OrionBench: Benchmarking Time Series Generative Models in the Service of the End-User

Sarah Alnegheimish [1] **Laure Berti-Equille** [2] **Kalyan Veeramachaneni** [1]

Abstract

Time series anomaly detection is a vital task in many domains, including patient monitoring in healthcare, forecasting in finance, and predictive maintenance in energy industries. This has led to a proliferation of anomaly detection methods, including deep learning-based methods. Benchmarks are essential for comparing the performances of these models as they emerge, in a fair, rigorous, and reproducible approach. Although several benchmarks for comparing models have been proposed, these usually rely on a one-time execution over a limited set of datasets, with comparisons restricted to a few models. We propose *OrionBench*– an end-user centric, continuously maintained benchmarking framework for unsupervised time series anomaly detection models. Our framework provides universal abstractions to represent models, extensibility to add new pipelines and datasets, hyperparameter standardization, pipeline verification, and frequent releases with published updates of the benchmark. We demonstrate how to use *OrionBench*, and the performance of pipelines across 17 releases published over the course of four years. We also walk through two real scenarios we experienced with *OrionBench* that highlight the importance of continuous benchmarking for unsupervised time series anomaly detection.

1. Introduction

As continuous data collection becomes more commonplace across domains, there is a corresponding need to monitor systems, devices, and even human health and activity in order to find patterns in collected data, as well as deviations from those patterns (Chandola et al.; Aggarwal, 2017). Over the past decade, tremendous progress has been made in using machine learning to perform various types of monitoring, including unsupervised time series anomaly detection. For example, in the past 5 years, Hundman et al. (2018) created a Long Short-Term Memory (LSTM) forecasting model to find anomalies in spacecraft data, Park et al. (2018) used LSTM variational autoencoders for anomaly detection in multimodal sensor signals collected from robotic arms, and Geiger et al. (2020) used generative adversarial networks for time series anomaly detection on widely used public datasets.

These methods have gained popularity given their unsupervised nature. With supervised learning, models learn patterns from human-labelled data and then use those patterns to detect other anomalies. That these models are restricted to previously labelled events, which are themselves difficult for humans to find, makes it more challenging for models to make useful predictions. Moreover, these models struggle to find "new" events that are interesting to the user. In contrast, with unsupervised learning, no ground truth is given to the model, revealing anomalies that may have otherwise gone unseen. This property is highly valuable to users, who are often unable to determine *what* they are looking for and *when* it will occur. In this paper, we focus on unsupervised models.

When *end-users* – defined here as people who are interested in training a model on their own data in order to find anomalies – attempt to use these models, they regularly run into particular challenges and pain points, which we highlight in Table 1.

One challenge is simply deciding which model to use. Rapid innovation in the machine learning space, where papers are regularly published presenting new state-of-the-art (SOTA) models, means many users are in a constant state of struggling to keep up. Moreover, if users do decide to use the latest pipeline – perhaps alongside their existing approach – they often find themselves unsure of how to get started, as research papers are full of new terminology published alongside obfuscated code. Another challenge comes with determining whether one model works better than another. Users might find that the new model did not actually improve on their existing model. Lastly, and more subtly, a new model can outperform existing models not because

[1]MIT LIDS, Cambridge MA [2]IRD ESPACE-DEV, Montpellier, France. Correspondence to: Kalyan Veeramachaneni <kalyanv@mit.edu>.

Table 1. A set of pain points experienced by end-users and their corresponding research questions we aim to address.

	End-User Pain Point		Research Question
Fear of missing out	Numerous generative modeling techniques are published, each promising better performance than all previous models. An end-user worries if the model they have been using is suboptimal and needs to updated.	**RQ1**	How can we support end-users in confidently making the decision of whether or not to adopt a new model?
Unable to parse complex jargon	Published work has a lot of complex machine learning specific jargon which makes it impossible for an end-user to approach implementation or delineate differences between what they have been using versus the new methods. Important components (e.g. pre-processing functionalities) are hidden behind the complexity of the model, when in reality these components are what made the model successful.	**RQ2**	How can we best represent models with proper abstractions, such that new models can be represented as a set of components and one can identify the differences between models easily?
Time lost with no improvement	With these two challenges above, end-users may spend substantial time trying to adapt a new model's code for their data, only to discover that the new model did not outperform their current model on their data.	**RQ3**	How can we provide end-users with ready-to-use models should they choose to adopt?

of the model itself, but due to the inclusion of important pre- and/or post-processing operations, a distinction that may not be apparent to the user. This entire process can be time-consuming, taking 6-12 months after the research is published for an end-user to figure out and decide whether or not to incorporate a newly published method. Figure 1(left) depicts this asynchrony between the research process and method usage.

It is important to note that an end-user is only interested in finding the best solution for their particular problem, and does not think about these modeling techniques in the way that researchers do. In recent years, benchmarks have become instrumental in gauging and comparing model performance for machine learning researchers (Coleman et al., 2019; Han et al., 2022). In this paper, we ask — can benchmarking frameworks also help to alleviate end-users challenges? Specifically, we ask if it possible to bring together both the researcher and the end-user to utilize benchmarking systems and reduce the time required to use the latest research.

We propose *OrionBench*– an end-user centric benchmarking framework for unsupervised time series anomaly detection. Figure 1(left) illustrates how in a regular setting researchers and end-users operate independently from one another, creating hurdles for end-users when adopting a new published model. With our system Figure 1(right), all researchers (Orion and ML researchers) directly contribute their model to *OrionBench*. This creates a single source of readily-available models for the end-user.

Three concrete innovations enable us to address end-users' common but critical concerns:

- A continuously running system, moving away from *point-in-time* evaluations.
- Abstractions that allow us to easily incorporate and assess new models, and isolate the factors that make them better.
- Seamless integration of the latest models into usable *pipelines* for end-users.

Below we highlight our framework's unique contributions. *OrionBench* is:

1. **A standardized framework** that enables the integration of new pipelines and datasets. *OrionBench* started with 2 pipelines in 2020, and as of now encompasses 12 pipelines, 28 primitives, 14 public datasets, and 2 custom evaluation metrics. Once integrated into the framework and benchmarked, a pipeline is seamlessly made available to the end-user through unified APIs.
2. **A continuously-run benchmark** with frequent releases. To date, we have 17 benchmark leaderboards covering almost four years, accumulating over 70,032 experiments. In addtion, we demonstrate the stability and reproducibility of *OrionBench*.
3. **An end-to-end benchmark** executable with a single command. Given a pipeline and datasets, the benchmark evaluates the performance of the pipeline on every signal according to time series anomaly detection-based metrics. We provide an extensive evaluation that illustrates the qualitative and computational performances of pipelines across all datasets according to time series anomaly detection-based metrics.
4. **Open-source** and publicly available: `https://github.com/sintel-dev/Orion`.[1]

[1]Reproducing paper figures and tables is available: `https://github.com/sarahmish/orionbench-paper`

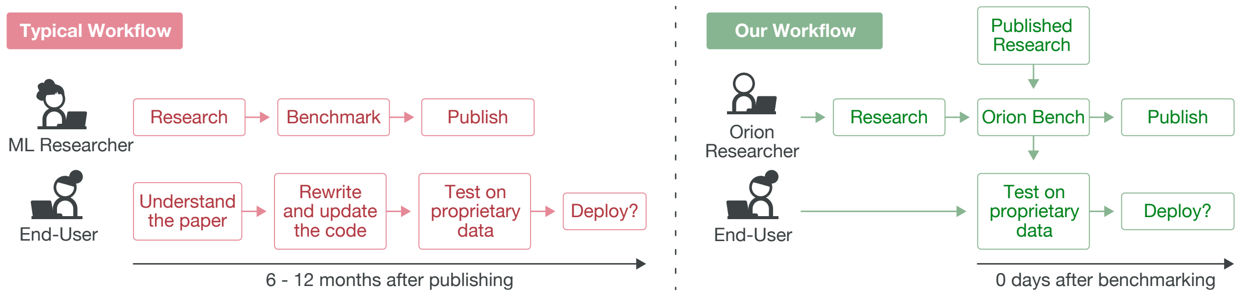


Figure 1. Typically researchers and end-users have independent processes. Researchers develop their method and benchmark it to publish their papers. Once these methods are publicized, end-users work on first understanding the model then adapting the code to work on their own data. After it is tested, end-users decide whether the performance is sufficient for it to be deployed or not. With `OrionBench`, we aim to have a single hub where researchers can benchmark their pipelines and become instantaneously available to end-users.

2. *OrionBench*

OrionBench is a benchmark suite within the Orion system (Alnegheimish, 2022). A researcher creates a new model and integrates it with Orion through a pull request. A benchmark run is executed and produces a leaderboard, and the model is then stored in the sandbox. This part of the workflow satisfies the goals of the researcher, which is comparing the performance of different models. To serve end-users, pipelines in the sandbox are tested by an Orion developer. Pipelines that pass the tests are verified and become available to end-users. This workflow is depicted in Figure 2. Five main properties enable our framework for benchmarking unsupervised time series anomaly detection models: abstractions that enable us to compose models as pipelines (directed acyclic graphs) of reusable components called *primitives*; hyperparameter standardization; extensions to add new pipelines and datasets; verification of pipelines; and continuous benchmark releases. Lastly, we conclude the section by illustrating how *OrionBench* benefits the end-user.

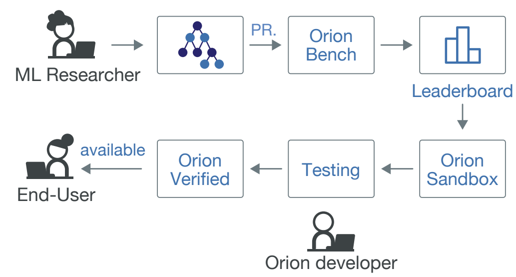


Figure 2. OrionBench integrates new models made by ML researchers and compares its performance to currently available models through the leaderboard. After testing the validity and reproduciblity of the model, it is transferred from "sandbox" to "verified" and becomes readily available to the end-user.

2.1. Abstracting Models into Primitives and Pipelines

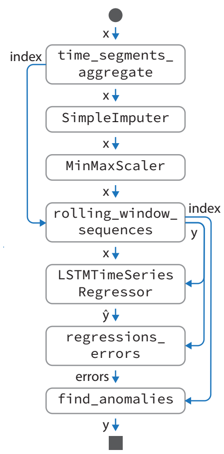
New unsupervised time series anomaly detection models are constantly being developed. This poses the question: *how do we uniformly represent these models?*

To accomplish this goal, we standardize models. The anomaly detection process starts with a signal $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T\}$ where T is the length of the time series and $\mathbf{x}_i \in \mathbb{R}^n$ and n is the number of channels. When the time series is a univariate signal then $n = 1$. The goal is to find a set of anomalous intervals $A = \{(t_s^1, t_e^1), \dots, (t_s^k, t_e^k)\}$ where $k \geq 0$. Each interval represents the start and end *timestamps* of the detected anomaly.

We adopt a universal representation of *primitives* and *pipelines* (Smith et al., 2020). *Primitives* are reusable basic block components that perform a single operation. Primitives can be single tasks, and range from data scaling to signal processing to model training. When primitives are stacked together, they compose *pipelines*. A pipeline is computed into its respective computational graph, similar to the `LSTM DT` pipeline and its primitives shown in Figure 3a, where the input is a uni- or multi-variate time series, and the output is a list of intervals of the detected anomalies. As portrayed in Figure 3b, we use the `fit` method to train the model and the `detect` method to run inference. With this standardization, we are able to treat all models equivalently. Primitives provide a code-efficient structure such that we can be modular and re-use primitives between pipelines. Moreover, it allows researchers to conduct ablation studies in order to attribute pipeline performance and the contribution of primitives.

2.2. Standardizing Hyperparameter Settings

Deep learning models require setting a multitude of hyperparameters, some of which are model-specific. This has made it more challenging to keep benchmarks fair and transparent. In *OrionBench*, hyperparameters are stored as `json` files to expose configurations in both machine- and human-



(a) Graph representation

```python
from orion import Orion
from orion.data import
    ↪ load_signal

train = load_signal(
    ↪ 'S-1-train')
test = load_signal('S-1-test')

orion = Orion(
    pipeline='lstm_dt'
)

# train pipeline
orion.fit(train)

# detect anomalies
anomalies = orion.detect(test)
```

(b) Usage in `python`

```
hyperparameters = {
    'time_segments_aggregate':
        ↪ {
        'interval': 21600
    },
    'rolling_window_sequences':
        ↪ {
        'target_column': 0,
        'window_size': 250
    },
    'LSTMTimeSeriesRegressor':
        ↪ {
        'epochs': 35
    },
    'find_anomalies': {
        'window_size_perc':
            ↪ 30,
        'fixed_threshold':
            ↪ false
    }
}
```

(c) Hyperparameter config

Figure 3. Example of `LSMT DT` pipeline. (a) Graph representation of the pipeline showcasing its primitives and data flow. (b) `python` usage example. (c) Subset of hyperparameter configuration in `json` format of the pipeline.

readable representations. Figure 3c is an example of the hyperparameter settings for `LSTM DT`.

To increase benchmark fairness, we standardize hyperparameters for both *global* and *local* hyperparameters. **Global** hyperparameters are shared between pipelines. They typically pertain to pre- and post- processing primitives. For example, in Figure 3c, `interval` is a global hyperparameter that denotes the aggregation level for the signal – here it is set to 6 hours of aggregation (21,600 seconds). Such hyperparameters are selected based on the characteristics of the dataset, and in some cases are dynamic. For example, `window_size_perc` sets the window size to be 30% of the signal length. **Local** hyperparameters such as `epochs` are pipeline-specific and are selected based on the authors' recommendation in the original paper. These hyperparameters are consistent across datasets per pipeline in order to alleviate any bias introduced by knowing the ground truth anomalies of the dataset.

2.3. Integrating New Pipelines and Datasets

A main pillar of open-source development is continuously maintaining and updating a library. Benchmark libraries are no different. For a library to grow, it is essential to keep introducing new pipelines and datasets to benefit the end-user.

ML researchers build new primitives and compose new pipelines easily in *OrionBench*. The framework provides templates to help guide researchers in this process. Moreover, ML researchers can utilize primitives in other packages given a corresponding `json` representation. It is often the case that pre- and post- processing primitives are reusable across pipelines (Alnegheimish et al., 2022). *OrionBench* first started with 2 pipelines, and now has 12 pipelines. The same applies to benchmark datasets. To make the data more accessible, we host publicly available datasets on an Amazon S3 instance. Signals can be loaded via `load_signal` command (as shown in Figure 3b) that will directly connect to S3 if the data is hosted there. Otherwise, it will search for the file locally. This enables users to also load their own private custom data for benchmarks.

2.4. Verifying Pipelines

We organize pipelines into *verified* pipelines and *sandbox* pipelines. When a new pipeline is proposed, it is categorized under "sandbox" until several tests and validations are made. The ML researcher opens a new pull request and is requested to pass unit and integration tests before the pipeline is merged and stored in the sandbox. Next, Orion developers test the new pipeline and verify its performance and reproducibility. One of the most commonly encountered situations is a mismatch between the researchers' comparison report and the results an Orion developer would get from running the same framework. A very common reason for this was that researchers had failed to update a hyperparameter setting. Once these checks are made, pipelines are transferred from "sandbox" to "verified". The increased reliability of verified pipelines enhances the end-user's confidence in adopting pipelines.

2.5. Releasing Regularly

The last requirement for an end-user-friendly benchmarking framework like *OrionBench* is to keep track of how benchmark results change over time. Most pipelines are stochastic in nature, meaning benchmark results can change from run to run. Moreover, when the underlying dependency packages (e.g. `TensorFlow`) introduce new versions, benchmark results can be affected or even compromised. Therefore, it is crucial to monitor the pipeline performances over time and prevent possible breakdowns due to backwards incompatibility.

This need is a main driver behind the creation of *OrionBench*. Benchmarking was introduced as a measure of *stability* and *reproducibility* testing, analogous to how Continuous Integration Continuous Deployment (CI/CD) tests have greatly increased the reliability of open-source libraries. *OrionBench* now serves as a test of pipeline stability over time. As of now, 17 releases have been published, and the *leaderboard* changes with each release (see Section 3.2).

2.6. Benefiting the End-User

OrionBench is available to the end-user on pypi, where they can install *OrionBench* through "pip install Orion". Then all verified pipelines are at the finger tips of the end-user where they train a pipeline and detect anomalies using `fit` and `detect` APIs, respectively. End-Users have access to a collection of models that they trust to perform as expected, fits their computational needs, and are continuously maintained and benchmarked.

3. Evaluation

We demonstrate the use of *OrionBench* on 12 pipelines ranging from classic to generative models and 14 datasets. We also lay out how benchmarking works as a mechanism to test pipeline stability. Moreover, we present two real-world scenarios in which *OrionBench* was used to ground unsupervised anomaly detection.

Datasets. Currently, the benchmark is executed on 14 datasets with ground truth anomalies. These datasets are gathered from different sources, including **NASA** [2], **NAB** [3], **UCR** [4], and **Yahoo S5** [5]. Collectively, these datasets contain 742 time series and 2,599 anomalies. The properties of each dataset, including the number of signals and anomalies, the average length of signal, and the average length of anomalies are presented in Table 2. The table makes clear how properties differ between datasets; for instance, NASA, NAB, and UCR contain anomalies that are longer than those in Yahoo S5, and the majority of anomalies in Yahoo S5's A3 & A4 datasets are point anomalies.

Table 2. Datasets Summary. There are 14 datasets with varying number of signals and anomalies. The table presents the average signal length and anomaly length for each dataset. All these datasets are publicly accessible.

	Dataset	# Signals	# Anomalies	Avg. Signal
NASA	MSL	27	36	4890.59
	SMAP	53	67	10618.86
NAB	Art	6	6	4032.00
	AWS	17	30	3980.35
	AdEx	5	11	1593.40
	Traf	7	14	2237.71
	Tweets	10	33	15863.1
Yahoo S5	A1	67	178	1415.9
	A2	100	200	1421.0
	A3	100	939	1680.0
	A4	100	835	1680.0
UCR	Natural	142	142	99973.33
	Distorted	92	92	49218.02
	Noise	16	16	39343.38
Total		742	2599	

Models. As of the writing of this paper, *OrionBench* includes 12 pipelines: `ARIMA` – Autoregressive Integrated Moving Average statistical model (Box & Jenkins, 1968); `MP` – Discord discovery through Matrix Profiling (Yeh et al., 2016); `AER` – AutoEncoder with Regression deep learning model with reconstruction and prediction errors (Wong et al., 2022); `LSTM-DT` – LSTM non-parametric Dynamic Threshold with two LSTM layers (Hundman et al., 2018); `TadGAN` – Time series Anomaly Detection using Generative Adversarial Networks (Geiger et al., 2020); `LSTM VAE` – Variational AutoEncoder with LSTM layers (Park et al., 2018); `LSTM AE` – AutoEncoder with LSTM layers (Malhotra et al., 2016); `Dense AE` – Similar to `LSTM AE`, with Dense layers (Sakurada & Yairi, 2014); `LNN` – Liquid Neural Network model, a variant of Liquid Time-Constant Networks (Hasani et al., 2021); `GANF` – Graph Augmented Normalizing Flows density-based model (Dai & Chen, 2022); `AT` – AnomalyTransformer model with association discrepancy (Xu et al., 2022); `Azure AD` – Microsoft Azure Anomaly Detection service (Ren et al., 2019).

Hyperparameters. Hyperparameter settings are an important part of model performance. As highlighted in Section 2.2, *OrionBench* seeks to provide a fair benchmark by standardizing hyperparameters. Moreover, we sort pipelines based on whether they are prediction-based or reconstruction-based (Alnegheimish et al., 2022) and set the hyperparameters based on those properties. The precise values are selected based on the configurations proposed

[2]https://github.com/khundman/telemanom
[3]https://github.com/numenta/NAB
[4]https://www.cs.ucr.edu/~eamonn/time_series_data_2018
[5]https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70

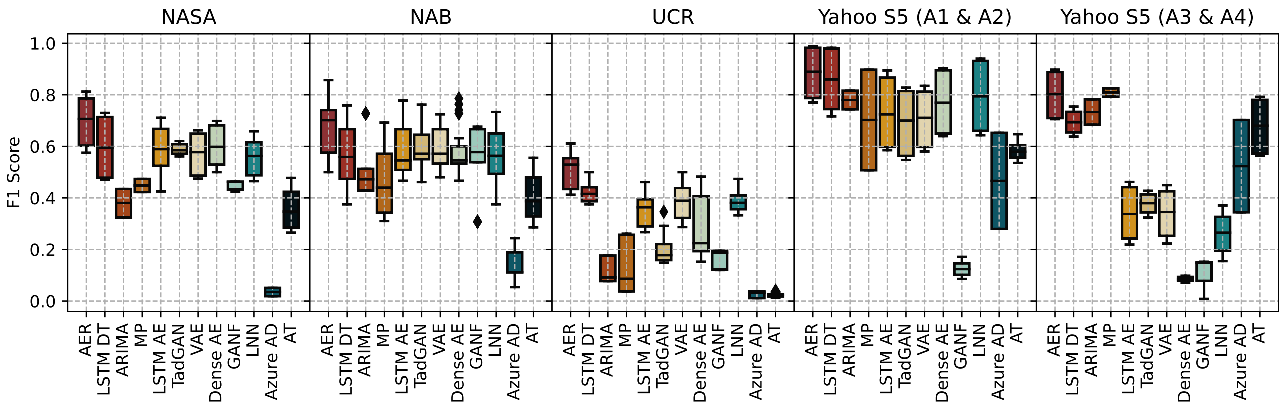


Figure 4. Distribution of F1 Scores across NASA, NAB, Yahoo S5, and UCR. Yahoo S5 was split into two subsets highlighting the F1 difference pipelines experience when detecting point anomalies.

```python
from orion.benchmark import benchmark

pipelines = [ 'arima', 'lstm_dt', 'aer', '...']
datasets = ['NAB', 'NASA', '...']
metrics = ['f1', 'precision', '...']

benchmark(pipelines=pipelines, datasets=datasets,
    ↪ metrics=metrics, rank='f1')
```

Figure 5. Benchmark command in `Python`. Running `benchmark()` with default settings will execute the benchmark on all pipelines and datasets currently integrated.

by the original authors in previous work (Hundman et al., 2018; Wong et al., 2022; Geiger et al., 2020; Dai & Chen, 2022). For example, reconstruction-based pipelines tend to have a smaller `window_size` compared to prediction-based pipelines given that reconstructing large segments is difficult. For example, prediction-based pipelines have a `window_size` of 250 data points, while reconstruction-based pipelines have a smaller `window_size` of 100, because the objective is to reconstruct the entire window rather than predict a few steps ahead. While some other methods alter the `window_size` based on the signal length (Malhotra et al., 2016), we provide the option to make these hyperparameters dynamic. For example, `window_size` can be set as 10% of the entire signal length.

Settings. We use *OrionBench* version `0.5.2`. The benchmark is executed to run for 5 iterations over all the pipelines and datasets.

Compute.. We set up an instance on the MIT SuperCloud (Reuther et al., 2018) with an Intel Xeon Gold 6249 processor of 10 CPU cores (9 GB RAM per core) and one NVIDIA Volta V100 GPU.

3.1. End-to-End Benchmark

The script in Figure 5 illustrates the few lines of code required for execution.

Benchmark Usage. *OrionBench* is available to all users through a single command, as illustrated in Figure 5. Users specify the list of pipelines, datasets, and metrics they are interested in and pass them to the `benchmark` function. The output result is stored as a detailed `.csv` file that signals performance metrics for each pipeline, such as accuracy, precision, recall, and F1 score. It also shows the status of the run – whether it was successful or not, total execution time, and the runtime for each internal primitive.

Qualitative Performance. Figure 4 depicts the F1 score obtained for each dataset on average. The score achieved by each pipeline differs based on the dataset and its properties. We can see that `AER` is the highest-performing pipeline overall. Another interesting observation is that `LSTM AE`, `TadGAN`, `VAE`, and `Dense AE` are not effective at detecting point anomalies. These pipelines are all reconstruction-based and are susceptible to anomalous regions when computing the deviation between the original and reconstructed signal, producing anomaly scores with reduced peaks at these points. Anomalies thus pass by undetected (Wong et al., 2022). This is clearly demonstrated in the Yahoo S5 datasets, where F1 scores for A3 & A4 datasets are low compared to those for A1 & A2. Furthermore, the `Azure AD` pipeline frequently flags segments as anomalous. This strategy works for datasets with a lot of anomalies, such as Yahoo S5. We therefore notice an increased F1 score there compared to other datasets.

Leaderboard. Table 3 compares the performance of each pipeline and reports the number of datasets for which it outperforms `ARIMA`. Out of 14 datasets, `AER` is outperforming `ARIMA` for 13, while `Azure AD` is performing worse for all datasets. Since the benchmark was executed for 5 iterations, we report the median result. However, pipelines are stochastic and might perform differently between runs. Therefore, end-users are left to wonder whether the rankings provided in the leaderboard are robust and trustworthy.

Table 3. Leaderboard showing number of datasets in which each pipeline outperformed `ARIMA`.

Pipeline	Outperform: `ARIMA, 1970` (Box & Jenkins, 1968)
`AER, 2022`	13
`LSTM DT, 2018`	10
`LSTM AE, 2016`	9
`TadGAN, 2020`	9
`VAE, 2018`	9
`Dense AE, 2014`	9
`LNN, 2021`	9
`GANF, 2022`	8
`MP, 2016`	7
`AT, 2022`	2
`Azure AD, 2019`	0

Using Spearman's rank correlation, $\rho = 0.916$, we find that the best pipelines are consistent across runs. Similarly, pipelines at the lower end of the table are stable in their rankings. On the other hand, the middle part of the table is subject to change as `TadGAN`, `LSTM AE`, `VAE`, `LNN`, and `Dense AE` compete with one another. The exact ranking of each pipeline in all runs is shown in Table 5 in the Appendix.

Computational Performance. In addition to quality performance, end-users are interested in pipelines' computational performance. Figure 6 illustrates how much time (in minutes) on average each pipeline needs depending on the signal length. Elapsed time includes the time it takes to train a pipeline and time it takes to run inference. The shortest signal in all datasets contains 750 data points, while the longest one contains 900,000 data points. On shorter signals, pipelines typically take seconds, while longer signals may take minutes or even hours to complete. The most time-consuming pipeline is `LNN` and in second place is `TadGAN`, which has more neural networks to train than other pipelines. On the longest signal in the dataset, it takes `LNN` and `TadGAN` approximately 5 and 2 hours total elapsed time respectively. Moreover, inference-only pipelines, such as `Azure AD`, are computationally fast and almost invariant to the length of the signal. some pipelines can become more demanding when the signal length increases such as `AT` where the runtime increased by a factor of $7\times$ and $8\times$ respectively.

Given the complexity of this model, end-users might want to select an alternative pipeline. Moreover, a user might sacrifice quality performance for computational efficiency, or vice versa. Individual end users can make their own decisions when weighing these tradeoffs.

3.2. Progression of Benchmarks

Stability. As we see in Figure 4, `AER` is the highest-performing pipeline: *Was this always the case? OrionBench* publishes benchmark results with every package release.

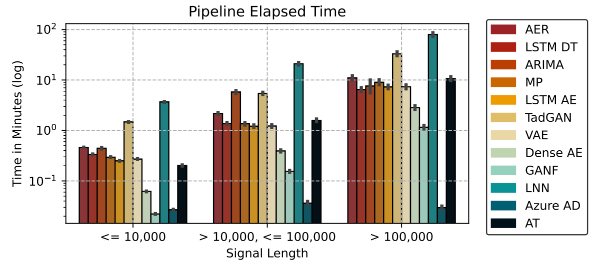


Figure 6. Average elapsed time of pipelines across different signal length groups.

Figure 8 depicts the average F1 score of four pipelines. These pipelines are chosen to show: the best-performing pipeline (`AER`), the worst-performing pipeline (`Azure AD`), the first implemented pipeline (`LSTM DT`), and the classic pipeline (`ARIMA`), that are currently available in our framework. The observed performance can change from one release to another for a number of reasons, including the stochastic nature of pipelines, internal changes in dependency packages, and dynamic thresholding. If we look closely at Figure 8, we notice three shifts (viewed as slopes) to `LSTM DT`, and only two to `AER`, `ARIMA`, and `Azure AD`.

First, in version update `0.1.3` $\rightarrow$ `0.1.4`, we saw a drop in F1 score due to an internal change in how we calculate the overall scores. The aggregation calculation became automated and was conducted on the dataset level rather than the signal level. Second, in version update `0.1.5` $\rightarrow$ `0.1.6`, there was an increase in performance that can be traced back to our hyperparameter setting modifications. Third, going from version `0.3.2` $\rightarrow$ `0.4.0` shifted our implementation from `TensorFlow` version 1 to 2, which impacted the underlying implementation. Lastly, after introducing a new dataset, namely UCR, we noticed a drop in the overall performance by pipelines in `0.5.0` $\rightarrow$ `0.5.1` because this was a more difficult dataset. Overall, the observed changes were minimal and could be traced back to alterations within our framework.

Pipeline Integration. Figure 7 showcases exactly when each pipeline was integrated to *OrionBench*. The first benchmark release, version `0.1.3`, in September 2020, featured only 2 pipelines. Over time, new models have been developed and integrated. As of today, *OrionBench* has 12 verified pipelines, ranging from classical models to deep learning and made by 5 different contributors.

3.3. OrionBench in Action

As anomaly detection models continue to be developed, *OrionBench* allows researchers and end-users to understand and compare these models. In this subsection, we walk through two real-world scenarios where benchmarking was useful for: (1) guiding researchers to develop a new model

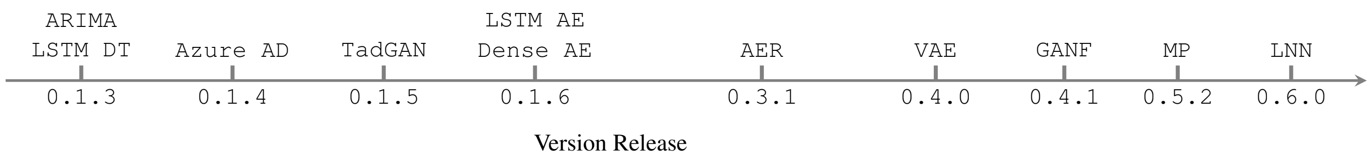


Figure 7. Timeline of pipeline introduction to the benchmark. *OrionBench* started with 2 pipelines; over the course of three years, 8 more pipelines were introduced at different stages.

Table 4. Comparison of anomaly detection benchmarks. A (✓) indicates the framework includes an attribute, while an (✗) indicates the attribute is absent. #Datasets and #Pipelines columns represent the number of currently available datasets and pipelines respectively. Columns under "Pipeline Type" represent whether certain pipeline types are supported including classic pipelines such as ARIMA, Deep Learning (DL) pipelines such as LSTM, and BlackBox (BBox) pipelines that are called externally through an API such as Azure's AD service. Columns under "Properties" represent whether a benchmark has certain properties, including custom evaluation methods for time series anomaly detection; Whether the benchmark is extensible and can integrate new datasets and pipelines, and If the benchmark is being released in periodic fashion with an updated leaderboard. The last two columns illustrate the last time a leaderboard has been published and where.

	Available		Pipeline Type			Properties			Published Leaderboard	
Framework	# Datasets	# Pipelines	Classic	DL	BBox	Evaluation	Extensible	Periodic	Last Update	Source
`Numenta` (Lavin & Ahmad, 2015)	7	4	✓	✗	✗	✓	✓	✗	Jun 2018	Github
`TSB-UAD` (Paparrizos et al., 2022)	18	12	✓	✓	✗	✓	✗	✗	Nov 2022	Github
`TODS` (Lai et al., 2021)	4	9	✓	✓	✗	✗	✗	✗	Dec 2021	Paper
`TimeEval` (Wenig et al.)	23	71	✓	✓	✗	✗	✓	✗	Aug 2022	Paper
`Exathlon` (Jacob et al., 2020)	10	3	✗	✓	✗	✓	✗	✗	Sep 2021	Paper
`Merlion` (Bhatnagar et al., 2021)	12	12	✓	✓	✗	✓	✗	✗	Sep 2021	Paper
OrionBench	14	12	✓	✓	✓	✓	✓	✓	Oct 2024	Github

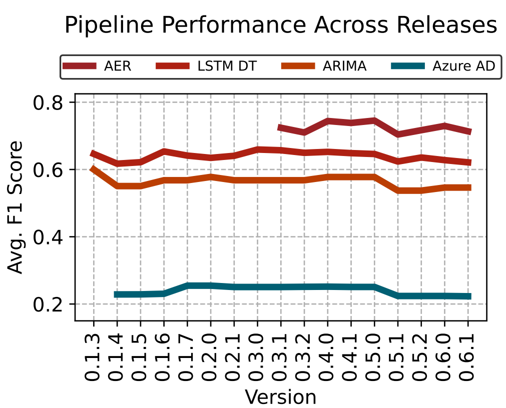


Figure 8. Monitoring pipelines' performance across all formal releases of *OrionBench*.

for unsupervised time series anomaly detection; (2) providing end-users with an existing SOTA model. We show that *OrionBench* is a commodity benchmarking framework.

Before walking through the aforementioned scenarios, we would like to describe the state of OrionBench. `LSTM DT` (Hundman et al., 2018) and `TadGAN` (Geiger et al., 2020) (which was developed by the Orion team) performed competitively against each other until version `0.3.1`, when `AER` (Wong et al., 2022) was introduced. Below, we illustrate the story behind the `AER` model and how we, the OrionBench developers, helped benchmark their model.

Scenario 1 – OrionBench guided a researcher to focus in the right direction. Researchers are eager to adopt the latest innovations in deep learning. An independent researcher was keen on introducing the attention mechanism to anomaly detection (Vaswani et al., 2017). While the model was promising in local experiments, to assure its performance we decided to run it through *OrionBench*. Unfortunately, the model could not do better than either `LSTM DT` or `TadGAN`. This reoriented the project and led to an investigation of the successes and limitations of pipelines. Subsequently, it led to a deep understanding of where prediction models prevailed compared to reconstruction models and vice versa. *OrionBench* helped guide this process by cross-referencing model performance with dataset properties. The conclusion was that prediction-based anomaly scores are better at capturing point anomalies than reconstruction-based anomaly scores. Moreover, reconstruction-based anomaly scores are better at capturing longer anomalies. Wong et al. (2022) uncovered more associations related to anomaly scores and error methods. The outcome of this investigation ultimately resulted in the `AER` model and is now the best-performing pipeline on *OrionBench*.

Scenario 2 – OrionBench enabled the addition of a latest model and provided an end-user with confidence in other models. We had been working with an end-user from a renowned satellite company for over four years when they approached us with interest in a new SOTA model. The model was GANF, (Dai & Chen, 2022) which had been featured in a news article [6] that caught their attention. New models are published that claim SOTA performance, beating existing models on their benchmarks. This is common for renowned companies that invest in creating high-performing models. The end-user wanted to know: *Should we adopt this model?* Several issues can prevent such models from living up to their promised performance in industrial and operational settings. Real-world datasets are inherently more complex than pristine benchmark datasets. Furthermore, authors often fine-tune a model to the benchmark datasets, neglecting others and causing their model to underperform on unseen datasets. *OrionBench*, as an independent benchmark, can help determine whether it makes sense to adopt a new model. We integrated GANF into *OrionBench*. As presented earlier in Figure 4, it was only competitive on the NAB dataset. However, due to the seamless integration of the pipelines into *OrionBench*, the end-user was still able to apply the pipeline to their own data and obtained valuable results. This emphasizes that the behavior of models differs from one dataset to another, and there is no one-pipeline-fits-all.

Similarly, LNN models (Hasani et al., 2021) have been utilized in a variety of applications, including robot control. A published news article[7] suggests that these models are able to perform any time series task. To test their ability to perform unsupervised anomaly detection, we implemented an LTC primitive and, shortly after, the LNN pipeline. Hasani et al. (2021) released an accompanying pip installable library, which has made creating the LNN pipeline straightforward. It took one week from its first commit to when it merged on the main branch and became sandbox-available. *OrionBench* has made it easier for us to incorporate new models and assess their anomaly detection capabilities.

4. Related Work

In this section, we walk-through some of the algorithms on unsupervised time series anomaly detection as well as benchmarking systems.

4.1. Unsupervised Time Series Anomaly Detection Algorithms

Many anomaly detection methods have emerged in the past few years (Chandola et al.; Blázquez-García et al., 2021; Goldstein & Uchida, 2016). These include statistical thresholding techniques (Patcha & Park, 2007), clustering-based methods (Münz et al., 2007; Syarif et al., 2012; Agrawal & Agrawal, 2015), and machine learning models (Hasan et al., 2019; Liu et al.). More recently, deep learning models have become popular and have been adopted for anomaly detection (Chalapathy & Chawla, 2019; Pang et al., 2021). Deep learning-based anomaly detection models for time series data rely mostly on unsupervised learning, because in most settings, there is no a priori knowledge of anomalous events. Malhotra et al. (2016) built an autoencoder with Long Short-Term Memory (LSTM) layers (Hochreiter & Schmidhuber, 1997) that learns to reconstruct 'normal' signal behavior. It uses the residual between the reconstructed signal and the original signal to locate anomalies. LSTM networks are practical at capturing the temporal dynamics in time series data. Hundman et al. (2018) used an LSTM forecasting model to predict the signal and paired it with a non-parametric threshold to mitigate false positives. Since then, generative models including variational autoencoders (VAE) (Park et al., 2018), Generative Adversarial Networks (GAN) (Geiger et al., 2020), and Transformers (Xu et al., 2022) have been adopted for unsupervised anomaly detection.

4.2. Time Series Anomaly Detection Benchmarks

There are several notable open-source time series benchmarking systems featuring unsupervised time series anomaly detection methods (Lavin & Ahmad, 2015; Paparrizos et al., 2022; Lai et al., 2021; Wenig et al.; Jacob et al., 2020; Bhatnagar et al., 2021). Table 4 highlights the key features present in each framework. While these benchmarks do exist, they are not directed towards end-users, and are usually not kept up-to-date. Table 4 shows the latest published results for each time series anomaly detection benchmark framework, whether the scoreboard has been updated on github, and the date of the latest update. Usually, these assessments are done once, during the production of related papers. We argue that this makes them *point-in-time* benchmarking frameworks.

In addition to unsupervised pipelines, some frameworks include supervised pipelines (Paparrizos et al., 2022; Wenig et al.). However, comparing supervised pipelines to unsupervised ones can be misleading, as labels are not available in most real-world scenarios. We address three key points with *OrionBench*: (1) time series anomaly detection requires careful consideration of how to evaluate pipelines; (2) integration of new pipelines and datasets needs to be seamless such that pipelines are available to end-users, (3) benchmarks need periodic releases and leaderboard updates to ensure results are trusted and pipelines are stable.

There are many other benchmarking frameworks, such as time series forecasting benchmarks (Alexandrov et al., 2020; Bauer et al., 2021; Taieb et al., 2012), and anomaly detection for tabular data (Campos et al., 2016; Han et al., 2022). However, these benchmarks inherently differ from our un-

[6] https://news.mit.edu/2022/artificial-intelligence-anomalies-data-0225

[7] https://news.mit.edu/2021/machine-learning-adapts-0128

supervised anomaly detection benchmark for time series data.

5. Conclusion

We present *OrionBench*– a continuous end-to-end benchmarking framework for unsupervised time series anomaly detection. The benchmark is open-source and publicly available: `https://github.com/sintel-dev/Orion`. As of today, the benchmark holds 28 primitives, 12 pipelines, 14 public datasets, and 2 custom evaluation metrics. We present the qualitative and computational performance of pipelines across all datasets. Moreover, we showcase results our benchmark has accumulated since 2020, highlighting its value for providing continuous evaluations that demonstrate the extensibility and stability of pipelines.

Although the benchmark compares different pipelines, there is no one pipeline will be the best choice for every dataset. Pipeline selection is still a challenging process that highly correlates with the characteristics of the dataset at hand and the type of anomalies present in the dataset. In our future work, we would like to focus on the suitability of pipelines and finding the relationship between various attributes of the input data and the efficacy of anomaly detection. Moreover, we invite ML researchers to contribute to *OrionBench*.

Acknowledgements

This research was supported by SES S.A., and Iberdrola and ScottishPower Renewables.

Change Log

Updated the title; Table 1 has been updated and now gives more details. Figure 6 now shows runtime groupby by three different signal length bins; Added release 0.6.1 to the results.

References

Aggarwal, C. *An Introduction to Outlier Analysis*. Springer, 2017.

Agrawal, S. and Agrawal, J. Survey on anomaly detection using data mining techniques. *Procedia Computer Science*, 60:708–713, 2015.

Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., Maddix, D. C., Rangapuram, S., Salinas, D., Schulz, J., Stella, L., Türkmen, A. C., and Wang, Y. GluonTS: Probabilistic and neural time series modeling in python. *Journal of Machine Learning Research*, 21(116):1–6, 2020.

Alnegheimish, S. Orion – a machine learning framework for unsupervised time series anomaly detection. Master's thesis, MIT, 2022.

Alnegheimish, S., Liu, D., Sala, C., Berti-Equille, L., and Veeramachaneni, K. Sintel: A machine learning framework to extract insights from signals. In *SIGMOD International Conference on Management of Data*, pp. 1855–1865, 2022.

Bauer, A., Züfle, M., Eismann, S., Grohmann, J., Herbst, N., and Kounev, S. Libra: A benchmark for time series forecasting methods. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, pp. 189–200, 2021.

Bhatnagar, A., Kassianik, P., Liu, C., Lan, T., Yang, W., Cassius, R., Sahoo, D., Arpit, D., Subramanian, S., Woo, G., et al. Merlion: A machine learning library for time series. *arXiv preprint arXiv:2109.09265*, 2021.

Blázquez-García, A., Conde, A., Mori, U., and Lozano, J. A. A review on outlier/anomaly detection in time series data. *ACM Computing Surveys*, 54(3):1–33, 2021.

Box, G. E. and Jenkins, G. M. Some recent advances in forecasting and control. *Journal of the Royal Statistical Society*, 17(2):91–109, 1968.

Campos, G. O., Zimek, A., Sander, J., Campello, R. J., Micenková, B., Schubert, E., Assent, I., and Houle, M. E. On the evaluation of unsupervised outlier detection: Measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 30:891–927, 2016.

Chalapathy, R. and Chawla, S. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3): 1–58.

Coleman, C., Kang, D., Narayanan, D., Nardi, L., Zhao, T., Zhang, J., Bailis, P., Olukotun, K., Ré, C., and Zaharia, M. Analysis of DAWNBench, a time-to-accuracy machine learning performance benchmark. *ACM SIGOPS Operating Systems Review*, 53(1):14–25, 2019.

Dai, E. and Chen, J. Graph-augmented normalizing flows for anomaly detection of multiple time series. In *International Conference on Learning Representations*, 2022.

Geiger, A., Liu, D., Alnegheimish, S., Cuesta-Infante, A., and Veeramachaneni, K. TadGAN: Time series anomaly detection using generative adversarial networks. In *IEEE International Conference on Big Data*, pp. 33–43, 2020.

Goldstein, M. and Uchida, S. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS One*, 11(4), 2016.

Han, S., Hu, X., Huang, H., Jiang, M., and Zhao, Y. ADBench: Anomaly detection benchmark. In *Advances in Neural Information Processing Systems*, volume 35, pp. 32142–32159, 2022.

Hasan, M., Islam, M. M., Zarif, M. I. I., and Hashem, M. Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches. *Internet of Things*, 7:100059, 2019.

Hasani, R., Lechner, M., Amini, A., Rus, D., and Grosu, R. Liquid time-constant networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7657–7666, 2021.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hundman, K., Constantinou, V., Laporte, C., Colwell, I., and Soderstrom, T. Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 387–395, 2018.

Jacob, V., Song, F., Stiegler, A., Rad, B., Diao, Y., and Tatbul, N. Exathlon: A benchmark for explainable anomaly detection over time series. *arXiv preprint arXiv:2010.05073*, 2020.

Lai, K.-H., Zha, D., Xu, J., Zhao, Y., Wang, G., and Hu, X. Revisiting time series outlier detection: Definitions and benchmarks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Lavin, A. and Ahmad, S. Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. In *2015 IEEE 14th International Conference on Machine Learning and Applications*, pp. 38–44, 2015.

Liu, D., Zhao, Y., Xu, H., Sun, Y., Pei, D., Luo, J., Jing, X., and Feng, M. Opprentice: Towards practical and automatic anomaly detection through machine learning. In *Proceedings of the 2015 Internet Measurement Conference*, pp. 211–224.

Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., and Shroff, G. LSTM-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, 2016.

Münz, G., Li, S., and Carle, G. Traffic anomaly detection using k-means clustering. In *GI/ITG-Workshop MMBnet*, volume 7, 2007.

Pang, G., Shen, C., Cao, L., and Hengel, A. V. D. Deep learning for anomaly detection: A review. *ACM Computing Surveys*, 54(2):1–38, 2021.

Paparrizos, J., Kang, Y., Boniol, P., Tsay, R. S., Palpanas, T., and Franklin, M. J. TSB-UAD: An end-to-end benchmark suite for univariate time-series anomaly detection. *Proceedings of the VLDB Endowment*, 15(8):1697–1711, 2022.

Park, D., Hoshi, Y., and Kemp, C. C. A multimodal anomaly detector for robot-assisted feeding using an LSTM-based variational autoencoder. *IEEE Robotics and Automation Letters*, 3(3):1544–1551, 2018.

Patcha, A. and Park, J.-M. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer Networks*, 51(12):3448–3470, 2007.

Pena, E. H., de Assis, M. V., and Proença, M. L. Anomaly detection using forecasting methods ARIMA and HWDS. In *2013 32nd International Conference of the Chilean Computer Science Society (sccc)*, pp. 63–66. IEEE, 2013.

Ren, H., Xu, B., Wang, Y., Yi, C., Huang, C., Kou, X., Xing, T., Yang, M., Tong, J., and Zhang, Q. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3009–3017, 2019.

Reuther, A., Kepner, J., Byun, C., Samsi, S., Arcand, W., Bestor, D., Bergeron, B., Gadepally, V., Houle, M., Hubbell, M., et al. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance Extreme Computing Conference*, pp. 1–6, 2018.

Sakurada, M. and Yairi, T. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the 2nd Workshop on Machine Learning for Sensory Data Analysis*, pp. 4–11, 2014. ISBN 9781450331593. doi: 10.1145/2689746.2689747.

Smith, M. J., Sala, C., Kanter, J. M., and Veeramachaneni, K. The Machine Learning Bazaar: Harnessing the ML ecosystem for effective system development. In *SIGMOD International Conference on Management of Data*, pp. 785–800, 2020.

Syarif, I., Prugel-Bennett, A., and Wills, G. Unsupervised clustering approach for network anomaly detection. In *Networked Digital Technologies*, pp. 135–145, 2012.

Taieb, S. B., Bontempi, G., Atiya, A. F., and Sorjamaa, A. A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition. *Expert Systems with Applications*, 39(8): 7067–7083, 2012.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

Wenig, P., Schmidl, S., and Papenbrock, T. TimeEval: A benchmarking toolkit for time series anomaly detection algorithms. *Proceedings of the VLDB Endowment*, 15 (12):3678–3681.

Wong, L., Liu, D., Berti-Equille, L., Alnegheimish, S., and Veeramachaneni, K. AER: Auto-encoder with regression for time series anomaly detection. In *2022 IEEE International Conference on Big Data*, pp. 1152–1161, 2022.

Xu, J., Wu, H., Wang, J., and Long, M. Anomaly Transformer: Time series anomaly detection with association discrepancy. In *International Conference on Learning Representations*, 2022.

Yeh, M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H. A., Silva, D. F., Mueen, A., and Keogh, E. Matrix Profile I: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th International Conference on Data Mining*, pp. 1317–1322, 2016.

Appendix

A. Limitations

We acknowledge several limitations of our framework and results. Black box pipelines such as Microsoft's `Azure AD` service lack certain levels of transparency. In Figure 8, we noticed that `Azure AD` improved in `0.1.6` → `0.1.7`. However, we have no knowledge on what has caused this improvement. Unlike other pipelines where we can cross reference change of behaviour to code modification or even updated dependency package releases. Nevertheless, we are still able to monitor the performance of black box pipelines and having some confidence in their stability.

In addition, benchmarks are notorious for requiring massive computing resources, and in this case it is no different. While the models can vary in usage, to perform a comprehensive benchmark, we utilize MIT supercloud (Reuther et al., 2018). When computing resources are limited, on-demand benchmark runs become difficult. We aim to alleviate this challenge with continuous and periodic running benchmarks. Moreover, with every introduction to a new model, a benchmark must be run to add the model to the leaderboard.

Lastly, and most importantly, there is no guarantee that these pipelines will deliver the same performance on real-world datasets. A clear example was demonstrated in Section 3.3 Scenario 2 where `GANF` produced valuable results for the end-user, however its results in the benchmark are not as promising as some other pipelines. This stresses the importance of pipeline selection based on the characteristics of the datasets and anomalies. Further research is needed to understand the suitability of unsupervised pipelines for a given dataset.

B. Primitives & Pipelines

B.1. Primitive Template

Abstractions in *OrionBench* of primitives and pipelines are universal representations of end-to-end models, from a signal to a set of detected anomalies. Compared to standard `scikit-learn` like code, it requires one additional step of creating `json` files to define these primitives. Figure 9 showcases a template that helps contributors to guides their own primitive.

Once primitives are built, they can be stacked to create a pipeline similar to the example shown in Figure 10. The example shows the `json` file representation of `LSTM DT` pipeline.

B.2. Pipelines

Currently in *OrionBench*, there are 9 readily available pipelines. They are all unsupervised pipelines. All pipelines and their hyperparameter settings for the benchmark can be explored directly: `https://github.com/sintel-dev/Orion/tree/master/orion/pipelines/verified`. Below we provide further detail on the mechanisms behind each pipeline.

`ARIMA` (Pena et al., 2013). `ARIMA` is an autoregressive integrated moving average model which is a classic statistical analysis model. It is a forecasting model that learns autocorrelations in the time series to predict future values prediction. Since then it has been adapted for anomaly detection. The pipeline computes the prediction error between the original signal and the forecasting one using simple point-wise error. Then it pinpoints where the anomalies are based one when the error exceeds a certain threshold. Particularly, `ARIMA` pipeline uses a moving window based thresholding technique defined in `find_anomalies` primitive.

`AER` (Wong et al., 2022). `AER` is an autoencoder with regression pipeline. It combines prediction and reconstruction models simultaneously. More specifically, it produces bi-directional predictions (forward & backward) while reconstructing the original time series at the same time by optimizing a joint objective function. The error is then computed as a point-wise error for both forward and backward predictions. As for reconstruction, dynamic time warping is used, which computes the euclidean distance between two time series where one might lag behind another. The total error is then computed as a point-wise product between the three aforementioned errors.

`LSTM DT` (Hundman et al., 2018). `LSTM DT` is a prediction-based pipeline using an LSTM model. Similar to `ARIMA`, it computes the residual between the original signal and predicted one using smoothed point-wise error. Then they apply a non-parametric thresholding method to reduce the amount of false positives.

`TadGAN` (Geiger et al., 2020). `TadGAN` is a reconstruction pipeline that uses generative adversarial networks to generate

```json
{
    "name": "orion.primitives.primitive.PrimitiveName",
    "contributors": ["Author <email>"],
    "documentation": "reference to documentation or paper if available.",
    "description": "short description.",
    "classifiers": {
        "type": "postprocessor",
        "subtype": "anomaly_detector"
    },
    "modalities": [],
    "primitive": "orion.primitives.primitive.PrimitiveName",
    "fit": {
        "method": "fit",
        "args": [
            {
                "name": "X",
                "type": "ndarray"
            },
            {
                "name": "y",
                "type": "ndarray"
            }
        ]
    },
    "produce": {
        "method": "detect",
        "args": [
            {
                "name": "X",
                "type": "ndarray"
            },
            {
                "name": "y",
                "type": "ndarray"
            }
        ],
        "output": [
            {
                "name": "y",
                "type": "ndarray"
            }
        ]
    },
    "hyperparameters": {
        "fixed": {
            "hyper_name": {
                "type": "str",
                "default": "value"
            }
        }
    }
}
```

Figure 9. Primitive template. The first section of the `json` describes metadata, the second part contains functional information including the names of the methods and their arguments, the third part defines the hyperparameters of the primitive.

```
{
    "primitives": [
        "mlstars.custom.timeseries_preprocessing.time_segments_aggregate",
        "sklearn.impute.SimpleImputer",
        "sklearn.preprocessing.MinMaxScaler",
        "mlstars.custom.timeseries_preprocessing.rolling_window_sequences",
        "keras.Sequential.LSTMTimeSeriesRegressor",
        "orion.primitives.timeseries_errors.regression_errors",
        "orion.primitives.timeseries_anomalies.find_anomalies"
    ],
    "init_params": {
        "mlstars.custom.timeseries_preprocessing.time_segments_aggregate#1": {
            "time_column": "timestamp",
            "interval": 21600,
            "method": "mean"
        },
        "sklearn.preprocessing.MinMaxScaler#1": {
            "feature_range": [
                -1,
                1
            ]
        },
        "mlstars.custom.timeseries_preprocessing.rolling_window_sequences#1": {
            "target_column": 0,
            "window_size": 250
        },
        "keras.Sequential.LSTMTimeSeriesRegressor": {
            "epochs": 35
        },
        "orion.primitives.timeseries_anomalies.find_anomalies#1": {
            "window_size_perc": 30,
            "fixed_threshold": false
        }
    }
}
```

Figure 10. `LSTM DT` pipeline example. This is the content present in the `json` file of the pipeline. The first section defines the stack of primitives used in the pipeline which will be computed to the graph shown previously in Figure 3a. The `init` argument initializes some of the hyperparameters for each primitive. This is a detailed version with full primitive names of the hyperparameters shown in Figure 3c.

a synthetic time series. To sample a "similar" time series, the model uses an encoder to map the original time series to the latent dimension. There are three possible strategies to compute the errors between the real and synthetic time series. Specifically, point-wise errors, area difference, and dynamic time warping. Most datasets are set to dynamic time warping (dtw) as error.

`MP` (Yeh et al., 2016). `MP` is a matrix profile method that seeks to find discords in time series. The pipeline computes the matrix profile of a signal, which essential provides the closes nearest neighbor for each segment. Based on these values, segments with large distance values to their nearest neighbors are anomalous. We use `find_anomalies` to set the threshold dynamically.

`VAE` (Park et al., 2018). `VAE` is a variational autoencoder consisting of an encoder and a decoder with LSTM layers. Similar to previous pipelines, it adopts reconstruction errors to compute the deviation between the original and reconstructed signal.

`LSTM AE` (Malhotra et al., 2016). `LSTM AE` is an autoencoder with an LSTM encoder and decoder. This is a simpler variant of `VAE`. It also uses reconstruction errors to measure the difference between the original and reconstruction signal.

`Dense AE`. `Dense AE` is an autoencoder where its properties are exactly similar to that of `LSTM AE` with the exception of the encoder and decoder layers.

`GANF` (Dai & Chen, 2022). `GANF` is density-based methods where they use normalizing flows to learn the distribution of the data with a graph structure to overcome the challenge of high dimensionality. The model outputs an *anomaly measure* that indicates where the anomalies might be. To convert the output into a list of intervals, we add `find_anomalies` primitive.

`Azure AD` (Ren et al., 2019). `Azure AD` is a black box pipeline which connects to Microsoft's anomaly detection service [8]. To use this pipeline, the user needs to have a subscription to the service. Then the user can update the `subscription_key` and `endpoint` in the pipeline `json` for usage.

`AnomTransformer (AT)` (Xu et al., 2022). `AnomTransformer` is a transformer based model using a new *anomaly-attention* mechanism to compute the association discrepancy. The model amplifies the discrepancies between normal and abnormal time points using a minimax strategy. The threshold is set based on the attention values.

C. Data

C.1. Data Format

Time series is a collection of data points that are indexed by time. There are many forms in which time series can be stored, we define a time series as a set of time points, which we represent through integers denoting *timestamps*, and a corresponding set of values observed at each respective timestamp. Note that no prior pre-processing is required as all pre-processing steps are part of the pipeline, e.g. imputations, scaling, etc.

C.2. Dataset Details

The benchmark currently features 11 publicly accessible datasets from different sources. Table 2 illustrates some of the datasets' properties. Below, we provide more detailed description for each dataset.

`NASA`. This dataset is a spacecraft telemetry signals provided by NASA. It was originally released in 2018 as part of the LSTM-DT paper (Hundman et al., 2018) and can be accessed directly from `https://github.com/khundman/telemanom`. It features two datasets: Mars Science Laboratory (MSL) and Soil Moisture Active Passive (SMAP). MSL contains 27 signals with 36 anomalies. SMAP contains 53 signals with 69 anomalies. In total, NASA datasets has 80 signals with 105 anomalies. This dataset was pre-split into training and testing partitions. In our benchmark, we train the pipeline using the training data, and apply detection to only the testing data.

`NAB`. Part of the Numenta benchmark (Lavin & Ahmad, 2015) is the NAB dataset `https://github.com/numenta/NAB`. This datasets includes multiple types of time series data from various applications and domains In our benchmark we selected five sub-datasets (name: # signals, # anomalies): artWithAnomaly (`Art`: 6, 6): this dataset was artificially generated; realAWSCloudwatch (`AWS`: 17, 20): this dataset contains AWS server metrics collected by AmazonCloudwatch service such as CPU Utilization; realAdExchange (`AdEx`: 5, 11), this dataset contains online advertisement clicking rate

[8] `https://azure.microsoft.com/en-us/products/cognitive-services/anomaly-detector/`

metrics such as cost-per-click; realTraffic (`Traf`: 7, 14): this dataset contains real time traffic metrics from the Twin Cities Metro area in Minnesota such as occupancy, speed, etc; and realTweets (`Tweets`: 10, 33): this dataset contains metrics of a collection of Twitter mentions of companies (e.g. Google) such as number of mentions each 5 minutes.

`Yahoo S5`. This dataset contains four different sub-datasets. A1 dataset is based on real production traffic of Yahoo computing systems with 67 signals and 179 anomalies. On the other hand, A2, A3 and A4 are all synthetic datasets with 100 signals each and 200, 939, and 835 anomalies respectively. There are many anomalies in this dataset with over 2,153 in 367 signals, averaging 5.8 anomalies in each signal. Most of the anomalies in A3 and A4 are short and last for only a few points in time. Data can be requested from Yahoo's website https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70. In our benchmark, we train and apply detection to the same entire signal.

`UCR`. This dataset was released in a SIGKDD competition in 2021 https://www.cs.ucr.edu/~eamonn/time_series_data_2018/UCR_TimeSeriesAnomalyDatasets2021.zip. It contains 250 signals with only one anomaly in each signal. The anomalies themselves were artificially introduced to the signal. More specifically, in many times they are synthetic anomalies, or a consequence of flipping/smoothing/interpolating/reversing/prolonging normal segments to create anomalies. The dataset was created to have more challenging cases of anomalies.

D. Evaluation

This section provides further details on our evaluation setup and obtained results. Code to reproduce Figures and Tables are provided https://github.com/sarahmish/orionbench-paper

D.1. Evaluation Setup

Results presented in Section 3 are reported based on version `0.5.0` of Orion which is also released on pip [9]. We recommend setting up a new python environment before installing Orion. Currently the library is supported in `python 3.6`, `3.7`, and `3.8`.

Evaluation Strategy. Measuring the performance of unsupervised time series anomaly detection pipelines is more nuanced than the usual classification metrics. *OrionBench* compares detected anomalies with ground truth labels according to well-defined metrics. This can be done using either weighted segment or overlapping segment (Alnegheimish et al., 2022). For our evaluation in this paper, we use *overlapping segment* exclusively. Using overlapping segment, for each experiment run (which is an evaluation of one pipeline on one signal), we record the number of true positives (TP), false positive (FP), and false negative (FN) obtained. Because anomalies are scarce and in many signals only one anomaly exists (or none), in many cases precision and recall scores will be undefined on a signal level. Therefore, we compute the scores on a dataset level.

$$precision = \frac{\sum_{s \in \mathcal{S}} TP_s}{\sum_{s \in \mathcal{S}} TP_s + FP_s} \qquad recall = \frac{\sum_{s \in \mathcal{S}} TP_s}{\sum_{s \in \mathcal{S}} TP_s + FN_s}$$

For a given dataset with a set of signals $\mathcal{S}$, we compute the total true positives, false positives, and false negatives within every signal in that set. Then we compute the score for each pipeline according to the metric of interest whether it is precision, recall, or f1 score. The computation of f1 score is standard from precision and recall ($f1 = 2 \times \frac{precision \times recall}{precision + recall}$).

Recorded Information. During the benchmark process, information regarding performance, computation, diagnostics, etc. gets recorded. Below we list all the information we store for each experiment. An experiment is defined as a single pipeline trained on a single time series then used for detection for the same time series.

- *dataset:* the dataset that the signal belongs to, e.g. `SMAP`.
- *pipeline:* the name of the pipeline, e.g. `AER`.
- *signal:* the name of the signal, e.g. `S-1`.
- *iteration:* each experiment can be run for k iterations.

[9] https://pypi.org/project/orion-ml/0.5.0/

- *f1, precision, recall:* the evaluated metrics, in many cases it is undefined.
- *tn, fp, fn, tp:* the evaluated number of true negatives, false positives, false negatives, true positives respectively. In overlapping segment approach, *tn* does not have a value given the nature of evaluation.
- *status:* whether or not the experiment ran from beginning to end without issue.
- *elapsed:* how much runtime each experiment took (includes training and inference).
- *run_id:* the process identification number.

The benchmark results are saved as `.csv` files and stored directly in the Github repositories: https://github.com/sintel-dev/Orion/tree/master/benchmark/results. Moreover, the pipelines used in each experiment are saved for reproduciblity measures. Due to their large size, we store these pipelines on a local server. However, part of our endeavour is to make these pipelines public as well such that they can be used and inspected.

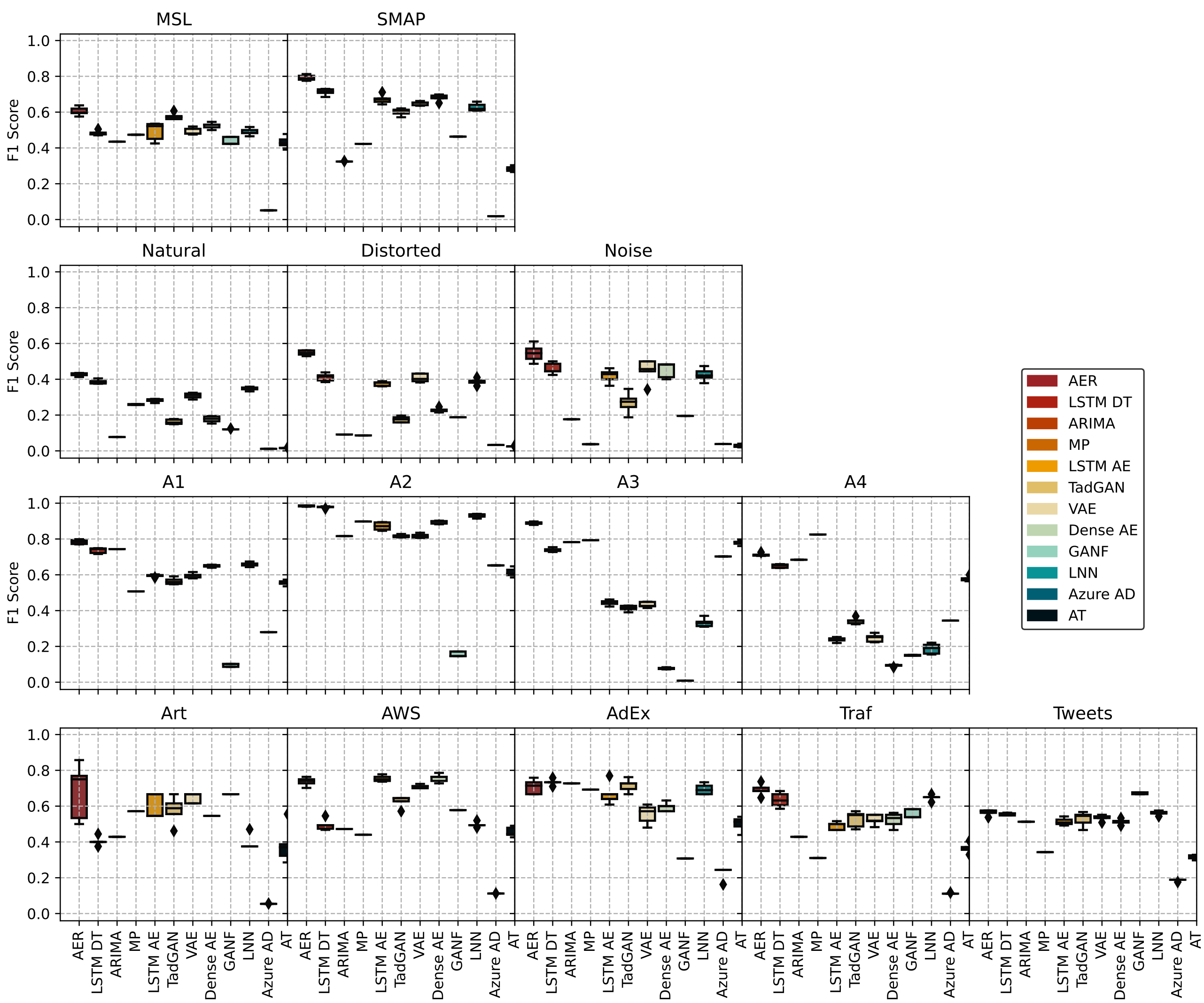


Figure 11. Distribution of F1 Score per Dataset. This Figure is a detailed version of Figure 4 where now every dataset is shown separately. On average, `AER` is the highest scoring pipeline for most datasets with the exception of AWS, AdEx, and Tweets. The performance of pipelines changes from one dataset to another, indicating there does not exists a single pipeline that will work perfectly for all datasets. One of the insights we find is how point anomalies in A3 & A4 present a challenge for reconstruction-based pipelines such as `LSTM AE`, `TadGAN`, `VAE`, and `Dense AE`.

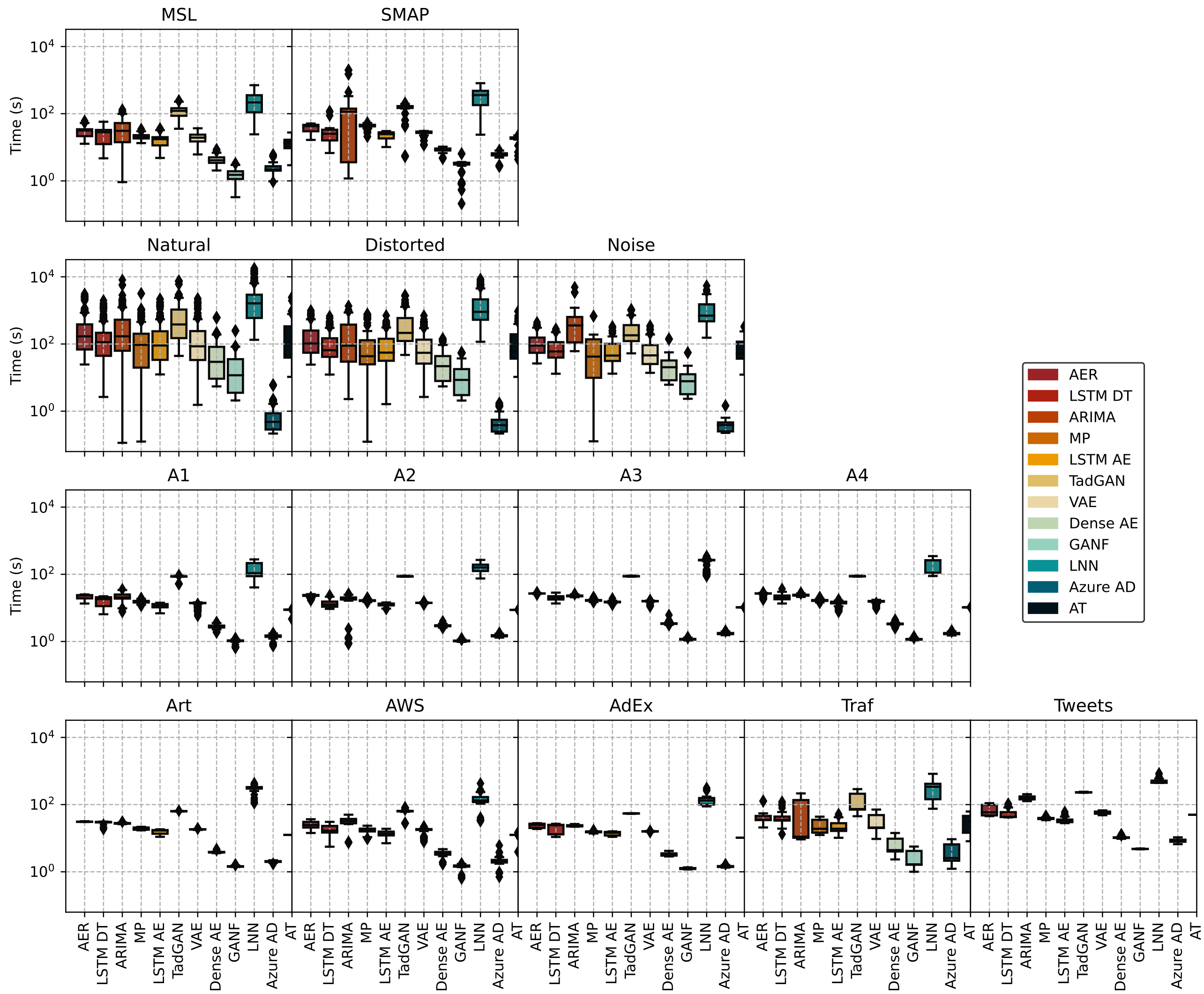


Figure 12. Runtime (in seconds) per dataset. Runtime is recorded as the wall time it takes to train a pipeline using `fit` and run inference using `detect`. The most time consuming pipeline is `TadGAN`.

D.2. Benchmark Results

Figure 11 illustrates the F1 score obtained per dataset. Observing the average values per dataset, `AER` seems to score the highest on most datasets. However, other pipelines such as `LSTM DT` are comparable or outperform `AER` in certain cases. Each pipeline has its strengths and the performance varies from one dataset to another.

Pipeline scalability is an important aspect to address for many end-users. The reported wall time of each pipeline per dataset is shown in Figure 12. `TadGAN` takes minutes to run while other pipelines seem to finish in several seconds. The fastest pipelines are `GANF` and `Azure AD`. `Azure AD` is an inference only pipeline, and `GANF` is fast to train.

D.3. Leaderboard

Table 5 shows the rank of each pipeline in 5 different benchmark runs. The rank is calculated from the order of the leaderboard (as shown in Table 3). If two pipelines have the same number of wins, the average F1 score is used as a tie-breaker.

Table 5. Rank of pipelines in five independent runs.

	Run				
Pipeline	#1	#2	#3	#4	#5
`AER`	1	1	1	1	1
`LSTM DT`	3	2	2	2	2
`LSTM AE`	2	5	3	4	4
`TadGAN`	6	7	5	5	6
`VAE`	4	3	4	7	5
`Dense AE`	7	4	6	6	7
`LNN`	5	6	7	3	3
`GANF`	8	8	8	8	8
`MP`	9	9	9	9	9
`AT`	10	10	10	10	10
`Azure AD`	11	11	11	11	11

D.4. Computational Cost Across Releases

In addition to quality stability shown in Figure 8, we can monitor the runtime execution of the benchmark. We illustrate the average runtime for each pipeline across 15 releases in Figure 13. There is a clear improvement in average runtime in release `0.2.1`. This increase in speed traces back to an internal change of the API's code. More specifically, pipelines builds were adjusted to only build once to reduce overhead during the `fit` and `detect` process. Looking back at the development plan, this is reflected in Issue #261 where we see exact alterations made to the code.

Moreover, in version `0.4.0`, the package migrated to `TensorFlow 2.0` which consequently made the pipelines faster in GPU mode. However, in version `0.4.1` the pipelines were executed without GPU, which is evident by the slight increase in runtime.

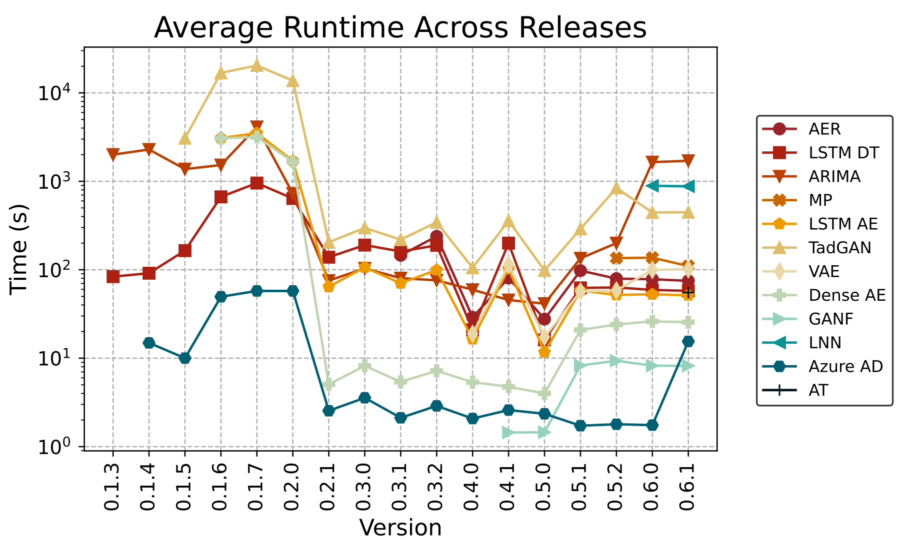


Figure 13. Average runtimes (in seconds) across Orion versions. Deep learning pipelines with LSTM layers are more sporadic across releases due to the availability of GPU. Pipelines such as `Azure AD` are black box pipelines that run inference alone making it fast. In version `0.2.1`, we migrated our benchmark to MIT Supercloud.

D.5. Qualitative Performance Across Releases

The detailed sheets of benchmark runs are stored directly in the repository: `https://github.com/sintel-dev/Orion/tree/master/benchmark/results`. The following tables report the F1 score, precision, and recall metrics for each pipeline across all releases as of today.

- version `0.6.1`, pipelines 12, datasets 12, release date: October 4th 2024.
- version `0.6.0`, pipelines 11, datasets 12, release date: February 13th 2024.
- version `0.5.2`, pipelines 10, datasets 12, release date: October 19th 2023.
- version `0.5.1`, pipelines 9, datasets 12, release date: August 16th 2023.
- version `0.5.0`, pipelines 9, datasets 11, release date: May 23rd 2023.
- version `0.4.1`, pipelines 9, datasets 11, release date: January 31st 2023.
- version `0.4.0`, pipelines 8, datasets 11, release date: November 10th 2022.
- version `0.3.2`, pipelines 7, datasets 11, release date: July 4th 2022.
- version `0.3.1`, pipelines 7, datasets 11, release date: April 26th 2022.
- version `0.3.0`, pipelines 6, datasets 11, release date: March 31st 2022.
- version `0.2.1`, pipelines 6, datasets 11, release date: February 18th 2022.
- version `0.2.0`, pipelines 6, datasets 11, release date: October 11th 2021.
- version `0.1.7`, pipelines 6, datasets 11, release date: May 4th 2021.
- version `0.1.6`, pipelines 6, datasets 11, release date: March 8th 2021.
- version `0.1.5`, pipelines 4, datasets 11, release date: December 25th 2020.
- version `0.1.4`, pipelines 3, datasets 11, release date: October 16th 2020.
- version `0.1.3`, pipelines 2, datasets 11, release date: September 29th 2020.

Pipeline	NASA		UCR			Yahoo S5				NAB				
	MSL	SMAP	Natural	Distorted	Noise	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score														
AER	0.605±0.02	0.792±0.02	0.427±0.01	0.547±0.01	0.546±0.05	0.784±0.01	0.984±0.00	0.889±0.01	0.712±0.01	0.682±0.16	0.737±0.02	0.708±0.04	0.691±0.03	0.565±0.02
LSTM DT	0.483±0.01	0.713±0.02	0.386±0.01	0.411±0.02	0.460±0.03	0.736±0.02	0.978±0.00	0.738±0.01	0.649±0.01	0.404±0.03	0.495±0.03	0.734±0.02	0.634±0.04	0.556±0.01
ARIMA	0.435±0.00	0.324±0.00	0.077±0.00	0.092±0.00	0.176±0.00	0.744±0.00	0.816±0.00	0.782±0.00	0.684±0.00	0.429±0.00	0.472±0.00	0.727±0.00	0.429±0.00	0.513±0.00
MP	0.474±0.00	0.423±0.00	0.259±0.00	0.086±0.00	0.037±0.00	0.507±0.00	0.897±0.00	0.793±0.00	0.825±0.00	0.571±0.00	0.440±0.00	0.692±0.00	0.310±0.00	0.343±0.00
LSTM AE	0.493±0.05	0.672±0.03	0.281±0.01	0.375±0.01	0.419±0.04	0.595±0.01	0.871±0.02	0.444±0.01	0.238±0.01	0.594±0.07	0.754±0.02	0.665±0.06	0.490±0.02	0.513±0.02
TadGAN	0.576±0.02	0.601±0.02	0.162±0.01	0.176±0.02	0.269±0.06	0.564±0.02	0.817±0.01	0.413±0.02	0.342±0.02	0.577±0.08	0.627±0.03	0.716±0.04	0.527±0.05	0.528±0.04
VAE	0.492±0.02	0.649±0.01	0.308±0.02	0.407±0.02	0.449±0.06	0.595±0.01	0.817±0.01	0.433±0.02	0.247±0.02	0.646±0.03	0.710±0.01	0.554±0.05	0.524±0.03	0.536±0.02
Dense AE	0.524±0.02	0.680±0.02	0.177±0.02	0.228±0.01	0.452±0.04	0.649±0.01	0.894±0.01	0.077±0.00	0.093±0.01	0.545±0.00	0.756±0.02	0.595±0.02	0.523±0.04	0.512±0.02
GANF	0.438±0.02	0.463±0.00	0.121±0.00	0.188±0.00	0.195±0.00	0.095±0.01	0.156±0.01	0.008±0.00	0.150±0.00	0.667±0.00	0.578±0.00	0.308±0.00	0.556±0.02	0.673±0.01
LNN	0.490±0.02	0.627±0.02	0.347±0.01	0.386±0.02	0.425±0.04	0.658±0.01	0.930±0.01	0.333±0.02	0.187±0.03	0.394±0.04	0.496±0.01	0.694±0.03	0.648±0.02	0.562±0.01
Azure AD	0.051±0.00	0.019±0.00	0.011±0.00	0.033±0.00	0.038±0.00	0.280±0.00	0.653±0.00	0.702±0.00	0.344±0.00	0.054±0.00	0.112±0.00	0.228±0.04	0.112±0.00	0.186±0.01
AT	0.431±0.03	0.283±0.02	0.016±0.00	0.026±0.00	0.028±0.01	0.555±0.01	0.613±0.02	0.779±0.01	0.577±0.01	0.384±0.10	0.456±0.03	0.501±0.04	0.365±0.03	0.315±0.01
Precision														
AER	0.601±0.05	0.832±0.02	0.338±0.01	0.478±0.02	0.502±0.05	0.809±0.02	0.983±0.01	0.992±0.00	0.927±0.01	0.602±0.13	0.804±0.04	0.566±0.04	0.562±0.02	0.572±0.03
LSTM DT	0.370±0.01	0.655±0.02	0.327±0.01	0.345±0.03	0.418±0.02	0.690±0.02	0.972±0.01	0.988±0.01	0.902±0.01	0.327±0.01	0.417±0.05	0.580±0.02	0.491±0.04	0.514±0.02
ARIMA	0.455±0.00	0.307±0.00	0.085±0.00	0.115±0.00	0.167±0.00	0.684±0.00	0.772±0.00	0.998±0.00	0.955±0.00	0.375±0.00	0.405±0.00	0.727±0.00	0.429±0.00	0.444±0.00
MP	0.346±0.00	0.291±0.00	0.217±0.00	0.047±0.00	0.019±0.00	0.448±0.00	0.824±0.00	0.952±0.00	0.946±0.00	0.500±0.00	0.314±0.00	0.600±0.00	0.205±0.00	0.324±0.00
LSTM AE	0.489±0.07	0.638±0.04	0.192±0.01	0.313±0.02	0.343±0.04	0.625±0.00	0.835±0.04	0.944±0.01	0.672±0.01	0.627±0.04	0.827±0.04	0.602±0.04	0.447±0.01	0.571±0.03
TadGAN	0.493±0.02	0.531±0.03	0.115±0.01	0.130±0.01	0.195±0.05	0.636±0.02	0.809±0.01	0.739±0.03	0.586±0.02	0.485±0.07	0.604±0.05	0.707±0.07	0.429±0.04	0.559±0.05
VAE	0.466±0.02	0.595±0.01	0.221±0.01	0.352±0.03	0.407±0.06	0.584±0.01	0.727±0.02	0.842±0.01	0.653±0.03	0.629±0.05	0.720±0.02	0.484±0.06	0.508±0.04	0.608±0.02
Dense AE	0.573±0.02	0.724±0.02	0.140±0.02	0.227±0.01	0.487±0.07	0.726±0.01	0.954±0.01	0.964±0.01	0.553±0.01	0.600±0.00	0.832±0.02	0.657±0.06	0.483±0.04	0.586±0.01
GANF	0.712±0.03	0.786±0.00	0.084±0.00	0.159±0.00	0.160±0.00	0.286±0.00	0.282±0.02	1.000±0.00	0.977±0.01	1.000±0.00	0.867±0.00	1.000±0.00	0.630±0.06	0.650±0.01
LNN	0.414±0.02	0.537±0.02	0.270±0.01	0.310±0.02	0.365±0.04	0.639±0.01	0.881±0.02	0.956±0.02	0.600±0.05	0.313±0.03	0.405±0.01	0.555±0.03	0.490±0.02	0.524±0.01
Azure AD	0.026±0.00	0.009±0.00	0.006±0.00	0.019±0.00	0.021±0.00	0.167±0.00	0.484±0.00	0.542±0.00	0.217±0.00	0.028±0.00	0.060±0.00	0.131±0.02	0.059±0.00	0.105±0.00
AT	0.282±0.03	0.170±0.01	0.008±0.00	0.013±0.00	0.014±0.00	0.497±0.01	0.494±0.03	0.85±0.01	0.758±0.01	0.253±0.10	0.325±0.02	0.352±0.03	0.229±0.02	0.188±0.01
Recall														
AER	0.611±0.02	0.755±0.01	0.577±0.01	0.641±0.02	0.600±0.06	0.761±0.01	0.985±0.00	0.805±0.01	0.578±0.01	0.800±0.22	0.680±0.02	0.945±0.05	0.900±0.08	0.558±0.02
LSTM DT	0.694±0.02	0.782±0.02	0.472±0.01	0.509±0.02	0.512±0.05	0.788±0.02	0.984±0.00	0.590±0.01	0.507±0.01	0.533±0.07	0.613±0.02	1.000±0.00	0.900±0.04	0.606±0.02
ARIMA	0.417±0.00	0.343±0.00	0.070±0.00	0.076±0.00	0.188±0.00	0.815±0.00	0.865±0.00	0.643±0.00	0.533±0.00	0.500±0.00	0.567±0.00	0.727±0.00	0.429±0.00	0.606±0.00
MP	0.750±0.00	0.776±0.00	0.320±0.00	0.500±0.00	0.625±0.00	0.584±0.00	0.985±0.00	0.679±0.00	0.732±0.00	0.667±0.00	0.733±0.00	0.818±0.00	0.643±0.00	0.364±0.00
LSTM AE	0.500±0.04	0.710±0.01	0.525±0.01	0.470±0.01	0.538±0.03	0.569±0.01	0.912±0.01	0.290±0.01	0.145±0.01	0.567±0.09	0.693±0.01	0.745±0.10	0.543±0.04	0.467±0.02
TadGAN	0.694±0.05	0.696±0.04	0.273±0.02	0.270±0.02	0.438±0.08	0.507±0.02	0.824±0.01	0.287±0.01	0.242±0.01	0.733±0.15	0.653±0.04	0.727±0.00	0.686±0.08	0.503±0.06
VAE	0.522±0.02	0.713±0.02	0.511±0.02	0.485±0.02	0.500±0.08	0.608±0.02	0.933±0.01	0.291±0.01	0.152±0.02	0.667±0.00	0.700±0.00	0.655±0.08	0.543±0.04	0.479±0.01
Dense AE	0.483±0.02	0.642±0.02	0.239±0.02	0.228±0.02	0.425±0.03	0.587±0.01	0.841±0.01	0.040±0.00	0.051±0.00	0.500±0.00	0.693±0.03	0.545±0.00	0.571±0.05	0.455±0.02
GANF	0.317±0.02	0.328±0.00	0.217±0.00	0.228±0.00	0.250±0.00	0.057±0.01	0.108±0.01	0.002±0.00	0.081±0.00	0.500±0.00	0.433±0.00	0.182±0.00	0.500±0.00	0.697±0.00
LNN	0.600±0.03	0.755±0.02	0.486±0.02	0.513±0.01	0.512±0.05	0.679±0.01	0.985±0.00	0.201±0.02	0.111±0.02	0.533±0.07	0.640±0.01	0.927±0.04	0.957±0.04	0.606±0.03
Azure AD	0.806±0.00	0.940±0.00	0.197±0.00	0.130±0.00	0.250±0.00	0.848±0.00	1.000±0.00	0.998±0.00	0.837±0.00	1.000±0.00	0.833±0.00	0.909±0.00	1.000±0.00	0.818±0.00
AT	0.928±0.03	0.854±0.02	0.690±0.03	0.750±0.03	0.762±0.05	0.628±0.02	0.809±0.02	0.718±0.02	0.465±0.02	0.867±0.07	0.767±0.04	0.873±0.05	0.914±0.03	0.958±0.05

Table 6. Benchmark Summary Results Version `0.6.1`

	NASA		UCR	Yahoo S5				NAB				
Pipeline	MSL	SMAP	UCR	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score						
`AER`	0.533	0.781	0.489	0.784	0.987	0.878	0.712	0.714	0.727	0.690	0.703	0.562
`LSTM DT`	0.466	0.694	0.390	0.735	0.980	0.743	0.637	0.400	0.507	0.714	0.585	0.603
`ARIMA`	0.525	0.411	0.153	0.728	0.856	0.797	0.686	0.308	0.382	0.727	0.467	0.514
`MP`	0.474	0.423	0.051	0.507	0.897	0.793	0.825	0.571	0.440	0.692	0.305	0.343
`LSTM AE`	0.462	0.662	0.330	0.600	0.864	0.444	0.247	0.667	0.737	0.500	0.467	0.508
`TadGAN`	0.568	0.610	0.177	0.593	0.805	0.377	0.308	0.667	0.667	0.696	0.516	0.531
`VAE`	0.538	0.627	0.354	0.570	0.809	0.427	0.244	0.667	0.712	0.480	0.483	0.508
`Dense AE`	0.493	0.688	0.204	0.644	0.891	0.084	0.086	0.545	0.778	0.600	0.563	0.533
`GANF`	0.462	0.463	0.143	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`LNN`	0.477	0.654	0.363	0.661	0.925	0.305	0.197	0.400	0.506	0.710	0.636	0.592
`Azure AD`	0.051	0.019	0.004	0.280	0.653	0.702	0.344	0.056	0.112	0.163	0.117	0.176
`AT`	0.449	0.303	0.021	0.583	0.617	0.772	0.576	0.385	0.423	0.474	0.338	0.315
						Precision						
`AER`	0.513	0.820	0.411	0.805	0.990	0.995	0.922	0.625	0.800	0.556	0.565	0.581
`LSTM DT`	0.358	0.638	0.329	0.690	0.975	0.988	0.896	0.333	0.439	0.588	0.444	0.550
`ARIMA`	0.477	0.303	0.144	0.670	0.769	0.998	0.955	0.286	0.342	0.727	0.438	0.486
`MP`	0.346	0.291	0.027	0.448	0.824	0.952	0.946	0.500	0.314	0.600	0.200	0.324
`LSTM AE`	0.429	0.627	0.242	0.630	0.814	0.941	0.681	0.667	0.778	0.462	0.438	0.577
`TadGAN`	0.481	0.540	0.130	0.629	0.777	0.716	0.531	0.556	0.636	0.667	0.471	0.548
`VAE`	0.599	0.558	0.271	0.556	0.709	0.856	0.640	0.667	0.724	0.429	0.467	0.577
`Dense AE`	0.515	0.741	0.171	0.725	0.949	0.976	0.534	0.600	0.875	0.667	0.500	0.593
`GANF`	0.750	0.786	0.105	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`LNN`	0.404	0.573	0.285	0.659	0.872	0.929	0.620	0.333	0.408	0.550	0.467	0.553
`Azure AD`	0.026	0.009	0.002	0.167	0.484	0.542	0.217	0.029	0.060	0.089	0.062	0.099
`AT`	0.297	0.193	0.011	0.517	0.498	0.855	0.747	0.250	0.297	0.333	0.206	0.189
						Recall						
`AER`	0.556	0.746	0.604	0.764	0.985	0.786	0.580	0.833	0.667	0.909	0.929	0.545
`LSTM DT`	0.667	0.761	0.480	0.787	0.985	0.595	0.495	0.500	0.600	0.909	0.857	0.667
`ARIMA`	0.583	0.642	0.164	0.798	0.965	0.663	0.535	0.333	0.433	0.727	0.500	0.545
`MP`	0.750	0.776	0.432	0.584	0.985	0.679	0.732	0.667	0.733	0.818	0.643	0.364
`LSTM AE`	0.500	0.701	0.520	0.573	0.920	0.291	0.151	0.667	0.700	0.545	0.500	0.455
`TadGAN`	0.694	0.701	0.276	0.562	0.835	0.256	0.217	0.833	0.700	0.727	0.571	0.515
`VAE`	0.583	0.716	0.512	0.584	0.940	0.284	0.151	0.667	0.700	0.545	0.500	0.455
`Dense AE`	0.472	0.642	0.252	0.579	0.840	0.044	0.047	0.500	0.700	0.545	0.643	0.485
`GANF`	0.333	0.328	0.224	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`LNN`	0.583	0.761	0.500	0.663	0.985	0.182	0.117	0.500	0.667	1.000	1.000	0.636
`Azure AD`	0.806	0.940	0.824	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818
`AT`	0.917	0.701	0.704	0.669	0.810	0.704	0.469	0.833	0.733	0.818	0.929	0.939

Table 7. Benchmark Summary Results Version `0.6.0`

	NASA		UCR	Yahoo S5				NAB				
Pipeline	MSL	SMAP	UCR	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score												
`AER`	0.587	0.819	0.476	0.799	0.987	0.892	0.709	0.714	0.741	0.690	0.703	0.638
`LSTM DT`	0.471	0.726	0.393	0.728	0.985	0.744	0.646	0.400	0.468	0.786	0.585	0.603
`ARIMA`	0.525	0.411	0.153	0.728	0.856	0.797	0.686	0.308	0.382	0.727	0.467	0.514
`MP`	0.474	0.423	0.051	0.507	0.897	0.793	0.825	0.571	0.440	0.692	0.305	0.343
`LSTM AE`	0.545	0.662	0.327	0.595	0.867	0.466	0.239	0.667	0.741	0.500	0.500	0.475
`TadGAN`	0.560	0.605	0.170	0.578	0.817	0.416	0.340	0.500	0.623	0.818	0.452	0.554
`VAE`	0.494	0.613	0.324	0.592	0.803	0.438	0.23	0.667	0.689	0.583	0.483	0.533
`Dense AE`	0.559	0.692	0.207	0.667	0.892	0.07	0.101	0.545	0.764	0.600	0.563	0.508
`GANF`	0.462	0.463	0.147	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`LNN`	0.517	0.618	0.362	0.652	0.938	0.331	0.191	0.375	0.481	0.714	0.667	0.575
`Azure AD`	0.051	0.019	0.015	0.280	0.653	0.702	0.344	0.056	0.112	0.163	0.117	0.176
Precision												
`AER`	0.564	0.867	0.395	0.830	0.990	0.993	0.917	0.625	0.833	0.556	0.565	0.611
`LSTM DT`	0.364	0.671	0.335	0.665	0.985	0.986	0.905	0.333	0.383	0.647	0.444	0.550
`ARIMA`	0.477	0.303	0.144	0.670	0.769	0.998	0.955	0.286	0.342	0.727	0.438	0.486
`MP`	0.346	0.291	0.027	0.448	0.824	0.952	0.946	0.500	0.314	0.600	0.200	0.324
`LSTM AE`	0.512	0.615	0.239	0.633	0.827	0.948	0.649	0.667	0.833	0.462	0.444	0.538
`TadGAN`	0.538	0.516	0.124	0.629	0.845	0.762	0.588	0.400	0.613	0.818	0.412	0.562
`VAE`	0.444	0.600	0.243	0.574	0.701	0.852	0.646	0.667	0.677	0.538	0.467	0.593
`Dense AE`	0.594	0.714	0.178	0.748	0.939	0.944	0.590	0.600	0.840	0.667	0.500	0.577
`GANF`	0.750	0.786	0.111	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`LNN`	0.434	0.520	0.288	0.632	0.895	0.954	0.590	0.300	0.388	0.588	0.500	0.525
`Azure AD`	0.026	0.009	0.008	0.167	0.484	0.542	0.217	0.029	0.06	0.089	0.062	0.099
Recall												
`AER`	0.611	0.776	0.600	0.770	0.985	0.809	0.578	0.833	0.667	0.909	0.929	0.667
`LSTM DT`	0.667	0.791	0.476	0.803	0.985	0.597	0.502	0.500	0.600	1.000	0.857	0.667
`ARIMA`	0.583	0.642	0.164	0.798	0.965	0.663	0.535	0.333	0.433	0.727	0.500	0.545
`MP`	0.750	0.776	0.432	0.584	0.985	0.679	0.732	0.667	0.733	0.818	0.643	0.364
`LSTM AE`	0.583	0.716	0.516	0.562	0.910	0.309	0.146	0.667	0.667	0.545	0.571	0.424
`TadGAN`	0.583	0.731	0.272	0.534	0.790	0.286	0.240	0.667	0.633	0.818	0.500	0.545
`VAE`	0.556	0.627	0.488	0.612	0.940	0.295	0.140	0.667	0.700	0.636	0.500	0.485
`Dense AE`	0.528	0.672	0.248	0.601	0.850	0.036	0.055	0.500	0.700	0.545	0.643	0.455
`GANF`	0.333	0.328	0.220	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`LNN`	0.639	0.761	0.488	0.674	0.985	0.200	0.114	0.500	0.633	0.909	1.000	0.636
`Azure AD`	0.806	0.940	0.176	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818

Table 8. Benchmark Summary Results Version `0.5.2`

	NASA		UCR	Yahoo S5				NAB				
Pipeline	MSL	SMAP	UCR	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score												
`AER`	0.587	0.775	0.474	0.780	0.987	0.869	0.686	0.769	0.750	0.733	0.611	0.581
`LSTM DT`	0.485	0.707	0.417	0.724	0.987	0.744	0.644	0.400	0.494	0.759	0.667	0.600
`ARIMA`	0.435	0.326	0.090	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`MP`	0.474	0.423	0.051	0.507	0.897	0.793	0.825	0.571	0.440	0.692	0.305	0.343
`LSTM AE`	0.479	0.662	0.332	0.619	0.874	0.460	0.227	0.667	0.750	0.615	0.471	0.533
`TadGAN`	0.568	0.590	0.173	0.552	0.806	0.408	0.321	0.571	0.603	0.583	0.529	0.606
`VAE`	0.486	0.649	0.339	0.556	0.817	0.415	0.236	0.462	0.737	0.538	0.483	0.533
`Dense AE`	0.537	0.641	0.202	0.640	0.885	0.078	0.102	0.545	0.800	0.632	0.500	0.508
`GANF`	0.462	0.463	0.147	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`Azure AD`	0.051	0.019	0.015	0.280	0.653	0.702	0.344	0.056	0.112	0.163	0.117	0.176
Precision												
`AER`	0.564	0.806	0.385	0.816	0.990	0.992	0.920	0.714	0.808	0.579	0.500	0.621
`LSTM DT`	0.373	0.650	0.352	0.680	0.990	0.988	0.892	0.333	0.404	0.611	0.545	0.568
`ARIMA`	0.455	0.311	0.102	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`MP`	0.346	0.291	0.027	0.448	0.824	0.952	0.946	0.500	0.314	0.600	0.200	0.324
`LSTM AE`	0.486	0.639	0.245	0.652	0.833	0.932	0.675	0.667	0.808	0.533	0.400	0.593
`TadGAN`	0.511	0.517	0.125	0.624	0.758	0.736	0.532	0.500	0.576	0.538	0.450	0.606
`VAE`	0.474	0.583	0.259	0.540	0.723	0.857	0.619	0.429	0.778	0.467	0.467	0.593
`Dense AE`	0.581	0.672	0.172	0.715	0.949	0.974	0.566	0.600	0.880	0.750	0.500	0.577
`GANF`	0.750	0.786	0.111	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`Azure AD`	0.026	0.009	0.008	0.167	0.484	0.542	0.217	0.029	0.060	0.089	0.062	0.099
Recall												
`AER`	0.611	0.746	0.616	0.747	0.985	0.773	0.547	0.833	0.700	1.000	0.786	0.545
`LSTM DT`	0.694	0.776	0.512	0.775	0.985	0.596	0.504	0.500	0.633	1.000	0.857	0.636
`ARIMA`	0.417	0.343	0.080	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`MP`	0.750	0.776	0.432	0.584	0.985	0.679	0.732	0.667	0.733	0.818	0.643	0.364
`LSTM AE`	0.472	0.687	0.512	0.590	0.920	0.306	0.137	0.667	0.700	0.727	0.571	0.485
`TadGAN`	0.639	0.687	0.280	0.494	0.860	0.282	0.230	0.667	0.633	0.636	0.643	0.606
`VAE`	0.500	0.731	0.492	0.573	0.940	0.274	0.146	0.500	0.700	0.636	0.500	0.485
`Dense AE`	0.500	0.612	0.244	0.579	0.830	0.040	0.056	0.500	0.733	0.545	0.500	0.455
`GANF`	0.333	0.328	0.220	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`Azure AD`	0.806	0.940	0.176	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818

Table 9. Benchmark Summary Results Version `0.5.1`

	NASA		UCR	Yahoo S5				NAB				
Pipeline	MSL	SMAP	UCR	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score												
`AER`	0.575	0.803	0.482	0.799	0.987	0.898	0.712	0.500	0.750	0.667	0.703	0.571
`LSTM DT`	0.471	0.730	0.393	0.743	0.980	0.734	0.639	0.400	0.494	0.710	0.632	0.560
`ARIMA`	0.435	0.326	0.090	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.533	0.658	0.311	0.593	0.852	0.452	0.252	0.545	0.737	0.667	0.500	0.542
`TadGAN`	0.571	0.592	0.172	0.547	0.809	0.427	0.324	0.667	0.645	0.727	0.486	0.552
`VAE`	0.475	0.653	0.340	0.599	0.806	0.424	0.227	0.667	0.700	0.609	0.516	0.542
`Dense AE`	0.500	0.692	0.199	0.656	0.902	0.080	0.094	0.545	0.764	0.600	0.467	0.508
`GANF`	0.462	0.463	0.147	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`Azure AD`	0.051	0.019	0.015	0.280	0.653	0.702	0.344	0.056	0.112	0.163	0.117	0.176
Precision												
`AER`	0.568	0.850	0.402	0.830	0.99	0.995	0.931	0.500	0.808	0.526	0.565	0.600
`LSTM DT`	0.364	0.667	0.332	0.696	0.975	0.979	0.898	0.333	0.404	0.550	0.500	0.500
`ARIMA`	0.455	0.311	0.102	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.513	0.608	0.224	0.629	0.802	0.946	0.690	0.600	0.778	0.615	0.444	0.615
`TadGAN`	0.473	0.529	0.124	0.621	0.803	0.736	0.569	0.556	0.625	0.727	0.391	0.640
`VAE`	0.432	0.600	0.256	0.582	0.714	0.834	0.639	0.667	0.700	0.583	0.471	0.615
`Dense AE`	0.571	0.714	0.166	0.739	0.955	0.975	0.558	0.600	0.840	0.667	0.438	0.577
`GANF`	0.750	0.786	0.111	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`Azure AD`	0.026	0.009	0.008	0.167	0.484	0.542	0.217	0.029	0.060	0.089	0.062	0.099
Recall												
`AER`	0.583	0.761	0.604	0.770	0.985	0.818	0.577	0.500	0.700	0.909	0.929	0.545
`LSTM DT`	0.667	0.806	0.484	0.798	0.985	0.587	0.496	0.500	0.633	1.000	0.857	0.636
`ARIMA`	0.417	0.343	0.080	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.556	0.716	0.508	0.562	0.910	0.297	0.154	0.500	0.700	0.727	0.571	0.485
`TadGAN`	0.722	0.672	0.280	0.489	0.815	0.300	0.226	0.833	0.667	0.727	0.643	0.485
`VAE`	0.528	0.716	0.504	0.618	0.925	0.284	0.138	0.667	0.700	0.636	0.571	0.485
`Dense AE`	0.444	0.672	0.248	0.590	0.855	0.042	0.051	0.500	0.700	0.545	0.500	0.455
`GANF`	0.333	0.328	0.220	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`Azure AD`	0.806	0.94	0.176	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818

Table 10. Benchmark Summary Results Version `0.5.0`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score					
`AER`	0.632	0.784	0.767	0.978	0.878	0.708	0.769	0.727	0.667	0.686	0.603
`LSTM DT`	0.481	0.708	0.726	0.973	0.740	0.638	0.400	0.474	0.759	0.649	0.560
`ARIMA`	0.435	0.324	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.514	0.686	0.600	0.888	0.443	0.227	0.667	0.800	0.522	0.500	0.483
`TadGAN`	0.482	0.573	0.612	0.818	0.371	0.327	0.615	0.645	0.538	0.512	0.551
`VAE`	0.538	0.635	0.556	0.803	0.457	0.257	0.545	0.700	0.593	0.571	0.567
`Dense AE`	0.545	0.683	0.629	0.877	0.087	0.098	0.545	0.741	0.632	0.571	0.500
`GANF`	0.462	0.463	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`Azure AD`	0.051	0.019	0.280	0.653	0.702	0.344	0.054	0.112	0.244	0.111	0.189
						Precision					
`AER`	0.600	0.845	0.795	0.970	0.991	0.923	0.714	0.800	0.526	0.571	0.633
`LSTM DT`	0.368	0.662	0.678	0.961	0.989	0.883	0.333	0.391	0.611	0.522	0.500
`ARIMA`	0.455	0.307	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.529	0.658	0.630	0.846	0.929	0.679	0.667	0.880	0.500	0.444	0.560
`TadGAN`	0.426	0.539	0.690	0.806	0.703	0.558	0.571	0.625	0.467	0.379	0.528
`VAE`	0.500	0.580	0.549	0.703	0.869	0.680	0.600	0.700	0.500	0.571	0.630
`Dense AE`	0.600	0.729	0.723	0.964	0.977	0.549	0.600	0.833	0.750	0.571	0.609
`GANF`	0.750	0.786	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`Azure AD`	0.026	0.009	0.167	0.484	0.542	0.217	0.028	0.060	0.141	0.059	0.107
						Recall					
`AER`	0.667	0.731	0.742	0.985	0.789	0.575	0.833	0.667	0.909	0.857	0.576
`LSTM DT`	0.694	0.761	0.781	0.985	0.591	0.499	0.500	0.600	1.000	0.857	0.636
`ARIMA`	0.417	0.343	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.716	0.573	0.935	0.291	0.137	0.667	0.733	0.545	0.571	0.424
`TadGAN`	0.556	0.612	0.551	0.830	0.252	0.231	0.667	0.667	0.636	0.786	0.576
`VAE`	0.583	0.701	0.562	0.935	0.310	0.158	0.500	0.700	0.727	0.571	0.515
`Dense AE`	0.500	0.642	0.556	0.805	0.046	0.054	0.500	0.667	0.545	0.571	0.424
`GANF`	0.333	0.328	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`Azure AD`	0.806	0.940	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818

Table 11. Benchmark Summary Results Version `0.4.1`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
					F1 Score						
`AER`	0.583	0.778	0.787	0.978	0.895	0.691	0.750	0.690	0.733	0.632	0.606
`LSTM DT`	0.457	0.707	0.743	0.980	0.748	0.651	0.421	0.474	0.733	0.649	0.571
`ARIMA`	0.435	0.324	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.493	0.662	0.588	0.885	0.446	0.237	0.667	0.712	0.615	0.552	0.492
`TadGAN`	0.543	0.620	0.558	0.828	0.428	0.321	0.571	0.585	0.583	0.516	0.559
`VAE`	0.533	0.634	0.575	0.833	0.444	0.230	0.545	0.689	0.615	0.483	0.533
`Dense AE`	0.545	0.683	0.646	0.902	0.082	0.075	0.545	0.755	0.600	0.581	0.483
`GANF`	0.462	0.463	0.086	0.171	0.008	0.152	0.667	0.578	0.308	0.583	0.667
`Azure AD`	0.051	0.019	0.280	0.653	0.702	0.344	0.054	0.112	0.244	0.111	0.189
					Precision						
`AER`	0.583	0.831	0.818	0.970	0.992	0.918	0.600	0.714	0.579	0.500	0.606
`LSTM DT`	0.348	0.639	0.691	0.975	0.986	0.901	0.308	0.391	0.579	0.522	0.500
`ARIMA`	0.455	0.307	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.486	0.605	0.623	0.849	0.939	0.658	0.667	0.724	0.533	0.533	0.536
`TadGAN`	0.489	0.538	0.631	0.826	0.766	0.560	0.500	0.543	0.538	0.471	0.543
`VAE`	0.513	0.590	0.561	0.742	0.855	0.639	0.600	0.677	0.533	0.467	0.593
`Dense AE`	0.600	0.750	0.722	0.960	0.952	0.507	0.600	0.870	0.667	0.529	0.560
`GANF`	0.750	0.786	0.281	0.300	1.000	0.986	1.000	0.867	1.000	0.700	0.639
`Azure AD`	0.026	0.009	0.167	0.484	0.542	0.217	0.028	0.060	0.141	0.059	0.107
					Recall						
`AER`	0.583	0.731	0.758	0.985	0.816	0.553	1.000	0.667	1.000	0.857	0.606
`LSTM DT`	0.667	0.791	0.803	0.985	0.603	0.510	0.667	0.600	1.000	0.857	0.667
`ARIMA`	0.417	0.343	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.731	0.556	0.925	0.293	0.145	0.667	0.700	0.727	0.571	0.455
`TadGAN`	0.611	0.731	0.500	0.830	0.297	0.225	0.667	0.633	0.636	0.571	0.576
`VAE`	0.556	0.687	0.590	0.950	0.300	0.140	0.500	0.700	0.727	0.500	0.485
`Dense AE`	0.500	0.627	0.584	0.850	0.043	0.041	0.500	0.667	0.545	0.643	0.424
`GANF`	0.333	0.328	0.051	0.120	0.004	0.083	0.500	0.433	0.182	0.500	0.697
`Azure AD`	0.806	0.940	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818

Table 12. Benchmark Summary Results Version `0.4.0`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score					
`AER`	0.579	0.770	0.793	0.978	0.888	0.721	0.800	0.727	0.690	0.667	0.571
`LSTM DT`	0.472	0.717	0.744	0.987	0.735	0.652	0.400	0.545	0.759	0.585	0.579
`ARIMA`	0.435	0.324	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.548	0.681	0.611	0.877	0.456	0.233	0.545	0.764	0.667	0.452	0.500
`TadGAN`	0.558	0.610	0.568	0.824	0.427	0.320	0.471	0.656	0.720	0.556	0.559
`VAE`	0.500	0.648	0.594	0.809	0.450	0.236	0.500	0.712	0.560	0.500	0.525
`Dense AE`	0.554	0.683	0.665	0.889	0.074	0.094	0.545	0.727	0.632	0.533	0.508
`Azure AD`	0.051	0.021	0.279	0.653	0.702	0.344	0.054	0.113	0.250	0.112	0.189
						Precision					
`AER`	0.550	0.855	0.812	0.970	0.996	0.930	0.667	0.800	0.556	0.545	0.600
`LSTM DT`	0.357	0.667	0.701	0.990	0.987	0.894	0.333	0.500	0.611	0.444	0.512
`ARIMA`	0.455	0.307	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.541	0.662	0.648	0.850	0.956	0.692	0.600	0.840	0.562	0.412	0.556
`TadGAN`	0.480	0.515	0.647	0.828	0.718	0.551	0.364	0.618	0.643	0.455	0.543
`VAE`	0.475	0.603	0.577	0.708	0.866	0.634	0.500	0.724	0.500	0.500	0.571
`Dense AE`	0.621	0.729	0.752	0.944	0.947	0.558	0.600	0.800	0.750	0.500	0.577
`Azure AD`	0.026	0.011	0.167	0.484	0.542	0.217	0.028	0.061	0.145	0.059	0.107
						Recall					
`AER`	0.611	0.701	0.775	0.985	0.802	0.589	1.000	0.667	0.909	0.857	0.545
`LSTM DT`	0.694	0.776	0.792	0.985	0.586	0.514	0.500	0.600	1.000	0.857	0.667
`ARIMA`	0.417	0.343	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.556	0.701	0.579	0.905	0.299	0.140	0.500	0.700	0.818	0.500	0.455
`TadGAN`	0.667	0.746	0.506	0.820	0.304	0.225	0.667	0.700	0.818	0.714	0.576
`VAE`	0.528	0.701	0.612	0.945	0.304	0.145	0.500	0.700	0.636	0.500	0.485
`Dense AE`	0.500	0.642	0.596	0.840	0.038	0.051	0.500	0.667	0.545	0.571	0.455
`Azure AD`	0.806	0.940	0.848	1.000	0.998	0.837	1.000	0.833	0.909	1.000	0.818

Table 13. Benchmark Summary Results Version `0.3.2`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
					F1 Score						
`AER`	0.600	0.785	0.745	0.985	0.881	0.721	0.471	0.727	0.733	0.600	0.562
`LSTM DT`	0.500	0.680	0.741	0.978	0.734	0.633	0.400	0.481	0.786	0.611	0.603
`ARIMA`	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.493	0.715	0.620	0.863	0.447	0.241	0.667	0.712	0.609	0.516	0.516
`TadGAN`	0.595	0.645	0.531	0.829	0.416	0.350	0.714	0.645	0.818	0.541	0.580
`Dense AE`	0.507	0.661	0.665	0.887	0.082	0.098	0.400	0.778	0.632	0.581	0.542
`Azure AD`	0.050	0.027	0.279	0.653	0.702	0.344	0.053	0.068	0.250	0.068	0.269
					Precision						
`AER`	0.618	0.810	0.760	0.985	0.983	0.907	0.364	0.800	0.579	0.462	0.581
`LSTM DT`	0.375	0.614	0.700	0.970	0.980	0.886	0.333	0.388	0.647	0.500	0.550
`ARIMA`	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.515	0.700	0.641	0.817	0.942	0.665	0.667	0.724	0.583	0.471	0.552
`TadGAN`	0.521	0.568	0.567	0.769	0.745	0.582	0.625	0.625	0.818	0.435	0.556
`Dense AE`	0.548	0.719	0.752	0.934	0.952	0.570	0.500	0.875	0.750	0.529	0.615
`Azure AD`	0.026	0.014	0.167	0.484	0.542	0.217	0.027	0.036	0.145	0.035	0.161
					Recall						
`AER`	0.583	0.761	0.730	0.985	0.798	0.598	0.667	0.667	1.000	0.857	0.545
`LSTM DT`	0.750	0.761	0.787	0.985	0.587	0.492	0.500	0.633	1.000	0.786	0.667
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.472	0.731	0.601	0.915	0.293	0.147	0.667	0.700	0.636	0.571	0.485
`TadGAN`	0.694	0.746	0.500	0.900	0.289	0.250	0.833	0.667	0.818	0.714	0.606
`Dense AE`	0.472	0.612	0.596	0.845	0.043	0.054	0.333	0.700	0.545	0.643	0.485
`Azure AD`	0.806	0.567	0.848	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818

Table 14. Benchmark Summary Results Version `0.3.1`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score					
`AER`	0.579	0.778	0.786	0.992	0.896	0.716	0.533	0.678	0.759	0.667	0.581
`LSTM DT`	0.486	0.703	0.752	0.985	0.743	0.635	0.400	0.545	0.733	0.667	0.580
`ARIMA`	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.500	0.690	0.625	0.867	0.446	0.238	0.667	0.764	0.583	0.500	0.500
`TadGAN`	0.512	0.658	0.566	0.858	0.422	0.331	0.714	0.625	0.750	0.588	0.559
`Dense AE`	0.554	0.661	0.650	0.874	0.087	0.090	0.545	0.778	0.526	0.516	0.464
`Azure AD`	0.050	0.020	0.279	0.653	0.702	0.344	0.053	0.068	0.250	0.068	0.269
						Precision					
`AER`	0.550	0.831	0.810	1.000	0.996	0.928	0.444	0.690	0.611	0.545	0.621
`LSTM DT`	0.366	0.642	0.716	0.985	0.988	0.886	0.333	0.500	0.579	0.545	0.556
`ARIMA`	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.500	0.641	0.658	0.824	0.961	0.672	0.667	0.840	0.538	0.444	0.556
`TadGAN`	0.440	0.564	0.626	0.815	0.735	0.577	0.625	0.588	0.692	0.500	0.543
`Dense AE`	0.621	0.719	0.732	0.922	0.956	0.519	0.600	0.875	0.625	0.471	0.565
`Azure AD`	0.026	0.010	0.167	0.484	0.542	0.216	0.027	0.036	0.145	0.035	0.161
						Recall					
`AER`	0.611	0.731	0.764	0.985	0.814	0.583	0.667	0.667	1.000	0.857	0.545
`LSTM DT`	0.722	0.776	0.792	0.985	0.595	0.495	0.500	0.600	1.000	0.857	0.606
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.746	0.596	0.915	0.291	0.145	0.667	0.700	0.636	0.571	0.455
`TadGAN`	0.611	0.791	0.517	0.905	0.296	0.232	0.833	0.667	0.818	0.714	0.576
`Dense AE`	0.500	0.612	0.584	0.830	0.046	0.049	0.500	0.700	0.455	0.571	0.394
`Azure AD`	0.806	0.940	0.848	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818

Table 15. Benchmark Summary Results Version `0.3.0`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score					
`LSTM DT`	0.476	0.741	0.739	0.990	0.753	0.644	0.400	0.537	0.714	0.703	0.556
`ARIMA`	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.500	0.658	0.584	0.877	0.444	0.262	0.667	0.724	0.667	0.471	0.475
`TadGAN`	0.575	0.659	0.564	0.853	0.439	0.370	0.615	0.656	0.640	0.541	0.559
`Dense AE`	0.523	0.661	0.665	0.891	0.072	0.109	0.400	0.764	0.571	0.552	0.517
`Azure AD`	0.050	0.020	0.279	0.653	0.702	0.344	0.053	0.068	0.250	0.068	0.269
						Precision					
`LSTM DT`	0.362	0.697	0.710	0.995	0.985	0.888	0.333	0.486	0.588	0.565	0.513
`ARIMA`	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.500	0.598	0.630	0.830	0.935	0.709	0.667	0.750	0.615	0.400	0.538
`TadGAN`	0.490	0.550	0.638	0.811	0.784	0.599	0.571	0.645	0.571	0.435	0.543
`Dense AE`	0.586	0.719	0.752	0.925	0.946	0.595	0.500	0.840	0.600	0.533	0.600
`Azure AD`	0.026	0.010	0.167	0.484	0.542	0.216	0.027	0.036	0.145	0.035	0.161
						Recall					
`LSTM DT`	0.694	0.791	0.77	0.985	0.610	0.505	0.500	0.600	0.909	0.929	0.606
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.731	0.545	0.930	0.291	0.160	0.667	0.700	0.727	0.571	0.424
`TadGAN`	0.694	0.821	0.506	0.900	0.305	0.267	0.667	0.667	0.727	0.714	0.576
`Dense AE`	0.472	0.612	0.596	0.860	0.037	0.060	0.333	0.700	0.545	0.571	0.455
`Azure AD`	0.806	0.940	0.848	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818

Table 16. Benchmark Summary Results Version `0.2.1`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
					F1 Score						
`LSTM DT`	0.460	0.703	0.752	0.980	0.733	0.643	0.400	0.481	0.643	0.684	0.568
`ARIMA`	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.480	0.690	0.600	0.870	0.436	0.243	0.545	0.750	0.615	0.571	0.525
`TadGAN`	0.558	0.650	0.559	0.890	0.412	0.374	0.500	0.677	0.692	0.500	0.567
`Dense AE`	0.529	0.661	0.652	0.899	0.087	0.092	0.545	0.741	0.632	0.552	0.517
`Azure AD`	0.050	0.020	0.279	0.653	0.702	0.344	0.053	0.068	0.250	0.068	0.269
					Precision						
`LSTM DT`	0.359	0.654	0.716	0.975	0.991	0.895	0.333	0.388	0.529	0.542	0.512
`ARIMA`	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.462	0.653	0.610	0.825	0.943	0.663	0.600	0.808	0.533	0.571	0.571
`TadGAN`	0.480	0.567	0.642	0.837	0.746	0.635	0.500	0.656	0.600	0.409	0.559
`Dense AE`	0.562	0.719	0.746	0.960	0.956	0.553	0.600	0.833	0.750	0.533	0.600
`Azure AD`	0.026	0.010	0.167	0.484	0.542	0.216	0.027	0.036	0.145	0.035	0.161
					Recall						
`LSTM DT`	0.639	0.761	0.792	0.985	0.581	0.502	0.500	0.633	0.818	0.929	0.636
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.731	0.590	0.920	0.283	0.149	0.500	0.700	0.727	0.571	0.485
`TadGAN`	0.667	0.761	0.494	0.950	0.284	0.265	0.500	0.700	0.818	0.643	0.576
`Dense AE`	0.500	0.612	0.579	0.845	0.046	0.050	0.500	0.667	0.545	0.571	0.455
`Azure AD`	0.806	0.940	0.848	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818

Table 17. Benchmark Summary Results Version `0.2.0`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score					
`LSTM DT`	0.447	0.671	0.724	0.975	0.751	0.637	0.400	0.488	0.733	0.619	0.535
`ARIMA`	0.435	0.326	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.500	0.667	0.593	0.859	0.422	0.223	0.667	0.724	0.720	0.552	0.540
`TadGAN`	0.465	0.646	0.549	0.843	0.492	0.388	0.667	0.623	0.741	0.476	0.523
`Dense AE`	0.563	0.710	0.656	0.889	0.084	0.094	0.545	0.800	0.632	0.500	0.517
`Azure AD`	0.061	0.021	0.276	0.653	0.702	0.344	0.053	0.068	0.286	0.068	0.269
						Precision					
`LSTM DT`	0.343	0.600	0.680	0.966	0.990	0.887	0.333	0.385	0.579	0.464	0.500
`ARIMA`	0.455	0.311	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.500	0.635	0.614	0.825	0.934	0.667	0.667	0.750	0.643	0.533	0.567
`TadGAN`	0.400	0.546	0.600	0.814	0.792	0.584	0.556	0.613	0.625	0.357	0.531
`Dense AE`	0.643	0.772	0.731	0.944	0.953	0.544	0.600	0.880	0.750	0.500	0.600
`Azure AD`	0.032	0.011	0.166	0.484	0.542	0.216	0.027	0.036	0.169	0.035	0.161
						Recall					
`LSTM DT`	0.639	0.761	0.775	0.985	0.605	0.497	0.500	0.667	1.000	0.929	0.576
`ARIMA`	0.417	0.343	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.701	0.573	0.895	0.273	0.134	0.667	0.700	0.818	0.571	0.515
`TadGAN`	0.556	0.791	0.506	0.875	0.357	0.291	0.833	0.633	0.909	0.714	0.515
`Dense AE`	0.500	0.657	0.596	0.840	0.044	0.051	0.500	0.733	0.545	0.500	0.455
`Azure AD`	0.806	0.940	0.815	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818

Table 18. Benchmark Summary Results Version `0.1.7`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score					
`LSTM DT`	0.466	0.689	0.739	0.975	0.746	0.645	0.400	0.500	0.759	0.585	0.551
`ARIMA`	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.507	0.667	0.601	0.880	0.445	0.231	0.667	0.712	0.667	0.516	0.508
`TadGAN`	0.517	0.634	0.551	0.841	0.484	0.376	0.571	0.689	0.769	0.563	0.559
`Dense AE`	0.507	0.693	0.665	0.904	0.078	0.090	0.545	0.786	0.600	0.581	0.533
`Azure AD`	0.061	0.021	0.276	0.653	0.702	0.344	0.053	0.068	0.286	0.068	0.269
						Precision					
`LSTM DT`	0.358	0.619	0.684	0.966	0.984	0.894	0.333	0.413	0.611	0.444	0.528
`ARIMA`	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.487	0.623	0.624	0.861	0.945	0.654	0.667	0.724	0.615	0.471	0.577
`TadGAN`	0.434	0.543	0.642	0.801	0.792	0.585	0.500	0.677	0.667	0.500	0.543
`Dense AE`	0.548	0.733	0.752	0.971	0.950	0.532	0.600	0.846	0.667	0.529	0.593
`Azure AD`	0.032	0.011	0.166	0.484	0.542	0.216	0.027	0.036	0.169	0.035	0.161
						Recall					
`LSTM DT`	0.667	0.776	0.803	0.985	0.601	0.504	0.500	0.633	1.000	0.857	0.576
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.528	0.716	0.579	0.900	0.291	0.140	0.667	0.700	0.727	0.571	0.455
`TadGAN`	0.639	0.761	0.483	0.885	0.348	0.277	0.667	0.700	0.909	0.643	0.576
`Dense AE`	0.472	0.657	0.596	0.845	0.040	0.049	0.500	0.733	0.545	0.643	0.485
`Azure AD`	0.806	0.940	0.815	1.000	0.998	0.837	1.000	0.733	0.909	1.000	0.818

Table 19. Benchmark Summary Results Version `0.1.6`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
F1 Score											
`LSTM DT`	0.480	0.718	0.747	0.975	0.739	0.649	0.400	0.474	0.741	0.684	0.583
`ARIMA`	0.344	0.309	0.744	0.816	0.782	0.684	0.429	0.472	0.727	0.429	0.513
`LSTM AE`	0.474	0.667	0.593	0.865	0.438	0.276	0.667	0.764	0.615	0.452	0.552
`TadGAN`	0.529	0.654	0.555	0.822	0.487	0.377	0.714	0.645	0.741	0.486	0.567
`Dense AE`	0.515	0.667	0.648	0.897	0.080	0.091	0.545	0.786	0.600	0.581	0.517
`Azure AD`	0.061	0.021	0.276	0.653	0.702	0.344	0.053	0.070	0.019	0.068	0.269
Precision											
`LSTM DT`	0.375	0.680	0.690	0.966	0.991	0.893	0.333	0.391	0.625	0.542	0.538
`ARIMA`	0.393	0.304	0.684	0.772	0.998	0.955	0.375	0.405	0.727	0.429	0.444
`LSTM AE`	0.450	0.635	0.629	0.833	0.931	0.711	0.667	0.840	0.533	0.412	0.640
`TadGAN`	0.451	0.573	0.592	0.792	0.764	0.575	0.625	0.625	0.625	0.391	0.559
`Dense AE`	0.567	0.712	0.719	0.955	0.975	0.586	0.600	0.846	0.667	0.529	0.600
`Azure AD`	0.032	0.011	0.166	0.484	0.542	0.216	0.027	0.037	0.009	0.035	0.161
Recall											
`LSTM DT`	0.667	0.761	0.815	0.985	0.589	0.510	0.500	0.600	0.909	0.929	0.636
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.727	0.429	0.606
`LSTM AE`	0.500	0.701	0.562	0.900	0.286	0.171	0.667	0.700	0.727	0.500	0.485
`TadGAN`	0.639	0.761	0.522	0.855	0.358	0.280	0.833	0.667	0.909	0.643	0.576
`Dense AE`	0.472	0.627	0.590	0.845	0.042	0.049	0.500	0.733	0.545	0.643	0.455
`Azure AD`	0.806	0.940	0.815	1.000	0.998	0.837	1.000	0.700	0.909	1.000	0.818

Table 20. Benchmark Summary Results Version `0.1.5`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score					
`LSTM DT`	0.532	0.704	0.735	0.980	0.743	0.653	0.400	0.462	0.467	0.615	0.548
`ARIMA`	0.344	0.307	0.744	0.816	0.782	0.684	0.429	0.472	0.538	0.429	0.513
`TadGAN`	0.575	0.644	0.626	0.700	0.494	0.381	0.714	0.677	0.800	0.450	0.592
`Azure AD`	0.061	0.021	0.271	0.653	0.697	0.337	0.053	0.068	0.019	0.068	0.269
						Precision					
`LSTM DT`	0.431	0.667	0.690	0.980	0.991	0.899	0.333	0.375	0.368	0.480	0.500
`ARIMA`	0.393	0.300	0.684	0.772	0.998	0.955	0.375	0.405	0.467	0.429	0.444
`TadGAN`	0.490	0.523	0.652	0.700	0.795	0.588	0.625	0.656	0.714	0.346	0.553
`Azure AD`	0.032	0.011	0.163	0.484	0.541	0.212	0.027	0.036	0.009	0.035	0.161
						Recall					
`LSTM DT`	0.694	0.746	0.787	0.980	0.594	0.513	0.500	0.600	0.636	0.857	0.606
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.636	0.429	0.606
`TadGAN`	0.694	0.836	0.601	0.700	0.359	0.281	0.833	0.700	0.909	0.643	0.636
`Azure AD`	0.806	0.940	0.787	1.000	0.978	0.816	1.000	0.733	0.909	1.000	0.818

Table 21. Benchmark Summary Results Version `0.1.4`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score					
`LSTM DT`	0.468	0.708	0.738	0.978	0.728	0.634	0.400	0.468	0.437	0.667	0.564
`ARIMA`	0.344	0.307	0.744	0.816	0.782	0.684	0.429	0.472	0.538	0.429	0.513
`Azure AD`	0.061	0.021	0.277	0.653	0.692	0.333	0.053	0.068	0.019	0.068	0.269
						Precision					
`LSTM DT`	0.379	0.662	0.679	0.970	0.984	0.890	0.333	0.383	0.333	0.545	0.489
`ARIMA`	0.393	0.300	0.684	0.772	0.998	0.955	0.375	0.405	0.467	0.429	0.444
`Azure AD`	0.032	0.011	0.184	0.484	0.537	0.216	0.027	0.036	0.009	0.035	0.161
						Recall					
`LSTM DT`	0.611	0.761	0.809	0.985	0.577	0.492	0.500	0.600	0.636	0.857	0.667
`ARIMA`	0.306	0.313	0.815	0.865	0.643	0.533	0.500	0.567	0.636	0.429	0.606
`Azure AD`	0.806	0.940	0.562	1.000	0.972	0.729	1.000	0.733	0.909	1.000	0.818

Table 22. Benchmark Summary Results Version `0.1.3`

	NASA		Yahoo S5				NAB				
Pipeline	MSL	SMAP	A1	A2	A3	A4	Art	AWS	AdEx	Traf	Tweets
						F1 Score					
`LSTM DT`	0.495	0.750	0.757	0.987	0.756	0.643	0.400	0.531	0.452	0.718	0.620
`ARIMA`	0.489	0.424	0.753	0.856	0.783	0.693	0.429	0.576	0.538	0.545	0.513
						Precision					
`LSTM DT`	0.364	0.680	0.721	0.975	0.986	0.890	0.333	0.472	0.350	0.560	0.579
`ARIMA`	0.393	0.300	0.690	0.772	0.998	0.955	0.375	0.567	0.467	0.429	0.444
						Recall					
`LSTM DT`	0.774	0.836	0.798	1.000	0.613	0.503	0.500	0.607	0.636	1.000	0.667
`ARIMA`	0.647	0.724	0.829	0.961	0.644	0.543	0.500	0.586	0.636	0.750	0.606