Exathlon: A Benchmark for Explainable Anomaly Detection over Time Series [Experiments, Analysis & Benchmark]

Vincent Jacob[†] Fei Song[†] Arnaud Stiegler[†] Bijan Rad[†] Yanlei Diao[†] Nesime Tatbul[*]
[†]Ecole Polytechnique, France [*]Intel Labs and MIT, USA
{vincent.jacob,fei.song,arnaud.stiegler,bijan.rad,yanlei.diao}@polytechnique.edu
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 experimental research domains. While advanced analytics tasks over time series data have been gaining lots of attention, lack of such community resources severely limits scientific progress. In this paper, we present Exathlon, the first comprehensive public benchmark for explainable anomaly detection over high-dimensional time series data. Exathlon has been systematically constructed based on real data traces from repeated executions of large-scale stream processing jobs on an Apache Spark cluster. Some of these executions were intentionally disturbed by introducing instances of six different types of anomalous events (e.g., misbehaving inputs, resource contention, process failures). For each of the anomaly instances, ground truth labels for the root cause interval as well as those for the extended effect interval are provided, supporting the development and evaluation of a wide range of anomaly detection (AD) and explanation discovery (ED) tasks. We demonstrate the practical utility of Exathlon's dataset, evaluation methodology, and end-to-end data science pipeline design through an experimental study with three state-of-the-art AD and ED techniques.

1 INTRODUCTION

Time series is one of the most ubiquitious types of data in our increasingly digital and connected society. Advanced analytics capabilities such as detecting anomalies and explaining them are crucial in understanding and reacting to temporal phenomena captured by this rich data type. *Anomaly detection* (AD) refers to the task of identifying patterns in data that deviate from a given notion of normal behavior [11]. 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: 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) 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 [31, 39]. However, progress has been rather slow and limited. While there is extensive research activity going on [10], proposed solutions have been mostly adhoc and far from 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., [4, 17]). Inspired by those efforts, in this paper, we propose *Exathlon, the first comprehensive public benchmark for explainable anomaly detection over high-dimensional time series data.*

Exathlon focuses on the familiar domain of metric monitoring in large-scale computing systems, and provides a benchmarking platform that consists of: (i) a curated anomaly dataset, (ii) a novel benchmarking methodology for AD and ED, and (iii) an end-to-end data science pipeline for implementing and evaluating AD and ED algorithms based on the provided dataset and methodology. More specifically, we make the following contributions in this work:

Dataset. We constructed Exathlon systematically based on real data traces collected from around 100 repeated executions of 10 distributed streaming jobs on a Spark cluster over 2.5 months. Inspired by chaos engineering in industry [5], our traces were obtained by disturbing more than 30 job executions with nearly 100 instances of 6 different classes of anomalous events (e.g., misbehaving inputs, resource contention, process failures). For each of these anomalies, we provide ground truth labels for both the root cause interval and the corresponding effect interval, enabling the use of our dataset in a wide range of AD and ED tasks. Overall, both the normal (*undisturbed*) and 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. Exathlon evaluates AD and ED algorithms in terms of two orthogonal aspects: functionality and computational performance. For AD, we primarily target *semi-supervised techniques* (i.e., with a model developed/trained using only 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 [11]) over *high-dimensional time series* (i.e., multi-variate with 1000s of dimensions). This decision is informed by our observation of this being the most common and inclusive usage scenario in practice. For ED, we broadly consider both *model-free* (e.g., [3, 61]) and *model-dependent* (e.g., [45]) techniques. AD functionality is evaluated under four well-defined model learning settings, based on four key evaluation criteria – anomaly existence, range detection, early detection, and exactly-once detection – using a novel range-based accuracy framework [52]. Similarly, ED

functionality is tested for two capabilities – local explanation and global explanation – each measured in terms of conciseness, consistency, and accuracy. Computational efficiency and scalability for both AD model training/inference as well as for ED execution can also be evaluated at varying data dimensions and sampling rates. Overall, Exathlon provides a rich and challenging testbed with a well-organized evaluation methodology (Table 2).

Data Science Pipeline. We designed an end-to-end pipeline for explainable time series anomaly detection. This pipeline includes all the data processing steps necessary to turn our raw datasets into AD and ED results together with their benchmark scores. Our design is modular and extensible. This not only makes it easy to implement new AD and ED techniques to benchmark, but also allows creating multiple variants of pipeline steps to experiment with and compare. For example, training data preparation for different AD learning settings or scoring AD results for different criteria levels can be easily configured, run, and compared in our pipeline (Figure 3).

Experimental Study. We provide the first experimental study evaluating and comparing a representative set of state-of-the-art AD and ED techniques to illustrate the usage and benefits of Exathlon. Results suggest that our dataset carries useful signals that can be picked up by the tested AD and ED algorithms in a way that can be effectively quantified by our evaluation criteria and metrics. Furthermore, we observe that our benchmarking framework exposes increasing levels of challenges to stress-test these algorithms in a systematic way (§6).

Compared to current public resources for time series AD research [2, 16, 19, 34, 44], a key contribution of Exathlon is that it comprehensively covers one challenging application domain end to end, as opposed to providing multiple smaller and simpler datasets from several independent domains. Furthermore, a public benchmark for time series ED research with ground truth labels is largely lacking today, making it hard to evaluate and compare an increasing number of published papers on this important topic. Thus, we believe Exathlon provides an opportunity for a more in-depth investigation and evaluation of models and algorithms in both time series AD and ED, potentially revealing new insights for accelerating research progress in explainable anomaly detection.

In the rest of the paper, we first briefly summarize related work. After presenting our dataset, evaluation methodology, and full pipeline design in more detail, we demonstrate the practical utility of Exathlon through an experimental analysis of three state-of-the-art AD and ED algorithms using a selected set of evaluation criteria and settings from our benchmark. Finally, we conclude with an outline of future directions. The dataset, code, and documentation for Exathlon are publicly available at https://github.com/anomalybench/anomaly-bench.git.

2 RELATED WORK

Datasets and Benchmarks. Benchmarks to evaluate database (DB) system performance have been around for more than 30 years [14, 15, 27]. In addition to industry-standard benchmarks for relational DB workloads such as TPC-C and TPC-H, new domain-specific benchmarks for emerging workloads have been proposed (e.g., Linear Road Benchmark for stream processing [1], YCSB for scalable key-value stores [13], BigBench for big data analytics [25]). The main focus of these benchmarks has been on computational performance. With recent benchmarks for ML/DL-based advanced data analytics such as ADABench and DAWNBench, there has been a focus shift toward end-to-end ML pipelines and new evaluation metrics such as time to accuracy [12, 43]. Like in DB and systems communities, the ML community has also been publishing datasets and benchmarks to support research in many problem domains from object recognition to natural language processing [4, 17, 41, 54]. Well-known data archives for time series research include: UCI [19], UCR [16], and UEA [2]. These archives provide real-world data collections created for general ML tasks, such as classification and clustering. While the need for systematically constructing AD benchmarks from real data has also been recognized by others [22], public availability of anomaly datasets is still limited [44]. To our knowledge, Numenta Anomaly Benchmark (NAB) is the only public benchmark designed for time series AD [34]. NAB provides 50+ real and artificial datasets, primarily focusing on real-time AD 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 ED. NAB also has several technical weaknesses that hinder its use in practice (e.g., ambiguities in its scoring function, missing values in its datasets) [49].

Anomaly Detection (AD). There is a long history of research in AD [11, 28]. The high degree of diversity in data characteristics, anomaly types, and application domains has led to a plethora of AD approaches from simple statistical methods [7] to distance-based [53], density-based [9, 11], and isolation forests [37] to deep learning (DL) methods [10]. It is beyond the scope of this paper to provide a complete survey; we refer the reader to recent survey papers for a full discussion of such methods [10, 11, 28]. In our experimental study, we particularly focus on three DL methods that represent the recent state of the art [40, 48, 58] (detailed in §6). Such DL methods have the potential to handle a variety of anomaly patterns, such as contextual and collective patterns [11], and overcome known limitations of previous density- and distance-based methods that are very sensitive to data dimensions.

Interpretable Machine Learning. Interpretable ML has recently attracted a lot of attention [42]. Relevant techniques generally belong to two broad families: *interpretable models* and *model-agnostic methods*. Interpretable models are those that directly build a human-readable model from the data (e.g., linear or logistic regression, decision trees or rules) [42]. In contrast, model-agnostic methods separate explanations from the ML model, hence offering the flexibility to mix and match ML models with interpretation methods. In the model-agnostic family, several methods obtain interpretable classifiers by perturbing the inputs and observing the response [33, 45, 46, 51]. LIME [45] explains a prediction of any classifier by approximating it locally with a visually interpretable, sparse linear model, and explains the overall model by selecting a set of representative instances with explanations. As such, it is generally considered a method for *local explanations*. We evaluate LIME in our experimental study. Anchors [46] improved upon LIME by replacing a linear model with a logical rule that explains a single data instance and offers better coverage of data points in the local neighborhood, but it does not support time series data like in our work. SHAP scores [38], RESP scores [6], axiomatic attribution [51] are also instance-level explanations that assign a numerical score to each feature, representing their importance in

the outcome. In contrast to local explanations, other work aims to explain a model via *global explanations*. Some of them approximate a DL model using a decision tree [24, 57], or by learning a decision set [32, 33] directly as explainable models. All of these methods suffer from lacking a benchmark dataset and evaluation methodology. The FICO challenge was designed to evaluate such methods using a home loan application dataset, with a known label (high or low risk) for each application, but it relies on manual evaluation of returned explanations by real-world data scientists [23]. As a result, it remains hard to compare different ED methods due to the lack of ground truth explanations and automated evaluation procedures.

Explaining Outliers in Data Streams. There is a handful of work in explaining outliers in data streams. Given normal and abnormal time periods by the user, EXstream finds explanations to best distinguish the abnormal periods from the normal ones [61]. MacroBase helps the user prioritize attention over data streams, with modules for both AD and ED tasks [3]. Its AD module uses simple statistical methods like MAD, which is known to be suitable only for detecting simple point outliers [11]. For a detected anomaly, MacroBase's ED module discovers an explanation in the form of conjunctive predicates, by using a frequent itemset mining framework that takes minimum support and risk ratio as input parameters. We evaluate both of these techniques in our experimental study.

Explaining Outliers in SQL Query Results. Scorpion explains outliers in group-by aggregate queries by searching through various subsets of the tuples that were used to compute the query answers [56]. Given a set of explanation templates by the user, Roy et al.'s approach performs precomputation in a given DB to enable interactive explanation discovery [47]. Similarly, given a table, El Gebaly et al.'s work constructs an explanation table and finds patterns affecting a binary value of each tuple [21]. These approaches target traditional DB workloads and are not applicable to our problem. There have been some recent industrial efforts on time series anomaly explanation and root cause analysis in DB systems [31, 39]. These approaches require a variety of inputs from the user, e.g., casual hypotheses [31], or labels of root causes [39], whereas our work focuses on semi-supervised learning for explainable AD. Moreover, these systems are largely based on proprietary code and datasets that are not accessible to the research community.

3 DATASET

The Exathlon 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: Spark Application Monitoring

Large-scale big data analytics applications are being deployed on Apache Spark clusters everyday. Monitoring the execution of these jobs to ensure their correct and timely completion via AD can be business-critical. For example, some of the largest e-commerce platforms run Spark jobs on petabytes of data each day to analyze purchase patterns, target offers, and enhance customer experiences [50]. Since results of these jobs affect immediate business decisions such as inventory management and sales strategies, they are often specified with deadlines. Anomalies that occur in job execution would prevent analytical jobs from meeting their deadlines and hence cause disruption to critical operations on those e-commerce platforms. We model this widespread and challenging AD use case in our benchmark.

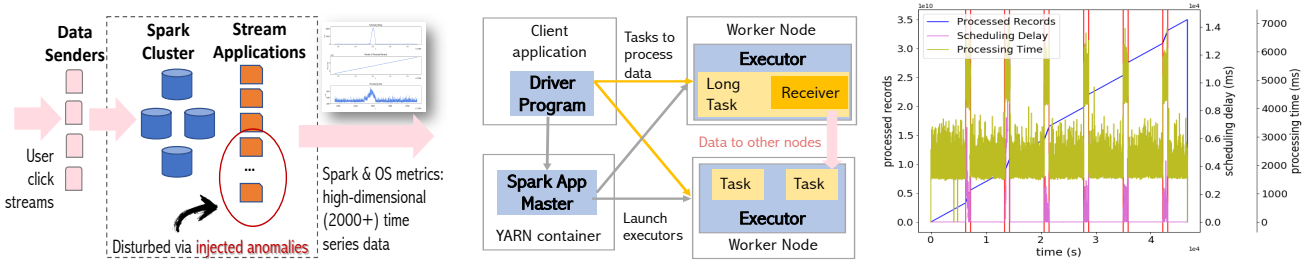
System Setup. Our Spark workload consists of 10 stream processing applications that analyze user click streams from the World-Cup 1998 website [35] (replicated with a scale factor for our long-running applications). As in Figure 1(a), *Data Sender* servers ingest the streams at a controlled *input rate* to a Spark cluster of 4 nodes, each with 2 Intel® Xeon® Gold 6130 16-core processors, 768GB of memory, and 64TB disk. Each 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. 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. Furthermore, given 32 cores, each node can run tasks from multiple applications concurrently. As common real-world practice, we run 5/10 randomly selected applications at a time. The placement of Driver and Executor processes to cluster nodes for these 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.

Trace Collection. We ran the 10 Spark streaming applications in our 4-node cluster over a 2.5-month period. The data collected from each run of a Spark streaming application is called a *Trace*. Some of the traces were manually pruned, because they were affected by cluster downtimes or the injected anomalies were not well reflected in the data due to failed attempts. After the manual pruning, we kept 93 traces to constitute the Exathlon dataset.

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). Table 1(a) gives a summary of the metrics collected per trace. The Driver offers 243 Spark UI metrics covering 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.

3.2 Undisturbed vs. Disturbed Traces

In generating our 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 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.



(a) Spark application monitoring (b) Spark execution environment (c) Bursty input

Figure 1: Spark application monitoring, and metrics observed in anomaly instances (a pair of red vertical bars marks a root cause event)

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 (RCI + EEI) min, avg, max	Data Items
Undisturbed	N/A	59	N/A	N/A	1.4M
Disturbed	T1: Bursty input	6	29	15m, 22m, 33m	360K
Disturbed	T2: Bursty input until crash	7	7	8m, 35m, 1.5h	31K
Disturbed	T3: Stalled input	4	16	14m, 16m, 16m	187K
Disturbed	T4: CPU contention	6	26	8m, 15m, 27m	181K
Disturbed	T5: Driver failure	11	9	1m, 1m, 1m	128K
Disturbed	T6: Executor failure		10	2m, 23m, 2.8h	

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

(b) Undisturbed traces, disturbed traces, and ground truth labels of 97 anomalies

Table 1: The Exathlon dataset

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 normal 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 6 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; please see Appendix B for further details.

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 1(c) 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.

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.

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.

Driver Failure (Type 5) and Executor Failure (Type 6): Hardware faults or maintenance operations may cause a node to fail all of a sudden, making all processes (drivers and/or executors) located on that node unreachable. Such processes must be restarted on another node, which causes delays. We created such 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. We also created anomalies by failing executor processes, which get restarted 10 seconds after the failure, but its effects on metrics such as processing delay may continue longer. We created 9 driver failures and 10 executor failures over 11 different traces.

The Ground Truth Table. For all of these 97 anomaly instances over 34 anomalous traces, we provide ground truth labels in the format shown in Table 1(b). Such labels include both *root cause intervals* (RCIs) and their respective *extended effect intervals* (EEIs). RCIs typically correspond to the time period during which DEG programs are running, whereas the EEIs are the time periods that start immediately after an RCI and end when important system metrics return to normal values or the application is eventually pushed to crash. The EEIs are manually determined using domain knowledge. Please see Appendix B for further details.

	Anomaly Detection (AD) Functionality	Explanation Discovery (ED) Functionality	Computational Performance
Evaluation Criteria	AD1: Anomaly Existence AD2: Range Detection AD3: Early Detection AD4: Exactly-Once Detection	ED1: Local Explanation ED2: Global Explanation	P1: AD Training Scalability P2: AD Inference Efficiency P3: ED Efficiency
Evaluation Metrics	Accuracy: *Range-based Precision, Recall, F-Score, AUPRC*	Conciseness Consistency: *Stability* (ED1), *Concordance* (ED2) Accuracy: *Point-based Precision, Recall, F-Score*	Time
Settings & Parameters	LS1: *1-App Many-Examples*, LS2: *N-App Many-Examples*, LS3: *1-App Few-Examples*, LS4: *N-App Few-Examples*		Dimensionality Cardinality Factor

Table 2: The Exathlon evaluation methodology and benchmark design

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, Exathlon is designed to evaluate AD and ED algorithms in two orthogonal aspects: functionality and computational performance. For each of these, the benchmark provides multiple evaluation criteria measured via the corresponding metrics, under a well-defined variety of experimental settings and parameters. This enables systematic analysis of both AD and ED capabilities. In terms of functionality, the evaluation criteria capture that an AD/ED algorithm is exposed to increasingly more challenging requirements as the functionality level is raised from one to the next. In terms of computational performance, Exathlon provides three complementary criteria that can be evaluated by varying dimensionality and size of the dataset. We present more details in the rest of the section.

4.1 Anomaly Detection (AD) Functionality

First and foremost, we designed Exathlon targeting *semi-supervised 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.

Evaluation Criteria. We identified four key criteria for evaluating AD functionality, as listed from basic towards advanced:

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

AD2 (*Range Detection*): The next expectation is to report not only the existence, also the precise time range of an anomaly. The wider a range of an anomaly that an AD method can detect, the better its understanding of the underlying real-world phenomena.

AD3 (*Early Detection*): The third expectation is to minimize the *detection latency*, i.e., the difference between the time an anomaly is first flagged and the start time of the corresponding RCI.

AD4 (*Exactly-Once Detection*): The last 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.

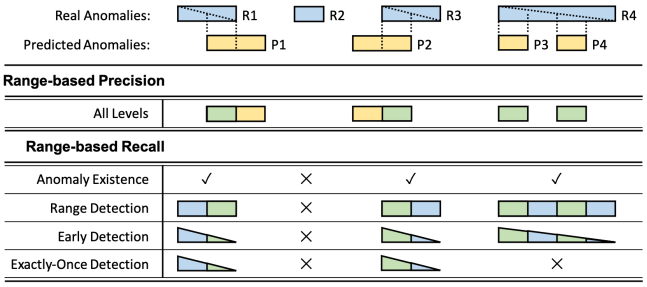


Figure 2: Range-based precision and recall at AD levels 1-4. Precision evaluates prediction quality (green out of yellow for each Pi). Recall evaluates anomaly coverage (green out of blue for each Ri).

Evaluation Metrics. To assess how well an AD algorithm can meet these four functionality levels, we use the customizable accuracy evaluation framework for time series [52]. This framework extends the classical precision/recall from point-based data to range-based data, by introducing a set of tunable parameters. By setting the values of these parameters in a particular way and applying the resulting precision/recall formulas to the output of an AD algorithm, one can assess how well that output measures up to the quality expectations represented by those parameter settings. We leverage this as a mathematical tool to quantify how well an AD algorithm meets AD1-AD4. Furthermore, we do this in a way that every level AD_i builds on and adds to the requirements of the previous level AD_{i-1}. This monotonic design ensures that the AD functionality score that an algorithm gets (Precision, Recall, or other metrics obtained by combining them, e.g., F-Score or Area Under the Precision/Recall Curve (AUPRC)) is always ordered as: $score(AD1) \geq score(AD2) \geq score(AD3) \geq score(AD4)$, which facilitates evaluating and interpreting results in a systematic way.

Figure 2 provides a simple example to illustrate how range-based precision and recall are computed for different AD levels. Given real anomaly ranges R1..R4 and predicted anomaly ranges P1..P4 produced by an AD algorithm, we first compute Precision/Recall for each range and then average them for overall Precision/Recall. Intuitively, Precision focuses on the size of TP ranges (colored green) relative to TP+FP ranges (colored yellow), and Recall focuses on the size of TP ranges relative to TP+FN ranges (colored blue). For AD1, Recall(Ri) is 1 if Ri is flagged, 0 otherwise. For AD2, Recall(Ri) is proportional to the relative size of the TP range. For AD3, Recall(Ri) is further weighted by position of the TP range relative to the start of Ri. Finally, at AD4, Recall(Ri) degrades to 0 for Ri that is not flagged exactly once. Precision(Pi) is computed in an analogous

way, except that AD levels about anomaly coverage quality (AD1 and AD3) are not relevant to it; rather the main focus is on the size and number of the real anomaly ranges that are successfully predicted. In our simple example, it turns out that all AD levels for Precision consider the same Pi subranges.

To achieve precision and recall at different AD levels, we set the tunable parameters of the range-based precision/recall framework with necessary modifications. The details are deferred to Appendix C due to space constraints. Further note that both the semi-supervised AD algorithms we investigate and the precision/recall for time series model we use to assess them focus on binary classification (normal vs. anomalous ranges). On the other hand, our dataset is inherently a multi-class one (normal ranges vs. six types of anomalous ranges). This raises a question about how to evaluate binary predictions under multi-class labels. We take a holistic approach, and evaluate the AD prediction results both globally and grouped by type, whenever this is reasonable and provides useful insights. For example, even though a binary predictor is not able to detect different anomaly types, we can still measure its resulting coverage (i.e., Recall) for each type. However, type-wise measurement is not entirely meaningful for Precision, since false positives (FPs) are essentially typeless.

Learning Settings. By default, our benchmark offers only undisturbed traces as training data for building an AD model, because it is most practical for real-world use. It can be seen as a *"noisy" semi-supervised anomaly detection* problem [11], meaning that the training data covers mostly the normal data but is subject to small amounts of noise. In this context, Exathlon also tests how well a learned AD model generalizes to different workload characteristics, including the Spark application (A), input rate (R), and concurrency (C) characteristics. We offer four alternative learning settings, considering the following two issues:

Modeling Subject (1-App vs. N-App Learning): Spark applications differ in workload characteristics (e.g., CPU intensive versus 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). Hence learning anomalies across multiple applications' traces requires more generalization power. (i) *1-App learning* focuses on training and evaluating an AD model on a single application basis. As such, there is no need for the model to generalize to other applications. (ii) *N-App learning* trains an AD model across multiple applications together, and hence needs to learn how to characterize different applications in the model.

Training Constraints (Many vs. Few Examples): Each of our traces contains data from a random sample of 5 out of 10 applications running at different input rates. An AD model trained with normal traces with certain (A, R, C) settings may later be subject to a new trace with a previously unseen (A, R, C) setting. A well-learned model is supposed to generalize across these differing workload settings. However, such generalization power is easier to achieve if the training data includes many examples of (A, R, C), or many (R, C) examples in the 1-App Learning setting. By default, Exathlon reserves the disturbed traces entirely for testing. As such, the training data includes only the 59 undisturbed traces, which is unlikely to cover most (A, R, C) values, hence indicating the *Few Example*-training scenario. This bears similarity with the few-shot learning problem [55] recently studied in the ML community, and poses a great challenge for learning. Exathlon also offers a simpler, yet less realistic, training setting, *Many Example*-training, if some algorithms cannot learn from limited examples. Here, we allow the AD algorithm to include in training also an earlier segment (with normal data only) of each disturbed trace, while testing on the anomalies from a later segment of the same trace. This way, the model is given a chance to "peek" at the normal state (including all workload characteristics) of a particular test trace. This simulates the scenario that there are many training examples of (A, R, C), some of which bear similarity with the current test trace.

By combining the above options, we obtain four learning settings to experiment with: *LS1: 1-App, Many-Examples*; *LS2: N-App, Many-Examples*; *LS3: 1-App, Few-Examples*; and *LS4: N-App, Few-Examples*.

4.2 Explanation Discovery (ED) Functionality

Once an anomalous instance is flagged by an AD method, the next desirable functionality is to find the best explanation for the anomaly detected, or more precisely, a human-readable formula offering useful information about what has led to the anomaly.

There have been many explanation discovery (ED) methods in recent work (see §2). These methods differ in the form of "explanation" provided: some return a logical formula as an explanation [3, 46, 61], others return a decision tree [57], and some others return a numerical score for each feature such as the coefficient in linear regression [45] or the SHAP score [38] (see Figure 6 for examples). Exathlon does not pose any restrictions on the form of explanation used. Instead, it takes an abstract view of explanations. Formally, we model each trace in the test dataset as a multi-dimensional time series, $[\ \boldsymbol{x}_1 \dots\ \boldsymbol{x}_t \dots\ \boldsymbol{x}_n]^T$, where each data item includes m features, $\boldsymbol{x}_t = (x_{t1}, \dots, x_{tm})$. A detected anomaly is a subsequence of the time series that starts at timestamp t and has duration w, $X_{t,w} = [\ \boldsymbol{x}_t \dots\ \boldsymbol{x}_{t+w}]^T$. If an AD method can provide only point-based detection, the duration w is set to 0. We denote the explanation generated for the anomaly $X_{t,w}$ as $F_{t,w}$ and treat it as a function of the features, $\mathbf{A} = (a_1, \dots, a_m)$, from the data:

$$F_{t,w}(a_1, \dots, a_m) \models X_{t,w}$$

where $\models$ means that $F_{t,w}$ "explains" the anomaly $X_{t,w}$. In addition, we define an extraction function, $G_{\mathbf{A}}$ over $F_{t,w}$, that returns the set of features used in the explanation (e.g., appearing in a logical formula or having non-zero coefficients in a regression model).

$$G_{\mathbf{A}}\left(F_{t,w}(a_1, \dots, a_m)\right) = \mathbf{A}_{t,w} \subseteq \mathbf{A}$$

Finally, we define the size of $F_{t,w}$ as the size of its feature set $\mathbf{A}_{t,w}$.

$$|F_{t,w}(a_1, \dots, a_m)| = |\mathbf{A}_{t,w}|$$

Evaluation Criteria: Subject of Explanation. The key distinction that Exathlon makes is whether an ED method is attempting to explain a single anomaly (*local*) or a broad set of anomalies (*global*).

ED1: *Local Explanation*: This corresponds to explaining one anomaly instance, offering a compact yet meaningful piece of information to help the user understand this particular instance. As mentioned by LIME [45], the explanation should be locally faithful. In our context, it means that the same explanation can hold over immediate "neighbors", which are anomaly instances of the same application and same anomalous type, and around the same time period.

ED2: *Global Explanation*: Alternatively, an ED method may attempt to explain a (potentially large) set of anomalies, called a global explanation. In general, it is not possible to find an identical succinct explanation for many different instances. Hence, a global explanation usually is composed of a set of explanations; e.g., LIME [45] chooses the most representative k instances to explain a model. In our benchmark, it makes most sense to construct a global model for a set of anomalies of the same type, but potentially from different applications or different runs of the same application. This helps us understand for "semantically similar" anomalies, whether an ED method can return explanations that are consistent, or even of predictive power of similar anomalies that arise in the future.

Evaluation Metrics. Exathlon evaluates both local and global explanations for three desired properties:

1. *Conciseness*: This corresponds to the number of features used in the explanation. Following the Occam's razor principle, humans favor smaller, and thus simpler explanations. As different ED methods return explanations of different forms, our benchmark counts the number of features used in the explanation as its conciseness measure. In the ED1 case, that is $|F_{t,w}|$ defined above. In the ED2 case, a global explanation includes a set of explanations, and its conciseness measure is the average of the size of each explanation.

2. *Consistency*: Anomalies of the same type occurring in a similar context should have consistent explanations. We customize this notion for ED1 and ED2, respectively. In both cases, we care only about the set of features employed in the explanation, without considering the numerical or categorical values used.

Stability (ED1) is the customized consistency measure for ED1. It means that the anomalies occurring in a similar context (e.g., for the same application, same run, and same time period) should have similar explanations, subject to small perturbation of the data.

Formally, we introduce a subsampling procedure over an anomaly $X_{t,w}$, which generates a set of samples, $\{X_{t,w}^{(i)}\}$. We denote the corresponding explanations generated for them as $\{F_{t,w}^{(i)}\}$. The extraction function on a set of explanations is defined to be the duplicate-preserving union (like Union All in SQL) of the extraction function of each respective explanation:

$$G_{\mathbf{A}}\left(\{F_{t,w}^{(i)}(a_1,\ldots,a_m)\}\right) = \bigcup_i G_{\mathbf{A}}\left(F_{t,w}^{(i)}\right) = \bigcup_i \mathbf{A}_{t,w}^{(i)} = \mathbf{A}_{t,w}^{\cup}$$

Finally, for each feature $a_j \in \mathbf{A}_{t,w}^{\cup}$, we count its frequency in this feature set and normalize it by the total size of the feature set. The consistency measure is computed by the entropy of the set of normalized frequencies of such features, a_j, $j = 1, 2, \ldots$:

$$H\left(A_{t,w}^{\cup}\right) = -\sum_i p(a_j)\log_2 p(a_j), \quad a_j \in \mathbf{A}_{t,w}^{\cup}$$

$$p(a_j) = \mathbb{1}_{\mathbf{A}_{t,w}^{\cup}}(a_j) \;/\; \left|\mathbf{A}_{t,w}^{\cup}\right|$$

Here $\mathbb{1}_{\mathbf{A}_{t,w}^{\cup}}(a_j)$ is an indictor function that counts the occurrences of a feature in a multi-set. For capturing consistency, our choice of entropy is motivated by information theory that a set of explanations that lack consistency will require using more bits to encode, hence a larger entropy value. In the ideal case, all explanations, $\{F_{t,w}^{(i)}\}$, are identical, and its entropy takes the minimum value 0 if the size of the explanation is 1 (denoted as H_1), the value 1 if the size is 2 (H_2), or the value 1.58 if the size is 3 (H_3). We consider an entropy value within $H_3 = 1.58$ a good score for consistency.

Concordance (ED2) is the customized consistency measure for ED2. Here, it means that the anomalies of the same type are expected to have consistent explanations, subject to larger amounts of deviation in data due to different time periods in the same run of a Spark application, different runs of the application, or even different Spark applications. Formally, we are given a set of anomalies, $\{X_{t_i,w_i}\}$. Denote their corresponding explanations as, $\{F_{t_i,w_i}\}$. The consistency measure of this set of explanations is computed similarly to that of ED1, except that we are replacing the subsampled anomalies, $\{X_{t,w}^{(i)}\}$, with the given set of anomalies, $\{X_{t_i,w_i}\}$.

3. *Accuracy*: The last property, which is also the hardest to achieve, is to view an explanation of an anomaly as a predictive model, apply it to other similar instances (defined above for ED1 and ED2, respectively), and them evaluate accuracy of such predictions.

Note that not all explanations can serve as a predictive model. Only those that are a function mapping a given data item to 0/1, $F_{t,w}: \boldsymbol{x}_t \in \mathbb{R}^m \rightarrow \{0,1\}$, can offer predictive power over test data. For example, a logical formula [3, 46, 61] or a decision tree [57] can be used to run prediction on new data items, but feature importance scores or SHAP scores [38] cannot. Even with those ED methods that return a predictive explanation, it is only a point-based predictive model. The literature largely lacks ED methods that can return explanations that characterize a temporal pattern. For this reason, Exathlon evaluates the accuracy of such explanations using point-based precision recall. Should new ED methods arise with explanations covering a temporal pattern, we will add range-based precision recall as accuracy metrics, like in the AD evaluation.

In the case of ED1, we are given a particular anomaly $X_{t,w}$. To measure the accuracy of an ED method, we subsample from $X_{t,w}$, yielding a sample, $X_{t,w}^{(i)}$. We run the ED method to generate an explanation, $F_{t,w}^{(i)}$. Then we run $F_{t,w}^{(i)}$ as a predictive model over a test dataset that includes the remainder of the anomalous data, $X_{t,w} - X_{t,w}^{(i)}$, as well as some normal data that immediately proceeds or follows $X_{t,w}$. For each test point, we obtain a 0/1 prediction and compare to the ground truth. We repeat this procedure for all test points to compute the final precision, recall, and F-score.

In ED2, we are given a set of anomalies X_{t_i,w_i}. We randomly split the set into a training set and a test set. We can run a suitable ED method to generate a global explanation from the training set, and then use it as a predictive model over the test set. For each anomaly in the test set, we compare the point-wise prediction against the ground truth and compute the precision, recall, F-score, similar to ED1. Note, however, that most ED methods hard-code constants in the explanation, which hardly generalizes to other anomalies in a different context, e.g., a different Spark application or input rate.

4.3 Computational Performance

Exathlon can also be used to evaluate computational performance.

Evaluation Criteria and Metrics. ML algorithm performance is typically measured in terms of the total time it takes for model training as well as for using that model for making predictions. For AD, we define P1 and P2 to evaluate training and inference performance, respectively. For ED, the time to discover each explanation, P3, is our third performance metric.

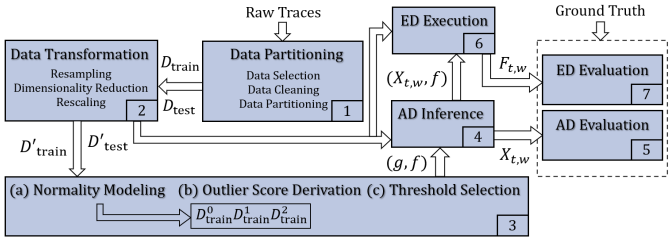


Figure 3: A pipeline for explainable AD on multivariate time series

Experimental Parameters. Exathlon offers scalability tests by varying the following two data-related parameters:
Dimensionality M: Our dataset consists of high-dimensional time series data. The 2,283 metrics (features) may be correlated and contain a lot of null values, which are representative of real-world datasets. The benchmark leaves it to each user algorithm as how it copes with the high dimensionality. The relevant techniques may include dimensionality reduction using linear transformation (e.g., PCA), or feature selection by leveraging the correlation structure in the data. Such choices are left to the discretion of each user algorithm, and Exathlon reports on the resulting dimensionality M used in AD and ED tasks.
Cardinality Factor α: Besides high dimensionality, our dataset also has high cardinality, $N = 2{,}335{,}781$ data items, which is significantly higher than the existing Numenta Anomaly Benchmark [34]. If the training time of an algorithm is too long, a user algorithm can choose to reduce the cardinality via resampling, i.e., by taking average of the data items in each l-second interval, which amounts to a cardinality factor $\alpha = 1/l$ and reduced data size of αN.

5 A FULL PIPELINE FOR EXPLAINABLE AD

Besides a curated dataset and an evaluation methodology, Exathlon also provides a full pipeline for explainable AD on high-dimensional time series data. Our pipeline is characterized as follows: (i) It consists of the typical steps in a deep ML pipeline, ranging from data partitioning, feature engineering, dimensionality reduction, to AD and ED. (ii) It implements a variety of AD and ED functionalities and evaluation modules that score them based on the metrics of the benchmark (see §4). (iii) It provides an open, modular architecture that allows different methods to be added and combined through the pipeline. Figure 3 provides an overview.

1. Data Partitioning. This initial phase takes as input the 93 raw traces, as described in §3. It first performs simple data cleaning, e.g., replacing missing data with a default value. It then performs data selection and partitioning of the 93 traces according to the learning settings described in §4: (i) In the *1-App* learning settings (LS1 and LS3), we consider each Spark application, i, separately. The data selection step collects all the traces related to i. The data partitioning step then takes all undisturbed traces of i into training data, D_i , and all disturbed traces of i into test data, $\tilde{D}_i$. These two datasets will be used to run the full pipeline to learn the model for i and conduct the final evaluation. (ii) In the *N-App* learning settings (LS2 and LS4), the data selection step collects all 93 traces. The data partitioning step then takes all undistributed traces as training data, D, and all disturbed traces as test data, $\tilde{D}$. (iii) The above implementations correspond to the *Few-Examples* learning settings (LS3 and LS4). If the pipeline is configured to run under the *Many-Examples* settings (LS2 and LS4), we further augment the training data obtained above, for *1-App* and *N-App* learning, respectively, with an earlier segment of each related test trace (mostly with normal data). The output of this phase is a pair of datasets, denoted as $(D_{\text{train}}, D_{\text{test}})$.

2. Data Transformation. As ML algorithms require data transformations to perform well, our pipeline offers the following steps:
(i) *Resampling* (optional): For the multi-variate time series in each trace, the user can choose to resample, by taking the average of data points in each l-second interval. This step reduces the *cardinality factor*, $\alpha = 1/l$, of the time series data, if the training time turns out to be too long for some ML algorithms.
(ii) *Dimensionality reduction*: Since our dataset includes $M = 2{,}283$ raw features, such high dimensionality may affect both model accuracy, known as the "curse of dimensionality", and training time. To reduce dimensionality, our pipeline offers a PCA-based method (with a parameter that controls different coverage of the data variance and the resulting feature set size), as well as a manually curated feature set with 19 features selected using domain knowledge.
(iii) *Rescaling*: Most ML algorithms require the features to be scaled into a range, e.g., between [0, 1], to better align features whose raw values may differ by orders of magnitude. A unique issue in our problem is that each test trace may represent a new context, e.g., a combination of input rate and concurrency not seen in training data. As a result, rescaling has to take into account this new context. Therefore, we provide a customized scaling method that rescales test data dynamically as we run an AD model over the data.

3. AD Modeling. The next phase takes the transformed training data and builds an AD model. Most AD methods build a model that describes the normal behavior in the data, called a "normality model", such that any future (test) data that deviates significantly from it will be flagged as an anomaly. Our pipeline offers an open architecture to embrace any AD method that builds such a normality model to detect anomalies. In this paper, we focus on recent DL-based AD methods [40, 48, 58], to explore their potential for handling the complexity of our dataset (high-dimensional, with noise), anomaly patterns (a variety of contextual and collective anomalies), and learning settings (noisy semi-supervised AD modeling).
(i) *Normality modeling*: The first step is to train a normality model based on a DL method of choice. Most DL methods take input data of fixed window size s. Given each of our traces, we create sliding windows of size s, with the slider parameter 1, and feed them as input to the model. Different DL methods model the data in the window by either trying to forecast the data point following the window (forecasting-based, e.g., LSTM [8]) or reconstructing the window via a succinct internal representation (reconstruction based, e.g., Autoencoder [29, 58] or GANs [48]). To train each specific model, we partition D_{train} into internal training (D^0_{train}), validation (D^1_{train}), and test (D^2_{train}) sets. The DL model is trained on D^0_{train}, with early stopping applied based on the model performance on D^1_{train}. Hyperparameter tuning is performed by choosing a configuration that maximizes model performance on D^2_{train}.
(ii) *Outlier score derivation*: We next build an initial AD model, $g : \boldsymbol{x} \in \mathbb{R}^m \rightarrow \mathbb{R}$, which maps each data point to an outlier score. For forecasting models, we compute the difference, d, between

the forecast and true values of each data point, and derive the outlier score v based on d; the higher the d value, the higher the v score. For reconstruction-based models, we treat the reconstruction error of each window as the v score for that window, and then derive the v score of each data point by averaging the scores of its enclosed sliding windows. Our pipeline implements the LSTM [8]; Autoencoder (AE) [29]; and BiGAN [48] for AD. Details of these models and the necessary modifications we made to suit our dataset are deferred to Appendix E.2 due to space constraints.

(iii) *Threshold selection*: The last step aims to find a threshold on the outlier score to return a 0/1 prediction. It returns a final AD model, $f : \boldsymbol{x} \in \mathbb{R}^m \rightarrow \{0, 1\}$, mapping each data point to 0/1. Exathlon does not offer labeled data for threshold selection. Hence, we provide unsupervised threshold selection fit on D^2_{train}. Among the methods listed in a recent survey [59], we choose three most used automatic techniques: SD, MAD, and IQR, with the possibility of repeating them multiple times to filter large outlier scores.

4. AD Inference. Once the AD model is built, the next phase of the pipeline runs the AD model over each test trace to detect anomalies. In the context of range based AD, predicted anomalies for a test trace are defined as sequences of positive predictions within that trace, denoted as $X_{t,w}$, which starts at t and has duration w.

5. AD Evaluation. The last AD phase evaluates the AD model for a given set of requirements. We evaluate both a model's ability to separate normal from anomalous data in the outlier score space and its final AD ability based on threshold selection. The separation ability (g) is assessed at the trace, application, and global levels. Global separation is reported as the AUPRC computed on all test data, while the application/trace-level separation is reported by computing an AUPRC for each application/trace, and averaging the results. The detection ability (f) is assessed by reporting its range-based precision, recall, and F-score, with parameters specified by the AD functionality. Recall is also reported by anomaly type.

6. ED Execution. For each test trace, AD inference reports a set of anomalies, and for each reported anomaly, the ED module returns an explanation for it. Our pipeline supports two families of ED methods. (i) *Model-free ED methods* do not require the access to an ML model. Instead, they only require the anomalous instance, $X_{t,w}$, and a reference dataset, to generate an explanation. Examples include EXstream [61] and MacroBase [3] (§2). Our implementation sets the reference dataset as the subset of data that immediately proceeds the detected anomaly, denoted by $X_{t,-w'}$, and was classified as normal. Then the pair of datasets, $(X_{t,w}, X_{t,-w'})$, are provided to the ED method to generate an explanation, $F_{t,w}$. (ii) *Model-dependent ED methods* take not only the anomalous instance, $X_{t,w}$, but also an AD model, $f : \boldsymbol{x} \in \mathbb{R}^m \rightarrow \{0, 1\}$. Examples include LIME [45], Anchors [46], and SHAP [38] (§2). In our implementation, we provide the AD model used in inference to the ED method.

7. ED Evaluation. After processing each test trace, we obtain a set of anomalies with their corresponding explanations. We then collect the explanations from all the test traces to run the final ED evaluation and compute conciseness, consistency, accuracy, and time metrics. Further details are given in Appendix D.

Sep Lvl	Method	Ave	AUPRC for Anomaly Types T1→T6					
Trace	**LSTM**	0.60	0.69	0.81	0.43	0.45	0.77	0.44
	AE	**0.73**	0.83	0.81	0.64	0.76	0.89	0.44
	BiGAN	0.61	0.91	0.76	0.15	0.70	0.64	0.51
App	**LSTM**	0.47	0.57	0.37	0.56	0.38	0.60	0.35
	AE	**0.57**	0.65	0.40	0.63	0.55	0.79	0.43
	BiGAN	0.52	0.81	0.36	0.25	0.54	0.69	0.48
Global	**LSTM**	0.41	0.56	0.32	0.53	0.25	0.53	0.27
	AE	**0.50**	0.60	0.36	0.54	0.47	0.68	0.37
	BiGAN	0.49	0.68	0.32	0.39	0.52	0.65	0.39

Table 3: Separation abilities of AD methods (LS4, $\text{FS}_{\text{custom}}$, AD2)

6 EXPERIMENTAL STUDY

In this section, we apply our benchmark to a select set of AD and ED methods. While a complete comparison of all related AD and ED methods is beyond the scope of this paper, analyzing the select methods allows us to demonstrate the value of our dataset and benchmark. Our analyses include the strengths and limitations of these AD and ED methods, challenges posed by our dataset and evaluation criteria, and some potential directions of future research.

6.1 Experimental Setup

In our experimental setup, we integrated into our pipeline three DL-based AD methods: LSTM [8], AE [29], and BiGAN [48]. We also integrated three recent ED methods: EXstream [61] and MacroBase [3] from the DB community for outlier explanation in data streams, and LIME [45], an influential method from the ML community. See Appendices E.2 and E.3 for details of these methods.

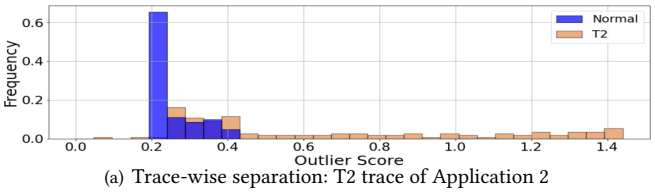
Besides the methods, our pipeline also needs to be configured with the following options: (a) **Learning Setting (LS1-4)**, as described in Table 2, with a default setting of LS4: N-App Few-Examples as it is the most realistic setting. (b) **Data Size and Feature Set (FS)**: Since some of the DL models (e.g., GANs) could not complete training on our cluster using the full dataset, we reduce the data size by setting the cardinality factor, $\alpha = 1/15$. We also use a reduced feature set of 19 features, $M = 19$, produced by either manual selection based on domain knowledge, denoted as $\text{FS}_{\text{custom}}$, or by PCA with the same number (19) of features, denoted as FS_{pca}. The choices of (a) and (b) determine the training data in a particular experiment. For each given training set, we allow each DL algorithm to train for 1.5 days (including hyperparameter tuning) to obtain an AD model. (c) **Level of AD Evaluation (AD1-4)**, as described in Table 2, with a default setting of AD2 (range detection).

6.2 AD Evaluation Results and Discussion

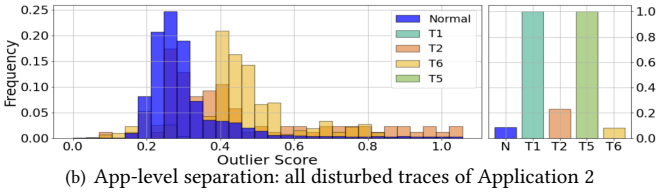
We begin by applying the LSTM [8], AE [29], and BiGAN [48] methods on our benchmark dataset and report on the AD metrics.

Experiment 1 (LS4, $\text{FS}_{\text{custom}}$, AD2). The first experiment presents a comparison of the three AD methods under a default setting, (LS4, $\text{FS}_{\text{custom}}$, AD2). Here, we focus on the model's ability to separate anomalous data from normal data, via the analysis of trace-level, application-level, and global AUPRC results summarized in Table 3.

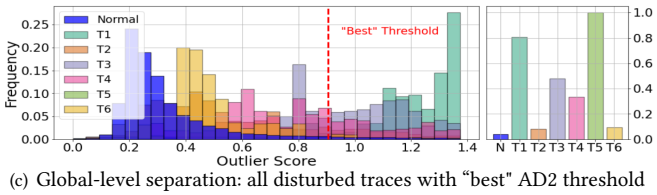
(1) *Separation Performance:* We first consider trace-level separation. All three methods achieved decent AUPRC scores for most (or a subset) of anomaly types, with AE achieving the highest score of 0.73. Figure 4(a) shows the distribution of outlier scores assigned by the AE method to the records in the T2 trace of Application 2.



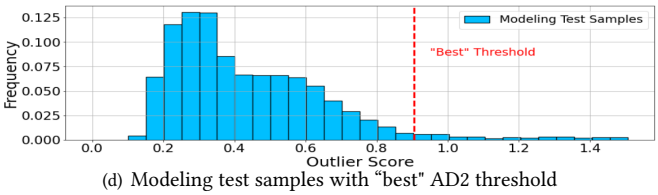
(a) Trace-wise separation: T2 trace of Application 2



(b) App-level separation: all disturbed traces of Application 2



(c) Global-level separation: all disturbed traces with "best" AD2 threshold



(d) Modeling test samples with "best" AD2 threshold

Figure 4: Outlier score distributions using the AE method (LS4, FS_{custom})

In this example, the normal records are separated from anomalous ones for most the data. This shows that the AD method has indeed captured useful signals from the data, performing better than naive classifiers that randomly assign a normal or abnormal label, or assigns each instance to the majority (normal) class.

Moving from the trace- to application- to global level separation, the AUPRC scores gradually decrease. This is because the separation of normal from anomalous instances in outlier score becomes increasingly harder as we broaden the contexts in which data is generated. Figure 4(b) shows the distributions of outlier scores assigned by the AE method to all disturbed traces of Application 2.[1] At the application level, the outlier scores assigned to normal instances spread more, while the ones assigned to T2 instances stay the same, decreasing the model's separation ability. For the global level, this trend is aggravated, as we can see in Figure 4(c).

To understand why, Figures 5(a) and 5(b) show the outlier scores of the T1 and T2 traces of Application 2. The outlier scores assigned to the T2 anomaly are in fact lower than the scores of some normal points in the T1 trace, for two reasons: (a) *Different contexts*: T1 and T2 traces were generated under different input rates, with the rate increase in T1 events around 2.5 times higher than in the T2 events. (b) *Noisy training data*: The normal data in T1 is "noisy". In fact, the normal records in the T1 trace that obtained higher outlier scores than the T2 anomaly exactly match the high processing delay due to Spark checkpointing activities. This indicates that the model has failed to capture these activities as normal behavior.

(2) *Method Comparison:* Regarding separation ability, the best performing method is AE, followed by BiGAN, then LSTM for all levels. AE (and BiGAN) typically produce *smooth* record-wise outlier scores, by taking averages over overlapping windows. The outlier scores produced by the LSTM, however, often exhibit discontinuous *spikes*. For the task of range detection (AD2), such frequent mixes of high and low values make it hard to produce continuous ranges of high outlier scores, penalizing recall when the outlier threshold is set high or precision when the threshold is set low.

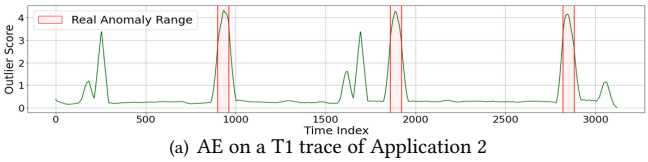
(3) *Anomaly Type Comparison:* For different anomaly types, Table 3 shows that at the global level, the best separated types are T1, T3 and T5 for LSTM and AE, and T1, T4 and T5 for BiGAN. The good performance for T1 (bursty input) and T5 (driver failure) across all methods are largely due to the fact these types have very visible impacts on many of the features output by FS_{custom}, e.g., features relating to the input rate, application delays and memory usage for bursty input, and virtually all features for driver failure. However, most methods offer poor separation for T6 (executor failure) anomalies, due to the limited impact such anomalies have on the FS_{custom} features, where the 6 executor features are *averaged across active executor spots*. As such, the impact of an executor going down is only visible during the (short) period of time for which it shuts down and potentially replaced.

AD1	TS	F1	Prec	Rcl	Rcl for Anomaly Types T1→T6					
LSTM	Best	0.80	0.73	0.91	1.00	1.00	1.00	0.80	1.00	0.61
	Med	0.77	0.67	0.96	1.00	1.00	1.00	1.00	1.00	0.67
AE	Best	0.62	0.60	0.72	1.00	0.75	0.92	0.47	1.00	0.31
	Med	0.59	0.54	0.76	1.00	0.88	1.00	0.57	1.00	0.31
BiGAN	Best	0.48	0.67	0.41	0.78	0.33	0.17	0.37	0.75	0.06
	Med	0.28	0.90	0.19	0.59	0.00	0.00	0.10	0.17	0.06
AD2	TS	F1	Prec	Rcl	Rcl for Anomaly Types T1→T6					
LSTM	Best	0.48	0.59	0.48	0.79	0.24	0.69	0.25	0.71	0.15
	Med	0.38	0.67	0.29	0.42	0.11	0.55	0.16	0.60	0.10
AE	Best	0.56	0.52	0.68	0.99	0.28	0.80	0.52	1.00	0.35
	Med	0.52	0.54	0.60	0.97	0.15	0.67	0.40	1.00	0.15
BiGAN	Best	0.37	0.67	0.28	0.66	0.05	0.06	0.19	0.65	0.00
	Med	0.17	0.90	0.10	0.30	0.00	0.00	0.06	0.17	0.00
AD3	TS	F1	Prec	Rcl	Rcl for Anomaly Types T1→T6					
LSTM	Best	0.40	0.59	0.37	0.64	0.12	0.57	0.15	0.70	0.14
	Med	0.29	0.67	0.20	0.27	0.04	0.37	0.10	0.58	0.08
AE	Best	0.54	0.47	0.71	1.00	0.21	0.86	0.61	1.00	0.38
	Med	0.51	0.54	0.57	0.96	0.06	0.62	0.37	1.00	0.14
BiGAN	Best	0.34	0.67	0.26	0.64	0.01	0.03	0.16	0.65	0.00
	Med	0.14	0.90	0.08	0.23	0.00	0.00	0.05	0.17	0.00

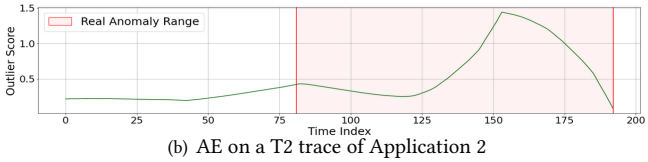
Table 4: Anomaly detection results (LS4, FS_{custom}, AD1-3)

Experiment 2 (LS4, FS_{custom}, AD2). We next examine how the separation abilities translate into actual AD performance via threshold selection. Detection metrics for AD2 (range detection) are reported in the middle section of Table 4. To generate the results, we run each of the STD, IQR and MAD thresholding techniques, leading to different AD performance results for each AD method. Table 4 reports the "best" (in terms of global F1-score) and median performance. The median performance is realistic, while we include the "best" only as an upper bound on performance, as it amounts to unrealistic tuning of the threshold on the test set directly.

[1]For readability, outlier scores greater than 3 times the IQR were grouped together and shown separately in the right plot, which shows the proportion of records with outlier scores beyond 3*IQR for each anomaly type.



(a) AE on a T1 trace of Application 2



(b) AE on a T2 trace of Application 2

Figure 5: Record-wise outlier scores on specific traces (LS4, FS_{custom})

Among the three methods, AE provides the best F1-score, due to its best separation ability reported in the previous experiment. However, F1-score of AE is not very high, 0.52 (median) or 0.56 (best). This is due to the difficulty in choosing a single threshold T on the outlier score for all traces and anomaly types in an *unsupervised* setting, where we select T by using part of the training data, D^2_{train}. Figure 4(d) shows the distribution of the outlier scores assigned to the D^2_{train} samples (the 3% largest were cut for readability), along with the best threshold found on them. This threshold is then used to flag anomalies in the test (disturbed) traces, as shown in Figure 4(c). We see that the anomalies whose scores lie left to T will be missed, penalizing recall, and the normal records whose scores lie right to T will lead to false positives, hurting precision.

Experiment 3. We next evaluate the AD methods under the different AD levels of the benchmark. Detection metrics for AD1-3 are reported in Table 4. The results for AD4 are mostly zeros for the three methods; hence we consider that none of them can support AD4. (AD1) Given our range-based anomalies, a good recall score is easier to reach under AD1. We observe a general increase in performance for all methods. LSTM becomes the best method, because its spikes inside a real anomaly range are now sufficient for getting a good recall score, while using a high threshold to ensure good precision. (AD3) As AD3 awards less recall scores for late detection, AE maintains its performance, indicating that its reported range anomaly is not concentrated at the end of the true range. For LSTM, the performance drops because the early detections it makes are more scattered and hence weigh less in recall.

Due to space constraints, we defer our additional experiments on learning settings and PCA results to to Appendix E.4.

Summary. Our main results are as follows: (1) Our benchmark data carries useful signals that AD methods can pick up to achieve good to modest trace-wise separation of anomalous data from normal data. (2) However, when an AD model has to handle different traces and anomalies, the separation becomes poorer, due to different data generation contexts and noise in training data. (3) The variety of our anomaly types present signals of different strength levels in the data, and hence can be used to stress-test the AD method. (4) For AD2, AE works the best while LSTM is the worse, mostly because the non-smooth outlier scores of LSTM make it hard to handle range anomalies. Our different AD levels also pose varying levels of challenges to the AD method. (5) Our problem setting also poses challenges in the AD modeling process, including unsupervised threshold selection and automated feature engineering.

6.3 ED Evaluation Results and Discussion

In this section, we report the results of running three ED algorithms, MacroBase [3], EXstream [61], and LIME [45], on our benchmark dataset to generate an explanation for each (correctly detected) anomaly. MacroBase and EXstream take an anomaly and a reference dataset, and explain the separation between these two datasets. LIME takes an anomaly and an AD model, for which we use AE, the best AD method described above, and explains why the AD model gives such (high) anomalous scores on the particular input. Since LIME can only explain anomalies of maximum (window) size s, if an anomaly is larger than s, we create multiple windows for LIME to explain. We now show how to use our benchmark metrics to analyze the strengths and limitations of each method, as well as to compare these methods (varying widely in the form of explanation, required input, and method used) in a common framework.

Table 5 summarizes the Exathlon metrics, *conciseness*, *consistency*, *accuracy*, and *running time*, for local (ED1) or global (ED2) explanations using the three ED methods. Conciseness for ED1 and ED2 is the same here, i.e., the average size of a set of explanations. Further, none of the ED methods can be used as a predictive model at the global level, hence lacking ED2 accuracy numbers. We also show examples of explanations: Figure 6(a) shows the explanation returned by each method for a stalled input anomaly (T3). Here we use only the feature index while Appendix E.1 shows the feature names. For LIME, the t suffixes indicate the location within its window, and the coefficients represent feature importance scores. Figure 6(b) shows the features returned by three methods for the same anomaly, under 5 different random subsamples (for LIME, different windows during the anomalous period). Figure 6(c) shows the features of 5 different anomalies of the same stalled input type.

MacroBase. (1) MacroBase generates explanations of 3.16 features on average across all anomaly types. But for some anomaly types, e.g., T3, it generates long explanations, using 6-7 features. The algorithm does not take compactness into consideration, but rather, prefers longer explanations to take care of correlated features. (2) In terms of stability (local consistency), it does not work very well, with an entropy score outside the idea range of $H_1 = 0$ and $H_3 = 1.58$. The reason relates to a correlation between conciseness and stability: as Table 5 shows for different anomaly types, the longer explanations tend to be less stable. See Fig. 6(b) for the MacroBase example. For global consistency, its concordance value further degrades, using inconsistent features in explanations of the same anomaly type. See Fig. 6(c) for an example. (3) Its ED1 accuracy is good for some anomaly types (e.g., T1-3), but poor for some other types (e.g., T5). (4) Its execution time is within a few seconds, except for T3 where the time is up to 18 seconds.

EXstream. (1) EXstream has good conciseness performance, as it uses multiple techniques to prune marginally related features. (2) In terns of stability, it achieves a good average score within $H_3 = 1.58$. However, its concordance value degrades drastically. The reason lies in the lack of the false positive filtering technique (as it requires user labeled data), so that explanations include incidentally correlated features. In the global setting, different irrelevant features may be included depending on the context, hence penalizing the

	MacroBase						EXstream						LIME			
	Concise	Consistency		Prec	Rcl	Time (sec)	Concise	Consistency		Prec	Rcl	Time (sec)	Concise	Consistency		Time (sec)
	ED1/2	ED1	ED2	ED1	ED1	ED1/2	ED1/2	ED1	ED2	ED1	ED1	ED1/2	ED1/2	ED1	ED2	ED1/2
T1	2.20	1.41	2.45	0.96	0.94	1.39	2.06	1.65	2.99	0.88	0.75	0.013	4.34	2.41	2.78	259
T2	1.71	1.18	1.32	0.94	0.96	1.16	3.71	2.10	3.59	0.77	0.57	0.0076	4.71	2.26	2.61	270
T3	6.5	3.05	3.31	0.97	0.96	17.67	1.66	1.11	3.02	0.98	0.91	0.011	4.25	2.47	2.64	258
T4	1.6	1.13	2.94	0.72	0.78	0.16	3.8	1.8	3.78	0.61	0.33	0.0093	2.75	1.81	2.42	256
T5	4.71	3.01	3.63	0.21	0.18	1.6	1.71	1.04	2.52	0.34	0.32	0.0055	3.16	2.37	2.0	267
T6	2.25	1.41	2.57	0.75	0.74	0.46	2.87	1.20	3.5	0.78	0.56	0.0075	3.14	2.13	2.6	263
Ave	3.16	1.86	2.70	0.75	0.76	3.74	2.63	1.48	3.23	0.72	0.57	0.0091	3.72	2.24	2.5	262

Table 5: Results of ED methods, MacroBase, EXstream, and LIME, in terms of conciseness, consistency, accuracy, and running time

MacroBase	LIME	
v_5 <= -0.0120	-1.07 < v_1_t-39 <= -0.20: 0.301	v_1_t-39 > 0.28: -0.152
v_0 <= -0.317	v_18_t-34 <= -0.53: 0.026	v_9_t-4 > 0.36: 0.024
v_2 <= -0.317	-3.57 < v_9_t-39 <= -0.73: 0.023	v_9_t-35 > 0.36:0.018
	v_9_t-37 <= -0.38: 0.022	v_9_t-11 <= -0.40: 0.015
EXstream	0.73 < v_9_t-38 <= 3.57: 0.022	v_12_t-31 > 0.44:0.015
v_5 <= -0.0120		

(a) Explanations of a stalled input anomaly

MacroBase	EXstream	LIME
[5, 2, 0]	[5]	[1, 6, 9, 12]
[5, 2, 0]	[5, 13]	[1, 6, 18]
[14, 2, 4, 0, 16, 5, 17, 15]	[5]	[1, 9, 12]
[14, 2, 4, 0, 16, 5, 17, 15, 6, 7]	[5]	[1, 9, 12]
[14, 2, 4, 0, 16, 5, 17, 15, 6]	[0]	[1, 9, 12]

(b) Stability of a stalled input anomaly

MacroBase	EXstream	LIME
[0, 2]	[5]	[1, 18, 9]
[0, 2, 4, 5, 14, 15, 16, 17]	[5]	[8, 1, 18, 9]
[0, 2, 5]	[5]	[1, 18, 12, 9]
[0, 2, 4, 5, 14, 15, 16, 17]	[14]	[1, 18, 12, 9]
[0, 2, 14, 15, 16, 17]	[2, 6, 16]	[1, 12, 4, 9]

(c) Concordance of 5 stalled input anomalies

Figure 6: Examples of the explanations, stability, and concordance of MacroBase, EXstream, and LIME

concordance. See Fig. 6(c) for the EXstream example. (3) The ED1 accuracy is good for some anomaly types (e.g., T1-3), but poor for other types (e.g., T3-6), especially with poor recall. (4) The execution time is very short as a stream processing algorithm.

LIME. (1) LIME tends to generate longer explanations, e.g., for anomaly types T1-T3. (2) The stability of individual explanations tends to be correlated with the length; longer explanations are shown to be less stable. Concordance across different explanations degrades modestly. (3) ED1 accuracy is not available for LIME, because it cannot be used as a predictive model: the coefficients of the returned features do not sum up to 1 and hence cannot be applied for prediction. (4) The execution time is poor, e.g, 262 seconds on average for finding an explanation.

Comparison. We next compare the three ED algorithms. With the understanding that *conciseness* and *stability* are the most basic requirements of ED algorithms, we first consider these aspects in comparison. MacroBase lacks a mechanism for minimizing the size of an explnation. LIME relies on sparse linear regression to select a few features. Neither is as effective as EXstream, which inspired by the non-monotone submodular optimization problem, eagerly prunes marginally related features. Stability appears to have positive correlation with conciseness. By offering longer explanations, MacroBase and LIME are also less stable.

For global explanations, *concordance* is harder to achieve than stability, as shown for all three methods. This means that the user is likely to see explanations built on different features for anomalies of the same type, which is undesirable. The lack of concordance indicates a direction for future research in the ED area.

For *accuracy* of a local explanation, MacroBase and EXstream return a logical formula that can be applied in a local neighborhood for prediction. MacroBase is more accurate as it pays the cost to evaluate a large number of feature combinations, while EXstream appears to "overfit" the data and suffers in recall. By returning only feature importance scores, LIME cannot be used for prediction.

In the global setting, neither of MacroBase or EXstream can be used for prediction, as they hard-code context-dependent constants in predicates, which do not generalize to other contexts (e.g., a different input rate). This phenomenon is intrinsic in point-based explanations that essentially check if a feature takes a value in a specific range. In order to free explanations from context-dependent predicates, the transition from point-based to temporal explanations that capture the causal relationship between events, independent of the context, could be a relevant direction for future research.

Regarding *efficiency*, EXstream takes about 0.01 second while MacroBase takes 0.1-18 seconds. LIME does not suit stream processing as it takes over 4 minutes to generate an explanation; as such, the application may have missed a window for taking corrective action, e.g., preventing application crash or denial of service.

7 CONCLUSIONS AND FUTURE DIRECTIONS

In this paper, we presented Exathlon – a novel public benchmark for explainable AD, and demonstrated its utility through an experimental analysis of selected AD and ED algorithms from recent literature. Our AD results show that Exathlon's 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 AD methods for semi-supervised learning under noisy training data, mixed anomaly types, and in few-shot learning settings. On the ED front, the literature lacked comparative analysis tools and studies. Our benchmark fills this gap by providing a common framework for analyzing the strengths and limitations of diverse ED methods in their conciseness, consistency, accuracy, and efficiency. These results call for new research to advance the current state of the art of AD and ED, as well as integrated solutions to anomaly and explanation discovery. For a true integration, ED methods should first become capable of discovering range-based explanations, which is also a key step towards automated root cause analysis (a.k.a., "why explanations"). Exathlon's dataset and extensible design are well-positioned to support research progress towards these goals in the long term. Going forward, we envision Exathlon 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 AD and ED community.

REFERENCES

[1] Arvind Arasu, Mitch Cherniack, Eddie F. Galvez, David Maier, Anurag Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. 2004. Linear Road: A Stream Data Management Benchmark. In *VLDB*.

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

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

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

[5] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. 2016. Chaos Engineering. *IEEE Software* 33, 3 (2016), 35–41.

[6] Leopoldo E. Bertossi, Jordan Li, Maximilian Schleich, Dan Suciu, and Zografoula Vagena. 2020. Causality-based Explanation of Classification Outcomes. In *Fourth Workshop on Data Management for End-To-End Machine Learning (DEEM)*. 6:1–6:10.

[7] Ana Maria Bianco, Marta Garcia Ben, Eunie Jr. Martinez, and Victor J. Yohai. 2001. Outlier Detection in Regression Models with ARIMA Errors using Robust Estimates. *Journal of Forecasting* 20, 8 (2001), 565–579.

[8] Loïc Bontemps, Van Loi Cao, James McDermott, and Nhien-An Le-Khac. 2016. Collective Anomaly Detection Based on Long Short-Term Memory Recurrent Neural Networks. In *International Conference on Future Data and Security Engineering (FDSE)*, Vol. 10018. 141–152.

[9] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: Identifying Density-based Local Outliers. In *ACM International Conference on Management of Data (SIGMOD)*. 93–104.

[10] Raghavendra Chalapathy and Sanjay Chawla. 2019. Deep Learning for Anomaly Detection: A Survey. *CoRR* abs/1901.03407 (2019). arXiv:1901.03407 http://arxiv.org/abs/1901.03407

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

[12] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Christopher Ré, and Matei Zaharia. 2019. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. *ACM SIGOPS Operating Systems Review* 53, 1 (2019).

[13] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *ACM Symposium on Cloud Computing (SoCC)*.

[14] The Standard Performance Evaluation Corporation. [n.d.]. SPEC Benchmarks. https://www.spec.org/.

[15] The Transaction Processing Council. [n.d.]. TPC Benchmarks. http://www.tpc.org/.

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

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

[18] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. 2017. Adversarial Feature Learning. In *International Conference on Learning Representations (ICLR)*.

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

[20] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. 2004. Least Angle Regression. *The Annals of Statistics* 32 (2004).

[21] Kareem El Gebaly, Parag Agrawal, Lukasz Golab, Flip Korn, and Divesh Srivastava. 2014. Interpretable and Informative Explanations of Outcomes. *Proceedings of the VLDB Endowment (PVLDB)* 8, 1 (2014), 61–72.

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

[23] FICO. 2018. Explainable Machine Learning Challenge. https://community.fico.com/s/explainable-machine-learning-challenge.

[24] Nicholas Frosst and Geoffrey E. Hinton. 2017. Distilling a Neural Network Into a Soft Decision Tree. In *International Workshop on Comprehensibility and Explanation in AI and ML*.

[25] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. 2013. BigBench: Towards an Industry Standard Benchmark for Big Data Analytics. In *ACM SIGMOD International Conference on Management of Data*.

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

[27] Jim Gray. 1993. *The Benchmark Handbook for Database and Transaction Systems*. Morgan Kaufmann.

[28] Manish Gupta, Jing Gao, Charu C. Aggarwal, and Jiawei Han. 2014. Outlier Detection for Temporal Data: A Survey. *IEEE Transactions on Knowledge and Data Engineering* 26, 9 (2014), 2250–2267.

[29] Geoffrey Hinton and Ruslan Salakhutdinov. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science* 313, 5786 (2006), 504 – 507.

[30] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

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

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

[33] Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. 2016. Interpretable Decision Sets: A Joint Framework for Description and Prediction. In *KDD*. 1675–1684.

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

[35] Boduo Li, Edward Mazur, Yanlei Diao, Andrew McGregor, and Prashant J. Shenoy. 2012. SCALLA: A Platform for Scalable One-Pass Analytics Using MapReduce. *ACM Transactions on Database Systems* 37, 4 (2012), 27.

[36] Dan Li, Dacheng Chen, Jonathan Goh, and See-Kiong Ng. 2018. Anomaly Detection with Generative Adversarial Networks for Multivariate Time Series. *CoRR* abs/1809.04758 (2018).

[37] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-Based Anomaly Detection. *ACM Transactions on Knowledge Discovery from Data* 6, 1 (2012), 3:1–3:39.

[38] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Annual Conference on Neural Information Processing Systems (NIPS)*. 4765–4774.

[39] 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. 2020. Diagnosing Root Causes of Intermittent Slow Queries in Large-Scale Cloud Databases. *Proceedings of the VLDB Endowment (PVLDB)* 13, 8 (2020), 1176–1189.

[40] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. 2015. Long Short Term Memory Networks for Anomaly Detection in Time Series. In *European Symposium on Artificial Neural Networks (ESANN)*. 89–94.

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

[42] Christoph Molnar. 2021. Interpretable Machine Learning: A Guide for Making Black Box Models Explainable. https://christophm.github.io/interpretable-ml-book/.

[43] Tilmann Rabl, Christoph Brücke, Philipp Härtling, Stella Stars, Rodrigo Escobar Palacios, Hamesh Patel, Satyam Srivastava, Christoph Boden, Jens Meiners, and Sebastian Schelter. 2019. ADABench - Towards an Industry Standard Benchmark for Advanced Analytics. In *TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC)*.

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

[45] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *SIGKDD*, 1135–1144.

[46] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-Precision Model-Agnostic Explanations. *AAAI*.

[47] Sudeepa Roy, Laurel Orr, and Dan Suciu. 2015. Explaining Query Answers with Explanation-Ready Databases. *Proceedings of the VLDB Endowment (PVLDB)* 9, 4 (2015), 348–359.

[48] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. 2017. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery. In *Information Processing in Medical Imaging (IPMI)*. 146–157.

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

[50] Spark-uses [n.d.]. How are Big Companies using Apache Spark. https://medium.com/@tao_66792/how-are-big-companies-using-apache-spark-413743dbbbae.

[51] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic Attribution for Deep Networks. *ICML*, 3319–3328.

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

[53] Luan Tran, Liyue Fan, and Cyrus Shahabi. 2015. Distance Based Outlier Detection for Data Streams. *Proceedings of the VLDB Endowment* 9, 4 (2015), 1089–1100.

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

[55] Yaqing Wang, Quanming Yao, James T. Kwok, and Lionel M. Ni. 2020. Generalizing from a Few Examples: A Survey on Few-shot Learning. *Comput. Surveys* 53, 3 (2020), 63:1–63:34. https://doi.org/10.1145/3386252

[56] Eugene Wu and Samuel Madden. 2013. Scorpion: Explaining Away Outliers in Aggregate Queries. *Proceedings of the VLDB Endowment (PVLDB)* 6, 8 (2013), 553–564.

[57] Mike Wu, Michael C. Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. 2018. Beyond Sparsity: Tree Regularization of Deep Models for Interpretability. *AAAI*.

[58] 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. 2018. Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications. In *International World Wide Web Conference (WWW)*. 187–196.

[59] Jiawei Yang, Susanto Rahardja, and Pasi Fränti. 2019. Outlier Detection: How to Threshold Outlier Scores?. In *International Conference on Artificial Intelligence, Information Processing and Cloud Computing (AIIPCC)*. 37:1–37:6.

[60] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. 2018. Efficient GAN-Based Anomaly Detection. *CoRR* abs/1802.06222 (2018). http://arxiv.org/abs/1802.06222

[61] Haopeng Zhang, Yanlei Diao, and Alexandra Meliou. 2017. EXstream: Explaining Anomalies in Event Stream Monitoring. In *International Conference on Extending Database Technology (EDBT)*. 156–167.

A PUBLIC REPOSITORY

The dataset, code, and documentation for Exathlon are publicly available at https://github.com/anomalybench/anomaly-bench.git.

B DETAILS ON DATA COLLECTION

In this section, we provide additional details on anomaly design and data collection.

B.1 Undisturbed and Disturbed Traces

Undisturbed Traces. To capture the normal state of execution, we ran experiments continually (5 concurrent programs at a time) for a month, using the input rate values and configurations of Spark parameters that suit the capacity of the cluster, and collected their traces. As noted above, there were instances of downtime of the cluster and we manually pruned the traces affected by the downtime. We consider the remaining traces representative of the normal state because the cluster was not interrupted or stress tested by disruptive external events. We obtained 59 such traces, constituting the main bulk of our dataset (around 15.3GB of data).

It is important to note that the undisturbed traces also exhibit a great deal of variability in the recorded metrics. For example, engineers often check the following metrics to see if their Spark streaming applications are making progress : (a) *scheduling delay:* the delay between the scheduling of a task and the start of its processing; (b) *processing time:* time taken to process a batch of data from the streaming input; (c) *number of processed records* in current execution. As Figure 7(a) shows, both scheduling delay and processing time can rise high beyond a normal range of values, while the number of processed records increases steadily over time. The spikes of the first two metrics are likely to be caused by other system operations, e.g., checkpointing in Spark or CPU usage by a DataNode in HDFS (Hadoop File System). Since such variations appear in almost every trace, they should be considered to be part of the normal state.

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 6 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.

Bursty Input (Type 1): For every application, the user expects a certain range of data input rates and configures Spark parameters to allocate sufficient resources for such input rates. However, on some special occasions (e.g., a special sales day for a retailer) the input rate can increase significantly, and existing resources are not sufficient for handling the higher data rate. Such phenomena can be reflected in the data, e.g., in increased values of processing time, memory usage, and scheduling delay, as shown in Figure 7(b). It is because each batch of data (e.g., in the past 10 seconds) increases dramatically in size, and the processing time of the batch increases accordingly. When the processing time exceeds the batch interval (e.g., 10 seconds), the application cannot process data fast enough; so data is put in memory by the receivers and such data experiences a higher delay before it is scheduled for processing. Early detection of busty input is helpful because it can lead to corrective action such as allocating more resources to the application.

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 and then set the input rate back to its normal range. We repeated this pattern multiple times during a given trace, with sufficient gap between two instances. As such, we created a total of 29 instances of this anomaly type over 6 different traces. The duration of each injected event (marked by its DEG execution), together with the type of the event, was recorded as the **root cause** as shown in Table 1(b).

Bursty Input Until Crash (Type 2): Spark developers rank the out of memory (OOM) condition to be the number one reason of Spark application failures. To capture such phenomena, we extended the bursty input design: instead of reducing the input rate back to a normal range, we kept the input at the same high rate until the application eventually crashed. Figure 7(c) shows how the system behaviors during such an event are reflected in the data: The application starts with a normal input rate. At some point, the input rate rises high, and processing time and scheduling delay build up until one of the executors fails due to out-of-memory error. Then a new executor is launched to replace the failed one, but the sustained high rate will make more executors to fail. When the number of failed executors reaches a threshold, the application will be killed by Spark, leading to an application crash.

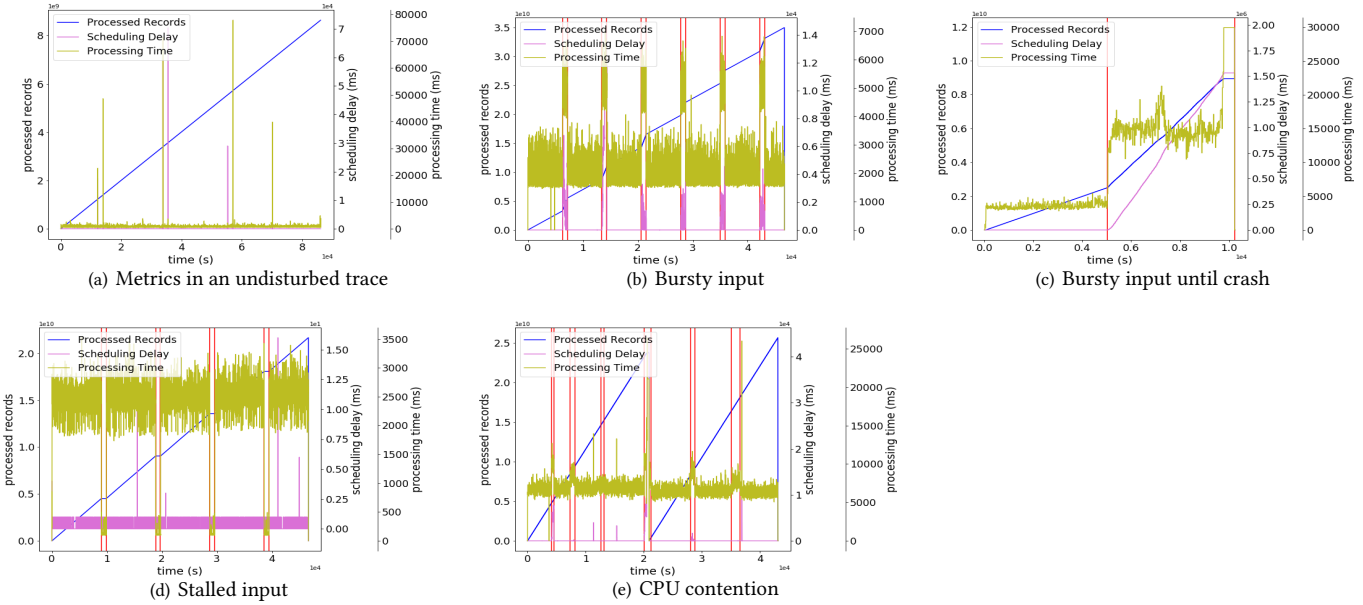
To instrument this anomaly type, we ran the DEG forever, first crashing the executors due to lack of memory, and eventually crashing the application. We injected this anomaly into 7 different traces.

Stalled Input (Type 3): Another type of input related anomaly is stalled input, i.e., no data is injected into the Spark application. This may indicate a failure of the data source (e.g., Kafka or HDFS) and render a waste of resources as the driver and executors are still running but with no data to process. This type of anomaly can be reflected in our collected metrics: since no data is coming from the receiver, the number of processed records (as well as other related metrics) does not increase, and the processing time is much lower than in the normal state.

To generate traces, 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.

CPU Contention (Type 4): A Spark cluster is usually run with a resource manager (e.g., YARN) that allocates resources to each application so that each of them has exclusive access to its own resources including CPU and memory. However, the resource manager cannot prevent external programs from using the resources that it allocated previously, which is a common reason for performance issues in distributed environments with high concurrency. For instance, it is not uncommon to have a Hadoop DataNode using a large amount of CPU on a node also used by a Spark application. This generates a contention for resources and will often slow down the progress of the Spark application.

We reproduced this type of 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



(a) Metrics in an undisturbed trace (b) Bursty input (c) Bursty input until crash

(d) Stalled input (e) CPU contention

Figure 7: Metrics observed in normal data and anomaly instances (a pair of red vertical bars marks a root cause event)

Figure 7(e) for a trace with multiple instances of this type of anomaly, where each instance is marked by a pair of red vertical bars. As a result, there are several intervals during which the processing time is higher than normal due to CPU contention on the affected node, meaning that one or more executors are using fewer resources than allocated. If the processing time is still lower the batch interval, the CPU contention does not affect much the progress of the application; otherwise, scheduling delay starts to build up. Besides processing time, CPU contention can have a more severe impact on the Spark application: if the driver is performing a lot of computation and CPU contention occurs to the driver node, it can cause a crash of the driver. Then the application master has to restart the driver, which will itself restart all the executors.

Driver Failure (Type 5) and Executor Failure (Type 6): Another type of anomalies includes abrupt failures in distributed systems. A common example is a node failure caused by a hardware fault or a maintenance operation. In this case, all the processes (drivers and/or executors) located on that node will be unreachable. Such processes must be restarted on another node, which causes processing delays. We created such 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. We also created anomalies by failing executor processes, which get restarted 10 seconds after the failure, but its effects on metrics such as processing delay may continue longer. We created 9 driver failures and 10 executor failures over 11 different traces.

B.2 Extended Effect Intervals

In this section, we describe the way that we generated the extended effect interval (see Table 1(b)) for each anomalous event. Each anomaly instance was initially labeled with the known type and the *root cause interval* (RCI). However, we observed that an injected event could have a long-lasting effect. For example, when an event caused CPU contention to running Spark applications, after its end time it would still take some time for the processing time and scheduling delay to come back to normal. Therefore, we seek to offer a larger interval for an anomaly detection algorithm to recognize the event. More specifically, we used domain knowledge to set the *extended effect interval* (EEI), that starts immediately after the RCI ends, and ends at a point after which we deem anomaly detection not helpful. This means that the EEI could end at the time of an application crash, or the point that the main metrics become normal in our traces.

(1) Bursty Input: The end of the EEI is set to the point when highly related metrics such as the processing time and scheduling delay come back to normal.
(2) Bursty Input Until Crash: The EEI is set to null. It is because the root cause event already ends at the time of the application crash.
(3) Stalled Input: The end of the EEI is set to the point when the processing time comes back to normal (the application restarts processing data at its usual rate).
(4) CPU Contention: a) If the effect is increased processing time, we set the end of EEI to the time when processing time and scheduling delay come back to normal. b) If the effect is the driver crash, we set the end of EEI to the time when the application restarts (typically 1 minute after the crash in practice).
(5) Driver Failure: The end of the EEI is set to when the application restarts.
(6) Executor Failure: If the executor is replaced, we observe an increase in scheduling delay during the replacement time. If it is not replaced, then the buildup in scheduling delay either lasts temporarily or until the application crashes. In all these cases, we set the end of the EEI to the point when the scheduling delay comes back to normal.

	Precision Parameters			Recall Parameters		
AD Functionality Level	α	δ	γ	α	δ	γ
AD1: Anomaly Existence	0	Flat	1	1	N/A	N/A
AD2: Range Detection	0	Flat	1	0	Flat	1
AD3: Early Detection	0	Flat	1	0	Front	1
AD4: Exactly-Once Detection	0	Flat	0	0	Front	0

Table 6: Range-based precision/recall parameter settings

Then the combined anomaly interval, RCI plus EEI, gives our benchmark more freedom to evaluate anomaly detection algorithms.

C DETAILS ON BENCHMARK DESIGN

C.1 Anomaly Detection (AD) Functionality

Table 6 further shows how to set the tunable parameters of the range-based precision/recall framework [52] to capture the requirements of AD1-AD4 in general. We provide a brief insight for these parameters here, and refer the reader to the original paper for further details [52]. The α parameter is to reward detecting the existence of an anomaly range and is only meaningful in the context of recall when we only care about anomaly existence. Therefore, it always gets a 0 value except when computing AD1's recall. The δ parameter captures the positional bias, i.e., the reward to be earned from which portion of the anomaly range was successfully detected. It is the most relevant for AD3 and beyond, where detection latency is considered. For recall of AD3-4, δ should be set to "Front" to favor early detection, and "Flat" in all other cases, where detection position does not matter. Finally, the γ parameter is to penalize fragmented ranges (i.e., an anomaly range detected as multiple subranges) and is, therefore, directly relevant to exactly-once detection. For AD4, we set γ to 0, indicating that fragmentation is strictly undesirable; for all others, it has no penalizing effect to the score. We would like to note that there is a fourth parameter, ω, which is to compute how much reward is earned from the size of the anomaly range that is correctly detected (relevant for AD2 and beyond). We use its default, additive definition in the original model with a minor normalization adjustment to ensure monotonicity.

D DETAILS ON PIPELINE DESIGN

7. ED Evaluation. After processing each test trace, we obtain a set of anomalies with their corresponding explanations. We then collect the explanations from all the test traces to run the final ED evaluation To compute the metrics proposed in our benchmark, we implement the following functions for each ED method: (i) The extraction function $G_{\mathbf{A}}(F_{t,w})$ is used to extract the feature set from each explanation, required for conciseness and consistency measures. We support extraction functions for common forms of explanations, including logical formulas, linear models, and decisions trees. (ii) A subsampling procedure is applied to each anomaly $X_{t,w}$ for computing the stability and accuracy measures in the ED1 case. Here, if an ED method can explain an anomaly of arbitrary duration, our sampling method performs a random split of the anomaly dataset $X_{t,w}$ (together with the reference set, if present) so that 80% of each dataset is used to construct an explanation from this sample. Some other ED methods can only explain an anomaly of a maximum size s, e.g., LIME [45]. If the duration w of the detected anomaly $X_{t,w}$ is larger than s, we create samples of size s that are evenly spread across $X_{t,w}$ and then call the ED method to explain each of these samples. The implementation of concordance and accuracy measures in the ED2 case directly follows their definitions.

E DETAILS OF OUR EXPERIMENTAL STUDY

All of our experiments were performed in a cluster of 20 compute nodes, each with 2 Intel® Xeon® Gold 6130 16-core processors, 768GB of memory, and 64TB disk.

E.1 Customized Feature Set

Our custom feature set was produced via the manual selection of relevant features resulting from domain knowledge and various visualizations. This feature set characterizes different aspects of a running application, with 9 features relating to the Spark application driver, 6 to the executors, and 4 to the OS metrics. We list the complete list of features below:

- **Driver features**
  - Processing delay, scheduling delay and total delay of the last completed batch.
  - First-order difference ($f_t := f_{t+1} - f_t$, simply called *difference* in the following) in number of completed batches, received records and processed records since the beginning of the application.
  - Difference in number of records in the last received batch.
  - Difference in total memory used (MB).
  - Difference in heap memory used by the JVM.
- **Executor features** – All averaged across active Spark executors (before differencing).
  - Difference in number of HDFS writing operations.
  - Difference in CPU time and runtime.
  - Difference in number of records read and written during shuffling operations.
  - Difference in heap memory used by the JVM.
- **OS features** – All for each of the four nodes of the mini-cluster the applications were run on.
  - Difference in global CPU percentage idle time

In the following, we list the feature names with their indexes, which are used in our explanations shown in Figure 6.

0. driver_Streaming_lastCompletedBatch_processingDelay_value
1. driver_Streaming_lastCompletedBatch_schedulingDelay_value
2. driver_Streaming_lastCompletedBatch_totalDelay_value
3. 1_diff_driver_Streaming_totalCompletedBatches_value
4. 1_diff_driver_Streaming_totalProcessedRecords_value
5. 1_diff_driver_Streaming_totalReceivedRecords_value
6. 1_diff_driver_Streaming_lastReceivedBatch_records_value
7. 1_diff_driver_BlockManager_memory_memUsed_MB_value
8. 1_diff_driver_jvm_heap_used_value
9. 1_diff_node5_CPU_ALL_Idle%
10. 1_diff_node6_CPU_ALL_Idle%
11. 1_diff_node7_CPU_ALL_Idle%
12. 1_diff_node8_CPU_ALL_Idle%
13. 1_diff_avg_executor_filesystem_hdfs_write_ops_value
14. 1_diff_avg_executor_cpuTime_count

15. 1_diff_avg_executor_runTime_count
16. 1_diff_avg_executor_shuffleRecordsRead_count
17. 1_diff_avg_executor_shuffleRecordsWritten_count
18. 1_diff_avg_jvm_heap_used_value

E.2 Details of AD Methods

We next describe in more detail the methods we used for the normality modeling, outlier score derivation and threshold selection steps of AD modeling when conducting our experiments.

LSTM. Long Short Term Memory (LSTM) [30] is a deep neural network structure designed for handling sequential data. In the context of AD, it is typically used as a forecasting based method, where the outlier score for a given data point is derived from its forecasting error by the model. Like in a recent work [8], our experiments used LSTM for AD by setting the outlier score of a point as its relative forecasting error. Unlike in [8], the scores produced here were however not further averaged but kept as is.

AE. An Autoencoder (AE) [29] is a deep neural network structure designed to map its input data to a latent representation in lower dimensional space (or *coding*), by learning to accurately reconstruct it through its structure. In the context of AD, autoencoders are typically used as reconstruction based methods, where the outlier score of a data sample is derived from its reconstruction error by the model. In our experiments, we used a dense autoencoder architecture, and defined the Mean Squared Error (MSE) of an input window as its outlier score.

BiGAN. Generative Adversarial Networks (GAN) [26] learn to generate samples with similar statistics as their training data, by training a generator network to map random noise in latent space to samples that appear realistic to a discriminator network. Bidirectional Generative Adversarial Networks (BiGAN) [18] are used to automatically learn the inverse mapping of the generator, through the joint training of an encoder network, that can be later used in pair with the generator to reconstruct some (real) input samples. As such, BiGAN can also be used in the context of AD as a reconstruction based method, with an additional flexibility in deriving the outlier score for a test window, leveraging the architectures of all its three networks [60]. In our pipeline, we experimented with network architectures allowing the usage of LSTM networks to better handle time sequences, like done in [36]. The outlier score of a test window was defined as the average of its MSE by the (encoder, generator) pair and its feature loss by the discriminator, as defined in [60].

Threshold Selection. Throughout our experiments, threshold selection was performed based on the distribution of the outlier scores assigned by an AD method to the samples of D^2_{train}. More precisely, we experimented with threshold definitions of the form:

$$\texttt{threshold} = S_1 + c \cdot S_2$$

Where S_1 and S_2 are two descriptive statistics of the D^2_{train} sample outlier scores, and c a constant we refer to as the *thresholding factor*. We also allowed this computation to be performed twice, removing for the second iteration the outlier scores that were above the threshold in the first iteration, in order to drop any obvious outliers that could prevent us from finding a suitable threshold.

For every AD Evaluation in our experiments, the following threshold selection methods were evaluated for $c \in \{1.5, 2, 2.5, 3\}$:

- **STD** : $S_1 = \bar{m}$, $S_2 = s$, *i.e.* sample mean and standard deviation.
- **MAD** : $S_1 = \text{median}, S_2 = \text{MAD}$, where $\text{MAD} = 1.4826 \cdot \text{median}(|X - \text{median}(X)|)$.
- **IQR** : $S_1 = Q_3$, $S_2 = Q_3 - Q_1 = \text{IQR}$.

This led to a total of 24 thresholding parameter combinations, and hence 24 final detection performances for each evaluated AD method. The numbers reported in Table 4 correspond to the best performance that was achieved in terms of F1-score in the disturbed traces, along with the median value for the 24 performances.

E.3 Details of ED Methods

We further integrated three recent ED methods into our pipeline: EXstream [61] and MacroBase [3] from the database community for outlier explanation in data streams, and LIME [45], an influential ED method from the ML community.

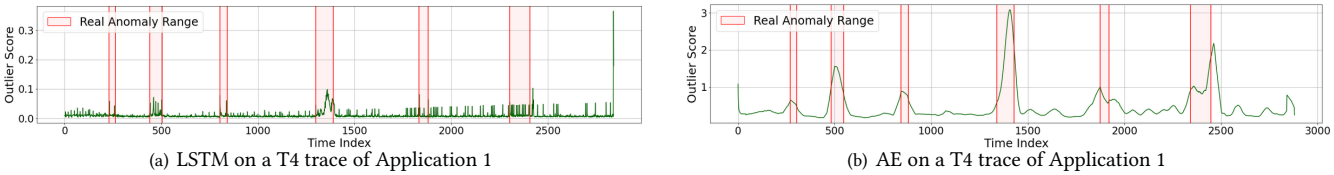
EXstream [61] takes an anomaly dataset and a reference dataset. It first calculates the entropy-based single-feature reward, which indicates the ability of this feature to separate normal data from anomalous data. Then it uses several heuristics to find a small set of features with high rewards to build the explanation. One step is false positive filtering that prunes features only incidentally correlated with the anomaly. This step requires generating extra labeled data based on user annotations of a particular time series. Since such user annotations are not available in our problem setting, we did not include this step in our implementation.

MacroBase [3] takes the same inputs as EXstream. It first calculates the risk ratio for each feature. Then it evaluates the same measure for feature combinations based on a frequent itemset mining procedure. It also employs several optimizations to boost the efficiency. Since it is designed for categorical features, in our implementation, we add a extra step transforming each numerical feature into categorical values (via equal width binning) .

LIME [45] is originally designed to provide an explanation for a classifier prediction. It also provides an interface for regression models, and for temporal input. Internally, LIME leverages Lasso [20] to identify k important features first, and then uses a customized loss function to learn a new linear model to approximate the original model output locally. In our implementation, we use the RecurrentTabularExplainer interface provided by LIME, and set $k = 5$.

E.4 Additional AD Evaluation Results

Experiment 1 (LS4, $\text{FS}_{\text{custom}}$, AD2): In method comparison, in terms of separation ability, the best performing method is AE, followed by BiGAN, and then LSTM for all levels. To understand why, we show in Figures 8(a) and 8(b) the outlier scores produced by the LSTM and AE methods, respectively, for the T4 (CPU contention) trace of Application 1. AE (and BiGAN) typically produce *smooth* record-wise outlier scores, by taking averages over overlapping windows. The outlier scores produced by the LSTM, however, often exhibit discontinuous *spikes*. For the task of range detection (AD2), such frequent mixes of high and low values make it hard to produce continuous ranges of high outlier scores, penalizing recall when the outlier threshold is set high or precision when the threshold is set low.



(a) LSTM on a T4 trace of Application 1

(b) AE on a T4 trace of Application 1

Figure 8: Record-wise outlier scores on specific traces (LS4, FS_{custom})

Method	Ave	AUPRC for Anomaly Types T1→T6					
LSTM	0.14	0.15	0.14	0.09	0.14	0.21	0.12
AE	0.39	0.43	0.39	0.18	0.49	0.49	0.38
BiGAN	0.34	0.40	0.29	0.21	0.47	0.37	0.29

Table 8: Global Separation Results (LS4, FS_{pca}, AD2)

LS	Method	Ave	AUPRC for Anomaly Types T1→T6					
LS1	**LSTM**	0.43	0.55	0.35	0.46	0.33	0.58	0.32
	AE	0.58	0.72	0.41	0.67	0.54	0.71	0.42
	BiGAN	0.55	0.82	0.35	0.37	0.53	0.77	0.47
LS2	**LSTM**	0.46	0.55	0.37	0.55	0.36	0.61	0.32
	AE	0.58	0.66	0.41	0.61	0.57	0.75	0.45
	BiGAN	0.54	0.72	0.33	0.65	0.56	0.68	0.31
LS3	**LSTM**	0.43	0.58	0.34	0.44	0.33	0.58	0.30
	AE	0.57	0.72	0.40	0.64	0.53	0.69	0.44
	BiGAN	0.52	0.83	0.31	0.33	0.51	0.72	0.43
LS4	**LSTM**	0.47	0.57	0.37	0.56	0.38	0.60	0.35
	AE	0.57	0.65	0.40	0.63	0.55	0.79	0.43
	BiGAN	0.52	0.81	0.36	0.25	0.54	0.69	0.48

Table 7: Application-Wise Separation Results (LS1:4, FS_{custom}, AD2)

Experiment 4 We next examine the effect of different Learning Settings under AD2. Table 7 reports the results. Looking at the application-level AUPRCs of the three methods, we see that AE and BiGAN follow the expected trend, with their average separation ability decreasing as we increase data variety and disable picking from the beginning of test traces. On the contrary, LSTM seems to perform better for LS2 and LS4 (N-App settings), maybe indicating that this method benefits more from the increase in data cardinality than is penalized by its increase in variety. Although the changes in results appear as minor in these examples, we can see that training models with different Learning Settings can help drawing some conclusions regarding their data needs and generalization ability.

Experiment 5 In this last experiment, we study the effect of feature extraction on the performance of the three methods, by comparing our custom feature selection (FS_{custom}) to applying PCA directly to the raw input features (FS_{pca}). From Table 8, we can see that the global separation abilities of all methods drop significantly when using FS_{pca} (especially for LSTM). This is largely due to the fact that PCA is not able to preserve the right signals for detecting a major portion of our anomalies.

When using FS_{pca}, we are not averaging across active executor features anymore, which likely plays a part in the fact that separation abilities for T6 separation were either maintained or increased for AE and BiGAN. Another aspect of FS_{pca} is that it selects features based on their variance, which means that features tending not to vary much in normal scenarios, like the scheduling delay or input rate, will be far less represented in the output space than features having naturally more activity, like CPU or memory related metrics. This likely explains why T4 separation ability is also maintained or increased for AE and BiGAN.

In summary, we can see that this benchmark can also enable us to analyze the effect of constructing different feature spaces on the overall AD down to the detection of specific types.