# ODIN AD: a framework supporting the life-cycle of time series anomaly detection applications

Niccolò Zangrando[0000−0002−4796−5649], Piero Fraternali[00000−0002−6945−2625], Rocio Nahime Torres[0000−0003−2865−0278], Marco Petri[0000−0001−5368−9196], Nicolò Oreste Pinciroli Vago[0000−0001−7906−4987], and Sergio Herrera[0000−0002−8903−0622]

Department of Electronics, Information, and Bioengineering, Politecnico di Milano, 20133 Milan, Italy
{niccolo.zangrando, piero.fraternali, rocionahime.torres, nicolooreste.pinciroli, sergioluis.herrera}@polimi.it, marco.petri@mail.polimi.it

**Abstract.** Anomaly detection (AD) in numerical temporal data series is a prominent task in many domains, including the analysis of industrial equipment operation, the processing of IoT data streams, and the monitoring of appliance energy consumption. The life-cycle of an AD application with a Machine Learning (ML) approach requires data collection and preparation, algorithm design and selection, training, and evaluation. All these activities contain repetitive tasks which could be supported by tools. This paper describes ODIN AD, a framework assisting the life-cycle of AD applications in the phases of data preparation, prediction performance evaluation, and error diagnosis.

**Keywords:** Time series · Anomaly detection · Data annotation · Model evaluation · Evaluation metrics.

## 1 Introduction

With the advent of IoT architectures, the analysis of numerical temporal data series is being increasingly applied in such industries as manufacturing and construction, in which machines, appliances, and whole systems are equipped with sensors producing timestamped numerical data streams. Applications include anomaly detection (AD) [5] whose primary focus is to find anomalies in the operation of working equipment at early stages to alert and avoid breakdowns. AD is also at the core of predictive maintenance (PdM) [29], which aims at optimizing the trade-off between run-to-failure and periodic maintenance, improving the Remaining Useful Life (RUL) of machines, and avoiding unplanned downtime. The development of an AD solution follows the typical life-cycle of a data-driven application, illustrated in Figure 1. Such a workflow differs from that of a traditional software system because it relies on predefined parametric algorithms that must be fit to the specific task and data at hand [7].

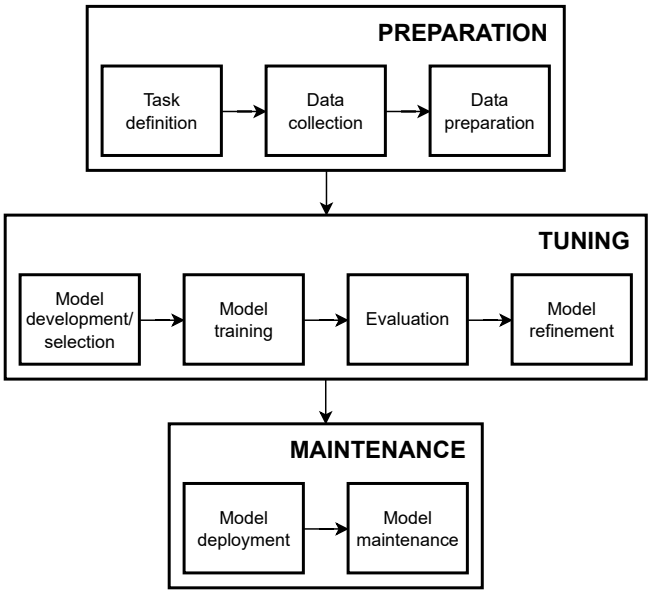


Fig. 1: Life-cycle of a data-driven application

The workflow illustrated in Figure 1 contains many repetitive tasks. In the preparation stage, the collected data must be annotated with ground truth labels (GT) for training and evaluation purposes. They could also be enriched with additional domain- or task-dependent meta-data, which could be exploited for performance diagnosis purposes [11]. In the tuning stage, the quality of an algorithm is assessed by computing general and task-specific prediction performance metrics. When multiple candidate algorithms are available, their performances must be compared head-to-head on the same data set. Their generalization capability must be checked too, by testing a model trained on the data from a specific source with the data of another distinct source of the same type. Model refinement also entails diagnosing the causes of prediction failures, which may require the categorization of errors into general and task-specific types, the attribution of errors to particular characteristics of the input data, and the quantification of the impact that a certain error type has on the prediction performance metrics.

All these activities are amenable to support by computerized tools. An ideal development environment should enable the data scientist to load multiple data sets and annotate each one with GT labels or with other meta-data, pick the selected algorithms from a library and execute them, and obtain per-algorithm and also comparative performances reports. In the refinement phase, it should be possible to break down the performance metrics based on user-defined criteria, classify errors with user-defined criteria, and assess the impact of the various types of errors on each metrics.

This paper presents ODIN AD, a framework supporting the development of AD applications on numerical uni- and multi-variate data series. ODIN AD offers the following features:

- Data ingestion. Temporal data series can be imported with CSV files. Existing meta-data can be imported too, in CSV format.
- Data annotation. If needed, a data set can be enriched with GT labels and custom meta-data, with a dedicated annotator GUI.
- Algorithm selection and execution. ODIN AD does not support model design and execution but lets the user import the predictions made with any algorithm into the workspace. One or more prediction CSV files can be loaded in the same analysis session.
- Model/algorithm evaluation under multiple configurations. The user can define the configuration of an evaluation session, by choosing the anomaly definition and matching strategy to use and the metrics to compute. ODIN AD implements 5 anomaly definition strategies, 4 anomaly to GT matching strategies, 9 performance metrics and 4 performance curves off-the-shelf. The users can plug in their own strategies and metrics.
- Error diagnosis. Prediction errors can be categorized with user-defined criteria and the impact of a specific type of error on the performance metrics can be quantified.
- Model/algorithm performance visualization and reporting. Prediction performance metrics can be displayed in a visualization GUI and embedded in a performance evaluation report.
- Performance comparison. The visualization and the reporting functions can be applied to a single algorithm or to multiple ones. In the latter case, the head-to-head comparison of the selected algorithms on all the chosen metrics is provided.

ODIN AD is algorithm-agnostic and designed to be extensible. Its architecture allows the integration of other input/output data formats, AD definition strategies, performance metrics, and visualization widgets.

## 2 Related work

Statistical and ML algorithms are applied to temporal data series for such applications as forecasting [19], anomaly detection [3, 27] and predictive maintenance [33]. The computer-based aid to AD application development mostly focuses on the evaluation phase. Benchmark data sets, such as SKAB [16] and NAB Benchmark [18], annotate data with GT labels and implement common evaluation metrics such as F1 score, NAB score, false alarm rate, and miss alarm rate.

Contributions such as [2, 10, 34] extend the support beyond the use of the basic performance measures in the evaluation phase. The work [2] generalizes the metrics provided by AD benchmarks by introducing the concept of Preceding Window ROC, which extends the popular ROC diagram to the case of time

series. Also, the evaluation process is adapted to better fit the needs of AD algorithm assessment, e.g., by rewarding early anomaly detection. The Darts library [10] assists time series analysis in general. It implements multiple models, from ARIMA to Deep Neural Networks (DNNs). It supports uni- and multi-variate series, meta-learning on multiple series, training on large datasets, model ensembles, and probabilistic forecasting. The library is designed for usability, achieved by wrapping the underlying functions under a simple and uniform Python interface. The RELOAD tool [34] aids the ingestion of data, the selection of the most informative features, the execution of multiple AD algorithms, the evaluation of alternative anomaly identification strategies, the computation of performance metrics, and the visualization of results in a GUI. RELOAD implements multiple metrics and algorithms off-the-shelf and has an extensible architecture. However, it does not support yet the breakdown of performance metrics by user-defined criteria and the characterization of errors.

The PySAD tool [32] supports AD application development on streaming data. It comprises pre-processors for data normalization and enables the execution of AD models and of model ensembles. It also features post-processors to transform model scores for learning purposes. Its evaluator module includes multiple AD metrics (e.g., precision, recall, and window score) and a wrapper to adapt Sklearn metrics so as to allow extensibility. Other tools, such as TagAnomaly [23], Curve [1] or TRAINSET [13], focus mainly on the data preparation step. They let developers annotate anomalies but do not offer the possibility to add meta-data to them. The Wearables Development Toolkit (WDK) [9] is a framework for the analysis of time series produced by wearable and IoT devices. It supports the annotation, the analysis, and the visualization of time series and the performance assessment of activity recognition algorithms.

In the specific field of intrusion detection, the work [22] describes CyberVTI, a client-server tool for AD in network time series. CyberVTI incorporates multiple state-of-the-art unsupervised algorithms and helps the analyst inspect their performances with different parameters. It supports the phases of data ingestion, data preparation, in which the imported data are validated, feature engineering, in which the features are selected, extracted, and normalized, and processing, in which the AD algorithms are executed.

All the mentioned tools that support the evaluation step restrict performance assessment to a few standard metrics. The use of DNNs for AD [3] is evidencing the limits of such a basic approach. DNNs have a complex architecture which makes their behavior hard to understand and debug. This characteristic demands more informative approaches to error diagnosis and model refinement. One possibility is to exploit the semantic richness of time series, which are characterized by many properties (e.g., the sampling frequency, the stationarity, and periodicity of the series, the type and physical characteristics of the signal and of the corresponding acquisition sensor). Such an abundance of significant input properties could be exploited to enable the breakdown of performance indicators and to correlate the errors with specific features of the input and with user-defined categories. The exploitation of data series semantic features, beyond those used

for training, and the characterization of errors in user-defined categories are distinctive capabilities of ODIN AD.

## 3 ODIN AD

In this section we illustrate the main functionalities of ODIN AD. The running example employed to produce the diagrams and the visualizations exploits the time series of the REFIT data set, specifically the fridge consumption data series of house 1 [26]. The GT labels used to compute the performance metrics have been created by three independent annotators. The algorithms used to produce the diagrams are GRU-autoencoder and LSTM-autoencoder [4, 20].

### 3.1 Data ingestion, analysis, and preparation

ODIN AD lets the user import the time series data, the GT labels, and the semantic input annotations. The artifacts follow the formatting guidelines common to most public datasets. Temporal series are imported as CSV files with a timestamp identifier followed by the feature values; GT data are encoded in JSON files listing the timestamps at which anomalies occur; input properties can be imported as CSV files with a timestamp identifier and a column per property.

When the GT labels are not available, ODIN AD lets the user define them with the anomaly annotator GUI shown in Figure 2. An anomaly is created by selecting a point or an interval on the time axis. Anomalies can be annotated with user-defined meta-data, which can be inserted and/or modified with the annotator GUI. In Figure 2 the custom annotations refer to a user-defined categorization of the anomalies. The anomaly annotations of the running example include: *Continuous OFF state*, when the appliance is in the low consumption state for a long time, *Continuous ON state*, when the appliance is in the consumption state for an abnormally long time, *Spike*, when the appliance has an abnormal consumption peak, *Spike + Continuous*, when the appliance has a consumption peak followed by a prolonged ON state, *Other*, when the anomaly does not follow a well-defined pattern.

Anomalies and their annotations can be deleted, updated, and exported to a CSV file.

When the same time series is annotated by more than one user, ODIN AD supports the analysis of the inter-annotator agreement over the anomalies and their associated properties, with the help of the diagrams shown in Figure 3.

In addition to the manually provided properties, ODIN AD supports the automatic extraction of properties from the data series, so as to speed up the annotation process. The current version of ODIN AD implements some basic property extractors: hour of the day, day of the week, month, and duration. The user can add her own extractors to automatically compute custom and domain-dependent properties. As an example, a custom extractor is implemented to automatically derive the anomaly annotations shown in Figure 2. The user can display and validate or modify the automatically extracted proposals in the

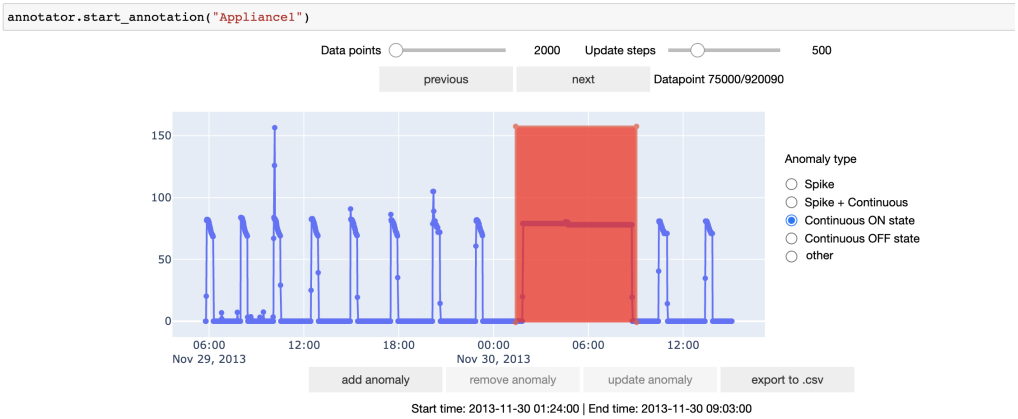


Fig. 2: The interface of the GT anomaly annotator at work on the running example time series. The user can specify the anomalies and add meta-data to them. The user has annotated the currently selected GT anomaly, shown in red, with the *Continuous ON state* label.

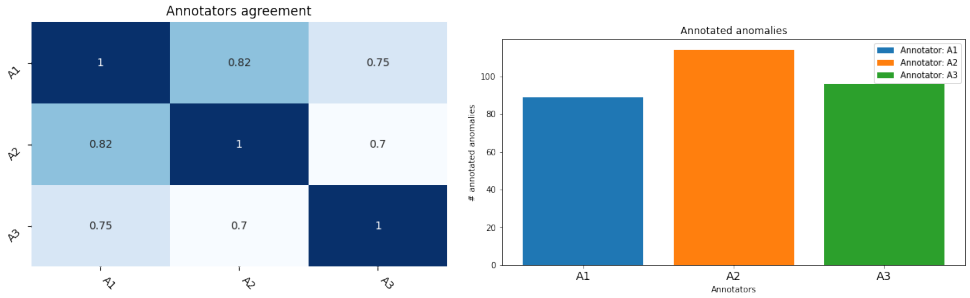


Fig. 3: Diagrams of the inter-annotation agreement: the annotator consensus diagram (left) shows the percentage of data points that are classified in the same way by each pair of annotators. The GT anomalies histogram plots the number of anomalies (points or intervals) created by each annotator.

annotator GUI. In this way, the GT semantic labeling process is accelerated. Figure 4 shows the distribution of the GT anomalies across the *duration* and *anomaly type* properties of the time series.

The temporal data series can be pre-processed before the application of AD algorithms. ODIN AD currently implements the following pre-processors: the stationarity pre-processor, implemented with the Dickey-Fuller test [6], the periodicity pre-processor, implemented with the Fast Fourier Transform method [15], and the seasonality, trend and residual decomposition pre-processor [12]. Residuals decomposition can be done with an additive model (addition of the decomposed values restores the original times series) or with a multiplicative one

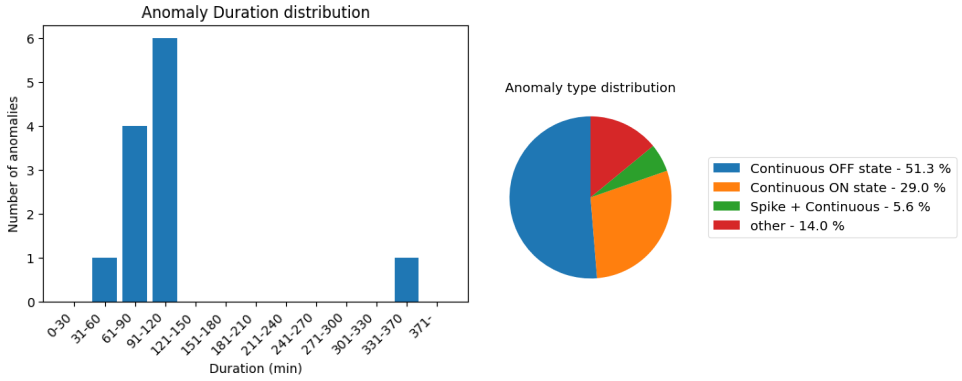


Fig. 4: ODIN AD shows the annotated anomalies in the running example time series distributed by the automatically extracted *duration* property (left) and by the manually refined *anomaly type* property (right).

(the original series is obtained by multiplying the decomposed values). ODIN AD also supports data transformations. The input time series and the output predictions can be manipulated using *scalers*. The current version of ODIN AD implements some predefined scalers (MinMaxScaler, StandardScaler) and can be extended with user-defined ones.

### 3.2 Execution

AD algorithms are executed outside ODIN AD and their output is imported into an analysis session. Depending on the AD approach, predictions can be structured as follows:

- If the AD algorithms exploits classification (e.g., OneCLassSVM, LocalOutlierFactor, Isolation Forest), for each timestamp the prediction contains the confidence score.
- If the AD algorithms exploit forecasting (e.g., ARIMA, LSTM) or reconstruction (e.g., LSTMAutoencoder, GRUAutoencoder) the prediction file can contain one value per timestamp (single-valued prediction) or multiple values per timestamp (multi-valued prediction). The latter case is relevant for the methods that exploit a sliding window (e.g., GRU, Autoencoder-based), which assign a different predicted/reconstructed value to the same data point based on the window used to compute the prediction.

### 3.3 Evaluation and refinement

ODIN AD supports the assessment of anomaly detectors under multiple anomaly definitions and matching strategies and performance metrics.

**Anomaly definition strategies.** An anomaly definition strategy specifies the way in which the data points of the input time series are compared with the predicted or reconstructed points of the anomaly detector in order to infer whether a point or an interval is anomalous. Each strategy takes in input a pair of entities to compare (points and/or sets of points) and returns a score value $s$ or a score vector $\mathbf{s}$ interpretable as the confidence with which the prediction is considered an anomaly. Different strategies can be adopted for the types of predictions computed by the AD algorithms. The current version of ODIN AD implements the following anomaly definition strategies:

- Absolute and Squared Error (AE/SE) [25]: computes the score $s$ as the absolute or squared error between the input and the predicted/reconstructed value. It applies to single-valued predictions.
- Likelihood: each point in the time series is predicted/reconstructed $l$ times and associated with multiple error values. The probability distribution of the errors made by predicting on normal data is used to compute the likelihood of normal behavior on the test data, which is used to derive an anomaly score. This method was introduced in [21]. It applies to single- and multi-valued predictions.
- Mahalanobis: as in the likelihood strategy, each point in the time series is predicted/reconstructed $l$ times. For each point, the anomaly score $s$ is calculated as the square of the Mahalanobis distance between the error vector and the Gaussian distribution fitted from the error vectors computed during validation [20]. It applies to single-valued and to multi-valued predictions.
- Windows strategy: the strategy computes a score vector $\mathbf{s}$ of dimension $l$ associated with each point. Each element $s_i$ of the score vector is the MAE (by default) or the MSE of the i-th predicted/reconstructed window that contains the point [17]. It applies to multi-valued predictions.

The AE, SE, Gaussian, and Mahalanobis strategies compute an anomaly score $s$. A threshold $\tau$ is then applied to such a value for classifying the point as normal or anomalous. The Windows strategy computes an array of anomaly scores and in this case, a point is considered anomalous if each element of the array is above the threshold.

**Anomaly matching strategies.** An anomaly matching strategy specifies the way in which an identified anomaly is compared to the GT, so as to categorize it as a true positive (TP), false positive (FP), true negative (TN), and false negative (FN). ODIN AD implements four strategies:

- Point to point: each anomalous point is compared only to the corresponding data series point using the GT label.
- Interval to interval: the Intersection over Union (IoU) metrics is calculated between the GT anomaly interval and the predicted anomaly interval and a threshold $\tau$ is applied to categorize the prediction (IoU $> \tau$ qualifies a TP). By default, the threshold is set to 0.5, but it can be modified.

– Interval to point(s): each predicted anomalous interval is considered a TP if it contains at least $X$ GT anomaly points. By default, X is set to 50% of the interval points.
– Point to interval: each predicted anomaly point is considered TP if it lies within the boundaries of a GT anomaly interval.

**Metrics** ODIN AD implements the basic time series metrics and diagrams (accuracy, precision, recall, F1 score, $F_{0-1}$ score, miss alarm rate, false alarm rate, NAB score, Matthews Coefficient, PR and ROC curves). The predefined portfolio can be extended with additional metrics.

**Analysis, reporting, and visualization.** The analysis and reports include:

– Performance summary. The values of the selected metrics (standard and custom) are organized in a comprehensive report.
– Performance per confidence threshold value. The performance metrics that exploit a threshold on the anomaly confidence score are plotted for each value used in their computation.
– Performance per IoU threshold value. The performance metrics that exploit a threshold on the overlap between the prediction and the GT interval are plotted for each value used in their computation.
– Confusion matrix. The TPs, FPs, FNs, and TNs are displayed in the usual tabular arrangement.
– Per-property metrics break down. One or more performance metrics are disaggregated by the values of a semantic property of the input. An example is presented in Figure 5: recall metrics break down by anomaly type and anomaly duration.

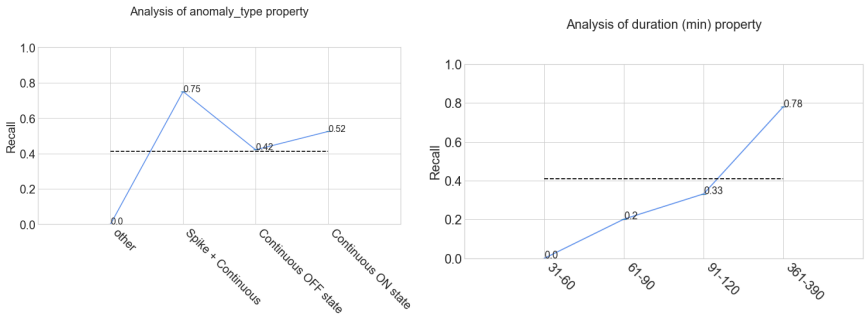


Fig. 5: Break down of the recall metrics based on the values of the *anomaly type* and of the *duration* property. The diagram shows that the algorithm identifies well anomalies of type *spike + continuous* and has more difficulty in detecting the continuous state types. Recall is maximal for long duration anomalies, which are rare and easier to detect.

– FP error categorization. FP errors are grouped into classes and for each class the performance improvement achievable if such errors were removed is computed. Figure 6 shows an example of FP error categorization and impact analysis predefined in ODIN AD. The user can define other categorizations and apply the break down to any metrics.
– Anomaly duration difference and distribution. The difference of duration between the GT and the TP anomalies and the distribution of the duration of the GT and of all the predicted anomalies are plotted.
– Calibration analysis. The confidence histogram and the reliability diagram [8] enable the assessment of how well the distribution of the predicted anomalies agrees with that of the real anomalies.

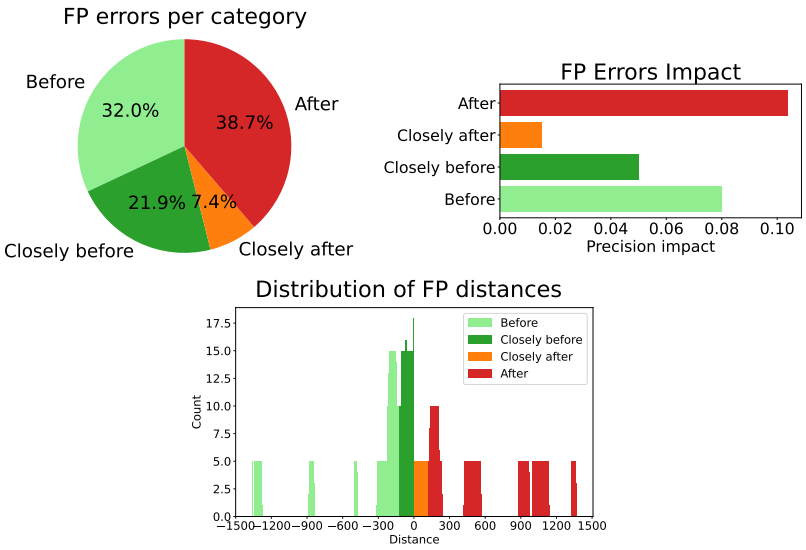


Fig. 6: The FP errors per category diagram (left) shows the distribution of the FPs across four categories. In this breakdown, FPs are grouped by position (before/after) and distance (close/not close) w.r.t. to the closest GT anomaly based on a distance threshold parameter. The FP errors impact diagram (right) shows the contribution of removing each error type on the precision metrics. The distance distribution diagram (bottom) shows the distribution of the distance between FPs and the closest GT anomaly. The analysis shows that most FPs are positioned after more than two hours w.r.t. the closest anomaly and that the post-anomaly false alarms impact the precision metrics the most.

In addition to the performance reports and diagrams, an interface permits the inspection of the data set and of the predictions, as shown in Figure 7. The user can import a data set and the predictions of one or more AD algorithms,

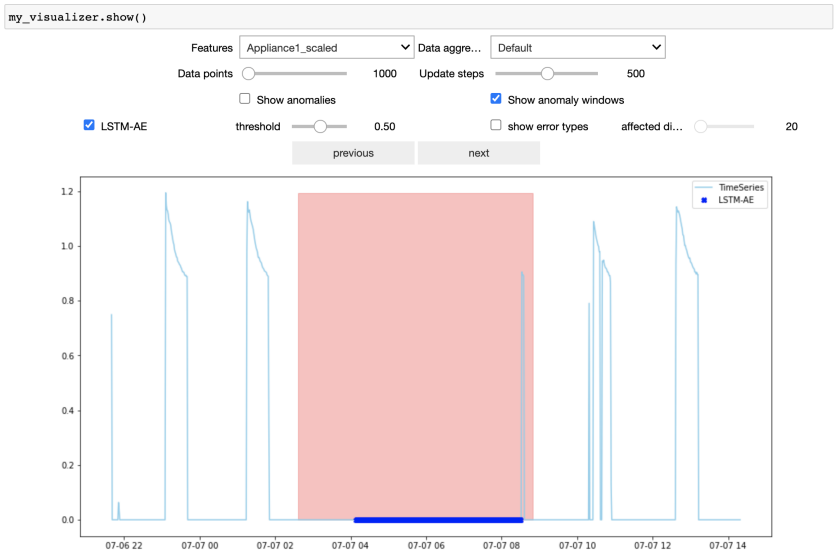


Fig. 7: The user interface of the anomaly visualizer showing the point-wise predictions of one model vs. the GT intervals. A scrollable window lets the user browse the time series. GT interval anomalies are highlighted as colored rectangles and the predicted anomalous points as dots. Adjusting the threshold value used by the anomaly definition strategy updates the diagram.

select the feature to display, in the case of multi-variate data series, and scroll the timeline to inspect the anomalies. The visualization can be configured by setting the slice of the time series to show, the number of points per slice and the granularity of the timeline. The default granularity is equal to the sampling frequency of the data set but can be adjusted by aggregating the data points. The user can also set the threshold value used by the anomaly definition strategy.

**Model comparison.** ODIN AD also supports the comparison of the result of multiple models applied to the same data set. The comparative diagrams can contrast the anomalies detected by the different models and the values of the performance metrics. Figure 8 shows the comparative performance diagram and anomaly visualization.

### 3.4 ODIN AD architecture

ODIN AD is open-source[1] and implemented in Python. The annotator and the visualizer are Jupyter Notebook applications.

Figure 9 illustrates the structure of the classes. The five main modules are: (1) DatasetAD, responsible for loading the dataset and pre-processing/analyzing

[1] https://github.com/rnt-pmi/odin

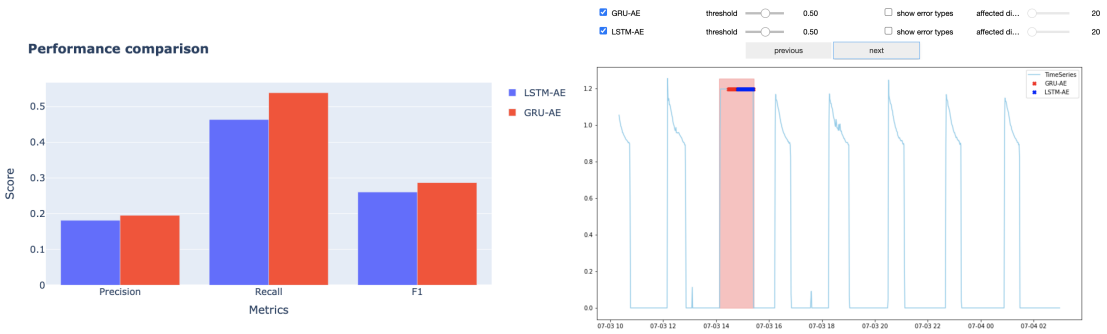


Fig. 8: The user interface of the anomaly visualizer for jointly assessing the results of multiple algorithms: head-to-head display of performance metrics (left) and comparison of identified anomalies (right).

it; (2) AnalyzerAD, for computing the performance metrics and diagrams based on the imported prediction files; (3) AnnotatorAD, for creating/editing GT and meta-data annotations; (4) ComparatorAD, for contrasting different models on the same dataset; (5) VisualizerAD for displaying the input time series and the predictions. Other modules permit the customization of the data set pre/post-processing (Scaler, PropertiesExtractor) and of the types of analysis (AnomalyDefinitionStrategy, AnomalyMatchingStrategy, CustomMetrics, ErrorCategory).

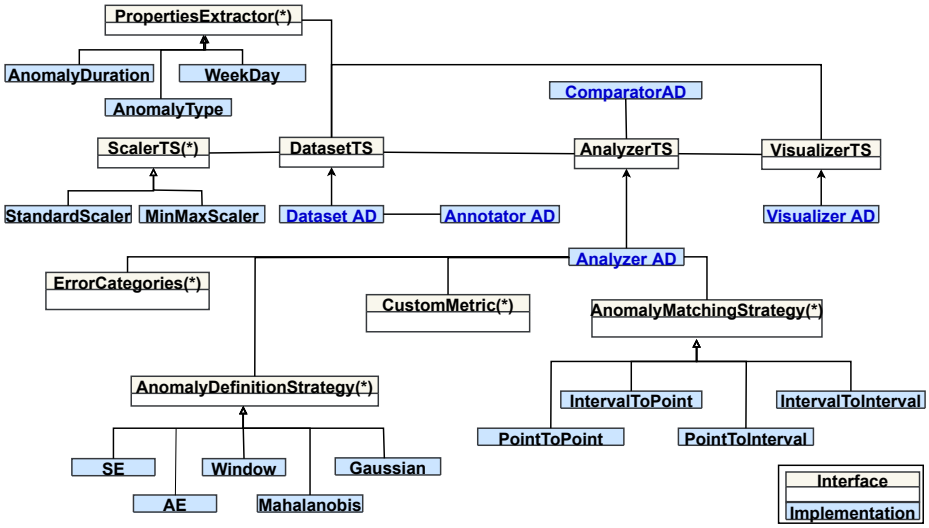


Fig. 9: Simplified class diagram of the architecture of ODIN AD. The components marked with (*) denote the entry points for the extension of ODIN AD.

### 3.5 Extending ODIN AD

ODIN AD is publicly available and the code repository contains a test suite that facilitates extension and bug checking. The "plug&play" architecture enables the customization of the anomaly definition and matching strategies, of the performance metrics, of the pre- and post-data processors, and of the property extractors. Next, we illustrate some examples of how the extension works.

Listing 1 shows how to add a custom anomaly definition strategy. The listing imports the necessary interface (line 1), defines a `CustomAnomalyDefinition` class implementing such interface (line 3), and codes the two required functions. The `get_anomaly_score` function computes the anomaly scores (line 5) and the `check_prediction_format` function verifies that the anomaly definition strategy works with the proper prediction formats, single or multi-valued (line 8).

Listing 1: Addition of a custom anomaly definition strategy

```
1 from TOOL.classes.TimeSeries import AnomalyDefinitionStrategy
2
3 class CustomAnomalyDefinition(AnomalyDefinitionStrategy):
4
5   def get_anomaly_scores(self, gt, predictions):
6     # returns the anomaly score for the input time series
7
8   def check_prediction_format(self, predictions):
9     # returns a Boolean indicating if the predictions have the valid
      ↪  type for the strategy
```

Listing 2 shows an example of custom metrics. It declares a new class that implements the `CustomMetricTS` interface (line 3), with the method `evaluate_metric` that actually computes the measure (line 4) given the GT, the predicted anomalies and the matching strategy. An instance of the new metrics is instantiated (line 18) and added to the `Analyzer` module (line 19).

Listing 2: Custom metrics implementation

```
1 from TOOL.classes.TimeSeries import CustomMetricTS
2
3 class MyEvaluationMetric(CustomMetricTS):
4   def evaluate_metric(self, gt, predictions, matchingStrategy):
5     # Parameters:
6     #  gt: contains the GT anomalies
7     #  predictions: contains the predicted anomalies
8     #  matchingStrategy: the selected by the user
9
10    # Returns:
11    #  metric_value: the calculated value in the set
```

```
12
13      #TODO: call metrics computation code using the matchingStrategy
14      metric_value = #...
15      std_deviation = # only if apply
16      return metric_value, std_deviation
17
18  my_evaluation_metric = MyEvaluationMetric("my metric name")
19  my_analyzer.add_custom_metric(my_evaluation_metric)
```

## 4 Conclusions

This paper presents ODIN AD, a framework supporting the life-cycle of AD applications in the phases of data preparation and model evaluation and refinement. Data scientists can load multiple data sets and annotate each one with GT labels or with other meta-data, import the predictions made by the algorithms of their choice, and obtain per-algorithm and also comparative performance reports. In the refinement phase, they can break down the performance metrics based on multiple criteria, classify errors in user-defined types, and assess the impact of the various types of errors on each metrics. ODIN AD is open source and designed with an architecture that eases the customization of such aspects as the anomaly definition and matching strategy, the performance metrics, and the pre- and post-processors.

The future work will focus on extending the support for the analysis of multivariate time series with algorithms that exploit forecasting and reconstruction. This will require the design and implementation of multi-valued anomaly definition strategies for multi-variate time series in which the user can select the features and the distance function to exploit for the definition of the anomaly. We will also improve the support for the analysis of periodic time series, by implementing more robust approaches to the periodicity detection, such as the one described in [28]. Finally, we aim at integrating interpretability techniques, such as [14, 24, 30, 31], within the performance-oriented analysis functionalities of ODIN AD.

**Acknowledgements** This work is supported by the project PRECEPT - A novel decentralized edge-enabled PREsCriptivE and ProacTive framework for increased energy efficiency and well-being in residential buildings funded by the EU H2020 Programme, grant agreement no. 958284.

## References

1. Baidu: Curve. https://github.com/baidu/Curve, [Online; accessed 16-June-2022]
2. Carrasco, J., López, D., Aguilera-Martos, I., García-Gil, D., Markova, I., García-Barzana, M., Arias-Rodil, M., Luengo, J., Herrera, F.: Anomaly detection in predictive maintenance: A new evaluation framework for temporal unsupervised anomaly detection algorithms. Neurocomputing **462**, 440–452 (2021)

3. Chalapathy, R., Chawla, S.: Deep learning for anomaly detection: A survey. arXiv preprint arXiv:1901.03407 (2019)
4. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)
5. Choi, K., Yi, J., Park, C., Yoon, S.: Deep learning for anomaly detection in time-series data: Review, analysis, and guidelines. IEEE Access (2021)
6. Dickey, D.A., Fuller, W.A.: Distribution of the estimators for autoregressive time series with a unit root. Journal of the American statistical association **74**(366a), 427–431 (1979)
7. Gharibi, G., Walunj, V., Nekadi, R., Marri, R., Lee, Y.: Automated end-to-end management of the modeling lifecycle in deep learning. Empirical Software Engineering **26**(2), 1–33 (2021)
8. Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q.: On calibration of modern neural networks. In: International Conference on Machine Learning. pp. 1321–1330. PMLR (2017)
9. Haladjian, J.: The Wearables Development Toolkit (WDK). https://github.com/avenix/WDK (2019)
10. Herzen, J., Lässig, F., Piazzetta, S.G., Neuer, T., Tafti, L., Raille, G., Van Pottelbergh, T., Pasieka, M., Skrodzki, A., Huguenin, N., et al.: Darts: User-friendly modern machine learning for time series. arXiv preprint arXiv:2110.03224 (2021)
11. Hoiem, D., Chodpathumwan, Y., Dai, Q.: Diagnosing error in object detectors. In: European conference on computer vision. pp. 340–353. Springer (2012)
12. Hyndman, R.J., Athanasopoulos, G.: Forecasting: principles and practice. OTexts (2018)
13. Inc, G.: Trainset. https://trainset.geocene.com/, [Online; accessed 17-June-2022]
14. Jacob, V., Song, F., Stiegler, A., Rad, B., Diao, Y., Tatbul, N.: Exathlon: A benchmark for explainable anomaly detection over time series. arXiv preprint arXiv:2010.05073 (2020)
15. Kao, J.B., Jiang, J.R.: Anomaly detection for univariate time series with statistics and deep learning. In: 2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE). pp. 404–407. IEEE (2019)
16. Katser, I.D., Kozitsin, V.O.: Skoltech anomaly benchmark (skab). https://www.kaggle.com/dsv/1693952 (2020). https://doi.org/10.34740/KAGGLE/DSV/1693952
17. Keras: Keras documentation: Timeseries anomaly detection using an autoencoder. https://keras.io/examples/timeseries/timeseries_anomaly_detection/
18. Lavin, A., Ahmad, S.: Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark. In: 2015 IEEE 14th international conference on machine learning and applications (ICMLA). pp. 38–44. IEEE (2015)
19. Mahalakshmi, G., Sridevi, S., Rajaram, S.: A survey on forecasting of time series data. In: 2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16). pp. 1–8. IEEE (2016)
20. Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., Shroff, G.: Lstm-based encoder-decoder for multi-sensor anomaly detection. arXiv preprint arXiv:1607.00148 (2016)
21. Malhotra, P., Vig, L., Shroff, G., Agarwal, P., et al.: Long short term memory networks for anomaly detection in time series. In: Proceedings. vol. 89, pp. 89–94 (2015)

22. Marques, P., Dias, L., Correia, M.: Cybervti: Cyber visualization tool for intrusion detection. In: 2021 IEEE 20th International Symposium on Network Computing and Applications (NCA). pp. 1–9 (2021). https://doi.org/10.1109/NCA53618.2021.9685543
23. Microsoft: Tag anomaly. https://github.com/microsoft/TagAnomaly, [Online; accessed 16-June-2022]
24. Mujkanovic, F., Doskoč, V., Schirneck, M., Schäfer, P., Friedrich, T.: timexplain–a framework for explaining the predictions of time series classifiers. arXiv preprint arXiv:2007.07606 (2020)
25. Munir, M., Siddiqui, S.A., Dengel, A., Ahmed, S.: Deepant: A deep learning approach for unsupervised anomaly detection in time series. Ieee Access **7**, 1991–2005 (2018)
26. Murray, D., Stankovic, L., Stankovic, V.: An electrical load measurements dataset of united kingdom households from a two-year longitudinal study. Scientific data **4**(1), 1–12 (2017)
27. Pang, G., Shen, C., Cao, L., Hengel, A.V.D.: Deep learning for anomaly detection: A review. ACM Comput. Surv. **54**(2) (mar 2021). https://doi.org/10.1145/3439950, https://doi.org/10.1145/3439950
28. Puech, T., Boussard, M., D'Amato, A., Millerand, G.: A fully automated periodicity detection in time series. In: Lemaire, V., Malinowski, S., Bagnall, A., Bondu, A., Guyet, T., Tavenard, R. (eds.) Advanced Analytics and Learning on Temporal Data. pp. 43–54. Springer International Publishing, Cham (2020)
29. Ran, Y., Zhou, X., Lin, P., Wen, Y., Deng, R.: A survey of predictive maintenance: Systems, purposes and approaches. arXiv preprint arXiv:1912.07383 (2019)
30. Rojat, T., Puget, R., Filliat, D., Del Ser, J., Gelin, R., Díaz-Rodríguez, N.: Explainable artificial intelligence (xai) on timeseries data: A survey. arXiv preprint arXiv:2104.00950 (2021)
31. Schlegel, U., Vo, D.L., Keim, D.A., Seebacher, D.: Ts-mule: Local interpretable model-agnostic explanations for time series forecast models. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 5–14. Springer (2021)
32. Yilmaz, S.F., Kozat, S.S.: Pysad: a streaming anomaly detection framework in python. arXiv preprint arXiv:2009.02572 (2020)
33. Zhang, W., Yang, D., Wang, H.: Data-driven methods for predictive maintenance of industrial equipment: A survey. IEEE Systems Journal **13**(3), 2213–2227 (2019). https://doi.org/10.1109/JSYST.2019.2905565
34. Zoppi, T., Ceccarelli, A., Bondavalli, A.: Evaluation of anomaly detection algorithms made easy with reload. In: 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE). pp. 446–455. IEEE (2019)