engineering proceedings

MDPI

Proceeding Paper

State of Non-Intrusive Automated Testing in Android Applications: A Scoping Review of Data Loss and Vulnerability Detection †

Rubén Guerrero [1,2,*], Geraldine Morán [3], Adriana Guamialama [3], Cristhian Tepud [3] and Samuel Lascano [1,2,3,*]

[1] Research Department, State Polytechnic University of Carchi, Tulcán 040101, Ecuador
[2] Computer Science Career, State Polytechnic University of Carchi, Tulcán 040101, Ecuador
[3] International Research Network and Observatory of Emerging Technologies, Tena 220101, Ecuador; geraldine.moran@upec.edu.ec (G.M.); adriana.guamialama@upec.edu.ec (A.G.); cristhian.tepud@upec.edu.ec (C.T.)
* Correspondence: ruben.guerrero@upec.edu.ec (R.G.); samuel.lascano@upec.edu.ec (S.L.)
† Presented at the Sixth International Conference on Information Systems and Software Technologies (ICI2ST 2026), Quito, Ecuador, 23–25 September 2026.

Abstract

This scoping review analyzes non-intrusive automated testing techniques in Android applications for data loss and vulnerability detection, following the PRISMA-ScR protocol across 127 primary studies (2020–2026). Static analysis (20.5%) and dynamic analysis (15.0%) dominate the corpus; automated GUI testing (13.4%) and robotic testing (3.1%) represent emerging non-intrusive directions. Only 38.6% of studies simultaneously achieve full non-intrusiveness and complete automation. False positive rates are reported in 1.6% of studies; scalability and performance impact go unreported in 85.0% and 72.4%, respectively, revealing critical barriers to operational adoption.

Keywords: Android applications; automated testing; data loss detection; non-intrusive testing; vulnerability detection; scoping review; mobile security

1. Introduction

The sustained expansion of mobile devices has transformed everyday habits, positioning Android applications as essential tools for communication, commerce, banking, and healthcare, among other domains [1]. Android has consolidated its status as the dominant mobile operating system worldwide, increasing the need to assure application quality before deployment [2]. Automated testing has become a key practice for verifying correct software behavior and improving user experience, particularly in an ecosystem characterized by high device and configuration diversity [3].

One of the most frequent problems in Android application development is data loss, which occurs when execution is interrupted by system events such as screen rotation or incoming calls [4]. When application state is not correctly preserved during these events, user information may be irrecoverably lost, directly affecting software reliability [5]. These failures are especially difficult to detect because they manifest only under specific states and inputs [6].

Security represents an equally critical concern. Mobile devices store large volumes of personal information, making Android a frequent target for cyberattacks and malicious applications [7]. The increasing sophistication of malware has exposed the insufficiency of traditional detection methods, requiring the adoption of more advanced approaches capable of identifying vulnerabilities at early stages [8].

Check for updates

Academic Editor: Fredy Gavilanes

Published: 18 September 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

Several tools have been developed to address data loss detection automatically. The Data Loss Detector (DLD) integrates an exploration strategy with customized test oracles, detecting 75% of faults in a reference benchmark built from 110 real faults [9]. Similarly, iFixDataloss detects and automatically repairs data loss issues, generating patches accepted by developers in real applications [10].

Despite these advances, many existing solutions rely on semi-intrusive methods requiring source code access or system instrumentation, limiting their applicability [11]. Non-intrusive testing has emerged as a methodologically distinct paradigm that evaluates applications exclusively through external interfaces, without modifying source code, injecting instrumentation, or altering system configurations. This characteristic enables testing in closed commercial environments where internal access is restricted, expanding the practical applicability of automated quality assurance beyond controlled laboratory settings. Robotic GUI-based approaches have been proposed and extended [12,13] to operationalize this paradigm by interacting with the device interface as an end user would, producing black-box test scenarios that preserve the integrity of the production environment. Visual robotic systems have further extended these capabilities to platforms with customized operating systems [14], incorporating deep learning techniques to improve GUI recognition [15].

This scoping review addresses three objectives: (i) characterizing non-intrusive automation techniques applied to Android testing for data loss and vulnerability detection; (ii) evaluating the reported effectiveness of automated tools; and (iii) identifying the main limitations and challenges associated with their implementation.

The remainder of this article is organized as follows: Section 2 describes the methodological design and study selection criteria; Section 3 presents the results of data extraction and technique classification; Section 4 discusses findings on effectiveness and limitations of non-intrusive approaches; Section 5 presents conclusions and future research directions.

2. Materials and Methods

This study was conducted as a scoping review following the PRISMA-ScR protocol. Its objective was to identify, analyze, and synthesize scientific evidence related to non-intrusive automated testing techniques in Android applications, the application was tested on Android 10.0/13.0/14.0 (API level 29; Google LLC, Mountain View, CA, USA), focusing on data loss and security vulnerability detection. A documentary analysis was performed with quantitative and descriptive synthesis of indicators such as detection rates and test coverage, as well as an interpretive analysis of approaches and limitations reported in the literature. The unit of analysis corresponded to techniques, tools, and software agents used for non-intrusive detection in Android applications. Both journal articles and conference papers were included, considering the rapid evolution of the area.

2.1. Research Questions

The following research questions (RQs) were formulated to guide the review:

- RQ1: What non-intrusive automated testing techniques are used in Android applications to detect data loss and security vulnerabilities?
- RQ2: How effective are non-intrusive automated tools in identifying information leakage and security failures in Android mobile applications?
- RQ3: What are the main limitations and challenges in implementing non-intrusive automated testing for vulnerability detection in Android applications?

2.2. Search Strategy

Literature search was conducted across five digital libraries widely used in Software Engineering: IEEE Xplore, Scopus, ScienceDirect, SpringerLink, and Google Scholar. The

inclusion of Google Scholar was motivated by its broader indexing coverage of conference proceedings and workshop papers in software engineering, which are frequently absent from subscription-based databases; searches were performed with explicit query constraints (exclusion of preprints, theses, and non-peer-reviewed repositories) and documented filters (2020–2026) to maximize reproducibility. The search strategy was based on Boolean combinations of terms related to automated testing, Android applications, software agents, and security. Keywords included "automated testing", "test automation", "Android applications", "data loss", "vulnerability detection", "non-intrusive testing", and "software agents", combined using AND and OR logical operators. The search process followed four main stages: (i) record identification in selected databases, (ii) duplicate removal using Zotero, (iii) title and abstract screening, and (iv) full-text eligibility assessment. Detailed search strategies and record counts per database are presented in Table 1.

Table 1. Search strategy and retrieval results for each database.

Database	Search Strategy (Keywords)	Initial	Dedup.
IEEE Xplore	"automated testing" AND Android	469	33
Google Scholar	"Android" "data leakage" "automated testing" "vulnerability" -arxiv.org -preprint -diva-portal.org -tesis -thesis (filters: 2020–2026)	93	50
Scopus	("Android" AND "data leakage") (filters: 2020–2026)	167	70
ScienceDirect	("Android" OR "mobile application") AND ("security testing" OR "vulnerability detection")	469	119
SpringerLink	("software testing" OR "test automation") AND (Android OR "mobile applications") (filters: 2020–2026, Computer Science)	1021	153
Total		2219	425

Note. Deduplication values correspond to internal deduplication within each database. Cross-database deduplication was performed subsequently during the PRISMA identification stage, yielding 383 unique records after removal of 42 duplicates across databases (see Figure 1). Google Scholar results were constrained to peer-reviewed sources by excluding preprint servers, institutional repositories, and thesis archives through explicit query operators. The 2219 figure represents the total raw retrieval across all databases before any deduplication. The 425 figure reflects records remaining after internal deduplication within each database. Cross database deduplication subsequently reduced this to 383 unique records.

2.3. Inclusion and Exclusion Criteria

The following inclusion criteria were established to ensure quality, technical relevance, and scientific validity:

- Publications from 2020 to 2026.
- Scientific studies, conference articles, and reviews focused on automated testing in Android applications.
- Research addressing non-intrusive testing techniques or automated testing approaches.
- Works incorporating software agents or methods for data loss and security vulnerability detection.
- Publications in English.

The following exclusion criteria were applied:

- Publications prior to January 2020.
- Studies unrelated to Android applications (e.g., iOS, web, or desktop systems).
- Works without a focus on software testing automation.

- Documents without a verifiable technical contribution, without defined methodology, or without full-text access.

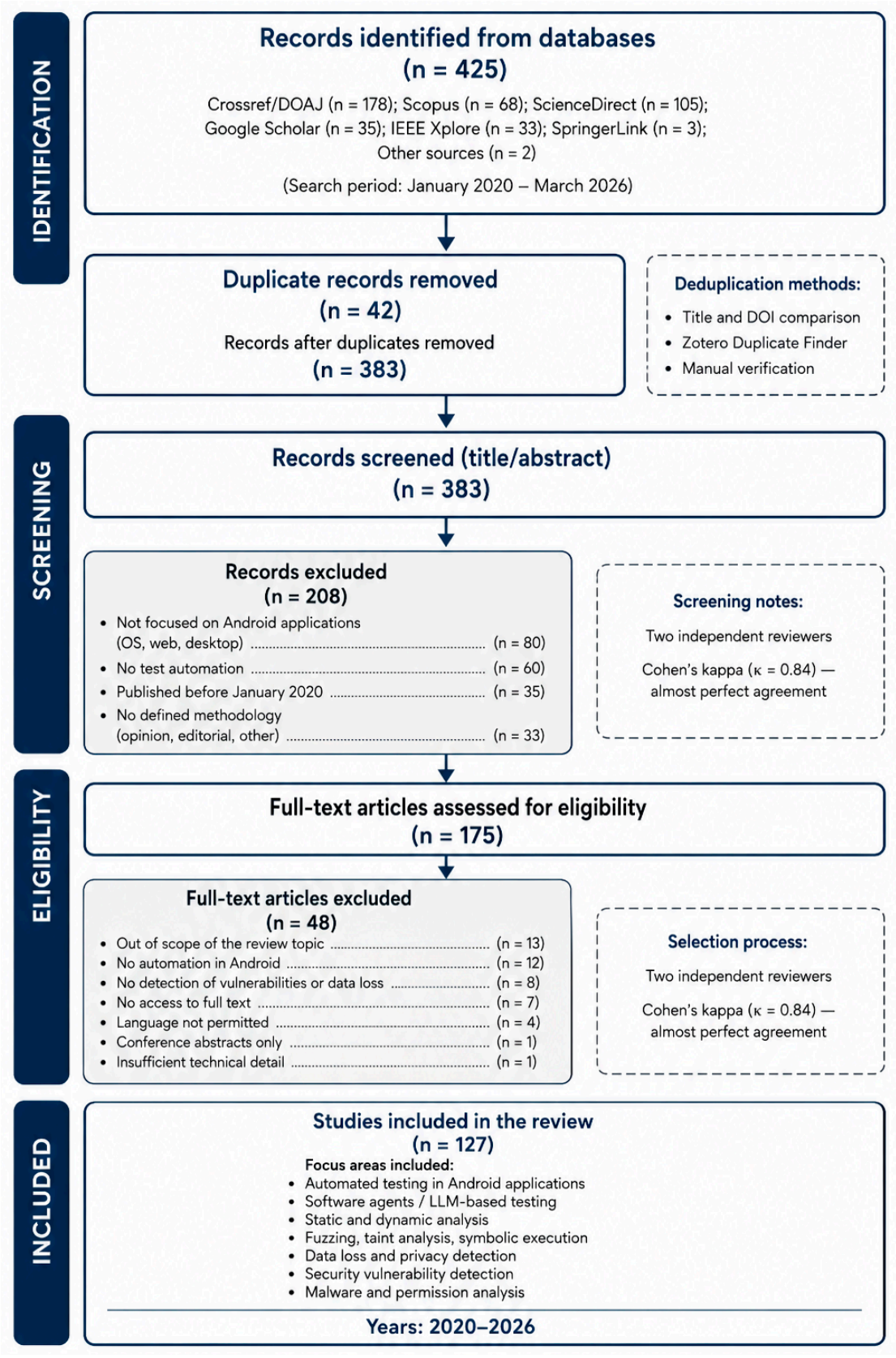


Figure 1. PRISMA-ScR flowchart of the study search and selection process.

The temporal restriction of 2020–2026 applies exclusively to the primary studies included in the scoping review corpus. References to works published prior to 2020 appear solely in the theoretical framework and introduction sections to provide foundational context for the reviewed techniques and tools.

2.4. Operational Definition of Intrusion Levels

Since the literature does not provide universally accepted definitions of intrusion levels in Android testing, this review adopts operational definitions to ensure consistent classification across studies.

A non-intrusive approach was defined as any testing technique capable of evaluating Android applications exclusively through externally observable interfaces (GUI interactions, visual recognition, runtime observation, network monitoring, or external sensors) without modifying the application binary, injecting instrumentation, requiring access to source code, or altering the operating system.

A semi-intrusive approach refers to techniques that require limited intervention, such as APK repackaging, bytecode instrumentation, debugging interfaces, or temporary runtime hooks, while preserving most application behavior.

An intrusive approach includes methods that require source-code modification, extensive instrumentation, operating-system customization, or invasive runtime alterations that significantly modify the execution environment.

2.5. Study Selection and Data Extraction

The document selection process followed four stages aligned with the PRISMA-ScR protocol: identification through execution of search strings in selected databases, duplicate removal, initial screening by title and abstract, and eligibility assessment through full-text reading. To systematize information from included studies ($n = 127$), a data extraction matrix was designed with variables oriented to characterize reported approaches: testing technique, tool employed, intrusion level and automation degree, vulnerability type, detection rate, false positives, benchmark used, scalability, performance impact, reported limitations, and suitability for non-intrusive testing.

To ensure classification reliability, title and abstract screening was performed independently by two reviewers (R.G. and G.M.), with disagreements resolved through discussion. Full-text eligibility assessment involved a third reviewer (S.L.) when consensus was not reached by the primary reviewers. Data extraction from the 127 included studies was distributed across all authors and cross-checked on a 20% random sample. Observed agreement on technique classification and intrusion level reached 89.0%, with a Cohen's kappa coefficient of $\kappa = 0.84$, indicating almost perfect agreement prior to final consensus.

2.6. Threats to Validity

Search completeness. Five databases were queried; however, gray literature, doctoral theses, and technical reports were excluded by design. Relevant evidence produced outside indexed academic channels may remain unrepresented in this synthesis.

Publication bias. The corpus is subject to the positive-results bias characteristic of software engineering literature, in which studies reporting successful detection rates are more frequently submitted and accepted than those reporting failures or null results. This limits the representativeness of the reported effectiveness figures.

Classification reliability. Intrusion level and automation degree were inferred from reported descriptions when the authors did not explicitly declare them. Studies lacking methodological transparency (18.9% for intrusion level; 20.5% for automation degree) were

classified as "not specified" rather than assigned to a category, introducing a conservative bias in the synthesis.

Temporal scope. The 2020–2026 boundary captures recent developments but excludes foundational work published before 2020 that continues to influence current tools and frameworks.

Generalizability. All included studies target the Android platform exclusively. Findings do not transfer directly to iOS, cross-platform frameworks (Flutter, React Native), or embedded systems running Android-derived operating systems.

3. Results

3.1. Search Results

The identification, screening, and eligibility process is summarized in the PRISMA flow diagram (Figure 1). In the identification stage, 425 records were retrieved from five databases; after cross-database deduplication using Zotero, 383 unique records were obtained. During screening, title and abstract review led to the exclusion of 208 records for not meeting inclusion criteria, resulting in 175 articles for full-text evaluation. In the eligibility stage, a full reading of 175 articles led to the exclusion of 48 documents for not being related to Android applications, not addressing automated testing or security, being outside the thematic scope, or lacking full-text access. Consequently, 127 studies met all established criteria and were incorporated into the documentary analysis and thematic synthesis. In accordance with PRISMA-ScR guidelines, no formal methodological quality assessment was performed, given that the purpose of this review is to identify the state of the art and existing gaps, rather than to compare statistical results. The relevance of selected studies was guaranteed through rigorous application of the defined inclusion and exclusion criteria.

3.2. Classification and Characterization of Studies

A data extraction matrix was designed to capture and classify the most relevant information from the 127 included studies across four dimensions: testing techniques, intrusion and automation levels, detected problem types and evidence quality, and scalability and performance reporting. Results are presented with their respective synthesis tables, enabling progressive and articulated responses to the three research questions.

3.2.1. Testing Techniques

Table 2 summarizes the distribution of primary testing approaches across the 125 studies with an identifiable technique (2 of the 127 included studies could not be classified due to insufficient methodological description). Static analysis accounts for the largest proportion (n = 26, 20.5%), followed by dynamic analysis (n = 19, 15.0%) and automated GUI testing (n = 17, 13.4%). Hybrid approaches combining static and dynamic analysis represent 10.2% of the corpus (n = 13). Emerging techniques—including robotic GUI testing (3.1%), machine learning-based approaches (2.4%), and formal verification (2.4%)—appear in a minority of studies. Twelve studies (9.4%) do not specify a primary technique, reflecting terminological heterogeneity across the literature.

A temporal inspection of the included studies suggests a gradual transition from predominantly static-analysis approaches during the early years of the review period (2020–2022) toward increasing adoption of GUI-based automation, robotic testing, and machine-learning-assisted techniques after 2023. Although static analysis remains the most frequently reported methodology, more recent publications increasingly emphasize practical deployment through black-box interaction and visual testing, indicating a shift toward operational applicability rather than purely analytical detection.

Table 2. Test automation techniques and analysis approaches identified in the ScR (multiple techniques per study allowed).

Approach/Primary Technique	*n*	%
Static analysis	26	20.5
Dynamic analysis	19	15.0
Automated GUI testing	17	13.4
Hybrid approaches (static + dynamic)	13	10.2
Not specified/other	12	9.4
Systematic reviews/surveys	10	7.9
Dynamic malware analysis	7	5.5
Fuzzing	6	4.7
Flow analysis/taint analysis	5	3.9
Robotic GUI testing	4	3.1
Formal verification/model-based	3	2.4
AI/Machine Learning	3	2.4
Total	125	98.4

Note. Of the 127 included studies, 2 could not be assigned to any primary technique category due to insufficient methodological description and are therefore excluded from this table. Some studies employ more than one methodological approach; consequently, frequencies may exceed 125. Percentages are calculated over the 125 studies with an identifiable primary technique. The sum of percentages does not reach 100% due to the two unclassified studies.

3.2.2. Intrusion and Automation Levels

Non-intrusive approaches appear in 57.5% of studies, while 52.8% report complete automation. The combined category simultaneously fully non-intrusive and fully automated—reaches only 38.6%, representing the maximum operational coverage observed in this corpus. An additional 18.9% of studies are non-intrusive but semi- automated, retaining partial operational dependency. The presence of 18.9% of studies not specifying their intrusion level and 20.5% not declaring their automation degree introduces methodological ambiguity that limits direct comparison between proposals. As summarized in Table 3, the non-intrusive suitability classification further shows that 38.6% of the studies achieve a high level of suitability, corresponding to approaches that are both fully non-intrusive and automated, whereas 18.9% are classified as medium-high because they are non-intrusive but retain some degree of operational intervention. Conversely, 16.5% of studies cannot be reliably classified due to insufficient information, while 15.7% are semi-intrusive and 7.9% require system or application modification, indicating progressively greater implementation constraints.

Table 3. Intrusion level, automation degree, and suitability for non-intrusive testing.

Indicator	Result	*n*	%
Intrusion level	Non-intrusive	73	57.5
Intrusion level	Semi-intrusive	20	15.7
Intrusion level	Intrusive	10	7.9
Intrusion level	Not specified	24	18.9
Automation degree	Fully automated	67	52.8
Automation degree	Semi-automated	34	26.8
Automation degree	Not specified	26	20.5
Non-intrusive suitability	High: fully non-intrusive and automated	49	38.6
Non-intrusive suitability	Medium-high: non-intrusive but semi-automated	24	18.9
Non-intrusive suitability	Undeterminable	21	16.5
Non-intrusive suitability	Medium: semi-intrusive	20	15.7
Non-intrusive suitability	Low: requires system/app modification	10	7.9

Note. Percentages are calculated over the total of 127 included articles.

3.2.3. Detected Problems and Evidence Quality

Data leakage and privacy issues constitute the dominant thematic focus (40.2%), followed by security vulnerabilities (15.7%) and malware detection (13.4%). Regarding evidence quality, 42.5% of studies present low quantitative evidence, while only 25.2% offer high-level evidence based on quantitative metrics and reproducible benchmarks. False positive rates appear in only 1.6% of the corpus, and detection rate metrics appear in 29.9%. As shown in Table 4, although 52.0% of studies report using benchmarks or datasets, only a quarter provide high-quality quantitative evidence, highlighting a persistent gap in rigorous and reproducible evaluation.

Table 4. Detected problems and evidence quality reported in studies.

Primary Result	*n*	%
Data loss/privacy	51	40.2
Security vulnerabilities	20	15.7
Malware/malicious behavior	17	13.4
Functional failures/GUI	17	13.4
Not specified	13	10.2
High evidence: metrics + benchmark	32	25.2
Medium evidence: partial evidence	41	32.3
Low evidence: no quantitative metrics	54	42.5
Studies reporting detection rate/accuracy	38	29.9
Studies reporting benchmark/dataset	66	52.0
Studies reporting false positives	2	1.6

Note. Percentages are calculated based on the total of 127 articles included. Primary result categories are not mutually exclusive; some studies address more than one problem type, and percentages do not sum to 100%.

3.2.4. Limitations, Scalability, and Performance

Scalability is unreported in 85.0% of studies, and performance impact in 72.4%. Dataset size and coverage represent the most frequent explicitly reported limitation (21.3%), followed by platform compatibility issues derived from Android ecosystem fragmentation. Only 33.9% of studies declare future work lines. As shown in Table 5, the high proportion of unreported scalability and performance indicators highlights a significant gap in assessing the practical deployment and long-term applicability of the proposed approaches.

Table 5. Limitations, scalability, and performance impact reported by studies.

Indicator	Result	*n*	%
Primary limitation	Not reported	41	32.3
Primary limitation	Dataset size/coverage	27	21.3
Primary limitation	Other reported limitations	24	18.9
Primary limitation	Dataset + platform compatibility	13	10.2
Primary limitation	Platform compatibility	5	3.9
Primary limitation	Scalability/performance	4	3.1
Scalability	Positively reported	9	7.1
Scalability	Partial	9	7.1
Scalability	Not reported	108	85.0
Performance impact	Reported	35	27.6
Performance impact	Not reported	92	72.4
Future work	Reported	43	33.9
Future work	Not reported	84	66.1

3.3. Comparative Tool Analysis

Table 6 compares representative tools based on technique, intrusion level, automation, detection performance, benchmarks, and scalability. Most tools offer full automation and are based on real-world applications or benchmark datasets, while non-intrusive approaches

generally demonstrate greater operational suitability. However, detection rates are explicitly reported for only a small subset of tools, and false positive rates are mostly unreported. Scalability is also limited or only partially demonstrated in most cases. Overall, Table 6 highlights a trade-off between non-intrusive operation and the availability of rigorous evidence of performance and scalability, with only a few tools combining full automation, quantitative detection results, benchmark-based evaluation, and practical scalability.

Table 6. Comparative analysis of representative tools identified in the ScR.

Tool/ Framework	Technique	Intrusion Level	Automation	Detection Rate	False Positive Rate	Benchmark Used	Scalability
DLD	Dynamic + GUI	Non-intrusive	Full	75% (110 faults)	Not reported	Custom (110 faults)	No
iFixDataloss	Static + Dynamic	Semi-intrusive	Full	Not specified	Not reported	Real apps	No
RoScript	Robotic GUI	Non-intrusive	Full	Not specified	Not reported	Touchscreen apps	Partial
R-DLD	Robotic GUI + Vision	Non-intrusive	Full	100% (30 apps, controlled)	Not reported	Benchmark apps	Partial
FlowDroid	Static taint analysis	Intrusive	Full	~93% (DroidBench)	Low (varies)	DroidBench	Partial
VLM-Fuzz	GUI Fuzzing + LLM	Non-intrusive	Full	Not specified	Not reported	Real apps	Partial
DroidRista	Static data flow	Semi-intrusive	Full	Not specified	Not reported	DroidBench + real apps	Partial
DroidHook	Dynamic sandbox	Semi-intrusive	Full	Not specified	Not reported	Real apps	No
ArgusDroid	Static (permission-API graph)	Non-intrusive	Full	Not specified	Not reported	Real apps	Partial
MobiLogLeak	Taint flow analysis	Semi-intrusive	Full	Not specified	Low	5000 real apps	Partial

Note. Data extracted from the 127 included studies. Tools were selected to represent each primary technique category identified in Table 2: data loss detection (DLD, iFixDataloss), robotic GUI testing (RoScript, R-DLD), static taint analysis (FlowDroid, DroidRista, MobiLogLeak), GUI fuzzing (VLM-Fuzz), dynamic sandbox analysis (DroidHook), and permission-based static analysis (ArgusDroid). R-DLD's 100% detection rate was obtained under controlled benchmark conditions and should not be interpreted as generalizable to production environments.

3.4. Thematic Synthesis and Research Questions

3.4.1. Testing Techniques (RQ1)

The most frequently employed techniques are static analysis (20.5%) and dynamic analysis (15.0%), both of which inspect or monitor applications without necessitating user-level interaction. Among non-intrusive approaches specifically, automated GUI testing (13.4%) and robotic testing (3.1%) represent the primary methodological directions. Only 38.6% of the corpus achieves the combination of full non-intrusiveness and complete automation that constitutes the operational target of this review. This figure establishes the current ceiling of the field: the majority of proposed solutions satisfy one criterion but not both simultaneously.

3.4.2. Effectiveness of Non-Intrusive Tools (RQ2)

The available evidence does not permit the identification of a dominant technique in terms of effectiveness, and this limitation is itself a substantive finding. Of the 127 included studies, only 29.9% report detection rate or accuracy metrics, and a mere 1.6% disclose false positive rates. Any effectiveness claim constructed on this corpus describes the behavior of a self-selected minority of studies whose reporting practices are unrepresentative of the field as a whole. Within the subset presenting high-level quantitative evidence (25.2%), hybrid static-dynamic approaches consistently report broader detection coverage, and automated

GUI testing demonstrates positive trends in black-box applicability. These conclusions, however, carry a structural caveat: studies that report metrics are disproportionately those with favorable results, which means the 70.1% of studies omitting detection figures are not neutral absences—they represent an absence of data that may introduce optimistic bias in any aggregated performance assessment. A literature in which three out of four studies cannot be compared quantitatively does not support effectiveness rankings; it severely limits the reliability of any effectiveness ranking derived from this corpus. The near-total absence of false positive reporting (1.6%) compounds this problem: detection capability without false positive characterization is an incomplete metric, as a tool generating unconstrained false alerts degrades security team performance through alert fatigue regardless of its detection rate. The systematic absence of false positive reporting limits the extent to which reviewed proposals can be evaluated for responsible operational deployment.

Consequently, comparative conclusions regarding the superiority of specific testing approaches should be interpreted as qualitative trends rather than evidence-based rankings, as the underlying reporting practices are highly heterogeneous and incomplete.

3.4.3. Limitations and Challenges (RQ3)

RQ3 is directly answered by the limitations landscape summarized in Section 3.2.4 (Table 5): dataset coverage, platform fragmentation, and the systematic non-reporting of scalability and performance data constitute the field's principal implementation barriers (Figure 2). Beyond these reporting gaps, a further structural challenge emerges, with 32.3% of studies omitting stated limitations and 18.9% omitting intrusion-level classification; thus, the corpus does not currently support direct cross-study comparison. This comparability gap is itself a primary practical obstacle facing adopters of non-intrusive testing.

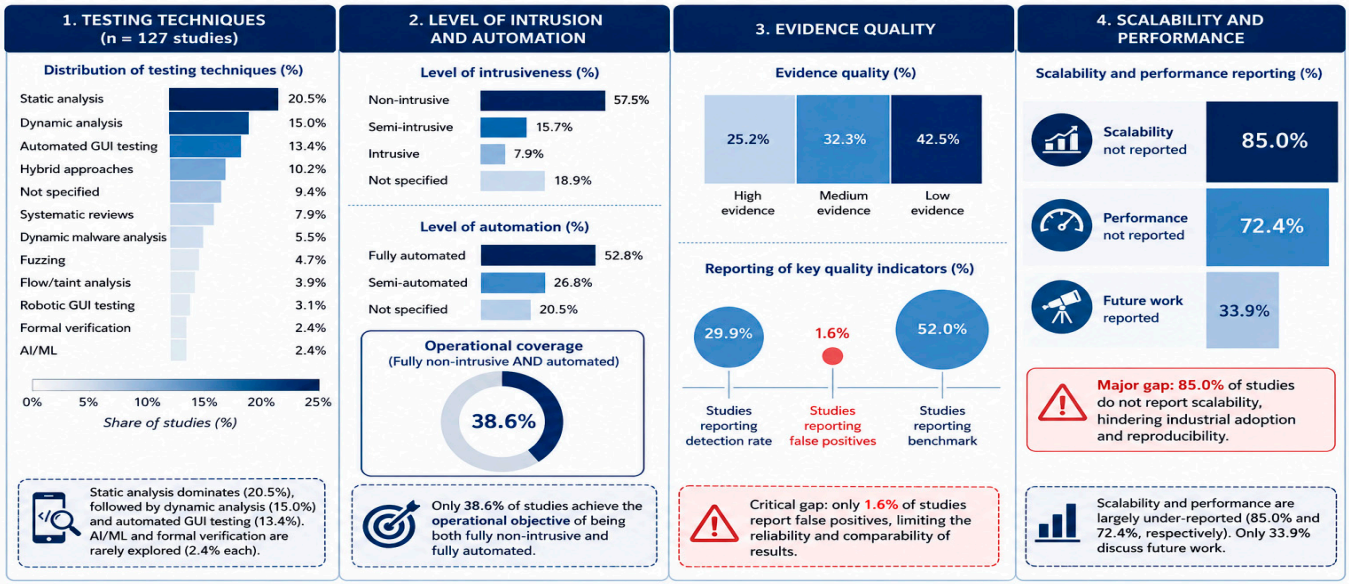


Figure 2. Multidimensional synthesis of the scoping review findings across four analytical dimensions: testing techniques distribution ($n = 127$ studies), intrusion and automation levels, evidence quality indicators, and scalability and performance reporting. Percentages are calculated based on the total number of included studies. The operational coverage indicator (38.6%) represents the subset of studies simultaneously achieving full non-intrusiveness and complete automation.

4. Discussion

The comparison highlights that, unlike previous studies, the present review jointly evaluates non-intrusiveness, automation, scalability, and evidence quality, thereby provid-

ing a more comprehensive analytical perspective on automated Android testing (Table 7).

Table 7. Comparison with previous literature reviews.

Review	Coverage	Non-Intrusive Focus	Automation Analysis	Scalability	Evidence Quality
[3]	Android testing	No	Partial	No	No
[7]	Android security	No	No	No	No
[14]	GUI testing	Partial	No	No	No
This review	Android testing + security	✓	✓	✓	✓

The volume of production in Android testing research contrasts with the field's limited empirical maturity. A corpus of 127 studies published within six years signals active scientific interest; the dominance of low-evidence studies (42.5%) signals that novelty is prioritized over validation. This pattern recurs across software engineering subfields and has been documented in systematic mapping studies of test automation more broadly [3]. The present review confirms that Android testing reproduces this structural tendency rather than departing from it.

Unlike previous reviews that primarily focus on Android malware detection, automated GUI testing, or software testing frameworks independently, the present scoping review integrates these research directions under the common perspective of non-intrusive automated testing. This broader perspective enables comparison between static, dynamic, GUI-based, robotic, and hybrid approaches using a unified analytical framework based on intrusion level, automation degree, evidence quality, scalability, and operational applicability.

One explanation for the dominance of static analysis may reflect, at least in part, the availability of established infrastructure and documented benchmarks, rather than demonstrated technical superiority alone. Established tools such as FlowDroid, Amandroid, and MalloDroid provide ready-made infrastructure, documented benchmarks (DroidBench, IccTA), and established evaluation protocols that reduce the entry barrier for new studies. Dynamic approaches and GUI testing, despite their higher ecological validity, require device farms, emulator configurations, and interaction scripting that represent non-trivial setup costs. Robotic testing adds physical infrastructure requirements. The observed distribution therefore encodes resource constraints and community conventions, not only technical merit.

The 18.9-percentage-point gap between studies classified as non-intrusive (57.5%) and those achieving full non-intrusiveness combined with complete automation (38.6%) constitutes the central structural finding of this review. Achieving non-intrusiveness without full automation produces a solution that is reproducible in principle but operationally dependent: a human operator must intervene at some stage, which limits deployment in continuous integration pipelines and large-scale testing environments. Conversely, full automation without non-intrusiveness enables scalability at the cost of applicability in closed or commercial environments. The combination of both properties in a single framework remains the field's unresolved design challenge.

Building on the effectiveness limitations discussed in Section 3.4.2, standardized false-positive reporting should become a minimum requirement in future empirical evaluations of non-intrusive testing tools, given its direct bearing on operational cost and alert-fatigue risk.

From a practical perspective of scalability substantially restricts the practical interpretation of existing proposals. Without evaluations across diverse devices, Android versions, and deployment conditions, the external validity of many reported solutions remains uncertain.

5. Conclusions

This scoping review mapped 127 primary studies on non-intrusive automated testing in Android applications published between 2020 and 2026, organized around four analytical dimensions: testing techniques, intrusion and automation levels, detected problem types and evidence quality, and scalability and performance reporting.

Finding 1—Incomplete paradigm consolidation. Non-intrusive approaches appear in 57.5% of the corpus, confirming a directional trend. The subset achieving simultaneous full non-intrusiveness and complete automation reaches only 38.6%, establishing the current operational ceiling of the field and revealing a persistent design gap between methodological intent and practical deployability.

Finding 2—Technical plurality without methodological convergence. Static analysis (20.5%) and dynamic analysis (15.0%) remain structurally dominant. Automated GUI testing (13.4%), robotic testing (3.1%), and machine learning-based approaches (2.4%) gain ground progressively. No single technique dominates in terms of both frequency and empirical rigor, placing the field in an active but unresolved transition phase.

Finding 3—Critical empirical deficit. High-quality quantitative evidence appears in 25.2% of studies, and false positive rates appear in just 1.6%—the central empirical gap identified in this review (see Sections 3.4.2 and 4).

Finding 4—Scalability and performance as unresolved debts. Scalability evaluation is absent in 85.0% of studies; performance impact in 72.4%—the field's largest unaddressed evaluation gap (see Section 4).

Priority research directions include: (1) standardized reporting frameworks mandating precision, recall, F1-score, false positive rate, and computational overhead as minimum metrics; (2) public benchmark datasets representative of Android's real-world application distribution across categories, API levels, and vulnerability types; (3) hybrid frameworks natively integrating visual GUI testing, runtime behavioral analysis, and machine learning within a fully non-intrusive, fully automated pipeline; and (4) controlled comparative experiments evaluating robotic non-intrusive testing against hybrid static-dynamic approaches on a shared benchmark with unified metrics.

Author Contributions: Conceptualization, R.G. and G.M.; methodology, R.G. and G.M.; software, R.G.; validation, S.L.; formal analysis, A.G. and C.T.; investigation, A.G. and C.T.; data curation, A.G. and C.T.; writing—original draft preparation, R.G. and G.M.; writing—review and editing, S.L.; supervision, S.L.; project administration, S.L.; funding acquisition, S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external or project-specific funding, and although the Universidad Politécnica Estatal del Carchi (UPEC), Ecuador, provides institutional support for publication costs as a benefit for its academic staff, this support was not applied to the present manuscript, therefore no grant or funding number is associated with this work.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data supporting the results of this study are available at: https://doi.org/10.5281/zenodo.20072778 (accessed on 29 May 2026).

Acknowledgments: The authors would like to thank the International Research Network and Observatory of Emerging Technologies for their support in the development of this scoping review. During the preparation of this manuscript, the authors used Claude 3.5 (Anthropic) to assist in linguistic refinement, structural organization, and formatting of LaTeX tables and citations. The authors reviewed and edited the output and took full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

PRISMA-ScR	Preferred Reporting Items for Systematic Reviews and Meta-Analyses extension for Scoping Reviews
GUI	Graphical User Interface
RQ	Research Question
ScR	Scoping Review
DLD	Data Loss Detector
AI	Artificial Intelligence

References

1. Baride, S.; Dutta, K. A cloud based software testing paradigm for mobile applications. *SIGSOFT Softw. Eng. Notes* **2011**, *36*, 1–4. [CrossRef]
2. Riccio, V.; Amalfitano, D.; Fasolino, A.R. Is this the lifecycle we really want? an automated black-box testing approach for Android activities. In *Companion Proceedings for the ISSTA/ECOOP 2018 Workshops*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 68–77. [CrossRef]
3. Kong, P.; Li, L.; Gao, J.; Liu, K.; Bissyandé, T.F.; Klein, J. Automated Testing of Android Apps: A Systematic Literature Review. *IEEE Trans. Reliab.* **2019**, *68*, 45–66. [CrossRef]
4. Amalfitano, D.; Amatucci, N.; Memon, A.M.; Tramontana, P.; Fasolino, A.R. A general framework for comparing automatic testing techniques of Android mobile apps. *J. Syst. Softw.* **2017**, *125*, 322–343. [CrossRef]
5. Konduru, A.; Shan, Z. Automated Testing to Detect Status Data Loss in Android Applications. In *2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA)*; IEEE: New York, NY, USA, 2022; pp. 155–160. [CrossRef]
6. Zhou, Y.; Song, W. DDLDroid: Efficiently Detecting Data Loss Issues in Android Apps. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*; Association for Computing Machinery: New York, NY, USA, 2023; pp. 703–714. [CrossRef]
7. Garg, S.; Baliyan, N. Android security assessment: A review, taxonomy and research gap study. *Comput. Secur.* **2021**, *100*, 102087. [CrossRef]
8. Palutla, D.V.; Bojjagani, S.; Mula, S.C.R.; Uyyala, R.; Sharma, N.K.; Morampudi, M.K.; Khan, M.K. Unveiling Android security testing: A Comprehensive overview of techniques, challenges, and mitigation strategies. *Comput. Electr. Eng.* **2025**, *127*, 110620. [CrossRef]
9. Riganelli, O.; Mottadelli, S.P.; Rota, C.; Micucci, D.; Mariani, L. Data loss detector: Automatically revealing data loss bugs in Android apps. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*; Association for Computing Machinery: New York, NY, USA, 2020; pp. 141–152. [CrossRef]
10. Guo, W.; Dong, Z.; Shen, L.; Tian, W.; Su, T.; Peng, X. iFixDataloss: A tool for detecting and fixing data loss issues in Android apps. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*; Association for Computing Machinery: New York, NY, USA, 2022; pp. 785–788. [CrossRef]
11. Freitas, D.; Miranda, B.; Iyoda, J. Toward a non-invasive robotic-supported approach for data loss detection in android applications. *Inf. Softw. Technol.* **2026**, *194*, 108072. [CrossRef]
12. Mao, K.; Harman, M.; Jia, Y. Robotic Testing of Mobile Apps for Truly Black-Box Automation. *IEEE Softw.* **2017**, *34*, 11–16. [CrossRef]
13. Qian, J.; Lv, G.; Jin, Y.; Shang, Z.; Yan, S.; Wang, Y.; Chen, L. Robotic Visual GUI Testing for Truly Non-Intrusive Test Automation of Touch Screen Applications. *IEEE Trans. Softw. Eng.* **2025**, *51*, 1205–1231. [CrossRef]
14. Yu, S.; Fang, C.; Tuo, Z.; Zhang, Q.; Chen, C.; Chen, Z.; Su, Z. Vision-Based Mobile App GUI Testing: A Survey. *ACM Comput. Surv.* **2025**, *58*, 1–46. [CrossRef]
15. Zhang, M.; Yu, J.; Xu, C.; Li, J.; Yin, X.; Liu, H. Towards Testing the Accessibility of Dynamic Visual Changes in Android Mobile GUI with Multi-Modal LLMs. *ACM Trans. Comput. Hum. Interact.* **2026**, *33*, 1–32. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.