Authors' Response

We are deeply grateful to the Editor for handling our manuscript and allowing us to revise it. We would also like to thank all Reviewers for their thoughtful and constructive feedback sincerely. Their valuable comments have greatly helped us improve the clarity, completeness, and overall quality of our work.

In response to the reviewers' feedback, we have carefully revised the manuscript and made the necessary modifications to address all concerns. A detailed, point-by-point response is provided below. The reviewers' comments are shown in italic, and all significant revisions in the manuscript are highlighted in blue for clarity.

We sincerely appreciate the opportunity to enhance our manuscript and believe that the revisions have substantially strengthened its technical depth and presentation. We hope that the revised version meets the reviewers' expectations, and we remain happy to provide further clarifications if needed.

Response to Reviewer 1

Response R1.1 [Main Idea & Insight] *The problem to be solved in this paper is clearly explained in the introduction. However, after reading the introduction, the main idea of the proposed approach has not been explicitly explained. What is/are your insight(s) to tackle the problem? Why you think the idea would work?*

We thank the reviewer for appreciating the clarity of our problem definition. We have substantially revised the Introduction (Section I) to explicitly articulate the main idea, key insight, and the rationale behind our approach.

1. Our Key Insight: We observe that the fundamental limitation of ERM under natural distribution drift is not merely the presence of changing data distributions, but the loss of temporal structure during training. By collapsing samples collected at different time periods into a single static distribution, ERM obscures which correlations are stable over time and which are transient, encouraging shortcut features that are predictive on average but brittle under temporal evolution.

Our key insight is that time itself provides weak but consistent supervision: timestamps are not noise, but implicit signals that reveal how malware behavior evolves. Correlations that persist across time are more likely to reflect stable malicious semantics, while those that vary with time tend to encode transient implementation details. Making temporal variation explicit during training enables the learning process to distinguish these two factors and prioritize representations that remain stable despite malware evolution.

2. Why this idea works: The effectiveness of TIF comes from rethinking ERM-based training under natural distribution drift and aligning the training objective with real-world deployment scenarios.

First, TIF changes what information is exposed during training. Instead of collapsing samples from different time periods into a single static distribution as in ERM, we treat time as an explicit source of environmental variation. This exposes temporally unstable correlations that would otherwise be averaged out, allowing the model to distinguish stable malicious semantics from transient, time-specific artifacts at the representation level.

Second, TIF changes how the model is optimized for generalization. By partitioning historical data into temporal environments, we instantiate the principle of Invariant Risk Minimization (IRM) by explicitly penalizing representations whose predictive power varies across time. This aligns the optimization process with the deployment scenario, where future samples naturally come from unseen time periods, and encourages the model to rely on temporally invariant correlations rather than spurious historical shortcuts. Importantly, TIF is not a naive application of invariant learning: the multi-proxy contrastive module is crucial for producing high-quality representations in the presence of heterogeneous malware families, while the two-stage optimization and environment-specific gradient alignment mitigate known instability issues of IRM and tailor TIF to the malware detection setting. These design choices ensure that temporal invariance is not only a theoretical objective, but also a practically learnable property under realistic malware evolution.

Response R1.2 [Definition of Temporal Invariance] *what is the definition of "temporal invariant"? I only knew the meaning of this term until I read the examples in Fig.2. That is too late and also informal.*

We agree that formally defining temporal invariance early in the paper is crucial for understanding our contribution. We have made two major revisions to address this:

1. To ensure the concept is understood immediately, we have revised the Introduction (Section I, Paragraph 2) to provide an explicit conceptual definition. We clarify that temporal invariance refers to the persistence of malicious intent and functional semantics, even as implementation details (e.g., APIs, obfuscation) evolve over time.

2. To address the concern about informality, we have introduced the explanation of temporal invariance in Section III-B, including inter- and intra-invariance. We further operationalize this notion in our framework by defining a *Temporally Invariant Feature* as one that simultaneously satisfies Stability (consistency across temporal environments, Eq. 6) and Discriminability (separability between classes, Eq. 8), with the combined formulation given in Eq. 10.

Response R1.3 [Training Cost] *How is the cost of using this temporal invariant training framework? Have you reported the time taken to perform training?*

We have added an Overhead Analysis section (Section V-G) to explicitly analyze the training cost of TIF. Instead of reporting hardware-dependent wall-clock time, we provide a theoretical complexity analysis to ensure generality across different model architectures and implementations. Our analysis decomposes the training procedure into two stages and shows that the additional cost introduced by multi-proxy contrastive learning (MPC) and invariant gradient alignment (IGA) is strictly lower-order compared to the dominant encoder forward/backward computation. As a result, the overall training cost incurs only a constant-factor increase over standard ERM-based training. Importantly, this overhead is confined to offline training and does not affect inference-time efficiency.

Response R1.4 [Case Study] *That is good to see the proposed approach indeed improved the F1-score (and other metrics) over the baselines. Can you give some case studies to showcase the superiority of the proposed approach? Such case studies would allow the readers to understand why you can improve the F1-score and mitigate the problem you aim to solve.*

To provide concrete and interpretable evidence explaining why our method improves F1-score and robustness under temporal drift, we have added a dedicated Case Study in Section V-I.

Specifically, we analyze the top-10 important features in the interpretable Drebin feature space, comparing TIF with the ERM-based baseline DeepDrebin [1] under two representative drift scenarios: intra-family drift (Airpush_tr $\rightarrow$ Airpush_te) and inter-family drift (Airpush_tr $\rightarrow$ Hiddad_te, where Hiddad does not appear in training). These settings directly correspond to the malware evolution scenarios discussed in our Motivation (Section III-B).

Our analysis shows that TIF consistently assigns higher importance to stable, high-level intent and capability features that are less sensitive to version- or family-specific implementation changes. Under intra-family drift, TIF emphasizes system-wide broadcast events such as `USER_PRESENT` and `PACKAGE_ADDED`, which are commonly associated with application activation and background service triggering and remain stable across different variants of the same family. In contrast, DeepDrebin places greater weight on specific APIs and permissions that are more prone to temporal variation.

This difference becomes more pronounced under inter-family drift. When evaluated on the unseen Hiddad family, TIF assigns higher importance to general network-related capabilities (e.g., `ACCESS_NETWORK_STATE`) and high-level lifecycle management, whereas DeepDrebin relies on more restrictive, version-reliant features. As a result, the dominant signals used by TIF exhibit better transferability across families.

Overall, this case study provides an interpretable explanation of why TIF achieves improved F1-score under temporal drift: by favoring more stable intent prerequisites over brittle implementation-specific details, the model reduces its sensitivity to natural distribution shifts induced by malware evolution.

Response R1.5 [Threat to Validity] *Please clarify the potential threats to the validity of your experiment in a dedicated section.*

We have added a dedicated Threat to Validity section (Section VI) to discuss factors that may affect the interpretation of our results. In this section, we clarify potential threats to internal validity (e.g., model complexity and hyper-parameter sensitivity), construct validity (e.g., using temporal partitioning as an agnostic proxy for malware drift), and external validity (e.g., focusing on natural temporal drift rather than adversarial shifts). These discussions help delineate the scope of our conclusions and the assumptions underlying our evaluation.

Response R1.6 [Artifacts] *Did you provide the artifact for your work to allow open science?*

Response to Reviewer 2

Response R2.1 [Detailed Comparisons & Reasons behind Good Results] *The authors discussed different drift-robust malware detectors in the Background section. I would love to see more in-depth and detailed comparisons between TIF and these techniques especially the most similar ones (i.e., the first category). While the current discussions discussed the limitations of these techniques, it is not very clear how and why TIF can address these limitations. From a very high-level understanding, both TIF and some existing techniques focus on the invariant features across different time periods. It is unclear what novel design of TIF makes it outperform the existing techniques. In other words, I would like to suggest the authors to add more detailed introductions to the most similar techniques, connect their limitations with the challenges discussed in the motivating example if possible, and add intuitions on how TIF can address these limitations.*

We have substantially revised the manuscript to clarify both the conceptual differences between TIF and the most related drift-robust techniques, and the reasons why TIF achieves superior robustness under realistic malware evolution.

From a theoretical perspective, existing static drift-robust methods address drift by attributing performance degradation to specific assumed causes or manifestations. As we discussed in Section II-B, APIGraph [2] assumes drift is primarily driven by API raplacement and stabilizes the feature space accordingly; T-stability [3] and related training-dynamics-based methods [4] assume that unstable features can be identified through monotonic temporal trends or learning-speed heuristics; family-aware methods [5] focus on the emergence of unseen families as the dominant source of drift. While effective under the particular scenarios they model, these approaches inherently rely on restrictive assumptions about how drift is reflected in the feature space. In practice, malware evolution is driven by multiple, interacting factors, and the same high-level behavior may persist even when APIs remain unchanged, but implementation strategies evolve. Under such conditions, assumptions tied to a single drift source may no longer hold and thus fail to generalize. In contrast, TIF does not assume any specific cause of drift. Instead, it takes a manifestation-oriented perspective: if malware evolution induces distribution shifts over time, then time itself provides weak but consistent supervision for exposing unstable correlations. TIF explicitly treats temporal partitions as environments and reformulates training to distinguish correlations that persist across time from those that vary with time, rather than predefining what should be stable. By breaking the ERM paradigm that collapses all temporal data into a single distribution, TIF enables the learning process to surface and suppress temporally unstable shortcuts while preserving invariant semantics. This design choice fundamentally differentiates TIF from prior approaches that either filter features based on assumed drift patterns or rely on post-hoc updates to compensate for drift.

Beyond this conceptual distinction, we have strengthened the empirical explanation through a case study (Section V-I). TIF consistently emphasizes features aligned with malicious intent and behavioral prerequisites (e.g., lifecycle events,

network capability, persistent background execution), rather than implementation-specific APIs or permissions that are brittle under evolution. In contrast, ERM-based and feature-constrained baselines increasingly rely on volatile, version-dependent signals. The case study provides concrete evidence that TIFś training objective leads to representations that are both more stable and more semantically meaningful under temporal drift.

Taken together, the novelty of TIF lies in changing the training paradigm itself: it leverages temporal structure to learn invariance without prespecifying drift causes and without relying on frequent retraining. This principled shift explains why TIF consistently outperforms existing drift-robust methods across diverse drift scenarios.

Response R2.2 [Baseline Selection] *How are the baselines selected? There are different existing techniques that aim to address the problem of evolving malware families. How did the authors come to these 3.*

We thank the reviewer for this question. Our baseline selection follows a principled criterion: we compare against methods that address malware distribution drift under the same task definition and evaluation protocol, and whose robustness mechanisms align with the problem formulation discussed in our Introduction (Section I) and Background (Section II-B).

Both static and continual learning baselines are included. For the static ones, we focus on approaches that enhance detector robustness under a static, offline training setting, without relying on deployment-time updates or auxiliary supervision. Within this scope, we select three representative baselines that correspond to complementary and widely adopted perspectives on handling malware drift. Rather than differing only by implementation details, these methods reflect distinct mainstream design philosophies: (i) APIGraph [2] represents approaches that explicitly assume the cause of drift, attributing performance degradation primarily to API-level implementation changes and addressing drift via feature selection; (ii) T-stability [3] represents approaches that characterize drift through its observable temporal heuristics, constraining features whose discriminative power varies across time; (iii) Guided Retraining [6] represents a general, model- and feature-agnostic representation enhancement strategy that improves robustness without making explicit assumptions about drift sources. Together, these methods span a spectrum from explicit assumptions about drift causes, to empirical observations of drift effects, to assumption-free representation enhancement. This categorization mirrors the landscape of existing solutions discussed in our Background section and enables a systematic comparison with our approach from multiple complementary angles.

In addition, we include HCC [7] as a continual learning-based baseline. Unlike the static methods above, HCC mitigates drift by periodically updating the model with newly observed samples during deployment. We include this baseline not as a direct competitor in the static setting, but to evaluate whether our framework can also serve as a robustness enhancement when integrated into a dynamic, deployment-time update pipeline, thereby assessing its effectiveness in both static and continual scenarios.

We do not include DOMR [5] because it relies on episodic meta-learning with one-vs-rest recognition heads tailored for identifying new malware families, which is incompatible with our setting involving hundreds of families and both inter- and intra-family drift. SCRR [4] is excluded due to the unavailability of its source code and its highly sensitive bi-level optimization procedure, making faithful reproduction without an official implementation impractical.

Response R2.3 [Continual/Incremental Learning Comperison] *As far as I understand, the authors did not compare with incremental training techniques as baselines. The authors may include a representative technique as baseline or give good reasons on why they chose not to include such a baseline.*

We clarify that our evaluation explicitly includes a representative continual learning baseline and a dedicated research question to assess performance under incremental model updates.

Specifically, we include HCC [7], a state-of-the-art continual learning framework for Android malware detection, as our representative baseline. HCC addresses distribution drift by periodically detecting drifted samples and updating the model during deployment, and is widely adopted in this domain. In the context of malware detection, incremental training is commonly realized as a form of continual learning, where models are periodically updated with newly observed samples to handle distribution drift. HCC follows this paradigm and therefore serves as a representative incremental/continual learning baseline.

Moreover, we design a dedicated research question (RQ4) (Section V-E) to systematically evaluate the interaction between TIF and continual learning. In this evaluation, we consider two complementary settings based on HCC: *(i) Without the continual update mechanism*, where model updates are disabled to isolate and directly compare representation quality. Under this setting, TIF achieves an average F1 improvement of 49.16% on drift samples compared to HCC, indicating substantially more stable and discriminative representations. *(ii) With the continual learning framework enabled*, where we follow HCC's update strategy to evaluate the full detection pipeline under realistic drift conditions. In this setting, TIF reduces the number of required model updates from 65 to 38, cutting labeling cost by 41.5% while maintaining comparable detection performance.

This separation allows us to disentangle improvements arising from stronger representations from those due to continual adaptation. Our results show that TIF learns more stable representations that delay performance degradation, thereby reducing update frequency and labeling cost when integrated into a continual learning framework, while maintaining detection performance.

Response R2.4 [Precision & Recall] *For the evaluation, the authors only presented F1-score and AUT. While I understand that F1-score is a combination of Precision and Recall. Can the authors still add the precision and recall values to provide a more comprehensive picture of the performance of TIF?*

We thank the reviewer for this suggestion. Malware detection is a highly imbalanced classification problem in both training and testing phases (malware: benign = 1:10), under which precision and recall are highly sensitive to class imbalance and threshold choice. Reporting either metric in isolation can therefore be misleading, especially under temporal drift, where score distributions shift over time. We adopt F1-score as the primary metric because it provides a balanced summary of precision and recall that is more robust to class imbalance. Based on this consideration, AUT(F1) is further used to characterize long-term performance stability, which would be less meaningful if applied to precision or recall alone due to their high variance in this setting. In addition, to demonstrate that our results reflect consistent trends rather than single-run variability, we have additionally reported the standard deviation across multiple runs in Table II. The low variance observed across runs indicates that the performance gains of TIF are stable and reproducible, providing further evidence of its effectiveness beyond point estimates of evaluation metrics.

Response R2.5 [Performance Trend of T-stability] *In table 1, interestingly, for Drebin with T-stability, the increases are higher in later years. The gaps between the improvement of T-stability and TIF is narrowing. Could the authors add some explanations on why this happened?*

We thank the reviewer for this insightful observation. The narrowing gap can be attributed to how different methods handle the trade-off between discriminability and temporal stability. In the early stages, TIF holds a clear advantage because it jointly optimizes for stability and discriminability in the representation space, allowing it to preserve features that are both predictive and consistent across time. In contrast, T-stability applies conservative, feature-level constraints that suppress temporally unstable features, even when they are highly discriminative in the early phase, leading to lower initial performance.

As time progresses, features that lack temporal robustness naturally lose their predictive value under time-aware evaluation. Since both T-stability and TIF emphasize temporally stable patterns, the effective feature sets used by the two methods become more similar over time, which explains the observed narrowing of the performance gap.

Despite this convergence, TIF consistently outperforms T-stability. T-stability operates purely at the feature level with linear constraints and follows a suppressive strategy, limiting its ability to model complex feature interactions. In contrast, TIF performs invariant learning in a non-linear representation space and explicitly optimizes both stability and discriminability, enabling it to retain more expressive and robust semantics throughout the evaluation period.

We note that our case study (Section V-I) analyzes feature importance in the Drebin space purely for interpretability. The observed feature-level behaviors are manifestations of the underlying representation learned by TIF, which is optimized in the representation space rather than through explicit feature-level constraints.

Response R2.6 [Explanation of Results in RQ2] *The evaluation setup is a bit confusing for RQ2. How are the values calculated in Figure 6? I'm not very sure about this. This makes it hard to interpret the results. Would the authors add some explanations on the experiment set up for this RQ?*

We have revised the manuscript to clarify both the experimental design and the calculation of Figure 6, making the evaluation procedure more explicit. First, we clarify the design motivation of RQ2, which is to examine whether the proposed framework can learn invariant representations under both intra-family and inter-family drift, corresponding to the two types of temporal invariance discussed in Section III-B. To this end, we explicitly define two evaluation settings in Section V-A1: 1) Closed-world, where test samples contain only malware families observed during training to evaluate intra-family invariance; and 2) Open-world, where test samples contain only malware families absent from the training set to evaluate inter-family invariance. To ensure fair comparison under highly imbalanced family distributions, test samples in each setting are sorted chronologically and partitioned into 10 equal-sized temporal segments, which are used consistently across RQ2.

Second, we recreated Figure 6 to make it cleaner, measuring representation stability. For each temporal test segment, we extract the hidden representations of malware samples from the encoder and compute their mean embedding. We then compute the cosine similarity between this mean test embedding and the mean malware embedding of the training set. This procedure is applied consistently under both closed-world and open-world settings. The results in Figure 6 therefore quantify how closely the representations learned at test time remain aligned with those learned during training. Higher similarity indicates more stable representations under temporal distribution drift. We have incorporated these explanations into Section V-C to improve clarity and interpretability.

Response R2.7 [RQ3 and Corresponding Figures] *The formatting for RQ3 and the figures may be revised.*

We have revised the presentation of RQ3 to improve clarity and interpretability. First, we clarify the evaluation protocol by defining the closed-world and open-world settings in the Dataset section (Section V-A1) and explaining that the indices in Table IV correspond to successive temporal test segments with equal sample sizes. Second, we refine the explanation of the Feature Contribution Score (FCS). We clarify that FCS measures the extent to which a model assigns importance to features that are discriminative in the current evaluation set, and that temporal invariance is inferred from the persistence of high FCS values across successive time segments rather than from the absolute value at a single time point. These revisions align the definition of FCS, the experimental protocol, and the presentation of Table IV, thereby improving the clarity of RQ3.

Response R2.8 [Application Statistic] *The authors may also add the number of apps for each month or each year when conducting an evaluation regarding the changes over time.*

To provide a clearer picture of dataset evolution over time, we add a new table (Table V-A1) that summarizes, for every evaluation year, the numbers of benign and malicious samples, as well as the number of seen and unseen malware families

appearing in the test set to make the temporal dynamics of the dataset explicit.

Response to Reviewer 3

Response R3.1 [Baseline Selection] *It is unclear why the authors selected the three techniques as representatives. Why did the authors not select the latest technique of each kind? I would like to see more explanations*

We thank the reviewer for this comment. Our baseline selection is guided by alignment with our problem setting and by representativeness of the mainstream robustness paradigms discussed in the Introduction (Section I) and Background (Section II-B), rather than by recency alone. As clarified in Response R2.2, our goal is to compare against methods that explicitly target robustness to evolving malware distributions *under the same static, offline training setting*, without relying on deployment-time updates or auxiliary supervision.

All selected baselines operate under the same static, offline training setting as our method, making them directly comparable. More importantly, they represent complementary and widely adopted perspectives on handling malware drift, more detailed information is also discussed in Response R2.2: (i) APIGraph [2] represents approaches that explicitly assume the cause of drift (e.g., API-level changes); (ii) T-stability [3] represents approaches that characterize drift through its observable temporal behavior of features; (iii) Guided Retraining [6] represents a general model and feature agnostic representation enhancement strategy that does not rely on any explicit assumption about drift causes.

In addition, we include a continual learning baseline, HCC [7], to evaluate whether our framework can also serve as a robustness enhancement when combined with deployment-time model updates. Together, these baselines are consistent with the categories of existing solutions outlined in our Introduction and Background, and enable a multi-angle evaluation of effectiveness across different mainstream design philosophies. Moreover, these methods represent the most recent reproducible works that satisfy the comparison criteria of TIF.

Response R3.2 [Dataset Information] *The authors may also add the number of apps for each month or each year when conducting an evaluation regarding the changes over time.*

Following this comment, we augment the dataset description with detailed temporal statistics. In particular, we include Table I that reports the yearly number of benign and malicious applications used in our evaluation, together with the number of visible (seen during training) and previously unseen malware families in each test year. This revision provides a transparent and fine-grained view of how the dataset evolves over time and how the evaluation settings relate to realistic deployment scenarios under temporal drift.

Response R3.3 [Latest Malware] *In the experiments, the malware samples are all before 2023. The latest malware samples in 2024 and 2025 are not included for experiments. I would suggest that the authors extend their experiments to include the latest samples.*

In response, we have extended our dataset to include malware samples collected up to June 2025. We note that, after mid-2025, the number of newly available malicious samples on AndroZoo becomes extremely limited (on average fewer than 15 malicious samples per month), which is insufficient to form a representative evaluation set and may introduce statistical bias. Therefore, we include all available samples up to June 2025 and clearly justify this cutoff in the revised manuscript. Accordingly, we update Table II, extend the experimental results to cover these newer samples, and revise the dataset description (Section V-A1) to explain the data selection and filtering process explicitly. These changes ensure that our evaluation reflects the most recent malware evolution while maintaining statistical reliability.

Response R3.4 [Statistical Analysis] *The experiments were repeated only three times (Section V), which is a relatively limited for evaluating model stability. Moreover, the lack of statistical analysis (e.g., variance, confidence intervals, or significance testing) reduces confidence in the reported improvements. Including such analyses would strengthen the experimental results.*

Following the suggestion, we further strengthen the analysis by explicitly reporting variability. Specifically, we add the standard deviation across three random seeds for TIF in the main result table (Table II). The results show consistently low variance across different runs, demonstrating that the observed performance gains are stable and reproducible rather than artifacts of a particular initialization.

Response R3.5 [Interpretation of Experimental Results] *Some experimental results lack sufficient interpretation. For example, in Table V, the best performance for years 2020–2022 is achieved by the MPC1+IGA combination, yet this finding is not discussed in depth. Similarly, in Table VI, the best performance for years 2019–2023 does not correspond to monthly segmentation, which appears to contradict the interpretation in Section V-F. Providing more detailed explanations for these inconsistencies would improve the clarity and credibility of the empirical analysis. Besides, the current analyses are mostly based on metric values. I would suggest that the authors perform a deeper analysis to find out some temporal invariants TIF learned when combined with each kind of detector for discussion.*

We revised explanations on these Tables and the corresponding results:

Interpretation of Table V: Since our samples are collected from real-world Android platforms, the dataset naturally reflects the evolution of malware development techniques. Starting from around 2020, Android malware increasingly adopts aggressive packing and obfuscation techniques, a widely observed trend in the mobile security ecosystem. Such samples weaken the reliability and consistency of static feature signals, making discriminative patterns noisier and less aligned with those observed during training [8]. As a result, the cross-environment consistency signal required for invariant learning becomes harder to estimate.

Under this regime, stronger invariance-inducing regularization introduced by MPC2 may over-constrain the representation space and suppress residual discriminative structures that remain useful, leading to diminished or unstable gains. In

contrast, MPC1 combined with IGA provides a more moderate regularization strength: MPC1 encourages partial sharing across environments without forcing strict alignment, while IGA suppresses environment-specific shortcuts. This balance makes MPC1+IGA more effective during the 2020–2022 period, where discriminative signals still exist but are increasingly noisy. The marginal increase observed in 2023 (from 0.662 to 0.665) is small and falls within expected year-to-year fluctuations. It does not indicate a systematic recovery of strong alignment benefits, but rather reflects the heightened sensitivity of heavily regularized models to year-specific drift characteristics under long-term evolution. Importantly, this observation does not imply a fundamental limitation of TIF. As discussed in Section V-E, we further demonstrate that TIF can be integrated into a continual learning framework to enable knowledge updating and progressively improved representations under evolving distributions although there are packed or obfuscated samples.

Interpretation of Table VI: The apparent inconsistency between the monthly result and quarterly result arises from the interaction between environment granularity and the temporal scale of distribution drift. Monthly segmentation provides finer-grained environments and is effective when distribution shifts are relatively mild, particularly in earlier test periods. In such cases, frequent temporal partitions help expose subtle variations and enable the model to learn invariances that are robust to short-term evolution. This explains why monthly segmentation can achieve strong performance in early stages. However, monthly partitions induce only small distributional contrasts, which may be insufficient to expose unstable correlations that emerge over longer time horizons. In contrast, quarterly segmentation introduces stronger temporal contrasts while maintaining sufficient samples per environment, enabling the model to better capture invariances that generalize across larger temporal gaps. Therefore, the superior performance of quarterly segmentation in later years reflects a trade-off between granularity and representativeness, and is consistent with the role of environment design discussed in Section, and we have revised this section for clearer illustration.

In addition, we have added a case study (Section V-I) to provide a qualitative analysis of the temporal invariants learned by TIF beyond metric-based evaluation. Using the interpretable Drebin [9] feature space and attribution analysis, we examine how TIF reshapes feature importance under both intra-family and inter-family temporal drift. The case study shows that TIF consistently emphasizes intent- and capability-level features (e.g., lifecycle events and general network access) that are intrinsic to malicious behavior and remain stable across time and families, while suppressing implementation-specific APIs and permissions that are brittle under evolution. These observations offer concrete evidence that TIF learns semantically meaningful temporal invariants rather than relying on transient implementation shortcuts, explaining its superior robustness under temporal drift.

Response R3.6 [Deeper Analysis on Temporal Invariants Learned by TIF] *Give a deeper analysis to find out some temporal invariants TIF learned when combined with each kind of detector for discussion.*

We have addressed it by adding a dedicated Case Study section (Section V-I) that provides a detailed interpretable analysis of the temporal invariants learned by TIF. In this case study, we analyze feature attributions in the interpretable Drebin feature space and examine how TIF reshapes feature importance under both intra-family and inter-family temporal drift, in comparison with an ERM-based detector. The analysis reveals that, when combined with standard malware detectors, TIF consistently emphasizes stable, intent- and capability-level features that persist across time and families, while suppressing volatile, implementation-specific APIs. This analysis provides concrete evidence that TIF learns semantically meaningful temporal invariants rather than relying on transient shortcuts, thereby explaining its superior robustness under temporal distribution drift.

Response R3.6 [Overhead] *What is the overhead of the approach?*

We have clarified the overhead of the proposed approach in Section V-G. Specifically, we present a detailed per-iteration complexity analysis of both training stages and show that the overall cost is dominated by the standard ERM encoder training, with the additional components contributing only a small constant-factor overhead. This conclusion follows from the fact that the introduced operations scale linearly with batch size and representation dimension, while the encoder forward/backward cost scales quadratically. Consequently, the proposed framework does not introduce asymptotic overhead, does not grow with deployment time, and incurs no inference-time cost. In addition, we utilize TIF in the continual learning setting, the result shows that purposed model decrease update cost by 41.5%.

TIF: Learning Temporal Invariance in Android Malware Detectors

Xinran Zheng*, Shuo Yang†, Edith C.-H. Ngai†, Suman Jana‡ and Lorenzo Cavallaro*

*University College London, London, UK
†The University of Hong Kong, Hong Kong SAR, China
‡Columbia University, New York, USA

***Abstract*—Learning-based Android malware detectors degrade over time due to natural distribution drift caused by malware variants and new families. This paper systematically investigates the challenges classifiers trained with empirical risk minimization (ERM) face against such distribution shifts and attributes their shortcomings to their inability to learn *stable* discriminative features. Invariant learning theory offers a promising solution by encouraging models to generate stable representations across environments that expose the instability of the training set. However, the lack of prior environment labels, the diversity of drift factors, and low-quality representations caused by diverse families make this task challenging. To address these issues, we propose TIF, the first temporal invariant training framework for malware detection, which aims to enhance the ability of detectors to learn stable representations across time. TIF organizes environments based on application observation dates to reveal temporal drift, integrating specialized multi-proxy contrastive learning and invariant gradient alignment to generate and align environments with high-quality, stable representations. TIF can be seamlessly integrated into any learning-based detector. Experiments on a decade-long dataset show that TIF excels, particularly in early deployment stages, addressing real-world needs and outperforming state-of-the-art methods.**

***Index Terms*—Malware Detection, Concept Drift, Invariant Risk Minimization**

I. Introduction

In open and dynamic environments, even the most effective malware detectors encounter significant challenges due to natural distribution drift[1], leading to performance degradation [10], [11]. This degradation arises from the continuous evolution of malware behavior and the emergence of new families that detectors have not previously encountered [12]. These new variants alter the underlying statistical properties of test samples [10], [13]–[15], thereby weakening detectors that rely on features derived from past training data.

To understand the rapid degradation of detectors under natural distribution drift, we revisit the standard training paradigm and attribute this vulnerability primarily to the failure to capture *temporal invariant semantics*. We observe that semantic invariance exists both within and across malware families; while implementation details (e.g., APIs or control flows) may change, the underlying malicious intent remains relatively stable. The failure to capture such semantics stems from the inherent limitation of Empirical Risk Minimization (ERM), the dominant machine learning training paradigm. ERM implicitly assumes all training samples are drawn from a single stationary distribution. By flattening malware instances collected at different times into a unified optimization objective, ERM treats all correlations homogeneously. This encourages the model to indiscriminately latch onto statistical shortcuts that are predictive on average but unstable over time. Consequently, the learned detector overfits to transient patterns, disregarding the invariant semantics implicitly present in historical data.

Recent research has focused on diagnosing the root causes of drift, yet these approaches are inherently limited. Some methods explicitly attribute the dynamic nature of malware to specific causes, such as unseen families [5], [11] or API renaming [2]. Others leverage observational heuristics of data dynamics, such as the distinct learning pacing of robust versus unstable correlations [4] or the monotonic evolution of feature importance over time [3]. However, natural drift is driven by multifaceted factors, where features often exhibit complex, non-linear dependencies. Simply targeting isolated drift sources yields only partial alleviation and often fails to generalize beyond specific target settings. Alternatively, Continual Learning (CL) addresses drift by continuously updating models with new data, without explicitly reasoning about the underlying causes [7], [16]. Yet, maintaining such systems is costly due to the constant need for expert annotation or the risk of noise from pseudo-labeling [17]. Moreover, performance degradation between updates undermines model robustness, compromising the reliability of continuous deployment. Taken together, these limitations underscore a central research question: whether invariant malicious semantics already embedded in historical training data can be explicitly exploited to address the limitations of ERM under natural distribution drift, without relying on drift factor modeling or frequent model updates.

Invariant learning theory [18] aims to address the shortcomings of ERM by learning invariant features/representations shared across different distributions, which aligns with our objective. It promotes the discovery of stable representations by dividing training data into distinct subsets or "environments" and encourages the model to minimize differences between them to capture stable elements across environments. This is based on two premises: a priori environment labels that

[1]For clarity, we use the terms drift, shift, concept drift/shift, natural drift/shift, and distribution drift/shift interchangeably throughout the text. This is in contrast to adversarial drift or adversarial evasive attacks, which are caused by identifying optimal input perturbations specifically crafted to cause a model's misclassification.

reveal instability [19], [20] and high-quality representations that adequately encode feature information [21]. These assumptions are not trivial for malware detection, as malware evolution is attributed to a variety of non-obvious factors, and the extremely unbalanced sample distributions and complex feature spaces due to multiple malicious families increase the complexity of learning high-quality representations, making invariance across environments difficult to explore.

In this paper, we present a **T**emporal **I**nvariant Training **F**ramework (TIF) to explicitly promote invariant representation learning under natural distribution drift. TIF first partitions training data temporally based on application observation dates, treating different time periods as distinct environments without relying on predefined environment labels. To support invariant learning in binary malware detection with heterogeneous and multi-family malware samples, we introduce a multi-proxy contrastive learning module that facilitates modular and expressive representation learning within each class. Moreover, TIF adopts a tailored invariant optimization strategy that encourages consistent optimization behavior for samples of the same class across temporal environments, thereby enhancing temporally invariant representations. Our framework is orthogonal to existing robust malware detectors: it requires no changes to feature spaces or model architectures and can be seamlessly integrated as an enhanced training paradigm for learning-based detectors. The main contributions of this paper are as follows:

- We formalize malware evolution under natural distribution drift from an invariant learning perspective, identifying the learning of discriminative and stable invariant representations as a key challenge for robust malware detection (Section III).
- We design a multi-proxy contrastive learning (Section IV-D) module to model heterogeneous multi-family malware distributions, together with an invariant gradient alignment (Section IV-E) module to enforce consistent optimization behavior across temporal environments, thereby promoting high-quality and stable invariant representations.
- We present TIF, a temporal invariant training framework for malware detectors that can be integrated with arbitrary learning-based detectors and feature spaces, enabling robust invariant representation learning over time by exposing and suppressing unstable information induced by drift (Section IV-F).
- We construct a 10-year dataset[2] and a series of experiments to evaluate the robustness of the TIF across various drift scenarios and feature spaces. The results show that TIF effectively slows performance degradation and learns invariant representations that consistently outperform state-of-the-art methods under diverse drift scenarios (Section V).

[2]We will open source the dataset metadata and code repository to foster reproducibility studies.

II. Background

In this section, we review the development of malware detectors robust to drift and the key components of invariant learning, laying the groundwork for proposing a temporal invariant representation learning solution tailored for Android malware detection.

A. Natural Drift of Malware

Natural drift refers to temporal changes in data distributions, often caused by malware evolution or emerging trends. Such drift violates the i.i.d. assumption and can degrade the performance of malware detectors by shifting decision boundaries over time. In Android malware, two primary sources of drift are commonly identified: 1) Technological evolution of malware: Even within the same family, malware may evolve due to system updates, framework changes, or advanced obfuscation techniques [22]. These modifications alter the feature distribution, causing models trained on earlier data to become outdated [23]. 2) Emergence of unknown malware families: Most learning-based detectors are trained on a fixed set of classes. However, new malware families may appear during deployment, exhibiting patterns unseen during training [24]. This leads to unreliable predictions and misclassifications.

B. Drift-robust Malware Detectors

Existing approaches to enhancing model robustness against drift can be broadly grouped into two categories. The first line of work improves robustness by explicitly modeling or attributing drift to specific causes. Representative examples include methods that assume drift primarily arises from API renaming or updating [2], the emergence of unseen malware families [5], or can be revealed through training dynamics [3], [4]. Specifically, for approaches based on training dynamics, Yang *et al.* [4] observe that spurious correlations tend to be learned faster during optimization and can thus be identified via learning speed disparities; Angioni *et al.* [3] assume that performance degradation is caused by a monotonic change in feature distributions over time, which is reflected as a decay in decision scores, and suppresses features whose importance varies significantly across temporal intervals; this assumption only applies to linear classifiers. By targeting a particular factor or proxy of drift, these approaches attempt to identify and filter unstable features accordingly. However, real-world malware evolution is driven by complex, multi-factor interactions that are difficult to isolate [25]. Heuristics based on a single assumed cause or manifestation often fail to capture the full spectrum of instability, leading to incorrect identification of stable semantics when multiple drift sources coexist.

The other category addresses drift by continuously incorporating new knowledge through incremental or continual learning [7], [16], where detectors are periodically updated by detecting drifted samples during deployment and retraining with newly labeled data. Although effective in extending model lifetime, this paradigm introduces a trade-off between performance and cost, as frequent updates require sustained

annotation effort and computational resources. Moreover, continual learning adapts models to evolving distributions without fundamentally addressing why the learned representations themselves fail to remain stable over time. The performance of detectors still declines over time, thereby weakening the robustness of updates, especially in long-term deployment settings.

In summary, prior drift-robust malware detectors either rely on rigid assumptions about the causes of drift, or compensate for drift by repeatedly updating models with new data. Given the inevitability and complexity of natural malware evolution, predefining what should remain stable is inherently challenging. This motivates a different perspective that designs learning objectives that explicitly encourage the extraction of drift-robust representations without assumptions about temporal drift causes and any relying on frequent retraining.

C. Invariant Learning

Assume that the training data $\mathcal{D}_{tr}$ is collected from multiple environments $e \in \mathcal{E}$, i.e., $\mathcal{D}_{tr} = \{D^e_{tr}\}_{e\in\mathcal{E}}$. Let the input space be $x \in \mathcal{X}$ and the target space be $y \in \mathcal{Y}$, and for the sample observations from each environment denoted $x, y \sim p(x, y|e)$, the samples within the environments obey an independent and identical distribution. Suppose a classification model f, denoted as a composite function $f = c \circ \phi$. where $\phi : \mathcal{X} \to \mathcal{H}$ denotes a feature encoder that maps the input samples into a feature representation space $\mathcal{H}$, and $\phi(x) \in \mathcal{H}$ is the "representation" of sample x. The $c : \mathcal{H} \to \mathcal{Y}$ denotes a classifier that maps the feature representation to the logits space of $\mathcal{Y}$.

1) Learning Invariant Representation: In test environments where distribution drift exists, the test data $\mathcal{D}_{te}$ may come from a distribution $p(x, y|e_{te})$ that does not appear in the training set, i.e. $e_{te} \notin \mathcal{E}$. Robustness to drift yields lower error rates on unknown test data distributions.

Invariant learning enhances the generalization ability of a model to unknown distributions by learning label distributions that are invariant across training environments. Its goal is to develop a classifier $c(\cdot)$ that satisfies the environmental invariance constraint (EIC) [26]:

$$\mathbb{E}\left[y \mid \phi(x), e\right] = \mathbb{E}\left[y \mid \phi(x), e'\right], \quad \forall e, e' \in \mathcal{E}, \tag{1}$$

where e and e' denote different environments to which the samples belong. This constraint is integrated into the training objective through a penalty term. Thus, the goal can be formalized as:

$$\min_{f} \sum_{e\in\mathcal{E}} R^e_{erm}(f) + \lambda \cdot \text{penalty}\left(\{S^e(f)\}_{e\in\mathcal{E}}\right), \tag{2}$$

where $R^e_{erm}(f) = \mathbb{E}_{p(x,y|e)}[\ell(f(x), y)]$ represents the expected loss on environment e. Empirical risk minimization (ERM) is to minimize this expected loss within each environment. $S^e(f)$ is some statistic of the model in e (see next), and the penalty is to constrain the change in this statistic to control the degree of deviation from the EIC. Optimizing this objective prevents mapping all x to the same value to satisfy environmental invariance, as it encourages the predictive utility of ϕ by minimizing empirical risk loss. In addition, the form of the penalty term is variable to achieve constraints for different objectives. Krueger et al. [27] proposed V-Rex such that $S^e(f) = R^e_{erm}(f)$, to minimize the variance of $S^e(f)$ in different environments. In CLOvE [28], the penalty is defined as the sum of the calibration errors of the model in each environment. One widely used scheme is Invariant Risk Minimization (IRM) and its practical variant IRMv1 proposed by Arjovsky et al. [29]. The penalty term is the sum of $S^e(f) = \|\nabla_w R^e_{erm}(w \circ \phi)\|^2$. w is a constant scalar multiplier of 1.0 for each output dimension, forcing the same classifier to be optimal in all environments. This formulation encourages the same classifier to be optimal across all environments, thereby promoting predictive invariance. Although IRMv1 explicitly targets this goal, it often suffers from unstable convergence and weak encoder supervision due to its bilevel optimization structure and use of a shared dummy classifier [30]. These limitations can cause the model to overfit to spurious correlations, especially when the number of training environments is small. Such underlying limitations can not be solved completely, but for specific application scenarios, there are mitigation strategies to realize a more robust IRM [31]. We explain this further in Section IV-E

2) Split Environments for Invariant Learning: Invariant learning relies on segmenting environments to highlight differences across them [18]. Early methods assumed prior knowledge of environment labels, which is often unavailable in practice [19], [32]–[34]. Recent approaches focus on invariant learning without predefined labels. Creager et al. [26] proposes estimating environment labels via prior clustering before applying invariant learning (EIIL), while others explore segmentation strategies such as natural clustering using dataset-provided labels or unsupervised clustering [35]. Experiments show natural clustering outperforms unsupervised methods. Regardless of the approach, effective segmentation requires environments to expose unstable information that can be ignored during invariant learning [18].

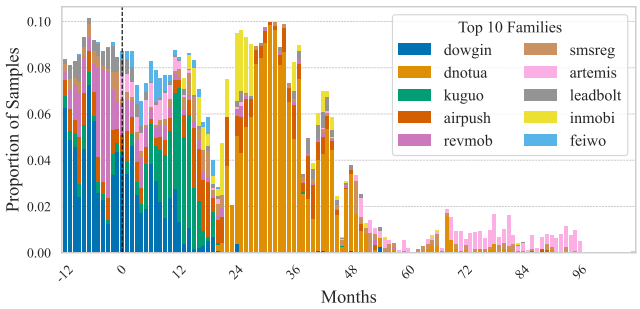


Fig. 1: The top-10 families' proportions vary over time; zero marks the test start, negatives indicate training.

III. Motivation

Learning-based malware detectors deliver outstanding performance in identifying potential threats; however, maintaining consistent performance under drift scenarios remains challenging. Figure 1 supports this claim, depicting the monthly proportion changes of the top-10 dominant families in our

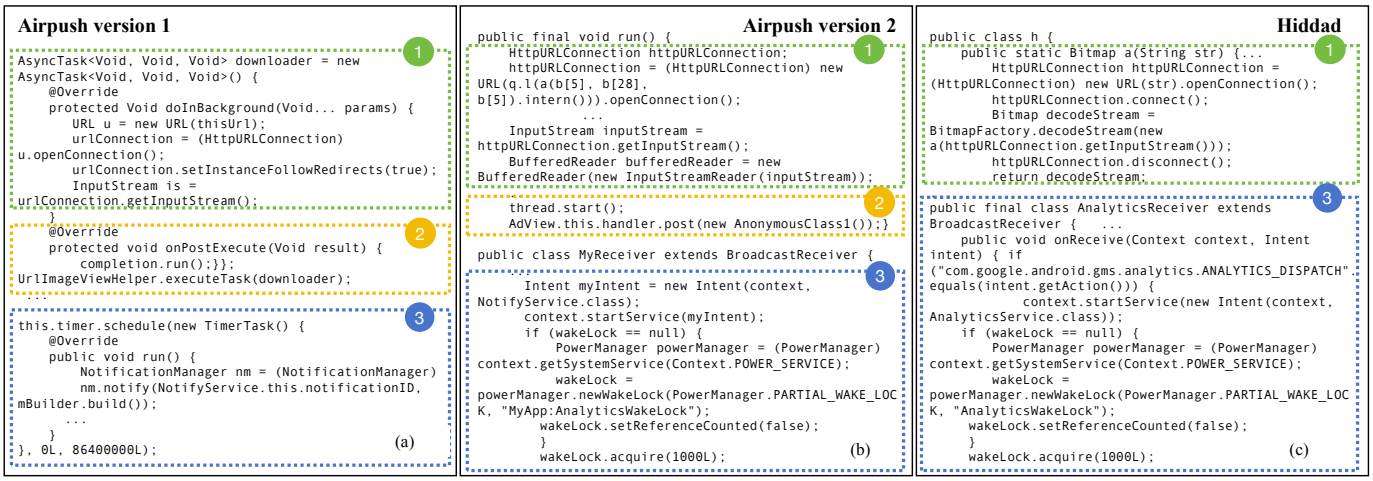


Fig. 2: (a), (b), and (c) show real code snippets from an early Airpush version, a later Airpush version, and the Hiddad adware family. Airpush's core behavior includes: (1) getting ad data from a specific URL, (2) asynchronous execution to avoid user interruption, and (3) pushing ads continuously through the background service. (a) and (b) demonstrate that both Airpush versions share invariant behaviors, with similar API calls and permissions despite implementation differences. Hiddad, while skipping step (2) for simpler ad display, shares steps (1) and (3) with Airpush, especially the newer version.

dataset (Section V-A1) across the training and testing phases. Notably, some families that dominate during training, such as Dowgin, gradually disappear in the testing phase, while new families, like Artemis, emerge. This dynamic evolution poses challenges for malware detectors' generalization across families. Even for families like Airpush, present in both phases, its feature distribution is also influenced by multiple factors including temporal fluctuations in proportions and API changes.

A. Threat Model

Malware natural drift refers to the gradual evolution of malware distributions over time, driven by the emergence of new families and modifications to existing ones. Without knowledge of detection methods, attackers can alter malware features, causing distribution shifts that undermine the long-term effectiveness of detection systems.

Our threat model focuses on natural drift, which differs from adversarial attacks that rely on knowledge of the detection scheme to identify optimal perturbations for evading models [36]. We find that even natural drift disrupts the adaptation capabilities of traditional methods, hindering their ability to generalize to new distributions. Therefore, in this work, we address the long-term challenges posed by malware natural drift by enhancing the robustness of detection systems against malware evolution. We consider adversarial drift to be out of scope for this paper and plan to explore it in future work.

B. Invariance in Malware Evolution

The challenges highlighted in Section III-A motivate the search for characteristics from training set which can be generalized to test samples. Indeed, definitions of malware families and types have inspired the exploration of such invariance by categorizing malware according to code structure, behavioral patterns and malicious intent. While specific implementations may vary due to evolutionary or obfuscation techniques, members of the same family typically exhibit behavioral patterns consistent with their overall goals. Furthermore, malware types similarly represent a wide range of operational intentions that remain stable across families. These internal consistencies form the basis of invariance, which we categorize as inter-family invariance and intra-family invariance: **Intra-family invariance**: Variants within a malware family maintain consistent operational behaviors and attack strategies, despite differences in implementation. **Inter-family invariance**: Common malicious patterns observed across malware families, such as resource abuse or evasion techniques.

To illustrate invariant malicious behaviors in drift scenarios, we select APKs from Androzoo[3] and decompile them using JADX[4] to obtain .java files. Our analysis focuses on core malicious behaviors in the source code. For intra-family invariance, we use versions of the Airpush family, known for intrusive ad delivery, from different periods. For inter-family invariance, we examine the Hiddad family, which shares aggressive ad delivery and tracking tactics but uses broader permissions, increasing privacy risks. Figure.2 shows code snippets with colored boxes highlighting invariant behaviors across samples. While Airpush uses asynchronous task requests, Hiddad relies on background services and scheduled tasks to evade detection.

Figure 2(a)[5] and (b)[6] show core code from this family in 2014 and later years, respectively. The 2014 version uses `NotifyService` and `TimerTask` to notify users every 24 hours, maintaining ad exposure. The later version, adapting to Android 8.0's restrictions, triggers `NotifyService`

[3]https://androzoo.uni.lu
[4]https://github.com/skylot/jadx
[5]MD5: 17950748f9d37bed2f660daa7a6e7439
[6]MD5: ccc833ad11c7c648d1ba4538fe5c0445

via `BroadcastReceiver` with `WAKE_LOCK` to sustain background activity. In Drebin's [9] feature space, these invariant behaviors are captured through features like `android_app_NotificationManager;notify`, `permission_READ_PHONE_STATE` and so on. Both implementations also use `HttpURLConnection` for remote communication, asynchronously downloading ads and tracking user activity, and sharing Drebin features such as `java/net/HttpURLConnection` and `android_permission_INTERNET`.

Similarly, Figure. 2(c)[7] shows a real sample from the Hiddad family, which uses HTTP connections for ad delivery, along with `AnalyticsServer` and `WAKE_LOCK` for continuous background services. Permissions like `android_permission_WAKE_LOCK` and API calls such as `getSystemService` reflect shared, cross-family invariant behaviors, whose learning would enhance model detection across variants.

Capturing the core malicious behaviors of Airpush aids in detecting both new Airpush variants and the Hiddad family, as they share similar malicious intents. These stable behaviors form consistent indicators in the feature space. However, detectors with high validation performance often fail to adapt to such variants, underscoring the need to investigate root causes and develop a drift-robust malware detector.

> ***Take Away**: The feature space of training samples contains invariance within and among malware families to be learned.*

C. Failure of Learning Invariance

Let $f_r \in \mathcal{R}$ be a sample in the data space with label $y \in \mathcal{Y} = \{0, 1\}$, where 0 represents benign software and 1 represents malware. The input feature vector $x \in \mathcal{X}$ includes features $\mathcal{F}$ extracted from f_r according to predefined rules. The goal of learning-based malware detection is to train a model $\mathcal{M}$ based on $\mathcal{F}$, mapping these features into a latent space $\mathcal{H}$ and passing them to a classifier for prediction. The process is formally described as follows:

$$\arg\min_{\theta} R_{erm}\left(\mathcal{F}\right), \tag{3}$$

where θ is the model parameter and $R_{erm}(\mathcal{F})$ represents the expected loss based on features space $\mathcal{F}$, defined as:

$$R_{erm}\left(\mathcal{F}\right) = \mathbb{E}[\ell(\hat{y}, y)]. \tag{4}$$

ℓ is a loss function. By minimizing the loss function, $\mathcal{M}$ achieves the lowest overall malware detection error.

1) Stability and Discriminability of Features: To investigate the drift robustness in malware evolution from the feature perspective, we define two key feature properties: stability and discriminability. Stability refers to a feature's ability to maintain consistent relevance across distributions, while discriminability reflects a feature's capacity to effectively distinguish categories. To avoid model-induced biases, we propose a modelless formal definition applicable to diverse architectures.

[7]MD5: 84573e568185e25c1916f8fc575a5222

Let f_j represent the j-th feature in the feature set $\mathcal{F}$, and S denote the set of all samples. To capture the behavior of feature f_j under different conditions, we compute its active ratio over a subset $S' \subseteq S$, representing how frequently or to what extent the feature is "active" within that subset. Specifically, for a binary feature space, feature f_j takes values 0 or 1 (indicating the absence or presence of the feature, respectively), the active ratio of f_j in the subset S' is defined as the proportion of samples where f_j is present, which is defined as Eq. 5:

$$r\left(f_j, S'\right) = \frac{1}{|S'|} \sum_{s \in S'} f_j(s). \tag{5}$$

The ratio measures how frequently the feature is activated within the subset S' relative to the total number of samples in the subset. At this point, we can define the stability and discriminability of features.

Definition 1. ***Stable Feature**: A feature f_j is defined as stable if its active ratio remains consistent across distinct temporal environments (i.e., data distributions from different time periods). Let the dataset S be partitioned into a sequence of temporal subsets $\mathcal{T} = \{S_{t_1}, S_{t_2}, \ldots, S_{t_m}\}$ based on the timestamps of the samples. Feature f_j is stable if, for any temporal subset $S_{t_k} \in \mathcal{T}$ of sufficient size:*

$$\forall S_{t_k} \in \mathcal{T}, |S_{t_k}| \geq n_0, \quad \left|r\left(f_j, S_{t_k}\right) - r\left(f_j, S\right)\right| \leq \epsilon, \tag{6}$$

where $\epsilon > 0$ is a small constant threshold. This condition ensures that the feature's prevalence is invariant to temporal evolution and does not rely on transient patterns specific to a single time window.

When we consider discriminability, there is a need to focus on the category to which the sample belongs. Thus, let $C = \{C_1, C_2, \ldots, C_k\}$ be a set of k classes, and $S_k \subseteq S$ be the subset of samples belonging to class C_k. The active ratio of feature f_j in class C_k is given by:

$$r\left(f_j, S_k\right) = \frac{1}{|S_k|} \sum_{s \in S_k} f_j(s). \tag{7}$$

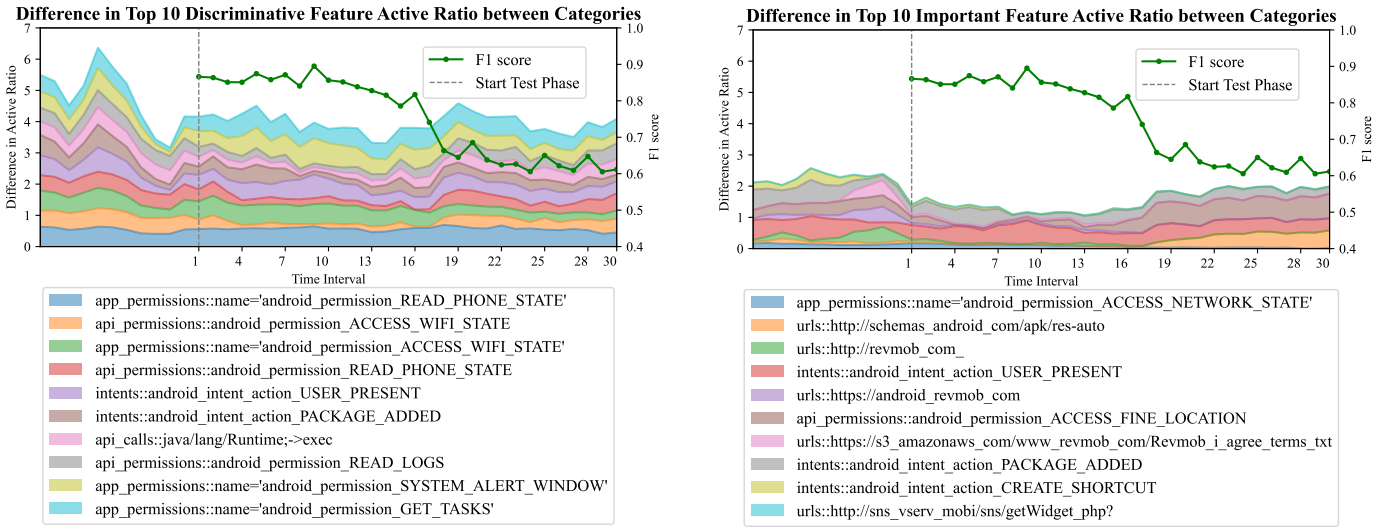
Definition 2. ***Discriminative Feature**: A feature f_j is discriminative if its active ratio differs significantly between at least two classes, C_p and C_q. Specifically, there exists a threshold $\delta > 0$ such that:*

$$\exists C_p, C_q \in C, p \neq q, \quad \left|r\left(f_j, S_p\right) - r\left(f_j, S_q\right)\right| \geq \delta. \tag{8}$$

Furthermore, the discriminative nature of the feature should be independent of the relative class sizes, meaning that the difference in activation should remain consistent despite variations in the proportion of samples in different classes. Mathematically, for any subset $\tilde{S}_p \subseteq S_p$ and $\tilde{S}_q \subseteq S_q$, where $\left|\tilde{S}_p\right| \neq |S_p|$ or $\left|\tilde{S}_q\right| \neq |S_q|$, the discriminative property still holds:

$$\left|r\left(f_j, \tilde{S}_p\right) - r\left(f_j, \tilde{S}_q\right)\right| \geq \delta. \tag{9}$$

Definition 3. ***Temporally Invariant Feature**: A feature f_j is temporally invariant if it simultaneously satisfies the stability*



(a) (b)

Fig. 3: (a) and (b) illustrate changes in the discriminability of the top 10 discriminative training features and the top 10 important testing features, respectively. "Discriminability" is defined as the absolute difference in active ratios between benign and malicious samples. The grey dotted line indicates the start of the testing phase, with preceding values representing each feature's discriminability across months in the training set.

and discriminability constraints across all temporal environments $\mathcal{T} = \{S_{t_1}, \ldots, S_{t_m}\}$. *Let* $S^+_{t_k}$ *and* $S^-_{t_k}$ *denote the malicious and benign subsets within the environment* S_{t_k}, *respectively. Formally:*

$$\forall S_{t_k} \in \mathcal{T}, \quad \underbrace{\left|r(f_j, S_{t_k}) - r(f_j, S)\right| \leq \epsilon}_{\textit{Stability}} \\ \wedge \quad \underbrace{\left|r(f_j, S^+_{t_k}) - r(f_j, S^-_{t_k})\right| \geq \delta}_{\textit{Discriminability}} \tag{10}$$

where ϵ *is the stability bound,* δ *is the discriminability threshold, and* $r(\cdot)$ *denotes the active ratio defined in Eq. 5.*

2) Failure Due to Learning Unstable Discriminative Features: The malware detector's strong performance within the same period indicates that it has effectively learned discriminative features to separate benign software from malware. However, performance degradation over time stems from the model's inability to capture stable discriminative features from the training set. To illustrate this, we randomly sample 110,723 benign and 20,790 malware applications from the Androzoo[8] platform (2014-2021). Applications are sorted by observation date, with 2014 samples used for training and the remainder divided into 30 test intervals with the same sample size. We extract Drebin [9] features, covering nine behavioral categories such as hardware components, permissions, and restricted API calls, and select the top 10 discriminative features based on active ratio differences to track over time. The model configuration follows DeepDrebin [1], a three-layer fully connected neural network with 200 neurons per layer. We evaluate performance in each interval using F1 scores. As shown in Figure 3, although the top 10 discriminative features maintain stable active ratios, the detector's performance consistently declines. We further examine feature importance over time using Integrated Gradients (IG) with added binary noise, averaging results across five runs to ensure robustness, as recommended by Warnecke et al. [37].

Figure 3 compares the active ratios of top discriminative (a) and important features (b). While stable discriminative features persist, ERM-based detectors often rely on unstable features with fluctuating effectiveness, resulting in poor generalization under drift. We observe that highly discriminative features, such as `api_calls::java/lang/Runtime;->exec` and `GET_TASKS`, are often linked to high-permission operations and potential malicious activity. These features, rarely seen in legitimate applications, reflect malware invariance, where core malicious intents persist despite evolving implementations.

> ***Take Away****: There are stable and highly discriminative features representing invariance in the training samples, yet current malware detectors fail to learn these features leading to decaying models' performance.*

D. Create Model to Learn Invariance

This discussion emphasizes the importance of learning stable, discriminative features for drift-robust malware detection. ERM captures both stable and unstable information correlated with the target variable [38], often relying on unstable features when they are highly correlated. The key challenge is to isolate and enhance stable features, aligning with the principles of invariant learning in Section II-C.

Invariant learning methods face challenges, as their success depends on effective environment segmentation to reveal unstable information [19], [20]. In malware detection, identifying variants that trigger distribution shifts is uncertain. High-quality representations from the encoder are also essential for

[8]https://androzoo.uni.lu

invariant predictors [21], yet Figure 3 shows that even during training, features fail to distinguish goodware from malware or capture malware's execution intent, relying on ambiguous cues.

To this end, we propose a temporal-aware environment segmentation method to reveal the instability of malware distribution drifts. In each environment, the ERM hypothesis guides the association with the target variable, and an encoder shared across environments attempts to learn all stable and unstable representations, after which it enhances the detector's generalization by minimizing the invariant risk to filter unstable elements.

***Take Away**: Invariant learning helps to learn temporal stable features, but it is necessary to ensure that the training set can expose unstable information and the encoder can learn good and diverse representations.*

IV. METHODOLOGY

A. Preliminary

1) Notations: We use uppercase and lowercase letters to denote matrices and vectors, respectively; $|\mathcal{D}|$ represents the total number of elements in set $\mathcal{D}$.

2) Problem setting: In Android malware detection, samples $x \in \mathbb{R}^d$ are divided into two parts: the labeled training data $\mathcal{D}_{tr} = (x_i^{tr}, y_i^{tr})_{i=1}^{|\mathcal{D}_{tr}|}$, where $y_i^{tr} \in \{0, 1\}$ represents the labels for benign and malicious applications, and the unlabeled test data $\mathcal{D}_{ts}$. Training data $\mathcal{T}_{tr}$ comes from a specific period $\mathcal{T}_{tr}$, with each sample x_i^{ts} ontains its corresponding observation timestamp t_i^{tr}. Test data $\mathcal{D}_{ts}$ comprises future samples $t_i^{ts} > \mathcal{T}_{tr}$ that appear progressively over time. We assume that the training and test data share the same label space and have unknown semantic similarities. The model is defined as $\mathcal{M} = c \circ \phi$, where c denotes the classifier and ϕ is the feature encoder. This work aims to enhance the encoder in capturing invariant semantic features from training data that generalize to unseen test data.

B. Overall Architecture

Section III-C2 highlights the need for learning temporally stable and discriminative features to improve robustness against distribution drift. We propose a temporal invariant training method, adaptable to any malware detector, which enhances robustness by extracting rich features and suppressing unstable factors through invariant risk minimization. The method comprises two components: **Multi-Proxy Contrastive Learning (MPC)**: representation diversity for benign and malware samples with dynamic proxies, encoding complex multi-family malware semantics into compact, discriminative embeddings. **Invariant Gradient Alignment (IGA)**: This component applies invariant risk minimization (IRM) to align gradients across environments, enabling the encoder to learn unified representations over time.

Figure 4 illustrates the overall framework of the proposed method, where Android applications are divided into non-overlapping environments based on their observation dates. The invariant training process includes two stages:

- Discriminative Information Amplification Stage: MPC is applied within each environment to minimize empirical risk, enabling the encoder to integrate discriminative features from each environment.
- Unstable Information Compression Stage: Building on the model trained in the first stage, MPC aligns features across environments, followed by IGA fine-tuning with IRM to suppress unstable information.

C. Training Environments Segmentation

Segmenting the training environment requires exposing unstable information. Malware distribution drift arises from multiple factors, such as application market demands or Android version updates. Dividing the training environment based on application observation dates effectively captures this mix of unknown causes. For a dataset with timestamps from $T_{\min}$ to $T_{\max}$, and a chosen time granularity Δ, samples are divided into t time windows, each representing an environment $e \in \mathcal{E}$, where $|\mathcal{E}| = t$. The resulting training set is an ordered multiset, $\mathcal{D} = \{D_1, D_2, ..., D_t\}$, with each environment D_t containing $|D_t|$ samples. A sample x_i with timestamp t_i is assigned to an environment using:

$$\mathcal{E}(x_i) = \left\lfloor \frac{t_i - T_{\min}}{\Delta} \right\rfloor. \tag{11}$$

As the environment index increases, timestamps approach the present. Time granularity impacts representation learning, balancing detailed distribution patterns against sample sufficiency. Finer granularity risks sparse samples, while coarser granularity may obscure shifts. Achieving balance requires careful consideration of label distribution to avoid misaligned representations. Section V-H explores the effects of granularity choices.

D. Multi-Proxy Contrastive Learning

Malware families have distinct distributions and imbalanced sample sizes, complicating malware category modeling in the embedding space. Benign samples may also display complex feature distributions due to factors like geographic location or user behavior. Treating all relationships within a single family equally can exacerbate imbalances, while overly homogenizing samples overlook valuable information. Thus, effective discrimination requires balancing the complex pairwise relationships within each category to ensure high-quality representations that capture diversity. To address this, we introduce a multi-proxy contrastive learning method, where each class is represented by multiple learnable proxy vectors, and samples are softly assigned to proxies to reflect their behavioral or structural variations.

For a batch of normalized feature embeddings $\mathbf{X}_c \in \mathbb{R}^{|\mathcal{B}_c| \times d}$ from category c , where $|\mathcal{B}_c|$ is the batch size and d is the feature dimension, we randomly initialize K learnable proxies for both benign and malicious categories. Specifically, the proxies for class c are denoted as $\mathbf{P}_c \in \mathbb{R}^{K \times d}$, where each row in $\mathbf{P}_c$ represents a proxy vector in the feature space. In this multi-proxy setup, each sample is softly assigned to all proxies in its category, which helps avoid over-reliance on any

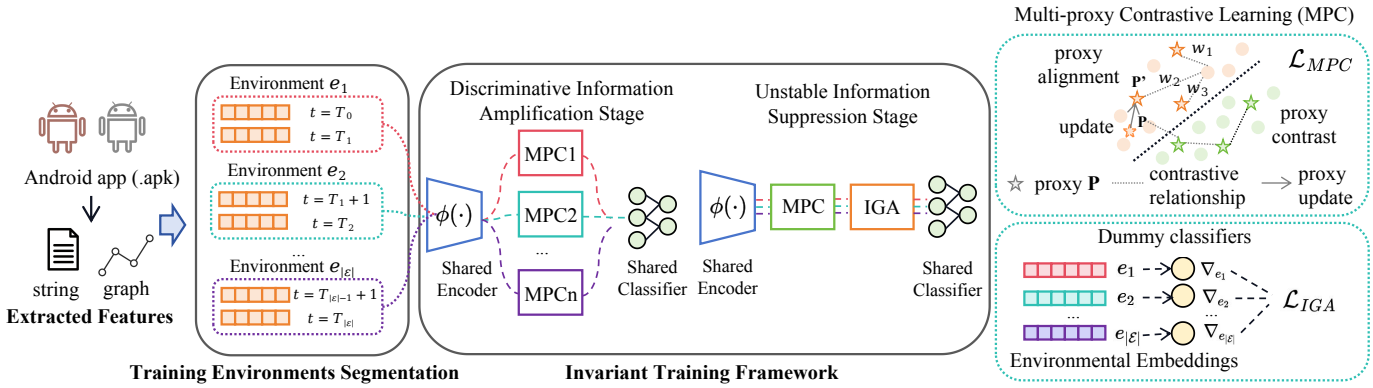


Fig. 4: The proposed invariant training framework and its core components

single proxy and mitigates the impact of noise. The similarity matrix between samples and all proxies in their category is computed and scaled by a temperature parameter τ:

$$\mathbf{S}_c = \mathbf{X}_c \mathbf{P}_c^\top / \tau, \quad \mathbf{S}_c \in \mathbb{R}^{|\mathcal{B}_c| \times K} \tag{12}$$

The assignment probability of the i-th sample is then calculated via a softmax:

$$p_{ij}^{(c)} = \exp(S_c^{(i,j)}) / \sum_{k=1}^{K} \exp(S_c^{(i,k)}) \tag{13}$$

, where $S_c^{(i,j)}$ denotes the similarity between the i-th sample and the j-th proxy in class c. For the sample over the corresponding proxies, the proxy alignment loss for each class c is denoted as:

$$\mathcal{L}_{pal}^c = -\frac{1}{|\mathcal{B}c|} \sum i = 1^{|\mathcal{B}c|} \sum j = 1^K p_{ij}^{(c)} \log p_{ij}^{(c)}. \tag{14}$$

Finally, the total proxy alignment loss across all classes is:

$$\mathcal{L}_{pal} = \frac{1}{C} \sum_{c=1}^{C} \mathcal{L}_{pal}^c. \tag{15}$$

Minimizing $\mathcal{L}_{pal}$ encourages samples of the same category to cluster around their most similar proxies, while still allowing flexibility in representation through multiple proxies. This not only enhances intra-class compactness and robustness to noise, but also captures potential sub-structures of each class.

The distribution of proxies determines the diversity and compactness of the embedding, all proxies should follow two principles: intra-class diversity and inter-class separation. To achieve this, we introduce two auxiliary regularization terms that operate directly on the proxy representations. **Intra-class diversity term**: This term aims to retain the internal diversity within each class to display multiple patterns. Thus, we encourage proxies within class to spread out across the hypersphere, allowing the model to represent sub-cluster variations. It is achieved by maximizing the average pairwise distance between proxies in each class:

$$\mathcal{L}_{intra} = -\frac{1}{C} \sum_{c=1}^{C} \frac{1}{\binom{K}{2}} \sum_{1 \le i < j \le K} \left\| \mathbf{P}_c^{(i)} - \mathbf{P}_c^{(j)} \right\|_2, \tag{16}$$

where $\mathbf{P}_c^{(i)}$ is the i-th proxy in class c and K is the number of proxies per class. The negative sign encourages diversity by maximizing distances. **Inter-class separation term**: To promote discriminative power, $\mathcal{L}_{inter}$ enforces a minimum margin m between class centers, preventing overlap or collapse of different category representations:

$$\mathcal{L}_{inter} = \frac{1}{C(C-1)} \sum_{c_1 \neq c_2} \Big[\left(m - \left\| \overline{\mathbf{P}}_{c_1} - \overline{\mathbf{P}}_{c_2} \right\|_2 \right) \cdot \mathbb{I}\left(\left\| \overline{\mathbf{P}}_{c_1} - \overline{\mathbf{P}}_{c_2} \right\|_2 < m \right) \Big] \tag{17}$$

where $\bar{\mathbf{P}}c = \frac{1}{K} \sum j = 1^K \mathbf{P}_c^{(j)}$ denotes the center of class c. We combine these regularization terms with the proxy alignment term to obtain the total multi-proxy contrastive loss $\mathcal{L}_{MPC}$ in this module, which can be formalized as:

$$\mathcal{L}_{MPC} = \mathcal{L}_{pal} + \lambda_{intra} \cdot \mathcal{L}_{intra} + \lambda_{inter} \cdot \mathcal{L}_{inter}, \tag{18}$$

where λ_{intra} and λ_{inter} balance the weight of these three loss functions.

E. Invariant Gradient Alignment

Section III-D highlights the prerequisites for learning invariant features: environment segmentation to expose unstable information and rich feature representations. Building on the definition of Invariant Risk Minimization (IRM), our goal is to guide encoders to focus on the stable aspects of these representations. Specifically, the encoder must ensure that representations of the same class across environments produce similar classifier gradients. IRMv1 [29] realizes this goal but as mentioned in Section II-C1, it suffers from suboptimal performance due to complex bilevel optimization and a shared dummy classifier. To leverage IRM under these conditions, we adopt a modified approach called Invariant Gradient Alignment (IGA) that mitigates IRM's deficiencies by introducing environment-specific dummy classifiers (scalar parameters s_e for each environment) to impose environment-aware gradient constraints. The objective function is shown in Eq.19.

$$\mathcal{L}_{IGA} = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \left\| \nabla_{s_e | s_e = 1.0} R^e(s_e \circ \phi) \right\|^2. \tag{19}$$

Here, s_e acts as a scalar for environment e, serving the role of a dummy classifier, set to 1.0 and updates through gradient backpropagation. $\mathcal{L}_{grad}$ evaluates how adjusting s_e minimizes

the empirical risk in environment e. $R^e(s_e \circ \phi)$ is represented as Eq. 20:

$$R^e(s_e \circ \phi) = \mathbb{E}^e \left[\mathcal{L}_{CLS} \left(s_e \left(\phi(x) \right), y \right) \right], \tag{20}$$

where $\mathcal{L}^e_{CLS}$ denotes binary cross-entropy for malware detection. The term $\phi(x)$ represents the output of the shared encoder for samples $x, y \sim p(x, y|e)$ from environment e. The gradient penalty term encourages uniform feature representation by aligning gradients of classifiers of all environments, thereby promoting consistent model performance.

F. Invariant Training Framework

Gradient adjustment for invariant learning classifiers relies heavily on the encoder's ability to learn rich representations [39]. Starting from random initialization often leads to suboptimal convergence. To overcome this, we propose a two-stage training strategy to first capture diverse features before applying invariant learning.

1) Discriminative Information Amplification: In the first stage, the multi-proxy contrastive learning module is applied to each environment to maximize the exploitation of discriminative features. Such initialization helps to amplify the discriminative power of the encoder in each environment. The optimization objective is defined in Eq.21:

$$\mathcal{L}_{DIA} = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} (\mathcal{L}^e_{CLS} + \alpha \cdot \mathcal{L}^e_{MPC}), \tag{21}$$

where $\mathcal{L}^e_{CLS}$ and $\mathcal{L}^e_{MPC}$ denote the classification loss and multi-proxy contrastive loss for environment e, respectively. Here, $\mathcal{L}^e_{CLS}$ is cross-entropy loss. Jointly minimizing the empirical risk across all environments equips the encoder with diverse feature representations, forming a robust foundation for invariant training.

2) Unstable Information Suppression: To mitigate overfitting to environment-specific features from the earlier stage, we reset the optimizer's parameters before the second training phase. This reset allows the model to refocus on the objectives of invariant learning. In this phase, we first apply a multi-proxy contrastive loss across all samples to enhance class representation learning. Next, invariant gradient alignment is used to harmonize classification gradients across environments. The updated optimization objective is defined in Eq. 22:

$$\mathcal{L}_{UIS} = \mathcal{L}_{CLS} + \alpha \cdot \mathcal{L}_{MPC} + \beta \cdot \mathcal{L}_{IGA}, \tag{22}$$

In training, the hyperparameters α and β balance the contributions of each loss term. This two-stage approach enables the model first to capture a broad feature set, then refine it for cross-environment invariance, enhancing generalization under distribution shifts. The proposed invariant training process is shown in Algorithm 1. This learning framework consists of two phases: discriminative information amplification and unstable information suppression. In Stage 1, the model enhances its ability to learn discriminative features. Specifically, in Line 6, the classification loss $\mathcal{L}^e_{CLS}$ is calculated for each environment to ensure correct class separation, and in Line 7, a multi-proxy contrastive loss $\mathcal{L}^e_{MPC}$ is computed to improve the discriminative power among samples. The combined loss for amplifying discriminative ability, including the classification and contrastive components, is computed in Line 9, and then the model is updated by backpropagation in Line 10. Stage 2 focuses on suppressing unstable information through invariant training. In Line 14, the optimizer is reset to remove the influence of previous training, while the model parameters from Stage 1 are retained in Line 15 to preserve the learned discriminative capabilities. Line 21 calculates the invariant gradient alignment loss $\mathcal{L}_{UIS}$. Line 22 calculates the overall classification loss for complete samples. The combined unstable information suppression loss is formed in Line 24, integrating the classification, contrastive, and alignment losses and updating the model in Line 25. This two-phase training process enables the model to learn discriminative and stable features, improving robustness and generalization in the face of distribution drift.

Algorithm 1 Algorithm of Invariant Training

Require: training dataset containing $|\mathcal{E}|$ environments $\mathcal{D}_{tr}$ and each sample x has a binary classification label $y_c, c \in [0, 1]$ and an environment label $y_e, e \in \mathcal{E}$, batch size $|\mathcal{B}|$, predefined number of epochs for stage 1 training N, hyperparameters α and β for loss weighting, model f with encoder network ϕ and predictor h.

Ensure:

1: **Stage 1: Discriminative Information Amplification**
2: **for** epoch $= 1$ to N **do**
3: **for** $i = 1, \ldots, |\mathcal{B}|$ **do**
4: **for** $e \in \mathcal{E}$ **do**
5: select mini-batch of samples $x^e_i \subseteq \mathcal{B}$
6: calculate classification loss $\mathcal{L}^e_{CLS}$
7: calculate multi-proxy contrastive loss $\mathcal{L}^e_{MPC}$
8: **end for**
9: obtain loss for empirical risk minimization $\mathcal{L}_{DIA} = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \mathcal{L}^e_{CLS} + \alpha \cdot \mathcal{L}^e_{MPC}$
10: update f by backpropagating $\mathcal{L}_{DIA}$
11: **end for**
12: **end for**
13: **Stage 2: Unstable Information Suppression**
14: Reset optimizer parameters
15: Continue with model parameters from Stage 1
16: **for** epoch $= N + 1$ to TotalEpochs **do**
17: **for** $i = 1, \ldots, |\mathcal{B}|$ **do**
18: **for** $e \in \mathcal{E}$ **do**
19: initialize dummy classifier $\{s_e = 1.0\}$
20: **end for**
21: calculate invariant gradient alignment loss $\mathcal{L}_{IGA}$
22: obtain classification loss $\mathcal{L}_{CLS}$
23: obtain multi-proxy contrastive loss $\mathcal{L}_{MPC}$
24: $\mathcal{L}_{UIS} = \mathcal{L}_{CLS} + \alpha \cdot \mathcal{L}_{MPC} + \beta \cdot \mathcal{L}_{IGA}$
25: update f by backpropagating $\mathcal{L}_{UIS}$
26: **end for**
27: **end for**

V. EVALUATION

This section evaluates the effectiveness of our proposed method in improving drift robustness for Android malware

TABLE I: Evaluation dataset. The dataset contains 357,344 applications from 2014 to 2025. The malware rate is about 10% per year. 2014 is the training year; there is no unseen family. Notably, *Unknown* is treated as one of *Seen* families in experiments.

Type		2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025
Benign		46536	25247	33731	55933	49641	21350	20643	19577	16480	8792	7305	3406
Malware	Seen Family	5154	7369	3304	2829	3537	1756	1786	1807	1271	115	972	6
	Unknown	263	391	690	4273	4623	43	78	86	80	742	264	58
	Unseen Family	/	2761	887	490	367	566	472	443	425	116	626	53
Total		51953	35768	38612	63525	58168	23715	22979	21913	18256	9765	9167	3523

TABLE II: AUT(F1,12m) of the candidate detectors on future year samples before and after adding different baselines and our framework to different feature spaces. w/o denotes the unmodified original classifier; all classifiers are trained based on 2014 samples. Δ denotes the performance percentage (%) between the proposed scheme and the base detector.

	Drebin		DeepDrebin				Malscan			BERTroid		
	w/o	T-stability	w/o	APIGraph	Guide Retraining	Ours	w/o	Guide Retraining	Ours	w/o	Guide Retraining	Ours
2015	0.858	0.824	0.859	0.868	0.814	$0.928_{\pm 0.012}$	0.842	0.851	0.902	0.771	0.772	0.833
Δ		↓ 3.96		↑ 1.05	↓ 5.24	↑ 8.03		↑ 1.07	↑ 7.13		↑ 0.01	↑ 8.04
2016	0.811	0.782	0.813	0.818	0.749	0.877	0.799	0.795	0.850	0.744	0.743	0.789
Δ		↓ 3.58		↑ 0.06	↓ 7.87	↑ 7.87		↓ 0.50	↑ 6.38		↓ 0.13	↑ 6.05
2017	0.750	0.717	0.752	0.754	0.656	0.815	0.739	0.734	0.783	0.708	0.713	0.752
Δ		↓ 4.40		↑ 0.03	↓ 12.77	↑ 8.38		↓ 0.68	↑ 5.95		↑ 0.71	↑ 6.21
2018	0.724	0.718	0.726	0.726	0.623	0.794	0.713	0.708	0.755	0.686	0.694	0.726
Δ		↓ 0.83		0.00	↓ 14.19	↑ 9.37		↓ 0.70	↑ 5.89		↑ 1.74	↑ 2.19
2019	0.700	0.726	0.713	0.713	0.575	0.769	0.700	0.694	0.741	0.660	0.670	0.701
Δ		↑ 3.71		0.00	↓ 19.35	↑ 7.85		↓ 0.86	↑ 5.86		↑ 2.42	↑ 6.36
2020	0.665	0.723	0.700	0.696	0.527	0.732	0.687	0.681	0.726	0.630	0.641	0.674
Δ		↑ 8.72		↓ 0.57	↓ 24.71	↑ 4.57		↓ 0.87	↑ 5.68		↑ 2.70	↑ 7.14
2021	0.645	0.716	0.689	0.686	0.497	0.717	0.676	0.669	0.713	0.609	0.623	0.653
Δ		↑ 11.01		↓ 0.44	↓ 27.87	↑ 4.06		↓ 1.04	↑ 5.47		↑ 3.28	↑ 7.72
2022	0.616	0.687	0.661	0.658	0.450	0.688	0.643	0.635	0.680	0.583	0.597	0.627
Δ		↑ 11.53		↓ 0.45	↓ 31.92	↑ 4.84		↓ 1.24	↑ 5.75		↑ 3.43	↑ 8.06
2023	0.610	0.662	0.638	0.635	0.413	0.665	0.615	0.607	0.652	0.558	0.574	0.602
Δ		↑ 9.01		↓ 0.47	↓ 34.64	↑ 4.23		↓ 1.30	↑ 6.02		↑ 2.87	↑ 8.42
2024	0.601	0.645	0.625	0.623	0.421	0.654	0.615	0.607	0.652	0.558	0.574	0.602
Δ		↑ 7.32		↓ 0.01	↓ 34.64	↑ 4.64		↓ 1.30	↑ 6.02		↑ 2.87	↑ 8.42
2025	0.595	0.636	0.616	0.619	0.416	0.647	0.615	0.607	0.652	0.558	0.574	0.602
Δ		↑ 6.86		↑ 0.01	↓ 34.64	↑ 5.03		↓ 1.30	↑ 6.02		↑ 2.87	↑ 8.42

detectors across various feature spaces. We also analyze the contributions of each component in the invariant learning framework. The evaluation addresses the following research questions:

- RQ1. Can our framework mitigate detector aging across different feature spaces?
- RQ2. Can our framework stabilize detectors in different drift scenarios?
- RQ3. Does our framework effectively learn invariant features of applications?
- RQ4. Can our framework be equipped to continual learning setting?

To ensure reliability, experiments were conducted using random seeds (1, 42, 2024), with results averaged. All experiments ran on an RTX A6000. The dataset and code will be released upon paper acceptance.

A. Evaluation Settings

1) Dataset: To evaluate the effectiveness of our approach under long-term malware evolution, we construct a dataset spanning 2014 to 2025 by extending the Transcendent [10], [25] dataset with newly collected samples from the Androzoo[9], a comprehensive repository aggregating apps from multiple sources. The original Transcendent dataset covers samples from 2014 to 2018, while samples from 2019 to 2025 are newly collected. Each sample is timestamped using its VirusTotal submission date[10], which we treat as the discovery time. Malware samples released in 2025 are sparsely updated on Androzoo (fewer than 15 new malware samples per month on average), making the data less representative. Therefore, our dataset includes samples collected up to June 2025. After filtering samples with failed downloads or feature extraction errors, the final dataset consists of 308,641 benign apps and 48,703 malicious apps, labeled based on VirusTotal reports. Apps flagged as malicious by more than four vendors are treated as malware. To track malware family evolution, we adopt Euphony [40] for family labeling; samples without an assigned family are labeled as unknown. To mitigate temporal and spatial biases, we apply the TESSERACT [13] methodology. In our experiments, the training set consists exclusively

[9]https://androzoo.uni.lu
[10]https://www.virustotal.com

of samples from 2014, while the test set spans the remaining years. The training data are further split into a proper training set (80%) and a validation set (20%). Detailed statistics of the dataset are reported in Table I.

For evaluation under temporal distribution shift, we further organize the test data according to both time and family availability. Specifically, we consider two evaluation settings: **Closed-world**, where test samples contain only malware families observed in the training set to test intra-family invariance, and **Open-world**, where test samples contain only malware families absent from the training set to test inter-family invariance. Due to the highly imbalanced and non-uniform appearance of malware families over time, directly evaluating year-by-year may bias the comparison. Therefore, for each setting, we sort test samples chronologically and partition them into 10 equal-sized temporal segments. These segments, indexed as 1-10 in subsequent tables and figures, represent successive evaluation intervals with comparable sample sizes rather than fixed calendar years.

2) Malware Ground Truth Selection: Malware labels are determined by multiple security engines, often with conflicting decisions. Researchers set thresholds to balance sensitivity and specificity; lower thresholds include borderline software, blurring classification boundaries. In this study, we use a low threshold ($vt = 4$) to evaluate performance in this challenging scenario.

3) Candidate Detectors: In Android malware detection, an APK file serves as input, containing the codebase (e.g., .dex files) and configuration files (e.g., manifest.xml), which offer behavioral insights such as API calls and permissions, then organized as different formats. We select three representative feature spaces:

Drebin: Drebin [9] is a malware detector using binary feature vectors from nine data types (e.g., hardware, API calls, permissions) for linear classification. To align with our framework's focus on neural network architectures, we include its deep learning variant, DeepDrebin [1], which uses the same feature space but employs a three-layer deep neural network (DNN) for feature extraction and classification.

Malscan: Malscan [41] adopts a graph-based feature space, extracting sensitive API calls from APKs and combining four centrality measures (degree, Katz, proximity, and harmonic wave centralities) as features. We concatenate these features according to the optimal method detailed in the paper for detection tasks.

BERTroid: BERTroid [42] leverages Android application permission strings, encoding them into dense representations with a BERT-based pre-trained model that captures contextual relationships among permissions. For this study, we use the pooled BERT output as the feature representation for malware detection.

For fair evaluation, we apply a two-layer linear classifier to each feature representation, with the final layer performing binary classification.

4) Baseline: To evaluate the robustness of our scheme across different feature spaces and training paradigms, we compare it with both static baselines and a continual learning baseline.

a) Static Baselines: We select representative static enhancement methods that improve the robustness of malware detectors under a fixed problem definition and offline training setting. These methods strengthen existing models without redefining the detection task or introducing new deployment assumptions, which align with our design purpose. Selected baselines cover different levels of abstraction in robustness enhancement, ranging from explicit feature manipulation to implicit representation learning.

APIGraph: APIGraph [2] represents an explicit feature selection strategy. It assumes that malware drift primarily arises from implementation-level API changes while preserving semantic behaviors, and stabilizes the feature space by clustering semantically similar APIs. This method relies on a predefined assumption about the source of drift and targets arbitrary API-based feature spaces. From the original method, we derive 2,000 clusters, replacing all features within the same cluster with the corresponding index.

T-stability: T-stability [3] represents an explicit feature constraint strategy. It assumes that feature drift follows consistent temporal trends and constrains features with unstable temporal behavior. This approach models drift patterns directly in the feature space with strong assumptions about feature evolution.

Guided Retraining: [6] represents an implicit representation enhancement strategy. Instead of modeling drift explicitly, it improves robustness by refining representations for difficult samples detected by the confusion matrix through supervised contrastive learning, without assuming specific drift sources or patterns and enables separate processing of easy and difficult samples during inference for better accuracy.

Together, these methods span a spectrum from explicit, assumption-driven feature-level robustness to implicit representation-level enhancement, enabling a principled comparison of different static robustness strategies.

b) Continual Learning Baselines: To evaluate the effectiveness of TIF applying on a continual learning framework to reduce cost while keeping reliability, we select **HCC** [7], a state-of-the-art continual learning framework for Android malware detection, which addresses distribution drift by periodically updating the detector with newly observed samples during deployment. HCC proposes enhanced representations with a drift sample detection and model updating strategy, serving as a comprehensive baseline.

5) Metrics: We evaluate different approaches using static and dynamic metrics. For static evaluation, the macro-F1 score is used to address the class imbalance in the Android dataset, offering a balanced assessment of precision and recall on a fixed-time test set. For dynamic evaluation, we employ the Area Under Time (AUT), a metric proposed by TESSERACT [13], which measures performance over time. AUT is calculated as:

$$\mathrm{AUT}(m, N) = \frac{1}{N-1} \sum_{t=1}^{N-1} \left(\frac{m(x_t + 1) + m(x_t)}{2} \right), \quad (23)$$

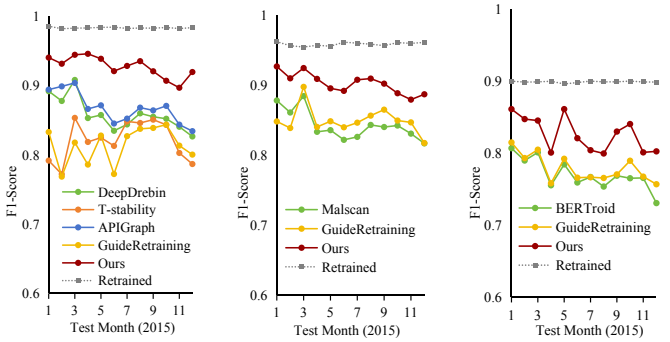


Fig. 5: Monthly performance of DeepDrebin (a), Malscan (b), and BERTroid (c) during the first test year (2015), with models initially trained on 2014 data. *Retrained* denotes detectors updated monthly with labeled test samples.

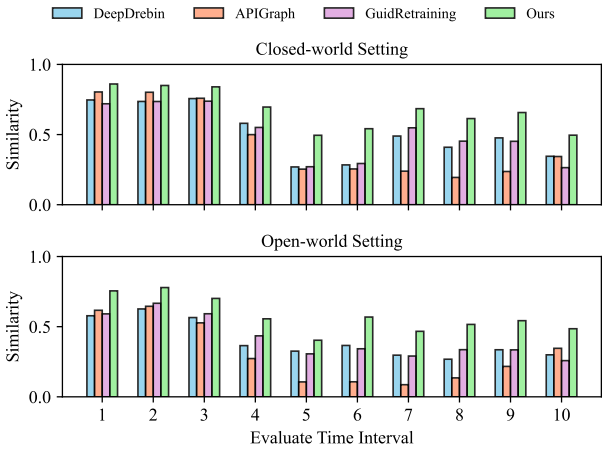


Fig. 6: Cosine similarity of malware representations across successive evaluation intervals under closed-world and open-world settings, computed using the DeepDrebin [1] model in the Drebin feature space [9]. Higher values indicate more stable representations.

where m represents the performance metric, with the F1-score selected as its instance. $m(x_t)$ denotes the performance metric at time t, and N is the number of monthly testing slots. AUT ranges from $[0, 1]$, with a perfect classifier achieving an AUT of 1 across all test windows.

B. Enhance Different Feature Space (RQ1)

This section evaluates the TIF for mitigating detector degradation across feature spaces. The classifier is trained on 2014 samples and tested monthly from 2015 to 2023, with annual results (AUT(F1, 12m)) summarized in Table II. To optimize short-term performance, we use monthly partitioning, with segmentation granularity discussed in Section V-H. Given that T-stability [3] is designed for the Drebin detector, we compare both the original Drebin [9] and its T-stability variant. APIGraph [2] supports Drebin and DeepDrebin but is incompatible with BERTroid [42] and Malscan [41], while GuideRetraining [6] applies to all detectors. We evaluate TIF against these baselines to assess absolute performance, noting that frequent switching between methods is impractical. Notably, TIF consistently outperforms others over the long term, even against the best yearly performances of alternative methods.

As drift increases over the test years, AUT declines for all methods, but TIF achieves up to an 8% improvement in the first year. This reflects real-world scenarios where detectors are retrained annually [2]. Figure 5 illustrates 2015 monthly performance, with a grey dashed line marking the upper performance bound. For each month n, training and validation sets include samples up to month n, simulating expert-labelled data for retraining. Without additional labels, TIF closely approaches this bound, demonstrating robust monthly gains. While AUT gains diminish over time, periodic retraining ensures manageable performance decline. T-stability exhibits lower performance on early test samples but degrades more slowly on distant ones, which is consistent with its design objective of suppressing temporally unstable features. By constraining features whose statistical contribution changes over time, T-stability reduces reliance on transient but initially discriminative signals, leading to conservative early performance but improved robustness under long-term drift. However, this feature-level, suppressive strategy prioritizes stability over discriminability and does not actively strengthen stable and discriminative feature combinations, which limits its overall performance compared to TIF.

> ***Take Away***: *TIF slows down the aging of detectors in each feature space and has significant performance gains in the years just after deployment.*

C. Robustness in Different Drift Scenarios (RQ2)

This section evaluates whether TIF learns stable representations under different distribution drift. We first evaluate the classification performance of the DeepDrebin [1] detector against baselines and TIF in different scenarios (closed-world and open-world), and compute AUT(F1, 12m) starting from the validation set. Table **??** shows that TIF improves AUT(F1, 12m) by 5.78% in the closed-world setting and by 3.17% in the open-world setting, indicating its effectiveness in learning inter- and intra-invariant representations.

To directly assess representation stability, we analyze the similarity between malware representations across time. Specifically, for each temporal test interval, we extract the hidden representations of malware samples from the encoder and compute their mean embedding. We then calculate the cosine similarity between this mean test embedding and the mean embedding of malware samples in the training set. Figure 6 plots this similarity across successive test intervals under both closed-world and open-world settings. Higher similarity indicates that the learned representations remain closer to those learned during training, reflecting greater temporal stability. As shown in Figure 6, TIF consistently achieves higher representation similarity than all baselines in the closed-world and open-world settings, respectively.

TABLE III: AUT(F1, 12m) of the closed-world drift and open-world test scenarios after adding different schemes in the DeepDrebin [1] detector. AG and GR represent baselines, APIGraph [2] and GuideRetraining [6], respectively

	Deepdrebin [1]							
	Closed world				Open world			
	w/o	AG	GR	Ours	w/o	AG	GR	Ours
1	0.911	0.920	0.906	0.949	0.903	0.905	0.871	0.918
2	0.880	0.891	0.861	0.927	0.860	0.863	0.788	0.879
3	0.855	0.865	0.808	0.902	0.817	0.822	0.714	0.839
4	0.811	0.821	0.727	0.861	0.764	0.771	0.657	0.788
5	0.777	0.789	0.663	0.827	0.727	0.735	0.602	0.752
6	0.758	0.770	0.627	0.805	0.705	0.713	0.541	0.730
7	0.742	0.754	0.598	0.788	0.689	0.698	0.497	0.715
8	0.733	0.645	0.573	0.778	0.678	0.687	0.443	0.705
9	0.728	0.739	0.552	0.771	0.671	0.679	0.692	0.697
10	0.714	0.723	0.529	0.756	0.658	0.665	0.415	0.682

TABLE IV: The sum of feature contribution scores (FCS) for the comparison schemes during training and testing in both closed and open-world settings. AG and GR represent the baselines APIGraph [2] and GuideRetraining [6].

	DeepDrebin							
	Closed world				Open world			
	w/o	AG	GR	Ours	w/o	AG	GR	Ours
Train	27.22	28.33	25.15	32.15	27.22	28.33	25.15	32.15
1	23.98	24.49	19.04	27.98	22.91	23.62	18.58	27.33
2	21.70	23.53	18.44	26.14	19.46	22.57	17.04	25.74
3	19.78	20.27	17.12	22.92	17.56	19.29	16.10	21.97
4	17.94	18.04	15.11	20.16	16.68	16.74	13.72	19.26
5	15.73	15.82	13.52	18.72	14.69	15.09	12.06	18.57
6	14.26	14.23	12.46	16.63	13.56	13.75	11.10	16.59
7	13.14	13.21	11.67	14.89	12.12	13.19	11.63	14.75
8	12.58	12.47	10.88	13.83	11.75	11.83	10.05	13.66
9	12.26	12.31	10.52	12.91	11.24	11.12	9.30	12.38
10	11.98	12.01	6.83	12.52	10.51	10.62	6.01	11.62

Take Away: *TIF generates stable feature representations for both evolved variants of existing families and new malware families.*

D. Effective Invariant Feature Learning (RQ3)

We investigate whether the proposed framework consistently focuses on discriminative features that remain effective under temporal distribution drift. As discussed in Section III-C2, ERM-based models often rely on transient correlations that are highly predictive at a specific time but degrade as data distributions evolve. We use the Drebin feature space as an example, because it has separable and interpretable characteristics. RQ3 examines this behavior from a feature-attribution perspective. To this end, we define the *Feature Contribution Score* (FCS) for each feature f_j as:

$$FCS_j = |r(f_j, S_m) - r(f_j, S_b)| \cdot IS_j, \tag{24}$$

where $r(f_j, S_m)$ and $r(f_j, S_b)$ denote the activation ratios of feature f_j in malware and benign samples, respectively, and

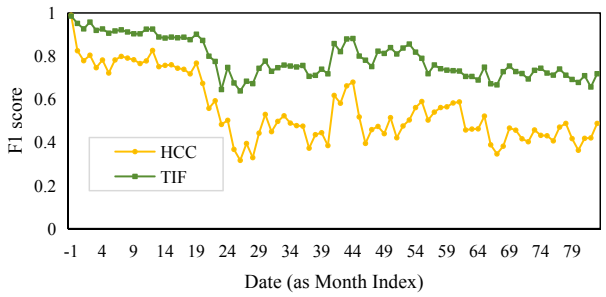


Fig. 7: F1 scores for TIF and HCC [7] in the no continual learning scenario. A subscript of -1 indicates the result of the model on the initial validation set.

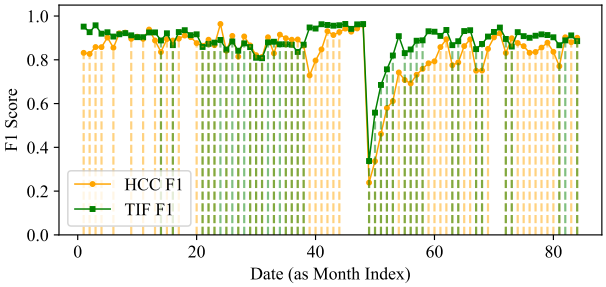


Fig. 8: Detection performance of HCC [7] and the proposed TIF-equipped framework on Android malware. Yellow dashed lines indicate update points of HCC; green lines represent TIF. Overlapping updates appear as brown due to color blending.

IS_j is the feature importance score computed via Integrated Gradients with respect to the malware category. The first term captures the discriminative strength of a feature in the current evaluation set, while IS_j reflects the extent to which the model relies on that feature. For a given model $\mathcal{M}$, we sum FCS over all features to obtain an aggregate score indicating the model's overall emphasis on discriminative features.

Importantly, FCS does not explicitly encode temporal information. Instead, feature stability is inferred from the persistence of high FCS values across successive temporal segments. Transient features may exhibit high FCS at a particular time, but their contribution typically decays as the distribution shifts. In contrast, effective invariant feature learning is characterized by a slower degradation of aggregate FCS over time, indicating that the model continues to emphasize features whose discriminative power remains valid. Using the closed-world and open-world evaluation protocol defined in Section V-A1, we compute FCS during training and across the 10 successive temporal test segments. As shown in Table IV, invariant training consistently yields higher and more persistent FCS values than ERM-based baselines in both settings. This suggests that TIF maintains attention to discriminative features that remain effective under temporal drift, providing quantitative evidence that complements the representation-level analysis in RQ2.

Take Away: *TIF enables the detector to learn stable, discriminative features from the training set.*

E. Effectiveness on Continual Learning (RQ4)

As discussed in Section II-B, continual learning frameworks mitigate this by updating models with drifted samples over time. Owing to its model- and feature-agnostic design, TIF can be integrated into continual learning frameworks to enhance representation stability.

To comprehensively evaluate the benefits of TIF, we consider two complementary settings on baseline, HCC [7]: **Without continual framework:** isolates the representation learning ability of the model by removing the update process, enabling a direct comparison of representation quality between TIF and HCC. **With continual framework:** evaluates the full detection pipeline under realistic drift, where models are periodically updated using newly observed samples. This separation allows us to disentangle improvements due to better representations from those due to continual adaptation.

1) Without Continual Framework: As noted in Section III-D, high-quality representations of complex malware families are essential for effective continual learning. To isolate representation performance, we remove the update stage and directly compare TIF and HCC at the representation level (Figure 7). TIF achieves an average F1 improvement of 49.16% on drift samples compared to HCC, whose hierarchical contrastive learning is limited by the large number of families and severe class imbalance. In contrast, TIF balances representation granularity and sample size with multiple proxies, and leverages invariant learning to focus on stable features. Benefited by high-quality representation, TIF helps models to remain robust at the beginning of the evaluation stage, delaying the time when the model first needs to be updated.

2) With Continual Framework: For fair comparison with HCC in a realistic drift-handling scenario, we follow its update scheduling scheme but remove the assumption of fixed monthly updates. In our experiment, because our method does not focus on threshold selection, the update trigger is illustrated using a simple F1-based rule: the model is evaluated on the next month's data, and if the F1 score falls below 0.90 (in this demonstration), an update is triggered by adding a fixed budget of low-confidence samples to the training set. We stress that this use of F1 is only for explanatory purposes in our controlled study to simulate an ideal drift detector. This allows us to strictly measure the lifespan of the representations. In a real deployment, such a rule would be replaced by an uncertainty-based criterion [11] or other update conditions [10], [25], many of which can be plugged into our framework without altering the rest of the design.

Following Section V-A1, we perform a longitudinal evaluation (2015-2021) using Drebin features [9] trained on 2014 data. This period is selected to rigorously assess representation robustness against behavioral drift while minimizing the confounding impact of feature depletion caused by post-2021 advanced packing. We adopt a probability-based update strategy, adding 100 low-confidence samples per step, consistent with HCC's best performance setup [7]. As shown in Figure 8, TIF demonstrates superior stability: it reduces update frequency from 65 (HCC) to 38, cutting labeling costs by 41.5% without compromising detection performance.

TABLE V: AUR(F1, 12m) after adding different components on Deepdrebin detector. • means with; ○ means without.

DeepDrebin [1]					
MPC1	○	•	•	•	•
MPC2	○	○	•	○	•
IGA	○	○	○	•	•
2015	0.859	0.902	0.902	0.900	0.928
2016	0.813	0.847	0.854	0.856	0.877
2017	0.752	0.783	0.782	0.793	0.815
2018	0.726	0.760	0.750	0.771	0.794
2019	0.713	0.747	0.742	0.765	0.769
2020	0.700	0.719	0.731	0.743	0.732
2021	0.689	0.704	0.720	0.731	0.717
2022	0.661	0.667	0.686	0.693	0.688
2023	0.638	0.637	0.658	0.662	0.665

TABLE VI: AUT (F1,12m) of the DeepDrebin [1] under different temporal environment segmentation methods.

	DeepDrebin [1]					
	w/o	monthly	quarterly	n=4	n=8	n=12
2015	0.859	0.928	0.915	0.914	0.902	0.905
2016	0.813	0.879	0.866	0.864	0.854	0.853
2017	0.752	0.816	0.801	0.800	0.783	0.799
2018	0.726	0.794	0.783	0.780	0.758	0.754
2019	0.713	0.769	0.775	0.762	0.745	0.741
2020	0.700	0.732	0.764	0.730	0.718	0.719
2021	0.689	0.717	0.750	0.709	0.701	0.705
2022	0.661	0.688	0.717	0.667	0.669	0.671
2023	0.638	0.665	0.690	0.634	0.643	0.645

***Take Away**: TIF learns stable representations that delay degradation under drift and reduce update frequency, lowering labeling cost without sacrificing accuracy.*

F. Ablation Study

This section evaluates the robustness of each component through an ablation study. Five configurations are tested: the base model, +MPC1, +MPC1 +MPC2, +MPC1 +IGA, and +MPC1 +IGA +MPC2, where MPC1/MPC2 represent multi-proxy contrastive learning in two training stage, and IGA denotes invariant gradient alignment which is only used in the second stage. Table V shows the AUT(F1, 12m) results for each setup with the Drebin detector.

MPC1 improves intra-environment representations, surpassing the baseline. Adding MPC2 and IGA enhances generalization by aligning gradients and capturing global features. The full configuration achieves the highest robustness by integrating stable and discriminative feature learning.

G. Overhead Analysis

We analyze the training overhead of TIF relative to standard ERM-based training. Let the training data be partitioned into E

temporal environments. In each iteration, a mini-batch of size B is sampled from each environment, yielding a total batch size $B_t = B \cdot E$. Let d denote the representation dimension, $C = 2$ the number of classes, and K the number of learnable proxies per class. We denote by $F(B_t)$ the dominant cost of one ERM training step, i.e., the forward and backward passes of the shared encoder and classifier. For common neural detectors, this cost is dominated by dense layers and scales at least as $F(B_t) = \Omega(B_t d^2)$.

Stage 1 (Discriminative Information Amplification) augments ERM with Multi-Proxy Contrastive learning (MPC). Beyond $F(B_t)$, MPC computes similarities between B_t sample representations (dimension d) and K proxies, incurring $\mathcal{O}(B_t K d)$, plus a proxy regularization cost of $\mathcal{O}(CK^2 d)$. Thus, the per-iteration cost is:

$$\mathcal{S}_1 = F(B_t) + \mathcal{O}(B_t K d) + \mathcal{O}(CK^2 d).$$

Stage 2 (Unstable Information Suppression) retains MPC and adds Invariant Gradient Alignment (IGA). IGA computes gradient norms w.r.t. environment-specific scalar dummy classifiers, introducing a strictly linear cost of $\mathcal{O}(B_t d)$. The total per-iteration cost becomes:

$$\mathcal{S}_2 = F(B_t) + \mathcal{O}(B_t K d) + \mathcal{O}(CK^2 d) + \mathcal{O}(B_t d).$$

Since Stage 1 is strictly less computationally expensive than Stage 2 ($\mathcal{S}_1 < \mathcal{S}_2$), we use $\mathcal{S}_2$ to upper-bound the cost of the entire training process. Using $F(B_t) = \Omega(B_t d^2)$ and dividing the lower-order terms by $F(B_t)$, we obtain:

$$\begin{aligned}\mathcal{S}_2 &= F(B_t)\left(1 + \mathcal{O}\left(\frac{K}{d}\right) + \mathcal{O}\left(\frac{1}{d}\right) + \mathcal{O}\left(\frac{CK^2}{B_t d}\right)\right) \\ &= F(B_t)\,(1+\gamma),\end{aligned}$$

where γ is a small constant determined by K and E (through $B_t = B \cdot E$). In our setting, with $K = 5$ and $K \ll d$, $\gamma \ll 1$. This overhead affects only offline training and does not introduce any inference-time cost. Our analysis confirms that despite the two-stage optimization, TIF incurs only an acceptable constant-factor overhead relative to standard ERM.

H. Environment Granularity Selection

Environment segmentation plays a critical role in invariant learning by controlling the trade-off between exposing temporal variation and ensuring sufficient samples per environment. We evaluate three segmentation strategies—monthly, quarterly, and equal-sized splits ($n = 4, 8, 12$), with results reported in Table VI. Equal-sized splits often introduce severe label imbalance, weakening environment-wise invariance estimation. Temporal segmentation better preserves natural evolution patterns. Among them, finer-grained (monthly) segmentation is effective under mild, short-term drift by exposing subtle variations and leads to strong performance in the early deployment stage, which aligns with our focus on initial deployment robustness. However, overly fine-grained partitions may reduce per-environment representativeness and provide insufficient distributional contrast under long-term drift. In contrast, quarterly segmentation introduces stronger temporal contrast while maintaining adequate samples per environment, yielding better generalization across larger temporal gaps. Overall, these results indicate that the optimal granularity depends on the scale of temporal drift rather than a single universally optimal choice.

I. Case Study

We conduct a case study to understand *why* TIF achieves superior robustness under temporal distribution drift, rather than merely *how much* it improves performance. We focus on the interpretable Drebin feature space [9] and analyze the top-10 important features identified by Integrated Gradients [43] for two malware families, Airpush and Hiddad (Table VII). The upper part of the table reports features emphasized by TIF, while the lower part shows those emphasized by the ERM-based baseline DeepDrebin. These two families both belong to the adware category and therefore share similar malicious intents, such as background activation and advertisement fetching, while differing substantially in their concrete implementations. This makes them a suitable pair for analyzing both intra-family drift ($\text{Airpush}_{tr} \rightarrow \text{Airpush}_{te}$) and inter-family drift under shared semantic intent ($\text{Airpush}_{tr} \rightarrow \text{Hiddad}_{te}$).

Under intra-family drift, TIF consistently prioritizes intent- and lifecycle-level features that correspond to explicit system broadcast events, such as `USER_PRESENT` and `PACKAGE_ADDED`. As shown in Table VII, these features directly indicate user presence and application installation, respectively, and serve as fundamental triggers for background service activation in adware. Because such broadcast events are defined at the Android framework level, they remain stable across different versions of the same malware family. In contrast, DeepDrebin assigns higher importance to implementation-specific features observed in the same table, including APIs such as `getRunningTasks` and `requestLocationUpdates`, as well as permissions like `ACCESS_FINE_LOCATION`. These features reflect concrete implementation choices that are sensitive to code refactoring and API usage changes within the family, and therefore exhibit weaker temporal stability.

The difference becomes more pronounced under inter-family drift. When evaluated on Hiddad, TIF emphasizes capability-level features that are intrinsic to adware functionality, where `ACCESS_NETWORK_STATE`, which indicates the general ability to access network connectivity for advertisement fetching or remote communication. This capability is required across different adware families, regardless of their internal implementations. DeepDrebin focuses on more restrictive and environment-dependent signals in the Hiddad setting, such as `ACCESS_WIFI_STATE`, `SYSTEM_ALERT_WINDOW`, and location-related APIs. These features encode specific runtime assumptions (e.g., Wi-Fi availability or overlay-based UI behaviors) that are not necessary for all adware variants and may change as families adopt alternative communication channels or interaction strategies. Consequently, such signals fail to generalize across families that share similar malicious intents but differ in implementation details.

TABLE VII: Top-10 important features for Airpush and Hiddad families from training (Airpush_tr) and test set (Airpush_te). Hiddad family (Hiddad_te) doesn't exist in the training set. The upper part is from TIF, and the lower part is from DeepDrebin [1]. Features 1-10 are sorted in descending order of importance score.

	Airpush_tr	**Airpush_te**	**Hiddad_te**
1	http://schemas_android_com/apk/res-auto	https://play_google_com/store/	intent_action_USER_PRESENT
2	https://play_google_com/store/	http://schemas_android_com/apk/res-auto	http://schemas_android_com/apk/res-auto
3	http://revmob_com	http://revmob_com	intent_action_PACKAGE_ADDED
4	https://android_revmob_com	https://android_revmob_com	permission_ACCESS_NETWORK_STATE
5	permission_ACCESS_NETWORK_STATE	intent_action_PACKAGE_ADDED	http://revmob_com
6	permission_READ_PHONE_STATE	permission_ACCESS_NETWORK_STATE	https://android_revmob_com
7	permission_ACCESS_FINE_LOCATION	com_google_android_gms_ads_AdActivity	com_google_android_gms_ads_AdActivity
8	intent_action_USER_PRESENT	permission_ACCESS_FINE_LOCATION	com_apperhand_device_android_AndroidSDKProvider
9	intent_action_PACKAGE_ADDED	intent_action_USER_PRESENT	permission_ACCESS_FINE_LOCATION
10	permission_GET_ACCOUNTS	http://xmlpull_org/v1/doc/features_html	getSubscriberId
1	http://schemas_android_com/apk/res-auto	https://play_google_com/store/	http://schemas_android_com/apk/res-auto
2	https://play_google_com/store/	http://schemas_android_com/apk/res-auto	intent_action_USER_PRESENT
3	permission_ACCESS_FINE_LOCATION	android/location/LocationManager;-requestLocationUpdates	intent_action_PACKAGE_ADDED
4	android/accounts/AccountManager;-getAccounts	http://xmlpull_org/v1/doc/features_html	android/location/LocationManager;-requestLocationUpdates
5	http://revmob_com_	permission_ACCESS_FINE_LOCATION	permission_ACCESS_WIFI_STATE
6	permission_GET_ACCOUNTS	https://market_android_com	android/app/ActivityManager;-getRunningTasks
7	permission_READ_HISTORY_BOOKMARKS	intent_action_PACKAGE_ADDED	permission_ACCESS_FINE_LOCATION
8	permission_ACCESS_WIFI_STATE	android/app/ActivityManager;-getRunningTasks	permission_SYSTEM_ALERT_WINDOW
9	https://android_revmob_com	permission_ACCESS_WIFI_STATE	http://xmlpull_org/v1/doc/features_html
10	permission_READ_PHONE_STATE	http://revmob_com	printStackTrace

Overall, TIF emphasizes high-level behavioral prerequisites inherent to the adware category, including intent-based activation and general network capability checks. By abstracting away from concrete API usages, these signals remain largely invariant to family-specific implementations and the evolution of the Android API. In contrast, DeepDrebin relies heavily on implementation-level APIs and permissions, whose semantics and availability are tightly coupled to specific Android versions and runtime conditions. As reflected by their variability across families and time in Table VII, such features constitute brittle indicators under temporal distribution shifts.

VI. Threats to Validity

We discuss several factors that may affect the interpretation of our results: 1) **Internal Validity.** Our proposed framework introduces additional components and training procedures, which may raise concerns regarding implementation complexity and sensitivity to hyper-parameter choices. To mitigate this issue, we evaluate TIF across multiple malware detectors and feature spaces, while keeping training protocols consistent with prior work. The observed performance trends remain consistent across these settings, suggesting that the reported improvements are not tied to a particular model configuration. 2) **Construct Validity.** Malware drift is a complex and multifaceted phenomenon whose underlying causes are largely unobservable. In this work, we do not attempt to explicitly model specific sources of drift. Instead, we adopt temporal partitioning as an agnostic proxy to capture the observable consequence of malware evolution, namely performance degradation over time. While this abstraction may not disentangle individual drift factors, it aligns with common practice in prior malware robustness studies and allows for a controlled and reproducible evaluation setting. 3) **External Validity.** Our evaluation focuses on natural temporal drift in real-world malware datasets. We do not consider adversarially crafted distribution shifts or adaptive attacks, which represent an important but distinct research direction. As such, the conclusions drawn in this work are limited to robustness under naturally occurring temporal evolution, and extending the framework to adversarial drift remains future work.

TABLE VIII: AUT (F1,12m) of the Deepdrebin [1] under different regularization methods.

Test Years	DeepDrebin [1]				
	ERM	Early Stopping	Dropout	$\ell 2$	Ours
2015	0.856	0.833	0.859	0.869	0.928
2016	0.808	0.795	0.818	0.822	0.877
2017	0.742	0.736	0.754	0.759	0.815
2018	0.703	0.701	0.726	0.727	0.794
2019	0.682	0.682	0.713	0.704	0.769
2020	0.664	0.668	0.696	0.681	0.732
2021	0.651	0.657	0.686	0.669	0.717
2022	0.628	0.635	0.658	0.641	0.688
2023	0.611	0.617	0.631	0.619	0.665

A. Comparison to the Regularization Method

ERM-trained models are easy to overfit to unstable features, limiting test set generalization [44]. Regularization methods, such as early stopping, $\ell 2$ regularization, and dropout, help mitigate this by constraining model parameters [45]. Unlike regularization, invariant learning focuses on stable features under drift scenarios. We compare these regularization methods with our invariant learning framework during the whole test phase (Table VIII). For reference, DeepDrebin (ERM) includes no regularization, while DeepDrebin [1] employs dropout with a hyperparameter of 0.2. Table VIII shows dropout improves performance under significant drift, outperforming early stopping and $\ell 2$ regularization, which fails to capture optimal features. Invariant training consistently outperforms, capturing stable, discriminative features that maintain performance across distributions.

VII. Conclusion

Android malware detectors suffer performance degradation due to natural distribution changes caused by malware evolution. We identify learnable invariant patterns among malware samples with similar intent, enabling a drift-stable feature space. To address this, we propose a temporal invariant training framework that organizes samples chronologically and integrates multi-proxy contrastive learning with invariant gradient alignment. Experiments demonstrate improved robustness across feature spaces and drifting scenarios, promoting stable and discriminative representations.

References

[1] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*. Springer, 2017, pp. 62–79.

[2] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, and M. Yang, "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, Oct 2020. [Online]. Available: http://dx.doi.org/10.1145/3372297.3417291

[3] D. Angioni, L. Demetrio, M. Pintor, and B. Biggio, "Robust machine learning for malware detection over time," *arXiv preprint arXiv:2208.04838*, 2022.

[4] Y. Yang, B. Yuan, J. Lou, and Z. Qin, "Scrr: Stable malware detection under unknown deployment environment shift by decoupled spurious correlations filtering," *IEEE Transactions on Dependable and Secure Computing*, 2024.

[5] T. Lu and J. Wang, "Domr: Towards deep open-world malware recognition," *IEEE Transactions on Information Forensics and Security*, 2023.

[6] N. Daoudi, K. Allix, T. F. Bissyandé, and J. Klein, "Guided retraining to enhance the detection of difficult android malware," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 1131–1143.

[7] Y. Chen, Z. Ding, and D. Wagner, "Continuous learning for android malware detection," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1127–1144.

[8] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, and C. Kruegel, "When malware is packin'heat; limits of machine learning classifiers based on static analysis features," in *Network and Distributed System Security Symposium*. Internet Society, 2020.

[9] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[10] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending transcend: Revisiting malware classification in the presence of concept drift," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 805–823.

[11] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "{CADE}: Detecting and explaining concept drift samples for security applications," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2327–2344.

[12] K. Pei, W. Li, Q. Jin, S. Liu, S. Geng, L. Cavallaro, J. Yang, and S. Jana, "Exploiting code symmetries for learning program semantics," in *Forty-first International Conference on Machine Learning (Spotlight)*, 2024. [Online]. Available: https://openreview.net/forum?id=OLvgrLtv6J

[13] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "{TESSERACT}: Eliminating experimental bias in malware classification across space and time," in *28th USENIX security symposium (USENIX Security 19)*, 2019, pp. 729–746.

[14] Z. Chen, Z. Zhang, Z. Kan, L. Yang, J. Cortellazzi, F. Pendlebury, F. Pierazzi, L. Cavallaro, and G. Wang, "Is it overkill? analyzing feature-space concept drift in malware detectors," in *2023 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2023, pp. 21–28.

[15] T. Chow, Z. Kan, L. Linhardt, L. Cavallaro, D. Arp, and F. Pierazzi, "Drift forensics of malware classifiers," in *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, 2023, pp. 197–207.

[16] T. Sun, N. Daoudi, W. Pian, K. Kim, K. Allix, T. F. Bissyande, and J. Klein, "Temporal-incremental learning for android malware detection," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 4, pp. 1–30, 2025.

[17] Z. Kan, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Investigating labelless drift adaptation for malware detection," in *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security*, 2021, pp. 123–134.

[18] Y. Chen, R. Xiong, Z.-M. Ma, and Y. Lan, "When does group invariant learning survive spurious correlations?" *Advances in Neural Information Processing Systems*, vol. 35, pp. 7038–7051, 2022.

[19] M. Srivastava, T. Hashimoto, and P. Liang, "Robustness to spurious correlations via human annotations," *International Conference on Machine Learning,International Conference on Machine Learning*, Jul 2020.

[20] L. Liu, M. Simchowitz, and M. Hardt, "The implicit fairness criterion of unconstrained learning," *arXiv: Learning,arXiv: Learning*, Aug 2018.

[21] M. Yang, Y. Zhang, Z. Fang, Y. Du, F. Liu, J.-F. Ton, J. Wang, and J. Wang, "Invariant learning via probability of sufficient and necessary causes," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[22] Y. Jiang, R. Li, J. Tang, A. Davanian, and H. Yin, "Aomdroid: detecting obfuscation variants of android malware using transfer learning," in *Security and Privacy in Communication Networks: 16th EAI International Conference, SecureComm 2020, Washington, DC, USA, October 21-23, 2020, Proceedings, Part II 16*. Springer, 2020, pp. 242–253.

[23] V. Y. F. Tan, P. L.A., and K. Jagannathan, "Recent advances in concept drift adaptation methods for deep learning," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, Jun 2022. [Online]. Available: http://dx.doi.org/10.24963/ijcai.2022/784

[24] E. Avllazagaj, Z. Zhu, L. Bilge, D. Balzarotti, and T. Dumitras, "When malware changed its mind: An empirical study of variable program behaviors in the real world." *USENIX Security Symposium,USENIX Security Symposium*, Dec 2020.

[25] R. Jordaney, K. Sharad, S. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," *USENIX Security Symposium,USENIX Security Symposium*, Dec 2016.

[26] E. Creager, J.-H. Jacobsen, and R. Zemel, "Environment inference for invariant learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 2189–2200.

[27] D. Krueger, E. Caballero, J.-H. Jacobsen, A. Zhang, J. Binas, D. Zhang, R. Le Priol, and A. Courville, "Out-of-distribution generalization via risk extrapolation (rex)," in *International conference on machine learning*. PMLR, 2021, pp. 5815–5826.

[28] Y. Wald, A. Feder, D. Greenfeld, and U. Shalit, "On calibration and out-of-domain generalization," *Advances in neural information processing systems*, vol. 34, pp. 2215–2227, 2021.

[29] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz, "Invariant risk minimization," *arXiv preprint arXiv:1907.02893*, 2019.

[30] E. Rosenfeld, P. Ravikumar, and A. Risteski, "The risks of invariant risk minimization," *arXiv preprint arXiv:2010.05761*, 2020.

[31] K. Yoshida and H. Naganuma, "Towards understanding variants of invariant risk minimization through the lens of calibration," *arXiv preprint arXiv:2401.17541*, 2024.

[32] K. Ahuja, E. Caballero, D. Zhang, J.-C. Gagnon-Audet, Y. Bengio, I. Mitliagkas, and I. Rish, "Invariance principle meets information bottleneck for out-of-distribution generalization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 3438–3450, 2021.

[33] E. Creager, J.-H. Jacobsen, and R. Zemel, "Environment inference for invariant learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 2189–2200.

[34] J. Liu, Z. Hu, P. Cui, B. Li, and Z.-J. Shen, "Heterogeneous risk minimization." *arXiv: Learning,arXiv: Learning*, May 2021.

[35] D. Teney, E. Abbasnejad, and A. van den Hengel, "Unshuffling data for improved generalization in visual question answering," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 1417–1427.

[36] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2020, pp. 1308–1325. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00073

[37] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck, "Evaluating explanation methods for deep learning in security," in *2020 IEEE european symposium on security and privacy (EuroS&P)*. IEEE, 2020, pp. 158–174.

[38] Y. Chen, W. Huang, K. Zhou, Y. Bian, B. Han, and J. Cheng, "Understanding and improving feature learning for out-of-distribution generalization," *Advances in Neural Information Processing Systems*, vol. 36, 2024.
[39] J. Zhang, D. Lopez-Paz, and L. Bottou, "Rich feature construction for the optimization-generalization dilemma," in *International Conference on Machine Learning*. PMLR, 2022, pp. 26 397–26 411.
[40] M. Hurier, G. Suarez-Tangil, S. K. Dash, T. F. Bissyandé, Y. L. Traon, J. Klein, and L. Cavallaro, "Euphony: harmonious unification of cacophonous anti-virus vendor labels for android malware," in *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 2017, pp. 425–435.
[41] Y. Wu, X. Li, D. Zou, W. Yang, X. Zhang, and H. Jin, "Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 139–150.
[42] M. Chaieb, M. A. Ghorab, and M. A. Saied, "Detecting android malware: From neural embeddings to hands-on validation with bertroid," *arXiv preprint arXiv:2405.03620*, 2024.
[43] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 3319–3328.
[44] J. Kukačka, V. Golkov, and D. Cremers, "Regularization for deep learning: A taxonomy," *arXiv preprint arXiv:1710.10686*, 2017.
[45] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.