AnomalyBench: An Open Benchmark for Explainable Anomaly Detection

Vincent Jacob
Ecole Polytechnique
vincent.jacob@polytechnique.edu

Fei Song
Ecole Polytechnique
fei.song@polytechnique.edu

Arnaud Stiegler
Ecole Polytechnique
arnaud.stiegler@polytechnique.edu

Yanlei Diao
Ecole Polytechnique
yanlei.diao@polytechnique.edu

Nesime Tatbul
Intel Labs and MIT
tatbul@csail.mit.edu

Abstract

Access to high-quality data repositories and benchmarks have been instrumental in advancing the state of the art in many domains, as they provide the research community a common ground for training, testing, evaluating, comparing, and experimenting with novel machine learning models. Lack of such community resources for anomaly detection (AD) severely limits progress. In this report, we present AnomalyBench, the first comprehensive benchmark for explainable AD over high-dimensional (2000+) time series data. AnomalyBench has been systematically constructed based on real data traces from ∼100 repeated executions of 10 large-scale stream processing jobs on a Spark cluster. 30+ of these executions were disturbed by introducing ∼100 instances of different types of anomalous events (e.g., misbehaving inputs, resource contention, process failures). For each of these anomaly instances, ground truth labels for the root-cause interval as well as those for the effect interval are available, providing a means for supporting both AD tasks and explanation discovery (ED) tasks via root-cause analysis. We demonstrate the key design features and practical utility of AnomalyBench through an experimental study with three state-of-the-art semi-supervised AD techniques.

1 Introduction

Anomaly detection (AD) refers to the task of identifying patterns in data that deviate from a given notion of normal behavior [10]. It finds use in almost every domain where data is plenty, but unusual patterns are the most critical to respond (e.g., cloud telemetry, autonomous driving, financial fraud management). AD over time series data has been of particular interest, not only because time-oriented data is highly prevalent and voluminous, but also more challenging to analyze due to its complex and diverse nature (e.g., multi-variate time series can consist of 1000s of dimensions; anomalous patterns may be of arbitrary length and shape; there may be intricate cause and effect relationships among these patterns; data is rarely clean). Furthermore, by helping uncover how or why a detected anomaly may have happened, *explanation discovery* (ED) (a.k.a., *root-cause analysis (RCA)*) forms a crucial capability for any time series AD system.

Recent advances in data science and machine learning (ML) significantly reinforced the need for developing robust anomaly detection and explanation solutions that can be reliably deployed in production environments [20, 29]. However, progress has been rather slow and limited. While there is extensive research activity going on [9], proposed solutions have been mostly adhoc and far from

Preprint. Under review.

being generalizable to realistic settings. We believe that one of the critical roadblocks to progress has been the lack of open data repositories and benchmarks to serve as a common ground for reproducible research and experimentation. Indeed, access to such community resources has been instrumental in advancing the state of the art in many other domains (e.g., [12, 35, 26, 4]). Inspired by those efforts, in this report, we present *AnomalyBench*, the first comprehensive public benchmark for explainable anomaly detection over high-dimensional time series data.

AnomalyBench focuses on the familiar domain of metric monitoring in large-scale computing systems, and provides a curated dataset and an evaluation methodology based on this dataset.

Dataset: We constructed AnomalyBench systematically based on real data traces collected from around 100 repeated executions of 10 distributed stream processing jobs on an Apache Spark cluster over a period of 2.5 months. More than 30 of these executions were disturbed by introducing nearly 100 instances of 5 different classes of anomalous events (e.g., misbehaving inputs, resource contention, process failures) [5]. For each of these anomaly instances, we provide ground truth labels for both the root-cause interval as well as the corresponding effect interval, thereby enabling the use of our dataset in both AD and ED tasks. Overall, both the normal (*undisturbed*) and the anomalous (*disturbed*) traces contain enough variety (including some noise due to Spark's inherent behavior) to capture real-world data characteristics in this domain (Table 1).

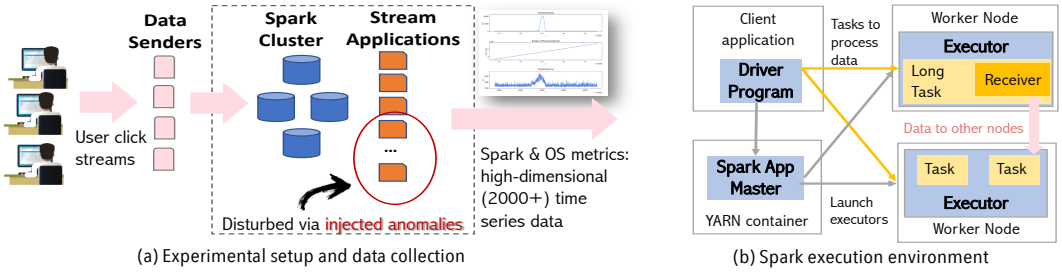
Evaluation Methodology: We designed AnomalyBench primarily targeting *semi-supervised deep learning based AD techniques* (i.e., trained only with normal data, possibly with occasional noise, and then tested against anomalous data) for *range-based anomalies* (i.e., contextual and collective anomalies occurring over a time interval instead of only at a single time point) over *high-dimensional* (i.e., multi-variate with 1000s of dimensions) time series. This decision is informed by our observation of this being the most common and inclusive usage scenario in practice. AnomalyBench evaluates a given AD approach A in terms of two key criteria: (i) functionality and (ii) generalization capability. Functionality refers to how well A can handle AD or ED. AD functionality is tested in terms of four complementary requirements (Existence, Latency, Duplicates-Free, Range Detection), whereas ED functionality is tested at two different levels (How-Explanation, Why-Explanation). Similarly, generalization capability is intended to test how well A's learned AD model generalizes to increasingly more challenging learning settings (1-App, N-App, N-App-Few-Shot). Overall, AnomalyBench provides a rich and challenging testbed with a well-organized evaluation methodology (Table 2).

Compared to current public time series datasets [2, 11, 13, 23, 33], the key contribution of AnomalyBench is that it comprehensively covers one challenging application domain end to end, as opposed to providing multiple small-scale and simple datasets from a bunch of independent domains. This provides an opportunity for a more in-depth investigation and evaluation of ML models, potentially revealing new insights for accelerating research progress in explainable anomaly detection. In the rest of the report, we first briefly summarize related work. After presenting our data collection and benchmark construction process in more detail, we demonstrate the practical utility of AnomalyBench through an experimental analysis of three state-of-the-art AD algorithms using a selected set of evaluation criteria from our benchmark. We finally conclude with an outline of future directions.

2 Related Work

Anomaly Detection and Explanation. There is a long history of research in anomaly detection (AD) [10, 16]. The high degree of diversity in data characteristics, anomaly types, and application domains has led to a plethora of AD approaches from simple statistical [6] to traditional machine learning (ML) [8, 27] to deep learning (DL) [9] methods. In our experimental study, we particularly focus on three DL methods that represent the recent state of the art [30, 48, 36] (detailed in §5). While gaining more traction lately, anomaly explanation (AE) is relatively less explored [3, 50, 41, 22, 20, 29, 34, 46]. We hope our benchmark brings more attention to this important problem.

Datasets and Benchmarks. Datasets and benchmarks like ours have been published to support the advancement of research in many different domains. Examples include: object recognition (ImageNet [12, 35], COCO [26], ObjectNet [4], COOS [28]), scene understanding (SUN Database [47]), natural language processing (WordNet [31], GLUE [45], SuperGLUE [44]), traffic forecasting (STREETS [40]), weather forecasting (ExtremeWeather [32], HKO-7 [38]), and visualization design (VizNet [19]). Well-known data archives used by ML and time series research communities include: UCI [13], UCR [11], and UEA [2]. Most of these archives provide a collection of real-world datasets created for general ML tasks such as classification and clustering. While the need for systematically constructing AD benchmarks from real data has been recognized by previous researchers [14], public availability of



(a) Experimental setup and data collection (b) Spark execution environment

Figure 1: Use case scenario: anomaly detection in Spark application monitoring

anomaly detection datasets is severely limited [33]. To our knowledge, Numenta Anomaly Benchmark (NAB) is the only public benchmark designed for time series anomaly detection [23]. NAB provides 50+ real and artificial datasets primarily focusing on real-time anomaly detection for streaming data. Compared to ours, each of these datasets is much smaller in scale and dimensionality, and does not capture any information to enable anomaly explanations. NAB also has several technical weaknesses that are heavily criticized for hindering its use in practice [39]. Lastly, there have been some recent industrial effects on time series anomaly explanation and root-cause analysis [20, 29], but they are largely based on proprietary code and datasets that are not accessible to the research community.

3 Data Collection

The AnomalyBench dataset has been systematically constructed based on real data traces collected from a use case scenario that we implemented on Apache Spark. In this section, we first describe this scenario, followed by the details of how we created the normal and anomalous data traces themselves.

3.1 Use Case Scenario: Spark Application Monitoring

Petabyte-scale big data analytics applications are being deployed on Apache Spark clusters everyday [42]. Monitoring the execution of these jobs to ensure their correct and timely completion via AD can be business-critical. We model this widespread and challenging AD use case in our benchmark.

System Setup. Our Spark workload consists of 10 stream processing applications that analyze user click streams from the WorldCup 1998 website [24] (replicated with a scale factor for our long-running applications). As seen in Figure 1(a), *Data Sender* servers ingest the streams at a controlled *input rate* to a Spark cluster of 4 nodes, each with 2 Intel® Xeon® Gold 6130 16-core processors, 768GB of memory, and 64TB disk. Each streaming application has certain workload characteristics (e.g., CPU or I/O intensive) and is executed by Spark in a distributed manner, as in Figure 1(b).

Submitted an application, Spark first launches a *Driver* process to coordinate the execution of this application. The driver connects to a resource manager (Apache Hadoop YARN), which launches *Executor* processes on a subset of cluster nodes where tasks (a unit of work on a data partition, e.g., `map` or `reduce`) will be executed in parallel. Further, given 32 cores each node can run tasks from multiple applications concurrently. As this is common practice in real-world production environments, we run 5/10 randomly selected applications at a time. The placement of Driver and Executor processes to cluster nodes for these applications is decided by YARN based on data locality, load on nodes. etc. Except for I/O activities, YARN offers container isolation for resource usage of all parallel processes.

Metrics Collected. During the execution of each application, we collected metrics from both the Spark Monitoring and Instrumentation interface (UI) and underlying operating system (OS). We call each such dataset a *Trace*. Table 1(a) gives a summary of the metrics collected per trace. The Driver yields 243 Spark UI metrics offering information such as scheduling delay, statistics on the streaming data received and processed, etc. Each executor provides 140 metrics on various time measurements, data sizes, network traffic, as well as memory and I/O activities. As we wanted to keep the number of metrics the same for all traces, we set a fixed limit of 5 for the number of Spark executors (3 active + 2 backup). This way, even if an active executor fails during a run and a backup takes over, the number of metrics collected stays the same, $5 \times 140 = 700$, with null values set for inactive executors. 335 OS metrics for each of the 4 cluster nodes are collected using the `Nmon` command, capturing CPU time, network traffic, memory usage, etc. All in all, each trace consists of a total of 2,283 metrics recorded each second for 7 hours on average, constituting a multi-dimensional time series.

Metric Type	Spark UI Driver	Spark UI Executor	OS (Nmon)
# of Metrics	243	5 x 140 = 700	4 x 335 = 1340
Total	2,283		

Frequency	1 data item per second
Data items	2,335,781
Duration	649 hours
Total size	24.6 GB

(a) Metrics and data size

Trace Type	Anomaly Type	# of Traces	Anomaly Instances	Anomaly Length min, avg, max	Data Items
Undisturbed	N/A	59	N/A	N/A	1.4M
Disturbed	Type 1: Bursty input	6	29	14m, 21m, 31m	360K
Disturbed	Type 2: Bursty input to crash	7	7	8m, 35m, 1.5h	31K
Disturbed	Type 3: Stalled input	4	16	14m, 15m, 15m	187K
Disturbed	Type 4: CPU contention	6	26	8m, 15m, 26m	181K
Disturbed	Type 5: Process failure	11	19	10s, 12m, 2.8h	128K

Ground truth label	(`app_id`, `trace_id`, `anomaly_type`, `root_cause_start`, `root_cause_end`, `extended_effect_start`, `extended_effect_end`)

(b) Normal and anomalous traces

Table 1: The AnomalyBench dataset

3.2 Trace Generation

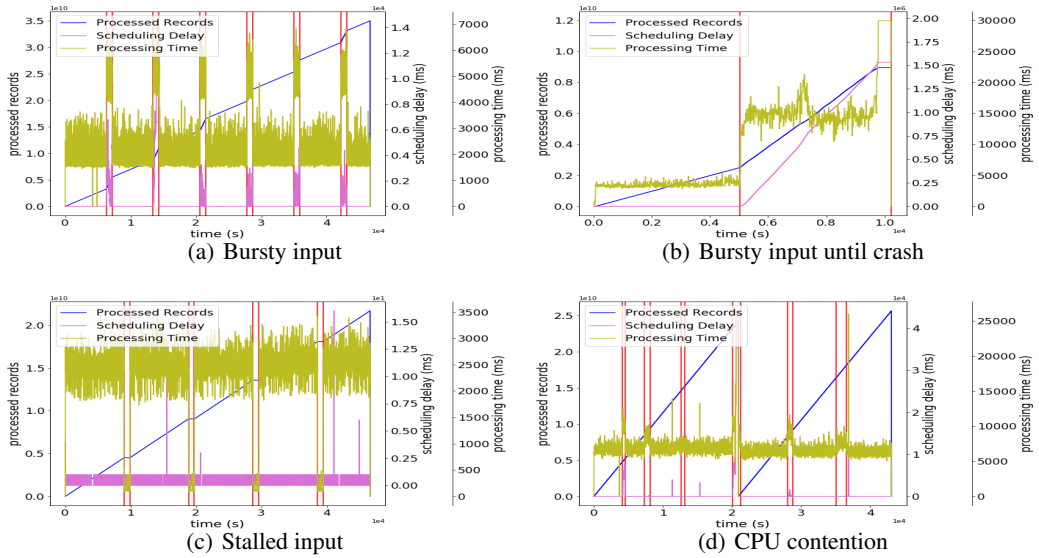
In generating our normal and anomalous traces, we followed the principles of chaos engineering (i.e., an approach devised by high-tech companies like Netflix for injecting failures and workload surges into a production system in order to verify/improve its reliability) [5]. Thus, we first generated *undisturbed traces* to characterize the normal execution behavior of our Spark cluster; we then introduced various anomalous events to generate *disturbed traces*. Table 1(b) provides an overview.

Undisturbed Traces. Uninterrupted executions of 5 randomly selected applications at a time, at parameter settings within the capacity limits of our Spark cluster, over a period of 1 month, gave us 59 undisturbed traces of 15.3GB in size. Any instances of occasional cluster downtime were manually removed from these traces. It is important to note that, although undisturbed, these traces still exhibit occasional variations in metrics due to Spark's inherent system mechanisms (e.g., checkpointing, CPU usage by a DataNode in the distributed file system). Since such variations do appear in almost every trace, we consider them as part of the normal system behavior. In other words, our normal data traces include some "noise", as most real-world datasets typically do.

Disturbed Traces. Disturbed traces are obtained by introducing anomalous events during an execution. Based on discussions with industry contacts from the Spark ecosystem, we came up with 5 types of anomalous events. When designing these, we considered that: (i) they lead to a visible effect in the trace, (ii) they do not lead to an instant crash of the application (since AD would be of little help in this case), (iii) they can be tracked back to their root causes. We briefly describe these anomalies below.

- **Bursty Input (Type 1):** To mimic input rate spikes, we ran a disruptive event generator (DEG) on the Data Senders to temporarily increase the input rate by a given factor for a duration of 15-30 minutes. We repeated this pattern multiple times during a given trace, creating a total of 29 instances of this anomaly type over 6 different traces. Please see Figure 2(a) for an example.
- **Bursty Input Until Crash (Type 2):** This is a longer variation of the Type 1 anomaly, where the DEG period lasts forever, crashing the executors due to lack of memory. When an executor crashes, Spark launches a replacement, but the sustained high rates will keep crashing the executors, until eventually Spark decides to kill the whole application. We injected this anomaly into 7 different traces. Figure 2(b) shows an example.
- **Stalled Input (Type 3):** This anomaly mimics failures of Spark data sources (e.g., Kafka or HDFS). To create it, we ran a DEG that set the input rates to 0 for about 15 minutes, and then periodically repeated this pattern every few hours, giving us a total of 16 anomaly instances across 4 different traces. Figure 2(c) shows an example.
- **CPU Contention (Type 4):** The YARN resource manager cannot prevent external programs from using the CPU cores that it has allocated to Spark processes, causing scheduling delays to build up due to CPU contention. We reproduced this anomaly using a DEG that ran Python programs to consume all CPU cores available on a given Spark node. We created 26 such anomaly instances over 6 different traces. See Figure 2(d) for a trace with multiple instances of this type of anomaly.
- **Process Failure (Type 5):** Hardware faults or maintenance operations may cause a node to fail all of a sudden, making all processes located on that node unreachable. Such processes must be restarted on another node, which causes delays. We created such anomalies by failing executor processes, which get restarted 10 seconds after the failure, but its effects on metrics such as processing delay continue longer. We also created anomalies by failing driver processes, where the number of processed records drops to 0 until the driver comes back up again in about 20 seconds.

For all of these 97 anomaly instances over 34 anomalous traces, we provide ground truth labels for both *root cause intervals* as well as their respective *extended effect intervals*. Root cause intervals



(a) Bursty input (b) Bursty input until crash

(c) Stalled input (d) CPU contention

Figure 2: Metrics observed for anomaly types 1-4: each pair of red vertical bars marks a root cause event

typically correspond to the time period during which DEG programs are running, whereas the extended effect intervals are the time periods that start immediately after a root cause interval and end when important system metrics return to normal values or the application is eventually pushed to crash. The effect intervals are manually determined using domain knowledge. The format of *ground truth labels* is shown in Table 1(b).

4 Benchmark Design

In this section, we present the evaluation methodology we designed to benchmark anomaly detection (AD) and explanation discovery (ED) algorithms based on the curated, high-dimensional time series dataset described in the previous section. As summarized in Table 2, AnomalyBench is designed to evaluate AD and ED algorithms in terms of two orthogonal aspects: functionality and generalization capability. For each of these, the benchmark provides multiple levels of challenges for systematically assessing the capabilities of AD and ED solutions. In terms of functionality, an AD algorithm gets more advanced, the more requirements it can meet, with AD1 being the most fundamental. So does an ED algorithm. In terms of generalization capability, LM3 represents the ultimate level of success. We briefly summarize all these levels below.

Functionality: Anomaly Detection (AD). First and foremost, we designed AnomalyBench targeting *semi-supervised deep learning based AD techniques* (i.e., trained only with normal data, possibly with occasional noise, and then tested against anomalous data) for *range-based anomalies* (i.e., contextual and collective anomalies occurring over a time interval instead of only at a single time point) over *high-dimensional* (i.e., multi-variate with 1000s of dimensions) time series. This decision is informed by our observation of this being the most common and inclusive usage scenario in practice. There are 4 complementary levels of AD functionality, listed from the most basic towards more advanced:

AD1 (*Existence*): The first expectation is to flag the existence of an anomaly somewhere within the *anomaly interval* (i.e., the root cause interval + the extended effect interval).

AD2 (*Latency*): The next expectation is to minimize the *detection latency*, i.e., the difference between the time an anomaly is flagged and the start time of the corresponding root cause interval.

AD3 (*Duplicates-Free*): The third expectation is to report each anomaly instance exactly once. Duplicate detections are undesirable, because they may not only redundantly cause repeated alerts for a single anomalous event, but also confusion if those alerts are for the same anomaly event or not.

AD4 (*Range Detection*): The last expectation is to report not only the existence, but also the precise time range of an anomaly. The wider a range of an anomaly interval an AD algorithm can detect, the better its understanding of the real-world phenomena underlying that anomaly.

In order to assess how satisfactorily an AD algorithm can meet these four functionality levels, we use the customizable evaluation framework recently proposed by Tatbul et al. [43].

Functionality		Generalization Capability
Anomaly Detection (AD)	**Explanation Discovery (ED)**	**Learning Method (LM)**
AD1: Existence	ED1: How-Explanation	LM1: 1-App Learning
AD2: Latency	ED2: Why-Explanation (RCA)	LM2: N-App Learning
AD3: Duplicates-Free		LM3: N-App Few-Shot Learning
AD4: Range Detection		

Table 2: The AnomalyBench evaluation methodology

Functionality: Explanation Discovery (ED). Time series anomalies are rarely independent events. Therefore, it is often desirable to find out the real reasons (i.e., the root cause events) that led to an observed anomaly. ED algorithms are expected to return compact, human-readable representations for such root causes, a.k.a., *an explanation*. Following the anomaly explanation model recently proposed by Zhang et al. [50], AnomalyBench differentiates between two forms of explanations:

ED1 (*How-Explanation*): The first level aims at explaining how the AD model came to recognize an anomaly. For example, the how-explanation for a bursty input anomaly could be a description such as "$processing_time > 10$s and $block_manager_mem > 10$GB". AnomalyBench evaluates how-explanation based on the following three criteria: (i) *Compactness*: This corresponds to the number of metrics used in the explanation; the smaller, the better (humans tend to favor more concise explanations). (ii) *Consistency*: Anomalies of the same type occurring in a similar context (e.g., for the same application) should have consistent explanations. AnomalyBench evaluates this criteria based on comparing histograms of metrics used in the explanations of similar types of anomalies. (iii) *Concordance*: This criteria is for identifying "discordant" metrics reported in explanations, i.e., those that are only incidentally correlated with an anomaly (e.g., metrics from node A used to explain a CPU contention anomaly on node B). AnomalyBench uses the information on root cause events provided in the ground truth table to assess concordance of a given how-explanation.

ED2 (*Why-Explanation*): The second level of explanation, referred to as why-explanation, aims to point to the root cause of the detected anomaly.

In general, identifying the actual root cause is much harder than explaining how an anomaly is flagged, but provides deeper insights about anomalies. Again, take the bursty input example. A why-explanation such as "$received_records[t] - received_records[t-1] > 10$M" would be more informative than the how-explanation given above. AnomalyBench provides root cause ground truth labels for all anomaly event instances in its dataset, which can be used for assessing if a given ED algorithm is competent at generating why-explanations.

Generalization Capability. AnomalyBench also tests how well a learned AD model generalizes to different application (A), input rate (R), and concurrency (C) characteristics, under three settings:

LM1 (*1-App Learning*): Spark applications differ in their workload characteristics (e.g., some are CPU intensive, while others are I/O intensive). Such characteristics may lead to different run-time observations (e.g., a CPU-intensive application would be more sensitive to CPU contention anomalies), making detecting anomalies across multiple applications' traces a more challenging task. 1-App learning focuses on training and evaluating an AD model on a single application basis. As such, it is the easiest level of generalization.

LM2 (*N-App Learning*): In N-App learning, the focus is on training an AD model to detect anomalies across multiple applications. Recall that each trace in our dataset contains data for 5 concurrently running applications (randomly chosen from 10 applications), running at different input rates. A well-learned model is supposed to generalize across these differing workload settings across multiple traces. In the LM2 setting, AnomalyBench allows the AD algorithm to use *any* normal data for training (including normal data from a disturbed trace, while testing on the anomalies from the same trace). This way, the model is given a chance to "peek" at the normal state (including all workload characteristics) of a particular trace, which will make it easier to detect anomalies later in the trace.

LM3 (*N-App Few-Shot Learning*): In a truly realistic setting, model training is unlikely to "peek" at normal portions of an anomalous trace that will later be used for testing. In other words, an AD model trained with normal traces with certain (A, R, C) settings may later be subject to a new anomalous trace with a previously unseen (A, R, C) setting. To simulate this scenario, AnomalyBench reserves a disturbed trace entirely for testing (in each leave-out experiment). Hence, the training problem at LM3 bears similarity with few-shot learning [37] and poses a greater challenge for learning.

	LSTM				VAE				GAN			
LM2	AUC	F1	Prec.	Rcl	AUC	F1	Prec.	Rcl	AUC	F1	Prec.	Rcl
T1	0.73	0.02	0.07	0.01	0.71	0.02	0.07	0.01	0.62	0.26	0.21	0.35
T2	0.98	0.76	0.72	0.81	0.98	0.76	0.71	0.81	0.77	0.26	0.16	0.69
T3	0.69	0.00	0.00	0.00	0.73	0.06	0.13	0.04	0.56	0.14	0.09	0.31
T4	0.97	0.70	0.80	0.63	0.97	0.68	0.78	0.60	0.80	0.37	0.26	0.67
T5	0.95	0.68	0.71	0.65	0.96	0.70	0.71	0.69	0.81	0.29	0.18	0.69
LM3	AUC	F1	Prec.	Rcl	AUC	F1	Prec.	Rcl	AUC	F1	Prec.	Rcl
T1	0.86	0.04	0.03	0.11	0.80	0.12	0.17	0.15	0.77	0.10	0.14	0.16
T2	1.00	0.87	0.96	0.82	1.00	0.83	0.73	1.00	0.84	0.72	0.71	0.86
T3	0.52	0.03	0.04	0.02	0.57	0.18	0.20	0.17	0.50	0.12	0.12	0.15
T4	0.86	0.55	0.47	0.81	0.73	0.34	0.21	0.95	0.89	0.62	0.55	0.80
T5	0.62	0.18	0.12	0.55	0.66	0.26	0.17	0.77	0.63	0.26	0.22	0.47

Table 3: Results of DL-based AD methods in the (**AD1, LM2**) and (**AD1, LM3**) settings of AnomalyBench

5 Experimental Study

In this section, we apply three DL-based AD methods on our dataset. By analyzing the results, we exhibit the value of our dataset for the task of AD. Further, we discuss various issues related to the dataset, the learning settings, and some potential future directions in AD.

5.1 Experimental Setup

Training Strategies. We split our dataset into 1) training including validation, 2) threshold selection (TS), and 3) test. We included only normal data in training and TS, because it is most practical for real-world use, i.e., excluding the known anomalies, but still including some random noise and unknown anomalies. It can be seen as a *"noisy" semi-supervised learning* problem. We implemented the learning methods LM1 - LM3 (see Table 2).

Anomaly Detection (AD). We consider recent deep learning based, semi or unsupervised methods, and also different modeling choices including discriminative, generative, and hybrid models (see [9], Table 3). As a result, we implemented the following three state-of-the-art AD methods:

1) LSTM Long Short-Term Memory (LSTM) [18] is a deep neural network structure designed for handling sequential data. In the context of AD, it tries to predict the values for an upcoming period, and uses the prediction error as the *anomaly score*, which is a discriminative approach. The first work to use LSTM for AD [30] works for univariate time series in the supervised setting. More recent work [7] supports collective AD. Our experiment follows the approach of the latter, defining the anomaly score of a test window as its *Averaged Relative Error* (ARE) by the LSTM model.

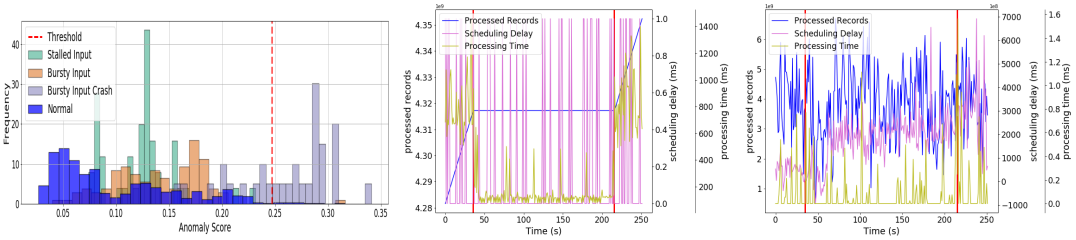
2) AE and VAE An Auto-Encoder (AE) [17] aims to learn an artificial neural structure such that the input data can be reconstructed via this structure. Variational Auto-Encoder (VAE) [21], which is able to learn a distribution of possible values, can use the reconstruction probability to compute an anomaly score for AD. Our work is the extension on recent work on VAE based AD [48] (which is a follow-up of [1]), where we make it applicable for multi-dimensional data.

3) GAN Generative Adversarial Networks (GANs) [15] learn to generate samples with similar statistics as their training data. In the context of AD, we expect the trained generator to learn a mapping from its latent space to realistic normal windows. Our experiment follows an approach similar to [36], where for each test window we iterate through latent space to find the representation yielding its most similar generated sample. The anomaly score of a window is then derived from the difference between its real and generated versions. Like in [25], we constructed the generator network using LSTM units rather than convolutional layers, for better handling time dependencies.

Threshold Selection. AnomalyBench does not offer labeled data for threshold selection. Hence, we use unsupervised threshold selection methods. Among the methods listed in a recent survey [49], we chose three most used automatic techniques: SD, MAD, and IQR. For each AD method (LSTM, VAE, or GAN), we run hyper-parameter tuning for each TS technique to find a set of potential thresholds. Then we apply these thresholds on the test dataset and report the one with best performance.

5.2 Results and Discussion

Table 3 summarizes the results of the three AD methods for AD1 (Existence) functionality. It shows the results first for the LM2 learning method, and then for LM3. (The results of LM1 are similar to those of LM2, hence omitted in the interest of space.) For each learning method, each row indicates



(a) Distribution of anomaly scores (b) Original metrics (c) Reconstructed metrics

Figure 3: Illustrating difficulty in semi-supervised AD under noise and information loss after PCA

the result for one type of anomaly. For each AD method, we report on four metrics: ROC AUC is used to judge how good the "anomaly score" is in numerical form. F1-score is used to judge how accurate the binary-valued AD is, which requires an additional threshold selection procedure. Precision and Recall focus on different accuracy aspects of the AD results, and F1-score is their harmonic mean. We make the following observations from Table 3, with additional evidence shown in Figure 3.

General Value. Consider LM2 results first. The AUC measurement (which does not depend on the threshold selection) clearly shows that our data set is valuable for AD no matter which methods we use. Most methods achieve good AUC values for anomaly types 2, 4, and 5, and modest values for the other two types. This shows that our data set can be used to evaluate/verify different AD methods.

AUC versus F1. The AUC performance is always better than the F1-score, often with a big gap between the two. This is due to the complexity of threshold selection. As we can see in Figure 3(a), when there are several anomaly types, they tend to have different distributions of anomaly scores, and some distributions overlap with that of normal data. The implications are two fold: First, the distribution of normal data overlaps significantly with those of T1 and T3 (which have the worst F1-score), making it hard to find a good threshold to separate the normal data from these anomalies. The fundamental reason is that the normal data contains noise, which is indicative of real-world datasets and calls for additional research on semi-supervised AD under noisy data. Second, if we move the threshold leftward in Figure 3(a), we achieve better F1-score for T1 and T3 types, but at the same time worse F1-scores for the other three types. Since in real-world datasets it is not possible to know all anomaly types in advance and separate data accordingly for training, this indicates the need for new threshold selection methods for handling mixed anomaly types in the data.

Anomaly Types. Some types of anomaly are harder to detect than others. As the table shows, all the methods found it harder to detect types T1 and T3. Our profiling points the reason to information loss in dimensionality reduction. Since our data is high-dimensional, common practice is to apply Principal Component Analysis (PCA) for reducing dimensionality [25] of the input to AD algorithms. However, the PCA model trained on normal data may deem some features with small variance unimportant, while these features behave differently in an anomaly and offer a strong signal for detection, which is unfortunately lost via PCA. In Figures 3(b) and 3(c), we show three metrics around a stalled input anomaly, both in original values and reconstructed values after applying PCA. The number of processed records, which we consider to be a strong signal for the anomaly, suffers from severe information loss after the transformation and hence may lead to the lower performance for T1 and T3. Finally, note that if we do not use PCA, the performance of most AD algorithms does not improve but the running time degrades, indicating that the lower layers of neural networks, known to perform feature engineering, may suffer from the same data loss problem.

Few-shot Learning. Finally, consider the difference when the learning method changes from LM2 to LM3. The AUC measurement improves somewhat for T1 and T2, but drops significantly for T3-T5 (see LSTM and VAE). Regarding F1-score, the drop is even more dramatic in T4-T5 (e.g., from 0.68 to 0.18). These results indicate the difficulty in the few-shot learning setting: the training data does not include normal data from the test trace, and hence has no chance to observe the particular test environment including the input rate, concurrency, and other factors. The poor performance of existing AD methods calls for new AD methods with proper generalization ability. More specifically, it should have the generalization power to forecast the normal behavior for unseen applications or unseen environments including the new input rate, concurrency, and even new hardware.

6 Conclusions

In this report, we presented AnomalyBench – a novel public benchmark for explainable anomaly detection. Further, we demonstrated the utility of AnomalyBench through an experimental analysis

of three recent DL-based AD algorithms, focusing on the most basic functionality of the benchmark (existence of an anomaly). Results show that our dataset is valuable for evaluating AD algorithms due to rich signals and diverse anomaly types included in the data. Yet more importantly, our results reveal the limitations of these state-of-the-art AD methods for semi-supervised learning under noise (i.e., training data is subject to noise and unknown anomalies), mixed anomaly types (i.e., anomaly types are unknown and mixed in the data), and in the few-shot learning setting (e.g., each test trace may represent an unseen application or unseen computing environment in terms the input rate, concurrency, or other system aspects). These results call for new research to advance the current state of the art of AD, as well as integrated solutions to anomaly and explanation discovery. Going forward, we envision AnomalyBench to develop into a collaborative community platform for fostering reproducible research and experimentation in the area. We intend to actively maintain and extend this platform, as well as welcoming feedback and contributions from the anomaly detection community.

Acknowledgments and Disclosure of Funding

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement n°725561).

References

[1] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Technical Report; SNU Data Mining Center: Seoul, Korea*, 2015.

[2] Anthony J. Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn J. Keogh. The UEA Multivariate Time Series Classification Archive, 2018. *CoRR*, abs/1811.00075, 2018.

[3] Peter Bailis, Edward Gan, Samuel Madden, Deepak Narayanan, Kexin Rong, and Sahaana Suri. MacroBase: Prioritizing Attention in Fast Data. In *ACM International Conference on Management of Data (SIGMOD)*, pages 541–556, 2017.

[4] Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Dan Gutfreund, Josh Tenenbaum, and Boris Katz. ObjectNet: A Large-Scale Bias-controlled Dataset for Pushing the Limits of Object Recognition Models. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 9453–9463, 2019.

[5] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal. Chaos engineering. *IEEE Software*, 33(3):35–41, 2016.

[6] A. M. Bianco, M. G. Ben, E. J. Martinez, and V. J. Yohai. Outlier Detection in Regression Models with ARIMA Errors using Robust Estimates. *Journal of Forecasting*, 20(8):565–579, 2001.

[7] Loïc Bontemps, Van Loi Cao, James McDermott, and Nhien-An Le-Khac. Collective anomaly detection based on long short-term memory recurrent neural networks. In *Future Data and Security Engineering - Third International Conference, FDSE 2016*, volume 10018, pages 141–152, 2016.

[8] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. *SIGMOD*, pages 93–104, 2000.

[9] Raghavendra Chalapathy and Sanjay Chawla. Deep Learning for Anomaly Detection: A Survey. *CoRR*, abs/1901.03407, 2019.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3):15:1–15:58, 2009.

[11] Hoang Anh Dau, Anthony J. Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn J. Keogh. The UCR Time Series Archive. *CoRR*, abs/1810.07758, 2018.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[13] Dheeru Dua and Casey Graff. The UCI Machine Learning Repository. `http://archive.ics.uci.edu/ml/`.

[14] Andrew F. Emmott, Shubhomoy Das, Thomas Dietterich, Alan Fern, and Weng-Keen Wong. Systematic Construction of Anomaly Detection Benchmarks from Real Data. In *ACM SIGKDD Workshop on Outlier Detection and Description (ODD)*, pages 16—21, 2013.

[15] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680, 2014.

[16] Manish Gupta, Jing Gao, Charu C. Aggarwal, and Jiawei Han. Outlier detection for temporal data: A survey. *IEEE Trans. Knowl. Data Eng.*, 26(9):2250–2267, 2014.

[17] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[19] Kevin Zeng Hu, Snehalkumar (Neil) S. Gaikwad, Madelon Hulsebos, Michiel A. Bakker, Emanuel Zgraggen, César A. Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çagatay Demiralp. VizNet: Towards a Large-Scale Visualization Learning and Benchmarking Repository. In *Conference on Human Factors in Computing Systems (CHI)*, page 662, 2019.

[20] Vimalkumar Jeyakumar, Omid Madani, Ali Parandeh, Ashutosh Kulshreshtha, Weifei Zeng, and Navindra Yadav. ExplainIt! - A Declarative Root-cause Analysis Engine for Time Series Data. In *ACM International Conference on Management of Data (SIGMOD)*, pages 333–348, 2019.

[21] Diederik P. Kingma and Max Welling. Stochastic gradient vb and the variational auto-encoder. In *International Conference on Learning Representations ICLR*, pages 14–16, 2014.

[22] Martin Kopp, Tomás Pevný, and Martin Holena. Anomaly Explanation with Random Forests. *Expert Systems with Applications*, 149, 2020.

[23] A. Lavin and S. Ahmad. Evaluating Real-Time Anomaly Detection Algorithms - The Numenta Anomaly Benchmark. In *IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 38–44, 2015.

[24] Boduo Li, Edward Mazur, Yanlei Diao, Andrew McGregor, and Prashant J. Shenoy. Scalla: A platform for scalable one-pass analytics using mapreduce. *ACM Trans. Database Syst.*, 37(4):27, 2012.

[25] Dan Li, Dacheng Chen, Jonathan Goh, and See kiong Ng. Anomaly detection with generative adversarial networks for multivariate time series. In *7th International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications on the ACM Knowledge Discovery and Data Mining conference*, 08 2018.

[26] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *European Conference on Computer Vision (ECCV)*, pages 740–755, 2014.

[27] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation-based anomaly detection. *ACM Trans. Knowl. Discov. Data*, 6(1):3:1–3:39, March 2012.

[28] Alex Lu, Amy Lu, Wiebke Schormann, Marzyeh Ghassemi, David Andrews, and Alan Moses. The Cells Out of Sample (COOS) Dataset and Benchmarks for Measuring Out-of-sample Generalization of Image Classifiers. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 1854–1862, 2019.

[29] Minghua Ma, Zheng Yin, Shenglin Zhang, Sheng Wang, Christopher Zheng, Xinhao Jiang, Hanwen Hu, Cheng Luo, Yilin Li, Nengjun Qiu, Feifei Li, Changcheng Chen, and Dan Pei. Diagnosing Root Causes of Intermittent Slow Queries in Large-Scale Cloud Databases. *Proceedings of the VLDB Endowment (PVLDB)*, 13(8):1176–1189, 2020.

[30] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. *ESANN*, pages 89–94, 04 2015.

[31] George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[32] Evan Racah, Christopher Beckham, Tegan Maharaj, Samira Ebrahimi Kahou, Mr. Prabhat, and Chris Pal. ExtremeWeather: A Large-Scale Climate Dataset for Semi-supervised Detection, Localization, and Understanding of Extreme Weather Events. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pages 3402–3413, 2017.

[33] Shebuti Rayana. Outlier Detection DataSets (ODDS) Library. `http://odds.cs.stonybrook.edu/`, 2016.

[34] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. pages 1135–1144, 2016.

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[36] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *Information Processing in Medical Imaging IPMI*, volume 10265, pages 146–157, 2017.

[37] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 815–823. IEEE Computer Society, 2015.

[38] Xingjian Shi, Zhihan Gao, Leonard Lausen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Deep Learning for Precipitation Nowcasting: A Benchmark and A New Model. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pages 5617–5627, 2017.

[39] N. Singh and C. Olinsky. Demystifying Numenta Anomaly Benchmark. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1570–1577, 2017.

[40] Corey Snyder and Minh Do. STREETS: A Novel Camera Network Dataset for Traffic Flow. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 10242–10253, 2019.

[41] Fei Song, Boyao Zhou, Quan Sun, Wang Sun, Shiwen Xia, and Yanlei Diao. Anomaly Detection and Explanation Discovery on Event Streams. In *International Workshop on Real-Time Business Intelligence and Analytics (BIRTE)*, pages 5:1–5:5, 2018.

[42] How are big companies using apache spark. `https://medium.com/@tao_66792/how-are-big-companies-using-apache-spark-413743dbbbae`.

[43] Nesime Tatbul, Tae Jun Lee, Stan Zdonik, Mejbah Alam, and Justin Gottschlich. Precision and Recall for Time Series. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 1924–1934, 2018.

[44] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 3266–3280, 2019.

[45] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *International Conference on Learning Representations (ICLR)*, 2019.

[46] Mike Wu, Michael C. Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. Beyond sparsity: Tree regularization of deep models for interpretability. *AAAI*, 2018.

[47] Jianxiong Xiao, Krista A. Ehinger, James Hays, Antonio Torralba, and Aude Oliva. SUN Database: Exploring a Large Collection of Scene Categories. *International Journal of Computer Vision*, 119(1):3–22, 2016.

[48] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, Jie Chen, Zhaogang Wang, and Honglin Qiao. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW*, pages 187–196, 2018.

[49] Jiawei Yang, Susanto Rahardja, and Pasi Fränti. Outlier detection: how to threshold outlier scores? In *Proceedings of the International Conference on Artificial Intelligence, Information Processing and Cloud Computing, AIIPCC*, pages 37:1–37:6, 2019.

[50] Haopeng Zhang, Yanlei Diao, and Alexandra Meliou. Exstream: Explaining anomalies in event stream monitoring. *EDBT*, pages 156–167, 2017.