FH University of Applied Sciences
TECHNIKUM WIEN

PROPOSAL

In Information and Communication Systems and Services

Precision at Pixel-Level: YOLOv8 vs. Conventional UI Testing

By: Nikolaus Rieder

Student Number: 2010258028

Supervisor: Dr. Dietmar Millinger

Wien, January 30, 2024

Contents

List of Abbreviations

CPL	Challenges, problemsand limitations
DUT	Device-under-test
LVGL	Light and versatile graphics library
ML	Machine learning
SiL	System-in-the-loop
SUT	System-under-test
TA	Test automation
UI	User interface
YOLO	You only live once

1 Short introduction

This exposé focuses on manual and automatic testing methods and highlights the problem of the continuous increase in regression testing in embedded user interface development. As tools which assist test engineers in assessing the visual aspects and functionality of devices are either lacking or not always applicable for embedded products, there exists a prevalence to employ humans to perform manual testing for quality assurance. UI technologies in web or mobile applications, have many existing solutions which can either directly access visual information and metadata about the user interface, or are capable of assisting software developers in exposing relevant information to testers. In those fields, it has also been shown that employing machine learning can greatly reduce the amount of test development labor involved in verifying user interfaces.

In the upcoming thesis, I plan to leverage existing machine learning models for object detection and bring these beneficial factors into the world of embedded systems, where the employment of these technologies has been lacking for UI testing. Through training of an ML model on the specifics of a widely used graphics library in microcontrollers and integration in a test automation solution for a distributed embedded system, it will be attempted to show that testing in those fields can benefit from state-of-the-art object detection. As an exemplary case, the user interface of a nurse call system will be tested using the implemented methods of the thesis and compared against the previous combined approach of manual and automatic testing.

In the following chapter, the existing problems in testing user interfaces at Schrack Seconet are described.

2 Problem description

Performing manual system testing on embedded devices, particularly in security systems like the Schrack Seconet nurse call system, is a labor-intensive process. It involves writing specifications and maintaining test cases, as well as executing these tests manually on the system. Specifically, the maintenance of these formally written test cases becomes increasingly complex and time-consuming with the product's time on the market. This problem is exaggarated in hardware products that inherently are part of the building infrastructure, where the general assumption is a long-time usage of the product itself. In the example of a nurse call system, a typical lifespan of devices on site can easily exceed 5 years, with the product itself being on the market for 10 years or more (as it is also the case for the Visocall IP system[1]). This means, for the firmware and software employed in these systems, a suite of regression tests is built up from the beginning of the product's development up to the end of its life cycle. To ensure quality of the product, the repeated execution of these regression tests is mandatory, especially if potential bugs in these systems can cause harm to human lives or infrastructure. Testing each one of those regression tests with every single build of the software during development is often not humanly manageable, so it is an acceptable approach, to perform a complete execution of the regression suite only once at the end of the development cycle. This is the general approach when the product development is organized in the V-Model, where the regression tests are executed in the verification phase.

For a long time, this approach was sufficient even if costly, but as time moves on and features get added, the test repository will grow in size. With every release, the time span of the manual execution time will increase and therefor stretch the release date of the software. In the last two decades, this problem was mitigated with the usage of test automation. It held the promise of reducing the workload for manual testers while also improving several factors of the test process *(e.g. execution time, defects found, invariant procedures...)*.

However, when it comes to user interfaces, it is still a common approach to perform manual testing, as it is difficult to automate in the field of embedded systems.

In mobile devices and web frontends, it is shown that test automation can be applied to improve the quality of the product, but this largely originates from the increase in frameworks and tools available to the test automation developer.

But legacy systems, which typically rely on hardware with limited performance capabilities, may lack the means to provide the TA developer with the necessary UI metadata to assess the visual aspects and functionality of the product. Such metadata includes the position, size, visual appearance and function of the user interface elements. In embedded systems, a firmware developer would have to implement functionality to expose this metadata to the test automation

---

The problem description is written from the personal expierence of the author.
He is employed at Schrack Seconet AG, with 7 years of experience in manual and automation testing at the company.

developer, increasing development time, complexity and cost.

One approach in system-in-the-loop systems testing relies on simulation of peripherals and hardware interfaces of the embedded system. In such systems the user interface will be rendered on a general purpose computer. The required UI object data for testing can then be obtained from the simulation, which then is easier to implement than exposing the metadata from the actual hardware. This approach is not always feasible, as the simulation of the hardware interfaces and peripherals can be very complex and time-consuming, especially if the firmware development does not have the time or skills to implement such a simulation. Additionally, it potentially hides bugs in the firmware, which would be exposed in a real-world scenario once the firmware is deployed on the actual product.

Another approach is the usage of additional and specialized testing hardware, connected to the embedded system. In the case of UI, controllers are required, which will read data being passed to the display and provide the test automation developer with the necessary testing metadata. This approach is also not always feasible, as it requires additional hardware to be developed and manufactured, which can increase cost and labor.

Since both of these approaches are not always achievable, it can be understandable why the tendency to perform manual UI testing on embedded systems is still prevalent. However, as development time and release cycles decrease through the usage of agile methodologies, the bottleneck of manual regression testing will become less manageable up to the point where there simply is not enough time to execute all test cases in a constrained timeframe. Looking at the already existing success of machine learning models in test automation for web frontends and mobile devices, it is fair to assume that the employment of this technology will improve quality for embedded systems. In the following pages, several research questions regarding the popular object detection model YOLOv8 are stated, which aim to assess the usage of machine learning in test automation for user interfaces on embedded devices. It is to be determined if the ability of such models can outperform the existing manual or automated testing methodologies used on such systems.

2.1 Problem context at Schrack Seconet

The implemented UI by Schrack Seconet uses the light and versatile graphics library[2] (LVGL), a popular choice in embedded hardware for developing user interfaces. This framework already comes with implementations that allow for simulation of the UI rendering on a development machine, which already improves development time since it doesn't require display hardware to work on UI design. But being a library targeted at embedded systems, there are no integrated testing tools that would allow a test automation developer to verify the functionality of those designs on the actual hardware. Since the nurse call system Visocall IP[1] is a distributed embedded system, it relies on multiple hardware devices to interact with each other and UI functionality is tied to the requests and responses of the system as a whole. When it comes to simulation, it marks unit testing of the user interface as an unusable solution, due to the inherent

The problem description is written from the personal expierence of the author. 4
He is employed at Schrack Seconet AG, with 7 years of experience in manual and automation testing at the company.

complexity of the system and the required simulation of multiple devices or peripherals for the UI to function properly. Instead, the developers at Schrack Seconet integrated driver modules into the firmware of UI devices, which allows for simulation of user interactions on the devices, mimicking human behavior on the display. The mimicked behavior includes interactions such as touch, press, release, drag and combined variations of the aforementioned.

However, this approach requires knowledge about the metadata of the UI by the test developer, so that the defined interactions are performed on the correct coordinates of the user interface element. Expected functionality of the device is then assessed by aggregation of device logging data and verification of expected signal states from accessible hardware outputs. These outputs typically are LEDs on the test device itself or other interconnected devices in the distributed system, which are connected through proprietary IP-based protocols or other proprietary hardware bus technologies employed by the Schrack Seconet system.

This approach works well for a given user interface design which is not affected by frequent changes, but it lacks verification on the visual aspect of the user interface itself. Furthermore, it comes with an inherent problem in maintenance and development time, since the test automation engineer is required to determine the element coordinates through reverse engineering or by inspecting the firmware codebase. The latter approach is generally frowned upon in our company, as we prefer a tendency towards black-box-testing, which mitigates the chance of overlooking software bugs due to code blindness. It is preferred for the test automation engineer to have as little knowledge about the implementation of the firmware as possible, so that the tester can focus on unbiased verification of the target device. It also would not be a viable approach to look through the codebase to determine the element coordinates, since it typically is not blatantly written inside the code and would require the tester to work through the programmatic abstractions.

The easiest solution to determine coordinates of UI elements is through the usage of screen dumps from the test device itself. The generated image file has the exact dimensions as the device display, and it is easy to determine coordinates through generally available image viewers, such as IrfanView[3].

The positions of UI elements is then embedded into automated test case routines, which will perform the defined UI interactions. The expectations on functionality is then verified through existing test framework capabilities. However, this approach comes with a caveat, as the hard-coded metadata is bound to become invalid when UI changes are introduced and implemented. This results in required rework of the affected test cases. This can be mitigated through abstractions (e.g. global variables, re-usable routines), but ultimately remains as labor-intensive maintenance to avoid false-positives in the testing process.

Another perspective on the current approach is the existing dependency on manual testing, since the described test method does not involve visual verification. The existing test automation can reliably assess the correct execution of functions in the system, but is incapable of finding errors that exist in the displayed image and animation on the device itself. The visual aspect of the UI plays an important in nurse call systems, since it involves multiple languages

---

The problem description is written from the personal expierence of the author. 5
He is employed at Schrack Seconet AG, with 7 years of experience in manual and automation testing at the company.

for text, standards for visual and acoustic signalization described in the VDE-0834[4], and also the visual prioritization of emergency calls.

Making use of the aforementioned screen dumps to assess these visual aspects is not a valid approach, since it does not account for display driver issues that may occur through either hardware defects or transmission errors from the device controller to the display controller.

A solution, which is capable of dealing with the dynamic nature of user interfaces, visual verification and future UI development changes is therefor highly sought after in the company. Making use of computer vision and machine learning in embedded systems testing is assumed to be a promising approach for the purpose of reducing the cost and time involved in manual and automatic testing of these user interfaces.

The problem description is written from the personal expierence of the author. 6
He is employed at Schrack Seconet AG, with 7 years of experience in manual and automation testing at the company.

3 Research state

The topic of the thesis is specific to applying an ML based object detection model to visual GUI testing (VGT), on devices of a distributed embedded systems. A direct comparison in that regard has not been found at the time of writing. But the comprehensive survey [5] on embedded systems testing serves as a good start for systematic approaches.

Existing research on UI testing from test targets of other fields such as software, mobile devices and web can be used to extract methodologies, comparative analysis and results, to draw conclusions about the proposed quality improvement when using an ML model approach for testing. The main differentiation in the thesis is the existing constraints when testing embedded systems, inherited largely from the difficulty of making visual information accessible for test automation purposes when working with embedded devices.

In an article [6], the method of using record-and-play is described, with the goal of improving test accuracy. It uses the display data sent to a host PC and analyzes events performed by a manual tester to record test cases. In another step, the recorded events are played back to the device. However, this approach is not very easy to perform on distributed embedded systems, with various displays involved. It also comes with a performance challenge and the cost of proprietary technologies used.

Since the thesis involves the transitioning from a manual system test suite for the embedded UI, to an automated one with ML capabilities, a study [7] in the software development field was found, which detailed the results and effects of such a transition. Their findings confirm that an automated visual testing approach can be beneficial and bring improvement. An additional later study on VGT by the same authors analyzed the challenges, problems and limitations (CPL) when it comes to transitioning to visual UI testing.[8] In their study, they discovered 58 CPLs, categorized into 26 types. A more thorough investigation of the study is still due in context of applicability on the embedded system to be tested, but it is anticipated that the analytic methodology of this study is applicable in the thesis.

In the field of mobile applications, a few studies were made using ML models to perform testing or create metadata of the graphical model of the UI. (i.e. element types, hierarchical structure, detection and recognition, state) [9]–[15] A common dataset used in these studies is the RICO dataset[16]. These studies share similarities of the upcoming research in regard to their usage of an ML model for gathering metadata about UI. Notably their software applications and user interfaces typically are written in languages and frameworks with better tooling support for performing visual automated testing.

An interesting case was made with the AI-driven tool "Owl Eye"[17], which can detect visual defects or inconsistencies in graphical user interface, based purely on provided screenshots.

Two models, Faster-RCNN and YOLOv8, were trained on the RICO dataset, with a concluding choice on YOLOv8 due to higher accuracy and faster training time.

These aforementioned studies hint towards a possible improvement through the usage of ML for visual testing and also the general improvement of quality, due to performing better than manual testing. As such, the thesis will explore if such an improvement can be made.

In regards to the testing framework used, it is already shown[18] in that Robot Framework can improve the quality of the testing process.

4 Research questions

In order to properly assess the effectiveness and applicability of visual testing with ML technologies on an embedded system, a few questions need to be raised.

As my thesis solely covers the employment of the YOLOv8 model, the main goal lies in comparing the trained model against the traditional testing method (i.e. combined manual and automatic testing for UI)

4.1 How does YOLOv8 enhance UI widget detection?

To answer this question, the accuracy of the model is compared against traditional methods. Since it can be assumed, that ML detection speed vs. human reverse engineering of UI element coordinates is an unfair comparison, the answer will focus on the precision the model can provide. Since the reliability of the model is critical, inherited from the critical functionality the UI of the security system performs, an extensive analysis of the model accuracy is required. It is however not anticipated to overcome the accuracy of a manual tester when it comes to identifying UI elements, as human levels of object detection have not been reached in that regard. Additionally, the thesis wants to determine if the test development speed will increase when employing the model. For this, we will factor in development time of automated test cases with and without the ML model.

4.2 What challenges does YOLOv8 overcome in automated embedded UI testing?

Errors during development of the test automation solution are common, and as such they will also appear during the integration of the model. It will be determined, if the integration of the model is less error-prone than the writing of traditional manual and automatic test cases. A list of CPLs types and categories will be taken from [8] and additional ones will be added in regard to the writing process of test cases, as it is considered to be a vital part of the improvement in the case.

An important challenge in regard to the traditional approach for the model is the experience of the manual testers and their familiarities with the UI design and critical functions of the system. If the ML model produces more false-positives and it is easier to embed errors in test routines, it

might fail adoption and also decrease its value in comparison to the traditional approach, given the required reliability.

The beginning of the test repository from the target system was over a decade ago, as such, many examples of UI bugs exist that were already detected through the traditional methods. The ML approach will be tested with these bug conditions to determine if they can be found with the new approach. Additionally, the ongoing development of a new user interface will be taken into account and a direct comparison of errors found with and without the model in place will be made.

In a statistical sense, the error reduction of post ML integration will be determined as well as the analysis of errors found in the sample bugs of the previous decade.

The overcoming of these challenges will be determined to assess the viability of using ML as a solution in testing critical embedded systems with user interfaces, given that the UI itself plays an important role in the emergency aspect of the tested system.

4.3 How effectively does YOLOv8 detect and classify Widgets in various UI layouts?

As a benchmark measure, the detection and classification accuracy of the model will be assessed. The results will be compared to other model performances. The model will also need to handle the case of varying user interfaces in an ongoing UI design process, meaning these metrics will be tracked over the entirety of the implementation and integration.

4.4 Which limitations of current testing methods does YOLOv8 address?

The limitations of the current methods reside in the man-hours involved for manual and automatic testing, the involved error rates with that method, and the bad scalability due to required specification changes. The man-hours for the development of the past and the ongoing UI testing will be taken into account, alongside a regression analysis to determine the improvements with the model usage.

5 Methodology

In the thesis, a multi-stage approach is proposed as showcased in Fig. 1.

The first stage involves synthesizing a dataset for training the machine learning model. This dataset will be created using two custom tools: a UI Generator and a UI Randomizer. The randomizer calls the generator to construct a dataset of UI screenshots with varying layouts and widget types.

A base model will then be trained on this varied and randomized UI dataset in the second stage, using the YOLOv8 model [19]. The model will learn the characteristics for detecting LVGl objects.

In the third stage, the model will be fine-tuned using proprietary UI screenshots from the actual embedded system under test. A docker image from this stage will be created to be deployed to a container host.

In the last stage, the Robot Framework [20] will fetch a screen dump from the device-under-test (DUT) and pass it to the UI detector container. Object detection will be performed and the returned metadata will be provided to a test case as metadata for UI interaction. When the test case starts, it will use the metadata to interact with the DUT.

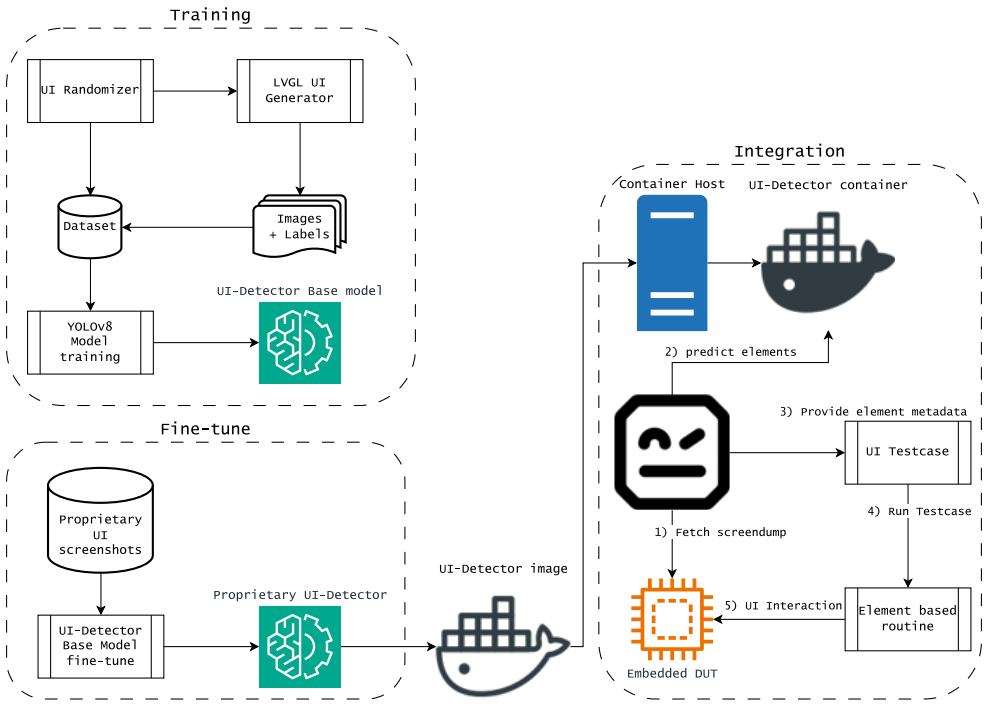

Figure 1: Proposed multi-stage approach.

5.1 Data Preparation

The first stage involves synthesizing a dataset for training the machine learning model. This dataset will be created using two custom tools: a UI Generator (Fig. 2) and a UI Randomizer (Fig. 3). The UI Generator, developed in C/C++, will produce images of user interfaces using a variety of widgets from the LVGL library, which is commonly employed in embedded systems for UI development. These generated images, along with their corresponding labels, serve as the initial training data for the YOLOv8 object detection model.

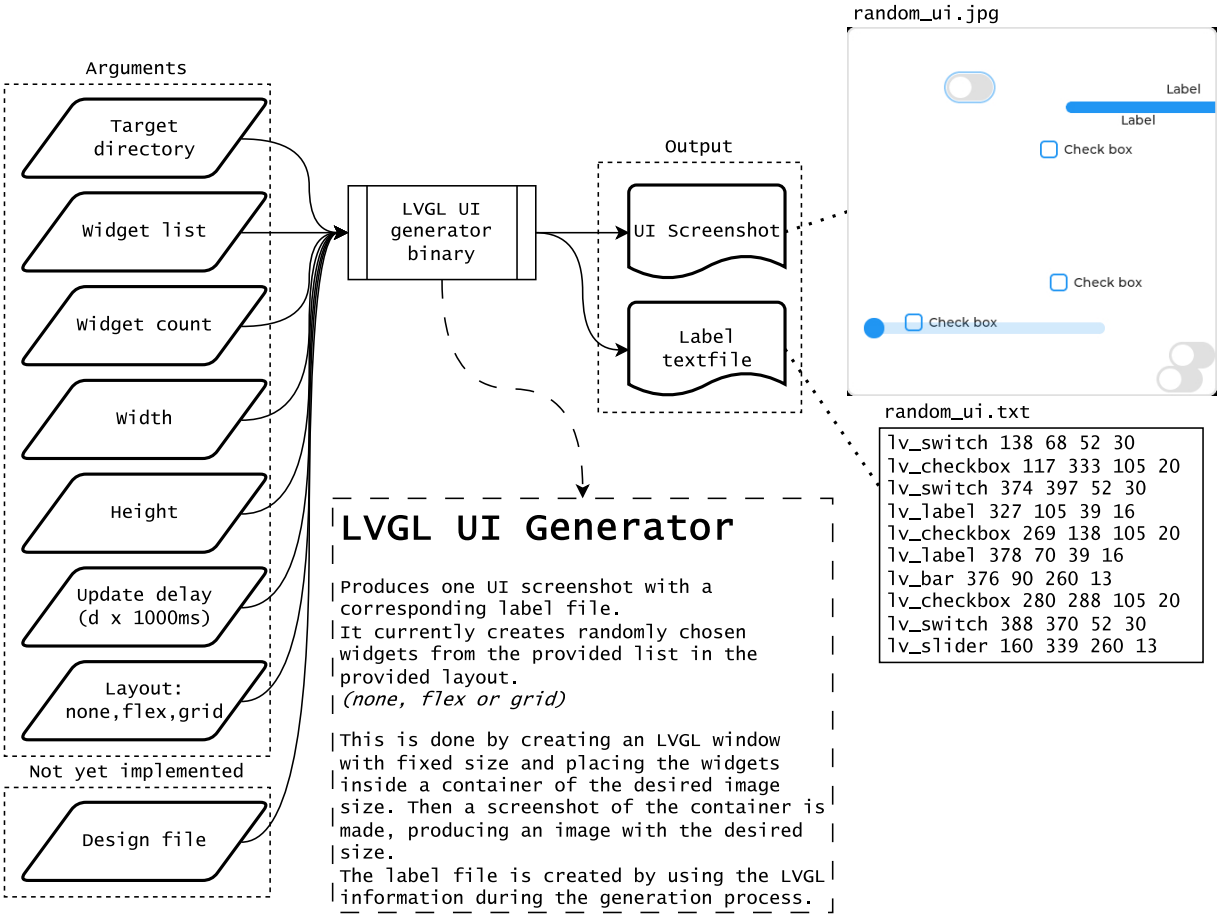

Figure 2: Diagram of the LVGL UI Generator binary.

The UI Randomizer, a Python script, will automate the generation of a large number of varied UI layouts by invoking the UI Generator with different parameters. This ensures a diverse training set, which is crucial for the robustness of the machine learning model. The randomizer will organize the generated data into training, validation, and testing sets, and preprocess the labels to convert widget names into class IDs and normalize pixel values for the object detection task.

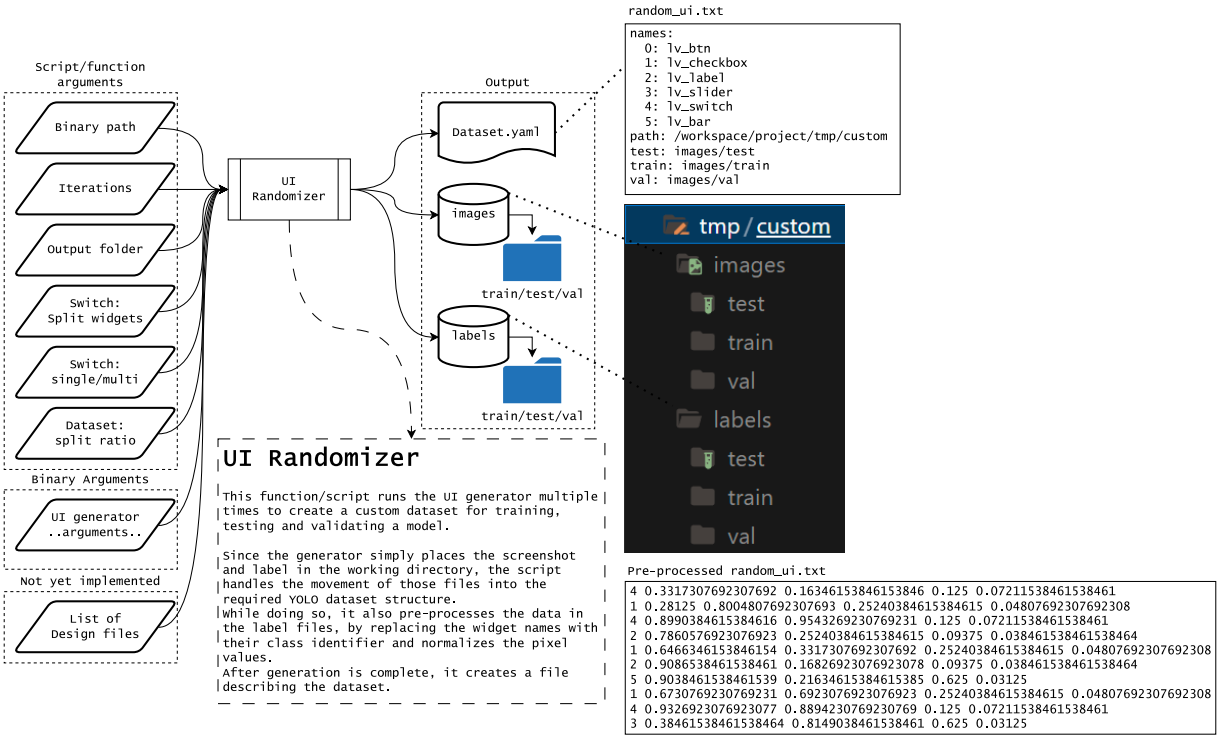

Figure 3: Diagram of the UI Randomizer script.

5.2 Model Training and Fine-Tuning

Once the dataset is prepared, the YOLOv8 model will be trained on the synthesized data. Training will be carried out until the model achieves a satisfactory level of accuracy on the validation set. After the base model training, fine-tuning will be conducted using proprietary UI screenshots from actual embedded systems, ensuring the model adapts to real-world examples similar to those it will encounter in production.

5.3 Integration into Testing Environment

The trained model will then be encapsulated within a Docker container, creating a portable and isolated UI-Detector. This detector will be integrated into a container host environment, which will manage the execution of the UI-Detector and facilitate interaction with the embedded

system under test. The container will be responsible for predicting UI elements and providing metadata essential for the test automation.

5.4 Automation of Testing Process

The final stage involves the actual automation of the testing process. This includes acquiring screendumps from the embedded device, which the UI-Detector will use to perform object detection. The identified UI elements and their metadata will be used in automated test cases, and the UI interaction routines will interact with the embedded DUT based on the test cases. The entire process aims to minimize manual intervention, reduce errors in test creation, and adapt quickly to changes in UI designs.

This comprehensive methodology integrates machine learning into the existing test framework, significantly enhancing the capabilities of visual regression testing in embedded systems. The use of Docker ensures a consistent and replicable testing environment, while the focus on automation aims to streamline the entire testing process, from data generation to test execution.

6 Thesis outline

1. Introduction
2. Existing literature
3. Problem Statement
    - Problem context at Schrack Seconet
4. Research Objectives
5. Research Questions
    - How does YOLOv8 enhance UI widget detection compared to conventional methods?
    - What challenges does YOLOv8 overcome in automated embedded UI testing?
    - How effectively does YOLOv8 detect and classify widgets in various UI layouts?
    - Which limitations of current testing methods does YOLOv8 address?
6. Tools and Technologies
    - YOLOv8
    - LVGL
    - Robot Framework
    - Docker
7. Methodology
    - Training
    - Integration
    - Automation
8. Implementation
    - Development Environment
9. Research Constraints and Limitations
10. Conclusion
11. Bibliography

Bibliography

[1] *Visocall IP | Moderne IP Kommunikationslösung*, https://www.schrack-seconet.com/de/healthcare/kommunikationssystem-visocall-ip/. (visited on 01/30/2024).

[2] *LVGL - Light and Versatile Embedded Graphics Library*, https://lvgl.io/. (visited on 01/30/2024).

[3] *IrfanView - Official Homepage - One of the Most Popular Viewers Worldwide*, https://www.irfanview.com/. (visited on 01/30/2024).

[4] *DIN VDE 0834-1 VDE 0834-1:2016-06 - Normen - VDE VERLAG*, https://www.vde-verlag.de/normen/0800319/din-vde-0834-1-vde-0834-1-2016-06.html. (visited on 01/30/2024).

[5] V. Garousi, M. Felderer, Ç. M. Karapıçak, and U. Yılmaz, "Testing embedded software: A survey of the literature," *Information and Software Technology*, vol. 104, pp. 14–45, Dec. 2018, ISSN: 09505849. DOI: 10.1016/j.infsof.2018.06.016. (visited on 01/24/2024).

[6] Y.-D. Lin, E. T.-H. Chu, S.-C. Yu, and Y.-C. Lai, "Improving the Accuracy of Automated GUI Testing for Embedded Systems," *IEEE Software*, vol. 31, no. 1, pp. 39–45, Jan. 2014, ISSN: 0740-7459. DOI: 10.1109/MS.2013.100. (visited on 01/25/2024).

[7] E. Alegroth, R. Feldt, and H. H. Olsson, "Transitioning Manual System Test Suites to Automated Testing: An Industrial Case Study," in *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, Luxembourg, Luxembourg: IEEE, Mar. 2013, pp. 56–65, ISBN: 978-0-7695-4968-2 978-1-4673-5961-0. DOI: 10.1109/ICST.2013.14. (visited on 01/24/2024).

[8] E. Alégroth, R. Feldt, and L. Ryrholm, "Visual GUI testing in practice: Challenges, problemsand limitations," *Empirical Software Engineering*, vol. 20, no. 3, pp. 694–744, Jun. 2015, ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-013-9293-5. (visited on 01/24/2024).

[9] M. D. Altinbas and T. Serif, "GUI Element Detection from Mobile UI Images Using YOLOv5," in *Mobile Web and Intelligent Information Systems*, I. Awan, M. Younas, and A. Poniszewska-Marańda, Eds., vol. 13475, Cham: Springer International Publishing, 2022, pp. 32–45, ISBN: 978-3-031-14390-8 978-3-031-14391-5. DOI: 10.1007/978-3-031-14391-5_3. (visited on 01/24/2024).

[10] J. Cheng and W. Wang, “Mobile Application GUI Similarity Comparison Based on Perceptual Hash for Automated Robot Testing,” in *2021 International Conference on Intelligent Computing, Automation and Applications (ICAA)*, Nanjing, China: IEEE, Jun. 2021, pp. 245–251, ISBN: 978-1-66543-730-1. DOI: 10.1109/ICAA53760.2021.00052. (visited on 01/24/2024).

[11] Y. Li, G. Li, L. He, J. Zheng, H. Li, and Z. Guan, *Widget Captioning: Generating Natural Language Description for Mobile User Interface Elements*, Oct. 2020. arXiv: 2010.04295 [cs]. (visited on 01/24/2024).

[12] B. Selcuk and T. Serif, “A Comparison of YOLOv5 and YOLOv8 in the Context of Mobile UI Detection,” in *Mobile Web and Intelligent Information Systems*, M. Younas, I. Awan, and T.-M. Grønli, Eds., vol. 13977, Cham: Springer Nature Switzerland, 2023, pp. 161–174, ISBN: 978-3-031-39763-9 978-3-031-39764-6. DOI: 10.1007/978-3-031-39764-6_11. (visited on 01/24/2024).

[13] T. Zhang, Y. Liu, J. Gao, L. P. Gao, and J. Cheng, “Deep Learning-Based Mobile Application Isomorphic GUI Identification for Automated Robotic Testing,” *IEEE Software*, vol. 37, no. 4, pp. 67–74, Jul. 2020, ISSN: 0740-7459, 1937-4194. DOI: 10.1109/MS.2020.2987044. (visited on 01/24/2024).

[14] T. Zhang, Z. Su, J. Cheng, F. Xue, and S. Liu, “Machine vision-based testing action recognition method for robotic testing of mobile application,” *International Journal of Distributed Sensor Networks*, vol. 18, no. 8, p. 155 013 292 211 153, Aug. 2022, ISSN: 1550-1329, 1550-1477. DOI: 10.1177/15501329221115375. (visited on 01/24/2024).

[15] S. N. Cavsak, A. Deliahmetoglu, B. T. Ay, and S. Tanberk, “GUI Component Detection Using YOLO and Faster-RCNN,”

[16] B. Deka, Z. Huang, C. Franzen, *et al.*, “Rico: A Mobile App Dataset for Building Data-Driven Design Applications,” in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, Québec City QC Canada: ACM, Oct. 2017, pp. 845–854, ISBN: 978-1-4503-4981-9. DOI: 10.1145/3126594.3126651. (visited on 01/25/2024).

[17] A. Gamal, R. Emad, T. Mohamed, O. Mohamed, A. Hamdy, and S. Ali, “Owl Eye: An AI-Driven Visual Testing Tool,” in *2023 5th Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, Giza, Egypt: IEEE, Oct. 2023, pp. 312–315, ISBN: 9798350381030. DOI: 10.1109/NILES59815.2023.10296575. (visited on 01/25/2024).

[18] J. T. Walker, “Software Test Automation with Robot Framework,” *International Journal of Computer Applications*, vol. 175, no. 25, pp. 8–14, Oct. 2020, ISSN: 09758887. DOI: 10.5120/ijca2020920784. (visited on 01/24/2024).

[19] G. Jocher, A. Chaurasia, and J. Qiu, *Ultralytics YOLO*, version 8.0.0, Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics.

[20] *Robot Framework*, https://robotframework.org/. (visited on 01/25/2024).

List of Figures