arXiv:2003.05731v4 [cs.LG] 5 Mar 2021

SUOD: Accelerating Large-Scale Unsupervised Heterogeneous Outlier Detection

Yue Zhao [*1] **Xiyang Hu** [*1] **Cheng Cheng** [1] **Cong Wang** [2] **Changlin Wan** [3] **Wen Wang** [1] **Jianing Yang** [1] **Haoping Bai** [1] **Zheng Li** [4] **Cao Xiao** [5] **Yunlong Wang** [5] **Zhi Qiao** [5] **Jimeng Sun** [6] **Leman Akoglu** [1]

Abstract

Outlier detection (OD) is a key machine learning (ML) task for identifying abnormal objects from general samples with numerous high-stake applications including fraud detection and intrusion detection. Due to the lack of ground truth labels, practitioners often have to build a large number of unsupervised, heterogeneous models (i.e., different algorithms with varying hyperparameters) for further combination and analysis, rather than relying on a single model. How to *accelerate* the training and scoring on new-coming samples by outlyingness (referred as prediction throughout the paper) with *a large number of unsupervised, heterogeneous* OD models? In this study, we propose a modular acceleration system, called SUOD, to address it. The proposed system focuses on three complementary acceleration aspects (data reduction for high-dimensional data, approximation for costly models, and taskload imbalance optimization for distributed environment), while maintaining performance accuracy. Extensive experiments on more than 20 benchmark datasets demonstrate SUOD's effectiveness in heterogeneous OD acceleration, along with a real-world deployment case on fraudulent claim analysis at IQVIA, a leading healthcare firm. We open-source SUOD for reproducibility and accessibility.

1 Introduction

Outlier detection (OD) aims at identifying the samples that are deviant from the general data distribution (Zhao et al., 2019b; Lai et al., 2020), which has been used in various applications (Chandola et al., 2009; Zha et al., 2020). Notably, most of the existing OD algorithms are unsupervised due to the high cost of acquiring ground truth (Zhao et al., 2019a). Model selection and hyperparameter tuning in OD have been shown to be non-trivial problems (Zhao et al., 2020; Li et al., 2020). To reduce the risk and instability of using a single OD model, practitioners prefer to build a large corpus of OD models with variation and diversity, e.g., different algorithms, varying parameters, distinct views of the datasets, etc (Aggarwal & Sathe, 2017). This approach is known as *heterogeneous OD*. Ensemble methods that select and combine diversified base models can be leveraged to analyze heterogeneous OD models (Aggarwal, 2012; Zimek et al., 2013; Aggarwal & Sathe, 2017), and more reliable results may be achieved. The simplest combination is to take the average or maximum across all the base models as the final result (Aggarwal & Sathe, 2017), along with more complex combination approaches in both unsupervised (Zhao et al., 2019a) and semi-supervised manners (Zhao & Hryniewicki, 2018).

However, training and prediction with a large number of heterogeneous OD models is computationally expensive on high-dimensional, large datasets. For instance, proximity-based algorithms, assuming outliers behave differently in specific regions (Aggarwal, 2013), can be prohibitively slow or even completely fail to work under this setting. Representative methods such as k nearest neighbors (kNN) (Ramaswamy et al., 2000), local outlier factor (LOF) (Breunig et al., 2000), and local outlier probabilities (LoOP) (Kriegel et al., 2009), operate in Euclidean space for distance/density calculation, suffering from the curse of dimensionality (Schubert et al., 2015). Numerous works have attempted to tackle this scalability challenge from various angles, e.g., data projection (Keller et al., 2012), subspacing (Liu et al., 2008), and distributed learning for specific OD algorithms (Lozano & Acuña, 2005; Oku et al., 2014). **However, none of them provides a comprehensive solution by considering all aspects of large-scale heterogeneous OD, leading to limited practicability and efficacy**.

To tap the gap, we propose a comprehensive acceleration framework called SUOD. As shown in Fig. 1, SUOD has three modules focusing on complementary levels: random projection (**data level**), pseudo-supervised approximation (**model level**), and balanced parallel scheduling

[*]Equal contribution [1]Carnegie Mellon University [2]Peking University [3]Purdue University [4]Arima Inc. [5]IQVIA [6]University of Illinois at Urbana-Champaign. Correspondence to: Yue Zhao <zhaoy@cmu.edu>, Xiyang Hu <xiyanghu@cmu.edu>.

Proceedings of the 4th MLSys Conference, San Jose, CA, USA, 2021. Copyright 2021 by the author(s).

(**execution level**). For high-dimensional data, SUOD generates a random low-dimensional subspace for each base model by Johnson-Lindenstrauss projection (Johnson & Lindenstrauss, 1984), in which the corresponding base model is trained. If prediction on new-coming samples is needed, fast supervised models are employed to approximate costly unsupervised outlier detectors (e.g., *k*NN and LOF). To train the supervised approximators, we regard the unsupervised models' outputs on the train set as "pseudo ground truth". Intuitively, this may be viewed as distilling knowledge from complex unsupervised models (Hinton et al., 2015) by fast and more interpretable supervised models. We also build a taskload predictor to reduce the scheduling imbalance in distributed environment. Other than generically assigning the equal number of models to each worker, our balanced parallel scheduling mechanism forecasts OD model cost, e.g., training time, before scheduling them, so that the taskload is evenly distributed among workers. Notably, all three acceleration modules are designed to be independent but complementary, which can be used alone or combined as a system. It is noted that SUOD is designed for offline learning with a *stationary assumption*, although it may be further extended to *online setting* for streaming data with extra effort (Wang et al., 2019a; 2020). It is noted that offline training and prediction is one of the primary scenarios in machine learning applications (Amazon, 2020; Nakandala et al., 2020).

Our contributions are summarized as follows:

1. **The First Comprehensive OD Acceleration System**: We propose SUOD, (to our knowledge) the most comprehensive system for heterogeneous OD acceleration by a holistic view on data, model, and execution level.
2. **Analysis of Data Compression**: We examine various data compression methods and identify the best performing method(s) for large-scale outlier ensembles.
3. **Exploration of Model Approximation for OD**: We analyze the use of pseudo-supervised regression models' performance in approximating costly unsupervised OD models, as the first research effort on this topic.
4. **Forecasting-based Scheduling System**: We fix an imbalance scheduling issue in distributed heterogeneous OD efficiency, saving up to 61% execution time.
5. **Effectiveness**: We conduct extensive experiments to show the effectiveness of the acceleration modules independently, and of the entire framework as a whole, along with a real-world case on fraud detection.

To foster accessibility and reproducibility, We release SUOD with industry-level implementation[1], which also becomes a core component of the leading OD library PyOD[2].

[1]https://github.com/yzhao062/SUOD
[2]https://github.com/yzhao062/PyOD

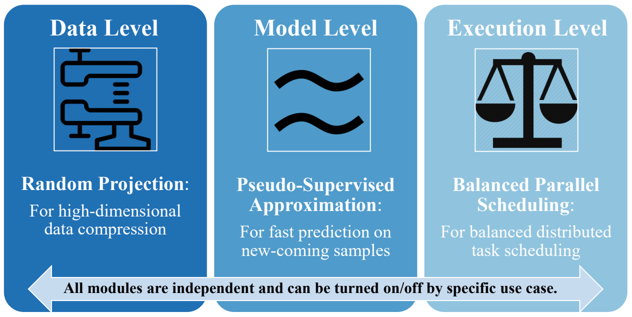
Data Level
Model Level
Execution Level
Random Projection:
For high-dimensional data compression
Pseudo-Supervised Approximation:
For fast prediction on new-coming samples
Balanced Parallel Scheduling:
For balanced distributed task scheduling
All modules are independent and can be turned on/off by specific use case.

Figure 1. SUOD focuses on three independent levels.

2 Related Work

2.1 Outlier Detection and Ensemble Learning

Outlier detection has numerous important applications, such as rare disease detection (Li et al., 2018), healthcare utilization analysis (Hu et al., 2012), video surveillance (Lu et al., 2017), fraudulent online review analysis (Akoglu et al., 2013), and network intrusion detection (Lazarevic et al., 2003). Yet, detecting outliers is challenging due to various reasons (Aggarwal, 2012; Zhao et al., 2019a;b). Most of the existing detection algorithms are unsupervised as ground truth is often absent in practice, and acquiring labels can be prohibitively expensive. We include 8 popular OD algorithms in this study for experimentation, including Isolation Forest (Liu et al., 2008), Local Outlier Factor (LOF) (Breunig et al., 2000), Angle-based Outlier Detection (ABOD) (Kriegel et al., 2009), Feature Bagging (Lazarevic & Kumar, 2005), Histogram-based Outlier Score (HBOS) (Goldstein & Dengel, 2012), and Clustering-Based Local Outlier Factor (CBLOF) (He et al., 2003). See Appendix B for details on OD models.

Consequently, relying on a single unsupervised model has inherently high risk, and outlier ensembles that leverage a group of diversified (e.g., heterogeneous) detectors have become increasingly popular (Aggarwal, 2012; Zimek et al., 2013; Aggarwal & Sathe, 2017). There are a group of unsupervised outlier ensemble frameworks proposed in the last several years from simple combination like averaging (Aggarwal & Sathe, 2017) to more complex model selection approaches like SELECT (Rayana & Akoglu, 2016), LSCP (Zhao et al., 2019a), and MetaOD (Zhao et al., 2020). In addition to fully unsupervised outlier ensembles, there are (semi-)supervised ensembling frameworks that can incorporate existing label information such like XGBOD (Zhao & Hryniewicki, 2018). For both unsupervised and (semi-)supervised methods, a large group of unsupervised, heterogeneous OD models are used as base for robustness and performance—SUOD is hereby proposed to accelerate for this scenario.

2.2 Scalability and Efficiency Challenges in OD

Efforts have been made through various channels to accelerate large-scale OD. On the **data level**, researchers try to project high-dimensional data onto lower-dimensional subspaces (Achlioptas, 2001), including simple Principal Component Analysis (PCA) (Shyu et al., 2003) and more complex subspace method HiCS (Keller et al., 2012). However, deterministic projection methods, e.g., PCA, are not ideal for building diversified heterogeneous OD—they lead to the same or similar subspaces with limited diversity by nature, resulting in the loss of outliers (Aggarwal, 2013). Complex projection and subspace methods may bring performance improvement for outlier mining, but the generalization capacity is limited. Hence, projection methods preserving pairwise distance relationships for downstream tasks should be considered. SUOD's considers both diversity induction and pairwise distance preservation for downstream tasks, leading to diversified and meaningful feature spaces (see §3.3).

On the **model level**, knowledge distillation emerges as a good way of compressing large neural networks (Hinton et al., 2015), while its usage in outlier detection is still underexplored. Knowledge distillation refers to the notion of compressing a large, often cumbersome model(s) into a small and more interpretable one(s). Under the context of OD, proximity-based models, such as LOF, can be slow (high time complexity) for predicting on new-coming samples with limited interpretability, which severely restricts their usability. SUOD adapts a similar idea to outlier mining by "distilling" complex unsupervised models. Although SUOD shares a similar concept as knowledge distillation for computational cost optimization, there are a few notable differences (see §3.4).

There are also engineering cures on the **execution level**. For various reasons, OD has no mature and efficient distributed frameworks with thousands of clusters—distributed computing for OD mainly falls into the category of "scale-up" that focuses on leveraging multiple local cores on a single machine more efficiently. To this end, specific OD algorithms can be accelerated by distributed computing with multiple workers (e.g., CPU cores) (Oku et al., 2014; Lozano & Acuña, 2005). However, these frameworks are not designed for a group of heterogeneous models but only a single algorithm, which limits their usability. It is noted that a group of heterogeneous detection models can have significantly varied computational cost. As a simple example, let us split 100 heterogeneous models into 4 groups for parallel training. If group #2 takes significantly longer time than the others to finish, it behaves like the bottleneck of the system. More formally, imbalanced task scheduling causes the system efficiency to be curbed by the worker taking the most time. There are also a line of system researches focus on more efficient task scheduling for "shorter tasks on larger degree of parallelism", e.g., Sparrow (Ousterhout et al., 2013) and Pigeon (Wang et al., 2019b). For instance, Sparrow discusses scheduling millions of tasks (at millisecond scale) on thousands of machines. Differently, Heterogeneous OD applications typically use a few OD models (e.g., 5 to 1,000) and each of them takes a few seconds to hours to run (a considerable number of tasks; each takes time to run), which is "longer tasks with a small number of workers". SUOD is, therefore, proposed to reduce the inefficiency in distributed heterogeneous OD specifically.

3 System Design

3.1 Problem Formulation

OD applications and research are primarily running a single, on-prime, powerful machine with multiple cores/workers, due to its high-stake nature (e.g., data sensitive of financial transactions). Therefore, we formulate unsupervised heterogeneous OD training and prediction tasks with:

- m unsupervised heterogeneous OD models, where $\mathcal{M} = \{M_1, ..., M_m\}$. We refer the combination of an algorithm and its hyperparamters as a model.
- train data $\mathbf{X}_{\text{train}} \in \mathbb{R}^{n \times d}$ without ground truth labels.
- (optional) test data $\mathbf{X}_{\text{test}} \in \mathbb{R}^{l \times d}$ for prediction. The OD models should be trained first.
- (optional) t available workers for distributed computing, e.g., t cores on a single machine. This constructs the worker pool as $\mathcal{W} = \{W_1, ..., W_t\}$. By default, a single worker setting ($t = 1$) is assumed.

Without SUOD, one will train each model in $\mathcal{M}$ on $\mathbf{X}_{\text{train}}$ iteratively, e.g., with a for loop. If there are multiple workers available ($t > 1$), a generic scheduling system will equally split m models into t groups, so each available worker will process roughly $\lceil \frac{m}{t} \rceil$ models. Prediction on new-coming samples $\mathbf{X}_{\text{test}}$ follows the similar manner as training. See Algorithm 1 for detailed symbol definition.

3.2 System Design

SUOD is designed to accelerate the above procedures with three independent modules targeting different levels (data, model, and execution). **Each module can be flexibly enabled or disabled** as shown in Algorithm 1. For high-dimensional data, SUOD can randomly project the original feature onto low-dimensional spaces (§3.3). Pairwise distance relationships are expected to be maintained, and the diversity is induced for ensemble construction. A fast supervised regressor could be used to approximate the output of each slow and costly unsupervised detector. We could use the supervised regressor for fast prediction (§3.4). If there are multiple available workers for distributed computing, we propose a forecasting-based scheduling mechanism (§3.5)

Algorithm 1 SUOD: Training and Prediction

Input: m unsupervised OD models $\mathcal{M}$; train data $\mathbf{X}_{\text{train}} \in \mathbb{R}^{n \times d}$; target dimension k; the number of available workers t (optional, default to 1); test data $\mathbf{X}_{\text{test}} \in \mathbb{R}^{l \times d}$ (optional); supervised regressor R (optional)
Output: Trained OD models $\mathcal{M}$; fitted pseudo-supervised regressors R (optional); test prediction results $\hat{y}_{\text{test}}$ (optional)

```
 1: for each model $M_i$ in $\mathcal{M}$ do
 2:   if random projection is enabled (§3.3) then
 3:     Initialize a JL transformation matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$
 4:     Get feature subspace $\psi_i := \langle \mathbf{X}_{\text{train}}, \mathbf{W} \rangle \in \mathbb{R}^{n \times k}$
 5:   else
 6:     Use the original space $\psi_i := \mathbf{X}_{\text{train}} \in \mathbb{R}^{n \times d}$
 7:   end if
 8: end for
 9: if number of available workers $t > 1$ then
10:   Scheduling the training of $m$ models to $t$ workers by minimizing Eq. 2 (see §3.5). Models are trained on the corresponding feature space $[\psi_1, ..., \psi_m]$.
11: else
12:   Train each model $M_i$ in $\mathcal{M}$ on its corresponding $\psi_i$.
13: end if
14: Return trained models $\mathcal{M}$

15: if Scoring on newcoming samples $\mathbf{X}_{\text{test}}$ then
16:   Acquire the pseudo ground truth $target^{\psi_i}$ as the output of $M_i$ on $\psi_i$, i.e., $target^{\psi_i} := M_i(\psi_i)$
17:   for each costly model $M_i$ in $\mathcal{M}_c$ do
18:     Initialize a supervised regressor $R_i$
19:     Train $R_i$ by $\{\psi_i, target^{\psi_i}\}$ (see §3.4)
20:     Predict by supervised $R_i$, $\hat{y}^i_{\text{test}} = R_i$.predict($\mathbf{X}_{\text{test}}$)
21:   end for
22:   Return $\hat{y}_{\text{test}}$ and approximation regressors $R$
23: end if
```

to reduce taskload imbalance in heterogeneous OD.

SUOD's API design follows `scikit-learn` (Pedregosa et al., 2011) and `PyOD` (Zhao et al., 2019b), with an *initialization*, *fit*, and *prediction* schema (see Codeblock 1).

```
import SUOD
# initialize a group of heterogeneous OD models
base_estimators=[
    LOF(n_neighbors=40), ABOD(n_neighbors=50),
    LOF(n_neighbors=60), IForest(n_estimators=100)]

# initialize SUOD with module flags
clf=SUOD(base_estimators=base_estimators,
    rp_flag_global=True,
    approx_clf=RandomForestRegressor(),
    bps_flag=True,
    approx_flag_global=True)

# fit and make prediction
clf.fit(X_train)
test_labels=clf.predict(X_test)
test_scores=clf.decision_function(X_test)}
```

Codeblock 1. SUOD's easy-to-use API (inspired by scikit-learn)

3.3 Data Level: Random Projection (RP) for Data Compression

For high-dimensional datasets, many proximity-based OD algorithms suffer from the curse of dimensionality (Lazarevic & Kumar, 2005). A widely used dimensionality reduction to cure this is the Johnson-Lindenstrauss (JL) projection (Johnson & Lindenstrauss, 1984), which has been applied to OD because of its great scalability (Schubert et al., 2015). Unlike PCA discussed in §2.2, JL projection could compress the data without heavy distortion on the Euclidean space—*outlyingness information is therefore preserved in the compression*. Moreover, its built-in randomness can be useful for diversity induction in heterogeneous OD—data randomness can also serve as a source of heterogeneity. Additionally, JL projection ($\mathcal{O}(ndk)$) is more efficient than popular PCA ($\mathcal{O}(nd^2 + n^3)$) with lower time complexity, where k is the target dimension for compression.

JL projection is defined as: given a set of n samples $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n\}$, each $\mathbf{x}_i \in \mathbb{R}^d$, let $\mathbf{W}$ be a $k \times d$ projection matrix with each entry drawing independently from a predefined distribution F, e.g., Gaussian, so that $\mathbf{W} \sim F$. Then the JL projection is a function $f : \mathbb{R}^d \to \mathbb{R}^k$ such that $f(\mathbf{x}_i) = \frac{1}{\sqrt{k}}\mathbf{x}_i\mathbf{W}^T$. JL projection randomly projects high-dimensional data (d dimensions) to lower-dimensional subspaces (k dimensions), but preserves the distance relationship between points. In fact, if we fix some $\mathbf{v} \in \mathbb{R}^d$, for every $\epsilon \in (0, 3)$, we have (Schubert et al., 2015):

$$P\left[(1-\epsilon)\|\mathbf{v}\|^2 \leq \|\frac{1}{\sqrt{k}}\mathbf{v}\mathbf{W}^T\|^2 \leq (1+\epsilon)\|\mathbf{v}\|^2\right] \leq 2e^{-\epsilon^2\frac{k}{6}} \tag{1}$$

Let $\mathbf{v}$ to be the differences between vectors. Then, the above bound shows that for a finite set of n vectors $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n\} \in \mathbb{R}^d$, the pairwise Euclidean distance is preserved within a factor of $(1 \pm \epsilon)$, reducing the vectors to $k = \mathcal{O}(\frac{log(n)}{\epsilon^2})$ dimensions.

Four distributions F for JL projection are considered in this study: (i) *basic*: the transformation matrix is generated by standard Gaussian; (ii) *discrete*: the transformation matrix is picked randomly from Rademacher distribution (uniform in $\{-1, 1\}$); (iii) *circulant*: the transformation matrix is obtained by rotating the subsequent rows from the first row which is generated from standard Gaussian and (iv) *toeplitz*: the first row and column of the transformation matrix are generated from standard Gaussian, and each diagonal uses a constant value from the first row and column. A more thorough empirical study on JL methods can be found in (Venkatasubramanian & Wang, 2011).

For $\mathbf{X}_{\text{train}}$, RP can reduce the original feature space d to the target dimension k. Specifically, SUOD initializes a JL transformation matrix $\mathbf{W} \in \mathbb{R}^{d \times k}$ by drawing from one of the four distributions F. $\mathbf{X}_{\text{train}}$ is therefore projected onto the k dimension feature space as $\mathbf{X}'_{\text{train}} = \langle \mathbf{X}_{\text{train}}, \mathbf{W} \rangle \in \mathbb{R}^{n \times k}$. The transformation matrix $\mathbf{W}$ should be kept for transforming newcoming samples: $\mathbf{X}'_{\text{test}} = \langle \mathbf{X}_{\text{test}}, \mathbf{W} \rangle \in \mathbb{R}^{m \times k}$. It is noted that RP module should be used with caution. First, projection may not be helpful or even detrimental for subspace methods like Isolation Forest and HBOS. Second, if the number of samples n is too small, the bound in Eq. (1) does not hold.

3.4 Model Level: Pseudo-Supervised Approximation (PSA) for Fast Prediction

PSA module is designed to speed up prediction on newcoming samples. Specifically, after the models in $\mathcal{M}$ are trained, SUOD uses PSA to approximate and replace each **costly unsupervised model** by a **faster supervised regressor** for **fast offline prediction**. Notably, only costly unsupervised models should be replaced; the cost can be measured through time complexity analysis. For instance, proximity-based algorithms like kNN and LOF are costly in prediction (upper bounded by $\mathcal{O}(nd)$), and can be effectively replaced by "cheaper" supervised models like random forest (Breiman, 2001) (upper bounded by $\mathcal{O}(ph)$ where p denotes the number of base trees and h denotes the max depth of a tree; often $p \ll n$ and $h \leq d$). This "pseudo-supervised" model uses the output of unsupervised models (outlyingness score) as "the pseudo ground truth"—*the goal is to approximate the output of the underlying unsupervised model*. It is noted that the approximator's prediction cost (i.e., time complexity) must be lower than the underlying unsupervised model, while maintaining a comparable level of prediction accuracy. For instance, fast (low time complexity) OD algorithms like Isolation Forest and HBOS should not be approximated and replaced. To facilitate this process, we predefine the pool of costly OD algorithm $\mathcal{M}_c$. If a model M_i is in $\mathcal{M}_c$, it will be approximated by default.

As shown in Algorithm 1, for each costly trained unsupervised model M_i belonging to $\mathcal{M}_c$, a supervised regressor R_i is trained by $\{\mathbf{X}_{\text{train}}, \mathbf{y}_i\}$; $\mathbf{y}_i$ is the outlyingness score by M_i on the train set (referred as pseudo ground truth)[1]. R_i is then used to predict on unseen data $\mathbf{X}_{\text{test}}$.

Remark 1: Supervised tree ensembles are recommended for PSA due to their outstanding scalability, robustness to overfitting, and interpretability (e.g., feature importance) (Hu et al., 2019). In addition to the execution time reduction, supervised models are generally more interpretable. For instance, random forest used in the experiments can yield feature importance automatically to facilitate understanding.

Remark 2: Notably, PSA may be viewed as using supervised regressors to distill knowledge from unsupervised OD models. However, it works in a fully unsupervised manner, unlike the classic distillation under supervised settings.

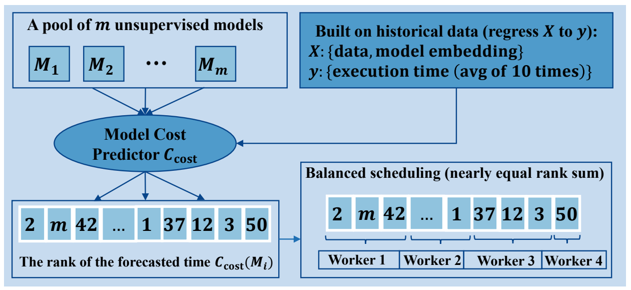
A pool of m unsupervised models
M_1
M_2
…
M_m
Built on historical data (regress X to y):
X: {data, model embedding}
y: {execution time (avg of 10 times)}
Model Cost Predictor C_{cost}
2 m 42 … 1 37 12 3 50
The rank of the forecasted time $C_{cost}(M_i)$
Balanced scheduling (nearly equal rank sum)
2 m 42 … 1 37 12 3 50
Worker 1
Worker 2
Worker 3
Worker 4

Figure 2. Flowchart of balanced parallel scheduling, which aims to assign nearly equal rank sum by model cost predictor C_{cost}.

3.5 Execution level: Balanced Parallel Scheduling (BPS) for Taskload Imbalance Reduction

Taskload Imbalance within Distributed Systems: It is likely to observe system inefficiency due to taskload imbalance among workers when a large number of heterogeneous models are used. For instance, we want to train 25 OD models with varying parameters from each of the four algorithms $\{k$NN, Isolation Forest, HBOS, OCSVM$\}$ (100 heterogeneous models in total). The existing distributed frameworks, e.g., the voting machine in scikit-learn (Pedregosa et al., 2011) or general frameworks like joblib[2], may simply split the models into 4 subgroups by order and schedule the first 25 models (all kNNs) on worker 1, the next 25 models on worker 2, etc. This does not account for the fact that within a group of heterogeneous models, the computational cost varies. Scheduling the task with the equal number of models can result in highly imbalanced load. In the worst-case scenario, one worker may be assigned significant more load than the rest, resulting in halt to the entire process. In this example, the kNN subgroup will be the system curb due to high time complexity. One naive solution is to shuffle the base models randomly. However, there is no guarantee this heuristic could work, and it may be practically infeasible.

The proposed BPS focuses on delivering a more balanced task scheduling among workers via forecasting their cost in advance. Ideally, all workers can finish the scheduled tasks within a similar duration and return the results. To achieve this goal, SUOD comes with a *model cost predictor* C_{cost} to forecast the model execution time (sum of 10 trials) given the meta-features (descriptive features) of a dataset (Zhao et al., 2020). C_{cost} is trained on 11 algorithm family with 47 benchmark datasets by 10-fold cross validation, yielding an effective regressor (random forest is used in this study). C_{cost}'s outputs show high Spearman's Rank correlation (Spearman, 1904) ($r_s > 0.9$) to the true

[1]If RP is enabled, $\mathbf{X}_{\text{train}}$ is replaced by the compressed space.

[2]https://github.com/joblib/joblib

model cost rank with low p-value ($p < 0.0001$), in all folds. Given a dataset and a model, C_{cost} can predict the execution time of the model's execution time with high accuracy. It is noted that the model cost predictor is only trained for the major methods in Python Outlier Detection Toolbox (PyOD) (Zhao et al., 2019b). For unseen models, they are classified as "unknown" to be assigned with the max cost to prevent overoptimistic scheduling.

As a result, a scheduling strategy is proposed by enforcing a nearly equal rank sum by the forecasted execution time among all available workers. Fig. 2 provides a simple example of scheduling m models to 4 workers. More formally, before scheduling m models for training (or prediction), cost predictor C_{cost} is invoked to forecast the execution time of each model M in $\mathcal{M}$ as $C_{\text{cost}}(M)$ and output the model cost rank of each model in $\{1, m\}$ (the higher the rank, the longer the forecasted execution time). If there are t cores (workers), each worker will be assigned a group of models to achieve the objective of minimizing the taskload imbalance among workers (Eq. 2).

$$\min_{\mathcal{W}} \sum_{i=1}^{t} \left| \sum_{M_j \in \mathcal{W}_i} C_{\text{cost}}(M_j) - \frac{m^2+m}{2t} \right| \tag{2}$$

Consequently, each worker is assigned with a group of models with the rank sum close to the average rank sum $\frac{(1+m)m}{2}/t = \frac{m^2+m}{2t}$. Indeed, the accurate running time prediction is less relevant as it depends on the hardware—the rank is more useful as a relevance measure with the transferability to other hardware. That is, the running time will vary on different machines, but the relative rank should preserve. One issue around the sum of ranks is the overestimation of high-rank models. For instance, rank f-th model will be counted f times more heavily than rank 1 model during the sum calculation, even their actual running time difference will not be as big as f times. To fix this, we introduce a discounted rank by rescaling model rank f to $1 + \frac{\alpha f}{m}$, where α denotes the scaling strength (default to 1). A larger α therefore puts more emphasis on costly models.

4 Experiments & Discussion

First, three experiments are conducted to understand the effectiveness of individual modules independently: **Q1**: how will different compression methods affect the performance of downstream OD accuracy (§4.1); **Q2**: will use pseudo-supervised regressors lead to more efficient prediction in comparison to the original unsupervised models (§4.2) and **Q3**: how does the proposed balanced scheduling strategy perform under different settings (varying number of models m, number of workers t, etc.) (§4.3). Then, the full SUOD with all three modules enabled is evaluated regarding time cost and prediction accuracy (on new samples) (§4.4). Finally, a real-world deployment case on fraudulent claim analysis at IQVIA (a leading healthcare organization), is described (§4.5). The details of experiment setting, e.g., datasets, evaluation metrics, and the pool of heterogeneous OD models, can be found in the Appendix.

4.1 Q1: Comparison of Model Compression Methods

In this section, **we demonstrate the effectiveness of RP module in high-dimensional OD tasks.** To evaluate the effect of data projection, we choose three costly outlier detection algorithms, ABOD, LOF, and kNN to measure their execution time, and prediction accuracy (ROC and P@N), before and after projection. These methods directly or indirectly measure sample similarity in Euclidean space, e.g., pairwise distance, which is prone to the curse of dimensionality, where data compression can help.

Table 1 shows the comparison results on four datasets (see Appendix Table A); the reduced dimension is set as $k = \frac{2}{3}d$ (33% compression). We compare the proposed four JL projection methods (see §3.3 for details of *basic*, *discrete*, *circulant*, and *toeplitz*) with **original** (no projection), **PCA**, and **RS** (randomly select k features from the original d features, used in Feature Bagging (Lazarevic & Kumar, 2005) and LSCP (Zhao et al., 2019a)). First, all compression methods show superiority regarding time cost. Second, **original** (no compression) method rarely outperforms, possibly due to the curse of dimensionality and lack of diversity (Zhao et al., 2019a). Third, **PCA** is inferior to **original** regarding prediction accuracy (see LOF performance in Table 1e, 1h, and 1k). The observation supports our claim that PCA is not suited in this scenario (see §2.2). Fourth, JL methods generally lead to equivalent or better prediction performance than **original** regarding both time and prediction accuracy. Lastly, among all JL methods, *circulant* and *toeplitz* generally outperform others. For instance, *toeplitz* brings more than 60% time reduction for kNN, demonstrating the effectiveness of RP and chosen as the default choice.

4.2 Q2: The Visual and Quantitative Analysis of PSA

Through both visualization and quantitative analysis, we observe PSA is useful for accelerating prediction of proximity-based OD algorithms. To better understand the effect of pseudo-supervised approximation, we first generate a synthetic dataset with 200 two-dimensional samples, consisting of 40 outliers generated by Normal distribution and 160 normal samples generated from Uniform distribution. In Fig. 3, we plot the decision surfaces of four costly unsupervised models (ABOD, Feature Bagging, kNN, and LOF) and of their corresponding supervised approximators (random forest regressor), with accuracy errors reported. In general, the faster pseudo supervised approximators do not lead to more errors, justifying the effectiveness of approx-

Table 1. Comparison of various data compression methods on different outlier detectors and datasets (see Appendix Table A). Each column corresponds to an evaluation metric (execution time is measured in seconds); the best performing method is indicated in **bold**. JL projection methods, especially *circulant* and *toeplitz*, outperform regarding both time cost and prediction accuracy.

(a) ABOD on **Cardio**

Method	Time	ROC	P@N
original	0.98	0.59	0.25
PCA	**0.82**	0.59	0.26
RS	0.92	**0.63**	**0.29**
basic	0.83	0.62	0.28
discrete	**0.82**	0.62	0.28
circulant	0.83	0.62	0.27
toeplitz	0.83	0.62	0.28

(b) LOF on **Cardio**

Method	Time	ROC	P@N
original	0.08	0.55	0.17
PCA	**0.04**	0.56	0.19
RS	**0.04**	0.57	0.15
basic	**0.04**	**0.60**	0.20
discrete	**0.04**	0.59	0.19
circulant	**0.04**	0.59	0.20
toeplitz	**0.04**	**0.60**	**0.21**

(c) kNN on **Cardio**

Method	Time	ROC	P@N
original	0.09	0.71	0.34
PCA	**0.03**	0.73	0.34
RS	**0.03**	0.69	**0.38**
basic	**0.03**	**0.74**	0.35
discrete	**0.03**	**0.74**	0.37
circulant	**0.03**	**0.74**	0.34
toeplitz	**0.03**	0.73	0.35

(d) ABOD on **MNIST**

Method	Time	ROC	P@N
original	12.89	0.80	**0.39**
PCA	8.93	**0.81**	0.37
RS	**8.27**	0.74	0.32
basic	8.94	0.80	0.38
discrete	8.86	0.80	**0.39**
circulant	9.33	0.80	0.38
toeplitz	8.96	0.80	0.38

(e) LOF on **MNIST**

Method	Time	ROC	P@N
original	7.64	0.68	0.29
PCA	4.92	0.67	0.27
RS	**3.65**	0.63	0.23
basic	4.87	0.70	0.31
discrete	5.21	0.70	**0.32**
circulant	5.06	0.69	0.31
toeplitz	4.97	**0.71**	0.31

(f) kNN on **MNIST**

Method	Time	ROC	P@N
original	7.13	**0.84**	**0.42**
PCA	3.92	**0.84**	0.40
RS	**3.33**	0.77	0.34
basic	4.17	**0.84**	**0.42**
discrete	4.11	**0.84**	0.41
circulant	4.13	**0.84**	0.41
toeplitz	4.11	**0.84**	**0.42**

(g) ABOD on **Satellite**

Method	Time	ROC	P@N
original	4.03	0.59	0.41
PCA	**3.01**	0.62	0.44
RS	3.53	0.63	0.44
basic	3.10	0.64	0.45
discrete	3.12	0.65	0.46
circulant	3.14	**0.66**	**0.48**
toeplitz	3.14	**0.66**	0.47

(h) LOF on **Satellite**

Method	Time	ROC	P@N
original	0.82	**0.55**	0.37
PCA	**0.23**	0.54	0.36
RS	0.39	0.54	0.37
basic	0.31	0.54	0.37
discrete	0.32	0.54	0.37
circulant	0.39	**0.55**	**0.38**
toeplitz	0.37	0.54	0.37

(i) kNN on **Satellite**

Method	Time	ROC	P@N
original	0.71	0.67	0.49
PCA	**0.18**	0.67	0.50
RS	0.31	0.68	0.49
basic	0.24	0.68	0.49
discrete	0.25	0.69	0.50
circulant	0.33	**0.70**	0.50
toeplitz	0.30	**0.70**	**0.51**

(j) ABOD on **Satimage-2**

Method	Time	ROC	P@N
original	3.68	0.85	0.28
PCA	**2.70**	0.88	0.30
RS	3.20	0.89	0.28
basic	2.78	0.91	0.29
discrete	2.79	0.91	**0.31**
circulant	2.85	0.91	0.29
toeplitz	2.83	**0.92**	0.30

(k) LOF on **Satimage-2**

Method	Time	ROC	P@N
original	0.79	0.54	0.07
PCA	**0.20**	0.52	0.04
RS	0.37	0.53	0.08
basic	0.29	0.52	0.08
discrete	0.30	0.53	0.07
circulant	0.43	**0.59**	**0.11**
toeplitz	0.32	0.54	0.09

(l) kNN on **Satimage-2**

Method	Time	ROC	P@N
original	0.68	0.94	**0.39**
PCA	**0.15**	0.94	**0.39**
RS	0.29	0.94	0.38
basic	0.23	0.94	0.38
discrete	0.20	0.95	0.37
circulant	0.36	**0.96**	0.37
toeplitz	0.25	**0.96**	**0.39**

imation. Fig. 3 subfigure 4 and 6 show that the approximators have even lower errors than the original (Feature Bagging and kNN). With a closer look at the decision surfaces, we assume that the approximation process improves the generalization ability of the model by "ignoring" some overfitted points. However, the approximation does not work with ABOD, possibly due to its extremely coarse decision surface (see Fig. 3, subfigure 1).

Table 2 and 3 compare prediction performance (scoring on new-coming samples) between the original unsupervised models and pseudo-supervised approximators on 10 datasets with 6 costly algorithms, regarding ROC and P@N. Since these algorithms are more computationally expensive than random forest regressors for prediction (by time complexity analysis), we skip the prediction time comparison where the gain is clear. Consequently, the focus is whether the approximators could predict unseen samples as good as the original unsupervised models. The tables reveal that not all the algorithms are suited for PSA, which is in line with the visual analysis. For instance, ABOD shows per-

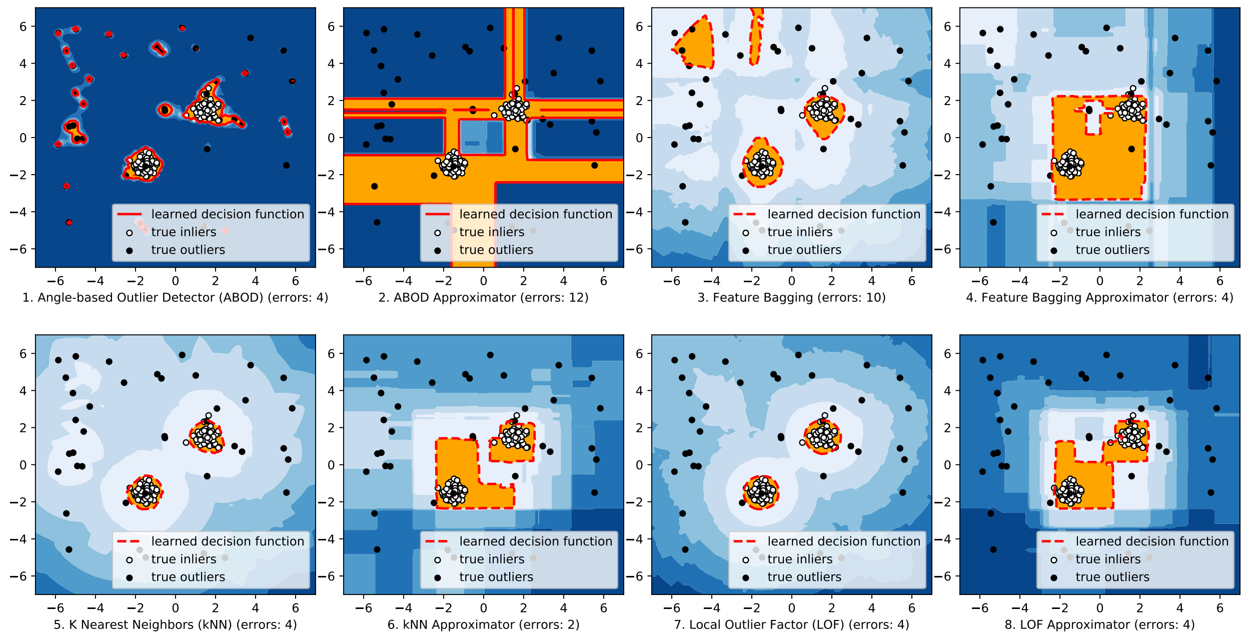

Figure 3. Decision surface comparison among unsupervised models and their pseudo-supervised approximators (in pairs). The approximator's decision boundary shows a tentative regularization effect, leading to even fewer detection errors.

formance decrease on half of the datasets. Notably, ABOD looks for a low-dimensional subspace to embed the normal samples (Aggarwal, 2013), leading to a complex decision surface to approximate. In contrast, proximity-based models benefit from the approximation. Both tables show, kNN, LoF, and akNN (average kNN) experience a performance gain. Specifically, all three algorithms yield around 100% ROC increase on **HTTP**. Other algorithms, such as Feature Bagging and CBLOF, show a minor performance variation within the acceptable range. In other words, it is useful to perform PSA for these estimators as the time efficiency is greatly improved with little to no loss in prediction accuracy.

4.3 Q3: Time Reduction of Balanced Scheduling

To evaluate the effectiveness of the proposed BPS algorithm, we run the following experiments by varying: (i) the size (n) and the dimension (d) of the datasets, (ii) the number of estimators (m) and (iii) the number of CPU cores (t). Due to the space limit, we only show the training time comparison between the generic scheduling and BPS on **Cardio**, **Letter**, **PageBlock**, and **Pendigits**, by setting $m \in \{100, 500\}$ and $t \in \{2, 4, 8\}$, consistent with the single machine setting in real-world applications.

Table 4 shows that the proposed BPS has a clear edge over the generic scheduling mechanism (denoted as **Generic** in the tables) that equally splits the tasks by order. It yields a significant time reduction (denoted as **% Redu** in the table), which gets more remarkable if more cores are used along with large datasets. For instance, the time reduction is more than 40% on **PageBlock** and **Pendigits** when 8 cores are used. This agrees with our assumption that model cost vary more drastically on large datasets given the time complexity increase non-linearly to the size—the proposed BPS method is particularly helpful.

4.4 Full System Evaluation with All Modules

Table 5 shows the performance of SUOD with all three modules enabled, even not all of them are always needed in practice. In total, 600 hundred randomly selected OD models from `PyOD` are trained and tested on 10 datasets. To simulate the "worst-case scenario", we build the model pool $\mathcal{M}$ by randomly select OD models, which minimizes the intrinsic task load imbalance. In real-world applications, this order randomization may not be possible as discussed in §3.5—two adjacent models are often from the same algorithm family and more prone to scheduling imbalance. **Although this setting will make the effectiveness of BPS module less impressive, we choose it to provide an empirical worst-case performance guarantee—the framework should generally perform better in practice.**

SUOD consistently yields promising results even we deliberately choose the unfavored setting. **Fit_B** and **Pred_B** denote the fit and prediction time of the baseline setting (no compression, no approximation, generic parallel task scheduling; see §2.2). In comparison, SUOD (denoted as **Fit_S** and **Pred_S**) brings time reduction on majority of the datasets with minor to no performance degradation. To measure the prediction performance, we measure the ROC and

Table 2. Prediction ROC scores of unsupervised models (Orig) and their pseudo-supervised approximators (Appr) by the average of 10 independent trials. The better method within each pair is indicated in **bold**. The approximators (Appr) outperform in most cases.

Dataset	**Annthyroid**		**Breastw**		**Cardio**		**HTTP**		**MNIST**		**Pendigits**		**Pima**		**Satellite**		**Satimage-2**		**Thyroid**	
Model	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr
ABOD	**0.83**	0.71	0.92	**0.93**	**0.63**	0.53	**0.15**	0.13	**0.81**	0.79	0.67	**0.82**	0.66	**0.70**	0.59	**0.68**	0.89	**0.99**	**0.96**	0.67
CBLOF	0.67	**0.68**	0.96	**0.98**	0.73	**0.76**	**1.00**	**1.00**	0.85	**0.89**	**0.93**	**0.93**	0.63	**0.68**	0.72	**0.77**	**1.00**	**1.00**	0.92	**0.97**
FB	**0.81**	0.45	**0.34**	0.10	0.61	**0.70**	0.34	**0.97**	0.72	**0.83**	0.39	**0.51**	0.59	**0.63**	0.53	**0.64**	0.36	**0.40**	**0.83**	0.46
kNN	**0.80**	0.79	**0.97**	**0.97**	0.73	**0.75**	0.19	**0.85**	0.85	**0.86**	0.74	**0.87**	0.69	**0.71**	0.68	**0.75**	0.96	**0.99**	0.97	**0.98**
akNN	0.81	**0.82**	**0.97**	**0.97**	0.67	**0.72**	0.19	**0.88**	0.84	**0.85**	0.72	**0.87**	0.69	**0.71**	0.66	**0.74**	0.95	**0.99**	0.97	**0.98**
LOF	0.74	**0.85**	0.44	**0.45**	0.60	**0.68**	0.35	**0.75**	0.72	**0.76**	0.38	**0.47**	0.59	**0.65**	0.53	**0.66**	0.36	**0.38**	0.80	**0.95**

Table 3. Prediction P@N scores of unsupervised models (Orig) and their pseudo-supervised approximators (Appr) by the average of 10 independent trials. The better method within each pair is indicated in **bold**. The approximators (Appr) outperform in most cases.

Dataset	**Annthyroid**		**Breastw**		**Cardio**		**HTTP**		**MNIST**		**Pendigits**		**Pima**		**Satellite**		**Satimage-2**		**Thyroid**	
Model	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr	Orig	Appr
ABOD	**0.31**	0.08	0.80	**0.83**	**0.27**	0.20	0.00	0.00	**0.40**	0.27	**0.05**	**0.05**	0.48	**0.52**	0.41	**0.46**	0.21	**0.64**	**0.36**	0.00
CBLOF	**0.25**	0.24	0.86	**0.90**	0.31	**0.34**	**0.02**	0.01	0.42	**0.48**	0.35	**0.36**	0.43	**0.48**	0.54	**0.57**	**0.96**	**0.96**	0.26	**0.38**
FB	**0.24**	0.02	0.03	**0.07**	0.23	**0.26**	0.02	**0.04**	0.34	**0.36**	0.03	**0.07**	0.37	**0.44**	0.37	**0.42**	0.03	**0.04**	**0.05**	0.02
kNN	0.30	**0.32**	**0.89**	**0.89**	0.37	**0.46**	**0.03**	**0.03**	0.42	**0.45**	**0.08**	0.06	**0.47**	**0.47**	0.49	**0.53**	0.32	**0.43**	0.33	**0.42**
AkNN	0.30	**0.33**	0.88	**0.89**	0.34	**0.40**	**0.03**	**0.03**	0.41	**0.45**	0.05	**0.13**	0.48	**0.49**	0.47	**0.52**	0.25	**0.43**	0.31	**0.44**
LOF	0.27	**0.36**	0.19	**0.35**	**0.23**	**0.23**	0.01	**0.03**	**0.33**	0.32	0.03	**0.08**	0.40	**0.44**	0.37	**0.42**	0.04	**0.07**	0.19	**0.25**

Table 4. Training time comparison (in seconds) between Simple scheduling and BPS against various number of OD models and workers. Percent of time reduction, Redu (%), is indicated in **bold**. BPS consistently outperform to Generic scheduling
.

Dataset	***n***	***d***	***m***	***t***	**Generic**	**BPS**	**Redu (%)**
Cardio	1831	21	500	2	240.12	221.34	**7.82**
Cardio	1831	21	500	4	185.44	154.43	**16.72**
Cardio	1831	21	500	8	140.63	120.02	**14.65**
Cardio	1831	21	1000	2	199.77	185.63	**7.08**
Cardio	1831	21	1000	4	130.82	110.60	**15.45**
Cardio	1831	21	1000	8	97.75	73.43	**24.88**
Letter	1600	32	500	2	111.95	109.52	**2.17**
Letter	1600	32	500	4	92.69	86.24	**6.94**
Letter	1600	32	500	8	57.21	48.72	**14.84**
Letter	1600	32	1000	2	224.61	222.59	**0.90**
Letter	1600	32	1000	4	228.08	172.07	**24.56**
Letter	1600	32	1000	8	109.50	89.51	**17.80**
PageBlock	5393	10	100	2	51.11	35.17	**31.19**
PageBlock	5393	10	100	4	42.49	16.23	**61.80**
PageBlock	5393	10	100	8	38.45	16.97	**55.86**
PageBlock	5393	10	500	2	197.84	137.46	**30.52**
PageBlock	5393	10	500	4	167.36	76.14	**54.51**
PageBlock	5393	10	500	8	127.08	66.29	**47.84**
Pendigits	6870	16	500	2	351.97	287.14	**18.42**
Pendigits	6870	16	500	4	288.51	146.50	**49.22**
Pendigits	6870	16	500	8	180.86	102.11	**43.33**
Pendigits	6870	16	1000	2	697.20	561.15	**19.51**
Pendigits	6870	16	1000	4	579.70	288.11	**50.33**
Pendigits	6870	16	1000	8	365.20	182.32	**50.08**

P@N by averaging the base model results (denoted as **Avg**_) and the maximum of average of the base models (denoted as **MOA**_), a widely used two-phase outlier score combination framework (Aggarwal & Sathe, 2017). Surprisingly, SUOD even leads to small performance boost in scoring new samples on most of the datasets (**Annthyroid**, **Cardio**, **MNIST**, **Optdigits**, **Pendigits**, and **Thyroid**). This performance gain may be jointly credited to the regularization effect by the randomness injected in JL projection (§3.3) and the pseudo-approximation (§3.4)—the baseline setting may be overfitted on certain datasets. It is noted that SUOD leads to more improvement on high-dimensional, large datasets. For instance, the fit time is significantly reduced on **Shuttle** (more than 50%). On the contrary, SUOD is less meaningful for small datasets like **Pima** and **Cardio**, although they may also yield performance improvement. Again, our settings mimics the worst case scenario for SUOD (the model order is already randomly shuffled) but still observe a great performance improvement; real-world applications should generally expect more significant results.

4.5 Real-World Deployment: Fraudulent Medical Claim Analysis at IQVIA

Estimated by the United States Government Accountability Office and Federal Bureau of Investigation, healthcare frauds cost American taxpayers tens of billions dollars a year (Aldrich et al., 2014; Bagdoyan, 2018). Detecting fraudulent medical claims is crucial for taxpayers, pharmaceutical companies and insurance companies. To further demonstrate SUOD's performance on industry data, we deploy it on a proprietary pharmacy claim dataset owned by IQVIA (a leading healthcare firm) consisting of 123,720 medical claims among which 19,033 (15.38%) are labeled as fraudulent. In each of the claim, there are 35 features including information such as drug brand, copay amount, insurance details, location and pharmacy/patient demographics. The current system in use is based on a group of selected detection models in `PyOD`, and an averaging method is applied on top of the base model results as the initial result. The cases marked as high risk are then transferred to human investigators in special investigation unit (SIU) for verification. It is important to provide prompt and accurate first-round screening for SIU, which leads to huge expense save.

Table 5. Comparison between the baseline (denoted as _B) and SUOD (denoted as _S) regarding time cost, and prediction accuracy (ROC and P@N). The better method within each pair is indicated in **bold** (Optdigits fail to yield meaningful P@N). SUOD generally brings time reduction with no loss in prediction accuracy on majority of datasets.

Data Information				**Time Cost (in seconds)**				**Ensemble Model Performance (ROC)**				**Ensemble Model Performance (P@N)**			
Dataset	***n***	***d***	***t***	**Fit_B**	**Fit_S**	**Pred_B**	**Pred_S**	**Avg_B**	**Avg_S**	**MOA_B**	**MOA_S**	**Avg_B**	**Avg_S**	**MOA_B**	**MOA_S**
Annthyroid	7200	6	5	73.91	**65.23**	47.48	**44.26**	0.91	**0.93**	0.91	**0.93**	0.46	**0.54**	0.46	**0.55**
Annthyroid	7200	6	10	71.00	**42.94**	44.68	**38.66**	0.91	**0.93**	0.92	**0.93**	0.46	**0.54**	0.46	**0.54**
Annthyroid	7200	6	30	42.80	**33.98**	30.92	**25.67**	0.91	**0.93**	0.92	**0.93**	0.46	**0.54**	0.46	**0.54**
Cardio	1831	21	5	**78.84**	79.70	**46.09**	46.68	0.91	**0.93**	0.91	**0.93**	0.46	**0.54**	0.45	**0.55**
Cardio	1831	21	10	72.04	**53.43**	44.57	**38.31**	0.91	**0.93**	0.91	**0.93**	0.46	**0.54**	0.46	**0.54**
Cardio	1831	21	30	47.53	**44.57**	**31.31**	31.43	0.91	**0.93**	0.92	**0.93**	0.46	**0.54**	0.46	**0.55**
MNIST	7603	100	5	856.53	**748.40**	453.39	**324.76**	0.77	**0.81**	0.77	**0.81**	0.29	**0.35**	0.28	**0.34**
MNIST	7603	100	10	726.76	**573.66**	367.85	**328.95**	0.78	**0.81**	0.78	**0.81**	0.29	**0.35**	0.30	**0.34**
MNIST	7603	100	30	357.40	**329.71**	260.80	**134.08**	0.78	**0.81**	0.78	**0.81**	0.29	**0.35**	0.29	**0.34**
Optdigits	5216	64	5	295.38	**267.71**	162.28	**149.19**	0.73	**0.75**	0.75	**0.77**	0.00	0.00	0.00	0.00
Optdigits	5216	64	10	247.24	**224.82**	136.12	**125.54**	0.73	**0.75**	0.74	**0.75**	0.00	0.00	0.00	0.00
Optdigits	5216	64	30	825.23	**791.95**	110.06	**62.63**	0.73	**0.75**	0.73	**0.76**	0.00	0.00	0.00	0.00
Pendigits	6870	16	5	287.75	**282.25**	184.20	**158.26**	0.92	**0.95**	0.92	**0.94**	0.19	**0.23**	0.19	**0.20**
Pendigits	6870	16	10	281.49	**155.06**	179.83	**160.94**	0.92	**0.95**	0.92	**0.94**	0.19	**0.25**	0.19	**0.23**
Pendigits	6870	16	30	149.93	**145.59**	104.25	**89.85**	0.92	**0.94**	0.93	**0.94**	0.19	**0.25**	0.19	**0.22**
Pima	768	8	5	**28.72**	31.94	**21.16**	23.79	**0.71**	**0.71**	**0.71**	0.70	**0.51**	**0.51**	**0.53**	0.51
Pima	768	8	10	27.38	**20.15**	**20.81**	25.03	**0.71**	0.70	**0.71**	0.70	**0.51**	**0.51**	**0.51**	**0.51**
Pima	768	8	30	19.36	**17.89**	**13.83**	17.43	**0.71**	0.70	**0.71**	0.70	**0.51**	0.50	**0.52**	0.50
Shuttle	49097	9	5	3326.54	**1453.93**	2257.50	**1956.12**	**0.99**	**0.99**	**0.99**	**0.99**	**0.95**	**0.95**	**0.95**	**0.95**
Shuttle	49097	9	10	2437.10	**1396.21**	1549.97	**1321.16**	**0.99**	**0.99**	**0.99**	**0.99**	**0.95**	**0.95**	**0.95**	**0.95**
Shuttle	49097	9	30	1378.29	**1258.69**	837.41	**651.00**	**0.99**	**0.99**	**0.99**	**0.99**	**0.95**	**0.95**	**0.95**	**0.95**
SpamSpace	4207	57	5	247.98	**244.39**	130.95	**110.08**	**0.57**	0.56	**0.56**	**0.56**	**0.45**	**0.45**	**0.46**	0.45
SpamSpace	4207	57	10	233.39	**186.91**	128.24	**115.83**	**0.57**	0.56	**0.56**	**0.56**	**0.46**	0.45	**0.46**	**0.46**
SpamSpace	4207	57	30	604.00	**538.91**	70.19	**61.38**	**0.57**	0.56	**0.57**	0.56	**0.46**	**0.46**	**0.46**	0.45
Thyroid	3772	6	5	87.90	**71.34**	49.51	**48.20**	0.91	**0.93**	0.91	**0.93**	0.46	**0.54**	0.46	**0.55**
Thyroid	3772	6	10	74.76	**46.91**	44.81	**38.60**	0.91	**0.93**	0.91	**0.93**	0.46	**0.54**	0.46	**0.54**
Thyroid	3772	6	30	45.84	**43.86**	28.90	**26.75**	0.91	**0.93**	0.92	**0.93**	0.46	**0.54**	0.46	**0.54**
Waveform	3443	21	5	167.98	**147.00**	109.94	**94.46**	**0.78**	0.76	**0.78**	0.76	0.11	**0.13**	0.11	**0.13**
Waveform	3443	21	10	154.72	**94.36**	91.69	**55.17**	**0.78**	0.76	**0.78**	0.77	**0.11**	**0.11**	**0.11**	**0.11**
Waveform	3443	21	30	97.11	**95.77**	53.47	**48.04**	**0.78**	0.76	**0.78**	0.76	0.11	**0.13**	0.11	**0.13**

SUOD is applied on top of the aforementioned dataset (74,220 records are used for training and 49,500 records are set aside for validation). Similarly to the full framework evaluation in §4.4, the new system with SUOD (all three modules enabled) is compared with the current distributed system on 10 cores. The fit time is reduced from 6232.54 seconds to 4202.30 seconds (32.57% reduction), and the prediction time is reduced from 3723.45 seconds reduced to 2814.92 seconds (24.40%). In addition to the time reduction, ROC and P@N also show improvements at 3.59% and 7.46%, respectively. Through this case, we are confident the proposed framework can be useful for many real-world applications for scalable outlier detection.

5 Conclusion & Future Directions

In this work, we propose SUOD to expedite the training and prediction of a large number of unsupervised heterogeneous outlier detection models. It consists of three modules with focus on different levels (data, model, execution): (i) Random Projection module compresses data into low-dimensional subspaces to alleviate the curse of dimensionality; (ii) Pseudo-supervised Approximation module could accelerate costly unsupervised models' prediction by replacing them by faster supervised regressors, which also brings the extra benefit, e.g., interpretability and (iii) Balanced Parallel Scheduling module ensures that nearly equal amount of workload is assigned to available workers in distributed computing. The extensive experiments on more than 20 benchmark datasets and a real-world claim fraud analysis case show the great potential of SUOD, and many intriguing results are observed. For reproducibility and accessibility, all code, figures, and datasets are openly shared[1].

More investigations are currently underway. First, we plan to demonstrate SUOD's effectiveness as an end-to-end framework on more complex downstream combination models like unsupervised LSCP (Zhao et al., 2019a) and supervised XGBOD (Zhao & Hryniewicki, 2018). Second, we would further emphasize the interpretability provided by the pseudo-supervised approximation, which can be beyond feature importance provided in tree regressors. Third, there is room to investigate why and how the pseudo-supervised approximation could work in a more strict and theoretical way. This study, as the first step, empirically shows that proximity-based models can benefit from the approximation. , whereas linear models may not. Lastly, we may incorporate the emerging automated OD, e.g., MetaOD (Zhao et al., 2020), to trim down the model space for further acceleration.

[1]`https://github.com/yzhao062/SUOD`

REFERENCES

Achlioptas, D. Database-friendly random projections. In Buneman, P. (ed.), *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA*. ACM, 2001. doi: 10.1145/375551.375608. URL https://doi.org/10.1145/375551.375608.

Aggarwal, C. C. Outlier ensembles: position paper. *SIGKDD Explor.*, 14(2):49–58, 2012. doi: 10.1145/2481244.2481252. URL https://doi.org/10.1145/2481244.2481252.

Aggarwal, C. C. *Outlier Analysis*. Springer, 2013. ISBN 978-1-4614-6396-2. URL http://dx.doi.org/10.1007/978-1-4614-6396-2.

Aggarwal, C. C. and Sathe, S. *Outlier Ensembles - An Introduction*. Springer, 2017. ISBN 978-3-319-54764-0. doi: 10.1007/978-3-319-54765-7. URL https://doi.org/10.1007/978-3-319-54765-7.

Akoglu, L., Chandy, R., and Faloutsos, C. Opinion fraud detection in online reviews by network effects. In Kiciman, E., Ellison, N. B., Hogan, B., Resnick, P., and Soboroff, I. (eds.), *Proceedings of the Seventh International Conference on Weblogs and Social Media, ICWSM 2013, Cambridge, Massachusetts, USA, July 8-11, 2013*. The AAAI Press, 2013. URL http://www.aaai.org/ocs/index.php/ICWSM/ICWSM13/paper/view/5981.

Aldrich, N., Crowder, J., and Benson, B. How much does medicare lose due to fraud and improper payments each year. *The Sentinel*, 2014.

Amazon. The total cost of ownership (tco) of amazon sagemaker., 2020. URL https://pages.awscloud.com/rs/112-TZM-766/images/Amazon_SageMaker_TC_uf.pdf. Accessed: 2-21-2021.

Bagdoyan, S. J. Medicare: Actions needed to better manage fraud risks. *https://www.gao.gov/assets/700/693156.pdf*, 2018.

Breiman, L. Random forests. *Mach. Learn.*, 45(1):5–32, 2001. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. Lof: Identifying density-based local outliers. In Chen, W., Naughton, J. F., and Bernstein, P. A. (eds.), *SIGMOD Conference*, pp. 93–104. ACM, 2000. ISBN 1-58113-217-4. URL http://dblp.uni-trier.de/db/conf/sigmod/sigmod2000.html#BreunigKNS00. SIGMOD Record 29(2), June 2000.

Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, 2009. doi: 10.1145/1541880.1541882. URL https://doi.org/10.1145/1541880.1541882.

Goldstein, M. and Dengel, A. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: Poster and Demo Track*, pp. 59–63, 2012. URL https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.401.5686&rep=rep1&type=pdf.

He, Z., Xu, X., and Deng, S. Discovering cluster-based local outliers. *Pattern Recognit. Lett.*, 24(9-10):1641–1650, 2003. doi: 10.1016/S0167-8655(03)00003-5. URL https://doi.org/10.1016/S0167-8655(03)00003-5.

Hinton, G. E., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

Hu, J., Wang, F., Sun, J., Sorrentino, R., and Ebadollahi, S. A healthcare utilization analysis framework for hot spotting and contextual anomaly detection. In *AMIA 2012, American Medical Informatics Association Annual Symposium, Chicago, Illinois, USA, November 3-7, 2012*. AMIA, 2012. URL http://knowledge.amia.org/amia-55142-a2012a-1.636547/t-003-1.640625/f-001-1.640626/a-044-1.641102/a-045-1.641099.

Hu, X., Rudin, C., and Seltzer, M. I. Optimal sparse decision trees. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 7265–7273, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/ac52c626afc10d4075708ac4c778ddfc-Abstract.html.

Johnson, W. B. and Lindenstrauss, J. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

Keller, F., Müller, E., and Böhm, K. Hics: High contrast subspaces for density-based outlier ranking. In Kementsietsidis, A. and Salles, M. A. V. (eds.), *IEEE 28th International Conference on Data Engineering (ICDE*

2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012, pp. 1037–1048. IEEE Computer Society, 2012. doi: 10.1109/ICDE.2012.88. URL https://doi.org/10.1109/ICDE.2012.88.

Kriegel, H., Kröger, P., Schubert, E., and Zimek, A. Loop: local outlier probabilities. In Cheung, D. W., Song, I., Chu, W. W., Hu, X., and Lin, J. J. (eds.), *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, pp. 1649–1652. ACM, 2009. doi: 10.1145/1645953.1646195. URL https://doi.org/10.1145/1645953.1646195.

Kriegel, H.-P., Schubert, M., and Zimek, A. Angle-based outlier detection in high-dimensional data. In Li, Y., Liu, B., and Sarawagi, S. (eds.), *KDD*, pp. 444–452. ACM, 2008. ISBN 978-1-60558-193-4. URL http://dblp.uni-trier.de/db/conf/kdd/kdd2008.html#KriegelSZ08.

Lai, K., Zha, D., Wang, G., Xu, J., Zhao, Y., Kumar, D., Chen, Y., Zumkhawaka, P., Wan, M., Martinez, D., and Hu, X. TODS: an automated time series outlier detection system. *CoRR*, abs/2009.09822, 2020. URL https://arxiv.org/abs/2009.09822.

Lazarevic, A. and Kumar, V. Feature bagging for outlier detection. In Grossman, R., Bayardo, R. J., and Bennett, K. P. (eds.), *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pp. 157–166. ACM, 2005. doi: 10.1145/1081870.1081891. URL https://doi.org/10.1145/1081870.1081891.

Lazarevic, A., Ertöz, L., Kumar, V., Ozgur, A., and Srivastava, J. A comparative study of anomaly detection schemes in network intrusion detection. In Barbará, D. and Kamath, C. (eds.), *Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, CA, USA, May 1-3, 2003*, pp. 25–36. SIAM, 2003. doi: 10.1137/1.9781611972733.3. URL https://doi.org/10.1137/1.9781611972733.3.

Li, W., Wang, Y., Cai, Y., Arnold, C. W., Zhao, E., and Yuan, Y. Semi-supervised rare disease detection using generative adversarial network. *CoRR*, abs/1812.00547, 2018. URL http://arxiv.org/abs/1812.00547.

Li, Z., Zhao, Y., Botta, N., Ionescu, C., and Hu, X. COPOD: copula-based outlier detection. In Plant, C., Wang, H., Cuzzocrea, A., Zaniolo, C., and Wu, X. (eds.), *20th IEEE International Conference on Data Mining, ICDM 2020, Sorrento, Italy, November 17-20, 2020*, pp. 1118–1123. IEEE, 2020. doi: 10.1109/ICDM50108.2020.00135. URL https://doi.org/10.1109/ICDM50108.2020.00135.

Liu, F. T., Ting, K. M., and Zhou, Z. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pp. 413–422. IEEE Computer Society, 2008. doi: 10.1109/ICDM.2008.17. URL https://doi.org/10.1109/ICDM.2008.17.

Liu, Y., Li, Z., Zhou, C., Jiang, Y., Sun, J., Wang, M., and He, X. Generative adversarial active learning for unsupervised outlier detection. *IEEE Trans. Knowl. Data Eng.*, 32(8):1517–1528, 2020. doi: 10.1109/TKDE.2019.2905606. URL https://doi.org/10.1109/TKDE.2019.2905606.

Lozano, E. and Acuña, E. Parallel algorithms for distance-based and density-based outliers. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), 27-30 November 2005, Houston, Texas, USA*, pp. 729–732. IEEE Computer Society, 2005. doi: 10.1109/ICDM.2005.116. URL https://doi.org/10.1109/ICDM.2005.116.

Lu, W., Cheng, Y., Xiao, C., Chang, S., Huang, S., Liang, B., and Huang, T. S. Unsupervised sequential outlier detection with deep architectures. *IEEE Trans. Image Process.*, 26(9):4321–4330, 2017. doi: 10.1109/TIP.2017.2713048. URL https://doi.org/10.1109/TIP.2017.2713048.

Nakandala, S., Saur, K., Yu, G., Karanasos, K., Curino, C., Weimer, M., and Interlandi, M. A tensor compiler for unified machine learning prediction serving. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*, pp. 899–917. USENIX Association, 2020. URL https://www.usenix.org/conference/osdi20/presentation/nakandala.

Oku, J., Tamura, K., and Kitakami, H. Parallel processing for distance-based outlier detection on a multi-core cpu. In *IEEE International Workshop on Computational Intelligence and Applications (IWCIA)*, pp. 65–70. IEEE, 2014.

Ousterhout, K., Wendell, P., Zaharia, M., and Stoica, I. Sparrow: distributed, low latency scheduling. In Kaminsky, M. and Dahlin, M. (eds.), *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013*, pp. 69–84. ACM, 2013. doi: 10.1145/2517349.2522716. URL https://doi.org/10.1145/2517349.2522716.

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., and Duchesnay, E. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011. URL http://dl.acm.org/citation.cfm?id=2078195.

Ramaswamy, S., Rastogi, R., and Shim, K. Efficient algorithms for mining outliers from large data sets. In Chen, W., Naughton, J. F., and Bernstein, P. A. (eds.), *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pp. 427–438. ACM, 2000. doi: 10.1145/342009.335437. URL https://doi.org/10.1145/342009.335437.

Rayana, S. and Akoglu, L. Less is more: Building selective anomaly ensembles. *ACM Trans. Knowl. Discov. Data*, 10(4):42:1–42:33, 2016. URL http://dblp.uni-trier.de/db/journals/tkdd/tkdd10.html#RayanaA16.

Schubert, E., Zimek, A., and Kriegel, H. Fast and scalable outlier detection with approximate nearest neighbor ensembles. In Renz, M., Shahabi, C., Zhou, X., and Cheema, M. A. (eds.), *Database Systems for Advanced Applications - 20th International Conference, DASFAA 2015, Hanoi, Vietnam, April 20-23, 2015, Proceedings, Part II*, volume 9050 of *Lecture Notes in Computer Science*, pp. 19–36. Springer, 2015. doi: 10.1007/978-3-319-18123-3_2. URL https://doi.org/10.1007/978-3-319-18123-3_2.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., and Williamson, R. C. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, July 2001. doi: 10.1162/089976601750264965. URL https://doi.org/10.1162%2F089976601750264965.

Shyu, M.-L., Chen, S.-C., Sarinnapakorn, K., and Chang, L. A novel anomaly detection scheme based on principal component classifier. Technical report, 2003.

Spearman, C. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904. ISSN 00029556. URL http://www.jstor.org/stable/1412159.

Venkatasubramanian, S. and Wang, Q. The johnson-lindenstrauss transform: An empirical study. In Müller-Hannemann, M. and Werneck, R. F. F. (eds.), *Proceedings of the Thirteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2011, Holiday Inn San Francisco Golden Gateway, San Francisco, California, USA, January 22, 2011*, pp. 164–173. SIAM, 2011. doi: 10.1137/1.9781611972917.16. URL https://doi.org/10.1137/1.9781611972917.16.

Wang, Z., Kong, Z., Chandra, S., Tao, H., and Khan, L. Robust high dimensional stream classification with novel class detection. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*, pp. 1418–1429. IEEE, 2019a. doi: 10.1109/ICDE.2019.00128. URL https://doi.org/10.1109/ICDE.2019.00128.

Wang, Z., Li, H., Li, Z., Sun, X., Rao, J., Che, H., and Jiang, H. Pigeon: an effective distributed, hierarchical datacenter job scheduler. In *Proceedings of the ACM Symposium on Cloud Computing, SoCC 2019, Santa Cruz, CA, USA, November 20-23, 2019*, pp. 246–258. ACM, 2019b. doi: 10.1145/3357223.3362728. URL https://doi.org/10.1145/3357223.3362728.

Wang, Z., Wang, Y., Lin, Y., Delord, E., and Khan, L. Few-sample and adversarial representation learning for continual stream mining. In Huang, Y., King, I., Liu, T., and van Steen, M. (eds.), *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, pp. 718–728. ACM / IW3C2, 2020. doi: 10.1145/3366423.3380153. URL https://doi.org/10.1145/3366423.3380153.

Zha, D., Lai, K., Wan, M., and Hu, X. Meta-aad: Active anomaly detection with deep reinforcement learning. In Plant, C., Wang, H., Cuzzocrea, A., Zaniolo, C., and Wu, X. (eds.), *20th IEEE International Conference on Data Mining, ICDM 2020, Sorrento, Italy, November 17-20, 2020*, pp. 771–780. IEEE, 2020. doi: 10.1109/ICDM50108.2020.00086. URL https://doi.org/10.1109/ICDM50108.2020.00086.

Zhao, Y. and Hryniewicki, M. K. XGBOD: improving supervised outlier detection with unsupervised representation learning. In *2018 International Joint Conference on Neural Networks, IJCNN 2018, Rio de Janeiro, Brazil, July 8-13, 2018*, pp. 1–8. IEEE, 2018. doi: 10.1109/IJCNN.2018.8489605. URL https://doi.org/10.1109/IJCNN.2018.8489605.

Zhao, Y., Nasrullah, Z., Hryniewicki, M. K., and Li, Z. LSCP: locally selective combination in parallel outlier ensembles. In Berger-Wolf, T. Y. and Chawla, N. V. (eds.), *Proceedings of the 2019 SIAM International Conference on Data Mining, SDM 2019, Calgary, Alberta, Canada, May 2-4, 2019*, pp. 585–593. SIAM, 2019a. doi: 10.1137/1.9781611975673.66. URL https://doi.org/10.1137/1.9781611975673.66.

Zhao, Y., Nasrullah, Z., and Li, Z. Pyod: A python toolbox for scalable outlier detection. *J. Mach. Learn. Res.*, 20:96:1–96:7, 2019b. URL http://jmlr.org/papers/v20/19-011.html.

Zhao, Y., Rossi, R. A., and Akoglu, L. Automating outlier detection via meta-learning. *CoRR*, abs/2009.10606, 2020. URL https://arxiv.org/abs/2009.10606.

Zimek, A., Campello, R. J. G. B., and Sander, J. Ensembles for unsupervised outlier detection: challenges and research questions a position paper. *SIGKDD Explor.*, 15(1):11–22, 2013. doi: 10.1145/2594473.2594476. URL https://doi.org/10.1145/2594473.2594476.

Supplementary Material

Details on Datasets and Models.

A Datasets and Setup

Table A.1 describes the selected outlier detection benchmark datasets, and more than 20 outlier detection benchmark datasets are used in this study[1,2]. The data size n varies from 452 (**Arrhythmia**) to 567,479 (**HTTP**) samples and the dimension d ranges from 3 to 274. For both random projection and parallel scheduling experiments, the full datasets are used for model building (training). For the pseudo-supervised approximation experiments and the full framework assessment, 60% of the data is used for training and the remaining 40% is set aside for validation. For all experiments, performance is evaluated by taking the average of 10 independent trials using area under the receiver operating characteristic (ROC) curve and precision at rank n (P@N)—here n denotes the actual number of outliers. Both metrics are widely used in outlier research (Zimek et al., 2013; Liu et al., 2020).

Table A.1. Selected real-world benchmark datasets

Dataset	Pts (n)	Dim (d)	Outliers	% Outlier
Annthyroid	7200	6	534	7.41
Arrhythmia	452	274	66	14.60
Breastw	683	9	239	34.99
Cardio	1831	21	176	9.61
HTTP	567479	3	2211	0.40
Letter	1600	32	100	6.25
MNIST	7603	100	700	9.21
Musk	3062	166	97	3.17
PageBlock	5393	10	510	9.46
Pendigits	6870	16	156	2.27
Pima	768	8	268	34.90
Satellite	6435	36	2036	31.64
Satimage-2	5803	36	71	1.22
seismic	2584	10	170	6.59
Shuttle	49097	9	3511	7.15
SpameSpace	4207	57	1679	39.91
speech	3686	400	61	1.65
Thyroid	3772	6	93	2.47
Vertebral	240	6	30	12.50
Vowels	1456	12	50	3.43
Waveform	3443	21	100	2.90
Wilt	4819	5	257	5.33

B Outlier Detection Models

As shown in Table B.1, we use a large group of outlier detection models in the experiment by varying algorithms and their corresponding hyperparameters. We build the model pool $\mathcal{M}$ by sampling from it.

[1]ODDS Library: http://odds.cs.stonybrook.edu

[2]DAMI Datasets: http://www.dbs.ifi.lmu.de/research/outlier-evaluation/DAMI

Table B.1. Outlier Detection Models in SUOD; see parameter definitions from PyOD (Zhao et al., 2019b)

Method	**Parameter 1**	**Parameter 2**
ABOD (Kriegel et al., 2008)	n_neighbors: [3, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	N/A
CBLOF (He et al., 2003)	n_clusters: [3, 5, 10, 15, 20]	N/A
Feature Bagging (Lazarevic & Kumar, 2005)	n_estimators: [10, 20, 30, 40, 50, 75, 100, 150, 200]	N/A
HBOS (Goldstein & Dengel, 2012)	n_histograms: [5, 10, 20, 30, 40, 50, 75, 100]	tolerance: [0.1, 0.2, 0.3, 0.4, 0.5]
iForest (Liu et al., 2008)	n_estimators: [10, 20, 30, 40, 50, 75, 100, 150, 200]	max_features: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
kNN (Ramaswamy et al., 2000)	n_neighbors: [1, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	method: ['largest', 'mean', 'median']
LOF (Breunig et al., 2000)	n_neighbors: [1, 5, 10, 15, 20, 25, 50, 60, 70, 80, 90, 100]	method: ['manhattan', 'euclidean', 'minkowski']
OCSVM (Schölkopf et al., 2001)	nu (train error tol): [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]	kernel: ['linear', 'poly', 'rbf', 'sigmoid']