International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V74I9P101

Volume 74 Issue 9, 1-15, September 2026

Review Article

A Critical Review of Android Malware Detection Techniques (2014–2026): Evaluating Static, Dynamic, Hybrid, and Deep Learning Approaches

Rishish Jain[1], D. A. Mehta[2]

[1,2]*Department of Computer Engineering, SGSITS, Indore, Madhya Pradesh, India.*

Corresponding Author : rishishjain2009@gmail.com

Received: 09 July 2026 Revised: 17 August 2026 Accepted: 02 September 2026 Published: 18 September 2026

Abstract - *Over the past decade, Android malware detection has advanced a great deal, yet a persistent gap remains between the accuracy figures reported on benchmarks and what these systems actually achieve once deployed. This paper reviews thirty-one Android malware detection systems and reputation-intelligence platforms published between 2014 and 2026, spanning static, dynamic, hybrid, and deep-learning approaches, and compares them against six practical criteria: detection accuracy, obfuscation resilience, computational overhead, scalability, temporal robustness, and practical deployability. Benchmark accuracy rose from around 94% for Drebin to 98.1% for SeGDroid over this period, but Pendlebury et al.'s TESSERACT framework shows that a large share of these reported numbers is inflated by temporal and spatial evaluation bias. The paper also looks at reputation-based platforms such as VirusTotal, MetaDefender, Hybrid Analysis, and ANY.RUN, Intezer Analyze, MalwareBazaar, and AndroZoo, comparing what each contributes to Android malware research and dataset construction. Drawing on this review, a generalized multi-stage detection pipeline is proposed, and the paper closes by outlining the research gaps that should guide future work on both the research and deployment sides of Android malware detection.*

Keywords - *Android Malware, Deep Learning, Machine Learning, Mobile Security, Static Analysis.*

1. Introduction

A high benchmark number does not by itself mean a security problem has been solved. What matters is whether the claimed performance holds up once the system leaves the lab, and in Android malware detection, that gap has stayed wide for years. Hundreds of papers over the last decade have reported classification accuracies that keep climbing, yet malware keeps growing at scale. More than 3.8 million new malicious Android applications were detected in 2023 alone [2].

Android carried more than three billion active devices in 2024 [1], making it by far the largest target for mobile threat actors. Its own openness works against it. Sideloading from third-party stores is allowed, and developers get API-level access to sensitive device resources, so no single defensive layer has been able to close the ecosystem off completely.

The field's own history illustrates the point. Drebin [3] reported 94% accuracy with a linear SVM in 2014. A decade later, SeGDroid [4] reported 98.1% using a sensitive Function Call Graph (FCG) neural network. On paper, this looks like steady progress. TESSERACT [5], however, found that when Drebin is re-evaluated under temporally consistent conditions, its accuracy falls to 65–75%, and similar drops recur throughout the field. The field has clearly moved forward, but none of the systems reviewed here is simultaneously accurate, obfuscation-resistant, computationally cheap, temporally stable, deployable, and explainable. Addressing that combination of shortfalls is the motivation for this review. The sections that follow look systematically across existing approaches, weigh their strengths against their limitations, and set out where the field still needs to go.

The rest of the paper is organised as follows: Section 2 sets out the evaluation framework; Sections 3–6 review static, dynamic, hybrid, and deep-learning approaches in turn; Section 7 looks at reputation-based intelligence platforms; Section 8 discusses cross-cutting issues; Section 9 critically examines the generalised detection pipeline; Section 10 gives the comprehensive comparison tables; Section 11 sets out the research gaps; and Section 12 concludes.

2. Review Methodology and Comparison Criteria

2.1. Search Strategy and Study Selection

This review follows a structured, though not formally registered, literature-identification process rather than an ad-hoc reading list. Candidate studies were identified through keyword searches of IEEE Xplore, the ACM Digital Library, ScienceDirect, SpringerLink, and arXiv, using combinations of the terms "Android malware detection," "static analysis," "dynamic analysis," "machine learning," "deep learning," "graph neural network," and "temporal robustness," together with forward and backward citation chasing from the more heavily cited papers in each cluster. The search window ran from January 2014, when Drebin [3] and FlowDroid [7] established the field's two dominant early paradigms, to March 2026. A candidate study was included if it (i) proposed or evaluated a system, framework, or platform for detecting Android malware; (ii) reported at least one quantitative performance figure (accuracy, F1-score, true-positive rate, or an equivalent metric) on a named dataset; and (iii) appeared in a peer-reviewed venue, an established conference proceedings, or, for the small number of 2025–2026 systems still moving through peer review, a citable preprint from a research group with a verifiable institutional affiliation. Studies were excluded if they addressed malware detection on platforms other than Android, focused solely on iOS or desktop threats, or provided no measurable evaluation. Screening titles and abstracts against these criteria, followed by a full-text read of the retained set, converged on the thirty-one detection systems and reputation-intelligence platforms discussed in Sections 3–7 — twenty-three detection systems tabulated in Table 2 and eight reputation-intelligence platforms tabulated in Table 3; every system named in the text also appears in Table 2 and the reference list, so the count is traceable rather than asserted. This is an assessment of selected representative studies spanning the field's major methodological families, not an exhaustive systematic review in the PRISMA sense — no formal registration, dual-reviewer screening, or inter-rater agreement statistics were used. That scope is appropriate for a critical, practitioner-oriented synthesis of a decade of work, but it is a real limitation, and it is stated here explicitly rather than implied by the word "review" in the title.

2.2. Comparison Criteria and Rating Thresholds

The thirty-one systems and platforms surveyed here are compared against six practical criteria, summarised in Table 1, chosen to reflect real deployment requirements rather than idealised benchmark conditions. Each system is rated Strong (S), Moderate (M), or Weak (W) on each criterion, based on the published evidence and, where available, independent replications.

The Strong / Moderate / Weak labels used throughout Sections 3–10 are assigned against the following approximate thresholds rather than applied impressionistically. For Detection Accuracy (DA): Strong = reported accuracy or F1 ≥ 97% AND, where a TESSERACT-style temporally consistent evaluation exists, a corresponding figure ≥ 85%; Moderate = 90–96.9% under standard evaluation, or a documented drop of 10–25 points under temporal evaluation; Weak = below 90%, or a documented temporal drop exceeding 25 points (as with Drebin [3], [5]). For Obfuscation Resilience (OR): Strong = evaluated against at least two obfuscation classes (renaming, control-flow, encryption, reflection) with under 5 points of accuracy loss; Moderate = evaluated against one obfuscation class or with 5–15 points of loss; Weak = not evaluated under obfuscation, or loss exceeding 15 points (e.g., Rastogi et al.'s 34-point drop for Drebin [6]). For Computational Overhead (CO): Strong = sub-second to low-seconds per-application analysis time reported; Moderate = tens of seconds to a few minutes; Weak = multiple minutes to hours per application, or reliance on GPU/DistilBERT-class inference not documented as real-time capable. For Scalability (SC): Strong = evaluated on a corpus of 50,000+ applications or explicitly reported as batch-parallelisable; Moderate = evaluated on 1,000–50,000 applications; Weak = evaluated on under 1,000 applications or reported as computationally prohibitive at scale. For Temporal Robustness (TR): Strong = explicit multi-year evaluation with under 10-point decline (e.g., MaMaDroid's 4-point decline [15]); Moderate = decline of 10–25 points, or drift acknowledged but only partially quantified; Weak = no temporal evaluation reported, marked '?' in Table 4 where the underlying paper gives no basis for even an approximate judgement. For Practical Deployability (PD): Strong = open dataset, reported inference cost, and no specialised hardware requirement; Moderate = at least one of these missing; Weak = requiring custom hardware (e.g., DL-Droid's physical device fleet [14]), enterprise-only APIs, or unreleased code and data. Where the source publication does not report the figure needed to apply a threshold, the rating is qualified in the running text (e.g., "DA: S (benchmark), W (temporal)") rather than presented as a single unqualified score, and the corresponding cell in Table 4 is marked '?' instead of guessed.

Table 1. Six Comparison Criteria

DA	True positive rate under temporally consistent (TESSERACT) evaluation.
OR	Performance under obfuscation: identifier renaming, control-flow changes, string encryption, API reflection.
CO	Analysis time and memory per application; on-device feasibility.
SC	Ability to process tens of thousands of apps without prohibitive cost.
TR	Accuracy degradation rate over months and years of real deployment.
PD	Realistic production feasibility: infrastructure needs, dependencies, and explainability.

DA=Detection Accuracy, OR=Obfuscation Resilience, CO=Computational Overhead, SC=Scalability, TR=Temporal Robustness, PD=Practical Deployability.

3. Static Analysis Approaches

Static analysis works on an application's artefacts — bytecode, the APK manifest, resource files, digital certificates — without ever running the app. That is what makes it fast, but it is also its core weakness: it reasons over syntactic representations of code, and an adversary can rewrite those without touching what the app actually does.

3.1. Drebin (Arp et al., NDSS 2014)

Drebin [3] is still the most cited baseline in Android malware detection. It trains a linear SVM on eight feature sets drawn from the AndroidManifest.xml and disassembled Dalvik Executable (DEX) bytecode, achieving 94% accuracy on 123,453 applications. Its interpretability – linear feature weights expose exactly which permissions or API calls contributed to a verdict – and sub-second inference time distinguish it from almost all subsequent systems.

The limitations are severe and well-documented. Rastogi et al. [6] showed that automated obfuscation toolkits reduce Drebin's detection rate by up to 34 percentage points for transformed variants of known malware. TESSERACT [5] found accuracy falling to 65–75% under temporally consistent evaluation.

Assessment: DA: S (benchmark), W (temporal); OR: W; CO: S; SC: S; TR: W; PD: M.

3.2. FlowDroid (Arzt et al., ACM PLDI 2014)

FlowDroid [7] implements context-sensitive, flow-sensitive, field-sensitive, and object-sensitive taint analysis, tracking sensitive data from sources such as GPS coordinates and device identifiers to sinks such as network connections across all Android component boundaries. On the DroidBench benchmark, it identifies 93% of true information leaks with zero false positives. This precision costs minutes to hours per application, ruling out large-scale deployment. Native code called via the Java Native Interface (JNI) lies outside FlowDroid's analysis scope – a gap increasingly exploited by malware that routes exfiltration through native libraries.

Assessment: DA: S (DroidBench); OR: M; CO: W; SC: W; TR: M; PD: W.

3.3. DroidAPIMiner (Aafer et al., 2013/2014)

DroidAPIMiner [8] concentrates on API call frequency and package-level provenance, achieving 92.5% accuracy on 1,727 applications using a k-Nearest Neighbour (k-NN) classifier. The fundamental limitation is blindness to call context: sendSMS() called from a background service in response to a command-and-control (C2) instruction is indistinguishable from the same call triggered by user interaction.

Assessment: DA: M; OR: W; CO: S; SC: S; TR: W; PD: M.

3.4. SigPID (Li et al., IEEE Trans. Ind. Inf. 2018)

Significant Permission IDentification (SigPID) [10] reduces 135 Android permissions to 22 discriminative ones via three-level pruning on Support, Confidence, and Negative Support metrics, achieving 93.62% accuracy comparable to classifiers using the full permission set. The specific 22-feature set ages as Android's permission landscape evolves.

Assessment: DA: M; OR: W; CO: S; SC: S; TR: W; PD: S.

3.5. PermPair (Arora et al., IEEE TIFS 2020)

PermPair [11] examines co-occurring permission pairs rather than individual permissions, capturing interaction effects that additive scores miss. An application requesting both READ_CONTACTS and INTERNET is more suspicious than one requesting either alone. Evaluated on three datasets, PermPair achieves up to 97.6% accuracy with reduced dimensionality.

Assessment: DA: S; OR: W; CO: S; SC: S; TR: M; PD: S.

4. Dynamic Analysis Approaches

Dynamic analysis takes the opposite approach: it watches an application's behaviour while it runs, so it captures runtime signals that survive syntactic changes to the code. The costs are execution time, incomplete code coverage, and the fact that sandbox-based implementations can be detected and evaded by environment-aware malware.

4.1. TaintDroid (Enck et al., ACM TOCS 2014)

TaintDroid [12] instruments the Dalvik virtual machine to propagate taint tags through bytecode instructions, monitoring sensitive data flows at near-zero runtime overhead. Deployment requires a modified Android runtime, restricting scope to managed devices. JNI-called native code is not tainted – a well-known and widely exploitable gap.

Assessment: DA: M (scope-limited); OR: S; CO: S; SC: M; TR: M; PD: W.

4.2. Transcend (Jordaney et al., USENIX 2017)

Transcend [13] is not a detector but a meta-system: using conformal prediction theory, it assigns credibility and confidence scores to flag predictions whose confidence falls below training-distribution thresholds, providing early warning of concept drift before it causes silent accuracy degradation. The limitation is dependency on a continuous monitoring infrastructure.

Assessment: DA: N/A; TR: S; PD: M.

4.3. DL-Droid (Alzaylaee et al., Comput. Secur. 2020)

DL-Droid [14] executes applications on physical Android devices via the Android Debug Bridge (ADB), making the execution environment indistinguishable from a

legitimate user device. A deep learning classifier trained on 420 stateful dynamic features achieves 97.8% with dynamic features alone, rising to 99.6% combined with static features. The 99.6% figure demands scrutiny: evaluation does not follow TESSERACT protocol, and a 120-second window may miss delayed malicious routines that activate after specific delays. The physical device fleet represents significant capital expenditure.

Assessment: DA: S (real-device); OR: S; CO: W; SC: W; TR: M; PD: W.

5. Machine Learning and Hybrid Approaches

5.1. MaMaDroid (Mariconti et al., NDSS 2017)

Malware Detection with Markov chains (MaMaDroid) [15] abstracts API calls to their package or family level before modelling call sequences as Markov chains, capturing transition probabilities between abstract behavioural states. This achieves two things simultaneously: resilience to identifier-level obfuscation, and capture of temporal ordering structure that frequency vectors discard. The 99% F1-score and only a 4-percentage-point decline over three years represent the field's best temporal robustness result of this era.

Assessment: DA: S; OR: S; CO: M; SC: S; TR: S; PD: M.

5.2. MalDozer (Karbab et al., Digit. Investig. 2018)

MalDozer [16] applies a Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) architecture to raw API method call sequences, achieving F1-scores of 96–99% with a false-positive rate of 0.06%. End-to-end learning eliminates manual feature engineering. The weakness is interpretability: no readable signals are available for analyst justification.

Assessment: DA: S; OR: M; CO: M; SC: M; TR: M; PD: M.

5.3. DeepRefiner (Xu et al., IEEE EuroS&P 2018)

DeepRefiner [17] introduces a two-layer cascade: a Multilayer Perceptron (MLP) static pre-filter passes uncertain cases to a Long Short-Term Memory (LSTM) dynamic analyser. This cascade insight – apply the expensive layer only to ambiguous cases – is practically important and underexplored. Evaluated on 110,000 applications, DeepRefiner achieves 97.74% accuracy. The dynamic layer uses emulator-based execution, retaining vulnerability to environment-aware evasion.

Assessment: DA: S; OR: M; CO: M; SC: M; TR: M; PD: M.

5.4. TESSERACT (Pendlebury et al., USENIX 2019)

TESSERACT [5] — Temporally Efficient Security-aware System Evaluation for malware Classification across Time and Space — is arguably the single most important methodological paper in this field, not because it detects malware but because it shows that most detection papers have been measuring the wrong thing.

It identifies two systematic biases. Spatial bias comes from test sets whose class balance does not match reality – real malware encounter rates run around 1–5%, not the roughly 50/50 split used in most evaluations. Temporal bias comes from testing on samples drawn from earlier time periods than the training data, which effectively tests a classifier against its own past rather than its future. Combined, these two biases inflate reported accuracy by 5–30 percentage points – a system reporting 99% on a standard benchmark may only reach 72–85% under TESSERACT-compliant evaluation. However, more than six years after publication, fewer than half of post-2019 papers actually adopt its protocol.

Assessment: Methodological contribution – essential for realistic evaluation; no detection capability.

Table 2. Comparison of Android Malware Detection Systems (2014–2026)

System	Year	Technique	Acc.	Main Contribution	Primary Limitation
Drebin	2014	Static SVM	94.0%	Permission/API based malware detection	Weak against modern obfuscation.
FlowDroid	2014	Taint Analysis	93%	Precise information flow tracking	Very high analysis time.
DroidAPIMiner	2014	API Mining	92.5%	API frequency analysis	No contextual understanding.
TaintDroid	2014	Dynamic	–	Runtime monitoring	Cannot inspect native code.
Apposcopy	2014	Semantic Analysis	–	ICC behaviour analysis	Manual rule maintenance.
MaMaDroid	2017	Markov Chains	99.0%	API abstraction	Call graph extraction overhead.
Transcend	2017	Concept Drift	–	Detects unreliable predictions	Not a malware detector.
SigPID	2018	Permission Selection	93.6%	22-permission model	Permission evolution.
MalDozer	2018	CNN+RNN	96.99%	Automatic feature learning	Low explainability.
DeepRefiner	2018	Cascade Learning	97.74%	Hybrid static-dynamic	Emulator dependence.
TESSERACT	2019	Evaluation	–	Temporal validation	Framework only.
PermPair	2020	Permission Pairs	97.6%	Permission interaction	Weak runtime protection.
DL-Droid	2020	Dynamic Learning	97.8%	Real-device execution	Needs physical devices.

AMalNet	2020	GCN	97.3%	Graph learning	GPU-intensive.
Lo et al.	2022	JK-GNN	96.2%	Improved graph representation	No temporal validation.
DeepCatra	2022	BiLSTM+GNN	–	Dual-view detection	High computational cost.
MASKDROID	2024	Masked GNN	97.5%	Self-supervised learning	Expensive training.
SeGDroid	2024	Sensitive FCG	98.1%	Sensitive API graph	Limited deployment study.
GIT-GuardNet	2025	Transformer+GAT	98.7%	Cross-attention fusion	Large inference latency.
DMLDroid	2025	Multimodal Fusion	98+%	Permission+Image+BERT	Heavy Transformer model.
Transformer Encoder	2025	Transformer	97+%	Pure attention architecture	Limited robustness.
ANAKIN	2026	GNN+Explainability	98+%	Explainable malware graphs	SOC validation is missing.
BERT–PSO	2026	BERT+PSO	97.87%	Transformer feature selection	Still computationally expensive.

6. Deep Learning And Gnn Approaches

6.1. AMalNet (Li et al., Comput. Secur. 2020)

AMalNet [18] applies Graph Convolutional Networks (GCNs) to Function Call Graphs (FCGs) extracted from APK bytecode, achieving 97.3% accuracy on Drebin, demonstrating better obfuscation resilience compared to sequence-based approaches.

Graph topology is preserved under identifier renaming – a GCN-based classifier requires adversaries to restructure the application's fundamental call architecture to evade detection.

Assessment: DA: S; OR: S; CO: W; SC: M; TR: ?; PD: M.

6.2. GNN with Jumping Knowledge (Lo et al., IEEE DSC 2022)

Lo et al. [19] address the over-smoothing problem of GNNs via Jumping Knowledge (JK), aggregating all GNN layer representations instead of only the last one. Performance improves 1–2 points over standard GCN on obfuscated variants. The technical contribution is solid; the practical improvement over AMalNet is incremental.

Assessment: DA: S; OR: S; CO: W; SC: M; TR: ?; PD: M.

6.3. DeepCatra (Wu et al., IEEE TIFS 2022)

DeepCatra [20] combines a bidirectional LSTM on temporal opcode sequences with a GNN on inter-component flow graph topology, achieving 2.7–14.6% F1 improvement over single-view baselines on over 18,000 applications. The dual-view architecture captures both sequential and structural information complementarily. Per-application analysis time is not reported, limiting deployment feasibility assessment.

Assessment: DA: S; OR: S; CO: W; SC: M; TR: ?; PD: M.

6.4. MASKDROID (ASE 2024)

MASKDROID [21] applies masked graph autoencoder pre-training to FCGs, learning structural representations from large unlabelled APK corpora before fine-tuning on labelled data. The improved out-of-distribution generalisation to unseen malware families is the most practically important result – generalisation to new families is the central challenge for deployed detection.

Assessment: DA: S; OR: S; CO: W; SC: M; TR: M; PD: M.

6.5. SeGDroid (Liu et al., Expert Syst. Appl. 2024)

SeGDroid [4] weights FCG edges connecting to sensitive Android APIs, concentrating graph learning on malware-relevant substructures. Cross-dataset generalisation is reported at 98.1% – a more stringent test than single-dataset evaluation.

Assessment: DA: S; OR: S; CO: M; SC: S; TR: M; PD: M.

6.6. GIT-GuardNet (Rashid et al., Sci. Rep. 2025)

GIT-GuardNet [22] fuses three analytical perspectives via cross-attention: static code attributes through a Transformer encoder, call graph structures through a Graph Attention Network (GAT), and temporal behaviour through a Temporal Transformer. Achieves 98.7% on CICAndMal2020 with resilience to gradient-based adversarial perturbations. Large model size creates inference latency, limiting real-time deployment.

Assessment: DA: S; OR: S; CO: W; SC: M; TR: ?; PD: M.

6.7. DMLDroid (Trung et al., 2025)

DMLDroid [45] combines three complementary modalities: permissions and intents (tabular features), DEX-file byte structure (image-based features), and API call-graph sequences encoded using a pre-trained DistilBERT model. A dynamic weighted fusion mechanism integrates these heterogeneous representations, enabling robust malware detection. Unlike most existing approaches, DMLDroid is explicitly evaluated against both code obfuscation techniques and GAN-generated adversarial examples, maintaining over 98% accuracy and F1-score on the CICMalDroid 2020 dataset. However, the DistilBERT encoder introduces substantial computational overhead, requiring several hours for training and considerably longer inference than lightweight static-analysis methods.

Assessment: DA: S; OR: S; CO: W; SC: M; TR: ?; PD: M.

6.8. Transformer-Encoder Static Detector (Shakib, Array 2025)

Shakib [46] proposes a Transformer-based static malware detector that relies solely on attention and encoder blocks without convolutional or recurrent networks. Static permissions and API-call features extracted using Androguard are processed through the Transformer architecture for malware classification. Although the model demonstrates competitive accuracy, its performance remains comparable to established static methods such as Drebin and SigPID. The findings suggest that attention mechanisms alone do not guarantee improved detection performance without richer graph-based or multimodal feature representations.

Assessment: DA: M; OR: W; CO: S; SC: S; TR: ?; PD: M.

6.9. ANAKIN (Cybersecurity 2026)

ANAKIN [47] constructs API graphs from Function Call Graphs (FCGs) and applies a Graph Neural Network (GNN) for Android malware detection. Its primary contribution is explainability through integration of the GNNExplainer algorithm, which identifies the API nodes and control-flow edges responsible for classification decisions. Experimental results on more than 26,000 APKs demonstrate higher accuracy than sequence-based API representations while providing interpretable predictions. Nevertheless, the practical effectiveness of these explanations has not yet been validated in real-world Security Operations Centre (SOC) environments. Recent explainable artificial intelligence (XAI) techniques, such as LIME [39], SHAP [40], and GNNExplainer [41], have significantly improved the interpretability of deep learning models. Recent surveys further highlight explainability as a key future direction for Android malware detection [43].

Assessment: DA: S; OR: S; CO: W; SC: M; TR: ?; PD: M.

6.10. BERT–PSO Framework (Banik and Singh, Discover Computing 2026)

BERT–PSO [48] extracts permissions and API-call sequences from decompiled applications and generates contextual embeddings using BERT. Particle Swarm Optimization (PSO) is then employed for feature selection, reducing the feature space by approximately 50% while maintaining high classification performance.

Evaluated on Drebin and AndroZoo datasets, the framework achieves 96.27–97.87% accuracy and additionally performs multi-year temporal validation, reporting an expected performance decline over evolving malware distributions. This makes it one of the few studies to explicitly quantify temporal robustness while combining transformer-based feature extraction with lightweight optimization techniques.

Assessment: DA: S; OR: M; CO: M; SC: S; TR: M; PD: S.

7. Reputation-Based Intelligence Platforms

Reputation-based threat intelligence platforms play a somewhat different but important role in Android malware detection. Rather than analysing APK structure or runtime behaviour directly, they pool crowdsourced intelligence from security researchers, antivirus vendors, and automated scanners into a single multi-source verdict on a submitted file. DL-Droid [14] was among the first systems to fold a VirusTotal API lookup into its pipeline, and many later systems followed suit, to the point that reputation lookups have become a standard building block of production-grade detection frameworks.

7.1. VirusTotal

VirusTotal [29] started life at Hispasec Sistemas in 2004, was acquired by Google in 2012, and moved under Google Security Operations in 2018. As of 2024, it has more than three million monthly users, holds records on over 3.7 billion files, and runs roughly six million analyses a day, scanning each submission with 70-plus antivirus engines, over ten dynamic sandboxes, and a set of static tools. Its API v3 returns threat scores, family labels, rich IoC relationship data, and AI-generated code summaries.

The Enterprise tier adds several capabilities on top of this. VT Intelligence is essentially a dedicated search engine over the malware corpus, built for in-depth profiling; VT Hunting tracks how threats evolve over time; and VT Graph lets an analyst explore relationships among URLs, domains, IP addresses, and malware samples visually. For bulk work, Enterprise adds file, URL, and sandbox feeds, plus VT Monitor for catching false positives and scanning software before release.

That said, the public API is capped at 500–1,000 requests a day, and the platform is fundamentally reactive: a genuinely new sample scores a detection ratio of zero until antivirus vendors update their signatures, which can take anywhere from hours to days. There is also a confidentiality concern – anything submitted through the public interface becomes visible to other users, which is a real risk for an organisation investigating a sensitive incident.

7.2. AndroZoo

AndroZoo [36] is maintained by the University of Luxembourg and holds a continuously growing collection of more than seven million Android applications, pulled mainly from Google Play but also from APKPure and AppChina. Each entry comes with VirusTotal scan results, a SHA-256 hash, package name, version code, and metadata on which market it was pulled from.

What makes AndroZoo valuable for research is that every entry carries a reliable first-seen timestamp, which lets researchers build temporally ordered datasets that satisfy TESSERACT's evaluation protocol — something static

benchmarks like Drebin cannot offer. For that reason, it is the dataset this review would recommend as a primary sample source for any study that wants its accuracy figures to mean something in deployment.

It is not without limitations, though: access needs a research agreement and API key, bandwidth is rate-limited, and because labelling is derived from VirusTotal detection ratios, AndroZoo inherits VirusTotal's blind spot on zero-day samples.

7.3. Comparative Review of Platforms

VirusTotal is not the only option; several other reputation and analysis platforms are available to the community, each with a somewhat different analytical angle. Table 3 lines VirusTotal up against six of the main alternatives.

Table 3. Comparative Analysis of Reputation-Based Malware Intelligence Platforms

Platform	AV Engines	API	Sample Type	Primary Strength	Key Limitation	Use Case
VirusTotal [29]	70+	v3 REST API	Public (shared with community)	Very large malware database with code summaries and threat links	May miss brand-new (zero-day) threats; daily limit 500 req/day	Best for large-scale malware reputation analysis
MetaDefender (OPSWAT) [30]	30–35+	REST API + on-prem SDK	Private	Fast scanning with strong detection and a private deployment option	Fewer AV engines than VirusTotal; less detailed for deep analysis	For orgs needing quick, secure incident response
Hybrid Analysis [31]	N/A	REST API (Paid)	Public / Paid private	Deep behaviour analysis incl. network activity and memory changes	Mainly Windows-focused; Android APK analysis is limited and slower	Android behaviour investigation, detailed analysis
ANY.RUN [32]	N/A	REST API	Paid private	Interactive sandbox – observe malware behaviour in real time	Paid for private mode; not ideal for full automation	Malware requiring user interaction
Intezer Analyze [33]	N/A	REST API	Enterprise private	Tracks malware family origin via code-reuse similarity	Limited AV comparison; files must be uploaded	Malware family tracking and APT attribution
Jotti's Malware Scan [35]	14+	N/A	Public	Free, instant multi-engine scan without registration	No behaviour analysis; limited AV coverage	Quick manual malware checks
MalwareBazaar [34]	None	REST API	Public	Open malware repository with YARA rules and downloadable samples	No AV scanning; depends on community tagging	Building malware datasets for research and training

IR = Incident Response. CDR = Content Disarm and Reconstruction. IoC = Indicator of Compromise. YARA = Yet Another Recursive Acronym (pattern-matching rule language).

7.3.1. OPSWAT MetaDefender

MetaDefender [30] sets itself apart from VirusTotal in two main ways. It can be deployed on-premises with more than 35 anti-malware engines, so samples can be multiscanned without ever leaving the corporate network — useful when confidentiality matters.

It also offers Content Disarm and Reconstruction (CDR), which strips potentially malicious content from a file and rebuilds a clean version rather than just flagging the threat. On typical hardware, a scan finishes in under 500 milliseconds.

7.3.2. Hybrid Analysis

Hybrid Analysis [31] runs on CrowdStrike's Falcon Sandbox and leans toward deep behavioural analysis rather than multi-engine scanning. Samples are executed in isolated Windows and Linux environments, and the platform captures network traffic, file-system changes, process activity, registry edits, and memory artefacts in more depth than VirusTotal's own sandbox integrations typically offer. For Android research specifically, though, the sandboxes are really built around Windows executables — Android APK analysis exists but is less thoroughly documented.

7.3.3. ANY.RUN Interactive Sandbox

ANY.RUN [32] takes a different approach again: instead of fully automated execution, the analyst interacts with the running sample directly — clicking, typing, navigating — which can surface execution paths an automated sandbox would miss entirely.

That matters for Android malware that only activates once a specific user action happens, such as a banking trojan waiting for the victim to open their banking app. A paid subscription is needed for API access and private sessions.

7.3.4. Intezer Analyze

Intezer [33] works differently again: instead of scanning with antivirus engines, it breaks the submitted binary into code segments and compares them against a database of known malware and legitimate software, looking for reused code.

That is particularly useful for repackaged Android apps, where a malicious payload has been stitched into a legitimate app's code – shared segments between the submission and a known malware family can still be spotted even when obfuscation hides the API-level signatures.

7.3.5. MalwareBazaar

MalwareBazaar [34], run by abuse.ch, is not a scanning service at all but an open repository of malware samples. Anyone can submit or download samples, and each one carries metadata such as YARA rule matches, file hashes, first-seen timestamps, and family tags assigned by reporters. Its API supports bulk downloads filtered by family, file type, or date, which makes it a useful complement to VirusTotal when building Android training datasets.

8. Cross-Cutting Issues

8.1. Concept Drift: The Field's Most Persistent Failure

Concept drift, the gap between the threats a model meets in deployment and the distribution it was trained on, has done more damage to real-world performance than any other single issue in this field. TESSERACT [5] quantified it directly, finding accuracy drops of 5–30% over two-to-three-year evaluation windows.

Transcend [13] can flag that drift is occurring, but does not adapt the model to it. The Efficient Concept Drift (ECD) framework [23] shows that retraining restores performance, though only when labelled data from the new distribution is actually available, and DREAM [24] reduces the labelling effort needed through contrastive autoencoder concept explanations without eliminating it.

Taken together, these results point to an unresolved gap: no published system keeps accuracy stable over multiple years without ongoing human involvement.

8.2. Adversarial Robustness: An Underexplored Vulnerability

Grosse et al. [25] showed at ESORICS 2017 that Android malware classifiers can be fooled by gradient-based perturbations of their input features. Small changes to the manifest, changes that leave the app's actual functionality untouched, are enough to slip past state-of-the-art classifiers.

Yumlembam et al. [26] later showed that VGAE-MalGAN attacks can substantially reduce GNN detection rates, although adversarial retraining recovers a meaningful part of that loss. Certified defences for Android's feature space remain an open problem.

8.3. Dataset Bias and Benchmark Age

Liu et al.'s 2025 benchmarking study [27] found that, once TESSERACT-compliant protocols are applied, Random Forest and CatBoost keep pace with GNN and Transformer models at a fraction of the computational cost. This is a useful check on the field, suggesting that some of the reported deep-learning advantage comes from experimental design choices rather than genuinely better generalisation.

A related problem is dataset age: the 2014 Drebin dataset, built from malware collected in 2010–2012, is still used to evaluate systems published as late as 2022–2024, and the field has not seriously addressed the methodological risk this creates.

9. The Generalised Detection Pipeline

9.1. Pipeline Design and Prior Usage

The eight-stage pipeline shown in Fig. 1 is this review's own synthesis rather than a copy of any single published system, but every stage is grounded in prior work. Cross-platform provenance metadata appears in an early form in RmvDroid [37]; static analysis via Androguard is used by both Drebin [3] and MaMaDroid [15]; graph construction is central to AMalNet [18], SeGDroid [4], and GIT-GuardNet [22]; Mohaisen et al. [38] were the first to formalise VirusTotal API integration as a detection layer; real-device ADB-based dynamic analysis is DL-Droid's [14] main contribution; multi-modal feature fusion is the defining choice behind DeepRefiner [17] and GIT-GuardNet [22]; and weighted risk fusion with per-layer verdicts, originally proposed for IoT security by Yumlembam et al. [26], is adapted here for Android.

None of the systems covered in this review combines all eight stages — multi-platform metadata, static analysis, graph construction, VirusTotal reputation, real-device dynamic analysis, multi-modal feature fusion, ML classification, and weighted risk fusion — into one self-improving architecture. This review proposes a combination as a concrete direction for future work, building on the evidence already gathered for each individual stage above.

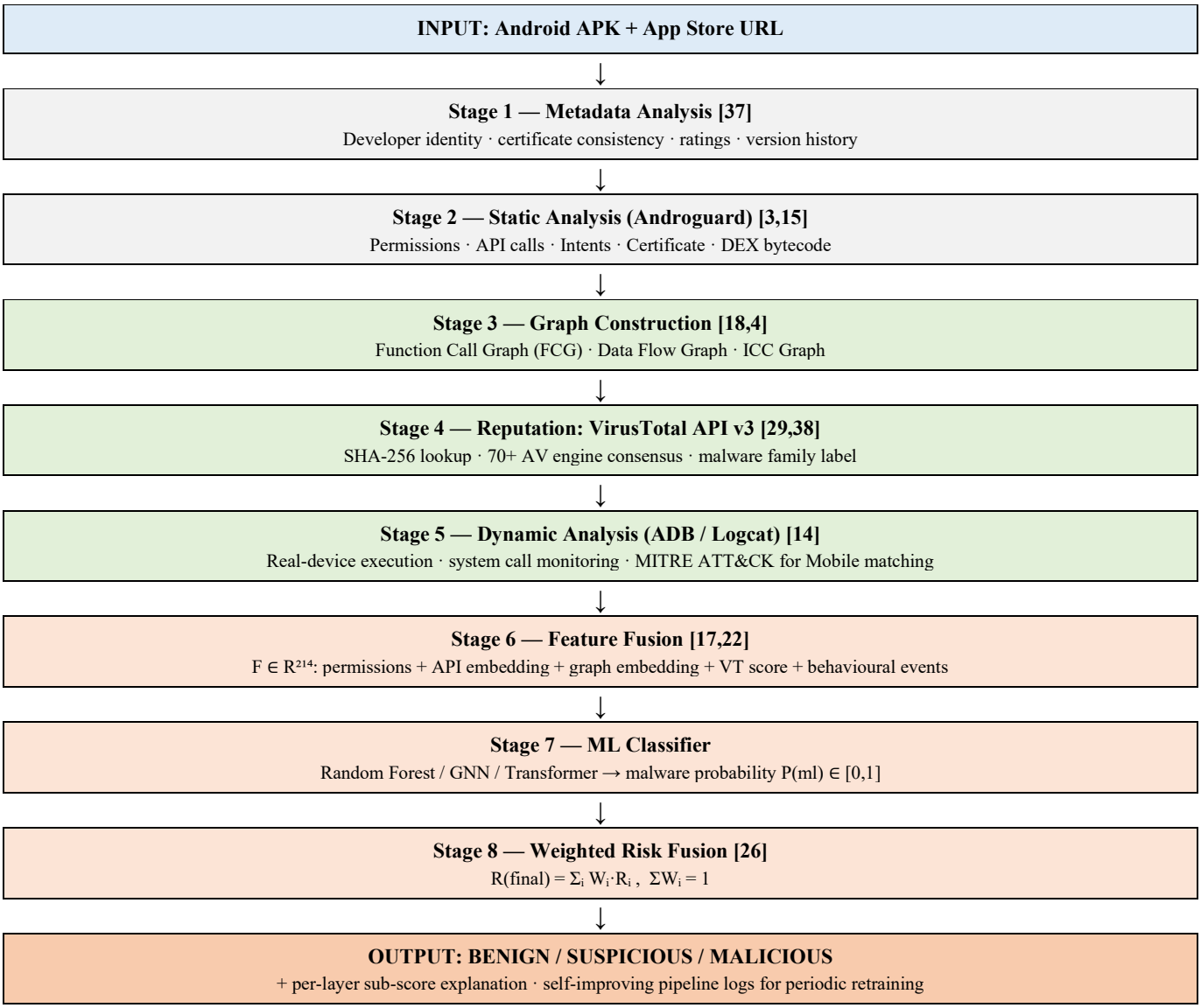

Fig. 1 Generalised eight-stage Android malware detection pipeline synthesised from the reviewed systems. Citations indicate the primary prior work underpinning each stage. Blue = static/metadata; Green = external/dynamic; Orange = ML/decision

9.2. Pipeline Limitations

The proposed eight-stage pipeline is a conceptual synthesis drawn from the literature rather than an architecture that has itself been experimentally validated, and this novelty carries real limitations that should be stated plainly.

First, no existing system uses all eight stages together. Each stage individually has prior-work support, but no published system has tested the full eight-stage combination end-to-end, so the performance figures discussed here are projections built up from individual-stage results rather than validated end-to-end numbers. This is the pipeline's biggest limitation and the most important thing for future work to address.

Second, cross-platform metadata is genuinely unexplored territory. Stage 1 pulls metadata simultaneously from Google Play, APKMirror, APKPure, and the Microsoft Store, a multi-platform cross-verification approach that no prior system has attempted. That is a real contribution, but it comes with practical fragility: scraping APKMirror and APKPure breaks whenever the sites change structure, Microsoft Store's Android support (via the Windows Subsystem for Android) is limited and not consistently populated, and there is no standard ground-truth dataset against which cross-platform metadata consistency scores could even be evaluated.

Third, Stage 4 inherits VirusTotal's own limits. The public API's 500–1,000 requests per day is not enough for large-scale deployment without an enterprise subscription, and a genuinely new sample not yet indexed by VirusTotal scores a detection ratio of zero. This stage contributes least precisely when a novel threat needs it most.

Fourth, Stage 5 needs real hardware. Running real-device dynamic analysis means maintaining a managed fleet of Android devices, which is not cheap to buy or operate, and a 120-second execution window can still miss malicious routines that only trigger after a delay, repeated interaction, or a specific environmental cue such as the presence of a targeted banking app.

Fifth, Stage 7's classifier has no certified robustness against adversarial attacks. It has not been tested against gradient-based or query-based attacks, and an attacker who can query the system repeatedly, something LLM-assisted malware generation is making easier, could plausibly craft samples that slip past the classifier without ever tripping the static or dynamic layers.

Finally, the self-improving loop in Stage 8 only counters concept drift if a steady stream of labelled new samples keeps coming in, and in practice, getting that ground truth means either analyst time or VirusTotal consensus, both of which add latency and cost.

10. Comprehensive Critical Comparison

The tables below summarise how the reviewed systems fare across the dimensions set out in Section 2, using S (Strong) where a system handles that dimension well, M (Moderate) where it partly handles it but has issues, W (Weak) where it clearly falls short, and ? where it was not evaluated. The core finding here is a simple one: not a single system in this review is rated Strong on all six dimensions at once.

10.1. Individual Criterion Breakdown

The following tables classify every reviewed system according to each evaluation criterion individually.

Table 4. Obfuscation Resilience (OR)

Rating	Systems
Strong	MaMaDroid, DL-Droid, AMalNet, Lo et al., DeepCatra, MASKDROID, SeGDroid, GIT-GuardNet, DMLDroid, ANAKIN.
Moderate	FlowDroid, MalDozer, DeepRefiner, BERT–PSO.
Weak	Drebin, DroidAPIMiner, SigPID, PermPair, Transformer Encoder.
Not evaluated	Transcend, TESSERACT.

Table 5. Computational Overhead (CO)

Rating	Systems
Strong	Drebin, DroidAPIMiner, SigPID, PermPair, Transformer Encoder.
Moderate	MaMaDroid, MalDozer, DeepRefiner, SeGDroid, BERT–PSO.
Weak	FlowDroid, DL-Droid, AMalNet, Lo et al., DeepCatra, MASKDROID, GIT-GuardNet, DMLDroid, ANAKIN.
Not evaluated	Transcend, TESSERACT.

Table 6. Temporal Robustness (TR)

Rating	Systems
Strong	MaMaDroid, Transcend, ECD, DREAM, BERT–PSO.
Moderate	FlowDroid, PermPair, MalDozer, DeepRefiner, DL-Droid, MASKDROID, SeGDroid.
Weak	Drebin, DroidAPIMiner, SigPID, AMalNet.
Not evaluated	Lo et al., DeepCatra, GIT-GuardNet, DMLDroid, Transformer Encoder, ANAKIN, TaintDroid, Apposcopy, TESSERACT.

Table 7. Practical Deployability (PD)

Rating	Systems
Strong	SigPID, PermPair, BERT–PSO.
Moderate	Drebin, DroidAPIMiner, MaMaDroid, MalDozer, DeepRefiner, Transcend, ECD, AMalNet, Lo et al., DeepCatra, MASKDROID, SeGDroid, GIT-GuardNet, DMLDroid, Transformer Encoder, ANAKIN.
Weak	FlowDroid, TaintDroid, Apposcopy, DL-Droid.
Not evaluated	TESSERACT.

Table 8. Overall Rating Summary Across All Evaluation Criteria

System	DA	OR	CO	SC	TR	PD
Drebin	S	W	S	S	W	M
FlowDroid	S	M	W	W	M	W
DroidAPIMiner	M	W	S	S	W	M
MaMaDroid	S	S	M	M	S	M
MalDozer	S	M	M	M	M	M
DeepRefiner	S	M	M	M	M	M
PermPair	S	W	S	S	M	S
DL-Droid	S	S	W	M	M	W
AMalNet	S	S	W	M	W	M
MASKDROID	S	S	W	M	M	M
SeGDroid	S	S	M	M	M	M
GIT-GuardNet	S	S	W	M	?	M
DMLDroid	S	S	W	M	?	M
Transformer Encoder	S	W	S	M	?	M
ANAKIN	S	S	W	M	?	M
BERT–PSO	S	M	M	M	S	S

10.2. Critical Finding

Across all of the systems reviewed, none earns a Strong rating on every practical deployment criterion at once. Traditional static approaches stay computationally cheap but continue to struggle with obfuscation and concept drift; dynamic and graph-based approaches buy more robustness at the cost of more compute; and the newer Transformer-based systems push benchmark accuracy higher still, without yet having the deployment validation or long-term temporal testing to back it up.

11. Research Gaps

Looking back across all thirty-one systems and platforms and a full decade of work, seven problems keep recurring. The first five are structural issues affecting the field as a whole; the last two are specific to the eight-stage pipeline proposed here, and are probably the most concrete starting points for future work.

11.1. No System Does Everything Well

Table 2 makes the trade-off visible: every system reviewed has at least one Weak rating somewhere. A system that is fast and lightweight (Strong CO) usually gets there by looking only at surface-level code features, which leaves it weak against obfuscation (Weak OR); a system that earns strong obfuscation resilience by running on a real device (Strong OR) is expensive to run at scale (Weak CO and PD). The deeper issue is that no published system has actually set out to be strong on all six dimensions at once – researchers typically pick one dimension to optimise, usually detection accuracy on a clean benchmark, and treat the rest as secondary. The result is a decade's worth of systems that look impressive in the lab but break down in one or more ways once deployed.

Going forward, systems should state their target trade-off up front – for instance, "designed for on-device deployment, and therefore accepting lower obfuscation resilience in exchange for low overhead" – and be evaluated against that stated goal rather than a generic benchmark.

11.2. Temporal Evaluation

TESSERACT [5], published in 2019, made a simple but damning point: mixing old and new samples into one dataset and splitting them randomly inflates reported accuracy by 5–30 percentage points relative to what a system would actually achieve once deployed. That is not a minor statistical wrinkle – it is a methodological flaw at the core of how the field evaluates itself.

Table 8 shows that 14 of the 21 systems covered here never test on a proper temporal split, and some of those papers were published in 2022, 2023, and 2024 – three to five years after TESSERACT made the issue impossible to miss. Practically, that means we still do not know how well most of these systems would hold up in real deployment, and the 98% figures reported for several GNN-based systems could well be much lower in practice.

Conferences and journals should treat TESSERACT-compliant temporal evaluation the same way they already treat a proper train/test split – as a condition of publication, not an optional extra.

11.3. Explainability

A security analyst who receives a malware verdict needs to know why the system flagged it – to explain the decision to a manager, document it for an audit, or decide whether to trust it. An output of "malicious: 94% confidence" with nothing behind it is not much use in a real SOC.

Drebin's linear SVM, back in 2014, was genuinely explainable: you could look at its feature weights and say, for instance, that an app was flagged because it requests READ_SMS and SEND_SMS together, which is unusual for a weather app.

That kind of explainability has largely disappeared as the field has moved through GNNs and Transformer-based systems. GIT-GuardNet and MASKDROID do use GNNExplainer to highlight which graph edges mattered, but nobody has actually tested whether that output is useful or interpretable to a real analyst.

Explainability needs to be tested with real security analysts working through realistic SOC scenarios – not just by checking whether a graph attention weight happens to be non-zero.

11.4. The LLM Threat

Pa Pa et al. [28] showed back in 2023 that large language models like ChatGPT can produce functional malware-like code on request. That changes the threat model qualitatively: an attacker can now describe the evasion they want in plain language, generate code, test it against a detector, see the verdict, and iterate – all in a matter of minutes.

Every system covered in this review is vulnerable to this to some extent, simply because all of them were designed on the assumption that malware is written by humans working with limited time and resources. Once generating malware becomes as easy as typing a description into a chat window, the economics of the attacker-defender relationship shift substantially. Systems built around fixed feature sets or statically trained classifiers – which is most of what is reviewed here – are particularly exposed.

Researchers need to start testing detection systems specifically against LLM-generated malware variants and building detection methods that hold up against this newer threat model.

11.5. The Lab-to-Real-World Gap

Perhaps the most telling finding of this whole review is what is not there: not one of the papers reviewed reports data from a sustained, real-world production deployment.

Every paper evaluates on a dataset, reports its accuracy numbers, and stops – none of them come back later and say "we ran this on real enterprise traffic for six months, here is what actually happened." That gap matters because everything that makes real deployment hard – unpredictable app diversity, varying device environments, malware authors adapting in response, alert fatigue, false-positive management –does not show up in a clean lab evaluation.

A system with 98% accuracy on a curated benchmark could easily throw hundreds of false alerts a day in production and be operationally useless.

At least a few research groups, ideally working with industry partners, need to commit to longitudinal deployment studies. That is slower and harder than publishing another benchmark paper, but it is really the only way to find out whether a decade of this research has produced anything that works at scale.

11.6. No Standardised Evaluation Protocol for Multi-Stage Hybrid Pipelines

As far as we are aware, the eight-stage pipeline proposed here is the first to combine cross-platform metadata aggregation with static, graph-based, reputation, and real-device dynamic analysis inside one weighted risk-fusion architecture. However, there is no accepted way to evaluate a system like this yet.

Existing work evaluates single- or two-stage systems on shared benchmarks such as Drebin or CICMalDroid, but no benchmark has been built to measure how much each stage in a multi-stage pipeline actually contributes, how the stages interact, or how gracefully the system degrades when one or more stages go down.

The practical consequence is that it is currently impossible to fairly compare this pipeline against any existing system, since no existing system takes the same inputs or produces the same structured output. Without a shared evaluation protocol, the field cannot really build on multi-stage architectures cumulatively – every group ends up starting over with its own dataset splits, feature definitions, and fusion weights.

What is needed is a standardised evaluation framework for multi-stage hybrid detection systems: a shared dataset with temporal ordering, cross-platform metadata, and VirusTotal labels; a standard ablation protocol for measuring each stage's marginal contribution; and a graceful-degradation benchmark that tests how the system behaves when individual stages fail or are unavailable.

11.7. Cross-Platform Metadata Verification Has No Ground Truth

Stage 1 of the proposed pipeline is also the only component, across every system reviewed here, that performs cross-platform metadata verification – checking whether developer identity, certificate fingerprints, and declared permissions line up consistently across Google Play, APKMirror, APKPure, and the Microsoft Store at the same time. That is a genuinely new contribution, since repackaged and redistributed malware often shows inconsistencies across platforms that APK-level analysis on its own cannot pick up.

The catch is that no published dataset offers verified ground-truth labels for cross-platform metadata consistency – there is no corpus of apps with known-good and known-inconsistent metadata across stores, no agreed set of consistency features, and no accepted threshold for what counts as a suspicious discrepancy. Without that ground truth, Stage 1's actual contribution to overall pipeline accuracy cannot be rigorously measured, only estimated from first principles.

There is also an operational fragility problem: scraping third-party stores like APKMirror and APKPure is vulnerable to site structure changes, rate limiting, and geographic access restrictions, any of which can quietly degrade Stage 1's data quality without setting off any alert. Making cross-platform metadata collection more robust – through official APIs, federated community datasets, or formal data-sharing agreements with store operators – is a prerequisite before this component could be deployed in production.

Before Stage 1's contribution can really be evaluated properly, the field needs a dedicated cross-platform Android metadata dataset with ground-truth consistency labels verified by independent researchers. That sits at the intersection of software supply-chain security, web data engineering, and malware analysis, and no existing dataset or benchmark currently covers it.

12. Conclusion

This paper has worked through thirty-one Android malware detection systems and reputation-intelligence platforms published between 2014 and 2026, across static, dynamic, hybrid, and deep-learning approaches. The goal went beyond comparing reported benchmark numbers, aiming instead to understand how these systems hold up under practical deployment conditions. That is why the comparison used six criteria: detection accuracy, obfuscation resilience, computational overhead, scalability, temporal robustness, and practical deployability.

The comparative review points to one central conclusion. Benchmark accuracy on its own does not tell you

much about real-world deployment performance. As the TESSERACT framework shows, temporally inconsistent evaluation can inflate reported accuracy by 5–30%, and a lot of work published even after 2019 still does not evaluate temporally. None of the systems reviewed here scores strongly across all six criteria. GNN- and Transformer-based approaches tend to reach higher detection accuracy but need more compute and have little evidence behind them for practical deployment. Lighter methods, such as SigPID and PermPair, are efficient and easy to deploy but remain limited against obfuscation. Reputation-based platforms like VirusTotal and AndroZoo remain valuable for dataset construction and malware analysis, but because they depend on previously known malware, they are less useful against zero-day threats.

Several important challenges also remain open. Explainability has not kept pace with deep learning and graph-based detection, and only a handful of studies have checked whether the explanations these systems produce are actually useful to analysts. Long-term, real-world deployment studies are still rare, which makes it hard to say how reliable existing systems really are in practice, and only a small number of recent studies address robustness against adversarial attacks or AI-generated malware.

Overall, future work in this area should prioritise realistic temporal evaluation, better explainability, real deployment testing, and stronger robustness against evolving malware. Progress on these fronts is what would turn Android malware detection systems from tools that are merely accurate on a benchmark into tools that are actually reliable once running in real-world Android environments.

Conflicts of Interest

The author(s) declare(s) that there is no conflict of interest regarding the publication of this paper.

Funding Statement

This research received no external funding.

AI Declaration

Claude (Anthropic) was used during the preparation of this manuscript to assist with language refinement, readability, and revision of selected passages. The authors independently reviewed and verified the manuscript, including its technical content, cited literature, numerical results, interpretations, and conclusions, and take full responsibility for the accuracy, originality, and final content of the paper.

References

[1] Statcounter, Mobile Operating System Market Share Worldwide, 2024. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/

[2] Google, Android Security Year in Review 2023, Google Security Blog, Technical Report, 2024. [Online]. Available: https://blog.google/products-and-platforms/products/android-enterprise/android-security-paper-2023/

[3] Daniel Arp et al., "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," *Proceedings of NDSS*, San Diego, CA, vol. 14, no. 1, pp. 23-26, 2014. [Google Scholar] [Publisher Link]

[4] Zhen Liu et al., "SeGDroid: An Android Malware Detection Method Based on Sensitive Function Call Graph Learning," *Expert Systems with Applications*, vol. 235, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[5] Feargus Pendlebury et al., "TESSERACT: Eliminating Experimental Bias in Malware Classification Space and Across Time," *Proceedings of 28th USENIX Security Symposium*, Santa Clara, CA, pp. 729-746, 2019. [Google Scholar] [Publisher Link]

[6] Vaibhav Rastogi, Yan Chen, and Xuxian Jiang, "Catch Me if you Can: Evaluating Android Anti-Malware Against Transformation Attacks," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 1, pp. 99-108, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[7] Steven Arzt et al., "FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps," *ACM Sigplan Notices*, vol. 49, no. 6, pp. 259-269, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[8] Yousra Aafer, Wenliang Du, and Heng Yin, "DroidAPIMiner: Mining API-level Features for Robust Malware Detection in Android," *International Conference on Security and Privacy in Communication Systems*, Springer, vol. 127, pp. 86-103, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[9] Yu Feng et al., "Apposcopy: Semantics-Based Detection of Android Malware through Static Analysis," *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Hong Kong, pp. 576-587, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[10] Jin Li et al., "Significant Permission Identification for Machine-Learning-Based Android Malware Detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216-3225, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[11] Anshul Arora, Sateesh K. Peddoju, and Mauro Conti, "*PermPair*: Android Malware Detection using Permission Pairs," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1968-1982, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[12] William Enck et al., "TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, pp. 1-29, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[13] Roberto Jordaney et al., "Transcend: Detecting Concept Drift in Malware Classification Models," *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC, pp. 625-642, 2017. [Google Scholar] [Publisher Link]
[14] Mohammed K. Alzaylaee, Suleiman Y. Yerima, and Sakir Sezer, "DL-Droid: Deep Learning based Android Malware Detection using Real Devices," *Computers & Security*, vol. 89, pp. 1-11, 2020. [CrossRef] [Google Scholar] [Publisher Link]
[15] Lucky Onwuzurike et al., "MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 2, pp. 1-34, 2019. [CrossRef] [Google Scholar] [Publisher Link]
[16] ElMouatez Billah Karbab et al., "MalDozer: Automatic Framework for Android Malware Detection using Deep Learning," *Digital Investigation*, vol. 24, pp. S48-S59, 2018. [CrossRef] [Google Scholar] [Publisher Link]
[17] Ke Xu et al., "DeepRefiner: Multi-Layer Android Malware Detection System Applying Deep Neural Networks," *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, London, UK, pp. 473-487, 2018. [CrossRef] [Google Scholar] [Publisher Link]
[18] Xinjun Pei, Long Yu, and Shengwei Tian, "AMalNet: A Deep Learning Framework based on Graph Convolutional Networks for Malware Detection," *Computers & Security*, vol. 93, 2020. [CrossRef] [Google Scholar] [Publisher Link]
[19] Wai Weng Lo et al., "Graph Neural Network-based Android Malware Classification with Jumping Knowledge," *2022 IEEE Conference on Dependable and Secure Computing (DSC)*, Edinburgh, United Kingdom, pp. 1-9, 2022. [CrossRef] [Google Scholar] [Publisher Link]
[20] Yafei Wu et al., "DeepCatra: Learning Flow- and Graph-based Behaviors for Android Malware Detection," *IET Information Security*, vol. 17, no. 1, pp. 118-130, 2022. [CrossRef] [Google Scholar] [Publisher Link]
[21] Jingnan Zheng et al., "MaskDroid: Robust Android Malware Detection with Masked Graph Representations," *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pp. 331-343, 2024. [CrossRef] [Google Scholar] [Publisher Link]
[22] Muhammad Usama Tanveer et al., "GIT-GuardNet: Graph-Augmented Multi-Modal Learning Framework for Robust Android Malware Detection," *Scientific Reports*, vol. 15, no. 1, pp. 1-17, 2025. [CrossRef] [Google Scholar] [Publisher Link]
[23] Borja Molina-Coronado et al., "Efficient Concept Drift Handling for Batch Android Malware Detection Models," *arXiv preprint*, pp. 1-18, 2023. [CrossRef] [Google Scholar] [Publisher Link]
[24] Yiling He et al., "Combating Concept Drift with Explanatory Detection and Adaptation for Android Malware Classification," *arXiv preprint*, pp. 1-19 2024. [CrossRef] [Google Scholar] [Publisher Link]
[25] Kathrin Grosse et al., "Adversarial Examples for Malware Detection," *Computer Security – ESORICS 2017*, Oslo, vol. 10493, pp. 62-79, 2017. [CrossRef] [Google Scholar] [Publisher Link]
[26] Rahul Yumlembam et al., "IoT-based Android Malware Detection using Graph Neural Network with Adversarial Defense," *arXiv preprint*, pp. 1-13, 2025. [CrossRef] [Google Scholar] [Publisher Link]
[27] Guojun Liu et al., "Benchmarking Android Malware Detection: Traditional vs. Deep Learning Models," *arXiv preprint*, 2025. [CrossRef] [Google Scholar] [Publisher Link]
[28] Yin Minn Pa Pa et al., "An Attacker's Dream? Exploring the Capabilities of ChatGPT for Developing Malware," *Proceedings of the 16th Cyber Security Experimentation and Test Workshop*, pp. 10-18, 2023. [CrossRef] [Google Scholar] [Publisher Link]
[29] Google Security Operations, VirusTotal API v3 Overview, 2024. [Online]. Available: https://docs.virustotal.com/reference/overview
[30] OPSWAT, MetaDefender: A More Private Alternative to VirusTotal, 2024. [Online]. Available: https://www.opswat.com/blog/metadefender-more-private-alternative-virustotal
[31] CrowdStrike, Hybrid Analysis, 2024. [Online]. Available: https://www.hybrid-analysis.com
[32] ANY.RUN, Interactive Online Malware Sandbox, 2024. [Online]. Available: https://any.run
[33] Intezer, Intezer Analyze, 2024. [Online]. Available: https://analyze.intezer.com
[34] abuse.ch, MalwareBazaar, 2024. [Online]. Available: https://bazaar.abuse.ch
[35] Jotti, Jotti's Malware Scan, 2024. [Online]. Available: https://virusscan.jotti.org
[36] Kevin Allix et al., "AndroZoo: Collecting Millions of Android Apps for the Research Community," *Proceedings of the 13th International Conference on Mining Software Repositories*, Austin, TX, pp. 468-471, 2016. [CrossRef] [Google Scholar] [Publisher Link]
[37] Haoyu Wang et al., "RmvDroid: Towards a Reliable Android Malware Dataset with App Metadata," *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, Montreal, QC, Canada, pp. 404-408, 2019. [CrossRef] [Google Scholar] [Publisher Link]
[38] Aziz Mohaisen, Omar Alrawi, and Manar Mohaisen, "AMAL: High-Fidelity, Behavior-Based Automated Malware Analysis and Classification," *Computers & Security*, vol. 52, pp. 251-266, 2015. [CrossRef] [Google Scholar] [Publisher Link]
[39] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA, pp. 1135-1144, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[40] Scott M. Lundberg, and Su-In Lee, "A Unified Approach to Interpreting Model Predictions," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, pp. 1-10, 2017. [Google Scholar] [Publisher Link]

[41] Zhitao Ying et al., "GNNExplainer: Generating Explanations for Graph Neural Networks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, pp. 1-12, 2019. [Google Scholar] [Publisher Link]

[42] Josh Achiam, "*GPT-4 Technical Report*," arXiv preprint, pp. 1-100, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[43] Maryam Tanha, and Somayeh Kafaie, "A Review of Explainable AI for Android Malware Detection and Analysis," *IEEE Access*, vol. 13, pp. 141958-141974, 2025. [CrossRef] [Publisher Link]

[44] Parvez Faruki et al., "Android Security: A Survey of Issues, Malware Penetration, and Defenses," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 998-1022, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[45] Doan Minh Trung et al., "DMLDroid: Deep Multimodal Fusion Framework for Android Malware Detection with Resilience to Code Obfuscation and Adversarial Perturbations," *arXiv preprint*, pp. 1-17, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[46] Md. Habibullah Shakib, "Android Malware Detection using Transformer, Decoder and Encoder Models," *Array*, vol. 28, pp. 1-13, 2025. [CrossRef] [Google Scholar] [Publisher Link]

[47] Giuseppina Andresini et al., "Anakin: Explainable Android Malware Detection with Graph Neural Networks," *Cybersecurity*, vol. 9, no. 1, pp. 1-37, 2026. [CrossRef] [Google Scholar] [Publisher Link]

[48] Abhinandan Banik, and Jyoti Prakash Singh, "A BERT and PSO Framework for Android Malware Detection using Real Permissions and API Calls," *Discover Computing*, vol. 29, no. 1, pp. 1-38, 2026. [CrossRef] [Google Scholar] [Publisher Link]