Towards Explainable Drift Detection and Early Retrain in ML-based Malware Detection Pipelines

Jayesh Tripathi[1], Heitor Gomes[2], and Marcus Botacin[1]

Texas A&M University (TAMU) - USA
{jtjayesh98,botacin}@tamu.edu
Victoria University of Wellington - NZ
heitor.gomes@vuw.ac.nz

Abstract. The current largest challenge in ML-based malware detection is maintaining high detection rates while samples evolve. Although multiple works have proposed drift detectors and retraining-aware pipelines that work with reasonable efficiency, none of these detectors and pipelines are currently explainable, which limits our understanding of the threats' evolution and the detector's efficiency. Despite previous works that presented taxonomies of concept drift events, no practical solution for explainable drift detection in malware pipelines existed until this work. Our insight to change this scenario is to split the classifier knowledge into two: (1) the knowledge about the frontier between Malware (M) and Goodware (G); and (2) the knowledge about the concept of the (M and G) classes. Thus, we can understand whether the concept or the classification frontier changed by measuring the variations in these two domains. We make this approach practical by deploying a pipeline with meta-classifiers to measure these sub-classes of the main malware detector. We demonstrate via 5K+ experiment runs the viability of our solution by (1) illustrating how it explains every drift point of the DREBIN and AndroZoo datasets and (2) how an explainable drift detector makes online retraining to achieve higher rates and requires fewer retraining points.

Keywords: Malware Detection · Concept Drift · Explainable AI
This is the public author version.

1 Introduction

Computer systems are targeted daily by a myriad of malware samples, such that we cannot keep up with this high volume of attacks without the help of automated tools, which are currently largely based on Machine Learning (ML). Although ML can be very effective, designing and deploying an efficient ML pipeline for malware detection has many challenges, as extensively discussed in related works [10]. The main challenge for ML-based malware detection nowadays is the degradation that the models face due to the attacker's new Tools, Tactics, and Procedures (TTPs), which require frequent classifier retrains. Constantly retraining the classifier is not efficient; thus, precise triggers for the retraining process are warranted. Currently, drift detection algorithms are the best triggers for a retraining task. Drift detectors work by observing the distribution of the predicted labels in comparison to ground-truth labels (e.g., provided by malware sandboxes or human analysts) and report significant growths in the prediction error rate, which is used as a proxy for the distribution change.

The drawback of current drift detection algorithms is that, by the nature of their design, they do not explain the causes of the drift but just report the drift detection

based on the observed error rates. Although it is enough to keep the pipeline operating at reasonable rates, this limits other applications, such as benchmarking and dataset characterization. Benchmarking is limited because it is impossible to know what the drift detector is recognizing as different in a stream. Therefore, it is hard to have a ground truth for eventual false positive reports. It also makes it hard to fairly compare two drift detectors as they detect different phenomena. In this scenario, identifying the best drift detectors for a given scenario has become a trial-and-error task rather than a scientific task. For similar reasons, the nonexplainability also limits the understanding of real scenarios, which limits incident response. Even if a drift detector points out that an in-the-wild stream of malware has changed its distribution, analysts cannot know what is present in the new samples that was not present in the previous ones, which limits remediation and prevention actions.

Ideally, Drift-aware pipelines should provide a clear view of what has changed from one period to another. Recent works in drift detection took their first steps in this direction. A proposed pipeline [11] innovates by retraining not only the classifiers but also the feature extractors when drift is detected. Thus, by comparing the vocabulary in different epochs, it is possible to know the malware features that gained and lost importance. This technique was used to identify, for example, when SMS sending permissions became less prevalent in Android malware because the OS stopped allowing apps with this permission in the app store. Despite this advance, this work still did not explain how the feature changes caused the models to change. A step in this direction was given by works that tried to model probabilistically and statistically the chances of a sample belonging to a distribution or not [23]. Again, despite being a significant advance, this approach still provides an incomplete treatment of the drift problem, even after its complement [6], because although identifying the best retraining moments, it still does not explain the drift points and changing features. Thus, the current scenario is: *Drift detectors that explain features do not explain models and vice-versa*.

We propose changing this scenario with a **practical** solution to explain drift events. This solution is based on the key insight that the classification process is composed of two different aspects: **(1)** the frontier between the classes and **(2)** the concept of the classes. This differentiation highlights the fact that sometimes drift is reported because **(i)** the frontier is misplaced and sometimes because **(ii)** the concept itself changed. We propose that if we measure these two aspects, we can explain drift events by assigning them to one of these two classes (frontier or concept change).

Unlike conformal evaluation [6], which also breaks down the problem in two, we do not rely on explicitly math modeling (e.g., via averages and variances) the problem, but on using their own ML classifiers (meta-models) for the task. This allows one to deploy our solution using the same ordinary drift detectors used in the main classifier (e.g., DDM, ADWIN, and so on) while still benefiting from the non-linearity of ML models to learn the concepts. We propose that detecting drift in the meta-models exposes the change in the frontier and the concepts. To test our hypothesis, we deploy the classifiers under test with a main drift detector in addition to **(1)** drift detectors in their classes, and **(2)** drift detectors in a second layer deployment of one-class versions of the main classifiers, specifically trained to recognize the concepts known by the main classifier.

We tested our solution via 5K+ experiments with the DREBIN and Androzoo datasets with different settings (e.g., detectors, imbalances, policies, etc.), and we found that:

- Our approach explains all classification points, including true drift points, but also false positive drift detections and bad frontiers caused by limited training data.
- Explaining detection and retraining are the two faces of the same coin, as recognizing true concept changes allows making online retraining procedures faster (early retrain), more effective (achieving higher rates), and more efficient (requiring fewer retrains) compared to traditional retraining.

In sum, our contributions are as follows.

- Proposing viewing concept drift as two separate problems: one is establishing the decision boundary, and another is learning the concept for the classes.
- Proposing using meta-classifiers as practical solutions to measure changes in the decision boundaries and the in-class concepts to explain drift detection decisions and suggesting early retraining.
- Evaluating our proposals on the DREBIN and AndroZoo datasets to demonstrate the practical viability of explaining concept drift events in malware detectors.

2 Analyzing Concept Drift

Understanding Classification Errors. When we train a binary classifier for a task like separating malware and goodware, we teach the classifier to divide the feature space with a decision frontier. The samples on one side of the frontier are assigned to one class, and the same is true for the other class. Figure 1 exemplifies it with a horizontal frontier in a 2D feature space for didactic purposes. It is key to note that the classifier might not learn the actual concept of the samples belonging to each class, i.e., what they have in common (the circle), but just a frontier that separates one from another.

In the ideal scenario, the frontier should adapt perfectly to the concept, but this is not what often happens in practice. This gap opens a space for positive and negative effects. On the positive side, some newer samples might still be classified in the right class even if they do not match the learned concept. To that, it is enough to be farther to the wrong class frontier. On the negative side, False Positives (FPs) might occur if the samples still belong to the same original concept but the initial frontier was improperly defined. This happens. For example, when the sampling process to define the training set was biased or did not consider enough representative samples. Figure 2 illustrates this case. Although the newer samples still belong to the same concept (the wider circle), they are flagged in the wrong class because the frontier is improperly defined. Ideally, the classifier should learn the actual concept to avoid these cases. In practice, the establishment of a new frontier (the diagonal line) is what is achieved with classifier retraining.

If the classifier learned the concept, FPs due to an improper frontier would not happen, but the classifiers would still make an increasing number of incorrect predictions over time. This is caused by the concept drift/evolution effects, i.e., the malware samples changing their characteristics significantly to avoid detection such that the learned concepts do not apply anymore. In these events, the newer samples tend to slowly deviate from the learned concept to multiple directions, as illustrated in Figure 3. When the samples drift towards the frontier, they might cross the boundaries and cause FPs.

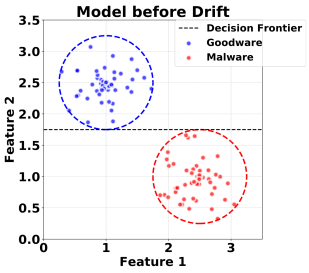

Fig. 1: **Initial Training.**

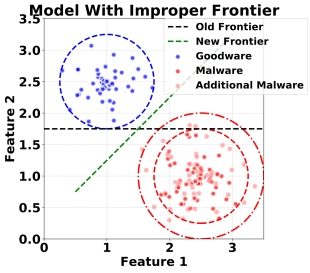

Fig. 2: **Additional Data.**

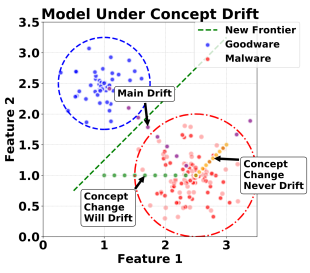

Fig. 3: **Multiple Drifts.**

Detecting vs. Explaining Drift. Constantly retraining the classifier to change the frontier is costly and inefficient. Thus, in practice, MLSec operators benefit from the loose coupling between concept and frontier to let systems operate while producing correct results–even outside the borders of the ideal concept–and only retrain the classifier when the frontier is affecting the correctness of results. The moment to change the frontier is indicated by drift detectors. The major limitation of this approach is that current drift detection algorithms and architecture do not explain their result, i.e., they do not draw a big picture of the phenomenon that is happening, as previously discussed. By observing only the main result, current solutions can't even tell if the result is wrong because the frontier is misplaced or because an actual concept change happened. Achieving greater explainability is possible by observing the multiple drift scenarios and mitigating the detector's blind spots, as follows.

Traditional drift detectors operate by only measuring the error rate in the classifier frontier. The underlying idea is that if the concept changes, more samples will cross the frontier, and thus, the error rate will increase. We can interpret this type of detector operation as measuring if the samples of our example are crossing the horizontal frontier from Figure 1 to Figure 2. The **first** blind spot is that samples cross the frontier not only when the concept changes but also when the frontier itself is misplaced. By only looking at the frontier, it is not possible to tell the cause of the drift alert. The **second** blind spot is that this type of detector is only triggered when the result is already affected, and this happens at a late drift stage. The concept might have started to change early and remained in the gap between the concept and the frontier before crossing it. In this case, the classifier did not benefit from this situation to start proactively retraining the classifier. The **third** blind spot is that drift detectors based on the frontier might be affected by class imbalance (depending on their internal construction, but we will remain agnostic here). If the number of samples crossing the frontier is relatively small, it is not "worthy" retraining the classifier. However, this decision is problematic in very imbalanced scenarios, where the majority class might always be correctly predicted, whereas the minority one might always be wrong. The **fourth** blind spot is that, due to practical storage and processing limits, the drift detectors do not observe the entire history of predictions but only a window, which makes them susceptible to FPs. If a sequence of wrong predictions (e.g., a few from each class) sequentially appears by simple randomness, the drift alarm will be triggered. Current drift detectors have no way to identify their own FPs. The **fifth** blind spot is that traditional drift detectors do not differentiate which classes are drifting but only observe if the frontier was crossed by

a significant amount of samples. Therefore, many drift detectors are not even able to explain which class is problematic.

Class-aware drift detectors is a potential solution for mitigating the blind spot of not telling which class crossed the line. By measuring the error rate in each class, it is possible to know which classes (malware, goodware, or both) have their samples crossed the lines. In the didactic example from Figure 2, the malware class is causing the drift identification event. It also solves the blind spot of not identifying the FP cases because if none of the classes have their samples crossing the frontier, an eventual drift identification event can only be caused by a rare sequence of wrong predictions within the limited window. However, since it keeps monitoring only the decision frontier, it does not solve the blind spot of only detecting the drift at a late stage, and it is not able to explain if the frontier was misplaced or if the concept actually changed.

Concept-Aware Detectors is our proposal for an ideal model of a detector that learns the concept of the samples (the circles in Figure 3) independently of the frontier. In this model, the samples are not only assigned to classes but also classified as belonging or not belonging to known concepts. This way, it is possible to know if a sample is part of a known concept regardless of where the frontier is placed, which allows differentiating the reasons for drift. If a drift detector is placed in the concept detector, we can identify if the concept actually changed. If there is no drift in the (ideal) concept classifier but a drift in the (real-world) main classifier, it means that the initial frontier was misplaced.

The concept-aware drift detector acts in tandem with the main classifying by helping to explain its drift and non-drift events. This interaction also allows for anticipating eventual drift occurrences. When the concept is actually changing, three cases might be identified, as illustrated in Figure 3: **first**, the concept might change in a direction that does not go towards the frontier (the parallel line to the frontier in Figure 3). In this case, the detection benefits from the concept-frontier gap to keep classifying the new samples correctly without the need for a retrain; **Second**, the new concept goes towards the frontier, but it has not crossed yet (the horizontal drift line in Figure 3). In this case, the classifier would benefit from an early retrain, as crossing the frontier is imminent; **Third**, the new concept went towards the frontier and crossed it (the diagonal drift line in Figure 3), which is the explanation for a true drift detection.

Based on the above discussion, we propose the following taxonomy on the type of information that the drift detection algorithm and/or architecture can provide and the case it identifies:

- **Type 1:** Main Classifier Drift. It detects whether a significant number of samples of any class crossed the detection frontier or not within a sampling window to the point of already harming the final classification result.
- **Type 2:** Sub-Class Drift. It detects whether a significant number of samples of a specific class crossed the detection frontier or not within a sampling window to the point of being noticeable but without guarantees that it affects the final classification result (contingent upon Type 1 detection).
- **Type 3:** Concept Change. It detects if a significant number of samples of a specific class do not match the previous knowledge the classifier had about that class, regardless of the correct class assignment (Type 1 and 2 events). The implications of the concept change causing drift or not are contingent on the following cases:

- **Case A:** Concept change without drift risk. If the concept changes in a direction that does not go toward the decision frontier, it cannot cause drift events.
- **Case B:** Concept change with imminent drift risk. If the concept changes towards the decision frontier (Type 2), it will eventually cause drift when crossing the frontier (Type 1). This point is a candidate for early retraining.
- **Case C:** Current Drift due to concept change. If the concept changes towards the frontier (Type 2) and crosses it (Type 1), concept drift is detected late.

These types of detectors are cumulative, i.e., a detector type 3 implies the deployment of type 1 and type 2 detectors. Type 3 detectors currently do not exist. The development of a practical type 3 architecture is this work's goal, as following detailed.

A practical solution for explaining drift events. A first idea for improving the explainability of drift detectors is to redesign the drift detection algorithms to be class- and concept-aware. This, however, imposes limits on the types of algorithms that can be used and significantly impacts the design of the malware detectors that become coupled to the used drift detectors. We aim to remain agnostic to the type of drift detection algorithm, and instead of algorithm redesign, we propose an external monitoring architecture that implements the class and concept-aware concepts independently of the detectors used.

We propose a 2-layer architecture where the first extends the Type 1 drift detectors already existing in current malware detection architectures to become Type 2 by instantiating copies of the main drift detector and directly linking them to the prediction of each class. The second layer consists of replicating the main classifier in 1-class settings to focus them on learning the class concept rather than the decision frontier, thus turning them into Type 3 detectors.

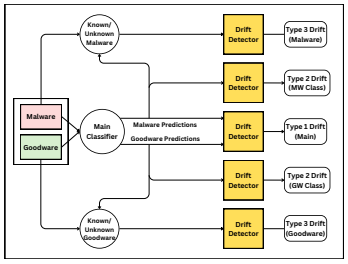

Fig. 4: **Drift-Explainable Architecture.**

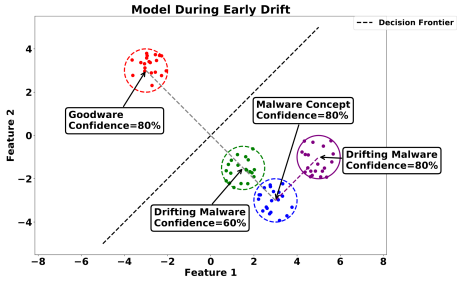

Fig. 5: **Direction-Change Drift Detection.**

In the proposed architecture, illustrated by Figure 4, the training process starts as usual, with malware and goodware ground-truth samples being provided to the main classifier (classifier under test). However, an additional step is introduced to train the second-layer classifier. To teach the classifier the concept of malware, for instance, the labels correctly predicted as malware after the training of the main classifier as provided as positive labels to the secondary classifier (*Known Malware*, classified correctly). The malware samples misclassified by the main classifier as goodware are provided to the second layer classifier as *Unknown Malware* samples. Thus, this second-layer classifier learns what the main classifier knows as malware and not its difference to goodware. The same process is repeated to train the goodware class. The prediction step also starts as usual, with the unknown sample being classified by the main classifier, whose predictions are later checked by the main drift detector. However, the labels of

each class are also forwarded to individual detectors. The sample is also classified in parallel by the second layer classifier, having its ground-truth class as a reference, to identify if it fits the previous classifier knowledge about that class, independently of the main classifier prediction.

The proposed architecture is capable of detecting Type 3 drift, which is enough to point out concept change events, but it is still not enough to anticipate concept drift detection occurrences. to that, an additional step is required to understand the direction of the concept change. For a practical implementation, we approximate the identification of a drift direction by the Equation 1: $Avg_{concept} - Avg_{drift} > k, \quad \text{given } k > 0$

The rationale for that is that the confidence level of the main classifier tells how close the samples are to the frontier. The lower the confidence, the closer to the frontier. If the concept is changing towards a lower confidence region, it is going closer to the frontier, and it is likely to cause a concept drift alarm. We here identify if the concept is going towards the frontier via the average confidence of the new concept cluster compared to the average confidence of the original concept cluster. It is key to highlight the difference between this and previous work's approach, as we do not simply trust in the overall change in the confidence level for all samples, but we rely on the second-layer model to cluster the samples whose confidence will be averaged.

Concept changes in different directions are illustrated in Figure 5, which shows two concepts (malware and goodware) originally positioned at 80% confidence level. When the concept changes laterally, over the same diagonal line, it triggers a Type 3 detection, but the new concept is not a candidate for early retrain, as it does not go toward the frontier, which is indicated by the fact that the average confidence remained the same (it is the same in all points over the diagonal line parallel to the main frontier). In turn, if the concept changes towards the perpendicular line to the frontier, it will trigger a Type 3 alarm, and it is a candidate for early retrain, as it tends to cross the frontier, which is indicated by the reduced confidence level (60%).

The relation between the drift event types in the classification stream. During the operation and/or the evaluation of a malware detection pipeline, the multiple drift detectors of our proposed architecture will trigger simultaneously, depending on the drift event that is happening. We summarized all valid states in Table 1, based on the previously introduced drift taxonomy.

The simplest situations are the ones when the detectors agree. When no drift detector fires, the operation is normal. When all detectors fire, there is a clear drift caused by concept change. The most challenging cases are when the detectors disagree. If only the Type 3 drift detector fires, it indicates an early drift case. Then, we need to check if the change is towards the frontier (Type 1) or not, if wanting to decide on an early retrain. If the Type 2 detector fires without a Type 3 one firing, this indicates that the frontier is the problem. If the Type 1 fires without the others, this indicates a false positive. Some detector-state combinations are not possible. For instance, if only the Type 3 detector fires, it can only be in case 1 or 2, but never in case 3 (crossing the frontier), as it requires the Type 1 drift to also fire. These states are our falseability points. We used them to validate our approach. If these cases appear in our test, this indicates that our hypothesis/theory does not hold.

Table 1: **Explaining Drift Events.** Information types for each combination of triggered detectors. Representing Triggered Detectors (✓) and Possible (△) and Not-Applicable (Ø) cases. Omitting Impossible cases.

Main Type			Cases			Conclusion
Type 1	Type 2	Type 3	Case A	Case B	Case C	
				Ø		Normal Operation
		✓	△			Early Concept Change with no impact on frontier
		✓		△		Early Concept Change with imminent impact on frontier
	✓			Ø		Bad Frontier detected without concept change hold by imbalance in main class
	✓	✓		△		Bad Frontier detected with concept change hold by imbalance in main class
✓				Ø		False Positive Drift Detection
✓		✓	△			False Positive with concept change in non-impactful direction
✓	✓			Ø		Bad Frontier detected without concept change, with impact in the main class
✓	✓	✓			△	Concept Change with Immediate Impact and Identification

3 Evaluation

3.1 Drift Detection Explained by Examples

Consider the Android malware detection case a representative example of the malware detection problem. For this demonstration, we considered the DREBIN [4] dataset, given its popularity and known drift cases. We also considered a simple model that classifies the APK's permissions by modeling them as a binary vector (as in DREBIN), which allows us to easily understand when features entered and left the concept, thus highlighting the concept change phenomenon. Whereas we initially enumerated all permissions in the dataset to create a feature vector of the ideal size, the feature vector is only filled on demand as the permissions appear in the data stream. In our experiment, the dataset was temporally ordered, as recommended by the best practices in the field [10, 25] to avoid the data snooping pitfall [3]. Since DREBIN is very imbalanced [14] and could affect the drift results, we undersampled each temporal bin to the 50-50 proportion. Our goal in this demonstration is not to perform a real-world characterization (which is done in the next step) but to highlight the studied drift events.

Observing drift in the main classifier output. Drift is an attacker-induced phenomenon that can be noticed in any reasonably long temporal stream observation. Figure 6 illustrates this phenomenon for the evaluated DREBIN dataset setting. It shows the curves derived from the initial training with samples from the first 3, 5, and 7 bins (epochs). In all cases, the accuracy significantly decreased, from more than 90% to less than 80%.

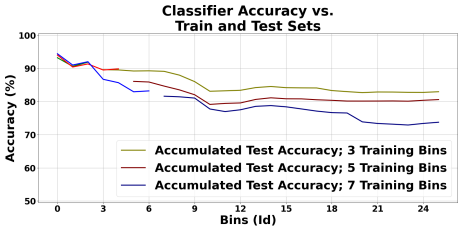

Fig. 6: **Concept drift in practice.** The classification accuracy decreases regardless of the initial training set size/period.

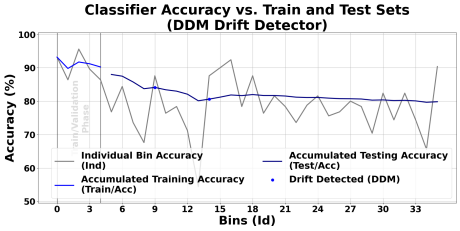

Fig. 7: **Drift tendency vs. instantaneous detection.** Drift points reported by the ADWIN algorithm.

Concept drift is a temporal trend, but drift detection algorithms observe the instantaneous error rate. When the drift effect significantly influences the classifier output, it is detected by a drift detection algorithm. Figure 7 shows how the Drift Detection

Algorithm (DDM) [16] detects multiple drift points in our tested DREBIN setting. It is key to notice that the drift effect is a tendency, which is shown by the curve of the accumulated accuracy results. The drift detection, however, happens in the samples received within a window (the bins), which exhibit higher variations in the instantaneous accuracy values. In this work, we measure drift in the instantaneous results, but we display them mapped to the accumulated ones to highlight the tendencies.

Different drift detectors detect different drift points. In addition to DDM, we also tested the EDDM [5] and ADWIN [9] drift detectors in multiple settings. We consider the policies of **(i)** never resetting the detector in the entire stream; **(ii)** resetting upon detections; and **(iii)** resetting at every epoch. Figure 8 illustrates how each detection algorithms detect a different number of points and at different epochs according to their internal working and parameters. We notice that although the stream is the same, the first time each detectors identify the drift occurrence is different and depends on a different policy. This characteristic of current drift detectors makes it hard to compare them and to explain their detections. Although we tested all settings in all experiments, in the rest of this paper, we only show results for the detector that presented the best results.

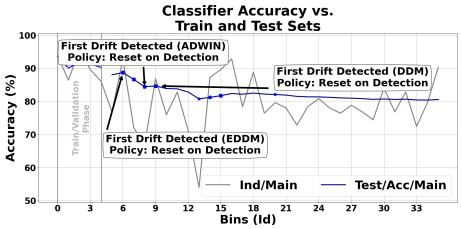

Fig. 8: **Comparing algorithms and policies.** Each one detects a different number of drift points/events and at different times.

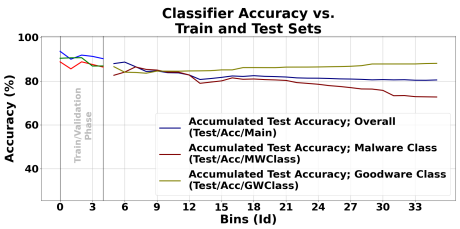

Fig. 9: **Separating detection rates per class** reveals that the drift in the MW class causes the global performance degradation.

Class-aware detectors provide an initial explanation level. The previously presented results with Type 1 drift detectors do not allow identifying which class caused the drift identification. By adopting a Type 2 detector, we can easily identify it. Figure 9 shows the overall and per-class accuracies in the tested DREBIN settings. It is clear that the classifier detection rate is the average of two cases: **(i)** higher goodware and **(ii)** lower malware detection rates. It shows both that: **(i)** goodware is easier to classify than malware; and **(ii)** the drift in the malware is responsible for the classifier performance degradation.

Sub-classes drift differently than the main classifier. We proposed not only to measure the accuracy of individual classes but actually detect drift in the subclasses. Figure 10 shows that each class presents a different number of drift points and that the drifts in the main classifiers are not simply the sum of the drifts in the classes. It also shows that the first drift point for each sub-class is identified at different moments. This happens when the same classifier is applied to all classes, which reinforces that each class has its own dynamics that should be individually monitored and understood.

Concepts are really different than frontiers. In the previous experiments, we reached the limit of the Type 2 drift detector. It explained the classes that caused drift and the moments each one drifted, but it still does not explain the reason. To achieve that, we introduced the Type 3 detector with the concept of the classes. The remaining question

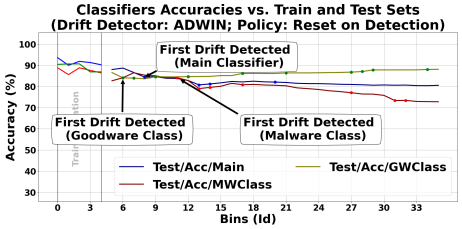

Fig. 10: **Main class vs. sub-classes.** A different number of drift points is identified in each class and at different epochs.

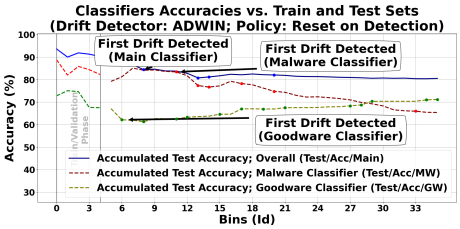

Fig. 11: **Drift in the classes self-recognition rates.** Drifts are represented both for the MW and GW meta-classifiers.

was if the 2-nd layer classifier would learn something different than the main classifier. Figure 12 and Figure 13 shows the accuracy for the self-recognition of the goodware and malware classes, respectively. It is possible to see both that: **(i)** the dynamics of these classifiers are different from the individual classes of the main classifier (Figure 9); and **(ii)** each one of the classes has its own dynamics. The observed results are compatible with the overall results that the malware class is causing drift because the classifier is losing its ability to recognize malware over time.

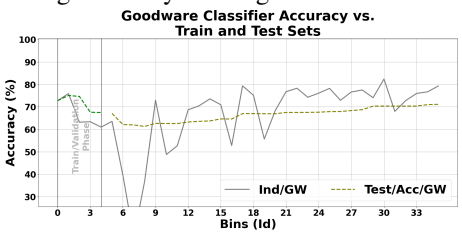

Fig. 12: **GW class self-recognition rate.**

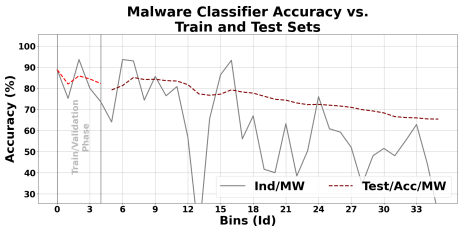

Fig. 13: **MW class self-recognition rate.**

A drift in the concept is really different than a drift in the frontier. A similar concern was if the concepts would drift and if these drift points would provide additional information than the drift points in the main classes. Figure 11 shows the drift points for the subclasses in comparison to the drift in the main classifier. It is possible to see that, once again, the drift dynamics for each concept/class are different among themselves and from the main classifier. The first time each concept drifts is noticeably different. Moreover, compared with the drift points for the subclasses of the main classifier (figure 10), the drift points are significantly different, which confirms the hypothesis that the second-layer classifier provides a different type of information about the drift events, which enables explaining them.

A Type-3 detector enables explaining the entire classifier operation and all drift events. Our key proposition is that looking at all types of drift detectors, we could explain the drift events, which was achieved in practice. Figure 14 shows the explanation of all points based on the detector's information combined as in Table 1, except for normal operation points, when nothing is plotted. It explains that the initial detection drop is due to the misplaced frontier and not because of actual concept change. This happens because the DREBIN dataset has an initial distribution very imbalanced towards goodware, and our artificial undersampling approach introduced a limited learning of the decision boundary. In turn, it further points out that later drift points are caused by

real concept changes, which is indicated by the drift in the concept class. In all cases, the drift point in the main classifier is explained by a drift point also in another curve. No impossible case was observed, thus confirming the approach's correctness.

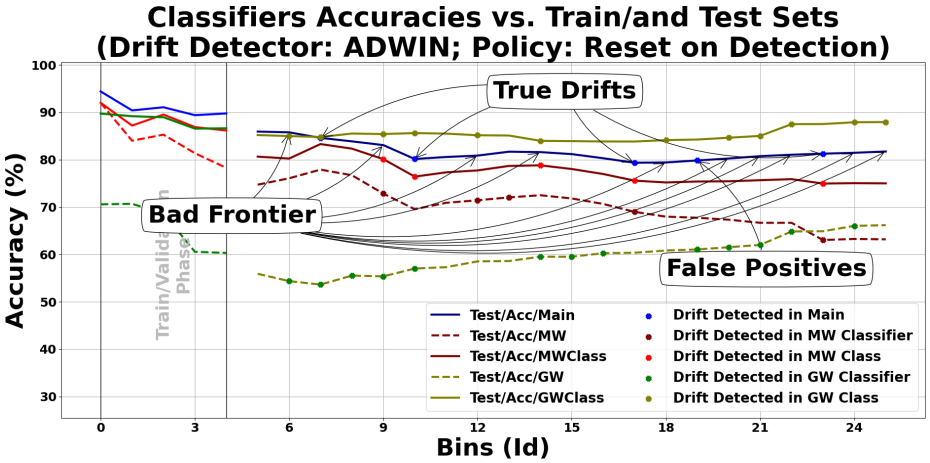

Fig. 14: **Explaining all operational points and all drift occurrences.** Omitting points of normal operation.

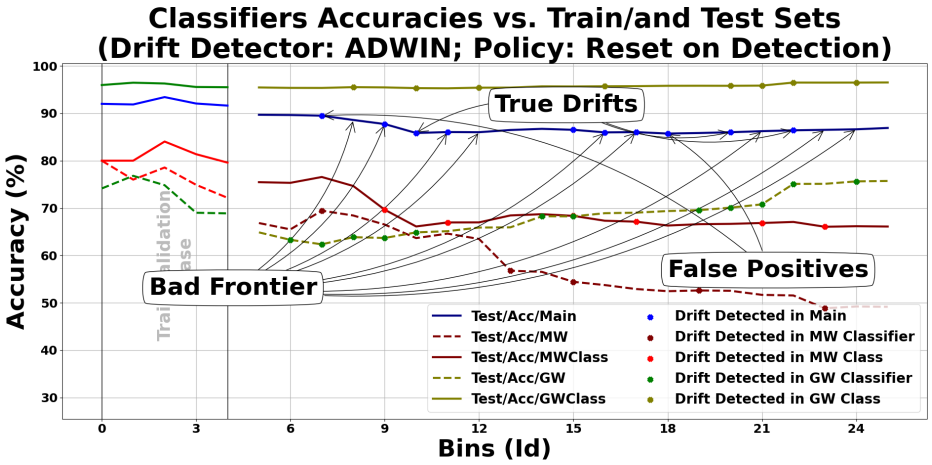

Fig. 15: **Explaining all drift detection points (2:1 balance).** We observe fewer frontier problems and more False Positives.

Type-3 drift detectors reveal the impact of imbalanced datasets in the drift detection. We attributed the previously identified frontier problems to the limited learning ability caused by the undersampling procedure. If our theory is correct, increasing the learning ability should result in fewer frontier problems being detected by our solution. We put this hypothesis to the test by increasing the proportion of goodware to malware (dataset balance) in an experiment.

Figure 15 shows the results for the 2:1 dataset imbalance. Our solution explained all drift and non-drift points, as expected. It also reported fewer frontier problems

in this scenario, thus confirming our hypothesis that Type-3 drift detectors allow the identification of not only real drift cases but also frontier problems.

Type-3 drift detectors reveal true False Positives in Type-1 drift detectors. A phenomenon first observed in the previous experiment, when increasing the dataset imbalance, is that a drawback of having a more defined frontier causes False Positives (FP) to appear. In fact, we further discovered that the more we increase the imbalance–from fully balanced (1:1) to the natural imbalanced (N:1)–the more FPs are observed, as reported in Figure 16.

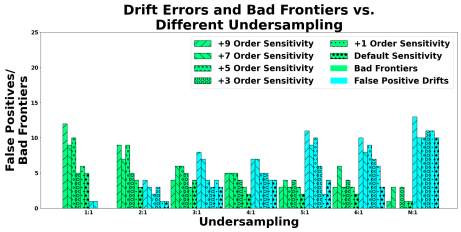

Fig. 16: **Drift Detector Calibration.** False Positives and bad frontiers are explained by the proposed approach.

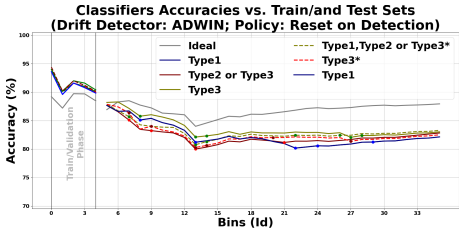

Fig. 17: **Early retraining on concept changes** leads to improved accuracy than retraining only upon main class drift.

We claim that our Type-3 solution can identify FPs in the Type-1 drift detectors. If our theory is correct and the reported points are true false positives, calibrating the drift detectors' sensitivity should decrease the FPs without affecting the frontier reports. In turn, if our theory is wrong and the reported FPs are true drift points, decreasing the drift detector sensibility should lead to more frontier problems as the samples evolve. We put our theory to the test by conducting experiments with different dataset imbalances. Figure 16 summarizes the results for the number of frontier problems and reported FPs. They confirm our hypothesis that our Type-3 solution reported true FPs. As we calibrated the drift detector sensitivity in different orders of magnitude, fewer FPs were reported. Moreover, the frontier problems decreased the more data the model had to learn.

Explainability allows early retrain. The previously presented results are based on the continuous run of the initial model. In most pipelines, however, the model is retrained upon the drift alarms. Our main goal in this work was not to make drift detectors faster or more precise but more explainable. However, if you have a better situational awareness–i.e., you know what is happening, thus when concepts are changing–you have a better trigger for retraining procedures. Therefore, we conclude that explainability and precise early retraining are the two faces of the same coin.

This fact is clearly illustrated in Figure 17, which shows the achieved accuracy when retraining models upon the trigger of the best and worst combinations of drift alarms (only Type 1, 2, or 3, or all combinations of Type 1, 2, and 3). The results are compared to the ideal case where the model was trained with the entire dataset, and no drift exists by definition/calibration. Retraining has a positive effect in all cases compared to not retraining. However, the best results are achieved when retraining upon Type 3 drift detectors and not when retraining upon a drift in the main classifier. This confirms the hypothesis that Type 3 detectors can identify concept change before the samples cross the decision boundary (Type 1 and 2), when it is already harming the detection results.

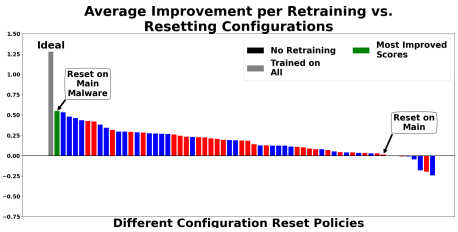

Fig. 18: **Retrain Effectiveness.** The vast majority of the proposed drift detection triggers lead to increased accuracy gains than the original Type 1 drift detector trigger.

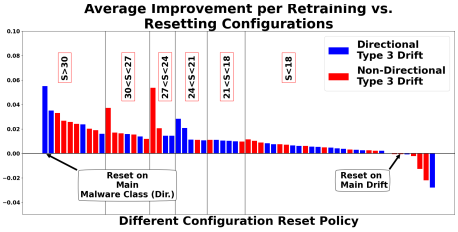

Fig. 19: **Retrain Efficiency.** The best cost-benefit between the amount of retrains and accuracy increase is achieved by identifying concept changes.

Figure 18 presents the distribution of accuracy gains for all combinations of retrain triggers. It is noticeable that retraining upon a drift identification in the vast majority of the proposed Type 2 and Type 3 detectors leads to increased gains than only retraining upon a Type 1 drift detection event.

Explainability makes retraining more efficient. Retraining procedures should be efficient in addition to effective, i.e., they should not only increase the overall detection rate but do it without spending excessive resources. A procedure that retrains the classifier every epoch is effective but not resource-efficient. We evaluated the efficiency of all retraining policies (the set of drift detectors that trigger a retrain) via their cost-benefit, i.e., how much they increase the total accuracy compared to the number of triggered retrains. Figure 18 demonstrated that the proposed directional approach was the most effective, but would it also be the most efficient?

Figure 19 shows the cost-benefit distribution for the multiple training strategies, i.e., their average accuracy increase per number of retrains. The results are broken down by their 10% increase intervals. This is required because, whereas a lower number of retrains is on average very efficient, it does not allow scaling the accuracy at higher levels. In turn, to raise the accuracy to cover all corner conditions, a series of additional retrains steps are required, which naturally decreases the efficiency. This is known as the 80:20 distribution or Pareto problem–when most of the gains come initially and a series of extra steps are required for the additional minor gains. This phenomenon is observed by the fact that the most efficient strategy is a non-directional drift strategy, but it did not achieve the highest increase levels. In turn, our proposed direction-aware retraining strategy was not only the most effective one but also the most efficient. Overall, directional strategies were the most effective for 3 out of the 5 ranges of accuracy gains.

3.2 Results Generalization

After showcasing our solution's potential via the above examples, we extended our analysis to evaluate how it generalizes. To do so, we repeated the previous experiments in a wide range of settings (e.g., different drift detectors, drift detection policies, and dataset imbalances) and assessed the outcomes' coherence. In total, we performed 5000 different runs of our solution over the DREBIN dataset. No run produced an "impossible" case, thus reinforcing our claims on the approach's correctness. Overall, the obtained large-scale results corroborated the previous experiments' results, which allows to explain the different effects datasets are subject to.

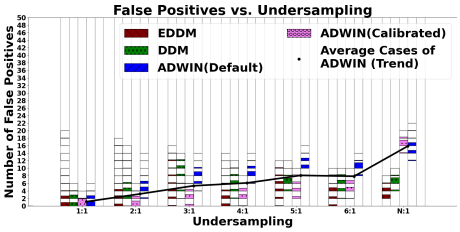

Fig. 20: **DREBIN: FP Results Distribution for different imbalances.** FPs grow with the imbalance for most detectors.

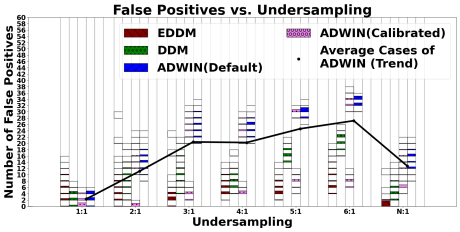

Fig. 21: **ANDROZOO: FP Results Distribution for different imbalances.** FPs grow with the imbalance for most detectors.

False Positives identification at scale. Figure 20 shows the FP distribution of different detector settings (e.g., detection policies) for different imbalances. For each possible FP value (y-axis), the stacked bars show the proportion of experiments that presented (hatched) or not (empty) that FP rate. The bars are stacked, covering all possible FP rates. We filled the bar representing the setting with the highest proportion to highlight patterns. In the case of ties, all settings were highlighted. We broke down the results by detectors.

The most prevalent results in each column shows that in general the more imbalanced the dataset, the more FPs happen, as previously showcased for the individual case. This happens for all detectors, but it is more pronounced for the default configuration of ADWIN (4th column). ADWIN calibration mitigates the problem (column 3) for most configurations (remember that each bar represents hundreds of runs). Retraining on drift detection (last column) also significantly mitigates the problem, but via a different mechanism: not because it is calibrated, but because of its continuous adjustments.

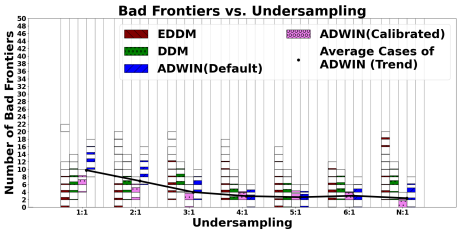

Fig. 22: **DREBIN: Detectors' Results Distribution.** Frontier problems for different undersamplings. The bigger the undersampling, the more frontier problems.

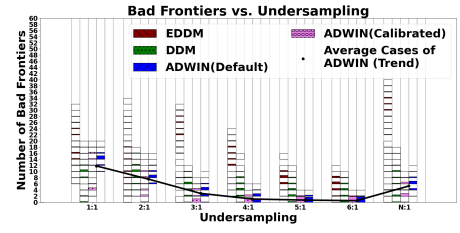

Fig. 23: **ANDROZOO: Detectors' Results Distribution.** Frontier problems for different undersamplings. The bigger the undersampling, the more frontier problems.

Bad Frontiers identification at scale. Figure 22 shows the distribution of bad frontier points reported by our solution in different dataset imbalances. In the overall case, as well as for the particular case previously demonstrated, the more undersampled the dataset is from the original distribution, the more frontier problems appear due to the limited data available to learn from. This effect is observed for all drift detection settings,

and it is mitigated by calibration and/or retraining. This result shows not only that the effect is real but also that our solution can explain it.

Retraining. Once we have demonstrated how drift events are explained in the general case by our proposed solution, we shift our attention to remonstrating how an explainable drift detection also favors more effective and efficient retraining processes. We computed the retraining statistics for the thousands of runs previously presented and analyzed the behavior of different drift detection strategies.

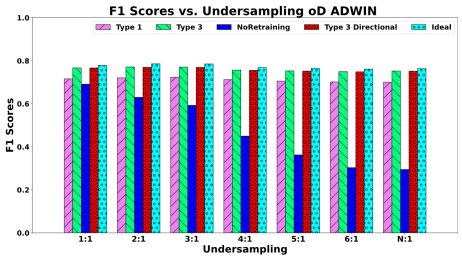

Fig. 24: **DREBIN: Average Retraining Results.** Average F1-score Under Time increase over the baseline when triggering retrains using different policies vs. the multiple dataset imbalances.

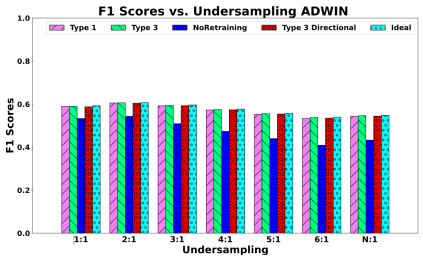

Fig. 25: **ANDROZOO: Average Retraining Results.** The imbalance effect is less pronounced in this dataset, but the Type-3 retraining strategy is still the superior one in all scenarios.

Figure 24 shows the average F1-score for the multiple detectors configurations in the tests with multiple, different dataset imbalances. It is clear that performing no retraining (i.e., having no drift detection) is the worst case of all. The greater the dataset imbalance, the more noticeable the impact becomes as more samples are considered, thus accumulating the effect of wrong predictions. In this sense, using even a traditional (Type 1) drift detector is a good mitigation strategy. Class-aware (Type 2) drift detectors present gains over Type 1 because detecting when one class is drifting instead of waiting for the impact in the two classes at the same time allows early retraining, so the classifier spends less time making wrong predictions. In this same line, our proposed explainable drift detector (Type 3) presents even greater performance gains, as it allows early retraining when the concepts change, which reduces even more the window in which the detector makes wrong predictions. For all imbalances, our solution was the one that got closer to the ideal scenario–where retraining was not needed because it was trained with the complete dataset, thus not presenting drift by default.

Androzoo Generalization. While the imbalances and size of the DREBIN dataset highlight multiple positive aspects of our proposed drift explanation, they could also potentially hide negative aspects. Therefore, we repeated the previous experiments with Androzoo, a much larger and more balanced dataset, to demonstrate that our solution also operates properly under these different biases. As for the DREBIN dataset, our proposed solution explained all event points for the Androzoo one, and no invalid point was observed. More than that, the trends observed for the DREBIN dataset were also

observed in Androzoo when different imbalances were enforced. Figure 21 shows the average number of FPs identified in the multiple Androzoo experiments for different dataset imbalances when considering ADWIN, the best detector in our tests (This time we omitted the results for the other detectors to not pollute the plot with redundant trends–they follow the same pattern as for DREBIN). As for DREBIN, the number of FPs increases with the imbalance. Also, similarly to DREBIN, the number of bad frontiers decreases with the imbalance, as shown in Figure 23. Finally, the overall performance (F1 under time) also increases for AndroZoo (Figure 25), as for DREBIN.

Runtime Performance Overhead Analysis. We conclude our evaluation with the analysis of the runtime performance overhead introduced by our solution. As a trade-off for making our solution agnostic to the main classifier, we made our solution bigger: as it requires 2 additional models, it is expected the total performance requirement to also increase by 2 times. We consider this an acceptable trade-off for one aiming to explain drifts. We measured the actual solution overhead via the total experiment runtime (training, predictions, and retraining) in our machines. While the baseline values might change from machine to machine, the relative overhead should be generalized consistently.

Table 2: **Runtime Performance Overhead for DREBIN and AndroZoo.** The cost of individual retrains is reduced, but the total execution time cost increases.

		DREBIN		AndroZoo	
Models (#)	**Retrain Policy**	**Total Time / Retrains (#)**	**Cost / Overhead / Normalized**	**Total Time / Retrains (#)**	**Cost / Overhead / Normalized**
1	Type 1	11.65s / 5	2.73s / 0x / 0x	470.4s / 5	94s / 0x / 0x
3	Type 1	53.53s / 5	10.7s / 3.92x / 3.92x	1688.6 / 5	337.7s / 3.6x / 3.6x
3	Type 2	105.77s / 13	8.13s / 7.74x / 2.98x	3563.9s / 15	237.6s / 7.5x / 2.5x
3	Type 3	102.70s / 13	7.90s / 7.52x / 2.89x	4161.3s / 19	219s / 8.84x / 2.32x

Table 2 shows the total execution time and overhead values normalized by the number of retrains for both datasets (different problem sizes). The first to second rows show the increase in the metrics scores caused by adding the external monitor to the system, i.e., by tripling the number of classifiers/models, but without actually using their information for classification. Although this scenario is not realistic in practice, it works as ground truth for isolating variables and measuring the individual impact of the architecture itself. The third and fourth rows exemplify cases that actually consider the results of the added classifiers, as used in practice, which allows measuring the impact of policies.

The results for the two datasets follow the same trend: The total execution time significantly increases when new the architecture is added (first to second line). It more than triples the total execution time, as not only the prediction and retraining time are tripled but there are also additional costs to compute drift distances and directions. The major overhead, however, comes from the additional number of retrains triggered by the new policies, which are required to increase the long-term detection accuracy. This overhead is non-linear, but exponential, because each new retrain includes all data from previous epochs, which makes each new retrain slower. Despite this effect, the normalize

time per retrain remains constant (and lower than the triple from the original architecture addition). In fact, the more efficient the policy, the smaller the cost of an individual retrain. For instance, Type 3 triggers retrains in the most appropriate moments, such that it can achieve the same accuracy as Type 2, with the same number of retrains, but by training in a much earlier scenario, by predicting the drift occurrence, thus having to process less data. In sum, our results allow us to conclude that adding meta-classifiers to explain drift detector significantly increases the absolute time taken to process the samples, but it makes the individual retrains much more efficient.

4 Discussion

Wider Implications. Whereas exemplified in the malware detection domain, where we have domain expertise, the hereby presented findings apply to any domain that can be modeled as a binary classification problem under the same constraints. Thus, we expect our proposed architecture to explain drift occurrences in varied tasks.
Feedback for Operators. We expect operators of malware detection pipelines (e.g., MLSecOps) to benefit from our solution not only to understand the drift points but also to understand the pipeline operation at any point. Our solution's ability to identify improper frontiers might assist operators in identifying when the classifier was trained with a limited amount of data or a non-representative dataset and is not operating well in practice, which is common during the initial phases of malware detection pipeline deployments.
Limitations. We propose a practical solution for explaining drift points. We do not provide mathematical guarantees of the optimality of the retraining results. Future works should provide it via additional math formulations on top of our developments.
Future Work. This work is a first step towards explaining drift detection points, but our work is not exhaustive. Further developments should expand our contribution from the binary classification domain to the multi-class domain. It is also key to make analysis more fine-grained and robust. As cases of particular research interest to applying this approach in real scenarios, we point out the following open research problems and future research directions:

- **Label Delays.** In real deployments, the true labels used for drift detection are not immediately available but arrive delayed, which causes a delay in drift detection. Whereas here we assumed an ideal scenario, as most of the literature, it is key to develop strategies to handle drift delays to foster the solution adoption. It is key to be mindful that, in practice, the explanations provided by our solution would be delayed by the same amount of time as the label arrival delay.
- **Virtual Drifts.** In practical scenarios, it is common that the ground truth labels used for drift detection might flip over time from one class to another. These flips might cause improper drift detection (virtual drift points). Whereas this work, as most in the literature, assumed an ideal scenario of no label flips, the case of label flips should be addressed in real-world deployments. For such, strategies for identifying false drift reports should be developed.
- **Intra-Class Drifts.** The Sample's concept drifts not only across classes but also inside the same class as they evolve. A single concept approach, as most in the

literature, is blind to such modifications. To be able to spot such cases, we should not only learn a single malware concept but also split the learned malware concept into multiple ones and temporally evaluate their drift dynamics.

Making retraining as practical as explanation. Our approach makes the explanation of drift events practical. We consider that requiring additional processing for this task is acceptable since explanation and validation tasks are usually performed in controlled environments without processing constraints. In turn, whereas we demonstrated that it would foster early retraining, further studies are warranted to obtain the best cost-benefit for its deployment in practice. A lighter version of the approach can be developed, such as considering only the concept's centroid as the average confidence.

Expansion for Heterogeneous Architectures. Our experiments considered the first and second-layer classifiers as having the same classifier. This is an experimental choice to isolate variables and allows us to claim that any observed effect is due to the concepts, not due to the architecture. However, it is possible to hypothesize future deployments with heterogeneous architectures, where the second-layer classifier is even more powerful than the first layer to better learn the concept from limited data amounts.

The data coupling invariant: Note that although the model can be decoupled in their architectures, they always should have a strong decoupling in their data, i.e., the first and second layer models should be trained with the same dataset and present the maximum performance on them. If the models are decoupled in data, the second layer model might report a drift point that does not exist in the primary model.

5 Related Works

Proposals for New Drift Detectors are the most common in the literature. They vary from eliminating the need for labels [30] to new models and architectures [2, 24] for increased detection rates. However, these works do not focus their developments on explaining their drift results, which is our focus in this work.

Model Monitoring is the MLSecOps task this paper targets. Previous works proposed architectures to model a given model operation [7,8], but these works only approached the problem quantitatively, providing a final score of the system operation. In our qualitative approach, we used scores to explain the model operation at each moment.

Classifying Drift Events is a common contribution in the literature [21]. Previous works proposed characterizing drift events in multiple dimensions [28], such as the introduction of a new class, the recurrence of the drift events, the impact extent, and so on. Whereas characterizing drift in all its dimensions, the malware detection problem offers some advantages. such as its limitation to a binary classification problem, with no introduction of new classes, which allows for relaxing some constraints from the general case. In this case, we adopt a more practical classification [17] that considers real and virtual drift occurrences. We extend these concepts to drift in the malware detector class or frontier.

The limit of the existing explainable solutions is to only explain parts of the drift phenomenon, such as only local regions [18], only the wrong features [27], or only feature-frequency change [20]. We here present explanations that cover all these scenarios. In addition, we present tests with real and not synthetic data [22].

The applications benefiting from drift explanation described in the literature range from data mining [1] to COVID identification [15]. Whereas drift is a fundamental phenomenon in malware detection, few drift detection architectures are designed for

the task, and the few existing ones [26] do not explain the phenomenon beyond feature change. Ideally, we would like to perform forensic procedures of malware detectors [13] but to explain the binary classification and not how new malware families appear.

Security-focused drift detectors are limited in either providing information only about the features that changed without explaining the model changes [11] or by determining model changes without explaining the features used in the models [23]. Conformal evaluation [6], the closest idea to ours, also split detection in two (confidence and credibility), but unlike our proposal, these information are not used to explain the classifier situation at each moment, but only to early retrain the classifier. 2-level architectures are becoming more common over time. A recent proposal also split the problem into two [19] but adopted a mathematical approach based on the latent space conformity to evaluate drift points. Our proposal uses ML classifiers to model such deviations to explain the points in addition to triggering a retrain on them.

Boundary-based vs. Distance-based Explanations. Drift explanations are typically classified in these two classes, each one presenting pros and cons. Our approach fits into the first category as it aims to explain frontier-crossing events. An exemplary work in the second category is CADE [29], which uses contrastive learning to measure the difference between goodware and malware classes. We consider that boundary-based approaches are more interesting solutions to the scenario presented here as they allow explaining every drift point, including false positive drift reports, whereas distance-based approaches only report true concept changes.

Explaining Detection vs. Explaining Drifts. The explanation of ML-based malware detection models is a growing topic, but most of the existing solutions focus on explaining the detection model itself. Popular metrics for it include, for instance, the fidelity of the model [12], but these metrics do not explain the malware evolution. In this sense, the here proposed drift explanation is a step ahead to complement (and not replace) previous explanation initiatives.

6 Conclusion

This work investigated the problem of explaining concept drift detection occurrences in malware detection pipelines. We proposed the idea of splitting the concept drift phenomenon into **(1)** classifier's frontier changes and **(2)** concept changes to explain better the drift causes. We evaluated the viability of our proposal via experiments with the DREBIN and AndroZoo datasets. We discovered that our approach not only explains all drift events, identifying true and false drift reports, but also that the reliance on an explainable method increases the detector performance by allowing the retraining to occur in the most promising points (true drift events).

Reproducibility. All code developed in this search is available at `https://github.com/Botacin-s-Lab/Concept.Drift.Explanation`

Acknowledgments. We thank the anonymous reviewers and shepherd for all the helpful insights. Marcus Botacin thanks NSF for the support via the CNS 2327427 grant.

References

1. Adams, J.N., van Zelst, S.J., Quack, L., Hausmann, K., van der Aalst, W.M., Rose, T.: A framework for explainable concept drift detection in process mining. In: Business Process Management: 19th International Conference, BPM 2021, Rome, Italy, September 06–10, 2021, Proceedings 19. pp. 400–416. Springer (2021)

2. Andresini, G., Pendlebury, F., Pierazzi, F., Loglisci, C., Appice, A., Cavallaro, L.: Insomnia: Towards concept-drift robustness in network intrusion detection. In: Proceedings of the 14th ACM workshop on artificial intelligence and security. pp. 111–122 (2021)
3. Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., Rieck, K.: Dos and don'ts of machine learning in computer security. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 3971–3988. USENIX Association, Boston, MA (Aug 2022), https://www.usenix.org/conference/usenixsecurity22/presentation/arp
4. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K.: Drebin: Effective and explainable detection of android malware in your pocket. In: NDSS. The Internet Society (2014), http://dblp.uni-trier.de/db/conf/ndss/ndss2014.html#ArpSHGR14
5. Baena-Garcıa, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., Morales-Bueno, R.: Early drift detection method. In: Fourth international workshop on knowledge discovery from data streams. vol. 6, pp. 77–86. Citeseer (2006)
6. Barbero, F., Pendlebury, F., Pierazzi, F., Cavallaro, L.: Transcending transcend: Revisiting malware classification in the presence of concept drift. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 805–823 (2022). https://doi.org/10.1109/SP46214.2022.9833659
7. Bhaskhar, N., Rubin, D.L., Lee-Messer, C.: An explainable and actionable mistrust scoring framework for model monitoring. IEEE Transactions on Artificial Intelligence (2023)
8. Bhatt, U., Xiang, A., Sharma, S., Weller, A., Taly, A., Jia, Y., Ghosh, J., Puri, R., Moura, J.M., Eckersley, P.: Explainable machine learning in deployment. In: Proceedings of the 2020 conference on fairness, accountability, and transparency. pp. 648–657 (2020)
9. Bifet, A., Gavalda, R.: Learning from time-changing data with adaptive windowing. In: Proceedings of the 2007 SIAM international conference on data mining. pp. 443–448. SIAM (2007)
10. Ceschin, F., Botacin, M., Bifet, A., Pfahringer, B., Oliveira, L.S., Gomes, H.M., Grégio, A.: Machine learning (in) security: A stream of problems. Digital Threats **5**(1) (mar 2024). https://doi.org/10.1145/3617897, https://doi.org/10.1145/3617897
11. Ceschin, F., Botacin, M., Gomes, H.M., Pinagé, F., Oliveira, L.S., Grégio, A.: Fast & furious: On the modelling of malware detection as an evolving data stream. Expert Systems with Applications **212**, 118590 (2023). https://doi.org/https://doi.org/10.1016/j.eswa.2022.118590, https://www.sciencedirect.com/science/article/pii/S0957417422016463
12. Chen, L., Yagemann, C., Downing, E.: To believe or not to believe: Validating explanation fidelity for dynamic malware analysis. In: CVPR Workshops. pp. 48–52 (2019)
13. Chow, T., Kan, Z., Linhardt, L., Cavallaro, L., Arp, D., Pierazzi, F.: Drift forensics of malware classifiers. In: Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security. pp. 197–207 (2023)
14. Daoudi, N., Allix, K., Bissyandé, T.F., Klein, J.: A deep dive inside drebin: An explorative analysis beyond android malware detection scores. ACM Trans. Priv. Secur. **25**(2) (may 2022). https://doi.org/10.1145/3503463, https://doi.org/10.1145/3503463
15. Duckworth, C., Chmiel, F.P., Burns, D.K., Zlatev, Z.D., White, N.M., Daniels, T.W., Kiuber, M., Boniface, M.J.: Using explainable machine learning to characterise data drift and detect emergent health risks for emergency department admissions during covid-19. Scientific reports **11**(1), 23017 (2021)
16. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: Bazzan, A.L.C., Labidi, S. (eds.) Advances in Artificial Intelligence – SBIA 2004. pp. 286–295. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
17. Gomes, H.M., Grzenda, M., Mello, R., Read, J., Le Nguyen, M.H., Bifet, A.: A survey on semi-supervised learning for delayed partially labelled data streams. ACM Comput. Surv. **55**(4) (nov 2022). https://doi.org/10.1145/3523055, https://doi.org/10.1145/3523055

18. Haug, J., Braun, A., Zürn, S., Kasneci, G.: Change detection for local explainability in evolving data streams. In: Proceedings of the 31st ACM International Conference on Information & Knowledge Management. pp. 706–716 (2022)
19. He, Y., Lei, J., Qin, Z., Ren, K.: Going proactive and explanatory against malware concept drift (2024)
20. Hinder, F., Vaquet, V., Brinkrolf, J., Hammer, B.: Model-based explanations of concept drift. Neurocomputing **555**, 126640 (2023). https://doi.org/https://doi.org/10.1016/j.neucom.2023.126640, https://www.sciencedirect.com/science/article/pii/S0925231223007634
21. Hu, H., Kantardzic, M., Sethi, T.S.: No free lunch theorem for concept drift detection in streaming data classification: A review. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **10**(2), e1327 (2020)
22. Jacob, V., Song, F., Stiegler, A., Rad, B., Diao, Y., Tatbul, N.: Exathlon: A benchmark for explainable anomaly detection over time series. arXiv preprint arXiv:2010.05073 (2020)
23. Jordaney, R., Sharad, K., Dash, S.K., Wang, Z., Papini, D., Nouretdinov, I., Cavallaro, L.: Transcend: Detecting concept drift in malware classification models. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 625–642. USENIX Association, Vancouver, BC (Aug 2017), https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney
24. Panda, P., Kancheti, S.S., Balasubramanian, V.N., Sinha, G.: Interpretable model drift detection. In: Proceedings of the 7th Joint International Conference on Data Science & Management of Data (11th ACM IKDD CODS and 29th COMAD). pp. 1–9 (2024)
25. Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., Cavallaro, L.: TESSERACT: Eliminating experimental bias in malware classification across space and time. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 729–746. USENIX Association, Santa Clara, CA (Aug 2019), https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury
26. Shaer, I., Shami, A.: Thwarting cybersecurity attacks with explainable concept drift. arXiv preprint arXiv:2403.13023 (2024)
27. Vishnampet, R., Shenoy, R., Chen, J., Gupta, A.: Root causing prediction anomalies using explainable ai. arXiv preprint arXiv:2403.02439 (2024)
28. Webb, G.I., Hyde, R., Cao, H., Nguyen, H.L., Petitjean, F.: Characterizing concept drift. Data Mining and Knowledge Discovery **30**(4), 964–994 (Apr 2016). https://doi.org/10.1007/s10618-015-0448-4, http://dx.doi.org/10.1007/s10618-015-0448-4
29. Yang, L., Guo, W., Hao, Q., Ciptadi, A., Ahmadzadeh, A., Xing, X., Wang, G.: CADE: Detecting and explaining concept drift samples for security applications. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 2327–2344. USENIX Association (Aug 2021), https://www.usenix.org/conference/usenixsecurity21/presentation/yang-limin
30. Zheng, S., van der Zon, S.B., Pechenizkiy, M., de Campos, C.P., van Ipenburg, W., de Harder, H., Nederland, R.: Labelless concept drift detection and explanation. In: NeurIPS 2019 Workshop on Robust AI in Financial Services: Data, Fairness, Explainability, Trustworthiness, and Privacy (2019)