and material science. Learning the chemistry, properties, and characteristics of molecules attract a lot of attention in drugs discovery and material science. In drug discovery,

GCN mode named DeepChemStable is used that predict the stability of a compound. Moreover, by learning the interaction between drug and target protein using a graph convolution network, it becomes easy to analyze the polypharmacy side effects. Prediction of protein interface has become a challenge in drug science [105]. Fout et al. construct a graph by employing a GCN model where each residue in a protein is considered as a node [46]. The sequence and structure of amino acids in each protein are considered as features of nodes. To predict the interface of protein, GC layers are used for different protein graphs.

Another application of GCN in chemistry is the prediction of molecular properties. Message-Passing Neural Networks (MPNNs) can be used to predict the properties of the molecule. PotentialNet uses graph convolution over chemical bonds to learn the characteristics of constituents. The bond-based and spatial-distance-based propagation was entailed, and a graph was gathered over the ligand atoms. Later on, a connected layer was formed for the prediction of the molecule's properties [44].

Besides the applications in social science, GCN has been applied in social network analysis. Fake news can be detected using a graph convolution network. Furthermore, GCN has been widely used for social media recommendations. The social recommendation aims to boost the recommendation performance. Ying et al. proposed a GCN-based model – PinSage – to find the interaction between pins and boards in Pinterest [198].

Another dimension in which graphical convolution help is fake news detection. Exposure to fake news misleads the masses. Automatic detection of fake news has become crucial as well as challenging. Guoyong et al. proposed a multi-depth GCN framework to detect the fake news spread on social media. The experiment of the model was conducted on one of the largest fake news datasets of the world – LIAR – to confirm that M-GCN outperforms among the latest five methods proposed [70].

Prediction of stock price movement is necessary as well as challenging in financial markets. Previously it has been inferred that stock price fluctuation depends upon its information thereby neglecting the cross-relation among involved stocks. Now it has become well known that the price of the stock is correlated with the price of other stocks. A graphical convolution network could predict the movement of stock. Jiexia et al. proposed a framework called Multi-GCGRU that is the integration of GCN and GRU to predict the stock movement. Experiments on stock indexes of China market showed that the proposed model is feasible to incorporate stock relationships [196].

GCN is an automatic learning build to tackle problems with arbitrary graphs. It has greater advances in deep learning so GCN has wide range of applications in social

and material science, theft and fraud detection, computer vision – images, videos, meshes, and point clouds – and natural language processing. Moreover, GCN help governments to minimize the traffic congestions and accidents by providing real-time traffic prediction data. In the same manner, it helps to predict the fluctuation in stock market by carefully understanding the other stocks. Lastly, GCN help in demand and supply of transportation sharing such as bike sharing. GCN is still Facing challenges due to some architecture problems. However, it is still have promising applications.

2.5.3 Generative Adversarial Network (GAN)

A generative Adversarial Network (GAN) is a model of machine learning that is operating with two neural networks. According to this theory, both networks working effectively on the same input and GAN predict the most accurate output [2]. GAN has based on AI technology that is the most recent and widely used in the Google search engine, business software, Intelligence forces and armed institutes including police for different purposes including image recognition and mapping. AI technology worked through three orbits which are artificial intelligence, machine learning and deep learning [32]. All these networks are connected with the neural nodes and resulted in an effective and accurate outcome.

GAN is the model and class of machine learning and process all the information according to the AI system. This model is widely using in voice, video and image recognition, processing and generation [179]. This model is working through the two networks which are generator Gx and discriminator Dx that enable the system to present accurate results [138]. In the era of modernisation and innovations, the use of this model for fake news, image and voice is increasing which is a negative factor of the technology and against ethics. There is a need to use this innovation in business, healthcare and other positive outcomes instead of the unethical use of the technology. This section of the literature review provides the GAN application in different sectors as major applications. The last part of the LR discusses the application of the GAN model in the deep learning area of AI.

GAN model is a multifunctional and diverse purpose-based tool that has a wide range of applications in different sectors. Healthcare, Business, Cybersecurity, Animation, Translation and Editing are the major are most important fields in which the GAN model is assisting to improve the quality and performance [76]. The healthcare sector is the most beneficiary of the neural network and GAN model because it is helping in the area of diagnosing and radiation to make appropriate images and observe the movement of internal body parts [197]. It is helping to develop the 3D

image for a better understanding of the malfunctioning and illness. It is a competent tool for drug recovery and tumours identification.

This tool of AI is also helping to effectively deal with business activities and improve decision-making efficacy. This model is applicable in the presentation of quality images and improves software authentication by selecting the appropriate network like in accounting software [169]. In the field of cybersecurity, this tool has vital applications in maintain privacy, detecting unauthorised users and recognise the official users and providing permission to use and log in. This tool of Machine Learning is increasing the quality of images and providing a better design for the one input into different outcomes.

There are different applications of the GAN model in deep learning to generate the most attractive designs, outputs and changed the images, videos into fake human images [22]. It converted the rough images into attractive colours and designs. The research studies in the AI and IT have demonstrated the followings applications of the GAN tool in deep learning.

It is the most effective application and feature of the GAN model that has been used in business, trade, healthcare and image editing based on deep learning. Through the neural coordination and supportive physical apparatus, this tool helped the detection of an object. For example, if the system has been stored and coded with different objects then GAN will help to detect accurately those objects [182]. If the hand, book or snacks will be pitted in the GAN then it will be detected accurately. This tool has been used in different supermarkets and departmental stores to recognise the specific object, code and bar. It is also used for the security purpose to detect any weapon and used for drug recognition at airports.

GAN applications has also widely used in cybersecurity, phone privacy and official data protection including private files. In this kind of working the model can recognise the fingerprint that has been registered in the system [72]. The image and face one time has been stored as the output and later this process will be worked. When any irrelevant image, fingerprint or face appeared on the sensor or camera then the access has been prohibited and neural networks recognised the transaction as unauthorised activity.

The GAN system has the ability due to its efficacy and feature of AI including graphic features to convert the 2D and simple objects and images into 3D images that are most easy to understand and work furthermore on the object. The field of architecture is based on this technology and maps are developing through the neural network's efficacy [204]. The 3D drafts and mapping are more accurate, feasible and

easy to work with. The best example of the 3D generation images from this tool is chair, table and a room development structure.

The GAN theory and neural networks are most important in business intelligence like google is worked according to the AI. There are different tools and software that are used in business studies to make the strategies, plans and reports related to finance, training and accounts. This model effective coordination of the neural nodes provided the most suitable design, strategy and techniques related to the on-going project [110]. For example, if there are already coded tools and techniques of management then working on new management will suggest the appropriate tools for recent use based on the previous practices and actions. In this way, the GAN is helping to resolve the account and management related issues.

It is the most effective feature and application of the tool that reduced the time of working from the start rather it is converted the image text into another language and reduce the manual working. This system worked with the online and offline tools that have been already made and languages are installed in these systems [78]. When the image of the English language is putting from the person to process it in the French language then neural coordination and nodes worked more accurately and selected the most similar and suitable words in the image. In this way, it reduced the working on the manual dictionary, image creation and translation.

The process of translation from image to text is lengthy because there is a need to first develop an image again and then translate the image text from basic language to the needed one. But the application and features of the GAN model are reducing the manual work and save time [86]. This dual neural network can change the image into text within seconds and there is no need for manual translation. Nowadays there are different features that GAN is supporting like a photo to emoji, face ageing and video prediction. There is a need to reduce the negative and fake use of the GAN model and make effective utilisation of the innovation.

2.6 Deep Learning Techniques for the Recognition of GUI Components

Implementation and application of a GUI is a multi-step process. In software development, the GUI application is a complex process that needs a proper identification of all the components, machine language, assembly, and complications. These complex steps are based on the initiation by developing the idea of required GUI, refinement

into visual design, assessed through relevant prototype or design. It is ended by identifying problems and acquiring user feedback. Due to the advancement, the GUI is altering and acquiring new tools and techniques to address user problems and provide convenience—deep learning tools and techniques at enhancing and improving the GUI components and their functions.

2.6.1 Deep Learning Tools for GUI Components

The most widely used and effective tools or software to recognize Graphical user interface components are Pix2code, Sketch2code, ReDraw, and Rico, which uses a CNN and KNN approach.

2.6.2 Pix2code

Beltramelli [14] describes an important development in this area, explaining how deep learning can transform screenshots of the GUI created by designers into computer code. Betramelli achieves 77% accuracy on three different platforms (iOS, Android, and Web-based technologies) by using deep learning to train the model's code to automatically generate a single image end-to-end. The authors believe that this is the first attempt to solve the problem of generating GUI code from visual input by using machine learning to understand potential variables rather than complex problem solving engineering [14]. The paper states that GUI components are synthetically generated, but the author does not give a way to generate DSL code.

The pix2code dataset maps bootstrap-based websites into a DSL consisting of 18 vocabulary tokens. It consists of 3,500 image and mark-up pairs, which was split into 80% train, 10% cross validation, and 10% test set. The dataset is rescaled by PyTorch from 2,400 $\times$ 1,380px to 224 $\times$ 224px in order to fit the requirements of the ResNet model. The ResNet model is used to extract features from the image (size 1 $\times$ 1 $\times$ 2,048px per screenshot) which it passes onto a decoder model.

The author relies on two methods in computer vision literature to capture images. The first is the CNN, and its unsupervised feature learning that performs GUI is mapped to learning representation. In addition, a recurrent neural network (RNN) is used to perform language modelling of descriptive text, which is associated with GUI image input. It is a Keras-based implementation, and the author calls this method 'pix2code model architecture', which is used to solve three phases of problems: computer vision creation of GUI, language modelling of computer code

understanding, and then using these outputs to generate objects represented by these variables described by computer code.

Another related work is a project developed by Emil Wallner, which is another Keras-based implementation of pix2code, using the same dataset. it differentiates itself from pix2code by replacing the pre-trained image features with a light CNN. Instead of using max-pooling to increase information density, it increases the strides. Andrew S. Lee also made improvements on the basis of pix2code. Unlike the single end-to-end pix2code model, his system follows an image captioning model previously created for PyTorch: an encoder CNN and a decoder RNN. As a whole, the system takes in a screenshot as input and outputs a sequence of indices (based on DSL language's vocabulary) which then convert into valid HTML.

2.6.3 ReDraw

A ReDraw dataset [122] comprises many Android screens captures, GUI metadata, and annotated GUI segment pictures. It incorporates 15 classifications like 'RadioButton, ProgressBar, Switch, Button, and Checkbox in 14,382 GUI pictures, and 191,300 annotated GUI segments'. The quantity of every segment comes to 5,000 after the dataset is handled. For more data about the dataset, see The ReDraw Dataset. This dataset is utilized for preparing and assessing the AI advancements of the CNN and K-closest neighbor (KNN) referenced in the ReDraw paper [122]. The paper proposes a technique for computerizing the change from GUI to code in three stages: Detection, Classification, and Assembly.

It starts with implementing or utilizing the CV technology to remove GUI metadata from the plan draft, for example, bounding boxes (positions and sizes). Then it uses a huge programming distribution center to perform information mining and dynamic examination to get the segments that show up in the GUI. After utilization and assessment, the outcome information uses a CNN technology dataset to figure out how to arrange the separated components into explicit sorts, like Radio, Progress Bar, and Button [168]. The third step is the Assembly of these components and the output Information.

2.6.4 Sketch-to-Code

The GUI design process involves many creative iterations. This process usually starts on a blank sheet of paper, where designers and software engineers share ideas and try

to design scenarios or workflows that their clients want [153]. Once there is a preliminary design, it is usually captured through photos and then manually translated into an application or web pages that can be accessed on computers or mobile devices. This translation requires time and effort and usually delays the design process.

For GUI designers, sketching is the most natural and intuitive form of expression, so they aim to skip the intermediate steps and generate the final product directly from a sketch. For example, designers from Airbnb believe that the time cost of validating an idea should be zero; that is, when they have an idea, they should be able to immediately generate an app prototype for testing. The technology that will emerge in the next few years will remove obstacles in the development process and allow people to design products more intuitively [167]. Because Airbnb has developed a mature design system, each component has its own name, and a machine is trained to identify different sketch symbols and corresponding components. After training, the machine can recognise most manuscripts and generate code directly from the combined symbols, which are represented as interfaces in browsers.

In addition, Microsoft has also developed an artificial intelligence web design tool called Sketch2Code, which can convert website sketches into functional HTML code [167]. Sketch2Code has been trained with pictures of different handwriting designs and marked with common HTML related element information, including text boxes, buttons and images. It can store the information associated with each step of the HTML generation process, including the original image, prediction results, layout and grouping information. It also supports Microsoft Azure and uses it as a back-end entry point to coordinate the generation process through interaction with services. The new designs can upload and view the generated HTML results through its website.

However, these companies have not disclosed their source code solutions. Although it is a promising example machine-aided design, how much of this model is fully trained end to end and how much it relies on hand made image functions are unclear.

2.6.5 RICO

Rico [36] is by a long shot the biggest versatile GUI dataset, made to help five sorts of information-driven applications: 'design search, GUI layout generation, GUI code generation, user interaction modelling and user perception prediction.' The Rico dataset contains 27 classes, in excess of 10,000 applications, and around 70,000 screen captures. From that point forward, there have been a few examinations and applications dependent on the Rico dataset, for instance, Learning Design Semantics for Mobile Apps. This paper presents a code-and visual-based technique to add

semantic comments to versatile GUI components. Using this technique, 25 GUI part classifications, 197 content button ideas, and 99 symbol classifications can be distinguished depending on the GUI screen captures and view order.

2.7 Evaluation Metrics for GUI Structure

2.7.1 Bilingual Evaluation Understudy (BLEU)

Bilingual evaluation understudy (BLEU) is a score for comparing a candidate translation of text to one or more reference translations [137]. It is particularly preferred as a metric that has been used widely to evaluate Natural Language Processing (NLP) systems and other branches of machine learning which help to produce language, specifically machine translation (MT) and Natural Language Generation (NLG) systems. BLEU matrices are being consumed widely by different organizations and other platforms in NLP for over fifteen years with an aim to evaluate NLP systems specifically within natural language generation and in machine translation [111]. This technology is perceived to play a major role in substituting actions taken by humans and by providing efficient alternatives to NLP for the past few years [142]. Furthermore, BLEU itself computes the word-based overlap with the gold-standard reference text [166]. Also, among its different applications within NLP, it is being used widely as an evaluation matric depending on the assumptions which let it correlate with and predict the real-world utility of the natural language processing systems that are measured extrinsically through task performance or user satisfaction. Therefore, through a survey conducted targeting the participants on one IT company, it was concluded that BLEU matric was used by their company to ease down the human efforts and indulge different aspects and calculations which helped the NLP based activities to stay efficient. BLEU retains a quality of computing word-based overlap with the gold-standard reference text, which lets it predict the real-world utility of the NLP systems [58]. One of the BLEU's applications targets the area of clinical medicine according to which, the BLEU metric is used to evaluate the AIDS diseases and its medication through its impact on viral load instead of assessing it explicitly whether it leads to longer or higher-quality life [152].

BLEU is particularly a score used for comparisons and contrasting the candidate translation of a certain text to one or more reference translations and it has been used differently in NLP products [47]. It is claimed that BLEU was originally developed for the translations purposed but today it is being consumed widely to evaluate text

generated for the task related to NLP. Additionally, regarding its application on Machine Translation (MT), it is claimed that the BLEU metric scores the translation on the scale of 0 to 1 intending to measure the tolerability of the MT output. This technique metric is applied on NLP systems so effectively which is proved through the closeness of score with MT output. Closer to one the test sentences score is the more overlap is present with the human reference translations and therefore, the better the NLP systems are estimated to be.

Some major applications of NLP include chatbots, text summarization, machine translation, and language models [81]. All of the NLP applications are known to generate some text as outputs additional to the automatic speech recognition and image captioning. All these products of NLP are protected by BLEU to reduce human errors and provide accurate translations. According to which there are a number of responses while translating a certain text and this condition is, even more, trickier in case images and video inputs which can be at the same time [172]. BLEU techniques are known to produce the best matching response to the original input. This problem has been solved through the BLEU score which is not even perfect as it contains numerous drawbacks. However, it is not complex and has different benefits due to which it is used frequently as a metric [119]. The most recognized approach of BLEU score is that the more closely the predicted text is to the human-generated target sentence, the perfect it is. As the score of BLEU are in the middle of 0 and 1, so the sentence score of 0.6 or 0.7 is considered as the best output score. BLEU score 1 is not realistic to achieve because two people cannot come up with a similar sentence variant for a certain problem so attainment of a perfect match is not possible.

It can be claimed that the most realistic results are achieved when the BLEU score is less than one [30]. Furthermore, N-gram and precision are considered as widely used concepts or applications of the BLEU that are used from regular text processing. These applications are not specific to BLEU or NLP which means they are employed by other output-based technologies too [199]. While investigating the drawbacks and benefits of BlEU score within NLP systems, this technique of machine translation is understandable and quick [27]. Also, it was assessed through this study that it collides and cooperates with the human ways of evaluating similar text and is language independent due to which it can be straightly applied to the NLP models. On the other hand, some drawbacks of the BLEU score were also assessed through this study according to which meanings of words are not considered due to which humans can change the word with synonyms if find useful. This lowers the accuracy of the BlEU score which is not desired and the BLEU score considers that word incorrect.

Furthermore, BLEU looks at the same word matches and considers the variants of the same word wrong. For example, raining and rain are not similar for the BLEU score which is a major shortcoming of BLEU applications within NLP. The BLEU scores are often not recommended by organizations because orders of words are not considered by it [210]. It is evident that sentence meanings get change by changing the order of words. For example, 'the guard arrived late due to the rain' and 'the rain arrived late due to the guard' are not giving a different meaning by just changing the order of the words. BLEU score will get these sentences the same which is a major drawback.

2.7.2 Website Structural Complexity (WSC) Metrics

Jin, Zhu and Hall [85] proposed an abstract model of website GUI as a directed graph, where a website can be modeled as a pair $< G, S >$, where $G =< V, E >$ is a directed graph representing the website; V is the set of nodes representing web pages; E is the set of edges representing links between web pages; and S is the start node of the graph, i.e. the home page of the website. The directed graph must also satisfy the condition that all nodes v in V are reachable, i.e. there is at least one path from the home page to node v. They suggested the use of the Number of Independent Paths (NOIP) as a measure of hypertext navigation complexity. The larger the NOIP, the more complex the website structure is.

This idea was further investigated in Zhang, et al [206]. Five website structural complexity (WSC) metrics were proposed. The metrics were evaluated against Weyuker [185]'s axiom system of software complexity.

$$WSC_1 = \sum_{i=1}^{n} outlink(i) \tag{2.1}$$

where: $outlink(i)$: out-link of a given page i, n: number of pages in a website. From graph theory, for all directed graphs, the sum of in-links of all nodes is equal to the sum of out-links, which is equal to the total number of clickable links. Therefore, we have that

$$WSC_1 = \sum_{i=1}^{n} inlink(i) = total\, number\, of\, links \tag{2.2}$$

WSC_1 catches the intuition that a small website with fewer pages and links are less complex than a large web site that has hundreds even thousands of pages and links. However, for comparison purposes, it is desirable to know its relative complexity

taking into consideration of the size. Dividing the overall complexity by the number of pages gives a normalized complexity.

$$WSC_2 = \frac{WSC_1}{n} = \frac{\sum_{i=1}^{n} outlink(i)}{n} \tag{2.3}$$

Informally, WSC_2 defines structural complexity as the average number of links per page.

As suggested in [85], the number of independent paths in a hyperlinked network of web pages can be used as a complexity metrics. Let NOIP(G) denote the number of independent paths in a graph model G. We define the following metrics.

$$WSC_3 = NOIP(G) \tag{2.4}$$

According to graph theory, the number of independent paths in a directed graph G can be calculated by the following formula [208].

$$NOIP(G) = e - n + d + 1 \tag{2.5}$$

where e is the total number of links in the graph, n is the number of nodes in the graph and d is the number of dead end nodes in the graph. We have that

$$WSC_3 = e - n + d + 1 \tag{2.6}$$

We can also define a relative complexity metric based on the average linear of independent paths as follows.

$$WSC_4 = \frac{WSC_3}{n} = \frac{e - n + d + 1}{n} \tag{2.7}$$

Not only does the number of out-link and in-links affect website structural complexity, but also the distribution of the links within a website. For a fixed number of links, a website in which links are concentrated in a few pages is more complex than one in which links are mostly evenly distributed. In the discussion of software structural complexity measurement, Belady and Evangelisti [13] applied interconnection matrix representation of partition to their study and suggested that complexity increases as the square of connections (fanout), where fanout is number of the calls from a given module. In website designs, all pages are connected by hyperlinks. This leads to the following metrics, WSC5, for website structural complexity.

$$WSC_5 = \frac{\sum_{i=1}^{n} outlink^2(i)}{n} \tag{2.8}$$

Other metrics of website structural complexity have also been proposed and investigated in the literature; see, e.g. [39] for a survey of web metrics. In comparison with assessment methods and analysis methods, navigability metrics have the advantages of objectiveness and the possibility of using automated tools to evaluate large-scale websites efficiently. Therefore, this paper takes this approach.

2.8 Summary

The analysis and review of current literature sources on the GUI and deep learning techniques revealed the advancements and the revolution in the fields. The graphical user interface components need classification and visual recognition, which is possible and convenient through deep learning techniques. Studies revealed that Computer vision is the most important aspect of deep learning, which uses different algorithms for object detection, tracking, classification, etc. By using the computer vision approaches, the graphical user interface components can be tracked and identified. The techniques and approaches use single shot and double shot detectors or neural networks for detection and classification purposes.

By reviewing the existing research on deep learning models and evaluation metrics for automatic GUI generation, a number of critical issues must be addressed, such as:

- Do the existing evaluation metrics for GUI structure work for the deep learning model?
- Can the performance of the existing pic2code model be further improved?
- Can we modify or improve the existing dataset for a better performance?
- Can we find a better model or approach to outperformanced the existing one?

To address the above questions, the following approaches are intended to apply:

- To develop a new metric to evaluate the accuracy for GUI layouts.
- To propose a modified framework for solving the problem of feature vector losses in pix2code framework. The outcome should outperform the existing method based on BLEU, a metric for NLP.
- To propose a new GAN model for GUI generation. To achieve satisfactory results, a new data augmentation method should be developed to address the issue of the insufficient data.

- To develop a scene graph model to generate GUI examples such as actual texts, images, and buttons. The new model is able to predict the object segmentation mask and frame in the GUI.Then a GUI generation method will be applied to convert the scene layout into a real GUI.

Chapter 3

Evaluating Semantic Similarity for Source Code Translation

3.1 Introduction

In the information world, which cannot be separated from computation and programming, translating source code from one programming language to another is an indispensable task. Software vendors often develop multiple versions of programming languages so the same product can run on different platforms (Windows, iOS, Android and UNIX). With the increasing research in statistical machine translation (SMT), researchers in the field of software engineering (SE) have been studying how to use natural language processing (NLP) technology and SMT modeling to translate the source code of different programming languages in recent years [130, 89, 132].

Source code translation must ensure that the code runs correctly, because it has strict specifications compared to ordinary text. In natural language translation assessment, there are some effective automatic measurement methods, such as BLEU scores [137]. Unfortunately, no automated metric has been successfully validated to assess the accuracy of source code translation.

Since the 1970s, similarity detection in source code plagiarism has been studied in academic and industry communities [124, 59]. Attribute counting is the earliest code-detection technology proposed and applied. Subsequently, a semantic measure based on code semantic information for similarity comparison emerged [37, 97]. In addition, the model based on Extensible Markup Language (XML) and the method based on semantic trees are also successfully applied to the partial similarity recognition system [189]. At present, semantic metrics are widely used in most similarity recognition systems because of their wide applicability and high detection accuracy. Stanford

University's Measure of Software Similarity (MOSS) is one of the most popular tools designed using semantic metrics technology [156].

Because most of code plagiarism detection technologies also use semantic measurement, the focus of this chapter is to judge the accuracy of code translation by referring to the semantic measurement method of code plagiarism detection. We can feasibly establish an automatic measurement method for evaluating the quality of SMT-based code translation tools by combining related code similarity detection technologies. Based on this, we propose code semantic metric (CSM) for empirical research on source code translation metrics, and seek to verify the following two questions:

- Whether existing code plagiarism detection tools can effectively compare to the translation quality of the SMT-based code translation model.
- Whether CSM can better reflect the similarity between reference code and translated code than existing code plagiarism detection tools.

3.2 Related Work

According to the ways in which the code features of the program are extracted, the similarity measurement techniques are divided into two categories: attribute-counting and structural metrics.

3.2.1 Attribute-Counting

The attribute-counting method mainly performs statistical processing on various attributes included in the source code, maps these attributes to the vector space and calculates the similarity between the two. software science metrics [59] is the earliest attribute-counting method. First, the metrics of software similarity are given, and several software metric features are defined. Then, the software metric features contained in the source code are statistically mapped to corresponding feature vectors. The cosine metric formula calculates the similarity between two vectors as the similarity between the two software programs.

Most of the subsequent attribute-counting techniques are based on software science metrics. In 1996, Sallies et al. [154] considered the six-part program attributes of capacity, control flow, data dependence, nesting depth and control structure in statistical software metrics and formed a six-element vector matrix. The similarity calculation was then performed on the sextuple vector. Experiments show that this

method's detection efficiency is better than that of software science metrics, but the detection accuracy is not ideal, and misjudgments occur. Some researchers have proposed to increase the dimensions of the feature vector based on software science metrics to improve detection accuracy, but the experimental results did not improve significantly. Verco et al. [175] pointed out that increasing the vector dimension does not reduce the error rate of detection or improve detection accuracy.

3.2.2 Structural Metrics

Compared with the attribute-counting method, structural metrics add more internal structure information and implicit semantic information to the detection process, and structural information (control flow, nesting relationship, calling relationship, etc.) inside the program. It performs in-depth analyses to generate data sequences representing the meaning of source code and then calculates the similarity of the data sequences. Structural metrics' detection accuracy is improved compared to the attribute-counting method. The detection method based on structural metrics usually includes the following two steps:

1) Source code formatting. This includes converting identifiers in the source code to specific symbols, filtering out blank lines and comment statements in the code, unifying uppercase and lowercase letters and more. There are many formatting methods. The methods used by researchers include: string-based, token-based, tree-based and semantic-based [84].

String-based: First, the source code is divided into strings by line; each program fragment contains adjacent strings. Second, it judges whether the strings in the two program fragments are the same. Finally, it judges whether plagiarism occurred according to the similarity of the fragments contained in the program. A more representative string-based detection method is the parametric matching algorithm proposed by Baker in 1995 [8]. The algorithm unifies the identifiers and literials in the source code and then compares the similarities. However, after unified formatting, the detection results will have a large deviation, which affects detection accuracy.

Token-based: The source code is lexically analysed and a sequence is generated; then, the same sequence fragment in the sequence generated by the two program codes is detected. Compared with string-based methods, token-based methods are more robust at detecting code formatting and code spacing. The detection efficiency of token-based methods is very high, but detection accuracy is still poor [146].

Tree-based: To perform lexical and syntactic analysis on the source code, and obtain corresponding abstract syntax trees (ASTs) [87]. If two sub-trees contained in

two ASTs are identical, they are determined to be similar. The similarity of programs is judged according to the similarity of the similar sub-trees contained in the AST species [7, 11, 10]. Compared to string-based and token-based methods, the detection accuracy of the tree-based method is significantly improved. However, the AST-based method's efficiency leads to higher optimization costs and poor detection efficiency [146].

Semantic-based: Komondoor et al. [96] first converted the source code into a program dependency graph (PDG), and then used program slicing [184] to determine whether the sub-graphs of the two program dependency graphs are identical or isomorphic, thereby determining whether plagiarism is suspected. The semantic-based method has high detection accuracy, but it is difficult to detect the source code of large data due to extremely high space-time complexity, which makes it impossible to obtain practical applications [146].

2) The similarity calculation is performed on the data sequence obtained by formatting the source code, and the similarity value between the two data sequences is obtained. Commonly-used methods are the vector space model method and string matching algorithms, including: the Levenshtein distance [128], cosine similarity [165], longest common subsequence [115], Greedy String Tiling (GST) [187] and Running Karp-Rabin Greedy String Tiling (RKR-GST) [188].

3.2.3 Hybrid Metrics

Structural metrics add more program structure information to the process of source code detection, which improves detection accuracy more than attribute counting. However, some complex plagiarism methods can not be detected without in-depth analyses of program data flow and control flow. To better balance detection efficiency and detection accuracy, most recently-developed source code plagiarism detection systems combine attribute-counting and structure metrics, such as JPlag [140], MOSS [156], Sim [53] and YAP3 [189]. MOSS, YAP3 and JPlag are among the most widely-used systems. MOSS's core algorithm is the Winnowing algorithm [109], and both YAP3 and JPlag use the RKR-GST algorithm.

There are many key issues to be solved in the field of source code plagiarism detection. Based on existing source code plagiarism detection research, this chapter will study the detection accuracy and detection efficiency of code plagiarism detection methods.

3.3 Methodology

This section describes the datasets and our proposed CSM. In evaluating the translated source code, in addition to comparing the structural similarity of the code, it is important to consider the semantics and functions of the generated code. The more similarities between the semantics and functions of the translated and reference codes, the better the translation quality. By evaluating existing code plagiarism detection tools and technical experiments in existing literature, we will provide empirical evidence to prove whether the accuracy of code translation detection using technology in these areas can improve detection performance.

3.3.1 Datasets

We choose two SMT-based datasets from lpSMT [129] and mppSMT [131] to evaluate algorithm performance. The SMT-based datasets focus on phrase and grammar translation. A total of 34,209 pairs of parallel methods corpuses written in Java and C# were collected from these datasets. The datasets were manually created by developers and were initially used in nine open-source systems developed in Java. They were then translated to C#. A total of 2,250 semantic scores were allocated manually by human judgement (0, 0.25, 0.5, 0.75 and 1, respectively). Each score index was calculated on the basis of each line in the translation result. The higher the score, the closer the translated code is to the reference code.

```
1   // Java code: ClientQueryResult.java
2   public ClientQueryResult(Transaction ta, int  initialSize ){
3     super(ta,  initialSize );
4   }
5
6   // C# code: ClientQueryResult.cs
7   public ClientQueryResult(Transaction ta, int  initialSize ) : base(ta,
            initialSize ) {}
8
9   // Translated by lpSMT:
10  public ClientQueryResult(Transaction ta, int  initialSize ) : base(ta {,
            initialSize ) ; }
```

Figure 3.1: lpSMT Example

3.3.1.1 lpSMT

The lpSMT dataset [129] can adapt to translation between phrases and produce translation codes with high lexical and code symbol sequence accuracy. When multiple tokens or sequences of token appear in the wrong place, the semantic accuracy of the code is directly affected. Figure 3.1 shows an example of the lpSMT model, which converts calls to parent class constructors through 'super'. In Java, 'super' is called inside the method body. Conversely, in C#, the call to the constructor is done through the base and occurs before the method body, i.e. in the method signature, such as 'base(ta, initialSize)'. However, in the translated version, the call is divided into two parts: one is in the method signature 'base(ta', while the other is in the method body ',initialSize);'. Therefore, the translation code is grammatically incorrect. In this case, lpSMT is obtained based on the transformation of method signatures and markers in the body, but it does not take into account that the entire grammar unit of the constructor invokes the parent in 'super' for conversion.

3.3.1.2 mppSMT

The mppSMT dataset [131] is mainly for syntax translation; that is, translating code in the syntactic structure first, and then aggregating the translation code of all structures to generate the final translation code. To improve semantic accuracy, mppSMT integrates type mapping and API usage between two languages in the translation process. Compared with existing technologies, this strategy is more effective at achieving higher grammatical and semantic accuracy [131]. An important feature of mppSMT is that a large part of the translated code is semantically correct, but it is obviously different from manual translation. Specifically, the correct code involves a) code with a local variable name different from the reference method, but all variables are renamed consistently; b) code that adds or deletes namespaces to a type (e.g., new P.A() and new A()); c) code that adds or deletes 'this' code in an existing or method has the same identity; d) syntax units and code for different API purposes, such as field access for getters or array access for method calls.

3.3.2 Our Proposed CSM

We design CSM as a comprehensive operational approach based on N-grams [20], Term frequency-inverse document frequency (TF-IDF) weighting scheme [61] and cosine similarity [165]. We name our methods CSM 3-gram and CSM 4-gram according to the N value as described in the following section.

Table 3.1: 2-grams of the Java Program 'Hello World'

2-grams code	count
System.	1
.out	1
out.	1
.println	1
println (	2
(‘	1
‘Hello	1
Hello World	2
World’	1
’)	1
);	1

3.3.2.1 N-gram

N-gram [20] is a statistical language model that is widely used in many domains, such as speech recognition, text recognition, machine translation, information retrieval, text classification and other fields. Program code text is essentially pure text, which has more structural features than natural language text, so N-grams can also play a role in detecting program code plagiarism. Using N-gram to represent the program code to be detected not only maintains the context order of the program code, but also transforms the program code into an N-gram set [20].

In the case of text processing, an N-gram is a sequential sequence, and N is the markup length in the sequential data. When N = 1, it discards all information about word order, so that all possible sequence alignments may produce the same vectors. To solve the problem of local sorting, N-grams must divide text into all sequences of length N: 2-grams, 3-grams, 4-grams, etc. For example, when N = 2, the Java program 'System.out.println ('Hello World');' is split into non-alphanumeric characters as 2-grams sequences (see Table 3.1). In addition to splitting text into words, a single character can also be used as a token for N-grams. For example, the character-level 4-grams in the text 'print('a')' is represented by the set prin,rint,int(,nt(‘,t(‘a,(‘a’,‘a’).

3.3.2.2 TF-IDF

A common situation with N-gram is that some unimportant sequences in the dataset appear more frequently than others. This requires scaling feature vector values ac-

cording to the relative importance of each feature. An effective way to scale the features is to use the TF-IDF weighting scheme [61].

TF-IDF is a statistical method used to assess the importance of a word to a document set or a document in a corpus. Word importance usually increases with the number of times it appears in documents, but inversely decreases with how often they appear in corpus [61]. The TF-IDF calculation is as follows:

$$TF - IDF(t, d, D) = TF(t, d) \cdot IDF(t, D) \tag{3.1}$$

Where $TF(t, d)$ is the frequency of the term t appearing in the current document d, and $IDF(t, D)$ is the frequency of the term t appearing in the whole text domain D. If a term appears in many texts, its IDF value will be high. For example, the word 'void' appears in almost all programs in C#. Although 'void' has a higher word frequency, it has a much lower importance than 'ref', which is used less frequently.

3.3.2.3 Cosine Similarity

Because TF-IDF can be used to extract terms in a document, and terms can often be used as the basis for us to judge whether two documents are similar [61]. Therefore, we can extract the terms between the two documents, and then use the cosine similarity to calculate the similarity between the terms of the two documents to obtain the similarity between the documents. We convert the terms of the two documents into word frequency vectors, and then calculate the cosine similarity between word frequency vectors to obtain the similarity of the corresponding documents.

In lpSMT and mppSMT, we select key terms from the documents of reference code and translated code. After that, we combine the key terms of the two documents into a set. For each term in the set, calculate its TF-IDF value in the reference code document and the translation code document respectively. Then, we can obtain the word frequency vector A of the reference code and the word frequency vector B of the translation code in both datasets.

The calculation of cosine similarity [165] can be regarded as the operation between two vectors:

$$cosinesimilarity(A, B) = \frac{A \cdot B}{\|A\| \, \|B\|} \tag{3.2}$$

After normalising the word frequency vectors before computing the distance matrix, the angle between two vectors will be calculated. When comparing the angle, the difference between the two directions is generally pointed out. When the angle is

0 degrees, the two directions are the same and the vectors coincide; when the angle is 90 degrees, the two directions are different and completely independent. The closer the cosine value is to 1, the closer the angle is to 0 degrees and the more similar the two vectors are. In this way, the similarity score for the two source codes is obtained.

3.4 Evaluation Results

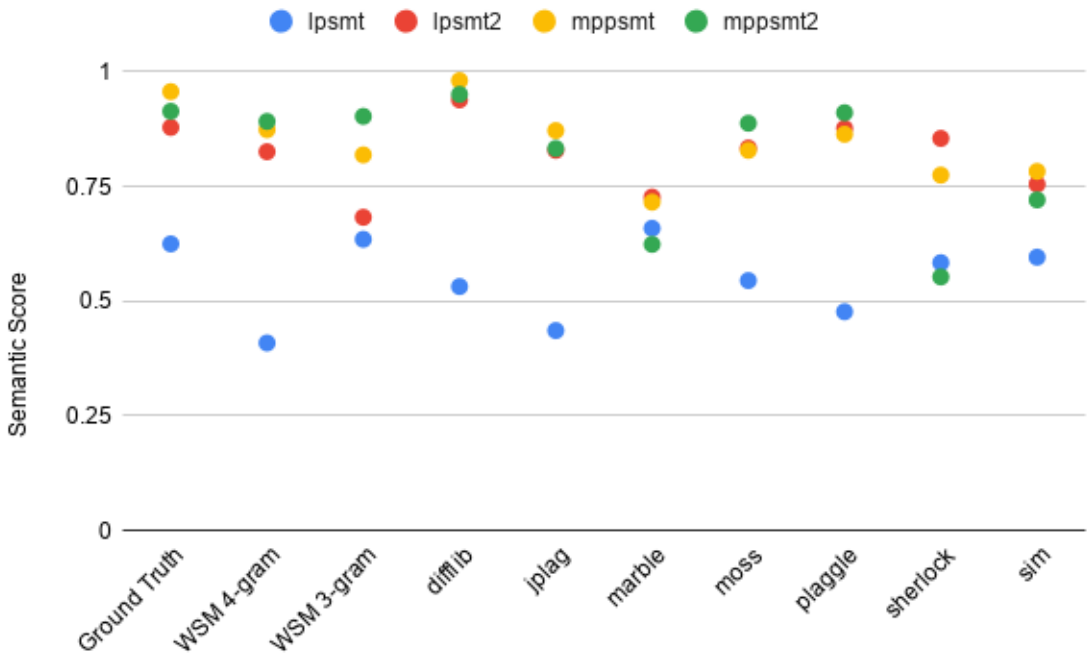


Figure 3.2: The Semantic Scores for Each Method on SMT Code Translation Datasets

We evaluated the performance of the highest results by running SMT-based code translation datasets through different code plagiarism detection tools (CSM 4-gram, CSM 3-gram, difflib, JPlag, MOSS, marble, plaggie, Sherlock and Sim).

We compared the scores of CSM and each code plagiarism tool with human judgement using semantic scoring. The results are shown in Figure 3.2. First, it should be clear that difflib (a Python diffing library) scored the closest to ground truth in all SMT-based code translation datasets. Difflib had the highest average accuracy of all tools. The reason may be that most code translation cases in datasets are easily detected; that is to say, they contain at least some identical parts. Sometimes, some of these tools may be fooled by peers who only have formatted characters (such as spaces, newlines and braces), just like in the mppSMT dataset.

Table 3.2: Average Precision Scores on Source Code Translation Datasets

Dataset	lpSMT	lpSMT2	mppSMT	mppSMT2	Average Scores
ground truth	0.645	0.879	0.957	0.914	0.849
CSM 4-gram	0.409	0.826	**0.874**	**0.892**	0.750
CSM 3-gram	**0.635**	0.683	0.819	**0.903**	0.760
difflib	0.532	0.939	**0.981**	0.951	**0.851**
JPlag	0.436	0.829	**0.872**	0.833	0.743
marble	**0.659**	0.727	0.716	0.624	0.682
MOSS	0.545	**0.834**	0.829	0.888	**0.774**
plaggie	0.477	**0.877**	0.864	**0.911**	**0.782**
Sherlock	0.584	**0.855**	0.775	0.553	0.692
Sim	**0.596**	0.755	0.783	0.721	0.714

* **Bold** font indicates the top three results when compared to the ground truth.

The proposed CSM method achieved strong performance on the majority of the datasets when N = 4, but not on lpSMT. Compared with other popular plagiarism tools, such as JPlag and MOSS, it achieves an average score closer to that of the human judgement. Table 3.2 shows that CSM 4-gram was the best performer when evaluated using mppSMT and mppSMT2.

The learning similarity model of the CSM method at N = 3 has not been significantly improved, and other datasets, except lpSMT, are not ideal. From this, we can conclude that the 3-gram model has not been well extended to these data sets. By contrast, JPlag, marble, Sherlock and Sim seem to be relatively weak, while difflib, MOSS and plaggie perform relatively well.

3.5 Discussion and Future Work

At present, our method has limitations. CSM uses N-gram sequence features and cosine similarity to judge the accuracy of translated code. There is no training process for the dataset before running. This indicates that the method only understands the importance of some similarities at runtime, based on the features in the dataset. This means that when the dataset is small, the distribution of some features may not represent the true distribution of tasks.

Future research directions may consider building a simpler monitoring model using deep learning and multiple similarity metrics. The model must understand the importance of each similarity metric. Another interesting direction is to use unsuper-

vised learning algorithms on multiple source code translation datasets and then use these representations as features of the supervised model.

3.6 Summary

In this chapter, we proposed CSM and compared its performance with the benchmark plagiarism detection tools. We validated using four SMT-based datasets from two sources, i.e. lpSMT, lpSMT2, mppSMT and mppSMT2. The N-gram model, based on TF-IDF weighting and cosine similarity, solved this problem well and achieved high scores on different datasets. TF-IDF's weighted features can also be used as part of the visualization method, because more unique parts of a word pair will show more significance than other similarities. To further improve the accuracy of code plagiarism detection, more empirical study is required. Overall, further improvement of CSM in the future will bring more reliable results.

Chapter 4

Automatic Graphical User Interface Generation with a Domain-Specific Language Model

4.1 Introduction

When people communicate with machines, the user interface (UI) is an indispensable tool [57]. Most user-oriented software applications rely on an attractive graphical user interface (GUI) to attract customers and facilitate the effective completion of a computing task [57]. When developing any GUI-based application, an important step is to draft and prototype design models, which help the UI instantiate to evaluate or prove abstract design concepts. In large-scale industrial environments, this process is usually accomplished by professional designers who have expertise in this field and can use image-editing software, such as Photoshop [74] or Sketch [29], to generate attractive, intuitive GUIs. After the initial design drafts are created, it is important to faithfully translate them into code so that the end user experiences the design and expected form of the user interface can be achieved (Figure 4.1).

Figure 4.1: A Web UI Design Workflow

Previous studies have shown that this process (usually involving multiple iterations) is challenging, time-consuming, and error-prone [133], especially if the design and implementation are performed by different teams [123]. In addition, UI teams usually adopt an iterative design process to collect feedback on GUI effectiveness at early stages. It is best to use prototypes because more detailed feedback can be collected; however, using current practices and tools is often too expensive [126]. In addition, past work on detecting GUI design violations in mobile applications has emphasised the importance of this issue from an industrial perspective [123]. Instead of spending scarce time and resources on iterative design and user interface coding, it is better to choose an accurate automation method. This will enable smaller companies to focus more on features and values rather than turning design into operational application code. Given the setbacks faced by front-end developers and designers in building accurate GUIs, automation support is clearly needed.

To help ease this process, some modern IDEs, such as Xcode [75], Visual Studio [117], and Android Studio [55], provide built-in GUI editors. However, recent studies have shown that using these editors to create complex, high-performance GUIs is cumbersome because users are prone to introducing errors and demonstration failures, even with simple tasks [200]. Other business solutions include collaborative GUI design and interactive preview design on target devices or browsers (with limited functionality using custom frameworks), but none provides an end-to-end solution that automatically converts mockups. Obviously, a tool that can partially automate this process could significantly reduce the burden of the design and development process.

Beltramelli [14] described an important development in this area called pix2code, explaining how deep learning can transform screenshots of a GUI created by designers into computer code. Since pix2code is focused on GUI layouts, graphical components and their relationships, the actual text value of the tag is ignored, and the resulting text portion is replaced with a specified number of random letters.

The aims of this chapter are two-fold: first, to design a framework based on pix2code that can automatically generate a specific platform code for a given GUI screenshot as an input. Second, to investigate the performance metrics used in domain-specific language evaluation. We hypothesize that the extended version of this method may reduce the time for manual GUI coding processes.

4.2 Related Work

Models and prototypes are used to collect feedback at the beginning of the design process. They help improve visual design and are meant to be used by design teams as communication tools to focus on the final appearance and solve layout problems for websites or applications [19]. The problem of automatically generating computer programs from a given specification has been studied since the early days of artificial intelligence (AI) [38]; however, recent research has focused on the possibility of generating source code from design models to save developers from labour-intensive and repetitive parts of the design process. As a result, deep learning applications are being explored as a potential solution to this problem. Because computing power is the biggest obstacle to front-end development automation, deep learning has been applied to the design model field. User interface code development for applications is a cumbersome and expensive practice, and users expect mobile and computer user interfaces to be highly customised and optimised for the specific tasks at hand [135]. A gap has been observed during production, and the conversion of user interface concepts to a working user interface code is done manually by programmers in a cumbersome, error-prone and expensive manner [133].

Many of the approaches discussed so far have relied on domain-specific languages (DSL; languages for specialised domains that are more restrictive than full-featured computer languages). The use of domain-specific languages limits the complexity of programming languages that need to be modelled and reduces the size of the space to be searched [14]. Betramelli's pix2code achieved 77% accuracy on three different platforms (iOS, Android, and web-based technologies) by using deep learning to train the model's code and automatically generate a single image end to end [14]. The authors believe that this is the first attempt to solve the problem of generating GUI code from visual input by using machine learning to understand potential variables rather than complex problem-solving engineering [14]. The paper further states that UI components are synthetically generated, but the author does not offer a way to generate DSL code.

Another related work is a project developed by Emil Wallner [1], which is another Keras-based implementation of pix2code, using the same dataset. It differentiated itself from pix2code by replacing the pre-trained image features with a light convolutional neural network (CNN). Instead of using max-pooling to increase information density, it increased strides. Lee et al. [103] also made improvements on the basis of pix2code. Unlike the single end-to-end pix2code model, their system followed an

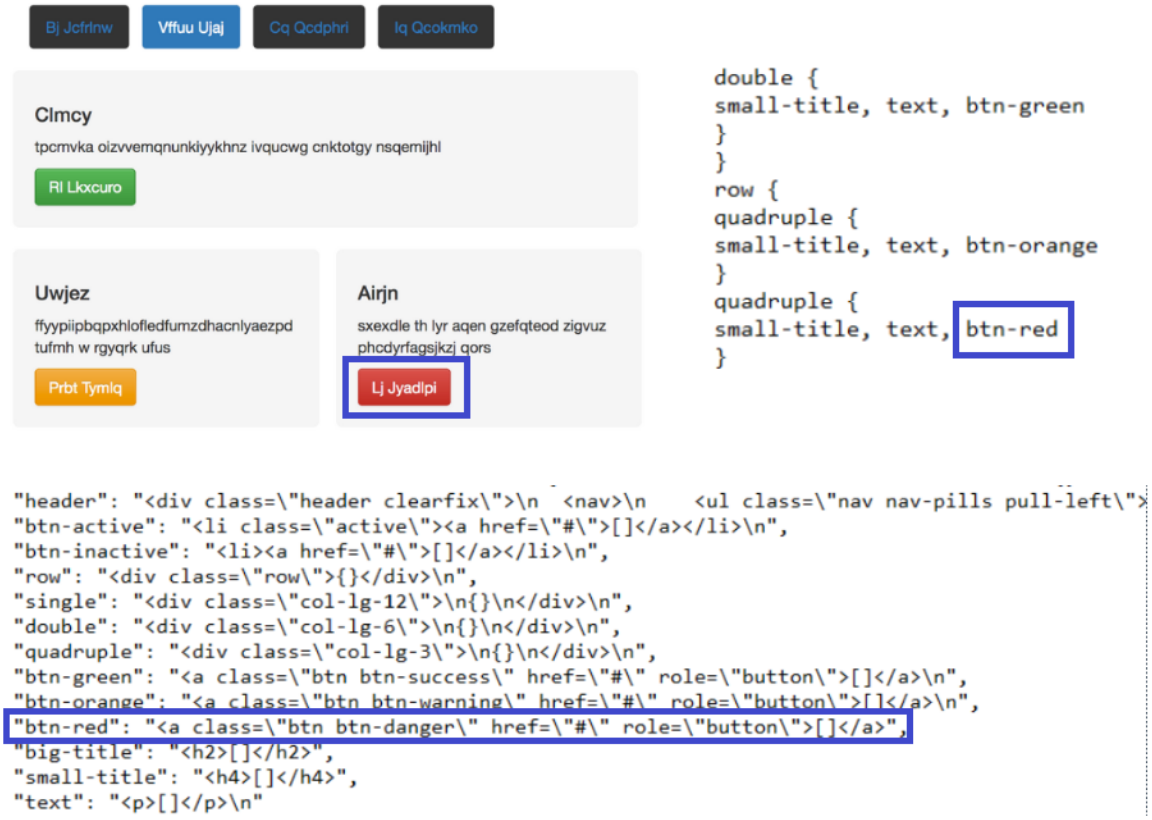


Figure 4.2: The Web DSL Token Mapping from Pix2code

image-captioning model previously created for PyTorch, with an encoder CNN and a decoder Recurrent neural network (RNN). As a whole, the system takes a screenshot as input and outputs a sequence of indices (based on DSL's vocabulary), which are then converted into valid HTML.

4.3 Approach

4.3.1 Model Architecture

As the framework of pix2code is based on Vinyals's image captioning model [177], the first input of Long short-term memory (LSTM) comes from the feature vector extracted by CNN. The feature vector of the image is the first input of the LSTM, and its information is captured in the hidden state of the LSTM. This can cause some of the information in the feature vector to be discarded as the length of the caption increases, thereby affecting the overall performance of the model [88].

To solve this issue, we redesigned the pix2code model's framework (Figure 4.3). The LTSMs are replaced by gated recurrent unit (GRU) [26] to improve the training

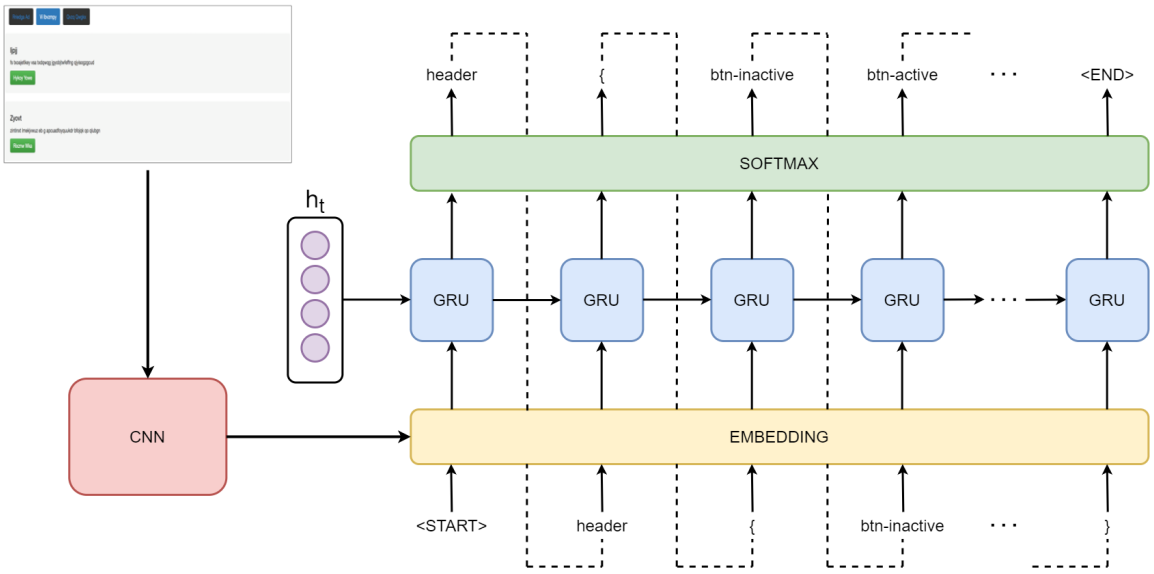


Figure 4.3: Our Proposed Framework Based on Pix2code [14]

rate, and the CNN's feature vector connects to the GRU's input as an embedded input to the GRU. Therefore, in theory, the model can capture all the future vector's information during the DSL token generation process.

Referring to the pix2code dataset, we preprocessed the data by resizing the input image to 256 × 256 pixels without retaining its aspect ratio and then normalised the pixel values. We used VGG16, VGG19 [164] and ResNet34 [63] as the encoders for our experiment. We adjusted the embedded size of GRU to 50, with 3 layers and 256 hidden units as a decoder.

4.3.2 Training and Sampling

When training the model, we divided an input into an image and its DSL token sequence, the label of which was the next token in the DSL file. The model compares its next token prediction with the actual next token prediction using a cross-entropy loss function.

During sampling, the image is processed through the CNN network, but text processing is only the seed of the starting sequence. In each step, the model's prediction of the next token in the sequence is appended to the current input sequence and entered as a new input sequence. This process is repeated, beginning with the <START> token, until the model predicts an <END> token or the process reaches a predefined limit on the number of tokens in each DSL file (Figure 4.4). After the

```
<START> header
<START> header {
<START> header { btn-inactive
<START> header { btn-inactive ,
<START> header { btn-inactive , btn-inactive
<START> header { btn-inactive , btn-inactive ,
<START> header { btn-inactive , btn-inactive , btn-inactive
<START> header { btn-inactive , btn-inactive , btn-inactive ,
<START> header { btn-inactive , btn-inactive , btn-inactive , btn-inactive
<START> header { btn-inactive , btn-inactive , btn-inactive , btn-inactive ,
<START> header { btn-inactive , btn-inactive , btn-inactive , btn-inactive , btn-inactive
<START> header { btn-inactive , btn-inactive , btn-inactive , btn-inactive , btn-inactive }
<START> header { btn-inactive , btn-inactive , btn-inactive , btn-inactive , btn-inactive } row
<START> header { btn-inactive , btn-inactive , btn-inactive , btn-inactive , btn-inactive } row {
<START> header { btn-inactive , btn-inactive , btn-inactive , btn-inactive , btn-inactive } row { quadrupl
e
<START> header { btn-inactive , btn-inactive , btn-inactive , btn-inactive , btn-inactive } row { quadrupl
e {
<START> header { btn-inactive , btn-inactive , btn-inactive , btn-inactive , btn-inactive } row { quadrupl
e { small-title
<START> header { btn-inactive , btn-inactive , btn-inactive , btn-inactive , btn-inactive } row { quadrupl
e { small-title ,
<START> header { btn-inactive , btn-inactive , btn-inactive , btn-inactive , btn-inactive } row { quadrupl
e { small-title , text
```

Figure 4.4: An Example of DSL Token Prediction

model generates the predicted token, the compiler converts the DSL token into HTML code that can be rendered in a web browser.

4.4 Experiment

4.4.1 Datasets

Beltramelli's pix2code dataset [14] contains 1,750 screenshots of synthetically generated websites and their associated source code, which are used as training features. Because each site generated in the dataset comprises only a few simple bootstrap elements (such as buttons, text boxes and divs), the 'vocabulary' of the model is limited to these features. However, this approach could be generalised to a larger vocabulary by increasing the number of elements. The DSL file is compiled with reference to the code in JSON format. The source code for each example comprises tokens in a DSL file, with each token corresponding to a piece of HTML code, as illustrated in Figure 4.2. The compiler is used to convert the DSL file into working HTML code.

4.4.2 Evaluation Metrics

In automatic translation evaluation, bilingual evaluation understudy (BLEU) [137] is an algorithm that must be mentioned. The basic assumption of BLEU is that if there are more N-grams to be co-produced with the reference translation, the more similar the description, the higher the quality of the translation. By counting the number of

```
1-gram: <START> @ header @ { @ btn-inactive @ } @ row @ { @ double @ { @ btn-orange @ } @ <END> (11/12)
2-gram: <START> header @ { btn-inactive @ } row @ { double @ { btn-orange @ } double @ { text @ } double @ { } @ } <END> (9/10)
3-gram: <START> header { @ btn-inactive } row @ { double { @ btn-orange } double @ { btn-orange } @ }  <END> (4/6)
4-gram: <START> header { btn-inactive @ } row { double @ { btn-orange } double @ { text } double @ { } } <END> (4/5)
```

Figure 4.5: An Example of MBLEU Score in DSL Tokens

co-occurring N-grams and adding a penalty factor to short sentences, the translation of the same topic can be evaluated through a reference translation. Since the field of code metrics has recently emerged, there is no corresponding method to measure the accuracy of DSL at this stage. Therefore, we designed a modified BLEU score (MBLEU) to evaluate the model. The formula to calculate the MBLEU score is as follows:

$$MBLEU = BP \cdot \exp(\sum_{n=1}^{N}(w_n log P_n) \tag{4.1}$$

where BP is a penalty factor for translations whose length is less than the reference value:

$$BP = \begin{cases} e^{1-c/r}, & \text{if } c > r \\ e^{1-r/c}, & \text{if } c \leq r \end{cases} \tag{4.2}$$

and P_n is the N-gram matching rate.

Take a sequence of tokens in a DSL file as an example (Figure 4.5). MBLEU divides sentences from one to four token sequences into four N-grams. In the prediction below, 'btn-orange' is a false prediction, and the actual correct token should be 'btn-green'. When $N = 4$ (4-gram), w_n will be 1/4 or 0.25. Then, the MBLEU score will be (11/12)$\times$ 0.25 + (9/10) $\times$ 0.25 + (4/6) $\times$ 0.25 + (4/5) $\times$ 0.25 = 0.22 + 0.22 + 0.16 + 0.2 = 0.8.

The sum also needs to be multiplied by the penalty BP of a sentence's length. In the 3-gram example, the token length is outside the measurement range, so the BP equals 1, and the result of the above becomes our final score. If MBLEU gets a score of 1.0, the correct elements will be in the correct position of the given source image. The lower the score, the greater the difference between the generated DSL token sequence and the real result, and the decoded HTML code will be different from the input sketch.

We compared the experimental scores of our model on MBLEU and BLEU-N [137] with the model from pix2code [14] in three different CNN types (Model-A: VGG16, Model-B: VGG19 and Model-C: ResNet34). Each score on methods ranges from 0 to 1, and a higher value gives a more accurate result for GUI code generation (Table 4.1). The results of evaluation metrics may change with further experiments in the future.

Table 4.1: Evaluation Results from the Metrics of Pix2code Dataset

Methods	CNN Types	BLEU-1	BLEU-2	BLEU-3	BLEU-4	MBLEU
Pix2code Model	VGG16	0.527	0.452	0.397	0.322	0.79
Model-A	VGG16	0.535	0.463	0.404	0.337	0.82
Model-B	VGG19	0.556	0.485	0.417	0.346	0.88
Model-C	ResNet34	**0.577**	**0.502**	**0.452**	**0.376**	**0.93**

4.5 Discussion

Through the experimental results for the pix2code dataset, we obtained a higher score when compared with pix2code model in the same CNN type (Model-A in Table 4.1). Our models also delivered the best performances on ResNet34 when compared with the other two CNN types (VGG16, VGG19).

At present, the model still has some limitations, which illustrate the following possible follow-up steps:

- Due to the limitations of the existing pix2code dataset, the model only trains a vocabulary of 16 elements, so it can only predict the DSL token specified in the data.
- Because CSS lacks style changes when making sketches part of existing datasets, there is still a significant difference compared to hand-drawn sketches.
- There are some shortcomings in the NLP evaluation metrics. The existing methods lack the ability to judge dependency relationships between different DSL tokens and their importance to the overall web page. It is necessary to improve the penalty factor or find a suitable alternative.

Existing work in the area of automatic GUI generation is still in the early stages of development; models such as Beltramelli's [14] have so far contained only a few parameters and have been trained on small datasets. There is further scope to focus on the more limited areas of a web-based GUI that do not require data synthesis. Because a large number of websites are already available online, and because new websites are created every day, this situation provides almost unlimited training data to extend deep learning methods and transform web-based design models into HTML/CSS code [14].

4.6 Summary

This chapter proposed a modified framework for solving the problem of feature vector losses in pix2code framework. The preliminary experimental results outperformed state-of-the-art methods based on BLEU and MBLEU. We demonstrated MBLEU is suitable for DSL evaluation but it is inconclusive due to a lack of datasets for further evaluation.

The next step could be attempting to create more elements to generate additional web examples, such as actual texts, images, drop-down menus, forms and bootstrap components. With increasing computer hardware performance, it is better to create a dataset that can be directly trained by HTML/CSS code than a DSL token sequence. A good way to generate more variants in hand-drawn sketch data might be to create a realistic hand-drawn website image using a generative adversarial network (GAN).

Chapter 5

Data Augmentation on Graphical User Interface Generation

5.1 Introduction

The automated generation of graphical user interface (GUI) design has recently become the subject of extensive research [98],[161]. Notably, deep learning technology has become a key method for promoting advancements in this field [120],[106]. However, this method usually requires a large amount of data as support, which is a difficult problem that cannot be avoided in the implementation of automated GUI generation. Presently, it is very difficult to obtain new data directly by using related techniques, such as object recognition in the user interface. Once an image is obtained, it must be manually annotated and classified according to its layout, and these are time-consuming tasks that professionals in the field have to perform accurately [183]. In addition, the existing GUI data sets lack standards and diversity in the classification of data objects. For example, the Rico dataset does not effectively classify the functional categories of GUIs, and there is a lack of interactive GUIs such as input forms and search pages. The distribution of data objects is lack of balance, which is too complicated in some GUIs and too simple in others [36].

Data augmentation is one of the effective methods for solving the problem of limited data at this stage [162],[163]. This technology enables the transformation of generating new training samples from an original dataset without changing the data category. The method has been successfully applied in many situations in the field of image processing, such as image classification, object recognition, semantic segmentation, and information retrieval. However, existing augmentation methods cannot be directly applied to relevant GUI design datasets due to varying data structures. In the case of natural images, the objects are usually invariant to orientation. Here,

operation methods, such as rotating, tilting, and changing colors, can be used to achieve the effect of data augmentation. In the case of a GUI, the objects must be arranged in an orderly manner in accordance with the layout design specifications [68]. Therefore, there are only a limited set of operations applicable in this domain.

In this chapter, we introduce a new GUI layout data augmentation method that directly synthesises a set of graphic elements in the layout. In this model, we use the Rico dataset [36] as the reference for the training data and pre-specify a set of fixed element categories (for example, 'text', 'button', or 'image'). In our network, each element is defined by its category probability and geometric parameter representation, which are bounding box keys. The generator takes the random sampling probability of graphical elements and geometric parameters as input and arranges the chosen elements in the layout. The output includes the deterministic category and geometric parameters of the design elements, which have been chosen according to the sampling probability. The generator has the function of replacing invariants. If the input elements are re-ordered, this function will generate the same layout.

For this structured data, we implement a two-stage operation mode. The first stage directly acts on the category probability and geometric parameters of the element with size adjustments. Although effective, it is not sensitive enough for misalignments and spacing in terms of exact pixels between elements. In the second stage, we propose the user interface generative adversarial network (UIGAN) for the generator based on work in the field of vision. Just as a person can judge a design by looking at rasterised images, by mapping different elements onto a two-dimensional layout, we can evaluate their relationship with the specific methods. The models can then be used for layout optimisation because they are specifically utilised to distinguish visual patterns including, but not limited to, image segmentation and occlusion. However, the key challenge is how to map the geometric parameters to pixel-level layouts. One method that can be applied is a spatial transformation network to decompose the graphic elements into bitmap masks [80].

We evaluate our method by generating a GUI layout from the frame of the markup. In summary, our model has the following contributions: (1) a differentiable wireframe generator that can determine the alignment based on the arrangement of the discrete elements and (2) UIGAN based on the generative adversarial network (GAN) [11] that directly creates structured data, which is represented in the GUI design as a set of resolution-independent markup graphic elements.

5.2 Related Work

5.2.1 Model-Based User Interface Automatic Generation

Early user interface generation methods were mainly model based. Modelers need to fully understand the whole system and define multiple models, including task, user, presentation, session, and platform models. The models can be represented by highly professional tags to make the interface easier to create and maintain.

Puerta [143] describes a model-based interface designer (MOBI-D), a comprehensive environment that supports user-centered design through model-based interface development. To solve the problems of transitioning from scenario to formal specification and unclear UI code generation, Elkoutbi et al. [43] proposed a requirements engineering method that generates a user interface prototype from the scenario and formal specification of the application. To model interactive operation objects and realise the cooperation between interactive objects and domain objects, Silva et al. [35] designed the unified modeling language for interactive applications (UMLi), an extension of UML that provides support for UI design.

Generating graphical user interface code using machine learning technology is a relatively new research field [18], [21]. DeepCoder is a system that can generate computer programs by using statistical prediction to enhance traditional search technology [9]. In this work, the author defines a programming language with sufficient expressiveness, including real-world programming problems. It can be predicted from input and output examples and obtains a model for mapping input and output example sets to program attributes. Experiments were carried out showing an order of magnitude in acceleration compared with standard program synthesis technology. This makes it possible to use this method to solve similar problems to the simplest when programming competitive websites.

In Gaunt et al.'s [49] research, the source code can be generated by learning the relationship between input and output examples through a differentiable interpreter. The author's goal here was to develop a new machine learning method based on a neural network and a graphical model and understand the ability of machine learning technology relative to traditional alternatives, such as the constraint-solving method based on the programming language community. The main contribution here was the proposal of TerpreT, a domain-specific language (DSL) used to express program synthesis problems. TerpreT is similar to a probabilistic programming language: the model consists of a specification for program representation (declaration of random variables) and an interpreter that describes how the program maps input to output

(connecting unknowns to the observed model). The reasoning task involves observing a set of input and output examples and inferring the underlying program.

In addition, Ling et al. [107] demonstrated program synthesis from mixed natural language and structured program specifications as input. It is worth noting that most of these methods rely on a DSL (such as a markup, programming, or modelling language). They are designed for a specific domain, but they are usually more restrictive than a fully functional computer language. Therefore, the use of a DSL limits the complexity of the programming language to be modelled and reduces the search space size.

There is less work to generate code through visual input—examples include hand drawing and UI screenshots. The pix2code project [14] was the first attempt to solve the problem of user interface code generated by visual input by understanding potential variables through a machine learning method rather than complex engineering heuristics. The author first generates a DSL from the prototype diagram and then compiles the DSL into source code. The author uses the design prototype map and the DSL context as training data, a convolutional neural network (CNN) to obtain image features, and two long short-term memory (LSTM) networks to understand the basic laws of the DSL contexts and the relationship between a DSL and a corresponding prototype map. On the whole, pix2code performs well, but there are some limitations, such as the need to formulate the code length range in advance and that pix2code does not consider the GUI hierarchy and code structure.

The attention-based layered decoding model of Zhu et al. [209] improved pix2code. The author proposed an attention-based code generation model, which can more finely describe GUI images and generate layered structured code consistent with the hierarchical expansion of GUI graphic elements. In addition, all the components can be extracted separately for end-to-end joint training. The experimental results show that the author's method had obvious better performance compared with the original pix2code in a public GUI code dataset and their own dataset.

Nguyen et al. [133] first proposed the technology of automatic reverse engineering of mobile application user interface (REMAUI). REMAUI automatically infers the source code of a mobile application user interface from a screenshot or conceptual design diagram of the user interface. On a given input bitmap, REMAUI identifies user interface elements through computer vision and optical character recognition (OCR) technology, infers the appropriate user interface hierarchy, and exports the results as source code for compilation and execution. The experimental evaluation results show that the UI generated by REMAUI was similar to the original UI at

the pixel level and the UI hierarchy at its runtime. However, REMAUI also has limitations. First, it does not support the classification of detected components into their local component types but uses the binary classification of text or images, which limits the practical applicability of this method; second, from a developer's point of view, it is not clear whether the GUI hierarchy generated by REMAUI is really useful because the GUI hierarchy is not evaluated.

Moran et al. [122] proposed redraw based on REMAUI. In contrast to other methods, redraw is not specific to any particular field. It uses data-driven methods to classify and generate a GUI hierarchy, can use a CNN to classify GUI components into their own types, and can use a data-driven iterative k-nearest neighbours (KNN) algorithm combined with computer vision technology to generate a GUI hierarchy.

5.3 Methodology

5.3.1 Basic Layout Manipulations

The first stage of the data augmentation task includes specifying a set of transformations so that the image classification problem is considered constant, including the X-shrinking and the zoom adjustment; the transformations do not change the image category. The X-shrinking refers to the proportional shrinkage of the width of each component in each GUI layout (Figure 5.1). Zoom adjustment is scaling the whole of each component in the GUI layout with the centre coordinate as the base point (Figure 5.2). With these two methods, the number of Rico datasets can be doubled each time they are scaled by a certain percentage (e.g., 5%, 10%). It should be noted that the image enhancement technique depends on the problem, and certain transformations should not be applied.

5.3.2 Layout Generation with GAN

The second stage is based on layout generation, in which all design elements are arranged in an appropriate size and position according to their content-based attributes (such as area, aspect ratio, and reading order). We use a generative adversarial network (GAN) to automatically generate the layouts of the Rico dataset.

A GAN [90] can create data similar to original data through the complex operation of a neural network. An excellent GAN can closely imitate the characteristics of the original data. The accuracy of its output directly affects the results of subsequent

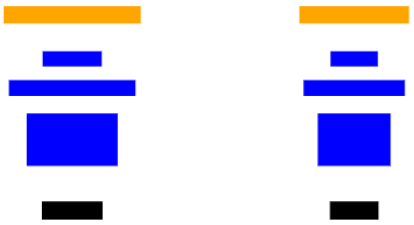


Figure 5.1: Transformation of X-shrinking

Figure 5.2: Transformation of zoom adjustment

research on behaviour recognition, an outcome that is very significant to follow-up work.

GANs are widely used in various fields [2]. However, they remain very rare in the field of GUI dataset generation. By analysing the characteristics of the layout structures in the Rico dataset, we find no great difference between the layout structure data and sequence data. Thus, GANs are also suitable for the expansion of the GUI datasets. For a generating process, a special network structure is necessary to generate satisfactory layout data.

After the selection of the generator and discriminator, we propose UIGAN, a modified GAN framework especially for GUI layout generation. The structure of UIGAN proposed in this chapter is shown in Figure 5.3.

As seen in the architecture of UIGAN, in addition to using a non-traditional recurrent neural network (RNN) [52] and a convolutional neural network (CNN) [160], the discriminator and generator must also be connected to the fully connected layer (FC). The generator mainly consists of three layers of RNN, which are used to generate characteristic data with time correlations. Each layer of the RNN includes 3n RNN nodes, where n represents the total number of component points in each frame of the layout data. The component points in the layout are obtained from the coordinate values of each vertex of the bounding box. It is observed that there are two fully connected layers, with each FC still containing 3n nodes.

The fully connected layer is used to further generate layout data. Through the

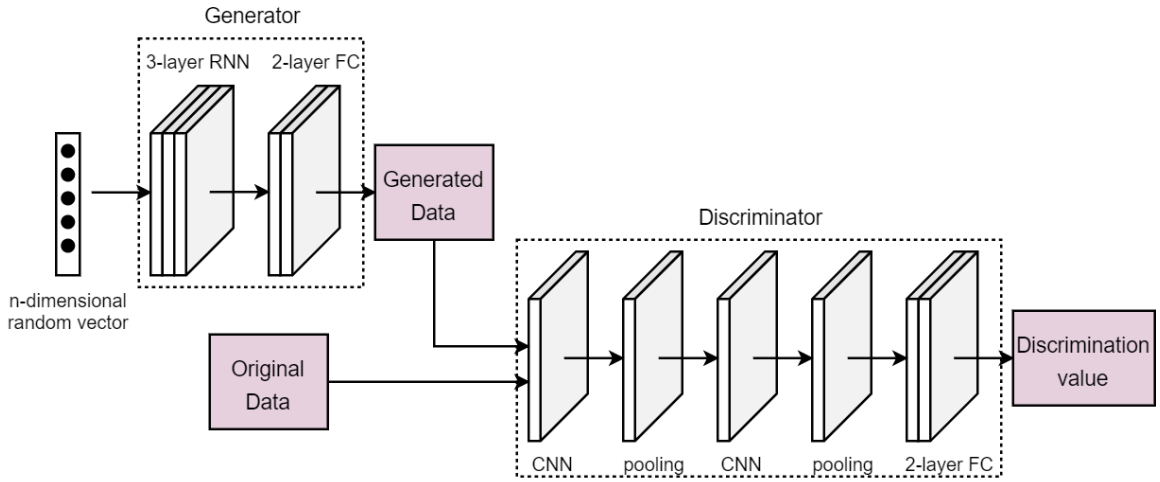


Figure 5.3: Architecture of UIGAN

fully connected layers, data with the same dimensions and time correlations with the original data can be generated. Through training, the sequence of the layout data that meets the needs can then be generated.

The discriminator is mainly composed of the CNN, pooling layer, and FC. Each CNN layer includes a 3×3 convolution kernel. The pooling layer adopts maximum pooling. The number of nodes in the FC of the first layer was set to 13, and the number of nodes in the FC of the second layer was set to one. The main task of the CNN is to extract the matrix's shape features. Because the layout data sequence is essentially the same as the image data, the trained CNN can effectively analyse the difference between the generated data and the original data. The three-layer FC's objective is to convert the extracted features into identification values to determine the data's authenticity.

5.3.3 Training of UIGAN

After preprocessing the dataset, it is also necessary to initialise the GAN's parameters. Generally, all parameters are either assigned a value of zero or conform to a normal distribution. After all initialisation work is completed, model training can be started.

The training process of UIGAN alternates between training the generator and training the discriminator [26]. In training the generator, one must ensure that the discriminator's parameters remain unchanged. In a similar vein, the process of training the discriminator requires the generator's parameters to be kept constant. This process is an iteration, and each iteration includes two forward propagations and two backpropagations.

In training the generator, a piece of data is selected from the dataset (that is, layout data is extracted, which is generally stored in the form of a matrix). The matrix's number of rows is the number of layouts, set as m, and n represents the total number of three-dimensional coordinate dimensions of all the component points from the layout. Thus, the data is expressed as follows:

$$I_{mn} = F(O) \tag{5.1}$$

where O is the original dataset, and F is the layout data selected from O.

After selecting an action, one must generally obtain the number of layouts m and the number of spatial coordinates n of the component points. To ensure that the generated data are as similar to the original data as possible, it is necessary to set the generated data matrix's number of cycle iterations to m. Doing so ensures that the number of rows and columns of the generated data matrix are equal to those of the original layout data.

Next, the system needs to generate a random vector set as z, input the random vector into the generator, and, after m iterations, generate a matrix with the same dimensions as the original data R according to the algorithm described above, which is called the pseudo layout sequence Z. This is expressed by the following:

$$S = G(Z) \tag{5.2}$$

The generated layout sequence S is then sent to the discriminator to generate an eigenvalue T as follows:

$$T = D(S) \tag{5.3}$$

Finally, the generator parameters are adjusted according to the following error formula:

$$L_g(S) = -\sum \log(D(G(S))) \tag{5.4}$$

The adaptive moment estimation (Adam) technique is used to optimise all the parameters of the generator. In the process of training the identifier, it is ensured that the generator parameters remain unchanged. Similarly, a piece of data in the real action sequence is selected, such as in Equation 5.1. After selecting an action, it is still necessary to obtain the number of layouts m and the number of spatial coordinates n of the component points. Similarly, the number of iterations of the generated loop is then set to m.

Through the method described above, the random variable z is obtained. After passing through the generator according to Equation 5.2, the pseudo layout data matrix is still obtained. It is then input into the discriminator to obtain the characteristic value T1 according to the following equation:

$$T_1 = D(S) \tag{5.5}$$

In a process different from the one utilised for training the generator, the original data R is input into the discriminator to obtain the eigenvalue T_2 as follows:

$$T_2 = D(R) \tag{5.6}$$

Adjust the parameters of the appraiser according to the error formula:

$$L_d(S, R) = -\sum(\log(T_1) + \log(1 - T_2)) \tag{5.7}$$

Finally, Adam is used to optimise all the parameters of the discriminator. The two processes are iterated continuously. When the discriminator can no longer identify the data created by the generator, the training is deemed complete.

After the training, the generator needs to be extracted from the UIGAN system, the required dataset expansion tool. The continuous input of random variables can generate a large number of datasets to supplement contexts with an otherwise insufficient number of GUI datasets.

5.4 Experiment

The computer used in the experiment had 32 GB of DDR4 RAM and 8 GB of graphics memory through an NVIDIA GeForce RTX2080 graphics card accelerator.

5.4.1 Dataset

An encoder can be trained by using the Rico dataset to understand the embedding of the GUI layout and add 64-dimensional vector annotations to each GUI to represent the encoded visual layout [36]. The vector representation is usually used to compute structurally (and typically semantically) similar GUIs and support an example-based dataset search. To create training input for the automatic encoder embedded with layout information, We constructed the layout for each GUI, captured the bounding box area of all the leaf elements in its view hierarchy, and distinguished text, images,

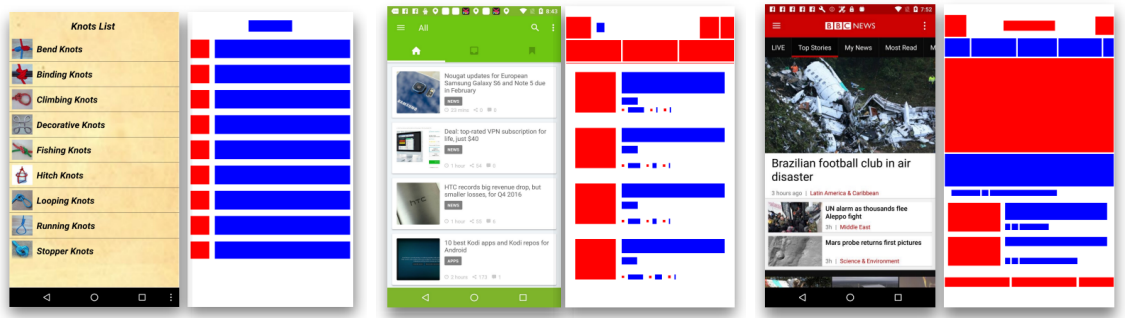


Figure 5.4: GUI layouts constructed for the Rico dataset

and buttons (Figure 5.4). Rico's view hierarchy eliminates the noise image processing or OCR technology normally required to create these inputs.

Because the Rico dataset is stored as a basic digital sequence, a layout sequence can be abstracted into a matrix and then further abstracted into a vector through the transformation method [36]. Therefore, the layout sequence can be transformed into a matrix or vector through abstract methods. When evaluating the similarity between two actions, one must only obtain the similarity of two matrices or vectors through mathematical methods.

5.4.2 Evaluation Metrics

Since the Rico dataset is stored as a sequence of basic numbers, each data object in the GUI has its vertex coordinates of the top-left and the parameters for its width and height. For a certain sequence of numbers, it can be abstracted into a matrix, and then through the conversion method, it can be abstracted into a vector. Therefore, a GUI data object sequence can be transformed into a matrix or vector by abstract methods. To compare the similarity between the generated GUI and the original GUI, we need to compare the coordinate parameters of each data object to judge the deviation of its position, and the length of width and height to judge the accuracy of its size.

At present, two evaluation methods were adopted:

The first method is Euclidean distance evaluation. This loss function is used for the similarity of matrices. Let the two matrices be A and B and the elements be a and b, respectively. The Euclidean distance can be expressed as follows:

$$L_e = \sum_{A,B} |a - b| \tag{5.8}$$

This method's main idea is to add the differences of numerical values to obtain the differences in the data and ignore the influence of other aspects. The advantages are intuitive and simple; the disadvantage is that significant calculation is necessary, especially for the square calculation of 1-norm. Because in Euclidean distance, the contribution of each coordinate is the same. When coordinates represent measured values, they often have random fluctuations of varying sizes.

We use cosine similarity to compensate for the defect of Euclidean distance. Cosine similarity measures the correlation between two vectors (x, y), which is defined as follows:

$$cos(x, y) = \frac{x \cdot y}{|x| \cdot |y|} \tag{5.9}$$

The calculation result is a decimal value from -1 to 1. Specifically, -1 means that the two vectors have opposite directions, 0 indicates that the two vectors are vertical, and 1 shows that the two vectors have the same direction.

This method describes the similarity of vectors in terms of vector direction. If the directions of the two vectors differ greatly, this will be reflected by this method. A combination of the two methods is used to measure the dataset generated by UIGAN from two angles and compare it with other semi-supervised and unsupervised algorithms.

Summarise the formula of cosine similarity and define the evaluation formula as follows:

$$E = \frac{L_e}{2 + cos(x, y)} \tag{5.10}$$

We have also tested other metrics such as Manhattan distance, Hamming distance, etc. We finally determined that, for the parameter setting of the Rico dataset, the combination of Euclidean distance and cosine similarity metric method can obtain the most accurate calculation results.

5.4.3 Experimental Results

Ideally, by the end of the UIGAN training, the generator and discriminator will have strong creation and identification abilities. The two form a state of confrontation: when the value of one loss function rises, the other will fall until the values balance.

The calculation results of the loss function are shown in Tables 5.1 and 5.2. By using Equation 5.5, increasing the iterations results in the identification of the original data converging to one, indicating that the identification ability is increasing.

Table 5.1: Calculation results of the Euclidean distance loss function

Number of Iterations	Original Data Loss	Generated Data Loss
0	0.81	0.82
2,000	0.97	0.84
4,000	0.98	0.8
6,000	0.99	0.85
8,000	0.99	0.86
10,000	0.99	0.86
12,000	0.99	0.86
14,000	0.99	0.87

Table 5.2: Calculation results of discriminator loss and generator loss

Number of Iterations	Discriminator Loss	Generator Loss
0	2.03	0.24
2,000	2.07	0.22
4,000	2.19	0.22
6,000	2.34	0.21
8,000	2.49	0.21
10,000	2.63	0.21
12,000	2.77	0.20
14,000	2.91	0.20

Table 5.3: Experimental results of GUI generation based on the Rico dataset

Data Object Category	Traditional GAN (E value)	UIGAN (E value)
Text	2,708.71	2,693.35
Image	2,723.54	2,634.47
Button	2,265.68	2,116.81

How to determine the loss result of the discriminator for the generator is indicated in Equation 5.5. From this, it can be seen that the result generated by the discriminator for the generator gradually increases and approaches one. Stated differently, the data produced by the generator becomes increasingly similar to the original data.

The change of the loss function of the discriminator in the training process corresponds to Equation 5.7, while the change of the loss function of the generator in the training process is represented by Equation 5.4.

It can be seen that the two trends begin at opposite ends of a range and finally converge, which fully reflects their confrontation operation. The changes in the above parameters reflect the training process of UIGAN. Under the described hardware conditions, the total time consumed by the whole training process was measured at about seven hours.

The process of traditional GAN—that is, the GAN model if the RNN module is deleted—is as follows: input the original data into the traditional GAN, use the GAN to generate data of a certain scale, then calculate the evaluation value of the dataset (as in Equation 5.10).

The experiment was carried out on the Rico dataset, and the results are shown in Table 5.3. Several layouts were randomly selected for the experiment. From the analysis of the Rico dataset experiment, it can be concluded that the similarity between the data generated by UIGAN and the original data was higher than that of the traditional GAN and the original data; that is, the RNN achieved the expected effect in the UIGAN system.

However, there exists a limited number of experiments and evaluation standards in related fields to serve as benchmarks in regards to judging whether the similarities between the dataset created by UIGAN and the original data were reasonable and well generated. Nevertheless, the experimental results returned an evaluation value smaller than that of traditional GAN, showing that the algorithm improved upon the GAN in the field of GUI generation to a certain extent. This could be attributed to the fact

that UIGAN can learn the dependencies between data sequences. The experimental results also show that UIGAN was able to generate similar GUI datasets.

5.5 Summary

The algorithm regularly deviates from the correct result of a GUI in generating a complex layout; thus, further improvements are necessary. On the whole, UIGAN performs well in experiments of dataset augmentation, going beyond initial expectations. Despite the lack of comparable experiments, the generated dataset meets the expected goal for GUI layout training and performs better than the traditional GAN based on the experimental results.

Future work will further classify the data objects in the existing datasets. For example, text objects and image objects in a GUI can be subdivided into 'TextView', 'ImageView', 'TextButton' and 'ImageButton' according to their functions. In addition, more categories can be extended, such as 'Headline', 'Icon' and 'Toolbar'. It can be seen that data expansion will have more potential to be tapped in the field of GUI generation.

Chapter 6

Scene Graph-to-UI Model for Graphical User Interface Layout Generation

6.1 Introduction

Graphical user interface design has become the subject of extensive research, and this includes its aesthetics [174]. Automatic GUI design allows users to complete tasks without expending unnecessary energy on the design itself. GUI design can also improve the usability of the interface. Designing a GUI presents a series of significant challenges. One of the most critical challenges is to represent abstract concepts in the visual language of graphics. GUI design should maintain a unified visual style and prioritize giving a distinct personality to each graphic [100].

Although significant progress has recently been made in the methods of generating authentic natural images, particularly in Graph Convolutional Networks (GCNs) [155], the current method of creating GUI design is much more primitive. This chapter proposes a scene graph to user interface (SG2UI) model that directly synthesises a group of graphic elements in the design. In this model, a fixed set of element classes (i.e., 'text', 'button', or 'image') are specified in advance. In the network of SG2UI, each element is represented by its class probability and geometric parameters. These are the keys of bounding box. The generator takes the graphical elements of random sampling class probability and geometric parameters as input and arranges them in the design. The output is comprised of the deterministic class probability and geometric parameters of the design elements. The generator has the invariant displacement function, which will generate the same layout if the input elements are reordered.

Figure 6.1: Our GUI generation model can support content aware layout generation. Given the input design category and keywords of the summary text content, it will automatically generate multiple layouts that conform to the visual and text content.

For structured data, we propose two discriminant networks. The first is structurally similar to a generator; it acts directly on the element's class probability and geometric parameters. Although effective, it is not sensitive enough to prevent dislocation and occlusion between elements. The second discriminator works in the field of vision. Just as one can judge a design by looking at rasterized images and mapping different elements into a two-dimensional layout, the relationship between them can be evaluated. However, the critical challenge lies in how to map geometric parameters to the pixel-level layout. One method entails decomposing graphic elements into bitmap masks using spatial transformation networks [80]. However, filling pixels in design elements can lead to occlusion and may have no effect on backpropagation, such as when small polygons are hidden behind large polygons. We evaluate the SG2UI model by generating GUI layouts from the marked boundaries.

Our proposed SG2UI model includes the following contributions: 1) The generator for directly synthesizing structured data is represented as a set of resolution-independent labelled graphic elements in the design; and 2) A differentiable wireframe render layer allows the discriminator to determine alignment based on the arrangement of discrete elements.

6.2 Related Work

6.2.1 Text-to-Image Generation

Image generation based on text description is the main research direction of the image generation model. If the model can realise the conversion from text to image, it shows that the model understands the image semantically. The GAN-INT-CLS model proposed by Reed et al. [149] is the first attempt to use the generative adversarial network (GAN) model to generate images from text sequences. By taking the text vector as the conditional input of the GAN model, the text-to-image generation is better realised, but it is mainly suitable for generating images with a resolution of 32×32 pixels. On this basis, Zhang et al. [201] proposed the StackGAN model, using two tandem GANs to generate an image with a resolution of 256×256 pixels. Reed et al. [150] proposed a GAWWN (learning what and where to draw) model for text-to-image generation based on position constraints. By capturing the positioning constraints of objects in the image, they learn the boundary box of objects in the image and improve the quality of the generated image. Zhang et al. [202] proposed the StackGAN++ model, which uses multi-pair generators and discriminators to solve the limitation of using only two pairs of StackGAN models, increase unconditional

image loss for the discriminator, and improve the ability of the discriminator. Xu et al. [193] proposed the AttnGAN model, introduced the attention mechanism based on the StackGAN++ model [192], matched different subregions of the image through relevant words in the natural language description, and proposed the deep attention multimodal similarity model (DAMSM) to calculate the loss between the subimage and the corresponding words, so that the text features have sufficient visual resolution.

The existing research on the text-to-image generation model is only applicable to image data sets containing a single object. When complex scene images containing multiple objects and relationships are encountered, the generated images will become chaotic. The reason is that text sequence is a linear structure, and it is difficult to transfer information between objects in text descriptions. Johnson et al. [86] proposed image generation from a scene graph (sg2im) model based on a scene graph. Compared with a text sequence, the input scene graph can more effectively represent the structural relationship between objects in the image and is more conducive to information transmission between objects. The sg2im model uses the graph convolutional network [94] to extract the features of the scene graph and generate an image containing multiple objects and relationships.

6.2.2 Sg2Im

The sg2im model takes the scene graph describing the object and its relationship as the input to generate a realistic image corresponding to the scene graph [86]. The model consists of a generator network and a discriminator network. The subnetworks of the sg2im model proposed in this chapter include the graph convolutional network (GCN), scene layout synthesis network (SLSN), cascaded refinement network (CRN) and discriminator network.

The GCN introduces the graph convolution idea proposed by Kipf and Welling [94] to process the model input scene graph in the way of spatial domain convolution, in order to obtain the abstract vector representation of each object in the graph, and the embedded vector aggregates the feature information of other objects in the graph. Each layer of the graph convolutional network obtains the abstract vector representation of the object and relationship in the graph by training three functions. The input of the function is the edge in the graph, and the output is the vector representation of the starting subject, relationship, and target object of the edge. Then, the final vector representation of all object nodes is obtained by an average pooling function. After stacking through multiple graphs, each final output object vector can aggregate the information of other objects along the edge of the graph.

The purpose of the SLSN is to synthesise a rough scene layout corresponding to the final generated image. The scene layout is similar to the semantic segmentation map of the generated image, which only contains object position information and edge contour information, but does not include object colour details. Therefore, in order to synthesise a scene layout with enough respect for facts, we need to predict the border and segmentation mask of each object. The sg2im model predicts the frame and segmentation mask of each object through the object layout network, including the box regression network and mask regression network. Finally, all object layouts in a single image are combined to obtain the scene layout of the whole image.

A cascaded refinement network is used to transform the synthetic scene layout into a generated image, which is similar to the inverse process of image semantic segmentation [112]. The network generates the final image from coarse to fine by gradually adding detailed information to the scene layout. CRNs are suitable for image generation with high resolution and high fidelity. Chen and Koltun [24] used a CRN to generate a photo-like real image containing hundreds of thousands of pixels.

Discriminator and generator network confrontation training can greatly improve the output of generator networks. The discriminator in the sg2im model consists of an image discriminator and an object discriminator. The image discriminator can improve the quality of the generated image. The object discriminator network ensures that the objects in the generated image are realistic enough. At the same time, an auxiliary classifier [136] is introduced into the object discriminator to ensure that the objects can be classified correctly.

Although the sg2im model generates complex scene images containing multiple objects and relationships, the quality of the final generated images is not high. Experiments show that only by using the object frame and segmentation mask provided by the real label to generate the image can we better distinguish the relationship between different objects in the image. The segmentation mask generated by prediction and the image generated by frame are relatively chaotic, indicating that the frame regression network in the object layout network cannot better locate the objects in the scene map, and the mask generated by the mask regression network cannot better represent the edge contour information of the objects. Therefore, we need to improve the object layout network to obtain a better object segmentation mask and frame.

6.3 Methodology

In this chapter, the SG2UI model is proposed to generate higher quality complex GUIs containing multiple objects and relationships. It is divided into two parts: 1) Feature extraction of the scene graph using a graph attention network to obtain object vectors with stronger expression ability; and 2) Improved mask regression network and frame regression network to obtain a more accurate object mask and frame, and synthesise a 2D scene layout closer to the generated GUI semantics.

6.3.1 Scene Graph Preprocessing

A scene graph is a way to sort data into a hierarchical structure [86]. Scene graph is usually used to describe the objects, attributes and relationships between objects in an image. It can be used for image retrieval, image generation, image/video action capture and special relationship detection [86].

Because it does not limit the types of objects, attributes and relationships that can be represented, this representation method can describe the visual scene in great detail. Figure 6.2 illustrates an example GUI scene graph, which shows that the object instance has been set to 'image, button and text' within GUI. Each component is represented using different colours. A relationship between components can be created by using 'above, below, left of, or right of'.

Given a set R of relationship types, we can define scene graph G as a tuple $G = (O, E)$, where $O = o_1, ..., o_n$ is a set of objects and $E \subseteq O \times R \times O$ is a set of edges, and n is the number of instance objects in the scene graph; $R = r_1, ..., r_m$ represents the relationship set between objects in the scene graph, and m is the number of relationships in the scene graph; $E = (o_i, r_1, o_j), ..., (o_p, r_m, o_q)$ represents all directed edges in the scene graph, and i, j, p, q represents the instance object label in the scene graph.

Embedding technology [25] is used to convert all objects and relationships in the scene graph into abstract embedding vectors. The object feature vector set is represented by $h_o = h_{o1}, ..., h_{on}$, and $h_r = h_{r1}, ..., h_{rm}$ represents the relational feature vector set. The feature dimension of the object feature vector set and the relationship feature vector is set to $F1$.

6.3.2 Feature Extraction of Scene Graph

The scene graph object layout network outputs the embedded vector expression of instance objects in the scene graph, and each vector aggregates the feature information

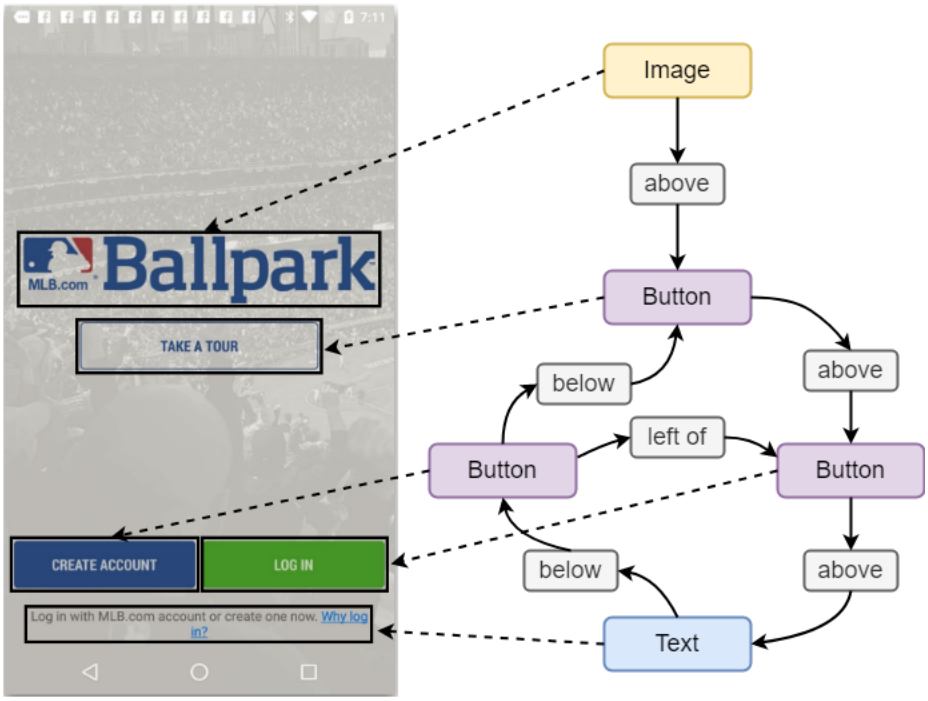


Figure 6.2: A Scene Graph Example in the GUI.

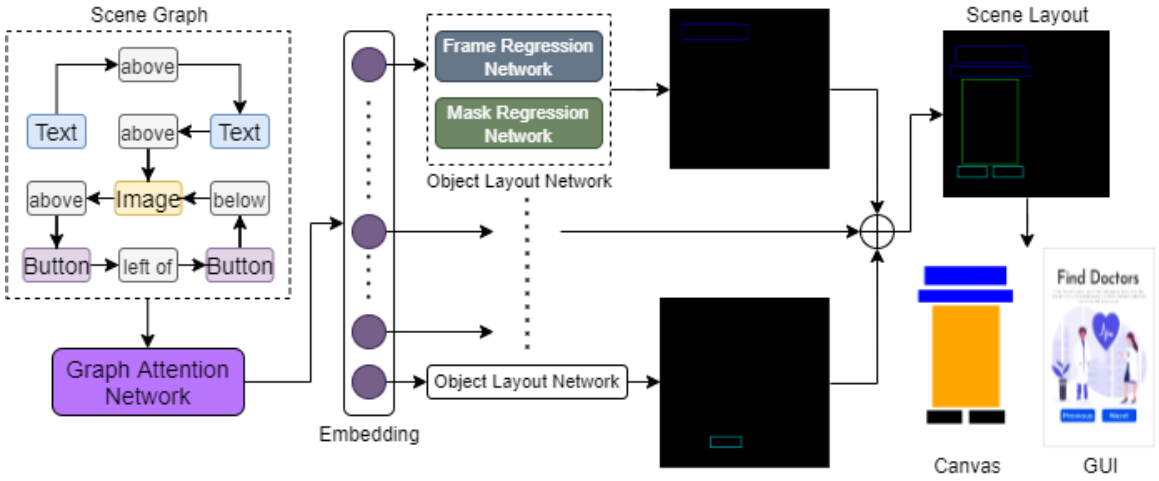


Figure 6.3: Architecture of SG2UI model.

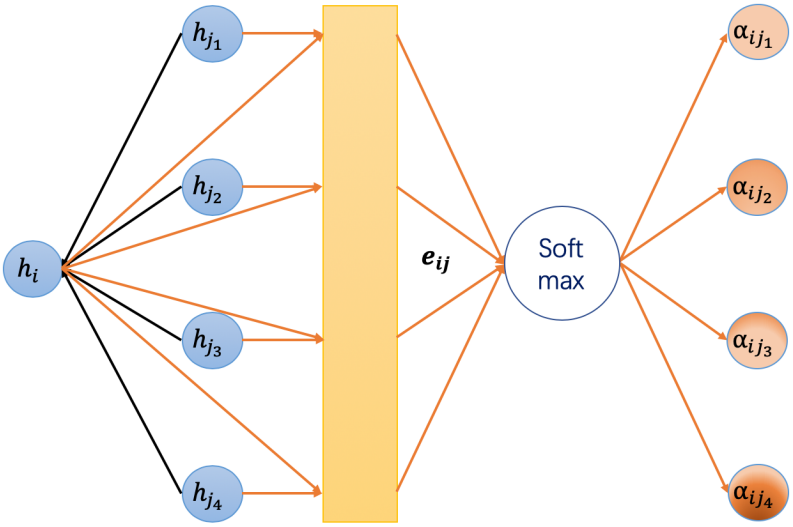


Figure 6.4: Architecture of graph attention network.

of all other objects and relationships [86].

In the SG2UI model, each layer converts the low-level feature vector representation corresponding to three elements in each edge (o_i, r_k, o_j) of the scene graph into high-level feature vector representation by training three learnable functions: g_s, g_p, and g_o. Therefore, the function inputs are the embedding vectors (h_{oi}, h_{rk}, h_{oj}) corresponding to the edges in the scene graph. By extracting the edge features in the scene graph, the information is aggregated along the edges of the scene graph. After multiple convolution layers, each final output object vector aggregates the feature information of all other objects and relationships.

In order to obtain more expressive object embedding vectors, a graph attention network is used as a scene graph feature extraction network. The graph attention network introduces an attention mechanism based on the GCN; that is, when each output eigenvector aggregates neighbourhood nodes, it assigns a learnable attention coefficient to all neighbourhood nodes, so that each object can have a different perception of all its neighbourhood nodes. The specific steps are as follows:

1) A shared parameter matrix $W \in R^{F_1 \times F_2}$ is used to convert all object vectors and relationship vectors in the scene graph into higher-level feature vectors to ensure that the object and relationship feature vectors have stronger expression ability. Then, the attention coefficient between objects is calculated by using the high-level feature vector of edges in the scene graph, which is calculated as:

$$e_{ij} = T(W[h_{oi}, h_{rk}, h_{oj}]) \tag{6.1}$$

where e_{ij} is the contribution of instance object o_j to instance object o_i in any edge (o_i, r_k, o_j) of the scene graph, where attention computing network T indicates a tensor merging operation.

2) The softmax function is used to standardise all neighbourhood node objects of each object, so that the coefficients can be easily compared between different nodes, specifically:

$$\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k \in N_i} exp(e_{ik})} \tag{6.2}$$

where N_i represents all first-order neighbourhood nodes of node i (including i itself). After the standardised attention coefficients are obtained, the linear combination of node features corresponding to them is calculated as the final output of each object node.

3) After processing multiple graph attention layers, each object obtains its corresponding final output feature vector, which contains the feature information of all other objects in the scene graph. This ensures that different objects of the same category have different abstract vector representations, and different outputs can be used to predict the object layout. Figure 6.4 shows the graph attention network structure.

6.3.3 Scene Layout Synthesis

When converting a scene graph into a GUI, it is necessary to synthesise the scene layout corresponding to the GUI as a transition. Therefore, the object vector output from the graph attention network is transferred to the object layout network, the object layout of each object is predicted, and then all object layouts in the graph are combined to obtain the GUI scene layout [86].

The object layout network is composed of three parts: the prediction object border, the partition mask frame regression network, and the mask regression network. In the SG2UI model, the frame regression network uses two-layer multi-layer perceptron (MLP) [86] to predict the relative GUI coordinates of the object frame; that is, $b = (x_0, y_0, x_1, y_1)$. The mask regression network uses a series of upper sampling layers and convolution layers to predict the binary mask m with a fixed size of $M \times M$, and then synthesises the scene layout of the GUI.

6.3.4 Object Layout Network Training

The object layout network introduces the training idea of the confrontation process. Therefore, in order to avoid an object layout with large errors generated in the early

stages affecting the subsequent training of the network, the whole generation network needs to be trained in stages. The specific network structure is shown in the scene layout synthesis network in Figure 6.3. The network first generates a scene layout diagram that respects the generated GUI, and then trains it to generate high-quality GUIs.

Finally, the improved frame regression network and mask regression network are used to generate the object segmentation mask of the frame. In order to better distinguish the outline of objects, a non-transparent canvas colour mask is used to label different objects.

6.3.5 Loss Function

The generation from scene graph to scene layout is realised through training the graph attention network and the object layout network. The network loss function is calculated as follows:

$$Loss = \sum_{i=1}^{n} C(m, m_i) \tag{6.3}$$

Where C represents the cross entropy loss of binary classification of the generated mask m and the real mask m_i.

6.3.6 Evaluating Indicator

To better evaluate the generated image quality, this section introduces the FID [66] concept for testing. It evaluates the generation model from two aspects: clarity and diversity. Mathematically, Frechet Distance is used to calculate the distance between two "multivariate" normal distributions [66]. For a "univariate" normal distribution, the Frechet distance is:

$$d(X, Y) = (\mu_X - \mu_Y)^2 + (\sigma_X - \sigma_Y)^2 \tag{6.4}$$

where μ and σ are the mean and standard deviation of the normal distribution, and X and Y are two normal distributions. FID is given by Frechet Instance of multivariate normal distribution:

$$FID = \|\mu_X - \mu_Y\|^2 - Tr(\Sigma_X + \Sigma_Y - 2\sqrt{\Sigma_X \Sigma_Y}) \tag{6.5}$$

where X and Y are real and false embeddings, respectively, and are assumed to be two multivariate normal distributions [66]. μ_X and μ_Y are the mean values of the

vectors X and Y. Tr is the trace of the matrix, and Σ_X and Σ_Y are the covariance matrices of vectors.

6.4 Experiment

The SG2UI model is trained and verified using the Rico dataset. First, the scene graph is synthesised using the annotation information provided by the GUI, and then the model is trained to generate a 256 × 256 pixel GUI. The quality of the generated GUI is evaluated by using the FID [66]. The accuracy of the objects in the generated GUI is evaluated using Intersection Over Union (IOU).

6.4.1 Dataset

The Rico dataset [36] contains design data for more than 9.3k Android applications, covering 27 categories. It exposes 66K visual, textual, structural, and interactive design properties of unique GUI screens. Each GUI annotates the objects in it, focusing on the object frame information and segmentation mask. Through this annotation information, the model input scene graph can be synthesised.

Specifically, the relative relationship between objects is marked according to the GUI layout coordinates of the objects, while the scene graph is constructed with six mutually exclusive geometric relationships 'left of', 'right of', 'above', 'below', 'inside', and 'surrounding'. At the same time, a special GUI object is added to all scene graphs for expansion and a special in-GUI relationship is added between each object and GUI object to ensure that all objects can be connected in the scene graph.

The experiment ignores objects whose coverage areas are less than 5% of the GUI but retains images with 3 to 8 objects. Finally, 11,550 GUI images meeting the requirements are obtained from the Rico training set and designated as the training set for this experiment, 1,024 images meeting the requirements are selected from the Rico verification set and designated as the verification set, and 2,048 images are selected as the test set.

6.4.2 Implementation

The actual GUI size in the Rico dataset (2,560 × 1,440 pixels) was too large to be used for neural network training. To adapt to the graph attention network, we normalised the Rico dataset before the training process, and preprocessed the training samples into a format that can be read by the network. First, the following parameters are

established: input object vector dimension $F1 = 128$, final output vector dimension $F2 = 128$, segmentation mask output dimension $M = 16$, and length and width of the generated GUI $W = H = 64$. The Adam method [93] with a learning rate of 0.0001 is used as the optimisation function for all model trainings. Batch size is set to 32 and 100,000 iterations are performed. The network structure design is as follows:

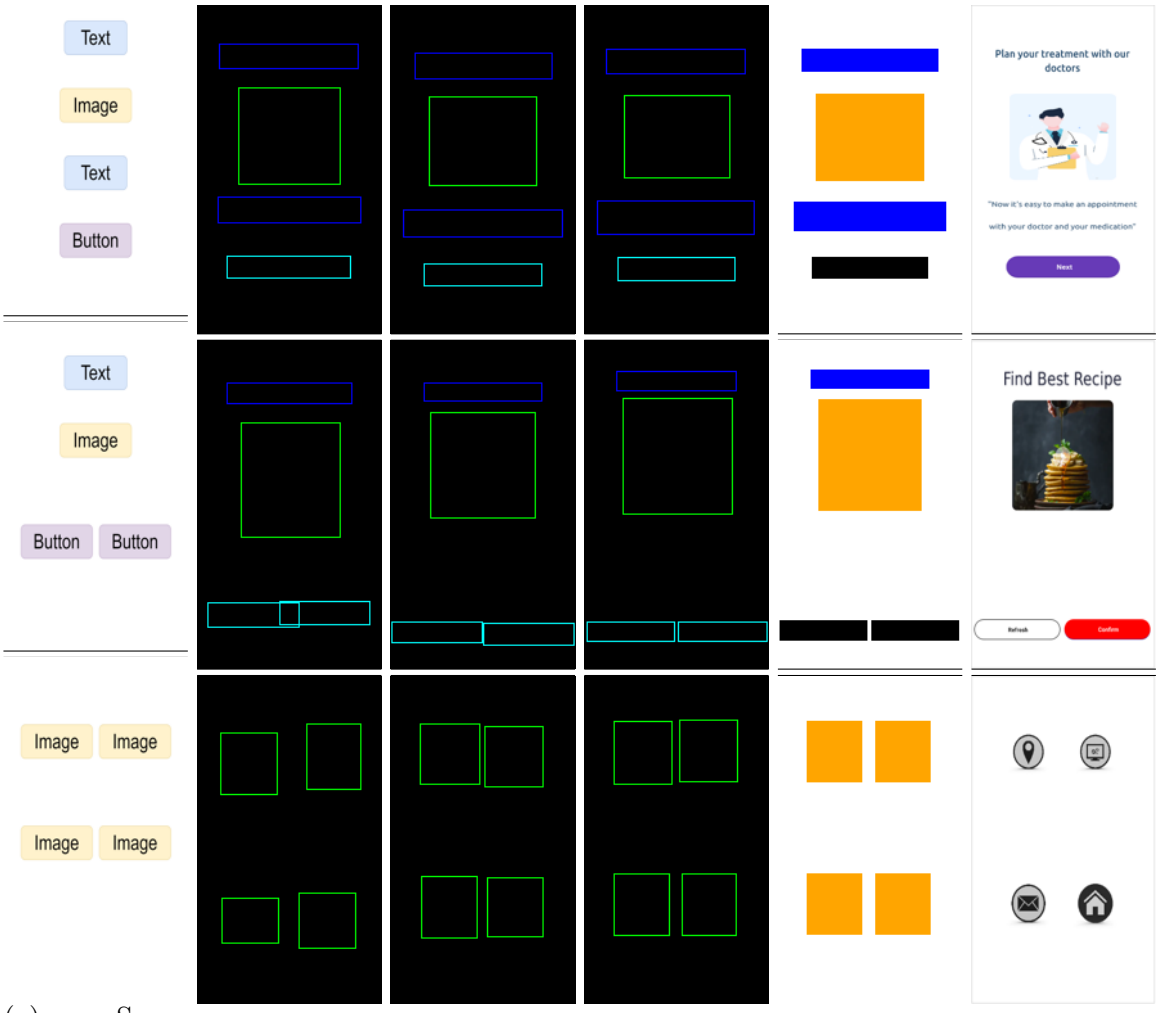
1) The graph attention network is composed of five identical graph attention layers in series. Each layer needs to train two parametric matrices. The object and relationship vector are transformed respectively to obtain the shared parametric matrix $W \in R^{F_1 \times F_2}$ and attention computing network T. The attention computing network is composed of two linear layers with ReLU function as the activation function. 2) The object layout network consists of a frame regression network, a mask regression network, and a mask discriminator network. Input is the object vector and output the four transformation coefficients of the frame.

The mask regression network requires a series of transpose convolution operations to realise the conversion from the object tensor to the mask tensor. Here, the upsample layer is connected with the convolution layer (conv). The convolution layers comprise 3×3 convolutions with a filling step of 1 with ReLU as the activation function. Herein, the batch normalisation proposed by Ioffe et al. [77] is introduced. In the last layer, to ensure that the output mask value is between (0, 1), the Sigmoid function is used as the activation function. The mask discriminator network is used to identify the real mask and generate the mask. Given that the classifier mainly learns the contour features of the mask for classification, a smaller convolution network can be used.

6.4.3 Qualitative Results

The Rico test set is used to verify the generalisability of the model. The final generated sample is shown in Figure 6.5. For a better comparison, Figure 6.5 also shows the corresponding reference GUI, the model input scene diagram synthesised according to the annotation information of the reference GUI, the reference text of one of the description images corresponding to the reference image, the 2D scene graph corresponding to the generated GUI layout, the generated GUI layout of the sg2im model [86] input with the same scene graph, and the generated GUI layout of the StackGAN model [201] input with reference text.

As can be seen in Figure 6.5, compared with the GUI layout generated by the sg2im model, the GUI layout generated by SG2UI is smoother, clearer and less different from the reference GUI. Meanwhile, the alignment accuracy of GUI layout generated by



(a) Scene Graph (b) StackGAN (c) sg2im (d) SG2UI (e) Canvas (f) GUI

Figure 6.5: GUI generated using the Rico test set.

StackGAN model is lower than that of our model, which shows that the method that makes use of the scene graph as input is more conducive to generating complex GUIs containing multiple objects and relationships than the method using text description as input.

In the Rico dataset, the test set samples are randomly divided into five groups, and the mean and standard deviation of each group are recorded. For the Sg2UI and sg2im models, the GUIs of each test set are synthesised into five different scene graphs to generate five sample GUIs. For the StackGAN model, each text description of each test set GUI generates a GUI with a size of 256 $\times$ 256 pixels. The GUI is reduced to 64 $\times$ 64 pixels by downsampling. The final results are shown in Table 6.1.

In addition to viewing the GUI layout, evaluating the quality of the generated

Table 6.1: Comparison of FID for 64 × 64 pixels GUIs generated by different methods

Methods	FID	*Loss*
Real GUIs	17.3 ± 0.4	0.212
StackGAN	9.4 ± 0.2	0.467
sg2im	8.3 ± 0.1	0.279
SG2UI(ours)	8.8 ± 0.1	0.248

Table 6.2: Statistics of predicted bounding boxes

Methods	R@0.3	R@0.5
sg2im	53.4	33.2
SG2UI(ours)	59.7	39.4

GUI can also detect the object frame predicted by the model. There are two common measurement methods. The first method entails calculating the IoU ratio between the predicted generated object frame b and the object frame B provided by the real label. Specifically:

$$IoU = \frac{b \bigcap B}{b \bigcup B} \tag{6.6}$$

The other measurement method entails generating the diversity of object frames, that is, the predicted changes in the object frames relative to other objects and relationships in the graph are evaluated by the standard deviation of the position and area of the object bounding boxes of each category.

Table 6.2 presents an evaluation of the accuracy and diversity of the prediction-generated frame by the SG2UI and sg2im models [86]. Herein, $R@t$ represents the recall rate of objects with different IoU thresholds, which is used to evaluate the accuracy of the prediction object frame, while σ_x and σ_{area} represent the average of the standard deviation of the frame position and area of objects in all categories.

If graph convolution is not used, the model can only predict a separate bounding box for the objects from each object category, but cannot realise the prediction generation of different objects from the same category, that is, $\sigma_x = \sigma_{area} = 0$.

The experimental data shows that compared with sg2im and StackGAN models, the SG2UI model discussed in this chapter is more conducive for the generation of complex scene graphs containing multiple objects and relationships. Notably, the

generated GUI quality is better. In addition, the improved object layout network is more accurate in predicting the position of objects in the GUI and can better distinguish different objects from within the same category.

6.5 Summary

With the aim to address the problem of existing text-to-image generation models being unable to generate multiple objects and relationships, a SG2UI model based on graph attention network scene graph-to-GUI generation is proposed in this chapter. First, the graph attention network is used as the feature extraction network of the input scene graph. Then, the improved object layout network is used to predict the object segmentation mask and frame in the GUI to obtain a more realistic scene layout. The canvas is used to convert the scene layout into a real GUI. Thereafter, the FID and perceptual loss are used to calculate the difference between the generated GUI and the real GUI.

The final qualitative experimental results demonstrate that the object details of the final GUI generated by the model are more apparent and the relationship between objects is more in line with the source information, indicating that the model improves the quality of the generated GUI to a certain extent. At the same time, the quantitative experimental results show that compared with the sg2im model proposed by Johnson et al. [86], the SG2UI model can obtain a higher FID score.

In Chapter 4, we proposed an improved method for some defects in the pix2code framework. This method implemented an encoder–decoder model. They trained per the GUI metadata and information in the screenshot. The target screenshot is first translated into a domain-specific language (DSL), then into GUI code. This method has some disadvantages: (I) It is only verified on a small number of synthetic applications, and there is no large-scale user interface mining; (II) It requires a DSL that must be maintained and updated over time, increasing the complexity and workload required to use the method. Therefore, it is difficult to judge this method's performance on actual GUI data. In contrast, SG2UI is trained on a large data set, Rico, collected through a new application for the automatic dynamic analysis of user interface mining. The data collection and training process can be fully automated and repeated, which helps reduce the burden on developers.

In this chapter, the SG2UI model successfully realises the generation of complex scene layouts containing multiple objects and relationships, but the input scene graph of the model depends on a large amount of image annotation information, which has

an adverse effect on the generation of objects and relationships that do not appear in the training dataset. Therefore, as follow-up work, a planned research direction is for the model to directly generate complex scene layouts through more easily obtained natural semantic text. Furthermore, new methods will be explored to capture objects in semantic text and determine the relationships between them.

Chapter 7

Conclusions

7.1 Introduction

The aim of this project is to develop a system that can automatically generate a specific platform code for a given graphical user interface (GUI) screenshot or GUI elements as inputs, it provides two main methods for GUI generation. The extended version of these methods might reduce the need to manually program GUIs. Users can choose which method to obtain the desired generation results according to their requests. The improved version of pix2code in Chapter 4 can obtain its HTML code through a GUI screenshot, and the CSM proposed in Chapter 3 can be used as one of its extended measurement methods. If users do not have GUI screenshots or just want to generate results through their own design inspiration, they can just give the keywords of elements in the GUI and complete it through the SG2UI model provided in Chapter 6. Chapter 5 provides the method of generating original GUI datasets to provide further data support for the above two generation methods.

This chapter summaries the findings of the study and gives conclusions drawn from the findings. Finally, some suggestions for future research will be given.

7.2 Research Findings

Table 7.1 illustrates the objectives and the outcomes of this thesis. The first objective focuses on the development of evaluation metrics for GUI code generation. Chapter 3 proposed CSM and compare its performances with the benchmark plagiarism detection tools. We validate on four SMT-based datasets from two sources, i.e lpSMT, lpSMT2, mppSMT and mppSMT2. The N-gram model based on TF-IDF weighting and cosine similarity solves this problem well, and has good scores on different datasets. TF-IDF weighted features can also be used as part of the visualization

Table 7.1: Objectives and outcomes of this thesis.

No	Objective(s)	Outcome(s)
1	Design a novel metric to evaluate the accuracy for GUI layouts.	This study has designed a code semantic metric (CSM) using n-gram sequence features and cosine similarity to measure the accuracy of translation code. (Chapter 3)
2	Propose an improved framework to solve the problem of feature vector losses in a pix2code framework to a certain extent.	An improved framework based on pix2code is developed to automatically generate a specific platform code for a given GUI screenshot as an input. (Chapter 4)
3	Synthesise realistic GUI images using generative adversarial networks.	The UIGAN is developed for GUI data augmentation. (Chapter 5)
4	Create a scene graph model to generate additional GUI examples such as actual texts, images, and buttons.	A scene graph-to-UI generation model based on a graph attention network is proposed to generate higher quality GUI layouts. (Chapter 6)

method, because more unique parts of a word pair will show more significance than other similarities.

At present, the model still has some limitations, which illustrate the possible follow-up steps:

- CSM uses n-gram sequence features and Cosine Similarity to judge the accuracy of translation code. There is no training process of dataset before running.
- The method only understands the importance of some similarities at runtime based on the features in the dataset. This means that when the data set is small, the distribution of some features may not represent the true distribution of tasks.

Chapter 4 proposed a modified framework for solving the problem of feature vector losses in pix2code framework to a certain extent. The preliminary experimental results outperformed the state-of-the-art methods based on BLEU and MBLEU. We demonstrate MBLEU is suitable for DSL evaluation.

At present, we found that the model have one or more of the following drawbacks:

- Due to the limitations of the existing pix2code dataset, the model only trains a vocabulary of 16 elements, so it can only predict the DSL token specified in the data.
- Because CSS lacks style changes when making sketch parts of existing datasets, there is still a big difference compared to human hand-drawn sketches.

- There are some shortcomings in the NLP evaluation metrics. The existing methods lack the ability to judge the dependency relationship between different DSL tokens and its importance to the whole web page. It is necessary to improve the penalty factor or find a more suitable alternative.

Chapter 5 introduced UIGAN, which performed well in experiments of dataset augmentation, achieving beyond initial expectations. The experimental results return an evaluation value smaller than that of traditional GAN; this shows that the algorithm has improved upon GAN in the field of GUI generation to a certain extent. This could be attributed to the fact that UIGAN can learn the dependencies between data sequences. The experimental results also show that UIGAN can generate similar datasets of GUI.

With an aim to address the problem of existing text-to-image generation models being unable to generate multiple objects and relationships, Chapter 6 proposed a SG2UI model based on graph attention network scene graph to GUI generation is proposed in this chapter. First, the graph attention network is used as the feature extraction network of the input scene graph. Then, the improved object layout network is used to predict the object segmentation mask and frame in the GUI to obtain a more realistic scene layout. The canvas is used to convert the scene layout into a real GUI. Thereafter, the FID and perceptual loss are used to calculate the difference between the generated GUI and the real GUI.

The final qualitative experimental results demonstrate that the object details of the final GUI generated by the model are more apparent and the relationship between objects is more in line with the facts, indicating that the model improves the quality of the generated GUI to a certain extent. At the same time, the quantitative experimental results show that compared with the sg2im model proposed by Johnson et al. [86], the SG2UI model obtained a higher FID score.

7.3 Future Work

The future of Automatic GUI generation with great promises, as major technology companies and developers have been attempting the application of machine learning in their own field. The following are some future work for GUI generation:

- In order to further improve the accuracy of code semantic metric in Chapter 3, future research directions will consider building a simpler monitoring model through deep learning and multiple similarity metrics. The evaluation model

needs to be able to understand the importance of each similarity metric. Another interesting direction is to use unsupervised learning algorithms on multiple source code translation datasets, and then use these representations as features of the supervised model.

- In Chapter 4, the future work could be trying to create more elements to generate additional web examples such as actual icons, image buttons, drop-down menus, forms, and bootstrap components. With the increasing performance of computer hardware, it is better to create a dataset that can be directly trained by HTML/CSS code than a DSL token sequence in the future. A good way to generate more variants in hand-drawn sketch data might be to create a realistic hand-drawn website image using a Generative Adversarial Network (GAN).

- There exists a limited number of experiments and evaluation standards in related fields on whether the similarities between the dataset generated by UIGAN and the original data are reasonable and well generated. At the same time, the algorithm regularly deviates from the correct result of GUI in the generation process of complex layout. Future work will focus on improving the existing GAN model and trying to realize the generation of more complex layout structure on GUI datasets such as Redraw [122].

- In Chapter 6, the SG2UI model successfully realises the generation of complex scene layouts containing multiple objects and relationships, but the input scene graph of the model depends on a large amount of image annotation information, which has an adverse effect on the generation of objects and relationships that do not appear in the training dataset. Therefore, in the follow-up work, a planned research direction is for the model to directly generate complex scene layouts through more easily obtained natural semantic text. Another research direction is to functionalize GUI by placing GUI design in the context of HCI design, especially leading from the GUI to the process of human-computer interaction and the hypertext nature of graphic user interface. Furthermore, new methods will be explored to capture objects in semantic text and determine the relationships between them.

Bibliography

[1] Screenshot to code. https://github.com/emilwallner/Screenshot-to-code/blob/master/README.md. Accessed: 2019-01-22.

[2] Alankrita Aggarwal, Mamta Mittal, and Gopi Battineni. Generative adversarial network: An overview of theory and applications. *International Journal of Information Management Data Insights*, page 100004, 2021.

[3] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

[4] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M Memon. Using gui ripping for automated testing of android applications. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 258–261. IEEE, 2012.

[5] Pirkka Åman and Lassi A Liikkanen. Interacting with context factors in music recommendation and discovery. *International Journal of Human–Computer Interaction*, 33(3):165–179, 2017.

[6] Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[7] Brenda S Baker. On finding duplication and near-duplication in large software systems. In *Proceedings of 2nd Working Conference on Reverse Engineering*, pages 86–95. IEEE, 1995.

[8] Brenda S Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM Journal on Computing*, 26(5):1343–1362, 1997.

[9] Matej Balog, Alexander L Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. *arXiv preprint arXiv:1611.01989*, 2016.

[10] Ira D Baxter, Christopher Pidgeon, and Michael Mehlich. Dms®: Program transformations for practical scalable software evolution. In *Proceedings of the 26th International Conference on Software Engineering*, pages 625–634. IEEE Computer Society, 2004.

[11] Ira D Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, pages 368–377. IEEE, 1998.

[12] Farnaz Behrang, Steven P Reiss, and Alessandro Orso. Guifetch: supporting app design and development through gui search. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, pages 236–246, 2018.

[13] Laszlo A Belady and Carlo J Evangelisti. System partitioning and its measure. *Journal of Systems and Software*, 2(1):23–29, 1981.

[14] Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 1–6, 2018.

[15] Carlos Bernal-Cárdenas, Kevin Moran, Michele Tufano, Zichang Liu, Linyong Nan, Zhehan Shi, and Denys Poshyvanyk. Guigle: A gui search engine for android apps. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 71–74. IEEE, 2019.

[16] Nigel Bevan. Measuring usability as quality of use. *Software Quality Journal*, 4(2):115–130, 1995.

[17] Barry Boehm. A view of 20th and 21st century software engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 12–29, 2006.

[18] Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. Programming with a differentiable forth interpreter. In *International conference on machine learning*, pages 547–556. PMLR, 2017.

[19] Tom Brinck, Darren Gergle, and Scott D Wood. *Designing Web sites that work: Usability for the Web*. Morgan Kaufmann Publishers, 2002.

[20] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer networks and ISDN systems*, 29(8-13):1157–1166, 1997.

[21] Rudy R Bunel, Alban Desmaison, Pawan K Mudigonda, Pushmeet Kohli, and Philip Torr. Adaptive neural compilation. *Advances in Neural Information Processing Systems*, 29:1444–1452, 2016.

[22] Yue Cao, Bin Liu, Mingsheng Long, and Jianmin Wang. Hashgan: Deep learning to hash with pair conditional wasserstein gan. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1287–1296, 2018.

[23] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation. In *Proceedings of the 40th International Conference on Software Engineering*, pages 665–676, 2018.

[24] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1511–1520, 2017.

[25] Xinxiong Chen, Lei Xu, Zhiyuan Liu, Maosong Sun, and Huanbo Luan. Joint learning of character and word embeddings. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[26] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[27] Ondřej Cífka and Ondřej Bojar. Are bleu and meaning representation in opposition? *arXiv preprint arXiv:1805.06536*, 2018.

[28] Ian G Clifton. *Android user interface design: Implementing material design for developers*. Addison-Wesley Professional, 2015.

[29] Bohemian Coding. Sketch. https://www.sketch.com/, 2019.

[30] Ander Corral and Xabier Saralegi. Elhuyar submission to the biomedical translation task 2020 on terminology and abstracts translation. In *Proceedings of the Fifth Conference on Machine Translation*, pages 813–819, 2020.

[31] Felipe Costa., Priscila Saito., and Pedro Bugatti. Video action classification through graph convolutional networks. In *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 4: VISAPP,*, pages 490–497. INSTICC, SciTePress, 2021.

[32] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[33] Zhiyong Cui, Kristian Henrickson, Ruimin Ke, and Yinhai Wang. Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. *IEEE Transactions on Intelligent Transportation Systems*, 21(11):4883–4894, 2019.

[34] Alastair H Cummings. The evolution of game controllers and control schemes and their effect on their games. In *The 17th annual university of southampton multimedia systems conference*, volume 21. Citeseer, 2007.

[35] P Pinheiro da Silva and Norman W Paton. A uml-based design environment for interactive applications. In *Proceedings Second International Workshop on User Interfaces in Data Intensive Systems. UIDIS 2001*, pages 60–71. IEEE, 2001.

[36] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 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*, pages 845–854, 2017.

[37] Ai-Ping Deng, GL Xu, and Ben Xiao. Research on a string matching algorithm for detecting plagiarized source code [j]. *Computer Engineering & Science*, 30(3):62–64, 2008.

[38] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdelrahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. *arXiv preprint arXiv:1703.07469*, 2017.

[39] Devanshu Dhyani, Wee Keong Ng, and Sourav S Bhowmick. A survey of web metrics. *ACM Computing Surveys (CSUR)*, 34(4):469–503, 2002.

[40] Kaize Ding, Qinghai Zhou, Hanghang Tong, and Huan Liu. Few-shot network anomaly detection via cross-network meta-learning. In *Proceedings of the Web Conference 2021*, pages 2448–2456, 2021.

[41] Chunning Du, Jingyu Wang, Haifeng Sun, Qi Qi, and Jianxin Liao. Syntax-type-aware graph convolutional networks for natural language understanding. *Applied Soft Computing*, 102:107080, 2021.

[42] Elfriede Dustin, Jeff Rashka, and John Paul. *Automated software testing: introduction, management, and performance*. Addison-Wesley Professional, 1999.

[43] Mohammed Elkoutbi, Ismaïl Khriss, and Rudolf K Keller. Automated prototyping of user interfaces based on uml scenarios. *Automated Software Engineering*, 13(1):5–40, 2006.

[44] Evan N Feinberg, Debnil Sur, Zhenqin Wu, Brooke E Husic, Huanghao Mai, Yang Li, Saisai Sun, Jianyi Yang, Bharath Ramsundar, and Vijay S Pande. Potentialnet for molecular property prediction. *ACS central science*, 4(11):1520–1530, 2018.

[45] Simon Fothergill, Helena Mentis, Pushmeet Kohli, and Sebastian Nowozin. Instructing people for training gestural interactive systems. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1737–1746, 2012.

[46] Alex M Fout. *Protein interface prediction using graph convolutional networks*. PhD thesis, Colorado State University, 2017.

[47] Gil Francopoulo, Joseph Mariani, and Patrick Paroubek. Predictive modeling: guessing the nlp terms of tomorrow. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 336–343, 2016.

[48] Wilbert O Galitz. *The essential guide to user interface design: an introduction to GUI design principles and techniques*. John Wiley & Sons, 2007.

[49] Alexander L Gaunt, Marc Brockschmidt, Rishabh Singh, Nate Kushman, Pushmeet Kohli, Jonathan Taylor, and Daniel Tarlow. Terpret: A probabilistic programming language for program induction. *arXiv preprint arXiv:1608.04428*, 2016.

[50] James H Gerlach and Feng-Yang Kuo. Understanding human-computer interaction for information systems design. *MIS quarterly*, pages 527–549, 1991.

[51] Kathrin M Gerling, Frank P Schulte, and Maic Masuch. Designing and evaluating digital games for frail elderly persons. In *Proceedings of the 8th international conference on advances in computer entertainment technology*, pages 1–8, 2011.

[52] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[53] David Gitchell and Nicholas Tran. Sim: a utility for detecting similarity in computer programs. In *ACM SIGCSE Bulletin*, volume 31, pages 266–270. ACM, 1999.

[54] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[55] Google. Android studio. developer.android.com/studio/index.html/, 2019.

[56] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 922–929, 2019.

[57] JoAnn T Hackos and Janice Redish. *User and task analysis for interface design*. Wiley New York, 1998.

[58] Md Akmal Haidar and Mehdi Rezagholizadeh. Textkd-gan: Text generation using knowledge distillation and generative adversarial networks. In *Canadian conference on artificial intelligence*, pages 107–118. Springer, 2019.

[59] Maurice Howard Halstead et al. *Elements of software science*, volume 7. Elsevier New York, 1977.

[60] Junwei Han, Dingwen Zhang, Gong Cheng, Nian Liu, and Dong Xu. Advanced deep-learning techniques for salient and category-specific object detection: a survey. *IEEE Signal Processing Magazine*, 35(1):84–100, 2018.

[61] David J Hand. Data mining. *Encyclopedia of Environmetrics*, 2, 2006.

[62] Yixue Hao, Min Chen, Long Hu, M Shamim Hossain, and Ahmed Ghoneim. Energy efficient task caching and offloading for mobile edge computing. *IEEE Access*, 6:11365–11373, 2018.

[63] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[64] Eric B Hekler, Predrag Klasnja, Jon E Froehlich, and Matthew P Buman. Mind the theoretical gap: interpreting, using, and developing behavioral theory in hci research. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3307–3316, 2013.

[65] Hadi Hemmati, Zhihan Fang, and Mika V Mantyla. Prioritizing manual test cases in traditional and rapid release environments. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–10. IEEE, 2015.

[66] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[67] Ken Hinckley and Daniel Wigdor. Input technologies and techniques. *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*, pages 151–168, 2002.

[68] Karen Holtzblatt and Hugh Beyer. Contextual design: evolved. *Synthesis Lectures on Human-Centered Informatics*, 7(4):1–91, 2014.

[69] Jason Hong. Matters of design. *Communications of the ACM*, 54(2):10–11, 2011.

[70] Guoyong Hu, Ye Ding, Shuhan Qi, Xuan Wang, and Qing Liao. Multi-depth graph convolutional networks for fake news detection. In *CCF International conference on natural language processing and chinese computing*, pages 698–710. Springer, 2019.

[71] Li Huang, Meiling He, Chong Tan, Du Jiang, Gongfa Li, and Hui Yu. Jointly network image processing: multi-task image semantic segmentation of indoor scene based on cnn. *IET Image Processing*, 14(15):3689–3697, 2020.

[72] Rui Huang, Shu Zhang, Tianyu Li, and Ran He. Beyond face rotation: Global and local perception gan for photorealistic and identity preserving frontal view synthesis. In *Proceedings of the IEEE international conference on computer vision*, pages 2439–2448, 2017.

[73] Sungjae Hwang, Andrea Bianchi, Myungwook Ahn, and Kwangyun Wohn. Magpen: magnetically driven pen interactions on and around conventional smartphones. In *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services*, pages 412–415, 2013.

[74] Adobe Inc. Adobe photoshop. https://www.adobe.com/Adobe/Photoshop/, 2019.

[75] Apple Inc. Xcode. https://developer.apple.com/xcode/, 2019.

[76] R Indhumathi and S Sathiya Devi. Healthcare cramér generative adversarial network (hcgan). *Distributed and Parallel Databases*, pages 1–17, 2021.

[77] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[78] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[79] Robert JK Jacob and Keith S Karn. Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. In *The mind's eye*, pages 573–605. Elsevier, 2003.

[80] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

[81] Rohan Jagtap, Dr Dhage, and N Sudhir. An in-depth walkthrough on evolution of neural machine translation. *arXiv preprint arXiv:2004.04902*, 2020.

[82] Bernard J Jansen. The graphical user interface. *ACM SIGCHI Bulletin*, 30(2):22–26, 1998.

[83] Tobias Skovgaard Jepsen, Christian S Jensen, and Thomas Dyhre Nielsen. Relational fusion networks: Graph convolutional networks for road networks. *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[84] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. Deckard: Scalable and accurate tree-based detection of code clones. In *Proceedings of the 29th international conference on Software Engineering*, pages 96–105. IEEE Computer Society, 2007.

[85] Lingzi Jin, Hong Zhu, and Patrick Hall. Adequate testing of hypertext applications. *Information and software technology*, 39(4):225–234, 1997.

[86] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1219–1228, 2018.

[87] Joel Jones. Abstract syntax tree implementation idioms. In *Proceedings of the 10th conference on pattern languages of programs (plop2003)*, page 26, 2003.

[88] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[89] Svetoslav Karaivanov, Veselin Raychev, and Martin Vechev. Phrase-based statistical translation of programming languages. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, pages 173–184. ACM, 2014.

[90] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[91] Phil Kim. Convolutional neural network. In *MATLAB deep learning*, pages 121–147. Springer, 2017.

[92] Tae San Kim, Won Kyung Lee, and So Young Sohn. Graph convolutional network approach applied to predict hourly bike-sharing demands considering spatial, temporal, and global effects. *PloS one*, 14(9):e0220782, 2019.

[93] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[94] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[95] Gregory Kipper and Joseph Rampolla. *Augmented Reality: an emerging technologies guide to AR*. Elsevier, 2012.

[96] Raghavan Komondoor and Susan Horwitz. Using slicing to identify duplication in source code. In *International static analysis symposium*, pages 40–56. Springer, 2001.

[97] Jens Krinke. Identifying similar code with program dependence graphs. In *Proceedings Eighth Working Conference on Reverse Engineering*, pages 301–309. IEEE, 2001.

[98] Andres Kull. Automatic gui model generation: State of the art. In *2012 IEEE 23rd International Symposium on Software Reliability Engineering Workshops*, pages 207–212. IEEE, 2012.

[99] Wataru Kumagai and Akiyoshi Sannai. Universal approximation theorem for equivariant maps by group cnns. *arXiv preprint arXiv:2012.13882*, 2020.

[100] Mary Ann Chiramattel Kunjachan. Evaluation of usability on mobile user interface. *University of Washington, Bothell*, 2011.

[101] Francis Quintal Lauzon. An introduction to deep learning. In *2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*, pages 1438–1439. IEEE, 2012.

[102] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[103] Andrew Lee. *Generating Wabpages from Screenshots*. Stanford University, 2018.

[104] Rogério de Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software engineering for self-adaptive systems II*, pages 1–32. Springer, 2013.

[105] Xiuming Li, Xin Yan, Qiong Gu, Huihao Zhou, Di Wu, and Jun Xu. Deepchemstable: chemical stability prediction with an attention-based graph convolution network. *Journal of chemical information and modeling*, 59(3):1044–1049, 2019.

[106] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. Humanoid: A deep learning-based approach to automated black-box android app testing. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1070–1073. IEEE, 2019.

[107] Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744*, 2016.

[108] Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. Deformable shape completion with graph convolutional autoencoders. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1886–1895, 2018.

[109] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine learning*, 2(4):285–318, 1988.

[110] Yufei Liu, Yuan Zhou, Xin Liu, Fang Dong, Chang Wang, and Zihong Wang. Wasserstein gan-based small-sample augmentation for new-generation artificial intelligence: a case study of cancer-staging data in biology. *Engineering*, 5(1):156–163, 2019.

[111] Zequn Liu, Ruiyi Zhang, Yiping Song, and Ming Zhang. When does maml work the best? an empirical study on model-agnostic meta-learning in nlp applications. *arXiv preprint arXiv:2005.11700*, 2020.

[112] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[113] Michael R Lyu. Software reliability engineering: A roadmap. In *Future of Software Engineering (FOSE'07)*, pages 153–170. IEEE, 2007.

[114] Martin Maguire. Context of use within usability activities. *International journal of human-computer studies*, 55(4):453–483, 2001.

[115] David Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM (JACM)*, 25(2):322–336, 1978.

[116] Mahsa Mesgaran and A Ben Hamza. Graph fairing convolutional networks for anomaly detection. *arXiv preprint arXiv:2010.10274*, 2020.

[117] Microsoft. Microsoft visual studio. https://visualstudio.microsoft.com/, 2019.

[118] Jd R Millán, Frederic Renkens, Josep Mourino, and Wulfram Gerstner. Noninvasive brain-actuated control of a mobile robot by human eeg. *IEEE Transactions on biomedical Engineering*, 51(6):1026–1033, 2004.

[119] Praful Mishra, Anmol Mishra, Aniket Bharti, and Sarita Ambadekar. Theoretical answer evaluation using lsa, bleu, wmd and fuzzy logic. *International Journal of Advanced Research in Computer Science*, 9(2), 2018.

[120] Mihir Mistry, Ameya Apte, Varad Ghodake, and SB Mane. Machine learning based user interface generation. In *International Conference on Intelligent Computing, Information and Control Systems*, pages 453–460. Springer, 2019.

[121] Paolo Montuschi, Andrea Sanna, Fabrizio Lamberti, and Gianluca Paravati. Human-computer interaction: Present and future trends. *Computing Now*, 7(9), 2014.

[122] Kevin Moran, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. Machine learning-based prototyping of graphical user interfaces for mobile apps. *IEEE Transactions on Software Engineering*, 46(2):196–221, 2018.

[123] Kevin Moran, Boyang Li, Carlos Bernal-Cárdenas, Dan Jelf, and Denys Poshyvanyk. Automated reporting of gui design violations for mobile apps. In *Proceedings of the 40th International Conference on Software Engineering*, pages 165–175. ACM, 2018.

[124] Maxim Mozgovoy et al. Desktop tools for offline plagiarism detection in computer programs. *Informatics in Education-An International Journal*, 5(1):97–112, 2006.

[125] Damian Mrowca, Chengxu Zhuang, Elias Wang, Nick Haber, Li Fei-Fei, Joshua B Tenenbaum, and Daniel LK Yamins. Flexible neural representation for physics prediction. *arXiv preprint arXiv:1806.08047*, 2018.

[126] Brad Myers, Sun Young Park, Yoko Nakano, Greg Mueller, and Andrew Ko. How designers design and program interactive behaviors. In *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 177–184. IEEE, 2008.

[127] Medhini Narasimhan, Svetlana Lazebnik, and Alexander G Schwing. Out of the box: Reasoning with graph convolution nets for factual visual question answering. *arXiv preprint arXiv:1811.00538*, 2018.

[128] Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

[129] Anh Tuan Nguyen, Tung Thanh Nguyen, and Tien N Nguyen. Lexical statistical machine translation for language migration. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 651–654. ACM, 2013.

[130] Anh Tuan Nguyen, Tung Thanh Nguyen, and Tien N Nguyen. Migrating code with statistical machine translation. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 544–547. ACM, 2014.

[131] Anh Tuan Nguyen, Tung Thanh Nguyen, and Tien N Nguyen. Divide-and-conquer approach for multi-phase statistical migration for source code (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 585–596. IEEE, 2015.

[132] Anh Tuan Nguyen, Zhaopeng Tu, and Tien N Nguyen. Do contexts help in phrase-based, statistical source code migration? In *2016 IEEE International*

Conference on Software Maintenance and Evolution (ICSME), pages 155–165. IEEE, 2016.

[133] Tuan Anh Nguyen and Christoph Csallner. Reverse engineering mobile application user interfaces with remaui (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 248–259. IEEE, 2015.

[134] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, 2015.

[135] Greg Nudelman. *Android design patterns: interaction design solutions for developers*. John Wiley & Sons, 2013.

[136] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *International conference on machine learning*, pages 2642–2651. PMLR, 2017.

[137] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[138] Raj Patel. A brief overview on generative adversarial networks. *Data and Communication Networks*, pages 267–277, 2019.

[139] Francesca Pistilli, Giulia Fracastoro, Diego Valsesia, and Enrico Magli. Learning graph-convolutional representations for point cloud denoising. In *European Conference on Computer Vision*, pages 103–118. Springer, 2020.

[140] Lutz Prechelt, Guido Malpohl, Michael Philippsen, et al. Finding plagiarisms among a set of programs with jplag. *J. UCS*, 8(11):1016, 2002.

[141] Roger S Pressman. *Software engineering: a practitioner's approach*. Palgrave macmillan, 2005.

[142] Feddy Setio Pribadi, Teguh Bharata Adji, Adhistya Erna Permanasari, and Takashi Ninomiya. Short answer scoring using w-bleu for regular assessment in vocational high school. *International Journal of Innovation and Learning*, 24(4):437–447, 2018.

[143] Angel R Puerta. A model-based interface development environment. *IEEE Software*, 14(4):40–47, 1997.

[144] Guocheng Qian, Abdulellah Abualshour, Guohao Li, Ali Thabet, and Bernard Ghanem. Pu-gcn: Point cloud upsampling using graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11683–11692, 2021.

[145] Zheng Qin, Xiang Zheng, and Jiankuan Xing. *Introduction to software architecture*. Springer, 2008.

[146] Chaiyong Ragkhitwetsagul, Jens Krinke, and David Clark. A comparison of code similarity analysers. *Empirical Software Engineering*, 23(4):2464–2519, 2018.

[147] Agyl Ardi Rahmadi and Aris Sudaryanto. Visual recognition of graphical user interface components using deep learning technique. *Jurnal Ilmu Komputer dan Informasi*, 13(1):35–45, 2020.

[148] Syed Fazle Rahman. *Jump Start Bootstrap*. SitePoint, 2014.

[149] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *International Conference on Machine Learning*, pages 1060–1069. PMLR, 2016.

[150] Scott E Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee. Learning what and where to draw. *Advances in neural information processing systems*, 29:217–225, 2016.

[151] Steven P Reiss, Yun Miao, and Qi Xin. Seeking the user interface. *Automated Software Engineering*, 25(1):157–193, 2018.

[152] Ehud Reiter. A structured review of the validity of bleu. *Computational Linguistics*, 44(3):393–401, 2018.

[153] Daniela Retelny, Michael S Bernstein, and Melissa A Valentine. No workflow can ever be enough: How crowdsourcing workflows constrain complex work. *Proceedings of the ACM on Human-Computer Interaction*, 1(CSCW):1–23, 2017.

[154] Philip Sallis, Asbjorn Aakjaer, and Stephen MacDonell. Software forensics: old methods for a new science. In *Proceedings 1996 International Conference Software Engineering: Education and Practice*, pages 481–485. IEEE, 1996.

[155] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[156] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85. ACM, 2003.

[157] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[158] Douglas C Schmidt. Model-driven engineering. *Computer-IEEE Computer Society-*, 39(2):25, 2006.

[159] Brian Shackel. Usability–context, framework, definition, design and evaluation. *Interacting with computers*, 21(5-6):339–346, 2009.

[160] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[161] Ben Shneiderman, Catherine Plaisant, Maxine S Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. *Designing the user interface: strategies for effective human-computer interaction*. Pearson, 2016.

[162] Patrice Simard, Bernard Victorri, Yann LeCun, and John S Denker. Tangent prop-a formalism for specifying selected invariances in an adaptive network. In *NIPS*, volume 91, pages 895–903. Citeseer, 1991.

[163] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3, 2003.

[164] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[165] Amit Singhal et al. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[166] Fadzilah Siraj and Mohamed Salem Ali. Evaluation of structured questions using modified bleu algorithm. *International Journal of Innovations in Engineering and Technology (IJIET)*, 7(4):374–383, 2016.

[167] Sarah Suleri, Vinoth Pandian Sermuga Pandian, Svetlana Shishkovets, and Matthias Jarke. Eve: A sketch-based software prototyping workbench. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–6, 2019.

[168] Xiaolei Sun, Tongyu Li, and Jianfeng Xu. Ui components recognition system based on image understanding. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 65–71. IEEE, 2020.

[169] Farbod Taymouri, Marcello La Rosa, Sarah Erfani, Zahra Dasht Bozorgi, and Ilya Verenich. Predictive business process monitoring via generative adversarial nets: The case of next event prediction. In *International Conference on Business Process Management*, pages 237–256. Springer, 2020.

[170] Paul Thagard and David Croft. Scientific discovery and technological innovation: Ulcers, dinosaur extinction, and the programming language java. In *Model-based reasoning in scientific discovery*, pages 125–137. Springer, 1999.

[171] Brian H Toby. Expgui, a graphical user interface for gsas. *Journal of applied crystallography*, 34(2):210–213, 2001.

[172] Ngoc Tran, Hieu Tran, Son Nguyen, Hoan Nguyen, and Tien Nguyen. Does bleu score work for code migration? In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pages 165–176. IEEE, 2019.

[173] Diego Valsesia, Giulia Fracastoro, and Enrico Magli. Image denoising with graph-convolutional neural networks. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2399–2403. IEEE, 2019.

[174] Paul Van Schaik and Jonathan Ling. The role of context in perceptions of the aesthetics of web pages over time. *International journal of human-computer studies*, 67(1):79–89, 2009.

[175] Kristina L Verco and Michael J. Wise. Plagiarism à la mode: a comparison of automated systems for detecting suspected plagiarism. *The Computer Journal*, 39(9):741–750, 1996.

[176] Nitika Verma, Edmond Boyer, and Jakob Verbeek. Feastnet: Feature-steered graph convolutions for 3d shape analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2598–2606, 2018.

[177] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[178] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018, 2018.

[179] Chia-Hung Wan, Shun-Po Chuang, and Hung-Yi Lee. Towards audio to scene image synthesis using generative adversarial network. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 496–500. IEEE, 2019.

[180] Chu Wang, Babak Samari, and Kaleem Siddiqi. Local spectral graph convolution for point set feature learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 52–66, 2018.

[181] Ge Wang, Jong Chu Ye, Klaus Mueller, and Jeffrey A Fessler. Image reconstruction is a new frontier of machine learning. *IEEE transactions on medical imaging*, 37(6):1289–1296, 2018.

[182] Wanwei Wang, Wei Hong, Feng Wang, and Jinke Yu. Gan-knowledge distillation for one-stage object detection. *IEEE Access*, 8:60719–60727, 2020.

[183] X Wang, Y Peng, L Lu, Z Lu, M Bagheri, and R Summers. Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *IEEE CVPR*, volume 7, 2017.

[184] Mark Weiser. Program slicing. In *Proceedings of the 5th international conference on Software engineering*, pages 439–449. IEEE Press, 1981.

[185] Elaine J Weyuker. Evaluating software complexity measures. *IEEE transactions on Software Engineering*, 14(9):1357–1365, 1988.

[186] Terry Winograd. From programming environments to environments for designing. *Communications of the ACM*, 38(6):65–74, 1995.

[187] Michael J Wise. String similarity via greedy string tiling and running karp-rabin matching. *Online Preprint, Dec*, 119(1):1–17, 1993.

[188] Michael J Wise. Neweyes: a system for comparing biological sequences using the running karp-rabin greedy string-tiling algorithm. In *ISMB*, pages 393–401, 1995.

[189] Michael J Wise. Yap3: Improved detection of similarities in computer program and other texts. In *ACM SIGCSE Bulletin*, volume 28, pages 130–134. ACM, 1996.

[190] Youzi Xiao, Zhiqiang Tian, Jiachen Yu, Yinshu Zhang, Shuai Liu, Shaoyi Du, and Xuguang Lan. A review of object detection based on deep learning. *Multimedia Tools and Applications*, 79(33):23729–23791, 2020.

[191] Dianxiang Xu, Lijo Thomas, Michael Kent, Tejeddine Mouelhi, and Yves Le Traon. A model-based approach to automated testing of access control policies. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 209–218, 2012.

[192] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

[193] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1316–1324, 2018.

[194] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph r-cnn for scene graph generation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 670–685, 2018.

[195] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7370–7377, 2019.

[196] Jiexia Ye, Juanjuan Zhao, Kejiang Ye, and Chengzhong Xu. Multi-graph convolutional network for relationship-driven stock movement prediction. In *2020*

25th International Conference on Pattern Recognition (ICPR), pages 6702–6709. IEEE, 2021.

[197] Xin Yi, Ekta Walia, and Paul Babyn. Generative adversarial network in medical imaging: A review. *Medical image analysis*, 58:101552, 2019.

[198] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983, 2018.

[199] Fei Yuan, Longtu Zhang, Huang Bojun, and Yaobo Liang. Simpson's bias in nlp training. *arXiv preprint arXiv:2103.11795*, 2021.

[200] Clemens Zeidler, Christof Lutteroth, Wolfgang Stuerzlinger, and Gerald Weber. Evaluating direct manipulation operations for constraint-based layout. In *IFIP Conference on Human-Computer Interaction*, pages 513–529. Springer, 2013.

[201] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 5907–5915, 2017.

[202] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan++: Realistic image synthesis with stacked generative adversarial networks. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1947–1962, 2018.

[203] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23, 2019.

[204] Suiyun Zhang, Zhizhong Han, Yu-Kun Lai, Matthias Zwicker, and Hui Zhang. Stylistic scene enhancement gan: mixed stylistic enhancement generation for 3d indoor scenes. *The Visual Computer*, 35(6):1157–1169, 2019.

[205] Tianpu Zhang, Weilong Ding, Tao Chen, Zhe Wang, and Jun Chen. A graph convolutional method for traffic flow prediction in highway network. *Wireless Communications and Mobile Computing*, 2021, 2021.

[206] Yanlong Zhang, Hong Zhu, and Sue Greenwood. Web site complexity metrics for measuring navigability. In *Fourth International Conference onQuality Software, 2004. QSIC 2004. Proceedings.*, pages 172–179. IEEE, 2004.

[207] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.

[208] Hong Zhu and Patrick AV Hall. Test data adequacy measurement. *Software Engineering Journal*, 8(1):21–29, 1993.

[209] Zhihao Zhu, Zhan Xue, and Zejian Yuan. Automatic graphics program generation using attention-based hierarchical decoder. In *Asian Conference on Computer Vision*, pages 181–196. Springer, 2018.

[210] Vlad Zhukov, Eugene Golikov, and Maksim Kretov. Differentiable lower bound for expected bleu score. *arXiv preprint arXiv:1712.04708*, 2017.