Analyzing Machine Learning Approaches for Online Malware Detection in Cloud

Jeffrey C Kimmell*, Mahmoud Abdelsalam†, and Maanak Gupta‡
*‡Dept. of Computer Science, Tennessee Technological University, Cookeville, Tennessee 38505, USA
†Dept. of Computer Science, Manhattan College, Riverdale, NY, USA
*jckimmell42@tntech.edu, †mabdelsalam01@manhattan.edu, ‡mgupta@tntech.edu

***Abstract*—The variety of services and functionality offered by various cloud service providers (CSP) have exploded lately. Utilizing such services has created numerous opportunities for enterprises infrastructure to become cloud-based and, in turn, assisted the enterprises to easily and flexibly offer services to their customers. The practice of renting out access to servers to clients for computing and storage purposes is known as Infrastructure as a Service (IaaS). The popularity of IaaS has led to serious and critical concerns with respect to the cyber security and privacy. In particular, malware is often leveraged by malicious entities against cloud services to compromise sensitive data or to obstruct their functionality. In response to this growing menace, malware detection for cloud environments has become a widely researched topic with numerous methods being proposed and deployed. In this paper, we present online malware detection based on process level performance metrics, and analyze the effectiveness of different baseline machine learning models including, Support Vector Classifier (SVC), Random Forest Classifier (RFC), K-Nearest Neighbor (KNN), Gradient Boosted Classifier (GBC), Gaussian Naive Bayes (GNB) and Convolutional Neural Networks (CNN). Our analysis conclude that neural network models can most accurately detect the impact malware have on the process level features of virtual machines in the cloud, and therefore are best suited to detect them. Our models were trained, validated, and tested by using a dataset of 40,680 malicious and benign samples. The dataset was complied by running different families of malware (collected from VirusTotal) in a live cloud environment and collecting the process level features.**

I. Introduction and Motivation

Cloud computing's convenience and scalability have made it the go to resource for many entities in both private and public sectors. One of the major cloud characteristics is offering resources on-demand, if and when needed using the pay-as you go model. In general, IT specialists at enterprises have to control and manage these resources which hinders the advantages offered by the cloud. Cloud automation tools have become the norm where IT personnel are able to automatically provision resources in the cloud. Such automation is achieved through tools (e.g., Puppet[1] and Chef[2]) by writing configuration scripts that are able to create, modify, and delete resources in the cloud. Just as such orchestration tools offer huge benefits to DevOps teams, they widen the security attack surface. In particular, VMs are often spawned using automatic configuration tools which means that a large group VM are similarly configured, if not exact copies. The inherent redundancies in these VMs could allow for malware to easily propagate across VMs, especially if there are flaws in these configuration scripts. The repercussions of compromising a group of VMs far outweighs those of a single compromised VM. Cloud infrastructure requires considerably major security implementations due to its inherent complexity and dynamic environment where threats are always changing and evolving. For the same reason, developing malware detection methods that are both accurate and fast is imperative [1].

Malware is a major threat to cloud infrastructures. Multiple malware detection methods have been proposed with pros and cons. Static malware detection [2], [3], [3], [4] is a popular method where the signature of an executable is analyzed and compared to a database of known malware signatures. Attackers have tried to limit the effectiveness of static analysis by implementing techniques such as obfuscation and packing. In addition, static malware analysis is limited to known malware executables and is unable to detect the ever-evolving zero-day malware. These two major limitations have led to extensive research on behavioral malware detection methods. Dynamic and online malware detection are two behavioral based methods. Dynamic malware detection methods work by running the malware executables in a secure environment, such as a sandbox and analyzing their behavior. By doing this, the detection system is able to analyze novel zero-day malware since it is not relying on previously known signatures but the actual behavior of the executable. However, attackers have been able to implement malware that can detect the use of tools such as a sandbox, and cease behaving maliciously in order to avoid its detection. Dynamic and static methods also share the same limitation where the detection system focus on identifying malware in the given executables before they are run on actual systems. However, it is common for malware to get into a system through vulnerabilities, hence bypassing these primitive detection approaches. Online malware detection [5]–[8] focuses on the behavior of a machine that it is trying to protect from malware. Rather than analyzing executables and their behavior, online methods monitor the performance of the entire virtual machine, and raises an alert if traces of malicious behavior is found at any time. As such, online malware detection methods are considered continuous real-time detection system, and overcomes the limitations of static and dynamic malware detection approaches.

Machine Learning (ML) and neural networks techniques are

[1]Puppet. https://puppet.com/
[2]Chef. https://www.chef.io/

2693-8340/21/$31.00 ©2021 IEEE
DOI 10.1109/SMARTCOMP52413.2021.00046

widely utilized in order to capture the behavior of malware in an accurate and efficient way [9]. This is due to the models' abilities to quickly process a significant amount of data generated by a VM to classify executables as malicious or benign. Online malware detection techniques are significantly impacted by the features chosen to capture the behavior of the malware present in a particular machine. For example, several works [2], [10], [11] are using system calls (the most widely used); however, it is very resource consuming and data can only be fetched through on-host collecting agent. Only few works [1], [5]–[8], [12], [13] use resource utilization metrics (also known as performance metrics) due to the fact that they are less expressive than system calls in terms of capturing the low-activity malicious behavior. However, performance metrics are more suitable to cloud environments since they are *cheaper* and can easily be fetched from the hypervisor (e.g., VMs introspection [14]).

In this paper, we analyze and compare the effectiveness of different online malware detection approaches that utilize process-level performance metrics. We provide an in depth analysis of various machine learning models which will work as a baseline for other works which focus primarily on one machine learning model. This is critical to motivate the use of *expensive* deep learning models which require huge amounts of training data and to prove their efficacy with respect to more fundamental machine learning models. To our understanding and literature review, this is the first work focusing on analyzing the efficiency of baseline machine learning models, which is important to *justify* the use of expensive deep learning techniques.

The *main contributions* of this paper are as follows:

- We analyze the effectiveness of different machine learning models for online malware detection.
- We demonstrate how the set of processes running in a VM can be represented as a sequence of system features.
- We conjecture that a Convolutional Neural Network (CNN) model is better suited for malware detection compared to traditional ML methods.

The remainder of the paper is structured as follows. Section II discusses the related works regarding cloud malware detection and use of various machine learning models. We also elaborate different ML models used in our work. Section III discusses the experimental cloud set up and methodology. The results generated by each of the models are discussed in Section IV. Section V offers comparison and analysis of different ML approaches used followed by limitations in Section VI. Section VII summarizes our work.

II. Background

We will outline related work in malware detection and summarize different ML models used in our work.

A. Related Works

There has been substantial work in the field of malware detection. Most recently, approaches that rely on machine learning techniques have gained traction. The high increase in cloud activity has also called for more attention towards methods that are specific to the cloud environment [1], [5]–[7], [10], [20], [21]. Table I shows some of the closely related work. We categorize the work with respect to the focus of the paper, the targeted environment, and the features used for detection as well as the ML algorithms used.

Dynamic Malware Detection. Dynamic malware detection approaches focus on running malware executables in a sandbox and closely monitor its behavior or system wide behavior. Most works target traditional host-based systems. Research in [11], [15] utilize system calls as features to train classical machine learning models (i.e. KNN, NB, SVC and DT) and neural networks, respectively. Other works [9], [20], [21] analyze the effectiveness of using CNN, RF and KNN models for malware detection and rely on features extracted from API calls. In addition, Joshi et al. [23] use the random forest classifier and monitor a VM's process behavior. However, their analysis of these approaches is not analyzed on a cloud environment. The main limitation of such approaches is the fact that they are performed in an isolated environment neglecting the unique cloud topology, including the infrastructure and its network communication channels. Even though dynamic analysis approaches can be adopted and used in online settings, collecting real-time metrics generated from the cloud environment is essential for cloud malware detection.

Online Malware Detection. Unlike static and dynamic analysis approaches where an executable is analyzed or monitored before it runs on a system, online malware detection approaches focus on continuously monitoring the entire systems under the assumption that a malware will eventually make its way into the system. The authors in [19] introduced a malware detection method that utilizes performance counters and [22] proposed the use of memory features; however, both of these works are targeting traditional host-based environment. Other works specifically target the cloud. Abdelsalam et al. [7] presented a CNN solution that focused on process level performance metrics with a relatively successful accuracy score of 90%. However, this work only examines CNN and does not provide a baseline of comparison with respect to traditional machine learning algorithm, which we aim to accomplish in this paper. In addition, we also categorize anomaly detection based approaches as online techniques, since they naturally focus on continuous monitoring of their target systems. Pannu et al. [18] use cloud performance metrics features and analyzed the effectiveness SVM and Gaussian based approaches. Even though their work focused on general anomaly detection within the cloud, it can be easily adopted and tailored to detect malware specifically. Similarly, Guan et al. [17] considered anomaly detection within a cloud environment where they analyzed system calls based on an ensemble of Bayesian predictors and decision trees. Their work also focused on cloud computing systems failure, not specifically malware. Other anomaly detection based approaches are focused on malware. Azmandian et al. [16] presented an intrusion detection system using system calls as features. Abdelsalam et al. [6] proposed

TABLE I

THIS TABLE SHOWS THE DIFFERENCES BETWEEN OUR WORK AND RELATED WORKS. THE SECTIONS IN RED INDICATE A DIFFERENCE IN FEATURES USED, ENVIRONMENT THAT THE SOLUTION WAS TESTED IN, OR FOCUS OF THE PAPER. A ✓ INDICATES THAT A PARTICULAR PAPER POSSESSES THIS ATTRIBUTE OR MODEL, WHEREAS A BLANK CELL INDICATES AN ABSENCE OF THIS ATTRIBUTE OR MODEL.

	Features					Domain		Focus			Models Used								
Paper	API Calls	Performance Metrics	System Calls	Performance Counters	Memory Features	Cloud Environment	Traditional Host-Based Environment	Dynamic Malware Detection	Online Malware Detection	Anomaly Detection	KNN	Naive Bayes	Neural Network	Random Forest	Boosted Trees	SVC	Clustering	Decision Trees	No Machine Learning
Firdausi et al. 2010 [15]			✓				✓	✓			✓	✓				✓		✓	
Azmandian et al. 2011 [16]			✓			✓			✓	✓	✓						✓		
Guan et al. 2012 [17]		✓				✓			✓	✓		✓						✓	
Pannu et al. 2012 [18]		✓				✓				✓						✓			
Demme et al. 2013 [19]				✓			✓		✓		✓			✓				✓	
Pirscoveanu et al. 2015 [20]		✓					✓	✓						✓					
Watson et al. 2015 [1]		✓				✓			✓	✓						✓			
Luckett et al. 2016 [11]			✓				✓	✓					✓						
Fan et al. 2016 [21]	✓						✓	✓			✓								
Tobiyama et al. 2016 [9]	✓						✓	✓					✓						
Abdelsalam et al. 2017 [6]		✓				✓			✓	✓							✓		
Xu et al. 2017 [22]					✓		✓		✓					✓					
Abdelsalam et al. 2018 [7]		✓				✓			✓				✓						
Dawson et al. 2018 [10]			✓			✓			✓	✓			✓						✓
Joshi et al. 2018 [23]		✓					✓	✓						✓					
Our Approach		✓				✓			✓		✓	✓	✓	✓	✓	✓			

a novel k-means clustering algorithm for detection purposes. This approach succeeded in detecting highly active malware, but was not successful in detecting low activity malware. Dawson et al. [10] fetch API calls through hypervisor to be used as features and use a non linear phase-space algorithm to detect anomalous behavior. Watson et al. [1] use performance metrics to build a one class SVC; however, the authors experimented on highly active malware which is easy to detect.

In this paper, we aim to address the following limitations:

1) Unlike traditional host-based approaches in [9], [11], [15], [19]–[23], we aim to focus on developing a cloud-specific approach. Our experiment deployment, which consists of a commonly used three-tier web architecture, gives our collected data the added benefit of being generated in an extremely realistic cloud environment. The different layers of this architecture allows the utilization of the cloud topology and provides an in-depth view at how a real system could be affected by malware.
2) Unlike dynamic analysis approaches in [9], [11], [15], [20], [21], we aim to focus on developing an online malware detection approach that is well suited for cloud environments.
3) Unlike the work in [9]–[11], [15], [16], [19], [21], [22], we focus on performance-level metrics which is more practical for cloud than costly features like system or API calls.
4) Unlike the work in [1], we aim to focus on a broad range of malware by using low-active malware from seven different categories.
5) Unlike the work in [7], we aim to provide a baseline comparison by employing various traditional machine learning algorithms to convey the importance of using deep learning algorithms for online malware detection in cloud.

B. Baseline Machine Learning Models

Here we outline different ML models used in our analysis.

1) Support Vector Classifier (SVC): SVC are supervised learning models that are used for classification. SVC's ability to use a non-linear kernel gives this method the ability to efficiently perform non-linear classifications. This also reduces the computational power required to calculate relationships in infinite dimensions. There is not always a possible linear classification discernible between features, so finding higher dimensional relationships between the supplied features allows SVC to make classifications that other methods, such as logistic regression, would not be able to make.

2) Random Forest Classifier (RFC): RFC are supervised learning models used for classification. An RFC works by fitting a collection of decision trees [24], and is therefore considered an ensemble learning method since it uses a collection of classifiers [25]. RFCs choose the best parameter at each node at random as opposed to decision trees where the

best parameter is selected based on all of the features [23]. This gives RFCs better scalability as well as reducing the risk of overfitting.

3) Nearest Neighbor: Also known as k-Nearest Neighbor (KNN), it is a supervised learning method of classification that relies on measuring the distances of samples that have close proximity. KNN assumes that samples with the same classification will be closer in distance, and uses this assumption to classify a new sample based on the k closest neighbors.

4) Gradient Boosted Trees: Gradient Boosted Trees, or Gradient Boosted Classifiers (GBC), like RFC, is an ensemble supervised learning method. GBCs work by creating many decision trees that handle specific decisions. A weak learner is considered a model that is only slightly better than guessing. However, weak learners are often able to make correct decisions on a very specific portion of the sample. The idea is to create an ensemble of enough weak learners so that the model as a whole can uses the decisions from the various learners to generate an overall accurate classification.

5) Naive Bayes: A Naive Bayes classifier generates classifications using the Bayes Theorem. Bayes Theorem is a method used to calculate conditional probability based off a set of features, however it requires a large pool of computational resources. Bayes Theorem assumes that all features are dependent on one another, this is what leads to the theorem becoming so computationally demanding. To remedy this, a simplified or *Naive* method was created by assuming that each of the features are independent; this assumption allows the theorem to be simplified which therefore reduces the required computational resources.

6) CNN: Convolutional Neural Networks are most commonly used for image recognition or use cases that involve visual data. We can imitate this with our data by shaping it so that it is represented in a two dimensional array, similar to an image. The CNN model that we employ in our paper is DenseNet-121 model, which is one of the state-of-the-art CNN models.

DenseNets [26] attempt to solve the issue of the ”vanishing gradient.” This problem arises when the neural network becomes so deep that the standard back-propagation fails to update the neurons at the early layers of the network from the changes made at the output. Because of this issue, neural networks were limited in deepness and complexity. DenseNets work around this issue by creating more channels that connect the hidden layers. In DenseNets, the outputs of every layer are passed to subsequent layers. Because of this, DenseNets do not need as many feature maps at every hidden layer since these feature maps are being used by every subsequent layer and not just the successor layer. In addition, DenseNets borrow the identity mapping feature from residual networks. This feature allows the gradient to flow through the model easily via the use of skip-connections. DenseNets are comprised of dense blocks and connected with transition layers that are made of a convolution and pooling layer. For simplicity, we will refer to it as CNN for the remainder of the paper.

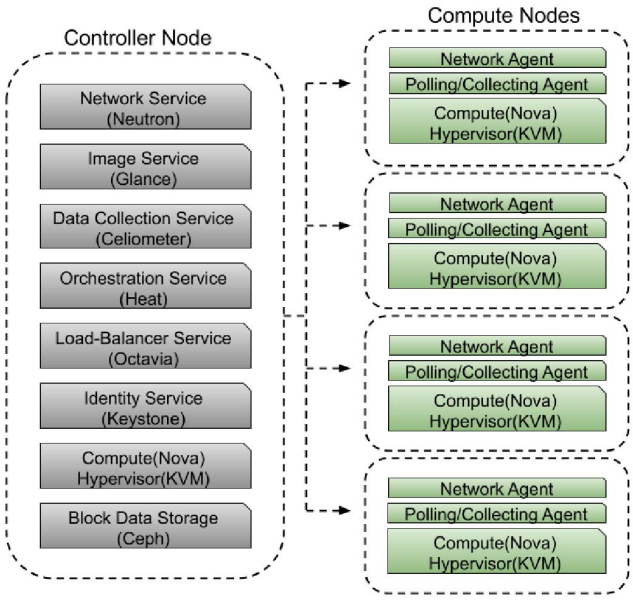

Fig. 1. Experimental Cloud Testbed Setup

III. Methodology and Experimental Setup

A. Experimental Setup

1) Testbed: Simulating real world data is an imperative aspect to creating reliable malware prediction models. In order to achieve this, we set up a cloud environment with traffic to emulate real world cloud behavior. OpenStack, a popular cloud platform, was installed on the testbed and consisted of a single control node and four compute nodes as shown in Figure 1. The control node is responsible for tasks such as the dashboard, storage, network, identity, and computing. The compute nodes only handle computing services, and each compute node is also supplied with agents for networking, polling, and collecting. Allowing malware to behave naturally is also key in the data collection process. To ensure the malware was able to behave as *intended*, all of the experiments to collect system features were conducted on machines that were connected to the Internet. This was done since some malware has the capability to detect a closed environment such as a sandbox, a limitation of dynamic malware analysis approaches. If a closed environment is detected, the malware may try to act as a benign process or may cease its behaviour as to not be detected. The Internet connection ensures that malware is able to communicate with its necessary command and control servers needed for malware to act in some cases. In addition, all firewalls and antivirus were disabled.

2) Malware Samples: The malware that was injected during our experiments were obtained from VirusTotal[3]. 113 samples were gathered in total and were chosen at random consisting of diverse malware families such as DoS, Backdoor, Trojan, Virus, among others.

[3]https://www.virustotal.com/gui/

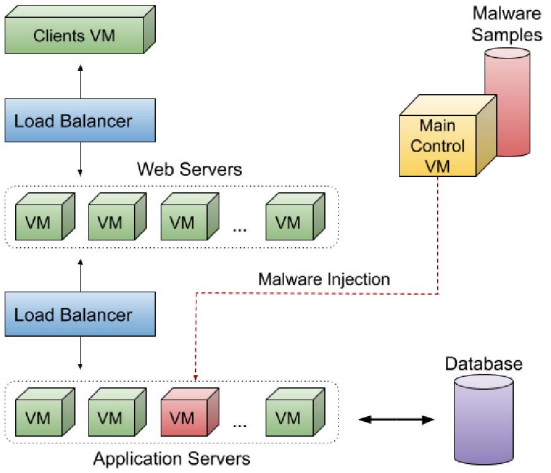

Fig. 2. Experiment Deployment

3) Experiment Deployment: In order to simulate a real world scenario, a three-tiered web architecture was used as shown in Figure 2. This architecture consisted of web-servers, application servers, and a database server. A front load balancer is tasked with handling and distributing clients requests to the appropriate web servers. An internal load balancer is used to connect web servers to the application servers and to distribute requests among the application servers, and the application servers are all connected to a single database server. An auto scaling policy is also utilized which is based on the average CPU usage and is applied independently to both the web and application servers. The policy states that if the average CPU utilization of all VMs belonging to the web or application layer exceeds 70%, new VMs are created and attached to the corresponding load balancer. Inversely, if the average CPU utilization falls below 40%, VMs are deleted to reduce resource usage. During our experiments, anywhere between 2 to 10 servers were spawned at each layer, depending on the overall traffic load. In order to uphold integrity, the traffic/requests were generated based on an ON/OFF Pareto distribution, this is done to mimic the realistic dynamic behavior of cloud infrastructures. A main control VM is used for keeping the malware executables in the database, injecting a single malware sample into an application server at a specific time, and deploying/destroying the experiment stack. OpenStack Heat orchestration service is used to deploy/destroy experiment stacks using *yaml* scripts.

4) Unique Processes: We collect system features (e.g. memory, cpu, input/output etc.) from all process that are running on the VM at certain times. The 45 features that were collected are used to represent the dynamic behavior of each of the processes [13]. Many of these processes are short lived, and also have their process IDs reassigned by the operating system. Due to these characteristics, it is often difficult to analyze the behavior of these processes. To remedy this, we utilize a method known as *unique processes* inspired by [7] which reduces dynamism. Traditional operating systems identify processes with a *pid*, whereas unique processes consider the actual behavior of a process and is identified using a tuple of two elements, *process name* and *the command used to run the process*. Processes that share common values in both fields within the tuple are clustered by taking the average of their measurements. By using this approach, we are also able to reduce the total number of processes within a given sample.

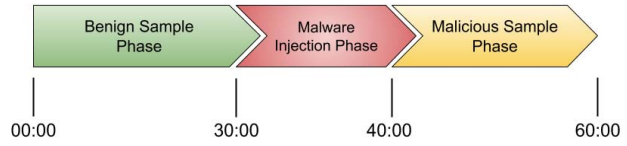

Fig. 3. Experiment Timeline

5) Data Collection: A total of 113 experiments were conducted each using a different malware executable. Each of the 113 experiments lasted for 1 hour and generated 360 data samples (sample collected every 10 seconds) for a total of 40,680 samples. Each experiment is split up into benign and infected phases as shown in Figure 3. The first 30 minutes is the benign phase, during this time there is no malware injected into the machine. Between minute 30 and 40, a single malware is injected into one of the application servers. The malware injection and execution times varies which adds a more dynamic nature to the experiments and ensures that a rigid injection and execution would not skew the results. Minute 40 is referred to as the malicious phase. During this time, malware is openly running on the machine. Data samples are collected every 10 seconds, this results in a total of 360 samples in total for each experiment which are stored in the database. After the experiment is completed, the main control VM destroys the entire experiment stack in order to prevent contamination between experiments.

6) Model Training: We used the scikit-learn[4] library for implementing our classical machine learning models, and Keras[5], a high-level API that runs on top of TensorFlow[6], for implementing our CNN model. The hyperparamaters for each of the models were determined by conducting random grid search to achieve optimal performance. The data collected was split with 60% of the data being used for training, 20% for validation, and 20% being used for testing. All the ML models were trained on a Windows machine equipped with an AMD Ryzen 5 2600 processor and 16 GB of RAM.

B. Evaluation

We use five standard metrics to measure the performance of different models, accuracy, precision, recall, and F1 as

[4]scikit-learn. https://scikit-learn.org/stable/
[5]Keras. https://keras.io/
[6]Tensorflow. https://www.tensorflow.org/

defined below where TP (True Positive), TN (True Negative), FP (False Positive), and FN (False Negative).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

IV. Results

Figure 4 shows the performance metric scores from each of our models. In regards to overall performance, the CNN model outperformed every other model with an F1 score of 91.5%. We use the F1 score as a basis for overall performance since it takes the precision and recall metrics into consideration and is therefore better at describing a model's overall performance than the accuracy metric. The RFC and SVC models had the next highest F1 scores with 84.15% and 83.47% respectively. The GBC, GNB, and KNN models did not perform as well with F1 scores of 76.57%, 64.61%, and 61.81%.

The recall metric quantifies the percentage of the infected samples that were detected, and the model that scored the highest in this metric was the GNB model with a score of 98.57%. However there is a caveat to this, its precision score, which measures how many of the samples that were labeled as infected were actually infected, was very low with a score of 48.06%. The CNN model achieved a recall score of 84.6% and a precision score of 100%. The next best model in regards to recall was the SVC model with a score of 80.91% and a precision score of 86.2%. The RFC model achieved a recall score of 72.80% but also had a very high precision score of 99.71%. The KNN model performed the worst, which was indicated earlier by the low F1 score, with a recall score of 57.67% and a precision score of 66.6%. The findings when analyzing the accuracy scores are consistent with our other findings, the CNN model outperforms the other models with the RFC and SVC models not too far behind.

Figure 5 shows the ROC curves and AUC scores for each of our models. Once again, the CNN model has the best AUC score of more than 99%. The RFC and SVC models follow closely scoring a 94% and a 93% respectively. The GBC model was able to score and AUC of 90% whereas the KNN model scored a 77% and the GNB model performed the worst with a score of 65%. These results are consistent with the results depicted in Figure 4.

Another important metric to consider when comparing the performance of various models is the **training time** of the model, as shown in Table II. Neural networks require models that use this methodology be trained for a certain number of epochs. If the model is trained for too many epochs *overfitting* could occur, but if the model is not trained for enough epochs, *underfitting* will occur. As such, we stop training if the validation accuracy is not increasing for specific number of epochs and we choose the model with the highest validation accuracy achieved during these epochs. The CNN model took 1683 seconds to train and required a total of 32 epochs. Other models do not train based on a certain number of epochs, therefore, the same technique doesn't apply when training the remainder of the models. In such case, the SVC model took the longest to train by far at 989 seconds. The GBC model took 167 seconds, the KNN model took 28 seconds, the RFC model took 20 seconds, and the GNB model only took 2 second to train. Note that the reported times solely include training time, excluding the time it takes to read and load the data samples.

TABLE II
Time Cost for the Models

Model	Time to Train (s)	Detection Time (ms)
CNN	1683	164
SVC	989	11
RFC	20	3900
KNN	28	118
GBC	167	40
GNB	2	.9

V. Comparison and Analysis

The superior metrics generated by the CNN model clearly indicate that this model is the best suited for our use case. The comparison of all the models can be found in Figure 4. The CNN model was able to detect 84.6% of all of the infected samples while also not falsely labeling benign samples as infected. In malware detection, while detecting every instance of malware is ideal, a high number of false positives can be just as much of a disruption as malware. That is why the high recall rate achieved by the GNB model is not as promising as it may seem. Its extremely low precision score of 48.06% indicates that this model generated a high number of false positives. In fact, 52% of the samples labeled as infected were actually benign. In a business case, having a model that generated these many false positives could hinder day to day activities by labeling essential, benign processes as malware. The SVC and RFC models produced similar accuracy and precision scores but differed in the recall and precision metrics. The SVC model was able to detect 80% of the total infected samples, but also has a lower precision score of 86%. The RFC model had a lower recall score of 72% but generated nearly no false positives with a precision score of 99%. These models will need security professionals to make a decision when implementing a malware detection system. There could be a situation where a company is willing to have a lower recall rate in exchange for nearly no false positives to ensure that the malware detection implementation does not severely hinder necessary activities. On the other hand, there may be a security critical use case where a high number of false negatives is very detrimental. In that case, it may be beneficial with a system

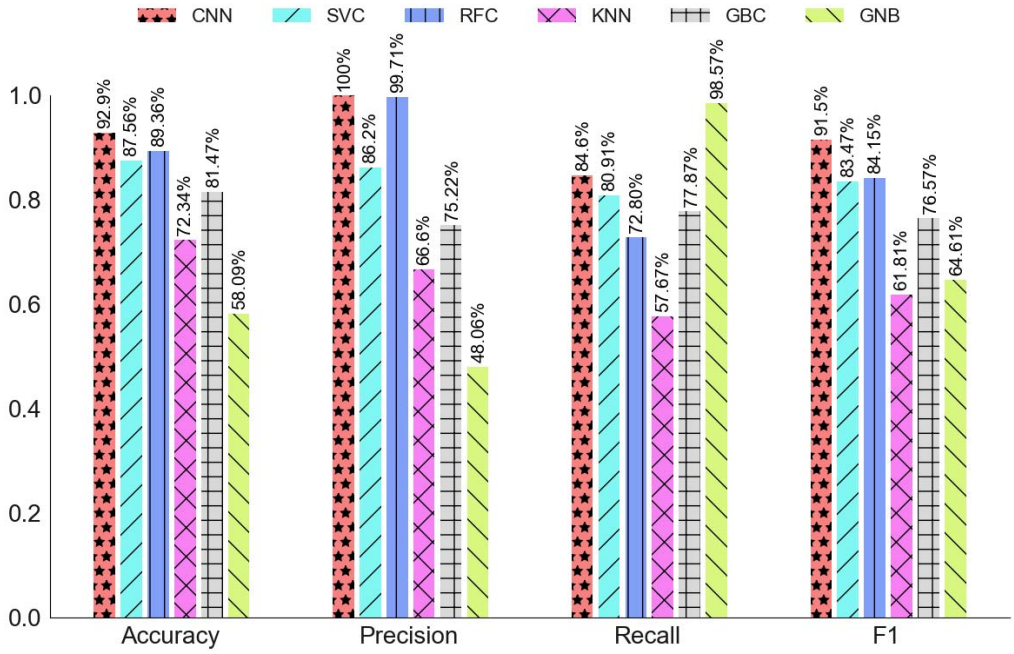

Fig. 4. Performance Metrics Comparison for Different Machine Learning Models

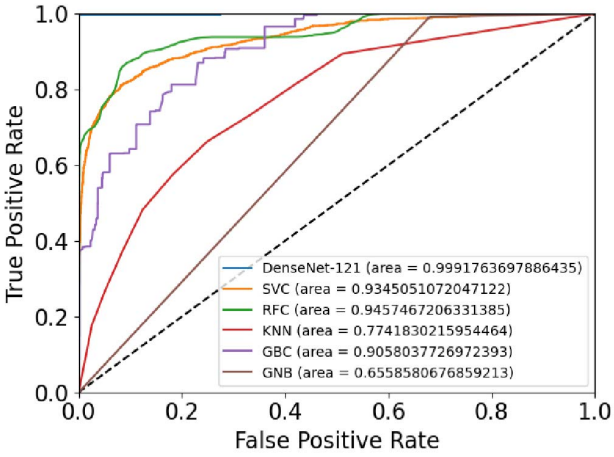

Fig. 5. Receiver Operating Characteristic (ROC) Curves

which is able to detect nearly all malwares even if a large portion of those classifications are false positives.

A. Cost Analysis

The time it takes to train these models, as shown in Table II, can also have an affect on which use case each model will be suitable for. There is a clear pattern of the more successful models requiring more time to train. The CNN model took the longest to train by far but also outperformed every other model by a convincing margin. The RFC model gave promising results with an F1 score of 84% and also only required 20 seconds to train. The RFC model reigns supreme of the SVC model in regards to time since the SVC model required over 900 seconds to train while still yielding similar results to the RFC model. In a general sense, it is usually worth sacrificing some time in order to train models that are as accurate as possible. That is why deploying a deep learning method such as the CNN model is preferred to the faster trained models such as SVC and RFC. Once these models are trained and deployed for online detection, an important aspect is how long will the models take to decide whether a given sample is benign or malicious. Detection time, also found in Table II, shows how long it took each model to classify a single sample. This metric indicates how fast each model will be able to process the input data and correctly classify an infected sample as such. The GNB model has the fastest detection time, however this is due to this model having a high false positive rate indicated by its low precision score. The RFC is the slowest model by far, this could be due to the input data having to parse through the various trees that make up the RFC which would cause this model to take longer to generate a prediction. A faster detection time is of course preferred, however if a model generates fast predictions that are incorrect, the benefit of a faster model is overshadowed by its inability to produce accurate results.

B. Overall Analysis

Despite its longer training time, the CNN model has proven to be the optimal model in our use case. The CNN model was

able to achieve the highest metric scores. The CNN model achieved high, balanced values between all of the metrics indicating that this model is able to correctly classify our samples as benign or infected, more so than the other models. The CNN model's longer train time can be attributed to its deep architecture. This model utilized the state of the art DenseNet-121 model, indicating that it contains 121 different layers within the network.

VI. Limitations

One limitation of our work is due to the relatively small number of malware samples used. We conducted 113 different experiments each with a different type of malware, but conducting more experiments with a wider range of malware could give us a better look into how malware affects the behavior of VMs in a cloud environment. Another limitation lies in the assumption that each VM can only be infected by a single malware, which helps in simplifying our analysis. In practice, a machine can be infected with multiple malware at the same time. That being said, our work aims to provide a fundamental step towards a more rigorous and complex analysis for multiple malware infection. Further work is needed in order to determine if our approach would be feasible given a situation where a VM is infected by multiple malwares.

In addition, the use of the unique processes approach could allow malware behavior to go unnoticed. Since a unique process sample is the average of a collection of processes sharing the same name and command line, a malware that mimics these same attributes will be counted within the average of this sample. This is a common drawback to methodologies that utilize meta-stats (e.g. average, standard deviation, etc.) However, the drawback of using meta-stats is confined to each unique process independently. This makes our approach partially immune to the meta-stats limitation as opposed to other approaches that use meta-stats of the entire system.

VII. Conclusion

In this paper we analyzed a variety of machine learning methods in order to determine which method is best for online malware detection in cloud. We find that, although it takes the longest to train, the DenseNet-121 (CNN) model has the best overall performance. The SVC and RFC models produced promising results that are not too far behind those produced by the CNN model, as well as being much quicker to train. However, when it comes to malware detection, taking the time to train a more accurate model is mostly preferred. The remaining models, KNN, GBC, and GNB, simply could not compete with the others. Due to the CNN model's success, it can be concluded that deep learning models are more adept at detecting malware within our dataset in cloud IaaS.

Acknowledgement

This research is partially supported by NSF Grants 2025682 at Tennessee Technological University and 2025686 at Manhattan College.

References

[1] M. R. Watson, A. K. Marnerides *et al.*, "Malware detection in cloud computing infrastructures," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 2, pp. 192–205, 2015.

[2] M. Alazab, S. Venkatraman *et al.*, "Zero-day malware detection based on supervised learning algorithms of api call signatures," 2010.

[3] A. Shalaginov, S. Banin, A. Dehghantanha, and K. Franke, "Machine learning aided static malware analysis: A survey and tutorial," in *Cyber Threat Intelligence*. Springer, 2018, pp. 7–45.

[4] S. A. Roseline, S. Geetha, S. Kadry, and Y. Nam, "Intelligent vision-based malware detection and classification using deep random forest paradigm," *IEEE Access*, vol. 8, pp. 206 303–206 324, 2020.

[5] M. Abdelsalam, R. Krishnan, and R. Sandhu, "Online malware detection in cloud auto-scaling systems using shallow convolutional neural networks," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2019, pp. 381–397.

[6] ——, "Clustering-based IaaS cloud monitoring," in *IEEE International Conference on Cloud Computing (CLOUD)*, 2017, pp. 672–679.

[7] M. Abdelsalam, R. Krishnan *et al.*, "Malware detection in cloud infrastructures using convolutional neural networks," in *IEEE International Conference on Cloud Computing (CLOUD)*, 2018, pp. 162–169.

[8] A. McDole *et al.*, "Analyzing CNN based behavioural malware detection techniques on cloud IaaS," in *International Conference on Cloud Computing (CLOUD)*. Springer, 2020, pp. 64–79.

[9] S. Tobiyama *et al.*, "Malware detection with deep neural network using process behavior," in *IEEE Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, 2016, pp. 577–582.

[10] J. A. Dawson *et al.*, "Phase space detection of virtual machine cyber events through hypervisor-level system call analysis," in *IEEE International Conference on Data Intelligence and Security*, 2018, pp. 159–167.

[11] P. Luckett *et al.*, "Neural network analysis of system call timing for rootkit detection," in *IEEE Cybersecurity Symposium*, 2016.

[12] A. McDole *et al.*, "Deep Learning Techniques for Behavioral Malware Analysis in Cloud IaaS," in *Malware Analysis using Artificial Intelligence and Deep Learning*. Springer, pp. 269–285.

[13] J. C. Kimmel *et al.*, "Recurrent neural networks based online behavioural malware detection techniques for cloud infrastructure," *IEEE Access*, vol. 9, pp. 68 066–68 080, 2021.

[14] T. Garfinkel *et al.*, "A virtual machine introspection based architecture for intrusion detection." in *NDSS*, vol. 3, no. 2003, pp. 191–206.

[15] I. Firdausi *et al.*, "Analysis of machine learning techniques used in behavior-based malware detection," in *IEEE Int. conference on advances in computing, control, and telecommunication technologies*, 2010.

[16] F. Azmandian *et al.*, "Virtual machine monitor-based lightweight intrusion detection," *ACM SIGOPS Operating Systems Review*, vol. 45, 2011.

[17] Q. Guan, Z. Zhang, and S. Fu, "Ensemble of bayesian predictors and decision trees for proactive failure management in cloud computing systems," *Journal of Communications*, vol. 7, no. 1, pp. 52–61, 2012.

[18] H. S. Pannu, J. Liu, and S. Fu, "Aad: Adaptive anomaly detection system for cloud computing infrastructures," in *IEEE Symposium on Reliable Distributed Systems*, 2012, pp. 396–397.

[19] J. Demme *et al.*, "On the feasibility of online malware detection with performance counters," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 559–570, 2013.

[20] R. S. Pirscoveanu *et al.*, "Analysis of malware behavior: Type classification using machine learning," in *IEEE Int. conference on cyber situational awareness, data analytics and assessment*, 2015, pp. 1–7.

[21] Y. Fan, Y. Ye, and L. Chen, "Malicious sequential pattern mining for automatic malware detection," *Expert Systems with Applications*, vol. 52, pp. 16–25, 2016.

[22] Z. Xu *et al.*, "Malware detection using machine learning based analysis of virtual memory access patterns," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 169–174.

[23] S. Joshi *et al.*, "Machine learning approach for malware detection using random forest classifier on process list data structure," in *Proc. of the Int. Conference on Information System and Data Mining*, 2018, pp. 98–102.

[24] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[25] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[26] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.