MCNDROID: A LONGITUDINAL MULTIMODAL BENCHMARK FOR ROBUST DRIFT DETECTION IN ANDROID MALWARE

Anonymous authors
Paper under double-blind review

ABSTRACT

Machine learning (ML) in real-world systems must contend with concept drift, adversarial actors, and a spectrum of potential features with varying costs and benefits. Malware naturally exhibits all of these complexities, but for the same reason, it is challenging to curate and organize data to study these factors. We present McNdroid, to our knowledge the largest longitudinal multimodal Android malware benchmark for malware detection and drift analysis. McNdroid spans 2013–2025, excluding 2015, and represents each application with three aligned modalities– static features from manifests and smali code, dynamic behavioral features from sandbox execution, and graph-based features from function-call graphs. Using temporally separated splits, we evaluate standard ML and deep-learning detectors across increasing train–test time gaps. Results show clear temporal degradation, while multimodal fusion outperforms the best single modality across long-term temporal gaps. Cross-modal agreement also declines over time, suggesting that drift affects both individual feature spaces and the consistency among modalities. We further analyze modality-specific drift, malware-family evolution, and temporal changes in model explanations. We publicly release McNdroid, benchmark splits, and code to support reproducible research on temporal generalization and robust multimodal learning in security-critical, non-stationary settings.

Data: `https://zenodo.org/records/19969833`
Code: `https://anonymous.4open.science/r/McNdroid-BDEC/`

1 INTRODUCTION

Malware detection is a large-scale and continuously evolving machine-learning (ML) problem Anderson & Roth (2018); Arp et al. (2014). Online malware-analysis services such as VirusTotal process approximately 1.8 million samples per day Virustotal. Traditional methods such as sandbox analysis, hand-crafted features, and signature matching, require substantial manual effort and often struggle with obfuscation Aonzo et al. (2020) and rapidly changing malware Nguyen et al. (2024); Rastogi et al. (2013). Although ML detectors can reduce reliance on manual rules, they face a different challenge. Both benign and malicious software evolve over time, changing the underlying data distribution and inducing concept drift, which can degrade models trained on historical data and lead to model aging Jordaney et al. (2017); Cai (2020); Pendlebury et al. (2019).

Evaluating this degradation requires benchmarks that capture more than one view of an application. Static features provide broad coverage but can be affected by obfuscation, repackaging, API substitution, and structural manipulation Zhang et al. (2020); He et al. (2023a); Zhang et al. (2025); Zhao et al. (2021); Haque et al. (2026). Dynamic traces capture runtime behavior Shijo & Salim (2015); Xu et al. (2024), while program graphs capture application structure and dependencies Freitas et al. (2021); Chen et al. (2025); Zhang et al. (2025). However, both are costly to extract at scale. *A useful benchmark should therefore provide aligned static, dynamic, and graph-based views over time, enabling evaluation of how each modality changes and fails under malware evolution.*

Existing Android malware benchmarks provide useful foundations, but they do not fully support multimodal analysis of malware evolution. DREBIN Arp et al. (2014) enabled large-scale Android malware detection, and TESSERACT Pendlebury et al. (2019) and APIGraph Zhang et al. (2020)

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

introduced temporal evaluation, but these benchmarks remain limited in coverage, representation, or long-term family-level analysis. LAMDA Haque et al. (2026), the closest related benchmark, provides 12 years of Android malware data but primarily uses static DREBIN-style features. Such static-only temporal benchmarks cannot reveal whether degradation stems from static artifacts, runtime behavior, code structure, or malware-family shifts. Hidden spatio-temporal biases can also lead to overly optimistic conclusions Chow et al. (2025). These gaps motivate a temporally structured multimodal benchmark with aligned static, dynamic, and graph-based views. Table 1 summarizes this gap by comparing existing APK benchmarks across characterization features, modality coverage, dataset scale, temporal span, artifact release, and dataset composition.

Existing multimodal benchmarks such as MULTIBENCH and MultiZoo support representation learning, fusion, robustness, and reproducible evaluation across several domains Liang et al. (2021; 2023). However, they are not designed for naturally occurring temporal shift, where modalities, labels, and cross-modal relationships evolve together. This limits their use for evaluating multimodal generalization under realistic deployment shift.

Android malware evolution offers a natural setting for this gap. Android applications can be represented through static, dynamic, and graph-based modalities, while distributions shift as ecosystems, malware families, APIs, and attacker strategies evolve Bavota et al. (2014); Yang et al. (2018); Fazzini et al. (2019). This makes malware evolution a useful testbed for studying multimodal models under realistic deployment shift, including fusion, modality robustness, graph learning, long-tailed labels, and continual adaptation over extended deployment periods.

We therefore introduce McNdroid, a large-scale benchmark for temporal multimodal evaluation in Android malware detection. McNdroid supports time-aware evaluation by training models on historical samples and testing them on future samples from shifted distributions Dragoi et al. (2022). Unlike single-modality benchmarks, it provides aligned static, dynamic, and graph-based views for each application, enabling analysis of how different modalities change and degrade over time. By preserving chronology, McNdroid supports realistic evaluation of model aging, adaptation, fusion, and generalization under deployment-like shift Chow et al. (2025). It also serves a broader testbed for multimodal learning under evolving and adversarial distributions.

In summary, our key contributions are:

- We introduce McNdroid, a large-scale benchmark for temporal multimodal Android malware detection, with 858,859 samples and 1,354 malware families.
- We provide three modalities–static, dynamic, and graph-based– for each application, capturing application artifacts, runtime behavior, and program structure.
- We provide a time-aware evaluation splits that train models on historical samples and test them on future samples, enabling realistic analysis of concept drift, modality-specific degradation, adaptation, and family-level evolution.
- We release pre-extracted feature representations and benchmark protocols to support reproducible research on multimodal fusion, graph learning, long-tailed classification, modality robustness, and generalization under non-stationary malware distributions.

Contributions in the Appendices. The appendices provide supporting analyses on dataset characterization, prior multimodal datasets, and the motivating case study (Appendices A–C); model descriptions and fusion strategies (Appendices D–E); extended temporal results, feature-space visualization, family stability, and feature importance (Appendices F–I); missing-modality sensitivity and prediction uncertainty under label drift (Appendices J–K); additional concept-drift adaptation, family-structure analysis, and continual learning (Appendices L–N); GNN baselines and computational resources (Appendices O–P); and security impact and responsible use of McNdroid (Appendix Q).

2 Related Work

Malware Datasets and Benchmarks. Large-scale Android malware datasets are essential for training and evaluating ML-based detectors. DREBIN Arp et al. (2014) remains one of the most widely used Android malware datasets based on static features. Other prior work has explored dynamic and graph-based representations, including MaMaDroid Mariconti et al. (2017), MalScan Wu et al. (2019), and MalNet Freitas et al. (2021). However, malware benchmarks can rapidly become

Table 1: Compact comparison of APK characterization features, modality, dataset scale, time coverage, artifact release, and dataset composition.

Approach	APK Characterization	Modality	Samples	Years	Artifact
Drebin Arp et al. (2014)	HC, AC, IN, PER, API, CS	S	129,013	3	✓
MamaDroid Mariconti et al. (2017)	API, PG	G	43,940	7	✓
HinDroid Hou et al. (2017)	API, PG	G	2,334	1	✗
Kim et al. Kim et al. (2018)	HC, AC, IN, PER, API, CS	S	41,260	–	✗
MalScan Wu et al. (2019)	API, PG	G	30,715	8	✓
SDAC Xu et al. (2020)	API, PG	G	70,142	6	✗
HomDroid Wu et al. (2021b)	API, PG	G	8,198	–	✗
Xmal Wu et al. (2021a)	PER, API	S	35,690	–	✓
RAMDA Li et al. (2021)	IN, PER, API	S	58,483	–	✓
MSDroid He et al. (2023b)	PER, API, CS, PG	G	81,790	6	✓
MalNet Freitas et al. (2022; 2021)	API, CS	S/I	1,262,024	–	✓
HiGraph Chen et al. (2025)	PER, API, CS, PG	G	595,211	11	✓
LAMDA Haque et al. (2026)	HC, API, CS	S	1,008,381	12	✓
MPDroid Zhang et al. (2025)	HC, API, CS	S/D	10,000	–	✗
McNdroid	**HC, AC, IN, PER, API, CS, PG, NET, FS, SYS, PROC, CMD, MOD, DB**	**S/D/G**	**858,859**	**12**	✓

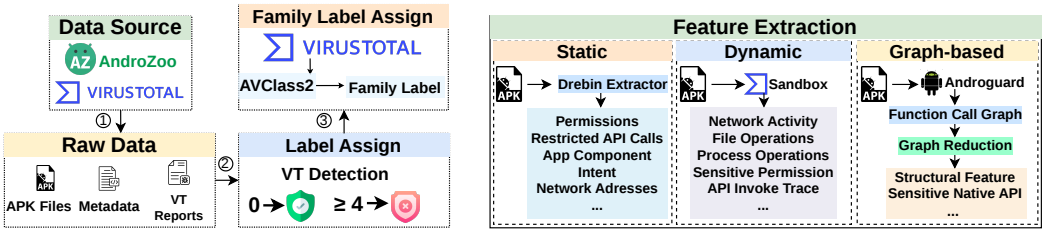
Legend: HC = Hardware / Resource Component; AC = Application Component; IN = Intent; PER = Permission; API = API Call; CS = Code String; PG = Program Graph Feature; NET = Network Artifact; FS = File-System Artifact; SYS = System Property; PROC = Process; CMD = Command Execution; MOD = Loaded Module; DB = Database Access. **Modality:** S = Static; D = Dynamic; G = Graph-based; I = Image-based.

outdated due to their evolving nature Pendlebury et al. (2019), and may suffer from sampling bias, data snooping, label noise, mixed-source inconsistencies, and unrealistic evaluation protocols Arp et al. (2022). Prior work such as TESSERACT Pendlebury et al. (2019) and Arp et al. Arp et al. (2022) provides methodological guidance for reliable temporal evaluation, while MPDroid Zhang et al. (2025) supports multimodal representation but *not longitudinal evaluation*. These limitations motivate benchmarks that jointly support both multimodal malware characterization and temporal evaluation, a gap summarized in Table 1.

Static and Graph-Based Analysis. Static analysis extracts APK features such as permissions, API calls, and services Arp et al. (2014). Graph-based approaches use function-call graphs (FCGs), control-flow graphs (CFGs), and related representations to capture application structure and behavior Wu et al. (2019); Freitas et al. (2021); Mariconti et al. (2017). API-sequence modeling further improves robustness to simple obfuscation. LAMDA Haque et al. (2026) is the closest longitudinal benchmark, but it primarily supports static analysis. As a result, it cannot evaluate how runtime behavior and program structure evolve alongside static features.

Dynamic and Behavioral Analysis. Dynamic analysis executes applications in controlled environments to observe runtime behavior, including system calls, API calls, dynamic loading, and network traffic Mahdavifar et al. (2020). TaintDroid Enck et al. (2014), DroidScope Yan & Yin (2012), CopperDroid Tam et al. (2015), and DVa Xu et al. (2024) show that runtime traces can reveal privacy leaks, high-level malicious behaviors, accessibility abuse, dynamic payloads, and persistence. VirusTotal reports also support static, dynamic, and behavioral studies VirusTotal (2025); Lashkari et al. (2018). However, dynamic analysis remains costly at scale and can suffer from low code coverage and evasion Kondracki et al. (2022). McNdroid provides pre-extracted dynamic features which allow temporal behavioral analysis without rerunning large-scale sandboxing.

Multimodal and Fusion Analysis. Prior works combine permissions, APIs, opcodes, strings, components, native code, and URLs, including approaches with feature-level explanations Zhu et al. (2019); Kim et al. (2018). MPDroid integrates static function-call and dynamic API-call graphs using multimodal pre-training, alignment, and fusion Zhang et al. (2025). These methods mainly focus on representation learning and detection performance and provide limited analysis of how different modalities evolve over time. McNdroid aligns static, dynamic, and graph-based representations



(a) Data collection strategy.

(b) Feature extraction pipeline.

Figure 1: Overview of the McNdroid dataset creation process.

across years and evaluates them under temporal splits to study per-modality degradation, fusion stability, and malware-family changes under long-term distribution shift.

3 McNdroid Creation

3.1 Dataset Collection Strategy

We construct McNdroid from AndroZoo Allix et al. (2016), which provides APK SHA-256 hashes, VirusTotal (VT) metadata, and submission timestamps. These metadata allow us to build a longitudinal dataset for studying Android malware evolution under temporal distribution shift. A schematic representation of our data collection strategy is depicted in Figure 1a.

Data Collection and Temporal Coverage. We collect APKs from 2013 to 2025. We exclude 2015 because AndroZoo contains too few valid entries for that year Allix et al. (2016). Removing this sparse year avoids distorted temporal comparisons and maintains a more consistent timeline.

Label Assignment and Noise Reduction. We assign labels using `vt_detection` count, following prior malware benchmarking practice Haque et al. (2026); Pendlebury et al. (2019). Applications with `vt_detection = 0` are labeled benign, while those with `vt_detection >= 4` are labeled malware Haque et al. (2026). Requiring at least four antivirus detections provides stronger cross-vendor agreement and helps reduce labeling noise. Samples with intermediate detection counts (1–3) are discarded as ambiguous. We release raw `vt_detection` values so researchers can vary thresholds and test sensitivity to alternative detection thresholds.

Deduplication and Integrity Checks. Following prior work Liu et al. (2026), we perform strict deduplication using SHA-256 hashes to eliminate redundant samples and prevent data leakage.

Cross-Modality Consistency Filtering. To support fair multimodal evaluation, we retain only APKs with valid static features, dynamic outputs, and function-call graphs. Samples are removed if any extraction pipeline fails, including APKTool errors, incomplete sandbox reports, or graph-construction failures. This intersection filtering reduces the dataset from approximately 1.5M APKs to 858,859 samples, trading coverage for complete and aligned multimodal views.

Family Label Allocation. To facilitate fine-grained analysis beyond binary classification, we assign malware family labels using AVClass2 Sebastián & Caballero (2020), a widely adopted tool for normalizing heterogeneous antivirus vendor outputs. AVClass2 aggregates noisy and often inconsistent VirusTotal VirusTotal (2025) labels into a unified and standardized family taxonomy. For each sample, we retrieve the corresponding VirusTotal report, and extract the inferred family label.

Evaluation Splits. We adopt an 80–20 train–test split with stratification to preserve the original malware-to-benign class distribution Haque et al. (2026). This ensures unbiased evaluation under realistic class imbalance without introducing distributional skew. Detailed dataset statistics and malware family distributions are provided in Appendix A.

3.2 Feature Extraction

Figure 1b summarizes the static, dynamic, and graph-based feature extraction pipeline from APKs.

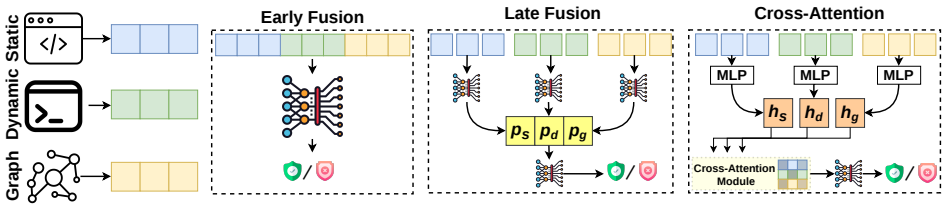


Figure 2: Overview of the multimodal fusion strategies evaluated in McNdroid.

Static Feature. Our first modality uses static features, which support efficient application profiling without execution. Following DREBIN Arp et al. (2014), we unpack each APK and parse `AndroidManifest.xml` and the disassembled `classes.dex` bytecode. We extract hardware components, requested and used permissions, app components, intents, suspicious API calls, and network addresses. These indicators are encoded as a high-dimensional binary bag-of-features vector, where each dimension records the presence or absence of a static attribute. To prevent temporal leakage, we construct the global vocabulary only from the 2013 training split. We then apply low-variance filtering with `VarianceThreshold` (0.001) to remove rare or uninformative attributes, reducing the static feature space from 501,525 to 2,390 features per application Rahman et al. (2025; 2022).

Dynamic Feature. Our second modality captures runtime behavior from structured VirusTotal sandbox reports VirusTotal (2025). The extracted signals cover network activity, file-system operations, process execution, inter-process communication, and API-call traces. We transform raw telemetry to a hybrid feature space–(i) explicit features for stable behavioral patterns, and (ii) hashed features for high-entropy or open-vocabulary artifacts. Explicit features include system-interaction counts, protocol usage statistics, telephony access, location access, reflection, privilege-escalation attempts, and persistence indicators. High-cardinality artifacts, such as domains, file paths, command strings, and textual fields, are mapped to fixed-dimensional vectors using feature hashing Weinberger et al. (2009); Anderson & Roth (2018); Joyce et al. (2025). This design preserves common behavioral signals while bounding the dimensionality of open-vocabulary artifacts. The resulting dynamic representation contains 17,483 features per application. We further apply `VarianceThreshold` (0.001) to the dynamic feature space, reducing its dimensionality from 17,483 to 2,293 features.

Function Call Graph (FCG). Our third modality captures program structure through function-call graphs (FCGs). Following prior work Mariconti et al. (2017); Xu et al. (2020); Chen et al. (2025), we extract FCGs with Androguard Desnos (2011) and model each application as a directed graph of function calls. Because raw FCGs are large, noisy, and API-heavy Shokouhinejad et al. (2025); Zhang et al. (2025), we reduce complexity in two steps: retaining local developer-defined functions, where malicious logic is often embedded Chen et al. (2025), and preserving edges incident to security-critical APIs identified by MalScan Wu et al. (2019). From the reduced graph, we extract structural statistics, sensitive-interaction ratios, and binary sensitive-API indicators. The sensitive-API vocabulary is constructed solely from training data, yielding a fixed-dimensional FCG representation of 2,793 features per application. We also release caller–callee relations for custom graph construction.

4 Benchmark Design

McNdroid provides a large-scale multimodal dataset and reproducible benchmark for Android malware detection under temporal distribution shift. McNdroid standardizes data curation, label assignment, feature extraction, and evaluation splits, enabling fair comparison across methods Arp et al. (2014); Haque et al. (2026); Chen et al. (2025); Mariconti et al. (2017); Zhang et al. (2020), modality-specific analysis, and robustness evaluation under realistic temporal shift.

Malware Detection under Distribution Shift. We formulate Android malware detection as supervised binary classification. Each application i is represented by aligned static, dynamic, and graph-based modalities, $x_i = (x_i^s, x_i^d, x_i^g)$, and a detector learns $f(x_i) \rightarrow y_i$, where $y_i \in 0, 1$ denotes benign or malicious. McNdroid assesses temporal generalization by training on historical samples and testing on future applications Haque et al. (2026); Dragoi et al. (2022), capturing distribution shifts arising from changes in application characteristics, malware families, APIs, and attacker behavior.

Table 2: Comparison of unimodal and best multimodal performances across three temporal splits. † Multimodal denotes the best fusion configuration per model–split pair.

Split	Model	Static		Dynamic		Graph		Multimodal†	
		F1	ROC-AUC	F1	ROC-AUC	F1	ROC-AUC	F1	ROC-AUC
IID	MLP	90.6±6.5	97.4±2.0	82.9±7.8	92.0±5.0	82.1±5.3	93.3±2.8	90.7±5.5	98.3±1.0
	LightGBM	91.6±5.5	98.4±1.2	85.2±7.7	94.0±4.2	91.5±5.1	98.1±1.3	92.6±4.5	98.7±0.9
	XGBoost	91.5±5.8	98.2±1.4	85.2±8.0	93.6±4.6	92.1±5.1	98.8±0.8	92.1±5.0	98.8±0.8
	SVM	90.0±5.3	97.1±2.0	76.9±6.0	89.3±3.9	86.2±5.5	95.0±2.7	90.5±4.9	98.1±0.9
	DetectBERT	89.6±5.6	97.9±1.2	81.3±7.7	90.4±5.6	0.0±0.0	50.0±0.0	90.7±5.8	98.1±1.3
	Transformer	88.5±6.5	97.7±1.4	80.4±7.6	91.1±4.9	41.8±10.3	66.9±4.0	91.0±4.1	97.9±1.0
NEAR	MLP	43.1±30.2	77.3±13.4	38.7±24.5	69.9±10.9	36.9±24.7	65.2±8.3	42.7±26.5	77.9±13.4
	LightGBM	40.3±31.8	82.3±11.0	34.7±28.3	74.5±12.4	32.6±28.6	81.5±8.2	39.7±28.6	78.9±14.6
	XGBoost	39.4±32.1	84.6±7.6	35.5±26.7	73.3±12.7	34.6±28.2	84.4±6.2	39.6±29.3	81.7±12.3
	SVM	43.9±30.4	73.9±16.5	40.2±19.9	71.2±9.4	31.1±26.6	77.4±6.3	46.2±26.6	78.0±12.1
	DetectBERT	42.1±30.9	70.5±22.4	39.7±23.4	67.0±13.9	0.0±0.0	49.9±0.1	43.9±24.4	78.2±11.4
	Transformer	39.5±30.2	68.4±24.3	39.2±25.1	68.4±14.8	29.4±17.5	67.5±9.7	44.8±27.2	80.8±9.91
FAR	MLP	28.2±16.9	81.2±12.4	32.2±22.9	63.7±6.3	15.4±13.3	53.3±13.4	29.0±21.0	79.4±12.3
	LightGBM	35.7±24.3	84.0±12.8	25.3±20.6	65.1±16.6	29.8±18.1	76.4±11.8	47.9±27.1	79.1±17.9
	XGBoost	31.5±23.4	85.3±12.2	27.7±22.2	65.6±14.0	27.1±15.0	77.1±11.9	42.7±25.7	80.3±16.3
	SVM	33.3±18.5	79.8±11.3	29.1±20.0	56.0±8.7	13.7±11.8	83.0±9.1	41.6±21.9	79.8±10.3
	DetectBERT	34.8±21.0	80.9±16.9	32.5±23.7	62.3±7.2	10.6±15.0	54.5±7.0	38.1±25.4	84.0±11.9
	Transformer	28.9±21.3	82.4±17.2	29.4±20.9	60.4±7.0	18.3±13.8	35.8±17.2	43.4±26.1	69.2±16.3

Temporal Generalization. To simulate deployment, McNdroid uses chronologically ordered splits following prior temporal malware evaluation protocols Haque et al. (2026); Yang et al. (2021). Applications are partitioned by year, and models are trained only on earlier years and evaluated on later years. Increasing the train–test gap creates stronger temporal shift, allowing us to measure performance degradation over time. This design prevents temporal leakage; unless otherwise stated, temporal experiments preserve chronological ordering and avoid random mixing across years.

Modality-Controlled Evaluation. Each application has aligned static, dynamic, and graph-based views. This alignment allows evaluation of (i) single-modality, (ii) pairwise-modality, and (iii) multimodal settings under identical sample sets and temporal splits. As a result, performance differences can be attributed to modality choice or fusion strategy, while the data remain unchanged.

Multimodal Fusion Protocols. McNdroid supports evaluation of three multimodal fusion strategies, summarized in Figure 2: (i) *early fusion*, where features from modalities are concatenated before classification to learn feature interactions Pérez-Rúa et al. (2019); Boulahia et al. (2021); (ii) *late fusion*, where modality-specific models are trained independently and their outputs are combined at the decision level Franco et al. (2020); and (iii) *cross-attention fusion*, where modality-specific embeddings are integrated through learnable attention mechanisms to capture cross-modal interactions Zhu et al. (2019). In early fusion, static, dynamic, and graph-based feature matrices are horizontally concatenated. We also evaluate pairwise variants—static–dynamic, static–graph-based, and dynamic–graph-based—to identify complementary modality pairs. In late fusion, modality-specific learners produce probability estimates that are used as meta-features for a second-stage learner Franco et al. (2020). In cross-attention fusion, each modality is projected into a shared embedding space using an MLP encoder Kim et al. (2018). Modality tokens then attend to one another through multi-head attention Zhu et al. (2019), and the context-aware representations are normalized, concatenated, and used for prediction. Before training and evaluation, we verify hash and label consistency across modalities so that all fusion strategies use matched samples.

5 Benchmark Validation under Temporal Distribution Shift

Experimental Setup. Following prior temporal evaluation studies Dragoi et al. (2022); Haque et al. (2026), we train models on 2013 samples and evaluate them on progressively later years to measure robustness under temporal distribution shift. We define three evaluation regimes: IID, using temporally adjacent 2014 samples as a near-distribution baseline; NEAR, using samples from 2016–2017; and FAR, using samples from 2018–2025. This protocol fixes the training distribution and increases the temporal distance of the test distribution, allowing us to measure how detector performance changes as software ecosystems, malware families, and behaviors evolve.

We first evaluate each modality independently to establish unimodal baselines. For static, dynamic, and graph-based representations, we train widely used malware detection models, including Linear

SVM, LightGBM, MLP, XGBoost, DetectBERT, and Transformer-based architectures Xiong et al. (2021); Anderson & Roth (2018); Haque et al. (2026). We then apply the same temporal protocol to multimodal models that fuse static, dynamic, and graph-based representations. This ensures that unimodal and multimodal methods are compared under identical samples, labels, and temporal splits. All experiments are repeated over five random seeds, and results are reported as *mean*±*std*, averaged over runs and years within each split. Table 2 compares unimodal baselines with multimodal variants.

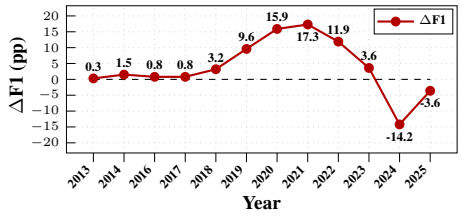


Figure 3: Year-wise multimodal F1-score gain over the strongest unimodal approach.

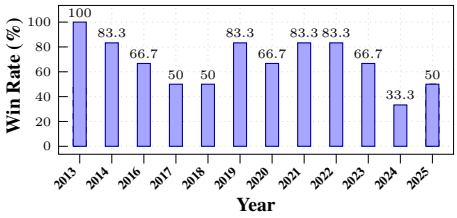


Figure 4: Year-wise multimodal win rate across six models.

Results. Table 2 shows substantial temporal degradation: all models perform best on IID, while F1-scores decline sharply on NEAR and FAR, indicating limited generalization to later malware distributions. Among unimodal representations, static features are the most stable, dynamic features provide moderate robustness, and graph-based features are more variable, with severe failures for some model–modality pairs. FAR performance is not uniformly worse than NEAR, showing that drift is not strictly monotonic with calendar distance; however, both temporal regimes remain well below IID. Multimodal fusion provides clearer gains under long-term drift: FAR LightGBM improves from $35.7_{\pm 24.3}$ with the best unimodal static representation to $47.9_{\pm 27.1}$ with fusion, while FAR Transformer improves from $29.4_{\pm 20.9}$ with the best unimodal dynamic representation to $43.4_{\pm 26.1}$.

Figure 3 reports year-wise F1 gains of the best multimodal configuration over the strongest unimodal input for the same architecture, with positive gains in 10 of 12 years and the largest improvements during 2019–2022 (+9.6 to +17.3 percentage points). However, gains become negative in 2024 and 2025 (-14.2 and -3.6), showing that fusion is not uniformly beneficial under severe temporal shift. Figure 4 complements this result by measuring how consistently multimodal gains occur across models, defined as the fraction of architectures whose best multimodal configuration outperforms the strongest unimodal configuration in the same test year. Multimodal fusion wins for all six models in 2013 and five of six in 2014, 2019, 2021, and 2022, but falls to 33.3% in 2024 and 50.0% in 2025. This decline aligns with Section 6.3, where cross-modal agreement decreases over time and becomes negative in 2024 and 2025, indicating that although the modalities provide complementary signals, their benefit weakens under severe temporal shift.

Interpreting temporal degradation. We observe severe degradation in 2017, 2018, 2024, and 2025. In 2017–2018, feature-space analysis (Appendix G) shows substantial malware–benign overlap, consistent with reduced class separability. In 2024–2025, degradation is associated with fewer malware samples (Appendix A) and declining cross-modal agreement, which becomes negative in 2024 and 2025 (Figure 7), causing fusion to combine conflicting evidence. Beyond these factors, Android evolution modifies APIs, permissions, policies, libraries, and framework usage Bavota et al. (2014); Yang et al. (2018); Fazzini et al. (2019). Malware also evolves in implementation strategies, API usage, obfuscation, and runtime behavior Arp et al. (2014); Xu et al. (2024); Haque et al. (2026). These shifts affect modalities differently: static features remain comparatively stable, graph representations exhibit the strongest structural drift, and dynamic features track changing runtime behavior and execution constraints. Class and family level analyses (Section 6.2 and Appendix H) further show that these changes vary across benign applications and malware families.

6 Comprehensive Multimodal Drift Analysis

6.1 Distributional Shift Across Time

Experimental Setting. We quantify temporal distribution shift by comparing each test year with the 2013 training reference using feature-wise two-sample Kolmogorov–Smirnov (KS) statistics Gao et al.

(2024); Kulinski et al. (2020); Berger & Zhou (2014). KS measures the maximum difference between two empirical distributions, making it suitable for measuring covariate shift across modalities.

Analysis. Figure 5 shows that modalities diverge from the 2013 distribution over time, indicating temporal covariate shift. The graph-based modality exhibits the largest drift, increasing from 0.001 in the 2013 comparison to 0.141 in 2025. This is consistent with the evolving Android ecosystem such as operating-system updates and API changes can deprecate, modify, or introduce APIs over time, which may alter function-call structure and graph-derived features. The high KS divergence for graph features aligns with our results where graph-based models show greater performance variability under temporal shift.

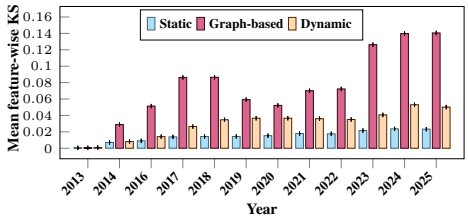


Figure 5: Distribution shift across modalities measured by the mean feature-wise KS statistic.

6.2 Feature Stability Across Modalities and Class Distributions

Setting. We further examine class-specific feature stability. For each modality and class, we estimate yearly feature distributions using normalized histograms and compare them with the 2013 reference using Jeffreys divergence Jeffreys (1946). This symmetric, non-negative measure captures discrepancies in both directions between distributions. We compute divergence per feature and average across features to obtain one value for each modality, class, and year Haque et al. (2026).

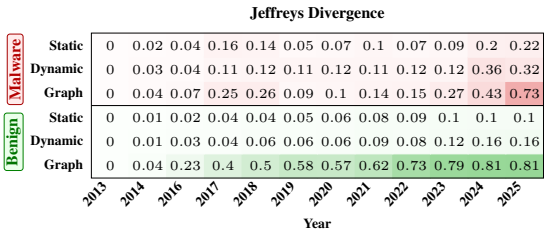


Figure 6: Temporal Jeffreys divergence across modalities for malware and benign samples.

Analysis. Figure 6 shows temporal Jeffreys divergence for malware and benign samples across the modalities. Malware shows low initial divergence from the 2013 baseline. Static and dynamic features remain relatively stable, while graph-based divergence increases in later years. Benign samples exhibit stronger drift, with gradual increases in static and dynamic divergence and a sharp rise in graph-based divergence, likely reflecting changes in APIs, frameworks, and development practices. Overall, static features are most stable, dynamic features show moderate drift, and graph-based representations exhibit the strongest temporal evolution, particularly for benign applications.

6.3 Cross-Modal Disagreement Analysis

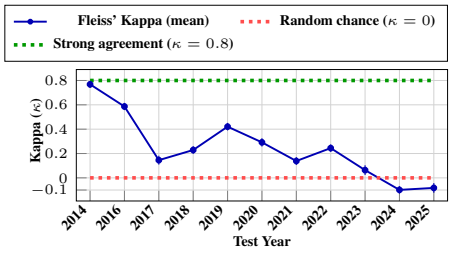


Figure 7: Year-wise Fleiss' Kappa with standard-deviation error bars.

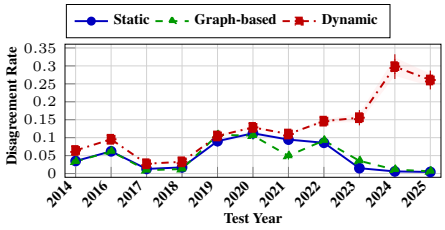


Figure 8: Year-wise disagreement across static, dynamic, and graph-based features.

Setting. To assess whether modalities make consistent predictions over time, we train one XGBoost classifier per modality on 2013 samples and evaluate them on later test years. We quantify inter-modality agreement using Fleiss' Kappa Fleiss (1971), treating each classifier as an independent rater. For each sample, we identify the modality whose prediction differs from the other two and aggregate these disagreement rates by modality and year.

Analysis. Figure 7 shows that cross-modal agreement is highest in 2014 ($\kappa = 0.768$) and declines over time, becoming negative in 2024 which indicates increasing disagreement among modality-

specific classifiers. Figure 8 shows that dynamic features are the primary source of this disagreement. The dynamic classifier differs from the other two modalities for approximately 30% of samples in 2024–2025. In comparison, the static and graph-based classifiers remain more consistent with each other. Overall, temporal drift affects both individual feature spaces and cross-modal agreement.

7 Concept Drift Adaptation Across Modalities

Setting. We evaluate concept drift adaptation (CDA) on McNdroid using three established approaches: HCC Chen et al. (2023), CADE Yang et al. (2021), and DAN–MMD Long et al. (2015); Li et al. (2025). For HCC and CADE, we follow the active learning framework of Chen et al. Chen et al. (2023), where target samples are periodically selected for labeling and used to update the detector under fixed labeling budgets. DAN–MMD represents domain adaptation, using the labeled target budget to align source and target representations by minimizing the maximum mean discrepancy between their latent feature distributions. We evaluate all methods across static, dynamic, graph-based, and fused representations under the same temporal protocol, using labeling budgets of 50, 100, 200, and 400 samples. We report the 400-sample setting in the main text and provide results for the remaining budgets in Appendix L.

Table 3: Performance of CDA methods across IID, NEAR, and FAR settings (budget = 400).

Method	Setting	Static	Dynamic	Graph-based	Feature Fusion
HCC	IID	95.9±0.3	73.9±3.7	90.3±1.6	93.9±1.3
	NEAR	87.0±1.2	70.7±1.1	73.0±1.7	82.9±0.7
	FAR	70.5±2.3	46.1±2.2	62.5±1.5	70.0±0.5
CADE	IID	92.4±0.3	76.9±0.9	86.1±1.1	82.0±4.0
	NEAR	52.2±7.9	36.2±4.9	29.4±4.7	41.6±3.9
	FAR	20.6±0.8	14.4±1.1	6.3±1.4	14.3±4.2
DAN–MMD	IID	83.9±1.8	67.4±1.7	79.8±2.2	80.2±1.5
	NEAR	50.7±11.2	46.9±1.5	44.1±11.1	39.0±7.5
	FAR	18.5±1.3	14.8±0.3	16.5±2.3	15.3±2.6

Analysis. Table 3 shows that HCC achieves the strongest overall performance across modalities and temporal regimes. Although all methods degrade from IID to NEAR and FAR, HCC remains substantially more robust, retaining FAR F1 scores of $70.5_{\pm 2.3}$ with static features and $70.0_{\pm 0.5}$ with feature fusion. In contrast, CADE and DAN–MMD deteriorate sharply under long-term drift, with FAR performance falling below 21% across all modalities. The largest degradation occurs for CADE on graph-based features, which drops to $6.3_{\pm 1.4}$. These results indicate that current CDA methods remain highly sensitive to modality-specific and long-term distribution shift.

8 Discussion and Limitation

McNdroid align static, dynamic, and graph-based features which allow future researchers to analyze their individual and joint evolution under shift in temporal distribution. Results show that degradation across all modalities, while multimodal fusion provides complementary signals to improve robustness. Graph-based features show the strongest temporal divergence, feature importance shifts over time, and cross-modal agreement declines in later years. These findings establish McNdroid as a diagnostic benchmark for studying modality-specific drift, cross-modal inconsistency, malware-family evolution, explanation drift, and temporal adaptation.

McNdroid has several limitations. VirusTotal labels and AVClass2 family assignments may contain noise despite conservative filtering. In addition, our sampling strategy is more balanced than typical deployment distributions to preserve malware-family diversity and sufficient malicious samples across years. While this may not fully reflect benign-heavy settings, it allows controlled evaluation of multimodal drift, family evolution, and robustness under temporal shift.

9 Conclusion

We present McNdroid, a 12-year longitudinal multimodal Android malware benchmark for analyzing changes in detection performance, malware behavior, benign distributions, and family composition. Our evaluations show that long-term evolution degrades robustness, shifts feature distributions, reduces cross-modal agreement, alters explanations, and challenges drift adaptation methods. With broad temporal coverage, diverse families, and aligned multimodal features, McNdroid supports reproducible evaluation of robust, interpretable, and adaptive malware detection under temporal shift.

10 Reproducibility Statement

We provide detailed documentation of the dataset construction, preprocessing pipeline, labeling procedure, feature extraction, data partitioning, and evaluation protocol in the main paper and supplementary material. To further support reproducibility, the dataset is publicly available on Zenodo at `https://zenodo.org/records/19969833`, and an anonymous project repository is available at `https://anonymous.4open.science/r/McNdroid-BDEC`. The repository includes the implementation, configuration files, and instructions required to reproduce the experiments reported in this paper. Additional implementation details and experimental settings are provided in the appendix and accompanying repository.

11 Ethics Statement

This work is intended to support defensive cybersecurity research on malware detection and related machine-learning tasks. The dataset does not distribute executable malware samples or other directly executable malicious artifacts. We release only the data representations and associated information necessary to support the research objectives of this work.

12 AI Use Statement

Generative AI tools were used only to improve the clarity, grammar, and presentation of the manuscript, and to assist with resolving LaTeX formatting issues. They were not used to create or label the dataset, design the experimental setup, conduct experiments, or interpret the results. All AI-assisted content was reviewed by the authors, who take full responsibility for the final manuscript.

References

Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *International Conference on Mining Software Repositories (MSR)*, 2016.

Hyrum S Anderson and Phil Roth. EMBER: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

Simone Aonzo, Gabriel Claudiu Georgiu, Luca Verderame, and Alessio Merlo. Obfuscapk: An open-source black-box obfuscation tool for android apps. *(SoftwareX)*, 2020.

John Arevalo, Thamar Solorio, Manuel Montes-y Gómez, and Fabio A González. Gated multimodal units for information fusion. *arXiv preprint arXiv:1702.01992*, 2017.

Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *USENIX Security Symposium*, 2022.

Lucas Baier, Tim Schlör, Jakob Schöffer, and Niklas Kühl. Detecting concept drift with neural network model uncertainty. *arXiv preprint arXiv:2107.01873*, 2021.

Gabriele Bavota, Mario Linares-Vasquez, Carlos Eduardo Bernal-Cardenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. The impact of api change-and fault-proneness on the user ratings of android apps. *IEEE Transactions on Software Engineering (TSE)*, 2014.

Vance W Berger and YanYan Zhou. Kolmogorov–smirnov test: Overview. *Wiley StatsRef: Statistics Reference Online*, 2014.

Said Yacine Boulahia, Abdenour Amamra, Mohamed Ridha Madi, and Said Daikh. Early, intermediate and late fusion strategies for robust deep learning-based multimodal action recognition. *Machine Vision and Applications*, 2021.

Haipeng Cai. Assessing and improving malware detection sustainability through app evolution studies. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2020.

Ricardo J. G. B. Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PA-KDD)*, 2013.

Santiago Castro, Devamanyu Hazarika, Verónica Pérez-Rosas, Roger Zimmermann, Rada Mihalcea, and Soujanya Poria. Towards multimodal sarcasm detection (an obviously perfect paper). In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

Han Chen, Hanchen Wang, Hongmei Chen, Ying Zhang, Lu Qin, and Wenjie Zhang. Higraph: A large-scale hierarchical graph dataset for malware analysis. *arXiv preprint arXiv:2509.02113*, 2025.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2016.

Yizheng Chen, Zhoujie Ding, and David Wagner. Continuous learning for android malware detection. In *USENIX Security Symposium*, 2023.

Theo Chow, Mario D'Onghia, Lorenz Linhardt, Zeliang Kan, Daniel Arp, Lorenzo Cavallaro, and Fabio Pierazzi. Breaking out from the tesseract: Reassessing ml-based malware detection under spatio-temporal drift. *arXiv preprint arXiv:2506.23814*, 2025.

Anthony Desnos. Androguard: Reverse engineering, malware and goodware analysis of android applications. `https://github.com/androguard/androguard`, 2011.

Marius Dragoi, Elena Burceanu, Emanuela Haller, Andrei Manolache, and Florin Brad. Anoshift: A distribution shift benchmark for unsupervised anomaly detection. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 2014.

Mattia Fazzini, Qi Xin, and Alessandro Orso. Automated api-usage update for android apps. In *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis (SIGSOFT)*, 2019.

Tiantian Feng, Digbalay Bose, Tuo Zhang, Rajat Hebbar, Anil Ramakrishna, Rahul Gupta, Mi Zhang, Salman Avestimehr, and Shrikanth Narayanan. Fedmultimodal: A benchmark for multimodal federated learning. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2023.

Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 1971.

Annalisa Franco, Antonio Magnani, and Dario Maio. A multimodal approach for human activity recognition based on skeleton and rgb data. *Pattern Recognition Letters*, 2020.

Scott Freitas, Yuxiao Dong, Joshua Neil, and Duen Horng Chau. A large-scale database for graph representation learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Scott Freitas, Rahul Duggal, and Duen Horng Chau. Malnet: A large-scale image database of malicious software. In *ACM International Conference on Information & Knowledge Management (CIKM)*, 2022.

Cuiying Gao, Gaozhun Huang, Heng Li, Bang Wu, Yueming Wu, and Wei Yuan. A comprehensive study of learning-based android malware detectors under challenging environments. In *IEEE/ACM International Conference on Software Engineering (ICSE)*, 2024.

Md Ahsanul Haque, Ismail Hossain, Md Mahmuduzzaman Kamol, Md Jahangir Alam, Suresh Kumar Amalapuram, Sajedul Talukder, and Mohammad Saidur Rahman. LAMDA: A longitudinal android malware benchmark for concept drift analysis. In *International Conference on Learning Representations (ICLR)*, 2026.

Md Kamrul Hasan, Wasifur Rahman, AmirAli Bagher Zadeh, Jianyuan Zhong, Md Iftekhar Tanveer, Louis-Philippe Morency, and Mohammed Ehsan Hoque. Ur-funny: A multimodal language dataset for understanding humor. In *Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

Ping He, Yifan Xia, Xuhong Zhang, and Shouling Ji. Efficient query-based attack against ml-based android malware detection under zero knowledge setting. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023a.

Yiling He, Yiping Liu, Lei Wu, Ziqi Yang, Kui Ren, and Zhan Qin. Msdroid: Identifying malicious snippets for android malware detection. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2023b.

Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

Harold Jeffreys. An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 1946.

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific Data*, 2016.

Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *USENIX Security Symposium*, 2017.

Robert J Joyce, Gideon Miller, Phil Roth, Richard Zak, Elliott Zaresky-Williams, Hyrum Anderson, Edward Raff, and James Holt. EMBER2024–A Benchmark Dataset for Holistic Evaluation of Malware Classifiers. *arXiv preprint arXiv:2506.05074*, 2025.

Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

TaeGuen Kim, BooJoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im. A multimodal deep learning method for android malware detection using various features. *IEEE Transactions on Information Forensics and Security (TIFS)*, 2018.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)*, 2017.

Brian Kondracki, Babak Amin Azad, Najmeh Miramirkhani, and Nick Nikiforakis. The droid is in the details: Environment-aware evasion of android sandboxes. In *Network and Distributed System Security Symposium (NDSS)*, 2022.

Sean Kulinski, Saurabh Bagchi, and David I Inouye. Feature shift detection: Localizing which features have shifted via conditional distribution tests. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Ravie Lakshmanan. Darkgate malware exploits samba file shares in short-lived campaign. *The Hacker News*, 2024. URL https://thehackernews.com/2024/07/darkgate-malware-exploits-samba-file.html.

Arash Habibi Lashkari, Andi Fitriah A Kadir, Laya Taheri, and Ali A Ghorbani. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *2018 International Carnahan Conference on Security Technology (ICCST)*, 2018.

Michelle A Lee, Brent Yi, Roberto Martín-Martín, Silvio Savarese, and Jeannette Bohg. Multimodal sensor fusion with differentiable filters. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020a.

Michelle A Lee, Yuke Zhu, Peter Zachares, Matthew Tan, Krishnan Srinivasan, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Learning multimodal representations for contact-rich tasks. *IEEE Transactions on Robotics (T-RO)*, 2020b.

Luis A Leiva, Asutosh Hota, and Antti Oulasvirta. Enrico: A dataset for topic modeling of mobile ui designs. In *International Conference on Human-Computer Interaction with Mobile Devices and Services*, 2020.

Adrian Shuai Li, Arun Iyengar, Ashish Kundu, and Elisa Bertino. Revisiting concept drift in windows malware detection: Adaptation to real drifted malware with minimal samples. In *Network and Distributed System Security Symposium (NDSS)*, 2025.

Heng Li, Shiyao Zhou, Wei Yuan, Xiapu Luo, Cuiying Gao, and Shuiyan Chen. Robust android malware detection against adversarial example attacks. In *The Web Conference (WWW)*, 2021.

Paul Pu Liang, Yiwei Lyu, Xiang Fan, Zetian Wu, Yun Cheng, Jason Wu, Leslie Chen, Peter Wu, Michelle A Lee, Yuke Zhu, et al. Multibench: Multiscale benchmarks for multimodal representation learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Paul Pu Liang, Yiwei Lyu, Xiang Fan, Arav Agarwal, Yun Cheng, Louis-Philippe Morency, and Ruslan Salakhutdinov. Multizoo and multibench: A standardized toolkit for multimodal deep learning. *Journal of Machine Learning Research*, 2023.

Jiahao Liu, Jun Zeng, Fabio Pierazzi, Ziqi Yang, Lorenzo Cavallaro, and Zhenkai Liang. Unraveling the key of machine learning-based android malware detection. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2026.

Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *Proceedings of the 32nd International Conference on Machine Learning*, Proceedings of Machine Learning Research. PMLR, 2015.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Samaneh Mahdavifar, Andi Fitriah Abdul Kadir, Rasool Fatemi, Dima Alhadidi, and Ali A Ghorbani. Dynamic android malware category classification using semi-supervised deep learning. In *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, 2020.

Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gianluca Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models. In *Network and Distributed System Security Symposium (NDSS)*, 2017.

Microsoft Defender Security Research Team. How artificial intelligence stopped an emotet outbreak. Microsoft Security Blog, 2018. URL https://www.microsoft.com/en-us/security/blog/2018/02/14/how-artificial-intelligence-stopped-an-emotet-outbreak/. Accessed: 2026-09-19.

Dung Thuy Nguyen, Ngoc N Tran, Taylor T Johnson, and Kevin Leach. Pbp: Post-training backdoor purification for malware classifiers. *arXiv preprint arXiv:2412.03441*, 2024.

Fernando Ortega and Vishnu Pratapagiri. Your mobile app, their playground: The dark side of virtualization, 2025. URL https://zimperium.com/blog/your-mobile-app-their-playground-the-dark-side-of-the-virtualization.

Jimin Park, AHyun Ji, Minji Park, Mohammad Saidur Rahman, and Se Eun Oh. MalCL: Leveraging gan-based generative replay to combat catastrophic forgetting in malware classification. In *AAAI Conference on Artificial Intelligence*, 2025.

Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *USENIX Security Symposium*, 2019.

Juan-Manuel Pérez-Rúa, Valentin Vielzeuf, Stéphane Pateux, Moez Baccouche, and Frédéric Jurie. Mfas: Multimodal fusion architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Mohammad Saidur Rahman, Scott E. Coull, and Matthew Wright. On the Limitations of Continual Learning for Malware Classification. In *Conference on Lifelong Learning Agents (CoLLAs)*, 2022.

Mohammad Saidur Rahman, Scott Coull, Qi Yu, and Matthew Wright. MADAR: Efficient continual learning for malware analysis with distribution-aware replay. In *Conference on Applied Machine Learning in Information Security (CAMLIS)*, 2025.

Vaibhav Rastogi, Yan Chen, and Xuxian Jiang. Droidchameleon: Evaluating android anti-malware against transformation attacks. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2013.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Scott Coull and Christopher Gardner. What are deep neural networks learning about malware? Google Cloud Blog, 2018. URL https://cloud.google.com/blog/topics/threat-intelligence/what-are-deep-neural-networks-learning-about-malware. Accessed: 2026-02-19.

Silvia Sebastián and Juan Caballero. Avclass2: Massive malware tag extraction from av labels. In *Annual Computer Security Applications Conference (ACSAC)*, 2020.

PV Shijo and AJPCS Salim. Integrated static and dynamic analysis for malware detection. *Procedia Computer Science*, 2015.

Hossein Shokouhinejad, Roozbeh Razavi-Far, Hesamodin Mohammadian, Mahdi Rabbani, Samuel Ansong, Griffin Higgins, and Ali A Ghorbani. Recent advances in malware detection: Graph learning and explainability. *arXiv preprint arXiv:2502.10556*, 2025.

Kimberly Tam, Salahuddin J Khan, Aristide Fattori, and Lorenzo Cavallaro. Copperdroid: automatic reconstruction of android malware behaviors. In *Network and Distributed System Security Symposium (NDSS)*, 2015.

Gido M van de Ven, Nicholas Soures, and Dhireesha Kudithipudi. Continual learning and catastrophic forgetting. *arXiv preprint arXiv:2403.05175*, 2024.

Valentin Vielzeuf, Alexis Lechervy, Stéphane Pateux, and Frédéric Jurie. Centralnet: a multilayer approach for multimodal fusion. In *European Conference on Computer Vision (ECCV) workshops*, 2018.

Virustotal. VirusTotal — virustotal.com. https://www.virustotal.com/gui/intelligence-overview. [Accessed 21-10-2025].

VirusTotal. VirusTotal – Stats, 2025. https://www.virustotal.com/gui/stats.

Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *International Conference on Machine Learning (ICML)*, 2009.

Bozhi Wu, Sen Chen, Cuiyun Gao, Lingling Fan, Yang Liu, Weiping Wen, and Michael R. Lyu. Why an android app is classified as malware: Toward malware classification interpretation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2021a.

Yueming Wu, Xiaodi Li, Deqing Zou, Wei Yang, Xin Zhang, and Hai Jin. Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019.

Yueming Wu, Deqing Zou, Wei Yang, Xiang Li, and Hai Jin. Homdroid: detecting android covert malware by social-network homophily analysis. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2021b.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *AAAI conference on artificial intelligence (AAAI)*, 2021.

Haichuan Xu, Mingxuan Yao, Runze Zhang, Mohamed Moustafa Dawoud, Jeman Park, and Brendan Saltaformaggio. {DVa}: Extracting victims and abuse vectors from android accessibility malware. In *USENIX Security Symposium*, 2024.

Jiayun Xu, Yingjiu Li, Robert H. Deng, and Ke Xu. Sdac: A slow-aging solution for android malware detection using semantic distance based api clustering. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2020.

Lok-Kwong Yan and Heng Yin. Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In *USENIX Security Symposium*, 2012.

Guowei Yang, Jeffrey Jones, Austin Moninger, and Meiru Che. How do android operating system updates impact apps? In *International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 2018.

Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. CADE: Detecting and explaining concept drift samples for security applications. In *USENIX Security Symposium*, 2021.

Amir Zadeh, Rowan Zellers, Eli Pincus, and Louis-Philippe Morency. Mosi: multimodal corpus of sentiment intensity and subjectivity analysis in online opinion videos. *arXiv preprint arXiv:1606.06259*, 2016.

AmirAli Bagher Zadeh, Paul Pu Liang, Soujanya Poria, Erik Cambria, and Louis-Philippe Morency. Multimodal language analysis in the wild: Cmu-mosei dataset and interpretable dynamic fusion graph. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

Samantha Zeitlin. Automating the security protections rapid response to malware. Elastic Security Labs, 2023. URL `https://www.elastic.co/security-labs/automating-security-protections-rapid-response-to-malware`. Accessed: 2026-09-19.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. *Journal of Machine Learning Research (JMLR)*, 2017.

Sanfeng Zhang, Heng Su, Hongxian Liu, and Wang Yang. Mpdroid: A multimodal pre-training android malware detection method with static and dynamic features. *Computers & Security*, 2025.

Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

Kaifa Zhao, Hao Zhou, Yulin Zhu, Xian Zhan, Kai Zhou, Jianfeng Li, Le Yu, Wei Yuan, and Xiapu Luo. Structural attack against graph based android malware detection. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

Dali Zhu, Tong Xi, Pengfei Jing, Di Wu, Qing Xia, and Yiming Zhang. A transparent and multimodal malware detection method for android apps. In *ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*, 2019.

OVERVIEW OF APPENDIX

The following appendices provide further information:

- **A Dataset Statistics**
- **B McNdroid vs Prior Multimodal Datasets**
- **C Motivating Case Study: Recent GodFather Variants**
- **D Model Descriptions**
- **E Multimodal Fusion Strategies**
- **F Extended Temporal Results**
- **G Feature-Space Visualization**
- **H Malware Family Temporal Stability**
- **I Feature Importance Across Modalities**
- **J Sensitivity to Missing Modalities**
- **K Prediction Uncertainty Under Label Drift**
- **L Additional Concept-Drift Adaptation Results**
- **M Unsupervised Family Structure Analysis**
- **N Continual Learning on McNdroid**
- **O Graph Neural Network Baselines**
- **P Computational Resources**
- **Q Security Impact and Responsible Use**

A DATASET STATISTICS

Table 4 summarizes the temporal composition of the dataset, reporting the number of malware and benign samples per year along with family-level statistics. Overall, the dataset exhibits substantial variation in both class distribution and family diversity across years, reflecting the evolving nature of the Android malware ecosystem. Earlier years (e.g., 2013–2014) show moderate family diversity, while later years (e.g., 2019–2022) exhibit a significant increase in the number of distinct malware families, indicating higher behavioral heterogeneity. The number of newly observed families also varies considerably across years, highlighting periods of rapid emergence of previously unseen malware variants during the observation period.

A notable characteristic of the dataset is the large proportion of singleton families—families represented by a single sample within a given year—which suggests a long-tailed distribution of malware. This reflects the presence of many rare or short-lived variants, a common phenomenon in real-world malware dynamics Lakshmanan (2024). In contrast, the number of samples labeled as *unknown* remains relatively small, indicating that the family labeling process is generally reliable.

Table 5 provides a fine-grained view of the temporal dynamics of the ten most prevalent malware families. The distribution reveals strong temporal shifts in family dominance: certain families (e.g., *dowgin*, *airpush*, *kuguo*) are prominent in earlier years, while others exhibit transient spikes or gradual decline over time. This non-stationary behavior highlights the presence of concept drift at the family level, where the relative prevalence of malware families changes significantly across years. Such dynamics underscore the importance of evaluating models under temporally evolving conditions.

B MCNDROID VS PRIOR MULTIMODAL DATASETS

Table 6 highlights the limitations of existing multimodal datasets for studying temporal multimodal generalization. Prior benchmarks span diverse sectors, including affective computing, healthcare, robotics, finance, human-computer interaction, multimedia, and cybersecurity. However, their evaluation tasks are primarily conventional in-distribution prediction problems, such as sarcasm detection,

Table 4: Year-wise distribution of malware and benign samples with corresponding malware family number.

Year	Malware	Benign	Valid Families	New Families	Singleton	Unknown
2013	38,324	58,226	230	230	1,411	27
2014	37,061	77,108	227	94	1,950	322
2016	30,683	39,516	350	159	4,545	154
2017	19,781	48,620	202	87	8,894	1,068
2018	34,254	45,688	345	132	19,059	1,182
2019	46,553	32,578	624	261	18,370	21
2020	41,086	25,164	558	123	25,919	14
2021	24,727	28,707	286	50	19,626	16
2022	44,133	36,649	641	159	24,558	3
2023	5,526	40,220	199	31	3,992	15
2024	760	54,869	61	14	604	0
2025	612	48,010	73	14	347	2

Table 5: Year-wise distribution of the top 10 malware families by total sample count, excluding benign, unknown, and singleton samples.

Year	dowgin	airpush	kuguo	smsreg	dnotua	gappusin	revmob	leadbolt	adwo	youmi
2013	6,839	5,615	5,295	1,041	2	3,247	296	1,807	2,822	1,004
2014	11,824 ↑	4,135 ↓	3,304 ↓	1,358 ↑	5 ↑	931 ↓	2,391 ↑	2,040 ↑	606 ↓	853 ↓
2016	2,259 ↓	2,227 ↓	4,025 ↑	1,089 ↓	35 ↑	1,659 ↑	1,869 ↓	930 ↓	947 ↑	1,168 ↑
2017	32 ↓	1,151 ↓	3 ↓	67 ↓	3,547 ↑	3 ↓	281 ↓	373 ↓	6 ↓	34 ↓
2018	17 ↓	1,178 ↑	8 ↑	142 ↑	5,053 ↑	5 ↑	170 ↓	527 ↑	6	35 ↑
2019	980 ↑	988 ↓	679 ↑	5,069 ↑	451 ↓	676 ↑	1,552 ↑	464 ↓	402 ↑	321 ↑
2020	619 ↓	522 ↓	533 ↓	2,245 ↓	616 ↑	303 ↓	67 ↓	137 ↓	123 ↓	347 ↑
2021	100 ↓	101 ↓	118 ↓	538 ↓	85 ↓	118 ↓	48 ↓	55 ↓	39 ↓	96 ↓
2022	570 ↑	159 ↑	774 ↑	2,874 ↑	772 ↑	336 ↑	43 ↓	62 ↑	96 ↑	372 ↑
2023	54 ↓	50 ↓	33 ↓	132 ↓	7 ↓	48 ↓	22 ↓	18 ↓	30 ↓	48 ↓
2024	0 ↓	3 ↓	0 ↓	7 ↓	1 ↓	0 ↓	0 ↓	0 ↓	0 ↓	0 ↓
2025	0	6 ↑	0	5 ↓	0 ↓	0	0	2 ↑	0	0

sentiment and emotion recognition, mortality and ICD-9 code prediction, object/contact/robot pose estimation, stock price and volatility forecasting, interface understanding, action recognition, movie genre classification, digit recognition, and malware classification. These datasets are valuable for studying multimodal fusion and representation learning, but they are not designed to evaluate model aging, concept drift, or distribution shift under naturally evolving deployment conditions.

McNdroid addresses these limitations by extending multimodal benchmarking to Android malware evolution. Unlike prior datasets in Table 6, McNdroid provides aligned static, dynamic, and graph-structured representations together with temporally ordered tasks for malware detection, family classification, temporal generalization, model aging, concept-drift detection, distribution shift analysis, and intra-/inter-family shift analysis.

C Motivating Case Study: Recent GodFather Variants

We use the recent virtualization-based GodFather Android malware reported by Zimperium zLabs Ortega & Pratapagiri (2025) as an external motivating case study, not as an part of this paper's contribution. This campaign highlights why Android malware detection benefits from multimodal representations. Unlike conventional overlay-based banking malware Xu et al. (2024), recent GodFather variants execute legitimate banking and cryptocurrency applications inside an attacker-controlled virtualized environment. This preserves the appearance of the original app while allowing the malware to intercept user inputs, alter runtime behavior, and exfiltrate sensitive information.

This design exposes complementary evidence across three modalities. Static Drebin-like features reveal the declared attack surface, including suspicious permissions, accessibility-service declarations, manifest components, encoded configuration strings, embedded payloads, target package names, C2 indicators, and virtualization or hooking artifacts. However, static analysis is weakened

Table 6: Representative multimodal datasets across different application sectors.

Sector	Dataset	Modalities / Data Types	# Samples	Prediction Task
Cybersecurity	**McNdroid (Ours)**	Static, Dynamic, and Graph structure	858,859	Malware detection, temporal generalization, concept-drift detection, distributional shift analysis, intra-family shift, inter-family shift and family classification
	MalNet Freitas et al. (2022; 2021)	Graph structure, node features	1,262,024	Malware detection and classification
Affective Computing	MUSTARD Castro et al. (2019)	Text, Video, Audio	690	Sarcasm detection
	CMU-MOSI Zadeh et al. (2016)	Text, Video, Audio	2,199	Sentiment analysis
	UR-FUNNY Hasan et al. (2019)	Text, Video, Audio	16,514	Humor detection
	CMU-MOSEI Zadeh et al. (2018)	Text, Video, Audio	22,777	Sentiment and emotion recognition
Healthcare	MIMIC Johnson et al. (2016)	Time-series, Tabular	36,212	Mortality and ICD-9 code prediction
Robotics	MuJoCo Push Lee et al. (2020a)	Image, Force, Proprioception	37,990	Object pose estimation
	Vision&Touch Lee et al. (2020b)	Image, Force, Proprioception	147,000	Contact and robot pose estimation
Finance	Stocks-F&B Liang et al. (2021)	Time-series financial indicators	5,218	Stock price and volatility prediction
	Stocks-Health Liang et al. (2021)	Time-series financial indicators	5,218	Stock price and volatility prediction
	Stocks-Tech Liang et al. (2021)	Time-series financial indicators	5,218	Stock price and volatility prediction
Human-Computer Interaction	Enrico Leiva et al. (2020)	Image, Structured layout	1,460	Interface design understanding
Multimedia	Kinetics400-S Kay et al. (2017)	Video, Audio, Optical flow	2,624	Human action recognition
	MM-IMDb Arevalo et al. (2017)	Text, Image	25,959	Movie genre classification
	AV-MNIST Vielzeuf et al. (2018)	Image, Audio	70,000	Digit classification
	Kinetics400-L Kay et al. (2017)	Video, Audio, Optical flow	306,245	Human action recognition

by anti-analysis techniques such as APK packaging manipulation, JADX-blocking fields, manifest obfuscation, and dynamically shifted logic. Dynamic analysis captures the executed attack workflow, including accessibility abuse, virtualized app launch, user redirection, runtime hooking, C2 communication, payload download, credential interception, and remote-control behavior. Yet these traces may be incomplete when behavior depends on target apps, granted permissions, live C2 infrastructure, regional conditions, delayed execution, or attacker-issued commands. Graph-based features provide the missing structural view by modeling how the host malware, accessibility service, virtual container, proxy activity manager, hooking framework, C2 handler, target-app interceptors, and sensitive source-to-sink paths are coordinated.

Thus, the GodFather case demonstrates the brittleness of single-modality detection. Static features describe what suspicious capabilities are present, dynamic features describe what behavior is observed at runtime, and graph-based features describe how malicious components are organized. Their fusion provides a more complete and explainable representation for detecting advanced Android banking malware under obfuscation, sandbox evasion, and modular attack design.

D Model Descriptions

Multi-Layer Perceptron (MLP). The MLP model is implemented as a four-layer fully connected neural network for malware classification, adapted from prior work Rahman et al. (2022; 2025). The hidden layers have dimensions 1024, 512, 256, and 128, respectively. Each hidden layer is followed by batch normalization, ReLU activation, and dropout with a rate of 0.5. For binary classification, the final classifier consists of a linear layer followed by a sigmoid activation, while for multi-class classification, a linear output layer is used. The model is trained using the Adam optimizer with a learning rate of 0.001 and a batch size of 512.

LightGBM. In addition to MLP, we employ LightGBM Ke et al. (2017), a gradient-boosted decision tree ensemble, for both binary and multi-class classification. The model is trained with up to 5000 estimators and a learning rate of 0.02. Each tree is allowed up to 256 leaves to provide sufficient model capacity. To mitigate overfitting, we apply 80% subsampling of both rows and features, along with L1 and L2 regularization. Early stopping is used based on validation performance, halting training if no improvement is observed for 100 rounds. For binary classification, the Area Under the Curve (AUC) metric is used, while multi-class classification uses multi-class log loss. These configurations follow standard practices of prior works in malware detection such as EMBER Anderson & Roth (2018) and TESSERACT Pendlebury et al. (2019).

XGBoost. The XGBoost model is configured with a maximum tree depth of 12 and a learning rate of 0.05. We use the histogram-based tree construction method with GPU acceleration for efficient training. The objective function is binary logistic loss for binary classification and soft-probability output for multi-class classification Chen & Guestrin (2016). The model is trained for up to 3000 boosting rounds, with additional regularization through row and feature subsampling.

Support Vector Machine (SVM). A linear Support Vector Machine is implemented using `LinearSVC`. The model is trained with a maximum of 20,000 iterations and a tolerance of 10^{-3} Chen et al. (2023). To obtain probabilistic outputs for ranking-based evaluation metrics, the classifier

is calibrated using `CalibratedClassifierCV` with sigmoid calibration. A `MaxAbsScaler` is applied within a pipeline to ensure compatibility with high-dimensional sparse features.

DetectBERT. DetectBERT is implemented as a lightweight transformer-based model for malware classification. The input feature vector is first projected into a hidden representation using a fully connected layer, followed by a learnable `[CLS]` token. The resulting sequence is processed by two stacked transformer layers utilizing Nyström-based self-attention Xiong et al. (2021), with residual connections and LayerNorm. The model supports multiple aggregation strategies, including `[CLS]`-based representation, summation, averaging, and random token selection. The final representation is passed through a linear classification head.

Transformer-based. We utilize a Vision Transformer (ViT)-based model adapted for malware classification using static feature vectors Haque et al. (2026). Each input sample is projected into a hidden-dimensional token via a linear embedding layer. A learnable `[CLS]` token is prepended, and positional embeddings are added. The resulting sequence is processed through a stack of transformer encoder blocks, each consisting of LayerNorm, multi-head self-attention, and a feed-forward network with GELU activation and residual connections. The final `[CLS]` representation is used for classification through a LayerNorm and linear output layer, enabling both binary and fine-grained multi-class malware classification.

E Multimodal Fusion Strategies

We consider four model-level strategies for integrating the three malware feature modalities: static, graph-based, and dynamic features. The first three strategies use conventional classifiers over fused feature spaces or prediction scores, while the fourth learns modality-specific embeddings and explicitly models cross-modal interactions through attention.

E.1 Pairwise Feature Fusion

Pairwise fusion evaluates whether two modalities provide complementary information when modeled jointly. For each modality pair (m_i, m_j), the corresponding feature vectors are concatenated into a single input representation: $x^{(i,j)} = [x^{(i)} \| x^{(j)}]$. We instantiate pairwise fusion for Static+Dynamic, Static+Graph-based, and Dynamic+Graph-based, training a separate classifier on each concatenated feature space. This provides a controlled way to assess complementary information between two modalities without requiring all three simultaneously. The selected feature are horizontally stacked, the classifier is fit on the fused representation.

E.2 Late Fusion

Late fusion combines modalities at the decision level by training one base classifier per modality and concatenating their predicted probabilities into a three-dimensional meta-representation:

$$z = [p_{\text{Static}}, p_{\text{Graph-based}}, p_{\text{Dynamic}}].$$

A stacker is then trained on out-of-fold predictions, which prevents reliance on inflated in-sample scores. At test time, it maps the three modality probabilities to the final prediction.

E.3 Cross-Attention Fusion

Cross-attention fusion replaces direct feature concatenation with a learned representation-level interaction mechanism Kim et al. (2018). Each modality is first processed by an MLP encoder Rahman et al. (2022): $h_m = E_m(x^{(m)})$, where E_m denotes the modality-specific encoder for modality m, and h_m is the corresponding dense modality token. Static, Graph-based, and Dynamic modalities are projected into a shared latent space using separate feed-forward encoders composed of linear transformations, normalization, nonlinear activation, and dropout. A learned modality embedding is added to each projected token to preserve modality identity in the shared representation space.

The fusion module performs directional cross-attention. Each modality token serves as a query and attends to the other two modality tokens as keys and values:

$$\tilde{h}_{\text{Static}} = \text{Attn}_{\text{Static}}(h_{\text{Static}}, [h_{\text{Graph-based}}, h_{\text{Dynamic}}]),$$

Table 7: Comparison of multimodal strategies across temporal splits using F1-score.

Split	Model	Static + Dynamic	Static + Graph-based	Dynamic + Graph-based	Late Fusion	Feature Fusion	Cross-Attention
IID	MLP	89.3±8.3	83.9±6.7	80.6±11.6	90.7±7.8	84.2±9.3	–
	LightGBM	90.6±8.7	91.9±7.4	91.4±7.2	92.6±6.4	91.6±7.6	–
	XGBoost	90.7±8.9	92.1±7.1	91.5±7.2	91.7±6.6	91.6±7.8	–
	SVM	89.8±7.6	89.5±8.9	87.1±8.7	90.5±7.0	89.2±9.4	–
	DetectBERT	89.4±7.9	89.9±5.7	87.1±7.2	90.7±8.3	89.9±5.6	–
	ViT	89.0±8.2	91.0±5.8	87.6±6.9	90.5±7.9	90.0±6.3	–
	Cross-Attention	–	–	–	–	–	90.0±8.6
NEAR	MLP	42.6±37.5	31.4±41.9	29.5±25.9	41.5±38.7	30.1±37.2	–
	LightGBM	39.7±40.5	39.5±44.2	36.9±39.5	36.5±42.9	39.4±42.8	–
	XGBoost	39.2±39.6	39.4±42.5	37.3±37.6	39.5±40.2	39.6±41.5	–
	SVM	46.2±37.7	39.3±39.8	33.3±35.4	41.3±42.3	39.3±41.2	–
	DetectBERT	44.0±34.5	36.1±43.8	41.1±36.1	42.8±41.7	38.8±42.1	–
	ViT	44.8±38.5	36.4±42.6	39.5±36.3	41.9±41.1	38.6±44.1	–
	Cross-Attention	–	–	–	–	–	40.8±40.7
FAR	MLP	28.2±18.2	15.6±13.1	15.0±11.2	29.0±22.5	20.3±16.0	–
	LightGBM	37.7±27.3	48.0±29.0	41.3±21.6	31.9±17.1	45.4±30.4	–
	XGBoost	33.2±22.4	42.7±27.5	27.2±20.2	33.6±20.3	42.0±30.3	–
	SVM	41.6±23.5	34.1±19.2	29.5±18.0	31.8±20.4	40.2±22.6	–
	DetectBERT	36.7±23.6	38.1±27.1	27.9±19.6	28.7±21.7	35.3±24.2	–
	ViT	34.2±21.7	43.4±27.9	25.6±16.1	31.1±18.9	33.7±21.6	–
	Cross-Attention	–	–	–	–	–	33.0±19.9

$$\tilde{h}_{\text{Graph-based}} = \text{Attn}_{\text{Graph-based}}(h_{\text{Graph-based}}, [h_{\text{Static}}, h_{\text{Dynamic}}]),$$
$$\tilde{h}_{\text{Dynamic}} = \text{Attn}_{\text{Dynamic}}(h_{\text{Dynamic}}, [h_{\text{Static}}, h_{\text{Graph-based}}]).$$

This design allows each modality to selectively incorporate complementary evidence from the other modalities while maintaining its own query-specific representation. The attended context is combined with the original modality token using a residual connection and then normalized. The three resulting modality-specific representations are concatenated to form the final fused representation, which is normalized once more before being passed to the classifier.

F Extended Temporal Results

Tables 7 and 8 report the F1-score and ROC-AUC results for all multimodal fusion strategies across IID, NEAR, and FAR temporal splits. A subset of these results is analyzed in the main paper; here, we provide the model- and metric-level comparison. Under IID evaluation, multimodal methods achieve uniformly strong performance, indicating that all modality combinations are effective when the test distribution is temporally close to the training distribution. In contrast, performance degrades substantially under NEAR and FAR splits. Among the fusion strategies, Static+Graph-based provides the clearest gains under long-term drift, achieving the strongest FAR F1-score for several model families, including LightGBM, XGBoost, DetectBERT, and ViT. However, this trend is not universal, as Static+Dynamic and late fusion remain competitive for specific classifiers. These results indicate that multimodal fusion utilizes complementary information from static, dynamic, and graph-based representations, but its effectiveness depends on both the modality pairing and the downstream model.

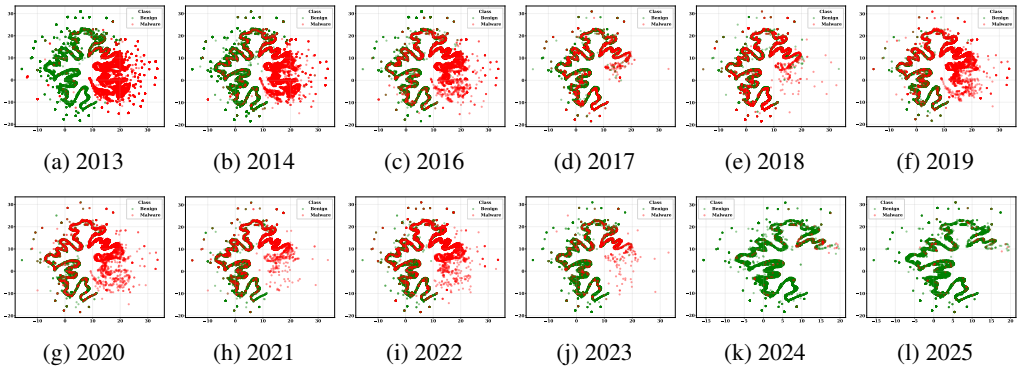
G Feature-Space Visualization

We use UMAP visualizations to assess the class-wise organization of learned representations across years and modalities. For each modality, we train an MLP using 2013 samples and extract 128-dimensional penultimate-layer embeddings from the fixed 2013-trained model for each year-specific test set. To obtain a consistent visualization space, we fit PCA with 50 retained components followed by UMAP on the 2013 training embeddings and apply the resulting fixed PCA–UMAP transformation to all test samples. This reference-fixed protocol avoids fitting a separate visualization for each year, ensuring that the observed year-wise malware–benign overlap, separation, and density patterns are measured in a common representation space induced by the source-year model. These results indicate that temporal drift affects each modality differently, motivating multimodal evaluation under modality-specific distribution shift. They also suggest a promising direction for future work: incorporating semantic API similarity, as explored by APIGraph Zhang et al. (2020), into graph-based representations to improve robustness against API-level and framework-level evolution.

Figures 9, 11, and 10 show yearly UMAP projections for static, graph-based, and dynamic representations, respectively. In 2013–2014, benign and malware samples are relatively separable across

Table 8: Comparison of multimodal strategies across temporal splits using ROC-AUC.

Split	Model	Static + Dynamic	Static + Graph-based	Dynamic + Graph-based	Late Fusion	Feature Fusion	Cross-Attention
IID	MLP	97.3±2.4	95.9±2.0	93.4±4.3	98.3±1.4	95.8±2.9	–
	LightGBM	98.3±1.8	98.7±1.2	98.1±1.9	98.7±1.2	98.6±1.4	–
	XGBoost	98.1±2.0	98.8±1.1	98.0±2.1	97.8±2.0	98.5±1.6	–
	SVM	97.0±2.8	96.4±3.8	95.5±3.8	98.1±1.3	96.3±4.0	–
	DetectBERT	96.3±3.4	96.1±3.5	94.5±4.3	98.1±1.8	95.6±3.7	–
	ViT	97.4±2.2	97.9±1.4	95.3±3.6	98.2±1.5	96.7±2.7	–
	Cross-Attention	–	–	–	–	–	97.8±2.1
NEAR	MLP	77.9±18.9	66.0±26.6	66.0±17.0	80.0±16.7	64.4±22.5	–
	LightGBM	78.9±20.7	73.9±28.3	86.1±7.9	82.4±16.0	80.7±18.9	–
	XGBoost	81.1±17.0	79.3±20.8	83.3±11.3	80.8±13.4	81.7±17.4	–
	SVM	78.0±17.1	75.3±16.0	78.8±8.5	79.1±17.8	77.5±14.4	–
	DetectBERT	78.2±16.2	82.4±7.8	80.1±8.2	80.7±16.9	82.6±7.9	–
	ViT	80.8±14.0	74.6±23.0	72.7±11.4	77.9±19.8	76.6±16.3	–
	Cross-Attention	–	–	–	–	–	73.1±25.9
FAR	MLP	76.7±10.1	72.8±10.3	69.1±6.8	79.3±13.1	69.8±10.8	–
	LightGBM	79.7±15.7	79.1±19.2	82.0±14.1	81.9±14.3	80.9±17.1	–
	XGBoost	80.1±14.6	80.3±17.4	77.6±14.5	79.6±13.8	80.1±16.8	–
	SVM	79.8±11.0	83.6±11.9	82.7±9.2	82.8±11.1	83.7±11.9	–
	DetectBERT	75.9±12.4	84.0±12.8	71.5±9.1	79.9±12.9	77.0±9.2	–
	ViT	76.6±9.8	69.2±17.4	56.1±5.5	79.1±13.4	62.6±6.2	–
	Cross-Attention	–	–	–	–	–	75.6±13.7



(a) 2013 (b) 2014 (c) 2016 (d) 2017 (e) 2018 (f) 2019

(g) 2020 (h) 2021 (i) 2022 (j) 2023 (k) 2024 (l) 2025

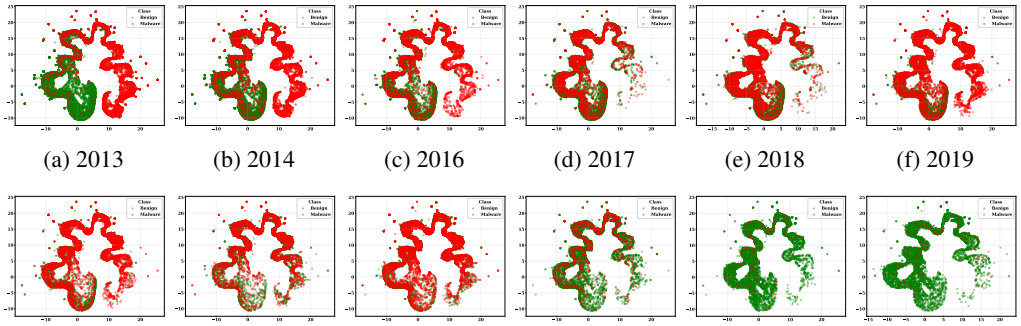
Figure 9: Year-wise visualization of the static feature space for malware and benign samples from 2013 to 2025, excluding 2015.

modalities, consistent with the strong IID results in Table 2. From 2016 onward, the embeddings become increasingly mixed, with the strongest overlap in 2017–2018, where malware samples occupy regions previously dominated by benign samples. This provides a representation-level explanation for the sharp degradation observed in Figure 3. Static features preserve the most coherent class structure among the three modalities, whereas graph-based features exhibit the largest structural instability. In particular, graph representations become less class-separable after 2017 and degrade further in later years, plausibly due to shifts in Android APIs, framework usage, third-party libraries, and malware implementation patterns that alter function-call structures Arp et al. (2014); Zhang et al. (2020); He et al. (2023a); Zhang et al. (2025); Zhao et al. (2021); Haque et al. (2026).

H Malware Family Temporal Stability

Setting. We analyze the temporal stability of malware families by quantifying how their feature distributions evolve across years within each modality. For each modality, we load yearly feature representations and associated metadata, and restrict the analysis to samples labeled as malware with valid family annotations across the observation period.

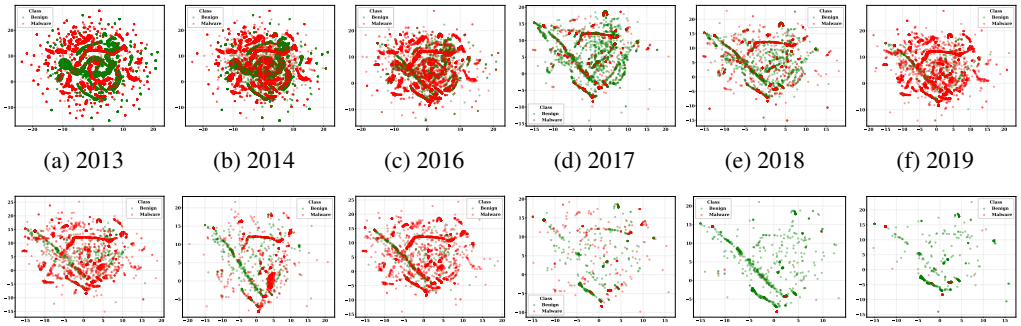
To ensure statistical reliability, we retain only families that appear consistently across multiple years with sufficient representation. For each feature dimension, empirical distributions are estimated using normalized histograms, and their discrepancy is measured using the Jeffrey divergence Jeffreys (1946). The divergence is computed independently for each feature and then averaged across all features to obtain a single drift score per family and year pair.



(a) 2013 (b) 2014 (c) 2016 (d) 2017 (e) 2018 (f) 2019

(g) 2020 (h) 2021 (i) 2022 (j) 2023 (k) 2024 (l) 2025

Figure 10: Year-wise dynamic feature space visualization of malware and benign samples from 2013 to 2025, excluding 2015.



(a) 2013 (b) 2014 (c) 2016 (d) 2017 (e) 2018 (f) 2019

(g) 2020 (h) 2021 (i) 2022 (j) 2023 (k) 2024 (l) 2025

Figure 11: Year-wise graph-based feature space visualization of malware and benign samples from 2013 to 2025, excluding 2015.

Analysis. Figure 12 presents a comparative analysis of stability scores across twenty distinct malware families, segmented by feature modality. Dynamic features consistently exhibit the highest stability, with median scores approximating 1.00 across the top 20 malware families. Static features, maintain high stability with narrow inter-quartile ranges. In contrast, graph based exhibit substantial volatility, they consistently yield the lowest stability scores and demonstrate the widest distribution of values across all analyzed families. Overall, these results confirm, that all while all three modalities produce stable representations for the majority of the malware families, graph-based are significantly more sensitive to intra-family structural diversity.

I Feature Importance Across Modalities

Setting. To understand which feature groups drive model decisions, we conduct a grouped SHAP-based importance analysis. We analyze feature importance across three modalities: static, graph-based, and dynamic features. Model explanations are computed using SHAP Lundberg & Lee (2017) values, where importance score is measured as the mean absolute SHAP contribution of each feature Haque et al. (2026). Feature-level scores are aggregated into semantic groups, and the top five groups are retained for each modality and temporal split.

Analysis. Table 9 summarizes the most influential feature groups across modalities under increasing temporal shift. For static features, importance is relatively distributed, with URL domains, permissions, and restricted APIs consistently dominant, though their contributions vary across NEAR and FAR settings. Graph-based features show a clear shift in reliance: API-level signals (e.g., TelephonyManager, WebView) decrease over time, while structural properties such as graph statistics become more prominent in the FAR setting. Dynamic features remain largely driven by string and

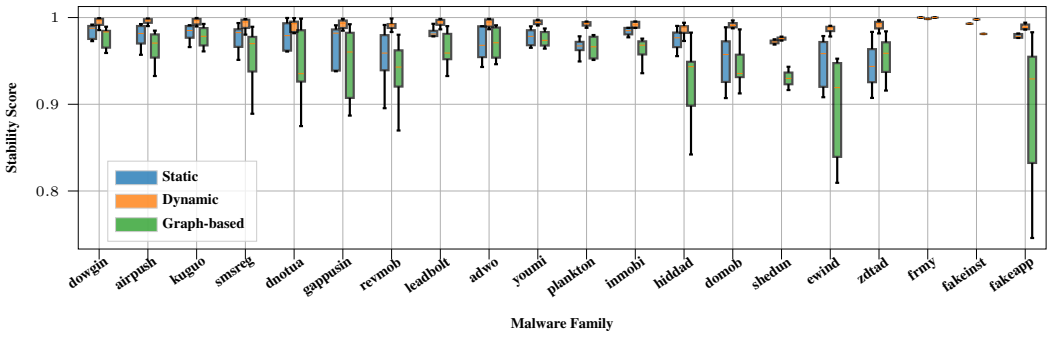


Figure 12: Per-family stability score distributions across three feature modalities for the top-20 Android malware families. Each box represents the interquartile range over all samples within a family.

Table 9: Top feature groups by normalized SHAP score, shown as % within each modality and split.

Modality	Top feature groups	IID	NEAR	FAR
Static	URL domains / Requested permissions / Restricted APIs Suspicious APIs / Intent filters	24.38±0.16 / 19.09±0.17 / 14.86±0.27 10.23±0.27 / 12.29±0.02	20.90±0.30 / 18.37±0.15 / 25.45±0.36 12.69±0.11 / 8.20±0.02	22.94±0.04 / 21.87±0.13 / 16.99±0.07 16.31±0.06 / 9.76±0.02
Graph-based	TelephonyManager / WebView / Activity APIs Graph statistics / View APIs	14.30±0.07 / 7.78±0.14 / 6.21±0.09 5.35±0.07 / 4.47±0.14	7.64±0.05 / 7.41±0.11 / 5.73±0.10 7.87±0.20 / 5.29±0.20	4.81±0.03 / 5.30±0.08 / 4.93±0.12 15.50±0.16 / 7.47±0.26
Dynamic	String / Path / Domain Component / Permissions	26.73±0.05 / 26.80±0.10 / 16.39±0.09 5.27±0.03 / 3.34±0.20	29.26±0.03 / 24.10±0.06 / 7.11±0.10 11.65±0.05 / 0.00±0.00	27.52±0.04 / 18.57±0.24 / 20.81±0.11 7.50±0.07 / 0.00±0.00

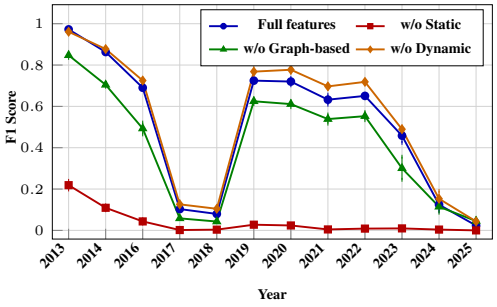


Figure 13: Year-wise F1 score comparison for full features vs missing modalities.

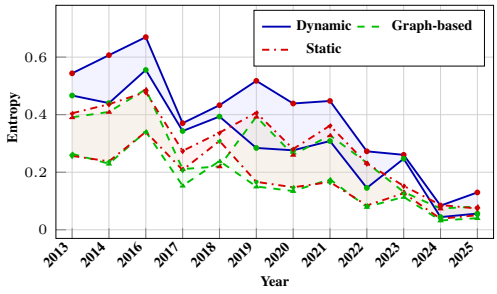


Figure 14: Entropy comparison across modalities and drifted–undrifted pairs.

path-based signals across all splits. Overall, these trends demonstrate that different modalities capture complementary aspects of the data under distribution shift.

J Sensitivity to Missing Modalities

Setting. In real-world scenarios, a detection system cannot always rely on the availability of all feature modalities due to noise, extraction failures, or incomplete data sources Liu et al. (2026). To assess sensitivity under such conditions, we simulate missing-feature settings by selectively removing or masking one or more modalities at inference time while keeping the model trained on complete data Feng et al. (2023). This setup allows us to systematically assess how performance degrades when features are unavailable and to identify which modalities are most critical for reliable detection.

Analysis. Figure 13 evaluates the sensitivity of McNdroid under missing-feature conditions. The results show that static features are the most indispensable modality: removing them causes F1 to collapse across nearly all test years, indicating that static representations provide the dominant signal for detection. Graph-based features are also beneficial, with their removal producing consistent but less severe degradation, while dynamic features appear less critical to overall detection performance and in several years their removal slightly improves F1.

Table 10: Performance of adaptation methods on the McNdroid dataset across IID, NEAR, and FAR settings using three modalities and feature fusion (budget = 50).

Method	Setting	Static	Dynamic	Graph-based	Feature Fusion
HCC Chen et al. (2023)	IID	89.38±0.96	72.59±0.28	83.63±0.88	86.78±0.25
	NEAR	77.47±1.05	56.21±2.51	65.99±1.32	70.62±1.17
	FAR	61.68±0.50	34.21±0.27	53.68±0.59	62.98±1.30
CADE Yang et al. (2021)	IID	87.12±0.69	72.65±0.99	79.72±0.88	80.34±1.85
	NEAR	47.17±0.70	30.99±2.90	30.40±6.83	34.97±8.82
	FAR	18.57±0.09	11.54±1.13	7.30±1.23	9.94±3.21
DAN–MMD Long et al. (2015)	IID	82.09±1.33	70.99±0.99	77.27±1.68	78.14±1.34
	NEAR	50.74±10.56	45.25±3.30	28.17±4.45	42.37±4.32
	FAR	26.72±2.79	19.31±0.57	16.84±11.75	17.53±8.81

Table 11: Performance of adaptation methods on the McNdroid dataset across IID, NEAR, and FAR settings using three modalities and feature fusion (budget = 100).

Method	Setting	Static	Dynamic	Graph-based	Feature Fusion
HCC Chen et al. (2023)	IID	91.70±0.54	73.78±0.25	87.09±0.67	89.58±1.23
	NEAR	82.63±0.45	62.03±6.60	61.01±0.73	77.65±0.93
	FAR	67.15±1.51	39.27±2.35	61.01±0.73	68.23±1.46
CADE Yang et al. (2021)	IID	87.69±0.32	70.68±1.27	80.82±1.63	83.92±2.06
	NEAR	46.44±5.12	23.46±5.91	29.98±2.71	42.16±1.16
	FAR	19.53±1.43	10.37±0.13	5.66±0.67	12.62±2.70
DAN–MMD Long et al. (2015)	IID	83.27±1.13	71.11±0.61	78.07±2.25	79.78±1.46
	NEAR	47.62±4.75	46.68±2.86	36.15±10.07	43.70±3.45
	FAR	21.10±2.96	17.37±0.94	20.39±7.31	14.27±3.77

K Prediction Uncertainty Under Label Drift

Setting. To evaluate the effect of label drift on model uncertainty, we analyze samples spanning the years 2013 to 2025 whose ground-truth labels were revised after initial annotation. For each year and each of the three feature modalities (static, graph-based, and dynamic), an XGBoost classifier is trained on 70% of the non-drifted samples and evaluated separately on the remaining 30% of non-drifted samples and on the full set of drifted samples. We measure prediction uncertainty using binary entropy and margin distance from the decision boundary Baier et al. (2021), and compare these distributions across drifted and non-drifted groups to assess whether label-drifted samples are systematically harder for the model to classify.

Analysis. Figure 14 reports the mean prediction entropy of drifted and non-drifted samples across the three feature modalities over time. The results show that uncertainty is consistently higher for drifted samples in the early years, with the gap being most pronounced in 2016, where static, graph-based, and dynamic modalities report entropy values of 0.479, 0.488, and 0.669 respectively, compared to 0.340, 0.341, and 0.555 for non-drifted samples. However, this separation narrows in later years and nearly disappears by 2023–2025. This trend should be interpreted cautiously, as the later years contain smaller drifted-sample sizes, which can make the entropy estimates less stable and reduce the reliability of the observed uncertainty gap.

L Additional Concept-Drift Adaptation Results

Tables 10, 11, and 12 show that the trends reported in Section 7 persist under smaller labeling budgets. HCC Chen et al. (2023) consistently provides stronger adaptation than CADE Yang et al. (2021) and DAN–MMD Long et al. (2015), but all methods exhibit substantial degradation when moving from IID to NEAR and FAR evaluation across all modalities. Comparing these results with the 400-sample setting in Table 3 further shows that increasing the monthly labeling budget improves some IID and NEAR results, but does not resolve the FAR-setting degradation. Thus, the failure mode is not simply a consequence of insufficient labels in a single adaptation cycle; rather, existing concept drift adaptation methods remain limited under accumulated temporal drift across static, dynamic, graph-based, and fused representations.

Table 12: Performance of adaptation methods on the McNdroid dataset across IID, NEAR, and FAR settings using three modalities and feature fusion (budget = 200).

Method	Setting	Static	Dynamic	Graph-based	Feature Fusion
HCC Chen et al. (2023)	IID	93.67±0.36	74.94±0.28	87.94±1.63	91.61±0.33
	NEAR	86.19±1.52	67.10±3.65	66.95±1.55	79.41±0.81
	FAR	70.13±1.20	45.27±0.49	64.93±0.60	70.42±0.15
CADE Yang et al. (2021)	IID	90.11±0.21	74.15±0.29	81.97±1.07	83.92±2.06
	NEAR	52.17±5.90	31.92±1.39	28.66±1.80	42.16±1.16
	FAR	21.12±1.26	11.30±0.51	5.29±0.38	12.62±2.70
DAN–MMD Long et al. (2015)	IID	84.79±1.90	69.53±1.08	79.50±2.07	80.69±1.23
	NEAR	50.97±9.06	48.09±1.81	37.08±7.17	44.38±2.72
	FAR	20.96±1.56	17.15±0.57	14.47±4.95	15.08±1.14

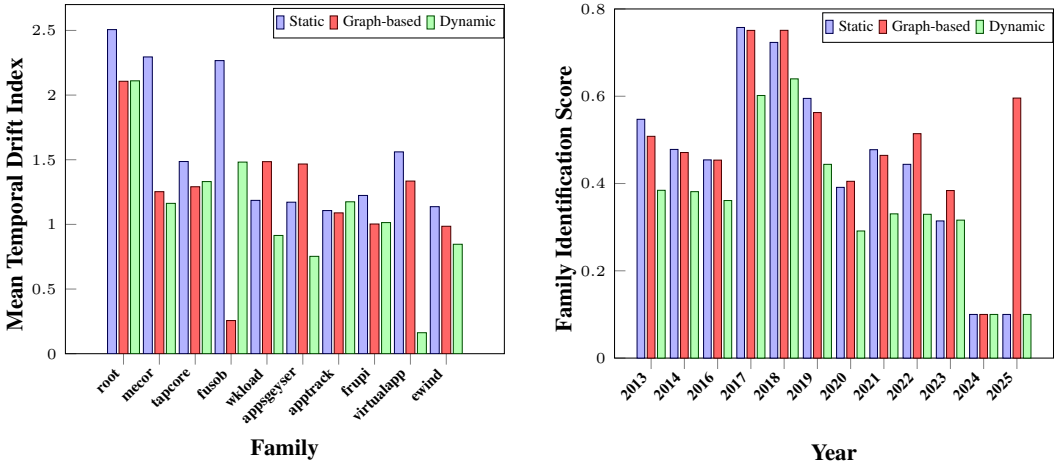


Figure 15: Comparison of modality-specific behavior across malware families and years. Left: mean temporal drift index by family. Right: family identification score by modality across years.

M Unsupervised Family Structure Analysis

We evaluate whether McNdroid representations preserve malware-family structure without supervision using HDBSCAN Campello et al. (2013) clustering across years and modalities. This analysis complements supervised detection by measuring whether static, dynamic, and graph-based features naturally form clusters aligned with malware-family labels.

Figure 15 summarizes modality-specific behavior at the family and year levels. The left panel shows that temporal drift varies substantially across malware families and modalities. Families such as `root`, `mecor`, and `fusob` exhibit high drift in static features, indicating large changes in manifest- and code-level artifacts over time. In contrast, dynamic and graph-based drift are lower for some families but higher for others, suggesting that malware evolution is not uniform across representations. This heterogeneity shows that family-level drift cannot be captured reliably by a single modality.

The right panel reports the family identification score across years. Static and graph-based representations generally provide stronger family-level structure than dynamic features, while dynamic features remain consistently lower, likely due to the sparsity and variability of sandbox-observed behavior. All modalities peak around 2017–2018, indicating stronger family separability in these years, but decline in later periods, especially after 2019. This trend suggests that family structure becomes less stable under long-term temporal shift. Together, these results show that McNdroid supports fine-grained analysis of both family-specific drift and modality-dependent family identification, complementing the supervised detection results.

Table 13: Average F1 scores (%) across years under domain-incremental continual learning for static, graph-based, dynamic, and feature-fusion representations.

(a) Static

Year	None	ER	EWC	SI	Joint
2013	41.8±0.2	40.1±0.3	41.9±4.2	41.9±4.2	42.0±0.1
2014	47.4±0.2	48.3±0.3	52.8±0.7	52.8±2.2	48.6±0.1
2016	67.3±0.1	69.8±0.2	67.6±1.3	61.1±0.4	69.5±0.1
2017	66.6±0.2	74.2±0.3	78.1±1.0	65.6±0.4	78.6±0.1
2018	69.6±0.1	73.7±0.3	73.5±0.1	68.7±0.7	80.5±0.1
2019	76.1±0.3	78.4±0.4	78.6±0.4	67.3±2.7	83.8±0.2
2020	75.9±0.1	78.2±0.3	78.6±0.3	70.6±0.4	84.8±0.1
2021	75.4±0.2	77.8±0.3	79.6±0.2	69.5±1.3	86.1±0.1
2022	72.0±0.2	74.3±0.3	76.5±0.3	71.8±1.4	86.8±0.1
2023	72.7±0.1	78.3±0.3	79.6±1.3	57.2±0.7	88.3±0.1
2024	50.9±0.3	76.3±0.4	76.9±0.9	53.2±0.5	89.3±0.1
2025	31.8±0.4	74.6±0.4	62.0±2.7	53.3±0.2	91.4±0.1

(b) Graph-based

Year	None	ER	EWC	SI	Joint
2013	27.0±0.2	26.7±0.3	26.6±2.8	26.6±2.8	27.6±0.1
2014	31.5±0.2	33.3±0.3	33.6±3.0	27.0±0.7	32.9±0.1
2016	59.8±0.2	58.2±0.3	59.3±2.7	61.7±2.0	68.2±0.1
2017	62.7±0.2	67.3±0.3	38.5±3.3	48.5±7.7	80.2±0.1
2018	62.3±0.2	73.7±0.3	71.9±2.3	68.6±1.2	80.9±0.1
2019	73.4±0.3	75.1±0.4	71.5±0.3	67.8±0.7	82.5±0.2
2020	69.9±0.2	72.7±0.3	65.9±4.1	67.8±0.7	83.5±0.1
2021	67.7±0.2	70.2±0.3	72.8±1.2	67.1±1.7	83.5±0.1
2022	67.8±0.2	70.1±0.3	67.5±1.9	62.6±0.4	84.9±0.1
2023	73.5±0.2	75.6±0.3	72.1±0.5	63.4±1.6	86.2±0.1
2024	12.0±0.4	73.5±0.4	0.9±1.2	0.0±0.0	87.8±0.1
2025	21.0±0.4	72.9±0.4	0.0±0.0	0.0±0.0	89.3±0.1

(c) Dynamic

Year	None	ER	EWC	SI	Joint
2013	37.9±0.2	37.8±0.3	40.0±2.6	40.0±2.6	39.5±0.1
2014	44.0±0.2	42.3±0.3	47.2±1.6	47.0±0.2	39.9±0.2
2016	59.5±0.2	57.2±0.3	59.7±0.4	49.6±0.4	58.1±0.2
2017	47.2±0.3	56.3±0.3	56.1±0.3	56.8±0.4	62.1±0.2
2018	44.8±0.3	55.1±0.3	54.4±0.6	54.5±0.5	54.3±0.2
2019	62.7±0.3	63.9±0.4	66.4±0.4	50.1±0.8	69.5±0.2
2020	61.5±0.2	62.2±0.3	64.6±0.2	50.9±1.7	70.3±0.1
2021	58.3±0.2	62.1±0.3	63.4±0.1	59.0±0.1	71.5±0.1
2022	59.4±0.2	63.5±0.3	60.9±0.1	57.3±0.2	72.1±0.1
2023	61.8±0.2	65.7±0.3	69.1±0.3	57.4±0.4	74.8±0.1
2024	42.2±0.4	64.6±0.4	61.8±1.5	54.1±0.4	74.9±0.1
2025	27.3±0.4	63.3±0.4	55.2±2.1	54.7±0.5	75.6±0.1

(d) Feature Fusion

Year	None	ER	EWC	SI	Joint
2013	44.8±0.2	44.7±0.3	36.7±5.8	36.7±5.8	43.2±0.1
2014	42.4±0.2	42.0±0.3	48.0±6.2	40.8±3.0	46.5±0.1
2016	65.4±0.1	65.5±0.2	64.7±1.6	59.8±3.2	65.8±0.1
2017	74.0±0.2	74.2±0.3	76.2±1.7	64.8±3.3	81.1±0.1
2018	68.9±0.2	72.8±0.3	76.9±0.9	59.9±15.4	79.4±0.1
2019	75.7±0.3	78.4±0.4	74.6±0.6	69.6±1.5	83.2±0.2
2020	74.0±0.1	76.7±0.3	74.8±1.8	67.8±5.0	84.1±0.1
2021	70.6±0.2	72.4±0.3	72.9±1.8	64.7±8.2	85.4±0.1
2022	69.4±0.2	73.2±0.3	67.5±3.8	64.7±2.3	85.9±0.1
2023	74.6±0.1	76.5±0.3	70.3±0.6	60.1±0.6	86.2±0.1
2024	54.4±0.3	77.6±0.4	35.4±18.2	0.0±0.0	87.5±0.1
2025	49.3±0.3	79.3±0.4	0.0±0.0	0.0±0.0	89.2±0.1

N CONTINUAL LEARNING ON MCNDROID

Continual learning (CL) studies how a model learns from a sequence of tasks while retaining knowledge from previous tasks van de Ven et al. (2024); Rahman et al. (2022). This is important for Android malware analysis because malware data is non-stationary: benign apps, malware behaviors, and malware families evolve over time Rahman et al. (2022); Park et al. (2025). Following standard CL definitions, we consider Domain Incremental Learning (Domain-IL) and Class Incremental Learning (Class-IL), which are two widely used CL scenarios in prior work.

We evaluate CL on our multimodal Android malware dataset using four feature settings: Static, Dynamic, Graph-based, and Feature Fusion. For Feature Fusion, samples are aligned by hash and features from all three modalities are concatenated. We compare five learning strategies: None/Naive, Joint, Experience Replay (ER), Elastic Weight Consolidation (EWC), and Synaptic Intelligence (SI). None sequentially trains the model using only the current task and therefore provides no explicit mechanism for retaining previous knowledge. Joint retrains the model using all training data observed up to the current task and serves as a cumulative-data reference. ER trains on the current task together with stored samples from previous tasks to mitigate catastrophic forgetting Rolnick et al. (2019). EWC constrains updates to parameters estimated to be important for previous tasks using a Fisher-information-based quadratic penalty Kirkpatrick et al. (2017). SI similarly regularizes parameter updates based on parameter importance accumulated during previous optimization trajectories Zenke et al. (2017). Unlike ER and Joint, EWC and SI do not require retaining previous training samples.

Domain-IL. In Domain-IL, each task corresponds to one calendar year from 2013 to 2025, excluding 2015. Each task contains both benign and malware samples, and the prediction target remains fixed as binary malware detection. Thus, the model must distinguish benign from malicious apps while adapting to temporal distribution shift.

For None/Naive, the model is trained sequentially on each year without access to data from previous years. Joint retrains the model using all training data observed from the first year up to the current

Table 14: Macro-F1 scores (%) across 11 tasks under class-incremental continual learning for static, graph-based, dynamic, and feature-fusion representations.

(a) Static

Task	None	ER	EWC	SI	Joint
1	67.1±0.6	67.1±0.8	68.0±0.2	68.0±0.2	67.1±0.5
2	31.3±0.8	59.3±1.0	40.5±1.8	63.5±0.5	68.7±0.5
3	9.0±0.5	53.6±1.0	8.6±5.3	51.4±2.7	66.4±0.6
4	1.3±0.3	49.7±1.1	0.3±0.2	43.1±0.5	64.5±0.6
5	0.0±0.1	45.9±1.1	0.1±0.0	37.2±1.1	63.5±0.6
6	0.0±0.1	41.6±1.2	0.0±0.1	26.2±2.4	62.5±0.7
7	0.0±0.1	39.7±1.2	0.2±0.3	30.3±1.0	61.7±0.7
8	0.0±0.1	36.3±1.2	0.2±0.3	17.1±1.8	60.4±0.7
9	0.0±0.1	34.0±1.2	0.1±0.2	27.3±1.0	59.0±0.8
10	0.0±0.1	32.0±1.3	0.0±0.0	19.4±2.3	59.8±0.8
11	0.0±0.1	31.2±1.3	0.0±0.0	22.7±1.4	59.3±0.8

(b) Graph-based

Task	None	ER	EWC	SI	Joint
1	17.3±0.7	17.3±0.9	18.3±0.5	18.3±0.5	17.3±0.6
2	0.2±0.2	15.4±1.0	0.2±0.1	1.2±0.1	15.1±0.7
3	0.0±0.1	12.9±1.1	0.1±0.1	0.1±0.0	16.6±0.7
4	0.0±0.1	10.0±1.0	0.0±0.0	0.0±0.0	16.1±0.8
5	0.0±0.1	10.2±1.0	0.0±0.0	0.0±0.0	14.0±0.8
6	0.0±0.1	6.6±0.9	0.0±0.0	0.0±0.0	12.3±0.8
7	0.0±0.1	5.9±0.9	0.0±0.0	0.0±0.0	11.2±0.8
8	0.0±0.1	4.8±0.8	0.0±0.0	0.0±0.0	10.9±0.8
9	0.0±0.1	3.6±0.8	0.0±0.0	0.0±0.0	9.3±0.7
10	0.0±0.1	2.0±0.7	0.0±0.0	0.1±0.0	9.8±0.7
11	0.0±0.1	3.6±0.8	0.0±0.0	0.0±0.0	11.0±0.8

(c) Dynamic

Task	None	ER	EWC	SI	Joint
1	51.7±0.6	51.7±0.8	51.6±0.3	51.6±0.3	51.7±0.5
2	8.3±0.5	45.2±0.9	23.4±1.4	49.5±0.5	52.7±0.6
3	1.0±0.2	41.7±1.0	0.7±0.4	25.6±3.3	49.0±0.6
4	2.1±0.3	39.5±1.0	0.1±0.1	21.3±1.4	48.1±0.7
5	0.9±0.2	36.4±1.1	0.3±0.2	24.6±2.0	48.1±0.7
6	0.7±0.2	34.6±1.1	0.1±0.0	8.3±0.7	44.5±0.7
7	1.0±0.2	31.7±1.1	0.1±0.1	15.3±2.3	45.3±0.8
8	0.4±0.2	29.0±1.1	0.0±0.0	10.4±1.8	44.7±0.8
9	0.1±0.1	26.9±1.1	0.0±0.0	6.7±1.3	43.0±0.8
10	0.0±0.1	25.7±1.2	0.1±0.0	9.4±2.5	43.2±0.8
11	0.0±0.1	25.2±1.2	0.1±0.1	7.0±2.5	43.3±0.8

(d) Feature Fusion

Task	None	ER	EWC	SI	Joint
1	33.8±0.7	32.7±0.9	32.9±0.8	32.9±0.8	35.1±0.5
2	2.1±0.3	30.5±1.1	2.0±0.2	9.7±2.1	29.5±0.7
3	0.1±0.1	27.1±1.1	0.8±0.4	0.4±0.1	29.4±0.7
4	0.0±0.1	22.7±1.2	0.0±0.0	0.0±0.0	29.2±0.7
5	0.0±0.1	19.7±1.2	0.0±0.0	0.0±0.0	28.3±0.8
6	0.0±0.1	13.7±1.1	0.0±0.0	0.0±0.0	22.5±0.8
7	0.0±0.1	10.6±1.0	0.0±0.0	0.0±0.0	22.1±0.8
8	0.0±0.1	13.7±1.1	0.0±0.0	0.0±0.0	20.0±0.8
9	0.0±0.1	8.9±0.9	0.0±0.0	0.0±0.0	20.6±0.8
10	0.0±0.1	9.3±0.9	0.1±0.1	0.6±0.3	18.6±0.8
11	0.0±0.1	8.8±0.9	0.0±0.0	0.0±0.0	16.4±0.8

year. ER trains on the current year together with replay samples from previous years. EWC and SI train sequentially without replay, using parameter-importance regularization to reduce changes to parameters important for previously learned tasks. We report task-wise performance for each modality and the fused multimodal representation.

Class-IL. In Class-IL, we focus on malware family classification. Following prior work Park et al. (2025), we select the 100 malware families with the largest numbers of samples and create a stratified 80/20 train/test split for each family. The task sequence is constructed such that Task 1 contains the first 50 malware families, and each subsequent task introduces 5 new families until all 100 families have been observed.

At each task, the model is trained according to the corresponding CL strategy and evaluated on the cumulative test set containing all malware families observed so far. None/Naive trains only on the current task, Joint trains on all observed tasks, ER trains on the current task together with replay samples from previous tasks, and EWC and SI sequentially update the model while regularizing parameters important to previously learned tasks. This setting evaluates whether the model can learn newly introduced malware families while preserving performance on previously seen families. As in Domain-IL, we evaluate static, dynamic, graph-based, and fused multimodal representations separately.

Results. Table 13 reports the F1 scores under the Domain-IL setting. Joint training generally provides the strongest performance, particularly in later years, reaching 91.4% for Static and 89.2% for Feature Fusion in 2025. Among continual-learning methods, ER and EWC provide the most consistent gains over None, although their relative performance varies across modalities and years. EWC is competitive with, and occasionally exceeds, ER in the earlier and intermediate years, particularly for Static and Dynamic features, but degrades sharply for Graph-based and Feature Fusion representations in 2024–2025. SI is generally weaker and exhibits greater instability, including near-zero performance for the Graph-based and Feature Fusion modalities in the final years. Overall, retaining access to previously observed data through Joint training remains substantially more effective under temporal domain shift, while regularization-based approaches such as EWC and SI provide less consistent

Table 15: Year-wise macro-F1 (%) of graph neural network models under temporal distribution shift.

Model	2013	2014	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025
GCN	79.8±0.2	**77.8±1.1**	62.2±1.1	**52.0±4.7**	**46.2±5.6**	**45.3±1.3**	36.1±0.7	39.6±0.4	39.1±0.8	50.4±0.6	44.7±1.2	43.3±1.4
GIN	**82.3±0.6**	76.2±1.3	61.4±1.6	49.5±1.0	41.9±0.8	44.5±0.8	36.1±0.3	**39.8±1.2**	38.3±0.4	52.7±1.3	47.7±1.8	47.1±2.2
GraphSAGE	76.8±0.3	75.5±0.9	**64.0±1.4**	49.9±3.4	41.6±1.7	43.7±2.4	**36.3±0.9**	38.2±0.4	**40.4±1.9**	**54.3±0.7**	**48.6±1.1**	**47.8±1.4**

protection than replay. Additionally, Table 14 reports the Macro-F1 scores under the Class-IL setting. Class-IL is substantially more challenging than Domain-IL: without an explicit retention mechanism, performance rapidly collapses toward 0%, indicating severe catastrophic forgetting. Joint training provides the strongest and most stable performance across tasks, while ER offers the most consistent mitigation among the continual-learning methods. SI can preserve substantial performance during the early tasks—for example, outperforming ER on Task 2 for Static and Dynamic representations—but its effectiveness decreases as additional classes are introduced. EWC provides limited protection in this setting and, similar to None, approaches near-zero Macro-F1 for most modalities after only a few tasks. These results indicate that class-incremental malware learning induces substantially stronger forgetting than domain-incremental learning, and that replay is more robust than parameter-regularization approaches under sustained class expansion.

O Graph Neural Network Baselines

Our original graph-based evaluation follows a feature-based design in which each function-call graph is summarized using structural statistics, sensitive-API indicators, and interaction ratios, producing a fixed-dimensional representation suitable for standard classifiers. This design is consistent with prior Android malware detectors such as MaMaDroid Mariconti et al. (2017), which derives Markov-chain features from API-call graphs, MalScan Wu et al. (2019), which summarizes function-call graphs using centrality-based features, and APIGraph Zhang et al. (2020), which incorporates graph-derived API semantics into conventional classifiers. Feature-based graph representations also provide practical advantages in scalability and inference cost, which remain important considerations for large-scale malware analysis Microsoft Defender Security Research Team (2018); Scott Coull and Christopher Gardner (2018); Zeitlin (2023).

To determine whether directly learning from graph topology improves robustness to temporal drift, we additionally evaluate three widely used graph neural network architectures: Graph Convolutional Network (GCN), Graph Isomorphism Network (GIN), and GraphSAGE, following the graph-learning setting used in MalNet-Graph Freitas et al. (2021). Unlike our feature-based graph models, these architectures operate directly on the caller–callee relationships encoded in the function-call graphs and learn application-level representations through message passing.

Table 15 reports the year-wise macro-F1 results. GIN achieves the strongest IID performance with 82.3% in 2013, but all three GNN architectures degrade substantially under temporal shift. GCN performs relatively better in several earlier years, whereas GraphSAGE achieves the strongest results in later years, particularly from 2022–2025. Nevertheless, the best macro-F1 falls below 50% by 2024–2025, indicating that directly learning from caller–callee graph structure does not eliminate long-term temporal degradation across the benchmark.

P Computational Resources

All dataset processing and experiments for McNdroid were conducted on a high-performance compute server with the following configuration:

- **CPU**: Dual-socket `Intel Xeon Gold 6430` with a total of 128 logical cores (64 physical cores, 2 threads per core).
- **Memory**: 512 GB RAM.
- **GPU**: 2× NVIDIA H100 NVL GPUs with 95.8 GB memory per GPU. Experiments were conducted under `CUDA 12.8` and driver version `595.71.05`.

This infrastructure enabled us to efficiently process nearly 900k million APKs, large-scale temporal benchmarking over 12 years of Android malware data.

Q SECURITY IMPACT AND RESPONSIBLE USE

McNdroid is designed to support research on robust Android malware detection under temporal distribution shift. By providing aligned static, dynamic, and graph-based representations over a 12-year period, the benchmark enables systematic study of model aging, modality-specific degradation, cross-modal robustness, malware-family evolution, and adaptation to changing threat distributions. These capabilities are relevant to practical settings such as mobile security, app-store vetting, endpoint protection, and adaptive threat monitoring.

The benchmark also improves reproducibility and accessibility for temporal malware research. Pre-extracted features, predefined temporal splits, and accompanying code allow researchers to evaluate detection and adaptation methods without reproducing large-scale APK processing, graph extraction, or sandbox execution. This reduces computational and infrastructure requirements while enabling more consistent comparison across methods.

McNdroid is intended for defensive security research and releases processed representations. As with other malware benchmarks, analyses of detector failures may reveal weaknesses that could potentially be misused; however, the released artifacts are designed primarily to support measurement, robustness analysis, and defensive model development. We therefore encourage use consistent with applicable data-use requirements and responsible security-research practices.