DRMD: Deep Reinforcement Learning for Malware Detection under Concept Drift

Shae McFadden[†‡§], **Myles Foley**[‡], **Mario D'Onghia**[§],
Chris Hicks[‡], **Vasilios Mavroudis**[‡], **Nicola Paoletti**[†], **Fabio Pierazzi**[§]
[†]King's College London, [‡]The Alan Turing Institute, [§]University College London

Abstract

Malware detection in real-world settings must deal with evolving threats, limited labeling budgets, and uncertain predictions. Traditional classifiers, without additional mechanisms, struggle to maintain performance under concept drift in malware domains, as their supervised learning formulation cannot optimize when to defer decisions to manual labeling and adaptation. Modern malware detection pipelines combine classifiers with monthly active learning (AL) and rejection mechanisms to mitigate the impact of concept drift. In this work, we develop a novel formulation of malware detection as a one-step Markov Decision Process and train a deep reinforcement learning (DRL) agent, simultaneously optimizing sample classification performance and rejecting high-risk samples for manual labeling. We evaluated the joint detection and drift mitigation policy learned by the DRL-based Malware Detection (DRMD) agent through time-aware evaluations on Android malware datasets subject to realistic drift requiring multi-year performance stability. The policies learned under these conditions achieve a higher Area Under Time (AUT) performance compared to standard classification approaches used in the domain, showing improved resilience to concept drift. Specifically, the DRMD agent achieved an average AUT improvement of 8.66 and 10.90 for the classification-only and classification-rejection policies, respectively. Our results demonstrate for the first time that DRL can facilitate effective malware detection and improved resiliency to concept drift in the dynamic setting of Android malware detection.

AAAI-26: This is an extended version of the paper *DRMD: Deep Reinforcement Learning for Malware Detection under Concept Drift* published at AAAI 2026.

Code: https://github.com/s2labres/DRMD

1 Introduction

Malware poses a significant and ever-evolving threat to mobile devices, personal computers, and enterprise systems. Every day, tens of thousands of new Android and Windows applications appear, far outpacing the capacity for manual analysis. Automated malware detection using machine learning approaches is therefore crucial to ensure user security. However, unlike domains that can directly process raw data, malware detection relies on vectorized abstractions to represent applications for classification. Therefore, feature extraction plays a pivotal role in determining the overall performance of a malware detection system. As a result, malware representations have been addressed using various feature spaces [Arp et al., 2014, Onwuzurike et al., 2019, Zhang et al., 2020b, Li et al., 2021], each incorporating different information (e.g. API calls or app permissions) into feature vectors for malware detection classifiers.

Traditional supervised classifiers struggle to maintain malware detection performance in isolation as benign and malicious behaviors evolve over time [Pendlebury et al., 2019, Miller et al., 2016, Allix et al., 2015]. While classification techniques from domains such as computer vision are effective on stationary data, they struggle to adapt to the dynamic nature of malware that actively seeks to evade detection [Kan et al., 2024]. Approaches such as active learning (AL) to periodically retrain on a subset of new samples [Settles, 2009, Chen et al., 2023] and classification with rejection to selectively abstain from classification [Jordaney et al., 2017, Barbero et al., 2022, Yang et al., 2021] can mitigate performance degradation caused by evolving threats.

In this work, we unify classification, AL, and rejection within a single decision framework using deep reinforcement learning (DRL). Prior work from Coscia *et al.* 2024 extended the Imbalanced Classification MDP (ICMDP) Lin et al. [2020] for Windows PE malware family classification to address class imbalance. While achieving impressive performance, prior work Coscia et al. [2024], Binxiang et al. [2019] does not unify classification and rejection in a single decision process, nor does it consider concept drift introduced by malware evolution. Furthermore, episodes in the ICMDP formulation involve classifying multiple samples: this introduces invalid sequential dependencies through the state transitions between samples.

We present the first approach in this domain that reformulates malware detection as a *one-step MDP*, or *contextual bandit*, where each episode corresponds to a single sample, thus avoiding spurious dependencies between states. Leveraging this formulation, we train a DRL-based Malware Detection (DRMD) agent that directly optimizes the long-term trade-off between detection accuracy, labeling cost, and misclassification risk under realistic concept drift. The policies learned by DRMD provide statistically significant gains over the best baseline in 75% of settings evaluated across different: feature spaces, datasets, and drift scenarios. These results demonstrate that DRMD is a promising enhancement to current malware detection pipelines. In summary, this paper makes the following key contributions:

- *Malware Detection MDP.* We introduce a novel one-step MDP formulation of malware detection that treats each sample independently, named the MD-MDP. This rectifies the spurious dependencies between samples present in prior work.
- *Integrated Rejection.* We extend the MD-MDP with a rejection action and reward structure, which enables learning a policy that balances classification accuracy and misclassification risk.
- *Time-Aware Evaluations.* We demonstrate that the policies learned from the MD-MDP consistently outperform Android malware detection approaches considered across multiple feature spaces and datasets in time-aware evaluations, achieving a relative AUT gain of 8.66 and 10.90 for the classification-only and classification-rejection policies, respectively.

2 Related Work

DRL has demonstrated excellent performance in a variety of decision-making and classification tasks. Lin *et al.* 2020 formulated ICMDP, forming states as samples, actions as predicted labels, rewarding +1 for correct and -1 for incorrect classification of minority class samples with majority class sample rewards down-scaled according to the class distribution. The use of a Deep Q-Network (DQN) for ICMDP (DQN-imb) has been successful on images in the healthcare domain, including stroke detection [Zuo et al., 2024], COVID-19 screening [Yang et al., 2024], and other healthcare tasks [Zhou et al., 2021, Jayaprakash et al., 2023, Usha Nandhini and Dharmarajan, 2023]. However, the application of DRL to malware detection is underexplored. SINNER Coscia et al. [2024] builds on the ICMDP framework and extends its reward function to the multi-class setting, employing a Dueling Double DQN to classify the family of Windows PE malware samples. Earlier work from Binxiang *et al.* 2019 similarly applies DQN to detect Windows PE malware. Although these approaches improve performance (c.f. Lin *et al.* 2020) by harnessing DRL, they do not consider concept drift, a core problem in malware detection. Furthermore, these approaches are not evaluated in conjunction with AL or rejection mechanisms. In contrast, our work combines classification, rejection, and AL for an agent to learn a single policy to effectively detect malware over time.

3 Android Malware Detection

A core challenge of malware detection arises from the scale of new samples that emerge daily and the limited capacity for manual review. Onwuzurike *et al.* 2019 estimated that an upper bound of 10,000 applications are submitted to the Google Play Store each day, while Miller *et al.* 2016 assumed the manual review capacity of an average company at only 80 samples per day. This significant gap necessitates the use of machine learning (ML) to accurately and consistently classify new applications based on previously labeled data.

Concept Drift A key limitation of traditional Supervised Learning (SL) classifiers is their reliance on the assumption that data follows an Independent and Identically Distributed (IID) structure [Bishop and Nasrabadi, 2006]. However, this assumption does not hold in the domain of Android malware, due to the continuous evolution of both benign applications (goodware) and malware over time [Pendlebury et al., 2019]. Goodware evolves naturally as software developers introduce new features, while malware adapts to better mimic goodware and evade detection. This continual evolution leads to a divergence between training and testing data that expands over time, a phenomenon known as concept drift [Moreno-Torres et al., 2012]. This performance degradation in malware classifiers poses a significant threat to the security of end users [Allix et al., 2015, Miller et al., 2016, Pendlebury et al., 2019, Kan et al., 2024].

AL and Rejection Mechanisms One of the primary strategies to mitigate concept drift is *active learning* (AL), which involves selecting a subset of new samples for manual labeling and periodic retraining [Settles, 2009]. The strategy for selecting the most informative samples is the key differentiator between selection approaches. However, manual labeling in the malware domain is resource-intensive. Therefore, AL techniques must carefully balance selecting the most informative samples while minimizing labeling costs. In contrast, rejection strategies identifies samples with a high probability of misclassification. These rejected samples are quarantined for manual inspection or other investigate processes. Both AL and rejection mechanisms in Android malware detection are typically employed periodically to mitigate concept drift, with common evaluations considering monthly periods for these mechanisms Kan et al. [2024], Chen et al. [2023], McFadden et al. [2024a]. In this work, we focus on the use of uncertainty-based AL and rejection [Settles, 2009]. The integration of other methods, such as conformal evaluation Jordaney et al. [2017], Barbero et al. [2022] or explanation-based approaches Yang et al. [2021], into the reward design of DRL-based detectors presents an interesting but non-trival direction for future work. This paper aims to establish a foundational basis for DRL-based malware detection integrating AL and rejection.

Time-Aware Evaluation Since concept drift in malware detection is inherently time-dependent, time-aware evaluations are essential to assess classifier performance in real-world scenarios. A valid time-aware evaluation must adhere to the following constraints [Pendlebury et al., 2019, Kan et al., 2024]. **(C1)** Training samples must always precede testing samples. **(C2)** All testing samples must be drawn from the same time window. **(C3)** The malware distribution in the testing set must reflect a realistic distribution. Violating these constraints can introduce biases that artificially inflate the performance of the classifier, leading to unrealistic expectations in practical deployments. In order to evaluate the performance of detectors, we use the Area Under Time (AUT) metric, introduced by

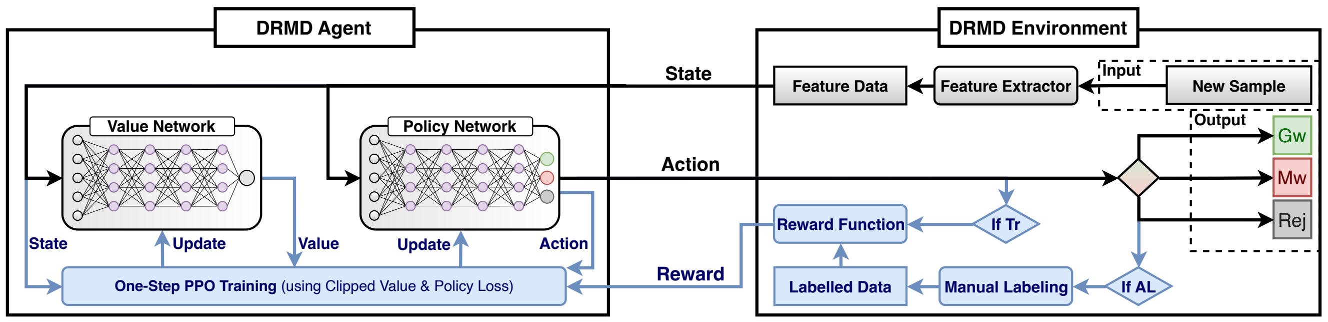

Figure 1: Overview figure showing the one-step classification of a sample using DRMD, with training components in blue.

Pendlebury *et al.*2019, and defined as:

$$AUT(P,N) = \frac{1}{N-1}\sum_{k=1}^{N-1}\frac{[P(X_{k+1})+P(X_k)]}{2} \quad (1)$$

where: $P(X_k)$ is the performance on samples X_k using metric P, with P being the F_1 score in our paper.

4 DRMD Design

DRMD is a DRL-based approach that, in its most complete form, unifies classification, AL, and rejection in a single malware detection pipeline. DRMD first formulates the Malware Detection MDP (MD-MDP), then uses a deep neural network to learn an effective detection policy. Concretely, MD-MDP extends ICMDP Lin et al. [2020] with corrected episode definitions and integrates the mechanisms necessary for effective malware detection in practice. See Figure 1 for an overview of DRMD.

Motivation The application of DRL to detection tasks has been shown to improve performance in more complex domains with class imbalance Zhou et al. [2021], Zuo et al. [2024], Yang et al. [2024]. However, the use of DRL in malware detection has seen little work Coscia et al. [2024], Binxiang et al. [2019] likely due to the drift and class imbalance of the domain. A policy for malware detection should consider the larger decision-making process of classification, rejection, and AL; thus allowing for the optimization of performance to be rejection-aware. Finally, the agent must adapt to concept drift within the domain constraints. As a result, DRL policies must move beyond the stationary perspectives of prior problem domains. Our MD-MDP and agent design achieve these considerations through a one-step episodic formulation that leverages an expanded reward function for temporal robustness and the integration of rejection.

4.1 MD-MDP

Prior work has applied the ICMDP to several domains such as malware family classification Coscia et al. [2024]. However, the formulation of ICMDP assumes state transitions between classification samples, thus treating independent samples as sequentially dependent. To overcome this, we cast malware detection as a one-step MDP (*aka* contextual bandit).

Episodes In our formulation, each episode consists of a single step: classifying or rejecting one sample. This corrects the state transitions introduced by the ICMDP formulation by isolating each sample into its own episode. Therefore, our MD-MDP formulation ensures that policy updates do not introduce spurious correlations as a result of artificial dependencies between independent samples.

States For episode e, the sample x^e consists of d features, where $F \subseteq \mathbb{R}^d$ represents the feature space, hence $x^e \in F$. The state consists of the features of the current sample; therefore, the state space and state are defined as $S = F$ and $s = x^e$, respectively.

Actions A chosen action $a \in A$ represents the decision of the agent for the current state s. The action space is $A = \{a_{gw}, a_{mw}, a_{rej}\}$, where a_{gw} classifies the sample as goodware, a_{mw} as malware and a_{rej} rejects the sample, abstaining from classification. *Active learning* is integrated by retraining on samples for which the agent selected a_{rej}.

Rewards The rewards need to encapsulate the dynamics of classifications and rejection. The three underlying components of classification in the domain are the accuracy of the predictions, the natural imbalance in the distribution of the samples, and the evolution of samples over time.

For *prediction accuracy*, correct classifications receive a positive reward and incorrect classifications incur a penalty. Given the true label $y^e \in \{0,1\}$ for a state $y(s^e) = y^e$, the accuracy reward is:

$$R_{\text{acc}}(s,a) = \begin{cases} +1 & \text{if } a = y(s), \\ -1 & \text{if } a \neq y(s). \end{cases} \quad (2)$$

For *class imbalance*, since Android malware samples are the minority class ($\hat{\sigma} \approx 10\%$) [Pendlebury et al., 2019], the rewards and penalties for malware samples ($y(s) = 1$) are proportionally increased. We use upscaling instead of majority class reward downscaling used by Lin *et al.* 2020 as we introduce additional scaling factors for temporal and rejection values. Therefore, the malware reward upscaling is defined as:

$$R_{imb}(s) = \begin{cases} {}^{1}/_{\hat{\sigma}} & \text{if } y(s) = 1, \\ 1 & \text{if } y(s) = 0. \end{cases} \quad (3)$$

For *temporal robustness*, we upscale the rewards for the current state s by it relative temporal position T_{pos}, in months, compared to the the first training state s^0, such that:

$$R_{tmp}(s) = {}^1\!/\!_2 \max\left(1,\ T_{pos}(s) - T_{pos}(s^0)\right) \tag{4}$$

For *rejection*, the reward is comprised of a flat rejection cost (R_{cost}), the inverted reward of the next most likely action ($R_{nml} = R_{clf}(s, a \neq a_{rej})$), as well as a rejection scaling factor based on the value of R_{nml} and $\hat{\sigma}$. The scaling factor balances the rewards for classifying and rejecting malware predictions while incentivizing rejection of uncertain goodware predictions. Together, these components penalize interference with correct classifications and incentivize rejection of incorrect classifications. The rejection reward is defined as:

$$R_{rej}(s,a) = \begin{cases} -R_{cost} - \frac{1}{\hat{\sigma}} R_{nml} & \text{if } R_{nml} \leq 0 \\ -R_{cost} - \hat{\sigma} R_{nml} & \text{else.} \end{cases} \tag{5}$$

The classification (R_{clf}) and complete DRMD (R_{cr}) reward functions are defined as:

$$R_{clf}(s,a) = R_{acc}(s,a) \times R_{tmp}(s) \times R_{imb}(s) \tag{6}$$

$$R_{cr}(s,a) = \begin{cases} R_{clf}(s,a) & \text{if } a \in \{a_{gw}, a_{mw}\} \\ R_{rej}(s,a) & \text{if } a = a_{rej} \end{cases} \tag{7}$$

Using our formulation, we create two types of policies for evaluation in Section 6. First, a Classification-Only Policy (π_{clf}) using the classification actions (a_{gw} and a_{mw}), and reward $R_{clf}(s,a)$. Second, a Classification-Rejection Policy (π_{cr}) that uses all actions, and reward $R_{cr}(s,a)$.

4.2 Agent Architecture

We use Proximal Policy Optimization (PPO) [Schulman et al., 2017], modified for one-step episodes, to train the DRMD agent. As the clipped update and actor-critic architecture of PPO allows efficient scaling to high-dimensional environments while maintaining training stability [Nguyen and Reddi, 2021]. Additionally, PPO has seen successful application to several other security domains such as automated cyber defense Foley et al. [2022a,b], Hicks et al. [2023], Bates et al. [2023], vulnerability discovery McFadden et al. [2024b], Foley and Maffeis [2025], and adversarial machine learning Tsingenopoulos et al. [2022, 2024]. The on-policy nature of PPO and its use of rollout updates facilitate learning on the most recent samples compared to experience replay buffers, used in off-policy methods, which would be more susceptible to concept drift.

While we have formulated the MD-MDP as a one-step MDP or contextual bandit problem, existing deep contextual bandits (DCBs) are not well-suited to our setting. First, DCBs assume stationary sub-Gaussian (bounded) reward noise Zhou et al. [2020], an assumption violated in malware domains due to concept drift. Existing work has explored the use of DCBs to facilitate AL Tae et al. [2024], Ban et al. [2024] but not the application of AL to DCBs.

The agent employs separate actor and critic neural networks, each consisting of four hidden layers with 512 neurons per layer, using LeakyReLU activations and dropout for regularization. The full architecture and hyperparameters are detailed in the Appendix C. During training, each epoch consists of one episode per training sample using the current policy. The collected states, actions, and rewards are then used to compute the clipped policy and value losses to update the actor and critic networks over mini batches. When new samples become available after the initial training, the agent will be fine-tuned on a sliding window of the newest samples (instead of on all samples) to focus updates on adapting to changes in the domain.

5 Experimental Settings

Datasets We use two Android malware datasets in our evaluation: Transcendent [Barbero et al., 2022] and Hypercube [Chow et al., 2025]. Transcendent provides an established Android malware dataset Kan et al. [2024], McFadden et al. [2024a], Herzog et al. [2025], while Hypercube is an updated dataset with newer malware samples. Transcendent consists of $259,230$ applications collected between 2014 and 2018. Hypercube contains $159,839$ applications between 2021 and 2023. For both datasets, we use the first year for training and subsequent years for testing, allowing evaluation of performance under concept drift. Both datasets contain a malware distribution of $\approx 10\%$ according to the spatial constraint of Tesseract [Pendlebury et al., 2019], and the samples have been labeled using a standard VirusTotal detection threshold ($VTT = 4$), according to the recommendations of Pendlebury et al. [2019], Kan et al. [2024].

Feature Spaces To ensure that the performance of DRMD is independent of a single feature representation, we evaluate it using two different Android malware feature spaces *for each dataset*. We use feature spaces that enable us to evaluate our approach across varying amounts of information per sample. First, we use the *Drebin* Arp et al. [2014] feature space, consisting of a sparse, high-dimensional binary vector, capturing static features extracted from the application's metadata and bytecode, including permissions, API calls, and network addresses. For our experiments, we follow Kan et al. in selecting the $10,000$ most informative Drebin features as it maintains performance while managing computational complexity. This representation has become the standard in Android malware detection [Pendlebury et al., 2019, Chen et al., 2023, Chow et al., 2025]. The second feature space that we use is *Ramda* Li et al. [2021], a compact feature space designed for adversarial robustness, which focuses on three critical static attributes: permissions, intent actions, and sensitive API calls. Each application is encoded as a 379-dimensional binary vector that captures the presence or absence of these features.

Baselines To compare our novel MD-MDP with the ICMDP, we evaluate the two model designs using the same (our) DRL agent architecture, thereby isolating the effect of this factor on the performance. The difference in performance is denoted as ΔIC. Moreover, we compare DRMD with a version of

Table 1: **Classification-Only Policy.** The AUT performance of DRMD across different settings, datasets, and feature spaces are reported alongside the performance differences compared to the best baseline (ΔBase), the ICMDP (ΔIC), and the DCBs (ΔDCB). All results are conducted over five runs and paired t-tests are used for statistical significance testing (* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$).

AL	Rej	Hypercube-Drebin				Hypercube-Ramda				Transcendent-Drebin				Transcendent-Ramda			
		AUT	ΔBase	ΔIC	ΔDCB	AUT	ΔBase	ΔIC	ΔDCB	AUT	ΔBase	ΔIC	ΔDCB	AUT	ΔBase	ΔIC	ΔDCB
0	0	63.33 ± 1.14	-1.15	+0.88	+5.15**	45.03 ± 0.68	+13.74***	+0.36	+4.19***	58.07 ± 0.53	+8.46***	+1.11	+10.52***	50.46 ± 0.87	+6.72**	+0.10	+13.05***
0	50	65.05 ± 0.98	+0.66	+0.48	+5.88**	45.84 ± 0.86	+18.69***	+0.31	+4.12***	59.44 ± 0.54	+11.48***	+0.49	+9.98***	52.22 ± 1.00	+9.64**	+0.20	+13.07***
0	100	67.13 ± 1.07	+3.59**	+0.34	+6.94**	46.87 ± 0.77	+26.65***	+0.20	+4.33***	59.91 ± 0.58	+11.88***	+0.50	+10.43***	52.55 ± 0.91	+10.41**	+0.37	+13.54***
0	200	70.48 ± 0.91	+6.42***	+0.14	+8.47**	48.89 ± 0.79	+29.77***	+0.63	+4.94***	61.92 ± 0.62	+14.53***	+0.57	+10.93***	53.41 ± 1.07	+13.66***	-0.12	+12.44***
0	400	75.02 ± 0.80	+9.98***	-0.43	+10.22***	52.50 ± 0.66	+32.92***	+1.60	+6.02***	63.26 ± 0.87	+18.29***	+0.29	+10.61***	55.39 ± 0.86	+21.66***	+1.02	+13.43***
50	0	74.05 ± 1.35	+6.01**	+5.56**	+15.13***	46.20 ± 0.79	+8.55***	+0.45	+4.30***	70.65 ± 0.92	+0.04	+5.31***	+12.61***	57.74 ± 2.42	+7.28**	+0.30	+11.30***
50	50	76.06 ± 1.59	+7.91***	+5.63**	+15.67***	46.37 ± 0.74	+11.69***	-0.33	+3.47***	73.00 ± 0.87	+3.96*	+5.10***	+13.29***	58.29 ± 2.51	+8.76**	+0.04	+10.05***
50	100	77.36 ± 1.66	+9.23***	+5.13**	+15.90***	46.86 ± 0.87	+16.00***	-0.19	+3.03***	73.04 ± 0.84	+2.26	+4.74***	+12.91***	59.89 ± 2.42	+10.31**	+0.38	+11.36***
50	200	79.16 ± 1.77	+10.78**	+4.12**	+15.86***	48.14 ± 0.75	+21.29***	+0.55	+2.86***	73.53 ± 1.92	+3.58*	+3.22*	+12.07***	60.20 ± 2.00	+12.63***	-0.27	+10.05***
50	400	81.89 ± 1.85	+13.13***	+2.84*	+15.85***	51.23 ± 1.13	+23.97***	+2.35**	+2.94***	74.82 ± 0.97	+4.43**	+3.10*	+12.04***	62.90 ± 2.08	+18.61***	+0.30	+11.46***
100	0	75.94 ± 0.93	+5.56***	+3.15	+15.10***	48.27 ± 0.67	+7.73***	+1.45*	+5.72***	71.06 ± 0.98	-0.74**	+4.50**	+10.71***	59.77 ± 1.60	+4.08**	+1.66	+10.88***
100	50	77.96 ± 0.99	+6.74***	+2.95	+15.76***	49.09 ± 0.71	+11.18***	+1.15*	+5.63***	73.32 ± 1.13	+2.40*	+4.81**	+11.26***	60.60 ± 1.88	+6.89**	+1.87	+10.08***
100	100	79.28 ± 1.15	+8.23***	+2.55	+15.94***	49.73 ± 0.77	+15.63***	+1.16*	+5.32***	73.51 ± 1.01	+1.45	+4.30*	+10.96***	61.92 ± 1.58	+8.01**	+2.02	+11.10***
100	200	81.10 ± 1.47	+10.53***	+2.01	+16.06***	51.04 ± 0.80	+23.68***	+1.74**	+4.96***	74.17 ± 1.52	+3.31*	+2.86	+10.69***	62.22 ± 1.45	+10.49***	+0.74	+9.75***
100	400	83.61 ± 1.68	+12.27***	+1.00	+15.78***	53.97 ± 1.13	+25.74***	+2.98**	+5.01***	74.90 ± 1.13	+3.80**	+1.98	+9.51***	65.03 ± 1.46	+16.82***	+1.76	+11.09***
200	0	76.51 ± 0.41	+2.42*	+3.22**	+13.89***	49.15 ± 0.95	+6.08***	+1.22	+6.03***	72.67 ± 0.38	-2.73***	+4.45***	+10.39***	60.14 ± 0.32	+0.06	+0.75	+8.91***
200	50	78.56 ± 0.25	+4.76***	+3.16**	+14.24***	50.12 ± 0.88	+8.43***	+1.05	+6.06***	74.98 ± 0.54	+0.02	+4.53***	+11.22***	60.78 ± 0.48	+2.12*	+0.96	+7.72***
200	100	79.74 ± 0.30	+6.15***	+2.88**	+14.26***	50.96 ± 0.76	+12.44***	+1.24	+5.94***	75.22 ± 0.51	-1.06	+4.24**	+10.84***	62.64 ± 0.23	+3.87***	+1.29*	+9.27***
200	200	81.43 ± 0.42	+8.07***	+2.57**	+14.04***	52.26 ± 0.78	+20.58***	+1.56*	+5.66***	76.38 ± 0.52	+1.85	+3.65**	+10.63***	62.58 ± 0.26	+6.18***	+0.11	+7.51***
200	400	83.86 ± 0.36	+10.21***	+2.11**	+13.69***	55.10 ± 0.81	+27.48***	+2.35**	+5.54***	76.65 ± 0.66	+1.08	+2.94**	+9.59***	65.50 ± 0.25	+14.52***	+1.00**	+8.80***
400	0	77.45 ± 0.38	+1.42	+3.26***	+12.77***	51.01 ± 0.54	+5.50***	+2.53**	+7.08***	72.50 ± 1.53	-8.33***	+3.41*	+8.87***	60.81 ± 0.76	-2.59**	+0.79	+7.75***
400	50	79.66 ± 0.32	+2.56*	+3.20***	+13.46***	52.27 ± 0.67	+8.61***	+2.53**	+7.34***	74.79 ± 1.58	-5.16**	+3.36*	+9.56***	62.12 ± 1.34	+0.16	+1.57	+7.10***
400	100	80.90 ± 0.38	+3.46**	+2.87**	+13.43***	53.19 ± 0.71	+12.55***	+2.63**	+7.29***	75.07 ± 1.68	-6.30*	+2.81	+9.17***	63.34 ± 0.95	+1.19	+1.06	+7.91***
400	200	82.53 ± 0.48	+5.21**	+2.59**	+13.13***	54.58 ± 0.72	+20.29***	+2.85**	+7.02***	76.04 ± 1.75	-4.04	+2.16	+8.62***	63.46 ± 0.74	+2.98**	+0.09	+6.23***
400	400	84.66 ± 0.52	+7.67**	+1.98*	+12.54***	56.94 ± 0.47	+29.07***	+3.11**	+6.15***	76.43 ± 1.72	-3.69	+1.65	+7.74***	66.39 ± 0.80	+10.68***	+1.15	+7.60***

our model architecture that uses Supervised Learning (SL) and standard cross-entropy loss, which we call SL-DRMD. In addition to SL-DRMD, for baselines, we also evaluate commonly used classifiers in Android malware detection: Drebin, DeepDrebin, and Ramda. The Drebin [Arp et al., 2014] classifier is a linear support vector machine (SVM) originally used for its corresponding feature space and serves as a foundational benchmark due to its simplicity. DeepDrebin [Grosse et al., 2017] applies a multilayer perceptron (MLP), with two hidden layer of 200 neurons, and represents a classical deep learning approach to Android malware detection without extensive modification. Finally, the Ramda [Li et al., 2021] classifier, developed for its corresponding feature space, uses a variational autoencoder (VAE) and a MLP to classify samples. Together, Drebin, DeepDrebin, Ramda, and SL-DRMD are evaluated across five random seeds on the four dataset-feature-space pairs and the difference between DRMD and the best performing baseline, for that setting, is reported as $\Delta Base$. Beyond ΔIC and $\Delta Base$, we also compare our agent architecture against two DCBs, NeuralTS Zhang et al. [2020a] and NeuralUCB Zhou et al. [2020]. ΔDCB reports the performance difference between DRMD and the best performing DCB using our state, action, and rewards. The hyperparameters for all of the baselines are provided in Appendix C.

6 Evaluation

This evaluation aims not only to investigate the base performance of DRMD but also to assess the impact that including rejection and AL (both monthly and integrated) has on model performance over time. We evaluate DRMD using five seeds, following the work of Alam *et al.* 2024, for each combination of dataset, feature space, and configuration, totaling 172 settings. Thus, we mitigate the impact of stochastic variance by capturing performance trends over multiple runs and across a wide range of conditions. Our results demonstrate that DRMD, both classification-only and classification-rejection policies, consistently outperform considered baselines, achieving an average AUT improvement of 8.66 and 10.90, respectively. Furthermore, DRMD significantly outperforms DCBs, NeuralTS and NeuralUCB, over all settings for the classification–only policy with an average $\Delta Base$ of 9.77.

6.1 Classification–Only Policy

First, we consider DRMD trained to learn a Classification–Only policy (π_{clf}), without the rejection action. Initially, when there is no AL or rejection (the first row of Table 1), DRMD surpasses the best baselines by 6.94 on average across five runs and all dataset-feature-space pairs. These initial results are promising, yet do not consider a complete malware detection pipeline. Thus, we consider malware detection pipeline configurations using monthly budgets for the uncertainty-based AL and rejection (Rej).

Monthly Rejection Here, we consider how integrating uncertainty-based monthly rejection affects performance when no AL is present. DRMD shows statistically significant improvements in performance over the best baseline in 15 of the 16 settings (rows 2-5 of Table 1), with an average $\Delta Base$ of 15.01. Increasing the number of uncertain samples rejected leads to increased performance for both AUT and $\Delta Base$. This means that the samples DRMD has high certainty on lead to more correct classifications than the baselines. Showing how the use of RL improves both overall accuracy and prediction certainty in malware detection.

Monthly Active Learning In the AL setting, both the DRMD policy and the baselines use monthly uncertainty-based AL to select samples for retraining at each testing month. DRMD demonstrates statistically significant improve-

Table 2: **Classification-Rejection Policy.** DRMD performance and average monthly rejection/selection rates across cost levels. $\Delta Base$ uses the seed-wise average rejections as the monthly rejection budget. All results are conducted as in Table 1.

	R_{cost}	Hypercube-Drebin			Hypercube-Ramda			Transcendent-Drebin			Transcendent-Ramda		
		AUT ± std	ΔBase	Rejected	AUT ± std	ΔBase	Rejected	AUT ± std	ΔBase	Rejected	AUT ± std	ΔBase	Rejected
IR	0	61.68 ± 1.71	-2.46	52.02	50.54 ± 2.82	+30.69***	533.80	60.70 ± 2.34	+14.08**	217.31	56.90 ± 0.74	+23.98***	599.47
	-0.1	63.17 ± 1.27	-0.90	58.17	50.99 ± 4.94	+31.14***	529.27	58.51 ± 4.34	+11.67*	203.50	55.84 ± 1.09	+22.91***	595.97
	-1	63.19 ± 2.51	-1.18	38.98	48.18 ± 3.91	+28.83***	353.58	58.96 ± 1.97	+11.39***	130.99	56.45 ± 1.36	+23.67***	541.41
	R_{cost}	AUT ± std	ΔBase	Rej&AL	AUT ± std	ΔBase	Rej&AL	AUT ± std	ΔBase	Rej&AL	AUT ± std	ΔBase	Rej&AL
$IRAL$	0	69.75 ± 3.13	+4.48**	28.95	64.35 ± 2.26	+33.84***	956.81	66.71 ± 3.20	-0.91	40.26	68.10 ± 1.73	+12.00***	506.85
	-0.1	67.51 ± 5.41	+3.82	21.99	65.39 ± 1.82	+36.65***	956.98	60.39 ± 8.24	-2.10	20.98	68.32 ± 6.43	+14.65*	869.61
	-1	64.61 ± 4.42	+1.00	24.09	57.41 ± 4.47	+29.98***	489.02	57.73 ± 5.70	-2.18	9.71	63.62 ± 0.96	+7.96***	211.16
	R_{cost}	AUT ± std	ΔBase	AL|Rej	AUT ± std	ΔBase	AL|Rej	AUT ± std	ΔBase	AL|Rej	AUT ± std	ΔBase	AL|Rej
$IRAAL$	0	72.37 ± 2.69	+5.00**	50|23.12	44.50 ± 2.25	+17.97***	50|360.80	70.93 ± 1.91	+2.41	50|50.07	61.09 ± 2.01	+13.51***	50|242.63
		72.05 ± 4.99	+3.34	100|15.34	50.86 ± 3.17	+22.98***	100|448.57	72.69 ± 0.29	+0.86	100|38.87	63.94 ± 0.90	+13.68***	100|279.51
		75.98 ± 1.95	+3.23***	200|13.37	56.23 ± 0.84	+27.36***	200|587.93	73.11 ± 1.22	-1.49*	200|57.27	64.95 ± 0.44	+13.53***	200|383.50
		78.13 ± 1.21	+1.95*	400|18.32	60.10 ± 1.49	+31.21***	400|724.25	74.66 ± 0.53	-6.53**	400|70.01	68.10 ± 3.31	+13.39**	400|466.65
	-0.1	69.68 ± 5.69	+4.73*	50|13.67	43.84 ± 2.45	+17.38***	50|344.86	69.74 ± 1.36	+1.07	50|34.48	61.15 ± 1.90	+12.51***	50|227.24
		75.29 ± 1.78	+4.03**	100|12.72	50.87 ± 1.88	+23.24***	100|404.13	72.49 ± 0.68	+0.99	100|39.14	63.23 ± 1.69	+12.96***	100|286.79
		76.80 ± 0.64	+3.66***	200|12.91	56.09 ± 2.14	+27.48***	200|553.63	73.76 ± 0.75	-0.55	200|60.88	65.10 ± 1.55	+12.76***	200|338.65
		77.48 ± 1.95	+1.95*	400|13.83	59.61 ± 0.64	+30.96***	400|684.16	74.57 ± 0.56	-6.63***	400|64.27	66.82 ± 1.49	+10.65**	400|401.47
	-1	66.84 ± 4.29	+3.74*	50|4.18	44.99 ± 2.34	+19.27***	50|192.70	65.04 ± 3.35	+1.56	50|6.58	60.17 ± 1.52	+10.65***	50|82.58
		69.18 ± 4.36	+3.74*	100|3.71	48.26 ± 1.44	+20.91***	100|211.69	69.03 ± 3.58	-1.32	100|12.30	63.56 ± 0.99	+9.70***	100|140.76
		70.00 ± 5.12	+2.35	200|3.73	53.45 ± 1.20	+26.06***	200|322.32	70.54 ± 1.43	-2.20	200|9.04	65.30 ± 0.90	+9.97***	200|235.29
		68.82 ± 6.23	-2.04	400|4.86	57.53 ± 1.21	+29.31***	400|503.93	72.56 ± 1.12	-4.57***	400|13.76	65.15 ± 1.17	+4.96**	400|239.47

ments in 9 of the 16 settings (where $AL > 0$ and $Rej = 0$ in Table 1), achieving an average $\Delta Base$ of 2.52. As in the rejection-only setting, both DRMD and the baselines use the same AL mechanism and budgets. Note that the diminishing returns of AL as performance improves McFadden et al. [2024a, 2023] coupled with the initial performance gains of DRMD contribute to the decreasing $\Delta Base$ as AL rates increase.

Monthly Rejection & Active Learning To represent a complete malware detection pipeline we combine both monthly AL and rejection in this evaluation, shown in Table 1 where $AL > 0$ and $Rej > 0$. DRMD has statistically significant improvements in 52 of the 64 settings, achieving up to 29.07 AUT improvement ($AL = 400$ and $Rej = 400$), and an average $\Delta Base$ of 8.71. Thus, when the rejection of samples at high risk of misclassification and the cost-sensitive drift adaptation of AL are applied together on DRMD, this performs better than when only using either individually, highlighting how our approach can yield improvements in real world settings.

MD-MDP vs ICMDP To isolate the difference from MD-MDP compared to ICMDP, we train a version of the DRMD agent using the transitions and reward structure of ICMDP Lin et al. [2020]. Concretely, episodes continue until either a) malware is misclassified or b) the agent runs out of training samples. Otherwise, states and actions are the same as in the MD-MDP. As a result, the next states are based on the next sample from the dataset. Rewards are structured as in R_{acc}, however, rewards from goodware samples are scaled down by $\hat{\sigma}$. For fairness, we use the same PPO architecture and hyperparameters to train both our agent and the ICMDP agent. The ΔIC columns in Table 1 show the gains or losses of the MD-MDP DRMD agent over the ICMDP DRMD agent across all combinations of dataset-feature-space pairs, AL rates, and rejection rates. Overall, the MD-MDP outperforms ICMDP in 97 out of 100 settings, with 45 settings being statistically significant, and achieves an average ΔIC of 1.94. These results demonstrate that beyond theoretical modeling discussions, the MD-MDP is empirically better suited for integration into wider malware detection pipelines.

One-Step PPO vs DCBs To isolate the benefit of our architecture and algorithm choice, we compare two versions of DRMD. The first is the standard version of DRMD and the second uses one of two DCBs, either NeuralTS or NeuralUCB, depending on which performs better for a given setting. The DCBs use the same architecture and algorithms as proposed in their original works. The ΔDCB columns in Table 1 shows the gains of one-step PPO over the DCBs across all combinations of dataset-feature-space pairs, AL rates, and rejection rates. Overall, the one-step PPO shows statistically significant improvements over the DCBs in all 100 settings. These results demonstrate that the architecture and one-step PPO algorithm chosen for DRMD is better suited, than standard DCBs, to the dynamics of the Android malware detection considered in this paper.

6.2 Classification-Rejection Policy

Extending from the Classification-only policy, the DRMD Classification-Rejection policy (π_Σ) integrates the *rejection action*, a_{rej}. Importantly, this action enables the agent to abstain from making a decision on a sample if there is insufficient information, high uncertainty, or concept drift. Note how, previously, rejections occurred only at the end of each month; however, the rejection action provides *real-time* abstention as the samples are seen, thus opening up the potential for real-time detection systems in future work. To maintain the capability for such decision making while facilitating AL, the samples resulting from the rejection action are used for retraining. We consider three different rejection costs (R_{cost}) to show how they effect the performance of DRL agents that can reject malware samples, which has not been considered in prior work. We present the results in Table 2, showing $\Delta Base$ as the comparison of DRMD to the best baseline. To ensure consistency in the number of rejections, the seed-wise average rejections

Table 3: **Ablation Study.** DRMD performance and rejection/selection rates starting with the simplest base approach and adding components of DRMD back sequentially. AUTs reported show the mean ± std performance of DRMD across five runs after adding the component, specified at the start of the row, to the version of DRMD from the previous row. '—' denotes rejection is not enabled and italics denote that the samples are also used for AL.

Added	Hypercube-Drebin		Hypercube-Ramda		Transcendent-Drebin		Transcendent-Ramda	
	AUT ± std	Rejected	AUT ± std	Rejected	AUT ± std	Rejected	AUT ± std	Rejected
Basic DRMD	59.22 ± 1.58	–	22.34 ± 1.52	–	46.53 ± 1.48	–	37.75 ± 2.21	–
Temporal Reward Scaling	56.15 ± 0.64	–	25.74 ± 1.55	–	46.18 ± 2.07	–	37.39 ± 2.44	–
Malware Reward Scaling	64.07 ± 1.77	–	43.95 ± 0.88	–	56.80 ± 0.65	–	49.80 ± 1.90	–
4 Hidden Network Layers	63.56 ± 1.69	–	43.44 ± 1.43	–	57.69 ± 0.83	–	49.90 ± 0.56	–
512 Neuron Hidden Layers	63.33 ± 1.14	–	45.03 ± 0.68	–	58.07 ± 0.53	–	50.46 ± 0.87	–
Integrated Reject Action	61.42 ± 0.90	0.00	44.90 ± 0.79	0.00	57.88 ± 2.22	0.00	50.28 ± 1.22	0.00
Reward Reject Outcome	61.68 ± 1.71	52.02	50.54 ± 2.82	533.80	60.70 ± 2.34	217.31	56.90 ± 0.74	599.47
Reject Cost (R_{cost}=-0.1)	63.17 ± 1.27	58.17	50.99 ± 4.94	529.27	58.51 ± 4.34	203.50	55.84 ± 1.09	595.97
Integrated Rejection with AL	64.24 ± 1.78	*9.70*	52.49 ± 1.07	*391.02*	59.47 ± 0.79	*17.11*	62.90 ± 1.07	*170.39*
Sliding Retraining Window	67.51 ± 5.41	*21.99*	65.39 ± 1.82	*956.98*	60.39 ± 8.24	*20.98*	68.32 ± 6.43	*869.61*
Augmented Active Learning	77.48 ± 1.95	*400*|13.83	59.61 ± 0.64	*400*|684.16	74.57 ± 0.56	*400*|64.27	66.82 ± 1.49	*400*|401.47

(rounded up) of DRMD is used as the budget for baseline rejections.

Integrated Rejection (IR) The IR setting introduces the rejection action without any additional AL on rejected samples. The results can be seen in Table 2, under '*IR*', and show statistically significant improvements over all baselines in 9 of the 12 settings with an average $\Delta Base$ of 16.15. This shows that the agent can effectively learn a policy that rejects samples likely to be misclassified. When comparing the two feature spaces (Ramda and Drebin), we see a disparity in the average number of rejections. Specifically, policies trained on Ramda reject more samples than those trained on Drebin. We hypothesize that the reduction of information encapsulated by Ramda results in greater prediction uncertainty compared to the Drebin feature space. Increasing R_{cost} intuitively reduces the number of rejected samples with only a minimal effect on $\Delta Base$. This implies that higher R_{cost} values lead DRMD to be more selective in rejection.

IR with Active Learning (IRAL) To go beyond the mitigation of concept drift and allow self-adaptation, we include the samples rejected in each period for retraining DRMD by updating the fine-tuning sample sliding window. The IRAL section in Table 2 shows the results of the combined rejection and AL policy across each combination of the dataset-feature-space-pair per rejection cost. Compared with the best baseline for each setting, DRMD shows statistically significant improvements in 7 of the 12 settings with an average $\Delta Base$ of 11.60. In Hypercube-Drebin and Transcendent-Drebin, retraining on rejected samples leads to higher confidence in classification actions. As R_{cost} increases, this leads to a reduction in the number of rejected and, thus, retraining samples in subsequent periods causing the observed drop in performance.

IR with Augmented Active Learning (IRAAL) To help mitigate the impact of conservative rejection policies and ensure stable retraining budgets for AL, the IRAAL section in Table 2 presents integrated rejection with augmented AL. Augmented AL extends the rejection-based sampling used in IRAL with uncertainty sampling so that, given a predetermined budget for AL, the model samples up to the predetermined labeling budget for retraining if the agent was conservative in rejections, and downsamples to the budget if the agent is rejecting more samples. We compare using the four different AL budgets as in Section 6.1. The results show statistically significant performance gains over the baseline approaches in 33 of 48 settings, with an average $\Delta Base$ of 10.75. Highlighting that the augmented AL budget improves stability and performance compared to the IRAL results.

Ablation Study In Table 3 we present a cumulative component breakdown of the AUT performance of DRMD across the four dataset–feature-space pairs. Starting from the *basic* DRMD (single 128-neuron layer, ±1 reward, no rejection), performance is uniformly low. Introducing *temporal and malware reward scaling* immediately increases AUT across all settings, highlighting the importance of incorporating both the spatial (malware distribution) and temporal aspects of the domain into the reward function. Increasing the capacity of the model, both by the number of *hidden layers* and by the number of their *neurons*, provides marginal improvements. Adding the *integrated reject action* without reward R_{rej} leads to the model not rejecting any samples and to negligible performance differences. Crucially, *rewarding rejected outcomes* provides a large performance increase as it allows the agent to learn to balance abstention and misclassification risk, with the *rejection cost* further refining this trade-off. Subsequently, *integrated rejection with AL* enables the agent to adapt to drift over time and coupled with the *sliding retraining window* provides performance increases in all but the Transcendent-Drebin setting where DRMD becomes too conservative in rejection. By rectifying this, *augmented active learning* yields a increase in AUT performance across most settings.

7 Conclusion

We introduce a new one-step MDP formulation for malware detection, MD-MDP, which enables the unification of DRL, cost-aware rejection, and AL to counteract concept drift in temporally split evaluations. We use MD-MDP to train the DRMD agent. Our results demonstrate that across two Android malware datasets and two feature spaces, DRMD outperforms the baselines of Drebin, DeepDrebin, Ramda, and SL-DRMD (SL version of DRMD) by an average of 8.66 AUT using the classification-only policy. Furthermore, we show that DRMD learns effective strategies that enable real-time detection, rejection, and AL leading to an average 10.90 AUT increase over the best baselines for all settings using the classification-rejection policy.

Future work could extend the MD-MDP and DRMD to include additional aspects of the malware detection pipeline, such as feature retrieval, as done in the image domain by Janisch *et al.* 2019. Additionally, extending the rejection reward design to integrate sources of information, such as conformal evaluation Jordaney et al. [2017], Barbero et al. [2022], explanation Yang et al. [2021], or contrastive learning Chen et al. [2023], provides a potentially valuable next step. Together these directions for future work represent the path forward to a DRL policy that unifies the complete Android malware detection problem.

Acknowledgments

This research was partially funded and supported by: the UK National Cyber Security Centre (NCSC); UK EPSRC Grant no. EP/X015971/2; the Defence Science and Technology Laboratory (DSTL), an executive agency of the UK Ministry of Defence, supporting the Autonomous Resilient Cyber Defence (ARCD) project within the DSTL Cyber Defence Enhancement programme.

References

Md Tanvirul Alam, Dipkamal Bhusal, and Nidhi Rastogi. Revisiting static feature-based android malware detection. *arXiv preprint arXiv:2409.07397*, 2024.

Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Are your training datasets yet relevant? In *International Symposium on Engineering Secure Software and Systems*, pages 51–67. Springer, 2015.

Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and Explainable Detection of Android Malware in your Pocket. In *Ndss*, volume 14, pages 23–26, 2014.

Yikun Ban, Ishika Agarwal, Ziwei Wu, Yada Zhu, Kommy Weldemariam, Hanghang Tong, and Jingrui He. Neural Active Learning Beyond Bandits. In *Proceedings of the International Conference of Learning Representations (ICLR)*. arXiv, April 2024. doi: 10.48550/arXiv.2404.12522. URL http://arxiv.org/abs/2404.12522. arXiv:2404.12522 [cs].

Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. Transcending Transcend: Revisiting Malware Classification in the Presence of Concept Drift. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 805–823. IEEE, 2022.

Elizabeth Bates, Vasilios Mavroudis, and Chris Hicks. Reward shaping for happier autonomous cyber security agents. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pages 221–232, 2023.

Liu Binxiang, Zhao Gang, and Sun Ruoying. A deep reinforcement learning malware detection method based on pe feature distribution. In *2019 6th International Conference on Information Science and Control Engineering (ICISCE)*, pages 23–27. IEEE, 2019.

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

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

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

Antonio Coscia, Andrea Iannacone, Antonio Maci, and Alessandro Stamerra. Sinner: A reward-sensitive algorithm for imbalanced malware classification using neural networks with experience replay. *Information*, 15(8):425, 2024.

Myles Foley and Sergio Maffeis. Apirl: Deep reinforcement learning for rest api fuzzing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 191–199, 2025.

Myles Foley, Chris Hicks, Kate Highnam, and Vasilios Mavroudis. Autonomous Network Defence Using Reinforcement Learning. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, pages 1252–1254. Association for Computing Machinery, 2022a. doi: 10.1145/3488932.3527286. URL https://doi.org/10.1145/3488932.3527286.

Myles Foley, Mia Wang, Zoe M, Chris Hicks, and Vasilios Mavroudis. Inroads into Autonomous Network Defence using Explained Reinforcement Learning. In *Conference on Applied Machine Learning in Information Security (CAMLIS)*, 2022b.

Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial examples for malware detection. In *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*, pages 62–79. Springer, 2017.

Alexander Herzog, Aliai Eusebi, and Lorenzo Cavallaro. Aurora: Are android malware classifiers reliable under distribution shift? *arXiv preprint arXiv:2505.22843*, 2025.

Chris Hicks, Vasilios Mavroudis, Myles Foley, Thomas Davies, Kate Highnam, and Tim Watson. Canaries and Whistles: Resilient Drone Communication Networks with (or without) Deep Reinforcement Learning. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence* *and Security*, AISec '23, pages 91–101. Association for Computing Machinery, 2023. ISBN 9798400702600. doi: 10.1145/3605764.3623986.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

Jaromír Janisch, Tomáš Pevnỳ, and Viliam Lisỳ. Classification with costly features using deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3959–3966, 2019.

SL Jayaprakash, Kolla Gnapika Sindhu, Tekumudi Vivek Sai Surya Chaitanya, Bachu Ganesh, et al. Meddqn: A deep reinforcement learning approach for biomedical image classification. In *2023 Global Conference on Information Technologies and Communications (GCITC)*, pages 1–7. IEEE, 2023.

Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting Concept Drift in Malware Classification Models. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 625–642, 2017.

Zeliang Kan, Shae McFadden, Daniel Arp, Feargus Pendlebury, Roberto Jordaney, Johannes Kinder, Fabio Pierazzi, and Lorenzo Cavallaro. Tesseract: Eliminating experimental bias in malware classification across space and time (extended version). *arXiv preprint arXiv:2402.01359*, 2024.

Heng Li, Shiyao Zhou, Wei Yuan, Xiapu Luo, Cuiying Gao, and Shuiyan Chen. Robust android malware detection against adversarial example attacks. In *Proceedings of the Web Conference 2021*, pages 3603–3612, 2021.

Enlu Lin, Qiong Chen, and Xiaoming Qi. Deep reinforcement learning for imbalanced classification. *Applied Intelligence*, 50(8):2488–2502, 2020.

Shae McFadden, Zeliang Kan, Lorenzo Cavallaro, and Fabio Pierazzi. Poster: Rpal-recovering malware classifiers from data poisoning using active learning. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 3561–3563, 2023.

Shae McFadden, Mark Kan, Lorenzo Cavallaro, and Fabio Pierazzi. The impact of active learning on availability data poisoning for android malware classifiers. In *Proceedings of the Annual Computer Security Applications Conference Workshops (ACSAC Workshops)*. IEEE, 2024a.

Shae McFadden, Marcello Maugeri, Chris Hicks, Vasilios Mavroudis, and Fabio Pierazzi. Wendigo: Deep reinforcement learning for denial-of-service query discovery in graphql. In *IEEE Workshop on Deep Learning Security and Privacy (DLSP)*, 2024b.

Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, et al. Reviewer integration and performance measurement for malware detection. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 122–141. Springer, 2016.

Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern recognition*, 45(1): 521–530, 2012.

Thanh Thi Nguyen and Vijay Janapa Reddi. Deep reinforcement learning for cyber security. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models (Extended Version). *ACM Transactions on Privacy and Security (TOPS)*, 22(2):1–34, 2019.

Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 729–746, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Burr Settles. Active learning literature survey. 2009.

Ki Hyun Tae, Hantian Zhang, Jaeyoung Park, Kexin Rong, and Steven Euijong Whang. Falcon: Fair Active Learning Using Multi-Armed Bandits. *Proceedings of the VLDB Endowment*, 17(5):952–965, January 2024. ISSN 2150-8097. doi: 10.14778/3641204.3641207. URL https://dl.acm.org/doi/10.14778/3641204.3641207. Publisher: Association for Computing Machinery (ACM).

Ilias Tsingenopoulos, Davy Preuveneers, Lieven Desmet, and Wouter Joosen. Captcha me if you can: Imitation games with reinforcement learning. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 719–735. IEEE, 2022.

Ilias Tsingenopoulos, Jacopo Cortellazzi, Branislav Bošanskỳ, Simone Aonzo, Davy Preuveneers, Wouter Joosen, Fabio Pierazzi, and Lorenzo Cavallaro. How to train your antivirus: Rl-based hardening through the problem space. In *Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 130–146, 2024.

A Usha Nandhini and K Dharmarajan. Drl-cnn technique for diabetes prediction. In *International Conference on Advancements in Smart Computing and Information Security*, pages 55–68. Springer, 2023.

Jenny Yang, Rasheed El-Bouri, Odhran O'Donoghue, Alexander S Lachapelle, Andrew AS Soltan, David W Eyre, Lei Lu, and David A Clifton. Deep reinforcement learning for multi-class imbalanced training: applications in healthcare. *Machine Learning*, 113(5):2655–2674, 2024.

Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. CADE: Detecting and Explaining Concept Drift Samples for Security Applications. In *USENIX security symposium*, pages 2327–2344, 2021.

Weitong Zhang, Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural thompson sampling. *arXiv preprint arXiv:2010.00827*, 2020a.

Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 757–770, 2020b.

Dongruo Zhou, Lihong Li, and Quanquan Gu. Neural contextual bandits with ucb-based exploration. In *International conference on machine learning*, pages 11492–11502. PMLR, 2020.

S Kevin Zhou, Hoang Ngan Le, Khoa Luu, Hien V Nguyen, and Nicholas Ayache. Deep reinforcement learning in medical imaging: A literature review. *Medical image analysis*, 73:102193, 2021.

Ting Zuo, Fenglian Li, Xueying Zhang, Fengyun Hu, Lixia Huang, and Wenhui Jia. Stroke classification based on deep reinforcement learning over stroke screening imbalanced data. *Computers and Electrical Engineering*, 114:109069, 2024.

A Computing Infrastructure

All experiments were conducted on a server using x86_64 architecture, AMD EPYC processors (16 CPU cores total), a NVIDIA H100 NVL GPU (CUDA 12.6) with 96 GB of memory, and 128 GB of RAM. The evaluation used Python 3.9.21 with core libraries including PyTorch 2.4.1+cu121, TensorFlow 2.18.0, scikit-learn 1.6.1, and NumPy 2.0.2. Further details regarding all library requirements can be found in the repository provided.

B Seeding

In order to improve the reproducibility of the experiments presented in the paper, the random seeds presented in Appendix C are used to seed $random$, $numpy.random$, and $torch.manual\_seed$.

C Hyperparameters

This section presents the complete hyperparameters and architecture for all approaches used in this paper.

C.1 Drebin

The hyperparameters used for all Drebin experiments was drawn directly from the updated time-aware evaluation work by Kan *et al.* 2024. The **hyperparameters** used are as follows, *Max Iterations*: 50000, *C*: 1, *Seeds*: [0, 1, 26, 42, 0x10c0ffee], *Classifier*: LinearSVC (sklearn.svm).

C.2 DeepDrebin

The hyperparameters used for all DeepDrebin experiments were drawn directly from the updated time-aware evaluation work by Kan *et al.* 2024. The **hyperparameters** used are as follows:

Training, *Epochs*: 10, *Batch Size*: 64, *Training/Validation Split*: 0.66/0.34, *Seed*: [0, 1, 26, 42, 0x10c0ffee], *Learning Rate*: 0.05, *Loss*: Cross Entropy, *Optimizer*: SGD.

Architecture, *Input*: n_features, *Layer Size*: 200, *Hidden Layers*: 2, *Activation Function*: ReLU, *Dropout*: 0.5, *Output*: 2.

C.3 Ramda

The hyperparameters used for Ramda follow the original work by Heng *et al.* 2021, extended to match our setting and hyperparameters used with the rest of the baselines. The **hyperparameters** used are as follows:

Training, *Epochs*: 10, *Batch Size*: 512, *Training/Validation Split*: 0.80/0.20, *Seeds*: [0, 1, 26, 42, 0x10c0ffee], *Learning Rate*: 1e-3, *Optimizer Epsilon*: 1e-8, *Optimizer*: Adam, *Reconstruction Weight*: 10.0, *KL Divergence Weight*: 1.0, *Feature Disentanglement Weight*: 10.0.

VAE, *Input*: n_features, *Hidden Layer Size*: 600, *Dropout*: 0.1, *Encoder Hidden Layers*: 2, *Bottleneck Layer Size*: 80, *Decoder Hidden Layers*: 2, *Activation Function*: ELU (first layer), Tanh (second layer) for encoder and reversed for decoder, *Input*: n_features, *V Threshold*: 30.

MLP, *Input*: 2 × Bottleneck Layer Size, *Layer Size*: 40, *Hidden Layers*: 3, *Activation Function*: Tanh (first layer) and ELU (second & third layers), *Dropout*: 0.1, *Output*: 2.

C.4 DRMD

The final hyperparameters used for DRMD are the same used in development, with the hyperparameters following from Schulman *et al.* 2017 and Huang *et al.* 2022. The **hyperparameters** used are as follows:

Training, *Epochs (Training Data)*: 5, *Epochs (Minibatches)*: 1, *Sliding Window Size*: 5000, *Minibatch Size*: 100, *Clip Coefficient*: 0.2, *Value Coefficient*: 0.5, *Entropy Coefficient*: 0.01, *Max Grad Norm*: 0.5, *Seeds*: [0, 1, 26, 42, 0x10c0ffee], *Learning Rate*: 2.5e-4, *Optimizer Epsilon*: 1e-5, *Optimizer*: Adam.

Actor, *Input*: n_features, *Layer Size*: 512, *Hidden Layers*: 4, *Activation Function*: LeakyReLU, *Dropout*: 0.5, *Output*: 3.

Critic, *Input*: n_features, *Layer Size*: 512, *Hidden Layers*: 4, *Activation Function*: LeakyReLU, *Dropout*: 0.5, *Output*: 1.

C.5 SL-DRMD

The hyperparameters of SL-DRMD are a combination of the DeepDrebin and DRMD hyperparameters. The **hyperparameters** used are as follows:

Training, *Epochs*: 5, *Batch Size*: 256, *Training/Validation Split*: 0.66/0.34, *Seeds*: [0, 1, 26, 42, 0x10c0ffee], *Learning Rate*: 0.05, *Loss*: Cross Entropy, *Optimizer*: SGD.

Architecture, *Input*: n_features, *Layer Size*: 512, *Hidden Layers*: 4, *Activation Function*: ReLU, *Dropout*: 0.5, *Output*: 2.

C.6 ICMDP

We use the same hyperparameters for the ICMDP version architecture as our proposed DRMD to isolate the modeling difference between the two versions, showing the impact of each version of the reward and state transition functions. We report the Gamma and GAE Lambda hyperparameters as they are relevant for the ICMDP version because of its multi-step episodes. The **hyperparameters** used are as follows:

Training, *Epochs (Training Data)*: 5, *Epochs (Minibatches)*: 1, *Sliding Window Size*: 5000, *Minibatch Size*: 100, *Clip Coefficient*: 0.2, *Value Coefficient*: 0.5, *Entropy Coefficient*: 0.01, *Max Grad Norm*: 0.5, *Gamma*: 0.99, *GAE Lambda*: 0.95, *Seeds*: [0, 1, 26, 42, 0x10c0ffee], *Learning Rate*: 2.5e-4, *Optimizer Epsilon*: 1e-5, *Optimizer*: Adam.

Actor, *Input*: n_features, *Layer Size*: 512, *Hidden Layers*: 4, *Activation Function*: LeakyReLU, *Dropout*: 0.5, *Output*: 2.

Critic, *Input*: n_features, *Layer Size*: 512, *Hidden Layers*: 4, *Activation Function*: LeakyReLU, *Dropout*: 0.5, *Output*: 1.

C.7 NeuralUCB & NeuralTS

The hyperparameters used forNeuralTS Zhang et al. [2020a] and NeuralUCB Zhou et al. [2020] follow their original works, extended to match our setting. The **hyperparameters** used are as follows:

Training, *Gradient Steps per Update*: 100, *Minibatch Size*: $|\mathcal{D}|$ (full batch over stored contexts), *Loss*: MSE, *Seeds*: [0, 1, 26, 42, 0x10c0ffee], *Learning Rate*: 1e-2, *Weight Decay*: $^{\lambda}/_{|\mathcal{D}|}$, *Optimizer*: SGD.

Arm Architecture, *Input*: $2 \times$ n_features (block-encoded context), *Layer Size*: 100, *Hidden Layers*: 1, *Activation Function*: ReLU, *Output*: 1.

Bandit Parameters, *Arms*: 2, *Style*: TS (NeuralTS) or UCB (NeuralUCB), (λ): 1, (ν): 1, *Gradient Estimation*: BackPACK.