Investigating Labelless Drift Adaptation for Malware Detection

Zeliang Kan[*†], Feargus Pendlebury[†‡‖], Fabio Pierazzi[*], Lorenzo Cavallaro[†]
[*]King's College London
[†]University College London
[‡]Royal Holloway, University of London
[‖]International Computer Science Institute

ABSTRACT

The evolution of malware has long plagued machine learning-based detection systems, as malware authors develop innovative strategies to evade detection and chase profits. This induces concept drift as the test distribution diverges from the training, causing performance decay that requires constant monitoring and adaptation.

In this work, we analyze the adaptation strategy used by DroidEvolver, a state-of-the-art learning system that self-updates using pseudo-labels to avoid the high overhead associated with obtaining a new ground truth. After removing sources of experimental bias present in the original evaluation, we identify a number of flaws in the generation and integration of these pseudo-labels, leading to a rapid onset of performance degradation as the model poisons itself. We propose DroidEvolver++, a more robust variant of DroidEvolver, to address these issues and highlight the role of pseudo-labels in addressing concept drift. We test the tolerance of the adaptation strategy versus different degrees of pseudo-label noise and propose the adoption of methods to ensure only high-quality pseudo-labels are used for updates.

Ultimately, we conclude that the use of pseudo-labeling remains a promising solution to limitations on labeling capacity, but great care must be taken when designing update mechanisms to avoid negative feedback loops and self-poisoning which have catastrophic effects on performance.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Security and privacy** → **Intrusion/anomaly detection and malware mitigation**.

KEYWORDS

Machine Learning; Malware Detection; Online Learning

ACM Reference Format:
Zeliang Kan, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2021. Investigating Labelless Drift Adaptation for Malware Detection. In *Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security (AISec '21), November 15, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3474369.3486873

AISec '21, November 15, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8657-9/21/11…$15.00
https://doi.org/10.1145/3474369.3486873

1 INTRODUCTION

Machine learning-based malware detectors operate in hostile, dynamically changing environments. Malware authors utilize obfuscation [1, 24] and evasion techniques [4, 42, 60] to avoid detection, develop new technologies to increase infectivity [55], and occasionally adopt new paradigms with greater profit potential (e.g., ransomware [27]). Additionally, the underlying platform continues to evolve, with new features, APIs, and programming practices further distorting the boundary between goodware and malware.

This activity causes fluctuations in the data distribution, a phenomenon known as *concept drift* [23, 25, 36], in which new examples begin to differ significantly from those observed during the training phase—i.e., the definition of what malware *is* changes over time. Depending on the root cause, concept drift can be sudden and dramatic, or subtle and gradual [21], but nevertheless it violates the i.i.d. assumption required by most classification algorithms. This violation causes an ongoing performance degradation that requires constant monitoring and adaptation [41].

One promising direction for overcoming this issue is concept drift *adaptation*, in which new knowledge is introduced to the classifier to reduce the cumulative prediction error. This class of methodologies includes active learning [49, 50] and online learning [9, 38, 39] techniques. While generally effective at mitigating performance degradation, these techniques require high-quality labels to be available during test time, which are usually expensive and time-consuming to obtain [35, 41].

A promising research direction that mitigates the high cost of labeling is *weak supervision* (e.g., semi-supervised learning), in which a model is trained using both labeled and unlabeled data. One branch of research focuses on the use of *pseudo-labels* to provide noisy—but sufficiently accurate—labels for new data with which to update the model [22, 30, 44]. An exemplary work in this area applied to Android malware detection is DroidEvolver [59], which proposes updating an ensemble of online learners using its predicted labels as pseudo-labels to eliminate labeling costs altogether.

In this work, we critically examine such a strategy in greater depth, using DroidEvolver [59] as a case study. First, we remove sources of experimental bias present in the initial evaluation, applying DroidEvolver to a dataset of 129,728 apps where malware is the minority class (~10% prevalence, as recommended in [41]). We observe a catastrophic *self-poisoning effect* which causes performance to degrade suddenly and significantly. To explain these effects, we identify several weaknesses in the system design relating to the generation of pseudo-labels: assumptions on dataset diversity, biased ensemble decision functions, and incorrect integration of predicted labels. We propose DroidEvolver++, a new variant of DroidEvolver, which addresses these issues, and we quantify the effect of each change through ablation studies.

This leads us to further explore the degree to which pseudo-labels might be useful in drifting security settings where the assumptions required for semi-supervised learning may be violated. We investigate the tolerance of DroidEvolver++ to label noise to demonstrate how the accuracy of pseudo-labels hinders the capabilities of the model. Similarly, we show how methodologies for ensuring high-quality pseudo-labels by thresholding on model confidence [30] and model uncertainty [44] can be applied to time-aware malware detection.

Ultimately, we conclude that the use of pseudo-labeling is still a promising solution to limitations on labeling capacity, but great care must be taken when designing the update mechanism to avoid negative feedback loops and self-poisoning. We urge caution when using predicted labels alone as pseudo-labels for malware detection.

In summary, we provide the following contributions:

- We identify shortcomings in DroidEvolver [59], the current state-of-the-art drift adaptation approach for malware detection (§3), and outline the lessons learned as well as proposing a more robust and effective variant (§4).
- We further explore the use of pseudo-labels for malware detection and the conditions under which they might still be a valuable strategy for updating models in the face of concept drift. We show that using a model's predicted labels as pseudo-labels greatly hinders its performance relative to the accuracy of its predictions, but that methods to improve the quality of pseudo-labeling can mitigate this to some degree (§5).
- To support future efforts in malware drift adaptation, we release the code for DroidEvolver++ and our implementation of alternative pseudo-label selection strategies (§8).

2 DRIFT ADAPTATION

In this section we provide some background on the problem of concept drift (§2.1) and how the use of online learning has been proposed to mitigate its impact (§2.2). Finally we give an overview of DroidEvolver [59], the drift adaptation approach that forms the core case study in our analysis (§2.3).

2.1 Concept Drift

Dataset shift is a common phenomenon in classification tasks when the joint distribution of inputs and outputs differs between training and test time [43]. Dataset shift can come in many forms: a change in the feature distribution (*covariate shift*), a change in the prevalence of a particular class (*prior probability* or *label shift*), or a change in the class definition itself (*concept shift*). These shifts are often intertwined and it can be difficult to attribute performance loss to a particular effect, so *concept drift* is often used as an umbrella term for shifts in general, particularly within the security literature [e.g., 15, 23, 46, 53]—we stick to this convention throughout this work.

Concept drift often affects real-world classifier deployments, either as a result of experimental bias during training and calibration [41] or due to a 'natural' change in the properties of the target classes over longer periods (e.g., the problem of aging faces in facial recognition [37]). Sources of drift in malware classification can be fairly benign, such as changes in market trends or new developer APIs [62]. However, the main driving force of drift is the development of new malware techniques to evade detection [1, 4, 42, 60], increase infection rates [55], and generate greater profits [27]. This results in an evolution of malware over time, which reduces the ability of classifiers to recognize newer examples [3, 25, 35, 41].

2.2 Online Learning for Malware Detection

In the online learning setting, data is provided as a stream of observations in sequence, rather than as a batch of examples.

Typically, an online learner will make a prediction for each new observation, and then subsequently update itself once the true label becomes available [9]. Online learners are useful for adapting to new patterns which makes them a useful candidate for tackling concept drift as malware evolves over time.

Another advantage of online learning is that it allows a trained detection system to be updated at a lower cost, as the system can be partially retrained using the new data only, and many methods reduce computation further (e.g., passive-aggressive classifiers [14] that update only when the model makes an incorrect prediction).

However, there are still limitations of online learners. In particular, online learners will gradually *unlearn* previously learned information and are also susceptible to *catastrophic interference* [20, 26, 28, 34] in which past information is forgotten completely and abruptly. Like all ML algorithms, they are also sensitive to the accuracy of new labels, but are specifically affected by whether labeling capacity can keep up with the volume of the incoming unlabeled data, in contrast to typical supervised batch learning where training only occurs after all ground truth labels have been obtained.

Online learning has been proposed for the detection of Android malware, most notably in the case of Casandra [38] and DroidOL [39]. Both build on Weisfeiler-Lehman graph kernels [51] to extract semantic features from the apps, while Casandra uses a Confidence Weighted algorithm [17] as its online learner and DroidOL uses a Passive Aggressive algorithm [14]. In the remainder of the section we will explore a more recent work, DroidEvolver [59], which departs from the previous methods by relying on *pseudo-labels* for updates, rather than ground truth labels.

2.3 Adaptation Without Labels: DroidEvolver

Here we provide an overview of DroidEvolver [59] as a case study in our analysis on the use of pseudo-labels for malware detection.

DroidEvolver employs an ensemble of five linear online learning models: Passive Aggressive (PA) [14], Online Gradient Descent (OGD) [63], Adaptive Regularization of Weight Vectors (AROW) [14], Regularized Dual Averaging (RDA) [58], and Adaptive Forward-Backward Splitting (Ada-FOBOS) [18]. Each uses a binary feature space where 0 and 1 indicate the absence or presence of an API call, respectively. API calls naturally reflect the evolution of both the Android framework and the apps themselves, and can be easily extracted from bytecode using static methods [7, 16]. The ensemble is trained using an initial dataset of labeled malware and goodware.

At test time, DroidEvolver uses the weighted sum of decision scores as the ensemble decision function to aggregate the predictions of the underlying models, however the predictions of *aging* models are excluded from the sum. To measure whether a model is aging or not, a fixed-length *app buffer* is maintained which holds a small set of apps that aim to be representative of the distribution up to the current test period. A *Juvenilization Indicator* (JI) score

is calculated as the proportion of apps in the app buffer, of the same class, which have decision scores greater than the new test object. If the JI score falls below or above a precalibrated lower and upper threshold, respectively, then the model is marked as aging. Note that this notion of *dissimilarity* for identifying drifting objects is essentially the same as the nonconformity score (NCM) used in Transcend [8, 23] and other methods derived from conformal prediction theory [40, 57].

Once a model is marked as aging, an *evolution* is triggered to revitalize the model. In this case, the update mechanism of the underlying online learner is invoked on the new drifting object, using the ensemble prediction as the label (i.e., the pseudo-label). Additionally, the feature set is extended to include any previously unseen features present in the new object. If either none or all of the models are aging, no update will occur.

To evaluate DroidEvolver, the original authors perform a comprehensive series of experiments, testing the performance with and without the presence of concept drift and measuring the overhead of the evolution process. They use a dataset of 68,016 apps spanning 6 years with a roughly balanced class ratio (~51% malware).

Note that we do not mean to diminish the research contributions of DroidEvolver, which was one of the first approaches to tackle the trade-off between performance over time and the efficiency of updating detection models. To this end, DroidEvolver significantly outperformed contemporary state-of-the-art approaches and it is a credit to the quality and openness of the work that we have been able to able to extend it and use it as our case study. Our intention is to build on DroidEvolver's contributions by refining our understanding of pseudo-labels in malware detection systems, to foster future work in the area of drift adaptation.

3 IDENTIFYING CHALLENGES IN PSEUDO-LABEL GENERATION

We assess the impact of experimental bias in the original evaluation of DroidEvolver [59] and identify weaknesses in its design.

3.1 Experimental Setup

Dataset. We use a dataset consisting of 129,728 Android applications with 116,993 goodware and 12,735 malware (a ratio of approximately 9:1, as suggested by Pendlebury et al. [41]). Features are binary, with 0 and 1 indicating the absence or presence of an API call, respectively. We use the DroidEvolver feature extraction script to build the feature space. The sample is taken from the public AndroZoo dataset [2] where each app is associated with VirusTotal (VT) detection metadata, which is used to derive labels. We follow examples in prior work [35, 41] and mark apps with 0 VT detections as goodware and apps with 4+ VT detections as malware. We note that removing grayware may positively inflate the results [6] and this should be taken into account when interpreting them. However, having a clean separation between malware and goodware reduces natural label noise and helps us more confidently control this variable to evaluate the systems' tolerance to label noise in later experiments (§5.2). The dataset spans three years. For performing a time-aware evaluation, we use the first year as training data and partition the remaining data into 24 test periods of one month each.

Metrics. To measure overall detection performance we use Precision, Recall, and the F_1 score. We also keep track of the *drift rate*, i.e., the proportion of new inputs in each test period that are identified as drifting. For DroidEvolver, drifting objects are those whose decision score falls outside the JI thresholds. DroidEvolver does not update models which are not aging, and models are marked as aging when new inputs are marked as drifting with respect to that model; therefore we are interested in maintaining a low drift rate over time. When ground truth labels are used for model updates, the drift rate reflects the labeling cost. When pseudo-labels are used, high drift rates increase the risk that no model in the ensemble will correctly classify new objects which leads to decay. Note that this metric relies on the ability of DroidEvolver to accurately identify drifting objects which may be undermined as the system deteriorates. In §6 we discuss the use of external drift detectors to support a pseudo-labeling system.

Vanilla baseline (PassiveAggressive). To act as a simple baseline and to demonstrate the presence of drift in the dataset, we use the Passive-Aggressive classifier [14] from the DroidEvolver ensemble *without* performing any model updates (PassiveAggressive). In this configuration, the classifier is equivalent to a linear support vector machine [13]. To measure the severity of the drift, we use Transcend [8, 23], a state-of-the-art approach to equip classifiers with a rejection option (as in abstaining classifiers), due to its similarity to the drift identification mechanism used by DroidEvolver. However, this is simply for the purpose of illustrating the drift; we do not reject any of the identified drifting points.

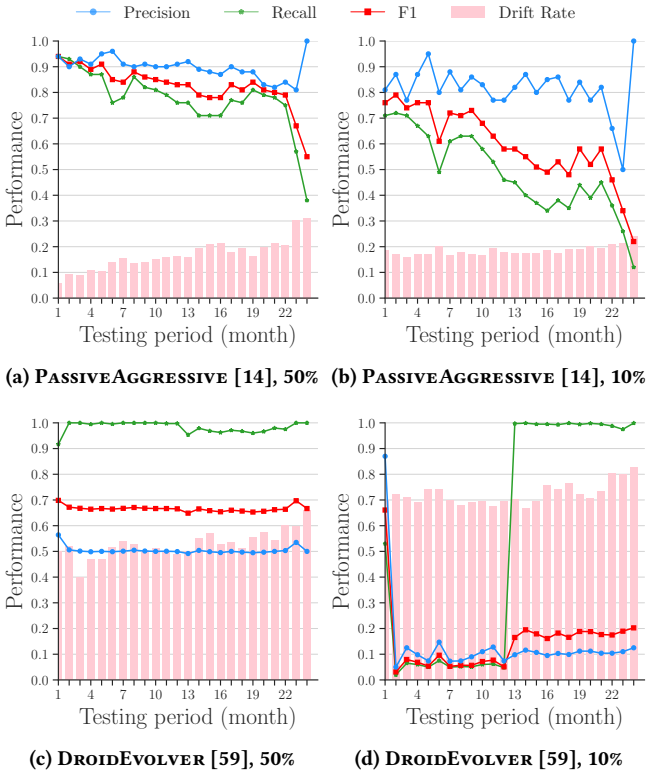
Threshold tuning. The pair of JI thresholds play a critical role in distinguishing between drifting and non-drifting points. To tune the thresholds, we initialize the model pool with the first eleven months of training data and use the subsequent month as a calibration set. We choose the JI threshold pair that performs the best on the calibration set, which is 0.3 and 0.7 for the lower and upper thresholds, respectively.

3.2 Assumptions on Data Distribution

Before analyzing the design of DroidEvolver itself, we first examine two assumptions regarding the data distribution, which may differ from a realistic setting.

Class balance. As shown by Pendlebury et al. [41], two forms of experimental bias, *spatial* and *temporal bias*, are a common cause for overinflated results in machine learning-based malware experiments. Temporal bias results when a dataset is temporally inconsistent, e.g., when the training data does not precede the test data or when classes are sampled from differing periods. Spatial bias refers to when an unrealistic ratio of malware to goodware is used in the test data. This is of particular importance in security, where the positive class is often the minority class. Overrepresenting this class leads to Precision being erroneously inflated [41]. As demonstrated in their time-aware evaluation, DroidEvolver's evaluation is temporally consistent and not affected by this experimental bias. We thus assess the impact of spatial bias on the system's performance.

We evaluate DroidEvolver and PassiveAggressive with two different dataset compositions. In the first, we downsample the



(a) PassiveAggressive [14], 50% (b) PassiveAggressive [14], 10%

(c) DroidEvolver [59], 50% (d) DroidEvolver [59], 10%

Figure 1: PassiveAggressive [14] without updates (top) vs. DroidEvolver [59] (bottom) applied to test data with ~50% malware (left column) vs. ~10% malware (right column).

amount of goodware to achieve a 50/50 balance between classes. In the second, we use our imbalanced dataset in its entirety.

Figures 1a and 1b show the performance of PassiveAggressive. The system clearly suffers from performance decay as the F_1 score gradually decays over time on both balanced (left) and imbalanced (right) datasets. In both cases, the rate of drifting objects gradually increases, demonstrating the presence of concept drift in the dataset. PassiveAggressive is sensitive to spatial bias [41], with performance decay more pronounced on the imbalanced data. The overall Precision in both cases is relatively stable, but Recall suffers greatly, indicating a large amount of False Negatives.

The performance of DroidEvolver is shown in Figures 1c and 1d. On the balanced dataset (left), performance quickly drops to an F_1 score of ~0.65, while the Precision drops to ~0.50, equivalent to random guessing in the balanced setting. The true base rate represented in the imbalanced case (right) shows a starker picture, with performance degrading severely in the first two months. In 2016 (i.e., from month 13), the model has high Recall, but this is an artefact of the class ratio as the majority of samples are predicted as malware, leading to low Precision that matches the malware base rate of ~0.1.

From these results, we can see that DroidEvolver, like PassiveAggressive, is indeed sensitive to spatial bias, which may have overinflated the results of the original evaluation. However, the low performance even in the balanced setting suggests that other factors may be at play, which we explore in §3.3.

Dataset diversity. We also reason briefly about other aspects of our dataset which may contribute to the lower performance. The dataset in the original evaluation spans from 2011 to 2016 inclusive, with roughly 5,000 malware and 5,000 goodware in each year. As the feature set is augmented over time, the number of features (i.e., API calls) grows from 14,327 to 52,001 features over the 6 year period. Our dataset contains just over twice as many apps and contains 105,092 distinct features in the 2014 training set alone. Even in the shorter time frame, when testing ends in Dec 2016, the number of recorded features has increased dramatically to 249,102, five times larger than the original evaluation. We hypothesize that the increased diversity and more abrupt onset of drift may make it difficult for DroidEvolver to adapt in time before negative feedback loops of the update mechanism take over. This highlights the sensitivity of models to specific datasets and we advise testing on more than one dataset where possible (although we recognize this is infeasible in many security settings where obtaining high quality datasets is challenging [6]).

3.3 Weaknesses in Pseudo-Label Generation

The previous experiment shows that DroidEvolver suffers from severe performance decay. As the non-updating PassiveAggressive outperforms DroidEvolver, and the degradation occurs faster than the naturally occurring drift illustrated in Figures 1a and 1b, we hypothesize that the model poisons itself due to weaknesses in the pseudo-label generation. Given this, we examine the update mechanism of DroidEvolver and identify the following flaws in addition to the erroneous dataset assumptions outlined in §3.2.

The ensemble is dominated by a subset of models. The pseudo-labels used for updating are derived from a weighted vote between the non-aging models in the model pool, specifically $\sum_{j=1}^{M} w_j \cdot x_i$ where w_j is the weight vector of the j^{th} model in the pool and x_i is the feature vector of the new test object. However, the different algorithms have very diverse ranges for the value of $w_j \cdot x_i$ (i.e., their individual decision functions). Therefore, algorithms that naturally produce outputs of a larger magnitude tend to dominate the weighted voting. In our experiments, the OGD and Ada-FOBOS classifiers have a larger decision output than the other three algorithms. This effectively decreases the model diversity in the ensemble, which increases the risk of performance degradation once the effectiveness of OGD and Ada-FOBOS drop.

Apps in the buffer are replaced randomly, causing a skew toward the majority class. DroidEvolver maintains a fixed-length app buffer, which contains a subset of apps representing the distribution up to the current test period. The decision scores of new inputs are compared to decision scores of apps in the buffer in order to calculate the JI score that measures whether a model is aging. The buffer is kept fresh by replacing apps each time a new sample is received. However, apps are replaced at random independent of their classes, which can cause apps in the buffer to skew towards a particular class. This problem is exacerbated when a realistic class balance is used (see §3.2) as one class quickly becomes underrepresented. In the extreme case, the buffer may

contain only samples of a single class as all apps of the other class have been replaced, which leads to errors in the JI computation.

The JI score of app buffer apps is not kept updated. DroidEvolver keeps track of the JI score for apps in the app buffer. However, these scores are not kept updated, which means that the JI score of new objects will be calculated using decision scores from many past models. These scores may not be representative of the current distribution, leading to incorrect decisions about which objects are drifting. Ideally, the JI score should be recomputed using fresh decision scores from each model.

The upper JI threshold causes high confidence predictions to be discarded. DroidEvolver uses both a lower and an upper JI threshold to identify drifting examples. The intuition is that objects which are very close or very far from the decision boundary with respect to other objects are more likely to be anomalous and thus drifting. While the lower threshold follows established results from other areas (e.g., the uncertainty sampling strategy from active learning [49] and rejection thresholds of Transcend [8, 23] rely on the same intuition), we argue that the upper threshold is harmful to the system. This is because it suggests that the points are clustered in a ball (or 'blob') in the decision region, with the densest region at the centroid representing the points most representative of the class. In such a case, it is possible to enter the class region from one boundary, pass through the densest region, and pass out through the opposing boundary. While this is true for many non-linear classifiers (e.g., support vector machines using an RBF kernel [13]), for the linear binary classifiers used in the ensemble, this is not the case. For these classifiers, as points move away from the decision boundary, they only become *more* representative of that class, i.e., the classifier is *more* confident of its prediction. The corollary of this is that DroidEvolver marks models that produce high-confidence predictions as aging, thus discarding high-quality pseudo-labels from the update mechanism, increasing its susceptibility to inaccurate predictions and self-poisoning.

4 DROIDEVOLVER++

To address the previously described shortcomings we propose an extension, DroidEvolver++. We hope that this will also provide a more stable baseline for future work to compare against. To evaluate, we measure the Precision, Recall, F_1 score, and drift rate, where the drift rate is the proportion of new inputs identified as drifting each period (see §3.1).

We add a calibration step to the model pool initialization step. This tuning step finds the best JI threshold for detecting drifting apps, as well as the ratio between goodware and malware in the app buffer. Algorithms 1 and 2 show the pseudo-code for these operations. Note that we eliminate the upper JI threshold in order to avoid discarding predictions with high confidence.

We change the logic for the pseudo-label generation to use the majority vote between all non-aging models (hard labels), instead of the original ensemble decision function $\Sigma_{j=1}^{M} w_j \cdot x_i$. This ensures specific classifiers do not dominate the decision due to the range of their decision function outputs.

We also fix the percentage of malware in the app buffer by only replacing apps with objects of the same class. This ensures the buffer does not become skewed towards a particular class or that a class loses all representation entirely.

Algorithm 1: JI Thresholds Tuning

```
Result: (τ0, τ1) Best JI Thresholds
x_train, x_validate = split (X_train);
models = modelpool_init (x_train);
best_score = 0.0;
buffer = buffer_generation(x_train, models, size = 2000);
for i ← 0 to 0.9 step 0.1 do
    for j ← i + 0.1 to 1.0 step 0.1 do
        F1 = DroidEvolver (models, x_validate, i, j, buffer);
        if F1 > best_score then
            τ0, τ1 = i, j;
        end
    end
end
Return (τ0, τ1);
```

We recompute the JI scores of apps in the app buffer each time a model updates. This ensures the system does not make decisions based on outdated information.

However, we recommend that, in the default configuration at least, ground truth labels are used in place of pseudo-labels for the model update. While this increases the cost of maintaining the system, it avoids the self-poisoning effects which render the models unusable during periods of extreme drift. Nevertheless, labeling pressure is still reduced as updates only happen when *aging* models appear in the model pool, and improving the stability of the system overall should reduce the rate at which models age. As DroidEvolver's original strength is that it does not require ground truth labels at all, we later propose an additional mechanism to improve the stability of the pseudo-labels to some degree (§5.1) and explore the settings in which pseudo-labels may remain effective (§5.2).

4.1 Tuning Class Ratio of the App Buffer

The fixed-length app buffer plays a vital role in distinguishing if a test object is drifting. The composition of apps in the buffer, and the degree to which they capture the current distribution, affects how the model is updated. In §3.2 we demonstrated that the random replacement of apps in the buffer of DroidEvolver can lead to failure due to the class imbalance in the data. However, as DroidEvolver++ ensures that apps only replace other apps of the same class, we can further tune the ratio of malware to goodware in the buffer before performing further experiments. Note that while the test dataset must follow a realistic malware-to-goodware ratio to avoid spatial bias, the ratio in the app buffer can be controlled.

To avoid data snooping, we use the first 11 months of 2014 as the training data and the final month of 2014 as the calibration data. After initializing the model pool, we perform a regular update on the calibration set. We test different malware rates in the range [0.1, 0.9] at increments of 0.1, following the procedure in Algorithm 2.

Figure 2 illustrates the performance of DroidEvolver++ for different ratios with the proportion of malware shown on the horizontal axis. The performance is erratic and does not show a strong

Algorithm 2: App Buffer Class Ratio Tuning

Result: r Best rate of malware in app buffer

x_{train}, $x_{validate}$ = split (X_{train});
models = modelpool_init (x_{train});
best_score = 0.0;
for $i \leftarrow 0.1$ **to** 0.9 ***step*** 0.1 **do**
    buffer = buffer_generation(x_{train}, *models*, ratio=i);
    F_1 = DroidEvolver (models, $x_{validate}$, τ_0, τ_1, buffer);
    if $F_1 > best\_score$ **then**
        $r = i$;
    end
end
Return r;

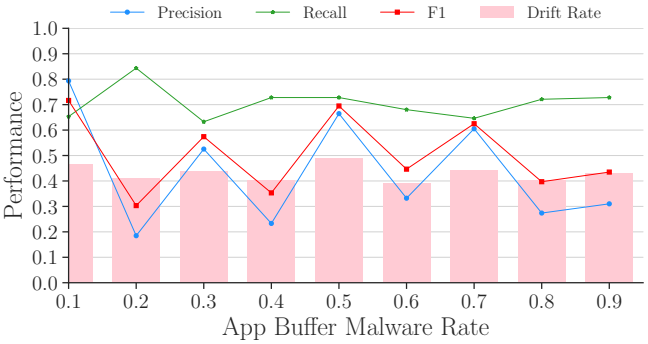


Figure 2: Impact of different malware-to-goodware ratios in the app buffer, DroidEvolver++ trained on the first 11 months of 2014 and calibrated on the final month of 2014.

trend. However, there appears to be some consistency when the malware rate is furthest from the calibration class distribution, at 0.8 and 0.9. Similarly, the peak performance is at 0.1, the value closest to malware rate in the calibration set, with an F_1 score of 0.72. Given these results, we fix the malware rate in the app buffer at 0.1 to approximate the expected rate at inference time.

4.2 DroidEvolver++ Ablation Study

In this section, we perform an ablation study to isolate each of our modifications and analyze their impact. We follow the same dataset and experimental setup as described in §3.1, considering only the more realistic imbalanced dataset setting.

We first conduct a control experiment with all the extensions of DroidEvolver++ activated. Then we disable each extension in turn, and compare the performance to the control. Figure 3a shows the performance of the control. The F_1 score begins at 0.75 in the initial test period, and remains relatively stable between 0.70 and 0.80 over the two-year period. Over the initial year, the F_1 score rises on average, peaking at 0.83—although this is somewhat expected given the use of ground truth labels. While the performance drops in the last three months, this is related to the very small number of samples in these months, as observed in prior work [41].

The drift rate stays relatively stable, averaging 0.50. Although much lower than DroidEvolver (cf. Figure 1d), this is a relatively high rate as each drifting point must be manually labeled. This is partially a cost of needing to maintain five models in the model pool—even if a point is considered drifting only for a single model, it must be labeled to update (and 'de-age') that model. In §5.1 we explore strategies to improve this performance-cost trade-off.

Modified Weighted Voting. We deactivate the modified weighted voting and revert to the original $\Sigma_{j=1}^{M} w_j \cdot x_i$ ensemble decision function. As shown in Figure 3b, the F_1 score does not change much compared to the control, decreasing by 0.05 in the first few months. However, the Precision drops considerably while Recall rises, indicating that the model is over-predicting the positive class.

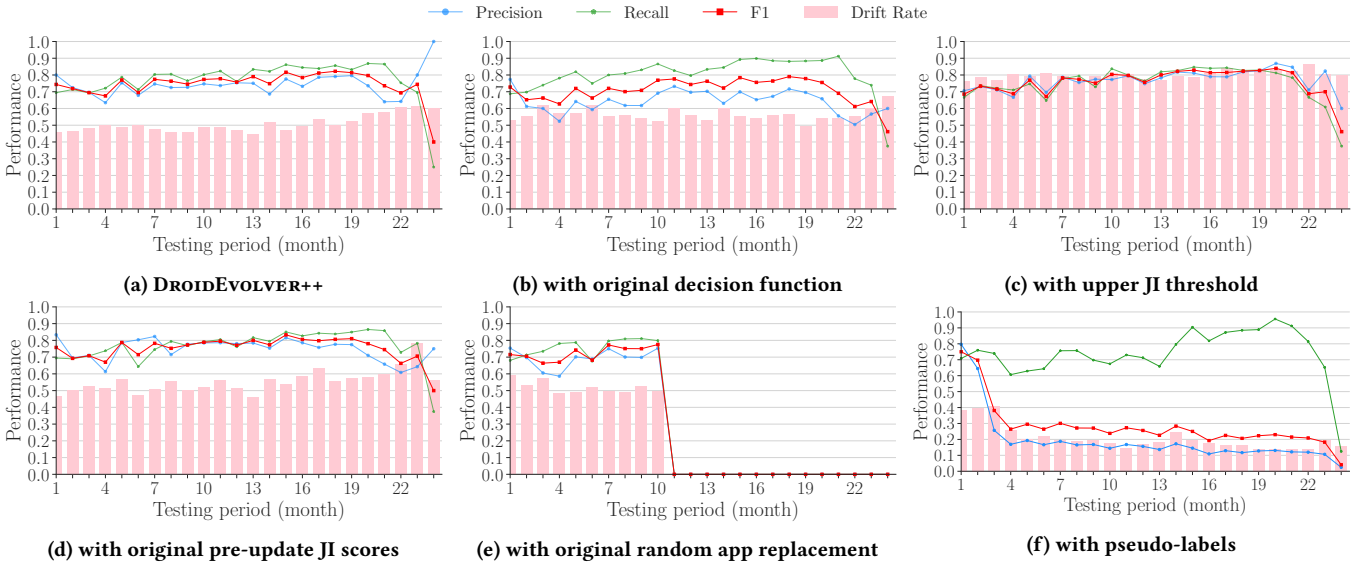
Additionally, the average drift rate increases slightly by 6.2%. With the drop in F_1 score, this suggests the JI comparison is marking more samples as drifting (i.e., the models are aging faster). Similarly, the performance degradation shows that the original weighted voting generates more mistakes than the modified majority vote of DroidEvolver++. In this experiment, the ensemble decision function is used to trigger the update, but is not used to produce a label for the update itself. As a result, though the decision may be incorrect, it will not poison the model pool for future predictions.

Upper JI Threshold. Next we reintroduce the upper JI threshold that DroidEvolver++ removes. As before, we use ground truth labels for updates. As shown in Figure 3c, the F_1 score is the most stable of the ablation settings. However, this is likely due to the increase in the number of updates caused by many more examples being marked as drifting—~80% over all test periods. As stated earlier, we aim to maintain as low a drift rate as possible to minimize the need for true labels. The average F_1 score of 0.75, only marginally different to the control, confirms our suspicion that the majority of predictions which have a JI above the upper threshold are actually high-confidence predictions—i.e., updating the model using their true label produces only minor gains in performance.

Updated App Buffer JI Scores. Next we deactivate the recomputation of JI scores for apps in the app buffer when the model updates. Here we see little change compared to the control, with comparable F_1 score, but more stable Precision and Recall. As the drift rate increases to a similar degree, we observe that there is some trade-off between the performance and the number of updates (labeling cost), similar to what was observed when the upper JI threshold was reintroduced.

App Buffer Replacements. Similarly we deactivate the requirement that app buffer replacements are class dependent. Still, ground truth labels are used for model updates in this experiment. As shown in Figure 3e, the evolution process terminates completely at the 11th month. This is because all malware in the app buffer has been replaced by goodware, and JI computation is no longer possible. To assess how typical this behavior is we repeat the experiment ten times and observe this phenomenon in four of those trials. During the ten months for which the updates succeed, the F_1 score is lower than that of the baseline on average, and the model exhibits the same tendency to overpredict malware as with the weighted voting.

We conclude that random replacement of the app buffer reduces the reliability of the system, especially given the imbalance between malware and goodware expected in the wild.



(a) DroidEvolver++ (b) with original decision function (c) with upper JI threshold

(d) with original pre-update JI scores (e) with original random app replacement (f) with pseudo-labels

Figure 3: Ablation study on DroidEvolver++ where each new component is in turn reverted back to its original form. All experiments in this study use ground truth labels except that shown in Figure 3f which uses pseudo-labels only. For those that use ground truth labels to perform updates, higher drift rates correspond to more updates and thus higher labeling costs.

Ground Truth vs. Pseudo-Labels. Finally, we evaluate whether, in light of the other improvements, DroidEvolver++ is able to operate using pseudo-labels, which is the core contribution of the original DroidEvolver. As shown in Figure 3f, the quality of the pseudo-labels is simply not high enough for this. The F_1 score drops significantly from 0.75 to 0.05 over 24 months, staying below 0.30 for most test periods. Precision decreases rapidly, and Recall increases slowly as the model poisons itself with spurious pseudo-labels and begins to overpredict the positive class. While the drift rate is very low, averaging 0.25, the low performance indicates that this is due to a failure of the system to recognize drifting objects.

This leads us to the conclusion that using a malware detector's own predicted labels as pseudo-labels is unlikely to be a viable solution to the trade-off between robustness to drift and labeling cost. In the following, we aim to explore this notion in more depth to analyze its strengths and limitations, and derive lessons learned.

5 THE LIMITS OF SELF-LEARNING IN MALWARE DETECTION

The core strength of DroidEvolver is the ability to use its own predicted labels as pseudo-labels for self-learning and eschew manual labeling entirely. However, our experiments in §3 and §4, show that the model can rapidly poison itself with catastrophic effects on the performance. Nevertheless, the proposal is still a tantalizing one, so in this section we examine in more depth whether higher quality pseudo-labels can be generated and if there are certain conditions that allow for self-learning with pseudo-labels to be more effective.

In the following, we use the same experimental setup as described in §3.1, initializing the model with data from 2014. However, due to the rapid performance degradation, we focus on 2015 alone as the test data. We use DroidEvolver++ exclusively to ensure the other DroidEvolver weaknesses (§3) do not act as confounding factors. As before, we measure Precision, Recall, F_1 score, and the drift rate, where the drift rate is the proportion of new inputs identified as drifting each period.

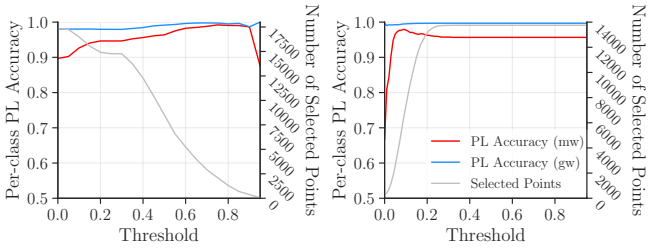
5.1 Uncertainty-Aware Pseudo-Label Selection

An important assumption for semi-supervised learning is that the decision boundary lies in low-density regions [12]. To achieve this, common pseudo-labeling methodologies aim to generate pseudo-labels using *high-confidence* predictions only [30, 52]. This intuition is straightforward to visualize for linear classifiers: high-confidence predictions are assigned to points furthest from the decision boundary. Such methods should reduce noise in the pseudo-labels which should mitigate—or at least delay—the onset of self-poisoning [22].

Recent work by Rizve et al. [45] extend this reasoning and observe that as many classifiers are poorly calibrated (i.e., their output probabilities do not align well with the true probabilities), confidence alone is insufficient as incorrect predictions may still be made with high confidence. To overcome this limitation, the authors propose selecting pseudo-labels which have high confidence but also low *prediction uncertainty*, as uncertainty can be interpreted as the quality of the calibration [29].

Rizve et al. [45] do not prescribe a specific measure of confidence and uncertainty in their work, and the metrics they use are specific to multiclass deep learning classifiers, however we can adapt the intuition to the ensemble of DroidEvolver++. Note that these measures are empirical approximations and should not be interpreted as having strong theoretical guarantees (in contrast to equivalent notions in Bayesian learning).

Confidence. As the confidence measure we use the average JI score across the non-aging models in the pool. Recall that the JI



(a) Confidence Tuning **(b) Uncertainty Tuning**

Figure 4: Tuning for confidence and uncertainty thresholds.

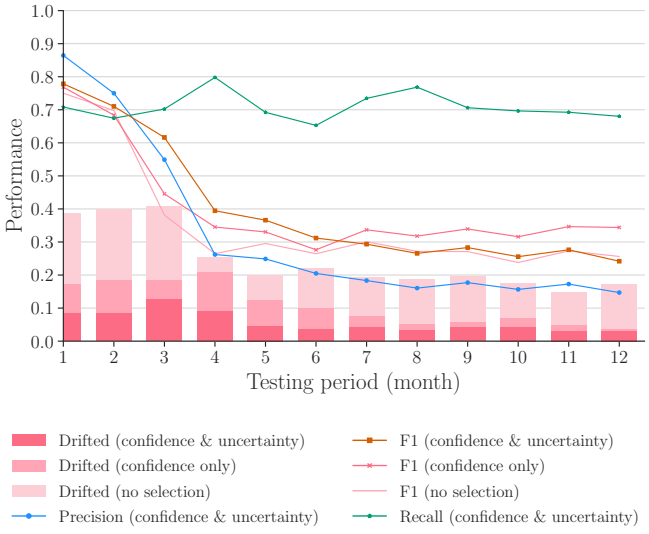


Figure 5: DroidEvolver++ with higher quality pseudo-labels selected using confidence and uncertainty thresholds.

score is computed as the proportion of apps in the app buffer, of the same class, which have decision scores greater than the given object. As the decision function for each model in the ensemble is simply the distance from the hyperplane, and the app buffer aims to be representative of the distribution as a whole, we can use JI as a proxy for distance from the hyperplane and thus confidence. The reason we cannot use the decision function outputs directly is because they are scaled differently for each model (§3), whereas the JI is normalized, similar to the credibility metric of Transcend [8, 23] and conformal prediction theory [57].

Uncertainty. As the uncertainty measure we use the standard deviation of the JI score between all non-aging models. If there is high uncertainty, we expect greater disagreement among models in the ensemble. Conversely if there is low uncertainty, we expect less disagreement. The motivation for using the JI score rather than the decision score directly is the same as for confidence. For both metrics we exclude aging models from the calculation as the decision of aging models is assumed to be untrustworthy.

Threshold Search. We aim to select pseudo-labels which are obtained with confidence above, and uncertainty below, a pair of thresholds. However, by filtering out low-quality pseudo-labels we also reduce the number of examples used to update the model which may also have adverse effects. The calibration process aims to find a suitable balance between these two variables.

To determine these thresholds we first split the training set into proper training set and calibration set at a ratio of 7:3 and use the proper training set to initialize the ensemble and the calibration set for the threshold search itself.

Figure 4 illustrates the threshold tuning process for both confidence (left) and uncertainty (right). Each plot shows the Accuracy of the pseudo-labels at different thresholds. As the benign class is the overwhelming majority, but is expected to exhibit less drift than the malicious class, we show Accuracy for each class separately. The gray line plot against the twin y-axis shows the number of points selected at that threshold.

For both metrics we see the Accuracy of the benign class stay above 98% throughout. For confidence (Figure 4a), Accuracy increases gradually for malware from 0.87 to 0.98 as the threshold increases from 0.0 to 0.8. Above this threshold the Accuracy decreases significantly, an artefact of the small number of selected pseudo-labels at this range. As a compromise between the number of selected pseudo-labels and the Accuracy, we select 0.5 as the confidence threshold. Note that this is stricter than the JI threshold used by DroidEvolver++ (0.3).

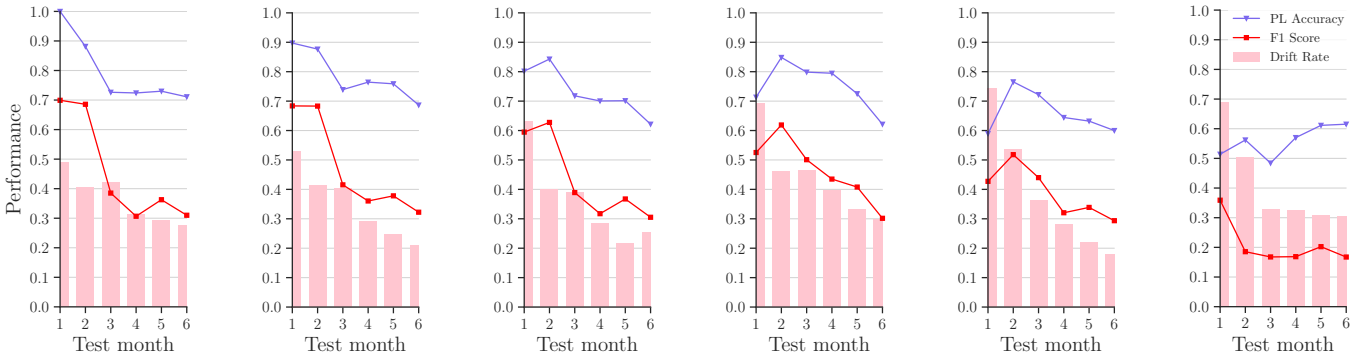
Uncertainty tuning is shown in Figure 4b. The calibration results show a similar trend although the distribution of selected pseudo-labels is skewed towards higher uncertainty values. We choose 0.1 as the threshold as this maximizes the Accuracy for both classes while retaining a reasonable number of selected pseudo-labels.

Results. Figure 5 illustrates the performance of DroidEvolver++ after the new thresholds are applied. The line without markers shows the original F_1 score of DroidEvolver++ using unfiltered pseudo-labels. As discussed previously, performance drops below 0.40 after two months.

We also consider the F_1 score when the confidence threshold is applied alone. The performance decay is still severe, stabilizing at ~0.35 F_1 score, but is slightly better than when there is no selection. In the majority of test periods, the drift rate is reduced by at least half of the original value, most notably in the first three months.

The other lines show the F_1 score, Precision, and Recall when both confidence and uncertainty thresholds are applied. The decay is delayed by a further month compared to using confidence alone, which shows that the uncertainty threshold is helpful in selecting better quality pseudo-labels. However, after the third month, the model begins to over-predict positive examples, with Recall staying relatively stable but Precision degrading. The drift rate is further reduced to less than 10% of all test objects for most test periods.

Notably, even though the performance decay is delayed when pseudo-labels are filtered this way, it is still eventually fatal. The ensemble decision function still requires a majority of the non-aging models to make correct predictions. However, it is likely that due to drift, inputs will inevitably be misclassified by the majority of the ensemble, which will cause the pseudo-labeling strategy to fail—that is, the low-density region of the decision boundary is eroded due to concept drift. This situation is more common to occur when the dataset distribution shifts suddenly [11].



(a) 0% noise (b) 10% noise (c) 20% noise (d) 30% noise (e) 40% noise (f) 50% noise

Figure 6: Exploration of pseudo-label error propagation in the first six months of the DroidEvolver++ evaluation. The update of the first month uses noisy ground truth labels where the true label is flipped with probability p% for p = 0, 10, 20, 30, 40, and 50. In the subsequent months, the update continues naturally using pseudo-labels. Additional plots are shown in Appendix A.

Given this, we conclude that while there may be hope for self-learning strategies in malware detection, it is likely better suited to a supporting role as the presence of concept drift requires externally generated up-to-date labels.

5.2 Tolerance to Pseudo-Label Noise

As we have seen, inevitable mistakes in the pseudo-labels can compound and quickly lead to self-poisoning. Here we examine how tolerant DroidEvolver++ is to this *pseudo-label noise* and how errors begin to propagate through the system and cause self-poisoning. Equally, this helps us determine if there is a certain quality of pseudo-labels for which a system can be self-sustaining.

In this experiment we use ground truth labels as pseudo-labels for the very first update in month 1. However, to simulate a certain proportion of incorrect pseudo-labels, whenever a model in the ensemble needs to be updated, we flip the label to be incorrect with some probability p. For example, a new object may appear with ground truth *benign* and be marked as drifting by a model. As the model is now considered aging, we must update it with the new object, but first we will flip the pseudo-label to *malicious* with probability p. After the very first update is completed, we continue the test phase as normal, using the usual generated pseudo-labels for months 2–6. Note that there is likely some naturally occurring label noise in the dataset, i.e., where the ground truth label is incorrect, but this should not significantly affect our results [19, 47]. Additionally we have removed grayware from the dataset—apps with 1–3 VirusTotal detections—to reduce this risk (see §3.1).

We repeat the experiment for different probabilities p: 0%, 10%, 20%, 30%, 40%, and 50%. For each, the model pool is initialized on the data from 2014 with the same starting app buffer, and we test using the first six months of 2015.

Figure 6 shows the performance of DroidEvolver++ for the different noise rates. The upper line depicts the Accuracy of the pseudo-labels while the lower line shows the F_1 score. The performance values in the second month is particularly informative for understanding how the initial pseudo-label error rate has affected the update. The drift rate increases from 49% where there is no error, up to 69% when half of the pseudo-labels are incorrect. This signals the instability of the system as the pseudo-label quality decreases. Interestingly, the Accuracy of the pseudo-labels in the second month is only minimally affected until the noise rate reaches 40% at which point performance suffers markedly. Notably, even when there are no mistakes in the initial pseudo-labels, errors compound at roughly similar rates so long as the majority of pseudo-labels are correct. We see a difference once noise levels reach 50%, at which point performance degrades almost immediately. This finding is further supported by additional results in Appendix A which include values for p = 60. Intriguingly, at higher noise levels the model is able to recover to a small degree, with pseudo-label Accuracy rising consistently for months 3–6—however it is not enough for the F_1 score to reach usable levels.

This result suggests that high-quality pseudo-labels alone are insufficient for generating further high-quality pseudo-labels in later months. Small error rates quickly compound to produce larger inaccuracies, which may explain why our pseudo-label selection in §5.1 was unable to further delay performance decay. Given these results it seems clear that manual intervention is needed to maintain the health of the system. However, we also observe that if the quality of pseudo-labels were to be sufficiently high *every month*, e.g., containing only 10% errors, the model would likely self-sustain itself. This suggests that a small amount of high-quality pseudo-labeled data may be able to augment manually labeled examples, for example, when labeling capacity is stretched. Similarly, when drift is shallow, as in the original DroidEvolver dataset, it is much more likely that the error rate of pseudo-labels will remain low.

6 DISCUSSION

We note that there are still limitations in DroidEvolver++ which would require moving further away from the DroidEvolver design. Notably, the inclusion of passive-aggressive algorithms in the model pool may limit the ability to self-learn, as these algorithms will only adjust their decision boundary when their prediction is incorrect, which will not occur if they agree with the majority vote that produces the pseudo-label.

Ultimately our results demonstrate that self-learning with pseudo-labels is intrinsically challenging in a drifting environment. If drift

is severe enough that a new instance is not correctly classified by any of the models in the ensemble, performance degradation will be inevitable without additional mitigations.

While the challenges are difficult to overcome, our results motivate several promising research directions for using pseudo-labels to combat concept drift in security tasks.

Alternative Pseudo-Labels. In this work we focus on the pseudo-labels as defined by DroidEvolver, i.e., self-learning with the predicted labels produced by the model itself. However, other pseudo-labels may be more stable and produce more promising results. For example, in *co-training*, multiple learners are used which each model the dataset in distinctly different ways [5, 10]. Having different 'views' of the data makes the ensemble more robust to drift in one particular representation—low quality pseudo-labels in one 'view' can be used to update a model that uses a different 'view', and vice-versa. Conversely, the models chosen by DroidEvolver are extremely similar and may degrade in similar ways such that they are not able to support one another.

Pseudo-Label Selection. We have shown that selecting higher quality labels can reduce poisoning effects and labeling cost, however by filtering pseudo-labels there is an inherent trade-off between the quality of the pseudo-labels and the amount available to use. Similarly, while the use of confidence and uncertainty thresholds is an improvement over the baseline, these metrics are still tied to the overall health of the system, similar to the pseudo-labels themselves. Using an additional framework dedicated to identifying drifting examples, such as CADE [61] or Transcend [8, 23], may help maintain the stability of the app buffer and may further help in identifying and rejecting low-quality pseudo-labels.

Complementary Approaches. Pseudo-labeling may be useful to complement other approaches such as active learning [49, 50]. Active learning is an extremely promising research avenue for reducing labeling burden as it has been shown that labeling just 1% of examples with a strategy such as uncertainty sampling [31] can significantly delay the onset of drift [41]. Pseudo-labeling may be used to augment a small amount of labeled data when labeling burden is strained, while the manually labeled examples ensure that the quality of pseudo-labels is kept high.

Robust Feature Spaces. Concept drift ultimately occurs in the feature space, and may be more or less severe depending on how the malware is represented [8]. As we have explored, pseudo-labels are more effective when the error rate of the generated pseudo-labels is low. As the error rate is a function of the severity of the drift, this motivates the development of more robust feature spaces (e.g., APIGraph [62] and Tong et al. [56]) which may then facilitate the successful use of pseudo-labels to reduce labeling cost.

7 RELATED WORK

Malware detection over time. Groundbreaking work by both Allix et al. [3] and Miller et al. [35] demonstrate how malware classifiers degrade over time and how training on "future" malware can bias evaluations. Pendlebury et al. [41] build on this work and identify new forms of temporal and spatial bias as well as exploring mitigation for time-related performance decay, including incremental learning, active learning, and classification with rejection (the third being explored more thoroughly by Jordaney et al. [23] and more recently by Barbero et al. [8]). One of the first methods to perform a temporal evaluation, MaMaDroid [33] periodically retrains the model on new labeled data once it becomes unusable and uses abstract APIs at the granularity of packages and families to reduce the effect of drift resulting from new API calls. Similarly, APIGraph [62] propose the augmentation of feature spaces with semantic graph-based features, which are more robust to concept drift. Our results from §5 motivate further research in this area of robust feature spaces, as pseudo-labeling strategies can be used more successfully when the drift is less severe and mistakes made by the pseudo-labeling mechanism are limited.

Online learning for malware detection. Two closely related approaches to DroidEvolver are DroidOL [39] and Casandra [38] which both use online learning to continually retrain the models. Both use ground truth labels for model updates, but must update with every new object, while DroidEvolver++ only updates models which are marked as aging. Furthermore, while DroidOL and Casandra each rely on a single learner, DroidEvolver's ensemble should mitigate bias introduced by any single detection method [59]. Additionally, neither implementations are publicly available which makes it difficult for new approaches to compare against them without significant engineering effort.

Drift identification. An orthogonal research direction to drift adaptation is drift identification. For these approaches, identifying drifting objects is the primary goal, after which they may be quarantined, explained, or sent for downstream processing. Transcend [8, 23] introduces the nonconformity measure on which DroidEvolver's JI calculation is based and uses it to reject low-quality predictions. CADE [61] focuses on explaining drift, using a distance-based metric to provide semantically meaningful explanations for new drifting objects. BBSE [32] focuses on prior probability shift (i.e., label shift) and is able to adjust classifiers to changes in the base rate. Also related are out-of-distribution detectors for adversarial examples [40, 48, 54] although these are largely confined to the computer vision domain.

8 AVAILABILITY

To support future efforts in malware drift adaptation, we release the code for DroidEvolver++ and our implementation of alternative pseudo-label selection methods at https://s2lab.cs.ucl.ac.uk/drift.

9 CONCLUSION

This work examined the use of pseudo-labels for combating concept drift in malware detection. Although pseudo-labels are a promising approach for avoiding the overhead of manual labeling, we identify several flaws in the strategy employed by the state-of-the-art drift adaptation technique, DroidEvolver [59]. Following this, we further explore the conditions under which pseudo-labels might be effective, investigating the impact of noisy labels and utilizing methods to ensure only high-quality pseudo-labels are used for updating the model. Ultimately, we conclude that the use of pseudo-labeling is still a promising solution to the overhead of manual labeling, but that great care must be taken when designing the update mechanism to avoid negative feedback loops and self-poisoning.

ACKNOWLEDGEMENTS

We thank the reviewers and our shepherd, Kathrin Grosse, for their constructive feedback. This research has been supported in part by the UK EP/P009301/1 EPSRC research grant and by the China Scholarship Council (CSC) of the Ministry of Education, P.R. China.

REFERENCES

[1] Hojjat Aghakhani, Fabio Gritti, Francesco Mecca, Martina Lindorfer, Stefano Ortolani, Davide Balzarotti, Giovanni Vigna, and Christopher Kruegel. 2020. When Malware is Packin' Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*.

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

[3] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2015. Are Your Training Datasets Yet Relevant? - An Investigation into the Importance of Timeline in Machine Learning-Based Malware Detection. In *ESSoS (Lecture Notes in Computer Science, Vol. 8978)*. Springer, 51–67.

[4] Hyrum S. Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. 2018. Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning. *CoRR* abs/1801.08917 (2018).

[5] Giuseppina Andresini, Feargus Pendlebury, Fabio Pierazzi, Corrado Loglisci, Annalisa Appice, and Lorenzo Cavallaro. 2021. INSOMNIA: Towards Concept-Drift Robustness in Network Intrusion Detection. In *Proc. of the ACM Workshop on Artificial Intelligence and Security (AISec)*.

[6] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *Proc. of the USENIX Security Symposium*.

[7] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*. The Internet Society.

[8] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2020. Transcending Transcend: Revisiting Malware Classification with Conformal Evaluation. *CoRR* abs/2010.03856 (2020).

[9] Avrim Blum. 1998. On-line algorithms in machine learning. In *Online algorithms*. Springer, 306–325.

[10] Avrim Blum and Tom M. Mitchell. 1998. Combining Labeled and Unlabeled Data with Co-Training. In *Proc. of the ACM Conference on Learning Theory (COLT)*.

[11] Dariusz Brzezinski and Jerzy Stefanowski. 2013. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems* 25, 1 (2013), 81–94.

[12] Olivier Chapelle and Alexander Zien. 2005. Semi-Supervised Classification by Low Density Separation. In *AISTATS*. Society for Artificial Intelligence and Statistics.

[13] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Machine Learning* 20, 3 (1995).

[14] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online Passive-Aggressive Algorithms. *Journal of Machine Learning Research (JMLR)* (2006).

[15] Amit Deo, Santanu Kumar Dash, Guillermo Suarez-Tangil, Volodya Vovk, and Lorenzo Cavallaro. 2016. Prescience: Probabilistic Guidance on the Retraining Conundrum for Malware Detection. In *Proc. of the ACM Workshop on Artificial Intelligence and Security (AISec)*.

[16] Anthony Desnos. [n. d.]. Androguard. Reverse engineering, Malware and goodware Analysis of Android applications. https://github.com/androguard. Accessed: May 2019.

[17] Mark Dredze, Koby Crammer, and Fernando Pereira. 2008. Confidence-weighted linear classification. In *Proc. of the International Conference on Machine Learning (ICML)*.

[18] John C. Duchi, Elad Hazan, and Yoram Singer. 2010. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. In *Proc. of the ACM Conference on Learning Theory (COLT)*.

[19] B. Frenay and M. Verleysen. 2014. Classification in the Presence of Label Noise: A Survey. *IEEE Transactions on Neural Networks and Learning Systems* (2014).

[20] Robert M. French. 1999. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences* 3, 4 (1999).

[21] João Gama, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *Comput. Surveys* (2014).

[22] Yves Grandvalet and Yoshua Bengio. 2004. Semi-supervised Learning by Entropy Minimization. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[23] Roberto Jordaney, Kumar Sharad, Santanu K. Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *Proc. of the USENIX Security Symposium*.

[24] Jinho Jung, Chanil Jeon, Max Wolotsky, Insu Yun, and Taesoo Kim. 2017. AVPASS: Leaking and Bypassing Antivirus Detection Model Automatically. In *Black Hat USA Briefings (Black Hat USA)*. Las Vegas, NV.

[25] Alex Kantchelian, Sadia Afroz, Ling Huang, Aylin Caliskan Islam, Brad Miller, Michael Carl Tschantz, Rachel Greenstadt, Anthony D. Joseph, and J. D. Tygar. 2013. Approaches to adversarial drift. In *Proc. of the ACM Workshop on Artificial Intelligence and Security (AISec)*.

[26] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler L. Hayes, and Christopher Kanan. 2018. Measuring Catastrophic Forgetting in Neural Networks. In *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*.

[27] Amin Kharraz, Sajjad Arshad, Collin Mulliner, William K. Robertson, and Engin Kirda. 2016. UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. In *Proc. of the USENIX Security Symposium*.

[28] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)* 114, 13 (2017), 3521–3526.

[29] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[30] Dong-Hyun Lee. 2004. Pseudo-Label: The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks. In *Proc. of the ICML Workshop on Challenges in Representation Learning (WREPL)*.

[31] David D. Lewis and William A. Gale. 1994. A Sequential Algorithm for Training Text Classifiers. In *SIGIR*. ACM/Springer, 3–12.

[32] Zachary C. Lipton, Yu-Xiang Wang, and Alexander J. Smola. 2018. Detecting and Correcting for Label Shift with Black Box Predictors. In *Proc. of the International Conference on Machine Learning (ICML)*.

[33] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon J. Ross, and Gianluca Stringhini. 2017. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*.

[34] Michael McCloskey and Neal J. Cohen. 1989. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. Psychology of Learning and Motivation, Vol. 24. Academic Press, 109–165.

[35] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, Anthony D. Joseph, and J. D. Tygar. 2016. Reviewer Integration and Performance Measurement for Malware Detection. In *Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*.

[36] Jose G. Moreno-Torres, Troy Raeder, Rocío Alaíz-Rodríguez, Nitesh V. Chawla, and Francisco Herrera. 2012. A unifying view on dataset shift in classification. *Pattern Recognition* (2012).

[37] Stylianos Moschoglou, Athanasios Papaioannou, Christos Sagonas, Jiankang Deng, Irene Kotsia, and Stefanos Zafeiriou. 2017. AgeDB: The First Manually Collected, In-the-Wild Age Database. In *CVPR Workshops*. IEEE Computer Society, 1997–2005.

[38] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, and Yang Liu. 2017. Context-Aware, Adaptive, and Scalable Android Malware Detection Through Online Learning. *IEEE Transactions on Emerging Topics in Computational Intelligence (TETCI)* (2017).

[39] Annamalai Narayanan, Yang Liu, Lihui Chen, and Jinliang Liu. 2016. Adaptive and scalable Android malware detection through online learning. In *Proc. of the International Joint Conference on Neural Network (IJCNN)*.

[40] Nicolas Papernot and Patrick D. McDaniel. 2018. Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning. *CoRR* abs/1803.04765 (2018). arXiv:1803.04765

[41] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *Proc. of the USENIX Security Symposium*.

[42] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. 2020. Intriguing Properties of Adversarial ML Attacks in the Problem Space. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*.

[43] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. 2009. *Dataset Shift in Machine Learning*. The MIT Press.

[44] Mamshad Nayeem Rizve, Kevin Duarte, Yogesh Singh Rawat, and Mubarak Shah. 2021. In Defense of Pseudo-Labeling: An Uncertainty-Aware Pseudo-label Selection Framework for Semi-Supervised Learning. In *Proc. of the International Conference on Learning Representations (ICLR)*.

[45] Mamshad Nayeem Rizve, Kevin Duarte, Yogesh S Rawat, and Mubarak Shah. 2021. In defense of pseudo-labeling: An uncertainty-aware pseudo-label selection framework for semi-supervised learning. *arXiv preprint arXiv:2101.06329* (2021).

[46] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. 2018. Microsoft Malware Classification Challenge. *CoRR* abs/1802.10135 (2018).

[47] Aleieldin Salem, Sebastian Banescu, and Alexander Pretschner. 2021. Maat: Automatically Analyzing VirusTotal for Accurate Labeling and Effective Malware Detection. *ACM Transactions on Privacy and Security (TOPS)* (2021).

[48] Vikash Sehwag, Arjun Nitin Bhagoji, Liwei Song, Chawin Sitawarin, Daniel Cullina, Mung Chiang, and Prateek Mittal. 2019. Better the Devil you Know: An Analysis of Evasion Attacks using Out-of-Distribution Adversarial Examples. *CoRR* abs/1905.01726 (2019).

[49] Burr Settles. 2009. *Active Learning Literature Survey.* Computer Sciences Technical Report 1648. University of Wisconsin–Madison.

[50] Burr Settles. 2012. *Active Learning.* Morgan & Claypool Publishers.

[51] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. 2011. Weisfeiler-Lehman Graph Kernels. *Journal of Machine Learning Research (JMLR)* (2011).

[52] Weiwei Shi, Yihong Gong, Chris Ding, Zhiheng Ma, Xiaoyu Tao, and Nanning Zheng. 2018. Transductive Semi-Supervised Deep Learning Using Min-Max Features. In *ECCV (5) (Lecture Notes in Computer Science, Vol. 11209).* Springer, 311–327.

[53] Anshuman Singh, Andrew Walenstein, and Arun Lakhotia. 2012. Tracking concept drift in malware families. In *Proc. of the ACM Workshop on Artificial Intelligence and Security (AISec).*

[54] Angelo Sotgiu, Ambra Demontis, Marco Melis, Battista Biggio, Giorgio Fumera, Xiaoyi Feng, and Fabio Roli. 2020. Deep neural rejection against adversarial examples. *EURASIP Journal on Information Security* 2020 (2020).

[55] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Lorenzo Cavallaro. 2017. The Evolution of Android Malware and Android Analysis Techniques. *Comput. Surveys* (2017).

[56] Liang Tong, Bo Li, Chen Hajaj, Chaowei Xiao, Ning Zhang, and Yevgeniy Vorobeychik. 2019. Improving Robustness of ML Classifiers against Realizable Evasion Attacks Using Conserved Features. In *Proc. of the USENIX Security Symposium.*

[57] Vladimir Vovk, Ilia Nouretdinov, Valery Manokhin, and Alexander Gammerman. 2018. Cross-conformal predictive distributions. In *Proc. of the PMLR Workshop on Conformal Prediction and its Applications (COPA)*, Vol. 91. PMLR.

[58] Lin Xiao. 2010. Dual Averaging Methods for Regularized Stochastic Learning and Online Optimization. *Journal of Machine Learning Research (JMLR)* (2010).

[59] Ke Xu, Yingjiu Li, Robert H. Deng, Kai Chen, and Jiayun Xu. 2019. DroidEvolver: Self-Evolving Android Malware Detection System. In *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P).*

[60] Weilin Xu, Yanjun Qi, and David Evans. 2016. Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers. In *Proc. of the Network and Distributed System Security Symposium (NDSS).*

[61] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadehand, Xinyu Xing, and Gang Wang. 2021. CADE: Detecting and Explaining Concept Drift Samples for Security Applications. In *Proc. of the USENIX Security Symposium.*

[62] Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. 2020. Enhancing State-of-the-art Classifiers with API Semantics to Detect Evolved Android Malware. In *Proc. of the ACM Conference on Computer and Communications Security (CCS).*

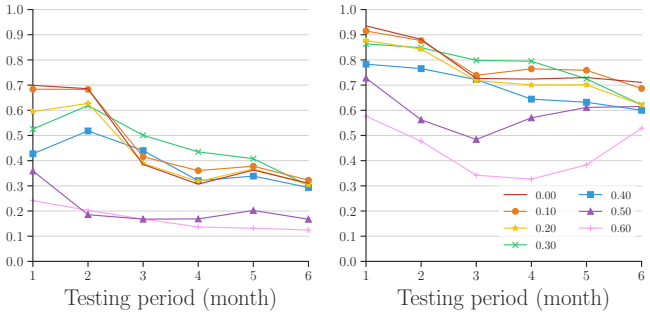
[63] Martin Zinkevich. 2003. Online Convex Programming and Generalized Infinitesimal Gradient Ascent. In *Proc. of the International Conference on Machine Learning (ICML).*

A ADDITIONAL RESULTS FOR NOISY PSEUDO-LABELS

In §5.2 we investigate how errors in the pseudo-label generation compound to create self-poisoning effects leading to the system becoming unusable. Here we offer an alternative view by overlaying the results at different noise levels. We include an additional result for a noise rate of 60% which further suggests that degradation is expedited when the majority of pseudo-labels are incorrect.

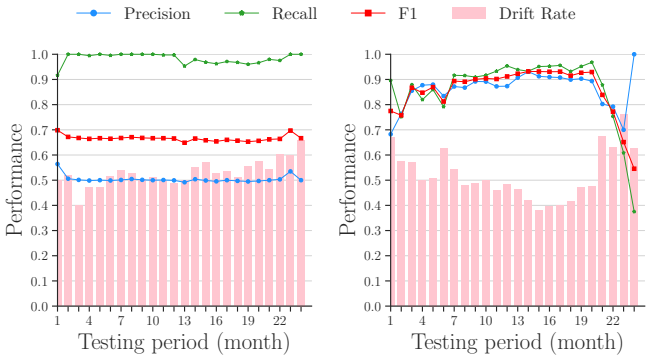
B ADDITIONAL RESULTS WITH BALANCED CLASS RATIO

For clarity, and to avoid introducing too many axes of comparison, our experiments focus on the imbalanced class setting evidenced by technical reports of 10% Android malware in the wild [41]. However, there may be domains in which testing on data with a balanced class ratio is a better approximation of the distribution in the wild. Figure 8 shows a final comparison between DroidEvolver and DroidEvolver++ applied the balanced setting described in §3.2.



(a) F_1 Score **(b) Pseudo-Label Accuracy**

Figure 7: Pseudo-label error propagation in the first six months of the DroidEvolver++ evaluation. The update of the first month uses noisy ground truth labels where the true label is flipped with probability p% for different values of p. In the subsequent months, the update continues naturally using pseudo-labels. See also Figure 6 for drift rates.



(a) DroidEvolver **(b) DroidEvolver++**

Figure 8: Final comparison between DroidEvolver and DroidEvolver++ applied to the balanced setting described in §3.2 with ~50% malware in the test set.