A Parallel and Scalable Framework for Insider Threat Detection

Abdoulaye Diop, Nahid Emad, Thierry Winter

A Parallel and Scalable Framework for Insider Threat Detection

Abdoulaye Diop, Nahid Emad, Thierry Winter

***Abstract*—In this article, we propose an innovative method for the detection of insider threats. This method is based on a unite and conquer approach used to combine ensemble learning techniques, which have the particularity of being intrinsically parallel. Furthermore, it showcases multi-level parallelism properties, offers fault tolerance, and is suitable for heterogeneous architectures. To highlight our approach's efficacy, we present a use case of insider threat detection on a parallel platform. This experiment's results showed the benefits of this method relative to its improvement of classification AUC-score and its scalability.**

***Keywords*—Bagging, Boosting, High performance computing, Unite and conquer, Insider threat, User behavior modeling**

I. Introduction

In the information and technology domain, informational data assets, and intellectual properties are a source of profit. Companies understand the strategic importance of protecting their information systems. They use tools to shield themselves against all kinds of cyberattacks. However, cyber attackers frequently target these assets and are continually upgrading and widening their attack options. Usually, the main focus of security specialists was the defense against external threats. They used tools based on databases of known threats patterns and experts' written-rules against cyber attacks. They monitored and controlled access to their companies' assets using identity access management (IAM) software. They also used security event information management (SIEM) and intrusion detection systems (IDS) to protect their systems and networks. However, none of these solutions can stop an insider attack efficiently. In a company environment, insiders are mostly employees who misuse their access rights, or hackers that exploit flaws of the authentication systems. A typical example of insider activities is the illegal share of confidential business information to competitor organizations in exchange for compensation (i.e. industrial espionage). Another common example is the sabotage of proprietary tools by insiders hired by a rival company. In 2018 studies from the Ponemon Institute evaluated the loss related due to insider attacks for some relatively big companies in the millions of dollars range [13].

In the cybersecurity domain, user and entity behavior analysis software are the tools used to stop insiders. Companies use these tools to determine if employee behavior is normal or abnormal. Employee behavior is hard to classify because it can change over time, its nature can diverge depending on their work requirements and their company structures. These tools mainly use behavior anomaly detection techniques to detect insiders. Most of the techniques used to identify malicious activities are based on supervised machine learning methods. Other approaches use semi-supervised, unsupervised machine learning, and graph analysis. Their goal is to detect a divergence from an employee behavior profile or to find outliers in the all-around company activity data. The proposed solutions are diverse, but they usually face the same problem of high false-positive/negative (FP/FN), and a lack of versatility. Depending on the company studied, a method initially performing well to detect an attack scenario can present an unstable detection accuracy. It makes sense to optimize a specific model for a particular company, but the cost of the software development and maintenance of these tools can represent a drawback. The data volume can also pose a challenge to implement a detection model. Machines limited in their computation power struggle to treat the massive amount of behavior data. A solution to this issue would be to build a model somehow adaptive, able to manage different and extensive data input. This model would have to consider detection accuracy, detection time, and maintainability constraints.

In this article, we propose an approach to counter this issue based on the use of unite and conquer and the ensemble learning principles. Our solution uses the collaboration of multiple machine learning methods to build individual post-login activity profiles. These profiles are used to identify a new activity record as regular or abnormal. The base methods of the ensemble of learners belong to the family of the unsupervised, supervised machine learning, and graph-based methods. We show that this approach makes it possible to obtain significant gains in accuracy relative to the base methods, which constitute the global method proposed. To exploit the potential parallelism of the proposed solution, we implement it with high-performance parallel computing techniques and show its efficiency also in terms of execution speed.

The rest of this paper is organized as follows. Section 2 presents some related works and defines our approach and issued detection models. Section 3 describes the algorithm and parallel programming models according to which we implemented the proposed approach. Section 4 presents the conditions of our experimentation. Section 5 showcases the results of our experiments and their analysis. Finally, Section 6 concludes this article and gives indications about some perspective of this work.

This project is funded by Atos, and the University of Versailles Paris Saclay. Abdoulaye Diop is with the Li-PaRAD and Maison de la Simulation laboratories, and Atos Evidian R&D, Les Clayes-sous-Bois 78340, France (e-mail:mamadou-abdoulaye.diop@atos.net). Nahid Emad is a member of the Li-PaRAD and Maison de la simulation laboratories, Versailles 78310 (e-mail: nahid.emad@uvsq.fr). Thierry Winter is with Atos Evidian R&D, Les Clayes sous Bois 78340, France (e-mail: thierry.winter@atos.net).

II. UCEL ALGORITHM FOR INSIDER THREAT DETECTION

A. Related work

In this section, we first present works done to combine bagging and boosting and then some example of solution for insider threat mitigation. Machine learning experts mainly combine bagging and boosting to deal with the bias-variance tradeoff. Kotsiantis et al. propose in [11] a bagging and boosting combination with sum rules voting as the mechanism collaboration. They build a global ensemble learning method with separated sub-ensemble, respectively, using bagging or boosting. They tested their solution on 36 well-known datasets from the UCI repository of machine learning and used decision tree (DT) C4.5, decision stump (DS), naive bayes network (NBN), and a rule learner (OneR) as base classifiers. With this study, they obtain better results using their approach compared to individual bagging and boosting with these base classifiers. Even though this approach helps to fix bias and variance issues individually, this proposition doesn't manage the tradeoff between them.

In the power plant management domain, [10] proposed a parallel combination of bagging and boosting for a regression problem. They used artificial neural networks (ANNs) as base classifiers of their ensemble learning solution for short term electricity load forecasting. They independently boosted ANNs on different bags of the training dataset. Here the sub-ensembles are all boosted ANNs. The final result of their forecasting is computed by averaging the result of each base method. They compared their effect against ANN, only bagged ANN, and only boosted ANN. They had the best results with their combination of bagging and boosting. In this approach, there is no collaboration between base methods. This could represent a missed opportunity to accelerate their convergence, hence faster training times.

Fauvel et al. proposed in [4] a hybrid ensemble method called local cascade ensemble (LCE) by combining bagging, boosting and mixture of expert (ME) [12] methods, in a decision tree scheme. They use their approach for a milk production industry to detect the cow estrus cycle. The ME method is a method based on a divide and conquer strategy. This method divides the problem space between classifiers, supervised by a gating network (i.e. a weighted average scheme). Each classifier is trained with a different part of the dataset. They use this combination of bagging and boosting to handle the bias-variance tradeoff, and use the diversification properties of ME to learn the specificities of parts of the dataset. Their approach showed better results when compared with other classifiers and commercial solutions for estrus detection. This approach is interesting; however, it is based on a divide and conquer strategy, which is not suitable for our insider threat detection problem.

These are a couple of examples of the use of the combination of bagging and boosting. To our knowledge, in the domain of insider threat detection, there is no work proposing to use a bagging and boosting combination to solve this. However, Hall et al. [8] studied the effect of boosting on classifiers trained to detect insiders' attack. They create a meta-learner by aggregating boosted classifiers using a probability vote. They tested their method by comparing the base classifiers firstly with their boosted version. They boosted artificial neural network (ANN), naive bayes network (NBN), support vector classifier (SVC), decision tree (DT), and logistic regression methods. They obtained mixed results by comparing the accuracy and the AUC-score of the base classifiers before and after boosting. This process improved NBN, SVC, DT, and LR slightly, not ANN and RF, which showed a slight decrease in accuracy and AUC-score. Using their meta-learner, they didn't get better efficiency. However, they have a better area under the ROC curve. These mix results might be a consequence of applying boosting to already high performing classifier, or not handling the high variance issues by not using the bagging method.

Despite the fairly recent consideration of the insider threat problem, significant literature exists on machine learning techniques and related ensemble learning methods for its handling. The latter mainly use methods based on unsupervised, supervised machine learning, and graph feature analysis. In the overall literature, supervised learning approaches showcase better performance than anomaly detection approaches. Particularly approaches based on neural networks. Yuan et al. [16] presented a deep learning approach to detect insiders. They use a combination of long short term memory (LSTM), and a convolutional neural network (CNN) to identify the abnormal behavior in multiple scenarios. They obtained an area under the curve (AUC) of 0.94 for their classifier. However, to work properly, they need substantial and balanced data. Since the activity dataset naturally contains more normal activity samples than insiders, this method can struggle to perform well and be susceptible to bias and variance issues.

Due to the nature of insider attack, using unsupervised learning and anomaly detection seems to be a natural decision to handle this type of issue. Employee behavior can be considered as the dominant normal activity class and the insider action as an anomaly. Tuor et al. [15] proposed unsupervised online approaches based on a deep neural network (DNN) and a recurrent neural network (RNN) (i.e. using an LSTM architecture) as a prospective filter for a human data analyst. Their best model obtained an anomaly score of in 95.53 percentile for the insider activity. Their goal was to diminish the workload of a security analyst.

Haidar et al. [7] gave us insight on how to boost two anomaly detection methods, (i.e. one-class support vector machine (OcSVM), isolation forest (IForest)) to handle the behavioral exception, but not the evolution over time. Using a periodic training scheme combined with a smart windowing of the training data can solve this problem. However, there is another problem than anomaly detection based solutions can face. They cannot characterize the source of the problem, which means that they can spot unusual or un-popular data samples, but cannot give insight into their causes.

Graph-based methods are another approach to tackle the problem of insiders. Based on the graph features analysis, the detection mechanism is to spot anomalous nodes or subgraphs. They share the same issue as anomaly detection approaches. They are also sensitive to exceptions, and they do not give

information on the nature of the anomaly. Chen et al. [1] used bipartite graphs to map user access logs. They then use the cosine similarity method to get the similarity between users and to detect if specific access is malicious or not. They based their approach on a correlation method used in collaborative filtering for recommendation systems.

B. Detailed contribution

In our work, we propose a new and customized approach to combine bagging and boosting. This is inspired by the previous work in the insider threat detection domain and the mitigation of high bias and high variance problem. We opt for the same strategy as the ME method. However, we diversify the distribution of the training dataset using bagging and boosting sampling techniques to handle the bias-variance tradeoff. So contrary to a divide and conquer strategy like ME, we propose a restarting strategy based on the unite and conquer approach, mixing individual classifier classification feedback to improve the training set. Hence in this article, we propose:

- A new iterative boosting and bagging combination relying on a restarting strategy inspired by the unite and conquer method, particularly well suited for insider detection problems.
- A fault-tolerant implementation strategy, to maximize the contribution of the best base methods or their combination.
- An implementation scheme combining anomaly detection and supervised learning methods depending on the available data.
- A custom scalable parallel implementation model that can deal with high data load, and that take advantage of the high-performance machine architectures.

C. Motivation

Insiders threat are reported in different types of companies' work structure (e.g. companies in information and technologies, finance, healthcare). This heterogeneous nature of employees' activity could appear even inside of a given organization. The insider attack scenarios are also diverse since they can target different types of assets of a company. As a consequence of this heterogeneity, a single detection method might not work to detect insiders in all cases [8].

Using an ensemble learning strategy can represent a solution to this issue. Most of the ensemble method uses a combination of classifiers to obtain a more accurate final prediction. A combination mechanism can help to detect different scenarios of an insider attack. For instance, we can compare the detection performance of the base methods composing the ensemble of learners. For the final prediction, one option is to select the best base method regarding a prediction metric. Another option can be to use a voting scheme to make all the base methods contribute to the class prediction.

In this article, we propose a way to detect insiders in multiple settings. Our approach combines bagging and boosting techniques by using a unite and conquer strategy. This approach consists of making collaborate several boosting methods (i.e. called co-methods) in a bagging context to solve the problem of insider threat detection. These two ensemble learning techniques are specifically used to handle the high bias (i.e. underfitting) and high variance (i.e. overfitting) problems. A classifier suffers from high bias when it is unable to fit the structure of the training set. On the other end, a classifier suffers from high variance when it is too specialized on the training set. In bagging, several subsets of the training dataset are stored in bags. A given learning method is applied in parallel to all these bags. In the case of classification, the final result will be chosen by a hard voting scheme. In boosting, a set of different weak learners is used sequentially to define a strong learner. Each learner is trained on data, taking into account the previous learner's success. After each training cycle, the data sample weights are redefined by increasing that of miss-predicted data. When all learners are trained and tested, a weighted voting mechanism is used to establish the final prediction.

Bagging solves the problem of overfitting by diversifying the training set distribution stochastically and share it with multiple classifiers. Boosting can handle underfitting by specializing a classifier list to a training set, capturing its underlying structure. However, this operation can result in over-specialization on the training set, increasing the overfitting risk. Hence a combination of the two methods can help to manage the bias-variance tradeoff. Balancing the bias and the variance is the ability for a classifier to generalize beyond its training set. This is an essential advantage for insider threat detection since this behavior analysis system is bound to analyze new activities records continuously. The intrinsic parallelism in bagging is an advantage of this method, mainly when most of today's applications deal with vast data quantities and need their processing and analysis on parallel and distributed architectures.

We call the proposed framework UCEL (i.e. for unite and conquer ensemble learning). Given several boosting methods able to learn employees' post-login behavior individually. The UCEL framework uses a combination in an extended manner of bagging and boosting methods to create profiles. Such a profile is built by learning the usual employee conduct and work patterns, from activity data recorded on their companies' information systems. These individual profiles can be used to analyze and classify the employee's recent or on-going activities as a normal activity, unusual, or insider activities. We name this particular instance of UCEL, a behavior profiler model (see *Fig1*). We give more detail about this framework in the *Section 3.F*.

D. Unite and conquer based method

The proposed model is based on a combination of bagging and boosting using a unite and conquer approach. Unite and conquer is an approach initially used in linear algebra to solve large-size sparse linear systems and/or eigenvalue problems [2]. This approach consists of making collaborate several iterative methods (i.e. also called co-methods) to solve the same problem. This process accelerates the system's overall convergence by making use of intermediate results issued from an iteration of each co-method by all the others. This can be seen as a set of collaborative co-methods that share

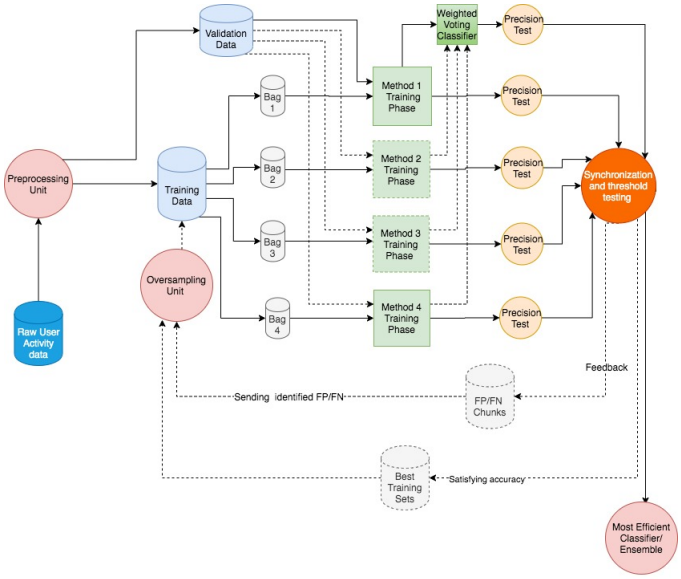


Fig. 1 Behavior profiler

their restarting cycle parameters to choose the best of them and to reach convergence more quickly. Precisely, the aim of this sharing is to combine the intermediate results in order to define the best restarting information for each cycle of each of the co-methods allowing the global method to reach convergence faster. Let P be a large numerical problem to solve (linear system, eigenvalue problem, etc), $L_1, L_2, \ldots, L_\ell$ be a set of iterative methods allowing to solve P, I_i^k be the initial condition (with $k = 0$) and restarting condition (with $k > 0$) of L_i (for $i = 1, \cdots, \ell$) and S_i^k be the approximated solution obtained by L_i at the end of its k-th iteration/cycle with I_i^k initial condition. The main steps of this approach to solve P are presented in algorithm 1. Let P_m represent the problem matrix, and S_f the final solution of the problem.

Algorithm 1 Unite and Conquer Algorithm (in: P_m; out: S_f)

1: **Start.** Choose a starting matrix $[I_1^0, \cdots, I_\ell^0]$, let $k = 0$.
2: **Iterate.** For $i = 1, \cdots, \ell$ do in parallel
3: Compute S_i^k of P_m by L_i with initial condition I_i^k
4: if (accuracy of S_i^k is good enough) then STOP
5: ***Share*** S_i^k information with all other processes j ($j = 1, \cdots, \ell$ and $j \neq i$).
6: **Restart.** Update initial condition $[I_1^{k+1}, \cdots, I_\ell^{k+1}]$ for restarting by $f(S_1^k, \cdots, S_\ell^k)$ and go to 2.

The restarting strategy of UC algorithm is its heart and corresponds to the updating of restarting condition I_i^k in the function of the obtained results $S_1^k, \cdots, S_\ell^k$ (step 6 of algorithm 1). When the ℓ processes of step 2 of algorithm 1 are running in an asynchronous way, step 6 will be part of "Iterate" (step 2) in the same way as "Share" step. In this case, the restarting condition of each process is a function of the most recent available results of all of ℓ processes. The success of the approach depends strongly on the quality of the restarting information at the beginning of a cycle of each process, which is a combination of the results obtained by all processes in their previous cycle. In other words, the definition of the function f, which describes this combination, in step 6, is of utmost importance for the rapid convergence of the UC algorithm. When the co-methods are the instances of the same iterative method, the corresponding unite and conquer method is also called *multiple X*, where *X* is the name of the co-method. Recall that an instance of an iterative method represents the method with a given set of parameters (i.e. inputs). A brief description of the multiple implicitly/explicitly restarted methods is presented in [2]. As shown in the papers [3] and [9] in the field of high-performance numerical computing, the asynchronicity of communications between co-methods (processes of algorithm 1, the possibility of overlapping these communications by co-method computations, the fault tolerance, and the potential load balancing are among the properties of these methods making them well adapted to parallel and/or distributed architectures such as GRID, supercomputers, or heterogeneous interconnected network of (super)computers. Finally, the most important feature of this approach is its simplicity of the concept. It can be applied to many iterative methods, as we will see an example in this paper with the collaboration between several boosting methods in a particular context of the bagging ones.

E. UCEL overview

In summary, UCEL is a multiple training cycle method that improves each co-methods classification performance during its iterations. This process is repeated until they reach a convergence state where their performances are enhanced and stabilized. Each co-method is trained with a bag of the training set. Then their performances are tested with the validation set. Individually the base methods weigh and share their miss classified data FP/FN sample to the rest of the co-methods. Following the boosting principle, each time a sample from the validation set is miss-classified by a co-method, its weight is increased based on the co-method performance metric at that iteration. Its chance to be selected to build the next cycles training data is increased. Bags of the original training data size are then created from the boosted training dataset and used for the new cycle. With this process, each co-method receives feedback on the data samples critical to classify by their peers correctly, to build better models.

We can compare our UCEL approach to classic divide and conquer based methods such as ME. It is an application of a divide and conquer strategy followed with a weighted voting scheme(i.e. probabilistic gating functions [12]) to classify new data. This means the problem space is divided into homogeneous parts, and there is no iterative improvement of expert (i.e. base methods). Unite and conquer learns the underlying global structure of data, where divide and conquer approaches learn the underlying substructure since the data is strictly divided. Depending on the machine learning problem, these different characteristics might be an advantage. However, for our insider threat detection problem, where we want to use insights from each co-method on statistically the same employee dataset at each iteration, it is not suitable to strictly divide the data. We can insist on the fact that the UCEL approach

is an improvement of ME. It switches the classic expert's consensus methods to predict a sample class by introducing collaboration between methods. This collaboration mechanism is based on the statistical data diversification of bagging and boosting, multiple training cycles, and the convergence's acceleration of each co-methods. The combination of bagging and boosting manage the bias and variance tradeoff since the training bags are built using a fusion of all the co-methods feedback information(i.e. performance measure and FP/FN). ME results would be almost equivalent to the first cycle results of our UCEL approach if we didn't apply bagging and strictly divided the dataset. Lastly, it is important to note that the error is individual to each base methods with divide and conquer approaches. With UCEL, each classifier is somehow responsible for the global error. Hence to decrease the global error, every co-methods need to be tweaked.

F. User behavior modeling with UCEL

In order to solve the problem of insider attacks, machine learning methods are used to analyze employee activity data. This data is used to model employee behavior by learning from their post-connection data (e.g. after logging on to a laptop or company server). Depending on the type of data available (e.g. labeled, unlabeled, or graph-based), we can use a variety of suitable methods. They vary from supervised classification methods, unsupervised classification methods (e.g. distance-based, density-based or hierarchical clustering), anomaly detection methods, or methods based on graph analysis. In a corporate environment, the vast majority of employees are not insiders; there is a natural class imbalance in their activity data. This means that there are many more samples of good activities recorded than there are of malicious activities. For example, if we focus on a single employee, insider action can be seen as a change in his/her usual work practices. These elements confirm that if we register for an extended period, the post-logging activities of companies' employees, we will most likely have an imbalanced dataset.

With this noticeable class imbalance, anomaly detection methods seem to be the most suitable for identifying insider activity. Indeed, due to the lack of abnormal examples, unsupervised and supervised classification methods might be inefficient for this kind of datasets. They are not disposing of enough information to learn the characteristics of abnormal examples. Hence It is hard to establish a decision boundary to distinguish normal and abnormal activities. Unsupervised anomaly detection methods are tuned to work with imbalanced data. These latter mostly set the majority class as the normal behavior model. Everything distant to the normal activity sample distribution (e.g. regarding the mean and the standard deviation) is then considered abnormal. However, anomaly detection approaches usually suffer from high FP/FN rates and only focus on detecting what is unusual in the dataset. Consequently, they don't precisely characterize the cause of the anomaly. Human action is needed to determine the causes of abnormalities. On the other end, supervised methods can be trained to detect anomalies related to the specific types of insider threat attacks, but as we stated before, they need a balanced and labeled dataset to perform well. Moreover, the same supervised classifier is not necessarily efficient in spotting different types of insider threat attacks. Hence, human action is required to label the data as standard and to choose and tune supervised classifiers.

Since the flow of activity data is continuous, labeling activity and balancing data before testing for insider threat might be more risky and costly for companies than the operation of characterizing anomalies when they are found. Hence, we propose to opt at first for semi-supervised anomaly detection based UCEL method. We can then combine it with a supervised learning-based UCEL method when we have enough labeled and balanced data (i.e. using human operators to label the data or oversampling strategies [14] to deal with the imbalances).

In a semi-supervised context [6], anomaly detection methods add samples with known labels to their data distribution to have extra information to their classification process. This action improves the decision boundary of classic unsupervised anomaly detection methods [7]. A behavior profiler with anomaly detection as co-methods can be alimented with a continuous feed of FP/FN samples, label by another security system, or human action. This profiler would work without a balanced dataset.

Before analyzing employee behavior by the proposed profilers, it is necessary to perform a data pre-processing step. In this step, we start by selecting samples from the raw activity data of a single employee and samples from an insider threat attack scenarios databases. We then perform classic feature engineering, with dataset cleaning, feature selection, scaling, and normalization. We finish by building a training, validation, and testing set from the cleaned data.

The second step is to apply the main principle of unite and conquer to machine learning. The idea is to design our classification method using the same architecture of the unite and conquer methods for restarted iterative methods. We establish correspondence points between the two techniques. The system matrix corresponds to the original training dataset. The subspace becomes a bag of data built with random sampling with replacement (RSR) and weighted random sampling (WRS). For instance, in the anomaly detection case, the co-methods or base methods can be mainstream methods such as: (IForest), (OcSVM), robust covariance (Robcov) and local outliers factor (LOF). We can also choose co-methods in the supervised learning case, such as multilayer perceptron (MLP), gaussian naive bayes (GNB), KNN, SVC, and others. We can also build a solution using the same method but with different hyperparameters (i.e. changing the initial condition). In other words, we can use the instances of the same base method, which is equivalent to create the particular case of the UCEL methods called multiple base method such as multiple IForest or multiple Robcov, etc. However, the advantage of using co-methods differently is that it helps to build a heterogeneous consensus on the nature hypothesis of an analyzed behavior.

If we focus on some base methods mechanisms, OcSVM is a variant of support vector machines classifier. SVM is a large-margin classifier that establishes a planar decision boundary between positive and negative examples. OcSVM

is one way to apply SVM for outlier detection. The decision boundary OcSVM opts for a spherical approach instead of planar approaches, as the support vector data description methods (SVDD) [6]. The goal is to find in a high dimension space, which is the minimal circumcising hypersphere that comprises only the good observations. IForest is a variant of decision trees and random forest algorithm. It uses a comparison of the depth of the tree branches to spot anomalies. The shorter branches are indicative of anomalies. The robust covariance/elliptic envelope method uses the assumption that the normal data belongs to a known Gaussian distribution. The outliers are spotted when they are too distant from the center of the distribution. The local outlier factor method studies the neighborhood of a data sample. It measures the local density of a given sample with respect to his neighbors. Outliers samples are detected when they present less density than their neighbors. The multilayer perceptron (MLP) is a feedforward ANN used in our case for classification. It is very efficient to fit a non-linear function to a dataset. K-nearest neighbors is a sample classification methods that output class membership of a sample, based on a popularity vote of the neighbor's samples. Quadratic discriminant analysis is a classifier using a quadratic decision boundary to separate the classes. It is important to note that none of the described classification methods individually can be considered as well adapted to all the situations. Each has unique advantages and disadvantages and can match the specificities of the companies activities. We proposed their combination using the UCEL framework to enjoy the benefits of these methods without their drawbacks. Hence the co-methods methods are chosen to be classic anomalies detection methods or supervised classification.

Individually the detection methods all operate differently to detect anomalies. However, they are mostly facing the same problem of high FP/FN rates, high bias, and high variance. We start the first iteration by training the co-methods with bags generated with an RSR on the initial training data. We then test the co-methods with the validation set by computing the AUC-score of each co-methods. After that step, we then try to combine the strength of each co-method and build a weighted voting classifier with their results. We then test its AUC-score against the score of the co-methods and the chosen detection threshold.

The third phase corresponds to the restarting step. We apply the principle of boosting on the previous training bag if the detection threshold is not reached. We start by gathering all the FP/FN of the co-methods and weigh them in the function of their popularity. The training data of the next iteration of a co-method is obtained by combining its most accurate training bag during the previous iterations, with the most popular FP/FN. Here, we use a RSR and a WRS again to build the new training bags. All the best bag correspondent to the co-method and WVC are stored in order to use them in the following iterations. They are updated when a new bag presents a better prediction score.

We repeat the same process until the detection threshold, or the desired number of iteration is reached. The restarting strategy is a critical part of our model. It launches a new cycle of classifier training and testing. In theory, the new period has better starting conditions than the previous ones. For now, we used a simple restarting strategy. Still, the proposition of the ones with more effectiveness and sophistication will be the subject of our future research works (i.e. we can consider sending in addition to the best bag or the best hyperparameters in a multi-UCEL case).

Depending on input data, even with the boosting process, some co-methods might not be efficient do detect the insiders. Hence, they are not contributing to accelerate the convergence through the iteration. In those cases, UCEL still provides good results because the focus is always shifted to the best method and the best bags. This highlights the fault-tolerance capabilities of this approach. The bags are stochastically selected. So in the same conditions, the performance might also drop if the selection is unlucky at that cycle. However, since we choose the best bag since the first iteration, combined with the situation where the classifier is mistaken, the precision is susceptible to oscillate and rise again. The stop condition for classic unite and conquer consists in reaching the chosen tolerance. The stop condition of our system is to have a base classifier AUC-score reaching the chosen threshold. At that point, the best-trained co-method or the best-weighted voting classifier is chosen to be the behavior profile.

III. Parallel programming model for UCEL

One important aspect of the UCEL framework is its intrinsic multi-level parallelism. That means we can exploit coarse-grain inter co-methods as well as fine-grain intra co-method parallelism. Moreover, communications between co-methods can be synchronous or asynchronous. In this article, we focus on a synchronous implementation. Despite a loss of time due to the synchronization, the advantage of this model is the simplicity of its implementation and the existence of a certain determinism in the calculations. Another important aspect of UCEL framework is its heterogeneity. Indeed, the processes corresponding to base classifiers could all be different. Thereby, it is possible to use an adapted hardware processor for each of them according to their natural parallelism. For instance, it would be more optimized to use a multicore node for an IForest method, and a GPU architecture for a neural network whose algorithm can present a high degree of data parallelism. In addition, the natural parallelism of the co-methods allows load balancing of the system by assigning each of them to a suitable hardware architecture. That is also to note that an algorithm which represents this framework is fault-tolerant. Indeed, the disappearance of a co-method, by any fault, does not prevent the other co-methods from working and making the algorithm work up to its end. We assume that the targeted parallel architecture has a set of nodes; each of them could be itself a parallel processor. For the parallel implementation of UCEL, we consider a programming model where the nodes of underlying architecture act as a computing server (SN) or controller (CN).

Under these hypotheses, each SN trains a classifier allowing predicting classes, selecting miss-classified data, and computing its AUC-score. Then, all of SN send the set of their results to the CN, which is in charge of determining

which is the best set among them before sending this set to all the SN. This computation task of the CN constitutes a synchronization step and can be seen as a critical section. Consequently, its execution time will have a significant impact on the execution speed of the whole algorithm. According to the best information received, SN update starting conditions for a new cycle of their corresponding boosting classifier. In each cycle, if the detection accuracy threshold is reached by one of the co-methods or by the voting classifier, all the processes stop. Otherwise, a new cycle is launched where the best new training bag is created and transferred to SN for a new cycle.

Let ℓ be the number of learners and bags of size m, $L^0 = [L_1^0, \cdots, L_\ell^0]$ be a set of initial learners, W_i^j be the set of miss-classified data issued from the jth cycle of the ith classifier and, B_{best}^j be the training set with the best AUC-score among $B_1^j, \cdots, B_\ell^j$. This bag is associated with L_{best} the most accurate learner and W_{best} the lightest false positive-negative set (with the smallest cardinal). A classifier is considered as sufficiently trained if its AUC-score is larger than a precision threshold θ. The algorithm 2, called *behavior profiler*, depicts a parallel implementation of the UCEL framework according to the above programming model.

Algorithm 2 Parallel behavior profiler
BP (in: T_S V_S, ℓ, q, θ; out: B_{best}, L_{best})

1: **Start.** Choose ℓ, m, L^0 the ℓ learners, ...
2: **Iterate.** For $i = 1, \cdots, \ell$ do in parallel
3: **Iterate.** For $j = 1, \cdots, q$
4: ***Training and testing on SN***$_i$
Train L_i^{j-1} on B_i^j, produce L_i^j, test L_i^j on V_S and select W_i^j
5: ***Communication: send from SN***$_i$ ***to CN***
Send (B_i^j, L_i^j, W_i^j, AUC-score(L_i^j)) from CN$_i$ to SN
6: ***Computation and stopping test on CN***
$WVC_j = V(L_i^j, AUC(L_i^j))$
$B_{best}^j, L_{best}^j, W_{best}^j = f(L_i^j, B_i^j, W_i^j, WVC_j)$
If (AUC-score(L_{best}^j) $> \theta$) then STOP all processes
7: ***Communication: send from CN to SN***$_i$
Send $(B_{best}^j, L_{best}^j, W_{best}^j)$ to all node i for $i \in [1, \ell]$.
8: ***Sampling on SN***$_i$
Set the bag $B_i^{j+1} = (1-\alpha) * W_{best}^j \cup (\alpha) * R_i^j$ where R_i^j is the set of $(m_i - k_i^j)$ correctly predicted data in B_{best}^j with $k_i^j = \mathbf{card}(W_{best}^j)$ and α is the updated weight given to miss-predicted data.
9: ***Result.***
Set L_{best} the best individual co-method or best weighted combination of co-methods during the iterations of all ℓ processes

V is a function that creates a weighted voting classifier and f a function that selects $(B_{best}^j, L_{best}^j, W_{best}^j)$ as the best results of each co-method received from all $i \in [1, \ell]$ processes.

IV. Experiments conditions

We implemented the behavior profiler *(BP)* with anomaly detection and supervised methods as co-methods. In order to evaluate the performances of this *BP*, we make use of two main performance metrics. The first one is the AUC-score allowing measurement of prediction performance and, consequently, (in)validating the approach. Indeed, from this information, we can see if *BP* algorithm improves the classical insider detection methods in terms of the FP/FN rate reduction and handling of the bias and variance tradeoff. The AUC corresponds to the area under the curve of a ROC(receiver operating characteristic curve). The ROC represents the ratio of the true positive rate (i.e. equal of the sensitivity, also known as recall) and the false positive rate (i.e. equal to $1 - specificity$, also known as the true negative rate). A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0. AUC is a desirable metrics because it is scale-invariant and classification-threshold-invariant. This metric allows us to modify the classification threshold for a more restrictive insider detection, without having an inaccurate measure. Also, it performs well with a highly imbalanced dataset. The second metric is speedup giving the gain of time due to the exploitation of parallelism in the *BP* algorithm. It highlights the gain in terms of the framework's execution time when it is run on a parallel architecture.

The dataset used for the experiments presented is the open-source version *CERT Insider threat R4.2*. The Computer Emergency Response Team (CERT) dataset is an artificial insider threat dataset created by the CERT National Center for Internal Threats (NITC) division. It is a set of data composed of employees' normal post-connection activity in a synthetic context and insider attack scenario perpetrated by synthetic malicious actors. These scenarios are abnormal and suspicious activities that can be dangerous for enterprises. The *R4.2* version is made up of several user activity data: *email.csv*, *file.csv*, *pyschometric.csv*, *logon.csv* and *device.csv*. We used all of these files except for *pyschometric.csv* to create a set of training, testing, and validation for our experiments. The profiler learns a behavior using the training set and the incorrectly classified elements in the validation set. To show the reliability of the *BP*, we considered three insider threat attacks scenarios. All of them are available in the R4.2 version of the CERT dataset (i.e. in the *scenario.txt* file).

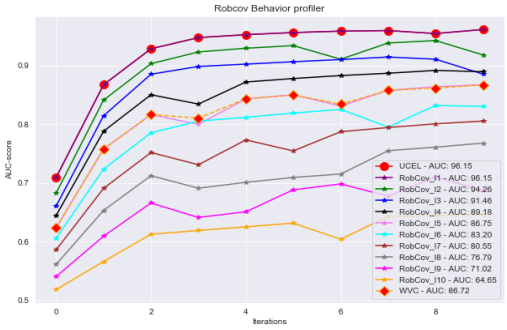
1) User who did not previously use removable drives or work after hours, begins logging in after hours, using a removable drive, and uploading data to wikileaks.org. He leaves the organization shortly thereafter.
2) User begins surfing job websites and soliciting employment from a competitor. Before leaving the company, she/he uses a thumb drive (at markedly higher rates than their previous activity) to steal data.
3) System administrator becomes disgruntled. Downloads a keylogger and uses a thumb drive to transfer it to her/his supervisor's machine. The next day, she/he uses the collected keylogs to log in as his supervisor and send out an alarming mass email, causing panic in the organization. She/he leaves the organization immediately.

The parallel architectures that used as support for the presented experiments is GRID5000 (or G5K). Grid'5000 is a

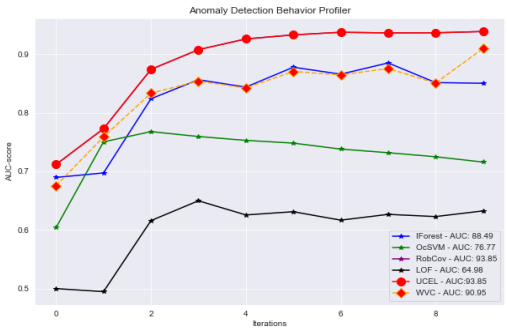
french national and international cluster constituted by several large homogeneous sub-clusters situated through France but also in Brazil and Luxembourg. This large-scale and flexible platform is a testbed for experiment-driven research in all areas of computer science. It focuses on parallel and distributed computing. G5K resources are 15000 cores, 800 compute-nodes, and also technologies such as GPU, SSD and Infiniband. For our experiments, we used mainly used nodes from the cluster of the Lille site.

V. Performance analysis

A. Model validation



(a) UCEL with *multiple Robcov* with 10 co-methods



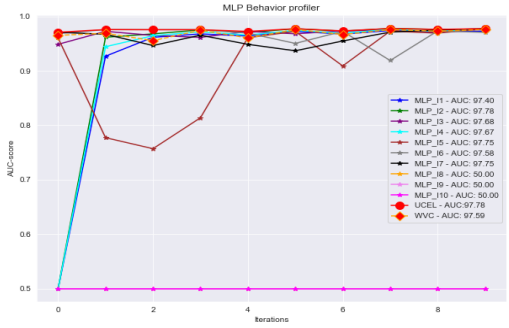
(b) UCEL with 4 different Anomaly Detection co-methods

Fig. 2 Training AUC-score evolution through iterations in an anomaly detection *BP* algorithm

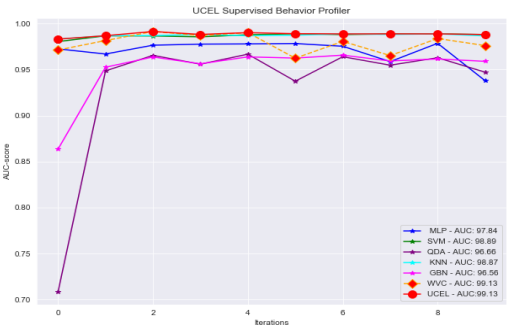
Method	TT without UCEL	TT with UCEL
Robcov	0.61 - 0.53	0.96 - 0.95
4AD	0.75 - 0.52	0.93 - 0.93
MLP	0.96 - 0.50	0.97 - 0.96
5SM	0.98 - 0.98	0.99 - 0.99

TABLE (I) Train and test with and without UCEL

First of all, we focus on *scenario 2* to check the benefice of the UCEL approach to manage the bias and variance tradeoff. The Figure 2 and Figure 3 depicts AUC-score in the function of the number of iterations of the UCEL framework. The curves in the figure represent the evolution of the training AUC-score of the co-methods. The sub-figure 2(a) and 2(b) represent respectively a UCEL execution of 10 instances of the robust covariance classifiers and 4 different anomaly detection



(a) UCEL with *multiple MLP* with 10 co-methods



(b) UCEL with 5 different supervised co-methods

Fig. 3 Training AUC-score evolution throughout iterations in a supervised *behavior profiler*

classifiers use as co-methods(i.e. IForest, OcSVM, Robcov, LOF). For example for *Robcov(k)*, k represent the k^{ieme} instance of a Robust Covariance classifier. Since the score is pretty low after the first iteration, we can suspect that, individually, the co-methods suffer either from underfitting, overfitting, or poor calibration of the hyperparameters or don't have enough sample to establish a correct decision boundary. UCEL boost their initial low AUC-score through the iterations.

We are particularly interested in the peak accuracy of the co-methods or the WVC. In Adaboost [5], the weighted voting systems weight positively the classifier with low error rates, and negatively the one with high error rates. In our case, we focus on the peak AUC-score of the learners. We will investigate in future work, other weighting and restarting functions (e.g. additionally to the FP/FN we will send the best hyperparameters).

Some co-methods showcase a little drop of performance after reaching their peak or oscillating between low and high values from an iteration to another. We suspect that this is due to the choice of the hyperparameters. The OcSVM co-method doesn't seem to benefit from this boosting strategy after the second iteration. We suspect that it is badly tuned. However, we need to do further investigation, particularly at the level of its objective function and the establishment of the decision boundary when we inject new elements. Haidar et al. [7] have proven the efficiency of an injection of false-positive

to OcSVM classifiers to improve its performance, however not in an iterative boosting manner. We also know that our model is sensitive to an unlucky random sampling. These cases can be spotted when the AUC-score is oscillating.

If the AUC threshold is not reached, UCEL selects the best method or *WVC* during the iterations. For this example, we choose the maximum number of iterations equal to 10. In sub-figure 2(a) and 2 (b) the precision threshold is never reached, but the AUC-score increase from 0.50 to 0.96 for (a), and 0.50 to 0.93 for (b). If we focus on the sub-figure(a), UCEL mostly improves the training accuracy from the methods with an initially low AUC-score. This is a direct consequence of the use of this particular combination of boosting and bagging that improves weak learners' training errors.

However, the Robcov instances (9) and (10) don't seem to be improving a lot by UCEL. This is indicative of a poor selection of hyperparameters. This implies that tuning a classifier plays a non-negligible role in the performance of UCEL. Hence, except for the instances (9) and (10), the methods starting with low training AUC-score get improved by the restarting process injection of the wrongly classified element. This also indicates that even when two of the co-methods do not contribute to the classification performance, the UCEL approach still allows the other co-methods to get better classification results.

Table I shows the result of the training and testing AUC-score without and with UCEL framework. Let *4AD* represents an execution UCEL with four anomaly detection methods and *5SM* a UCEL with five different supervised methods. The train-test(TT) results showcase high bias and high variance issues from the best method of the ten Robcov instances, and the four anomaly detection classifiers without UCEL. This points out that UCEL helps to manage bias and variance tradeoff to obtain better testing results.

In the supervised learning case, the sub-figures 3 (a) and 3 (b) respectively present 10 MLP instances in (a) and 5 different supervised classifiers in (b). In this case, we also remark that UCEL only improves the co-methods with starting low AUC-score and doesn't improve the ones with an already high score. This a consequence of boosting and bagging working well only with weak learners as base methods. Strong learners can get improvement using this strategy, but not to the same extent than weak learners [8]. For instance, the KNN, GNB, and SVM classifiers are not improved by the UCEL process. Their AUC-score stays rather good and stable through the iterations. This is also indicative of well-tuned classifiers for this problem.

However, in (b), the WVC of UCEL using all the co-methods produces better results than the individual classifiers. Hence the WVC was then chosen as the best model for the behavior profiler. In table I, we can observe that the best method of 10 MLP instances is still suffering from overfitting since the train test score varies from 0.96-0.50. The UCEL framework fixes this issue and helps to obtain a train-test score of 0.97-0.96. In the example with the five supervised learning methods, the AUC-score is 0.98, so pretty high for the individual co-methods. Despite that, UCEL adds an improvement of 1% to their scores. We can conclude that UCEL helps to improve the class prediction performance of the training and the testing error by managing the bias-variance tradeoff. Even if the classifiers are already performing well, UCEL might add a slight improvement with its weighted voting mechanism.

Considering the overall performance of UCEL for the three scenarios of insider attacks, we obtain mostly satisfying results. Table II presents the test results of 10 Robcov, 4 AD, 10 MLP, and 5SM methods. In most of the scenarios, we tend to see better classification results for supervised methods than anomaly detection methods. This confirms the best strategy is to adopt the use of semi-supervised methods when the data is imbalanced, and then use supervised methods when the companies dispose of enough feedback. They're also the possibility to apply oversampling techniques to the imbalanced dataset before using supervised learning methods [14].

Scenario	10 Robcov	4AD	10 MLP	5 SM
1	0.95	0.95	0.95	0.96
2	0.95	0.93	0.96	0.99
3	0.82	0.64	0.98	0.93

TABLE (II) Test results for 3 types of insider attack

B. Parallel performance analysis

We highlighted that the UCEL approach gives reliable results to detect insiders. To study the performance of the parallel version of the behavior profiler, we ran an implementation of parallel algorithm 2 on the *GRID5000* platform. On the Lille cluster of GRID5000 we used 9 nodes with 4 cores each for our experimentation. The Python language and the *Multiprocessing*, *Multithreading*, and *mpi4py* APIs and libraries are used to express the algorithm' parallelism. The performance of the implemented *BP* is measured in terms of speedup that we can obtain when dataset size increases. We recall that speedup represents the ratio of the serial and parallel execution time of an implementation.

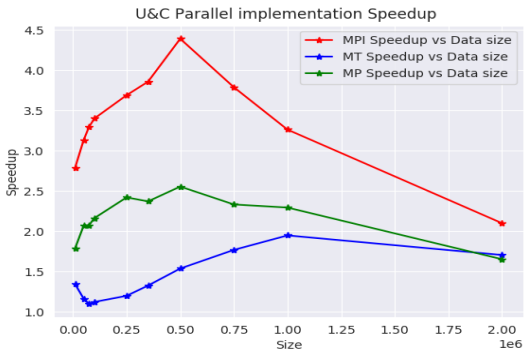


Fig. 4 Parallel *behavior profiler* on GRID'5K (8 co-methods run on 9 nodes)

Figure 4 shows a significant speedup which reaches 4.3 when the *mpi4py* library is used. This is because the co-methods are working in parallel for their training and testing phase. This confirms that *mpi4py* is more fit to benefit from cluster hardware than the other one. Even though the execution is always faster for the parallel implementation, we can

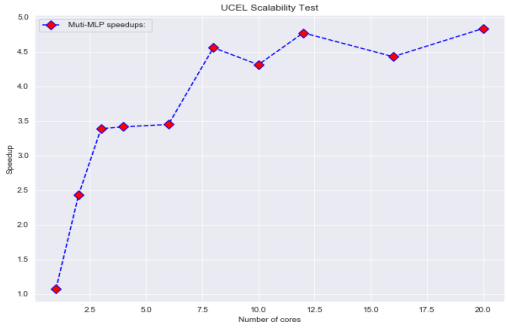


Fig. 5 Parallel *behavior profiler* strong scalability on GRID'5K (10 co-methods)

notice a limitation on the speedup gains when the number of data records increases past 500000. A small drop in the performance might occur due to the stochastic nature of the algorithm used. However, after 500000 records, the performance loss is continuous, and for all the parallelization libraries. This drop in performance is probably due to the multiplications of the communications at the synchronization steps. The co-methods have relatively different execution times. So the synchronization imposes the faster methods to wait for the slowest before restarting a cycle. Clearly, we can conclude that a better execution time can be obtained when *behavior profiler* model is run in a high performance mode. However, the synchronizations in UCEL algorithms, limit those benefits. We will take into account the asynchronous communications in the future implementation of our *BP* model. Figure 5 presents the strong scalability test of supervised BP. We run this test using a UCEL implementation with 10 MLP as co-methods, and a fixed activity dataset size of 500000 entries. The result of this test highlights that the speedup of our parallel behavior profiler rises from 1 to approximately 4.5 when we increase the number of processing cores. This figure demonstrates the scalability character of the UCEL implementation.

VI. Conclusion

User behavior analysis software has become essential tools to counter insider threats. Based on the analysis of user post logging activities, these tools' goal is to detect insiders in a company environment. In this work, we presented a new detection method based on the application of a unite and conquer approach to ensemble learning techniques. It helps us address issues that insider detection software faces today, such as high FN/FN rates and versatility. We highlighted that this framework, called UCEL, gives reliable results to detect insiders. Indeed, we have shown that UCEL increases the performances of all weak learners, composing the behavior profiler. On the other hand, UCEL doesn't have a beneficial impact on strong learners. But the presence of these latter, as co-methods, has a positive impact on the global method's results. We studied three insider attack scenarios, and we showed that the UCEL framework is reliable and gives good results. Besides, the study of these scenarios confirmed that the best detection strategy is to adopt semi-supervised methods when the data is imbalanced and then to use supervised methods when the companies dispose of enough labeled data. To study the performance of a parallel version of our behavior profiler, we ran a kind of parallel client-server implementation of the algorithm 2 on the *GRID5000* platform. The presented results show that we can obtain up to 4.3 speedup. Nevertheless, due to a synchronization step after each iteration, this speedup decreases when the dataset size increases. The solution to this issue is to use asynchronous communications in the algorithm. Indeed, asynchronous communications would allow overlapping communications with computation.

References

[1] You Chen, Steve Nyemba, Wen Zhang, and Bradley Malin. Specializing network analysis to detect anomalous insider actions. *Security informatics*, 1(1):5, 2012.

[2] Nahid Emad and Serge Petiton. Unite and conquer approach for high scale numerical computing. *Journal of computational science*, 14:5–14, 2016.

[3] Nahid Emad, S-A Shahzadeh-Fazeli, and Jack Dongarra. An asynchronous algorithm on the netsolve global computing system. *Future Generation Computer Systems*, 22(3):279–290, 2006.

[4] Kévin Fauvel, Véronique Masson, Elisa Fromont, Philippe Faverdin, and Alexandre Termier. Towards sustainable dairy management-a machine learning enhanced method for estrus detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3051–3059, 2019.

[5] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[6] Nico Görnitz, Marius Kloft, Konrad Rieck, and Ulf Brefeld. Toward supervised anomaly detection. *Journal of Artificial Intelligence Research*, 46:235–262, 2013.

[7] Diana Haidar and Mohamed Medhat Gaber. Adaptive one-class ensemble-based anomaly detection: an application to insider threats. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2018.

[8] Adam James Hall, Nikolaos Pitropakis, William J Buchanan, and Naghmeh Moradpoor. Predicting malicious insider threat scenarios using organizational data and a heterogeneous stack-classifier. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 5034–5039. IEEE, 2018.

[9] Haiwu He, Guy Bergère, and Serge Petiton. A hybrid gmres/ls-arnoldi method to accelerate the parallel solution of linear systems. *Computers & Mathematics with Applications*, 51(11):1647–1662, 2006.

[10] AS Khwaja, A Anpalagan, M Naeem, and B Venkatesh. Joint bagged-boosted artificial neural networks: Using ensemble machine learning to improve short-term electricity load forecasting. *Electric Power Systems Research*, 179:106080, 2020.

[11] S Kotsiantis and P Pintelas. Combining bagging and boosting. *International Journal of Computational Intelligence*, 1(4):324–333, 2004.

[12] Saeed Masoudnia and Reza Ebrahimpour. Mixture of experts: a literature survey. *Artificial Intelligence Review*, 42(2):275–293, 2014.

[13] Ponemon. 2018 coast of insider threat global organizations, ponemon insitute research report. *Managed Security services*, https://www.observeit.com/ponemon-report-cost-of-insider-threats/(Last accessed 4), 2018.

[14] Naghmeh Moradpoor Sheykhkanloo and Adam Hall. Insider threat detection using supervised machine learning algorithms on an extremely imbalanced dataset. *International Journal of Cyber Warfare and Terrorism (IJCWT)*, 10(2):1–26, 2020.

[15] Aaron Tuor, Samuel Kaplan, Brian Hutchinson, Nicole Nichols, and Sean Robinson. Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. *The Workshops of the The Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[16] Fangfang Yuan, Yanan Cao, Yanmin Shang, Yanbing Liu, Jianlong Tan, and Binxing Fang. Insider threat detection with deep neural network. In *International Conference on Computational Science*, pages 43–54. Springer, 2018.