

Minimizing Compute Costs: When Should We Run More Expensive Malware Analysis?

Andre T. Nguyen^{1,2,3,4}, Richard Zak^{2,3,4}, Luke E. Richards^{2,3,4}, Maya Fuchs^{2,3}, Fred Lu^{2,3,4}, Robert Brandon^{2,3,5}, Gary Lopez Munoz^{2,3,5}, Edward Raff^{2,3,4}, Charles Nicholas⁴ and James Holt²

¹JURA Bio, Inc. (work done while at the Laboratory for Physical Sciences and Booz Allen Hamilton)

²Laboratory for Physical Sciences

³Booz Allen Hamilton

⁴University of Maryland, Baltimore County

⁵Microsoft (work done while at the Laboratory for Physical Sciences and Booz Allen Hamilton)

Abstract

As organizations in government and industry increasingly rely on digitized data and networked computer systems, they face a growing risk of exposure to cyber attacks. Automated methods such as machine learning based malware detection algorithms have helped analysts to sift through large amounts of data. However, it is still too expensive to always run the best algorithms when massive amounts of new data are generated every day.

In this work, we demonstrate the benefits of leveraging uncertainty estimation when multiple algorithms with different strengths and costs are used as a part of a larger machine learning malware detection system. In particular, we introduce a novel method in which cheaper machine learning algorithms can choose to defer to costlier models when their own predictions are uncertain and the more expensive model is expected to do well.

We first use this method to detect specific capabilities in executable files, then extend it to general malware detection. In both cases, we are able to maintain high accuracy while minimizing the use of the more costly algorithms. With capability detection, we achieve an average 99.9% of correctly labeled capabilities for half the computational cost of using the expensive model throughout. For general malware detection, using this method to strategically balance the use of static and dynamic analysis saves a year's worth of compute time.

Keywords

machine learning, malware detection, cyber security, uncertainty

1. Introduction

As evidenced by current events, malware is a serious threat to companies, governments, schools, and even health care systems. It has cost billions in damages [1, 2] and even taken lives [3]. As computer networks grow in size, so do the challenges cybersecurity professionals face in securing them. With more connected devices, more users, and more complex systems, the volume of malware attack opportunities increases exponentially. It's clear, then, that automated malware detection is absolutely crucial, but the sheer volume of incoming data can make it very computationally challenging. Many anti-virus (AV) vendors see over 2 million new malicious files each month [4], and benign files on a network tend to outnumber malicious files at a ratio of 80:1 [5].

The collection and release of malware datasets has allowed for a vast array of machine learning (ML) based malware detection (MLMD) algorithms to be developed with the goal of classifying new files as benign (not malware) or malicious (malware) [6]. Traditionally, anti-virus systems used static and signature-driven systems (i.e., systems that looked for specific software known to be malicious) to detect malware [7]. More recently, dynamic software analysis has become increasingly popular. This involves running evaluated software in a secure environment to directly observe whether or not it behaves maliciously. Egele et al. [8] provide a survey on challenges, features, techniques, and tools for dynamic malware analysis.

CAMLIS'22: Conference on Applied Machine Learning in Information Security (CAMLIS), October 20–21, 2022, Arlington, VA

✉ an@jurabio.com (A. T. Nguyen); edraff@lps.umd.edu (E. Raff); holt@lps.umd.edu (J. Holt)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).



CEUR Workshop Proceedings (CEUR-WS.org)

Using machine learning to automate static and dynamic analysis enables faster investigation of more files and allows human analysts to focus on “hard” samples. While powerful, anti-virus and dynamic analysis tools have limitations. In particular, these methods can be time consuming when the data volume and velocity are high [9], with certain methods such as dynamic analysis having particularly high computational cost. Unsurprisingly, these algorithms also vary widely in their strengths, weaknesses, and costs, both in terms of model execution and especially feature extraction.

Recent works have demonstrated the benefits of incorporating uncertainty quantification in MLMD systems. Leveraging uncertainty allows for better identification of model errors, uncovering of new malware families, predictive performance under extreme false positive constraints, and out of distribution data detection [10, 11, 12]. These previous works have focused exclusively on static analysis based malware detection using a single ML algorithm.

In the following sections, we introduce a novel method in which cheaper ML algorithms can choose to defer to costlier models under two conditions: their own predictions are uncertain, and the more expensive model is likely to do well. We note that the main contribution of this paper is this deferral methodology; the determination of computational expense can vary from use case to use case.

Despite this variation, we find that by using uncertainty quantification combined with auxiliary prediction targets, we can consistently minimize compute costs while maximizing overall system accuracy. Specifically, we estimate uncertainty using Bayesian and ensembling approaches, and we train the cheaper algorithms to predict the success of the more expensive algorithms. The result is an easy to implement approach that is applicable to a wide variety of AV systems.

We first use this approach to speed up capability detection in executable files, using CAPA as the expensive model and MalConv as the cheaper one. We then show that this approach can improve general malware detection, where the cheaper model uses static analysis and the expensive model uses dynamic analysis. By using the expensive models only as-needed, we are able to improve accuracy while minimizing the use of more costly algorithms.

2. Related Work

The majority of the existing research on malware detection with machine learning has focused on the automation of static malware analysis. Raff et al. [13] introduce MalConv, a convolutional neural network for malware detection that operates on the raw byte sequences of files. Raff et al. [14] develop a new approach to temporal max pooling that improves MalConv’s memory and computational costs.

Understanding when a machine learning model is uncertain about its prediction is critical in high-risk applications such as cyber defense. However, the modeling of uncertainty is generally missing from the machine learning for malware literature. Three recent works have demonstrated the benefits of incorporating uncertainty quantification in MLMD systems. In particular, leveraging uncertainty allows for the identification of model errors, the uncovering of new malware families, improved predictive performance under extreme false positive constraints, and better out-of-distribution data detection [10, 11, 12].

We note that these previous works have focused exclusively on static analysis based malware detection using a single ML algorithm. In the current work, we explore the benefits of uncertainty estimation in settings where multiple, diverse algorithms with different strengths and costs are used as part of a larger MLMD system.

2.1. Uncertainty Estimation

The Bayesian framework allows for the principled modeling of uncertainty in machine learning and decision making [15]. Within this framework, probabilities represent degrees of belief, as opposed to the frequentist interpretation of probabilities as long-run frequencies [16]. Bayesian inference uses Bayes’ Theorem to update beliefs (represented in the form of probability distributions) when new data is observed.

In the context of machine learning, a Bayesian update takes the following form, where θ represents model parameters, D represents the data, and M represents the model class:

$$P(\theta|D, M) = \frac{P(D|\theta, M)P(\theta|M)}{P(D|M)}$$

$P(\theta|D, M)$ is the posterior belief about the model parameters given the data, $P(D|\theta, M)$ is the likelihood of the data given the model parameters, $P(\theta|M)$ is the prior belief about model parameters, and $P(D|M)$ is the marginal likelihood or evidence. These are related by Bayes' rule.

For prediction, the posterior predictive can be computed as follows, where D is the training data and D^* is the test data: $P(D^*|D) = \int P(D^*|D, \theta)P(\theta|D)d\theta$. When data points are conditionally independent given model parameters, then $P(D^*|D, \theta) = P(D^*|\theta)$, and we can write:

$$P(D^*|D) = \int P(D^*|\theta)P(\theta|D)d\theta$$

The posterior predictive is an example of Bayesian model averaging [17], a posterior weighted average of $P(D^*|\theta)$.

As exact Bayesian inference cannot be done for Bayesian deep learning models, approximate inference methods need to be used.

A straightforward approach is to use a Laplace approximation to model the posterior over neural network weights as a Gaussian [18, 19]. Ritter et al. [20] construct a Kronecker factored Laplace approximation to the posterior. Kristiadi et al. [21] add uncertainty units to a pre-trained network. The units are trained post-hoc using an uncertainty-aware loss. This addition improves uncertainty under a Laplace approximation.

Proper Markov Chain Monte Carlo (MCMC) methods will always perfectly sample from the posterior given enough time. The MCMC method of choice is Hamiltonian Monte Carlo (HMC) as described in Neal [22] under the name "hybrid Monte Carlo." Betancourt [23] provides an in-depth introduction to HMC. While HMC is notoriously difficult to tune, methods such as the No-U-Turn Sampler of Hoffman and Gelman [24] attempt to automate the tuning. While considered the gold standard, HMC isn't scalable because the method requires gradient computations that use the entire dataset.

As MCMC is hard to scale and tune in practice, variational inference is often used instead [25, 26]. This converts the integration problem into an optimization problem, where the posterior is approximated using a simpler variational distribution. In particular, the exact posterior $p(\cdot|D)$ is approximated by a variational approximation $q(\cdot)$ by minimizing the Kullback-Leibler divergence:

$$\begin{aligned} q^* &= \arg \min_{q \in Q} \text{KL}(q(\cdot)||p(\cdot|D)) \\ q^* &= \arg \min_{q \in Q} \int q(\theta) \log \frac{q(\theta)}{p(\theta|D)} d\theta \\ q^* &= \arg \min_{q \in Q} \int q(\theta) \log \frac{q(\theta)p(D)}{p(\theta, D)} d\theta \\ q^* &= \arg \min_{q \in Q} \log p(D) - \int q(\theta) \log \frac{p(\theta, D)}{q(\theta)} d\theta \end{aligned}$$

The $\log p(D)$ term does not depend on q , so it can be dropped from the optimization, and the subtracted term that remains is called the evidence lower bound (ELBO).¹ So the optimization becomes

$$q^* = \arg \max_{q \in Q} \text{ELBO}(q)$$

¹ELBO is a lower bound since the KL divergence is always positive, and as a result $\log p(D) \geq \text{ELBO}$.

The scaling of Bayesian methods to sizeable datasets has largely been made possible by stochastic variational inference [27]. Black box automatic stochastic variational inference methods [28, 29, 30, 31] that only require the specification of the model log-likelihood have improved the usability and adoption of Bayesian methods, and have sped up model iteration. Variational inference for neural networks was first introduced in the early 1990’s [32]. Graves [33] revived interest in the topic by introducing a *stochastic* variational method for inference in neural networks, which was later improved by Blundell et al. [34].

Gal and Ghahramani [35], Gal [36] introduce an easy to implement approach to variational inference in Bayesian neural networks. In particular, they show that a neural network with dropout [37, 38] is equivalent to an approximation of a deep Gaussian process [39], and training with dropout effectively performs variational inference for the deep Gaussian process model. For reference, dropout is a technique commonly used to reduce overfitting in neural networks by randomly dropping units during training, applied before every weight layer. Leaving dropout on at test time allows for sampling from the posterior distribution. This is the approach used for the Bayesian MalConv model [12] that we will use as the cheaper, static model in our experiments.

Two different kinds of uncertainty can be measured [36]. The first, aleatoric uncertainty, is a result of inherent stochasticity in the input data. More training data will not reduce the aleatoric uncertainty associated with a prediction, as this uncertainty is caused by noise in the prediction time input data. The second, epistemic uncertainty, is caused by a lack of training data resembling the prediction time input data. In regions of the input space that lack training data, model parameters are less constrained and different model parameter settings that produce diverse and potentially conflicting predictions can be comparably likely under the posterior.

For classification tasks where aleatoric and epistemic uncertainty don’t need to be differentiated, uncertainty can be measured using the predictive distribution entropy:

$$H[\mathbb{P}(y|x, D)] = - \sum_{y \in C} \mathbb{P}(y|x, D) \log \mathbb{P}(y|x, D)$$

Aleatoric uncertainty can be measured using expected entropy:

$$u_{alea} = \mathbb{E}_{\mathbb{P}(\theta|D)} H[\mathbb{P}(y|x, \theta)]$$

Mutual information can be used to measure epistemic uncertainty:

$$u_{epis} = I(\theta, y|D, x) = H[\mathbb{P}(y|x, D)] - \mathbb{E}_{\mathbb{P}(\theta|D)} H[\mathbb{P}(y|x, \theta)]$$

Monte Carlo estimates that are obtained by sampling from the posterior can be used to approximate the terms of these equations for our Bayesian models [40]. In particular, $\mathbb{P}(y|x, D) \approx \frac{1}{T} \sum_{i=1}^T \mathbb{P}(y|x, \theta_i)$ and $\mathbb{E}_{\mathbb{P}(\theta|D)} H[\mathbb{P}(y|x, \theta)] \approx \frac{1}{T} \sum_{i=1}^T H[\mathbb{P}(y|x, \theta_i)]$, where the θ_i are samples from the posterior over models, and T is the number of samples. For ensemble models which are not explicitly Bayesian (because each ensemble member receives the same weight), uncertainties can be computed in a similar way. In this case, the θ_i are no longer samples from a posterior, but multiple independent trainings of a model with T different random seeds. In our experiments, we measure overall uncertainty using predictive entropy.

2.2. Learning to Reject/Defer

Rejection Learning is a framework where a machine learning model can reject a data point at prediction time and choose to not make a prediction if it is not confident or is uncertain [41, 42, 43, 44, 45]. Importantly, Rejection Learning does not take into account any downstream models nor decision makers that will process rejected data points.

Madras et al. [46] extends and generalizes this idea with Adaptive Rejection Learning, also known as Learning to Defer. Unlike Rejection Learning, Learning to Defer allows models to learn to adapt

to the strengths, weaknesses, and biases of downstream decision makers and models. Mozannar and Sontag [47] use the method of Madras et al. [46] as a baseline and also tackle the problem of Learning to Defer. They provide a particularly good toy example of why the rejector and classifier need to be trained jointly. Similarly, Wilder et al. [48] introduce methods to optimize teams consisting of humans and ML algorithms.

Our approach differs from that of Madras et al. [46] in two ways. First, their approach can be seen as a mixture of Bernoullis: a binary deferral decision gating variable is used to both express uncertainty and to try to predict which model will have a lower loss. Our approach, on the other hand, explicitly decouples uncertainty from the prediction of a downstream model’s performance. This separation allows for an easier interpretation of the reasons for a deferral decision, and in many cases likely results in an easier learning problem. Second, the Bayesian uncertainty approach described in Appendix F of Madras et al. [46] uses only uncertainty to reject, and, unlike our work, does not provide a way to incorporate the prediction of a downstream model’s performance.

2.3. Active Feature Acquisition

An area related to Learning to Defer is active feature acquisition, the sequential decision process of whether to query more features or not under budget constraints. Gao and Koller [49] introduce an active classification process that combines multiple classifiers and features at test time using an instance-specific decision path. Similarly, Saar-tsechansky et al. [50] use Value of Information to rank feature acquisition options. There are many ways to measure the value of information, including information gain, classification loss, and decision robustness [51]. Active feature acquisition is useful not only for minimizing costs, but also for applications such as fairness. For instance, Noriega-Campero et al. [52] discuss active feature acquisition in the context of fair classification.

Xu et al. [53, 54] develop algorithms that select features individually for each test data point and reduce trees of classifiers into more efficient cascades of classifiers. Des Jardins et al. [55] construct a cascaded ensemble of classifiers to selectively acquire missing features at both train and test time. Wang et al. [45] develop Prediction Cascades, which use additional augmenting classifiers that evaluate the distributional output of the earlier classifier and estimate its uncertainty. Many cascade methods have an assumed ordering of cascade members, so are unable to adaptively reorder members based on the specific datum at hand. Gao and Koller [49] differs by balancing expected classification gain with computational cost. Observations are selected dynamically based on previous observations to achieve instance-specific decision paths.

Cascades are not the only approach, however; Xu et al. [56] don’t use a cascade of classifiers, instead choosing to extend stage-wise regression. They incorporate feature extraction cost into the objective during training, which minimizes test-time computation. Active feature acquisition can also be formulated as a Markov Decision Process [57, 58, 59] which allows for techniques from reinforcement learning to be used.

We note that active feature acquisition is likely excessive for our malware detection use case, as we have two classifiers with a clear ordering of static analysis then dynamic analysis.

More broadly, this problem is fundamentally related to the task of estimating the value of information [60]; that alone is a highly multidisciplinary problem that is rooted in Information Theory [61]. Behrens et al. [62] show evidence that humans modulate their learning rates based on environment volatility and uncertainty, and in a way that can be predicted by a Bayesian learner. This supports the hypothesis of Bayesian reasoning in humans.

Feltham [60] provide a formal framework for measuring the value of information in the context of accounting. In particular, they develop a framework to calculate the expected payoff for an information system from the perspective of the decision maker. Future research could include building off of this work in information theory by developing bespoke measures and cost functions that are applicable to cyber security and malware detection.

3. Data and Models

For malware and benign samples, we use the EMBER2018 dataset, which consists of portable executable files (PE files) scanned by VirusTotal in or before 2018 [63]. The dataset contains 600,000 labeled training samples and 200,000 labeled testing samples, with an equal number of malicious and benign samples in both sets. The malicious samples are also labeled by malware family using AVClass [64]. The train/test split in the data is temporal. For each sample, EMBER2018 includes vectorized features which encode general file information, header information, imported functions, exported functions, section information, byte histograms, byte-entropy histograms, and string information [63]. There are also 1.1TB of raw PE files that are not available as part of EMBER2018; these can be downloaded via VirusTotal.

Following the methodology of Nguyen et al. [11], we use a dropout Bayesian MalConv model [12] when working with raw binaries, and an LGBM model [65] for the EMBER extracted features. We also develop a model for dynamic analysis logs, and extract capabilities in the executable files. The latter is done using CAPA, with the SMDA recursive disassembler as a backend.²

For a comparison of computational costs, a 16 sample Bayesian MalConv model can process an average of about 51.2 EMBER2018 files per second on a NVIDIA Tesla P100 GPU. This is about 0.02 seconds per file for feature extraction, prediction, and sampling. The EMBER features take on average about 0.09 seconds to extract per file. CAPA feature extraction takes about 45.75 seconds per model on average. Finally, running dynamic analysis on a file takes an average of 526 seconds, or 8 minutes and 46 seconds. Note that these exclude the downstream prediction model, so the actual cost is slightly higher. In other words, running MalConv on a file is over 26,300 times faster than running a dynamic analysis model.

We note that we do not use larger datasets such as SoReL-20M of Harang and Rudd [66] due to the considerable computational cost. Running 20 million files through dynamic analysis would take over 333 years!

3.1. Dynamic Analysis Features and Model

We use a proprietary dynamic execution sandbox provided by a United States based Software Security company. The dynamic analysis tool uses libvirt to work with QEMU³ to manage Virtual Machines (VMs) running Windows 7 32-bit.

The dynamic analysis tool puts a sample into the VM, starts a custom monitoring application, runs the sample, saves the monitor's log file, and pulls the log from the VM to the host. Then, the tool kills the VM image and creates a new one from a template. This way, the VM is clean for every sample.

These dynamic analysis logs were featurized with a domain knowledge-driven approach similar to that of Anderson and Roth [63]. The dynamic analysis engine captures broad information about DNS, registry, file, process, and security events from the dynamic execution of the sample. These events contain raw information which we used to extract features for each file. The features include event subtype counts, event subtype information counts, number of unique processes by user, PID counts by user, event key frequencies, count of registry bytes written, registry key path information, timing information, authentication signature information, hashed event message, hashed command line information, and counts of unnamed events. We then train a LGBM model on top of these features.

We note that it is possible that some executables did not exhibit all their behavior or functionality during dynamic analysis. This could happen for a number of reasons. It could be because they were not in the proper environment. For instance, if the executable targets a specific Windows build, it may fail to run anywhere else. It might also not have access to all the resources needed for execution. The executable might not run if it can't connect to the internet, or to its command-and-control server. It's also possible that the executable has some built-in, anti-reverse engineering techniques. In that case, it might wait a few days before executing, for example.

²<https://github.com/mandiant/capa>

³<https://www.qemu.org/>

The point is, some malicious files may not act malicious during dynamic analysis. This means that even though dynamic analysis is more expensive than static analysis, it does not always result in better malware detection when used in isolation. We will show that judiciously running dynamic analysis on a subset of files will both improve malware detection accuracy and keep analysis costs low.

4. Predicting Capabilities in Executable Files

While most Machine Learning approaches to malware have focused on the detection of maliciousness, there has been less work on the identification of specific behaviors, which is a critical component to malware analysis [67].

We investigated the prediction of CAPA outputs using an 8 sample Bayesian MalConv model, as well as an 8 sample LGBM ensemble trained on the EMBER features. For prediction targets, we used the 350 CAPA rules from the Mandiant standard collection⁴, plus a target for failures, for a total of 351 non-exclusive labels for each file.

Figure 1 shows detection accuracy for each CAPA rule for both MalConv and the LGBM model. We note that accuracy is high across rules, suggesting that CAPA may not need to be run for all samples when outcomes can be predicted. The LGBM model is consistently more accurate than MalConv across rules, but requires more expensive feature extraction. Figure 2 shows the area under the receiver operating characteristic curve (AUC) for each CAPA rule, for both MalConv and the LGBM model. While LGBM generally does better than MalConv in terms of AUC, MalConv is better in the worst case.

Running CAPA is no small feat. CAPA is more than 2200 times more expensive than MalConv in terms of time, mainly due to the costs associated with disassembly. We can drastically reduce these computational costs by using CAPA exclusively when a prediction model, such as MalConv or LGBM, is highly uncertain. Specifically, CAPA should only deal with files for which the uncertainty is above some threshold.

Figure 3 shows the performance of such an approach for various uncertainty thresholds. The results show that we can achieve an average 99.9% of correctly labeled rules while running CAPA on less than half of the data.

The thresholds are measured by predictive entropy, and correspond to the proportion of the test data that is sent to CAPA. To simulate real world deployment of such a system, we plot two types of uncertainty thresholds: ideal and actualized. Ideal thresholds are those needed for the 99.9% average across all files; actualized are those chosen using a validation set. The validation and test sets were temporally split into samples first seen in November 2018 and December 2018, respectively.

We note that the ideal and actualized uncertainty thresholds are very close, and that compute time spent on CAPA extraction can be cut by more than half. This leads to substantial gains in a deployed production setting, where it is not feasible to run CAPA on all of the data.

In especially computationally-constrained settings, it might not be possible to run CAPA at all. In these cases, MalConv alone can achieve a 98% average of correctly labeled CAPA rules. This result suggests that MalConv is capable of learning features that are predictive of capabilities in executables.

5. Deferring to More Expensive Models

Dynamic malware analysis is more expensive than static malware analysis. Human malware analysis is more expensive than automated malware analysis. Ideally, we would like to minimize the analysis cost needed to determine if a file is malicious or benign. In other words, we want to run cheap, automated static analysis first, and only run more expensive analysis (such as dynamic analysis) if the static analysis model is uncertain. Additionally, we want to train the static model to be adaptive to the dynamic model's strengths and weaknesses. Even if uncertain, the static model should guess if the dynamic model will perform worse given the type of data being predicted on.

⁴<https://github.com/mandiant/capa-rules>

CAPA Prediction Accuracies By Rule

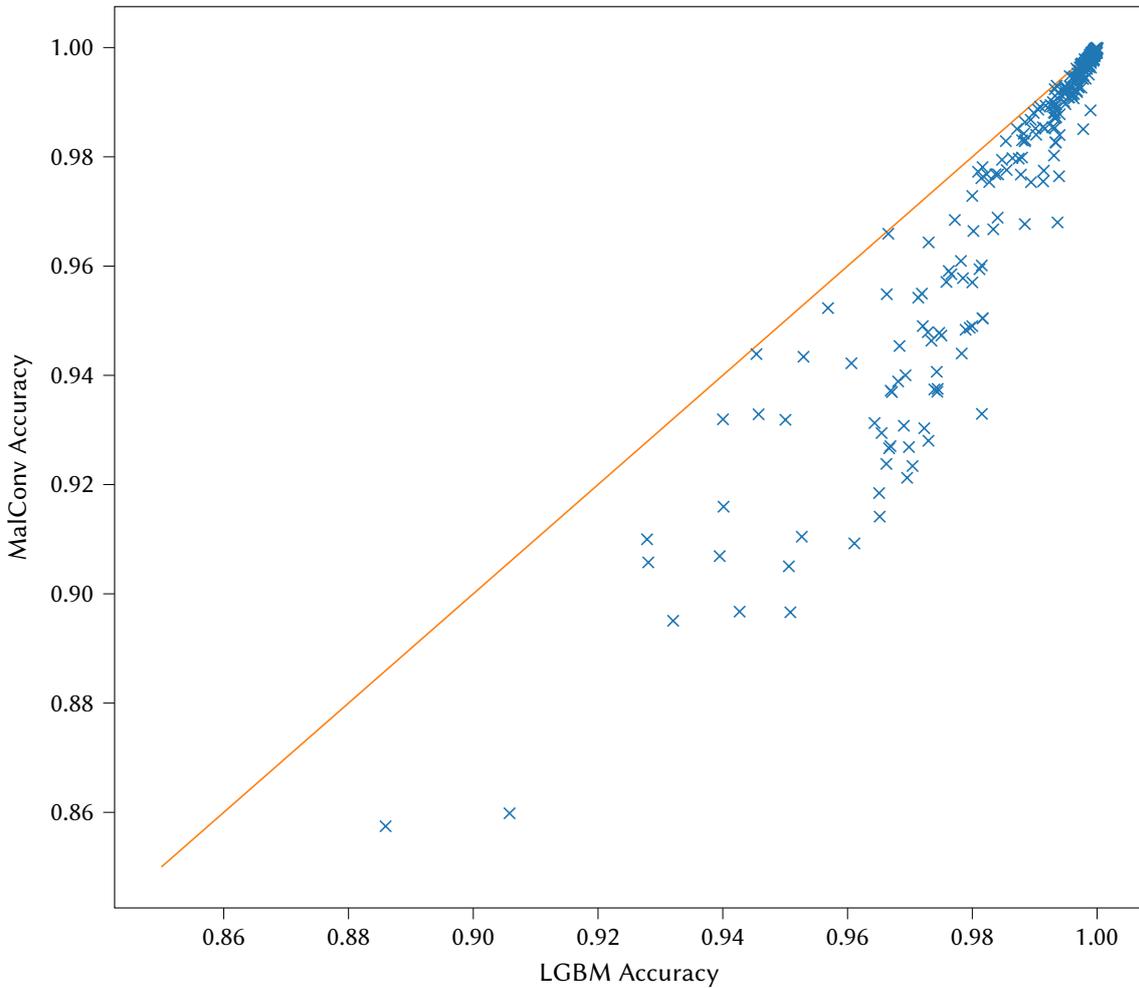


Figure 1: Detection accuracy for each CAPA rule for both MalConv and the LGBM model. The LGBM model is consistently more accurate than MalConv, but also more expensive.

Figure 4 shows the performance using Bayesian MalConv as the static model against various predictive entropy thresholds, which represent different total runtimes. Following the naming conventions from Madras et al. [46], “reject” means letting the dynamic model predict when Bayesian MalConv’s uncertainty is above a certain threshold; “defer” means letting the dynamic model predict only when Bayesian MalConv’s uncertainty is above a certain threshold, and when it guesses that the dynamic model will make the correct prediction.

To enable the Bayesian MalConv model to defer, we train it with two equally-weighted prediction tasks: the malware classification task, and the prediction of whether or not the dynamic model will be accurate on the sample. Algorithm 1 describes the training procedure for the Defer approach, algorithm 2 describes the deployment time prediction procedure for the Reject approach, and algorithm 3 describes the deployment time prediction procedure for the Defer approach. PyTorch is used to develop all of our code [68], the Adam optimizer with default recommended settings is used for training in all of our experiments [69], and a single NVIDIA Tesla P100 GPU was used.

Thresholds are chosen using a validation set. In Figure 4, we plot the actualized uncertainty thresholds that maximize malware detection accuracy. The validation and test sets are split into samples from November 2018 and December 2018 respectively. We also plot the accuracy achieved by an ensemble model, which averages the predictions of the MalConv and dynamic models together. Unsurprisingly, the ensemble model outperforms both MalConv and the dynamic model used in isolation. However, both MalConv and dynamic analysis need to be run on every single file in order to compute the ensemble

CAPA Prediction AUCs By Rule

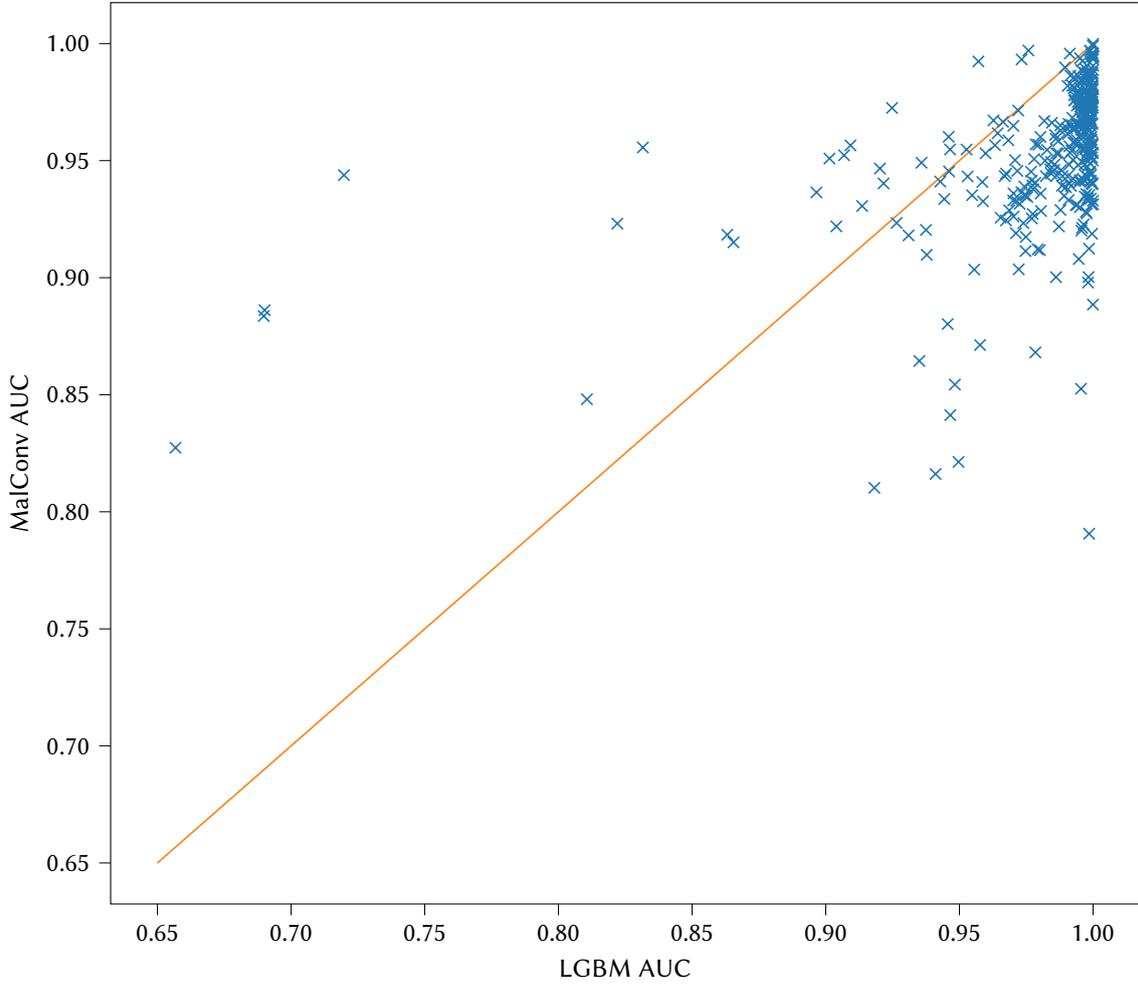


Figure 2: The AUC for each CAPA rule for both MalConv and the LGBM model. While LGBM generally does better than MalConv, MalConv is better in the worst case.

Algorithm 1 Training procedure for the Defer approach.

- 1: $f.train(X_{train}, Y_{train})$ \triangleright Train the dynamic model f (including feature extraction) on dataset of executables X_{train} , and maliciousness labels Y_{train} .
 - 2: $P_{train,f} \leftarrow f.predict(X_{train})$ \triangleright Compute dynamic model's predictions, $P_{train,f}$.
 - 3: $I_{train} \leftarrow (P_{train,f} == Y_{train})$ \triangleright Compute a new set of labels, I_{train} , corresponding to the dynamic model's success on individual files.
 - 4: $g.train(X_{train}, [Y_{train}, I_{train}])$ \triangleright
 Train the static model g (including feature extraction) on dataset of executables, X_{train} , to predict maliciousness labels, Y_{train} , and the dynamic model's success, I_{train} . The loss function used for training is $loss = ce(g.predict_Y(X_{train}), Y_{train}) + ce(g.predict_I(X_{train}), I_{train})$, where ce is the Cross Entropy Loss, $g.predict_Y$ predicts maliciousness, and $g.predict_I$ predicts the dynamic model's success.
-

predictions, so the cost is significantly higher.

We highlight that the rejection model with an actualized threshold achieves a test accuracy roughly equal to that of the ensemble model, while requiring dynamic analysis to be run on only 13.2 percent of the test data. This saves a year's worth of compute time compared to the ensemble model. The defer approach achieves a higher accuracy than that of the ensemble, and only requires dynamic analysis to be run on 17.3 percent of the test data.

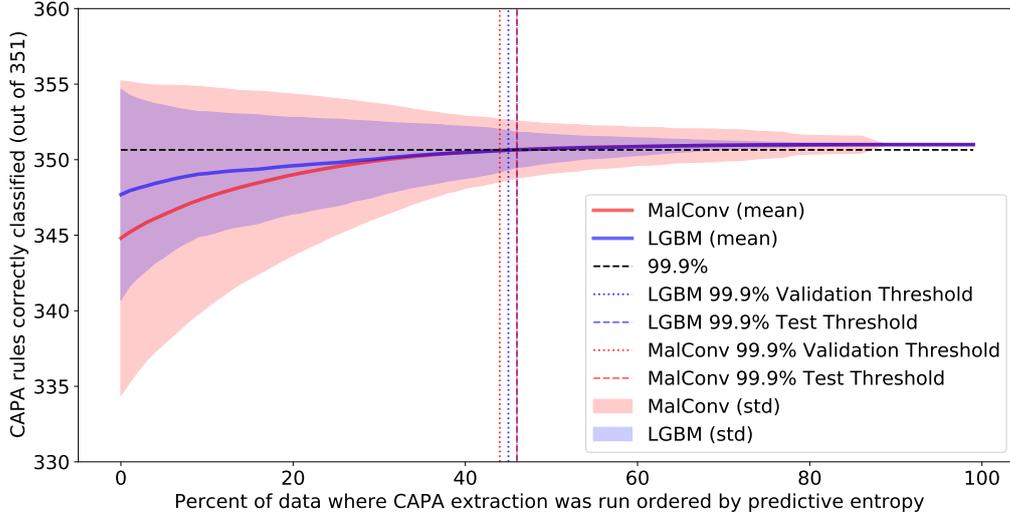


Figure 3: Performance with uncertainty-based rejection at various uncertainty thresholds. The x-axis maps the percentage of the data that requires expensive analysis. The far right represents CAPA running on all the data, and the far left only uses cheaper predictive models (MalConv or LGBM).

Algorithm 2 Making predictions in deployment using the Reject approach. In this setting, the static model g is trained to predict only maliciousness.

- 1: $Y_{pred} \leftarrow g.predict_Y(X_{test})$ \triangleright Predict maliciousness using the static model on a deployment time test input X_{test} .
 - 2: $U_{pred} \leftarrow g.uncertainty_Y(X_{test})$ \triangleright Compute the static model's uncertainty (as measured by predictive entropy) for its maliciousness prediction.
 - 3: **if** $U_{pred} > threshold$ **then** \triangleright If the static model's uncertainty is above the desired threshold, make a prediction using the dynamic model.
 - 4: $Y_{pred_{dynamic}} \leftarrow f.predict(X_{test})$
 - 5: **return** $Y_{pred_{dynamic}}$
 - 6: **else** \triangleright If the static model's uncertainty is below the desired threshold, return the static model's prediction.
 - 7: **return** Y_{pred}
 - 8: **end if**
-

Even if the uncertainty threshold is set too high, accuracy with the deferral approach never dips below that of the MalConv model. This confirms that MalConv is able to accurately predict when the dynamic model will make a mistake. This, along with MalConv's ability to accurately predict CAPA outputs, suggests that MalConv is capable of uncovering useful features in binary data.

Finally, in Figure 5, we show a similar analysis, but instead of the dynamic model as the expensive one, we use an LGBM model trained on EMBER features. In this case, running the more expensive model on all of the data (the ensemble method) offers the best accuracy, but is also the most expensive method by far. Both the reject and defer methods show the desired behavior of improving accuracy while limiting compute costs.

Algorithm 3 Making predictions in deployment using the Defer approach.

- 1: $Y_{pred} \leftarrow g.predict_Y(X_{test})$ \triangleright Predict maliciousness using the static model on a deployment time test input X_{test} .
 - 2: $U_{pred} \leftarrow g.uncertainty_Y(X_{test})$ \triangleright Compute the static model's uncertainty (as measured by predictive entropy) for its maliciousness prediction.
 - 3: $I_{pred} \leftarrow g.predict_I(X_{test})$ \triangleright Predict dynamic model's success using the static model.
 - 4: **if** $U_{pred} > threshold$ **and** $I_{pred} == 1$ **then** \triangleright If the static model's uncertainty is above the desired threshold and the static model predicts that the dynamic model will be successful, make a prediction using the dynamic model.
 - 5: $Y_{pred_{dynamic}} \leftarrow f.predict(X_{test})$
 - 6: **return** $Y_{pred_{dynamic}}$
 - 7: **else** \triangleright Else, return the static model's prediction.
 - 8: **return** Y_{pred}
 - 9: **end if**
-

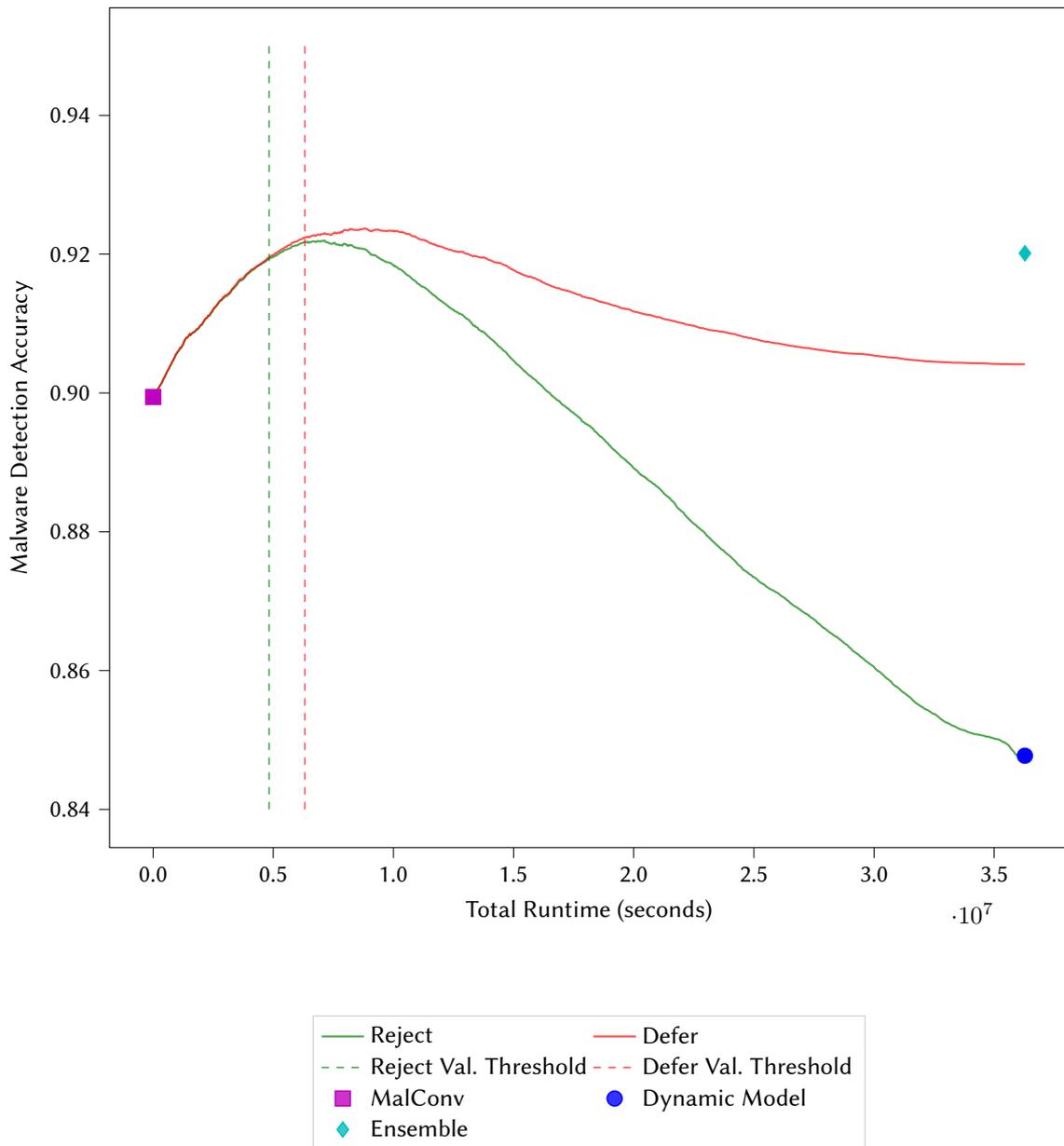


Figure 4: Performance with Bayesian MalConv as the cheap model that runs on all files, followed by a more expensive dynamic model. Both “Reject” and “Defer” require MalConv’s uncertainty to be above a certain threshold before turning to the dynamic model, but “Defer” also needs MalConv to predict that the dynamic model is likely to succeed.

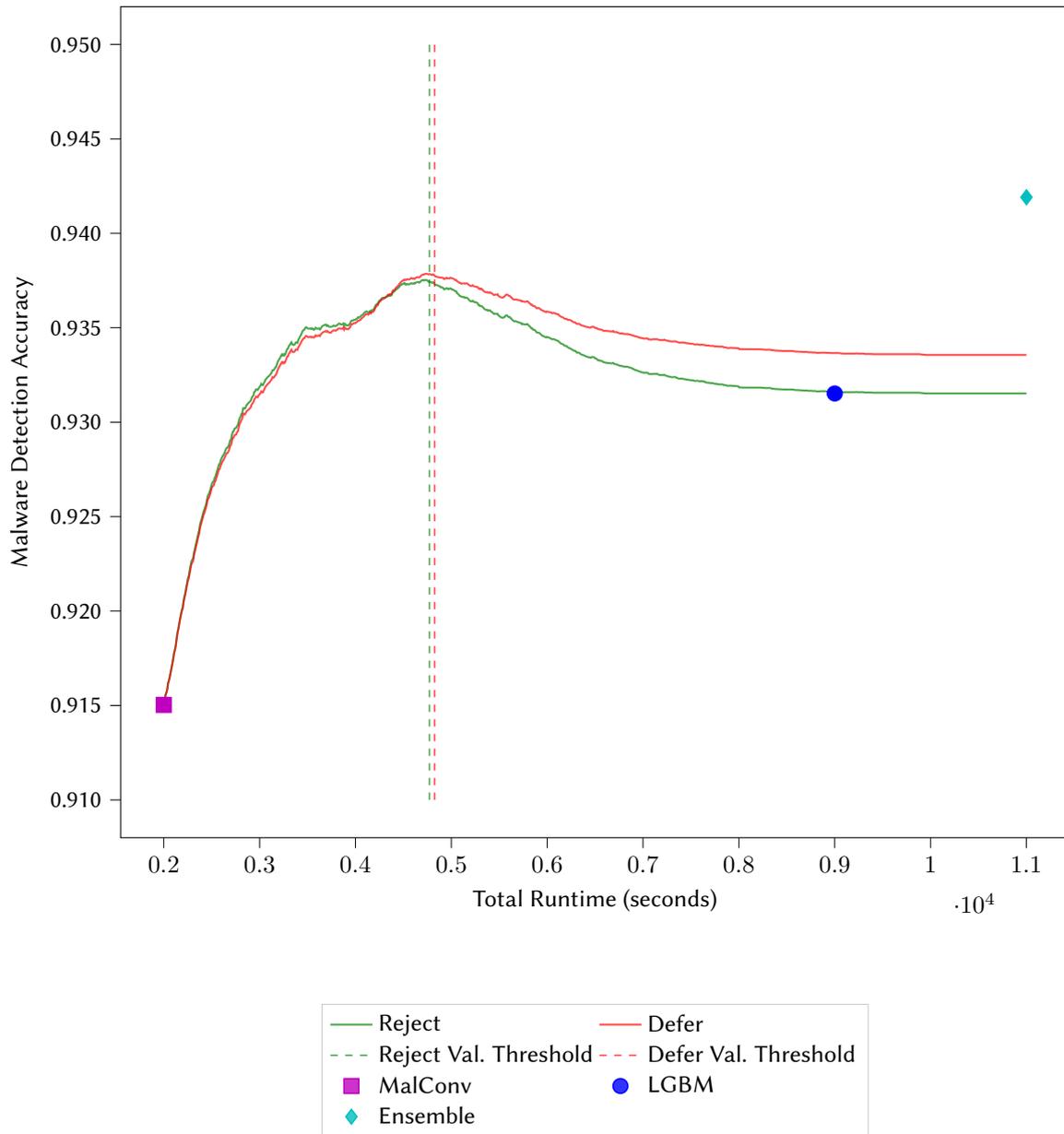


Figure 5: The performance with Bayesian MalConv as the cheap model, followed by LGBM on EMBER features as the expensive model, for various predictive entropy thresholds.

6. Conclusions

We have shown that cheaper static analysis features can be used to predict the outcomes of more expensive disassembly based analysis. We have also developed a dynamic analysis model that is complementary to static analysis models. In both cases, we showed how uncertainty can be used to minimize compute costs. Future work should explore chaining together a larger number of algorithms and approaches, examine more refined measures of cost, and assess applicability in other areas of cyber security.

References

- [1] R. Anderson, C. Barton, R. Boehme, R. Clayton, M. J. G. Van Eeten, M. Levi, T. Moore, S. Savage, Measuring the Changing Cost of Cybercrime, Workshop on the Economics of Information Security (WEIS) (2019). URL: https://weis2019.econinfosec.org/wp-content/uploads/sites/6/2019/05/WEIS_2019_paper_25.pdf.
- [2] P. Hyman, Cybercrime: it's serious, but exactly how serious?, Communications of the ACM 56 (2013) 18–20. doi:10.1145/2428556.2428563.
- [3] B. M. Eddy, N. Perlroth, Cyber Attack Suspected in German Woman's Death, The New York Times (2021).
- [4] E. C. Spafford, Is Anti-virus Really Dead?, Computers & Security 44 (2014) iv. doi:10.1016/S0167-4048(14)00082-0.
- [5] B. Li, K. Roundy, C. Gates, Y. Vorobeychik, Large-Scale Identification of Malicious Singleton Files, in: 7TH ACM Conference on Data and Application Security and Privacy, 2017.
- [6] E. Raff, C. Nicholas, A Survey of Machine Learning Methods and Challenges for Windows Malware Classification, arXiv preprint arXiv:2006.09271 (2020) 1–48. URL: <http://arxiv.org/abs/2006.09271>.
- [7] C. Wressnegger, K. Freeman, F. Yamaguchi, K. Rieck, Automatically inferring malware signatures for anti-virus assisted attacks, ASIA CCS 2017 - Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security (2017) 587–598. doi:10.1145/3052973.3053002.
- [8] M. Egele, T. Scholte, E. Kirda, C. Kruegel, A survey on automated dynamic malware-analysis techniques and tools, ACM Computing Surveys 44 (2012). doi:10.1145/2089125.2089126.
- [9] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, M. Van Steen, Prudent practices for designing malware experiments: Status quo and outlook, Proceedings - IEEE Symposium on Security and Privacy (2012) 65–79. doi:10.1109/SP.2012.14.
- [10] M. Backes, M. Nauman, LUNA: Quantifying and Leveraging Uncertainty in Android Malware Analysis through Bayesian Machine Learning, Proceedings - 2nd IEEE European Symposium on Security and Privacy, EuroS and P 2017 (2017) 204–217. doi:10.1109/EuroSP.2017.24.
- [11] A. T. Nguyen, E. Raff, C. Nicholas, J. Holt, Leveraging Uncertainty for Improved Static Malware Detection Under Extreme False Positive Constraints, IJCAI-21 1st International Workshop on Adaptive Cyber Defense (2021). URL: <http://arxiv.org/abs/2108.04081>.
- [12] A. T. Nguyen, F. Lu, G. Lopez Munoz, E. Raff, C. Nicholas, J. Holt, Out of Distribution Data Detection Using Dropout Bayesian Neural Networks, in: AAAI, 2022. URL: www.aaai.org.
- [13] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, C. Nicholas, Malware Detection by Eating a Whole EXE, Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence (2017). URL: <http://arxiv.org/abs/1710.09435>.
- [14] E. Raff, W. Fleshman, R. Zak, H. S. Anderson, B. Filar, M. Mclean, Classifying Sequences of Extreme Length with Constant Memory Applied to Malware Detection, in: AAAI, 2021.
- [15] J. M. Bernardo, A. F. Smith, Bayesian Theory, 2008. doi:10.1002/9780470316870.
- [16] H. Liu, L. Wasserman, Bayesian Inference, in: Statistical Machine Learning, In Preparation., 2014, pp. 299–351.
- [17] J. A. Hoeting, D. Madigan, A. E. Raftery, C. T. Volinsky, Bayesian Averaging Models, Statistical Science 14 (1999) 382–417.

- [18] D. J. C. MacKay, A Practical Bayesian Framework for Backpropagation Networks, *Neural Computation* 472 (1992) 448–472.
- [19] E. Daxberger, A. Kristiadi, A. Immer, R. Eschenhagen, M. Bauer, P. Hennig, Laplace Redux – Effortless Bayesian Deep Learning (2021). URL: <http://arxiv.org/abs/2106.14806>.
- [20] H. Ritter, A. Botev, D. Barber, A Scalable Laplace Approximation for Neural Networks, in: *ICLR*, 2018.
- [21] A. Kristiadi, M. Hein, P. Hennig, Learnable Uncertainty under Laplace Approximations (2020). URL: <http://arxiv.org/abs/2010.02720>.
- [22] R. M. Neal, Bayesian Learning for Neural Networks, Ph.D. thesis, University of Toronto, 1995.
- [23] M. Betancourt, A Conceptual Introduction to Hamiltonian Monte Carlo (2017). URL: <http://arxiv.org/abs/1701.02434>.
- [24] M. D. Hoffman, A. Gelman, The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo, *Journal of Machine Learning Research* 15 (2014) 1593–1623.
- [25] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, L. K. Saul, Introduction to variational methods for graphical models, *Machine Learning* 37 (1999) 183–233. doi:10.1023/A:1007665907178.
- [26] D. M. Blei, A. Kucukelbir, J. D. McAuliffe, Variational Inference: A Review for Statisticians, *Journal of the American Statistical Association* 112 (2017) 859–877. doi:10.1080/01621459.2017.1285773.
- [27] M. D. Hoffman, D. M. Blei, C. Wang, J. Paisley, Stochastic variational inference, *Journal of Machine Learning Research* 14 (2013) 1303–1347.
- [28] A. Kucukelbir, A. Gelman, Automatic Differentiation Variational Inference 18 (2017) 1–45.
- [29] R. Ranganath, S. Gerrish, D. M. Blei, Black box variational inference, *Journal of Machine Learning Research* 33 (2014) 814–822.
- [30] M. K. Titsias, M. Lázaro-Gredilla, Doubly stochastic variational bayes for non-conjugate inference, 31st International Conference on Machine Learning, *ICML 2014* 5 (2014) 4056–4069.
- [31] D. Duvenaud, R. P. Adams, Black-Box Stochastic Variational Inference in Five Lines of Python (2014) 1–4.
- [32] G. E. Hinton, D. van Camp, Keeping neural networks simple by minimizing the description length of the weights, *COLT* (1993) 5–13. doi:10.1145/168304.168306.
- [33] A. Graves, Practical variational inference for neural networks, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011* (2011) 1–9.
- [34] C. Blundell, J. Cornebise, K. Kavukcuoglu, D. Wierstra, Weight uncertainty in neural networks, 32nd International Conference on Machine Learning, *ICML 2015* 2 (2015) 1613–1622.
- [35] Y. Gal, Z. Ghahramani, Dropout as a Bayesian approximation: Representing model uncertainty in deep learning, 33rd International Conference on Machine Learning, *ICML 2016* 3 (2016) 1651–1660.
- [36] Y. Gal, Uncertainty in Deep Learning, Ph.D. thesis, University of Cambridge, 2016.
- [37] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, *arXiv preprint arXiv:1207.0580* (2012) 1–18. URL: <http://arxiv.org/abs/1207.0580>.
- [38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *Journal of Machine Learning Research* 15 (2014) 1929–1958.
- [39] A. C. Damianou, N. D. Lawrence, Deep Gaussian Processes, *Artificial intelligence and statistics, PMLR* 31 (2013) 207–215.
- [40] L. Smith, Y. Gal, Understanding measures of uncertainty for adversarial example detection, 34th Conference on Uncertainty in Artificial Intelligence 2018, *UAI 2018* 2 (2018) 560–569.
- [41] C. Cortes, G. DeSalvo, M. Mohri, Learning with rejection, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9925 LNAI (2016) 67–82. doi:10.1007/978-3-319-46379-7_{_}5.
- [42] C. Chow, An Optimum Character Recognition System Using Decision Functions, *IRE Transactions*

- on Electronic Computers EC-7 (1958) 180. doi:10.1109/TEC.1958.5222530.
- [43] C. K. Chow, On optimum recognition error and reject tradeoff, 1969.
 - [44] J. Attenberg, P. Ipeirotis, F. Provost, Beat the Machine: Challenging Workers to Find the Unknown Unknowns, AAAI (2011).
 - [45] X. Wang, Y. Luo, D. Crankshaw, A. Tumanov, F. Yu, J. E. Gonzalez, IDK Cascades: Fast deep learning by learning not to Overthink, 34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018 2 (2018) 580–590.
 - [46] D. Madras, T. Pitassi, R. Zemel, Predict responsibly: Improving fairness and accuracy by learning to defer, Advances in Neural Information Processing Systems 2018-Decem (2018) 6147–6157.
 - [47] H. Mozannar, D. Sontag, Consistent Estimators for Learning to Defer to an Expert (2020). URL: <http://arxiv.org/abs/2006.01862>.
 - [48] B. Wilder, E. Horvitz, E. Kamar, Learning to Complement Humans (2020) 1526–1533. doi:10.24963/ijcai.2020/212.
 - [49] T. Gao, D. Koller, Active classification based on value of classifier, Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011 (2011) 1–9.
 - [50] M. Saar-tsechansky, P. Melville, F. Provost, Active Feature-Value Acquisition (????) 1–32. URL: <papers://5e3e5e59-48a2-47c1-b6b1-a778137d3ec1/Paper/p1779>.
 - [51] S. Chen, A. Choi, A. Darwiche, Value of information based on decision robustness, Proceedings of the National Conference on Artificial Intelligence 5 (2015) 3503–3510.
 - [52] A. Noriega-Campero, B. Garcia-Bulle, M. A. Bakker, A. S. Pentland, Active fairness in algorithmic decision making, AIES 2019 - Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society (2019) 77–83. doi:10.1145/3306618.3314277.
 - [53] Z. Xu, M. J. Kusner, K. Q. Weinberger, M. Chen, Cost-sensitive tree of classifiers, 30th International Conference on Machine Learning, ICML 2013 28 (2013) 133–141.
 - [54] Z. E. Xu, M. J. Kusner, K. Q. Weinberger, M. Chen, O. Chapelle, Classifier cascades and trees for minimizing feature evaluation cost, Journal of Machine Learning Research 15 (2014) 2113–2144.
 - [55] M. Des Jardins, J. MacGlashan, K. L. Wagstaff, Confidence-based feature acquisition to minimize training and test costs, Proceedings of the 10th SIAM International Conference on Data Mining, SDM 2010 (2010) 514–524. doi:10.1137/1.9781611972801.45.
 - [56] Z. Xu, K. Q. Weinberger, O. Chapelle, The Greedy Miser: Learning under test-time budgets, Proceedings of the 29th International Conference on Machine Learning, ICML 2012 2 (2012) 1175–1182.
 - [57] H. Shim, S. J. Hwang, E. Yang, Joint active feature acquisition and classification with variable-size set encoding, Advances in Neural Information Processing Systems 2018-Decem (2018) 1368–1378.
 - [58] T. Rückstieß, C. Osendorfer, P. Van Der Smagt, Sequential feature selection for classification, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7106 LNAI (2011) 132–141. doi:10.1007/978-3-642-25832-9{_}14.
 - [59] G. Dulac-Arnold, L. Denoyer, P. Preux, P. Gallinari, Datum-Wise classification: A sequential approach to sparsity, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 6911 LNAI (2011) 375–390. doi:10.1007/978-3-642-23780-5{_}34.
 - [60] G. A. Feltham, The value of information, The Accounting Review (1968).
 - [61] J. McCarthy, Measures of the Value of Information, Proceedings of the National Academy of Sciences 42 (1956) 654–655. doi:10.1073/pnas.42.9.654.
 - [62] T. E. Behrens, M. W. Woolrich, M. E. Walton, M. F. Rushworth, Learning the value of information in an uncertain world, Nature Neuroscience 10 (2007) 1214–1221. doi:10.1038/nn1954.
 - [63] H. S. Anderson, P. Roth, EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models, arXiv preprint arXiv:1804.04637 (2018). URL: <http://arxiv.org/abs/1804.04637>.
 - [64] M. Sebastián, R. Rivera, P. Kotzias, J. Caballero, AVCLASS: A Tool for Massive Malware Labeling,

- in: International symposium on research in attacks, intrusions, and defenses., 2016. URL: <https://github.com/malicialab/avclass>.
- [65] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, LightGBM: A Highly Efficient Gradient Boosting Decision Tree, Technical Report, ??? URL: <https://github.com/Microsoft/LightGBM>.
- [66] R. Harang, E. M. Rudd, SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection (2020). URL: <http://arxiv.org/abs/2012.07634>.
- [67] M. R. Smith, A. J. Carbajal, C. C. Lamb, N. T. Johnson, R. Ramyaa, S. J. Verzi, J. B. Ingram, E. Domschot, W. Philip Kegelmeyer, Mind the Gap: On Bridging the Semantic Gap between MachineLearning and Malware Analysis, arXiv (2020) 49–60.
- [68] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An imperative style, high-performance deep learning library, Advances in Neural Information Processing Systems 32 (2019).
- [69] D. P. Kingma, J. Ba, Adam: A Method for Stochastic Optimization, International Conference for Learning Representations (2015) 1–15. URL: <http://arxiv.org/abs/1412.6980>. doi:<http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503>.