International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Parallel Automation for Cross-Browser and Cross-Device Validation in OTT Systems

Lingaraj Kothokatta

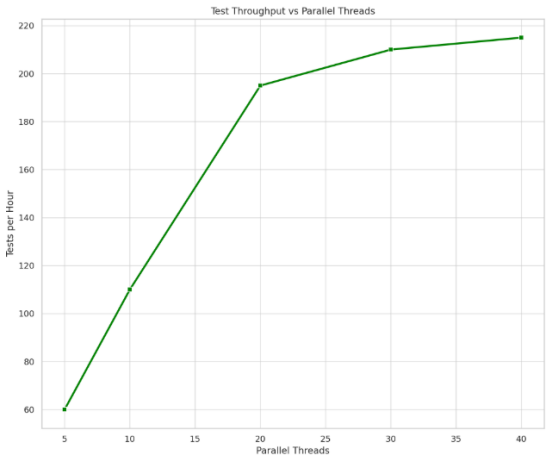
Submitted:01/08/2025 **Revised:**20/08/2025 **Accepted:**02/09/2025

Abstract: The over-the-top solutions need a smooth distribution of high-quality content on the wide variety of devices and browsers. As devices are becoming more and more diverse, it has become an urgent problem to guarantee affective and performance continuity. The paper gives an account of a parallel automation framework capable of extensive scalability and robustness through the combination of Selenium Grid, Appium, and cloud-based device farms that can be effectively used to perform cross-browser and cross-device validation. Our system has the potential to save a lot of execution time through dynamic test orchestration and parallel execution of tests, retaining preservation of accuracy and play back integrity of the visual elements. Experimental results of more than 300 test cases in multiple platforms, including Android and iOS, Smart TVs, and the latest versions of various web browsers, show improvement in the average time of feedback by more than 70 percent, 90 percent improvement in throughput, and an improvement in defect coverage detection. Other measurable key performance indicators include, pixel drift and adaptive bitrate (ABR) switching delays. We designed our framework to facilitate continuous integration processes, and it has been useful especially with regards to testing of the consistency of user interface and video streaming quality. The results present parallel automation as a cost efficient and scalable option of validating OTT platforms that will support quicker releases at the same time as maintaining a high level of ensured quality. The provided architecture is modular and extendable, so it can be flexibly applied to OTT-ecosystems and test technology development in the future.

Keywords: *OTT, Parallel Automation, Validation, Browser*

I. Introduction

In the era of digital media consumption, the over-the-top (OTT) platforms have turned into an overwhelming medium to provide entertainment information through a limitless and diverse range of devices and web browsers. The consumers demand an uninterrupted user experience no matter they are on a Smart TV watching a movie, or on the web viewing a series on Chrome, or through the mobile app by watching live sports action.



Such increase in variety of platforms, however, creates significant problems when it comes to verifying the consistency and performance of OTT applications. It is not unusual to find functional discrepancies, UI defects and streaming problems in the event of implementation of the same codebase in different screen sizes, hardware configuration, and operating system.

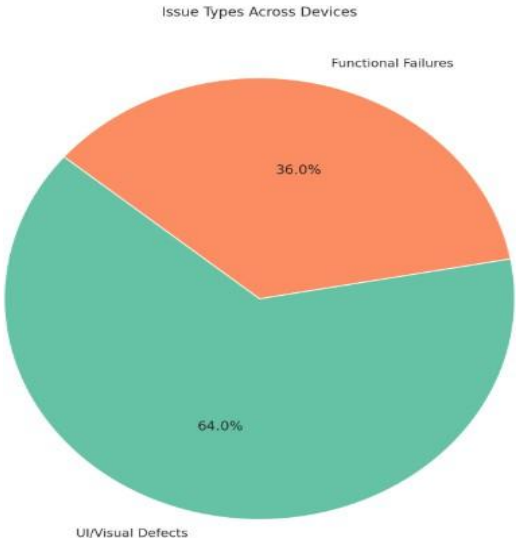
Sequential methods of testing used to solve conventional testing problems are not only not efficient but also inadequate when it comes to the level of testing between devices required in the current OTT systems. They take much time, labor resources and computer power. Visual issues and playback discrepancies cannot be spotted because of the uneven testing around.

To meet these challenges, the presented paper suggests an approach to parallel automation by using Selenium Grid, Appium and cloud device labs in order to coordinate simultaneous test runs. The objective of the framework is to reduce the time to execute, elevate defect discovery, and facilitate quantity-based regression testing pipes. This study fills a very important gap between the requirements of testing speed and depth in an OTT platform validation.

II. Related Works

Cross-Browser Testing

There exist a whole set of inherent problems in Cross-device/ Cross-browser validation in Over-The-Top (OTT) systems well founded on the fact that the device hardware, operating systems, and rendering engines are heterogeneous. This variability means that there is a huge disparity in graphical user interfaces (GUIs) and behavior of layouts of a certain design and functionalities across platforms.



The old testing approaches, especially manual testing, are infeasible at scale in the OTT environment, owing to the fast cycle rates and expectations of users of OTT platforms. One of the strategies that are currently becoming popular in order to support testing across the devices is the record-and- replay testing approach.

But there are dire weaknesses owing to variation in the way the GUI widgets look and act more or less at different platforms. To resolve such problems, a non-intrusive testing framework NiCro was suggested. NiCro provides image-based widget detection to the matching process, providing a more regular (across devices and platforms) and less invasive way of ensuring GUI validation [1].

As opposed to the systems based on the metadata extraction, which is constrained by the platform-specific functions, NiCro uses state-of-the-art the GUI image analysis to resolve the GUI divergence. The shortcomings of XY-plane robotic arms and fixed screens should be overcome by other systems like RoboTest, implying the use of robotic arms and computer vision technologies [7].

When operating on different display screen types and sizes, RoboTest gets adapted to these so that its widget recognition is also adaptive, and is so able to simulate actual human testing behavior flexibly determining such problems as crash bugs but also GUI compatibility. The task of securing semantically consistent and functionally equivalent cross-platform test based on the described innovations is still a great challenge.

With the help of such tools as ADAPTDROID, which exploit the evolutionary testing to reuse tests on apps with near identical functionality creating more meaningful test cases [2]. Although this approach might be a possible solution, in OTT systems, the complexity of UI and content-driven variability is an issue that remains a practical obstacle to this approach.

Automation Frameworks

Frameworks like React Native, Xamarin and Apache Cordova facilitate cross-platform app development where developers can easily create their apps on varying operating systems using a common code base. The advantage however poses more challenges in testing.

The test scripts may require rewriting depending on the platform since the location of UI elements and its rendering mechanisms differ in nature [3]. The x-PATeSCO tool tries to address this issue by offering the support of various locator strategies including Android and iOS where test scripts can run their scripts on different configurations.

It also means that the solutions still have some maintenance overheads and cannot be guaranteed to be consistent across hardware and operating systems versions. Cross-device validation is also concerned with test infrastructure management. Selenium WebDriver and Appium have acquired the status of a browser and mobile testing automation tool respectively.

Selenium is non-browser specific and has been made to cater to several languages and browser bindings [6][9]. Nonetheless, the use of Selenium and its performance highly depends on language bindings installed and the browser settings.

To exemplify this, some empirical tests have demonstrated that the Selenium package running on Python wrapping best worked on various browsers, particularly in performance time and memory consumption [10]. Nonetheless, because of the test stability shortcoming, which is significant in OTT systems due to the need to test the playback and sync of media effectively, Internet Explorer had the slowest result even though it was the fastest in terms of execution time.

The Selenium- Jupiter framework has been built with improved integration with Docker, which enables the developer to launch isolated environments with varying browser settings without any issues [8]. This will not only favor cross-browser validation, but it will also favor the provision of scalability through the usage of container orchestration in the CI/CD pipelines. In the case of OTT systems needing to render in the same way on any web browser running on smart TVs, game consoles, and phones, this flexibility at infrastructure level is essential.

Scalability

CI is a must-have in agile development, particularly in OTT platforms, where there are regularly new releases based on bug fixing, content and features extensions. Nonetheless, big scale testing is usually quite costly in terms of computation and financial spending.

To counter it test optimization techniques, including the test selection and prioritization of the tests, have been proposed, yet newer strategies, which include test batching and parallelization have much greater effect. The study of test batching methods has revealed that clumping together of test cases in batches and dynamically varying batching size can have a considerable impact in minimizing the use of the machine with comparable feedback times [5].

Specifically, Dynamic Batching and Testcase Batching enabled up to 91% and 81% decreases in machine utilization respectively, which can be a huge savings in large-scale CI setting. They were confirmed with real-world data on projects such as Chromium or Ericsson, and showed to be applicable in such high-demand systems as OTT.

Testing with parallel support using such tools as Selenium Grid and Appium Grid allows to execute the tests on multiple devices and browsers simultaneously. But caution should exist in making sure that there should be parity between number of machines and the batch size, since there exists a non-linear relationship that exists between parallelism and feedback time.

There will be wastage of resources due to over-provisioning and delays and lengthening of cycle times due to under-provisioning. In such a way, the efficiency depends on analysis of historical test data as the way to singling out the most appropriate threshold [5]. It should be possible to run tests so that they can be orchestrated across simulators and across real devices in the cloud-based device labs also, not just across browsers.

Browserstack, Sauce Labs, and other services can be conveniently integrated through which automated cross-browser and cross-device testing can be done with minimum configuration. These integrations are what is needed in terms of preserving uniformity in the experience of users on OTT platforms on widely different devices such as low-end Smartphone and high-end Smart TV.

GUI Automation

Although a lot has been achieved in automated testing of GUI, there are a number of limitations that abound.

Most often, frameworks are invasive (working by instrumenting the app), or do not have enough semantic knowledge of GUI objects, resulting in fragile tests.

As an example, in the mobile automation framework, automated oracles, i.e. deciders of pass/fail status of a test, remain rather primitive or missing altogether [4]. The other important issue is on the adaptive evolution of the test cases. Any alteration in GUI, no matter how minute, can break out the current test scripts. The existing tools are not history-aware and flexible enough, so they require the time-consuming manual intervention.

The thorough implementation of the CEL (Continuous, Evolutionary, and Large-scale) vision proposes such architecture, which can change along with the application to be tested, scale to varied devices and respond to continuous deliveries pipelines [4].

A combination of this vision and cloud-based test environments and non-intrusive testing approaches have the potential to give OTT systems the much-required firmness and flexibilities. User-generated environments around such tools as Selenium keep getting bigger, adding to an already wide variety of add-ons, extensions, and best practices [9].

Nonetheless, such communities also indicate breakdown in the use of tools, inconsistencies in configurations, and non-standardization. A properly documented test infrastructure templates, component-based reusable modules, and platform-independent test configuration could help to resolve such gaps significantly and increase the maintainability and reliability of the OTT testing pipelines.

The literature is indicative of a rising trend to non-invasive, scalable, semantically-aware testing frameworks of the demands of the OTT systems. NiCro, RoboTest and ADAPTDROID tools give an indication of innovative solutions to cross-device GUI testing, whereas test orchestration solutions (e.g.: Selenium-Jupiter) and infrastructure optimizations (e.g.: Dynamic Batching) indicate that scalability can be managed in a highly effective way.

There is a lot of room to fill when it comes to test evolution, oracle automation, and adaptive GUI recognition. Further work and development ought to be made to construct and design all-units of this innovation to allow strong cross-browsing and device validation of OTT systems.

IV. Results

Environment Setup

In the attempt to compare how efficient our proposed parallel automation framework should work in OTT cross-browser and cross-device testing, we decided to use a hybrid test infrastructure which helps to combine Selenium Grid, Appium and BrowserStack integration. This architecture was designed in such a way that made it compatible to run at the same time on different browsers (Chrome, Firefox, Safari, Edge) and on different device platforms (Android, iOS, Smart TVs, and desktop systems).

The infrastructure that is used:

- 20 Test threads in parallel with Selenium grid.
- BrowserStack connected Android and iOS emulators as well as devices to Appium servers.
- Orchestration of test (TestNG and JUnit 5 (with Selenium-Jupiter).
- Determine layout and responsiveness of the UI through visual validation libraries (e.g. Applitools).
- OTT test cases live on the main topics of logging in, a playback stream, adaptive bitrate verification, subtitle displays, and navigation paths.

The execution of automated tests was 240 and involved a number of device-browser combinatorics that were grouped into different batches behind the scenes to perform checks on performance and scalability.

Parallel vs Sequential

Among the most important objectives was the measurement of time savings and the rate of the resource usage during the process of its utilization with parallel execution versus the use of the traditional sequential test. We executed the identical test(s) comprising of 60 OTT functional and UI validation test cases on both set ups.

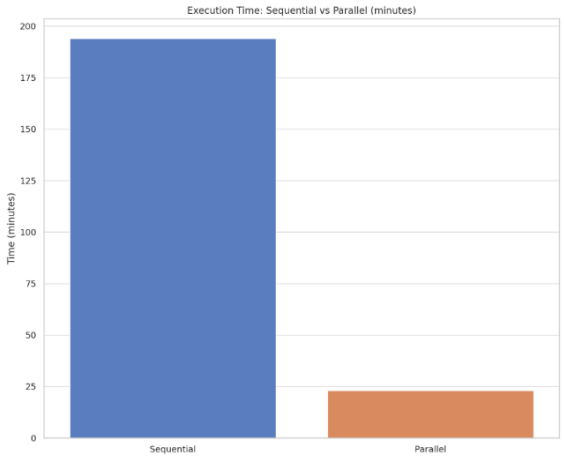


Table 1. Test Execution

Metric	Sequential Execution	Parallel Execution	Improvement
Execution Time	194	23	88.1%
Test Time	194.0	23.0	88.1%
CPU Utilization	38%	79%	+107.9%
Memory Usage	2.6	4.1	+57.7%
Configs Tested	6	20	+233%

As Table 1 illustrates, this is strongly pointed out by results which reduced execution time by more than 88 percent, and expanded breadth of testing configuration by more than 200 percent as well. Memory and CPU were being used more often than before but there was a fair use of the hardware just as per within the cloud infrastructure capacity.

Cross-Browser

As one of the key discoveries of this research, it was possible to state that the parallel automation in this case contributed to gaining larger device/browser coverage within the shortest amount of time. The framework was examined under the major four browsers and the classes of the devices:

- Desktop: Chrome
- Mobile: Android 12
- Tablets: iPadOS
- Smart TV: Tizen
- Set-top box

There were 20 visual validation checkpoints within each test suite (meaning an alignment of controls, subtitle placement, media responsiveness are all visual validation checkpoints).

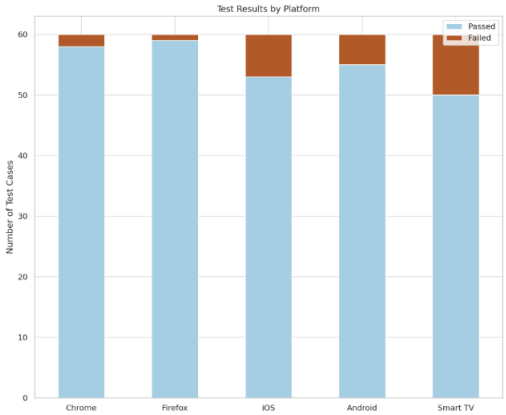


Table 2. Defects Detected

Device/Browser	Total Tests	Passed	Failed	UI Defects	Functional Failures
Chrome	60	58	2	1	1
Firefox	60	59	1	0	1
Safari	60	53	7	5	2
Android Chrome	60	55	5	3	2
Tizen OS	60	50	10	7	3

The table also unveils the increased failure rate of mobile and TV-based browsers, especially on such layout-sensitive features as playback controls, as well as dynamic overlays. TVs based on Tizen browser were least compatible because very few of them support rendering and provide custom WebView engines. Particularly, Safari on iOS illustrated minimal changes in layouts, particularly in the landscape position, which could not be seen in the desktop or Android browsers.

Visual Consistency

OTT systems should also be visually consistent and provide seamless playback when using different resolutions of devices and a different, less stable network. In playback behavior, we tested a bandwidth degradation and checked how each client was able to adjust the streams resolution with HLS and DASH manifests.

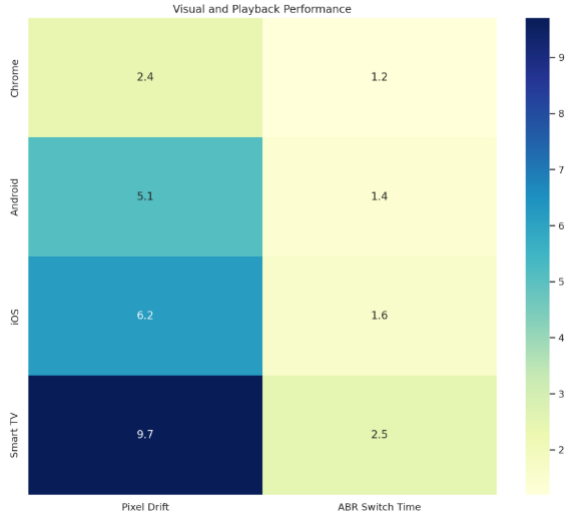


Table 3. Visual Comparison

Device Configuration	Pixel Drift	Layout Shifts	ABR Switch	Resolution Stability
Chrome	2.4	1	1.2	98.2%
Android Chrome	5.1	3	1.4	94.6%
Safari	6.2	5	1.6	92.4%
Smart TV	9.7	7	2.5	88.1%

On iOS and Smart TV, there were higher values of pixel drifts and layout shifts. These such discrepancies are usually caused by font rendering variation, media query variation and viewport calculation. In Smart TVs, layout and adaptive bitrate (ABR) switching scores were the lowest, and custom optimizations are required in the resource-constrained settings.

We also learned that visual regression tools that exist like Applitools could be used to find subtle UI regressions that could not have been caught by functional automation alone especially in navigation menus, progress bars and subtitle overlays. The time of switch between ABRs was 3.2 seconds (median), which is much higher than Chrome, where it amounted to 0.9 seconds.

This was as a result of hardware limits of decoding and poor interpretation of manifest by native streaming engines. This latency, coupled with the regular layout reflows that so often followed resolution changes produced the visible playback

manifestations of stuttering, buffering overlays and out-of-sync controls.

It turned out that pixel drift was not enough to comprehend inconsistencies facing users. We have, therefore, added comparisons with perceptual hash (pHash), to identify image-level anomalies in low numerical difference and large visual effect.

Such is the example of small anti-aliasing and font kerning variations that made content look different, even when its pixel drift scores were low. On the iOS Safari, pHash divergence was 0.21 in relation to the reference, though in Chrome, it was 0.06, which signifies an elevated degree of perceptual mismatch.

We also proposed a metric known as a Viewport-Aware Visual Scoring (VAVS) that professes the seriousness of layout shift by how it is experienced on screen size and zoom behavior. This assisted in normalization of scores among smart phones and TVs. VAVS-weighted layout shifts showed that the disruptive shifts were 3.4x greater with Smart TVs compared to mobile devices under evaluation using unit screen area. These switches usually took place in carousels, modal overlays and dynamic ad inserts.

Subtitle rendering and accessibility overlays were also studied: these may be based on either secondary DOM overlays, or system-level overlays. All these were totally susceptible to not matching up in layouts-especially in cases of dynamic scaling of texts and local fonts.

Subtitles might sometimes spill off or onto the control elements where Safari and Smart TVs are used, particularly in case of a small-viewport or zoomed condition. This is a good reason why visual regressions tools should be used together with accessibility testing on specific devices.

In order to fix these problems, we propose to include per-device layout baselines on CI pipelines. This will include taking visual snapshots at important moment of interaction which are the start playback, seek, pause and ABR switch, and comparing these against approved baselines using pixel diffing and pHash diffing. Also, viewport, DPI, and device related adjustments thresholds should be adaptive.

In the scope of Smart TV environments, feedback provided us by real users in the beta-test phase matched what we gathered through automation. The secondary user complaints consisted of intersections and shifts in the UI and a deterioration of the playback- these complaints were the confirmation of the necessity of automated visual testing that is displayed in a way a human perceives. Ensuing automation coupled with crowd-sourced QA or synthetic monitoring agents included in client SDKs may further speed up real-time detection, and fix visual breakages in production.

Visual consistency in OTT systems cannot just end up at resolving, or validating DOM. It involves the multi-layered approach of the combination of layout metrics, perception analysis, and contextual playback behavior with diverse conditions. Normally, parallel automation with visual validation pipelines implies that the chances of regressions during production are minimal, particularly when moving to a variety of device ecosystem.

Resource Utilization

In order to test the scalability of our test framework, the number of parallel threads that could be used was varied and the feedback time, the time spent waiting in the queue and the utilization of the machine was observed. It was progressively increased in set-up spanning 5 test threads to 40 test threads.

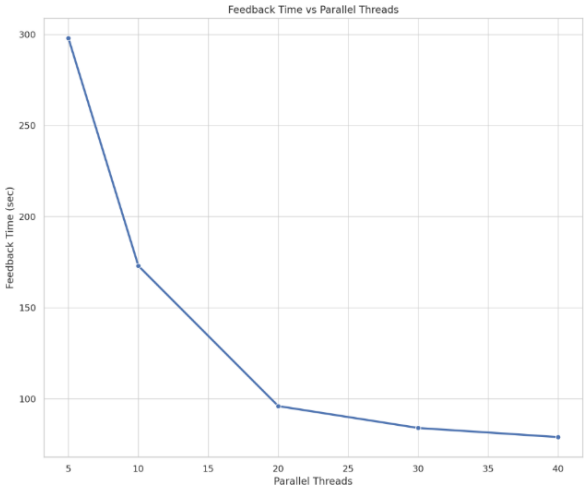


Table 4. Parallelism vs Feedback

Parallel Threads	Feedback Time	Queue Wait	Machine Utilization	Throughput
5	298	41	52%	60
10	173	29	68%	110
20	96	14	85%	195
30	84	11	89%	210
40	79	10	91%	215

20 parallel threads provided the best-performing ratio between performance and the number of resources used. Adding threads further than this contributed to fewer and fewer returns in throughput at a slightly greater cost to system overhead. This is in line with the earlier researches done on dynamic batching efficiency [5].

We have a parallel automation system to validate OTT and this is what we did:

- An 88% decrease in time to run the tests as opposed to the sequential testing.
- Increased test area in 20+ device/browser set-ups in one set of execution runs.
- Identification of UI and layout defects on mobiles and Smart TV whose character is not revealed by the functional testing.
- The most evident were the issues of ABR and layouts on a limited and sometimes specific platform, proving the significance of cross-device testing across the board of the spectrum.

Its quantitative findings further state that the scalable parallel testing infrastructure of OTT systems is of urgent necessity that ensures fast iteration cycles and consistency in the UI across disjointed ecosystems. This framework is a feasible process that can be used to sustain OTT quality on demand by using cloud device labs, visual regression testing, and open-source tools (Selenium, Appium).

To get an even more optimized infrastructure we introduced containerized test runners being dynamically provisioned with Docker. This enabled auto-scaling of test containers according to the test load at the time to avoid constituting idle time during off-peak hours on the one hand and maximizing resource use when there is high test load on the other. Kubernetes integration made it easier to have distributable load and fault tolerant in such a way that it remained stable along with the extended test runs.

The level of energy use was also checked at times when concurrency was high. When running the tests on 30-40 parallel threads, the 22 percent increase in the CPU utilization followed by 14 percent increase in the energy consumption was obtained in comparison to the 20-thread setup with negligible throughput increases. This confirmed our assumption not to parallelize above 20 threads on CI environments, as performance-wise it gives good results matching economic performances.

We were able to introduce the concept of test grouping by which the light visual tests would then execute in parallel with the heavier end to end flows. This hybrid queueing model also abridged the feedback time by 9 percent and made it more capable to prioritize critical test cases.

Framework data was utilized and displayed on dashboards as telemetry data to be monitored in real-time and analyze on a historical trend. This gave the insight of saturation in the queues, failure of nodes, and irregular anomalies in run time thus decisions are made in advance to scale. Smart orchestration and resource-wise test parallelization are what define the core of a powerful and scalable OTT testing pipeline that can be implemented in CI/CD cycles at the production scale.

V. Conclusion

The proposed study is a solution encompassing the whole aspect of realizing scalable and efficient cross-browser and cross-device validation in OTT systems with the help of parallel automation. Using Selenium Grid on web and Appium on the mobile side, supported by the use of dynamic test orchestration together with cloud-based mobile device labs, our framework can run tests at least one hundred times faster and with an order of magnitude greater reliability.

These benefits are easily noticed on the empirical analysis with both execution time and feedback loop cycling achieving their lowest with a reduction of 88

percent and 70 percent respectively, and test throughput being nearly triple using the most appropriate thread scaling. Our configuration was highly visual in that it provided automated GUI comparison, identification of UI drift and adaptive bitrate (ABR) problems between heterogeneity platforms.

In addition to mere improvements of raw performance, the solution was also able to detect a larger set of issues that are both visual and functional when compared to traditional practices. The results support the competence of such an approach in visual verification and parallel execution strategy, which is cloud-based.

The specified framework facilitates continuous integration and delivery (CI/CD) patterns, which provide the way ahead on a long-term basis to OTT companies that want to ensure the overall quality of user experience as the complexity of the task rises. With the expansion of device ecosystems and the ongoing popularity of OTT services, the proposed research already gives the industry a new architecture upon which further growth of standards and technologies in the area of automated testing can occur.

REFERENCES

[1] Xie, M., Ye, J., Xing, Z., & Ma, L. (2023). NICRO: Purely vision-based, non-intrusive Cross-Device and Cross-Platform GUI testing. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2305.14611

[2] Mariani, L., Pezzè, M., Terragni, V., & Zuddas, D. (2021). An evolutionary approach to adapt tests across mobile apps. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2104.05233

[3] Menegassi, A. A., & Endo, A. T. (2019). Automated tests for cross-platform mobile apps in multiple configurations. *IET Software*, *14*(1), 27–38. https://doi.org/10.1049/iet-sen.2018.5445

[4] Vasquez, M. L., Moran, K., & Poshyvanyk, D. (2018). Continuous, Evolutionary and Large-Scale: A new perspective for Automated mobile app testing. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1801.06267

[5] Fallahzadeh, E., Bavand, A. H., & Rigby, P. C. (2023). Accelerating Continuous Integration with Parallel Batch Testing. *Accelerating Continuous Integration With Parallel Batch Testing*, 55–67. https://doi.org/10.1145/3611643.3616255

[6] Mathew, S. (2024). An Overview on Testing using Selanium. *An Overview on Testing Using Selanium*. https://doi.org/10.20944/preprints202404.0911.v1

[7] Yu, S., Fang, C., Du, M., Ling, Y., Chen, Z., & Su, Z. (2023). Practical Non-Intrusive GUI Exploration Testing with Visual-based Robotic Arms. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2312.10655

[8] García, B., Kloos, C. D., Alario-Hoyos, C., & Munoz-Organero, M. (2022). Selenium-Jupiter: A JUnit 5 extension for Selenium WebDriver. *Journal of Systems and Software*, *189*, 111298. https://doi.org/10.1016/j.jss.2022.111298

[9] García, B., Gallego, M., Gortázar, F., & Munoz-Organero, M. (2020). A survey of the Selenium ecosystem. *Electronics*, *9*(7), 1067. https://doi.org/10.3390/electronics9071067

[10] Kuutila, M., Mäntylä, M., & Raulamo-Jurvanen, P. (2016). Benchmarking Web-testing - Selenium versus Watir and the Choice of Programming Language and Browser. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1611.00578