Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: www.al-kindipublisher.com/index.php/jcsts

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

| RESEARCH ARTICLE

A Scalable Framework for Hardware-in-the-Loop (HIL) Firmware Validation in Complex IOT Ecosystems

Sohan Singh Thakur Kaling
Independent Researcher, USA
Corresponding Author: Sohan Singh Thakur Kaling, **E-mail**: singhsohan@gmail.com

| ABSTRACT
The complexity of Internet of Things devices poses significant validation challenges with their progression from isolated functions to ecosystems of interconnected systems. The software-hardware interaction bugs often cannot be identified during traditional testing, especially in power management, communication protocols, and sensor interfaces. The described scalable Hardware-in-the-Loop framework deals with these issues by introducing a hybrid between declarative and procedural architecture that decouples the implementation of tests and their specification. Python procedural engine is a hardware abstraction engine, and YAML configuration can be used to write domain experts without understanding abstract frameworks. Such an architecture is bringing validation out of the manual and subjective process, and into an automated and deterministic process, which is part of development pipelines. The framework allows prompt feedback on changes in the code, concurrent running of tests, and extensive data gathering that will reveal hidden peculiarities. Using physical hardware testing during the development lifecycle, as opposed to the end of it, improves the product quality, provides faster development speed despite increased testing needs.

| KEYWORDS

Hardware-in-the-Loop (HIL), Firmware Validation, Configuration-as-Code, Test Automation, CI/CD Integration.

| ARTICLE INFORMATION

ACCEPTED: 12 November 2025 **PUBLISHED:** 01 December 2025 **DOI:** 10.32996/jcsts.2025.7.12.24

1. Introduction

1.1. The Growing Challenge of IoT Firmware Validation

The rapid expansion of Internet-of-Things technology has fundamentally transformed embedded systems from simple isolated devices into complex interconnected networks, creating unprecedented firmware validation challenges. Modern IoT devices incorporate sophisticated power management systems, real-time communication protocols, and intricate sensor networks that must function reliably across diverse operating environments. The validation landscape has become increasingly complex as these systems evolve beyond traditional embedded architectures.

Research reveals a critical pattern in embedded system failures: approximately 64% of critical defects occur at the hardware-software interaction boundary rather than within isolated software components [1]. These defects typically manifest in scenarios that traditional software-only testing methodologies fail to address effectively:

- Power state transitions during sleep/wake cycles and battery charging
- Communication protocol handshakes in wireless connectivity
- Sensor calibration and feedback loop processing
- Timing-sensitive operations requiring precise synchronization

The validation challenge intensifies significantly when organizations maintain a unified firmware codebase supporting multiple hardware variants. Each hardware configuration introduces new variables into the testing matrix, creating exponential growth in test permutations. This combinatorial explosion renders comprehensive manual testing logistically impossible within modern development timeframes that have compressed from months to weeks due to market pressures.

Manual laboratory testing approaches introduce substantial variability based on operator expertise and equipment configuration. The inherent limitations in repeatability and scalability become particularly problematic when applied to product families requiring continuous regression testing throughout accelerated development cycles. Without standardized automation, validation becomes a significant bottleneck that conflicts directly with market-driven timeline requirements.

1.2. Economic and Market Implications

The financial stakes of effective validation have escalated alongside market growth projections exceeding $1.3 trillion by 2026 [2]. The IoT landscape shows remarkable expansion across diverse sectors including industrial automation, healthcare monitoring, and smart infrastructure. This growth has been accompanied by escalating customer expectations regarding reliability and performance.

The consequences of firmware defects extend far beyond immediate remediation costs to encompass reputational damage, regulatory compliance issues, and potential product recalls, factors that can determine market success or failure. This reality creates a fundamental tension between accelerating development cycles and ensuring comprehensive quality validation. Organizations face mounting pressure to simultaneously decrease time-to-market while enhancing product quality, as early market entrants often establish significant competitive advantages through ecosystem effects and customer acquisition.

1.3. The Paradigm Shift in Validation

Traditional sequential development approaches, where hardware validation follows software development as a discrete phase, have become increasingly untenable. This separation creates substantial feedback delays between code implementation and discovery of hardware-dependent issues, significantly increasing remediation costs compared to early-stage detection.

Forward-thinking organizations have begun implementing integrated methodologies that incorporate hardware-dependent testing throughout the development lifecycle. This paradigm shift acknowledges the fundamental interconnection between firmware and hardware in IoT systems and the necessity of concurrent validation rather than sequential verification.

Effective validation methodologies must now bridge the gap between digital simulation and physical hardware interaction while generating quantitative, reproducible results compatible with modern development workflows. This integration requirement has driven interest in automated, scalable frameworks that can validate firmware against actual hardware without creating bottlenecks in accelerated development cycles.

The Hardware-in-the-Loop validation framework presented in subsequent sections addresses these challenges through a structured, repeatable approach to firmware testing that integrates seamlessly with contemporary software development practices while accommodating the physical realities of embedded systems. By enabling concurrent validation of hardware-dependent functionality throughout development, this approach transforms validation from a sequential bottleneck into an integrated quality assurance mechanism that supports both rapid development and robust product reliability.

2. Current Challenges in Embedded System Validation

2.1. The Evolution of Validation Requirements

The transformation of embedded systems from isolated devices to interconnected IoT ecosystems has fundamentally altered validation requirements. Traditional validation methodologies developed for previous generations of embedded technology operate on assumptions that no longer apply in complex, multi-connected environments. These conventional approaches face increasing inadequacy as IoT devices incorporate sophisticated communication protocols, dynamic power management systems, and complex sensor networks operating under variable environmental conditions.

2.2. The Limitations of Manual Testing

Despite recognized shortcomings, manual testing procedures remain surprisingly prevalent across the embedded systems industry. This approach typically involves engineers connecting oscilloscopes, power supplies, and other measurement equipment to devices under test while visually assessing signal waveforms and system responses. This methodology introduces several critical limitations:

2.2.1. Subjectivity and Inconsistency

Visual inspection inherently introduces subjective interpretation into what should be an objective analysis process. The assessment of waveform characteristics, timing relationships, and signal quality becomes dependent on individual judgment rather than quantitative metrics. This subjectivity creates significant variation in test outcomes between different validation sessions or when conducted by different personnel, undermining result consistency and reliability.

2.2.2. Documentation Deficiencies

Manual testing processes frequently lack standardized documentation practices. Test procedures, measurement criteria, and acceptance thresholds often exist as tribal knowledge rather than formal specifications. This results in validation artifacts of varying quality and completeness, creating challenges for knowledge transfer, regulatory compliance, and defect reproduction.

2.2.3. Scalability Constraints

As product complexity increases, manual testing creates severe scalability bottlenecks. The linear relationship between feature count and testing time becomes unsustainable as development schedules compress. Validation cycles consume disproportionate timeline allocations that conflict directly with accelerated development expectations, forcing quality compromises to meet market windows.

2.3. Sequential Methodology

Perhaps most critically, conventional validation approaches position hardware testing as a sequential activity following firmware development rather than an integrated, parallel process. This creates substantial feedback delays between code implementation and the discovery of hardware-dependent issues. Research examining embedded systems development workflows demonstrates that this temporal disconnect significantly increases remediation costs compared to early-stage detection, introducing inefficiencies that propagate throughout the development lifecycle [3].

2.3.1. The Simulation Shortfall

Software simulation environments, while valuable for certain aspects of validation, demonstrate fundamental limitations when addressing hardware-dependent behaviors. These environments operate on abstract representations of hardware rather than modeling the complex electrical and timing characteristics of physical devices.

Simulations struggle to accurately replicate critical phenomena including:

- Power supply fluctuations and brownout conditions
- Electromagnetic interference effects on signal integrity
- Temperature variations affecting component performance
- Timing jitter and clock synchronization issues
- Sensor input variations under real-world conditions

These physical-layer behaviors significantly influence system operation in deployment environments but frequently evade detection in simulation-only validation approaches.

2.4. Enterprise Solutions and Adoption Barriers

Industries like automotive and aerospace have developed sophisticated hardware-in-the-loop testing methodologies to address these challenges. However, these solutions typically involve specialized test equipment with proprietary interfaces and complex software environments. These enterprise-grade systems introduce significant adoption barriers within consumer electronics ecosystems due to:

- Integration challenges with modern development frameworks
- High implementation complexity requiring specialized expertise
- Substantial capital investment requirements
- Architectures designed for singular complex systems rather than distributed device ecosystems

These adoption barriers create a significant validation gap wherein many embedded systems projects remain inadequately tested for physical-layer phenomena. This gap creates elevated risk profiles particularly in power management, communication protocols, and sensor interfaces, precisely the areas where critical IoT failures most commonly occur [4].

The fundamental disconnect between validation requirements and available methodologies necessitates new approaches that effectively bridge simulation and physical hardware while remaining compatible with the economic and operational constraints of IoT development.

Challenge Category	Description	Impact
Manual Testing Limitations	Subjective assessment of waveforms	Inconsistent test outcomes
Documentation Issues	Non-standardized documentation	Variable quality artifacts
Scalability Constraints	Single device testing bottlenecks	Timeline allocation problems
Sequential Methodology	Hardware testing after development	Delayed feedback loops
Simulation Inadequacy	Abstract representation of hardware	Missing physical phenomena
Enterprise Solution Barriers	Specialized equipment requirements	Adoption challenges in IoT

Table 1: Current Challenges in Embedded System Validation [3, 4]

3. The HIL Framework: A Declarative-Procedural Approach

3.1. Architectural Innovation

The Hardware-in-the-Loop validation framework introduces a transformative architecture for embedded systems testing through its hybrid declarative-procedural model. This innovative approach directly addresses the fundamental challenges of IoT firmware validation by establishing a clear separation between test implementation (how tests execute) and test specification (what should be tested). This separation represents a significant departure from traditional monolithic test scripts where test logic and test parameters are tightly coupled, creating maintenance challenges and expertise barriers.

3.2. The Procedural Foundation

The framework's procedural component consists of a Python-based test engine built on the PyTest framework, providing comprehensive hardware abstraction capabilities for diverse test equipment interaction. This abstraction layer encapsulates the complexity of communicating with hardware test equipment:

- Programmable power supplies that simulate various power conditions
- Electronic loads that mimic real-world device power consumption
- Data acquisition systems capturing high-resolution signal measurements
- Communication analyzers monitoring protocol exchanges

This layer presents a consistent interface accessible through straightforward function calls like set_voltage(), measure_current(), or capture_waveform(). By isolating hardware communication details from test logic, the framework significantly reduces the technical expertise required for test development while maintaining compatibility with continuous integration environments.

The procedural foundation implements specialized functions for common validation scenarios that occur across multiple product variants:

- Power transition analysis during sleep/wake events
- Communication protocol verification for wireless interfaces
- Sensor characterization under variable environmental conditions
- Timing analysis for critical operational sequences

This creates a standardized validation vocabulary that can be consistently applied across different product variants and test configurations, eliminating the need to reinvent testing approaches for each new device variant [5].

3.3. The Declarative Layer

Complementing this procedural foundation, the declarative component leverages YAML as a human-readable configuration language for defining test parameters, sequences, and acceptance criteria. YAML's straightforward syntax enables domain experts to create comprehensive test specifications without requiring detailed knowledge of the underlying test framework implementation.

Test definition occurs through YAML configuration files that specify:

- Simple parameters (voltage thresholds, timing requirements)
- Complex sequences (multi-phase charging algorithms)
- Communication events (protocol handshakes)
- Pass/fail criteria (acceptable measurement ranges)

The declarative layer supports hierarchical configuration structures with inheritance capabilities, enabling the creation of base test definitions that can be extended or specialized for specific device variants. For example, a base battery charging test can be defined once and then extended with variant-specific parameters for different battery capacities or charging profiles.

This configuration-driven methodology transforms test definitions into self-documented, version-controlled artifacts within the development ecosystem, enabling application of software engineering best practices to test maintenance. Changes can be tracked, reviewed, and reverted using the same tools and processes applied to source code [6].
Separation of Responsibilities

The separation of test specification (parameters in YAML) from test implementation (code in Python) creates a natural division of responsibilities that aligns with organizational expertise boundaries:

- Firmware engineers define what should be tested via configuration files
- Test automation specialists maintain the underlying framework
- Hardware engineers establish measurement criteria and thresholds
- Quality assurance teams orchestrate test execution and reporting

This alignment eliminates expertise bottlenecks that frequently occur with monolithic test implementations, where specialized knowledge of both test frameworks and hardware characteristics is required.

The approach has demonstrated particular effectiveness in environments with evolving requirements and expanding device portfolios. Test modifications can be implemented through configuration changes rather than code modifications, significantly accelerating the adaptation of test suites to accommodate new hardware variants or updated specifications. By enabling domain experts to directly contribute to test definitions without programming expertise, the framework removes significant friction from the validation process while enhancing test coverage and relevance.

Component	Implementation	Function	Benefits
Procedural Engine	Python-based with PyTest	Hardware abstraction	Reduced complexity barrier
Declarative Configuration	YAML format	Test parameter definition	Domain expert empowerment
Hierarchical Configuration	Inheritance capabilities	Base/variant test definitions	Reduced configuration duplication
Hardware Interface	Abstraction layer	Equipment communication	Cross-platform consistency
Function Library	Specialized test functions	Common validation scenarios	Standardized test vocabulary
Configuration-as-Code	Version-controlled artifacts	Test maintenance	Software engineering practices

Table 2: HIL Framework Components - Declarative-Procedural Model [5, 6]

4. Performance Analysis: Automation vs. Manual Testing

4.1. Fundamental Performance Disparities

Comparative evaluation of testing methodologies reveals fundamental performance disparities between automated and manual approaches for embedded systems validation. These differences manifest across multiple dimensions including efficiency, reproducibility, and detection capability. A systematic assessment of these methodologies demonstrates quantifiable advantages that automated frameworks provide over traditional manual approaches, particularly for complex IoT devices with sophisticated hardware-software interactions.

4.2. The Setup Overhead Challenge

Manual testing methodologies involve significant setup overhead, requiring precise configuration of measurement equipment, calibration of power supplies, and preparation of data collection systems. These preparatory activities typically consume 40-60% of total testing time and require specialized knowledge of equipment operation and configuration. The process typically involves:

- Physical connection of oscilloscopes, power supplies, and data acquisition equipment
- Manual calibration of measurement systems to ensure accuracy

- Configuration of trigger conditions for capturing relevant events
- Establishment of appropriate measurement ranges and scales
- Preparation of data recording mechanisms for post-test analysis

These preparation practices introduce substantial variability based on operator experience and equipment familiarity. An experienced test engineer might complete setup procedures in half the time required by a novice, creating inconsistent testing timelines that complicate project scheduling and resource allocation. This variability becomes particularly problematic in team environments where different engineers may execute different portions of the test suite.

4.3. Throughput and Resource Limitations

Beyond setup inefficiencies, manual testing inherently limits throughput to a single device configuration at any given time. Each test sequence requires continuous operator attention and interaction, creating a strict one-to-one relationship between test engineers and devices under test. This limitation creates significant validation bottlenecks during critical development phases when multiple product variants require simultaneous assessment.

The resource constraints become particularly acute during pre-release validation cycles where comprehensive regression testing across multiple device configurations creates schedule pressure that frequently results in compromised test coverage. Organizations often face difficult decisions about which test cases to prioritize and which to defer or eliminate when manual resources cannot accommodate complete validation.

4.4. The Automation Alternative

Automated test frameworks address these limitations through standardized setup procedures implemented via programmatic control of test equipment. Once configured, these systems can execute complex test sequences without human intervention, dramatically reducing variability between test iterations. The standardization transforms validation from an unpredictable, resource-constrained activity into a deterministic process with well-defined completion parameters that can be effectively integrated into development planning.

The automation architecture enables parallel execution across multiple test stations, allowing simultaneous validation of different device configurations or test scenarios. This parallelization creates multiplicative efficiency gains that scale with infrastructure investment. A single test engineer can supervise multiple automated test stations, each executing different test sequences or validating different device variants concurrently.

4.5. Data Fidelity and Anomaly Detection

Automated validation frameworks deliver substantial improvements in test fidelity through comprehensive data collection and consistent evaluation criteria. Manual testing typically relies on visual assessment of oscilloscope displays, creating scenarios where subtle anomalies may escape detection due to:

- Limited sampling frequency during visual observation
- Operator fatigue during extended test sessions
- Attention is divided across multiple measurement points
- Difficulty distinguishing minor variations from normal behavior
- Inconsistent scrutiny across different test sections

This approach also introduces interpretation variance, where identical hardware conditions might yield different test outcomes based on subjective judgment. What one engineer might classify as acceptable ripple in a power supply waveform, another might flag as problematic. This subjectivity creates inconsistent quality standards across testing sessions and undermines the reliability of validation results.

Furthermore, manual procedures typically focus on capturing discrete measurements at specific test points rather than continuous waveform data, creating potential blind spots for transient phenomena or intermittent failures. Critical events occurring between measurement points frequently escape detection entirely, particularly timing-sensitive anomalies that appear and disappear within microseconds.

4.6. Continuous Data Acquisition

Automated approaches address these limitations through high-resolution data acquisition throughout the entire test sequence. Modern data acquisition systems can sample at rates exceeding millions of samples per second across multiple channels

simultaneously, creating comprehensive digital records of system behavior. This continuous monitoring enables detection of subtle pattern variations and momentary anomalies that frequently elude manual observation.

Advanced signal processing algorithms can analyze these comprehensive datasets to identify deviations from expected behavior patterns that would be virtually impossible to detect through visual inspection. Pattern matching, statistical analysis, and anomaly detection algorithms can process gigabytes of signal data to identify subtle precursors to failure modes or performance degradation.

4.7. Development Process Integration

The integration of automated validation within development workflows provides additional quality benefits by enabling comprehensive nightly regression testing across the entire product portfolio. This continuous validation process identifies potential issues introduced by seemingly unrelated code changes before they propagate into formal release candidates.

Automated systems can execute hundreds or thousands of test cases overnight, providing developers with comprehensive feedback at the start of each workday. This rapid feedback cycle dramatically compresses the time between introducing a defect and discovering it, significantly reducing remediation complexity. Early detection substantially reduces remediation costs compared to late-stage discovery, particularly for complex issues involving interactions between firmware components, hardware subsystems, and environmental conditions [8].

The cumulative effect of these performance improvements transforms validation from a development bottleneck into a quality enabler that accelerates rather than impedes development velocity.

Aspect	Manual Testing	Automated Framework
Setup Process	Equipment configuration by the operator	Programmatic control
Execution Time	Extended duration	Reduced validation cycles
Repeatability	Operator-dependent variations	Consistent results
Data Collection	Discrete measurement points	Continuous waveform capture
Assessment	Visual inspection of oscilloscopes	Automated analysis against thresholds
Throughput	Single device configuration	Parallel execution capability
Detection Capability	Visible anomalies only	Subtle pattern variations
Regression Testing	Limited by resources	Comprehensive nightly testing

Table 3: Testing Methodology Comparison [7, 8]

5. Enterprise Integration: HIL in CI/CD Pipelines

5.1. Transforming Development Methodology

The integration of Hardware-in-the-Loop testing within Continuous Integration/Continuous Deployment pipelines represents a fundamental transformation in embedded systems development methodology. Traditional development approaches have historically maintained strict separation between hardware validation and software development, creating isolated silos with limited information exchange. This separation creates a sequential workflow where firmware is developed, then built, then manually deployed to hardware for validation, a process that might occur weeks or months after the initial code creation.

This traditional approach introduces significant inefficiencies as hardware-dependent issues frequently remain undiscovered until late-stage integration testing. By this point, the development team may have built substantial additional functionality on potentially flawed foundations, creating complex dependency chains that complicate remediation. The resulting rework cycles introduce costly delays and frequently necessitate difficult compromise decisions between schedule adherence and comprehensive issue resolution.

5.2. Orchestration Infrastructure

Effective implementation of continuous validation requires thoughtful integration of physical testing within established software development infrastructures. This integration necessitates sophisticated orchestration systems capable of seamlessly connecting:

- Version control platforms manage code repositories
- Build automation tools generating firmware binaries

- Deployment systems transferring builds to physical devices
- Test execution frameworks controlling hardware test equipment
- Results processing engines are analyzing captured data
- Reporting systems communicating findings to development teams

Rather than developing specialized systems from the ground up, research indicates that adaptation of existing CI/CD platforms significantly improves adoption rates across engineering teams. By leveraging familiar tools like Jenkins, GitHub Actions, or GitLab CI with appropriate extensions for hardware interaction, organizations can minimize learning curves and implementation friction. This compatibility with established workflows accelerates organizational acceptance and reduces resistance to process changes [9].

5.3. The Continuous Validation Pipeline

The transition to HIL-integrated development pipelines transforms validation from a discrete phase following development into an integral quality gate throughout the entire process. This transformation begins with automated test triggering mechanisms connected directly to repository events such as:

- Pull request creation proposing code changes
- Commit submissions to development branches
- Merge operations combining feature branches
- Tag application marking potential release candidates

These events automatically initiate validation pipelines that evaluate the hardware impact of proposed changes. Upon activation, the pipeline automates firmware compilation using consistent build environments that eliminate "works on my machine" inconsistencies. The resulting binary is then deployed to physical devices in test environments without manual intervention requirements that would otherwise create workflow bottlenecks.

5.4. Test Execution and Analysis

Test execution proceeds according to predefined validation suites with intelligent prioritization based on modified components and risk assessment algorithms. This enables efficient resource utilization by focusing validation effort on areas most likely affected by recent changes rather than executing the entire test suite for minor modifications.

Upon completion, automated analysis engines process captured data, including:

- Waveform recordings from analog measurements
- Communication logs from protocol exchanges
- Timing measurements from critical operations
- Power consumption profiles during various activities

This analysis generates comprehensive reports directly linked to the originating code changes in version control systems. The linkage between code modifications and validation results creates clear traceability that enhances debugging efficiency and accountability.

5.5. Quality and Efficiency Benefits

This integrated approach delivers substantial quality improvements by identifying hardware-dependent issues at the earliest possible stage, when the causal changes remain fresh in developers' minds and before dependent functionality has been constructed on potentially flawed foundations. Early detection dramatically reduces the complexity of issue resolution and minimizes rework requirements.

The shift from post-development validation to continuous in-process testing represents a fundamental change in quality assurance philosophy that aligns with broader industry movements toward continuous delivery methodologies and agile development practices. Organizations implementing this approach consistently report reduced defect remediation costs alongside improvements in overall development velocity despite more comprehensive testing regimens [10].

By transforming hardware validation from a sequential bottleneck into a parallel, continuous activity, HIL-integrated CI/CD pipelines enable organizations to simultaneously improve product quality while accelerating development cycles, resolving the traditional tension between speed and reliability that has historically challenged embedded systems development.

Integration Element	Function	Benefit
Test Triggering	Repository event connection	Immediate evaluation
Build Automation	Firmware compilation	Workflow continuity
Deployment	Binary transfer to physical devices	Eliminated manual intervention
Test Execution	Prioritization by component risk	Optimized resource utilization
Results Processing	Waveform and log analysis	Linked to code changes
Report Generation	Comprehensive documentation	Development visibility
Quality Gate	Continuous validation	Early issue identification
Pipeline Adaptation	Modified existing CI/CD tools	Enhanced adoption rates

Table 4: CI/CD Pipeline Integration Elements [9, 10]

6. Conclusion

The Hardware-in-the-Loop validation system focuses on the primary issues of software testing in IoT by introducing a disruptive solution in the context of bridging the gap between digital simulation and the physical interaction with hardware. The framework, by not relating test logic and configuration and by linking directly to the development pipeline, will make validation at an earlier stage in the development lifecycle, where the problem can be detected and addressed at the lowest possible cost. The declarative-procedural architecture allows domain experts to develop end-to-end test specifications, and the standardized hardware abstraction allows end-to-end testing to be done on all product variants in a consistent, reproducible way. Subtle anomalies that could not have been detected otherwise are brought to light due to automated data collection and analysis, which does away with subjective assessment. Since IoT ecosystems have become increasingly complex and the market demand requires faster development cycles, automated validation frameworks are now simply a necessity, but not an option. The integrated and scalable nature of the discussed approach allows organizations to improve product quality and reliability at the same time, shorten the development schedules, and make the validation process a competitive edge, rather than a bottleneck of the development process in an increasingly competitive market.

Funding: This research received no external funding.
Conflicts of Interest: The authors declare no conflict of interest.
Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors, and the reviewers

References

[1] Andreas B and Jacob R, (2025) Automated Software Testing Using Pytest, Linköping University, 2025. Available: https://www.diva-portal.org/smash/get/diva2:1971591/FULLTEXT01.pdf
[2] Axel J and Simon P, (2024) Microservice integration testing with hardware-in-the-loop in CI/CD pipelines, Chalmers University of Technology, 2024. Available: https://odr.chalmers.se/server/api/core/bitstreams/29ff05b5-4d9c-44ef-86eb-dff820c84dc1/content
[3] Franc M et al., (2022) Hardware-in-the-Loop Simulations: A Historical Overview of Engineering Challenges, MDPI, 2022. Available: https://www.mdpi.com/2079-9292/11/15/2462
[4] MarketsandMarkets, (2024) IoT Market, 2024. Available: https://www.marketsandmarkets.com/Market-Reports/internet-of-things-market-573.html
[5] Per E S, (2021) Automated System-Level Software Testing of Industrial Networked Embedded Systems, arXiv:2111.08312v1, 2021. Available: https://arxiv.org/pdf/2111.08312
[6] Shachar S et al., (2019) Security Testbed for Internet-of-Things Devices, IEEE, 2019. Available: https://ieeexplore.ieee.org/abstract/document/8565917
[7] Siva, (2025) YAML for Scalable and Simple Test Automation, Codoid, 2025. Available: https://codoid.com/automation-testing/yaml-for-scalable-and-simple-test-automation/
[8] Srinivasa V, (2024) Exploring Challenges And Opportunities In Test Automation For IoT Devices And Systems, ResearchGate, 2024. Available: https://www.researchgate.net/publication/382737784_EXPLORING_CHALLENGES_AND_OPPORTUNITIES_IN_TEST_AUTOMATION_FOR_IOT_DEVICES_AND_SYSTEMS
[9] Vahid G et al., (2018) Testing embedded software: A survey of the literature, ScienceDirect, 2018. Available: https://www.sciencedirect.com/science/article/abs/pii/S0950584918301265
[10] Xinyue W et al., (2025) Shift-Left Techniques In Electronic Design Automation: A Survey, arXiv:2509.14551v1, 2025. Available: https://arxiv.org/pdf/2509.14551