International Journal of Research and Development in Applied Science and Engineering (IJRDASE)
ISSN: 2454-6844

Automation Framework Design and Performance Evaluation for Android App GA Analytics

Aditi Rastogi, Dr. Rohitashwa Pandey
Department of Computer Science & Engineering
Bansal Institute of Engineering and Technology, Lucknow, India
aditirasph@gmail.com, rohitashwapandey1982@gmail.com

Abstract: Mobile applications play a critical role in modern digital ecosystems, where both analytics accuracy and application performance significantly influence business decisions and user experience. However, existing testing approaches often treat analytics validation and performance monitoring as separate processes, leading to inefficiencies and unreliable results. This paper proposes a unified Android automation framework designed to validate Google Analytics (GA) events and monitor runtime performance metrics in a deterministic and scalable manner. The framework integrates Appium for UI automation, Cucumber for behavior-driven test definition, and Android Debug Bridge (ADB) for system-level data extraction. Unlike traditional methods that rely on network interception, the proposed system utilizes Android Logcat as a single source of truth for capturing and validating analytics events. In addition, the framework automates the collection and evaluation of key performance indicators such as CPU usage, memory consumption, startup time, UI rendering performance, battery usage, and network data consumption. The implementation on a real-world Android application demonstrates improved testing efficiency, reduced debugging time, and enhanced reliability of analytics data. The results indicate that the proposed framework effectively bridges the gap between analytics validation and performance testing, providing a comprehensive solution for modern mobile application quality assurance.

Keywords: Android Automation Framework, Google Analytics (GA), Mobile Application Testing, Analytics Validation.

1. Introduction:

Mobile applications have become an integral part of modern digital ecosystems, serving millions of users across diverse domains such as media, finance, e-commerce, and entertainment. With increasing competition and user expectations, the success of a mobile application is no longer determined solely by feature availability, but by the accuracy of analytics data and the consistency of application performance. Analytics events guide critical business decisions, marketing strategies, content prioritization, and user engagement analysis, while performance stability directly influences user retention, satisfaction, and brand credibility. In Android applications, Google Analytics (GA) is widely adopted to track user interactions, navigation flows, feature usage, and behavioural patterns. These analytics events form the backbone of data-driven decision-making. However, inaccuracies in analytics events—such as missing events, incorrect parameters, duplicate triggers, or delayed firing—can lead to misleading insights and flawed conclusions. Traditionally, analytics validation is performed manually using tools like Firebase Debug View, log inspection, or network proxies. Such approaches are often slow, environment-dependent, and unsuitable for large-scale regression testing. At the same time, mobile application performance has emerged as a critical quality attribute. Metrics such as application start-up time, CPU utilization, memory consumption, UI rendering smoothness, and battery usage significantly impact the end-user experience. Performance regressions introduced during feature development or SDK upgrades often go unnoticed until reported by users in production. Manual performance testing is time-consuming and inconsistent, while profiling tools are typically used only during development and not integrated into automated test pipelines. The absence of a unified and automated solution that validates both analytics correctness and runtime performance behaviour create a major gap in mobile quality assurance practices. Most existing frameworks focus either on UI automation or performance profiling in isolation, while analytics validation remains largely manual and unreliable. This gap becomes more evident in large applications with frequent releases, multiple user journeys, and diverse device configurations.

This work aims to bridge this gap by proposing an end-to-end Android automation framework that validates Google Analytics events and runtime performance metrics in a deterministic, scalable, and developer-friendly manner. The framework leverages Appium for realistic UI interaction, Cucumber (BDD) for readable and maintainable test scenarios, and Android Debug Bridge (ADB) for low-level system access. Instead of intercepting network traffic, the framework directly reads analytics payloads from Android Logcat, treating it as the single source of truth for GA validation. This approach improves reliability, reduces execution time, and eliminates dependency on external tools or network conditions. In addition to analytics validation, the framework automates the collection and validation of key performance indicators such as cold, warm, and hot start behaviour, CPU usage, Janky frames, Java heap memory, native heap memory, battery consumption, and network data usage. By comparing baseline metrics with post-interaction measurements, the framework enables early detection of performance regressions and helps maintain consistent application behaviour across releases. The CNBC Android application is selected as the case study for this

research due to its real-world complexity, high user traffic, and analytics-intensive workflows. The implementation and results demonstrate how the proposed framework enhances testing efficiency, reduces debugging time, and increases confidence in both analytics data and application performance. This paper establishes the foundation for the thesis by outlining the motivation, relevance, and problem context. The subsequent chapters explore related work, system architecture, implementation details, experimental results, and future enhancements of the proposed automation framework

2. Literature Review:

Recent advancements in mobile applications, software reliability, and intelligent monitoring systems have significantly improved failure detection, testing efficiency, and performance optimization. Several studies have focused on leveraging machine learning, data analytics, and automation frameworks to enhance system robustness and user experience. Kumar et al. [1] proposed an LSTM-based approach for early system failure detection through log analysis, demonstrating how deep learning models can effectively identify anomalies in system behavior. Similarly, Kim et al. [10] introduced an automated abnormal log detection technique that analyzes historical logs to provide debugging insights, emphasizing the importance of log mining in proactive system maintenance. Earlier work by Lo et al. [6] utilized discriminative pattern mining to classify software behaviors, forming a foundational approach for modern anomaly detection systems.

Security and vulnerability detection in mobile applications have also been widely explored. Mendoza and Gu [2] highlighted inconsistencies between web APIs and mobile applications, exposing potential security vulnerabilities. Lima et al. [8] proposed a comprehensive security monitoring framework for mobile devices, focusing on real-time threat detection and system protection. These studies underline the growing need for robust security mechanisms in increasingly complex mobile ecosystems.

Testing and quality assurance of mobile applications have seen significant improvements through automation and user-centric approaches. Melnyk et al. [3] introduced dynamic test case prioritization based on real user behavior data, improving testing efficiency and relevance. Baek and Bae [11] proposed automated model-based GUI testing techniques, while Subramanian et al. [12] developed a behavior-driven test automation framework. Additionally, Kothokatta [13] emphasized parallel automation for cross-browser and cross-device validation, which is critical in modern multi-platform environments. Baktha [14] evaluated various Android testing frameworks within Agile and DevOps practices, highlighting their performance and adaptability.

Performance optimization and system monitoring have also been key research areas. Kovacs [4] presented a data-driven framework for optimizing retail application performance using advanced monitoring metrics and anomaly detection. Kanagarathinam et al. [16] proposed an application prioritization engine to enhance real-time performance in smartphones, ensuring efficient resource allocation. Oliveira et al. [5] provided a comprehensive survey on provenance analytics, which aids in understanding workflow execution and improving reproducibility in computational experiments.

In the context of system reliability and energy efficiency, Banerjee et al. [7] addressed debugging energy-related failures in mobile applications, highlighting the trade-offs between performance and power consumption. Kumar [9] explored trustworthy binary classification in dynamic systems under uncertainty, contributing to reliable decision-making models in uncertain environments.

Furthermore, the integration of DevOps practices has accelerated mobile application development and deployment. Ruiz et al. [15] discussed the role of CI/CD pipelines in enabling faster and more reliable software releases, ensuring continuous integration and delivery in dynamic development environments

3. Methodology:

The Android App GA Analytics & Performance Automation Framework is designed as an end-to-end, unified system for validating Google Analytics (GA) events and runtime performance metrics in Android applications. The primary objective of the framework is to provide a single source of truth for analytics correctness and performance stability by combining UI automation, system-level diagnostics, and structured validation mechanisms [17, 18].

(a) System Overview: At a high level, the framework follows a layered architecture with clear separation of concerns and high maintainability. The execution begins with behaviour-driven test definitions using Cucumber, where analytics expectations and performance thresholds are expressed in a business-readable format. User interactions are executed through Appium-based UI automation, which simulates real user behaviour such as navigation, scrolling, clicks, and media playback. These interactions act as triggers for both analytics event generation and performance metric capture. The use of the Page Object Model (POM) ensures that UI logic is reusable, scalable, and resilient to UI changes.

For analytics validation, the framework adopts a Logcat-driven approach instead of traditional network interception. Google Analytics events generated by the application are directly extracted from Android Logcat, treating it as the authoritative source of analytics data. A polling-based validation mechanism waits for expected events within a configurable timeout window, parses raw log entries into structured event objects, and validates event names and parameters. In case of failure, all observed analytics events are automatically dumped, enabling instant debugging and complete visibility into analytics behaviour.

In parallel, the framework includes a performance automation module that captures key runtime metrics using Android's native diagnostic tools accessed via ADB. Metrics such as cold, warm, and hot start behaviour, CPU usage, janky frames, Java heap memory, native heap memory, battery consumption, and network data usage are collected in a deterministic manner. Baseline metrics are compared with post-interaction values and validated against predefined thresholds to detect regressions

early in the development lifecycle. Device and application state management is handled through ADB commands, ensuring consistent execution conditions across test runs. This includes clearing application data, force-stopping processes, managing Logcat lifecycle, and extracting device and OS metadata.

(b) Technology Stack: The proposed framework for automated GA validation and Android performance monitoring relies on a carefully selected set of technologies that ensure reliability, scalability, and maintainability. Each component of the stack serves a specific purpose, working together to provide a seamless and automated solution for data collection, verification, and performance analysis.

Java serves as the primary programming language for the framework. Its platform independence, strong object-oriented capabilities, and extensive ecosystem make it ideal for building scalable automation scripts. Java enables seamless integration with Appium, TestNG, and other dependencies, forming the backbone of the automation logic.

Appium is the chosen mobile automation tool, allowing the framework to interact with Android UI elements across multiple devices and OS versions. Its support for both native and hybrid apps ensures that the framework can validate user interactions, GA events, and app navigation flows consistently. Using Appium, tests are executed in real device environments, providing realistic insights into user behavior and app performance.

Cucumber supports Behavior-Driven Development (BDD) using Gherkin syntax. This allows test scenarios to be written in plain English, bridging the gap between technical and non-technical stakeholders. By defining test behavior in a human-readable format, the framework facilitates collaboration between QA engineers, developers, and product teams, ensuring that GA tracking and performance validation meet business requirements.

TestNG is integrated to manage test execution, grouping, and reporting. Its robust features for parallel execution, dependency management, and detailed reporting enhance the efficiency of the test cycles. Combined with Maven, TestNG ensures smooth build management, dependency resolution, and consistent execution of tests across environments.

ADB (Android Debug Bridge) provides the capability to interact directly with Android devices and emulators. It allows the framework to capture runtime performance metrics such as CPU usage, memory consumption, frame rendering times, and app startup times. ADB commands also enable log extraction and manipulation, which are critical for verifying GA events and analyzing app behavior under different scenarios.

Android Logcat is employed for real-time logging of app events, system messages, and analytics data. Captured logs serve as a single source of truth for verifying GA events, detecting discrepancies, and monitoring the application's performance. By automating log analysis, the framework eliminates the need for manual inspection, reducing errors and increasing testing efficiency.

Maven is used for project and dependency management. It streamlines the build process, ensuring that all required libraries and tools are correctly configured and versioned. Maven also supports integration with CI/CD pipelines, allowing automated test execution as part of the deployment workflow.

(c) Architecture Design

The framework is designed using a **layered architecture**, ensuring modularity, maintainability, and scalability. This architecture separates concerns into distinct layers, each responsible for a specific aspect of automation, analytics validation, or performance monitoring. The layered design enables seamless integration of Behavior-Driven Development (BDD), UI interactions, analytics validation, and runtime performance analysis, while keeping the framework organized and extensible.

1. BDD Layer: Topmost layer, implements **Behavior-Driven Development (BDD)** using Cucumber and Gherkin feature files. Test scenarios are in English to describe user interactions, app behavior, and GA event tracking. This layer ensures collaboration between technical and non-technical stakeholders, allowing product owners, QA engineers, and developers to understand and validate the requirements.

2. Page Object Layer: The **Page Object Model (POM)** forms the next layer, encapsulating the app's UI elements and interactions. Each screen or component of the Android application is represented by a page object, containing locators and reusable methods for interaction. This abstraction reduces code duplication, simplifies test maintenance, and isolates UI changes from the test logic. Appium communicates with the device to perform actions like clicks, scrolls, and input, ensuring that automation mimics real user behavior.

3. Analytics Engine Layer: The **Analytics Engine** is responsible for validating GA events and user interactions. It leverages Android Logcat and ADB to capture runtime logs and analytics payloads. The engine parses these logs to verify that events are triggered as expected, including event names, parameters, and sequences. By centralizing analytics verification, this layer provides a single source of truth for data accuracy, eliminating manual verification and reducing the risk of human error.

4. Performance Utilities Layer:

The **Performance Utilities** layer focuses on runtime metrics such as CPU usage, memory consumption (Java Heap and Native Heap), GPU overdraw, frame rendering, and app startup times (cold, warm, and hot starts). Using ADB commands, this layer collects and analyzes performance data, allowing the framework to detect inefficiencies or potential bottlenecks.

Collected metrics are stored and processed to provide insights into app responsiveness, smoothness, and device resource utilization.

5. Integration and Reporting:
The architecture integrates these layers with **TestNG** and **Maven** to manage execution, reporting, and dependency management. TestNG enables parallel execution of scenarios, grouping of tests, and generation of detailed reports. Maven ensures consistent builds, resolves dependencies, and facilitates integration with CI/CD pipelines for automated execution.
By employing this layered architecture, the framework achieves a **clear separation of concerns**, allowing each layer to evolve independently. It enhances code maintainability, improves test readability, and ensures that analytics validation and performance monitoring are fully automated. This design also supports scalability, enabling the addition of new test scenarios, app modules, or performance metrics without major restructuring. Overall, the architecture provides a structured and efficient foundation for reliable, data-driven validation of Android applications.

(c) GA Analytics Automation Framework
The GA Analytics Automation Framework is designed to provide end-to-end validation of Google Analytics (GA) **events** in the Android application without relying on network interception(Figure 1). This approach ensures deterministic, accurate, and reproducible results, addressing the limitations of traditional manual or proxy-based GA validation techniques.
1. Architecture and Workflow: The framework follows a layered architecture, integrating BDD scenarios, page objects, an analytics engine, and performance utilities. GA validation is achieved through the Analytics Engine Layer, which monitors log outputs in real-time, identifies GA events, and verifies the event names, parameters, and sequences.
The typical workflow is as follows:
Step 1: Scenario– Test scenarios in Cucumber in Gherkin syntax, describe user interactions and GA events.
Step 2: Execution via Appium – Appium automates interactions on real devices or emulators, simulating user behaviour such as clicks, scrolls, or navigation flows.
Step 3: Log Capture – Android Logcat captures events, ADB commands extract logs and performance metrics.
Step 4: Event Parsing and Validation – The Analytics Engine parses logs, matches captured GA events against expected events defined in JSON or external configuration files, and identifies discrepancies.
Step 5: Reporting – TestNG generates structured reports with detailed success and failure information, including screenshots, log snippets, and performance metrics (Figure 2).

2. Deterministic Validation: By capturing analytics events at the device level, the framework ensures **deterministic validation**, meaning the test results are consistent and repeatable regardless of network conditions. This method prevents data loss or timing issues that may occur in network-based approaches.

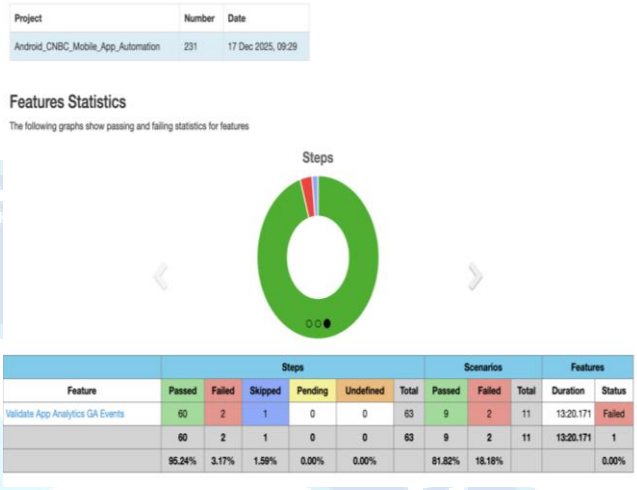

Figure 1. Feature Statistics

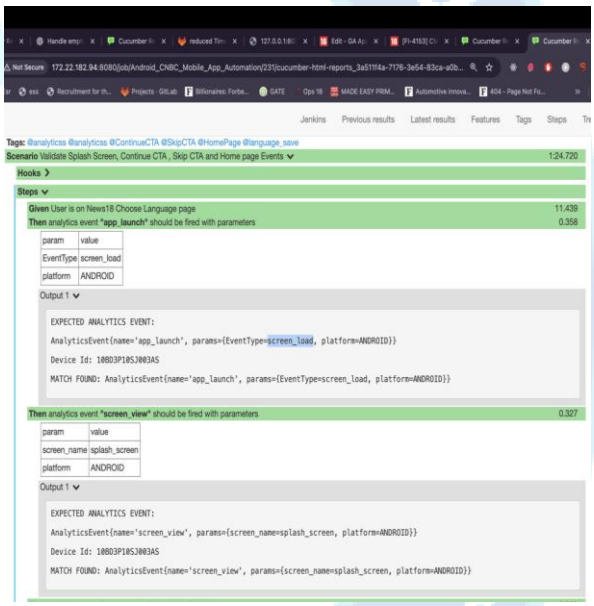

Figure 2. CUCUMBER Report

4. Results
The implementation of the proposed framework resulted in a significant reduction in debugging time and a marked improvement in analytics reliability. Automated GA validation eliminated manual log inspection and reduced human error. Performance automation enabled early detection of regressions across CPU, memory, rendering, and startup metrics. The framework provided deterministic, repeatable results across devices and environments, increasing confidence in analytics data and application performance. Overall, the solution proved scalable, maintainable, and effective for real-world Android application testing.

(a) CPU Usage Analysis
CPU utilization is a fundamental metric reflecting application efficiency and resource management. Excessive CPU usage can

lead to UI lag, increased battery consumption, and device overheating. The framework captures CPU usage during user interactions using the Android system command dumpsys cpuinfo. CPU metrics are collected before, during, and after critical test scenarios to understand how user actions impact processing load. The captured values are validated against predefined thresholds to identify abnormal spikes or sustained high CPU usage. This automated analysis enables early detection of inefficient loops, background threads, or analytics-heavy operations that could degrade performance. By correlating CPU usage with specific test scenarios, the framework provides actionable insights into performance bottlenecks and helps maintain optimal runtime efficiency.

(b) Janky Frames Analysis

Smooth UI rendering is essential for a responsive and visually pleasing user experience. The framework measures UI rendering performance using gfxinfo framestats**,** focusing on identifying Janky frames—frames that exceed the rendering deadline and cause visible stutter. During automated test execution, frame rendering data is collected while performing scrolls, animations, and screen transitions. The framework analyses frame timing to calculate the percentage of Janky frames and overall rendering consistency. High Jank percentages indicate layout inefficiencies, excessive overdraw, or complex UI hierarchies. This automated approach ensures that UI performance regressions are detected early, allowing teams to optimize layouts and rendering pipelines before release.

(c) Java Heap Memory Analysis

Java heap memory management plays a vital role in application stability and performance. Inefficient memory usage or leaks can lead to frequent garbage collection, UI freezes, or crashes. The framework monitors **Java heap usage** during extended user journeys and repetitive interactions. Heap metrics are collected using ADB commands that track memory allocation trends over time. Sudden increases or non-recovering memory patterns indicate potential leaks or inefficient object handling. Automated monitoring enables consistent detection of memory-related issues that are difficult to identify through manual testing. This analysis ensures that the application maintains healthy memory behaviour across sessions and devices.

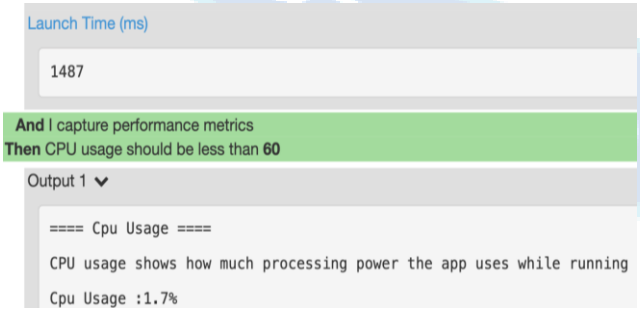

Figure 3. CPU usage report

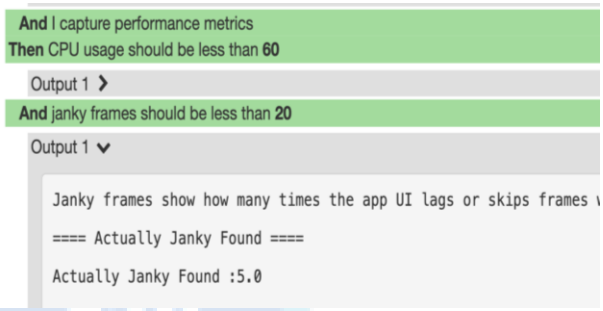

Figure 4. Janky frame report

(d) Native Heap Memory Analysis

Native heap memory is primarily used by media processing, graphics rendering, and native libraries. Issues in native memory often result in crashes that are harder to debug than Java memory leaks. The framework monitors native heap usage to identify abnormal growth patterns during runtime. Native memory metrics are captured during media playback, animations, and heavy UI operations. By analyzing these metrics, the framework helps detect issues in image decoding, video rendering, or native integrations. This proactive monitoring strengthens app stability and prevents critical runtime failures.

(e) Network and Data Usage Analysis:

Network efficiency is a key performance factor, especially for analytics-heavy applications. The framework captures RX (received) and TX (transmitted) data usage during user interactions to ensure optimal network consumption. Data usage metrics are collected using ADB commands while executing analytics and content-driven scenarios. Excessive data transfer may indicate redundant API calls, inefficient payloads, or unnecessary background syncs. Automated validation ensures that analytics tracking does not negatively impact data consumption or user costs.

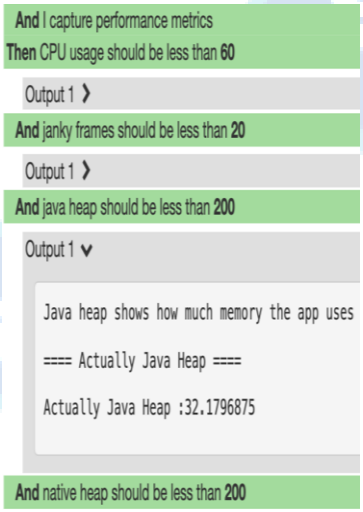

Figure 5. JAVA heap report

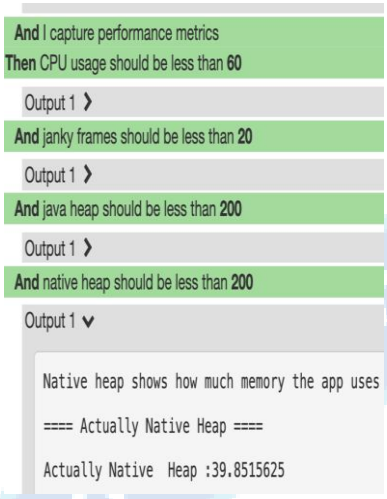

Figure 6. Actual native heap report

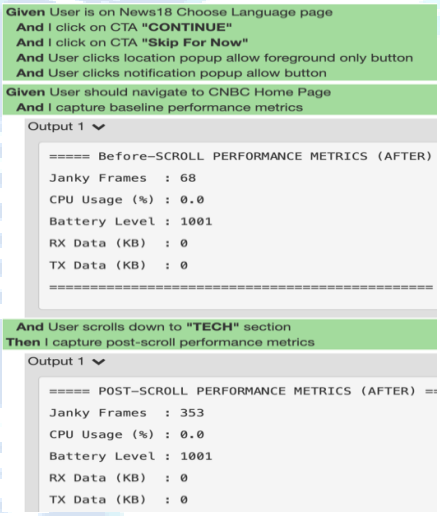

Figure 7. Battery impact report

(f) Battery Impact Analysis: Battery consumption is closely correlated with CPU usage, UI rendering, and network activity. The framework analyzes battery impact by correlating metrics captured across CPU, rendering, and memory layers.Instead of isolated battery readings, the framework focuses on identifying patterns that contribute to battery drain, such as sustained CPU spikes or excessive janky frames. This holistic approach ensures that performance optimizations translate into real-world battery efficiency.

(g) CI/CD Integration: The framework is designed to scale seamlessly across CI/CD pipelines and multiple devices. Maven-based builds, TestNG execution, and HTML reporting allow the framework to integrate easily with continuous integration tools. Automated execution in CI environments ensures consistent analytics and performance validation for every build. Multi-device execution enables broader coverage, improving confidence in production releases. Reports generated during CI runs can be archived as build artifacts for long-term analysis.

Figure 8. Complete real time performance analysis report

6. Conclusion:

This study presented a comprehensive and unified automation framework for validating Google Analytics events and monitoring runtime performance in Android applications. By combining UI automation, log-based analytics validation, and system-level performance monitoring, the framework addresses a significant gap in existing mobile testing practices. The use of Android Logcat as a single source of truth ensures deterministic and reliable analytics validation, eliminating dependencies on network conditions and reducing inconsistencies associated with traditional methods.

References:

[1] Kumar T, Singhal A, Priyadarshini R. Early System Failure Detection through System Log Analysis: An LSTM Approach. In 2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT) 2024 Jun 24 (pp. 1-7). IEEE.

[2] Mendoza A, Gu G. Mobile application web api reconnaissance: Web-to-mobile inconsistencies & vulnerabilities. In2018 IEEE Symposium on Security and Privacy (SP) 2018 May 20 (pp. 756-769). IEEE.

[3] Melnyk, Andrii, Lesia Dmytrotsa, Oleh Palka, Yaroslav Vasylenko, and Nataliya Klymuk. "Dynamic test case prioritisation for mobile applications based on real user behaviour data." (2025).

[4] Kovacs, A. (2026). Data Driven Optimization of Retail Application Performance Through Advanced Monitoring Metrics and Anomaly Detection Frameworks. Stanford Database Library of American Journal of Applied Science and Technology, 6(01), 175-181.

[5] Oliveira W, Oliveira DD, Braganholo V. Provenance analytics for workflow-based computational experiments: A survey. ACM Computing Surveys (CSUR). 2018 May 23;51(3):1-25.

[6] Lo D, Cheng H, Han J, Khoo SC, Sun C. Classification of software behaviors for failure detection: a discriminative pattern mining approach. InProceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining 2009 Jun 28 (pp. 557-566).

[7] Banerjee A, Guo HF, Roychoudhury A. Debugging energy-efficiency related field failures in mobile apps. InProceedings of the International Conference on Mobile Software Engineering and Systems 2016, (pp. 127-138).

[8] Lima A, Rosa L, Cruz T, Simões P. A security monitoring framework for mobile devices. Electronics. 2020 9(8):1197.
[9] Kumar H. Trustworthy Binary Classifications in Dynamic Systems Under Uncertainty, Doctoral dissertation, Georgia Institute of Technology, 2024.
[10] Kim J, Savchenko V, Shin K, Sorokin K, Jeon H, Pankratenko G, Markov S, Kim CJ. Automatic abnormal log detection by analyzing log history for providing debugging insight. InProceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice 2020 Jun 27 (pp. 71-80).
[11] Baek, Young-Min, and Doo-Hwan Bae. "Automated model-based android gui testing using multi-level gui comparison criteria." In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, pp. 238-249. 2016.
[12] Subramanian, Ramaswamy, Ning Chen, and Tingting Zhu. "Behavior Driven Test Automation Framework." In Proceedings of the International Conference on Software Engineering Research and Practice (SERP), pp. 17-23. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2017.
[13] Kothokatta, Lingaraj. "Parallel Automation for Cross-Browser and Cross-Device Validation in OTT Systems." International Journal of Research Publications in Engineering, Technology and Management (IJRPETM) 8, no. 5 (2025): 12919-12928.
[14] Baktha, Kishore. "Evaluating the Performance and Capabilities of Popular Android Mobile Application Testing Automation Frameworks in Agile/DevOps Environment." (2020).
[15] Carlos Ruiz, Zafón, and Goytisolo Juan. "The Role of DevOps in Mobile Application Development: CI/CD Pipelines for Faster Releases." Information Horizons: American Journal of Library and Information Science Innovation 1, no. 2 (2023): 4-12.
[16] Kanagarathinam, Madhan Raj, Krishna M. Sivalingam, and Gunjan Kumar Choudhary. "Application prioritization engine for enhancing real-time performance in smartphones." IEEE Transactions on Network and Service Management 21, no. 1 (2023): 773-788.
[17] Sahay, S., Anurag Banoudha, and Raghawendra Sharma. "On the use of ANFIS for Ground Water Level Forecasting in an Alluvium Area." International Journal of Research and Development in Applied Science and Engineering 2, no. 1 (2012): 1-7.
[18] Sahay, S., Anurag Banoudha, and Raghawendra Sharma. "Comparative study of soft computing techniques for ground water level forecasting in a hard rock area." International Journal of Research and Development in Applied Science and Engineering 4, no. 1 (2013): 1-6.