Machine Learning manuscript No.
(will be inserted by the editor)

RADE: Resource-Efficient Supervised Anomaly Detection Using Decision Tree-Based Ensemble Methods

Shay Vargaftik · Isaac Keslassy · Ariel Orda · Yaniv Ben-Itzhak

Received: date / Accepted: date

Abstract The capability to perform anomaly detection in a resource-constrained setting, such as an edge device or a loaded server, is of increasing need due to emerging on-premises computation constraints as well as security, privacy and profitability reasons. Yet, the increasing size of datasets often results in current anomaly detection methods being too resource consuming, and in particular decision-tree based ensemble classifiers.

To address this need, we present RADE – a new resource-efficient anomaly detection framework that augments standard decision-tree based ensemble classifiers to perform well in a resource constrained setting. The key idea behind RADE is first to train a small model that is sufficient to correctly classify the majority of the queries. Then, using only subsets of the training data, train expert models for these fewer harder cases where the small model is at high risk of making a classification mistake.

We implement RADE as a *scikit-learn* classifier. Our evaluation indicates that RADE offers competitive anomaly detection capabilities as compared to standard methods while significantly improving memory footprint by up to $12\times$, training-time by up to $20\times$, and classification time by up to $16\times$.

Keywords Resource efficient machine learning · Fast machine learning · Anomaly detection · Supervised learning · Decision-tree based ensemble methods

Shay Vargaftik
VMware Research
E-mail: shayv@vmware.com

Isaac Keslassy
VMware Research
Technion
E-mail: isaac@ee.technion.ac.il

Ariel Orda
Technion
E-mail: ariel@ee.technion.ac.il

Yaniv Ben-Itzhak
VMware Research
E-mail: ybenitzhak@vmware.com

1 Introduction

Machine learning (ML) based anomaly detection is exceptionally challenging and has a wide range of applications in domains such as finance, fraud detection, surveillance, health care, intrusion detection, and medical diagnosis. For example, an anomalous network traffic pattern could mean that a hacked device sends out sensitive data to an unauthorized destination. Anomalies in credit card transactions could indicate credit card or identity theft. Also, anomaly readings from various sensors could signify a faulty behavior in hardware or a software component of an automotive.

Employing anomaly detection in a resource-constrained setting or device is an on-going challenge, due to the following and often contradicting requirements. It has to:

- Locally carry out a substantial amount of computation, storage, and communication.
- Adhere to resource constraints such as limited memory, network connectivity and CPU due to timing or power constraints.
- Deal with the increasing amounts of data and available information.

A key supervised anomaly detection ML solution for tabular data is to employ decision-tree-based ensemble methods (DTEMs). DTEMs rely on either bagging or boosting techniques to improve their detection capabilities and offer appealing robustness, ease of use, and generalization properties as we next describe.

Bagging (or bootstrap aggregation). Bagging methods such as tree bagging [6], random forest [8] and extra trees [34] offer a controlled variance which is used to improve classification accuracy as well as achieve better generalization properties. In particular, Random Forest (RF) [8] is the most well-known and widely employed decision-tree-based bagging method. In a nutshell, RF grows each decision tree using a random subsample of the data and possibly different split features in tree nodes, resulting in different and weakly correlated trees. Then, a majority vote is used to determine the classification.

Several studies [37, 80, 86, 100] have proposed using RF for supervised anomaly detection. For instance, [100] employed RF for anomaly detection by using data mining techniques to select features and handle the class imbalance problem; and, [80] provided a scalable implementation of intrusion detection system.

RF is a popular classifier as it often offers appealing properties in comparison to other classification methods, such as Neural Networks [3], Support Vector Machines [32], Fuzzy Logic methods [9], and Bayesian Networks [45]. Specifically, RF offers : (1) robustness and moderate sensitivity to hyper-parameters; (2) parallel and fast training; (3) capability to deal with imbalanced datasets; (4) embedded feature selection and ranking capabilities; (5) handling missing, categorical and continuous features in many cases; (6) interpretability for advanced human analysis for further investigation or whenever such capability is required by regulations [70], *e.g.,* in order to understand the underlying risks. To that end, an RF can be interpreted by different methods, such as [4].

All these aforementioned advantages are repeatedly pointed out in the literature via analysis as well as comparative tests (see [36, 57, 69] and references therein) especially for intrusion detection (IDS) [37, 86, 100], fraud detection [97] and anomaly detection [80, 101].

Are bagging methods a natural fit for resource-constrained settings?
Traditional bagging classifiers tend to be memory bound and slower at classification [2, 51, 56, 92]. Moreover, standard bagging methods are typically less suitable for acceleration using embedded systems (*e.g.,* FPGAs, or GP-GPUs). This is mainly due to the need to traverse

numerous large decision trees. Indeed, the classification time of a large RF makes it infeasible for resource-constrained devices and its large memory footprint makes it infeasible for devices with little memory.

Boosting. Unlike bagging, boosting primarily offers controlled bias (and also variance). Many popular decision-tree-based boosting methods such as GBDT [30,38], XGBoost [18], LightGBM [43], CatBoost [64] and AdaBoost [29] employ the boosting concept, usually, by using iterative training. For example, in Adaptive Boosting, a weak classifier such as a *stump*[1] is added at each iteration (unlike bagging methods that use full-grown trees) and typically weighted with respect to its accuracy. Then, the data weights are readjusted such that a higher weight is given to the misclassified instances when considering all previous trees. In Gradient Boosting, a small decision tree (*e.g.,* with 8-32 terminal nodes) is added at each iteration and scaled by a constant factor. Then, a new tree is grown to reduce the loss function of the previous trees. For both methods, the next trees are trained with more focus on previous misclassifications.

Decision-tree-based boosting methods are known to be among the best off-the-shelf supervised learning methods available [52, 71, 72, 74], achieving appealing accuracy with only modest memory footprint, as opposed to RF that is usually memory bounded. Boosting methods also share many of the aforementioned advantages offered by RF [39].

That being said, boosting methods are also more sensitive to overfitting than RF, especially when the data is noisy [25]. Often, they are also harder to tune since they have more hyperparameters and higher sensitivity to these hyperparameters.

Are boosting methods a natural fit for resource-constrained settings?
The training of boosting classifiers is often significantly longer than bagging classifiers such as RFs, mainly since the trees are built sequentially and compute-intensive tasks such as classification and data weights readjustments take place at every iteration. In this compute-intensive process, it is often the case that the same training data is scanned and processed multiple times. For a resource-constrained device, this means that off-premise training may be required (*e.g.,* over the cloud), which in turn requires transferring the dataset and receiving the trained model over the network. However, that may require considerable network bandwidth and connectivity. Local training of boosting ML models is preferable when the network connectivity is limited or expensive (in terms of cost or energy), and when the data privacy is of great concern. Boosting methods also admit slower classification as the number of trees increases [1, 67] which makes them less suitable for resource-constrained settings and devices.

Motivation. Our comparison (Section 3.4) between bagging (RF) and boosting (XGBoost) to well-known classifiers including Decision-tree (DT), K-nearest-neighbors (KNN), Multilayer Perceptron (MLP), and Support Vector Classifier (SVC) shows that RF and XGBoost offer a balanced trade-off between ML and system performance (*i.e.,* ML scores vs. time and memory) where the best method often depends on the specific dataset and the application.

Nevertheless, both bagging and boosting are challenged by the continuous growth of datasets [50] in terms of the number of features and data instances alongside the increasing demand for lower memory footprint, faster training, and lower classification time.

With powerful under-loaded servers at hand, a sufficiently large and adequately trained DTEM may offer satisfactory anomaly detection capabilities. However, for a resource-constrained setting or device, this would typically lead to at least one of the following drawbacks:

- Large memory footprint.

[1] A one-level decision tree.

– Long training time.
– High energy consumption.
– Long classification time.

This is because in such cases the available hardware resources are low. For example, the clock speed of edge devices varies from tens of MHz (*e.g.,* Arduino MKR1000 with 48 MHz ARM) and up to 1 GHz (*e.g.,* Raspberry Pi 3 and Dell Edge 3000 series) with a low core count (*e.g.,* 1-4) and constrained power consumption. Their memory and RAM/L2 cache sizes vary from 256KB/no-cache (Arduino MKR1000) and up to 1GB/512KB (Raspberry Pi 3) and 2GB/1MB (Dell Edge 3000 series). This is in contrast to servers. For example, a `t2.2xlarge` AWS EC2 instance offers 8 cores at 2.3-GHz, 32 GB RAM, and 4.5 MB/45 MB of L2/L3 caches (Intel Broadwell E5-2686v4).

Contributions. Our aim is to make a step forward in overcoming the aforementioned performance drawbacks, which we find to be essential for efficient execution of DTEM-based anomaly detection in a resource-constrained settings and devices.

Accordingly, in this paper, we present, design, and evaluate RADE. RADE is a resource-efficient DTEM-based anomaly detection approach that merely augments standard DTEM classifiers with competitive anomaly detection capabilities and while offering significant savings in resource usage. In particular, in comparison to standard state-of-the-art DTEMs, RADE models are smaller in terms of memory footprint; faster to train in terms of required resources; offer lower response time / higher throughput. In turn, this also means that a classification query does not require too much computation resources.

The core idea behind RADE is based on two observations:

1. We can build a small model that is sufficient to classify correctly the majority of the classification queries.
2. For these harder cases, where the small model is not sufficiently confident and is at high risk of making a classification mistake, we can train expert models using only subsets of the training data.

Due to its modular design, RADE is orthogonal to the discussed bagging and boosting techniques and can augment them to form a more resource-efficient DTEM classifier.

We implement RADE as a scikit-learn classifier [11] and conduct extensive evaluation using different publicly available datasets. Evaluation results over different AWS EC2 instances and a Raspberry Pi 3 device are consistent and indicate that RADE offers competitive anomaly detection capabilities as compared to the monoloithic state-of-the-art DTEMs while significantly reducing model memory footprint ($12\times$), training-time ($20\times$), and classification time (by up to $16\times$).

2 RADE

In the following, we describe RADE's architecture in detail and provide some intuition as to why it is expected to be resource-efficient in comparison to standard monolithic DTEMs while offering competitive anomaly detection capabilities.

2.1 Observations

As mentioned, RADE has a hierarchical structure in which we first train a small ML model using the entire training dataset and only then train two expert ML models, using only sub-

sets of the training dataset, to complement the small ML model. We now detail the two observations that led us to such architecture.

1. **We can build a small ML model that is sufficient to classify the majority of the classification queries correctly.** An appealing property offered by DTEMs is that every classification query not only receives a label but it also has a corresponding empirical measure termed *confidence level* which is a measure of how confident the ML model is in its classification result [8]. For a DTEM, the confidence level is calculated by incorporating the individual votes of all trees into a single class distribution vector and taking the maximal value[2]. We heuristically use the confidence level as an indication to which data instances the small model finds easy (*i.e.,* instances with a high classification confidence) and which it finds hard (*i.e.,* instances with a low classification confidence). Specifically, we find that a small DTEM model can be used to correctly classify the majority of classification queries (but not necessarily all) by requiring a sufficiently high classification confidence level. That is, the classification result of a small model is *valid* only if its corresponding confidence level is sufficiently high, rather than accepting any classification confidence.
2. **For these harder cases, where the small ML model is not sufficiently confident and is at high risk of making a classification mistake, we can train expert ML models using only subsets of the training data.** With a small ML model that offers a valid classification result only for the high confidence queries, we remain with the harder queries, *i.e.,* that fraction of queries without a valid classification by the small model due to an insufficient confidence level. For this fraction of harder queries we build expert ML models. To do so in a resource efficient manner, we want to identify only the relevant subsets of the training data that will train the expert models to succeed specifically where the small model has low confidence in its classifications and is more likely to make a classification mistake. To do so, we observe that it is possible to leverage the small model we have already built. Specifically, we classify the entire training dataset by the small model and obtain the classification confidence level of each data instance. Roughly speaking, we "filter" data instances that the small model finds easy and train the experts using only data instances it finds hard. As we later detail, we use two expert models, each of which is responsible to handle a specific type of possible misclassifications by the small model.

2.2 RADE's Architecture

RADE's architecture is illustrated in Figure 1. We next detail RADE's classification and training processes as well as introduce new hyper-parameters and provide further intuition into our design choices.

Classification. Algorithm 1 describes the procedure for a classification by RADE. First, we classify an arriving data instance by the coarse-grained model (M_{cg} in line 1). Whenever the resulting confidence level is greater than or equal to the *classification confidence threshold* (CCT) the classification by the coarse-grained model is *valid* and therefore returned (lines 2-4). Otherwise, if the resulting confidence level is lower than CCT, the query is forwarded to one of the fine-grained models, which is chosen according to the coarse-grained

[2] For example, if a classification output is (Normal=0.78, Anomaly=0.22) that means that the instance is classified as Normal with a classification confidence level of 0.78.

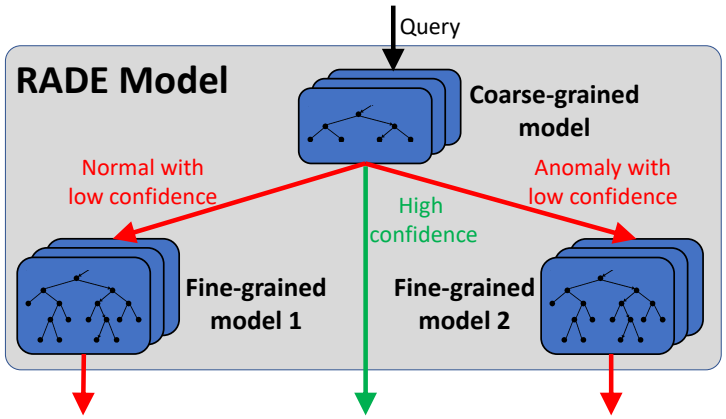


Fig. 1: RADE'S architecture. Upon a classification query, it is first classified by the small ML model (termed coarse-grained). Only if the classification confidence by the coarse-grained model is not sufficiently high, the query is forwarded to an expert ML model (termed fine-grained) for re-classification. Fine-grained model 1 is responsible for low-confidence Normal classifications and Fine-grained model 2 for low-confidence Anomaly classifications, of the coarse-grained ML model.

Algorithm 1 RADE classification

Input: Unlabeled data point x, confidence threshold **CCT**.
1: obtain coarse-grained distribution: $d_x = M_{cg}(x)$
2: classify: $y = argmax(d_x)$
3: if $max(d_x) \geq$ **CCT**:
4: return y
5: else:
6: $y == Normal\ ?\ c = 1 : c = 2$
7: obtain fine-grained distribution: $\bar{d}_x = M^c_{fg}$
8: classify: $\bar{y} = argmax(\bar{d}_x)$
9: return $\bar{y}$

classification (lines 5-9). Specifically, if the coarse-grained low-confidence classification is Normal, then the instance is forwarded to fine-grained model 1 (M^1_{fg}) which is trained to distinguish between Normal instances that are correctly classified by the coarse-grained model and Anomaly instances. Likewise, if the coarse-grained low-confidence classification is Anomaly, then the instance is forwarded to fine-grained model 2 (M^2_{fg}) which is trained to distinguish between Normal instances that are misclassified by the coarse-grained model and Anomaly instances.

Note that Algorithm 1 introduces the new hyper-parameter CCT. Roughly speaking, CCT introduces a trade-off between error rate and resource usage. We would like to choose its value low enough such that most of the classification queries are classified by the coarse-grained model but high enough such that we keep low and competitive error rate.

Training. Algorithm 2 describes the procedure for training a RADE model. It begins with the training of a coarse-grained model using all training data (line 1). Next, we train the fine-grained models. To that end, we classify each data instance in the training data by the coarse-grained model (lines 5). Then, if the confidence level (*i.e.,* $max(d_x)$) of the data

Algorithm 2 RADE training

Input: Labeled training data set X, confidence threshold **TCT**.
1: train M_{cg} using X
2: set: $data\_mask\_1 = \emptyset$
3: set: $data\_mask\_2 = \emptyset$
4: for each $x \in X$:
5: obtain coarse-grained distribution: d_x = $M_{cg}(x)$
6: if $max(d_x) <$ **TCT**:
7: if $x.label == Anomaly$:
8: update: $data\_mask\_1.append(True)$
9: update: $data\_mask\_2.append(True)$
10: else:
11: classify: y = $argmax(d_x)$
12: if $y == Normal$:
13: update: $data\_mask\_1.append(True)$
14: update: $data\_mask\_2.append(False)$
15: else:
16: update: $data\_mask\_1.append(False)$
17: update: $data\_mask\_2.append(True)$
18: else:
19: update: $data\_mask\_1.append(False)$
20: update: $data\_mask\_2.append(False)$
21: train M^1_{fg} using $X[data\_mask\_1]$
22: train M^2_{fg} using $X[data\_mask\_2]$

instance is lower than the *training confidence threshold* (TCT), the labeled data instance is used by both experts if its label is Anomaly (lines 8-9) or otherwise used by a single fine-grained model according to the prediction made by the coarse-grained model (lines 11-17). Notice that in lines 11-17, the data instances are used by a specific fine-grained model *according to their low-confidence coarse-grained classification, and not according to their labels*. The reason is that we train the fine-grained models to succeed specifically where the coarse-grained model is insufficiently confident and is more likely to make a mistake.

2.3 Why use all low-confidence anomalies for the training of both experts?

As illustrated in Figures 2(a) and 2(b), the data instances that are used to train fine-grained model 1 contain: (1) *all* low confidence Anomaly instances and, (2) low confidence Normal instances that are correctly classified by the coarse-grained model. Intuitively, this model becomes an expert in distinguishing between Normal instances that are correctly classified by the coarse-grained model and Anomaly instances. Likewise, the data instances that are used to train fine-grained model 2 contain: (1) *all* low confidence Anomaly instances and, (2) low confidence Normal instances that are misclassified by the coarse-grained model. Intuitively, this model becomes an expert in distinguishing between misclassified Normal instances by the coarse-grained model and Anomaly instances.

Naturally, for a training dataset, the Anomaly class is considerably smaller (usually by orders of magnitude) than the Normal class in terms of the number of instances, and its low-confidence subset is even smaller. This fact results in two potential drawbacks, which we mitigate by using the low-confidence Anomaly labeled instances in the training of both fine-grained models (lines 8-9 in Algorithm 2), as we describe in the following:

- *Less accurate Anomaly coarse-grained classification:* Since the coarse-grained model is trained using a rather small number of Anomaly instances as compared to the number of Normal instances, its classifications over these instances, as we find, are very noisy with many misclassifications as compared to the Normal instances. Namely, the classification distribution vector over these instances has a significant variance, which is even more severe for the low-confidence Anomaly subset. This makes the classification of the coarse-grained model as to which fine-grained model we need to send a specific low-confidence Anomaly instance less reliable (unlike for the Normal instances).
- *Increased overfitting likelihood by the fine-grained models:* Due to the small cardinality of the low-confidence subset of the Anomaly instances, it is more likely for a fine-grained classifier to receive a non-sufficient number of such instances for training. This, in turn, increases the likelihood of overfitting the model. That is, it is more likely for the training of a fine-grained classifier to terminate in a state in which it has a nearly perfect score for the low-confidence subset of the Anomaly instances that were used for its training, but this fine-grained model is likely to be less accurate at a classification of a low-confidence Anomaly instance that have not been used for its training and thus is more likely to be too different from other labeled instances this fine-grained classifier was trained on.

Therefore, using all the low-confidence subset of Anomalies by both fine-grained models reduces the likelihood of both drawbacks and makes the fine-grained models better experts for those queries in which the coarse-grained model is more likely to make a classification mistake. As we show in our evaluation, this design choice results in a very low overhead in terms of the number of data instances used for the fine-grained models training.

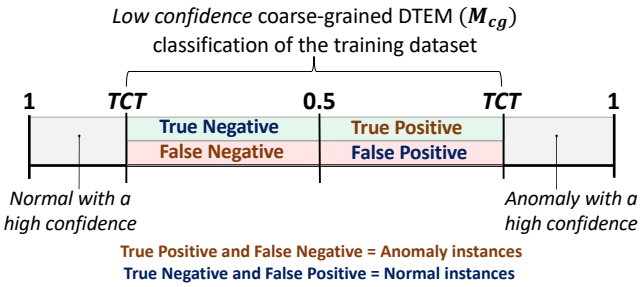
Evidently, many other data augmentation techniques such as synthesizing new cases, adding random noise and optimizing the oversampling percentage can be used by any classifier to potentially boost performance for imbalanced classification tasks (*e.g.,* [17], [102] and references therein). However, our usage of anomaly instances in both experts is unique to RADE's construction (as we have two fine-grained models) and is irrelevant to traditional state-of-the-art DTEMs. On the other hand, other techniques, as those mentioned above, are relevant to all DTEMs, including RADE, and are orthogonal to our contribution. In particular, after the dataset for each fine-grained model is selected, any of the aforementioned subsampling or up-sampling techniques can be employed when training an expert (e.g., balanced or stratified sampling within its dataset).

In Section 3.6, we show evaluation results where we subsample and oversample the low confidence anomalies for the training of the two expert models, shedding light on our default design choice.

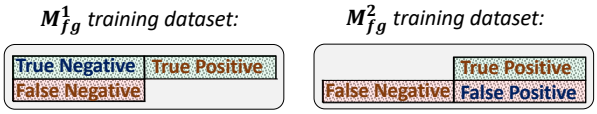
2.4 Putting it all together

Note that Algorithm 2 introduces the new hyper-parameter TCT. TCT should be chosen with respect to CCT. Roughly speaking, TCT introduces a tradeoff between the size of the fine-grained models and their training complexity to robustness against high-variance classification distributions by the coarse-grained model as we next describe.

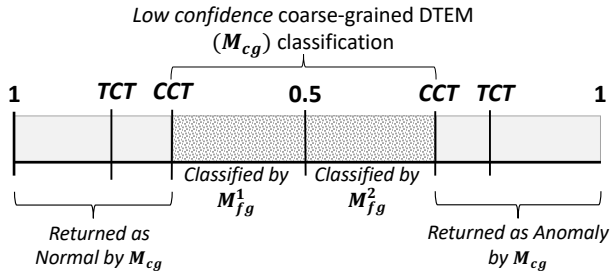
Figure 2(c) depicts an illustration of RADE's TCT driven training and CCT driven classification and how the two hyper-parameters relate. As illustrated, we may use *different* TCT and CCT values where we are interested in a subset of values such that TCT$\geq$CCT. The intuition for why it may be of interest to set TCT$>$CCT rather than TCT$==$CCT, is that it allows us to train the fine-grained models with a bigger subset of the labeled data instances



(a) We classify the training dataset using the coarse-grained model. Only the instances with low resulting classification confidence are used for the training of the fine-grained models.



(b) Illustrating the dataset of each fine-grained model. Low confidence anomalies are included in both subsets.



(c) Illustrating RADE's confidence level driven classification. CCT is chosen to be less or equal to TCT. This results in the fine-grained models being trained using a superset of instances in terms of their confidence level by the coarse-grained model in comparison to the set that is being forwarded to them for classification.

Fig. 2: RADE confidence level driven training and classification. The training confidence threshold (TCT) at the coarse-grained model determines the training data for each fine-grained classifier. For classification, the classification confidence threshold (CCT) at the coarse-grained model determines whether a classification query is returned directly by the coarse-grained model or which fine-grained model is queried for the latter case.

as compared to the classification subset that is re-classified by them. Tuning TCT, as we empirically find in our evaluations, often improves the anomaly detection capabilities for a modest price in training time and fine-grained model sizes.

2.5 On RADE and resource-efficiency

We next discuss intuition as to why RADE results in lower memory footprint, lower training time, and lower classification time as compared to standard monolithic DTEMs.

The training time and training complexity of DTEMs depend on the amount of available training data instances, the number of features (*i.e.,* the dimensionality of the data), the

number of trees and their depth limitations (if there are any). Whereas, the classification time mostly depends on the model size (*i.e.,* the number of trees and their depths). Clearly, the smaller model size of a coarse-grained classifier directly improves all of the criteria mentioned above. Beyond being fast to train and quick to query, its small size, as we find in our evaluation, often enables it to be fully/largely contained in cache memory even for small cache sizes, such as of edge devices. Additionally, as mentioned, the fine-grained classifiers are being trained by subsets of the training data, which reduces their training time, and, often, their size. Indeed, our evaluation shows that the size of the fine-grained models is considerably smaller as compared to their corresponding monolithic models for all tested data sets and classification methods.

Essentially, when considering a RADE model (*i.e.,* both the coarse-grained and the fine-grained models), on average, the classification time of RADE equals to a weighted average of the latencies according to the fraction of the classifications that are served by the coarse-grained and fine-grained models. Since the aim is for the coarse-grained model to serve most of the classifications, the averaged classification time is expected to significantly improve over standard monolithic DTEMs.

To summarize, both the coarse-grained and fine-grained models contribute to the overall improvements of RADE, in the following ways: The coarse-grained model is (1) based on a small classifier and, (2) serves most of the classification queries. The fine-grained models are (1) being trained using only subsets of the training data, (2) smaller as compared to the corresponding monolithic DTEMs and, (3) serve only a small fraction of the classification queries.

3 Evaluation

In this section, we conduct an evaluation of RADE and compare it to monolithic state-of-the-art DTEMs using different real-world datasets. We evaluate over different AWS EC2 instances and a Raspberry Pi 3 Model B Plus device. We start by briefly describing RADE's modular implementation in §3.1, introduce configuration notations in §3.2, detail the evaluation datasets in §3.3 and compare our chosen DTEM baselines to other non-DTEM state-of-the-art classifiers in §3.4.

Then, in §3.5 we turn to the evaluation of our first observation, *i.e.,* that a small (*i.e.,* coarse-grained) DTEM model is sufficient to classify the majority of the classification queries correctly. We continue to compare RADE models against monolithic RF and XGBoost in §3.7. Then, we take a deeper look into RADE's components in §3.8 showing its unique properties including the especially small size of the coarse-grained model, the reduced data subsets that are used for the training of the fine-grained models and the small data fractions that are classified by them. We also briefly discuss the new hyperparameters and our default configuration for RADE. Finally in §3.9, we conduct evaluation over Raspberry Pi 3 B Plus showing the advantages of RADE over monolithic models when employed over a resource-constrained device.

3.1 RADE's Implementation

We implement RADE as a *scikit-learn* classifier [63]. RADE's design is modular such that it supports any DTEM classifier that implements scikit-learn's API.

The code is available in [78].

	# Instances (n)	# Features (d)	Anomalies [%]
covertype	286048	54	0.96%
kddcup99	976158	41	0.35%
creditcardfraud	284807	30	0.17%
seismic	2584	18	6.58%
mammography	11183	6	2.32%
communities	1994	127	2.86%

Table 1: Evaluation datasets.

In our evaluation, we focus on the state-of-the-art widely used DTEMs

- Random forest [76] as a representative bagging DTEM.
- xgboost-v0.90 [66] as a representative boosting DTEM.

3.2 Classifier Configuration Notations

For ease of exposition, we use the following notations.

Monolithic classifier configuration. We denote a classifier C with T trees and a tree depth limitation of D, by $C(T, D)$. A classifier without a tree depth limitation is denoted by $C(T, None)$. In our evaluation C can be either a random forest (RF), or XGBoost (XGB). For instance, a RF with 10 trees, each limited to a depth of 5 is denoted by $RF(10, 5)$.

RADE classifier configuration. We denote a RADE classifier by a tuple $RADE(C(T_{cg}, D_{cg}), C(T_{fg}, D_{fg}), \text{CCT}, \text{TCT})$, that states the coarse-grained model, followed by the fine-grained models, followed by the classification and training confidence thresholds. For instance, RADE(RF(10,5),RF(25,20),0.8,0.9) is a RADE model with $RF(10, 5)$ as the coarse-grained model, $RF(25, 20)$ as each of the fine-grained models, CCT $= 0.8$, and TCT $= 0.9$.

3.3 Datasets

Table 1 summarizes the datasets we use in our evaluation. Three of them are of a higher interest to our study since they are both larger and thus more resource consuming and with a lower percentage of anomalies. All three are from different domains and use cases as we next describe:

- **covertype [54].** This dataset is used in predicting forest cover type from cartographic variables [5, 31, 59, 61]. This study includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. Class 2 is considered as Normal, and class 4 as Anomaly (0.96%).
- **kddcup99 [55].** This dataset is a popular benchmark and is widely used for the evaluation of IDS systems [24, 42, 73]. It was used for The Third International Knowledge Discovery and Data Mining Tools Competition in which the task was to build a network intrusion detector. This database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment. In our evaluation, we treat all intrusions (*e.g.,* DoS, Probe, R2L) as Anomalies (0.35%) and all non-hostile connections as Normal.

Dataset	Model	Macro F1	AUC Score	Kappa Score	Training Time [sec]	Classification Latency [sec]	Model Size [MB]
communities	DT	0.607±0.058	0.607±0.067	0.215±0.116	0.046±0.01	0.0013±0.0001	0.005±0.0
	KNN	0.493±0.0	0.5±0.0	0.0±0.0	0.014±0.001	0.0477±0.0003	2.657±0.0
	MLP	0.561±0.041	0.547±0.033	0.128±0.078	1.083±0.908	0.0018±0.0002	0.302±0.001
	SVC	0.491±0.061	0.635±0.055	0.08±0.048	0.26±0.008	0.1219±0.0041	1.015±0.034
	RF	0.591±0.062	0.561±0.043	0.199±0.124	0.299±0.014	0.0046±0.0001	0.396±0.014
	XGBoost	0.617±0.052	0.582±0.041	0.238±0.104	0.464±0.004	0.0039±0.0002	0.057±0.002
creditcardfraud	DT	0.87±0.004	0.867±0.011	0.741±0.009	13.504±1.447	0.0173±0.001	0.018±0.0
	KNN	0.542±0.01	0.523±0.006	0.085±0.02	1.109±0.058	4.6257±0.0183	91.569±0.0
	MLP	0.837±0.046	0.837±0.088	0.675±0.091	362.529±22.293	0.5022±0.0585	0.084±0.0
	SVC	0.307±0.01	0.572±0.017	0.001±0.0	5905.542±192.291	953.3254±35.9696	40.944±0.479
	RF	0.923±0.01	0.884±0.013	0.846±0.018	143.845±5.947	1.0881±0.0277	1.83±0.088
	XGBoost	0.92±0.01	0.888±0.017	0.841±0.02	43.287±0.321	0.3027±0.0136	0.066±0.001
kddcup99	DT	0.998±0.001	0.997±0.001	0.996±0.002	4.938±0.478	0.1914±0.0024	0.006±0.0
	KNN	0.997±0.0	0.995±0.001	0.994±0.0	212.902±6.352	81.9521±0.3294	428.133±0.0
	MLP	0.998±0.0	0.997±0.001	0.996±0.001	751.677±336.662	1.4721±0.0691	0.106±0.001
	SVC	0.799±0.035	0.995±0.001	0.599±0.07	16199.047±5556.238	4128.2109±979.2873	57.734±13.313
	RF	0.999±0.0	0.998±0.001	0.998±0.001	28.934±3.024	1.3763±0.0254	0.549±0.032
	XGBoost	0.999±0.0	0.998±0.001	0.996±0.001	63.554±2.03	0.8775±0.0327	0.057±0.0
mammography	DT	0.789±0.022	0.782±0.029	0.579±0.044	0.024±0.002	0.0003±0.0	0.017±0.001
	KNN	0.803±0.023	0.752±0.023	0.607±0.045	0.019±0.007	0.1924±0.0028	0.809±0.0
	MLP	0.828±0.01	0.779±0.01	0.657±0.02	5.27±0.39	0.0027±0.0001	0.028±0.0
	SVC	0.698±0.014	0.903±0.008	0.404±0.027	0.615±0.061	0.1788±0.0095	0.099±0.005
	RF	0.828±0.017	0.767±0.021	0.659±0.038	0.531±0.019	0.0272±0.0008	1.516±0.075
	XGBoost	0.83±0.013	0.781±0.017	0.661±0.026	0.293±0.003	0.0074±0.0002	0.061±0.001

Table 2: ML and resource consumption comparison among different monolithic classifiers.

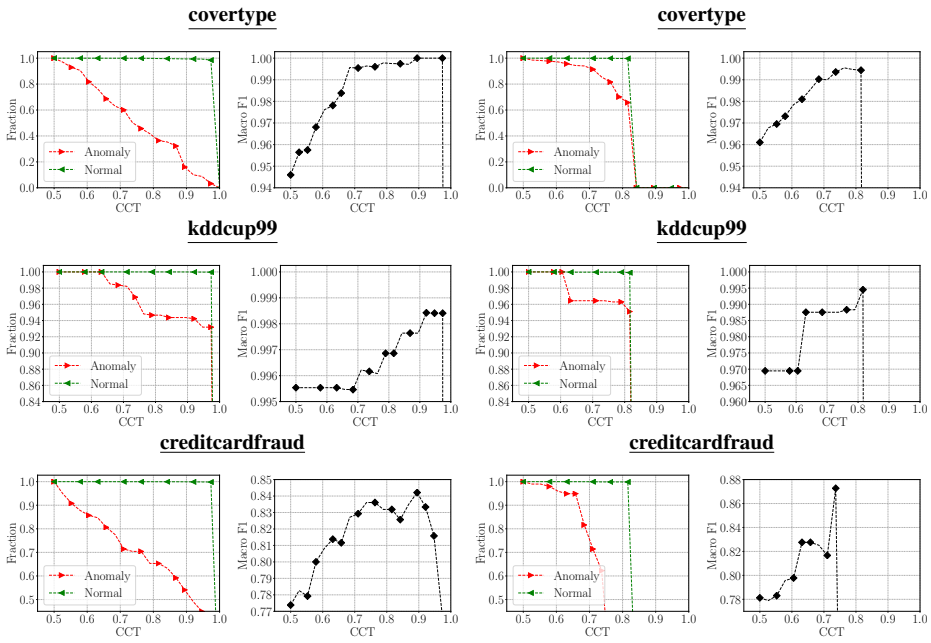
– **creditcardfraud [53].** This is a popular dataset that is used for anomaly and fraud detection benchmarking. [13, 14, 21, 22, 48]. The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, with frauds which we treat as Anomalies (0.17%) where all the rest are legitimate (Normal) transfers.

While we give more focus to the three aforementioned largest datasets, all datasets, including *seismic*, *mammography* and *communities* are widely used and publicly available via the UCI repository [26]. *creditcardfraud* is available via Kaggle [53] and *kddcup99* and *covertype* are also available via scikit-learn's *datasets* module.

3.4 Monolithic DTEMs as Baselines

In this paper, we focus on DTEMS. Nevertheless, we next compare our chosen baselines, *i.e.,* RF and XGBoost, to several non-DTEM state-of-the-art classifiers: Decision-tree (DT), K-nearest-neighbors (KNN), Multi-layer Perceptron (MLP), and Support Vector Classifier (SVC). We perform the ML performance comparison over three different scores: Macro-F1, area-under-the-curve (AUC), and Kappa [20].

The results are depicted in Table 2. Both RF and XGBoost offer an appealing tradeoff between ML performance and resource consumption. In particular, in our datasets, they mostly achieve the best ML performance across all three ML metrics while being competitive in resource consumption with other classifiers (except KNN and SVC, whose resource consumption is usually higher than the other methods and MLP whose training time is usually higher than the other methods). Note that DT is often less resource-consuming than other methods. However, it mostly achieves lower ML scores than RF and especially XGBoost.



(a) Classification by a small RF with 10 trees each of which is limited to a depth of 5 – $RF(10, 5)$.

(b) Classification by a small XGBoost with 10 trees each of which is limited to a depth of 3 – $XGB(10, 3)$.

Fig. 3: The useful classification fraction and its resulting macro F_1 score by a small ML model (coarse-grained model), when a classification is *valid* only if its confidence level is greater than or equal to a given classification confidence threshold (CCT).

3.5 Small ML model with a high confidence?

We turn to the evaluation of our first observation (the first of two observations on which RADE's architecture relies on):

"We can build a small ML model that is sufficient to classify the majority of the classification queries correctly" (§2.1)

Setup. We conduct evaluation over:

- A small RF with only 10 trees, each of which is limited to a depth of 5 (*i.e.,* $RF(10, 5)$).
- A small XGBoost with only 10 trees, each of which is limited to a depth of 3 (*i.e.,* $XGB(10, 3)$).

We present the results for these specific hyperparameters since these are, in fact, the default sizes of our coarse-grained models in RADE. In these experiments we focus on the 3 largest datasets, *i.e., covertype*, *kddcup99* and *creditcardfraud*.

In each experiment, we sweep over CCT and measure the valid classification fraction of Anomaly and Normal instances as well as their Macro F1 score[3].

[3] We use F1 score since it is more useful than Accuracy, for an uneven class distribution. Indeed, this is the case for anomaly detection purposes. Macro F1 score is the non-weighted mean of the Anomaly and Normal F1 scores.

RF(10,5). Figure 3(a) shows the evaluation results. Several phenomena are evident:

- As CCT increases, the fraction of valid classifications decreases. This phenomenon is especially significant for the anomalies. Indeed, this comes in line with the intuition that anomalies, due to their significantly smaller number and possible uniqueness, are much harder to classify correctly and with a high confidence for this small model.
- Unless CCT is set too high or too low, the resulting fraction of the valid classifications has a higher Macro F1 score. Again, this comes in line with the intuition that higher confidence is a good indication for a correct classification. It is important to note, however, that this phenomenon is not necessarily monotonic. For example, for the *creditcardfraud* dataset, CCT $= 0.85$ leads to a higher score than CCT $= 0.9$. This means that several classifications that were correct with a confidence level between 0.85 and 0.9 became not valid with the increase of CCT.
- Unless CCT is set too high, the resulting fraction of non-valid classifications is at most $\approx 1\%$. This is because over 99% of the data points are normal, so the non-valid anomalies account only for a small fraction of the entire dataset.
- As can be seen by the *kddcup99* and *creditcardfraud* datasets, we occasionally find few misclassifcations with a high confidence. These are often points that are misclassified even by large monolithic models since they are located deeply in the domain of the opposite class.
- If CCT is set too high, all / nearly all classification results are not valid and therefore their respective fractions drop.

To summarize, we find that working with a small model by setting a sufficient CCT results in a high fraction of valid classifications with a Macro F1 score that is competitive to large monolithic models as we later show.

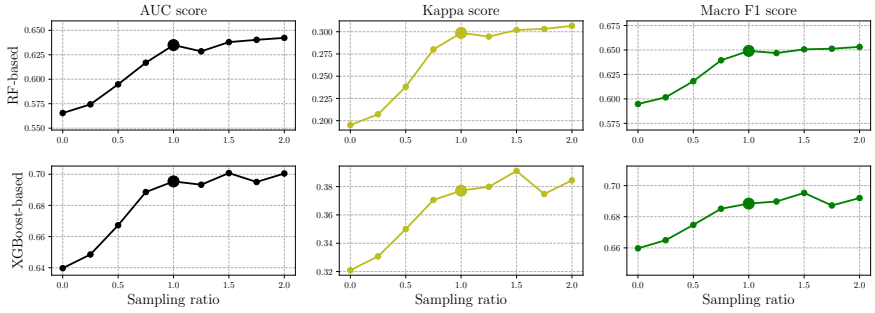
XGB(10,3). Figure 3(b) shows the evaluation results. All the results follow similar lines to the results in Figure 3(a). The main difference is that the confidence levels are, in general, lower than those by the RF. This should be taken into consideration when setting hyperparameters for RADE as we later discuss.

3.6 The use of low confidence anomalies for the training of the expert models

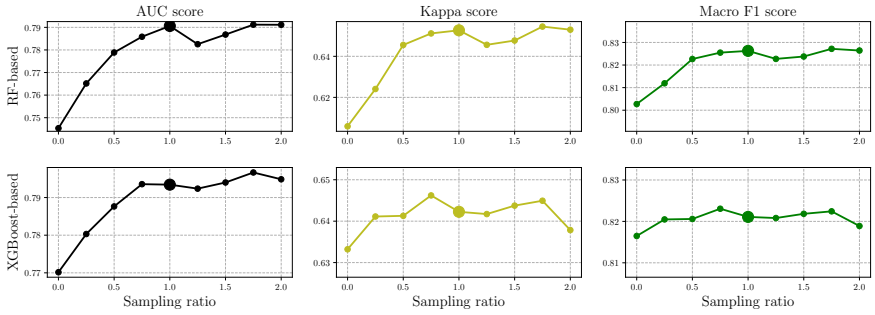
This section examines the effect of sub- and over-sampling the low-confidence anomalies for the training of RADE's expert models. Figure 4 shows how the sampling ratio affects RADE's ML scores (*i.e.,* AUC, Kappa, and Macro F1) for different datasets and both RADE flavors (RF- and XGboost-based).

Sub-sampling the low-confidence anomalies results in degraded ML scores, which aligns with intuition. That is, low-confidence anomalies form a minority set with often unique examples. Similar examples at the test data are expected to result in low confidence classification by the coarse-grained model. Thus, omitting similar examples from the training of the fine-grained models increases the possibility of misclassification. It is also evident that increasing the number of such examples (*i.e.,* oversampling) results in a limited impact on the ML performance for our DTEM based classifiers.

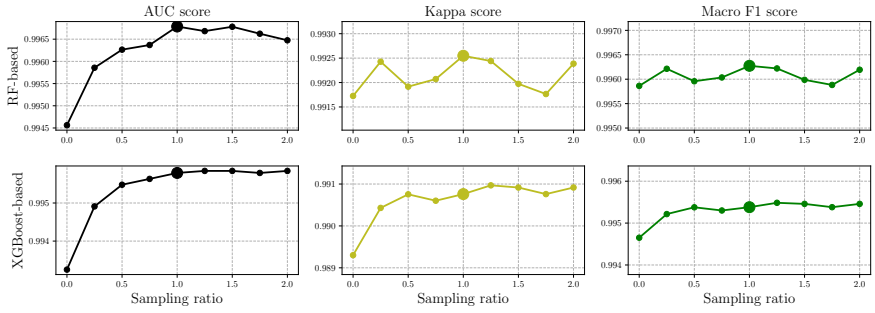
Using all the low-confidence anomalies for the training of both fine-grained models (which is equivalent to a sampling ratio of 1 in the figure) emerges as a solid operating point across all tests we conducted. Indeed, this is our default design choice for RADE.



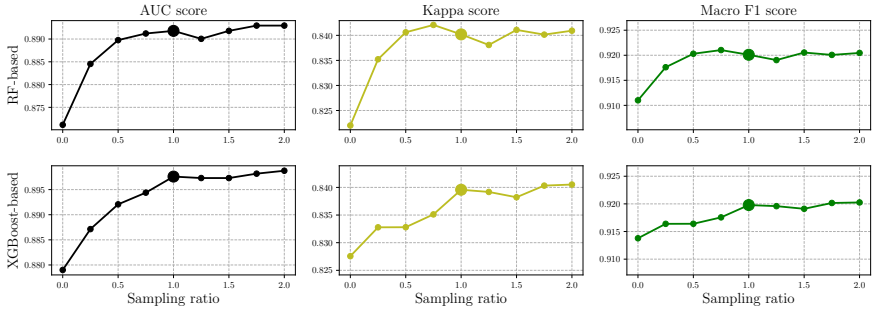
(a) communities



(b) mammography



(c) covertype



(d) creditcardfraud

Fig. 4: The effect of sub- and over-sampling the low-confidence anomalies on RADE's performance. Our design choice for RADE (i.e., sampling ratio of 1) is marked by a bigger marker.

Classifier	Dataset	Model	Macro F1	AUC	Kappa Score
RF	covertype	Monolithic	0.998±0.001	0.999±0.001	0.996±0.001
		RADE	0.996±0.001 (-0.002)	0.996±0.001 (-0.003)	0.992±0.001 (-0.004)
	kddcup99	Monolithic	0.999±0.0	0.998±0.001	0.998±0.001
		RADE	0.998±0.001 (-0.001)	0.997±0.001 (-0.002)	0.996±0.001 (-0.001)
	creditcardfraud	Monolithic	0.923±0.01	0.884±0.013	0.846±0.018
		RADE	0.92±0.008 (-0.003)	0.891±0.013 (+0.007)	0.841±0.016 (-0.006)
	seismic	Monolithic	0.502±0.013	0.509±0.007	0.031±0.023
		RADE	0.522±0.02 (+0.019)	0.52±0.012 (+0.011)	0.062±0.035 (+0.031)
	mammography	Monolithic	0.828±0.017	0.767±0.021	0.659±0.038
		RADE	0.826±0.017 (-0.002)	0.786±0.02 (+0.019)	0.653±0.034 (-0.006)
	communities	Monolithic	0.591±0.062	0.561±0.043	0.199±0.124
		RADE	0.65±0.036 (+0.059)	0.629±0.034 (+0.067)	0.301±0.072 (+0.101)
XGBoost	covertype	Monolithic	0.996±0.001	0.995±0.001	0.992±0.002
		RADE	0.994±0.001 (-0.002)	0.994±0.002 (-0.002)	0.988±0.003 (-0.004)
	kddcup99	Monolithic	0.999±0.0	0.998±0.001	0.996±0.001
		RADE	0.996±0.001 (-0.003)	0.992±0.002 (-0.006)	0.989±0.004 (-0.008)
	creditcardfraud	Monolithic	0.92±0.01	0.888±0.017	0.841±0.02
		RADE	0.92±0.01 (+0.000)	0.896±0.014 (+0.008)	0.841±0.02 (-0.000)
	seismic	Monolithic	0.508±0.019	0.512±0.01	0.041±0.033
		RADE	0.49±0.012 (-0.018)	0.503±0.006 (-0.009)	0.01±0.02 (-0.031)
	mammography	Monolithic	0.83±0.013	0.781±0.017	0.661±0.026
		RADE	0.812±0.018 (-0.018)	0.769±0.019 (-0.012)	0.625±0.035 (-0.036)
	communities	Monolithic	0.617±0.052	0.582±0.041	0.238±0.104
		RADE	0.674±0.069 (+0.057)	0.632±0.059 (+0.050)	0.351±0.136 (+0.113)

Table 3: RADE vs. monolithic DTEMs over `t2.2xlarge` AWS EC2 instance. RADE has a competitive ML performance in terms of Macro F1, AUC and Kappa scores. The values in the parenthesis show the difference between RADE and the monolithic models.

Clearly, for each specific dataset and model, one can optimize over the sampling ratio; for example, for the communities dataset, using 1.5 instead of 1 slightly improves the performance for the XGBoost-based flavor.

3.7 RADE vs. Monolithic DTEMs

Setup. We report results conducted over a general purpose environment by a `t2.2xlarge` AWS EC2 instance that provides a balance of compute and memory, with 8 vCPUs and 32 GB RAM running Ubuntu 16.04 LTS. All classifiers are configured to run on a single core for fair training and classification time comparison.

Runs. For each classifier and dataset, we perform 10 runs with different seeds and report mean and standard deviation values.

Models. We compare between our *default* RADE models to the default models of the monolithic RF and XGBoost. That is, we compare between

- RF(100,–) and RADE(RF(10,5),RF(25,20),0.8,0.9).
- XGB(100,3) and RADE(XGB(10,3),XGB(30,3),0.8,0.8).

Results. The results are summarized in Table 3 (ML scores) and Table 4 (resource consumption metrics). For both RF and XGBoost, RADE is competitive in comparison to the monolithic models in all ML scores.

In terms of resource usage, it is evident how RADE offers large savings in training time, classification time, and memory footprint. These savings become more significant as the datasets become larger. For example, for the *creditcardfraud* dataset, RADE's training time

Classifier	Dataset	Model	Training Time [sec]	Classification Time [sec]	Model Size [MB]
RF	covertype	Monolithic	8.6±0.4	0.607±0.0148	1.69±0.04
		RADE	3.0±0.0 (2.9x)	0.0736±0.0008 (8.2x)	0.39±0.03 (4.3x)
	kddcup99	Monolithic	28.9±3.0	1.3763±0.0254	0.55±0.03
		RADE	5.7±0.2 (5.1x)	0.2258±0.0015 (6.1x)	0.07±0.0 (8.2x)
	creditcardfraud	Monolithic	143.8±5.9	1.0881±0.0277	1.83±0.09
		RADE	6.9±0.1 (20.9x)	0.0655±0.0008 (16.6x)	0.14±0.01 (12.8x)
	seismic	Monolithic	0.2±0.0	0.0065±0.0002	1.9±0.03
		RADE	0.1±0.0 (3.3x)	0.0007±0.0 (8.9x)	0.27±0.02 (7.0x)
	mammography	Monolithic	0.5±0.0	0.0272±0.0008	1.52±0.07
		RADE	0.2±0.0 (3.2x)	0.0021±0.0001 (12.9x)	0.2±0.01 (7.6x)
	communities	Monolithic	0.3±0.0	0.0046±0.0001	0.4±0.01
		RADE	0.1±0.0 (4.7x)	0.0015±0.0 (3.0x)	0.08±0.0 (4.9x)
XGBoost	covertype	Monolithic	23.8±0.2	0.2628±0.0098	0.05±0.0
		RADE	5.4±0.0 (4.4x)	0.0983±0.0006 (2.7x)	0.04±0.0 (1.2x)
	kddcup99	Monolithic	63.6±2.0	0.8775±0.0327	0.06±0.0
		RADE	13.8±1.3 (4.6x)	0.4403±0.0114 (2.0x)	0.03±0.0 (2.1x)
	creditcardfraud	Monolithic	43.3±0.3	0.3027±0.0136	0.07±0.0
		RADE	6.9±0.1 (6.3x)	0.0654±0.0005 (4.6x)	0.05±0.0 (1.4x)
	seismic	Monolithic	0.1±0.0	0.0026±0.0001	0.06±0.0
		RADE	0.0±0.0 (2.6x)	0.0008±0.0 (3.3x)	0.03±0.0 (2.0x)
	mammography	Monolithic	0.3±0.0	0.0074±0.0002	0.06±0.0
		RADE	0.1±0.0 (2.2x)	0.0013±0.0001 (5.6x)	0.04±0.0 (1.5x)
	communities	Monolithic	0.5±0.0	0.0039±0.0002	0.06±0.0
		RADE	0.1±0.0 (4.1x)	0.0025±0.0001 (1.6x)	0.03±0.0 (1.9x)

Table 4: Resource consumption measurements for the evaluations in Table 3 – RADE vs. monolithic DTEMs over `t2.2xlarge` AWS EC2 instance. RADE offers significant savings in training time, classification time and memory footprint.

Classifier	Dataset	Model Size [KB]	Coarse-Grained Model Size [KB]	Fine-grained training data [M^1_{fg}, M^2_{fg}]	Fine-grained test data [M^1_{fg}, M^2_{fg}]
RF	covertype	391.01	25.86 (6.6%)	1.27%, 0.94%	0.38%, 0.6%
	kddcup99	66.80	26.88 (40.2%)	0.07%, 0.0%	0.0%, 0.0%
	creditcardfraud	143.21	31.77 (22.2%)	0.09%, 0.07%	0.03%, 0.04%
	seismic	271.94	37.95 (14.0%)	17.44%, 0.44%	6.59%, 0.07%
	mammography	198.39	35.92 (18.1%)	2.66%, 1.36%	1.16%, 0.53%
	communities	81.32	26.33 (32.4%)	6.0%, 0.0%	3.58%, 0.0%
XGBoost	covertype	44.13	8.27 (18.7%)	0.48%, 0.31%	0.24%, 0.28%
	kddcup99	27.30	7.72 (28.3%)	0.07%, 0.01%	0.07%, 0.0%
	creditcardfraud	46.12	8.13 (17.6%)	0.23%, 0.16%	0.11%, 0.14%
	seismic	28.83	7.85 (27.2%)	25.56%, 0.93%	24.88%, 0.01%
	mammography	40.21	8.05 (20.0%)	5.19%, 1.73%	4.25%, 1.12%
	communities	29.76	8.64 (29.0%)	11.57%, 0.28%	11.19%, 0.06%

Table 5: A deeper look into RADE's training and classification. It is evident how the coarse-grained model is sufficiently small to fit into the cache memory of even a small edge device. The percentage in the parenthesis show the coarse-grained size out of the total size of RADE. It is also notable how often the fractions that are passed for the training of the fine-grained models and for their classification are small.

is 6.3× faster than that of XGBoost. For the same dataset, RADE's model size is 12.8× smaller than that of RF.

We have also tried different hyperparameters for both RADE and the monolithic models and found the above results to be representative. In fact, a more careful choice of hyperparameter for RADE often leads to larger gains than reported in this paper. However, such

Classifier	Dataset	Model	Macro F1	AUC Score
RF	covertype	RADE	0.996±0.001	0.996±0.001
		RADE (coarse-grained only)	0.937±0.007	0.891±0.011
	kddcup99	RADE	0.998±0.001	0.997±0.001
		RADE (coarse-grained only)	0.997±0.001	0.994±0.001
	creditcardfraud	RADE	0.92±0.008	0.891±0.013
		RADE (coarse-grained only)	0.899±0.014	0.858±0.018
	seismic	RADE	0.522±0.02	0.52±0.012
		RADE (coarse-grained only)	0.486±0.007	0.501±0.004
	mammography	RADE	0.826±0.017	0.786±0.02
		RADE (coarse-grained only)	0.774±0.021	0.707±0.021
	communities	RADE	0.65±0.036	0.629±0.034
		RADE (coarse-grained only)	0.593±0.043	0.561±0.03
XGBoost	covertype	RADE	0.994±0.001	0.994±0.002
		RADE (coarse-grained only)	0.982±0.001	0.982±0.003
	kddcup99	RADE	0.996±0.001	0.992±0.002
		RADE (coarse-grained only)	0.985±0.002	0.972±0.003
	creditcardfraud	RADE	0.92±0.01	0.896±0.014
		RADE (coarse-grained only)	0.889±0.011	0.864±0.014
	seismic	RADE	0.49±0.012	0.503±0.006
		RADE (coarse-grained only)	0.485±0.008	0.501±0.004
	mammography	RADE	0.812±0.018	0.769±0.019
		RADE (coarse-grained only)	0.776±0.023	0.711±0.022
	communities	RADE	0.674±0.069	0.632±0.059
		RADE (coarse-grained only)	0.61±0.05	0.574±0.035

Table 6: RADE vs. coarse-grained model only. It is evident how both scores are consistently and significantly higher for RADE. This indicates that the aforementioned small fractions of test data in Table 5 that are passed to the fine-grained model are misclassified by the coarse-grained model but are correctly classified by the fine-grained models.

Classifier	Dataset	Macro F1	AUC Score	Fine-grained training data % $[M^1_{fg}, M^2_{fg}]$	Fine-grained test data % $[M^1_{fg}, M^2_{fg}]$	Training Time [sec]	Classification Time [sec]	Model Size [MB]
RF	covertype	0.997 (+0.001)	0.998 (+0.002)	100.0%, 0.96%	0.6%, 0.74%	5.7 (191.9%)	0.0745 (101.1%)	0.48 (123.5%)
	kddcup99	0.999 (+0.001)	0.997 (+0.001)	99.88%, 0.0%	0.01%, 0.0%	15.6 (272.6%)	0.2369 (104.9%)	0.16 (245.2%)
	creditcardfraud	0.92 (-0.000)	0.887 (-0.004)	99.99%, 0.18%	0.06%, 0.07%	42.2 (614.2%)	0.0649 (99.1%)	0.49 (344.8%)
	seismic	0.511 (-0.011)	0.513 (-0.006)	99.99%, 0.66%	19.54%, 0.07%	0.1 (135.2%)	0.0009 (126.6%)	0.52 (189.4%)
	mammography	0.818 (-0.009)	0.766 (-0.020)	99.94%, 2.34%	2.53%, 0.83%	0.3 (169.3%)	0.0023 (109.1%)	0.43 (217.9%)
	communities	0.614 (-0.036)	0.578 (-0.051)	99.97%, 0.0%	9.66%, 0.0%	0.1 (179.3%)	0.0015 (100.7%)	0.13 (157.2%)
XGBoost	covertype	0.994 (+0.000)	0.993 (-0.001)	99.97%, 0.99%	99.04%, 0.96%	12.4 (230.4%)	0.1989 (202.3%)	0.04 (101.7%)
	kddcup99	0.997 (+0.001)	0.995 (+0.003)	100.0%, 0.17%	99.67%, 0.16%	35.4 (257.2%)	0.5201 (118.1%)	0.03 (115.2%)
	creditcardfraud	0.917 (-0.004)	0.891 (-0.005)	99.98%, 0.19%	99.85%, 0.15%	19.5 (283.0%)	0.1387 (211.9%)	0.05 (101.6%)
	seismic	0.485 (-0.004)	0.501 (-0.002)	99.99%, 1.32%	99.85%, 0.01%	0.1 (140.5%)	0.0013 (155.9%)	0.03 (105.0%)
	mammography	0.799 (-0.013)	0.749 (-0.020)	99.91%, 2.42%	98.82%, 1.18%	0.2 (152.1%)	0.0031 (230.3%)	0.04 (102.7%)
	communities	0.64 (-0.034)	0.598 (-0.034)	99.99%, 0.29%	99.28%, 0.06%	0.2 (178.6%)	0.0033 (132.1%)	0.03 (103.0%)

Table 7: Comparing the default RADE configurations to RADE configurations with an addition of 0.1 to the CCT and TCT values (*i.e.,* , to RADE(RF(10,5),RF(25,20),0.9,1.0) and RADE(XGB(10,3),XGB(30,3),0.9,0.9)). The values in the parenthesis show the difference between these configurations and the default ones. It is evident how the further increase in the thresholds show similar performance in terms of Macro F1 and AUC scores while consuming considerably more resources.

careful tuning is out of the scope and motivation of our paper since we target training in resource-constrained settings.

3.8 Taking a deeper look into RADE

We rerun the experiments but now take a deeper look into RADE's training and classification.

In Table 5 we summarize the results of the memory footprint of the coarse-grained model in comparison to the entire memory footprint of RADE. We also measure the fractions of the training dataset that is used for the training of each fine-grained model and the fractions that are passed to each during the classification of the test dataset.

It is evident how the coarse-grained model is sufficiently small and hence can fit into the cache memory of even a small edge device, requiring no more than 40 KB. This further contribute to the improvement of the classification latency, since many expensive cache misses are alleviated for most of the instances. Also, the data fractions that are passed to the fine-grained models for both training and classification are small. Indeed, this is expected according to the previous analysis in §3.5 showing that the coarse-grained model manages to handle the absolute majority of queries with a sufficiently high confidence.

Now we take a deeper look into the ML performance in terms of F1 and AUC scores of RADE in comparison to its coarse-grained model – that is, by allowing the coarse-grained model to classify all queries disregarding their confidence. Formally, we compare between:

- RF(10,5) and RADE(RF(10,5),RF(25,20),0.8,0.9).
- XGB(10,3) and RADE(XGB(10,3),XGB(30,3),0.8,0.8).

The results are summarized in Table 6. It is evident how both scores are consistently and significantly higher for RADE. Indeed, this means that the aforementioned small fractions of test data in Table 5 that are passed to the fine-grained models are misclassified by the coarse-grained model but are correctly classified by the fine-grained models that are trained to be experts for such particular situations.

Finally, we look at how a different choice of hyperparameters affects RADE's performance. Specifically, we use our default RADE configuration and a RADE model with +0.1 to the default CCT and TCT values. That is, we compare between:

- RADE(RF(10,5),RF(25,20),0.8,0.9)
  and
  RADE(RF(10,5),RF(25,20),0.9,1.0)

- RADE(XGB(10,3),XGB(30,3),0.8,0.8)
  and
  RADE(XGB(10,3),XGB(30,3),0.9,0.9)

The results are depicted in Table 7.

Increasing the default CCT and TCT results in bigger subsets of the dataset used by fine-grained models during training and forwarded to them during classification. As a result, an increase in the classification and training times is expected, while the Macro F1 and AUC scores might be increased since the fine-grained models handle a higher fraction of the dataset. However, our evaluation demonstrates how such a further increase in the thresholds results in similar Macro F1 and AUC scores while consuming considerably more resources.

Indeed, we found our default configuration to be favorable over a variety of tests. It offers a good balance between the coarse-grained and the fine-grained models, such that RADE achieves competitive Macro F1 and AUC scores with relatively low resource consumption.

Classifier	Dataset	Model	Training Time [sec]	Classification Latency [sec]	Model Size [MB]
RF	covertype	Monolithic	140.1±7.2	4.6534±0.1354	1.7±0.05
		RADE	29.1±0.3 (4.8x)	0.5636±0.0061 (8.3x)	0.38±0.03 (4.4x)
	kddcup99	Monolithic	47.3±0.8	2.0941±0.0208	0.62±0.02
		RADE	6.4±0.8 (7.3x)	0.2696±0.0069 (7.8x)	0.06±0.01 (10.6x)
	creditcardfraud	Monolithic	1045.7±43.9	8.0629±0.184	1.82±0.09
		RADE	51.1±0.2 (20.5x)	0.4813±0.0021 (16.8x)	0.14±0.01 (13.4x)
	seismic	Monolithic	1.6±0.0	0.0563±0.0006	1.88±0.04
		RADE	0.5±0.1 (3.0x)	0.0067±0.0002 (8.4x)	0.27±0.02 (7.1x)
	mammography	Monolithic	3.3±0.1	0.1898±0.0019	1.51±0.07
		RADE	1.2±0.0 (2.8x)	0.0155±0.0002 (12.3x)	0.19±0.01 (7.9x)
	communities	Monolithic	1.7±0.0	0.0391±0.0011	0.38±0.01
		RADE	0.5±0.0 (3.7x)	0.0113±0.0003 (3.4x)	0.08±0.0 (5.0x)
XGBoost	covertype	Monolithic	459.3±5.9	1.7061±0.0251	0.07±0.0
		RADE	54.0±0.1 (8.5x)	0.8791±0.0043 (1.9x)	0.05±0.0 (1.5x)
	kddcup99	Monolithic	105.0±2.6	0.6846±0.0317	0.06±0.0
		RADE	9.5±1.0 (11.1x)	0.2794±0.0102 (2.4x)	0.02±0.0 (3.7x)
	creditcardfraud	Monolithic	1147.6±12.9	2.2038±0.0405	0.12±0.0
		RADE	83.2±0.1 (13.8x)	0.5758±0.0034 (3.8x)	0.04±0.0 (2.7x)
	seismic	Monolithic	1.3±0.0	0.0204±0.0003	0.13±0.0
		RADE	0.2±0.0 (5.1x)	0.0047±0.0002 (4.4x)	0.03±0.0 (4.1x)
	mammography	Monolithic	3.0±0.0	0.0721±0.0017	0.12±0.0
		RADE	0.8±0.0 (3.9x)	0.0084±0.0001 (8.6x)	0.05±0.0 (2.7x)
	communities	Monolithic	2.6±0.1	0.0229±0.0005	0.05±0.0
		RADE	0.5±0.0 (5.2x)	0.0176±0.0004 (1.3x)	0.03±0.0 (1.6x)

Table 8: Evaluation over Raspberry Pi 3 B+ device.

3.9 Evaluation over other environments

We evaluate RADE over other environments to cover different processors, memory sizes and operating systems. More specifically, in the following we compare RADE to the monolithic DTEMs over:
(1) An edge-device environment by Raspberry Pi 3 Model B Plus Rev 1.3 with 1-GB RAM running Ubuntu 20.04 LTS; and
(2) A cost-effective environment by a `a1.medium` AWS EC2 instance with single vCPU (a custom built AWS Graviton Processor with 64-bit Arm Neoverse core) and 2-GB memory, running Amazon Linux 2.

Tables 8 and 9 compare the training time, classification time and memory size of RADE and the monolithic models, for the same DTEMs and datasets, as in Tables 3 and 4. As can be seen, the resource savings by RADE over these environments repeat the same trend as presented in Table 4. Again, it is evident how the resource savings gap increases for the larger datasets (*i.e., covertype*, *kddcup99*, and *creditcardfraud*).

To summarize, RADE achieves resource savings on different environments, including general-purpose, cost-saving and edge-device.

4 Related Work

We next overview related work, including anomaly detection over edge-devices, DTEM optimizations techniques, related ML models and, finally general ML optimization techniques. For comprehensive anomaly detection techniques survey beyond our DTEM scope the reader is referred to [16, 89].

Classifier	Dataset	Model	Training Time [sec]	Classification Latency [sec]	Model Size [MB]
RF	covertype	Monolithic	23.0±1.2	0.8415±0.0193	1.7±0.06
		RADE	8.5±0.1 (2.7x)	0.1086±0.0016 (7.7x)	0.39±0.03 (4.3x)
	kddcup99	Monolithic	121.0±6.2	3.1705±0.0668	0.66±0.04
		RADE	18.7±0.2 (6.5x)	0.4662±0.0024 (6.8x)	0.07±0.0 (9.5x)
	creditcardfraud	Monolithic	268.5±11.7	1.4155±0.0325	1.82±0.1
		RADE	15.1±0.1 (17.8x)	0.0945±0.0004 (15.0x)	0.14±0.01 (12.7x)
	seismic	Monolithic	0.5±0.0	0.0119±0.0002	1.89±0.05
		RADE	0.2±0.0 (2.8x)	0.0014±0.0 (8.5x)	0.27±0.02 (7.0x)
	mammography	Monolithic	1.1±0.0	0.0421±0.0008	1.51±0.07
		RADE	0.5±0.0 (2.3x)	0.0038±0.0001 (11.1x)	0.2±0.01 (7.6x)
	communities	Monolithic	0.7±0.0	0.0074±0.0001	0.39±0.01
		RADE	0.2±0.0 (3.7x)	0.002±0.0 (3.6x)	0.08±0.0 (4.8x)
XGBoost	covertype	Monolithic	64.0±0.9	0.514±0.0106	0.07±0.0
		RADE	12.3±0.0 (5.2x)	0.26±0.0014 (2.0x)	0.05±0.0 (1.5x)
	kddcup99	Monolithic	206.9±6.0	2.0517±0.0591	0.06±0.0
		RADE	34.3±4.8 (6.0x)	0.8053±0.011 (2.5x)	0.03±0.01 (2.3x)
	creditcardfraud	Monolithic	156.4±3.8	0.8468±0.0141	0.12±0.0
		RADE	16.6±0.1 (9.4x)	0.1652±0.0032 (5.1x)	0.04±0.0 (2.9x)
	seismic	Monolithic	0.3±0.0	0.0081±0.0001	0.13±0.0
		RADE	0.1±0.0 (4.4x)	0.0013±0.0 (6.1x)	0.03±0.0 (4.1x)
	mammography	Monolithic	0.7±0.0	0.0277±0.0005	0.13±0.0
		RADE	0.3±0.0 (2.4x)	0.0026±0.0 (10.5x)	0.04±0.0 (2.9x)
	communities	Monolithic	0.8±0.0	0.0071±0.0002	0.05±0.0
		RADE	0.2±0.0 (5.1x)	0.0048±0.0001 (1.5x)	0.03±0.01 (1.7x)

Table 9: Evaluation over `a1.medium` AWS EC2 instance.

4.1 Anomaly detection over edge-devices

Several works target anomaly detection over edge devices. For instance, [91] introduces the importance of anomaly detection for edge health-care analytics. [68] presents SVELTE, an intrusion detection solution for edge devices, specifically designed for securing 6LoWPAN networks.

These works present a ML-based solution for a specific use-case, whereas in RADE we aim at a resource-efficient framework that supports many use-cases (as we demonstrate via the evaluation) and that can augment any DTEM classifier for better resource efficiency.

4.2 DTEM optimizations

There exist several approaches that aim at optimizing the memory layout of bagging DTEMs and in particular RFs. For instance, in [92] proposes achieving deterministic classification time by constructing random forests composed of many small trees rather than fewer deep trees. Some studies, *e.g.,* [2, 10] optimize memory-layouts of RFs, which, in turn, reduces the cache misses and accelerates classification time.

Much effort has been made to address the drawbacks of boosting methods as well. For instance, advanced implementations of the tree-based gradient boosting idea include: XGBoost [18] that supports parallelism and uses pre-sorted and histogram-based algorithms for computing the best split; LightGBM [43] that uses Gradient-based One-Side Sampling (GOSS) to filter out the data instances for finding a split value; CatBoost [64] that implements *ordered boosting*, a permutation-driven alternative to the classic algorithm and an innovative algorithm for processing categorical features.

For Adaptive boosting (*i.e.,* AdaBoost), some recent approaches targeting to accelerate its slow training [19, 60, 77]. For instance, [77] uses a new sampling strategy (WNS) that

selects a subset of the data at each iteration and by that reducing the number of data points onto which AdaBoost is applied.

Both the aforementioned bagging and boosting optimization techniques are orthogonal to RADE and can be used as building blocks for RADE's coarse-grained and fine-grained models to form even less resource consuming models (*e.g.,* in our evaluation, we use XGBoost and RF as building blocks).

4.3 Related ML models

Cascading. Cascading models are mainly used for object detection (*e.g.,* image, video) with increasing complexity feature extraction and evaluation stages (*i.e.,* DNN) [27, 79, 93]. Recently, such methods were proposed for ranking [46], anomaly detection [58], medicine [41, 95] and more. The main target of cascading models is fast classification. The price for this speed is usually a resource-consuming and long training as well as increased memory footprint (in a cascading model, many models are built, and numerous parameters are jointly optimized towards an optimization target).

Conversely, RADE's design, while it is in the spirit of a cascading model, concentrates not only on fast classification but also on low memory footprint and fast training resulting in the ability to run such models on resource-constrained devices.

Stacking. Stacking [7, 35, 90, 96] is an ensemble method that can also be viewed as a cascading model. It mainly differs from bagging and boosting by considering heterogeneous weak learners and combining them using a meta-model.

RADE inherently differs from the stacking approach, since RADE's fine-grained models are trained by fractions of the original dataset, while in stacking, the $i+1$'th model is trained using the classification output of the i'th model.

DNN approaches. Some ML works introduce approaches that aim at reducing certain resource consumption in the context of DNNs. BranchyNet [87] presents a deep neural network (DNN) architecture with an additional side branch classifiers, such that some queries can be correctly classified by early layers. This work mainly targets fast classification. [44] presents partitioning of a deep network between the edge and host platforms, such that it aims at the tradeoff between the required compute over the edge device and the required throughput between the edge and the host. [88] presents a similar hierarchical Neural Network approach, with the additional support for aggregating information from different edge devices to the cloud. RADE, on the other hand, allows employing the entire model over the edge-device, without the need for additional remote host/cloud computation.

[75] presents an approach to anomaly detection that uses auto-encoders, deployed on each edge device, to perform analytics and identify anomalous observations in a distributed fashion. A centralized server aggregates the updated models and distributes them back to the edge devices when a connection is available. However, this architecture requires bandwidth and connectivity between the edge devices and a central server which is used also for training synchronization. RADE, on the other hand, targets local training.

Delegation. [28] presented the concept of delegating classifiers and is mostly related to our work. Indeed, there are similarities between RADE and the general structure of the method proposed by [28]. Nevertheless, in addition to many technical details that target resource efficient anomaly detection by DTEMs, there are also few key conceptual differences between RADE and [28]: (1) We use two expert models on the same "delegation" level and not a single one as suggested by [28]. Each of these experts is responsible for capturing different concepts "delegated" by the small model. This is tailored for the binary class problem that

we consider; (2) [28] consider a strict delegation rule. To quote: *"The same threshold is used for training and for prediction (the delegating decision rule). The question arises on how to determine this threshold."*. We found this to be extremely limiting and often insufficient for our use-case. RADE introduces two thresholds, for training and classification (TCT and CCT); (3) The delegation rule suggested by [28] delegates a certain *percentage* of the harder labeled instances (this is also the case for the imbalanced case). This causes classification inconsistency of the resulting classifier. In particular, training the classifier with the same hyperparameter, the same data, and the same random seed still results in different classification results. This happens, for example, when training data is introduced in batches. RADE, on the other, results in consistent classification results.

4.4 General ML optimizations

Feature selection. Feature selection [23, 47, 85] methods are used to select a subset of features that are sufficient for classification. Feature selection may even improve accuracy via noise reduction as well as to reduce the training time by decreasing the size of the dataset. RADE is orthogonal to feature selection and can leverage it to meet specific system requirements (*e.g.,* further reducing the memory footprint of the coarse-grained model).
Sampling. A common technique to reduce the training complexity is to use progressive sampling [49, 62, 65]. These methods systematically increase the sample size until specific performance criteria are met. Additionally, stratified sampling techniques [12, 17, 98] can be employed to that end. For example, Instance hardness (IH) was recently presented [15,40,81–83,94,99] that mainly targets to alleviate the class imbalance problem. Such sampling techniques are orthogonal to any classifier, and hence applicable to RADE as well by subsampling the entire training data before training a RADE model or only filtered subsets of the datasets when training RADE's fine-grained models.

5 Conclusions and Future Work

This paper presents RADE, a resource efficient DTEM anomaly detection framework. RADE augments standard monolithic DTEM classifiers. System-wise, RADE improves the model memory footprint (by up to $12\times$), training time (by up to $20\times$) and classification time (by up to $16\times$), as compared to the monolithic models. ML-wise, RADE offers competitive anomaly detection capabilities.

Due to its properties, RADE is an appealing fit for employing ML-based anomaly detection in resource-constrained settings. RADE alleviates the memory-bound problem of bagging methods, and the compute-bound problem of the boosting methods. More specifically, the size of RADE's coarse-grained model (up to 40KB in our evaluation) allows it to be entirely stored in the cache, and serve most of the classification queries. Furthermore, the reduction of the training and classification times by RADE is an indication that it requires less compute and memory operations. In turn, this suggests that RADE requires less power [33], which is important for battery-based or power-constrained devices.

To conclude, we point out several important directions for future work. We have tested the RADE framework on three different platforms and over two different DTEM classifiers. Nevertheless, it is of interest to test RADE over other system environments and real-world scenarios. Furthermore, this paper demonstrates how RADE is built using either a random

forest or an XGBoost as a building block. However, other DTEMs, including boosted random forests [56], can be used and evaluated as building blocks as well. Such a broader investigation may introduce or reveal new challenges that are not captured by our evaluation. Finally, while we consider a static scenario where the training data is available offline, it is of interest to investigate whether the RADE approach can include dynamic updates, such as done by [84] for streaming scenarios.

6 Declarations

Funding

This work was partly supported by the Israel Science Foundation (grant No. 1119/19), the Hasso Plattner Institute Research School, the Technion Hiroshi Fujiwara Cyber Security Research Center, and the Israel Cyber Bureau.

Conflict of interest/Competing interests

The authors declare that they have no conflict of interest.

Availability of data and material

The data used for the simulation results are available online, see Section 3.3 for more details.

Code availability

The code is available in [78].

References

1. Appel, R., Fuchs, T., Dollár, P., Perona, P.: Quickly boosting decision trees–pruning underachieving features early. In: International conference on machine learning, pp. 594–602 (2013)
2. Asadi, N., Lin, J., De Vries, A.P.: Runtime optimizations for tree-based machine learning models. IEEE Transactions on Knowledge and Data Engineering **26**(9), 2281–2292 (2014)
3. Ashfaq, R.A.R., Wang, X.Z., Huang, J.Z., Abbas, H., He, Y.L.: Fuzziness based semi-supervised learning approach for intrusion detection system. Information Sciences **378**, 484–497 (2017)
4. Banerjee, M., Ding, Y., Noone, A.M.: Identifying representative trees from ensembles. Statistics in medicine **31**(15), 1601–1616 (2012)
5. Blackard, J.A.: Comparison of neural networks and discriminant analysis in predicting forest cover types. (2000)
6. Breiman, L.: Arcing classifiers (technical report). University of California, Department of Statistics (1996)
7. Breiman, L.: Stacked regressions. Machine learning **24**(1), 49–64 (1996)
8. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
9. Bridges, S.M., Vaughn, R.B., et al.: Fuzzy data mining and genetic algorithms applied to intrusion detection. In: Proceedings of 12th Annual Canadian Information Technology Security Symposium, pp. 109–122 (2000)
10. Browne, J., Tomita, T., Mhembere, D., Burns, R., Vogelstein, J.: Forest packing: Fast, parallel decision forests. arXiv preprint arXiv:1806.07300 (2018)

11. Buitinck, L., Louppe, G., Blondel, M., et al.: API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pp. 108–122 (2013)
12. Bunkhumpornpat, C., Sinapiromsaran, K., Lursinsap, C.: Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem. In: Pacific-Asia conference on knowledge discovery and data mining, pp. 475–482. Springer (2009)
13. Carcillo, F., Dal Pozzolo, A., Le Borgne, Y.A., Caelen, O., Mazzer, Y., Bontempi, G.: Scarff : a scalable framework for streaming credit card fraud detection with spark. Information Fusion **41** (2017). DOI 10.1016/j.inffus.2017.09.005
14. Carcillo, F., Le Borgne, Y.A., Caelen, O., Kessaci, Y., Oblé, F., Bontempi, G.: Combining unsupervised and supervised learning in credit card fraud detection. Information Sciences (2019). DOI 10.1016/j.ins.2019.05.042
15. Cavalcanti, G.D., Soares, R.J.: Ranking-based instance selection for pattern classification. Expert Systems with Applications **150**, 113269 (2020)
16. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM computing surveys (CSUR) **41**(3), 1–58 (2009)
17. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: synthetic minority over-sampling technique. Journal of artificial intelligence research **16**, 321–357 (2002)
18. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pp. 785–794. ACM (2016)
19. Chu, F., Zaniolo, C.: Fast and light boosting for adaptive mining of data streams. In: Pacific-Asia conference on knowledge discovery and data mining, pp. 282–292. Springer (2004)
20. Cohen, J.: A coefficient of agreement for nominal scales. Educational and psychological measurement **20**(1), 37–46 (1960)
21. Dal Pozzolo, A.: Adaptive machine learning for credit card fraud detection (2015)
22. Dal Pozzolo, A., Caelen, O., Johnson, R., Bontempi, G.: Calibrating probability with undersampling for unbalanced classification (2015). DOI 10.1109/SSCI.2015.33
23. Dash, M., Liu, H.: Feature selection for classification. Intelligent data analysis **1**(1-4), 131–156 (1997)
24. Dhanabal, L., Shantharajah, S.: A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. International Journal of Advanced Research in Computer and Communication Engineering **4**(6), 446–452 (2015)
25. Dietterich, T.G.: An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. Machine learning **40**(2), 139–157 (2000)
26. Dua, D., Graff, C.: UCI machine learning repository (2017). URL `http://archive.ics.uci.edu/ml`
27. Felzenszwalb, P.F., Girshick, R.B., McAllester, D.: Cascade object detection with deformable part models. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 2241–2248. IEEE (2010)
28. Ferri, C., Flach, P., Hernández-Orallo, J.: Delegating classifiers. In: Proceedings of the twenty-first international conference on Machine learning, p. 37 (2004)
29. Freund, Y., Schapire, R.E., et al.: Experiments with a new boosting algorithm. In: icml, vol. 96, pp. 148–156. Citeseer (1996)
30. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
31. Gama, J., Rocha, R., Medas, P.: Accurate decision trees for mining high-speed data streams. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 523–528. ACM (2003)
32. Gan, X.s., Duanmu, J.s., Wang, J.f., Cong, W.: Anomaly intrusion detection based on pls feature extraction and core vector machine. Knowledge-Based Systems **40**, 1–6 (2013)
33. García-Martín, E., Rodrigues, C.F., Riley, G., Grahn, H.: Estimation of energy consumption in machine learning. Journal of Parallel and Distributed Computing **134**, 75–88 (2019)
34. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Machine learning **63**(1), 3–42 (2006)
35. Graczyk, M., Lasota, T., Trawiński, B., Trawiński, K.: Comparison of bagging, boosting and stacking ensembles applied to real estate appraisal. In: Asian conference on intelligent information and database systems, pp. 340–350. Springer (2010)
36. Habeeb, R.A.A., Nasaruddin, F., Gani, A., Hashem, I.A.T., Ahmed, E., Imran, M.: Real-time big data processing for anomaly detection: A survey. International Journal of Information Management (2018)
37. Hasan, M.A.M., Nasser, M., Pal, B., Ahmad, S.: Support vector machine and random forest modeling for intrusion detection system (ids). Journal of Intelligent Learning Systems and Applications **6**(01), 45 (2014)

38. Hastie, T., Tibshirani, R., Friedman, J.: Unsupervised learning. In: The elements of statistical learning, pp. 485–585. Springer (2009)
39. Jin, Y., Chuang, J.: Medium corporation. tree Boosting with XGBoost. Why Does XGBoost Win Every Machine Learning Competition? `https://bit.ly/2TVWit1` (2017). [Online; accessed 7-September-2019]
40. Kabir, A., Ruiz, C., Alvarez, S.A.: Mixed bagging: A novel ensemble learning framework for supervised classification based on instance hardness. In: 2018 IEEE International Conference on Data Mining (ICDM), pp. 1073–1078. IEEE (2018)
41. Karegowda, A.G., Jayaram, M., Manjunath, A.: Cascading k-means clustering and k-nearest neighbor classifier for categorization of diabetic patients. International Journal of Engineering and Advanced Technology **1**(3), 147–151 (2012)
42. Kayacik, H.G., Zincir-Heywood, A.N., Heywood, M.I.: Selecting features for intrusion detection: A feature relevance analysis on kdd 99 intrusion detection datasets. In: Proceedings of the third annual conference on privacy, security and trust, vol. 94, pp. 1723–1722 (2005)
43. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. In: Advances in Neural Information Processing Systems, pp. 3146–3154 (2017)
44. Ko, J.H., Na, T., Amir, M.F., Mukhopadhyay, S.: Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. arXiv preprint arXiv:1802.03835 (2018)
45. Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. In: Proceedings of the 10th ACM conference on Computer and communications security, pp. 251–261. ACM (2003)
46. Kveton, B., Szepesvari, C., Wen, Z., Ashkan, A.: Cascading bandits: Learning to rank in the cascade model. In: International Conference on Machine Learning, pp. 767–776 (2015)
47. Kwak, N., Choi, C.H.: Input feature selection for classification problems. IEEE transactions on neural networks **13**(1), 143–159 (2002)
48. Lebichot, B., Le Borgne, Y.A., He, L., Oblé, F., Bontempi, G.: Deep-Learning Domain Adaptation Techniques for Credit Cards Fraud Detection, pp. 78–88 (2020). DOI 10.1007/978-3-030-16841-4_8
49. Leite, R., Brazdil, P.: Improving progressive sampling via meta-learning on learning curves. In: European Conference on Machine Learning, pp. 250–261. Springer (2004)
50. Li, M., Andersen, D.G., Park, J.W., Smola, A.J., Ahmed, A., Josifovski, V., Long, J., Shekita, E.J., Su, B.Y.: Scaling distributed machine learning with the parameter server. In: OSDI, vol. 14, pp. 583–598 (2014)
51. Liaw, A., Wiener, M., et al.: Classification and regression by randomforest. R news **2**(3), 18–22 (2002)
52. Liu, S., Xiao, J., Liu, J., Wang, X., Wu, J., Zhu, J.: Visual diagnosis of tree boosting methods. IEEE transactions on visualization and computer graphics **24**(1), 163–173 (2017)
53. Machine Learning Group - ULB: Credit Card Fraud Detection Dataset. `https://www.kaggle.com/isaikumar/creditcardfraud` (2013). [Online; accessed 26-August-2019]
54. Machine Learning Repository - UCI: Forest Cover Type Dataset. `https://archive.ics.uci.edu/ml/datasets/Covertype` (1998). [Online; accessed 26-August-2019]
55. Machine Learning Repository - UCI: KDD Cup 1999 Dataset. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html` (1999). [Online; accessed 26-August-2019]
56. Mishina, Y., Murata, R., Yamauchi, Y., Yamashita, T., Fujiyoshi, H.: Boosted random forest. IEICE Transactions on Information and systems **98**(9), 1630–1636 (2015)
57. Moustafa, N., Hu, J., Slay, J.: A holistic review of network anomaly detection systems: A comprehensive survey. Journal of Network and Computer Applications (2018)
58. Muniyandi, A.P., Rajeswari, R., Rajaram, R.: Network anomaly detection by cascading k-means clustering and c4. 5 decision tree algorithm. Procedia Engineering **30**, 174–182 (2012)
59. Obradovic, Z., Vucetic, S.: Challenges in scientific data mining: Heterogeneous, biased, and large samples. Tech. rep., Citeseer (2004)
60. Olson, M.: Jousboost: An r package for improving machine learning classifier probability estimates (2017)
61. Oza, N.C., Russell, S.: Experimental comparisons of online and batch versions of bagging and boosting. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 359–364. ACM (2001)
62. Parthasarathy, S.: Efficient progressive sampling for association rules. In: 2002 IEEE International Conference on Data Mining, 2002. Proceedings., pp. 354–361. IEEE (2002)
63. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine Learning in Python . Journal of Machine Learning Research **12**, 2825–2830 (2011)

64. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., Gulin, A.: Catboost: unbiased boosting with categorical features. In: Advances in Neural Information Processing Systems, pp. 6638–6648 (2018)
65. Provost, F., Jensen, D., Oates, T.: Efficient progressive sampling. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 23–32. ACM (1999)
66. PyPI, XGBoost: XGBoost. https://pypi.org/project/xgboost/ (2018). [Online; accessed 7-September-2019]
67. Ravanshad, A.: Medium corporation. gradient boosting vs random forest. https://bit.ly/2lT5IZ4 (2018). [Online; accessed 7-September-2019]
68. Raza, S., Wallgren, L., Voigt, T.: Svelte: Real-time intrusion detection in the internet of things. Ad hoc networks **11**(8), 2661–2674 (2013)
69. Resende, P.A.A., Drummond, A.C.: A survey of random forest based methods for intrusion detection systems. ACM Computing Surveys (CSUR) **51**(3), 48 (2018)
70. Right to explanation: Wikipedia, the free encyclopedia (2019). URL https://en.wikipedia.org/wiki/Right_to_explanation. [Online; accessed 26-August-2019]
71. Roe, B.P., Yang, H.J., Zhu, J.: Boosted decision trees, a powerful event classifier. In: Statistical problems in particle physics, astrophysics and cosmology, pp. 139–142. World Scientific (2006)
72. Roe, B.P., Yang, H.J., Zhu, J., Liu, Y., Stancu, I., McGregor, G.: Boosted decision trees as an alternative to artificial neural networks for particle identification. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **543**(2-3), 577–584 (2005)
73. Sabhnani, M., Serpen, G.: Application of machine learning algorithms to kdd intrusion detection dataset within misuse detection context. In: MLMTA, pp. 209–215 (2003)
74. Schapire, R.E.: The boosting approach to machine learning: An overview. In: Nonlinear estimation and classification, pp. 149–171. Springer (2003)
75. Schneible, J., Lu, A.: Anomaly detection on the edge. In: MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM), pp. 678–682. IEEE (2017)
76. Scikit - RF: sklearn.ensemble.randomforestclassifier. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier (2019). [Online; accessed 7-September-2019]
77. Seyedhosseini, M., Paiva, A.R., Tasdizen, T.: Fast adaboost training using weighted novelty selection. In: The International Joint Conference on Neural Networks, pp. 1245–1250. IEEE (2011)
78. Shay Vargaftik, Yaniv Ben-Itzhak: RADE's code. https://research.vmware.com/projects/efficient-machine-learning-classification (2019)
79. Shen, H., Han, S., Philipose, M., Krishnamurthy, A.: Fast video classification via adaptive cascading of deep models. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3646–3654 (2017)
80. Singh, K., Guntuku, S.C., Thakur, A., Hota, C.: Big data analytics framework for peer-to-peer botnet detection using random forests. Information Sciences **278**, 488–497 (2014)
81. Sleeman IV, W.C., Krawczyk, B.: Bagging using instance-level difficulty for multi-class imbalanced big data classification on spark. In: 2019 IEEE International Conference on Big Data (Big Data), pp. 2484–2493. IEEE (2019)
82. Smith, M.R., Martinez, T.: A comparative evaluation of curriculum learning with filtering and boosting in supervised classification problems. Computational Intelligence **32**(2), 167–195 (2016)
83. Smith, M.R., Martinez, T., Giraud-Carrier, C.: An instance level analysis of data complexity. Machine learning **95**(2), 225–256 (2014)
84. Tan, S.C., Ting, K.M., Liu, T.F.: Fast anomaly detection for streaming data. In: Twenty-Second International Joint Conference on Artificial Intelligence (2011)
85. Tang, J., Alelyani, S., Liu, H.: Feature selection for classification: A review. Data classification: Algorithms and applications p. 37 (2014)
86. Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.: A detailed analysis of the kdd cup 99 data set. In: Symposium on Computational Intelligence for Security and Defense Applications (CISDA), pp. 1–6. IEEE (2009)
87. Teerapittayanon, S., McDanel, B., Kung, H.: Branchynet: Fast inference via early exiting from deep neural networks. In: Pattern Recognition (ICPR), 2016 23rd International Conference on, pp. 2464–2469. IEEE (2016)
88. Teerapittayanon, S., McDanel, B., Kung, H.: Distributed deep neural networks over the cloud, the edge and end devices. In: Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on, pp. 328–339. IEEE (2017)

89. Thudumu, S., Branch, P., Jin, J., Singh, J.J.: A comprehensive survey of anomaly detection techniques for high dimensional big data. Journal of Big Data **7**(1), 1–30 (2020)
90. Ting, K.M., Witten, I.H.: Stacking bagged and dagged models (1997)
91. Ukil, A., Bandyoapdhyay, S., Puri, C., Pal, A.: Iot healthcare analytics: The importance of anomaly detection. In: Advanced Information Networking and Applications (AINA), 2016 IEEE 30th International Conference on, pp. 994–997. IEEE (2016)
92. Van Essen, B., Macaraeg, C., Gokhale, M., Prenger, R.: Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga? In: 20th International Symposium on Field-Programmable Custom Computing Machines, pp. 232–239. IEEE (2012)
93. Viola, P., Jones, M., et al.: Rapid object detection using a boosted cascade of simple features. CVPR (1) **1**(511-518), 3 (2001)
94. Walmsley, F.N., Cavalcanti, G.D., Oliveira, D.V., Cruz, R.M., Sabourin, R.: An ensemble generation method based on instance hardness. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2018)
95. Wang, Y., Patrick, J.: Cascading classifiers for named entity recognition in clinical notes. In: Proceedings of the workshop on biomedical information extraction, pp. 42–49. Association for Computational Linguistics (2009)
96. Wolpert, D.H.: Stacked generalization. Neural networks **5**(2), 241–259 (1992)
97. Xuan, S., Liu, G., Li, Z., Zheng, L., Wang, S., Jiang, C.: Random forest for credit card fraud detection. In: Networking, Sensing and Control (ICNSC), 2018 IEEE 15th International Conference on, pp. 1–6. IEEE (2018)
98. Yen, S.J., Lee, Y.S.: Cluster-based under-sampling approaches for imbalanced data distributions. Expert Systems with Applications **36**(3), 5718–5727 (2009)
99. Zhang, J., Wang, T., Ng, W.W., Zhang, S., Nugent, C.D.: Undersampling near decision boundary for imbalance problems. In: 2019 International Conference on Machine Learning and Cybernetics (ICMLC), pp. 1–8. IEEE (2019)
100. Zhang, J., Zulkernine, M.: Network intrusion detection using random forests. In: PST. Citeseer (2005)
101. Zhao, Z., Mehrotra, K.G., Mohan, C.K.: Online anomaly detection using random forest. In: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, pp. 135–147. Springer (2018)
102. Zheng, Z., Cai, Y., Li, Y.: Oversampling method for imbalanced classification. Computing and Informatics **34**(5), 1017–1037 (2016)