# TVTAT - A Real Time Camera Imaging Testing Tool for Smart TVs: Preliminary Results

Carlos Alberto Hagge da Cunha Filho
SIDIA Institute of Technology
Manaus, Amazonas, Brazil
carlos.cunha@sidia.com

Hugo Abreu Mendes
SIDIA Institute of Technology
Manaus, Amazonas, Brazil
hugo.mendes@sidia.com

Adriano Rodrigues de Paula
SIDIA Institute of Technology
Manaus, Amazonas, Brazil
adriano.paula@sidia.com

Ravi Barreto Doria Figueiredo
SIDIA Institute of Technology
Manaus, Amazonas, Brazil
ravi.figueiredo@sidia.com

Jessamine Maria de Lima Azevedo*
IAV do Brasil
São Paulo, São Paulo, Brazil
jessamine.azevedo@sidia.com

## ABSTRACT

Test automation tools that can accurately control a smart TV device are rare to be found. The difficulty of creating a system that is able to control such a device generates specific needs. In addition, the Brazilian digital television system supports DTVPlay, a middleware that provides the ability to broadcast interactive applications written in HTML5, NCL and Lua, that must be fully implemented on at least 90% of televisions manufactured in Brazil. Thus, there is a standard to be followed and a set of tests that need to be performed with each new middleware release. This work presents an automation tool called TV Test Automation Tool (TVTAT), that performs non-invasive tests on smart TVs, mainly but not restricted to DTVPlay tests. TVTAT uses real-time computer vision techniques such as optical character recognition, image pattern matching and color verification to assert that the middleware's implementation is according to the published specification. The results of some test scenarios are presented, demonstrating that there are trends that can be found either in application performance situations or in tests of availability of transmitted applications that depend on DTVPlay.

## KEYWORDS

Testing Tools, Software Testing, Smart TV Testing, DTVPlay, Digital Television

## 1 INTRODUCTION

Smart TVs and their ecosystem are becoming increasingly more complex, this fact and the demands for quality in tech products means that assuring this quality through testing is becoming a more difficult. Testing smart TV products, its features, native applications, and its use as a platform for third-party applications presents distinguished challenges to be overcome by an automated test tool, however it is an indispensable part of its development. When field tests are not correctly conducted, the final user's experience may be negative, in addition to it, fixes for issues found on software and hardware already in production are much more expensive to deploy.

In a test environment, to ensure the precise measurement and identification of a process or service event, one has a few options:;

- Create an externally visible code checkpoint, e.g. a log event written to a serial port in the smart TV's internal software routines;
- Observe an event displayed on the smart TV display.

Today, both methods of measurement are used to perform evaluations during a test. However, there are still challenges related to the second method. The observation of a given visual event could be done in two ways:

- Human verification;
- Automated verification using image processing or computer vision techniques.

Automatic testing systems can assist testers so they perform fewer repetitive tasks, freeing them time to focus on more analytical activities. Those tools rely on computer vision to mimic/improve human visual analysis of events displayed by the system under test. Some automatic testing systems were already a known and useful solution for non-smart TVs [18, 21]. With the release of new, more complex operating systems for smart TVs and their increasing number of embedded applications from third parties, the automation of this type of system test is an area of growing importance [2, 5, 9, 11, 19].

The TVTAT automates the testing of smart TV in order to ensure their quality, functionality and performance. Figure 1 presents a comparison of a hypothetical test case tested manually and with TVTAT. The Test Case Data is responsible to generate a set of the instructions to simulate interactions with the TV through infrared blaster commands or power cords turning off. TVTAT is responsible to execute those instructions and verify if they were executed correctly using computer-vision techniques.

In this work, two experimental applications of the tool are presented. The first application generates a set of results when testing the TizenOS implementation of the Integrated Services Digital Broadcasting (ISDB-Tb) used by the Brazilian Digital Television System (SBTVD). It provides a middleware for interactivity features named DTVPlay, previously known as Ginga [6, 15], focusing on the features used to implement Nested Context Language (NCL) and HyperText Markup Language Version 5 (HTML5) applications.

*Former employee that can be contated through the email jessamine.azevedo@gmail.com.

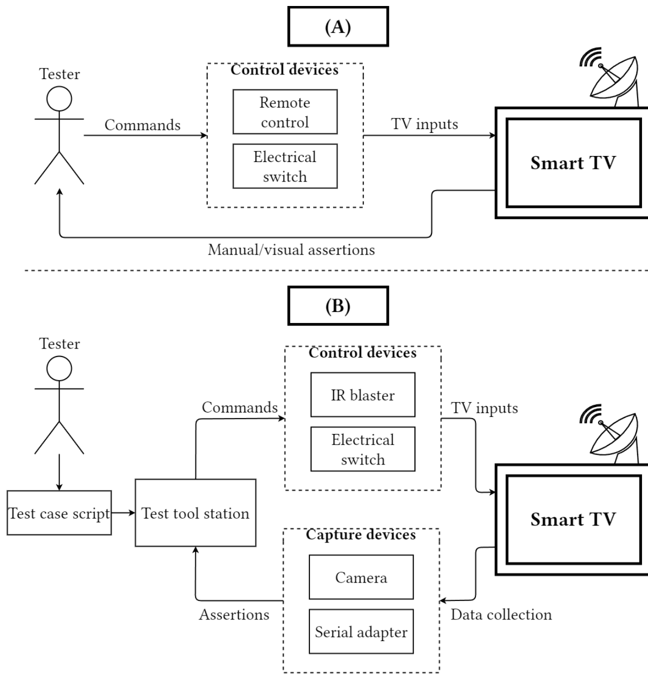



**Figure 1: Overview of the proposed tool. In (A), the manual test loop of a hypothetical test case being performed by a tester is shown. In (B), the testing process is automated using TVTAT and the repetitive test loop of commands and assertions is handled by TVTAT using hardware devices.**

The second application is a benchmark among different TV models to measure performance of content availability in a video streaming app. The goal of this work is to automate the tests that are made manually and guarantee the robustness and performance, diminishing the time of the realization of the tests.

The remaining of this paper is organized as follows: Section 2 summarizes the related works in the literature about automated tests in smart TV. Section 3 gives a brief Introduction on DTVPlay. Section 4 explains the materials and methods, the algorithms used by the tool and how a tester can create test scripts that can be executed by the tool. In Section 5, experiments and results on benchmarks, the data collected by the tool when executing a DTVPlay test suite and performance tests of a smart TV application are presented and analysed. Finaly, the conclusion is presented in Section 6.

## 2 RELATED WORKS

Yeh presents in [28] a tool called Sikuli, which uses a visual approach to automate graphical user interfaces using computer vision methods on screenshots. It provides a visual scripting Application Programming Interface (API) for automating Graphical User Interface (GUI) interactions, using patterns on the screen to direct mouse and keyboard events. Some scripts are presented on the work, that automate some suitable tasks, such as map navigation and bus tracking.

In [10] there is a focus on automating the generation of test cases for web applications, based on a Markov reward process with the reward being modeled by a function dependent on the number of asserts failed, code failure and timeout. The experiment shows a significant improvement on the defect detection capability of test cases generated through Markov reward process. It also presents an automated framework to test the smart TV apps.

In [1] it is shown that there is an intrinsic challenge in automatic testing of smart TV software with open problems related to algorithms for test case generation. It shows that one of the most crucial problem to be solved is the test generation strategy, and presents an automated framework to test the smart TV apps. It also presents an algorithm called EvoCreeper that detects all necessary view needed in the model for test generation.

Maia et al [20] presents a methodology for automatically testing the module Ginga CC Webservices, using DTV receivers that include the DTVPlay middleware. Maia shows an improvement by a factor of three, e.g. from 395 minutes using a manual approach to 131 minutes using an automated approach.

The approach used in TVTAT distinguishes itself from the state of the art by providing a unique set of features: support for any device that has a display and an IR receiver, and assertions that are non-invasive and real-time. This set of features enables it to better simulate the way the end user interacts with the device. The methodology presented in [20] is the one that most closely resembles the one used by TVTAT, however a major difference is the camera usage. In [20], the camera's main and only duty is to capture image evidences, but assertions are done using the images.

## 3 DTVPLAY

Ginga is an interoperability layer of the Brazilian digital TV system capable of decoding interactive applications sent by broadcasters. Interactive applications allow TV users to interoperate with the digital TV context. DTVPlay is the new generation of Ginga that integrates the broadcaster with internet services creating a Hybrid TV concept.

The DTVPlay architecture is composed of multiple subsystems that decode broadcaster apps at Digital Television (DTV) receiver, control app life cycle, perform media playback and monitor broadcast events. Among the supported DTVPlay applications are NCL, Lua, HTML5, etc.

DTVPlay's specifications are defined by several technical standards published by the Brazilian Association of Technical Standards (ABNT), the list can be found in [14] and is composed of the Ginga Common Core (Ginga-CC) subsystem, supporting NCL and HTML5 applications [24]. These specifications form the basis of one the test suites presented in this work.

DTVPlay NCL features are based on the principle of closed loop information that allows the user, or the terminal, to send information back to the broadcaster via Transmission Control Protocol (TCP) [8]. Therefore, Testing DTVPlay is a necessity to assert that all features work as expected and are ready for the broadcasters to make applications based on this system available [3, 27]. Recent works demonstrate the challenges around middleware testing, as the lack of regulation about its implementation can lead to standardization issues [12].

# 4 MATERIALS AND METHODS

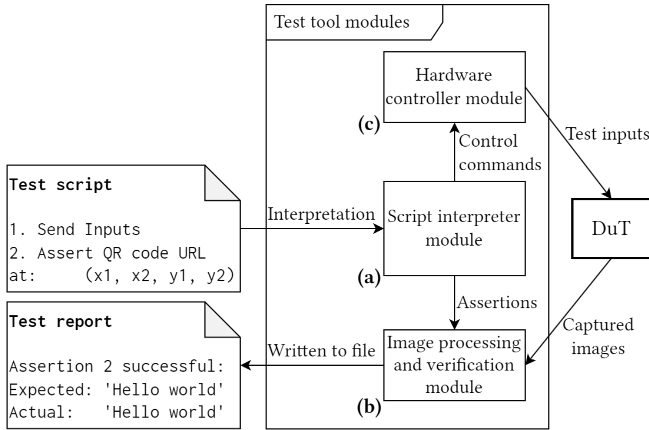



**Figure 2: Flow of data between the tool's modules during the execution of an assertion defined in test script.**

The system's architecture and its flow of information are shown in Figure 2, separated into three main modules: (a) script interpreter; (b) image processing and verification; and (c) hardware controller. The script interpreter controls the execution process, by reading the test case scripts, executing the commands and assertions in them. Verification tasks are delegated to the image processing and verification module, that uses the images collected by the camera's video stream to assert that the state of the Device under Test (DuT) is the one expected by the test case. The hardware controller module is tasked with sending inputs to the DuT, simulating user interactions defined by the test case.

In Figure 3, the connections of the TVTAT with front-end, back-end and controlled devices are shown. The web front-end consist of a page whose main role is to make the live video being recorded by the tool to be more accessible to multiple remote users while the devices are in the testing process. The HTML5 page with static components is served by an HTTP server, that contains dynamic components rendered and made interactive by JavaScript components. The back-end is responsible for starting the Flask (HTTP) server and the JavaScript application that reads the local FFmpeg video stream, encodes it to MPEG-2 and pushes the encoded video data to the client using a websocket.

The desktop GUI is based on Tkinter, the GUI framework from Python's standard library, and is the main GUI used by the testers. Their features include a built-in text editor for writing automation scripts, camera configuration screens, homography settings, manual testing and management of the automated test case execution sequence.

Still on the back-end, there is the portion of the TVTAT, in which frame preprocessing and frame verification for test case events are carried out. The back-end is also responsible for recording the test case video using FFmpeg, which can be used for verifications after those performed automatically. In addition, TVTAT exposes an API that can be used to control Transport Stream files that are transmitted by a modulator device, using the command line program

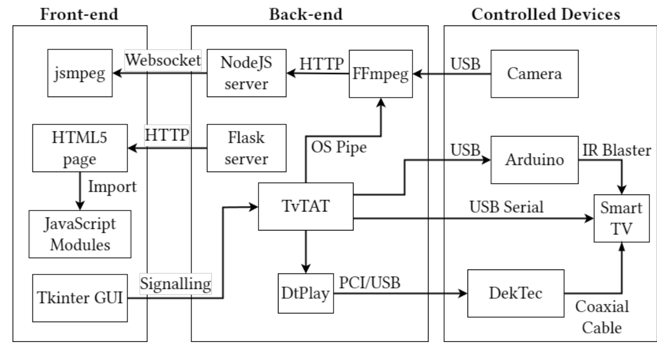



**Figure 3: Flow of data between the tool, controlled devices and front-ends**

DtPlay [4]. Also, the API allows the use of the Arduino device to control TV navigation through infrared commands, capture logs and control TV via serial interface.

## 4.1 Test case scripting

Test tools can be classified as scripted or scriptless [7], the first reads a set of already predefined instructions created either by the tool developers or a test case creator. Scriptless test tools try to navigate through the functionalities of the system under test using some Action Selection Mechanism (ASM) [7].

In TVTAT, the test case scenarios are already pregenerated, and their behavior is well-defined by the DTVPlay specification. For storing the execution sequence of the sets of test cases that compose a scenario, TVTAT uses a file format based on JavaScript Object Notation (JSON) format. These files are created and edited by the user through TVTAT's GUI.

The test scripts can be written in two languages: a TVTAT domain-specific language, called TV Test Automation Tool Script (TVTATScript), or Python 3. TVTATScript files are parsed and interpreted by the TVTAT application, while the Python scripts call the application's classes and procedures directly. The diagram shown in Figure 4 describes some steps executed by the script interpreter in an example test case:

- **(A)** The expected user input to the TV is read from the script and sent using external hardware;
- **(B)** an expected response is read from the script, in this case a QR code with a specific value should be displayed at some position on the screen. The tool then verifies this in real time, reading the camera's video stream;
- **(C)** the expected and actual result are stored and afterwards written to a test report that is visible to the user.

Using the test scripts, the tester can control the TV through an infrared blaster that emulates a user's remote control, and toggle the TV's power supply by sending commands to an Arduino electrical relay connected to the TV's power line. Additionally, the user can create assertions using verification methods that act upon the frames captured by the connected camera, collect evidence as video recordings and screenshots, call other TVTAT scripts, and implement basic logic using conditionals and counter statements.

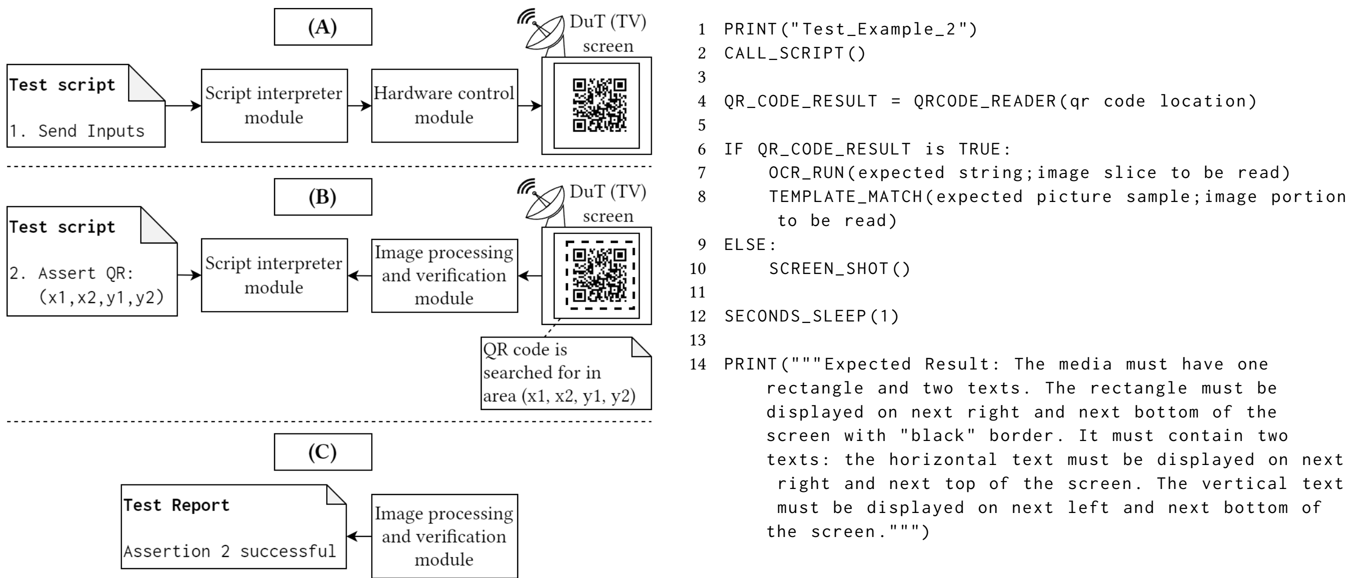



**Figure 4: Overview of interactions between the DuT and test script commands being executed by the test tool, where each of the steps is sequential in time.**

Two examples of test case scripts are shown below in the form of pseudocode. In Pseudocode 1, the test involves reading a QR code that marks the start of the test case, indicated in line 5 of the script. If the QR code is read correctly, line 7, it performs color verification, line 8. Otherwise, it performs a screen capture, line 10. Then, the expected result is printed and the artifacts are saved.

**Pseudocode 1: Example TVTATScript language with QR code reader and color verification**

```
1  PRINT("Test_Example_1")
2
3  CALL_SCRIPT()
4
5  QR_CODE_RESULT = QRCODE_READER(qr code location)
6
7  IF QR_CODE_RESULT is TRUE:
8      COLOR_VERIFY(color hex code; image slice location)
9  ELSE:
10     SCREEN_SHOT()
11
12 SECONDS_SLEEP(1)
13
14 PRINT("""Expected Result: The rectangle with "blue"
       color and with "fill" mode must be displayed on left
        and top of the screen.""")
```

In the second script represented by the Pseudocode 2, the process is analogous to the first, but instead of a color verification, it expects the detection of an object dynamically generated by the test and rendered by DTVPlay on the TV's screen, the tests consists in trying getting text content with Optical Character Recognition (OCR), line 7, then, a template match is executed, line 8.

**Pseudocode 2: Example TVTAT language with OCR and template match**

```
1  PRINT("Test_Example_2")
2  CALL_SCRIPT()
3
4  QR_CODE_RESULT = QRCODE_READER(qr code location)
5
6  IF QR_CODE_RESULT is TRUE:
7      OCR_RUN(expected string;image slice to be read)
8      TEMPLATE_MATCH(expected picture sample;image portion
        to be read)
9  ELSE:
10     SCREEN_SHOT()
11
12 SECONDS_SLEEP(1)
13
14 PRINT("""Expected Result: The media must have one
       rectangle and two texts. The rectangle must be
       displayed on next right and next bottom of the
       screen with "black" border. It must contain two
       texts: the horizontal text must be displayed on next
        right and next top of the screen. The vertical text
        must be displayed on next left and next bottom of
       the screen.""")
```

## 4.2 Image processing

TVTAT makes use of different image processing techniques to improve its assertion capability. Figure 5 demonstrates the frame flux for user visualization and processing for tests verification. The tool implements different types of verification methods, the main ones being: QR code reader, template matching, color verification and OCR, that defines which frame processing algorithm will be used for validation. The QR code assertion command uses the QR code detection method from OpenCV, the picture command uses the OpenCV template match method, following this logic, for color verification and optical character recognition, the Python colormath library and Tesseract OCR were used, respectively.

The main preprocessing done to the captured video frames is homography and perspective transformation, to enable flexibility in the camera positions and angles used in the test bed setup [16]. To set up the homography, the tester can use a graphical user interface window. There, the correct camera can be selected on a menu and the homography coordinates can be set by dragging the edges of the homography polygon to match the edges of the TV display that is shown in a live video stream from the selected camera. Finally, the user will save the configuration by clicking on the save button. The result is that the processed frames contain only the TV image as a rectangle, with the background and perspective removed.

The homography plus the perspective correction runs for every frame, the user is capable of checking the raw and post homography videos to check whether the quality is sufficient before starting recording the test cases. Test cases written from multiple camera angles can be executed, reused and mixed due to this feature.

Most verification methods also crop a region of interest from the frame before processing to reduce the required processing requirements. This region is user-defined during the writing of the test case script.

*4.2.1 Optical Character Recognition.* The OCR method used in TVTAT is provided by the pytesseract package, a wrapper for the Tesseract-OCR Engine [25]. By default, Tesseract is set to use Brazilian Portuguese and English dictionaries for detection, but additional

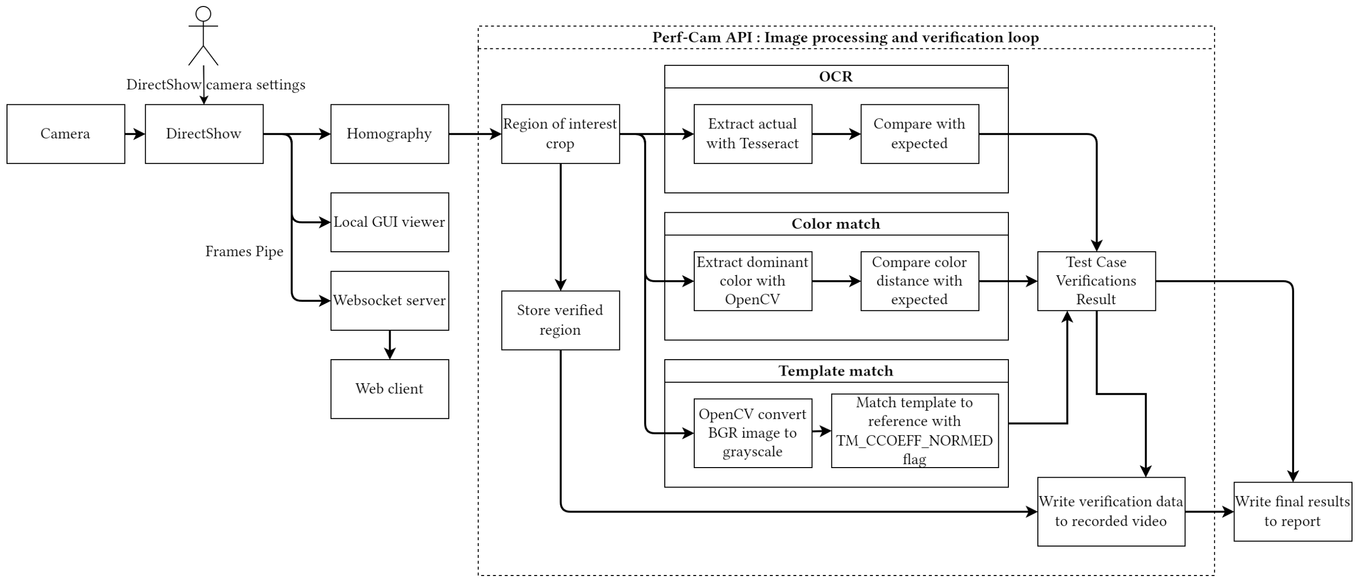



**Figure 5: Image processing overview**

languages can be installed. The string capture on the images by the OCR is triggered for specific frames, when is expected to find specific text on frames.

*4.2.2 Template match.* Asserts that a rectangular area captured by the tester is similar to the defined area in the frames of the captured video, in a given location. Before the template match function is executed, the passed frame region is converted from the default BGR (blue, green, red) color format to grayscale. A user-defined tolerance parameter is also used to adjust the similarity required between the expected and actual samples to pass the assertion. The template match method used is from the Python OpenCV library, with the normalized correlation coefficient operation [16].

*4.2.3 Color Verification.* Color assertions are made using color comparisons between the captured pixels and an RGB value defined in the test case script, with an additional tolerance value. The color comparison uses the CIE 2000 color difference formula, that returns a numeric value corresponding to the difference between the two given colors. Its implementation is provided by the colormath package [26], that attempts to account for differences in human color perception. The dominant colors from cropped from ROI images are calculated using the Kmeans method with the centers initialized in random positions. The Kmeans algorithm is performed to get clusters of colors, with the centroid of each cluster being the most relevant colors for that image [16].

## 4.3 DTVPlay Test Suite Metrics

The manual execution of the test cases using DTVPlay test suite will be compared to the proposed tool. A subset of the test suite will be manually executed, while the automated test suite will run completely. Time efficiency of both methods will be compared.

Identifying the test cases that are flaky [22] is crucial for the tester and the development team, because failed tests scenarios can lead to unwanted and unnecessary troubleshooting work. To help identify types of tests cases automated by the TVTAT that have unpredictable behavior, the Recursive Feature Elimination (RFE) algorithm was used with Gradient Boosting Classifier (GBC) [13, 17].

This algorithm tests a classification model with each of the features separately and then selects the features that lead to a more accurate model. The parameters chosen as inputs to the RFE were: the feature tested by the test case, the verification type, the execution time and whether the assertion passed or not, in other words, how flaky the test case is. The output of the algorithm is subjected to an ordinal ranking system based on how correlated each feature is with the success or failure of the test cases, a low ranking is associated with uncorrelated test case behavior.

To make this evaluation, two experiments were made, both with a loop of 30 evaluations, wherein executed with the RFE and GBC, the parameters of the GBC were set to randomly vary as follows [23]:

- learning rate - Any random float number from 0 to 0.1. The contribution of each tree decreases as learning rate increases. Between learning rate and number of estimators, there is a trade-off. Values must fall between the $[0.0, \infty]$ range.
- subsample - Any random float number bellow than 1.0. The percentage of samples that will be utilized to suit each base learner individually. Stochastic Gradient Boosting is the result if the value is less than 1.0. number of estimators is a parameter that is affected by subsample. A reduction in variance and an increase in bias result from selecting a subsample size of 1.0. Values must fall within the $[(0.0, 1.0]$ range.
- number of estimators - Any random item number from 10 to 300. The quantity of boosting steps to take. A large number

typically yields better performance because gradient boosting is fairly resistant to over-fitting. Values must fall within the $[1, \infty]$ range.

### 4.4 Application Benchmarking Approach

To validate the usefulness of TVTAT when interacting with heterogeneous devices, the same smart TV application was tested in different models and brands of smart TV devices. The application has similar GUI in all devices. TVTAT interacts in the same way as an end user, by using the basic input and outputs of the device. These are the display for output and the IR receiver for input. This enables TVTAT to test different devices without modifications to the test script.

This benchmarking provides data about the application's performance among environments. This enables the testers of the application to identify performance differences and trends of the tested application in different devices. To analyse these trends, simple linear regression is performed on the resulting data points.

## 5 EXPERIMENTS AND RESULTS

### 5.1 Testbed Setup

The Hardware used to perform the experiments is summarized below:

- High Definition Webcam connected at the host desktop;
- USB-Serial converter;
- Infrared (IR) sender device driven by microcontroller;
- Microcontroller to drive relay module;
- Transport Stream Modulator.

The TV is the device tested by TVTAT, and the Windows desktop is where TVTAT is installed and executed. The interactions between the TVTAT service and the TV happen through peripheral devices attached to the desktop: the IR sender, power relay, modulator and serial connection. On its usual configuration, the IR sender microcontroller and power relay are both Arduino modules. All verification procedures are performed using image processing, using the connected webcam as the source of frames.

### 5.2 DTVPlay Manual Test Suite

To establish a baseline to be compared to the results of the automated tests, a set of manual tests was done. A team of 3 testers was assembled, and a set of test cases was selected. This set was composed of 20 randomly selected test cases for each NCL functional area in Table 1, for a total 0f 220 test cases.

The testers operated the TV and made visual assertions as specified by the test scripts. The time taken for each test was recorded. In this manual test experiment, each test case took an average of 100 seconds to be completed.

### 5.3 DTVPlay Automated Test Suite

In this experiment, the tool is used to test a set of features offered by DTVPlay. The connected modulator device is used to transmit a digital television signal to the TV over a coaxial cable. These signal streams are stored in MPEG Transport Stream (TS) files inside the Windows file system, where each TS file corresponds to a DTVPlay test case. Embedded in these TS files are the test NCL scripts that should trigger behaviors visible in the TV's display, drawn over the digital TV transmission by DTVPlay.

**Table 1: NCL functional areas tested and number of assertions**

| Functional Area | Assert. | Functional Area | Assert. |
|---|---|---|---|
| Component | 2243 | Presentation Control | 157 |
| Presentation | 1208 | Linking | 130 |
| Interface | 1159 | Animation | 102 |
| Connectors | 572 | Transition | 80 |
| Layout | 341 | Structure | 60 |
| Reuse | 256 | — | — |

The testing of the features offered by NCL is split into 11 sets of test cases, where each evaluates one of the functional areas of NCL, as shown in Table 1. Additionally, some functional areas are currently not tested by the tool, these areas are Components Context, Persistent, Transitions Effects, Settings and Canvas. Assertions are classified based on the type of verification used, which can be Screenshot, QR Code, Template Match, Color Verify, or OCR.

TVTAT generates a report file for each test cycle, containing actual and expected results, execution time and the success status of the test, which may be unsuccessful due to failures or deficiencies of the tool and environment, or a DTVPlay implementation problem for this test case. The set of results assessed by this work corresponds to a total of 3049 executed tests cases. Each test case has a varying number of assertions, for a total of 7669. Of this total, 6308 assertions are classified by the functional area they test, according to Table 1. An additional 1361 assertions are not classified. For this result the test suite was executed three times: on the first execution, all of its test case were executed. On the second cycle only the failing test cases from the first cycle were executed again. So, the third cycle only the failing test cases from the second cycle were executed again.

*5.3.1 Result Analysis.* The boxplots of the features tested by the test suite were generated and are displayed on Figure 6. Those plots display the dynamic of each feature on the overall test case execution. It is visible that there are notable differences of execution time between successful and failed test cases, and the tendency that failed test cases almost certainly will present a longer execution time than successful test cases. This is due to the way the tool works, it generates failures when a timeout is reached without the expected event being found, but generates successes at the moment the event is found, skipping the timeout. For some cases, it can be seen that the samples correspond only to outliers, meanwhile other cases shows that there a tiny number of samples for compute the boxplot, that is the case for the Transition and Reuse function areas.

The experiment differs on the presence of the execution time as input for the RFE to process. The result is shown in Figure 7. The bar plots use the mean ranking for each feature with an error bar delimiting the 95% confidence interval for each rank. The result of the first Figure 7a and second Figure 7b demonstrate how the influence of execution time changes the perception of most important feature for classification, the Reuse functional area is shifted to the first place when execution time is not taken into account by the

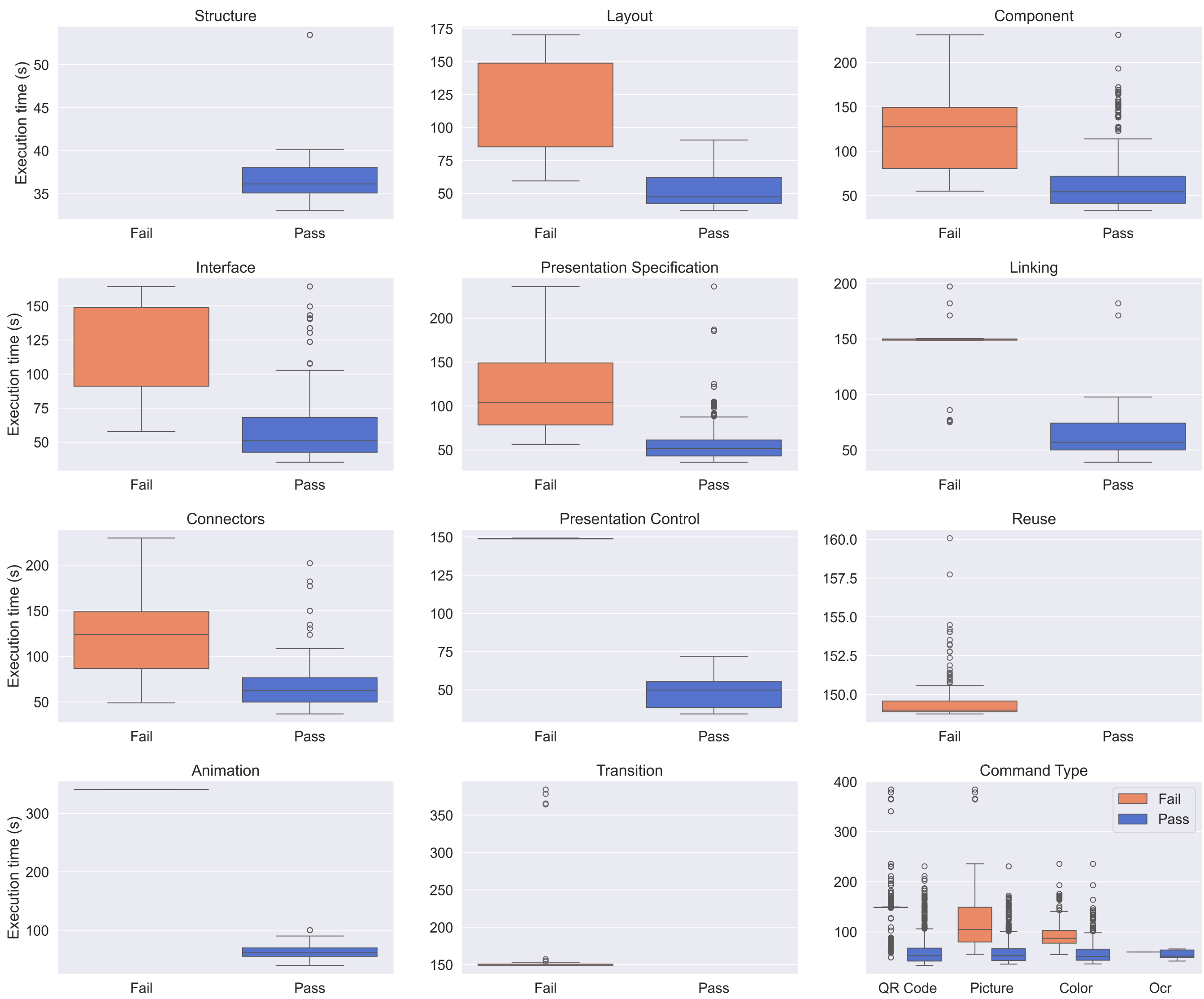



**Figure 6: Boxplot of first set features based on the result and execution time: Structure, Layout, Component and Interface; Presentation Specification, Linking, Connectors and Presentation Control; Reuse, Animation and Transition.**

GBC. The other features have an error bar varying from 1 to 3 in ranking.

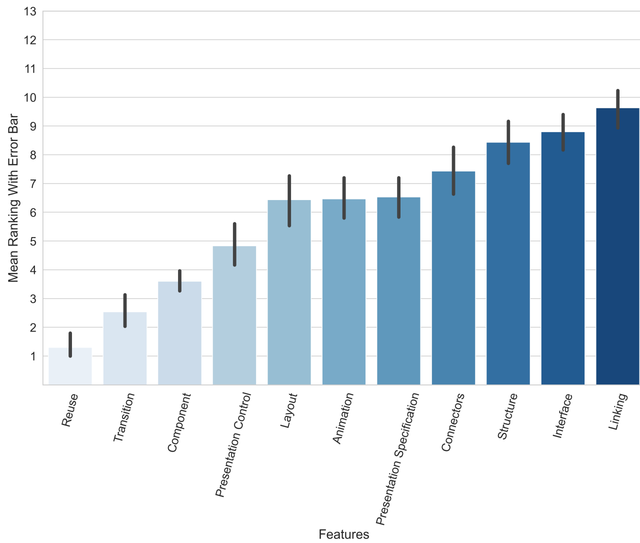
## 5.4 Application Performance Benchmarks

In this experiment, the tool was deployed to execute benchmark performance tests on different TV models. In this benchmark, the performance of a generic streaming application installed on all the TVs was measured. The common procedure consisted of defining start and finish points, measuring the time at each point and computing the time difference between the points, resulting in the execution time for the test case scenario. Based on the common procedure, two test case scenarios were defined. These two scenarios collect four different time measurements, $t1$, $t2$, $t3$, and $t4$. Diagrams of the steps in each scenario are shown in Figure 8 and Figure 9, and their executions are described in the following paragraphs.

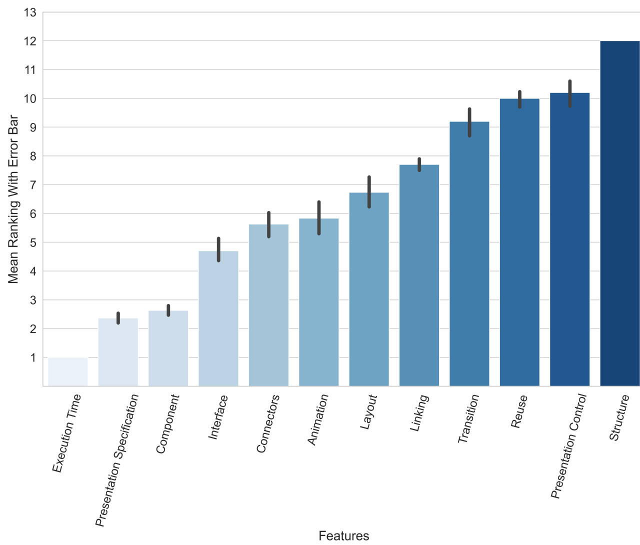
*5.4.1 Content Load Tests.* In the content load scenario, tool measures the time between when the user sees the application's home screen, **START**, and when any video content starts being displayed, **END**. There are two variations for this scenario: loading a video content $t1$, and loading live-streamed content $t2$.

The steps shown in the block diagram in Figure 8 are enumerated and detailed below:

(1) the application is at the home screen, the test tool records the time;
(2) IR commands are sent to navigate the home screen and open a video or a live stream;




(a) without execution time on the experiment.




(b) with execution time on the experiment.

**Figure 7: Mean ranking of the features using RFE.**

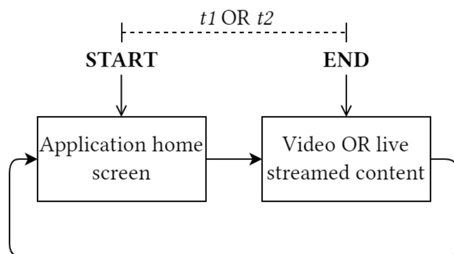



**Figure 8: Block diagram of the content load application performance benchmark scenario.**

(3) the test tool waits for the first chunk of the stream to load and be displayed, then records the time;
(4) back to 1.

*5.4.2 Cold Boot Tests.* In the cold boot scenario, the TV starts in a powered off state, with the relay where its power supply is connected toggled off. When the test tool sends the command to toggle on the relay, the first time measurement, **START**, is taken. The TV boots up due to the toggling of its power, while the test tool waits for a valid template match of the TV home screen. When the home screen is found, it navigates to the application and launches it, waiting again for a successful template match against a sample of the application home screen. When the application home screen is found. The second time measurement, **MIDDLE**, is made. The tool navigates to a video content, opens it and waits for the content to start being loaded and displayed on the screen. Then, it measures the time again, **END**. From this scenario, two different times are derived: the time between **START** and **MIDDLE**, measuring the time $t3$ between the TV being powered on and the application home screen being displayed, and the time between **START** and **END**, measuring the time $t4$ between the TV being powered on and a video content inside the application being displayed.

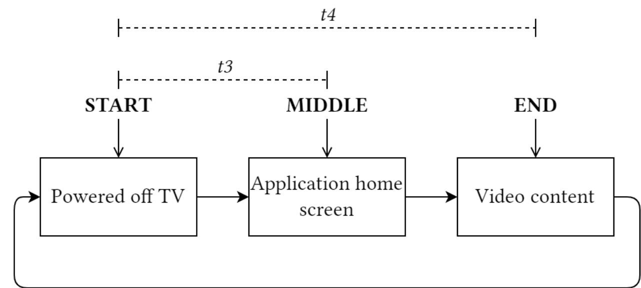



**Figure 9: Block diagram of the cold boot application performance benchmark scenario.**

The steps shown in the block diagram in Figure 9 are enumerated and detailed below:

(1) the TV has its power supply turned off;
(2) the power supply is turned on by toggling a relay, the test tool records the time;
(3) the test tool waits for the TV home screen to load;
(4) IR commands are sent to navigate the home screen and open the application;
(5) the test tool waits for the application's home screen to load, then records the time;
(6) the test tool navigates to open a video content;
(7) the test tool waits for the first chunk of the video to load and be displayed, then records the time;
(8) back to 1.

*5.4.3 Result Analysis.* All test scenario variations ran in a loop, 31 repetitions were performed for each one of them. Figure 10 shows the performance trends in the loading times for all scenarios. The confidence intervals, and error bar are shown in the plots for every experiment. This graph creates an average every 5 samples,

generates an error bar, and uses these sample averages as stakes that summarize the experiment up to that point.

Figure 10a and Figure 10b present the measured times for the video and live stream load test cases. In Figure 10a, certain models tend to lose performance over time, such as models A, E and D in the static content test, and only model D in the case of live content. One of the models has a tendency to improve performance, model C in Figure 10a. In the remaining cases, the models tend to have stable content availability times.

Figure 10d and Figure 10c present the measured times for the cold boot performance test cases. It is possible to see that certain models are considerably faster than others, and no trend is noticeable as the experiment continued.

## 6 CONCLUSION

In this work a test automation tool called TVTAT is presented, that provides tools and graphical interfaces that assist in creating smart TV test cases and organizing them into scenarios consisting of test case sequences. Its test script runner enables offloading the massive work of testing all aspects of DTVPlay with the automation tool.

From the obtained results, and the methodology presented, it is possible to identify the intrinsic characteristics and challenges present in smart TV testing using real-time image validation with cameras. The tests examined in this work include DTVPlay middleware and streaming applications. TVTAT speeds up the process of these tests, e.g., for a test with the manual execution time of 100 seconds, it is reduced to 86.23 seconds, with no human intervention, freeing many man-hours of repetitive tasks. The fact that the execution time of a test is correlated with its failure rate means that the real-time capture process could have a more accurate reactivity during tests in order to detect more quickly that a test scenario is failing, therefore opening up the possibility for future performance improvements.

The use of TVTAT smart TV automation is very promising, the tool avoids invasive control of the TV and is able to perform the exact same test on multiple smart TV models. Future work is expected to add tests for the HTML5 feature of DTVPlay, optimize the image preprocessing to have a faster and more efficient execution of each type of command, and improve the performance and stability of TVTAT.

## 7 ACKNOWLEDGEMENT

This work is the result of a PD&I project Autotest Contex Learning, realized by Sidia Institute of Science and Technology with partnership with Samsung Eletrônica da Amazônia Ltda, using the funding's of the federal law nº 8.387/1991, its dissemination and advertising being in accordance with the provisions of article 39 of Decree No. 10.521/2020.

## REFERENCES

[1] Bestoun S. Ahmed and Miroslav Bures. 2019. Testing of smart TV applications: Key ingredients, challenges and proposed solutions. In *Proceedings of the Future Technologies Conference (FTC) 2018: Volume 1*. Springer, 241–256.

[2] Bestoun S. Ahmed, Angelo Gargantini, and Miroslav Bures. 2020. An Automated Testing Framework For Smart TV apps Based on Model Separation. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 62–73.

[3] Gabriella Alves, Rennan Barbosa, Raoni Kulesza, and Guido L.S. Filho. 2015. A Software Testing Process for Ginga Products. In *Applications and Usability of Interactive TV: Third Iberoamerican Conference, jAUTI 2014, and Third Workshop on Interactive Digital TV, Held as Part of Webmedia 2014, João Pessoa, PB, Brazil, November 18-21, 2014. Revised Selected Papers 3*. Springer, 61–73.

[4] Gleb Avdeyenko and Teodor Narytnik. 2021. Hardware and Software Complex for Digital Television Signals Generation and Research. In *2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T)*. IEEE, 7–12.

[5] Mohammad Yusaf Azimi, Celal Cagin Elgun, Atil Firat, Ferhat Erata, and Cemal Yilmaz. 2023. AdapTV: A Model-Based Test Adaptation Approach for End-to-End User Interface Testing of Smart TVs. *IEEE Access* 11 (2023), 32095–32118.

[6] Gabriel Baum and Luiz Fernando G. Soares. 2012. Ginga middleware and digital TV in Latin America. *IT Professional* 14, 4 (2012), 59–61.

[7] Axel Bons, Beatriz Marín, Pekka Aho, and Tanja E.J. Vos. 2023. Scripted and scriptless GUI testing for web applications: An industrial case. *Information and Software Technology* 158 (2023), 107172.

[8] Rodrigo Braga, Volnei Klehm, Lauro Gama, Michael Mello, Taynara Paiva, Dina Nogueira, Ruddá Beltrão, Addison Lima, Leonardo Santana, Lois Nascimento, et al. 2019. NuGingaJS: a full portable ITU-T H. 761 Ginga middleware for DTV and IPTV. In *Proceedings of the 25th Brazillian Symposium on Multimedia and the Web*. 257–264.

[9] Miroslav Bures, Miroslav Macik, Bestoun S Ahmed, Vaclav Rechtberger, and Pavel Slavik. 2020. Testing the usability and accessibility of smart tv applications using an automated model-based approach. *IEEE transactions on consumer electronics* 66, 2 (2020), 134–143.

[10] Jing Cao, Xiaoqiang Liu, Hui Guo, Lizhi Cai, and Yun Hu. 2021. Test case generation for web application based on markov reward process. In *Journal of Physics: Conference Series*, Vol. 1792. IOP Publishing, 012039.

[11] Khasim Vali Dudekula, Hussain Syed, Mohamed Iqbal Mahaboob Basha, Sudhakar Ilango Swamykan, Purna Prakash Kasaraneni, Yellapragada Venkata Pavan Kumar, Aymen Flah, and Ahmad Taher Azar. 2023. Convolutional Neural Network-Based Personalized Program Recommendation System for Smart Television Users. *Sustainability* 15, 3 (2023), 2206.

[12] Bruno Farias, Ivo Machado, Eddie B de Lima Filho, Cláudio Pinheiro, Petrina Kimura, Leonardo Cordeiro, and Daniel Xavier. 2022. A Methodology for Emulating, Developing, and Testing the Middleware DTV Play in Personal Computers. In *2022 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 1–6.

[13] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[14] Fórum SBTVD [n. d.]. Normas Técnicas Sistema Brasileiro de TV Digital Terrestre. https://forumsbtvd.org.br/legislacao-e-normas-tecnicas/normas-tecnicas-da-tv-digital/portugues/. Accessed: 2023-05-25.

[15] Ginga website [n. d.]. Ginga Kernel Description. http://ginga.org.br/en.html. Accessed: 2023-05-25.

[16] Itseez. 2015. Open Source Computer Vision Library. https://github.com/itseez/opencv.

[17] Hyelynn Jeon and Sejong Oh. 2020. Hybrid-recursive feature elimination for efficient feature selection. *Applied Sciences* 10, 9 (2020), 3211.

[18] Mihajlo Katona, Ivan Kastelan, Vukota Pekovic, Nikola Teslic, and Tarkan Tekcan. 2011. Automatic black box testing of television systems on the final production line. *IEEE Transactions on Consumer Electronics* 57, 1 (2011), 224–231.

[19] Mumtaz Khan, Shah Khusro, Iftikhar Alam, Shaukat Ali, Inayat Khan, et al. 2022. Perspectives on the design, challenges, and evaluation of smart TV user interfaces. *Scientific Programming* 2022 (2022).

[20] Orlewilson B. Maia, Andre R. da Silva Conceição, Manoel J. de Souza Júnior, Fabrício Izumi, Eddie B. de Lima Filho, and Paulo Corrêa. 2022. A Real-Time Analyzer for Testing DTV Play. In *2022 IEEE International Conference on Consumer Electronics (ICCE)*. 01–05. https://doi.org/10.1109/ICCE53296.2022.9730583

[21] Dusica Marijan, Vladimir Zlokolica, Nikola Teslic, Vukota Pekovic, and Tarkan Tekcan. 2010. Automatic functional TV set failure detection system. *IEEE Transactions on Consumer Electronics* 56, 1 (2010), 125–133.

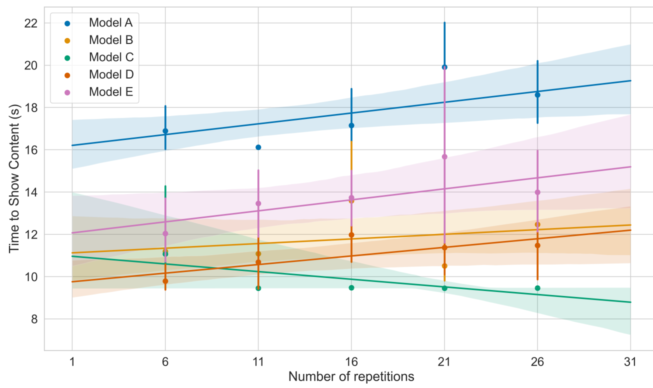
[22] Owain Parry, Gregory M. Kapfhammer, Michael Hilton, and Phil McMinn. 2021. A Survey of Flaky Tests. *ACM Trans. Softw. Eng. Methodol.* 31, 1, Article 17 (oct 2021), 74 pages. https://doi.org/10.1145/3476105

[23] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

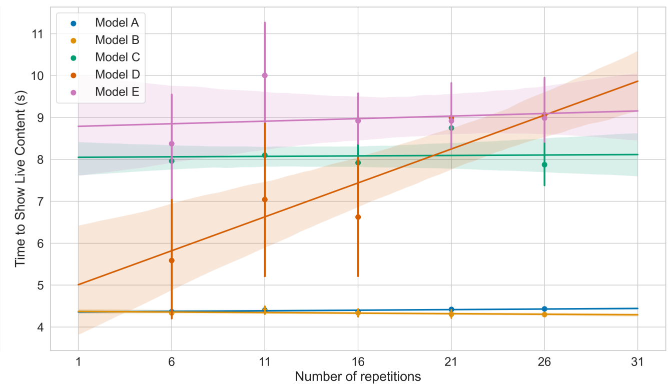
[24] Jônatas Rech, Vinícius Freitas, Bruno Farias, Eddie B de Lima Filho, Jeferson Costa, Ivo Machado, Xianpan Chen, Cláudio Pinheiro, and Daniel Xavier. 2021. A methodology for providing encrypted-content decoding in dtv play. In *2021 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 1–6.

[25] Ray Smith. 2007. An overview of the Tesseract OCR engine. In *Ninth international conference on document analysis and recognition (ICDAR 2007)*, Vol. 2. IEEE, 629–633.

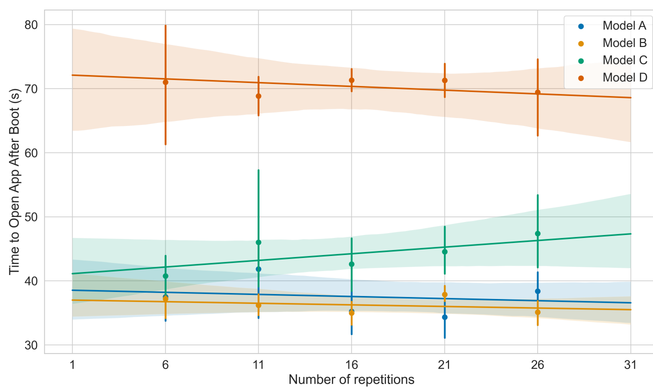
[26] Greg Taylor. 2017. python-colormath Documentation. (2017).



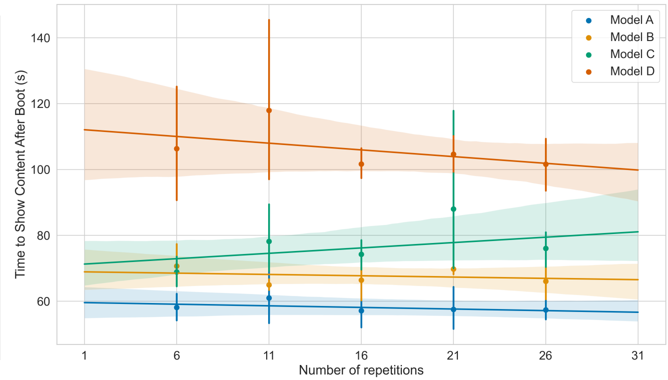

**(a) Show App content per repetition.**




**(b) Show App Live content per repetition.**




**(c) Launch App per repetition after Cold Booting with Relay.**




**(d) Launch App and show content per repetition after Cold Booting with Relay.**

**Figure 10: Binned Scatter plot of Time Performance Metric for Generic Streaming App with confidence interval and error bar.**

[27] Tiago H Trojahn, Juliano L Goncalves, Julio CB Mattos, Luciano V Agostini, and Leomar S Rosa. 2011. Tests and performance analysis of media processing implementations for the middleware of Brazilian Digital TV system using different scenarios. In *2011 Fifth FTRA International Conference on Multimedia and Ubiquitous Engineering*. IEEE, 95–100.

[28] Tom Yeh, Tsung-Hsiang Chang, and Robert C Miller. 2009. Sikuli: using GUI screenshots for search and automation. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*. 183–192.