Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Separating Behavioral Effect from Predictive Value in Dynamic Android Malware Detection: An Empirical Decomposition Study

Özgür TONKAL

ozgur.tonkal@samsun.edu.tr

Samsun University

Hakkı Batuhan SÜTÇÜ

Samsun University

Research Article

Keywords:

Posted Date: November 6th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11184308/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Separating Behavioral Effect from Predictive Value in Dynamic Android Malware Detection: An Empirical Decomposition Study

Anonymous Author[1*]

[1*]Anonymous Institution, Anonymous.

Corresponding author(s). E-mail(s): anonymous@anonymous.invalid;

Abstract

Dynamic Android malware detectors routinely combine behavioral views, but a reported score rarely separates class discrimination, execution-exposure effects, redundancy, and incremental value. This paper measures that separation empirically; it does not propose a new representation-learning or fusion mechanism. We use a study-specific paired corpus built from AndroZoo applications. After 300-s AndroPyTool executions, SHA-256 reconciliation and output-quality control reduced 13,654 audited unique APKs to a common predictive cohort of 11,457 applications (5,525 benign and 5,932 malware) with usable Strace and DroidBox outputs. We extracted 122 syscall features, actual execution duration, three compact same-PID sequence descriptors, and 74 DroidBox runtime features. Behavioral effects used rank-based statistics and Fisher exact tests under Benjamini-Hochberg correction; prediction used five-fold outer, three-fold inner nested cross-validation. Five core representations were tested across seven model families, compact-sequence increments with the three primary models, and paired class-stratified bootstrap analysis used pooled out-of-fold predictions. Execution duration was strongly associated with total syscall volume (Spearman $\boldsymbol{\rho \approx 0.727}$); per-runtime rates reduced the median absolute feature-duration association from 0.254 to 0.129 (about 49%) but did not consistently improve discrimination. Same-PID ordering was behaviorally rich, with 773 of 1,142 bigrams and 4,411 of 5,956 trigrams significant after this correction, yet the compact descriptors added little after fusion. Strace and DroidBox provided reciprocal incremental gains, and fusion outperformed the best single-modality representation across all seven model families, with XGBoost reaching the highest observed fold-mean ROC-AUC (0.947) and PR-AUC (0.952). Behavioral enrichment and predictive utility are therefore distinct properties of a dynamic representation.

Keywords: Android malware, Dynamic analysis, System calls, Empirical evaluation, Execution-duration normalization, Multimodal fusion

1 Introduction

Android remains a dominant mobile computing platform and therefore a persistent target for malicious applications. Google reported that in 2025 it prevented more than 1.75 million policy-violating apps from being published on Google Play, while Play Protect scanned more than 350 billion Android apps per day and its real-time scanning identified more than 27 million new malicious applications from sources outside Google Play [1]. Kaspersky separately reported a 56% year-on-year increase in Android Trojan-banker attacks in 2025 and 255,090 unique Trojan-banker

installation packages, a 271% increase over 2024 [2]. These figures are vendor telemetry rather than epidemiological estimates, but they illustrate the continuing scale and evolution of the mobile threat landscape. The detection problem is consequently not only one of identifying known signatures, but of constructing behavioral representations that remain informative when malware families, packing strategies, user interactions, and execution environments vary. Machine-learning-based Android malware detection has therefore evolved from permission- and API-centric static representations toward dynamic system calls, runtime events, behavior graphs, sequential models, and multimodal representations [3–6].

Dynamic analysis is attractive because it observes behavior that survives many source-level transformations, but it introduces its own experimental confounders. Two applications executed for different amounts of time do not have equal opportunity to emit syscalls or trigger higher-level events. Automated stimulation, emulator or real-device choice, sandbox visibility, anti-analysis behavior, and process scheduling can all change the measured trace, while code transformations can challenge static and signature-oriented detectors [7–12]. Consequently, a large behavioral effect or a high classification score does not by itself establish that a representation captures malware-specific information rather than exposure time, tooling artifacts, or information redundant with another modality.

System-call representations have a long history in mobile malware research. Crowdroid demonstrated behavior-based discrimination from Linux system-call activity [13], Deep4MalDroid introduced deep learning over syscall graphs [14], and later work used syscall sequences, recurrent models, ensembles, and temporal analyses [15–19]. At a higher abstraction level, dynamic sandboxes and behavior monitors expose network activity, information leakage, cryptographic use, file operations, services, receivers, and related semantics. Hybrid and multimodal approaches increasingly combine such views with static, graph-based, API, or behavioral features [20–25].

These advances leave several methodological questions insufficiently separated. Observation time is known to affect captured behavior, yet feature-level dependence on each APK's actual runtime and the effect of per-runtime rate normalization are rarely reported explicitly in Android malware evaluation. Syscall ordering is also established, but process-local transitions are often collapsed into a single global sequence. In addition, statistical enrichment does not imply incremental predictive utility: a motif may have a large odds ratio while adding little once occurrence statistics or a higher-level semantic view are available. Finally, although multimodal fusion is well established, reciprocal measurement of how much low-level Strace and high-level DroidBox views add to one another under identical held-out APK folds remains uncommon. These distinctions motivate the evidence-decomposition design used here.

The aim is measurement rather than mechanism design. We introduce no new representation-learning or fusion architecture; every representation and learner examined here is an established one. What the study contributes is evidence about how much each component actually carries under a single leakage-controlled protocol, together with a reusable procedure for asking that question of a new dynamic representation.

Dataset construction and pre-model auditing are integral to the evaluation design. Rather than starting from a precomputed benchmark feature table, we constructed a study-specific paired dynamic-analysis corpus from AndroZoo applications [26]. Applications were selected using explicit metadata criteria and dynamically analyzed to obtain low-level Strace traces and high-level DroidBox event streams. Before any feature modeling or classifier development, APK identities and dynamic-output validity were audited: 13,654 SHA-256-unique APKs (6,972 benign and 6,682 malware) formed the initial audited corpus, and output-quality control plus SHA-256 modality matching produced the common predictive cohort of 11,457 APKs (5,525 benign and 5,932 malware). This cohort is the population used for representation-matched machine-learning comparisons and pooled out-of-fold evaluation. Modality-specific behavioral statistics retain their corresponding analysis-ready denominators where applicable. The corpus additionally preserves actual runtime duration and PID-aware syscall ordering, enabling the nuisance, sequence, and cross-abstraction analyses developed in this study. The overall study workflow and cohort-construction sequence are summarized in Figure 1.

The study makes the following empirical contributions:

- We establish and pre-model audit the paired analytical cohort used throughout the study, reconciling 13,654 SHA-256-unique APK identities and dynamic outputs to a common Strace–DroidBox predictive cohort of 11,457 applications used consistently across representation-matched machine-learning comparisons.
- We quantify execution duration as a nuisance variable for syscall representations and show that per-runtime rates substantially reduce median duration dependence while producing model-family-dependent predictive effects.
- We analyze process-aware same-PID syscall ordering at scale, separating statistically enriched bigram/trigram behavior from the incremental predictive utility of a compact sequence representation.
- We quantify reciprocal cross-abstraction complementarity between Strace and DroidBox using identical held-out folds and paired class-stratified pooled out-of-fold bootstrap intervals, rather than relying on score differences alone.
- We test whether the core occurrence, rate, semantic, and fusion conclusions persist across logistic regression, random forest, histogram gradient boosting, RBF-SVM, ExtraTrees, XGBoost, and multilayer perceptron models; compact-sequence increments are evaluated separately with the three primary model families.

2 Related Work and Positioning

2.1 Evaluation realism and dynamic behavioral evidence

Android malware research is highly sensitive to dataset provenance, labeling, duplicate control, class balance, and time. DREBIN established an influential explainable static benchmark, while AndroZoo enabled large-scale longitudinal app collection [3, 26]. CICMalDroid2020 subsequently provided a labeled corpus with static and dynamic features for malware-category research [27]. Subsequent work showed that noisy antivirus consensus can undermine ground truth and that data freshness strongly influences measured performance [28, 29]. Miranda et al. [30] explicitly demonstrated how biased Android malware datasets can invalidate reported conclusions and proposed dataset-debiasing guidance, while Molina-Coronado et al. [31] identified duplicate samples, label attribution, class imbalance, evasion, and app evolution as major sources of optimistic evaluation. KronoDroid explicitly addressed time by combining static and dynamic features over a broad temporal span, and later studies demonstrated that concept drift and cross-device differences can materially distort Android malware evaluation [12, 32]. A recent field-wide review similarly argues that high reported accuracy does not make Android malware detection a solved problem because methodological flaws, dataset limitations, and long-term validity remain open challenges [33]. Recent longitudinal work further treats temporal drift as an explicit deployment problem rather than an implicit property of random cross-validation [34, 35].

Our study does not claim temporal generalization because reliable sample-level collection or first-seen timestamps are not preserved in the study corpus. Instead, the contribution is a carefully audited paired dynamic corpus and leakage-controlled within-corpus evaluation. This boundary is important: temporal generalization, family-disjoint generalization, and device transfer are distinct experimental questions.

Dynamic Android malware analysis ranges from high-level API or event monitoring to lower-level kernel and system-call observation. Crowdroid used system-call behavior for unsupervised anomaly detection; Afonso et al. combined dynamically observed Android API calls and system-call traces; DynaLog automated behavioral characterization; DL-Droid emphasized real-device execution and stateful stimulation; and Deep4MalDroid modeled syscall graphs with deep learning [8, 10, 13, 14, 36]. DroidCat later used app-level dynamic profiling as a complementary classification view [37]. More recent work has continued to exploit runtime system calls using conventional ensembles, sequence models, dataset construction, and semantic characterization [15, 17–19].

Observation conditions nevertheless matter. Küchler et al. showed that additional sandbox execution time changes the behavior exposed by malware, while Ficco demonstrated the value of

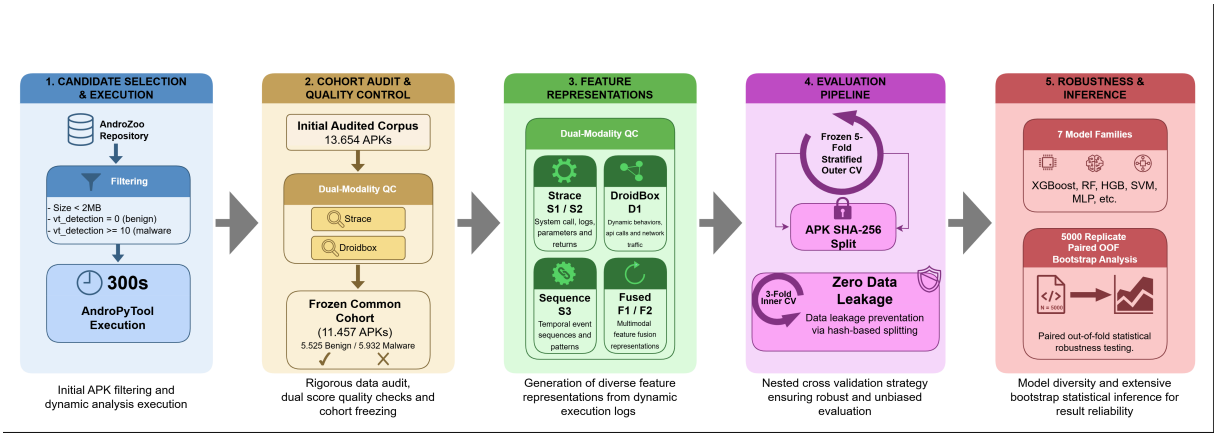


Fig. 1 Study overview. The initial audited set of 13,654 SHA-256-unique APKs (6,972 benign and 6,682 malware) was dynamically executed for 300 s, reconciled by SHA-256, and quality controlled to obtain the common paired cohort of 11,457 APKs (5,525 benign and 5,932 malware). The figure uses the S1–S4, D1, F1, and F2 representation identifiers defined in Table 3. Compact-sequence increments are evaluated with the three primary models, whereas the seven-model robustness benchmark covers the five core representations S1, S2, D1, F1, and F2.

combining observation windows and detectors in malware analysis [11, 38]. In Android specifically, Guerra-Manzanares et al. showed that dynamic system-call representations are affected by time and by the source device/environment [12, 16]. These studies motivate, but do not answer, our nuisance-control question: how strongly are individual syscall features coupled to each APK's actual runtime, and what changes when counts are converted to per-runtime rates?

2.2 Sequence modeling and multimodal behavior

Sequential structure has been explored at multiple abstraction levels. MaMaDroid, originally introduced at NDSS 2017 and subsequently expanded in a DOI-indexed journal version, models sequences of abstracted API calls as Markov chains, demonstrating that behavioral ordering can remain useful even when exact APIs evolve [39]. Sequence-oriented Android detection has also used API-call deep learning and opcode sequences [40, 41]. At the syscall level, recurrent and graph-based models have shown that ordering and local structural information can improve Android malware characterization [14, 15]. Semantics-aware approaches such as DroidSIFT similarly model dependencies among Android behaviors rather than treating features as independent counts [42]. Process-aware malware analysis outside the narrow Android-syscall setting further suggests that intra- and inter-process relations can reveal behavior obscured by globally aggregated traces.

Our sequence question is deliberately narrower. We preserve process identity when defining the primary transition semantics and test same-PID bigrams and trigrams. We then separate two claims that are often conflated: whether a motif is behaviorally enriched and whether a compact summary of process-aware order provides incremental predictive information after occurrence statistics and high-level runtime semantics are available.

Combining heterogeneous evidence is not new. DroidDetector associated static and dynamic features in a deep-learning detector; DroidFusion used classifier-level fusion; Kim et al. combined multiple Android feature families in a multimodal deep model; and later hybrid systems expanded this strategy [20, 43–46]. Gao et al. explicitly studied complementary features resilient to obfuscation; Chen et al. fused graph-attention and multimodal representations; and MPDroid introduced multimodal pre-training over static and dynamic information [21–23]. Recent hybrid and multi-view systems continue this trend, including the hybrid framework of Asmitha et al., LensDroid, and AppPoet [24, 25, 47].

Accordingly, our novelty claim is not 'multimodal fusion'. The specific contribution is

to quantify reciprocal incremental information between two dynamic abstraction levels collected for the same APK identities: Strace provides low-level system interaction while DroidBox exposes interpretable high-level runtime semantics. By using the same predefined test folds and paired out-of-fold predictions, we can distinguish a genuine modality increment from a change caused by different samples or partitions.

2.3 Measurement platforms and study positioning

Dynamic-analysis platforms differ substantially in what they expose and therefore in the behavioral representation they produce. DroidBox provides a fixed, interpretable high-level event ontology for network, file, leakage, cryptographic, service, receiver, SMS/call, and related Android behaviors [48, 49]. CopperDroid instead uses virtual-machine introspection to reconstruct operating-system and Android IPC semantics, including behavior originating in native code [9]. Frida offers flexible Java/native hooking and tracing, but its observations depend on the chosen hooks, scripts, and deployment mode, and instrumentation artifacts can themselves be detected [50, 51]. These tools are therefore not interchangeable measurement instruments. We retain DroidBox as the fixed semantic view already paired with Strace for the audited APK identities, without claiming that it is the preferred platform for a new deployment; transferability to VMI, standardized hook-based pipelines, and real devices remains an external-validation question.

Table 1 places the present work against closely related dataset, dynamic-analysis, sequence, multimodal, and temporal studies. The comparison emphasizes which ingredients are established and which combination is distinctive here.

3 Materials and Methods

3.1 Study design, dataset construction, and cohort audit

The study uses an evidence-decomposition design rather than a single end-to-end classifier comparison. Seven research questions are evaluated on a common APK SHA-256 identity base: RQ1, how benign and malicious applications differ in low- and high-level runtime behavior; RQ2, how predictive syscall occurrence is by itself; RQ3, whether execution duration behaves as a nuisance variable and whether per-runtime rates reduce that dependence; RQ4, whether process-aware syscall ordering is behaviorally enriched and incrementally predictive; RQ5, whether Strace and DroidBox provide complementary information; RQ6, the within-corpus discrimination achieved by fused dynamic representations; and RQ7, whether the core occurrence, rate, semantic, and fusion conclusions persist across model families. The compact sequence increments are evaluated separately with the three primary model families.

Candidate APKs were obtained from the AndroZoo research repository using its APK metadata and download infrastructure [26, 52]. AndroZoo metadata includes SHA-256 identifiers, APK size, and VirusTotal-derived detection information; the vt_detection field records the number of VirusTotal engines that flagged an APK as malicious. We used this field to define conservative operational labels: APKs with vt_detection=0 were eligible for the benign candidate class, whereas APKs with vt_detection$\geq$10 were eligible for the malware candidate class. Applications with intermediate detection counts (1–9) were not selected, reducing label ambiguity between the two operational classes. Candidate APKs were additionally restricted to files smaller than 2 MB. This threshold is an eligibility constraint on the study population rather than a predictive feature, and it is the strongest selection criterion applied in this work: both operational classes are drawn from the small-APK end of the Android size distribution. Because APK size covaries with code volume, and code volume plausibly covaries with how much behavior a fixed execution window can expose, the criterion may also interact with the duration analysis reported below. Its effect is not quantified against an unfiltered comparator population; the resulting limits on generalization are stated in the Threats to Validity section. No release-date, dex-date, market, or other temporal eligibility filter was applied.

Eligible APKs were dynamically analyzed with AndroPyTool, an automated Android analysis framework integrating DroidBox-based high-level behavioral monitoring and Strace-based low-level system-call tracing [53]. Each analyzed application was executed in the emulator environment

Table 1 Positioning of representative Android malware studies relative to the present work. The table emphasizes each study's principal analytical focus rather than treating unreported components as methodological deficiencies.

Study	Analysis / representation	Evaluation emphasis	Sequence / multimodal emphasis	Primary relevance
DREBIN [3]	Static permissions/API/code	Benchmark	Explainable static representation; dynamic order and fusion are not central contributions	Explainable static baseline
CopperDroid [9]	Dynamic VMI + IPC reconstruction	Sandbox measurement	Process/IPC semantics rather than paired low/high-level fusion	Instrumentation richness
DL-Droid [10]	Dynamic real-device runtime features	Device realism	Runtime behavior on real devices; process-aware ordering is not the central question	Execution realism
KronoDroid [32]	Hybrid static + dynamic features	Temporal dataset construction	Hybrid feature design with explicit time organization	Time-aware dataset construction
Guerra-Manzanares et al. [16]	Dynamic system calls	Temporal drift	System-call behavior under temporal change	System-call drift
MaMaDroid [39]	Static-sequential API Markov chains	Temporal discussion	Ordering at an abstracted API level	Sequence abstraction
Bhat et al. [17]	Dynamic system calls + ensembles	Conventional split	Occurrence-oriented syscall prediction with ensemble learners	Syscall prediction
MaDroid [18]	Dynamic runtime syscall sequences	Benchmark dataset	Sequence-focused, multi-featured representation	Modern sequence dataset
Chen et al. [22]	Graph + multimodal features	Conventional split	Graph structure and deep multimodal fusion	Deep fusion
MPDroid [23]	Static + dynamic pre-training	Conventional split	Multimodal pre-training; same-PID syscall order is not the central focus	Multimodal pre-training
LensDroid [25]	Complementary application-behavior views	Multiple scenarios	Multi-view complementary evidence	Complementary views
Sabbah et al. [35]	Static + dynamic longitudinal features	Chronological evaluation	Temporal drift and adversarial robustness	Longitudinal robustness
Present study	Paired Strace + DroidBox + same-PID descriptors	Nested CV; within-corpus	Runtime nuisance control, compact same-PID ordering, and paired cross-abstraction increments	Evidence decomposition and reciprocal complementarity

with a 300-s execution window. The retained study archive supports reproducibility of the post-execution audit and downstream analyses, but it does not preserve a complete environment specification sufficient to reconstruct every emulator and stimulation parameter independently. This limitation is stated explicitly below. Before feature engineering or classifier development, the resulting dynamic archive underwent an APK-level cohort audit. The initial audited corpus contained 13,654 SHA-256-unique APKs: 6,972 operationally benign and 6,682 operationally malicious applications. APK identities were reconciled across dynamic outputs using SHA-256 rather than filenames or execution-specific identifiers. Empty or invalid DroidBox records were excluded; among records with otherwise usable DroidBox output, two APKs lacked usable Strace traces. Only APKs with usable observations for both dynamic modalities were retained for paired predictive comparison. This quality-control and modality-matching procedure yielded a clean common predictive cohort of 11,457 APKs (5,525 benign and 5,932 malware). The cohort was defined before predictive modeling and reused across all representation-matched machine-learning experiments. In the reproducibility archive, this audit is materialized as the canonical clean_matched_index.parquet index; the audit logic precedes the downstream analysis notebooks.

The study-specific data resource used here is the audited paired dynamic behavioral corpus constructed from AndroZoo applications rather than a claim of originating the underlying APK binaries. Its analytical value lies in explicit APK identity control, pre-model dynamic-output quality control, aligned Strace and DroidBox observations for the same SHA-256 identities, measured execution duration, process-aware syscall ordering, predefined evaluation partitions, and out-of-fold predictions. The 11,457 applications constitute the common predictive evaluation cohort rather than the training size of any single model. Modality-specific descriptive and inferential analyses may retain their own analysis-ready denominator; in particular, the DroidBox behavioral-statistics table contains 11,459 valid records, whereas all paired predictive comparisons are restricted to the 11,457-APK common cohort. Because no temporal eligibility filter was imposed, sample-level temporal metadata were not incorporated into the evaluation table, and no chronological folds were constructed, the predictive evaluation estimates within-corpus rather than temporal generalization. The audited cohort sizes are summarized in Table 2, and the predictive representations used throughout the experiments are listed in Table 3.

3.2 Dynamic measurement and feature construction

DroidBox was retained as the fixed high-level semantic sensor because the central experiment requires paired low- and high-level observations for the same APK identities. Its event ontology provides interpretable runtime categories such as network activity, file operations, information leakage, cryptographic use, services, dynamically loaded code, receivers, SMS/calls, and identifier access [48, 49]. The choice is therefore about measurement consistency rather than technological superiority. CopperDroid provides richer VMI-based reconstruction of operating-system and Android IPC behavior [9], whereas Frida supports flexible Java/native instrumentation whose observations depend on the selected hooks, scripts, and deployment mode; instrumentation artifacts may themselves be detectable [50, 51].

Replacing DroidBox with CopperDroid, Frida, or another contemporary sensor would require re-executing the corpus and would change the measurement instrument and potentially the observed behavior. Such an experiment is valuable, but it addresses tool transferability rather than the controlled cross-abstraction comparison studied here. We therefore treat DroidBox as a legacy but controlled semantic sensor and reserve replication with contemporary VMI, standardized hook-based instrumentation, or real devices for external validation; DL-Droid illustrates why execution platform and stimulation realism can materially affect dynamic Android analysis [10].

Strace parsing produced a canonical 122-syscall vocabulary. For each APK, S1 contains the raw count of each syscall. Execution duration was derived from the trace timestamps; all 11,457 common-cohort samples had valid positive durations, including explicit correction of 41 midnight rollovers. To examine exposure-time dependence, we computed Spearman correlations

Table 2 Audited corpus and common predictive cohort.

Cohort	Total	Benign	Malware	Role
Initial audited corpus	13,654	6,972	6,682	SHA-256-unique APKs before paired-output QC
Common predictive cohort	11,457	5,525	5,932	Usable paired Strace-DroidBox cohort; fixed before modeling

Table 3 Predictive representations and their scientific roles.

ID	Representation	Modality	Scientific role
S1	122 raw syscall counts	Strace	Occurrence baseline
S2	122 duration-normalized syscall rates	Strace	Runtime nuisance-control representation
S3	3 compact process-aware sequence descriptors	Sequence	Sequence-only compact representation
S4	S2 + S3 (125)	Strace + sequence	Incremental ordering evaluation
D1	74 DroidBox runtime features	DroidBox	High-level semantic representation
F1	S2 + D1 (196)	Multimodal	Two-modality fusion
F2	S2 + D1 + S3 (199)	Multimodal	Complete primary representation

between feature values and actual runtime. S2 divides each syscall count by execution duration, yielding per-runtime syscall rates.

Two additional transformations—relative frequencies and centered log-ratio (CLR) features with a pseudocount of 0.1—were used only as nuisance-audit comparators, not as primary predictive representations. This distinction is deliberate: compositional normalization and runtime normalization answer different methodological questions. We also examined dependence on total syscall volume to test whether each transformation merely re-expresses activity scale.

The analysis-ready DroidBox table contained 11,459 valid records and 74 predictive features. Continuous/count features summarize event frequencies and diversity, including received actions, network endpoints, data leaks, file/network operations, cryptographic activity, and related semantics. Binary indicators capture the presence of interpretable behaviors such as identifier access and information leakage. High-cardinality identity strings and sample-specific tokens were excluded to reduce memorization and leakage risk. DroidBox behavioral statistics use this analysis-ready table; for D1, F1, and F2 predictive experiments, DroidBox features were restricted by SHA-256 to the 11,457-APK common cohort before nested cross-validation.

The sequence parser was independently reconciled with the canonical syscall parser: all 11,457 samples matched total syscall count and unique-syscall count exactly, with zero outside-vocabulary calls. The primary ordering semantics preserve process identity: a transition is counted only when adjacent calls belong to the same PID. Global chronology was retained as a sensitivity analysis, because interleaving multiple processes may introduce scheduler-driven transitions that are not local behavioral dependencies.

Same-PID bigrams were retained when document frequency was at least 1%, yielding 1,142 tested motifs; trigrams used a 0.5% threshold,

yielding 5,956 motifs. Behavioral prevalence differences were evaluated with Fisher exact tests and odds ratios under BH-FDR correction. For prediction, a deliberately compact S3 representation was defined with three descriptors—cross-PID transition ratio, same-PID self-transition ratio, and same-PID unique trigram count—after a redundancy audit showed very high correlations among broader sequence descriptors. Therefore, S3 tests the utility of this compact process-aware summary, not the full predictive capacity of thousands of n-grams.

3.3 Statistical analysis and representation ablations

Continuous Strace and DroidBox features were compared between benign and malware classes using the Mann–Whitney U test with rank-biserial correlation as the effect size. Binary/rare prevalence differences used Fisher exact tests with corrected odds ratios and 95% confidence intervals. Multiplicity was controlled with the Benjamini–Hochberg procedure at $q<0.05$ [54, 55]. These behavioral analyses use the corresponding analysis-ready modality table rather than forcing every descriptive statistic onto the predictive complete-case cohort; the paired predictive experiments, by contrast, use the 11,457-APK common cohort. Underflowed p- or q-values are reported using scientific lower-bound notation rather than as zero.

The experimental design uses controlled representation increments. S1 tests raw syscall occurrence; S2 tests duration-normalized rates; S3 tests the compact sequence summary; S4 adds S3 to S2; D1 tests DroidBox alone; F1 fuses S2 and D1; and F2 adds S3 to F1. This design supports direct questions about nuisance control (S1 versus S2), sequence increment (S2 versus S4 and F1 versus F2), and reciprocal modality complementarity (D1 versus F1 and S4 versus F2).

3.4 Predictive evaluation protocol

A five-fold stratified outer partition was defined from the pre-model common cohort with random state 42 and reused across all representation comparisons. APK SHA-256 was the split identity, and the integrity audit confirmed zero train/test SHA overlap in every model-representation-fold unit. Each of the 11,457 cohort APKs appears in exactly one outer test fold, so pooled OOF evaluation covers the complete cohort while no individual outer-fold model is trained on all 11,457 APKs. Hyperparameter selection was performed only within the corresponding outer-training partition using three-fold stratified inner cross-validation with fold-specific random states 43-47 (42 + outer-fold index). All scaling, transformation, and model fitting occurred within the training pipeline. No global supervised feature selection used outer-test information.

The primary benchmark used logistic regression, random forest, and histogram gradient boosting across all seven representations (S1–S4, D1, F1, and F2). A secondary robustness benchmark added RBF-kernel support-vector machines, ExtraTrees, XGBoost, and multilayer perceptrons for the five core representations S1, S2, D1, F1, and F2, producing 35 model–representation combinations and 175 outer-fold evaluations. Accordingly, seven-model robustness claims apply to the core occurrence/rate/semantic/fusion representations, whereas S3/S4 sequence increments are supported by the three primary model families. These families span linear, bagged-tree, boosted-tree, kernel, and neural decision functions [56–59].

ROC-AUC and PR-AUC were treated as the principal discrimination metrics; F1, balanced accuracy, sensitivity, specificity, precision, and accuracy were retained as supplementary diagnostics. PR-AUC was retained because it is sensitive to positive-class retrieval behavior and can be more informative than ROC-AUC as class prevalence changes [60, 61].

To quantify the uncertainty of representation increments, pooled out-of-fold predictions from the same APKs were paired and resampled with a class-stratified bootstrap of 5,000 replicates. For each comparison we report the observed delta in ROC-AUC or PR-AUC, a 95% bootstrap interval, and the direction of the difference across the five outer folds. This is prediction-level uncertainty conditional on the trained nested-CV models; it is not a formal hypothesis test over the entire stochastic training process [62].

4 Results

4.1 Behavioral differences and syscall baseline (RQ1–RQ2)

Strace and DroidBox both showed broad class-associated effects. Of the 122 relative syscall-frequency features, 104 were significant after BH-FDR correction. The largest rank-biserial effects were uname (r_rb=0.377), bind (0.376), connect (0.336), socket (0.326), and close (0.306), all enriched toward malware under the adopted sign convention. Rare syscall prevalence yielded still larger localized effects: chdir had a corrected odds ratio of approximately 70.70 (95% CI 29.22–171.03), while utimes and setsid showed large but less precise estimates because of sparse cells. Representative Strace effects are reported in Table 3.

DroidBox provided complementary high-level separation. Forty-nine of 57 tested continuous features were significant after FDR correction. The strongest continuous effects included received-action count (r_rb=0.409), unique received-action count (0.398), data-leak count (0.351), and unique opened-network-host count (0.329). Binary semantics also showed large prevalence ratios, including IMSI access (OR≈31.58), ICCID access (≈21.73), phone-number access (≈12.57), data-leak presence (≈10.86), and IMEI access (≈10.20). Representative DroidBox effects are reported in Table 4, and selected effects from both abstraction levels are visualized in Figure 2.

Raw syscall occurrence provided a strong within-corpus baseline. Logistic regression achieved mean ROC-AUC/PR-AUC of 0.833/0.847 on S1, while random forest and histogram gradient boosting achieved 0.923/0.934 and approximately 0.927/0.935, respectively. In the seven-model benchmark, XGBoost reached 0.929/0.938 on S1. These results confirm that low-level syscall composition contains substantial discriminative information even without high-level semantics or compact sequence descriptors.

4.2 Execution-duration nuisance and process-aware ordering (RQ3–RQ4)

Execution duration was strongly associated with total syscall volume (Spearman $\rho \approx 0.7266$). Across individual raw syscall features, the median absolute correlation with duration was 0.2536. Per-runtime rates reduced this value to 0.1288, corresponding to a 49.2% reduction in the typical feature-duration association. However, normalization did not eliminate duration dependence: the number of features with high residual correlation did not decrease monotonically, and the transformation should therefore be interpreted as nuisance reduction rather than complete nuisance removal.

The broader nuisance audit reinforced that transformation choice matters. Median absolute association with total syscall volume was 0.262 for raw counts, 0.365 for relative frequencies, 0.489 for CLR features, and 0.175 for per-runtime rates. Thus, compositional normalization is not interchangeable with exposure-time normalization. The corresponding duration- and activity-volume associations are summarized in Table 5 and visualized in Figure 3.

Predictively, the S2-S1 direction depended on the learner. RBF-SVM and MLP benefited from rates, whereas logistic regression, random forest, histogram gradient boosting, ExtraTrees, and XGBoost were modestly better on raw counts. This establishes an important distinction: reducing a nuisance association can improve representation validity without necessarily maximizing discrimination for every model family.

Same-PID sequence analysis revealed extensive class enrichment. Of 1,142 retained bigrams, 773 (67.7%) were significant after BH-FDR correction; of 5,956 retained trigrams, 4,411 (74.1%) were significant. Representative malware-enriched bigrams included close→chdir (malware prevalence 5.70% versus benign 0.07%; corrected OR≈74.24, 95% CI 29.27–188.32), ioctl→pipe (OR≈44.03), and mkdir→pread (OR≈34.12). Some trigrams were extremely enriched but sparse, including getuid32→writev→access (OR≈466.9), and are therefore interpreted with their wide confidence intervals. Aggregate same-PID sequence results and representative motifs are reported in Table 7, with selected signatures visualized in Figure 4.

Behavioral enrichment did not translate into equally large incremental discrimination for the three-feature compact sequence block. The S3 sequence-only representation was substantially weaker than S1/S2 or D1 (HGB ROC-AUC≈0.702, PR-AUC≈0.719). Adding S3 to rates

Table 3 Representative Strace behavioral effects. For relative frequencies, Effect is rank-biserial correlation and class values are medians. For rare-prevalence rows, Effect is corrected odds ratio and class values are prevalence percentages.

Family	Feature	Effect	Benign	Malware	95% CI	BH q
Relative frequency	uname	+0.377	0.0000	0.0000	—	< machine precision
Relative frequency	bind	+0.376	0.0000	0.0002	—	4.20e-268
Relative frequency	connect	+0.336	0.0002	0.0007	—	7.91e-212
Relative frequency	socket	+0.326	0.0003	0.0007	—	3.29e-199
Relative frequency	close	+0.306	0.0057	0.0122	—	6.80e-176
Relative frequency	poll	+0.289	0.0000	0.0000	—	1.57e-245
Relative frequency	rename	+0.270	0.0000	0.0000	—	1.76e-171
Relative frequency	fsetxattr	+0.268	0.0000	0.0000	—	1.31e-300
Relative frequency	getsockopt	+0.260	0.0002	0.0005	—	6.64e-127
Relative frequency	fsync	+0.250	0.0000	0.0001	—	4.50e-118
Rare prevalence	chdir	70.70	0.090	6.018	29.22–171.03	1.80e-94
Rare prevalence	utimes	49.58	0.000	0.438	3.02–813.82	1.86e-07
Rare prevalence	setsid	34.57	0.000	0.303	2.08–573.75	2.82e-05
Rare prevalence	ftruncate64	0.18	0.851	0.152	0.09–0.36	1.86e-07

(S2→S4) produced small changes; for HGB, the observed ROC-AUC increment was +0.0020 and its paired pooled-OOF bootstrap interval excluded zero. This interval is conditional on the single predefined outer partition and does not capture training-partition or seed-to-seed variability, so the result is interpreted as a small observed increment rather than as a general superiority claim. After Strace and DroidBox were already fused, F1→F2 deltas were near zero and all primary paired bootstrap intervals included zero. The result is therefore not that sequence information is uninformative, but that the tested compact sequence summary is largely redundant once the two main dynamic views are jointly available.

4.3 Cross-abstraction complementarity and multimodal discrimination (RQ5–RQ6)

The strongest and most stable predictive result was reciprocal modality complementarity. Adding Strace rates to DroidBox (D1→F1) improved ROC-AUC for logistic regression by 0.0313, random forest by 0.0324, and histogram gradient boosting by 0.0417; all three ROC intervals excluded zero and all five outer folds were positive. PR-AUC showed the same direction. Conversely, adding DroidBox to Strace-plus-sequence (S4→F2) improved ROC-AUC by 0.0795 for logistic regression, 0.0153 for random forest, and 0.0231 for histogram gradient boosting, again

Table 4 Representative DroidBox behavioral effects. Continuous rows use rank-biserial correlation; binary rows use corrected odds ratios and prevalence percentages.

Family	Feature	Effect	Benign	Malware	95% CI	BH q
Continuous	recvsaction_count	+0.409	0.000	1.000	—	< machine precision
Continuous	unique_received_action_count	+0.398	0.000	1.000	—	< machine precision
Continuous	dataleaks_count	+0.351	0.000	0.000	—	< machine precision
Continuous	unique_opennet_host_count	+0.329	0.000	1.000	—	5.26e-261
Continuous	dataleak_tag_taint_imei_count	+0.323	0.000	0.000	—	< machine precision
Continuous	unique_opennet_port_count	+0.314	0.000	1.000	—	6.06e-243
Continuous	network_total_event_count	+0.307	0.000	3.000	—	6.00e-225
Continuous	opennet_count	+0.306	0.000	1.000	—	1.02e-222
Continuous	path_dev_ratio	-0.304	0.059	0.042	—	1.45e-174
Continuous	network_send_ratio	+0.302	0.000	0.000	—	1.40e-258
Binary prevalence	dataleak_tag_taint_imsi_present	31.58	0.561	15.321	22.09–45.14	1.56e-226
Binary prevalence	dataleak_tag_taint_iccid_present	21.73	0.290	6.101	13.25–35.64	3.21e-83
Binary prevalence	dataleak_tag_taint_phone_number_present	12.57	0.434	5.292	8.32–18.99	1.27e-62
Binary prevalence	dataleaks_present	10.86	5.954	40.772	9.60–12.28	< machine precision

with positive paired evidence. The two modalities therefore contribute non-redundant information despite both being collected dynamically. The paired bootstrap estimates are reported in Table 7 and summarized visually in Figure 5.

Across the full seven-model benchmark, the top-performing region was dominated by F1 and F2. XGBoost F1 achieved mean ROC-AUC 0.9465±0.0011 and PR-AUC 0.9517±0.0015, while XGBoost F2 achieved 0.9473±0.0012 and 0.9524±0.0018. The corresponding pooled OOF values for XGBoost F2 were 0.9473 ROC-AUC and 0.9523 PR-AUC. These two aggregation modes are reported separately because the mean of fold metrics is not mathematically identical to a

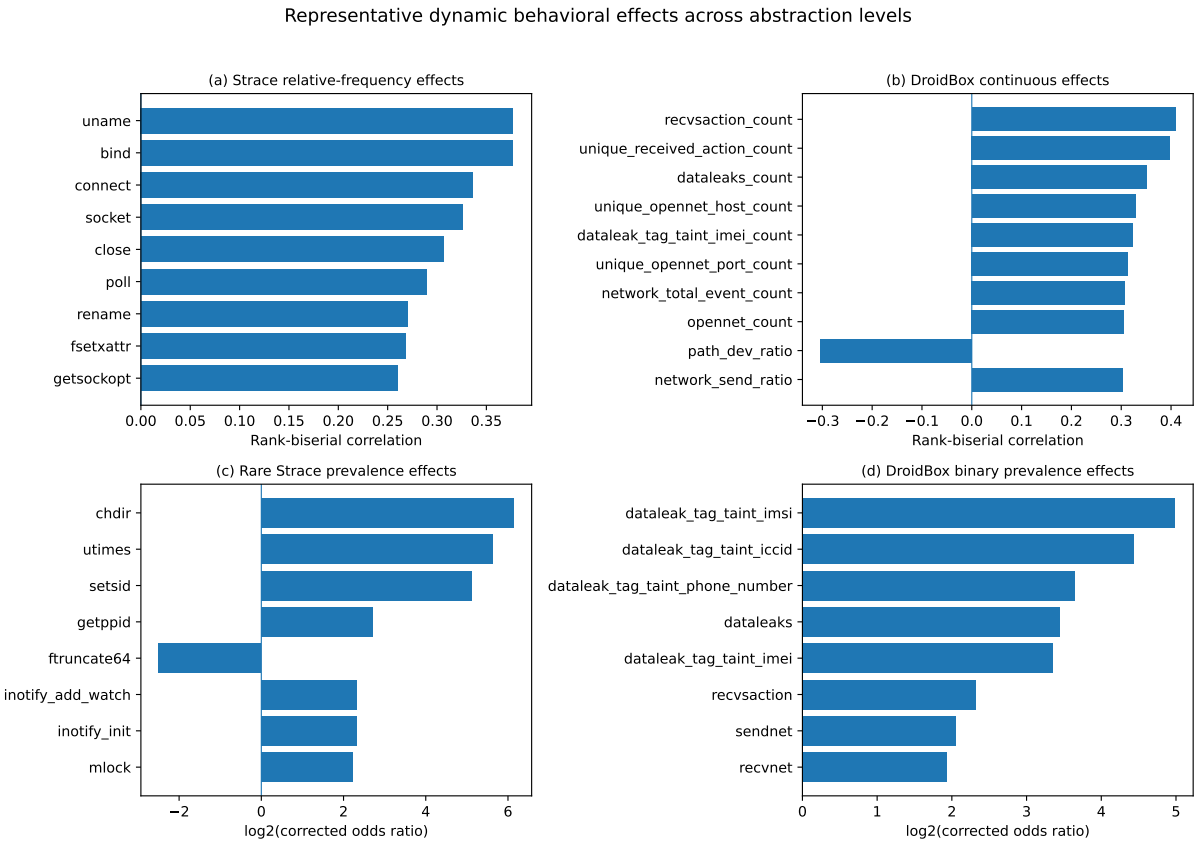


Fig. 2 Representative dynamic behavioral effects across abstraction levels. Panels (a) and (b) show signed rank-biserial effects for Strace and DroidBox continuous features. Panels (c) and (d) show $\log_2$-transformed corrected odds ratios for selected rare Strace and binary DroidBox events. Full multiplicity-corrected results are provided in Supplementary Tables S01–S04.

metric computed over the concatenated OOF predictions. The seven-model benchmark is reported in Table 8; pooled out-of-fold XGBoost ROC and precision–recall curves are shown in Figure 6. Exact model-family values are retained in the table rather than repeated in a second heatmap.

F2 was numerically the highest observed configuration, but the incremental F1→F2 evidence shows that the compact sequence descriptors add only marginally after Strace and DroidBox have been fused. Therefore, the scientific interpretation is that F1 captures nearly all of the multimodal gain, not that F2 is a substantively different predictor.

4.4 Model-family robustness and representation sensitivity (RQ7)

RQ7 concerns the five core representations S1, S2, D1, F1, and F2; the S3/S4 compact-sequence increments are not part of the seven-model robustness grid. For every evaluated model family, the best fusion representation (F1 or F2) exceeded the best single-modality representation (S1, S2, or D1) in mean ROC-AUC. The gain was approximately +0.032 for logistic regression, +0.011 for random forest, +0.019 for histogram gradient boosting, +0.047 for RBF-SVM, +0.017 for ExtraTrees, +0.018 for XGBoost, and +0.032 for MLP. PR-AUC showed the same qualitative pattern. In contrast, the effect of rates relative to raw counts changed sign across model families. The representation-effect sensitivity values are reported in Table 9 and summarized in Figure 7.

Table 5 Execution-duration and activity-volume nuisance audit.

Representation	Nuisance variable	Statistic	Interpretation
Total syscall count	Execution duration	0.7266	Global activity–runtime association
Raw syscall counts	Duration	0.2536	Median $\|\rho\|$ across 122 features
Duration-normalized rates	Duration	0.1288	49.2% lower than raw median $\|\rho\|$
Raw syscall counts	Total syscall volume	0.2622	Feature-volume dependence
Relative frequencies	Total syscall volume	0.3646	Higher than raw
CLR (pseudocount 0.1)	Total syscall volume	0.4891	Highest of tested transforms
Duration-normalized rates	Total syscall volume	0.1749	Lowest of tested transforms

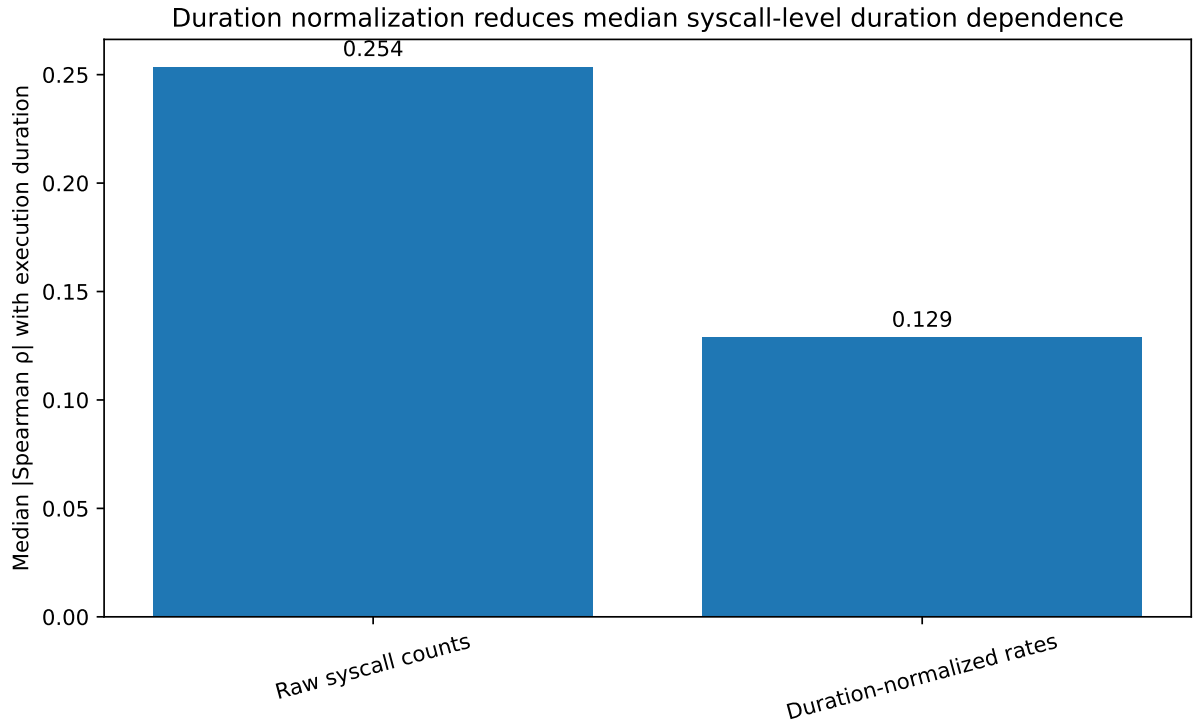


Fig. 3 Execution-duration dependence before and after per-runtime normalization. Per-runtime rates substantially reduce the median absolute feature–duration association but do not eliminate nuisance dependence.

5 Discussion

5.1 Behavioral evidence, nuisance control, and sequence utility

The behavioral results show that benign and malicious applications differ across both low-level

Table 6 Representative same-PID sequence motifs, continuing Table 7.

Table 7 Process-aware same-PID sequence evidence.

Family	Retained/tested	BH-FDR significant	Significant (%)	Retention threshold
Same-PID bigrams	1,142	773	67.7%	DF $\geq$1%
Same-PID trigrams	5,956	4,411	74.1%	DF $\geq$0.5%

Table 8 Representative same-PID sequence motifs, continuing Table 7.

Representative motif	Type	Malware prev.	Benign prev.	Corrected OR	95% CI / note
close→chdir	Bigram	5.698%	0.072%	74.24	29.27–188.32
chdir→execve	Bigram	5.698%	0.072%	74.24	29.27–188.32
ioctl→pipe	Bigram	—	—	44.03	See full table
mkdir→pread	Bigram	—	—	34.12	See full table
getsockopt→ fstat64	Bigram	—	—	13.82	See full table
uname→futex	Bigram	—	—	0.077	Benign-enriched
getuid32→ writev→access	Trigram	sparse	sparse	466.89	29.11–7488.15
madvise→ munmap→open	Trigram	sparse	sparse	257.36	Wide CI

and high-level runtime views. More than 100 syscall-relative features and most tested DroidBox continuous features survived multiplicity correction, indicating that class-associated behavior is distributed rather than confined to a small signature set. This is consistent with prior dynamic Android studies using syscalls, APIs, and behavioral graphs [8, 13, 17, 18]. At the same time, the extreme odds ratios observed for some rare syscalls and sequence motifs should not be interpreted as direct measures of predictive necessity: prevalence effects and incremental information answer different questions.

Simple syscall occurrence nevertheless remains a strong baseline. Tree and boosting models achieve high discrimination without sequence or high-level semantic features, reinforcing the need to compare richer representations against competent occurrence baselines rather than against weak reference models. This observation is consistent with the continued effectiveness of syscall-based Android detection [18, 19].

The duration analysis separates representation validity from predictive optimization. A correlation of approximately 0.73 between runtime and total syscall count shows that exposure opportunity is a major component of the raw trace. Per-runtime rates substantially reduce the typical feature-duration association, yet the raw-versus-rate predictive ordering changes across learners. A representation can therefore be less coupled to a nuisance variable without becoming the most discriminative input for every model family.

The comparison with relative-frequency and CLR transformations further indicates that not all normalizations address the same nuisance. Relative and CLR representations can still remain strongly associated with overall activity volume. This distinction should be considered in future dynamic malware studies whenever observation windows vary between samples.

Process-aware ordering adds a second caution. Same-PID n-grams produced thousands of significant effects and several very large odds ratios, supporting the premise that behavioral order contains useful structure [15, 39]. However, the predictive experiment deliberately used only three compact sequence descriptors after a redundancy audit. Their small increment, especially after Strace and

Table 7 Paired class-stratified pooled out-of-fold bootstrap analysis (5,000 replicates). Intervals quantify prediction-level uncertainty conditional on the trained nested-CV models.

Comparison	Model	Metric	Observed Δ	95% bootstrap CI	CI excludes 0	Positive folds
D1→F1	HGB	PR-AUC	+0.0384	[+0.0341, +0.0426]	Yes	5/5
D1→F1	HGB	ROC-AUC	+0.0417	[+0.0377, +0.0456]	Yes	5/5
D1→F1	LR	PR-AUC	+0.0253	[+0.0198, +0.0307]	Yes	5/5
D1→F1	LR	ROC-AUC	+0.0313	[+0.0272, +0.0354]	Yes	5/5
D1→F1	RF	PR-AUC	+0.0275	[+0.0238, +0.0314]	Yes	5/5
D1→F1	RF	ROC-AUC	+0.0324	[+0.0287, +0.0362]	Yes	5/5
F1→F2	HGB	PR-AUC	-0.0002	[-0.0016, +0.0012]	No	3/5
F1→F2	HGB	ROC-AUC	+0.0005	[-0.0006, +0.0016]	No	3/5
F1→F2	LR	PR-AUC	+0.0004	[-0.0005, +0.0013]	No	4/5
F1→F2	LR	ROC-AUC	+0.0005	[-0.0004, +0.0014]	No	3/5
F1→F2	RF	PR-AUC	+0.0006	[-0.0008, +0.0019]	No	3/5
F1→F2	RF	ROC-AUC	+0.0007	[-0.0002, +0.0017]	No	4/5
S2→S4	HGB	PR-AUC	+0.0012	[-0.0004, +0.0028]	No	4/5
S2→S4	HGB	ROC-AUC	+0.0020	[+0.0005, +0.0036]	Yes	5/5
S2→S4	LR	PR-AUC	+0.0004	[-0.0006, +0.0014]	No	5/5
S2→S4	LR	ROC-AUC	+0.0005	[-0.0006, +0.0016]	No	3/5
S2→S4	RF	PR-AUC	-0.0000	[-0.0011, +0.0011]	No	3/5
S2→S4	RF	ROC-AUC	+0.0005	[-0.0003, +0.0014]	No	4/5
S4→F2	HGB	PR-AUC	+0.0195	[+0.0164, +0.0228]	Yes	5/5
S4→F2	HGB	ROC-AUC	+0.0231	[+0.0201, +0.0262]	Yes	5/5
S4→F2	LR	PR-AUC	+0.0723	[+0.0653, +0.0792]	Yes	5/5
S4→F2	LR	ROC-AUC	+0.0795	[+0.0728, +0.0858]	Yes	5/5
S4→F2	RF	PR-AUC	+0.0114	[+0.0083, +0.0144]	Yes	5/5
S4→F2	RF	ROC-AUC	+0.0153	[+0.0121, +0.0185]	Yes	5/5

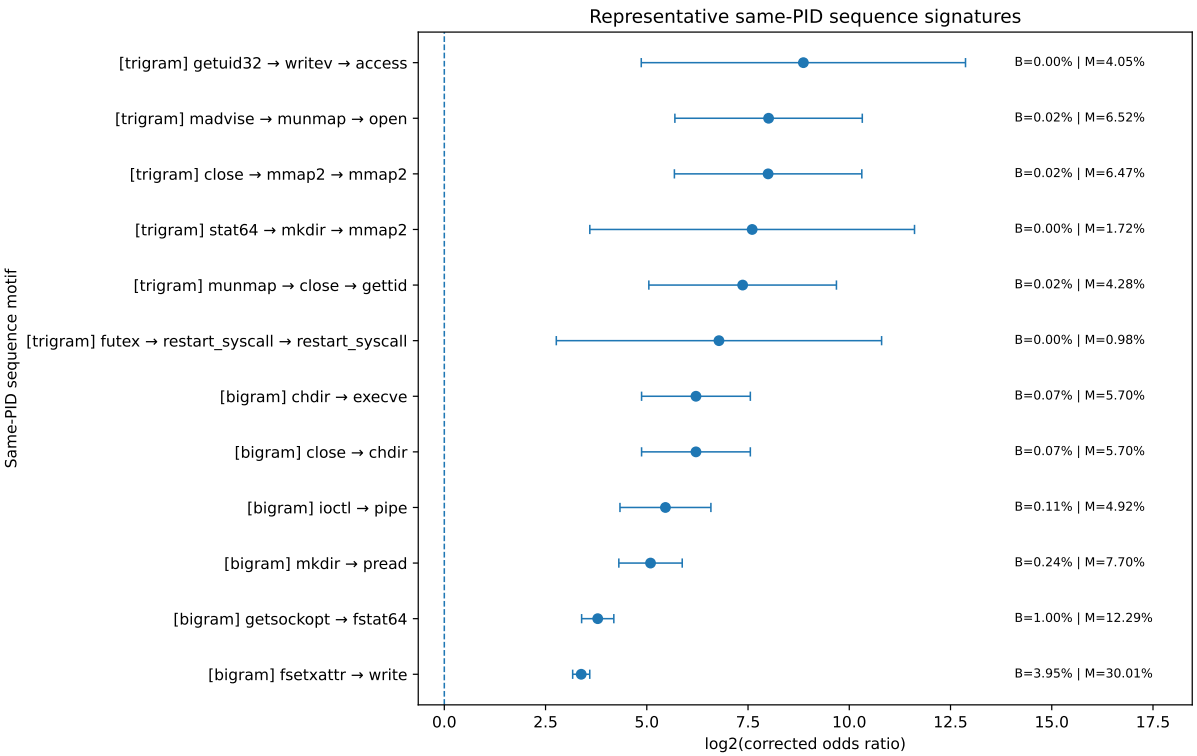


Fig. 4 Representative same-PID sequence signatures. Corrected odds ratios are shown on a logarithmic scale; extreme sparse effects should be interpreted together with confidence intervals and prevalence.

Table 8 Seven-model robustness benchmark. Cells report fold-mean ROC-AUC/PR-AUC; complete standard deviations and secondary metrics are provided in Supplementary Table S08 (Online Resource 1).

Model	S1	S2	D1	F1	F2
Logistic Regression	0.833/0.847	0.818/0.830	0.866/0.877	0.898/0.902	0.898/0.903
Random Forest	0.923/0.934	0.918/0.929	0.901/0.912	0.933/0.940	0.934/0.941
HistGradient Boosting	0.927/0.935	0.920/0.930	0.903/0.914	0.945/0.950	0.945/0.950
RBF-SVM	0.858/0.868	0.872/0.879	0.877/0.883	0.923/0.926	0.924/0.927
ExtraTrees	0.922/0.934	0.918/0.929	0.903/0.914	0.938/0.944	0.939/0.945
XGBoost	0.929/0.938	0.925/0.934	0.906/0.917	0.947/0.952	0.947/0.952
MLP	0.882/0.889	0.888/0.895	0.889/0.897	0.920/0.923	0.921/0.925

DroidBox were fused, should therefore be interpreted as evidence about this compact summary under the tested protocol rather than as evidence that sequence modeling in general is uninformative.

5.2 Cross-abstraction complementarity and model-family robustness

The most stable predictive finding is reciprocal cross-abstraction complementarity. Strace improves DroidBox, and DroidBox improves Strace plus the compact sequence block. This

Table 9 Representation-effect sensitivity across model families.

Model	Δ ROC-AUC (S2-S1)	Best fusion - best single-modality ROC-AUC
Logistic Regression	-0.0146	+0.0319
Random Forest	-0.0051	+0.0110
HistGradientBoosting	-0.0065	+0.0185
RBF-SVM	+0.0145	+0.0469
ExtraTrees	-0.0044	+0.0165
XGBoost	-0.0042	+0.0178
MLP	+0.0063	+0.0317

reciprocal pattern is more informative than a single fusion score because it shows that each abstraction level contains information not fully represented by the other: syscalls expose generic operating-system interaction, whereas DroidBox summarizes semantically meaningful Android events. The finding is compatible with prior multimodal and complementary-feature studies, but the paired design here isolates the increment on the same APK identities and held-out folds.

The representation-level conclusions also translate into high within-corpus discrimination. XGBoost F2 reached approximately 0.947 ROC-AUC and 0.952 PR-AUC by fold mean, with closely matching pooled OOF values. These scores are not used to claim state-of-the-art superiority over studies built on different corpora, label policies, Android versions, stimulation methods, or temporal protocols. More importantly, F1 captured nearly all of the fusion gain, so the main conclusion is complementarity between Strace and DroidBox rather than a substantive advantage from adding the compact sequence block to the fused representation.

Robustness across model families further separates stable findings from learner-specific effects. Fusion improved on the best single-modality representation for all seven model families, whereas the raw-versus-rate preference changed sign across learners. This contrast strengthens the complementarity conclusion while cautioning against presenting any preprocessing transformation as universally beneficial.

5.3 Practical evidence-decomposition checklist

For practitioners evaluating a new dynamic malware representation, the study suggests a compact five-step protocol before interpreting a high classification score as evidence of a useful behavioral signal. First, fix sample identity and cohort membership before model development. Second, quantify dependence on exposure-related variables such as actual runtime or total activity volume. Third, separate behavioral enrichment from incremental predictive value. Fourth, test modality increments on the same held-out samples rather than comparing scores obtained from different cohorts or partitions. Fifth, distinguish conclusions that persist across learner families from effects that are specific to a model or preprocessing choice. This checklist summarizes the analytical logic used in the present study; it is a decision procedure for interpreting representations, not a claim that every component requires the same sensor or classifier.

5.4 Dataset, tooling, and practical implications

The paired corpus is useful not only because it preserves multiple analytical layers, but because its common predictive population was established before modeling. SHA-256 reconciliation and modality-specific output-quality control reduced an initial audited set of 13,654 unique APKs to a paired predictive cohort of 11,457 applications with usable Strace and DroidBox observations. This pre-model complete-case rule makes representation and model-family comparisons refer

to the same APK population, although exclusion of incomplete dynamic executions can itself induce selection effects. Modality-specific behavioral statistics retain their analysis-ready populations where appropriate, so this complete-case rule should not be misread as retroactively redefining every descriptive table. The archived analytical layers include SHA-256 identity, low-level traces, high-level events, actual runtime, PID-aware order, outer-fold assignments, and OOF predictions. KronoDroid demonstrates the value of dataset design that explicitly records time and multiple feature sources, while DL-Droid demonstrates the importance of real-device execution [10, 32]. The analytical contribution of this paired resource is complementary: paired abstraction levels, explicit cohort auditing, and nuisance-aware derivatives are central, but sample-level temporal metadata were not incorporated into the present evaluation table and no chronological partition was constructed. A SHA-256-based temporal enrichment from source metadata is therefore a feasible extension rather than a validated result of the current study.

The DroidBox choice also frames future work. Replicating the same paired-increment experiment with CopperDroid-style VMI, a standardized Frida hook specification, or modern real-device instrumentation would reveal whether the observed complementarity is tool-independent. Such a study should keep sample identity, stimulation, and runtime exposure controlled so that changes can be attributed to the measurement platform rather than to a different corpus.

6 Threats to Validity

6.1 Measurement and construct validity

Dynamic traces are partial observations of application behavior. Non-triggered code, environment-dependent payloads, anti-analysis logic, and stimulation coverage can limit what is observed. DroidBox is a legacy Dalvik-era framework and does not provide the instrumentation breadth of modern VMI or hook-based systems, while Strace records low-level interaction without directly encoding application intent. The operational labels are also derived from AndroZoo VirusTotal metadata: vt_detection=0 defines the benign candidate class and vt_detection$\geq$10 the malware candidate class, with intermediate values excluded. These rules reduce ambiguity but do not constitute absolute ground truth. Rate normalization reduces duration dependence but cannot remove every exposure or activity-level effect.

The $<$ 2 MB APK-size eligibility criterion is the strongest selection constraint in this study and warrants separate emphasis. Applications distributed in practice, and benign store-distributed applications in particular, are frequently much larger, so neither operational class here is representative of the size distribution encountered in deployment. Because APK size covaries with code volume, the criterion may shift class composition and may also interact with how much behavior a fixed 300-s execution window can expose, which is the same quantity examined in the duration analysis. Quantifying this size-selection effect would require executing an unrestricted candidate set under the same protocol and comparing the resulting cohorts; that comparator was not collected, so the effect is reported here as an acknowledged and unquantified limitation. Results should accordingly be read as applying to small-APK populations analyzed under this protocol rather than to the full Android application-size distribution.

The duration analysis quantifies runtime dependence of syscall activity, but it does not separately estimate class-specific runtime distributions or duration-only discrimination. Runtime is therefore treated here as an operational exposure-related nuisance variable; the current evidence does not establish whether runtime itself is also class-associated.

The retained archive does not preserve a complete environment-level specification for every emulator and stimulation parameter (for example, all image/API details, stimulation settings, network state, reset policy, and parallel execution settings). Consequently, the reproducibility claim is strongest for cohort reconciliation, feature construction, statistical analysis, nested cross-validation, and result generation from the retained outputs, rather than for exact re-creation of the original dynamic-execution environment.

6.2 Evaluation and statistical validity

APK SHA-256 was used first to reconcile the paired dynamic corpus and subsequently as the split identity. The common predictive cohort was fixed before modeling, and outer train/test SHA-256 overlap was verified to be zero. Preprocessing and tuning occurred within training partitions, reducing identity leakage and preventing performance-driven cohort selection. These controls do not guarantee malware-family separation, and near-duplicate applications from the same family may remain. In addition, complete-case selection requires usable outputs from both dynamic modalities and can therefore favor applications that execute successfully in the analysis environment.

Behavioral statistics use multiplicity correction and effect sizes, but sparse motifs can still yield extreme odds ratios with wide confidence intervals. The paired bootstrap quantifies uncertainty in OOF prediction differences conditional on the fitted models; it does not propagate every source of training-process uncertainty. Intervals including zero for F1→F2 should therefore be read as a negligible observed increment under the tested protocol, not as proof of exact equality.

6.3 External validity and temporal scope

The study estimates leakage-controlled within-corpus generalization, not chronological future-app performance. No temporal eligibility filter was applied during candidate selection, sample-level temporal metadata were not incorporated into the evaluation table, and chronological folds were not constructed. Temporal drift, family-disjoint transfer, cross-device and Android-version transfer, geographic variation, adversarial adaptation, and zero-day deployment are therefore outside the validated scope. This boundary is important because prior work shows that dataset bias, duplication, labeling policy, app evolution, and device/environment changes can materially alter reported Android malware performance [29–31, 33].

7 Conclusion

This study decomposes dynamic Android malware evidence rather than treating a high classification score as a sufficient explanation. Across 11,457 APKs with paired Strace and DroidBox views, low- and high-level runtime behavior differed extensively between benign and malicious applications. Actual execution duration was strongly coupled to syscall volume, and per-runtime rates reduced typical duration dependence by approximately half, although their predictive effect remained model-family dependent. Process-aware same-PID ordering exposed thousands of enriched motifs, yet the tested three-descriptor compact sequence summary added little in the three primary-model comparisons once syscall rates and high-level semantics were jointly represented. This should not be generalized to richer n-gram or sequence models. In contrast, Strace and DroidBox showed robust reciprocal complementarity: each added measurable information beyond the other, and fusion gains for the five core representations persisted across seven model families.

The central contribution is therefore empirical rather than architectural: the study shows that behavioral enrichment, nuisance dependence, redundancy, incremental value, and robustness across learners are separable properties of a dynamic representation, and that a representation can score well on one while contributing nothing on another. Future work should extend this framework to timestamped collections, family-disjoint protocols, contemporary VMI or Frida-based instrumentation, real devices, and explicit adversarial or concept-drift evaluation. Such extensions would test whether the cross-abstraction complementarity observed here transfers beyond the present leakage-controlled within-corpus setting.

Supplementary Information

The supplementary document (Online Resource 1) contains the full feature-level statistics, the sequence-motif tables, the redundancy audit, the incremental-value uncertainty analyses, the fold-level results for all model–representation combinations, and the selected hyperparameters. The machine-readable versions of those tables are provided as a data archive (Online Resource 2).

Statements and Declarations

Competing Interests

The authors have no relevant financial or non-financial interests to disclose.

Funding

The authors did not receive support from any organization for the submitted work.

Ethics Approval

Not applicable. This study did not involve human participants or animals. It analyses publicly catalogued Android application packages obtained from a research repository.

Data Availability

The underlying Android application package binaries were obtained from AndroZoo and cannot be redistributed by the authors. Study-specific sample-level and derived artifacts are not publicly released because they remain part of ongoing research and are subject to applicable access and redistribution constraints. Access may be considered by the corresponding author upon reasonable request.

Code Availability

The analysis code and reproducibility documentation are archived in a public repository with a versioned archival release. The identifying repository links are withheld here to preserve double-anonymous review and are supplied through the submission system.

Author Contributions

Author contributions are supplied through the submission system, as naming the individual contributors here would breach double-anonymous review.

Declaration of Generative AI and AI-Assisted Technologies in the Manuscript Preparation Process

During the preparation of this work, the authors used ChatGPT (OpenAI) in order to support English language refinement and editorial formatting. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

References

[1] Kaza, V.: Keeping Google Play & Android app ecosystems safe in 2025. Google Security Blog, February 19, 2026. Accessed September 20, 2026 (2026). https://blog.google/security/keeping-google-play-android-app-ecosystem-safe-2025/

[2] Kaspersky: The number of Trojan banker attacks on smartphones increased by 56% in 2025. March 5, 2026; accessed September 20, 2026 (2026). https://www.kaspersky.co.uk/about/press-releases/the-number-of-trojan-banker-attacks-on-smartphones-increased-by-56-in-2025

[3] Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., Rieck, K.: DREBIN: Effective and explainable detection of Android malware in your pocket. In: Network and Distributed System Security Symposium (2014). https://doi.org/10.14722/ndss.2014.23247

[4] Islam, R., Sayed, M.I., Saha, S., Hossain, M.J., Masud, M.A.: Android malware classification using optimum feature selection and ensemble machine learning. Internet of Things and Cyber-Physical Systems **3**, 100–111 (2023) https://doi.org/10.1016/j.iotcps.2023.03.001

[5] Muzaffar, A., Hassen, H.R., Lones, M.A., Zantout, H.: An in-depth review of machine learning based Android malware detection. Computers & Security **121**, 102833 (2022) https://doi.org/10.1016/j.cose.2022.102833

[6] Rahima Manzil, H.H., Naik, S.M.: Detection approaches for Android malware: Taxonomy and review analysis. Expert Systems with Applications **238**, 122255 (2024) https://doi.org/10.1016/j.eswa.2023.122255

[7] Rastogi, V., Chen, Y., Jiang, X.:

DroidChameleon: Evaluating Android anti-malware against transformation attacks. In: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, pp. 329–334 (2013). https://doi.org/10.1145/2484313.2484355

[8] Afonso, V.M., Amorim, M.F., Grégio, A.R.A., Junquera, G.B., Geus, P.L.: Identifying Android malware using dynamically obtained features. Journal of Computer Virology and Hacking Techniques **11**(1), 9–17 (2015) https://doi.org/10.1007/s11416-014-0226-7

[9] Tam, K., Khan, S.J., Fattori, A., Cavallaro, L.: CopperDroid: Automatic reconstruction of Android malware behaviors. In: Network and Distributed System Security Symposium (2015). https://doi.org/10.14722/ndss.2015.23145

[10] Alzaylaee, M.K., Yerima, S.Y., Sezer, S.: DL-Droid: Deep learning based Android malware detection using real devices. Computers & Security **89**, 101663 (2020) https://doi.org/10.1016/j.cose.2019.101663

[11] Küchler, A., Mantovani, A., Han, Y., Bilge, L., Balzarotti, D.: Does every second count? Time-based evolution of malware behavior in sandboxes. In: Network and Distributed System Security Symposium (2021). https://doi.org/10.14722/ndss.2021.24475

[12] Guerra-Manzanares, A., Luckner, M., Bahsi, H.: Concept drift and cross-device behavior: Challenges and implications for effective Android malware detection. Computers & Security **120**, 102757 (2022) https://doi.org/10.1016/j.cose.2022.102757

[13] Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: Behavior-based malware detection system for Android. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 15–26 (2011). https://doi.org/10.1145/2046614.2046619

[14] Hou, S., Saas, A., Chen, L., Ye, Y.: Deep4MalDroid: A deep learning framework for Android malware detection based on Linux kernel system call graphs. In: IEEE/WIC/ACM International Conference on Web Intelligence Workshops, pp. 104–111 (2016). https://doi.org/10.1109/WIW.2016.040

[15] Xiao, X., Zhang, S., Mercaldo, F., Hu, G., Sangaiah, A.K.: Android malware detection based on system call sequences and LSTM. Multimedia Tools and Applications **78**, 3979–3999 (2019) https://doi.org/10.1007/s11042-017-5104-0

[16] Guerra-Manzanares, A., Luckner, M., Bahsi, H.: Android malware concept drift using system calls: Detection, characterization and challenges. Expert Systems with Applications **206**, 117200 (2022) https://doi.org/10.1016/j.eswa.2022.117200

[17] Bhat, P., Behal, S., Dutta, K.: A system call-based Android malware detection approach with homogeneous & heterogeneous ensemble machine learning. Computers & Security **130**, 103277 (2023) https://doi.org/10.1016/j.cose.2023.103277

[18] Duan, G., Liu, H., Cai, M., Sun, J., Chen, H.: MaDroid: A maliciousness-aware multifeatured dataset for detecting Android malware. Computers & Security **144**, 103969 (2024) https://doi.org/10.1016/j.cose.2024.103969

[19] Malik, S., Singh, N., Tripathy, S.: Semantic characterization of Android malware through runtime system call analysis. Journal of Information Security and Applications **98**, 104406 (2026) https://doi.org/10.1016/j.jisa.2026.104406

[20] Kim, T., Kang, B., Rho, M., Sezer, S., Im, E.G.: A multimodal deep learning method for Android malware detection using various features. IEEE Transactions on Information Forensics and Security **14**(3), 773–788 (2019) https://doi.org/10.1109/TIFS.2018.2866319

[21] Gao, C., Cai, M., Yin, S., Huang, G., Li, H., Yuan, W., Luo, X.: Obfuscation-resilient

Android malware analysis based on complementary features. IEEE Transactions on Information Forensics and Security **18**, 5056–5068 (2023) https://doi.org/10.1109/TIFS.2023.3302509

[22] Chen, S., Lang, B., Liu, H., Chen, Y., Song, Y.: Android malware detection method based on graph attention networks and deep fusion of multimodal features. Expert Systems with Applications **237**, 121617 (2024) https://doi.org/10.1016/j.eswa.2023.121617

[23] Zhang, S., Su, H., Liu, H., Yang, W.: MPDroid: A multimodal pre-training Android malware detection method with static and dynamic features. Computers & Security **150**, 104262 (2025) https://doi.org/10.1016/j.cose.2024.104262

[24] Asmitha, K.A., Vinod, P., Rehiman, R.K.A., Raveendran, N., Conti, M.: Android malware defense through a hybrid multi-modal approach. Journal of Network and Computer Applications **233**, 104035 (2025) https://doi.org/10.1016/j.jnca.2024.104035

[25] Meng, Z., Zhang, J., Guo, J., Wang, W., Huang, W., Cui, J., Zhong, H., Xiong, Y.: LensDroid: Detecting Android malware by visualizing app behaviors from multiple complementary views. IEEE Transactions on Information Forensics and Security **20**, 2915–2929 (2025) https://doi.org/10.1109/TIFS.2025.3547301

[26] Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y.: AndroZoo: Collecting millions of Android apps for the research community. In: Proceedings of the 13th International Conference on Mining Software Repositories, pp. 468–471 (2016). https://doi.org/10.1145/2901739.2903508

[27] Mahdavifar, S., Abdul Kadir, A.F., Fatemi, R., Alhadidi, D., Ghorbani, A.A.: Dynamic Android malware category classification using semi-supervised deep learning. In: IEEE DASC/PiCom/CBDCom/CyberSciTech, pp. 515–522 (2020). https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094

[28] Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W.: Deep ground truth analysis of current Android malware. In: Detection of Intrusions and Malware, and Vulnerability Assessment. Lecture Notes in Computer Science, vol. 10327, pp. 252–276 (2017). https://doi.org/10.1007/978-3-319-60876-1_12

[29] Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., Cavallaro, L.: TESSERACT: Eliminating experimental bias in malware classification across space and time. In: 28th USENIX Security Symposium, pp. 729–746 (2019). https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury

[30] Miranda, T.C., Gimenez, P.-F., Lalande, J.-F., Viet Triem Tong, V., Wilke, P.: Debiasing Android Malware Datasets: How Can I Trust Your Results If Your Dataset Is Biased? IEEE Transactions on Information Forensics and Security **17**, 2182–2197 (2022) https://doi.org/10.1109/TIFS.2022.3180184

[31] Molina-Coronado, B., Mori, U., Mendiburu, A., Miguel-Alonso, J.: Towards a fair comparison and realistic evaluation framework of Android malware detectors based on static analysis and machine learning. Computers & Security **124**, 102996 (2023) https://doi.org/10.1016/j.cose.2022.102996

[32] Guerra-Manzanares, A., Bahsi, H., Nõmm, S.: KronoDroid: Time-based hybrid-featured dataset for effective Android malware detection and characterization. Computers & Security **110**, 102399 (2021) https://doi.org/10.1016/j.cose.2021.102399

[33] Guerra-Manzanares, A.: Machine Learning for Android Malware Detection: Mission Accomplished? A Comprehensive Review of Open Challenges and Future Perspectives. Computers & Security **138**, 103654 (2024) https://doi.org/10.1016/j.cose.2023.103654

[34] Sun, T., Daoudi, N., Pian, W., Kim, K., Allix, K., Bissyandé, T.F., Klein, J.: Temporal-incremental learning for Android

malware detection. ACM Transactions on Software Engineering and Methodology **34**(4), 101–110130 (2025) https://doi.org/10.1145/3702990

[35] Sabbah, A., Kharma, M.F., Jarrar, R., Zein, S., Mohaisen, D.: Adversarial vulnerability under temporal concept drift: A longitudinal study of Android malware detection. Expert Systems with Applications **332**, 133560 (2027) https://doi.org/10.1016/j.eswa.2026.133560

[36] Alzaylaee, M.K., Yerima, S.Y., Sezer, S.: DynaLog: An automated dynamic analysis framework for characterizing Android applications. In: 2016 International Conference on Cyber Security and Protection of Digital Services, pp. 1–8 (2016). https://doi.org/10.1109/CyberSecPODS.2016.7502337

[37] Cai, H., Meng, N., Ryder, B.G., Yao, D.: DroidCat: Effective Android malware detection and categorization via app-level profiling. IEEE Transactions on Information Forensics and Security **14**(6), 1455–1470 (2019) https://doi.org/10.1109/TIFS.2018.2879302

[38] Ficco, M.: Malware analysis by combining multiple detectors and observation windows. IEEE Transactions on Computers **71**(6), 1276–1290 (2022) https://doi.org/10.1109/TC.2021.3082002

[39] Onwuzurike, L., Mariconti, E., Andriotis, P., De Cristofaro, E., Ross, G., Stringhini, G.: MaMaDroid: Detecting Android malware by building Markov chains of behavioral models (Extended Version). ACM Transactions on Privacy and Security **22**(2), 14–134 (2019) https://doi.org/10.1145/3313391

[40] Karbab, E.B., Debbabi, M., Derhab, A., Mouheb, D.: MalDozer: Automatic framework for Android malware detection using deep learning. Digital Investigation **24**, 48–59 (2018) https://doi.org/10.1016/j.diin.2018.01.007

[41] Pektaş, A., Acarman, T.: Learning to detect Android malware via opcode sequences. Neurocomputing **396**, 599–608 (2020) https://doi.org/10.1016/j.neucom.2018.09.102

[42] Zhang, M., Duan, Y., Yin, H., Zhao, Z.: Semantics-aware Android malware classification using weighted contextual API dependency graphs. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1105–1116 (2014). https://doi.org/10.1145/2660267.2660359

[43] Yuan, Z., Lu, Y., Xue, Y.: DroidDetector: Android malware characterization and detection using deep learning. Tsinghua Science and Technology **21**(1), 114–123 (2016) https://doi.org/10.1109/TST.2016.7399288

[44] Yerima, S.Y., Sezer, S.: DroidFusion: A novel multilevel classifier fusion approach for Android malware detection. IEEE Transactions on Cybernetics **49**(2), 453–466 (2019) https://doi.org/10.1109/TCYB.2017.2777960

[45] Mahdavifar, S., Alhadidi, D., Ghorbani, A.A.: Effective and efficient hybrid Android malware classification using pseudo-label stacked auto-encoder. Journal of Network and Systems Management **30**(1), 22 (2022) https://doi.org/10.1007/s10922-021-09634-4

[46] Nasser, A.R., Hasan, A.M., Humaidi, A.J.: DL-AMDet: Deep learning-based malware detector for Android. Intelligent Systems with Applications **21**, 200318 (2024) https://doi.org/10.1016/j.iswa.2023.200318

[47] Zhao, W., Wu, J., Meng, Z.: AppPoet: Large language model based Android malware detection via multi-view prompt engineering. Expert Systems with Applications **262**, 125546 (2025) https://doi.org/10.1016/j.eswa.2024.125546

[48] Lantz, P., Desnos, A., Yang, K.: DroidBox: An Android application sandbox for dynamic analysis. Open-source project documentation. Accessed September 20,

2026 (2011). https://github.com/pjlantz/droidbox

[49] Alazab, M., Moonsamy, V., Batten, L., Lantz, P., Tian, R.: Analysis of malicious and benign Android applications. In: 32nd IEEE International Conference on Distributed Computing Systems Workshops, pp. 608–616 (2012). https://doi.org/10.1109/ICDCSW.2012.13

[50] Frida: Android. Frida documentation. Retrieved September 20, 2026 (n.d.). https://frida.re/docs/android/

[51] Soriano-Salvador, E., Guardiola-Múzquiz, G.: Detecting and bypassing Frida dynamic function call tracing: Exploitation and mitigation. Journal of Computer Virology and Hacking Techniques **19**(4), 503–513 (2023) https://doi.org/10.1007/s11416-022-00458-7

[52] Alecci, M., Ruiz Jiménez, P.J., Allix, K., Bissyandé, T.F., Klein, J.: AndroZoo: A retrospective with a glimpse into the future. In: Proceedings of the 21st International Conference on Mining Software Repositories, pp. 389–393 (2024). https://doi.org/10.1145/3643991.3644863

[53] Martín, A., Lara-Cabrera, R., Camacho, D.: Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset. Information Fusion **52**, 128–142 (2019) https://doi.org/10.1016/j.inffus.2018.12.006

[54] Mann, H.B., Whitney, D.R.: On a test of whether one of two random variables is stochastically larger than the other. The Annals of Mathematical Statistics **18**(1), 50–60 (1947) https://doi.org/10.1214/aoms/1177730491

[55] Benjamini, Y., Hochberg, Y.: Controlling the false discovery rate: A practical and powerful approach to multiple testing. Journal of the Royal Statistical Society: Series B **57**(1), 289–300 (1995) https://doi.org/10.1111/j.2517-6161.1995.tb02031.x

[56] Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning **20**, 273–297 (1995) https://doi.org/10.1007/BF00994018

[57] Breiman, L.: Random forests. Machine Learning **45**, 5–32 (2001) https://doi.org/10.1023/A:1010933404324

[58] Friedman, J.H.: Greedy function approximation: A gradient boosting machine. The Annals of Statistics **29**(5), 1189–1232 (2001) https://doi.org/10.1214/aos/1013203451

[59] Chen, T., Guestrin, C.: XGBoost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785–794 (2016). https://doi.org/10.1145/2939672.2939785

[60] Fawcett, T.: An introduction to ROC analysis. Pattern Recognition Letters **27**(8), 861–874 (2006) https://doi.org/10.1016/j.patrec.2005.10.010

[61] Saito, T., Rehmsmeier, M.: The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. PLOS ONE **10**(3), 0118432 (2015) https://doi.org/10.1371/journal.pone.0118432

[62] Efron, B.: Bootstrap methods: Another look at the jackknife. The Annals of Statistics **7**(1), 1–26 (1979) https://doi.org/10.1214/aos/1176344552

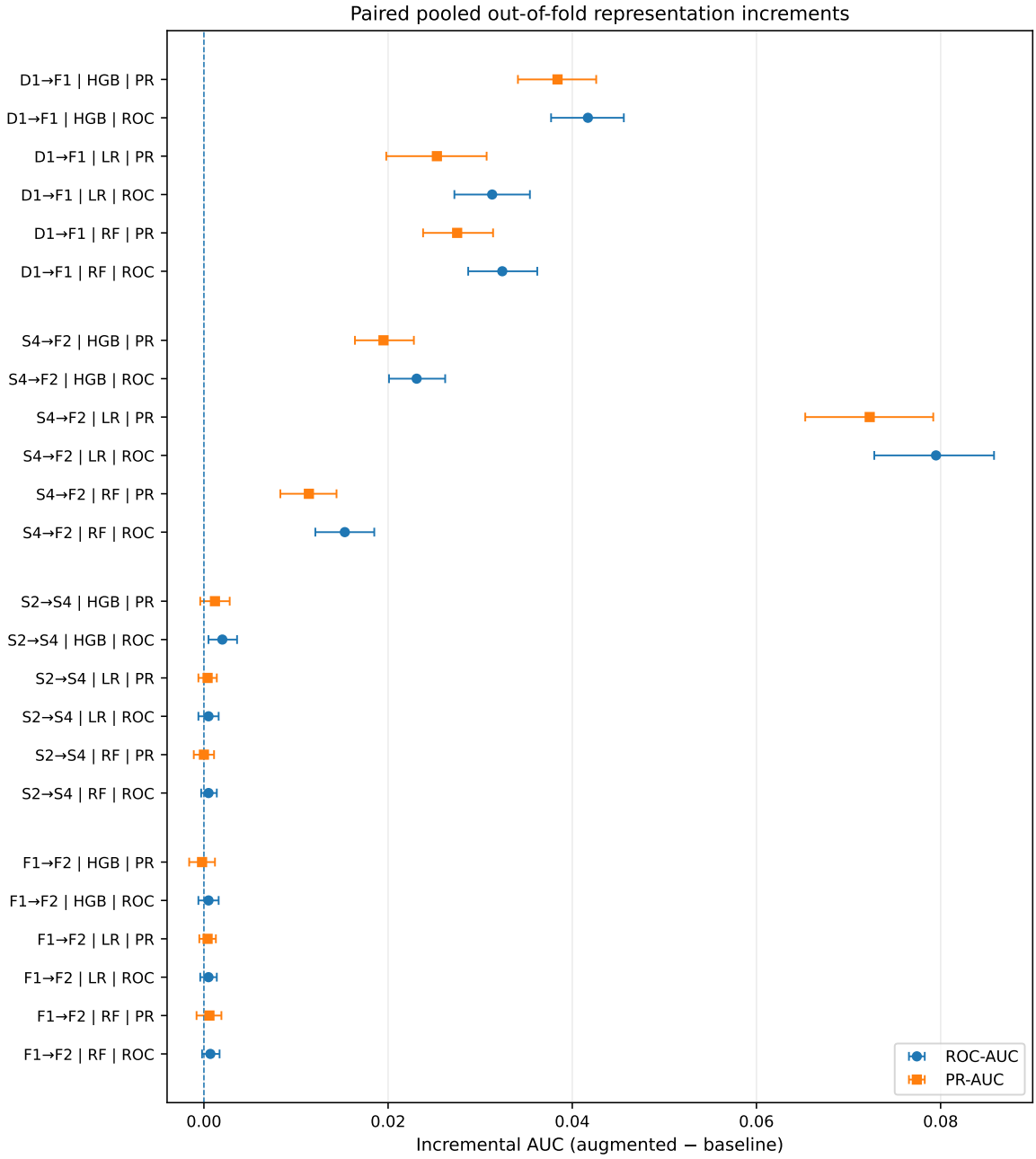


Fig. 5 Incremental value of additional dynamic representations under paired pooled out-of-fold bootstrap analysis. The largest and most consistent gains are D1→F1 and S4→F2; S2→S4 is small and F1→F2 is negligible. Points are observed pooled-OOF differences; intervals are prediction-level bootstrap intervals conditional on the fitted models.

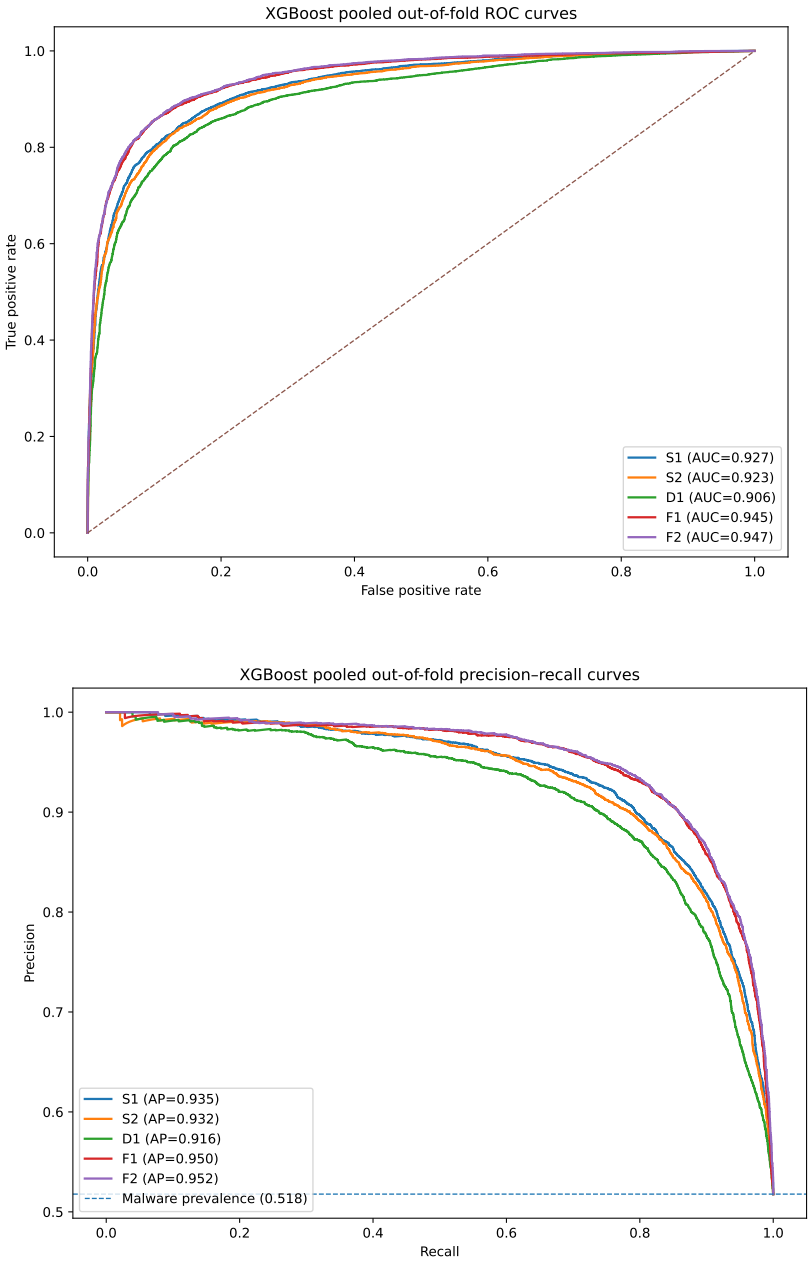


Fig. 6 Pooled out-of-fold XGBoost ROC and precision–recall curves for S1, S2, D1, F1, and F2. Curves are computed from held-out OOF predictions for all 11,457 APKs; the precision–recall panel includes the malware-prevalence baseline.

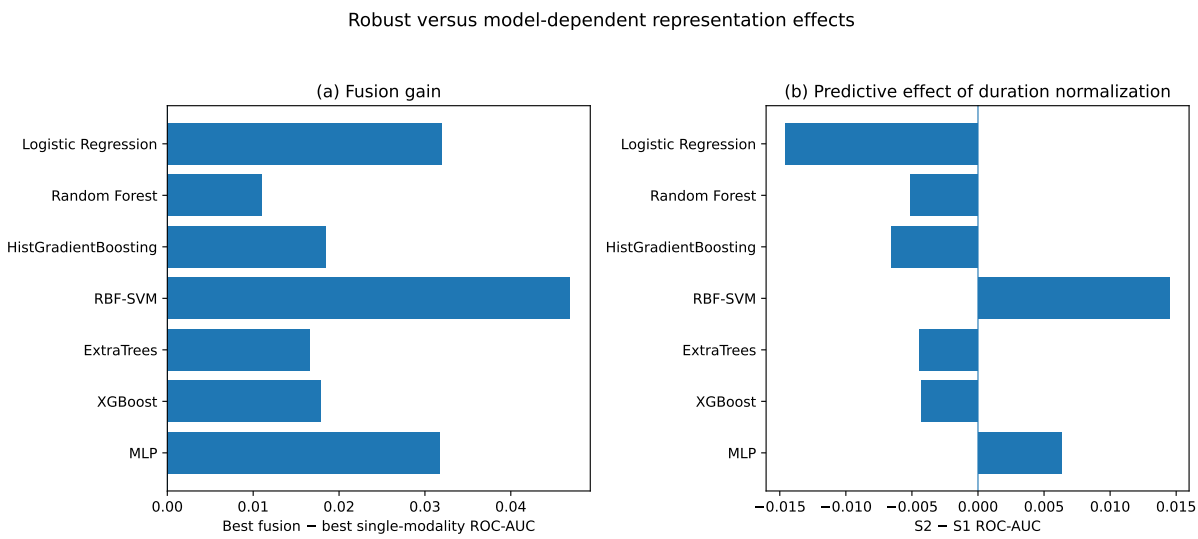


Fig. 7 Robust versus model-dependent representation effects. Fusion gains are positive across all seven model families, whereas the predictive effect of duration normalization changes sign across learners.