KeAi
CHINESE ROOTS
GLOBAL IMPACT

International Journal of Innovation Studies

A COMPREHENSIVE SURVEY OF ADAPTIVE RANDOM TESTING AND HYBRID APPROACHES IN CYBERSECURITY SOFTWARE TESTING

D.Indhumathi[1], Dr. A. Selvakumar[2],
[1]Research scholar, Department of Computer Science, Sree Saraswathi Thyagaraja College, Thippampatti, Pollachi, Bharathiar University, Coimbatore, India. E-mail:indhumathidhanaraj@gmail.com
[2]Associate Professor, Department of Computer Science, Sree Saraswathi Thyagaraja College, Thippampatti, Pollachi, Bharathiar University, Coimbatore, India. E-mail:arshariniselva@gmail.com

Abstract
Cybersecurity software testing has shifted from basic, random approaches to advanced, intelligent frameworks that can identify complex vulnerabilities in modern, distributed systems. This survey offers a thorough review of testing methodologies, focusing on Adaptive Random Testing (ART) and Partition-Based ART (PB-ART) as key techniques that connect traditional and smart testing approaches. It traces the evolution of methods from foundational techniques such as fuzzing, mutation testing, and vulnerability assessment platforms to more sophisticated, hybrid approaches that leverage machine learning for prioritization, clustering, anomaly detection, and optimization. The review also examines specialized frameworks and applications in areas such as IoT, cloud computing, API security, and DevSecOps. Analysing strengths, limitations, and performance metrics highlights research gaps, including input-space explosion, benchmark realism issues, machine learning reliability, and automation challenges. The three-phase framework captures how testing methods have progressed from simple input strategies to highly adaptive systems capable of managing complex cybersecurity environments. Overall, the survey consolidates current knowledge, offers insights across various testing contexts, and suggests future directions for more effective and automated vulnerability detection in critical software systems.

Keywords: Adaptive Random Testing, Cybersecurity Testing, Machine Learning-Based Fuzzing, Hybrid Testing Frameworks, Vulnerability Detection

1. Introduction

Cybersecurity has become a critical global concern as digital systems continue to expand across web, mobile, cloud, and IoT infrastructures, creating increasingly complex attack surfaces that adversaries constantly exploit [1]. As organizations adopt interconnected services and data-driven architectures, vulnerabilities within software components become prime entry points for malicious actors aiming to compromise system confidentiality, integrity, or availability, resulting in substantial economic and operational damage [2].

The growth of cyberattacks demonstrates that many incidents originate from undetected flaws, such as input validation issues, insecure authentication flows, weak session handling, or poor configuration management. Conventional software engineering practices frequently fail to anticipate these emerging threats, and limited testing has emerged as an essential discipline focused on identifying weaknesses early in the development lifecycle. This increasing

dependence on automated tools, structured methodologies, and intelligent algorithms reflects the rising need for adaptive, scalable, and proactive testing strategies. Given this scenario, understanding the evolution of cybersecurity testing approaches becomes vital for developing robust defences. This survey aims to review advancements comprehensively from the baseline to intelligent and real-world testing frameworks.

Although numerous testing methodologies exist, they often fall short in addressing the full spectrum of vulnerabilities that modern software systems exhibit due to their scale and complexity.

Traditional fuzzing generates large volumes of random input but lacks strategic guidance, causing significant resource consumption with minimal effectiveness in exposing logic-based faults. Mutation testing simulates program variations but remains computationally expensive and difficult to integrate into large-scale systems.

Manual penetration testing demands expert involvement and cannot achieve consistent, repeatable results across diverse environments. Adaptive Random Testing (ART) and Partition-Based extension (PB-ART) improve input distribution but still struggle to capture deeper, context-specific vulnerabilities in web, cloud, and IoT environments.

Meanwhile, cyberattacks grow increasingly sophisticated, leveraging subtle weaknesses that standard automation tools fail to detect. The challenge is further intensified by high-dimensional input spaces, dynamic system behaviours, and the lack of intelligent decision-making in selecting effective test cases. These limitations highlight the pressing need for improved cybersecurity testing methodologies combining intelligence, efficiency, and adaptability [3].

Cybersecurity testing is essential because modern systems operate in highly interconnected and adversarial environments where even a single vulnerability can propagate across networks, causing widespread damage. As software becomes distributed across cloud platforms, microservices, APIs, and IoT ecosystems, attack surfaces expand rapidly and demand rigorous, continuous testing strategies [4]. Intelligent testing mechanisms help identify vulnerabilities that traditional techniques often miss by improving input diversity, prioritization, and context-awareness. Furthermore, automated cybersecurity testing assists developers in integrating security checks directly into DevSecOps pipelines, ensuring vulnerabilities are detected early rather than after deployment. Techniques such as ML-based prioritization, anomaly detection, and optimization-driven input selection significantly enhance the capability of identifying critical flaws in complex systems. These approaches also reduce redundant test cases, optimize resource usage, and support scalability for large-scale software systems. Without systematic cybersecurity testing, organizations risk exposure to high-impact attacks, data breaches, and regulatory non-compliance. Therefore, adopting advanced intelligent testing methods is crucial for ensuring software resilience and maintaining secure digital infrastructures.

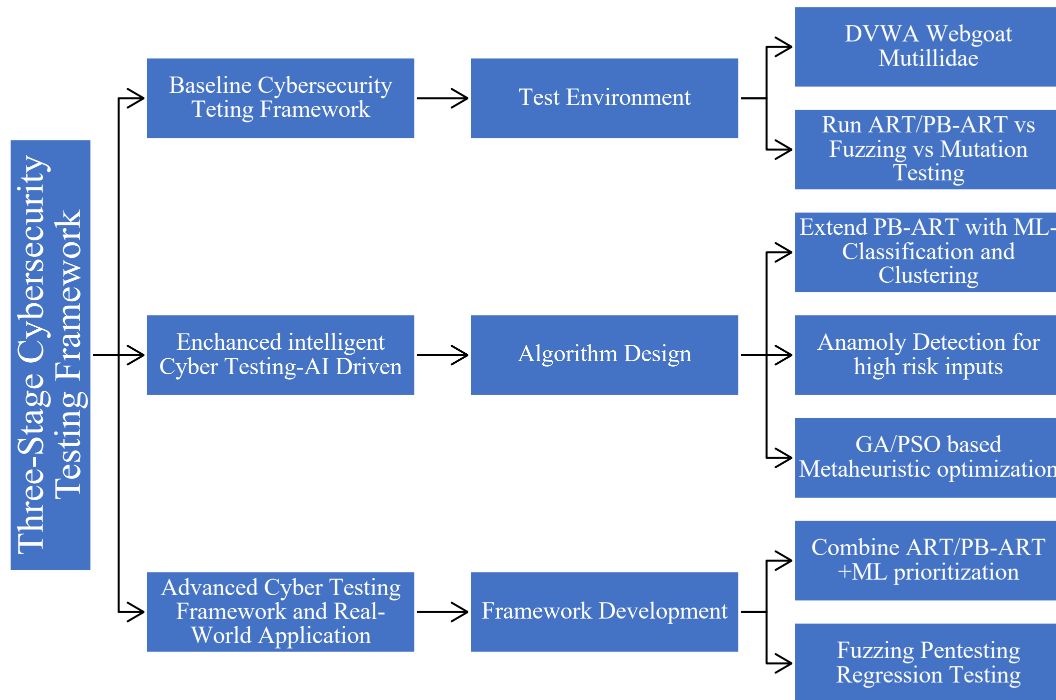

Figure 1. Framework For Cybersecurity Testing

This survey consolidates and analyses the complete landscape of cybersecurity testing approaches by aligning them with a structured three-phase research framework. First, it examines baseline testing techniques[1], including fuzzing, mutation testing, ART, and PB-ART, focusing on their roles in early vulnerability identification. Second, it explores intelligent hybrid methods[2] incorporating machine learning-based input prioritization, clustering, anomaly detection, and optimization. Third, it reviews advanced cybersecurity testing[3] frameworks designed for real-world applications across IoT, cloud computing, API security, and DevSecOps environments. The survey summarizes strengths, limitations, and comparative insights across all phases, emphasizing how testing methodologies evolve from simple random testing to highly optimized, intelligent, and adaptive systems. Finally, it highlights research gaps and future opportunities to enhance automated cyber testing, contributing to a comprehensive understanding of current and emerging practices in the domain.

2. Related Works

2.1 Baseline Testing Approaches

Fuzzing represents one of the foundational baseline techniques in cybersecurity software testing, operating on the principle of feeding programs with large volumes of semi-random, malformed, or boundary-sensitive inputs to uncover vulnerabilities that conventional deterministic testing often fails to detect [5]. At its core, fuzzing leverages the unpredictability of input diversity to stimulate unexpected program behaviours, including crashes, memory leaks, buffer overflows, logic inconsistencies, or other states that compromise software robustness. Modern fuzzing has evolved far beyond early random-input methods, incorporating sophisticated strategies such as coverage-guided fuzzing, grammar-based generation, hybrid concolic fuzzing, and machine-learning-assisted mutation heuristics. These enhancements enable fuzzers to intelligently explore code paths, understand program structures, and iteratively refine input sequences to trigger deeper execution branches. Mutation-based fuzzing stands out in particular due to its ability to derive new test cases from existing valid inputs,

introducing modifications that preserve structural correctness while increasing the likelihood of inducing anomalous states. This balance of randomness and structural awareness allows fuzzers to achieve high-throughput execution and broad input exploration [4].

A central feature of modern fuzzing is the integration of coverage-guided mechanisms, exemplified by tools such as AFL++, libFuzzer, and Honggfuzz, which instrument target programs to record execution paths, allowing the fuzzer to prioritize input seeds that exercise novel code regions [6]. This feedback loop substantially enhances efficiency by discarding unproductive inputs and intensifying exploration in areas with higher crash probability. Particularly in cybersecurity contexts, coverage-guided fuzzing has proven effective in discovering memory corruption vulnerabilities, type confusion errors, protocol desynchronization bugs, and deserialization weaknesses. Grammar-aware fuzzers further expand capability by adhering to strict input specifications (e.g., JSON, XML, HTML, or custom protocol formats, ensuring that generated inputs remain syntactically valid while probing semantic boundaries. This is critical in web applications, interpreters, compiler front-ends, and network protocol implementations, where malformed structure may simply be rejected without revealing deeper weakness.

Mutation testing serves as an essential complement to fuzzing by evaluating the quality and completeness of test suites. Unlike fuzzing, which focuses on uncovering vulnerabilities, mutation testing introduces small, systematic changes-called “mutants”- into program code, measuring whether existing tests can detect the resulting deviations [7]. The mutation score, defined as the proportion of mutants successfully identified by the test suite, quantifies the test suite’s fault-detection strength. In the context of cybersecurity, mutation testing is especially valuable because it can introduce subtle security-related modifications, such as altering validation routines, modifying sanitization logic, weakening cryptographic checks, or bypassing boundary validations. These domain-specific mutants reflect realistic vulnerability patterns, enabling researchers to evaluate not just functional correctness but also resilience against exploitation. Mutant-based evaluations are increasingly integrated with fuzzing workflows, where fuzzers are assessed based on their ability to kill mutants, thus correlating fuzzing effectiveness with security robustness.

Recent research trends reveal hybrid approaches that combine mutation testing and fuzzing to maximize vulnerability detection. For example, mutant-assisted fuzzing alters program behaviour dynamically during execution to reveal hard-to-reach states or rare execution paths that would otherwise remain dormant [8]. Similarly, concolic fuzzing integrates symbolic execution to solve constraints and guide fuzzers toward paths requiring precise input sequences. These hybrid approaches significantly expand fuzzing capability by reducing randomness, increasing precision, and enabling exploration of code guarded by complex conditions. They are particularly effective in discovering vulnerabilities in dense logic systems such as authentication modules, stateful APIs, and financial transaction engines. The combination of mutation and fuzzing techniques forms a strong baseline within Phase I because it allows the identification of structural weaknesses, evaluates detection capability, and produces rich behavioural data essential for Phase II intelligent testing approaches.

Deliberately vulnerable web applications play a critical role in Phase I by serving as standardized, controlled environments where baseline testing techniques such as fuzzing, mutation testing, and input manipulation can be systematically applied and evaluated. Three

major platforms DVWA, WebGoat, and Mutillidae are widely used due to their open-source availability, structured vulnerability sets, and realistic architecture [9]. These platforms emulate common web application components such as form inputs, session management, database backends, authentication modules, and parameterized endpoints. Their intentionally insecure configurations make them ideal for Phase I because they allow unrestricted experimentation without regulatory concerns or damage to production systems. More importantly, they provide ground truth for evaluating baseline testing performance, enabling researchers to benchmark vulnerability coverage, execution consistency, and repeatability before transitioning to more complex hybrid testing strategies in later phases.

DVWA (Damn Vulnerable Web Application) is one of the most widely used platforms for vulnerability-focused testing due to its multi-level structure, where each vulnerability is implemented in increasing difficulty modes low, medium, high, and sometimes impossible. This allows researchers to observe how fuzzers and mutation-based approaches behave under different levels of input validation and security-hardening measures [10]. DVWA includes vulnerabilities such as SQL injection, Command injection, XSS (reflected and stored), CSRF, brute-force authentication, insecure file uploads, and weak CAPTCHA validation. For fuzzing research, DVWA is particularly useful because its endpoints accept a variety of user-controlled parameters, enabling parameter-based fuzzing, payload injection, bypass testing, and anomaly detection. Researchers can analyse how fuzzers interact with server responses, whether input sanitizers block mutation-based payloads, and how baseline test metrics change across difficulty levels.

WebGoat, developed by the OWASP Foundation, adopts a pedagogical structure where each vulnerability is presented as a lesson that simulates a common developer error or security oversight. This makes WebGoat valuable for evaluating baseline testing techniques on vulnerabilities requiring contextual understanding or multi-step exploitation [11]. Unlike DVWA, which focuses on direct vulnerabilities, WebGoat presents challenges such as business logic bypass, complex session manipulation, parameter tampering across workflow steps, and chained vulnerabilities. These conditions are ideal for testing whether mutation strategies and fuzzing tools can navigate multi-step processes, maintain session consistency, and generate payloads that exploit contextual flaws. WebGoat's modular structure allows researchers to isolate and test specific vulnerability classes, making it helpful for comparative analysis across multiple categories.

Mutillidae expands the vulnerability landscape further by offering a broad array of challenges that mimic common patterns found in modern full-stack applications [12]. In addition to traditional injection flaws, Mutillidae includes insecure APIs, weak session handling, broken access control, misconfigured security headers, caching issues, open redirects, and insecure cookies. Its flexibility and large vulnerability surface make it particularly useful for large-scale fuzzing campaigns. Because Mutillidae simulates complex real-world interactions between server components, it allows fuzzers to explore deeper backend logic, multi-parameter requests, and server-side processing chains. This is beneficial for evaluating how well mutation strategies reveal vulnerabilities dependent on internal logic variations rather than input structure alone.

Code coverage stands as one of the most widely used metrics, measuring the proportion of program instructions, branches, or paths exercised by a test suite [13]. High coverage reflects

broader exploration of program logic, increasing the likelihood of revealing hidden vulnerabilities. Branch coverage and control-flow-depth metrics provide additional granularity, showing whether fuzzing reaches decision structures essential for triggering conditional vulnerabilities. Mutation score, derived from mutation testing, measures how many injected mutants are detected by the test suite, quantifying test Adequacy and assessing whether fuzzing successfully exposes semantic deviations.
Crash-based metrics are equally important. Unique crash count measures how many distinct vulnerabilities or crash types a baseline technique uncovers, which helps differentiate meaningful findings from redundant or repeated failures [14]. Time-to-first-crash reflects how quickly a technique exposes an initial vulnerability, providing insights into early-stage effectiveness. Stack-trace deduplication identifies structurally similar crashes, enabling statistical analysis of vulnerability diversity. These metrics collectively allow researchers to determine whether fuzzing is uncovering genuinely distinct flaws or repeatedly triggering the same vulnerability under different conditions.
In web environments, request-response latency, payload generation speed, and server-side anomaly patterns also contribute to evaluating testing strength [15]. Such metrics help determine whether a method can sustain extended testing campaigns or whether resource constraints limit performance. Coverage growth rate, a dynamic metric tracking how quickly new program areas are explored, is especially useful in comparing fuzzers and hybrid methods. Improvements in ML-based prioritization, ART hybridization, clustering, or anomaly-guided heuristics can only be validated when compared against these foundational measures [16]. By establishing quantitative ground truth, baseline metrics create a reliable foundation for evaluating whether advanced testing methods deliver meaningful advancements in vulnerability detection or merely add computational complexity without significant benefit.

2.2 Intelligent Hybrid Testing Approaches

Unlike baseline fuzzing, which distributes input attempts broadly across the input domain, ML-based prioritization analyses historical execution traces, code paths, coverage patterns, server responses, and anomaly frequencies to guide input generation toward regions with increased vulnerability density [17]. At its core, ML prioritization replaces uniform or random sampling with statistically informed ranking, significantly reducing unnecessary test execution while improving detection depth. Modern implementations leverage supervised learning architectures such as logistic regression, random forests, gradient boosting algorithms, and neural networks to classify inputs based on their likelihood of triggering anomalous or security-critical behaviours [18,19].
A key strength of ML-driven prioritization lies in its ability to incorporate multi-dimensional execution features. Test inputs are assessed not only by their syntactic form but also by their runtime impact, including branch traversal depth, input entropy, taint propagation characteristics, response-time deviations, and structural attributes. By recognizing correlations between these features and vulnerability triggers, ML models can identify subtle behavioural signals that traditional fuzzing fails to exploit. This multi-feature integration becomes particularly impactful when dealing with logic-heavy security flaws such as authentication bypasses, state-transition vulnerabilities, and sequence-dependent exploits [20].
Moreover, ML-based systems frequently employ adaptive retraining throughout testing campaigns. As new crashes or anomalies occur, their corresponding feature vectors are

reintegrated into the training dataset, enabling the model to progressively refine its understanding of the target system's vulnerability landscape. This dynamic learning cycle resembles reinforcement-based guidance, wherein feedback iteratively improves prioritization accuracy. Such adaptive models excel in environments with nonlinear or evolving security properties common in multi-step workflows and dynamically rendered web applications [21].

In many cybersecurity contexts, vulnerabilities depend on highly specific input patterns that are rarely encountered through random fuzzing alone. Examples include deserialization flaws requiring precise object structures, boundary-dependent logic flaws, timing-based side-channel anomalies, and multi-parameter dependency weaknesses. ML-based prioritization significantly increases the probability of exercising these security-sensitive paths by filtering inputs according to modelled risk likelihood, thereby accelerating discovery and improving coverage in narrow or complex execution regions [22].

The role of interpretability within ML prioritization is also significant. Feature-importance mappings, decision-path visualizations, and probability-weighted ranking insights provide a deeper understanding of why particular inputs are more likely to succeed. These interpretability outputs enrich subsequent hybrid testing stages—especially clustering, anomaly-guided exploration, and optimization-driven search—by revealing structural characteristics of the most impactful inputs [23]. Thus, ML-based prioritization serves not only as a filtering mechanism but also as an analytical layer informing the broader hybrid testing process.

Clustering and anomaly detection techniques further expand the capabilities of Phase II by introducing unsupervised learning mechanisms that structure and interpret the input domain without requiring ground-truth vulnerability labels. These techniques provide behavioural segmentation of the application under test, enabling testers to identify unexplored execution regions, structural input groupings, and potential vulnerability hotspots. Clustering is particularly valuable in early testing stages, where labelled data from crashes or security violations may still be sparse [24].

Clustering algorithms such as k-means, DBSCAN, hierarchical clustering, and spectral clustering organize test inputs into groups based on feature similarity—whether syntactic, semantic, behavioural, or performance-related. These clusters represent distinct behavioural zones within the input space. Inputs that lead to deep authentication logic, complex parsing routines, or backend computation pathways naturally form separate clusters. By analysing these cluster boundaries, testers can identify underrepresented regions that may contain rare or logic-intensive vulnerabilities. This provides a structured alternative to random selection, guiding testing toward areas with greater potential security significance [25].

Clustering also acts as a behaviour-informed enhancement of PB-ART (Partition-Based Adaptive Random Testing). Unlike baseline PB-ART—which partitions the input domain based on geometry or simple heuristics—cluster-driven partitioning reflects actual program behaviour, ensuring that test inputs are drawn from meaningful and diverse execution categories. This reduces input redundancy and increases the probability of triggering vulnerabilities deeply embedded in conditional branches, workflow dependencies, or stateful interactions [26].

Anomaly detection plays a complementary role by identifying deviations from normal execution behaviour. Methods such as Isolation Forest, Local Outlier Factor (LOF), One-Class SVM, and autoencoder-based reconstruction networks evaluate execution traces for unusual

timing signatures, abnormal output patterns, unexpected sanitization interactions, resource anomalies, and rare control-flow deviations. Each detected anomaly serves as a potential indicator of security-weak program states, often preceding or correlating with vulnerability exposure. Incorporating these signals into input selection allows testers to prioritize promising test cases that merit further exploration [27].

Together, clustering and anomaly detection construct behavioural maps highlighting structural relationships and irregularities within the input space. These maps later integrate directly into optimization-driven techniques by serving as guides for mutation, expansion, and refinement. Cluster centroids define semantic neighbourhoods for exploration, while anomaly scores serve as fitness indicators for optimization algorithms. This synergy ensures that intelligent hybrid testing strategies focus computational effort on behaviourally significant and security-relevant regions of the program [28].

Optimization-driven testing represents the final enhancement layer within Phase II, incorporating metaheuristic algorithms such as Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) to systematically evolve high-impact test inputs. These algorithms treat vulnerability discovery as an optimization problem, employing fitness functions based on coverage growth, anomaly intensity, crash likelihood, or partition distance. Through iterative refinement, they navigate complex input landscapes more efficiently than random or brute-force strategies [29].

Genetic Algorithms apply evolutionary principles—including selection, crossover, and mutation—to generate successive generations of candidate inputs. In cybersecurity contexts, GA proves particularly effective for crafting inputs capable of bypassing sanitization logic, traversing deeply nested branches, or matching multi-field validation constraints. GA-driven approaches have been shown to excel in testing scenarios involving sophisticated authentication mechanisms, multi-step transaction flows, and IoT firmware runtimes with unique stateful behaviours.

Particle Swarm Optimization complements GA by modelling input generation as a search conducted by a swarm of particles that adjust their trajectories based on personal and collective performance. PSO is well-suited for fine-grained exploration of numeric boundaries, timing-sensitive operations, and parameter-rich input domains. Contemporary PSO-based fuzzers have demonstrated strong results in uncovering protocol desynchronization flaws, processor-level inconsistencies, and boundary-condition vulnerabilities by converging rapidly on high-risk evaluation regions [28].

Collectively, GA and PSO provide the optimization backbone of Phase II's hybrid testing methodology. Their integration with ML-based prioritization, clustering, and anomaly detection enables a coordinated and intelligent testing strategy capable of navigating complex, multi-dimensional, and highly constrained cybersecurity environments—far beyond what traditional fuzzing or ART-based methods can achieve.

2.3 Advanced Frameworks and Domain Applications

Integrated cybersecurity testing frameworks represent the next evolutionary stage beyond the hybrid techniques introduced in Phase II. These frameworks unify diverse testing components, fuzzing engines, ML-driven prioritization, clustering modules, optimization algorithms, and runtime monitoring into cohesive ecosystems capable of supporting continuous, scalable, and multi-layered security assessments. Unlike standalone testing tools, integrated frameworks

provide an end-to-end methodology that connects test generation, execution, evaluation, and adaptation under a unified architecture [29]. Their design is driven by the need to manage complex systems with large attack surfaces, distributed components, and dynamic operational states.
Modern frameworks typically incorporate modular pipelines, enabling different testing strategies to collaborate rather than operate in isolation. For example, a framework may initiate broad-spectrum fuzzing to identify general anomalies, then automatically transition to ML-based prioritization or PSO-enhanced exploration once specific vulnerability-prone regions are identified. Such adaptive orchestration enhances efficiency by distributing computational effort across methods that are best suited to the current testing stage [25]. This layered integration significantly reduces redundancy, improves coverage, and ensures that testing remains aligned with dynamic application states.
A crucial advantage of integrated frameworks is their support for real-time data exchange across components. Coverage profiles, anomaly scores, cluster boundaries, and optimization feedback are synthesized into a shared knowledge base accessible to all modules. This cross-layer communication enables informed decision-making, such as selecting high-risk partitions for deeper ART-based sampling or allocating resources to fuzzing engines that demonstrate higher crash-density potential [26]. The result is a testing environment that is both reactive and predictive, capable of adjusting strategies based on emerging patterns within the system under test.
In practical deployments, integrated frameworks have been shown to improve testing throughput, reduce time-to-exploit discovery, and enhance resilience against evolving cyber threats. They are particularly effective for large-scale enterprise systems, multi-tenant environments, and distributed architectures where manual coordination of multiple tools is impractical. The holistic nature of such frameworks enables seamless scaling, automated orchestration, and continuous testing support, making them foundational to Phase III's advanced testing methodologies [30].
Cybersecurity testing in IoT, cloud, and API ecosystems introduces unique challenges due to their distributed architectures, resource constraints, and multi-layered interfaces. IoT systems often operate with heterogeneous hardware, limited memory, and proprietary protocols, making them difficult to assess using traditional fuzzing or mutation testing. Phase III methodologies extend hybrid ART-based and ML-assisted techniques to address these domain-specific constraints by incorporating context-aware analysis and device-level behavioural modelling [31]. These enhancements are critical in detecting firmware vulnerabilities, unsafe protocol interactions, and logic flaws arising from constrained device configurations.
Cloud environments further complicate security testing by introducing virtualization layers, dynamic resource allocation, and multi-tenant isolation requirements. Hybrid testing approaches must account for ephemeral infrastructure, API-driven orchestration, and distributed microservices. Intelligent testing strategies incorporate cloud-native telemetry—execution logs, service meshes, network traces—to guide fuzzing and anomaly detection with contextually rich behavioural signals. These signals support vulnerability discovery across inter-service communication layers, where traditional code-centric testing may overlook systemic risks.

API testing represents a critical domain within both IoT and cloud systems due to the heavy reliance on REST, GraphQL, and RPC-style interfaces. Hybrid testing engines integrate parameter-aware mutation, state-tracking fuzzing, and ML-driven prioritization to explore complex dependency structures common in modern APIs. API workflows often involve authentication tokens, chained requests, and multi-step logic sequences that require context maintenance throughout the test execution process. Modern frameworks employ reinforcement-style feedback loops to adaptively refine API request sequences, increasing the likelihood of exposing authorization flaws, workflow inconsistencies, or rate-limiting bypasses.

The integration of cybersecurity testing within DevSecOps pipelines represents a critical objective of Phase III, ensuring that security validation occurs continuously and automatically throughout the software lifecycle. Traditional security testing approaches are often executed post-development, resulting in delayed vulnerability disclosure and costly remediation cycles. By embedding hybrid testing techniques—fuzzing, ART strategies, ML-based prioritization, anomaly detection, and metaheuristic optimization—directly into CI/CD workflows, developers can perform security assessments concurrently with code changes, minimizing exposure windows and improving software resilience [31].

A key enabler of DevSecOps integration is the automation and modularity of modern cybersecurity testing frameworks. Containerized fuzzing engines, scalable ML inference modules, and automated fault analysers can be triggered during build and deployment events, ensuring that every code revision undergoes security evaluation. Coverage tracking, anomaly scoring, and optimization-driven input generation feed directly into automated decision systems that determine whether a build should pass, be flagged for review, or undergo deeper security inspection. This continuous validation ensures that vulnerabilities introduced during rapid development cycles are caught early, reducing long-term risk [31].

Hybrid testing integration within DevSecOps also requires efficient resource allocation to maintain development velocity. Intelligent prioritization enables pipelines to execute high-value tests first, using ML models to filter inputs based on predicted risk while clustering-based partitioning ensures broad behavioural coverage. Optimization algorithms such as GA and PSO further reduce test redundancy by evolving test inputs that retain high crash potential while minimizing execution cost. These strategies align with CI/CD constraints, where limited time windows necessitate highly efficient testing methodologies.

Furthermore, DevSecOps integration emphasizes reproducibility and traceability. Test results, crash logs, feature vectors, and optimization histories are archived within version-controlled repositories, enabling researchers and developers to track vulnerability evolution and reproduce behaviours across builds. These records support root-cause analysis, model retraining, and continuous improvement of testing strategies. The combination of systematic logging, automated risk evaluation, and adaptive testing transforms DevSecOps pipelines into resilient environments capable of long-term security monitoring [32].

Importantly, the shift toward DevSecOps-oriented cybersecurity testing leverages real-time telemetry from runtime environments such as Kubernetes clusters, serverless compute instances, and API gateways. This operational data provides valuable insight into how applications behave under production load, allowing hybrid testing engines to generate test cases that mimic realistic attack patterns. By aligning testing with real-world operational states,

DevSecOps pipelines bridge the gap between pre-deployment testing and in-production risk detection, enhancing overall cyber defence posture [33].
Together, these advancements position DevSecOps-integrated cybersecurity testing as an indispensable requirement for modern software ecosystems, supporting continuous assurance, rapid vulnerability detection, and robust lifecycle security management.

Table1. Comparative Analysis of Cybersecurity Testing Methods, Algorithms and Performance Characteristics

Title	Author &Year	Testing Approach	Algorithm/ Technology/software approach	Pros	Cons
Navigating the Cyber security Landscape: A Comprehensive Review of Cyber-Attacks, Emerging Trends, and Recent Developments	Mallick et al., 2024	Conceptual cyber security analysis	Machine learning, deep learning, Big-data analytics, Blockchain-based security	Provides modern cyber threats and attack patterns, integrates technical and non-technical cybersecurity solutions	Lacks experimental validation, performance evaluation metrics not provided
Test-driven development with mutation testing – an experimental study	Roman et al., 2021	Hybrid software testing	Test driven Development (TDD), Mutation testing	Enhances test strength and software quality, detects post release defects compared to traditional TDD	Evaluated in academic environment, lacking focus on adaptive testing methods
Development of Vulnerable Web Application Based on OWASP API Security Risks	Idris et al., 2021	Security testing and vulnerability-based learning environment	OWASP API security Risk modelling, Gamification-based security training, Containerization	Supports safe experimentation using containerization deployment, better understanding of real-	Lacks quantitative performance evaluation, does not include hybrid automated

				world API vulnerabilitie s	testing mechanisms
Slicing Through the Noise: Efficient Crash Deduplicatio n via Trace Reconstructi on and Fuzzy Hashing	Pang et al., 2024	Fuzz testing optimization	Execution trace analysis, Backward data dependency Slicing, Fuzzy hashing, similarity-based Greedy clustering, intel Processor trace	Reduces redundant crash reports, improves vulnerability analysis efficiency,	Focused on post-crash analysis rather than test generation ,evaluated on limited number of programs
Machine Learning-Based Anomaly Detection for Cyber Threat Prevention	Mizan ur et al., 2025	AI-based security testing/ anomaly detection approach	Decision tree, Random Forest, support vector machine, regression analysis	Support early cyber threat detection, applicable for real time monitoring environments , identifies adoption trends across sectors	High false positive rate, difficulty in detect zero-day attacks, data imbalances challenges
CovRL: Fuzzing JavaScript Engines with Coverage-Guided Reinforceme nt Learning for LLM-based Mutation	Eom et al., 2024	Hybrid intelligent Fuzz testing	LLM-based fuzzing framework, reinforcement learning, TF-IDF coverage mapping, coverage guided mutation	Reduces semantic mutation errors, integrates learning directly with coverage feedback	Computatio nal complexity due to LLM usage, evaluated mainly on JavaScript engines
BehAVExplo r: Behavior Diversity Guided Testing for Autonomous	Cheng et al., 2023	Behaviour-guided Hybrid Fuzz testing	BehaviorMiner, Temporal Feature Extraction clustering based Behavior Abstraction,	Reduces redundant failures, improves violation	Domain specific evaluation limited to ADS environment

Driving Systems			Apollo autonomous driving platform, LGSVL simulation environment	discovery efficiency,	, not directly applied to cyber security software
Revisiting Neural Program Smoothing for Fuzzing	Nicolae et al., 2023	Machine learning-Guided fuzz testing	Neural program smoothing, neural networks, Gradient based mutation, Neuzz++, MLfuzz benchmarking platform	Identifies limitations of neural fuzzing approaches, proposes reproducible benchmarking platform	ML based Fuzzers often underperform, high computational cost, scalability challenges
MOCK: Optimizing Kernel Fuzzing Mutation with Context-aware Dependency	Xu et al., 2024	Coverage guided hybrid kernel fuzz testing	Coverage guided fuzzing, language model-guided dependency learning, Linux kernel testing environment	Effectively explores deep kernel states, improves syscall sequence generation	High computational complexity due to integration of model, kernel specific applicability
Optimization Design of Privacy Protection System Based on Cloud Native	Zhang et al., 2022	Data anonymization-based security approach	Data anonymization algorithms, Risk analysis models, utility evaluation model, privacy protection modelling, spring boot, MyBatis, JSON web token, cloud service microservice containers,	Provides data privacy protection in private enterprise network, scalable cloud native micro-service deployment, supports configurable anonymization models	Performance metric for attack detections are not experimentally quantified, focuses on privacy preservation

Continuous Fuzzing: A Study of the Effectiveness and Scalability of Fuzzing in CI/CD Pipelines	Klooster et al., 2023	Continuous security Fuzz testing	Coverage-guided fuzz testing, commit triage analysis, continuous testing optimization, OSS-Fuzz environment/ CI-CD development pipelines	Significantly reduces unnecessary fuzzing effort, enables faster vulnerabilities discovery during deployment	Evaluated on limited software libraries, lacking focus on adaptive test generation

3.Overview of ART and PB-ART

3.1 Adaptive Random Testing

Adaptive Random Testing is a test generation technique designed to improve failure detection by ensuring that randomly generated inputs are evenly spread across the input domain rather than clustered in specific region [34]. The fundamental rationale behind ART is that failure-causing inputs tend to form contiguous or patterned regions, and therefore distributing test cases across the space increases the likelihood of intersecting fault-prone areas. ART incorporates distance-based selection strategies, where new candidate inputs are chosen based on their separation from previously executed tests, improving diversity and distribution.
Several ART variants which enhance the input dispersion using geometric or probabilistic rules. Firstly, Distance-based ART is a class of ART techniques where newly generated test case is chosen in order that geometric or semantic distance from all previously executed test case is enhanced. The primary objective is to impose spatial diversity in input domain whereby the probability of intersecting clustered failure regions is increased. The commonly used distance metrics consists of Euclidian distance, Manhattan distance, Hamming Distance and embedding-based distance. Secondly, Restriction-based ART (RB-ART) generated new test cases by imposing spatial restriction that stops them from being too near to before implemented test cases. Instead of explicitly increasing the distance RB-ART discard candidate inputs that destroy predefined proximity threshold. Lastly, Grid-based ART divides the input domain into different test cell and choose the test case from unvisited cells. Rather than utilizing distance calculations, diversity is enforced through structured spatial division.

Compared with pure random testing, ART consistently yields higher fault-detection effectiveness, especially in programs where input-related failures exhibit spatial patterns. Its computational efficiency also makes it suitable for large-scale systems with wide input ranges. ART’s application extends across numeric systems, GUI testing, and input-space modelling scenarios. Although not inherently intelligent, ART’s structured randomness provides a foundation for integrating advanced selection mechanisms. Ongoing research continues to refine ART heuristics for improved coverage and scalability [28].

3.2 Partition-Based ART

Partition-Based Adaptive Random Testing (PB-ART) enhances traditional ART by dividing the input domain into discrete partitions and selecting representative test cases from each region to maximize input coverage and reduce sampling bias [35]. This partitioning strategy allows the tester to achieve better structural balance by ensuring that no segment of the input space is overrepresented or neglected. By selecting one or more candidates from each partition, PB-ART improves distribution uniformity and increases the probability of intersecting failure clusters efficiently. The technique supports various partitioning approaches, including equal partitioning, adaptive partitioning, and semantic-based grouping. Firstly, equal partitioning is a constant input space division strategy where the complete input domain is partitioned into equal sized and non-overlapping sub-regions prior to test implementation. Each partition is treated uniformly and test cases are generated to make sure unbiased sampling across all partitions. Secondly, Adaptive partitioning is instant input space partition technique in which divisions are refined, integrated or subdivided during test execution based on suggestion such as failure detection, coverage information or input density. Finally, Semantic-based partitioning divides the input domain based on behavioural, logical or contextual similarity other than geometric or structural boundaries. Inputs are divided as per their semantic meaning, execution behaviour or functional characteristics. PB-ART is particularly helpful in multidimensional or highly structured domains, where direct distance computation may become computationally expensive. The method also facilitates integration with constraint-based or hybrid test generation mechanisms. Due to its structured organization, PB-ART improves both efficiency and consistency when compared with general ART methods.

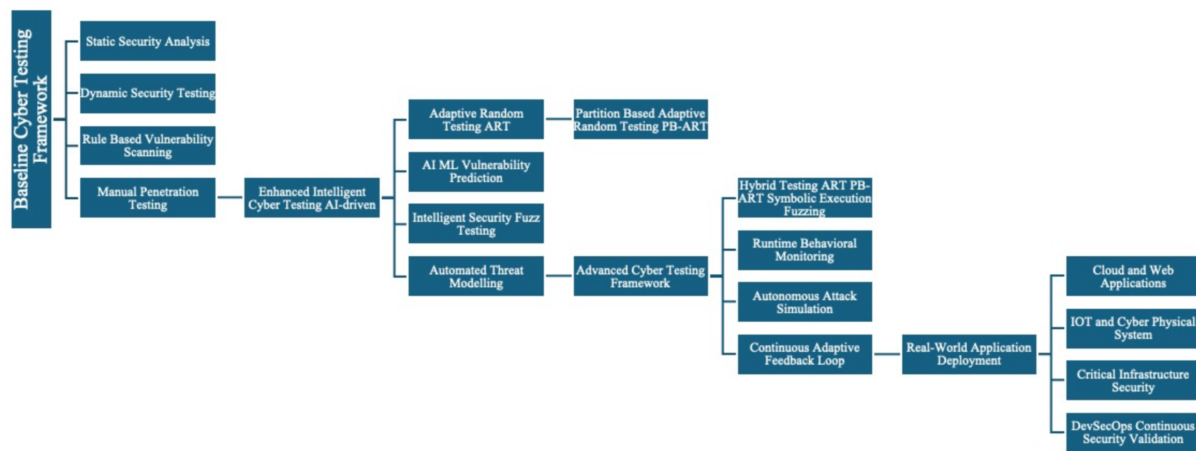

Figure 2. Framework to Illustrate Overview of ART and PB-ART

3.3 Benefits and Limitations

Adaptive Random Testing and Partition-Based ART provide several advantages, including improved input diversity, reduced clustering of test cases, and higher fault-detection capability compared with purely random testing approaches. ART-based approaches are computationally lightweight, making them practical for testing environments with large input spaces or limited resources. PB-ART further improves structural coverage by enforcing domain partitioning, ensuring more balanced sampling across the entire input region. Both methods are relatively

simple to implement and can be adapted to multiple testing contexts, including numeric computations, configuration testing, API fuzzing, and property-based testing workflows. However, ART and PB-ART still depend on randomness and may struggle with highly complex failure patterns that do not correlate well with spatial distribution. They also lack contextual or semantic awareness of program behaviour, limiting their ability to detect logic-based security vulnerabilities. Additionally, determining optimal distance metrics or partitioning strategies can be challenging. Advanced systems such as cloud platforms, machine-learning models, or dynamic web applications may require more intelligent or adaptive prioritization. Despite these limitations, ART-based approaches form a strong baseline for advanced hybrid testing strategies.

3.4 Relevance to Cybersecurity

ART and PB-ART play an important role in cybersecurity testing because their diverse input-generation strategies help uncover faults that arise from unexpected or malformed inputs frequently exploited by attackers. Cyber systems often involve vast and complex input spaces, including user-provided parameters, API payloads, network packets, authentication tokens, and configuration options. ART's even distribution of test inputs increases the likelihood of triggering vulnerabilities such as buffer overflows, input validation errors, server-side crashes, and unexpected exception states. PB-ART offers additional benefits in cybersecurity by ensuring systematic exploration of partitioned domains, which is particularly effective in web applications, protocol testing, and interface-driven systems with structured input categories. These methods also serve as foundational components for hybrid systems that incorporate intelligent prioritization, clustering, or machine-learning-based selection strategies. ART-based algorithms support scalable and automated vulnerability detection pipelines, making them well-suited for integration with fuzzers, static–dynamic hybrid test frameworks, and DevSecOps environments. Their computational efficiency enables continuous and regression testing of security-critical applications. Thus, ART and PB-ART remain highly relevant as baseline strategies within modern cybersecurity testing research.

4. Research Gaps and Challenges

Cybersecurity testing techniques illustrate a clear progression from foundational methods to highly adaptive and integrated strategies capable of addressing modern software complexity. Each group of approaches demonstrates distinct strengths, limitations, and methodological implications, contributing cumulatively to a more resilient testing ecosystem.

The baseline techniques-fuzzing, mutation testing, and evaluations on vulnerable platforms-are highly effective for uncovering low-level implementation flaws, memory corruption issues, and input-dependent vulnerabilities. These methods benefit from simplicity, speed, and broad applicability but are constrained by randomness and limited insight into deeper program logic. Intelligent hybrid testing techniques build on this foundation by incorporating machine learning, clustering, anomaly detection, and metaheuristic optimization. These strategies significantly enhance targeting precision, behavioural diversity, and analytical richness. Their limitations include reliance on quality behavioural data, increased computational demand, and the need for careful parameter tuning.

Advanced frameworks, domain-focused testing for IoT, cloud, and API systems, and integration with automated development pipelines extend cybersecurity testing into real-world deployment environments. These approaches support scalability, automation, and continuous

assessment. However, they introduce challenges related to orchestration complexity, varied domain requirements, and increased need for specialized configuration.

The earlier analysis highlights several overarching insights. First, vulnerability detection becomes more effective when guided by behavioural features—coverage patterns, anomaly indicators, cluster structures—rather than purely random sampling. Second, specialized systems such as IoT devices, distributed cloud services, and multi-step APIs require tailored testing strategies due to their architectural diversity. Third, combining ML-based prioritization, clustering methods, and optimization algorithms consistently outperforms any standalone technique by integrating multiple perspectives on input diversity and system behaviour. Finally, the shift toward automated, continuously integrated testing emphasizes the importance of ongoing security validation rather than isolated testing cycles.

Table 2. Comparative Analysis of Existing Works

Work	**Approach Type**	**Vulnerability Detection**	**Testing Efficiency Gain (%)**	**Diversity Handling**	**Computational Cost**	**Real-World Validation**
CovRL	AI + Reinforcement Learning Hybrid Fuzzing	High	Medium	Yes	High	Yes
MOCK	Context-Aware Kernel Fuzzing	High	Medium	Yes	High	Yes
BehAVExplor	Behavior-Guided Hybrid Fuzzing	High	Medium	Yes	High	Simulation
CI/CD	DevSecOps Security Testing	Medium	Medium	Partial	Medium	Yes
Crash Method	Fuzzing Analysis Optimization	Medium	High	Partial	Medium	Yes
ML Anomaly	AI-Based Threat Detection	Medium	Low	No	Medium	Yes
VAIS API	Vulnerability Learning	Low	Low	No	Low	Partial

	Environmen t					
Cloud- PPS	Privacy Protection Framework	Low	Medium	No	Medium	Enterpris e

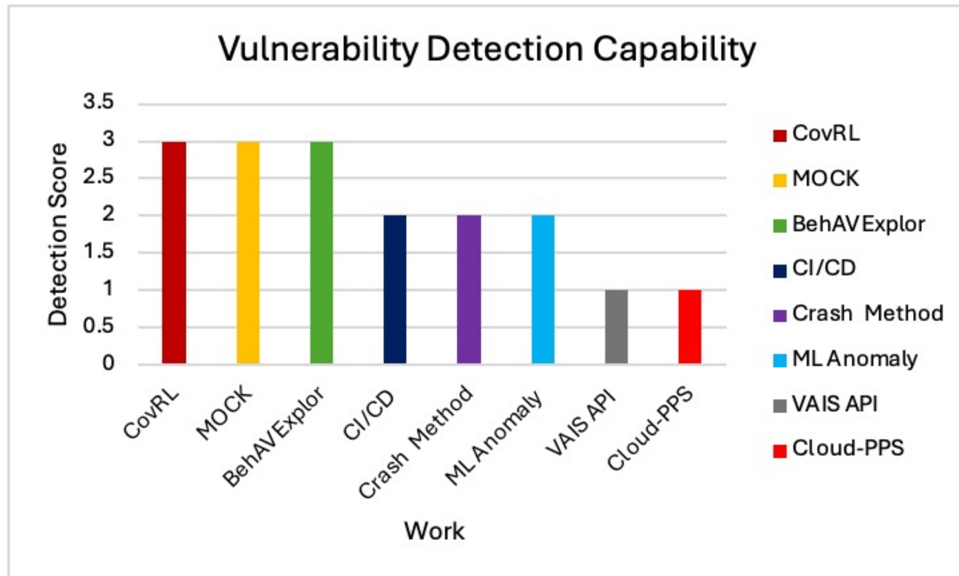

Figure 3. Comparison of Vulnerabilities Detection Capability

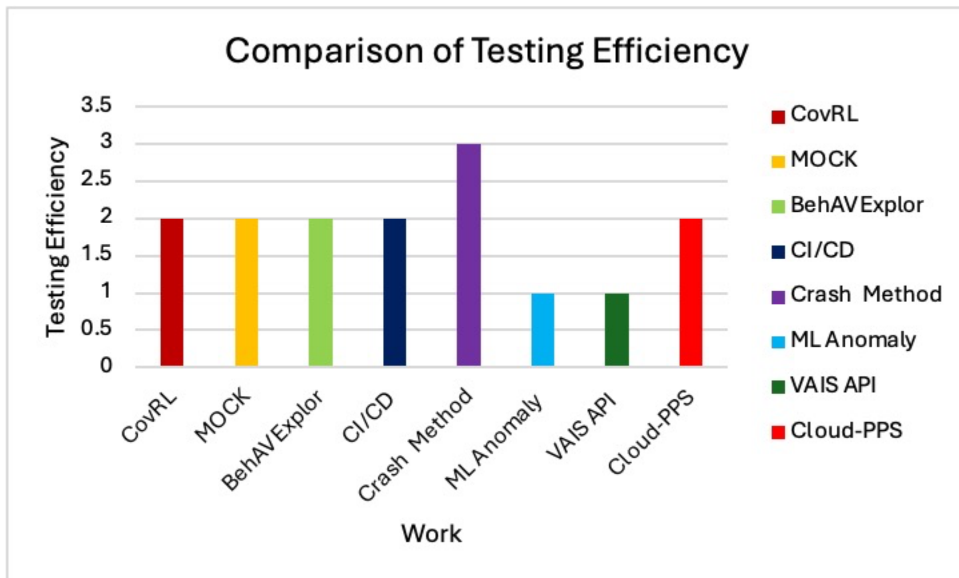

Figure 4. Comparison of testing efficiency

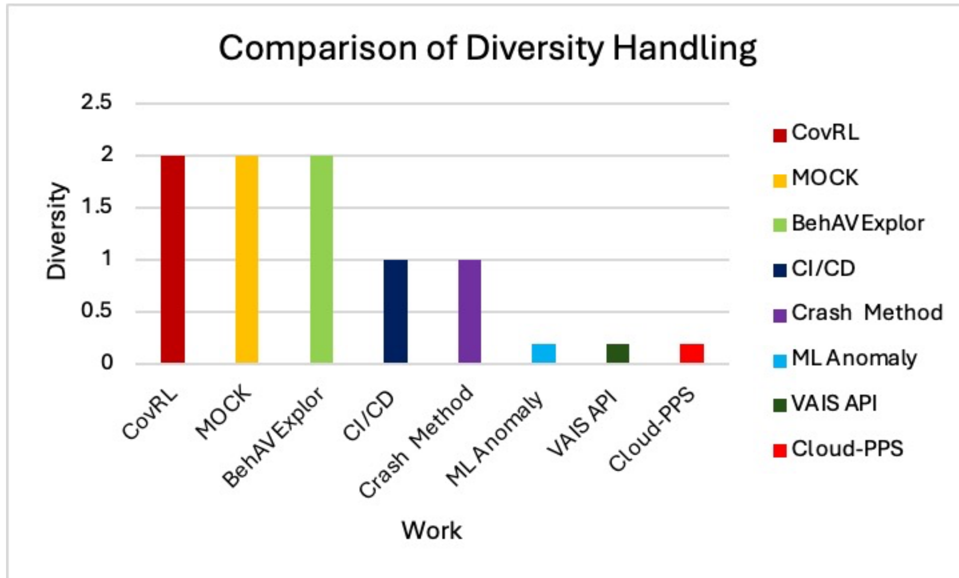

Figure 5. Comparison of Diversity Handling

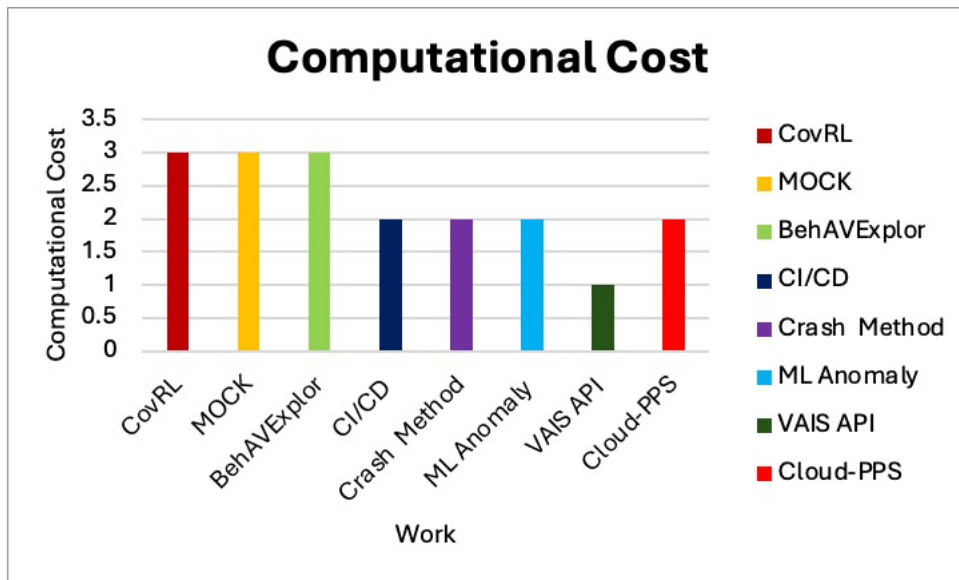

Figure 6. Comparison of Computational Cost

Figure 7. Comparison of Real-World Validation

Gaps in Current Methods

Despite these advances, notable gaps remain. Existing behavioral models often struggle with sparse or ambiguous signals, limiting their ability to detect nuanced vulnerabilities. Many integrated frameworks still lack generality across heterogeneous systems, particularly in resource-constrained or proprietary environments. Optimization methods risk premature convergence if fitness functions are not carefully constructed. Finally, real-world deployments demand faster, lighter testing mechanisms that many current techniques struggle to provide.

Evolving threats

Cyber attackers continuously develop new exploit patterns, evasion techniques, and multistage attack chains that surpass the detection capabilities of traditional testing tools. These evolving threats include polymorphic payloads, adaptive malware, and logic-based bypass techniques that are difficult to model in advance. Testing systems often depend on previously seen behavior, making them slow to respond to newly emerging vulnerabilities. As a result, outdated heuristics or static analysis rules frequently miss subtle, innovative attack vectors. This constant evolution creates a moving target, requiring testing methods to adapt rapidly to remain effective.

Input space explosion

Modern applications—especially those built on distributed microservices, IoT ecosystems, and multi-parameter APIs—produce enormous and highly diverse input domains. Each parameter, configuration, or workflow path introduces additional permutations, causing the input space to grow exponentially. Even advanced prioritization strategies, ML-driven guidance, and optimization algorithms struggle to fully explore these vast domains. Limited computational resources further restrict the depth of exploration, leaving many execution paths untested. Consequently, significant portions of the application remain unexercised, increasing the likelihood of undetected vulnerabilities.

Benchmark limitations

Widely used benchmarking platforms provide controlled environments but fail to mirror the operational complexity of real systems. They often lack features such as dynamic request routing, distributed workload balancing, layered authentication, or device-specific constraints

found in production environments. As a result, vulnerabilities identified on these benchmarks may not generalize, while real-world weaknesses may remain untested. Furthermore, simplified application logic and predictable data flows reduce the realism of evaluation results. This disconnect limits the reliability of benchmarking as an indicator of real-world testing effectiveness.

ML model reliability

Machine learning components used for prioritization, clustering, or anomaly detection rely on consistent and representative data to function accurately. However, production environments frequently generate noisy, incomplete, or heavily imbalanced datasets, which degrade model performance. Models may learn misleading correlations, leading to inaccurate input ranking or false anomaly signals. Changes in application behavior—such as software updates or configuration shifts—can cause models to drift, reducing long-term reliability. Ensuring continuous retraining and validation becomes challenging, especially in fast-changing deployment environments.

Automation issues

Fully automated cybersecurity testing pipelines often struggle with environmental inconsistencies, dependency conflicts, and variations in execution context. These issues can cause test failures unrelated to security, reducing the accuracy and stability of automated workflows. Additionally, orchestrating multiple testing tools—fuzzers, monitoring systems, ML modules—requires careful synchronization that is difficult to maintain over time. Limited visibility into distributed runtime states further complicates automation, as important behavioral signals may be missed. Together, these factors introduce blind spots and reduce the overall reliability of continuous testing systems.

5. Conclusion

This survey has examined the evolution of cybersecurity software testing techniques with a particular focus on Adaptive Random Testing (ART) and Partition-Based ART (PB-ART). By reviewing baseline security testing practices, hybrid intelligent approaches, and the progressive integration of automation and machine learning, the study highlights how current solutions partly address but do not fully overcome challenges such as input-space explosion, benchmark limitations, and the difficulty of ensuring effective vulnerability detection across modern cyber domains. The comparative insights and identified research gaps provide a consolidated understanding of where existing methods fall short and where future advancements are necessary. Future work will initially focus on strengthening the core ART and PB-ART mechanisms by improving input-space modeling and distance-based selection strategies. Refining these baseline techniques can increase test diversity and vulnerability exposure in early testing stages. This foundational improvement will also support more advanced intelligent approaches in later phases.

References

1. Mallick, M. A. I., & Nath, R. (2024). Navigating the cyber security landscape: A comprehensive review of cyber-attacks, emerging trends, and recent developments. World Scientific News, 190(1), 1-69.
2. Elder, S., Rahman, M. R., Fringer, G., Kapoor, K., & Williams, L. (2024). A survey on software vulnerability exploitability assessment. *ACM Computing Surveys*, *56*(8), 1-41.

3. Görz, P., Mathis, B., Hassler, K., Güler, E., Holz, T., Zeller, A., & Gopinath, R. (2023). Systematic assessment of fuzzers using mutation analysis. In *32nd USENIX Security Symposium (USENIX Security 23)* (pp. 4535-4552).
4. Reddy, K. J. (2025). Traditional Approaches and Limitations. In *Innovations in Neurocognitive Rehabilitation: Harnessing Technology for Effective Therapy* (pp. 39-51). Cham: Springer Nature Switzerland.
5. Yu, Z., Liu, Z., Cong, X., Li, X., & Yin, L. (2024). Fuzzing: Progress, Challenges, and Perspectives. *Computers, Materials & Continua*, *78*(1).
6. Alsaedi, A., Alhuzali, A., & Bamasag, O. (2022). Effective and scalable black-box fuzzing approach for modern web applications. *Journal of King Saud University-Computer and Information Sciences*, *34*(10), 10068-10078
7. Roman, A., & Mnich, M. (2021). Test-driven development with mutation testing–an experimental study. *Software Quality Journal*, *29*(1), 1-38.
8. Petrović, G., Ivanković, M., Fraser, G., & Just, R. (2021). Practical mutation testing at scale. *arXiv preprint arXiv:2102.11378*.
9. Idris, M., Syarif, I., & Winarno, I. (2021, September). Development of vulnerable web application based on OWASP API security risks. In *2021 International Electronics Symposium (IES)* (pp. 190-194). IEEE.
10. Wang, J., & Liu, X. (2023, May). Research on Software Security Based on DVWA. In *2023 IEEE 3rd International Conference on Electronic Technology, Communication and Information (ICETCI)* (pp. 38-42). IEEE.
11. Koman, J., & Janiszewski, M. (2025). SCAnME-scanner comparative analysis and metrics for evaluation: J. Koman, M. Janiszewski. *International Journal of Information Security*, *24*(3), 147.
12. GULEC, U., & IONESCU, V. M. (2019). IMPLEMENTING A NETWORK SECURITY LABORATORY AT UNIVERSITY UNDERGRADUATE LEVEL. *University of Pitesti Scientific Bulletin Series: Electronics and Computer Science*, *19*(2), 37-44.
13. Rawat, S., Jain, V., Kumar, A., Cojocar, L., Giuffrida, C., & Bos, H. (2017). Vuzzer: Application-aware evolutionary fuzzing. In *2017 Network and Distributed System Security (NDSS) Symposium:[Proceedings]* (pp. 1-14). Internet Society.
14. Pang, L., Qian, C., Kuang, X., Qin, J., Zang, Y., & Zhang, J. (2024). Slicing Through the Noise: Efficient Crash Deduplication via Trace Reconstruction and Fuzzy Hashing. *Electronics*, *13*(23), 4817.
15. Mizanur, M., Kumer, S., & Reza, N. (2025). Machine Learning-Based Anomaly Detection for Cyber Threat Prevention. Journal of Primeasia, 6(1), 1-8.
16. Zhang, A., Zhang, Y., Xu, Y., Wang, C., & Li, S. (2023). Machine Learning-Based Fuzz Testing Techniques: A Survey. *IEEE Access*, *12*, 14437-14454.
17. Wang, Y., Jia, P., Liu, L., Huang, C., & Liu, Z. (2020). A systematic review of fuzzing based on machine learning techniques. *PloS one*, *15*(8), e0237749.
18. Nicolae, M. I., Eisele, M., & Zeller, A. (2023, November). Revisiting neural program smoothing for fuzzing. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 133-145).

19. Xu, J., Zhang, X., Ji, S., Tian, Y., Zhao, B., Wang, Q., ... & Chen, J. (2024). Mock: optimizing kernel fuzzing mutation with context-aware dependency. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.
20. Eom, J., Jeong, S., & Kwon, T. (2024). Covrl: Fuzzing javascript engines with coverage-guided reinforcement learning for llm-based mutation. *arXiv preprint arXiv:2402.12222*.
21. Li, Z., Zhou, S., Luo, X., Wei, H., & Zhang, G. (2025). CAFLnet: a network protocol fuzzing framework based on selection algorithm with enhanced contextual information. *Cybersecurity*, *8*(1), 81.
22. Shi, J., Zou, W., Zhang, C., Tan, L., Zou, Y., Peng, Y., & Huo, W. (2022). CAMFuzz: Explainable Fuzzing with Local Interpretation. *Cybersecurity*, *5*(1), 17.
23. Fang, Z., Gu, M., Zhou, S., Chen, J., Tan, Q., Wang, H., & Bu, J. (2024). Towards a Unified Framework of Clustering-based Anomaly Detection. *arXiv preprint arXiv:2406.00452*.
24. Wei, J., Chen, P., Dai, J., Sun, X., Zhang, Z., Xu, C., & Wang, Y. (2025). HuntFUZZ: Enhancing error handling testing through clustering based fuzzing. *Journal of Computer Security*, *33*(5), 334-359.
25. Pei, H., Yin, B., Xie, M., & Cai, K. Y. (2021). Dynamic random testing with test case clustering and distance-based parameter adjustment. *Information and Software Technology*, *131*, 106470.
26. Karczmarek, P., Kiersztyn, A., Pedrycz, W., & Al, E. (2020). K-means-based isolation forest. *Knowledge-based systems*, *195*, 105659.
27. Cheng, M., Zhou, Y., & Xie, X. (2023, July). Behavexplor: Behavior diversity guided testing for autonomous driving systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 488-500).
28. Chen, C., Xu, H., & Cui, B. (2021). PSOFuzzer: a target-oriented software vulnerability detection technology based on particle swarm optimization. *Applied Sciences*, *11*(3), 1095.
29. Fu, Y. F., Lee, J., & Kim, T. (2023). autofz: Automated fuzzer composition at runtime. In *32nd USENIX Security Symposium (USENIX Security 23)* (pp. 1901-1918).
30. Zhang, Y., Zhang, S., Guo, C., Zhang, L., Sun, Y., & Huang, H. (2022, July). Optimization Design of Privacy Protection System Based on Cloud Native. In *International Conference on Artificial Intelligence and Security* (pp. 599-615). Cham: Springer International Publishing.
31. Klooster, T., Turkmen, F., Broenink, G., Ten Hove, R., & Böhme, M. (2023, May). Continuous fuzzing: a study of the effectiveness and scalability of fuzzing in CI/CD pipelines. In *2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT)* (pp. 25-32). IEEE.