BOND: Benchmarking Unsupervised Outlier Node Detection on Static Attributed Graphs

Kay Liu[1,*], **Yingtong Dou**[1,8,*], **Yue Zhao**[2,*], **Xueying Ding**[2], **Xiyang Hu**[2], **Ruitong Zhang**[3], **Kaize Ding**[4], **Canyu Chen**[5], **Hao Peng**[3], **Kai Shu**[5], **Lichao Sun**[6], **Jundong Li**[7], **George H. Chen**[2], **Zhihao Jia**[2], **Philip S. Yu**[1]
[1]University of Illinois Chicago [2]Carnegie Mellon University
[3]Beihang University [4]Arizona State University [5]Illinois Institute of Technology
[6]Lehigh University [7]University of Virginia [8]Visa Research
benchmark@pygod.org

Abstract

Detecting which nodes in graphs are outliers is a relatively new machine learning task with numerous applications. Despite the proliferation of algorithms developed in recent years for this task, there has been no standard comprehensive setting for performance evaluation. Consequently, it has been difficult to understand which methods work well and when under a broad range of settings. To bridge this gap, we present—to the best of our knowledge—the first comprehensive benchmark for unsupervised outlier node detection on static attributed graphs called BOND, with the following highlights. (1) We benchmark the outlier detection performance of 14 methods ranging from classical matrix factorization to the latest graph neural networks. (2) Using nine real datasets, our benchmark assesses how the different detection methods respond to two major types of synthetic outliers and separately to "organic" (real non-synthetic) outliers. (3) Using an existing random graph generation technique, we produce a family of synthetically generated datasets of different graph sizes that enable us to compare the running time and memory usage of the different outlier detection algorithms. Based on our experimental results, we discuss the pros and cons of existing graph outlier detection algorithms, and we highlight opportunities for future research. Importantly, our code is freely available and meant to be easily extendable:
`https://github.com/pygod-team/pygod/tree/main/benchmark`

1 Introduction

Outlier detection (OD) on a graph refers to the task of identifying which nodes in the graph are outliers. This is a key machine learning (ML) problem that arises in many applications, such as social network spammer detection [77], sensor fault detection [27], financial fraudster identification [22], and defense against graph adversarial attacks [33]. Unlike classical OD on tabular and time-series data, graph OD has additional challenges: (1) the graph data structure in general carries richer information, and thus more powerful ML models are needed to learn informative representations, and (2) with more complex ML models, training can be more computationally expensive in terms of both running time and memory consumption [35, 47], posing challenges for time-critical (i.e., low time budget) and resource-sensitive (e.g., limited GPU memory) applications.

Despite the importance of graph OD and many algorithms being developed for it in recent years, *there is no comprehensive benchmark on graph outlier detection*, which we believe has hindered the development and understanding of graph OD algorithms. In fact, a recent graph OD survey calls for "system benchmarking" and describes it as "the key to evaluating the performance of graph OD techniques" [55]. We remark that there already are benchmarks for general graph mining (e.g.,

[*]Equal Contribution.

36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks.

OGB [30]), graph representation learning [26], graph robustness evaluation [95], graph contrastive learning [97], graph-level anomaly detection [85],[1] as well as benchmarks for tabular OD [6] and time-series OD [46]. These do not cover the specific task we consider, which we now formally define:

Definition 1 *(Unsupervised Outlier Node Detection on Static Attributed Graphs (abbreviated as OND)) A static attributed graph is defined as $G = (V, E, \mathbf{X})$, where $V = \{1, 2, \ldots, N\}$ is the set of vertices, $E \subseteq \{(i, j) : i, j \in V \text{ s.t. } i \neq j\}$ is the set of edges, and $\mathbf{X} \in \mathbb{R}^{N \times D}$ is the node attribute matrix (the i-th row of $\mathbf{X}$ is the feature vector in $\mathbb{R}^D$ corresponding to the i-th node in the graph). Given the graph G, the goal of the problem OND is to learn a function $f : V \to \mathbb{R}$ that assigns a real-valued outlier score to every node in G. The outlier nodes are then taken to be the k nodes with the highest outlier scores, for a user-specified value of k. This problem is unsupervised since in learning f, we do not have any ground truth information as to which nodes are outliers or not.*

Note that there are other graph OD problems (e.g., feature vectors could be time-dependent, there could be supervision in terms of some outliers being labeled, the nodes and edges could change over time, etc) but we focus on the problem OND stated above as it is the most prevalent [16, 55]. For OND, the status quo for how algorithms are developed has the following limitations:

- **Lack of a comprehensive benchmark:** often, only a limited selection of OND algorithms is tested on only a few datasets, making it unclear to what extent the empirical results generalize to a wider range of settings. Here, we remark that this issue of generalization is exacerbated by the fact that across different applications, what constitutes an outlier can vary drastically and, at the same time, also be difficult to precisely define in a manner that domain experts agree upon.
- **Limited outlier types taken into account:** typically, only a few types of outliers are considered (e.g., specific kinds of synthetic outliers are injected into real datasets), making it difficult to understand how graph OD algorithms respond to a wider variety of outlier nodes, including ones that are "organic" (non-synthetic).
- **Limited analyses of computational efficiency in both time and space:** Existing work mainly focuses on detection accuracy, with limited analyses of running time and memory consumption.

To address all the above limitations, we establish *the first comprehensive benchmark for the problem of OND* that we call BOND (short for benchmarking unsupervised outlier node detection on static attributed graphs). To accommodate many algorithms, we specifically create an open-source Python library for Graph Outlier Detection (PyGOD)[2], which provides more than ten of the latest graph OD algorithms, all with unified APIs and optimizations. Meanwhile, PyGOD also includes multiple non-graph baselines, resulting in a total of 14 representative and diverse methods for OND. We remark that this library can readily be extended to include additional OD algorithms.

Our work has the following highlights:

1. **The first comprehensive node-level graph OD benchmark**. We examine 14 OD methods, including classical and deep ones, and compare their pros and cons on nine benchmark datasets.
2. **Consolidated taxonomy of outlier nodes.** We group existing notions of outlier nodes into two main types: structural and contextual outliers. Our results show that most methods fail to balance the OD performance of these two major outlier types.
3. **Systematic performance flaw found for existing deep graph OD methods.** Surprisingly, our experimental results in BOND reveal that most of the benchmarked deep graph OD methods have suboptimal OD performance on organic outliers.
4. **Evaluation of both detection quality and computational efficiency**. In addition to common *effectiveness* metrics (e.g., ROC-AUC), we also measure the running time and GPU memory consumption of different algorithms as their *efficiency* measures.
5. **Reproducible and accessible benchmark toolkit**. To foster accessibility and fair evaluation for future algorithms, we make our code for BOND freely available at: `https://github.com/pygod-team/pygod/tree/main/benchmark`

We briefly describe existing approaches for OND in §2. We provide an overview of BOND in §3, followed by detailed experimental results and analyses in §4. We summarize the paper and discuss future work in §5.

[1] Graph-level anomaly detection refers to when we have a set of graphs and want to find which graphs are significantly different from the majority of graphs; in contrast, the graph OD we focus on in this paper is for detecting outliers at the node level for a specific graph.

[2] A Python library for Graph Outlier Detection (PyGOD): `https://pygod.org/`

2 Related Work

In this section, we briefly introduce related work on outlier node detection. Please refer to [2] and [55] for more comprehensive reviews of classical and deep-learning-based graph outlier detectors. We have implemented most of the discussed algorithms in this section in the PyGOD.

Classical (non-deep) outlier node detection. Real-world evidence suggests that the outlier nodes are different from regular nodes in terms of structure or attributes. Thus, early work on node outlier detection employs graph-based features such as centrality measures and clustering coefficients to extract the anomalous signals from graphs [2]. Instead of handcrafting features, learning-based methods have been used to more flexibly encode graph information to spot outlier nodes. Examples of these learning-based methods include ones based on matrix factorization (MF) [3, 50, 60, 69], density-based clustering [7, 29, 75], and relational learning [42, 63]. As most of the methods above have constraints on graph/node types or prior knowledge, we only include SCAN [75], Radar [50] and ANOMALOUS [60] in BOND to represent methods in this category.

Deep outlier node detection. The rapid development of deep learning and its use with graph data has shifted the landscape of outlier node detection from traditional methods to neural network approaches [55]. For example, the autoencoder (AE) [38], which is a neural network architecture devised to learn an encoding of the original data by trying to reconstruct the original data from the encoding, has become a popular model in detecting outlier nodes [16, 25, 39, 64, 70, 80]. AEs can be learned in an unsupervised manner as we are aiming to reconstruct the original data without separately trying to predict labels. The heuristic behind AE-based outlier detection is that we can use the AE reconstruction error as an outlier score; a data point that has a higher reconstruction error is likely more atypical.

More recently, graph neural networks (GNNs) have attained superior performance in many graph mining tasks [17, 18, 40, 71, 82]. GNNs aim to learn an encoding representation for every node in the graph, taking into account node attributes and also the underlying graph structure. The encoding representations learned by GNNs turn out to capture complex patterns that are useful for OD. As a result, GNNs have also become popular in detecting outlier nodes in graphs [19, 72, 87, 54, 76]. Note that GNNs can be combined with AEs; in constructing an AE, we need to specify encoder and decoder networks, which could be set to be GNNs.

We point out that it is also possible to adopt a Generative Adversarial Network (GAN) for outlier node detection [11]. GANs learn how to generate fake data that resemble real data by simultaneously learning a generator network (that can be used to randomly generate fake data) and a discriminator network (that tries to tell whether a data point is real or fake). Naturally, outliers could be deemed to be data points that are considered more "fake".

Among the many deep outlier node detectors, we have thus far implemented nine (see Table 2) for inclusion in BOND, where we have tried to have these be somewhat diverse in their methodology.

3 BOND

In this section, we provide an overview of BOND. We begin by defining two outlier types in §3.1. We then elaborate on the datasets (§3.2), algorithms (§3.3), and evaluation metrics (§3.4) in BOND.

3.1 Outlier Types

Many researchers have defined fine-grained outlier node types from different perspectives [2, 3, 16, 33, 50, 55]. In this paper, we group existing outlier node definitions into two major types according to real-world outlier patterns: *structural outliers* and *contextual outliers*, which are illustrated in Figure 1 and defined below.

Definition 2 *(Structural outlier) Structural outliers are densely connected nodes in contrast to sparsely connected regular nodes.*

Structural outliers arise in many real-world applications. For example, members of organized fraud gangs who frequently collude in carrying out malicious activities can be viewed as forming dense subgraphs of an overall graph (with nodes representing different people) [2]. As another example, coordinated bot accounts retweeting the same tweet will form a densely-connected co-retweet graph [29, 58]. Note that some papers [50, 55] also regard isolated nodes that do not belong to any communities as structural outliers (i.e., they have only a few edges connecting to any communities), which is different from our definition above. Since there is no existing OD method that we are aware of for detecting these isolated outlier nodes, we do not cover this type of outlier in BOND.

Definition 3 *(Contextual outlier) Contextual outliers are nodes whose attributes are significantly different from their neighboring nodes.*

An example of a contextual outlier is a compromised device in a computer network [2]. The definition of a contextual outlier is similar to how outliers are defined in classical proximity-based OD methods [1].

Some researchers call a node whose attributes differ from those of all other nodes as a *global outlier* [55] or a contextual outlier [50]. We do not consider these outliers in BOND as we find that they do not actually use the graph structure; instead, these outliers could be detected using tabular outlier detectors [28, 92].

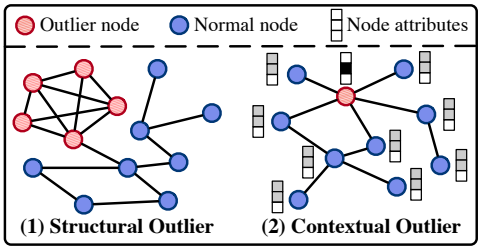

Figure 1: An illustration of structural vs. contextual outliers.

What we defined as a contextual outlier in Definition 3 is also referred to as an attribute outlier [16] or a community outlier [50, 55] in previous work. We argue that calling these *contextual outliers* is a more accurate terminology. The reason is that an "attribute outlier" sounds like it only depends on attributes (i.e., feature vectors), which would correspond to a global outlier [55], but confusingly this is not what is meant by the terminology. Meanwhile, the terminology of a "community" in graph theory has often been in reference to the density of edges among nodes and so a "community outlier" might be misconstrued to be what we call a *structural outlier* as in Definition 2. For the remainder of the paper, by "structural" and "contextual" outliers, we always go by Definitions 2 and 3 respectively.

Importantly, note that in real datasets, the organic (non-synthetic) outlier nodes present do not need to strictly be either a structural or a contextual outlier. In fact, what precisely makes them an outlier need not be explicitly stated and they could be neither a structural nor a contextual outlier, or they could even appear as a mixture of these two types! This makes detecting organic outliers more difficult than detecting synthetic outliers that follow a specific pattern such as those of Definitions 2 and 3.

3.2 Datasets

To comprehensively evaluate the performance of existing OND algorithms, we have investigated various real datasets with organic outliers used in previous literature. Note that some standard datasets are beyond the scope of the problem OND that we consider or do not make use of either the graph structure or node attributes/feature vectors. For example, YelpChi-Fraud [22], Amazon-Fraud [22], and Elliptic [74] are three graph datasets designed for supervised node classification; however, the fraudulent nodes have limited outlier pattern in terms of graph structure. Bitcoin-OTC, Bitcoin-Alpha, Epinions, and Amazon-Malicious from [43] are four bipartite graphs where nodes do not have attributes. DARPA [78], UCI Message [94], and Digg [94] are three dynamic graphs with organic edge outliers which are also out of our problem scope.

In BOND, we use the following datasets. First, since there are a limited number of open-source graph datasets with organic outlier nodes, we include three real datasets with no organic outliers that we inject synthetic outlier nodes into. Specifically, we use node classification benchmark datasets (**Cora** [66], **Amazon** [67], and **Flickr** [81]) from three domains with different scales. Next, we use six real datasets that contain organic outliers (**Weibo** [86], **Reddit** [44, 73], **Disney** [65], **Books** [65], **Enron** [65], and **DGraph** [32]). Finally, we also use purely synthetic data generated using the random algorithm by [36] that is able to produce graphs with varying scales; this random generation procedure provides a controlled manner in which we can evaluate different OD algorithms' computational efficiency in terms of both running time and memory usage. Some basic statistics for the real datasets used are given in Table 1 with more dataset details available in Appx. A.1.

To make synthetic outlier nodes of the two types we defined in §3.1 and to "camouflage" them so that they are more difficult to detect using simple OD methods, we slightly modify a widely-used approach [16, 25, 11, 80] (described below). These synthetic outliers are used with the real datasets that lack organic outliers (Cora, Amazon, Flickr) and also with the randomly generated graph data. In the random outlier injection procedures to follow, for the given graph G that we are working with, we treat the vertex set as fixed. To inject structural outliers, we modify the edges present, whereas to inject contextual outliers, we modify the feature vectors of randomly chosen nodes.

Injecting random structural outliers. The basic strategy is to create n non-overlapping densely connected groups of nodes, where each group has exactly m nodes (so that there are a total of $m \times n$

Table 1: Statistics of real datasets used in BOND (* indicates that outliers are synthetically injected).

Dataset	#Nodes	#Edges	#Feat.	Degree	#Con.	#Strct.	#Outliers	Ratio
Cora* [66]	2,708	11,060	1,433	4.1	70	70	138	5.1%
Amazon* [67]	13,752	515,042	767	37.2	350	350	694	5.0%
Flickr* [81]	89,250	933,804	500	10.5	2,240	2,240	4,414	4.9%
Weibo [86]	8,405	407,963	400	48.5	-	-	868	10.3%
Reddit [44, 73]	10,984	168,016	64	15.3	-	-	366	3.3%
Disney [65]	124	335	28	2.7	-	-	6	4.8%
Books [65]	1,418	3,695	21	2.6	-	-	28	2.0%
Enron [65]	13,533	176,987	18	13.1	-	-	5	0.4‰
DGraph [32]	3,700,550	4,300,999	17	1.2	-	-	15,509	0.4%

Table 2: Algorithms implemented in BOND and their characteristics: whether designed for graphs (row 3), whether neural networks are used (row 4), and what the core idea for the method (row 5).

Alg.	LOF	IF	MLPAE	SCAN	Radar	ANOMA-LOUS	GCNAE	DOMI-NANT	DONE/ AdONE	Anomaly-DAE	GAAN	GUIDE	CONAD
Year	2000	2012	2014	2007	2017	2018	2016	2019	2020	2020	2020	2021	2022
Graph	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Deep	✗	✗	✓	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓
Core	N/A	Tree	MLP+AE	Cluster	MF	MF	GNN+AE	GNN+AE	MLP+AE	GNN+AE	GAN	GNN+AE	GNN+AE
Ref.	[5]	[52]	[64]	[75]	[50]	[60]	[39]	[16]	[4]	[25]	[11]	[80]	[76]

structural outliers injected). To do this, for each $i = 1, \ldots, n$, we randomly sample m nodes to form the i-th group (these m nodes are sampled uniformly at random from nodes that have not been previously chosen to form a group); for these m nodes, we first make them fully connected and then drop each edge independently with probability p.

Injecting random contextual outliers. To inject a total of o contextual outliers, we first sample o nodes from the vertex set V without replacement; these are the nodes whose attributes we aim to modify as to turn them into contextual outliers. We denote the set of these o nodes as V_c (so that $o = |V_c|$), and refer to the remaining nodes $V_r := V \setminus V_c$ as the "reference" set. For each node $i \in V_c$, we randomly choose q nodes without replacement uniformly at random from the reference set V_r. Among these q reference nodes chosen, we find the one whose attributes deviate the most (in terms of Euclidean distance) from those of node i. We then change the attributes of node i to be the same as those of this most dissimilar reference node found.

For more details about synthetic outlier injection, see Appx A.1.3.

3.3 Algorithms

Table 2 lists the 14 algorithms evaluated in the benchmark and their properties. Our principle for selecting algorithms to implement in BOND is to cover representative methods in terms of the published time ("Year"), whether they use the graph structure ("Graph"), whether they use neural networks ("Deep"), and what the core idea is behind the method ("Core"). By including non-graph OD algorithms (LOF, IF, MLPAE), we can investigate the advantages and deficiencies of graph-based vs. non-graph-based OD algorithms in detecting outlier nodes. Similarly, incorporating three classical OD methods (clustering-based SCAN, MF-based Radar, and ANOMALOUS) helps us understand the performance of classical vs. deep OD methods. We select a wide array of GNN-based methods including the vanilla GCNAE; the classic DOMINANT; AnomalyDAE, an improved version of DOMINANT; and also GUIDE and CONAD, two state-of-the-art methods in this category with different data augmentation techniques. Besides GNN-based BOND methods, two methods encoding graph information using other models (DONE/AdONE and GAAN) are also included. Please refer to Appx. A.2 for a more detailed introduction of the methods benchmarked in BOND.

3.4 Evaluation Metrics

Detection quality measures. We follow the extensive literature in graph OD [15, 69, 87] to comprehensively evaluate the outlier node detection quality with three metrics: (1) ROC-AUC reflects detectors' performance on both positive and negative examples, while (2) Average Precision focuses more on positive examples, and (3) Recall@k evaluates the examples with high predicted outlier scores. See Appx. A.3 for more details.

Efficiency measures in time and space. Another important aspect of graph-based algorithms is their high time and space complexity [20, 35], which imposes additional challenges for large, high-dimensional datasets on hardware like GPUs with limited memory (e.g., out-of-memory errors).

Table 3: ROC-AUC (%) comparison among OD algorithms on three datasets with synthetic outliers, where we show *the avg perf.* ± *the STD of perf.* (*max perf.*) of each. The best algorithm by expectation is shown in **bold**, while the max performance per dataset is marked with underline. OOM denotes out of memory with regard to GPU (_G) and CPU (_C).

Algorithm	Cora	Amazon	Flickr
LOF	69.9±0.0 (69.9)	55.2±0.0 (55.2)	41.6±0.0 (41.6)
IF	64.4±1.5 (67.4)	51.3±3.0 (57.9)	57.1±1.1 (58.8)
MLPAE	70.9±0.0 (70.9)	74.2±0.0 (74.2)	72.4±0.0 (72.5)
SCAN	62.8±4.5 (72.6)	62.2±4.9 (71.1)	62.4±12.4 (75.0)
Radar	65.0±1.3 (66.0)	71.8±1.1 (73.4)	OOM_G
ANOMALOUS	55.0±10.3 (68.0)	72.5±1.5 (75.5)	OOM_G
GCNAE	70.9±0.0 (70.9)	74.2±0.0 (74.2)	71.6±3.1 (72.4)
DOMINANT	82.7±5.6 (84.3)	81.3±1.0 (82.2)	78.0±12.0 (84.6)
DONE	82.4±5.6 (87.9)	82.8±8.8 (93.7)	**84.7**±2.5 (89.0)
AdONE	81.5±4.5 (87.4)	**86.6**±5.6 (92.3)	82.8±3.2 (89.0)
AnomalyDAE	**83.4**±2.3 (85.3)	85.7±2.9 (90.8)	65.6±3.5 (70.4)
GAAN	74.2±0.9 (76.1)	80.8±0.3 (81.5)	72.4±0.2 (72.5)
GUIDE	74.7±1.3 (77.5)	OOM_C	OOM_C
CONAD	78.8±9.6 (84.3)	80.5±4.0 (82.2)	65.1±2.5 (67.4)

Table 4: ROC-AUC (%) comparison among OD algorithms on six datasets with organic outliers, where we show *the avg perf.* ± *the STD of perf.* (*max perf.*) of each. The best algorithm by expectation is shown in **bold**, while the max performance per dataset is marked with underline. OOM denotes out of memory with regard to GPU (_G) and CPU (_C). TLE denotes time limit of 24 hours exceeded.

Algorithm	Weibo	Reddit	Disney	Books	Enron	DGraph
LOF	56.5±0.0 (56.5)	**57.2**±0.0 (57.2)	47.9±0.0 (47.9)	36.5±0.0 (36.5)	46.4±0.0 (46.4)	TLE
IF	53.5±2.8 (57.5)	45.2±1.7 (47.5)	**57.6**±2.9 (63.1)	43.0±1.8 (47.5)	40.1±1.4 (43.1)	**60.9**±0.7 (62.0)
MLPAE	82.1±3.6 (86.1)	50.6±0.0 (50.6)	49.2±5.7 (64.1)	42.5±5.6 (52.6)	73.1±0.0 (73.1)	37.0±1.9 (41.3)
SCAN	63.7±5.6 (70.8)	49.9±0.3 (50.0)	50.5±4.0 (56.1)	49.8±1.7 (52.4)	52.8±3.4 (58.1)	TLE
Radar	**98.9**±0.1 (99.0)	54.9±1.2 (56.9)	51.8±0.0 (51.8)	52.8±0.0 (52.8)	**80.8**±0.0 (80.8)	OOM_C
ANOMALOUS	**98.9**±0.1 (99.0)	54.9±5.6 (60.4)	51.8±0.0 (51.8)	52.8±0.0 (52.8)	**80.8**±0.0 (80.8)	OOM_C
GCNAE	90.8±1.2 (92.5)	50.6±0.0 (50.6)	42.2±7.9 (52.7)	50.0±4.5 (57.9)	66.6±7.8 (80.1)	40.9±0.5 (42.2)
DOMINANT	85.0±14.6 (92.5)	56.0±0.2 (56.4)	47.1±4.5 (54.9)	50.1±5.0 (58.1)	73.1±8.9 (85.0)	OOM_C
DONE	85.3±4.1 (88.7)	53.9±2.9 (59.7)	41.7±6.2 (50.6)	43.2±4.0 (52.6)	46.7±6.1 (67.1)	OOM_C
AdONE	84.6±2.2 (87.6)	50.4±4.5 (58.1)	48.8±5.1 (59.2)	53.6±2.0 (56.1)	44.5±2.9 (53.6)	OOM_C
AnomalyDAE	91.5±1.2 (92.8)	55.7±0.4 (56.3)	48.8±2.2 (55.4)	**62.2**±8.1 (73.2)	54.3±11.2 (69.1)	OOM_C
GAAN	92.5±0.0 (92.5)	55.4±0.4 (56.0)	48.0±0.0 (48.0)	54.9±5.0 (61.9)	73.1±0.0 (73.1)	OOM_C
GUIDE	OOM_C	OOM_C	38.8±8.9 (52.5)	48.4±4.6 (63.5)	OOM_C	OOM_C
CONAD	85.4±14.3 (92.7)	56.1±0.1 (56.4)	48.0±3.5 (53.1)	52.2±6.9 (62.9)	71.9±4.9 (84.9)	34.7±1.2 (36.5)

Therefore, we measure efficiency in time and space respectively, we use (1) wall-clock time and (2) GPU memory consumption. We provide more details in Appx. A.3.

4 Experiments

We design BOND to understand the detection effectiveness and efficiency of various OD algorithms in addressing the problem OND. Specifically, we aim to answer: **RQ1** (§4.1): How effective are the algorithms on detecting synthetic and organic outliers? **RQ2** (§4.2): How do algorithms perform under two types of synthetic outliers (structural and contextual)? **RQ3** (§4.3): How efficient are algorithms in terms of time and space? Note that due to space constraints, for detection quality, we focus on the ROC-AUC metric, deferring results using the AP and Recall@k metrics to Appx. C.

Model implementation and environment configuration. Most algorithms in BOND are implemented via our newly released PyGOD package [53], and non-graph OD methods are imported from our earlier work [92]. Although we tried our best to apply the same set of optimization techniques, e.g., vectorization, to all methods, we suspect that further code optimization is possible. For more implementation details and environment configurations, see Appx. B.

Hyperparameter grid. In real-world settings, it is unclear how to do hyperparameter tuning and algorithm selection for unsupervised outlier detection due to the lack of ground truth labels and/or universal criteria that correlates well with the ground truth [55, 93]. For fair evaluation, when we report performance metrics in tables, we apply the same hyperparameter grid (see Appx. B) to each applicable algorithm and report its *avg. performance* (i.e., "algorithm performance in expectation"), along with the *standard deviation* (i.e., "algorithm stability") and the *max* (i.e., "algorithm potential").

4.1 Experimental Results: Detection Performance on Synthetic and Organic Outliers

Using the nine real datasets described in §3.2, we report the ROC-AUC score of different OD algorithms in Tables 3 and 4. Below are the key findings from these tables.

In terms of avg. performance, no outlier node detection method is universally the best on all datasets. Tables 3 and 4 show that only three of 14 methods evaluated (AnomalyDAE, Radar, ANOMALOUS) have the best avg. performance (for the ROC-AUC metric) on two datasets (Cora and Books for AnomalyDAE; Weibo and Enron for Radar and ANOMALOUS). The classical methods Radar and ANOMALOUS both have the best performance on Weibo and Enron (see Table 4) but they are worse than many deep learning methods on detecting synthetic outliers (see Table 3). Additionally, there is a substantial performance gap between the best- and worst-performing algorithms, e.g., DONE achieves $2.06\times$ higher average ROC-AUC compared to LOF on Flickr.

Most methods evaluated fail to detect organic outliers. Since most methods we evaluated are designed to handle structural and contextual outliers as defined in §3.1, to figure out the reason behind the failure and success in detecting organic outliers, we analyze the organic outlier patterns in terms of metrics related to the definitions of structural and contextual outliers. We first show that the success of most methods on Weibo (see Table 4) is because the outliers in Weibo exhibit the properties of both structural and contextual outliers. Specifically, in Weibo, the average clustering coefficient [24] of the outliers is higher than that of inliers (0.400 vs. 0.301), meaning that these outliers correspond to structural outliers. Meanwhile, the average neighbor feature similarity [22] of the outliers is far lower than that of inliers (0.004 vs. 0.993), so that the outliers also correspond to contextual outliers. In contrast, the outliers in the Reddit and DGraph datasets have similar average neighbor feature similarities and clustering coefficients for outliers and inliers. Therefore, their abnormalities rely more on outlier annotations with domain knowledge, and so supervised OD methods are more effective than unsupervised ones on Reddit (best AUC: 0.746 in [73] vs. 0.604 in Table 4) and DGraph (best AUC: 0.792 in [32] vs. 0.620 in Table 4) than unsupervised ones.

Deep learning methods and other methods using SGD may be sub-optimal on small graphs. The outliers on Disney, Books, and Enron also have similar outlier patterns defined in §3.1. However, most of the deep learning methods evaluated do not work particularly well on Disney and Enron compared to classical baselines. The reason is that Disney and Books have small graphs in terms of #Nodes, #Edges, and #Feat. (see Table 1). The small amount of data could make it difficult for the deep learning methods to encode the inlier distribution well and could also possibly lead to overfitting issues. Meanwhile, classical methods Radar and ANOMALOUS also perform poorly on Disney and Books; these methods use SGD, which we suspect could be problematic for such small datasets.

Different categories of methods evaluated are good at detecting different types of outliers. According to Tables 3 and 4, many deep graph-based methods are good at detecting synthetic outliers but are useless in detecting organic outliers. Meanwhile, non-graph-based methods have advantages when outliers do not follow taxonomies (Reddit and DGraph). These observations corroborate our understanding of unsupervised OD algorithms—their effectiveness depends on whether the underlying data distribution satisfies structural properties that the algorithms exploit.

In terms of standard deviation of the ROC-AUC metric, among the deep learning methods, some are noticeably less stable than others.[1] Certain deep learning methods, e.g., GAAN, exhibit insensitivity to hyperparameters, where the ROC-AUC standard deviation is mostly below 1%. Meanwhile, the methods that are unstable tend to involve more complex loss terms (e.g., weighted combination of multiple losses); for instance, DONE and AdONE achieve the highest *max* performance (i.e., the numbers in parentheses in Tables 3 and 4, and not the *avg.* performance) among deep graph methods on three datasets, while showing high ROC-AUC standard deviation across hyperparameters tested. Here, we emphasize that in practice for unsupervised OD, there being no labels means that hyperparameter tuning is far less straightforward, so stability (in OD detection quality) across hyperparameters is a desirable property of an algorithm.

4.2 Experimental Results: Detection Performance on Structural and Contextual Outliers

We report the ROC-AUC metric of different algorithms on three datasets with two types of injected synthetic outliers (contextual and structural outliers from §3.2) in Table 5. Note that we only consider synthetic outliers here since we have generated them so that we know exactly which nodes are contextual vs. structural outliers. Our main findings are as follows.

[1] For a specific method, the standard deviation of the method's ROC-AUC scores across hyperparameters (what we have called "algorithm stability") and the maximum ("algorithm potential") are in general monotonically related (i.e., when the standard deviation increases, then the maximum *minus the mean* also tends to increase; and vice versa), so that our finding here is both for algorithm stability and potential.

Table 5: ROC-AUC (%) comparison among OD algorithms on three datasets injected with contextual and structural outliers, where we show *the avg perf.* ± *the STD of perf.* (*max perf.*) of each. The best algorithm by expectation is shown in **bold**, while the max performance per dataset is marked with underline. OOM denotes out of memory with regard to GPU (_G) and CPU (_C). Reconstruction-based MLPAE and GCNAE perform best w.r.t contextual outliers, while there is no universal winner for both types of outliers.

	Cora		Amazon		Flickr	
Algorithm	Contextual	Structural	Contextual	Structural	Contextual	Structural
LOF	87.1±0.0 (87.1)	52.4±0.0 (52.4)	61.9±0.0 (61.9)	48.0±0.0 (48.0)	34.2±0.0 (34.2)	49.1±0.0 (49.1)
IF	77.5±2.2 (81.8)	51.4±2.3 (56.2)	51.8±6.0 (64.3)	50.9±0.8 (52.2)	63.1±2.1 (66.5)	50.9±0.3 (51.5)
MLPAE	**88.9**±0.0 (88.9)	52.5±0.0 (52.5)	**98.6**±0.0 (98.6)	49.0±0.0 (49.0)	**94.4**±0.1 (94.5)	50.0±0.1 (50.3)
SCAN	49.8±0.5 (51.7)	80.0±13.4 (95.9)	48.7±1.1 (49.9)	78.0±11.9 (94.0)	50.2±0.1 (50.3)	86.8±21.1 (99.7)
Radar	50.2±0.6 (51.0)	78.4±3.4 (81.6)	84.9±3.7 (88.2)	59.0±1.7 (61.3)	OOM_G	OOM_G
ANOMALOUS	51.1±1.3 (53.5)	69.3±16.2 (90.8)	85.4±0.9 (87.2)	59.5±2.5 (62.7)	OOM_G	OOM_G
GCNAE	**88.9**±0.0 (88.9)	52.5±0.0 (52.5)	**98.6**±0.0 (98.6)	49.0±0.0 (49.0)	88.7±9.2 (94.5)	50.0±0.2 (50.3)
DOMINANT	71.9±6.6 (74.4)	93.0±4.6 (95.3)	69.0±3.6 (71.3)	93.6±2.8 (94.4)	69.0±4.5 (71.0)	**96.3**±10.4 (98.9)
DONE	70.2±8.3 (80.0)	92.8±5.6 (99.8)	82.4±11.1 (95.1)	90.2±8.0 (99.4)	85.7±2.1 (88.7)	85.5±3.1 (89.1)
AdONE	73.9±5.0 (78.0)	91.3±4.8 (97.3)	78.0±10.6 (95.1)	85.7±11.8 (97.0)	80.2±4.3 (88.1)	87.1±1.9 (89.7)
AnomalyDAE	80.2±2.8 (87.2)	90.2±4.6 (95.9)	88.2±9.6 (98.3)	85.5±10.1 (94.3)	80.0±7.4 (93.3)	56.6±1.7 (59.0)
GAAN	88.7±0.1 (88.8)	61.0±0.8 (62.5)	98.5±0.1 (98.6)	64.2±2.0 (67.3)	94.1±0.3 (94.4)	50.3±0.3 (50.9)
GUIDE	88.3±0.8 (88.7)	61.8±2.4 (71.1)	OOM_C	OOM_C	OOM_C	OOM_C
CONAD	72.5±5.8 (74.4)	**93.6**±4.8 (95.4)	69.4±2.8 (71.3)	**94.1**±0.4 (94.3)	65.8±0.9 (67.4)	68.3±0.6 (69.1)

For GNNs, the reconstruction of the structural information appears to play a significant role in detecting structural outliers. Specifically, the performance gap on structural outliers between GCNAE and DOMINANT is over 40%. By taking a closer look into these two algorithms, they differ only in that DOMINANT has a structural decoder that aims to reconstruct the adjacency matrix of the graph. That DOMINANT performs significantly better than GCNAE suggests that reconstructing information on graph structure is helpful in identifying structural outliers, which intuitively makes sense as these outliers are defined in terms of graph structure.

Low-order structural information (i.e., one-hop neighbors) is sufficient for detecting structural outliers. DOMINANT and DONE achieve similar mean ROC-AUC scores (∼92%) on detecting structural outliers in the Cora and Amazon datasets even though DOMINANT encodes 4-hop neighbor information whereas DONE only encodes 1-hop neighbor information. This observation could facilitate outlier node detection model design since encoding high-order information usually imposes a higher computational cost, and multi-hop neighbor aggregation may even lead to the over-smoothing problem in GNNs [8].

No method achieves high detection accuracy for both structural and contextual outliers. For instance, none of the methods reaches 85% detection AUC on both structural and contextual outliers. Moreover, on Flickr with structural outliers injected, most attributed graph OD methods that are supposed to detect structural outliers have worse average ROC-AUC scores than that of SCAN, whereas SCAN is a non-attributed graph OD method detecting structural outliers by clustering on nodes. The above result suggests that the common approach that arbitrarily combines the structural and contextual loss terms with fixed weights (that are hyperparameters) can struggle to balance performance in detecting both outlier types. How to detect these two different types of outliers consistently well remains an open question.

We visualize the efficiency in time (wall clock running time) and space (GPU memory consumption) of selected algorithms in Figure 2. The complete results are available in Appx. C.3. All algorithms are evaluated under randomly generated graphs with the same sets of injected outliers to guarantee fairness. The running time in Figure 2 (left) is the sum of training and computing outlier scores. We only measure the GPU memory consumption of the different methods (as opposed to the CPU memory consumption) because it is often the bottleneck of OD algorithms [89]. For more details on the generated graphs and experimental settings of this section, see Appx. A.1.2 and Appx. B, respectively. Our key findings from Figure 2 are as follows.

4.3 Experimental Results: Computational Efficiency

Time efficiency. Classical methods that we evaluated take less time than the deep learning ones, which tend to be more complicated and learn more flexible models. Among the GNN-based methods, GUIDE consumes far more time compared to others. The reason is that GUIDE uses a graph motif counting algorithm (which is #P-complete [13]) to extract the structural features and consumes much more time on the CPU. CONAD takes the second-most amount of time due to its use of contrastive learning (which uses pairwise comparisons within mini-batches).

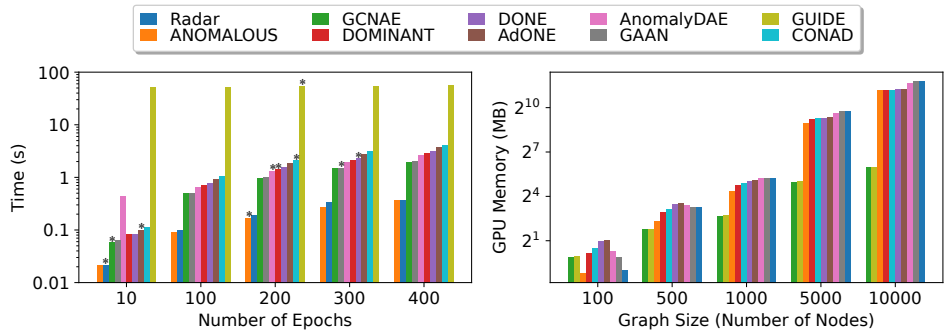

Figure 2: Wall-clock running time (left) and GPU memory consumption (right) of different methods. ($*$ denotes the best performance of each method among five different numbers of epochs.)

Space efficiency. According to Figure 2 (right), GCNAE and GUIDE consume much less GPU memory than the other methods as the graph size increases. GCNAE saves more memory due to its simpler architecture. GUIDE consumes more CPU time and RAM to extract low-dimensional node motif degrees, thereby saving more GPU memory. Though classical methods have the advantage in terms of running time, most of them cannot be deployed in a distributed fashion due to the limitation of "global" operators like matrix factorization and inversion. One advantage of deep models is that they can be easily extended to minibatch and distributed training via graph sampling. Another advantage is that deep methods can be easily integrated with existing deep learning pipelines (e.g., graph pretraining module that obtains node embeddings).

5 Discussion

We have established BOND, the first comprehensive benchmark for unsupervised outlier node detection on static attributed graphs. Our benchmark has empirically examined the effectiveness of a diverse collection of OD algorithms in terms of synthetic vs. organic outliers, structural vs. contextual outliers, and computational efficiency. Importantly, a major goal in our development of BOND is to make it easy to extend so that further progress can be made in better understanding existing algorithms and developing new ones to address OND. We conclude this paper by discussing future research directions for OND in general (§5.1), and then specific to benchmarking (§5.2).

5.1 Future Directions in Addressing the Problem OND

Our experimental results on real data (§4.1 and §4.1) reveal substantial detection performance differences between algorithms, with none of them being the universal winner. Even for a single algorithm, there is also the issue of hyperparameter tuning (e.g., the detection performance of AnomalyDAE on Weibo varies by as much as 14% across hyperparameters). However, the fundamental problem is that because the problem OND is unsupervised, it is not straightforward deciding on the "right" choice of algorithm or hyperparameter setting. Any quantitative metric we define to help with model selection or hyperparameter tuning will require assumption(s). Separately, the available computational budget and the size of the dataset to be analyzed can limit what methods can even be used (our results in §4.3 show that some methods are much more computationally expensive than others).

Opportunity 1: designing "type-aware" detection algorithms. A major finding of our experimental results is that which OD algorithm works best heavily depends on the type of outlier encountered (synthetic vs. organic, structural vs. contextual). Put another way, if we expect to see a particular type of outlier, then this should inform the choice of OD algorithm to apply (e.g., MLPAE and GCNAE for contextual outliers). Of course, this would require us to have some a priori knowledge or guess as to what the outliers look like in a dataset. Importantly, in real applications, we might not need to detect *all* outliers. For example, practitioners may want to focus only on high-value or high-interest outliers (e.g., illegal trades that affect revenue the most [37] can be considered as contextual outliers, so MLPAE and GCNAE could be good choices). Following recent advances in categorizing outlier types in tabular data [34], we call for attention to identifying more fine-grain outlier types in OND and figuring out which algorithms are well-suited to these different outlier types. Once we have this sort of information, there could be opportunities for automatically choosing a single or combining multiple OD algorithms, accounting for outlier types (e.g., using an ensembling approach like [91]).

Opportunity 2: synthesizing more realistic and flexible outlier nodes. The organic outliers encountered in our real data experiments (§4.1) can be complex and composed of multiple outlier types

(possibly including types beyond just the structural and contextual ones we focused on). Our experimental results show that there is a wide detection performance gap on synthetic vs. organic outliers, calling for more realistic outlier generation approaches. To improve existing generation methods in BOND, we could use a generative model to fit the normal samples, and then perturb the generative model to generate different types of outliers. This approach has been successful in tabular OD [68].

Opportunity 3: understanding the sensitivity of OND algorithms to hyperparameters and designing more stable methods. We had pointed out in §4.1 that some deep learning methods are more stable (across hyperparameters) than others in terms of standard deviation in achieved ROC-AUC scores. This phenomenon extends of course to classical methods as well. Better understanding the drivers of these algorithms' (in)sensitivity to hyperparameters would help us better design algorithms that are more stable with respect to hyperparameter settings. In turn, this could help ease the burden of unsupervised hyperparameter tuning. In tabular OD tasks, researchers have developed methods for outlier detection that are more stable with respect to hyperparameters such as the robust autoencoder [96], RandNet [9], ROBOD [21], and ensemble frameworks [91]. Perhaps some techniques from these tabular OD methods could be incorporated into specific methods for OND to improve stability.

Opportunity 4: developing more efficient OND algorithms. Our results on computational efficiency (§4.3) show that some algorithms take substantially more time and/or memory to execute than others. Meanwhile, most algorithms tested ran out of memory on the million-scale DGraph dataset (Table 4). We suggest developing more scalable algorithms for OND, which could mean more optimized implementations of existing algorithms and also the development of new algorithms. We point out several lines of work that could be helpful in this endeavor. First, there are existing approaches for making GNNs more scalable [35, 49] but these have yet to be specialized to address OND. Separately, most existing autoencoder-based methods that we tested reconstruct a complete graph adjacency matrix (e.g., DOMINANT, DONE, AnomalyDAE, GAAN), which is memory intensive (scaling quadratically with the number of nodes). Developing a more memory-efficient implementation of this step would be interesting. Next, approximating the node motif degree in GUIDE is possible, which can significantly reduce both computation time and space [10, 79]. Lastly, we mention that some recent work [90, 89] accelerates tabular OD via distributed learning, data and model compression, and/or quantization. These ideas could be extended to algorithms for OND.

Opportunity 5: meta-learning to assist model selection and hyperparameter tuning. Recent work on general graph learning [59] and unsupervised OD model selection on tabular data [93] shows that under meta-learning frameworks, we can identify good OD models to use for a new task (or dataset) based on its similarity to meta-tasks where ground truth information is available. A similar approach also works for unsupervised hyperparameter tuning [88]. We suggest exploring a meta-learning framework for algorithm selection and hyperparameter tuning in solving OND, including quantifying task similarity between graph OD datasets. Such a framework would require some but not all datasets to have ground truth, and that datasets with ground truth can be related to the ones without.

5.2 Future Directions in Improving Our Benchmark System BOND

Extending detection tasks to different "levels". In BOND, we focus on node-level detection with static attributed graphs due to their popularity, while there are more detection tasks at different levels of a graph. Recent graph OD algorithms extend to edge- [83], subgraph- [70], and graph-level [62, 85] detection. Future comprehensive graph OD benchmarks can include these emerging graph OD tasks.

Incorporating supervision. Although BOND focuses on unsupervised methods, there can be cases where a small set of labels (either for OD or relevant tasks) are available so that (semi-)supervised learning is possible (e.g., [22, 84]). Extending BOND to handle supervision would particularly be beneficial in addressing algorithm selection and hyperparameter tuning challenges.

Curating more datasets. Thus far, we have only included nine real datasets in BOND. Adding more datasets over time would be beneficial, especially ones with organic outliers. With a much larger collection of datasets (e.g., ≥ 20), one could run statistical tests for comparison [14], which has been used in OD tasks with tabular datasets [51, 93]. Similar to tabular OD [23], one can convert existing multi-classification graph datasets (e.g., ones from Open Graph Benchmark (OGB) [30]) into OD datasets by treating one or combining several small classes to be treated as a single "outlier" class, with all other classes considered "normal".

Acknowledgments

Funding. This work was supported in part by NSF under grants III-1763325, III-1909323, III-2106758, and SaTC-1930941. K.S. is supported by a Cisco Research Award. G.H.C. is supported by NSF CAREER award #2047981.

Author contributions. Conceptualization: K.L., Y.D., and Y.Z. Investigation and experiments: K.L., Y.D., Y.Z., X.D., X.H., R.Z., K.D., and C.C. Writing – original draft: K.L., Y.D., and Y.Z. Writing – review & editing: H.P., K.S., L.S., J.L., Z.H., and P.S.Y. Extensive reviewing & editing: G.H.C.

For any correspondence, please refer to Hao Peng.

References

[1] C. C. Aggarwal. An introduction to outlier analysis. In *Outlier analysis*. Springer, 2017.

[2] L. Akoglu, H. Tong, and D. Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015.

[3] S. Bandyopadhyay, N. Lokesh, and M. N. Murty. Outlier aware network embedding for attributed networks. In *Proceedings of the AAAI conference*, volume 33, pages 12–19, 2019.

[4] S. Bandyopadhyay, S. V. Vivek, and M. Murty. Outlier resistant unsupervised deep architectures for attributed network embedding. In *Proceedings of the WSDM*, pages 25–33, 2020.

[5] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[6] G. O. Campos, A. Zimek, J. Sander, R. J. Campello, B. Micenková, E. Schubert, I. Assent, and M. E. Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data mining and knowledge discovery*, 30(4):891–927, 2016.

[7] D. Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *European conference on principles of data mining and knowledge discovery*, pages 112–124. Springer, 2004.

[8] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445, 2020.

[9] J. Chen, S. Sathe, C. Aggarwal, and D. Turaga. Outlier detection with autoencoder ensembles. In *Proceedings of the 2017 SIAM international conference on data mining*, pages 90–98. SIAM, 2017.

[10] Z. Chen, L. Chen, S. Villar, and J. Bruna. Can graph neural networks count substructures? *Advances in neural information processing systems*, 33:10383–10395, 2020.

[11] Z. Chen, B. Liu, M. Wang, P. Dai, J. Lv, and L. Bo. Generative adversarial attributed network anomaly detection. In *Proceedings of the ACM CIKM*, pages 1989–1992, 2020.

[12] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. Nus-wide: a real-world web image database from national university of singapore. In *Proceedings of the ACM international conference on image and video retrieval*, pages 1–9, 2009.

[13] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10):1367–1372, 2004.

[14] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[15] K. Ding, J. Li, N. Agarwal, and H. Liu. Inductive anomaly detection on attributed networks. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 1288–1294, 2021.

[16] K. Ding, J. Li, R. Bhanushali, and H. Liu. Deep anomaly detection on attributed networks. In *Proceedings of the SDM*, pages 594–602. SIAM, 2019.

[17] K. Ding, J. Wang, J. Caverlee, and H. Liu. Meta propagation networks for graph few-shot semi-supervised learning. In *Proceedings of the AAAI Conference*. AAAI, 2022.

[18] K. Ding, Z. Xu, H. Tong, and H. Liu. Data augmentation for deep graph learning: A survey. *arXiv preprint arXiv:2202.08235*, 2022.

[19] K. Ding, Q. Zhou, H. Tong, and H. Liu. Few-shot network anomaly detection via cross-network meta-learning. In *Proceedings of the Web Conference 2021*, pages 2448–2456, 2021.

[20] M. Ding, K. Kong, J. Li, C. Zhu, J. Dickerson, F. Huang, and T. Goldstein. Vq-gnn: A universal framework to scale up graph neural networks using vector quantization. *Advances in Neural Information Processing Systems*, 34, 2021.

[21] X. Ding, L. Zhao, and L. Akoglu. Hyperparameter sensitivity in deep outlier detection: Analysis and a scalable hyper-ensemble solution. *arXiv preprint arXiv:2206.07647*, 2022.

[22] Y. Dou, Z. Liu, L. Sun, Y. Deng, H. Peng, and P. S. Yu. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the ACM CIKM*, pages 315–324, 2020.

[23] A. Emmott, S. Das, T. Dietterich, A. Fern, and W.-K. Wong. A meta-analysis of the anomaly detection problem. *arXiv preprint arXiv:1503.01158*, 2015.

[24] G. Fagiolo. Clustering in complex directed networks. *Physical Review E*, 76(2):026107, 2007.

[25] H. Fan, F. Zhang, and Z. Li. Anomalydae: Dual autoencoder for anomaly detection on attributed networks. In *Proceedings of the IEEE ICASSP*, pages 5685–5689, 2020.

[26] S. Freitas, Y. Dong, J. Neil, and D. H. Chau. A large-scale database for graph representation learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[27] A. Gaddam, T. Wilkin, M. Angelova, and J. Gaddam. Detecting sensor faults, anomalies and outliers in the internet of things: A survey on the challenges and solutions. *Electronics*, 9(3):511, 2020.

[28] S. Han, X. Hu, H. Huang, M. Jiang, and Y. Zhao. Adbench: Anomaly detection benchmark. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[29] B. Hooi, H. A. Song, A. Beutel, N. Shah, K. Shin, and C. Faloutsos. Fraudar: Bounding graph fraud in the face of camouflage. In *KDD*, 2016.

[30] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22118–22133. Curran Associates, Inc., 2020.

[31] K. Huang, T. Fu, W. Gao, Y. Zhao, Y. H. Roohani, J. Leskovec, C. W. Coley, C. Xiao, J. Sun, and M. Zitnik. Therapeutics data commons: Machine learning datasets and tasks for drug discovery and development. In *Proceedings of the Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[32] X. Huang, Y. Yang, Y. Wang, C. Wang, Z. Zhang, J. Xu, and L. Chen. Dgraph: A large-scale financial dataset for graph anomaly detection. *arXiv preprint arXiv:2207.03579*, 2022.

[33] V. N. Ioannidis, D. Berberidis, and G. B. Giannakis. Unveiling anomalous nodes via random sampling and consensus on graphs. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5499–5503. IEEE, 2021.

[34] C. I. Jerez, J. Zhang, and M. R. Silva. On equivalence of anomaly detection algorithms. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2022.

[35] Z. Jia, S. Lin, M. Gao, M. Zaharia, and A. Aiken. Improving the accuracy, scalability, and performance of graph neural networks with roc. *Proceedings of the MLSys*, 2:187–198, 2020.

[36] D. Kim and A. Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. *arXiv preprint arXiv:2204.04879*, 2022.

[37] S. Kim, Y.-C. Tsai, K. Singh, Y. Choi, E. Ibok, C.-T. Li, and M. Cha. Date: Dual attentive tree-aware embedding for customs fraud detection. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2880–2890, 2020.

[38] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[39] T. N. Kipf and M. Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[40] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the ICLR*, 2017.

[41] B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. In *European conference on machine learning*, pages 217–226. Springer, 2004.

[42] D. Koutra, T.-Y. Ke, U. Kang, D. H. P. Chau, H.-K. K. Pao, and C. Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 245–260. Springer, 2011.

[43] S. Kumar, B. Hooi, D. Makhija, M. Kumar, C. Faloutsos, and V. Subrahmanian. Rev2: Fraudulent user prediction in rating platforms. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 333–341, 2018.

[44] S. Kumar, X. Zhang, and J. Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1269–1278, 2019.

[45] K.-H. Lai, D. Zha, G. Wang, J. Xu, Y. Zhao, D. Kumar, Y. Chen, P. Zumkhawaka, M. Wan, D. Martinez, and X. Hu. TODS: An automated time series outlier detection system. In *Proceedings of the AAAI Conference*, pages 16060–16062, 2021.

[46] K.-H. Lai, D. Zha, J. Xu, Y. Zhao, G. Wang, and X. Hu. Revisiting time series outlier detection: Definitions and benchmarks. In *Advances in neural information processing systems*, 2021.

[47] K.-H. Lai, D. Zha, K. Zhou, and X. Hu. Policy-gnn: Aggregation optimization for graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 461–471, 2020.

[48] J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1):5–es, 2007.

[49] G. Li, M. Müller, B. Ghanem, and V. Koltun. Training graph neural networks with 1000 layers. In *International conference on machine learning*, pages 6437–6449. PMLR, 2021.

[50] J. Li, H. Dani, X. Hu, and H. Liu. Radar: Residual analysis for anomaly detection in attributed networks. In *IJCAI*, pages 2152–2158, 2017.

[51] Z. Li, Y. Zhao, X. Hu, N. Botta, C. Ionescu, and G. Chen. Ecod: Unsupervised outlier detection using empirical cumulative distribution functions. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2022.

[52] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 6(1):1–39, 2012.

[53] K. Liu, Y. Dou, Y. Zhao, X. Ding, X. Hu, R. Zhang, K. Ding, C. Chen, H. Peng, K. Shu, et al. PyGOD: A python library for graph outlier detection. *arXiv preprint arXiv:2204.12095*, 2022.

[54] Y. Liu, Z. Li, S. Pan, C. Gong, C. Zhou, and G. Karypis. Anomaly detection on attributed networks via contrastive self-supervised learning. *IEEE transactions on neural networks and learning systems*, 2021.

[55] X. Ma, J. Wu, S. Xue, J. Yang, C. Zhou, Q. Z. Sheng, H. Xiong, and L. Akoglu. A comprehensive survey on graph anomaly detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[56] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pages 43–52, 2015.

[57] E. Müller, P. I. Sánchez, Y. Mülle, and K. Böhm. Ranking outlier nodes in subspaces of attributed graphs. In *2013 IEEE 29th international conference on data engineering workshops (ICDEW)*, pages 216–222. IEEE, 2013.

[58] D. Pacheco, P.-M. Hui, C. Torres-Lugo, B. T. Truong, A. Flammini, and F. Menczer. Uncovering coordinated networks on social media: Methods and case studies. *ICWSM*, 21:455–466, 2021.

[59] N. Park, R. Rossi, N. Ahmed, and C. Faloutsos. Autogml: Fast automatic model selection for graph machine learning. *arXiv preprint arXiv:2206.09280*, 2022.

[60] Z. Peng, M. Luo, J. Li, H. Liu, and Q. Zheng. Anomalous: A joint modeling approach for anomaly detection on attributed networks. In *IJCAI*, pages 3513–3519, 2018.

[61] J. W. Pennebaker, M. E. Francis, and R. J. Booth. Linguistic inquiry and word count: Liwc 2001. *Mahway: Lawrence Erlbaum Associates*, 71(2001):2001, 2001.

[62] C. Qiu, M. Kloft, S. Mandt, and M. Rudolph. Raising the bar in graph-level anomaly detection, 2022.

[63] S. Rayana and L. Akoglu. Collective opinion spam detection: Bridging review networks and metadata. In *KDD*, 2015.

[64] M. Sakurada and T. Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA workshop on machine learning for sensory data analysis*, pages 4–11, 2014.

[65] P. I. Sánchez, E. Müller, F. Laforet, F. Keller, and K. Böhm. Statistical selection of congruent subspaces for mining attributed graphs. In *2013 IEEE 13th international conference on data mining*, pages 647–656. IEEE, 2013.

[66] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[67] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[68] G. Steinbuss and K. Böhm. Benchmarking unsupervised outlier detection with realistic synthetic data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(4):1–20, 2021.

[69] H. Tong and C.-Y. Lin. Non-negative residual matrix factorization with application to graph anomaly detection. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, pages 143–153. SIAM, 2011.

[70] H. Wang, C. Zhou, J. Wu, W. Dang, X. Zhu, and J. Wang. Deep structure learning for fraud detection. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 567–576. IEEE, 2018.

[71] J. Wang, K. Ding, L. Hong, H. Liu, and J. Caverlee. Next-item recommendation with sequential hypergraphs. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, 2020.

[72] X. Wang, B. Jin, Y. Du, P. Cui, Y. Tan, and Y. Yang. One-class graph neural networks for anomaly detection in attributed networks. *Neural computing and applications*, 33(18):12073–12085, 2021.

[73] Y. Wang, J. Zhang, S. Guo, H. Yin, C. Li, and H. Chen. Decoupling representation learning and classification for gnn-based anomaly detection. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1239–1248, 2021.

[74] M. Weber, G. Domeniconi, J. Chen, D. K. I. Weidele, C. Bellei, T. Robinson, and C. E. Leiserson. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591*, 2019.

[75] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 824–833, 2007.

[76] Z. Xu, X. Huang, Y. Zhao, Y. Dong, and J. Li. Contrastive attributed network anomaly detection with data augmentation. In *Proceedings of the PAKDD*, 2022.

[77] J. Ye and L. Akoglu. Discovering opinion spammer groups by network footprints. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 267–282. Springer, 2015.

[78] M. Yoon, B. Hooi, K. Shin, and C. Faloutsos. Fast and accurate anomaly detection in dynamic graphs with a two-pronged approach. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 647–657, 2019.

[79] X. Yu, Z. Liu, Y. Fang, and X. Zhang. Count-gnn: Graph neural networks for subgraph isomorphism counting. *under review*, 2021.

[80] X. Yuan, N. Zhou, S. Yu, H. Huang, Z. Chen, and F. Xia. Higher-order structure based anomaly detection on attributed networks. In *Proceedings of the IEEE Big Data Conference*, pages 2691–2700, 2021.

[81] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

[82] D. Zha, K.-H. Lai, K. Zhou, and X. Hu. Towards similarity-aware time-series classification. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pages 199–207. SIAM, 2022.

[83] G. Zhang, Z. Li, J. Huang, J. Wu, C. Zhou, J. Yang, and J. Gao. efraudcom: An e-commerce fraud detection system via competitive graph neural networks. *ACM Transactions on Information Systems (TOIS)*, 40(3):1–29, 2022.

[84] G. Zhang, J. Wu, J. Yang, A. Beheshti, S. Xue, C. Zhou, and Q. Z. Sheng. Fraudre: Fraud detection dual-resistant to graph inconsistency and imbalance. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 867–876. IEEE, 2021.

[85] L. Zhao and L. Akoglu. On using classification datasets to evaluate graph outlier detection: Peculiar observations and new insights. *Big Data*, 2021.

[86] T. Zhao, C. Deng, K. Yu, T. Jiang, D. Wang, and M. Jiang. Error-bounded graph anomaly loss for gnns. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1873–1882, 2020.

[87] T. Zhao, T. Jiang, N. Shah, and M. Jiang. A synergistic approach for graph anomaly detection with pattern mining and feature learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[88] Y. Zhao and L. Akoglu. Towards unsupervised hpo for outlier detection. *arXiv preprint arXiv:2208.11727*, 2022.

[89] Y. Zhao, G. H. Chen, and Z. Jia. TOD: Tensor-based outlier detection. *arXiv preprint arXiv:2110.14007*, 2021.

[90] Y. Zhao, X. Hu, C. Cheng, C. Wang, C. Wan, W. Wang, J. Yang, H. Bai, Z. Li, C. Xiao, Y. Wang, Z. Qiao, J. Sun, and L. Akoglu. SUOD: accelerating large-scale unsupervised heterogeneous outlier detection. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems 2021, MLSys 2021, virtual, April 5-9, 2021*. mlsys.org, 2021.

[91] Y. Zhao, Z. Nasrullah, M. K. Hryniewicki, and Z. Li. Lscp: Locally selective combination in parallel outlier ensembles. In *SDM*, pages 585–593. SIAM, 2019.

[92] Y. Zhao, Z. Nasrullah, and Z. Li. PyOD: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.

[93] Y. Zhao, R. Rossi, and L. Akoglu. Automatic unsupervised outlier model selection. *Proceedings of the NeurIPS*, 34:4489–4502, 2021.

[94] L. Zheng, Z. Li, J. Li, Z. Li, and J. Gao. Addgraph: Anomaly detection in dynamic graph using attention-based temporal gcn. In *IJCAI*, pages 4419–4425, 2019.

[95] Q. Zheng, X. Zou, Y. Dong, Y. Cen, D. Yin, J. Xu, Y. Yang, and J. Tang. Graph robustness benchmark: Benchmarking the adversarial robustness of graph machine learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[96] C. Zhou and R. C. Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 665–674, 2017.

[97] Y. Zhu, Y. Xu, Q. Liu, and S. Wu. An empirical study of graph contrastive learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

A Additional Details on BOND

A.1 Additional Dataset Information

A.1.1 Real Data

Cora [66] is a citation graph with nodes representing machine learning papers and edges representing papers' citation relationships. The node features are sparse bag-of-words (BoW) vectors extracted from the paper document, and their labels represent one of the seven classes.

Amazon [67] is a segment of the Amazon co-purchase graph [56], where nodes represent goods and edges indicate that two goods are frequently bought together. Notably, node features are BoW-encoded product reviews, and class labels are given by the product category.

Flickr [81] datasets originates from NUS-wide[12], a real-world web image database from the National University of Singapore. The SNAP website[1] collected Flickr data from four different sources including NUS-wide, and generated an undirected graph. A node in the graph represents one image uploaded to Flickr. If two images share some common properties (e.g., same geographic location, same gallery, comments by the same user, etc.), one edge is made between these two nodes. The node feature is composed of a 500-dimensional vector of the images provided by NUS-wide. Eighty-one tags of each image are manually merged into seven classes, and each image is assigned to one of the seven classes.

Weibo [86] is a user-posts-hashtag graph from Tencent-Weibo, a Twitter-like platform in China. This dataset collects information from 8,405 users with 61,964 hashtags. We use the user-user graph [2] provided by the author, which connects users who used the same hashtag. Temporal information was used to label the users. If a user made at least five suspicious events, he/she is labeled as a suspicious user; if no suspicious event was made, he/she is a benign user. There are a total of 868 suspicious users and 7,537 benign users. The suspicious users are regarded as outliers in the graph. Since the ground truth was generated using time information, the timestamps are not used to create raw user features. Therefore, the raw feature vector has two parts: (1) for each user, the one-hot vectors of his/her posts are summed where each one-hot vector represents the location where a post was made. Then the #dimension of the summed vector is reduced to 100 using SVD and (2) for each user, the #dimension of the BoW vectors extracted from post texts is reduced to 300. The final node feature is the concatenation of the location vector and the BoW vector. Note that Weibo is a directed graph; the remaining datasets used in our benchmark are undirected graphs.

Reddit [44, 73] is a user-subreddit graph extracted from a social media platform, Reddit [3]. This public dataset consists of one month of user posts on subreddits [4]. The 1,000 most active subreddits and the 10,000 most active users are extracted as subreddit nodes and user nodes, respectively. This results in 168,016 interactions. Each user has a binary label indicating whether it has been banned by the platform. We assume that the banned users are outliers compared to normal Reddit users. The text of each post is converted into a feature vector representing their LIWC categories [61] and the features of users and subreddits are the feature summation of the posts they have, respectively.

Disney [57] and **Books** [65] come from the Amazon co-purchase networks [48]. Disney is a co-purchase network of movies, where the attributes include prices, ratings, number of reviews, etc. The ground truth labels (i.e., whether it is an outlier) are manually labeled by high school students by majority vote. The second dataset, Books, is a co-purchase network of books on Amazon, which has similar attributes to the Disney dataset. The ground truth labels are derived from `amazonfail` tag information. More information about the datasets can be found on the project website [5].

Enron [65] is an email network dataset extracted from [41]. Each email is regarded as a node, and the messages between email addresses represent edges. The email addresses having sent spam messages are taken as outliers. Each node contains 20 attributes describing aggregated information about the average content length, the average number of recipients, or the time range between two emails.

[1] `http://snap.stanford.edu/`

[2] `https://github.com/zhao-tong/Graph-Anomaly-Loss/tree/master/data/weibo_s`

[3] `https://www.reddit.com/`

[4] `http://files.pushshift.io/reddit/`

[5] `https://www.ipd.kit.edu/~muellere/consub/`

DGraph [32] is a large-scale attributed graph with 3M nodes, 4M dynamic edges, and 1M ground-truth nodes. The nodes represent user accounts in a financial company providing personal loan services, and the edge between two nodes represents one account that has added another account as an emergency contact. For all the accounts with at least one borrowing record, the outliers are the accounts with overdue history, and the inliers are the accounts without overdue. Note there are also 2M accounts/nodes without any borrowing at all. The 17 node features are encoded from the user profile information like age and gender.

A.1.2 Random Graph Generation Method

We leverage a random graph generation method used in [36] to create an arbitrary OND graph for benchmarking. Specifically, the implementation in PyG [1] is used with 2 classes, node_homophily_ratio=0.5, average_degree=5 and num_channels=64. We use the generated random graphs to benchmark algorithms' efficiency and scalability. We generate a random graph **Gen_Time** with num_nodes_per_class=500 (1000 in total) as the graph data to test the runtime. To benchmark the scalability, we generate multiple random graphs **Gen_100**, **Gen_500**, **Gen_1000**, **Gen_5000** and **Gen_10000** with num_nodes_per_class equal to 50, 250, 500, 2500 and 5000, respectively.

A.1.3 Outlier Injection Details

For injecting structural outliers, we use $p = 0.2$. For contextual outliers, we set q equal to m in structural outlier injection. The other parameters used in outlier injection are shown in Table 6. Note that m is set to be approximate twice the degree of the graph. For real datasets, we keep a similar outlier ratio (i.e., the number of outliers injected is approximately 5% of the total number of nodes; as a reminder, for structural outliers, we inject $m \times n$ outliers and for contextual outliers, we inject o outliers; $o = m \times n$ in our setting). We keep a similar number of outliers for generated graph datasets of various sizes. The statistics of the generated graphs are shown in Table 7.

Table 6: Parameters used in synthetic outliers injection.

	Cora	Amazon	Flickr	Gen_Time	Gen_100	Gen_500	Gen_1000	Gen_5000	Gen_10000
Degree	4.1	37.5	10.6	5	5	5	5	5	5
n	70	350	2240	10	1	1	1	1	1
m	10	70	20	10	10	10	10	10	10

Table 7: Statistics of generated datasets in BOND.

Dataset	#Nodes	#Edges	#Feat.	Degree	#Con.	#Strct.	#Outliers	Ratio
Gen_Time	1,000	5,746	64	5.7	100	100	189	18.9%
Gen_100	100	618	64	6.2	10	10	18	18.0%
Gen_500	500	2,662	64	5.3	10	10	20	4.0%
Gen_1000	1,000	4,936	64	4.9	10	10	20	2.0%
Gen_5000	5,000	24,938	64	5.0	10	10	20	0.4%
Gen_10000	10,000	49,614	64	5.0	10	10	20	0.2%

A.2 Description of algorithms in the benchmark

LOF [5]. LOF is short for the Local Outlier Factor. LOF computes the degree of an object as abnormality, and the degree depends on how isolated the object is with respect to its surrounding neighborhood. Note that LOF only uses node attribute information, and the neighborhood is composed of k-nearest-neighbors.

IF [52]. Isolation Forest (IF) is a classic tree ensemble method used in outlier detection. It builds an ensemble of base trees to isolate the data points and defines the decision boundary as the closeness of an individual instance to the root of the tree. It only uses node attributes of data.

MLPAE [64]. The MLPAE is a vanilla autoencoder with multiple layer perceptron (MLP) as encoder and decoder. The encoder takes the node attribute as the input to learn its low-dimensional embedding

[1] `https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html#torch_geometric.datasets.RandomPartitionGraphDataset`

and a decoder reconstructs the input node attribute from the node embedding. The outlier score of a node is the reconstruction error of the decoder.

SCAN [75]. SCAN is a structural clustering algorithm to detect clusters, hub nodes, and outlier nodes in a graph. Since the structural outliers exhibit clustering patterns on graphs, we use SCAN to detect clusters and the nodes in detected clusters are regarded as structural outliers in the graph. SCAN only takes the graph structure as the input.

Radar [50]. Radar is an anomaly detection framework for attributed graphs. It takes the graph structure and node attributes as the input. It detects outlier nodes whose behaviors are singularly different from the majority by characterizing the residuals of attribute information and its coherence with network information. The outlier score of a node is decided by the norm of its reconstruction residual.

ANOMALOUS [60]. ANOMALOUS performs joint anomaly detection and attribute selection to detect node anomalies on attributed graphs based on the CUR decomposition and residual analysis. It takes the graph structure and node attribute as the input, and the outlier score of a node is decided by the norm of its reconstruction residual.

GCNAE [39]. GCNAE is the autoencoder framework with GCNs [40] as the encoder and decoder. It takes the graph structure and node attributes as input. The encoder is used to learn a node's embedding by aggregating its neighbor information. The decoder reconstructs the node attribute by applying another GCN to node embeddings and graph structures. Similar to MLPAE, the outlier score of a node is the reconstruction error of the decoder.

DOMINANT [16]. DOMINANT is one of the first works that leverage GCN and AE for outlier node detection. It uses a two-layer GCN as the encoder, a two-layer GCN decoder to reconstruct the node attribute, and a one-layer GCN and dot product as the structural decoder to reconstruct the graph adjacency matrix. The reconstruction errors of both decoders are combined as the outlier scores of the nodes.

DONE [4]. DONE leverages a structural and an attribute AE to reconstruct the adjacency matrix and node attribute. The encoders and decoders are composed of MLPs. The node embeddings and outlier scores are optimized simultaneously with a unified loss function.

AdONE [4]. AdONE is a variant of DONE, which uses an extra discriminator to discriminate the learned structure embedding and attribute embedding of a node. The adversarial training approach supposes to better align the two different embeddings in the latent space.

AnomalyDAE [25]. AnomalyDAE also utilizes a structure AE and attribute AE to detect outlier nodes. The structure encoder of AnomalyDAE takes both the adjacency matrix and node attribute as input; the attribute decoder reconstructs the node attribute using both structure and attribute embeddings.

GAAN [11]. GAAN is a GAN-based outlier node detection method. It employs an MLP-based generator to generate fake graphs and an MLP-based encoder to encode graph information. A discriminator is trained to recognize whether two connected nodes are from the real or fake graph. The outlier score is obtained by the node reconstruction error and real-node identification confidence.

GUIDE [80]. GUIDE is similar to DONE and AdONE with two different AEs, but it pre-processes the structure information before feeding it into the structure encoder. Specifically, node motif degree is used to represent the node structure vector which could encode higher-order structure information.

CONAD [76]. CONAD is one of the BOND methods that leverage graph augmentation and contrastive learning techniques. It imposes prior knowledge of outlier nodes via generating augmented graphs. After encoding the graphs using Siamese GNN encoders, the contrastive loss is used to optimize the encoder, and the outlier score of the node is obtained by two different decoders like DOMINANT.

A.3 Description of Evaluation Metrics

ROC-AUC (AUC). AUC computes the Area Under the Receiver Operating Characteristic Curve (ROC-AUC) from predicted outlier scores. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. In the BOND benchmark,

we regard the outlier nodes as the positive class and compute the AUC for it. AUC equals 1 means the model makes a perfect prediction, and AUC equals 0.5 means the model has no class-separation capability. AUC is better than accuracy when evaluating the outlier detection task since it is not sensitive to the imbalanced class distribution of the data.

Average Precision (AP). AP summarizes the precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight. AP is a metric that balances the effects of recall and precision, and a higher AP indicates a lower false-positive rate (FPR) and false-negative rate (FNR). FPR and FNR have equal importance for most outlier detection applications as more misclassified normal samples could worsen legit users' experience.

Recall@k. The outliers are usually rare in contrast to enormous normal samples in the data, and the outliers are of the most interest to outlier detection practitioners. We propose to use Recall@k to measure how well the detectors rank outliers over the normal samples. We set k as the number of ground truth outliers in each dataset. The Recall@k is computed by the number of true outliers among the top-k samples in the outlier ranking list divided by k. A higher Recall@k score indicates a better detection performance, and Recall@k equals 1 means the model perfectly ranks all outliers over the normal samples.

Runtime. Due to the coverage of both classical algorithms and neural network methods, we consistently measure the model runtime as the duration between the experiment starts and ends, mimicking the real-world applications without specific differentiation between CPU and GPU time.

GPU Memory. Notably, GPU memory is often the bottleneck of machine learning algorithms due to its limitation in extension. In BOND, we report the max active GPU memory for running an algorithm.

B Additional Experimental Settings and Details

Environment. The key libraries and their versions used in the experiment are as follows: Python=3.7, CUDA_version=11.1, torch=1.10, pytorch_geometric>=2.0.3, networkx=2.6.3, numpy=1.19.4, scipy=1.5.2, scikit-learn=0.22.1, pyod=1.0.1, pygod=0.3.0.

Hardware configuration. All the experiments were performed on a Linux server with a 3.50GHz Intel Core i5 CPU, 64GB RAM, and 1 NVIDIA GTX 1080 Ti GPU with 12GB memory.

More model implementation details. To include a large number of algorithms, we build Python Graph Outlier Detection (PyGOD)[1] [53], which provides more than 10 latest graph OD algorithms; all with unified APIs and optimizations. We tried our best to apply the same set of optimization techniques to each dataset. For Radar and ANOMALOUS, we use gradient descent instead closed-form optimization provided in official implementation due to fairness and efficiency concerns. For all deep algorithms, we implement sampling and minibatch training on large graphs (e.g., Flickr). See our library source code for more details. Meanwhile, we also include multiple non-graph baselines (LOF and IF) from our early work Python Outlier Detection (PyOD) [92].

Hyperparameter grid

The hyperparameter space is shown in Table 8. The candidates of hyperparameters are listed in square brackets. In each trial, a value is randomly chosen among candidates. The results (mean, std, max) are reported among 20 trials.

Due to the large graph size, full batch training on Flickr cannot fit in single GPU memory. Minibatch training and different batch size, sampling size, and the number of epochs are used on Flickr. Because of the complexity of real datasets, automated balancing by the standard deviation for weight alpha cannot balance well. Thus, three candidates are attempted. As Reddit has a lower feature dimension, we reduce hidden dimension values on Reddit.

How we determine the optimal performance in runtime comparison.

The optimal performance is determined by the ROC-AUC score. Taking the computational cost into account, we expect a reasonable score within as few training epochs as possible. Thus, when the

[1] PyGOD: `https://pygod.org/`

Table 8: Hyperparameters in different algorithms. The values in "[]" are candidates. We present these common hyperparameters shared by multiple algorithms on the top, and also specify some algorithm-specific hyperparameters at the bottom. Refer to PyGOD doc for more details.

Algorithm	Hyperparameter	Cora	Amazon	Flickr	Weibo	Disney	Books	Enron	DGraph	Reddit	Gen
Common	*dropout*	[0, 0.1, 0.3]									
	learning rate	[0.1, 0.05, 0.01]									
	weight decay	0.01									
	batch size	full batch		64	full batch				64	full batch	
	sampling	all neigh.		3	all neigh.				3	all neigh.	
	epoch	300		2	300				2	300	
	alpha	auto			[0.8, 0.5, 0.2]						auto
	hid. dim.	[32, 64, 128, 256]				[8, 12, 16]				[32, 48, 64]	
SCAN	*eps*	[0.3, 0.5, 0.8]									
	mu	[2, 5, 10]									
AnomalyDAE	*theta*	[10, 40, 90]									
	eta	[3, 5, 8]									
GAAN	*noise dim.*	[8, 16, 32]									
GUIDE	*struct. hid.*	[4, 5, 6]									

score converges (i.e., the score increment of consequence epochs is less than 0.5%), we mark the current epoch as optimal.

C Additional Experimental Results

C.1 Additional Results on Real Dataset Detection Performance

Table 9: Average Precision (%) comparison among OD algorithms on three datasets with synthetic outliers, where we show *the avg perf.* ± *the STD of perf.* (*max perf.*) of each. The best algorithm by expectation is shown in **bold**, while the max performance per dataset is marked with underline. OOM denotes out of memory with regard to GPU (_G) and CPU (_C).

Algorithm	Cora	Amazon	Flickr
LOF	12.2±0.0 (12.2)	5.6±0.0 (5.6)	5.4±0.0 (5.4)
IF	10.1±0.7 (11.5)	6.2±1.3 (9.5)	8.2±0.6 (9.3)
MLPAE	13.2±0.0 (13.2)	**34.8**±0.0 (34.9)	18.7±0.0 (18.7)
SCAN	12.0±5.8 (21.6)	13.8±10.5 (33.8)	24.6±20.6 (50.4)
Radar	7.5±0.4 (7.9)	12.3±1.5 (14.2)	OOM_G
ANOMALOUS	5.9±1.7 (8.8)	11.7±1.3 (15.3)	OOM_G
GCNAE	13.2±0.0 (13.2)	**34.8**±0.0 (34.9)	18.0±2.8 (18.6)
DOMINANT	20.0±3.0 (20.8)	16.0±0.9 (16.6)	**28.6**±11.8 (36.4)
DONE	**25.0**±8.8 (42.4)	19.3±7.7 (36.5)	20.0±2.7 (24.9)
AdONE	19.3±4.2 (29.2)	23.7±4.7 (31.2)	18.2±3.5 (25.1)
AnomalyDAE	18.3±2.1 (21.3)	24.0±7.2 (33.4)	12.3±3.8 (18.2)
GAAN	14.6±0.4 (15.1)	34.5±0.3 (34.8)	18.6±0.1 (18.7)
GUIDE	14.0±0.5 (14.8)	OOM_C	OOM_C
CONAD	17.1±5.5 (20.7)	15.5±2.0 (16.6)	8.8±1.1 (10.0)

Table 10: Average Precision (%) comparison among OD algorithms on six datasets with organic outliers, where we show *the avg perf.* ± *the STD of perf.* (*max perf.*) of each. The best algorithm by expectation is shown in **bold**, while the max performance per dataset is marked with underline. OOM denotes out of memory with regard to GPU (_G) and CPU (_C). TLE denotes time limit of 24 hours exceeded.

Algorithm	Weibo	Reddit	Disney	Books	Enron	DGraph
LOF	15.8±0.0 (15.8)	**4.2**±0.0 (4.2)	5.2±0.0 (5.2)	1.5±0.0 (1.5)	0.0±0.0 (0.0)	TLE
IF	12.9±2.6 (19.8)	2.8±0.1 (2.9)	**10.1**±4.5 (22.6)	1.9±0.2 (2.7)	0.1±0.0 (0.1)	**1.8**±0.0 (1.9)
MLPAE	52.8±9.9 (64.5)	3.4±0.0 (3.4)	5.9±0.8 (7.9)	1.8±0.3 (2.5)	0.1±0.0 (0.1)	0.9±0.0 (1.0)
SCAN	17.3±3.4 (20.5)	3.3±0.0 (3.3)	5.0±0.3 (5.5)	2.0±0.1 (2.1)	0.0±0.0 (0.1)	TLE
Radar	**92.1**±0.7 (92.9)	3.6±0.2 (3.9)	7.2±0.0 (7.2)	2.2±0.0 (2.2)	**0.2**±0.0 (0.2)	OOM_C
ANOMALOUS	**92.1**±0.7 (92.9)	4.0±0.6 (5.1)	7.2±0.0 (7.2)	2.2±0.0 (2.2)	**0.2**±0.0 (0.2)	OOM_C
GCNAE	70.8±5.0 (80.9)	3.4±0.0 (3.4)	4.8±0.7 (5.8)	2.1±0.4 (3.5)	0.1±0.0 (0.1)	1.0±0.0 (1.0)
DOMINANT	18.0±10.2 (36.2)	3.7±0.0 (3.8)	7.6±5.0 (23.2)	2.2±0.6 (4.1)	0.1±0.1 (0.4)	OOM_C
DONE	65.5±13.4 (77.3)	3.7±0.4 (4.5)	5.0±0.7 (6.4)	1.8±0.3 (2.6)	0.1±0.0 (0.1)	OOM_C
AdONE	62.9±9.5 (74.4)	3.3±0.4 (4.0)	6.1±1.5 (11.7)	2.5±0.3 (3.2)	0.1±0.0 (0.1)	OOM_C
AnomalyDAE	38.5±22.5 (77.3)	3.7±0.1 (3.8)	5.7±0.2 (6.3)	**3.5**±1.4 (7.8)	0.1±0.0 (0.1)	OOM_C
GAAN	80.3±0.2 (80.7)	3.7±0.1 (3.9)	5.6±0.0 (5.6)	2.6±0.8 (5.6)	0.1±0.0 (0.1)	OOM_C
GUIDE	OOM_C	OOM_C	4.8±0.9 (6.9)	1.9±0.3 (3.1)	OOM_C	OOM_C
CONAD	15.6±6.9 (31.7)	3.7±0.3 (4.6)	6.0±1.4 (11.5)	2.5±0.8 (4.9)	0.1±0.0 (0.3)	0.9±0.0 (0.9)

Table 11: Recall@k (%) comparison among OD algorithms on three datasets with synthetic outliers, where we show *the avg perf.* $\pm$ *the STD of perf.* (*max perf.*) of each. The best algorithm by expectation is shown in **bold**, while the max performance per dataset is marked with underline. k is set as the number of outliers in labels. OOM denotes out of memory with regard to GPU (_G) and CPU (_C).

Algorithm	Cora	Amazon	Flickr
LOF	15.2±0.0 (15.2)	5.2±0.0 (5.2)	8.4±0.0 (8.4)
IF	14.6±1.7 (18.1)	5.0±2.0 (9.9)	12.6±1.2 (14.7)
MLPAE	18.8±0.0 (18.8)	45.0±0.0 (45.1)	28.0±0.1 (28.1)
SCAN	17.5±8.9 (32.6)	16.1±11.2 (32.7)	27.3±22.5 (51.9)
Radar	4.2±0.5 (5.1)	18.0±5.2 (23.8)	OOM_G
ANOMALOUS	3.4±2.3 (10.1)	13.6±4.6 (25.1)	OOM_G
GCNAE	18.8±0.0 (18.8)	45.0±0.1 (45.1)	26.8±5.0 (28.1)
DOMINANT	23.8±4.0 (26.1)	19.4±0.9 (20.2)	**38.9**±17.2 (48.4)
DONE	**27.9**±9.8 (44.9)	20.6±9.4 (39.6)	23.6±2.3 (26.6)
AdONE	22.0±5.9 (34.8)	26.9±5.9 (38.2)	19.3±4.4 (27.0)
AnomalyDAE	21.6±1.9 (25.4)	30.1±12.0 (44.8)	17.7±7.7 (27.8)
GAAN	19.6±0.3 (20.3)	**45.4**±0.1 (45.7)	28.0±0.1 (28.1)
GUIDE	18.8±0.6 (20.3)	OOM_C	OOM_C
CONAD	19.8±7.3 (25.4)	18.6±3.0 (20.2)	12.1±2.4 (14.3)

Table 12: Recall@k (%) comparison among OD algorithms on six datasets with organic outliers, where we show *the avg perf.* $\pm$ *the STD of perf.* (*max perf.*) of each. The best algorithm by expectation is shown in **bold**, while the max performance per dataset is marked with underline. OOM denotes out of memory with regard to GPU (_G) and CPU (_C). TLE denotes time limit of 24 hours exceeded.

Algorithm	Weibo	Reddit	Disney	Books	Enron	DGraph
LOF	22.0±0.0 (22.0)	**4.4**±0.0 (4.4)	0.0±0.0 (0.0)	0.0±0.0 (0.0)	0.0±0.0 (0.0)	TLE
IF	13.8±6.4 (24.3)	0.1±0.1 (0.3)	**9.2**±8.3 (16.7)	1.1±1.6 (3.6)	0.0±0.0 (0.0)	0.1±0.1 (0.4)
MLPAE	48.9±11.0 (62.1)	3.0±0.0 (3.0)	0.0±0.0 (0.0)	0.9±1.6 (3.6)	0.0±0.0 (0.0)	**0.5**±0.1 (0.6)
SCAN	23.8±7.0 (30.5)	2.7±0.3 (3.0)	7.5±11.2 (33.3)	0.7±1.4 (3.6)	0.0±0.0 (0.0)	TLE
Radar	**86.4**±0.8 (87.4)	2.1±0.8 (3.5)	0.0±0.0 (0.0)	0.0±0.0 (0.0)	0.0±0.0 (0.0)	OOM_C
ANOMALOUS	**86.4**±0.8 (87.4)	4.0±1.9 (7.9)	0.0±0.0 (0.0)	0.0±0.0 (0.0)	0.0±0.0 (0.0)	OOM_C
GCNAE	67.6±5.2 (77.3)	3.0±0.0 (3.0)	0.0±0.0 (0.0)	0.7±1.8 (7.1)	0.0±0.0 (0.0)	0.4±0.0 (0.4)
DOMINANT	19.7±13.8 (37.4)	0.9±0.4 (2.7)	3.3±6.7 (16.7)	1.6±3.1 (10.7)	0.0±0.0 (0.0)	OOM_C
DONE	65.4±12.4 (76.3)	2.8±1.6 (5.7)	0.0±0.0 (0.0)	1.1±1.6 (3.6)	0.0±0.0 (0.0)	OOM_C
AdONE	64.3±7.6 (74.3)	1.0±1.2 (3.8)	1.7±5.0 (16.7)	**3.0**±1.7 (7.1)	0.0±0.0 (0.0)	OOM_C
AnomalyDAE	42.2±23.7 (75.7)	0.9±0.5 (3.0)	0.0±0.0 (0.0)	2.7±2.2 (7.1)	0.0±0.0 (0.0)	OOM_C
GAAN	77.1±0.2 (77.4)	1.1±0.4 (2.2)	0.0±0.0 (0.0)	1.8±1.8 (3.6)	0.0±0.0 (0.0)	OOM_C
GUIDE	OOM_C	OOM_C	0.0±0.0 (0.0)	0.4±1.1 (3.6)	OOM_C	OOM_C
CONAD	20.3±13.3 (37.1)	1.3±1.6 (7.6)	0.8±3.6 (16.7)	1.7±2.9 (10.7)	0.0±0.0 (0.0)	0.4±0.1 (0.6)

C.2 Additional Results on Performance Variation under Different Types of Outliers

Table 13: Average Precision (%) comparison among OD algorithms on three datasets injected with contextual and structural outliers, where we show *the avg perf.* ± *the STD of perf.* (*max perf.*) of each. The best algorithm by expectation is shown in **bold**, while the max performance per dataset is marked with underline. OOM denotes out of memory with regard to GPU (_G) and CPU (_C).

	Cora		Amazon		Flickr	
Algorithm	Contextual	Structural	Contextual	Structural	Contextual	Structural
LOF	12.2±0.0 (12.2)	3.1±0.0 (3.1)	3.2±0.0 (3.2)	2.6±0.0 (2.6)	3.5±0.0 (3.5)	2.5±0.0 (2.5)
IF	9.2±1.1 (11.7)	3.0±0.3 (3.6)	4.7±2.1 (10.5)	2.6±0.1 (2.7)	7.2±1.0 (9.1)	2.6±0.0 (2.6)
MLPAE	13.7±0.0 (13.7)	3.1±0.0 (3.1)	56.7±0.1 (56.9)	2.5±0.0 (2.5)	**24.8**±0.0 (24.9)	2.6±0.0 (2.6)
SCAN	2.6±0.0 (2.7)	21.6±17.1 (61.0)	2.5±0.0 (2.5)	22.4±24.4 (71.0)	2.5±0.0 (2.5)	59.1±36.8 (94.8)
Radar	2.5±0.1 (2.6)	5.7±0.8 (6.6)	12.5±2.3 (14.3)	3.4±0.1 (3.6)	OOM_G	OOM_G
ANOMALOUS	2.7±0.2 (3.4)	5.1±2.5 (13.5)	10.4±1.4 (13.7)	3.4±0.4 (4.0)	OOM_G	OOM_G
GCNAE	13.7±0.0 (13.7)	3.1±0.0 (3.1)	**56.8**±0.1 (56.9)	2.5±0.0 (2.5)	18.9±9.1 (24.9)	2.6±0.0 (2.6)
DOMINANT	6.3±1.1 (6.9)	19.2±4.7 (22.1)	4.7±0.5 (5.2)	16.5±2.4 (17.4)	4.8±0.5 (5.1)	**61.3**±13.7 (66.9)
DONE	7.0±2.2 (11.9)	**31.9**±25.9 (89.0)	12.0±4.9 (20.6)	**23.0**±18.1 (69.1)	14.9±2.2 (18.0)	9.8±2.0 (13.6)
AdONE	8.9±1.3 (10.9)	16.2±6.2 (32.8)	14.5±7.4 (32.5)	13.1±8.3 (34.1)	11.4±2.6 (16.5)	10.2±1.5 (13.0)
AnomalyDAE	9.3±1.7 (13.2)	14.8±4.9 (24.7)	21.9±17.7 (47.7)	11.6±5.4 (17.2)	9.4±6.6 (24.2)	3.1±0.4 (3.8)
GAAN	**14.1**±0.1 (14.2)	4.5±0.1 (4.7)	52.5±1.4 (56.2)	3.6±0.2 (4.0)	24.7±0.2 (24.9)	2.6±0.0 (2.6)
GUIDE	13.7±0.5 (14.0)	3.9±0.2 (4.8)	OOM_C	OOM_C	OOM_C	OOM_C
CONAD	6.5±1.1 (6.9)	20.0±4.6 (22.2)	4.8±0.5 (5.2)	16.9±1.0 (17.4)	4.5±0.3 (4.9)	5.4±0.3 (5.8)

Table 14: Recall@k (%) comparison among OD algorithms on three datasets injected with contextual and structural outliers, where we show *the avg perf.* ± *the STD of perf.* (*max perf.*) of each. The best algorithm by expectation is shown in **bold**, while the max performance per dataset is marked with underline. k is set as the number of each type of outliers in labels. OOM denotes out of memory with regard to GPU (_G) and CPU (_C).

	Cora		Amazon		Flickr	
Algorithm	Contextual	Structural	Contextual	Structural	Contextual	Structural
LOF	12.9±0.0 (12.9)	5.7±0.0 (5.7)	1.1±0.0 (1.1)	3.7±0.0 (3.7)	9.0±0.0 (9.0)	2.2±0.0 (2.2)
IF	13.1±3.3 (20.0)	3.8±2.0 (8.6)	4.4±3.5 (13.4)	2.1±0.7 (3.1)	13.8±1.9 (17.7)	2.5±0.3 (3.0)
MLPAE	15.7±0.0 (15.7)	7.1±0.0 (7.1)	**55.1**±0.1 (55.1)	2.3±0.0 (2.3)	**29.8**±0.1 (30.0)	2.9±0.0 (2.9)
SCAN	2.8±1.1 (4.3)	25.9±19.9 (61.4)	2.2±0.4 (2.9)	**25.1**±26.8 (70.3)	2.8±0.2 (3.1)	59.7±37.1 (96.4)
Radar	1.4±0.0 (1.4)	0.6±0.7 (1.4)	15.1±6.5 (20.6)	1.5±1.1 (2.3)	OOM_G	OOM_G
ANOMALOUS	1.8±1.4 (4.3)	1.2±2.4 (8.6)	5.9±5.4 (19.1)	0.6±1.4 (4.6)	OOM_G	OOM_G
GCNAE	15.7±0.0 (15.7)	7.1±0.0 (7.1)	**55.1**±0.1 (55.1)	2.3±0.0 (2.3)	21.6±12.7 (30.1)	2.8±0.2 (3.0)
DOMINANT	8.4±3.1 (10.0)	17.2±5.2 (30.0)	4.9±0.8 (6.0)	10.1±1.9 (10.9)	3.6±0.2 (4.0)	**70.6**±15.7 (76.0)
DONE	8.6±3.9 (18.6)	**31.4**±26.2 (82.9)	11.6±4.0 (21.1)	23.7±19.9 (73.1)	22.4±3.0 (25.8)	5.1±1.0 (6.7)
AdONE	11.4±3.1 (17.1)	13.4±7.2 (35.7)	18.1±7.4 (30.6)	10.4±9.8 (30.9)	17.7±3.2 (23.9)	4.4±1.4 (7.7)
AnomalyDAE	12.0±2.9 (17.1)	16.0±4.2 (22.9)	23.5±21.4 (53.4)	8.8±2.9 (12.9)	10.1±10.5 (30.2)	3.6±1.7 (6.9)
GAAN	15.9±0.4 (17.1)	8.2±0.6 (8.6)	54.9±0.6 (56.3)	3.8±0.3 (4.6)	**29.8**±0.1 (30.1)	2.9±0.0 (3.0)
GUIDE	**16.9**±0.6 (17.1)	8.5±0.3 (8.6)	OOM_C	OOM_C	OOM_C	OOM_C
CONAD	9.1±2.4 (10.0)	17.2±3.9 (18.6)	5.3±0.8 (6.0)	10.2±1.2 (10.9)	7.8±1.9 (10.1)	8.3±1.4 (9.7)

C.3 Additional Results on Efficiency and Scalability Analysis

Table 15: Time consumption (s) comparison among OD algorithms on five different numbers of epochs. For non-iterative algorithms, i.e., LOF, IF, and SCAN, we report the total runtime.

Algorithm	10	100	200	300	400
LOF	0.10	0.10	0.10	0.10	0.10
IF	0.09	0.09	0.09	0.09	0.09
MLPAE	0.04	0.46	0.82	1.37	1.74
SCAN	0.02	0.02	0.02	0.02	0.02
Radar	0.02	0.10	0.19	0.34	0.36
ANOMALOUS	0.02	0.09	0.17	0.26	0.36
GCNAE	0.06	0.49	0.96	1.45	1.94
DOMINANT	0.08	0.70	1.41	2.10	2.79
DONE	0.08	0.77	1.53	2.30	3.08
AdONE	0.10	0.91	1.81	2.71	3.62
AnomalyDAE	0.43	0.64	1.28	1.92	2.55
GAAN	0.06	0.49	0.98	1.47	1.97
GUIDE	50.77	51.92	53.40	54.27	55.21
CONAD	0.11	1.04	2.07	3.07	4.10

Table 16: GPU memory consumption (MB) comparison among deep algorithms on five different graph sizes (number of nodes). Note that GPU memory measurement does not apply to algorithms like LOF, IF, and SCAN.

Algorithm	100	500	1000	5000	10000
MLPAE	0.66	2.10	3.90	19.41	38.52
GCNAE	0.92	3.40	6.38	31.18	62.11
GUIDE	0.96	3.46	6.47	31.45	62.60
Radar	0.49	9.32	36.73	871.32	3450.88
ANOMALOUS	0.44	5.03	20.50	482.47	2293.76
DOMINANT	1.09	7.58	27.15	591.74	2324.48
DONE	1.95	10.93	32.74	624.41	2385.92
AdONE	1.99	11.38	33.60	657.54	2447.36
AnomalyDAE	1.21	10.54	36.94	794.81	3112.96
GAAN	0.90	9.51	36.87	871.17	3450.88
CONAD	1.39	8.77	29.48	604.32	2344.96

D Long-term Maintenance and Development Plan

We commit to maintaining and developing BOND and PyGOD in the long run, as many of our open-source outlier detection works (e.g., PyOD [92], SUOD [90], and TODS [45]). More specifically, we will focus on improving on two aspects of graph OD tasks, namely datasets (Appx. D.1) and algorithms (Appx. D.2)

D.1 Enriching Graph OD Datasets

We will keep monitoring the coming datasets suited for BOND tasks, and enrich our testbed with more datasets. There are three main approaches for this:

1. ***Directly including new graph OD datasets***. We will keep checking graph OD papers to include their newly introduced datasets.
2. ***Adapting graph datasets for graph OD tasks***. As we have discussed in future directions in §5, we could repurpose existing graph datasets for OD. For instance, given a graph dataset with multiple types of transactions for node classification, we could combine the rare classes together as anomalies, and the common transactions as the normal class. Most of the time, we could find some semantic meaning for the combined rare classes, e.g., fraud and mistakes. This adaptation process has been widely used in tabular OD [23] and has proven to be useful [6]. Specifically, Open Graph Benchmark (OGB) [30], and therapeutic data commons (TDC) [31] can serve as natural sources for building graph OD datasets, and we will start from these repositories.
3. ***Planting more types of synthesized outliers into plain graphs.*** Our experimental results and analysis suggest that the existing synthesizing approaches are too naive and not similar to most organic outliers. Our future plan includes: 1) adopting other outlier generation approaches from [75, 3]; 2) generating outliers using learning-based methods like GAN [68].

With more graph OD datasets (e.g., # datasets $>= 20$), we could conduct more in-depth (group-wise and pairwise) statistical analysis [14], which has not been possible in BOND works. We will keep updating the benchmark site[1] for newly added datasets.

D.2 Emerging Graph OD Algorithms

Regarding graph OD algorithms, we will keep maintaining and improving PyGOD in multiple aspects:

1. ***Monitoring and adding outlier node methods to PyGOD*** for both benchmark and general usage.
2. ***Optimizing its accessibility and scalability*** with the latest development in graph learning [35], which may bring us new insights into outlier node detection's scalability.
3. ***Incorporating automated machine learning*** to enable intelligent model selection and hyperparameter tuning [59, 93], which may unlock some interesting perspectives of graph OD.
4. ***Extending the scope from static attribute outlier node detection to more graph tasks***, e.g., outlier detection in edges and sub-graphs. This will lead to other interesting aspects of graph OD.

Robustness and Quality. While building PyGOD, we follow the best practices of system design and software development. First, we leverage the continuous integration by *GitHub Actions*[2] to automate the testing process under various Python versions and operating systems. In addition to the scheduled daily test, both commits and pull requests trigger the unit testing. Notably, we enforce all code to have at least 90% coverage[3]. Following the `PEP8` standard, we enforce a consistent coding style and naming convention, which facilitates community collaboration and code readability.

In the long term, we envision PyGOD could keep evolving to support more comprehensive benchmarking, as well as other graph detection tasks and benchmarks.

[1] `https://github.com/pygod-team/pygod/tree/main/benchmark`

[2] Continuous integration by GitHub Actions: `https://github.com/pygod-team/pygod/actions`

[3] Code coverage by Coveralls: `https://coveralls.io/github/pygod-team/pygod`