

# Automated Feedback Test Generation and Functionality Testing for UI Development with Self-Guided Recommendation

by

Md.Tanvir Hasan

18101087

S.M. Hazzaz Durjoy

18101384

Jahidul Hasan

18101213

Mahazabin Binte Zafar

18101337

A thesis submitted to the Department of Computer Science and Engineering  
in partial fulfillment of the requirements for the degree of  
B.Sc. in Computer Science

Department of Computer Science and Engineering  
School of Data and Sciences  
Brac University  
January 2023

© 2015. Brac University  
All rights reserved.

# Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

**Student's Full Name & Signature:**

Md. Tanvir Hasan

---

Md. Tanvir Hasan  
18101087



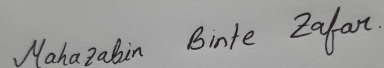
---

S.M. Hazzaz Durjoy  
18101384

Jahidul

---

Jahidul Hasan  
18101213



---

Mahazabin Binte Zafar  
18101337

# Approval

The thesis titled “Automated Feedback Test Generation and Functionality Testing for UI Development with Self-Guided Recommendation” submitted by

1. Md.Tanvir Hasan(18101087)
2. S.M. Hazzaz Durjoy (18101384)
3. Jahidul Hasan(18101213)
4. Mahazabin Binte Zafar(18101337)

Of Fall, 2022 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on 19 January 2023.

## Examining Committee:

Supervisor:  
(Member)



---

Dr. Muhammad Iqbal Hossain  
Associate Professor  
Department of Computer Science and Engineering  
Brac University

Co-Supervisor:  
(Member)

---

Name of Supervisor  
Designation  
Department of Computer Science and Engineering  
Brac University

Head of Department:  
(Chair)

---

Sadia Hamid Kazi, PhD  
Chairperson and Associate Professor  
Department of Computer Science and Engineering  
Brac University

## **Ethics Statement (Optional)**

This is optional, if you don't have an ethics statement then omit this page

# Abstract

UI development is the most integral part of the Software Development Life Cycle and testing the functionality of the UI is also as much important during the Software Testing Life Cycle of any software project. Without proper testing, we will remain unaware of how an application is working and in which scenario the application fails to do what was intended. Graphical User Interface is directly related to user experience and pretty much determines the future usage of that application, so it is important to make the application responsive, user-friendly, and simple with a proper functioning interface. UI development and testing the functionality is very important and also a time and resource-consuming process as the GUI has to be fault free. To do that, every permutation of the GUI functionality needs to be tested to launch the application in time. An android application consists of many core and composite components and to make such a seamless and fault-free application, every component's functionality has to fulfill its required functions. On one hand, testing the functionality of these components is necessary, on the other hand, the position of these components on the user interface is also as much important. As a result, a recommendation system is a must for the UI developer to save time during UI development which will make it easier for the developer to place each component in the correct position. Testing out all the functionality of a component is time-consuming. Even though some tools make the testing easier, the necessity of an automated tool is felt during the UI development that can make sure that the UI developer is getting necessary recommendations of the component's position and also able to check for himself if the components are reacting as it was intended to do. As a result, a huge amount of time and resources will be saved during software development. So in this paper, we intended to work on proposing a system that may give us the privilege of UI development and component functionality testing automation. functionality testing automation.

**Keywords:** Software Testing, Functionality Testing, YOLOv5, Object Detection, Computer Vision, Feedback Test Generation, UI Development, Self-Guided Recommendation, Android Application, Machine Learning.

## **Dedication (Optional)**

To our beloved parents and those who believed in us.

## **Acknowledgement**

First and foremost, thanks to the Great Almighty, our thesis was completed without any serious setbacks.

Moreover, we are thankful for the utmost contribution, time and support from our respected supervisors Dr. Muhammad Iqbal Hossain

Finally, to our parents and loved ones for their unwavering support.

# Table of Contents

Declaration	i
Approval	ii
Ethics Statement	iii
Abstract	iv
Dedication	v
Acknowledgment	vi
Table of Contents	vii
List of Figures	ix
List of Tables	x
Nomenclature	xi
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Problem Statement . . . . .	3
1.3 Objective and contributions . . . . .	4
1.4 Thesis Structure . . . . .	4
<b>2 Related Work</b>	<b>6</b>
2.1 Literature Review . . . . .	6
2.2 YOLOv5 . . . . .	17
2.3 Pytorch Framework . . . . .	18
2.4 Computer Vision . . . . .	18
2.5 Object Detection . . . . .	19
<b>3 Model &amp; Dataset</b>	<b>21</b>
3.1 Dataset description . . . . .	21
3.1.1 Data preprocessing . . . . .	21
3.1.2 Feature selection . . . . .	24
3.2 Model description . . . . .	24
3.2.1 Single Stage Object Detector . . . . .	24
3.2.2 Other important part for improved result . . . . .	25



<b>4</b>	<b>Implementation and Result Analysis</b>	<b>26</b>
4.1	Implementation . . . . .	26
4.1.1	Hardware Specification . . . . .	26
4.1.2	Environment Setup . . . . .	26
4.1.3	Package Installation . . . . .	26
4.1.4	Custom Model Configuration . . . . .	27
4.1.5	Automation System . . . . .	28
4.2	Result Analysis . . . . .	28
4.2.1	Comparative Analysis . . . . .	28
4.2.2	Selected Model Result Analysis . . . . .	30
4.2.3	System Analysis . . . . .	33
4.2.4	Performance Analysis . . . . .	37
<b>5</b>	<b>Conclusion</b>	<b>40</b>
5.1	Challenges . . . . .	40
5.2	Future Prospect . . . . .	40
5.3	Conclusion . . . . .	41
	<b>Bibliography</b>	<b>44</b>

# List of Figures

3.1	Summary of RICO dataset . . . . .	21
3.2	Labeling images using LabelImg package . . . . .	22
3.3	Data after Image Labeling . . . . .	23
3.4	Visual Representation how labeling works . . . . .	23
4.1	Comparison Graph of YOLOv5 Variants . . . . .	30
4.2	confusion matrix of our custom model . . . . .	31
4.3	System Overview . . . . .	32
4.4	Detection(Left side - the Application, Right side - Detection) . . . . .	33
4.5	No recommendation Perfect case (Left side - the Application, Right side - Recommendation) . . . . .	34
4.6	recommendation Perfect case (Left side - the Application, Right side - Recommendation) . . . . .	34
4.7	Functional Testing Error Check Result (No Error) . . . . .	35
4.8	Functional Testing Error Check Result (Error Found) . . . . .	36
4.9	Result . . . . .	37
4.10	F1 - confidence curve . . . . .	37
4.11	Precision Confidence Curve . . . . .	38
4.12	Recall Confidence curve . . . . .	38
4.13	Precision Recall curve . . . . .	39

# List of Tables

3.1	Features . . . . .	24
4.1	Component division on Train and Validaion . . . . .	29
4.2	Confusion Matrix Results . . . . .	30

# Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

*AUI* Attachment Unit Interface

*AUT* Application Under Test

*BCE* Binary Cross Entropy)

*CIoU* Complete intersection over union

*CNN* Convolutional Neural Network

*CPU* Central Processing Unit

*DRL* Deep Reinforcement Learning

*GPU* Graphics Processing Unit

*GUI* Graphical User Interface

*IDE* Integrated Development Environment

*LOC* Lines of Code

*mAP* Mean Average Precision

*MBT* Model Based Testing

*QA* Quality Assurance

*RAM* Random Access Memory

*SDLC* Software Development Life Cycle

*Silu* Sigmoid Linear Unit

*STLC* Software Testing Life Cycle

*UI* User Interface

*URL* Uniform Resource Locator

*UX* User Experience

*YOLO* You Only Look Once

# Chapter 1

## Introduction

The UI development phase is the most important phase among other phases of software development. Whether or not it will draw people and keep them from using the application depends on the user interface development phase. Creative and better usable UI is what ensures people to recommend others to use your application and be loyal users. A well-designed UI is capable of making any application easy to use and navigate as the user pleases without getting lost in the application. It also makes the application more enjoyable, and efficient which leads to a positive user experience. On the other hand, a poorly designed UI can be frustrating to use and a user may not come back again to use that particular application and also may hinder the efficiency of any work that is done on the application. A good UI, on one hand, improves the user experience as well as on the other hand increases the productivity and efficiency of a user.[9] There are different components of an android UI. These components work according to how the developer has programmed them to do. After developing the UI, a functionality test, which is also known as black box testing, is conducted on it to see whether all the components are working or not. If not, then feedback is given to the developer to improve them according to feedback. Buggy and non-functional components can ruin the user experience and users may not think about using the application again.[33]

During UI development, the developer must keep in mind that the UI should be easy to use for the user. To ensure that, the developer must keep in mind the placement, color contrast, and size of the components that are being placed on the UI. Many core and composite components exist in android applications. Among them are back buttons, settings-button, share-button, cross-buttons, check-box, switch-button, search-button, and so on. The placement of these components on an android application UI is very important as well as the functionality of these buttons. Because these buttons should be easily accessible and placed in such a position that the user will not feel any discomfort pressing and doing the functions that these buttons are programmed to do without any error. This is something that the UI developer must keep in mind while designing the application user interface and also while implementing the component's functions. Every UI that is developed by the UI developer, is evaluated by the QA team before any further development is done. It takes quite some time before the QA team finalizes the UI design and functionality of these components, several steps are taken and feedback is sent back to the developer to make certain changes about the position, color, and other things about the components depending on the requirement which is a time-consuming process

and takes about 2-3 weeks, 5-6 weeks and 9-10 weeks for small, medium and big size applications respectively and 20 to 30 percent of the overall development time is estimated to do testing such as functionality testing, stress testing and so on.

The purpose of our research is to mitigate this time-consuming process and automate this so that the UI developer can get proper auto-generated recommendations about the core and some of the composite component placement so that he can design and develop the UI according to the requirements that were set in term of the project and also perform the functionality test by himself without having any testing knowledge to ensure that the components that he placed on the UI are working as they were programmed to do and finalize the UI way before the deadline of the UI development.

## 1.1 Motivation

Software Development Life Cycle is a process that is followed while developing a software project. SDLC consists of a well-detailed plan which describes how to develop, maintain and enhance the software. This software development life cycle ensures the quality and overall development process. On the other hand, STLC is the software testing life cycle where the software goes through various testing phases to ensure that the software is bug-free. There are various stages of a typical software development cycle. These are planning, defining, designing, building, testing, and finally deployment. Planning and requirement analysis being the most important as well as a fundamental stage, inputs from customers, sales department, and some other entity is taken to plan the basic approach. After this stage, these requirements are documented and defined, and approved by the customer or by the market analyst. After the requirements are finalized, a design team draws up a mockup and sends it for approval and after being approved this design is sent to the UI developer to start designing the application. During the UI development phase the developer being completed with the design, sends it to the QA testing team where the UI is evaluated and tested on various test cases and eventually is sent back with additional feedback to the UI developer about the components design, placement, visibility and functionality and so on. After that, the UI developer makes changes according to the feedback and again sends it to the QA tester for approval. After the UI is finalized, further developments are done, and then again after proper testing on the application it is deployed in the market.[32]

The UI design is the most crucial part of the whole software development process. While developing the UI, the developer must send the developed UI to the QA team to ensure that the UI is up to mark. This means if this UI has met the requirements or not and if it is properly functioning. Otherwise, feedback is sent to the UI developer to make those changes according to the QA tester. While the UI is evaluated by the QA team, some things are taken into consideration and among them, the placement and the functionality of the components are crucial parts. If the components are not placed in certain places of the UI, the interface may become difficult for the user to use. In order to ensure that, the UI developer has to send the user interface design to the QA team several times for weeks before the UI is finalized. And another QA tester does a functionality test on the UI to see if the components are behaving the way they are supposed to. And this process of getting feedback and making changes takes place several times before the UI is finalized and further development is done on that project. A tremendous amount of time

is taken in this process while the UI developer goes back and forth with the QA tester teams to finalize the UI. This process takes a lot of time from the total time that was given for the development of the whole application. Additionally, further development and testing are required to release the application on the market which takes about 20 to 30 percent of the overall development time given. Furthermore, it is also seen that in some cases, while the deadline is way past over, still the testing phase is not completed. As a result, the deployment of the application is delayed. In our research, we are focusing on giving self-guided recommendations to the UI developer and an automated functional testing tool so that the developer can place each component in the proper position on the UI and also check if the components are working properly or not by the click of a button. One of the important factors that the UI developer must keep in mind is the position and the functionalities of the core and composite components of any android application. Thus we are focused on creating such a system that can give the UI developer recommendations regarding the positions of the android core and composite components which he can follow and design the UI and also conduct an automated functional test of the components very quickly so that during the testing phase it doesn't take so much time for the tester to finalize and deploy the application.

## 1.2 Problem Statement

As we can see, during the UI development, in order to check if the UI design has met the requirements or not, the developer sends the developed UI to a QA testing team and after the design is evaluated and approved, another QA testing team does a black-box testing also known as the functional testing on the application to see if the components are behaving as they should. A lot of processes and steps are involved from design approval to black-box testing. Usually, 20-30 percent of the total development times are given to testing but most of the time it has been seen that the deadline passes but the necessary testing is not done yet on the application. Furthermore, after testing if errors are found, feedback is sent to the developer to make changes and improvements according to the feedback. And thus, it is again sent for testing. This keeps going on until the QA testing team finally clears the application for deployment. It is safe to say that a tremendous amount of time is needed to finalize the application design and to conduct proper testing.

During software development, many automation tools are available to make the process easier and to save time. Such as a tool like Jira that can be used to do requirement gathering and Gherkin can be used to manage the requirements. Additionally, many intelligent IDEs are available to coders which help them to auto-generate codes and solve errors. Furthermore, tools like Taurus and Junit are used to do Unit Testing of python and java applications respectively. Similarly, Selenium is an automated web application testing tool that is used to write functional tests automatically and Ranorex is also an available automatic functional GUI testing tool. Besides, there are tools like Fabric, Packer, and Docker which are used to automate the deployment of an application.[30]

So as we can see, most software development phases have different tools that help automate the processes in order to save time. But during UI development, there are no such tools or methodology that can mitigate some of these stages where the UI developer has to get feedback from the QA tester about components placements and

also could conduct quick functional testing on the components in order to see if the components functional code implementations that the developer has programmed, works properly or not and make those improvements that are required.

Therefore the main question and the problem that we are trying to address is that, How can an UI developer use automation during UI development to mitigate the various steps involved in finalizing the UI and component functional testing by the QA testing team and save time from both SDLC and STLC phases thus deploying the application in time?

### 1.3 Objective and contributions

If we think about overall software development we see that UI development, backend development, and database design take a lot of time. And even after that rigorous amount of testing is needed to make sure that the application is properly usable or not. During this testing phase the bugs and errors that are found, need to be improved by the UI developer, backend developer and database deployment team, and again sent for testing. This process goes on and on until the application has no error and is approved for deployment.

GUI testing automation is a big sector. But testing comes after the development is done. Without the development of an application being completed, the QA testing team does not conduct tests to ensure the quality and usability of the application. No automation tools are used during the development phase by the UI developer who designs and implements the functionality of these android core and composite components. As a result, the UI developer has to check by themselves if the implementations of the component functions are working or not, one by one during developing the user interface. Our main objective of this research is that a UI developer can use these automation tools and get the recommendation of the position of the component and conduct auto-functional testing without sending it to the QA testing team, hence saving so much time from the software testing life cycle.

### 1.4 Thesis Structure

Our research paper starts with introduction talking about how SDLC and STLC plays a vital role in any software project and successful deployment. Behind all research there is a motivation that drives the researcher to conduct through research and complete the paper. In the motivation part we have explored the real reason why we have taken such an aspect to conduct our research. Furthermore, we have identified the research problem and what are the objectives that we are pursuing and how our research can contribute to this very aspect of software engineering. Moving on to the Background chapter, we have conducted a through literature review on research papers that align with our research and did a detailed summary. Additionally, we have explained some methodologies such as YOLOv5, object detection, pytorch framework and computer vision that are some of the important aspect of our research. Moving ahead, in the Model and Dataset chapter, we portrayed how the dataset was collected and pre-processed for further training purpose and a detailed statistic which shows the components details and the amount of component found in the dataset. Further going into the paper, we have described the model in similar



details as the features that YOLOv5 uses to produce their better accuracy. In the Implementation and Result Analysis chapter at first we have explained in details the total implementation guide for someone who might go through our research paper to implement the system step by step. Additionally, we moved into result analysis where initially we explained our reasoning for choosing YOLOv5s as model for our system and did a comparative analysis of all the variants of YOLOv5 trained with the same dataset. Furthermore, we have showed the results and other metrics showing how YOLOv5s model performed and showed the test results step by step. As the last part of this chapter we have put down the various performance metrics which portrayed the performance of YOLOv5s model. Lastly, in the Conclusion chapter we talked about the various challenges we faced while conducting our research and what are the future prospect that our research can possibly fulfill.

# Chapter 2

## Related Work

### 2.1 Literature Review

The authors in this paper stated that for many GUI automation and GUI testing jobs, the identification of graphical user interface (GUI) features are essential. According to the authors the initial step in performing GUI testing or GUI refactoring is obtaining the precise positions and categories of GUI objects. The authors of this paper constructed User Interface Element Recognition (UIED), a toolkit created to offer users a straightforward and user-friendly framework for successful GUI component recognition. To accommodate various and complex GUI pictures, UIED includes many classification technique, incorporating traditional computer vision (CV) algorithms and deep learning frameworks. Additionally, it does have cutting-edge detection approach that are innovative and specifically tailored to GUI elements. The recognition outcome can be changed as well as edited over an interactive map using the author's toolkit. In the end, it outputs the identified UI parts from the GUI image into configuration items which may then be subsequently altered in well-liked UI design programs like Sketch and Photoshop. According to evaluations, UIED can recognize objects accurately and is helpful for subsequent operations. When the tool is ready, it will be integrated with their current UI2CODE research, which seeks to generate code from a specific GUI picture, for use in continued expansion. By allowing designers to submit new GUI designs to their system and quickly receive useable code, computation will significantly aid GUI development[20].

In this paper, "Fragility of Layout-Based and Visual GUI Test Scripts: An Assessment Study on a Hybrid Mobile Application", the authors The fragility of the methodologies, i.e., the frequent requirement for updating test cases whenever the application's user interface (UI) changes, was cited by the authors of this research as a potential explanation for why developers don't employ automated GUI testing. The authors of this research set out to quantify the upkeep required by test scenarios for hybrid mobile apps and to identify related fragility reasons. They used both layout-based and visual tools for evaluation. They discovered that 20% of layout-based test techniques and 30% of visual testing methods required at least one revision. Moreover, 3-4% of the test procedures had fragilities introduced. There are three basic methods for creating mobile apps: hybrid, web-based, and native. While hybrid apps only need to be maintained for a small number of native components, native apps must be developed and updated independently for each OS. The benefit of developing hybrid mobile applications is that it requires less work to cre-

ate cross-platform applications. Cordova, Xamarin, Flutter, React Native are few examples of frameworks that exist for the development of hybrid apps. But hybrid app comes with the cost of little bit of performance reduction as well as reduction of feature for specific platform and sometimes with less user fondness. High usability and few flaws should be promised by well-designed apps to their consumers. It is crucial to thoroughly evaluate their GUIs (graphical user interfaces), which are how the majority of their functions are shown. Testing tools are not widely used for all types of mobile applications. Because of the constant growth of their GUI, mobile apps are particularly vulnerable to fragility. We have compared the two strategies in terms of both the convenience and effectiveness of test suites given on a native software. The authors employed two distinct methods to examine the GUI testing procedure in the domain of hybrid mobile applications. Both are third generation or graphical based (EyeAutomate) and second generation or layout based (Appium). The study's findings are interpreted in accordance with the needs of developers of Android apps and testers as well as experts working to address problems with the tools or approaches employed. The study is conducted on the hybrid app PoliTO App, which was created using Apache Cord and made available on the Google Play Store, App Store as well as Microsoft Store. Students, professors, and researchers can log in to the app to access a variety of functions, including news, maps, class schedules, issue reporting, and more. degree programs and details on university employees. The authors tried to get answers to three questions in the paper. First question was, if the suites written in layout based and visual based tools spot the malfunction in the app, the amount required for the maintenance effort needed for the used tool and the kind of fragilities that are encountered during the evaluation. After doing an evaluation, the authors came to the conclusion that the Layout-based and Visual test suites, respectively, required at least one change to 20% and 30% of the test methods. By assigning unique IDs to each widget in the GUI structure of the AUT, fragility may be easily avoided. In order to make changes, the authors said they intended to allow the conversion of Espresso test cases for native languages to EyeAutomate and Sikuli scripts. Additionally, they intend to create automated tools that will automatically fix the locators or oracles that require layout-based and visual tests. [12]

The authors bring attention in this paper, "Fastbot2: Reusable Automated Model-based GUI Testing for Android Enhanced by Reinforcement Learning", industrial apps undergo regular updates to meet evolving consumer requests and ongoing testing is essential for prompt feedback of app quality. Existing testing methods, however, are inefficient and useless since they just run each version of the test from scratch without using the results of earlier tests. At ByteDance, the authors introduced a reusable automated tool that leverages model-based testing (MBT) to accelerate development of industrial apps. To solve the problem the first challenge they had to solve was to store knowledge from previous run effectively. They suggested a probabilistic model for this, on which MBT may be developed to remember similar information from every test cycle. How to properly use the previous knowledge to direct GUI testing presented the writers with their second problem. To produce GUI events, traditional MBT methods go across the model. The essential discovery is to use a statistical model to carry out supervised model-based assessment immediately. The authors suggested a model-based, automated, and reinforcement-enhanced GUI testing method. acquiring the necessary practical skills for extensive testing. Their

final version, FASTBOT2, performs better than modern MBT tools, Stoa and Ape and has additionally been deployed successfully in pipelines of the CI at ByteDance. After evaluation the authors found that their version FASTBOT2 obtained the maximum activity saturation in each individual version as well as the greatest amount of aggregate exposure over all ten iterations of both applications. For Douyin, FASTBOT2 was able to detect much more crashes than all other tools. This further proved the point that the authors tried to establish that retaining knowledge from previous test run can perform more coverage.[27]

In this paper, “Improving Crowd-Supported GUI Testing with Structural Guidance”, the authors stated that crowd testing for Graphical user interface is easier as well as faster than a dedicated team that checks the quality of the software. In crowd testing a large number of crowd testers are hired by the developers to test the GUI of the software. But the problem that arises with crowd testing is the low test coverage due to the focus being on insufficient number of common UI navigation paths. This approach has the ability to reduce redundant effort for the GUI testing procedure. To solve this problem of low coverage of crowd testing the authors suggested two approaches. They are: Interactive event-flow graphs and GUI-level guidance. For the Interactive event-flow graphs, it remembers the already explored cases as well as it aggregates each tester’s interaction and visualizes the whole in a graph that is directed in a single direction. This opens a path for the crowd who tests the software to interact with provided graphs to make unexplored navigation for the paths, thus improving the coverage of the test cases. For the use of graphs to augment the GUI (GUI-level guidance), the authors used this process to guide the testers to avoid only exploring the paths that are commonly explored. The method the authors provided was able to effectively assist the testers in avoiding wasting time on pointless test cases, according to the assessment the authors conducted with 30 crowdsourced volunteers on 11 distinct test websites. While redundancy was decreased, the methods were also able to achieve the coverage of untrained testers by 55%. The authors suggested that these methods will be really beneficial for the development of more robust software that can perform for missions in more critical settings which will increase the efficiency. These efficiencies are achieved not only via testing more thoroughly. The authors also integrated the two methods in various parts of the pipeline of the development. This ensures the development of a much more reliable software in the early stage of the development phase.[16]

In this paper, “Scripted GUI testing of Android open-source apps: evolution of test code and fragility causes”, the authors stated that from the existing evidence it is proven that the mobile apps don’t get as much attention as their desktop versions. There is a great amount of lack of care in the testing of the mobile version. Even in all these the part that gets neglected the most is the GUI testing. Both web based and mobile applications suffer from the consequence due to testing fragility. The fragilities that the applications have to face are the failed GUI test class, sometimes the needed intervention as a result of the change in AUI or GUI arrangement as well as definition. In this paper the authors set the goal to determine what kind of diffusion in the test classes popular automated GUI frameworks for Android applications can generate. The authors also examined the amount that is necessary to retain the test classes updates as well as the number of code churn that exist in the test suites. Finally they also checked the amount of code churn along with the modification that happens in the AUI which resulted in the modification in the first place. In addition

to a taxonomy of 28 potential reasons for modifications to test code, the authors established 12 metrics to describe the evolution of test classes and test methods. For performing the experiment the authors selected six popular open-source GUI Automation Frameworks that are used in Android apps. By mining the GitHub repositories that contained the tools, the authors were able to assess their dissemination and generated their own set of metrics for the projects. The authors used the Grounded Theory technique to manually review various test documents written with the selected tools in the following section in order to establish a classification of causes for modifications to test code. After conducting the experiment the authors discovered that none of the GUI automation frameworks under consideration had a significant level of adoption among the open-source Android projects that are hosted on GitHub. The authors discovered that test suites needed modification frequently for projects that used tests made with the chosen frameworks. They found that around 8% of modified LOCs that were modified by developers belonged to test code. Furthermore, around 50% on average of those modifications were the result of changes in GUI definition and arrangement. Moreover, the evaluation shows that in the course of a standard Android open-source project, test code that used GUI automation frameworks required considerable interventions. For future work the authors intend to create automated tools that can identify patterns associated with fragility when changes are made to production code such as rearranging the GUI, renaming widgets, or altering the text shown to the user. Furthermore, these tools can also alert developers to possibly fragile classes and methods in the test suite. They plan these tools to be incorporated as plug-ins for well-known IDEs (e.g., Android Studio). The authors wish to incorporate testing of Android applications created with hybrid frameworks and apply the evolution and fragility metrics to other test methods, frameworks, and platforms (e.g., Flutter or Cordova). Finally, rather than just focusing on Android (or mobile, in general) apps, the classification of test management causes will be extended by the authors to include any GUI-based applications.[13]

In this paper, "Comparing the effectiveness of capture and replay against automatic input generation for Android graphical user interface testing", the authors stated that the two practical and affordable as opposed to conventional test-case-based techniques for graphical user interface testing of Android apps include test exploration and completely automated testing tools. According to the authors, without the need for extensive programming or testing expertise or prior test case construction, the test can be performed using capture and replay technologies that directly transform circumstances of the execution that are registered by testers while testing the test cases. The main advantage is that without the need for a tester, the latter tools can test Android GUIs. The authors discover that even though both of these methodologies are frequently used, no empirical research has been done to compare their effectiveness and provide insights that might help a project manager create a successful testing strategy. The authors discussed two studies that they ran to contrast the efficiency of exploratory testing methods that uses a capture and replay tool (Robotium Recorder) versus three free automatic testing tools (AndroidRipper, Sapienz, and Google Robo). The authors asked 20 students majoring in computer engineering to record testing executions while being restricted by time limit and denied accessibility to the source code. This experiment led the finding that the results that were done by those students were marginally, but not significantly, superior to

those of fully automated tools. After that the authors instructed again the students who were instructed before to increase the testing coverage gained by utilizing the source code and with a less time constraint. They were also allowed to use coverage of earlier tests. The findings of the second trial revealed that, particularly in long/complex execution circumstances, students outperformed the automated tools. The acquired results offer helpful cues for selecting testing approaches that incorporate both qualitative testing process and automation testing. Additionally, the authors carried out a thorough qualitative review of the coverage gap discovered by the testers as well as the AIG tools. They were able to look at both methods' weaknesses towards this analysis. They discovered that a time constraint and a deficiency of knowledge about the AUT are biggest barriers to the performance of human testers. The AIG tools' efficiency was constrained by methodological and technical errors. The technical problems are caused by incompatible GUI events and widgets and may be quickly fixed by technical tool improvements that are tailored to the ongoing development of the Android foundation. For their future work, the authors intend to recreate the experiment in an industrial setting and assess how the testers' proficiency level affects the efficacy of existing test suites they generate. Additionally, the authors plan to design and conduct additional research to compare the expenses of using automated technologies with manual testing. They also have the intention to take into account additional testing efficiency indicators, like the ability to discover faults.[21]

In this paper, "Automating GUI Testing with Image-Based Deep Reinforcement Learning", the authors stated that GUI is so important because visually interfaceable systems are necessary for the users to interact between current software applications and accessories. This is a reason why the GUIs has to be checked, and this helps the app devs to look for probable malfunction and lack of consistent functionality, to make sure of easier and reliable use case for the users. The authors drew attention to the fact that automation in GUI testing can increase the efficiency of manual GUI testing because it saves time and effort. The authors also admitted that even though regular GUI testing technologies that are automated reduce the necessary work of testing manually, most of the time they create continuity problems for cases that are generated for testing which lead to less efficiency and higher cost in the long run. The authors approached these difficulties by providing a deep reinforcement learning-based (DRL) approach in the case of adaptive and automated GUI testing to address these problems. The authors suggested assessing the effectiveness of an image-based DRL approach. Then they modified the A3C technique for GUI testing in order to simulate how a real person might interact with a GUI. The authors fed the algorithm GUI screenshots as source and then allowed the algorithm to decide the process it wanted to use to engage between the elements of the GUI. The authors note that when compared to a few baseline techniques, their solution can yield exploration efficiency gains of up to six times. Furthermore, in their experimental GUI testing environment, their solution outperforms unfamiliar human testes and is nearly as effective as a familiar human tester. In a period of 10,000 completed actions, the author's DRL resolution finds six times as many distinct application situations than Q-learning and random testing, and four times more distinct URLs in a duration of 100 clicks than Q-learning and random testing. The authors' approach allowed them to find a few new vulnerabilities that traditional automated tests were unable to find, like those brought on by repeatedly moving sequentially

away from and back to the same display. These factors led the authors to claim that image-based DRL discovery is possible using as a practical GUI test methodology. The authors wished to compare their solution to existing automated GUI testing techniques, like Humanoid and QDroid, for use in future work. In addition, they plan to investigate the potential for identifying logical mistakes in addition to the present capacity for detecting crashes and unhandled exceptions. Last but not least, the authors also consider the prospect of increasing the human benchmarking, leaving it to further studies to expand the amount of people whose testing capability is analyzed in order to attain a more universal human standard against which they might measure their methodology.[17]

According to the authors of this research of the paper, "Benchmarking Automated GUI Testing for Android against Real-World Bugs", over the past ten years there has been significant, ongoing improvement in automated GUI testing, which has helped to ensure the dependability of Android apps. Numerous testing methods and tools, in particular, have already been created as well as shown that they are very efficient in identifying malfunction issues as well as surpass their corresponding preceding tasks in terms of the quantity of discovered malfunctions. But the authors wanted to ask how well and completely can these techniques uncover crash issues in actual use is the big question. They discovered that this hasn't been thoroughly investigated, necessitating a ground-truth benchmark using actual bugs. The authors state that previous researches aren't able to provide a straight, comprehensive solution to this topic as those just compare tools with a few particular apps. The authors provided the very first quantitative ground-truth assessment of autonomous GUI testing for Android to supplement earlier research and address the aforementioned query. To achieve this, the authors put a lot of manual labor into setting up the Themis benchmark set, which consists of two parts: (1) one cautiously crafted metadata containing 52 actual, reproducible crash bugs (which took two people several months to collect and validate); and (2) one coordinated, scalable architecture with six modern, cutting-edge testing tools. The authors spent over 10,920 CPU hours performing the entire evaluation. 18 of the gathered actual flaws that are impossible to find using any program., which the authors found to be a serious flaw in the softwares' ability to identify problems. This systematic research carried out by the authors additionally detects five significant regular issues that the existing softwares confront as well as yields further data regarding factors influencing these tools in bug discovery and prospects for tool development. Moreover, their task gives fresh, precise information, which are most frequently challenging to find as well as completely undiscovered. In order to comprehend and evaluate the efficacy of current testing instruments, this study offers a fresh perspective that is complementary to earlier research. It also serves as a baseline for further study in this area. In order to examine current testing techniques, the authors concluded that their study offers a fresh, complementary viewpoint from earlier research.[24]

The authors of this work present a novel, "Practical GUI Testing of Android Applications via Model Abstraction and Refinement", completely based on automated models, as an efficient way of approaching testing of Android apps. The authors' approach involved dynamically optimizing the model by utilizing the runtime information while testing, in contrast to current model-based solutions which use an invariable GUI model to guide testing because in static GUI testing model's abstraction does not change during testing, making it frequently inaccurate. The author's

study reveals that this ability of model evolution greatly increases model precision and, as a result, dramatically increases testing effectiveness when compared to previous methodologies. With APE, the authors have put their technique into practice. APE exceeds the most advanced Android GUI evaluation methodology in relation to coverage of the evaluation as well as which amount of unidentified crashes that were detected in 15 huge, popular application that are available the Google Play Store. The authors did another examination of APE on 1,316 well-known programs in order to highlight its efficacy and usability further. This evaluation discovered 537 distinct crashes. Their evaluation was able to detect 38 crashes that have been reported; 13 have been corrected, and 5 have been verified. The authors concluded that among 15 very commonly valid benchmark applications, APE regularly beats state-of-the-art techniques which don't require any model system nor founded by the basis test models that are static, in the segment of coverage of the code (14-78% comparative enhancement) and the quantity of issues discovered (61 distinct events as opposed to 31-44). According to the evaluation of the authors it shows that APE is efficient, useful, and promising. Currently, the author's industry partner has implemented APE and integrated it into their testing procedure.[14]

The writers of this research paper, "Artificial Intelligence in Automated System for WebInterfaces Visual Testing", take into account an AI method for providing visual testing as well as the systematic approach that is linked into useful automated test suites. Thus, visual modifications in the graphical user interface of the program being tested were monitored and analyzed. The authors offered a tool that is designed to address the issues with the current method of typical snapshot visual testing as a solution. Testing the graphical user interface (GUI) is a pivotal step when guaranteeing the standard of software applications as it ensures a good experience for the users. In the test program, the GUI serves as the main node from which all functionalities can be accessed for the user. Because graphical user interfaces are made to work with people rather than machines, it is challenging to adequately test programs using them. This leads to the reuse of test cases from earlier test runs almost impossible in most cases. Additionally, the authors noted that interfaces are fundamentally non-static and subject to constant modification as a result of functionality updates, enhanced usability, shifting requirements, or altered circumstances. Additionally, this makes it more difficult to design and maintain examples of tests without turning towards expensive and human inspection that takes a lot of time. The authors talked about a suggested automated web interface visual assessment system that compares images using machine vision capabilities also as an ai - powered technique. The generated interface for test validation (specifically, web pages) as well as the anticipated illustration featuring placement of the imagery components along the webpage, for instance, a customer terminal, are compared. To perform the experiment the author's used scripting dialect Python, JavaScript, library TensorFlow, testing environment Cypress, and database MySQL for creating an automated system for visual web interface testing. The authors covered the theoretical underpinnings of automated visual assessment in the first section of the study, as well as the methods and tools that are already in use. The authors conducted a thorough review of the fundamental methodologies and methods for machine vision testing, including pixel-by-pixel picture comparison and ways for element style verification, for the following sub-task. The authors talk about picture segmentation clustering techniques (K-Means and Mean Shift). The authors did a



thorough examination and comparative analysis before choosing their image processing techniques. To develop a truly automated GUI testing system the stages and aspects of developing a web application were taken into account, software for the creation for the right methodology had been chosen, picture segmentation methods had been used, the methodology design had been established, one method to use as imagery test method had been created, as well as the effectiveness was evaluated thoroughly. This helped the author to create automation process for imagery evaluation of web junctions employing artificial intelligence techniques for photo classification as well as much more accurate exploration for the outcome of this paper. [18]

With 85% of the Smart Mobile OS Market Dominance, Android platform of Google leads mobile operating systems industry, according to findings of the authors. Android applications frequently allow end users to engage with them via a graphical user interface (GUI). The authors stress that any software's lack of reaction ability in an effective way to GUI operations frequently prevents it from carrying out a user's objective. GUI testing may be carried out manually or automatically using test scripts. According to a study the authors reviewed, Only 14% of the more than 600 open-source Android software projects have cases for test generation, as well as mere 9% of those have viable evaluation scenarios holding more than 40% code coverage. The authors emphasize the necessity for ongoing study as a result of lack of test code coverage of current methodologies, even though research has already been done on building automated GUI test methods for Android applications. Applications for Android are created using certain elements of the Android programming framework. Through a variety of input devices, including physical keyboards, software keyboards, and touchscreens, users can interact with Android applications. Numerous motions, including tap, drag, slide, pinch, and rotate are supported by Android. The authors claimed that in order to design automated testing methods for Android applications, it is required to take into account these interaction mechanisms as well as the distinctive design of the Android framework. According to the authors, one common method for evaluating Android applications is random test generation. Giving the Application Under Test (AUT) random GUI events as it runs is a straightforward strategy. This method has the drawback that portions of the program that need more in-depth testing could not get as much attention as those that have been investigated already. This issue is made much worse when the AUT needs to repeatedly execute particular occurrences in order to navigate previously undiscovered areas of the application. In contrast to random test generation, the author of this study used reinforcement learning using trigger preference interrogative to methodically produce test suites accompanied by better coverage of the code. Trial-and-error interactions are used to register occurrences which are most probable to uncover new conditions as well as return incomplete levels of exploration in their proposed Q-learning-based technique. An episode is selected by the Q-learning algorithm using a given set of events in a given state based on linked rewards earned from prior interactions. In this method events that have not yet been investigated offer greater rewards and higher priority when it comes to being chosen next. The first thing that the authors contributed in this paper is Automated GUI test case generation and evaluation of Android apps using reinforcement learning as well as a Q-learning-based test generating technique. Secondly they performed an empirical study that contrasts coverage of the chunk of the Q-learning-based approach as well

as randomized case creation for testing test creation in eight Android applications. They addressed the aforementioned shortcomings of randomized case creation for testing in domain of GUI-based Android applications by using reinforcement learning methods. Without the necessity for an existing abstract model of the AUT, the author's test creation approach leverages dynamic event extraction to choose events from the GUI while the application is running. The authors proposed that additional AUT components would probably undergo more comprehensive Q-learning testing. The authors observed that their suggested approach produces test suites which provide in the range from 3.31% and 18.83% better block-level test insurance as opposed to generating test cases in a randomized way, after empirically analyzing the methodology on eight Android applications.[9]

In this paper, "Test - Duo : A framework for generating and executing automated acceptance tests from use cases", the author explained that acceptance testing generally denotes a test process that ensures whether the system will be accepted by the user or not. Even if we use tools for acceptance testing the errors can be massive. Acceptance testing tools have been improving in recent years. Even after that the possibility of failing acceptance tests is huge. It is not only prone to error but also requires a lot of labor. Tools like FIT/FitNesse, Robot Framework have been used to reduce the labor and automate the task of test running and collecting information. The main objective of the paper is to leverage automated acceptance testing platforms by further automating the tracing of another task as a sequence of execution steps. It will take in data and provide the result for individual scenarios. The approach that has been used by them is - the test cases are made clear by breaking down every step of the use case. The data that are collected and will be given as input will be pooled. The anticipated results are noted down after that. The framework that they have worked on works in an iterative way. Then the uni tests go through a selective search regime under the selected platform. The test duo is responsible for recording each sequence step after the completion of the single steps fails. Then it fixes the set and provides that as output. First steps are not automated and are done manually. The last two steps are automated. The test duo framework works due to collaboration between test director and test driver. While the test detector sends the command as its step, the test driver waits for the command to get started with its step. In the test director framework test the inputs and desired result goes to the test steps. Then it commands for directing the test drive. Test driver receives commands from the Robot framework implemented on top of it. The test director generates failed cases and uses them as test cases as well whereas the test driver takes in command and uses the iterative way to perform command as test keyword. State space scheme is operated by the test director to be able to produce test steps. The initial node starts with the start node and branches down to use cases. The state cases branch out and every arc represents a different arc directed by the test duo framework. The node can be true or false. The goal nodes are nodes that receive assertion failure. The objective is to start from a node and end at a failure node. The failure node is the expected result in this scenario. The method of state space search can be any algorithm. It can be depth first search, breadth first search, iterative deepening. The objective is to find the shortest path to the failed sequence. The test drive being a fixed ROBOT framework is responsible to collaborate with the test director. A custom keyword is being fetched for delivering commands. Upon receiving the command the test driver sets up, tears down or ter-

minates the test. From the keyword table the keywords are matched and are hence given to the Robot framework. The data given in input is then used as a parameter to check the desired result. Static parameter and dynamic parameter are used as the main keyword for the framework. If in the annotation phase the keyword can be accessed then it can be created statically. If the data can be generated logically it will be termed as dynamic data. This way by generating the keyword and calling functions the expected result is found out. The test duo framework hence works to automatically generate acceptance testing.[2]

In this paper, "Identifying Infeasible GUI Test Cases Using Support Vector Machines and Induced Grammar", the author explained that Model based GUI testing is the gateway to automated test cases generation. Test cases are the ways to find out the faults and errors of an application. After the completion of a project it is run through a series of tests to ensure that the application is working according to the required specifications. But test cases terminate or have possibility of termination if one or two events aren't working or are not accessible. This creates a lot of waste resources due to inefficiency. Which in terms had made the developers try and find a way to produce a scenario which will allow them to have accessible and feasible test cases which will not waste their resources. Feasibility has always been a priority in these test cases. But to make the test cases feasible all the test cases are to be written, then executed and then the developers get to figure out the test cases that aren't working and then it can the process is run recurringly until other factors terminate this test cases. In this paper they have shown how to avoid inefficient, infeasible error prone test cases. They have used two supervised machine learning processes to support this idea that they have come up with. The two proposed methods to be taken in action support vector machine and grammar induction. Three feature extraction techniques had been implemented to find out infeasible GUI test cases in various applications. They also tried to use the algorithm to train the model to categorize the test cases of various and all lengths. They have used the SVM implementation in Matlab. The function `svmtrain()` and `svmclassify()` are used to implement the course of action. Gaussian Radial Basis Function kernel with a hard scaling of factor of 1 is used in the functions. This function allows more detailed and more analyzed and more complex margin to carry out the kernel function. SVM is applied following the normal steps that are normally being performed in these cases. Standard approaches for all machine learning techniques are executed in the SVMs which is most commonly well known to be called data modeling phase. Data is modeled in a way where the data can be used in a particular way that will provide the best output of the scenario. Even though SVM requires its data to be actual vectors, test cases on the other hand are clusters of IDs that are converted into actual vectors. They were able to extract three other algorithms which help them create vectors. The vectors were used to create three categories of SVM input. They were called Basic, Pairwise and Full Pairwise. The production of the three categories are done in four stages. Stage one is responsible for assigning values for the first N vector attributes. N here denotes a number of different IDs. Where i stands for index and index denotes the number of times an event i appears in the test cases. From this stage we will get the result as a N-dimensional Basic vector. This obtained result will be then used as input in Pairwise and full pairwise vectors. A string of events will let us know if the test case is infeasible. The prediction of the infeasibility of an event will be deduced from the next event that follows. Initially the test cases

use regular expressions. Then in the later phases the test cases keep changing the expression iteratively by returning a certain expression following a regular pattern. After the iterative changes finally fail to change further then only the set is declared as final set and is added to the set of regular expressions. The usefulness of utilizing two supervised learning algorithms are described in this paper elaborately. The two supervised algorithms are support vector machines, and induced grammars. Even though the results of induced grammars are really limited because rigorous use of this algorithm will not be cost effective for us, induced grammar was efficient in finding out the infeasible test cases more effectively than all the other types of algorithm in use. A cost effective and optimized grammar induction will be really helpful when it comes to finding failed infeasible test cases. Besides this the grammar induction will help in acquiring information about the GUI constraints. Which in terms will help us get rid of redundant test cases. The grammar that has been used in the described paper is used on a small scale. A large number of data with an optimized grammar induction will be really efficient in finding out and automating ways to get rid of such redundancies. Faster algorithms with more complex grammar is not a rare idea in the recent field of work. Even though the grammar used here is fairly simple, the effectiveness can not be overlooked. The pairwise algorithm was the most efficient in case of the effectiveness of the three extracted algorithms. Since this algorithm allowed us to train test cases of one length and could let us execute the algorithm in another length made it really feasible for us to use. The overall results denoted that the opted result is possible. It made a clear statement that test case feasibility is not a distant dream.[1]

The goal of the paper, "Mobile GUI Testing Fragility: A Study on Open-Source Android Applications", was to guess the adoption of GUI frameworks. It figures out the amount of changes that need to be made to keep the test classes the latest. 21 matrices were taken into consideration to calculate the adoption of testing tools. The ever changing nature of test cases were taken into consideration and hence the fragility was assumed. The fact that GUI tests are overly fragile creates hindrance in the automation of the GUI tests. It is overly sensitive and hence the automation can be difficult. The good part of Android operating system is it is not restricted to one source of Outsourcing. It can be available to a lot of marketplaces, can be available to different platforms and can be perceived by consumers at any given time. The marketplace for android software is huge but it has its downs as well. Since the availability is so extended the competition is also huge. Scope of errors are thin since lack of opportunities is huge. Since many applications are available for free any competitor can take advantage of it and can take inspiration to build a better error free version of the applications that are released. That's why the key is to keep the consumers hooked to our application. It can be done through mitigating errors, bugs and giving a friendly UI that will keep the consumers hooked. There is no room for softwares to casually crash. This will drive the consumers away and take them to a competitor providing similar experience but with less hindrance. The softwares has to work as the developers had told it will work. Such errors or failure will not only drive consumers away but also will get us feedback that will not be so positive. Applications will not just provide the required functionality. It will also provide some additional requirements that are strictly nonfunctional. This will give the users a better experience and will allow them to feel like they are having a personal experience. So, providing such an application is not easy and has to go through

a series of test scenarios. Without going through extensive testing an application is bound to fail. Failing or crashing while on the market is not an experience any developer would want to provide his consumers. So to solve the problem the test cases need to be adaptive. They need to be trained to learn and adapt with the upcoming changes and they need to be able to evolve with the test cases for it to work properly. Adaptive maintenance is the key to avoid the problems that coexist with GUI testing fragility. So to see if a test case inhibits fragility or not the thing that is to be done is to use an automation tool. JUnit shows a single test case as a single test method. These single test methods are then made into a series of test methods and then added to the java file of test cases. A test case will be called fragile as soon as it will go through any modifications or changes. This test class will also be called fragile if the methods inside the classes go through modifications. A test class on the other hand will be considered non fragile if the test methods do not go through modifications. The main goal of this paper or the things they have tried to achieve is they took a snapshot of the GUI testing frameworks. A set of six tools has been found out which can help us in automation of test cases. The tools are UIAutomator, Selendroid, Espresso, Robotium, Appium and Robolectric. Besides other findings they were able to find out that GUI testing framework is really thin on adoption. The common testings are used to enable the GUI fragility testing but they do not provide us with a lot of insight. 8% improvement on the fragility if the GUI testing has been cited upon using the researched frameworks.[10]

## 2.2 YOLOv5

YOLOv5 is a computer vision model. YOLO is translated to You Only Look Once. YOLOv5 is the 5th version of the model You Only Look Once. The main objective of this model is object detection. So YOLO is responsible for object detection being a computer vision model.

YOLOv5 has been found in 4 main versions. Small, medium, large and extra large are the versions which come with version 5. These 4 versions have been known to offer more accuracy as we go higher. All these versions take a different approach and different amounts of time to be trained.

YOLO network has 3 main pieces. These three main pieces are called backbone, neck and head.

The backbone works with the help of CNN. The backbone features different pixelated images and these are clustered to train the system. The neck mixes up the images and combines the images. Then the images are combined and passed forward and then the object is detected. While the head works as head and predicts the steps like a head would work.

The training process is done in 2 steps in this system :

Data augmentation is done to transform the base data. The model is then exposed to a wide range of variations. The training is done separately.

The second step is loss calculation. Loss calculation is also really self explanatory. YOLO calculates the total loss from different functions. The goal is to have maximum precision. These two steps help train the system and are really efficient in that way. Since it only has two steps it is really easy to train while comparing it to the other existing frameworks in the market.

The reason why YOLO is really widely used is because of few efficient functionalities. It is really easy to install and can be installed without any hassles. Since Torch is a lightweight library it is really easy to install. The training of YOLO is really fast and can be done really easily. YOLO is extremely easy to use on mobile devices that's why it is widely used by developers.

That is why it has been used really widely all over the world and will be used as an advanced framework in future as well.[31]

## 2.3 Pytorch Framework

Pytorch is an integration of Python and Torch. Python being the programming language used in this case and Torch being the library used in this case has been turned into a framework. It is an open source machine learning framework. Open source considering torch being an open source library has given us immense privilege to work without hesitation. Torch is used to produce deep neural networks. The language in which torch has been developed is the Lua scripting language. This has been working as a deep learning research platform for the developers for a long time. The main objective of this framework is to build a bridge between prototype and deployment. It quickens the process time of the research.

So the programming language in use in this framework is Python. Python being a very dynamic programming language allows this framework to work in diverse functionality. It is also possible to produce computation graphs using this framework.

The main benefit of this framework is that the developers don't have to wait around to finish the entire code to see what works for them and what doesn't work for them. It gives the opportunity to test out sections of codes as soon as it is written. The whole code need not be written to test if the new portion added works or not. It gives more opportunity to the developers to have less bugs as small portions of code can be tested along the way.

This is not the only benefit that the PyTorch framework can give us. The PyTorch framework has many more diverse features that benefit us. It is really easy to use the PyTorch framework since it is based on Python which is widely known as an easy to learn and easy to use language. It is really easy to debug because there exists python tools in the marketplace. Open source library is also available which enables easy access. Open neural network is also available. In short this framework is really user friendly and enables us to work efficiently without much hassle.[29]

## 2.4 Computer Vision

Another field of artificial intelligence is computer vision. Computer Vision is a technology that allows computers to deduce a meaningful interpretation of a still or moving image. It takes any form of image as input and tries to comprehend the meaning from the image based on the previous training it went through. The idea of computer vision is watch, observe and comprehend.

The idea of computer vision is mimicking human vision. The concept of computer vision came in generation trying to work out how humans brain and eye functions together. How human eyes aren't limited to seeing only. It infers and comprehends and speculates. The eyes are well trained from the time a child is born. It sees

objects and corresponds meaning with what it is seeing. Computer vision is also trying to do that. Whether an object is in motion or is stationary or if there are errors in the image is what computer vision is trying to work on.

Machine training is how developers are trying to achieve this functionality. But its capability will not be limited to the time to perceive images by camera or human eyes. It will have to be faster than that of optic nerves because it will have to process mass data in a seemingly insignificant amount of time and has to process the errors in that frame. It is estimated to go ahead of human capabilities.

So the question is how computer vision works. Computer vision works on a simple idea that it will be fed data and will be trained to recognize the meaning of the data fed. Mass data will be given as input so that whenever a similar object comes into vision the technology will be able to recognize what it means.

Taking an example of a car. It will be fed images of a car from all possible angles so that whenever a computer sees a car it will be able to tell that this is a car. Video information will also be fed to the system that will be able to say if the object is moving or stationary or accelerating.

The technologies that have helped developers achieve this computer vision to some extent is convolutional neural network (CNN) and the machine learning technology called deep learning. So a deep learning algorithm works on the principle that if the system is fed enough data it will be able to go through the data and will be able to go through all data to correspond a meaning with the data. The algorithm is a self learning algorithm. It teaches itself about the programme about recognizing images.

While CNN breaks the whole image into readable pixels and gets an idea of what the image is trying to be perceived as. It uses the labels to deduce mathematical functions that help it figure out the image from the deduction. Neural networks run CNN to see if it is correct or not.

Similar to the human eye, computer vision works to figure out the exact image, distance, position of the object. This has been a really great help in the wide horizon in the AI world.[36]

## 2.5 Object Detection

In the recent era and the over evolving software era deep learning has been a crucial part of the Artificial intelligence area. AI is deeply rooted in our life now. Our day to day life is entangled with AI. And among the areas of AI object detection is a prominent sector. It concerns the area of deep learning and computer vision.

So object detection is as self explanatory as it gets. It's a technology that detects objects. Through deep learning this technology is able to detect any image. This will go through extensive training to be able to detect the images. These images can vary from inanimate objects like houses, cars to living objects like human beings, cats, dogs. Object detection can figure out what the object is in the bounding box. In object detection the object is detected considering its basic form and is classified accordingly. To identify these objects the Softmax function is taken into consideration.

Object detection can be both machine learning and deep learning based. In the machine learning based object detection with help of Image based feature extraction

technique the images are extracted manually. Images of different objects are fed to the AI and trained to detect the object.

Whereas on the other hand deep learning based object detection uses the pre existing algorithms to extract the features automatically and train itself to learn and detect images. Both ways come with its own sets of perks and cons.

Image classification is also an important part of object detection. In this case an image is considered as an input. The image is fed to CNN. And the image will have a correspondent class with images. The images will be categorized and classified accordingly.

A notable problem is that an object is not only an object, in an image there can be multiple objects so classifying them into a single class will not be possible. This creates hindrance in the object detection technology. Two types of problems may arise during object detection. Multiple images may exist in a single frame and the second problem is the prediction of coordinate values of the bounding box object.

Object localization comes in handy in this case. In object localization the object that is the main object in the image is localized and is detected. Where object detection fails to come up with detection of a single object the object localization saves the day. On the other hand, image classification works really differently. Where the image localization and detection detects the object image classification shows us the possibility of an object existing in an image.

Object detection is a really important feature in the field of Artificial intelligence. It gives us a wide horizon to work with.[19]



# Chapter 3

## Model & Dataset

### 3.1 Dataset description

We have used a dataset that was created by RICO, a data driven design group from the University of Illinois for their research. The RICO dataset contains 66,261 unique android app UI screenshots that was taken from 9,772 android applications. Some categories were excluded such as media players and photo editors. These 9,772 android applications have an average rating of 4.1 stars on google play store. In order to create this dataset, RICO downloaded 9,722 applications from google play store and the screenshots were taken by 13 workers that were recruited from UpWork in order to take screenshots. These workers were instructed to work on individual applications for no more than 10 minutes and take screenshots of each screen that each application offered.[7]

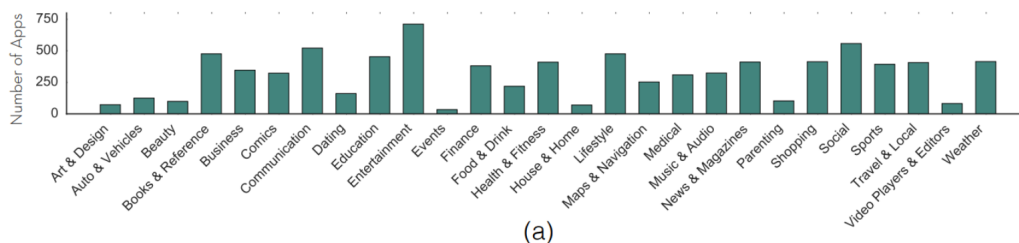


Figure 3.1: Summary of RICO dataset

During our training we found that, with a dataset above 2000 images, we were getting “data-loader exit unexpectedly” error. Since our hardware was not powerful enough to train this vast dataset offered by RICO, we had to reduce the number of data taken into training and validating the model. Because of this hardware restriction we reduced the dataset to 1,769 screenshots from the RICO dataset which contained android core components and some composite components. And also found that, with this hardware available to us, a dataset between 1,700-1,800 images created optimal and steady results while training and validating the model. As a result we worked with 1,769 images for our thesis purpose.

#### 3.1.1 Data preprocessing

In order to use our dataset to train, at first we had to label each of the core and composite components using LabelImg package which is available on GitHub. LabelImg

is a popular image annotation tool which was created by Tzutalin with additional help of many other contributors. Even though this Labellmg package is no longer actively being developed, this flexible image labeling tool has become a part of Label Studio Community. (heartexlabs, n.d.-c)

We ran this Labellmg on our system and created a box on each of the components that we wanted for our research. After putting a label on the components, a text file was saved on the system which contained the serial number of the component including the name that we provided for the component and the four x (max, min) and y (max, min) coordinate values that represented that the box that was put on the component for our model to train. An example is given below,

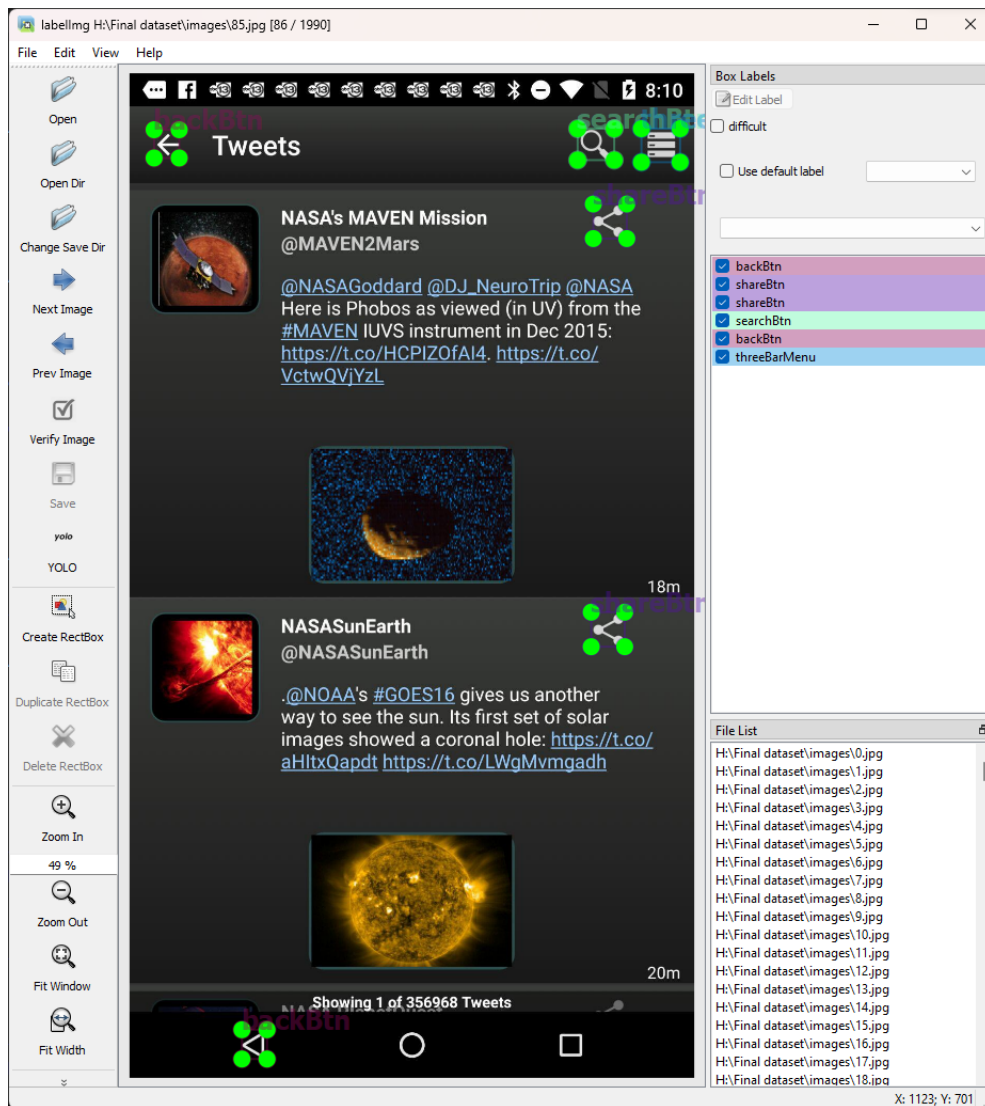


Figure 3.2: Labeling images using Labellmg package

After putting the labels on the above image we get a text file. An example of the text file is given below,

After going through all 1769 images, we have labeled all core and composite components that were found on the screenshots in order to train and validate our model. The visual representation of this data after image labeling is given below,[35]

```
85.txt - Notepad
File Edit View
1 0.065741 0.070312 0.044444 0.028125
4 0.847222 0.553646 0.059259 0.034375
4 0.850926 0.147396 0.059259 0.034375
11 0.825926 0.070833 0.064815 0.031250
1 0.221296 0.966667 0.044444 0.029167
9 0.940741 0.071875 0.068519 0.033333
Ln 6, Col 38 100% Unix (LF) UTF-8
```

Figure 3.3: Data after Image Labeling



Figure 3.4: Visual Representation how labeling works

### 3.1.2 Feature selection

As for the core and some composite components that we have taken into consideration were number picker, back button, settings button, add button, share button, info button, three dot menu button, switch button, cross button, three bar menu button, reload button, search button and check button. After labeling each of the images for our training and validating our model, we get the following numbers of core and composite components shown in the table below,

Name of the Component	Numbers of component detected
Number Picker	43
Back Button	2101
Settings Button	153
Add Button	172
Share Button	113
Info Button	162
Three Dot Menu	420
Switch Button	87
Cross Button	222
Three Bar Menu	435
Reload Button	65
Search Button	253
Check Box	545

Table 3.1: Features

## 3.2 Model description

Yolov5 is a real time single stage object detection model built on single convolutional neural network(CNN) architecture to detect object and classification cases in single forward pass network. Which involves input taking an image and in the output case giving the bounding boxes of the component and the class probabilities of all the components detected in the image individually. Which makes the model more efficient and because it is using cross scale feature pyramids, strong back bone network and the use of anchor boxes the accuracy is very good. [26]

### 3.2.1 Single Stage Object Detector

For both object detection purposes yolov5 uses a single CNN Network on a single forward pass and comprises Backbone, Neck and Head. Backbone implies a pretrained network which reduces the spatial resolution while increasing the feature resolution and takes the important characteristics representation from an image. To generalize properly on different scale and size of a component and getting the feature pyramids model neck is used. Finally, the most important part which increases the accuracy by implementing the bounding boxes with confidence and index of the component and where the final stage operation is being performed is the model head.

In case of Yolov5 head and neck is using PANet (Path Aggregation Network) and SPP (Spatial Pyramid Pooling) and the backbone consist of the CSP( Cross Stage

Partial ) method implemented in the Darknet53 Convolutional network called CSP-Darknet53.[26]

### **3.2.2 Other important part for improved result**

YOLOv5 uses Sigmoid Linear Unit (Silu) and sigmoid function as activation function. Silu is implemented for convolution operation in the hidden layer while sigmoid function is used in the output layer. For Loss function BCE(Binary Cross Entropy) and CIoU( Complete intersection over union) which is used respectively for classes,object and location loss. There is also a focus layer which is used to improve speed by reducing parameters and replacing the first three layers which makes some changes in mAP(mean Average Precision). The grid sensitivity is eliminated which means even if the component is at the edge the model can detect it which was hard for the previous version of yolo.[26]

# Chapter 4

## Implementation and Result Analysis

### 4.1 Implementation

In order to make sure that everything runs properly and smoothly there are some pre-requisite like a proper environment setup with all the packages installed that will ensure the system will work without any error and give meaningful results.

#### 4.1.1 Hardware Specification

Hardware specification basically means the hardware we used to train our data and carry out the experiment mentioned above is given below, The GPU was used mostly for data training which was Zotac Nvidia RTX 2060 with 6.00 GB of VRAM. There was 32GB of DDR4 2400MHz RAM available. Our system was powered by Intel Core-i7 8700K which is a 6 core, 12 thread processor. Windows was used to implement the feedback and functional testing system and it was installed in a 512GB M.2 SSD.

#### 4.1.2 Environment Setup

In order to implement the proposed system the first thing that needs to be done is make sure the environment is properly configured. For our case we have used python 3.9 in the conda environment to easily be able to install packages while isolating the whole environment.

#### 4.1.3 Package Installation

In order to run the system, some packages need to be installed for YOLOv5 to train the custom models and for setting up the recommendation and functional system. These additional packages ensure that the model we used YOLOv5 and our auto recommendation and function testing works seamlessly. List of the packages and how to install is given below,

## Package for YOLOv5

First we will need to clone the YOLOv5 model from github (<https://github.com/ultralytics/yolov5.git> ) then in the YOLOv5 install all the packages listed in the requirement.txt file.

## Package for Recommendation Testing System

For our proposed system which gives auto recommendation and functional testing, we need packages such as pygetwindow, numpy, pandas, math, matplotlib, csv, opencv-python, ipywidgets, time, pyscreenshot, networkx.

Among these packages the most important packages are Opencv-python which is used to show a real time window. Additionally, Matplotlib is generally used for visualization. In order to control and manage the inputs, the ipywidgets package is installed. We also needed to manipulate and read csv files, for that purpose Pandas package is needed and installed. In order to show real time changes and detection from the window Pyscreenshot package is needed for taking screenshots from the active application window. As we are showing real time detection and recommendation we need to choose the application the UI developer is using to develop the UI and for that Getwindow is installed. And last but not the least, Networkx is installed to generate graphs.

### 4.1.4 Custom Model Configuration

Though we are using the YOLOv5 model it was trained on different datasets and different classes of components so we need to ensure that we can use this model by doing necessary modifications for our use case.

#### YAML File Configuration

The YAML file ensures the whereabouts of training and validation images and their corresponding labels and number of component classes and their name which is supposed to be detected by the model. We have created a custom YAML file with the location of our custom datasets training and validation folder with the 13 components of classes we have chosen to detect and give assistance with.

#### Training Custom Model

Though YOLOv5 has few number of pre-trained models like YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x etc they were trained using dataset where the component classes we are using are missing so we need to create a custom model according to our need by training the model according to our dataset.

#### Training on Custom Dataset Configuration

We have trained our custom model with 300 epoch with a batch size of 32 and selected a pre-trained model of YOLOv5. In order to access the location of the images, the custom yaml file has to be configured and to get the result faster we have cached the training image data on the RAM and used the GPU.

### 4.1.5 Automation System

For an automation system we need to ensure that the custom trained model is properly loaded, while grabbing the window screenshot. We need to make sure that the unnecessary parts of the images are cropped as much as possible. The process of the application which we are supposed to screenshot and show real time feedback is correct, then the least confidence value which helps to make sure the component detection is highest while false detection is lowest as possible, is tuned according to which value gives the best result. Then the location file needs to be properly loaded to make sure the system finds the file so that it can show the recommended components location. Lastly, to re-run the application we need to ensure that the real time window is closed beforehand.

## 4.2 Result Analysis

While researching for object detection models, we have come across some object detection models that we could have used for the purpose of our research. Our entire research is focused on how we could use object detection algorithms to do automation in the software development phase and save time during development. There are various object detection algorithms such as faster R-CNN[3], mask R-CNN [8], YOLOv3, YOLOv5, YOLOv4, YOLOv7, YOLOv8[5][25], SSD[4], SSD MobileNet [23] and RefineNet[11] which falls under the category of 1-stage and 2-stage detector respectively. Due to the fact that, 1-stage detector has a method that can solve classification and localization problems simultaneously than the 2-stage detector, it was a better suited algorithm for our research. Despite the fact that Fast R-CNN was popular and commonly used in the past, it has some inefficiency problems in both learning and execution speed. In terms of YOLOv7, it is a bit complicated to set up and takes lots of time which beats the main purpose of our research. YOLOv8 was just recently released but it is still under extensive development to improve the features but it has the potential to become the best object detection algorithms among the YOLO models. In case of processing speed and accuracy, YOLOv5 is quite faster than Fast R-CNN[6]. Even though the basic working principle of YOLOv5 and YOLOv4 is similar[15], YOLOv5 is an improvement of base YOLOv4 and in terms of performance in precision, recall and average precision, YOLOv5 has proven to be more effective and efficient compared to YOLOv3, YOLOv4 and Faster R-CNN.[28][22] In terms of our thesis prospect, the best thing about YOLOv5 is that it is very easy to set up in order to do object detection. Thus we have selected YOLOv5 for our research purpose.

### 4.2.1 Comparative Analysis

YOLOv5 has 5 different versions n, s, m, l, x respectively. Extra small (nano), small size, medium size, large size and extra large size model and among these variants of YOLOv5 there is not much working principle differences except for the number of parameter and for more the additional parameter is, more cuda memory is required to train and slower it is to run.

For our use case, we have trained n(nano), s(small), and m(medium) size variants of YOLOv5. In the case of l(large) and x(extra large) it requires too much resources



like time and gpu memory to train properly and as they will run much slower than their counterparts which defeats the very purpose of our thesis which is to minimize time usage. Thus, we did not use them.

In the hopes of getting better performance, we created variations of our dataset and trained the n, s, l variants of YOLOv5. Variations of the dataset is,

- datasetvariationTVNB - 1468 Training image, 300 validation image with no background image
- datasetvariationTVSB - 1415 Training image, 353 validation image with 102 background image

Components	datasetvariationTVNB		datasetvariationTVSB	
	Train components	Val Components	Train components	Val Components
NumberPicker	39	4	36	7
Back	1763	338	1697	404
Settings	125	28	121	32
add	150	22	150	22
share	99	14	90	23
info	143	19	138	24
3dotmenu	363	57	350	70
switch	74	13	70	17
cross	205	17	199	23
3barmenu	362	73	350	85
reload	55	10	52	13
search	207	46	199	54
check	502	43	465	80
background	0	0	102	49

Table 4.1: Component division on Train and Validaion

We trained s, m , n models with the parameter image size of 640, batch size 32 epoch 300 and used device 0 which refers to training by using gpu and we got following results in detection

**Note:** In nTVNB - n refers to YOLOv5 model and TVNB refers Dataset

Components	nTVNB	nTVSB	sTVNB (Selected)	sTVSB	mTVNB	mTVSB
NumberPicker	0.5	0.29	0.5	0.29	0.5	0.29
Back	0.99	0.99	1	1	0.99	1
Settings	0.75	0.78	0.75	0.81	0.75	0.81
add	0.59	0.68	0.86	0.68	0.95	0.77
share	1	0.96	1	0.96	1	0.96
info	0.84	0.92	0.84	0.92	0.95	0.83
3dotmenu	0.91	0.91	0.93	0.94	0.88	0.93
switch	0.77	0.71	0.85	0.76	0.77	0.82
cross	0.94	0.91	0.88	0.83	0.65	0.78
3barmenu	0.92	0.92	0.95	0.91	0.86	0.92
reload	0.6	0.46	0.4	0.38	0.4	0.46
search	1	1	0.83	0.94	0.85	0.93
check	0.51	0.75	0.53	0.72	0.53	0.76

Table 4.2: Confusion Matrix Results

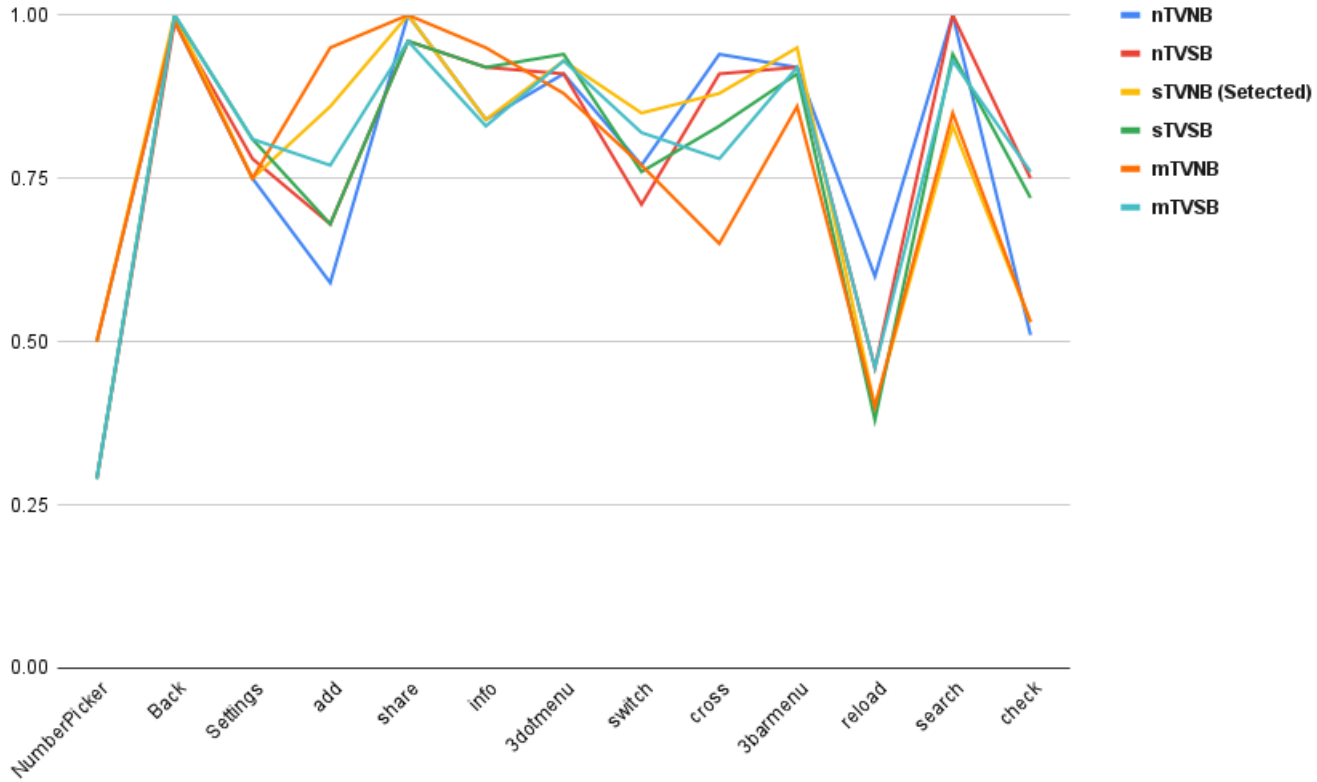


Figure 4.1: Comparison Graph of YOLOv5 Variants

## 4.2.2 Selected Model Result Analysis

After carefully observing the result from the training it can be said that the YOLOv5s model trained with datasetvariationTVNB dataset is giving us the best detection result for our dataset. So we are going forward with it and using it in our system for automation and testing purposes.

## Trained Model Analysis

For our system to work properly we need the YOLOv5 model to properly detect components. After using datasetvariationTVNB dataset on training the YOLOv5s model, we see the best detection result for most of the components we have selected for our case. Here is the confusion matrix of individual component detection for the model,

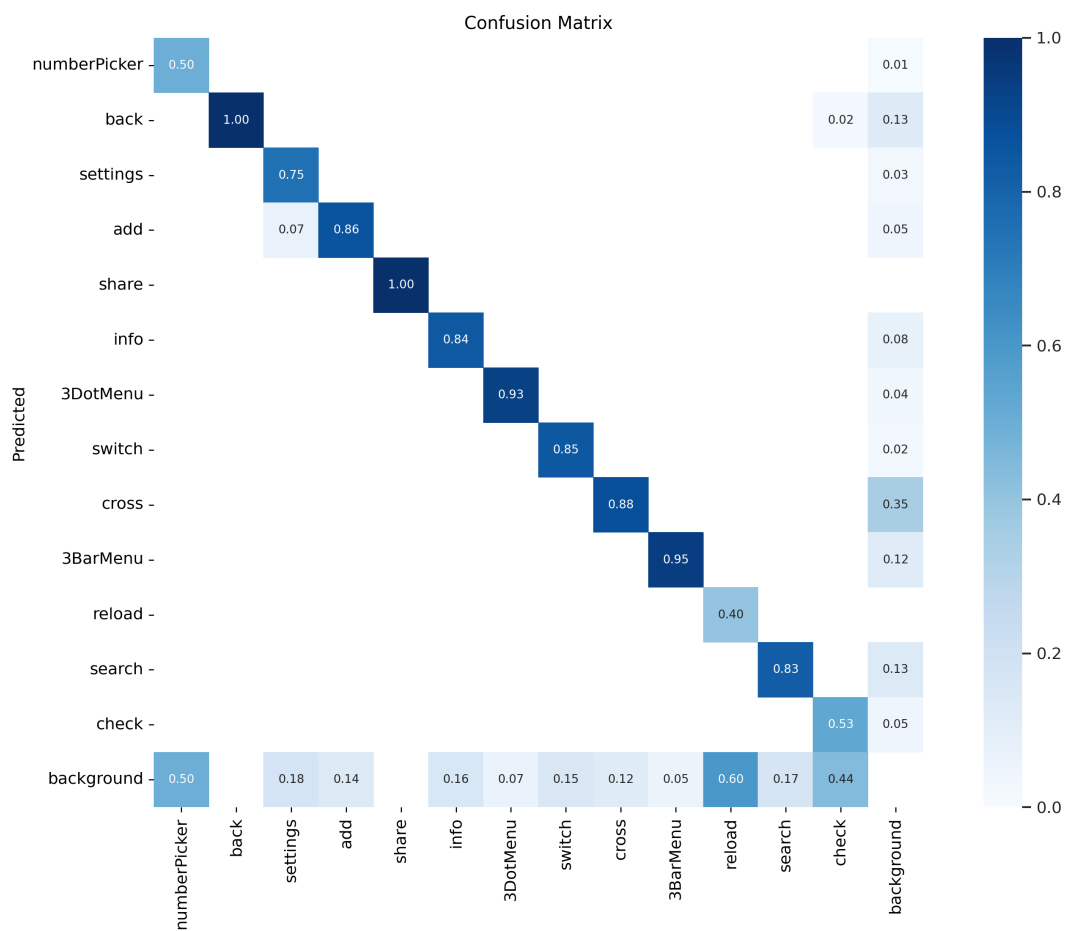


Figure 4.2: confusion matrix of our custom model

## System Overview

This is a step by step representation of our thesis from start to finish where we have illustrated the Data processing (Collecting, Labeling, Splitting), Training and core component of our thesis which is the system we have worked on to automate an important and integral part of software development and testing.

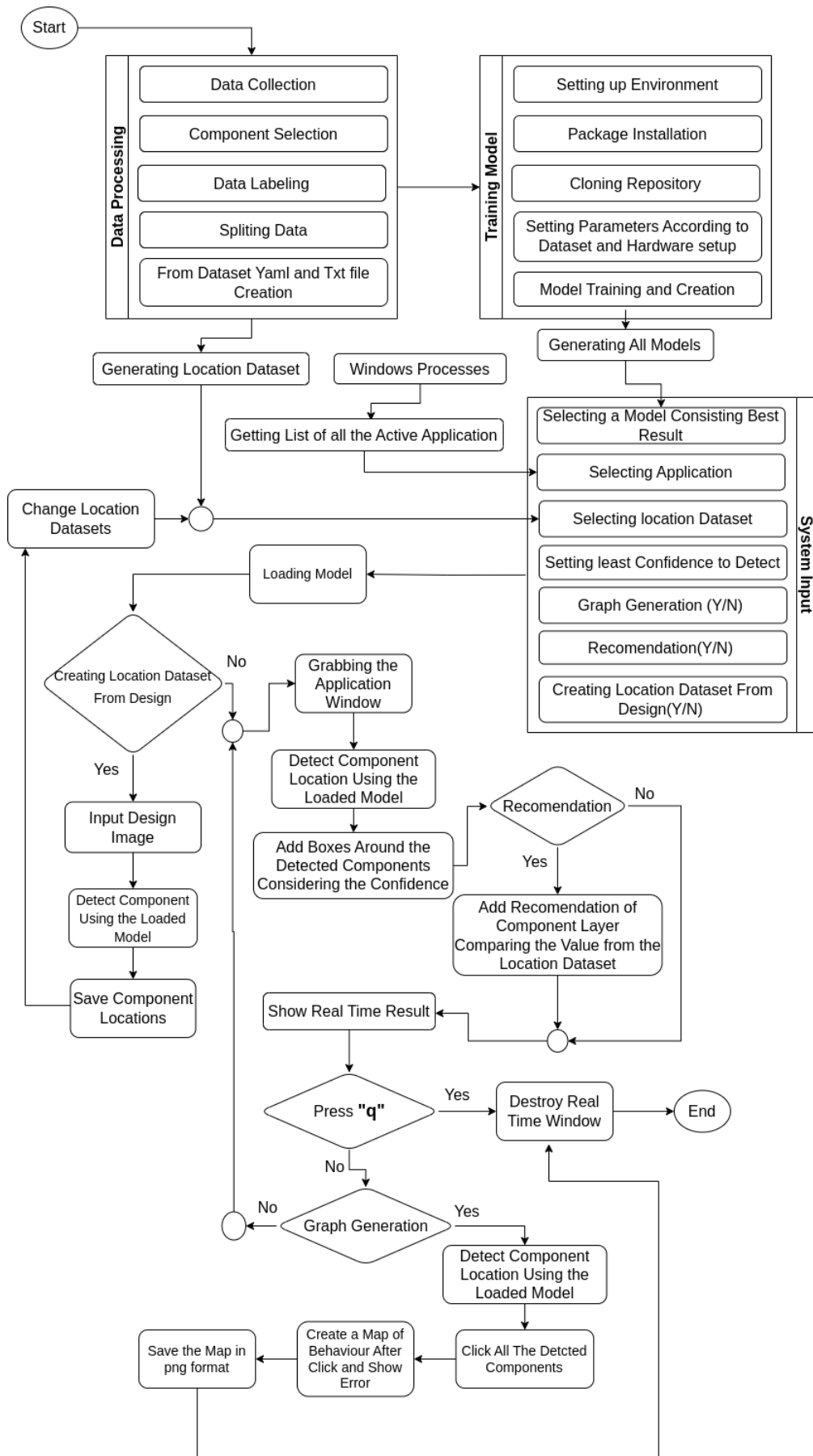


Figure 4.3: System Overview

### 4.2.3 System Analysis

After ensuring that the model gives good enough component detection we give the model as input in our system and the system will extract the names and coordinates of the components which our system will use to do real time Detection, Recommendation and functionality test with its graph.

#### Detection

Here in the detection process it takes the snapshot of the application window and and detects its components using the model and shows it through our system.

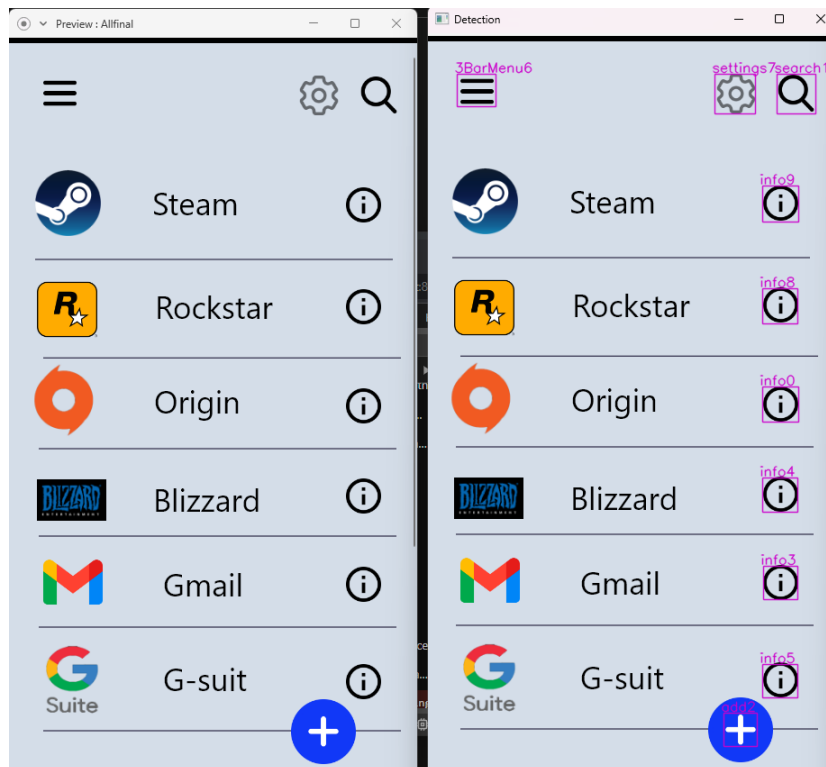


Figure 4.4: Detection(Left side - the Application, Right side - Detection)

#### Recommendation

In the first figure we can see that there is no recommendation because according to our system after detection and comparing the locations with the given recommended location it has suggested there is no need make any changes in the UI

In the second figure it can be seen that the system is suggesting to change the position of the components because it has found that after checking the recommended location provided it is better to change the UI.

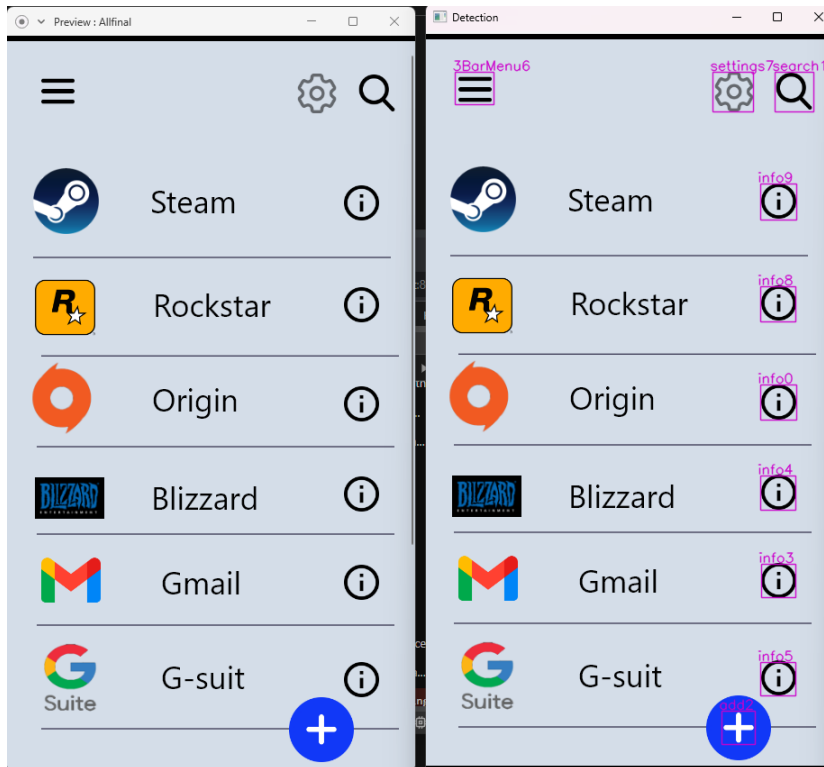


Figure 4.5: No recommendation Perfect case (Left side - the Application, Right side - Recommendation)

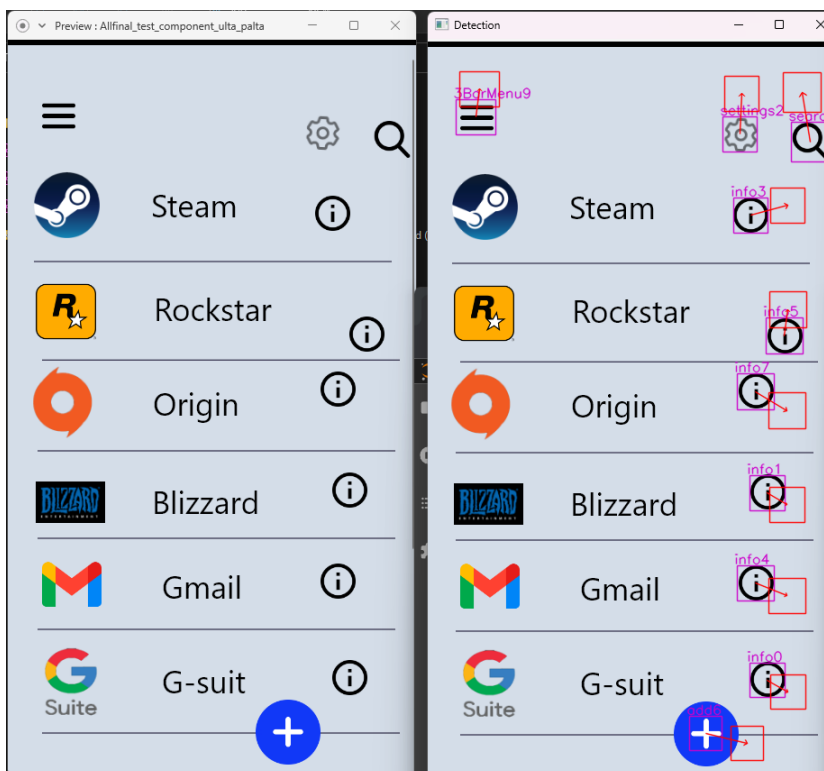


Figure 4.6: recommendation Perfect case (Left side - the Application, Right side - Recommendation)

## Functionality Testing & Graph Generation

In the first figure we can see that there is no error found because the model has detected all the components furthermore it was able to press all of them and the components are working thus it explored every page that it was programmed to. In the second figure there was an error found. It could happen in a few cases such as a model detecting something other than a component as a component and pressing it or the functionality of the component was not added and the UI developer can trace through the graph to find where the first error occurred and fix it.

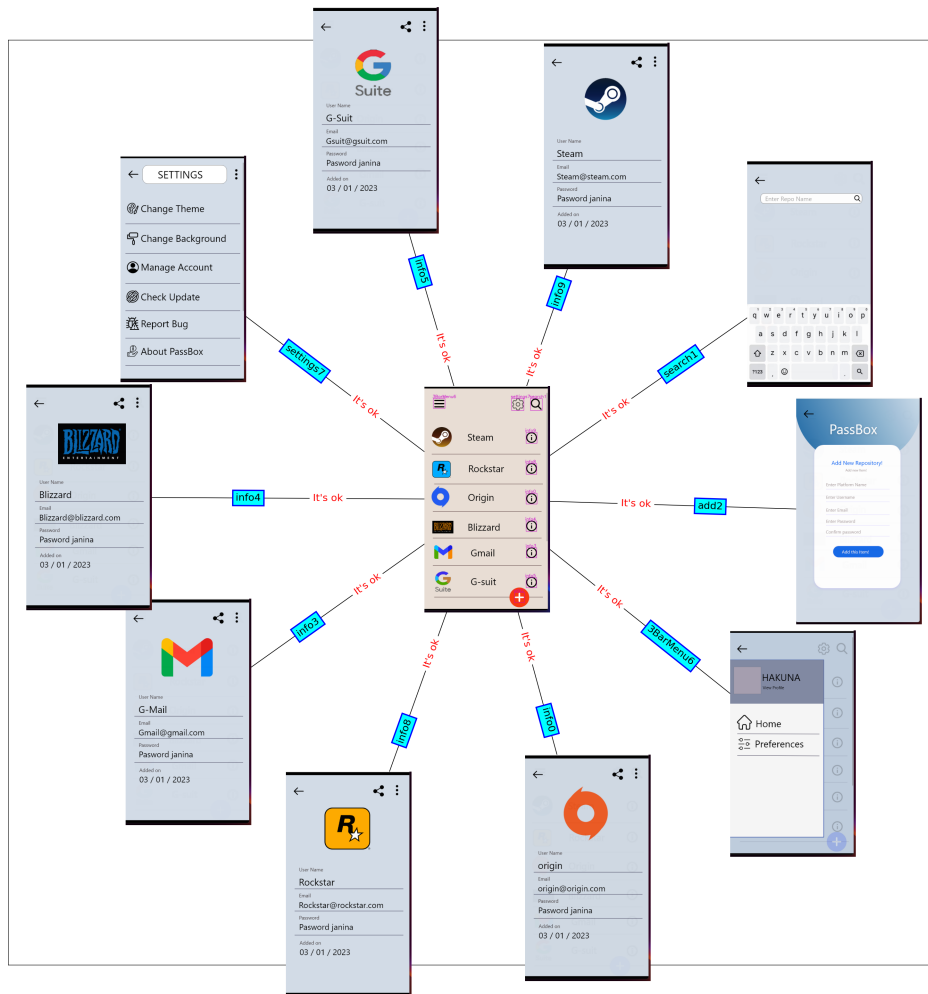


Figure 4.7: Functional Testing Error Check Result (No Error))

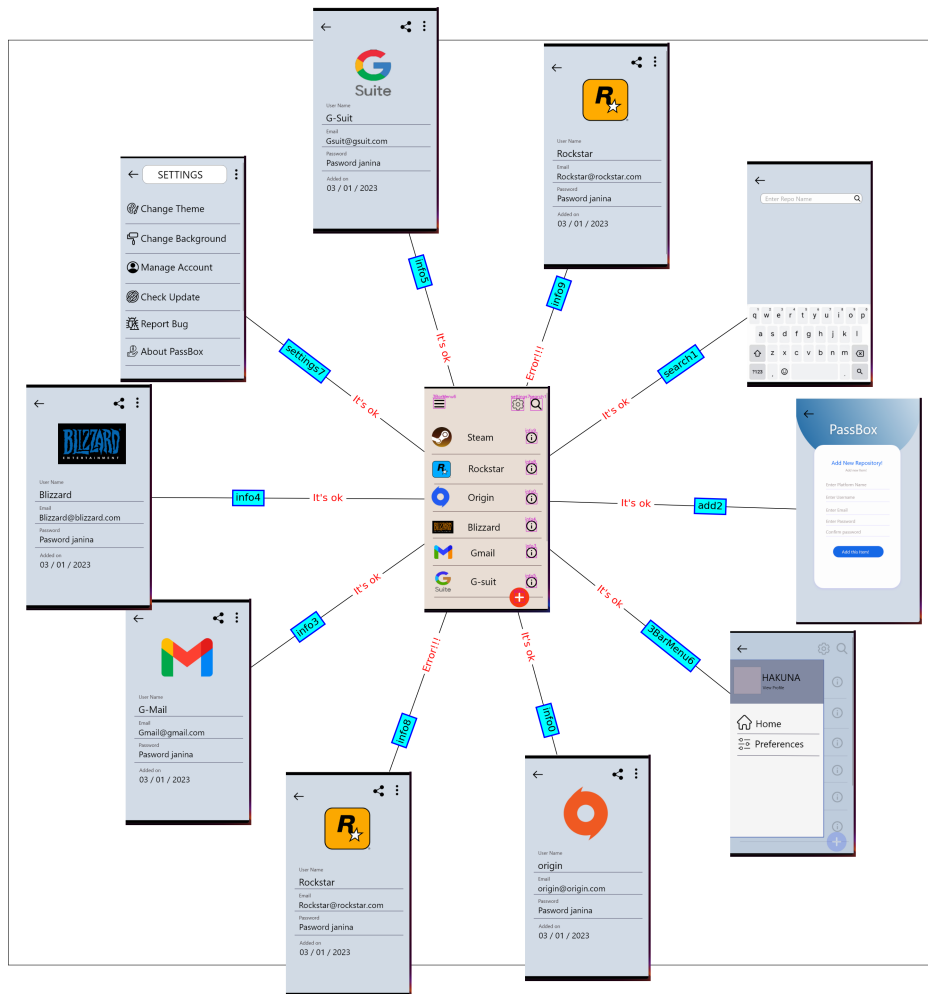


Figure 4.8: Functional Testing Error Check Result (Error Found)



## 4.2.4 Performance Analysis

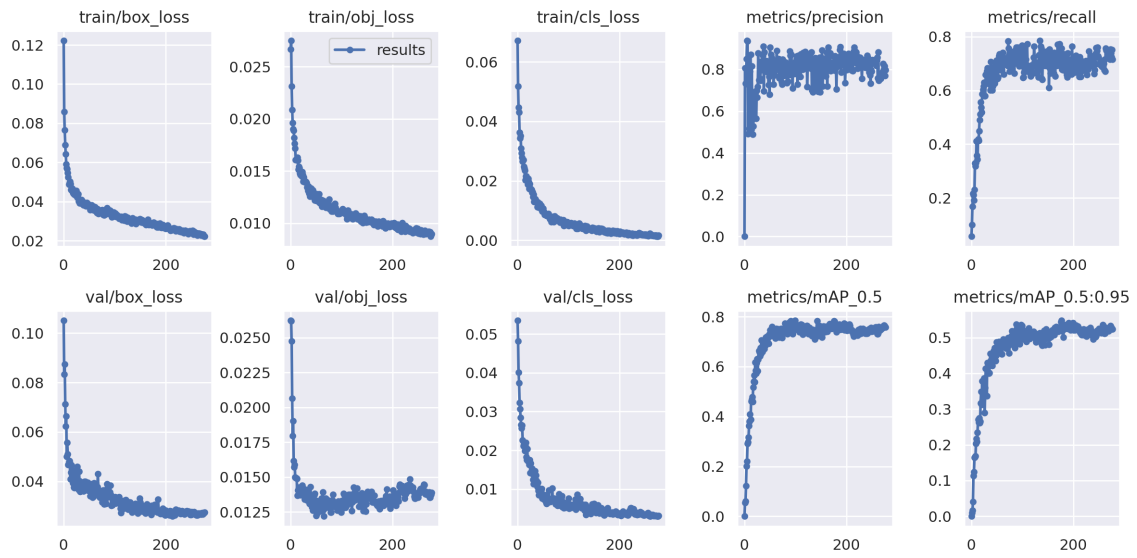


Figure 4.9: Result

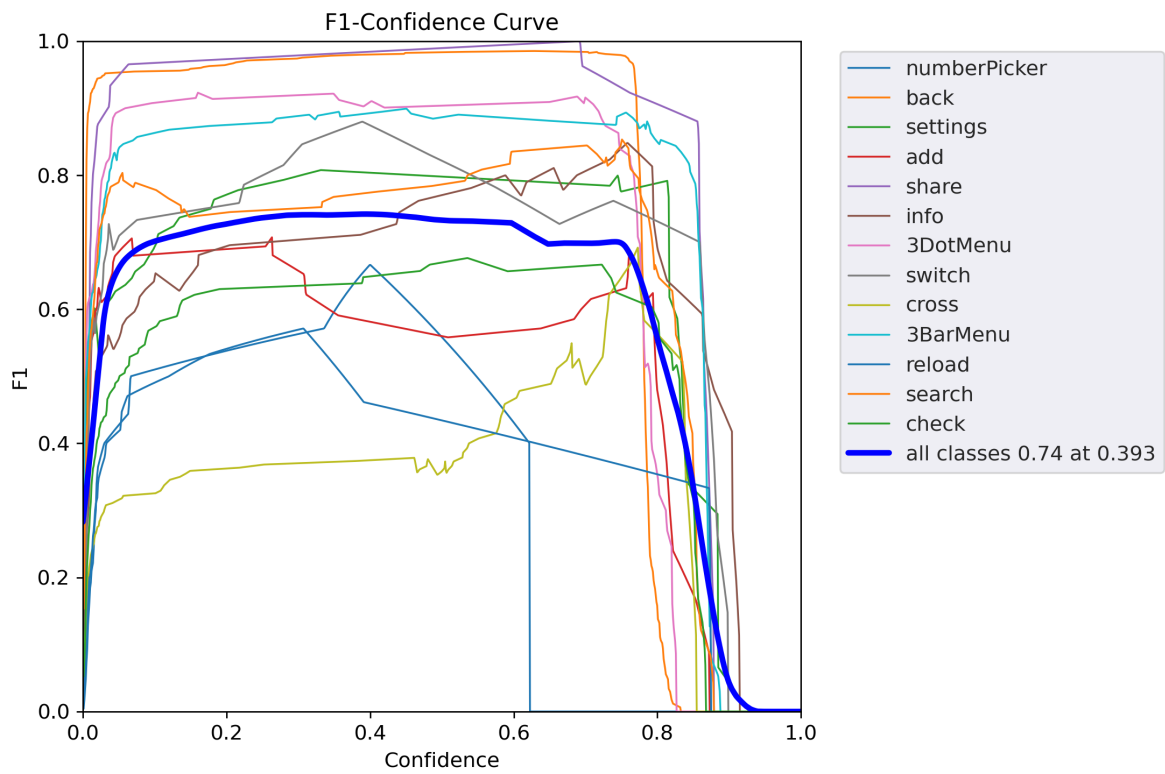


Figure 4.10: F1 - confidence curve

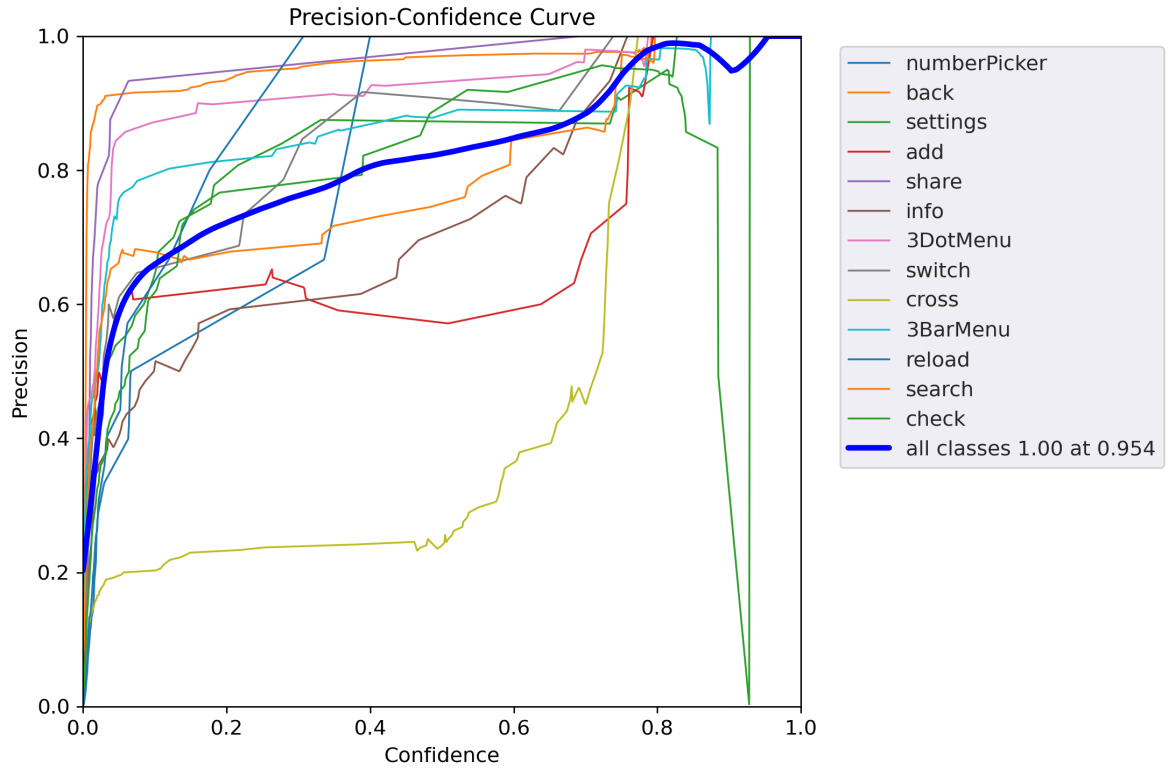


Figure 4.11: Precision Confidence Curve

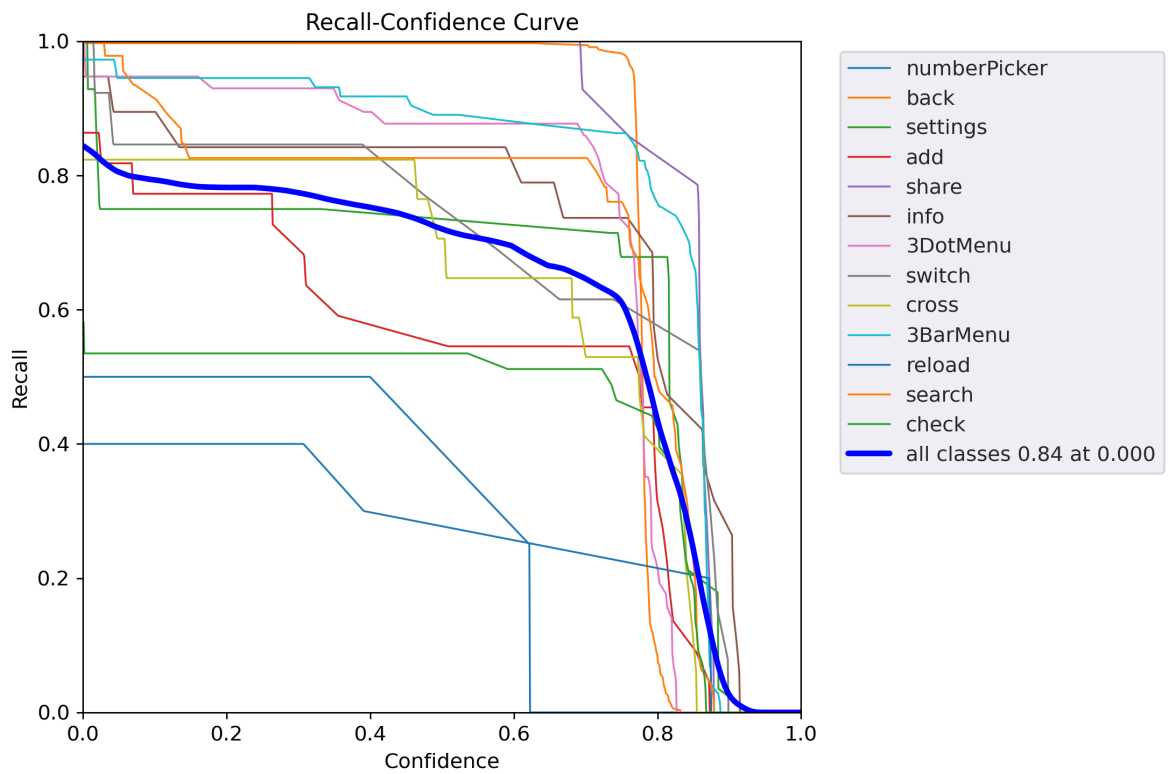


Figure 4.12: Recall Confidence curve

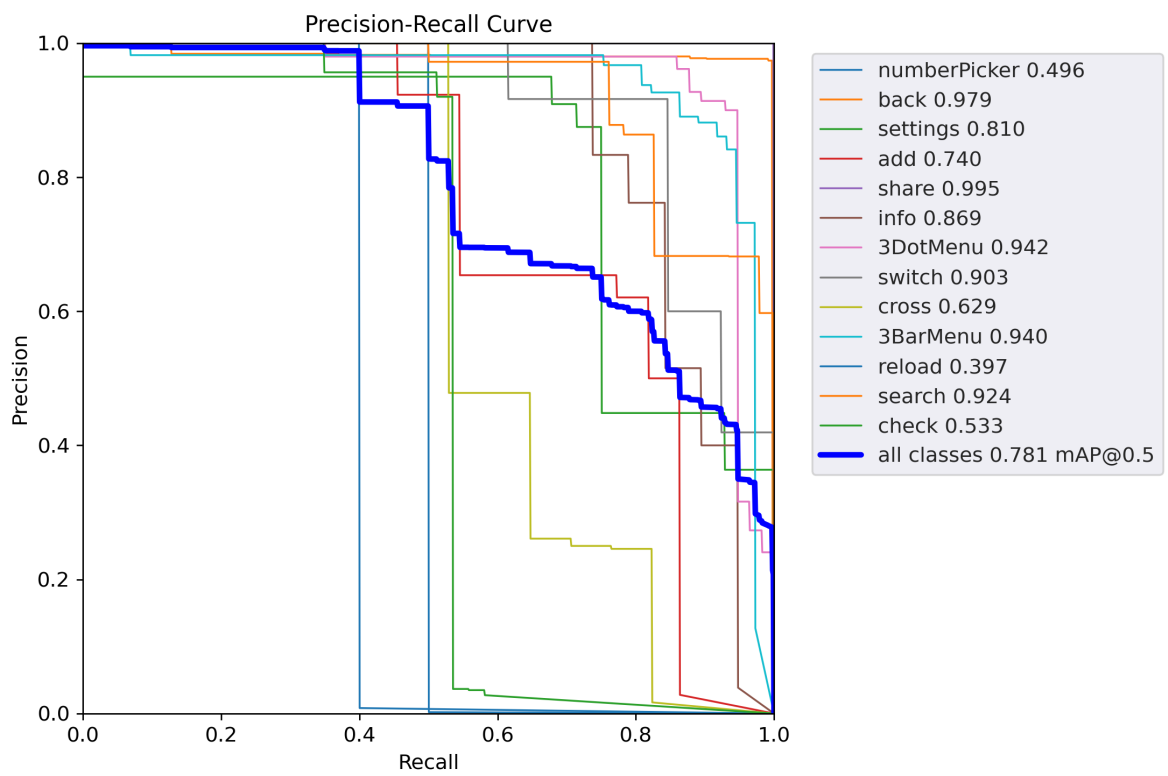


Figure 4.13: Precision Recall curve

# Chapter 5

## Conclusion

### 5.1 Challenges

The most hardest challenges that we have faced while conducting our research was dataset pre-processing. At first we were unable to find a suitable dataset that we could use for our research purposes. We needed an android application screenshots dataset. So that we could train the YOLOv5s variant of the YOLOv5 object detection model that we have selected. Getting the dataset was not that much hard. Although it took quite some time before we came across the RICO dataset, which was suitable for our research purpose. The hardest part during our research was to label the components from the dataset. The dataset was created and used by RICO in their research where they try to understand the best practices for android application design.[7] As the dataset were not labeled and we have focused on some of the core and composite components, we had to label each images by hand using the LabelImg package that was available in github which was so much challenging because we had to go through images one by one and label them with core and composite components. Another minor challenge that we faced was during training the model. As we did not have that much resourceful hardware, we faced problems while setting the batch size for training. It is recommended that, for better training results, use the largest batch size. Smaller batch size creates poor batchnorm statistics and which was recommended to avoid.[22][34] As a result, we have used batch size 32 that was within the acceptable range of our hardware in order to train the model.

### 5.2 Future Prospect

YOLOv5 recently has been used in many researches that deal with object detection and has gained popularity since its release. For our research purpose we have only considered android application UI development automation. But a rich and diversified dataset of any application can be used to train the model and use our system to work on any kind of GUI and do automated testing during the development phase. For any web application or website, our system can view recommendations and do functionality testing on those application's respective components. One of them can be a test input box for a website or web application. If enough data can be feed into the model to train it accordingly, with enough twerks in the coding part, our system can do functionality testing on the text input box for UX. Moreover, in future, this

system can be developed to do further extensive functionality testing. Additionally, after further growth this system can be turned into a modular system. Meaning that additional features can be developed in future and can be added as modules as development continues on our system. As one actively working part of our research is giving object recommendations, our system can be used to give recommendations on architectural designs after training the model with relevant dataset.

### **5.3 Conclusion**

Software development has multiple phases, in which UI development and testing takes a significant amount of time. The back and forth between UI developer and the QA team can be reduced to a remarkable amount by making the UI testing automated and also making it happen in real time. We aimed to achieve both of these goals in this paper. We fed YOLOv5 screenshots of Android app UI to train our model. By following our methodology UI developers will be able to write and test code simultaneously for the application's User Interface components placement through which users interact with the application. Our system also enables UI developers to evaluate if the components are working properly that they have written so far without the need of checking everything later by manual testers. So, as we have emphasized already our system is obtaining the results of UI bugs, which traditionally get detected in a later testing phase. Thus, our system can assist to deliver the same quality Android application with a decreased amount of time. As a result this methodology will be beneficial for both industrial level Android application developers as well as small companies who will be able to save time for UI testing and allocate those resources where needed to develop applications more efficiently.

# Bibliography

- [1] R. Gove and J. Faytong, “Identifying infeasible gui test cases using support vector machines and induced grammars,” in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, IEEE, 2011, pp. 202–211.
- [2] C.-Y. Hsieh, C.-H. Tsai, and Y. C. Cheng, “Test-duo: A framework for generating and executing automated acceptance tests from use cases,” in *2013 8th International Workshop on Automation of Software Test (AST)*, IEEE, 2013, pp. 89–92.
- [3] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *Advances in neural information processing systems*, vol. 28, 2015.
- [4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in *European conference on computer vision*, Springer, 2016, pp. 21–37.
- [5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.
- [6] L. Zhang, L. Lin, X. Liang, and K. He, “Is faster r-cnn doing well for pedestrian detection?” In *European conference on computer vision*, Springer, 2016, pp. 443–457.
- [7] B. Deka, Z. Huang, C. Franzen, J. Hibsichman, D. Afergan, Y. Li, J. Nichols, and R. 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*, 2017, pp. 845–854.
- [8] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.
- [9] D. Adamo, M. K. Khan, S. Koppula, and R. Bryce, “Reinforcement learning for android gui testing,” in *Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, 2018, pp. 2–8.
- [10] R. Coppola, M. Morisio, and M. Torchiano, “Mobile gui testing fragility: A study on open-source android applications,” *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 67–90, 2018.
- [11] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, “Single-shot refinement neural network for object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4203–4212.

- [12] R. Coppola, L. Ardito, and M. Torchiano, “Fragility of layout-based and visual gui test scripts: An assessment study on a hybrid mobile application,” in *Proceedings of the 10th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, 2019, pp. 28–34.
- [13] R. Coppola, M. Morisio, M. Torchiano, and L. Ardito, “Scripted gui testing of android open-source apps: Evolution of test code and fragility causes,” *Empirical Software Engineering*, vol. 24, no. 5, pp. 3205–3248, 2019.
- [14] T. Gu, C. Sun, X. Ma, C. Cao, C. Xu, Y. Yao, Q. Zhang, J. Lu, and Z. Su, “Practical gui testing of android applications via model abstraction and refinement,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, IEEE, 2019, pp. 269–280.
- [15] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “Yolov4: Optimal speed and accuracy of object detection,” *arXiv preprint arXiv:2004.10934*, 2020.
- [16] Y. Chen, M. Pandey, J. Y. Song, W. S. Lasecki, and S. Oney, “Improving crowd-supported gui testing with structural guidance,” in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.
- [17] J. Eskonen, J. Kahles, and J. Reijonen, “Automating gui testing with image-based deep reinforcement learning,” in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, IEEE, 2020, pp. 160–167.
- [18] K. Ivanova, G. V. Kondratenko, I. V. Sidenko, and Y. P. Kondratenko, “Artificial intelligence in automated system for web-interfaces visual testing,” in *COLINS*, 2020, pp. 1019–1031.
- [19] A. Patel, *What is object detection?* Jun. 2020. [Online]. Available: <https://medium.com/ml-research-lab/what-is-object-detection-51f9d872ece7>.
- [20] M. Xie, S. Feng, Z. Xing, J. Chen, and C. Chen, “Uied: A hybrid tool for gui element detection,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1655–1659.
- [21] S. Di Martino, A. R. Fasolino, L. L. L. Starace, and P. Tramontana, “Comparing the effectiveness of capture and replay against automatic input generation for android graphical user interface testing,” *Software Testing, Verification and Reliability*, vol. 31, no. 3, e1754, 2021.
- [22] Z. Li, K. Lu, Y. Zhang, Z. Li, and J.-B. Liu, “Research on energy efficiency management of forklift based on improved yolov5 algorithm,” *Journal of Mathematics*, vol. 2021, 2021.
- [23] W. Rahmani and A. Hernawan, “Real-time human detection using deep learning on embedded platforms: A review,” *Journal of Robotics and Control (JRC)*, vol. 2, no. 6, pp. 462–468, 2021.
- [24] T. Su, J. Wang, and Z. Su, “Benchmarking automated gui testing for android against real-world bugs,” in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 119–130.

- [25] D. Thuan, “Evolution of yolo algorithm and yolov5: The state-of-the-art object detection algorithm,” 2021.
- [26] C. Imane, *Yolo v5 model architecture [explained]*, Nov. 2022. [Online]. Available: <https://iq.opengenus.org/yolov5/>.
- [27] Z. Lv, C. Peng, Z. Zhang, T. Su, K. Liu, and P. Yang, “Fastbot2: Reusable automated model-based gui testing for android enhanced by reinforcement learning,” in *37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–5.
- [28] H. Wang, S. Zhang, S. Zhao, Q. Wang, D. Li, and R. Zhao, “Real-time detection and tracking of fish abnormal behavior based on improved yolov5 and siamrpn++,” *Computers and Electronics in Agriculture*, vol. 192, p. 106512, 2022.
- [29] K. Yasar and S. Lewis, *What is pytorch?* Nov. 2022. [Online]. Available: <https://www.techtarget.com/searchenterpriseai/definition/PyTorch>.
- [30] P. Narasimman, *Top 30 software testing tools for 2023*, Jan. 2023. [Online]. Available: <https://www.knowledgehut.com/blog/software-testing/software-testing-tools>.
- [31] J. Solawetz, *What is yolov5? a guide for beginners*. Jan. 2023. [Online]. Available: <https://blog.roboflow.com/yolov5-improvements-and-evaluation>.
- [32] *Sdlc - overview*. [Online]. Available: [https://www.tutorialspoint.com/sdlc/sdlc\\_overview.htm](https://www.tutorialspoint.com/sdlc/sdlc_overview.htm).
- [33] *Software testing overview*. [Online]. Available: [https://www.tutorialspoint.com/software\\_engineering/software\\_testing\\_overview.htm](https://www.tutorialspoint.com/software_engineering/software_testing_overview.htm).
- [34] Ultralytics, *Tips for best training results · ultralytics/yolov5 wiki*. [Online]. Available: <https://github.com/ultralytics/yolov5/wiki/Tips-for-Best-Training-Results>.
- [35] —, *Train custom data · ultralytics/yolov5 wiki*. [Online]. Available: <https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data>.
- [36] *What is computer vision?* [Online]. Available: <https://www.ibm.com/topics/computer-vision>.