AutoTSAD: Unsupervised Holistic Anomaly Detection for Time Series Data

Sebastian Schmidl
Hasso Plattner Institute,
University of Potsdam
Potsdam, Germany
sebastian.schmidl@hpi.de

Felix Naumann
Hasso Plattner Institute,
University of Potsdam
Potsdam, Germany
felix.naumann@hpi.de

Thorsten Papenbrock
Philipps University of Marburg
Marburg, Germany
papenbrock@informatik.uni-marburg.de

ABSTRACT

Detecting anomalous subsequences in time series data is one of the key tasks in time series analytics, having applications in environmental monitoring, preventive healthcare, predictive maintenance, and many further areas. Data scientists have developed various anomaly detection algorithms with individual strengths, such as the ability to detect repeating anomalies, anomalies in non-periodic time series, or anomalies with varying lengths. For a given dataset and task, the best algorithm with a suitable parameterization and, in some cases, sufficient training data, usually solves the anomaly detection problem well. However, given the high number of existing algorithms, their numerous parameters, and a pervasive lack of training data and domain knowledge, effective anomaly detection is still a complex task that heavily relies on manual experimentation.

We propose the unsupervised AutoTSAD system, which parameterizes, executes, and ensembles various highly effective anomaly detection algorithms. The ensembling system automatically presents an aggregated anomaly scoring for an arbitrary time series without a need for training data or parameter expertise. Our experiments show that AutoTSAD offers an anomaly detection accuracy comparable to the best manually optimized anomaly detection algorithms, and can significantly outperform existing method selection and ensembling approaches for time series anomaly detection.

PVLDB Reference Format:
Sebastian Schmidl, Felix Naumann, and Thorsten Papenbrock. AutoTSAD: Unsupervised Holistic Anomaly Detection for Time Series Data. PVLDB, 17(11): 2987 - 3002, 2024.
doi:10.14778/3681954.3681978

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://hpi.de/naumann/s/autotsad.

1 ANOMALY DETECTION IN TIME SERIES

A *data series* is an ordered sequence of real-valued data points recorded in equidistant intervals based on some continuous measure, such as temperature, mass, angle, position, or speed. If the order is based on time, the sequence is referred to as a *time series*. Because many data series analytics algorithms are agnostic to the reference measure, we use the term *time series* throughout this paper. While the data points of a time series can consist of multiple real-valued variables (*multivariate* time series), we focus on time series with only a single variable (*univariate* time series). "An *anomaly* in such a time series is either a point [...] or a sequence of points [...] that deviates significantly w. r. t. some measure, model, or embedding from the regular patterns of the time series." [61] The anomalies in a time series can differ in their lengths, magnitudes, and shapes. Figure 1 depicts a time series with three different typical anomalies: (i) A *flip* (reversal in time), (ii) an *amplitude* (scalar magnification), and (iii) an *outlier* (single anomalous value). In this example, each anomaly type occurs only once, but in many datasets, anomalies might re-appear and occur in different types.

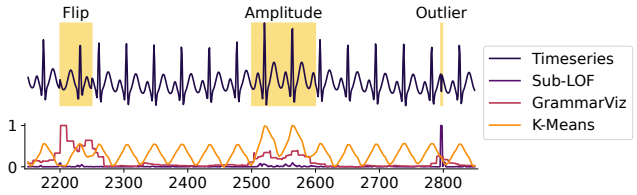

Figure 1: A time series with three anomalies. The anomaly scores of the three algorithms each mark a different anomaly.

Time series anomaly detection (TSAD) is an important task for many areas, ranging from healthcare monitoring [4, 22], over aircraft manufacturing processes [5, 74], network intrusion detection [40], and earth sciences [16] to finance applications [64]. Anomalies in time series can describe important events, such as structural defects, heart failures, adversarial attacks, earthquakes, or special environmental phenomena. These events need to be detected to act on them, prevent damage, or trigger further investigations.

Because detecting anomalies in time series is a difficult task that usually cannot be solved via visual inspection alone, researchers in different areas have developed a multitude of TSAD algorithms with different strengths and for different types of use cases [50, 61, 79]. Achieving good detection results on a given dataset in a non-specifically addressed domain is, however, still a challenge due to the manual experimentation needed for the following choices:

Algorithm selection: The applicability of a TSAD algorithm to a certain use case depends not only on different time series properties but also on the shape and types of the (yet unknown) anomalies [61]. Because analyzing time series properties is a complex task in itself, and the anomalies are unknown upfront, identifying an appropriate algorithm is challenging. Given the at least 175 TSAD algorithms with different strengths and requirements [50, 61, 79], practitioners need to test many algorithms to detect all anomalies.

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:10.14778/3681954.3681978

Algorithm parameterization: TSAD algorithms expose various hyperparameters, such as *window size*, *node degree*, *neighbor count*, and various *thresholds* [62]. The specification of these hyperparameters is difficult because (i) their meaning and implications are known only to domain experts, (ii) they usually have a significant impact on the algorithm's performance, (iii) they cover an (often infinitely) large domain of potential values, and (iv) their optimal values depend on the given time series and anomaly properties [62]. So even given the optimal algorithm for a specific anomaly detection setup, configuring that algorithm to detect the actually relevant anomalies remains a challenge.

Algorithm ensembling: Time series often contain different motifs and different types of anomalies (e. g. in Figure 1). The algorithm selection and parameterization, however, tailor the detection process to a specific target anomaly. To find *all* anomalies with potentially different properties and in potentially different contexts, multiple algorithm executions are needed. This demand is particularly apparent in scenarios with only little to no knowledge about the to-be-expected anomalies. An effective anomaly detection solution, therefore, needs to run an ensemble of appropriate algorithm configurations and intelligently combine the results [1, 13, 58].

Training data generation: In general, method selection techniques [68] and hyperparameter optimization (HPO) algorithms [20, 47] can (partially) solve the algorithm selection, parameterization, and ensembling challenges. They, however, require sufficient quantities of labeled training data, which are not available in typical anomaly detection scenarios. Using training data from known time series is also rarely possible due to the strong dependence of hyperparameter settings on the properties of a target dataset. Practical anomaly detection attempts, therefore, spend significant efforts on data labeling, which is not only expensive and time-consuming but also subjective and error-prone (as shown by the manually created labels in many real-world benchmark datasets [75]). Obtaining suitable training data in a systematic and preferably automatic way for a specific target dataset, therefore, remains a challenge.

To address all four challenges, we propose AutoTSAD, a novel, unsupervised system that automatically ensembles TSAD algorithms for a given univariate anomaly detection task. AutoTSAD can be applied without prior knowledge about the properties and anomalies of an input time series, and requires no configuration. The system automatically optimizes, executes, and ensembles multiple highly efficient, but specialized TSAD algorithms. The result of the anomaly detection is an aggregated anomaly score. Because each internal TSAD algorithm has a special view on the time series and a specific interpretation of anomalies, their ensemble can detect different types of anomalies. AutoTSAD also exposes the ranking of anomaly scores from the ensembled components for interactive exploration and explainability. Its technical contributions are:

(1) A **data generation** module that creates diverse training datasets for specific input time series with a novel regime creation procedure and a comprehensive anomaly injection strategy (cf. Section 4.2).

(2) An **algorithm optimization** module that creates promising algorithm configurations for a broad spectrum of anomaly types based on novel parameter seeding, study pruning, and instance selection strategies (cf. Section 4.3).

(3) A **scoring ensembling** module for TSAD algorithms calculates score ranks and a final combined score with a novel composition of scoring, ranking, and aggregation approaches for time series anomaly scores (cf. Section 4.4).

We evaluate AutoTSAD on 106 benchmark datasets and compare its anomaly detection quality to six baselines, including three anomaly detection ensembling methods [13, 58], a state-of-the-art TSAD method selection approach [29], and two synthetic baselines that summarize the results of 158 TSAD algorithms from a recent benchmark [61] and the performance of our base components respectively (cf. Section 5). The evaluation shows that AutoTSAD achieves significantly better results than state-of-the-art TSAD algorithms, ensembling techniques, and method selection approaches.

2 RELATED WORK

Time series anomaly detection (TSAD). Research in TSAD has a long history, and well over 150 algorithms have been proposed and surveyed [8, 11, 15, 19, 61]. The algorithms stem from various research communities, such as deep learning [31, 33, 36, 67, 76], classic machine learning [56, 77], outlier detection [12, 17, 46, 48], signal analysis [57, 70], or data mining [10, 38, 63, 81]. Due to the typical lack of training data, most of the algorithms are unsupervised (no training data) or semi-supervised (anomaly-free training data) [61]. Because anomalies are context-dependent and manifold, anomaly detection algorithms specialize on specific types of anomalies and time series and, thus, exhibit unique individual strengths. Existing benchmarks agree that there is no single universally best TSAD algorithm, parameterization is hard, and algorithm selection is challenging [35, 52, 61, 71]. For this reason, AutoTSAD uses a carefully chosen set of eight very well performing base algorithms with unique strengths and method family properties.

Algorithm Selection. Algorithm selection approaches, also called meta-learning approaches, try to identify the optimal algorithm for a given dataset based on different dataset characteristics [60]. Most approaches and, in particular, TSAD meta-learning techniques [68, 78, 80] rely on labeled datasets. They can, however, be applied only if labeled training datasets with representative anomalies are available, which is rarely the case. AutoTSAD, in contrast, does not require any external training data and solves not only the algorithm selection but also the algorithm parameterization task.

The unsupervised method selection technique TSADAMS [29] circumvents the use of training data with three surrogate metrics and Robust Rank Aggregation (RRA) to select the possibly best method. The base methods in TSADAMS are semi-supervised forecasting algorithms, which require training data without anomalies. Similar to TSADAMS, AutoTSAD also uses injected anomalies for performance assessment and algorithm selection. However, it uses *unsupervised* algorithms as base methods, regime segmentation in the data generation, algorithm hyperparameter optimization for fine-tuning, and algorithm ensembling (instead of selection), which results in clearly better detection scores in our evaluation.

Algorithm Ensembling. AutoTSAD is an unsupervised, heterogeneous, bias- and variance-reducing, model-centered ensembling system [1]. There exist many unsupervised outlier ensembling techniques of various types (sequential vs. independent, bias- vs. variance-reduction, data-centric vs. model-centric, …), but all of

these techniques have been proposed for relational data and point outliers [1]. The SELECT algorithm [58] is an unsupervised ensembling technique for outlier detection in point data, but in the same category as AutoTSAD. SELECT demonstrates that selectively combining the results of different outlier detectors yields superior performance compared to the individual approaches and standard ensembling techniques, as the combination reduces both variance and bias. SELECT's applicability to TSAD is, however, limited (cf. Section 5.2). AutoTSAD adopts SELECT's selective ensembling idea and applies it to time series (anomaly scores), but it utilizes proxy metrics computed on synthetic training data for the instance selection, it ranks algorithm instances instead of individual data points, and it uses score-wise instead of rank-wise combination methods. AutoTSAD is the first heterogeneous, bias- and variance-reducing ensembling approach designed for TSAD. In contrast to AutoTSAD and SELECT, also homogeneous ensembling approaches exist. Examples are Sub-IF [48] and deep-learning-based ensembles, such as RAE-Ensemble [41] or CAE-Ensemble [13]. Such ensembling approaches mainly reduce variance, which leaves room for improvement that AutoTSAD can successfully leverage (cf. Section 5.2). With Sub-IF, we show that homogeneous ensembling approaches can be used as an internal algorithm in AutoTSAD.

Hyperparameter optimization (HPO). AutoTSAD performs HPO to find suitable hyperparameter values for the system's base anomaly detection algorithms. HPO takes a labeled training data corpus, a target algorithm, and an optimization criterion as input; it, then, executes the algorithm on the data corpus with different hyperparameter values to maximize the algorithm's performance w. r. t. the optimization criterion. Most approaches choose values randomly (*random search*) [7], systematically and exhaustively (*grid search*) [24, 32, 44], or via exploration-exploitation-style sample generation (*Bayesian optimization (BO)*) [54, 65, 66].

Opprentice [47] and Isudra [20] are two supervised HPO systems for TSAD algorithms that, in contrast to AutoTSAD, require labeled training data. HYPEX [62] is an HPO framework that learns a parameterization model from time series characteristics; like AutoTSAD, it also alleviates the need for labeled training data, but it is less exact as it cannot specialize for specific (and different) types of anomalies. *Combined Algorithm Selection and Hyperparameter Optimization (CASH)* systems run the optimization of hyperparameters and the selection of an algorithm in one process [25]. AutoTSAD also serves both tasks, but instead of identifying a single best algorithm, it finds an ensemble of multiple, possibly diverse algorithms.

3 TIME SERIES AND ANOMALIES

AutoTSAD is a system that detects anomalies in time series datasets. To provide the formal background for the presentation of AutoTSAD, this section introduces the formal definitions for time series, subsequences, and anomalies.

As already stated in Section 1, time series are *univariate*, if they follow a single random variable, and *multivariate*, if they record multiple random variables. We focus on *univariate time series* and assume equidistant data points, which is a valid assumption for most anomaly detection scenarios. This assumption is inherited from the existing TSAD algorithms, which AutoTSAD uses internally; noncomplying data series need to be discretized. Local regions of a time series are called *subsequences*, and *anomalies* in a time series are regions, i. e., subsequences, that deviate from the norm:

Definition 3.1 (time series). A univariate *time series* $T \in \mathbb{R}^n$ is a sequence of real-valued points $T_i \in \mathbb{R}$, where $i \in |T|$ and $|T| = n$. We denote the length of the time series T as $|T|$ or n, and the i^{th}-point of the sequence as T_i.

Definition 3.2 (subsequence). A *subsequence* $T_{i,m}$ of a time series T is a continuous subset of the values in T starting from index i with length $m = |T_{i,m}|$: $[T_i, T_{i+1}, \ldots, T_{i+m-1}]$, where $0 \leq i \leq n - m$. Usually $m \ll n$. For a given time series T, the set of all subsequences in T of length m is defined as $\mathcal{T}_m = \{T_{0,m}, T_{1,m}, \ldots, T_{n-m,m}\}$.

Definition 3.3 (anomaly cf. [61]). A time series *anomaly* is a subsequence $T_{i,m}$ of length $1 \leq m < n$ that deviates w. r. t. some characteristic embedding, model, and/or similarity measure from frequent patterns in the time series T.

This anomaly definition covers subsequence anomalies as well as point anomalies, which are subsequences with a length of 1 ($m = 1$). The definition's dependence on some characteristic embedding, model, and/or similarity measure already shows the fuzzy nature of anomalies and, therefore, the need for holistic detection approaches.

Time series anomaly detection (TSAD) is the process of marking anomalies within a time series. Most TSAD algorithms compute anomaly scores to assess the abnormality of the points in the time series. However, the different method families use different ways to compute this anomaly score, such as probabilities, distances, or forecasting errors for points or subsequences. Thus, their meaning, range, and ability to contrast normal and abnormal points varies widely [42]. In line with related work [61], we use a point-wise unified result format for all internally used algorithms called *scoring* and transform the output of the algorithms into this format:

Definition 3.4 (scoring cf. [61]). Given a time series T, a time series *scoring* $S = [S_1, S_2, \ldots, S_n]$ with $S_i \in \mathbb{R}$ is the result of a TSAD algorithm that assigns each point $T_i \in T$ an anomaly score $S_i \in S$. For any two scores S_i and S_j, it must be true that if $S_i > S_j$, then T_i is more anomalous than T_j (in their respective contexts).

The transformation of a scoring into binary labels (0 for normal and 1 for abnormal points) is usually done by applying a threshold to the scoring in a final step. If needed, e. g., for ranking, we use the common thresholding strategy *2σ-thresholding* that computes a threshold $\theta_{2\sigma}$ from the mean μ and the standard deviation σ of the scoring S: $\theta_{2\sigma}(S) = \mu_S + 2 \cdot \sigma_S$. To measure the quality of an anomaly scoring independent of a specific threshold, we utilize the *Area Under the range-based Precision-Recall-Curve* [51] (RANGE-PR-AUC) metric. RANGE-PR-AUC extends the popular point-based *Area Under the Precision-Recall Curve* [21, 55] (AUC-PR) metric, such that it puts the same weight on all to-be-detected anomalies in one time series regardless of their lengths. AutoTSAD uses RANGE-PR-AUC to optimize TSAD algorithms, rank their results, and judge the final anomaly scores in our evaluation.

A *snippet* in a periodic time series is a subsequence with a specific, re-occurring pattern [34]. A time series that follows the same repeating snipped is called a *regime*. Regimes often describe states, such as heart rates at different activities. Because TSAD algorithms are most effective when tuned on specific regimes, we use regiming for training data generation.

4 AUTOTSAD

AutoTSAD is an anomaly detection system that utilizes a diverse ensemble of state-of-the-art anomaly detection algorithms for an exhaustive and unsupervised analysis of univariate time series. In this section, we introduce this novel system that, similar to existing TSAD algorithms, takes a single univariate time series T as input and produces an anomaly scoring S as output. The general anomaly detection approach works as follows: AutoTSAD, first, uses the input time series as a seed to generate a diverse set of training time series that cover the different motifs of the input time series and a wide range of potential anomalies. With this training data, our system, then, automatically parameterizes and selects promising algorithm configurations in a joint optimization attempt. These configurations, i. e., algorithm instances, are subsequently executed on the original input time series, resulting in a comprehensive set of more or less relevant anomaly scorings. Afterward, AutoTSAD ranks the scorings and combines the top-ranked results into a final aggregated scoring S. In addition to S, AutoTSAD can also provide a score ranking as output that users can interactively explore.

In this section, we give an overview of AutoTSAD's architecture (Section 4.1) and, then, explain AutoTSAD based on its sequentially executed modules: (i) *data generation* (Section 4.2), (ii) *algorithm optimization* (Section 4.3), and (iii) *scoring ensembling* (Section 4.4).

4.1 Architecture

AutoTSAD consecutively executes the three modules *data generation*, *algorithm optimization*, and *scoring ensembling*. As shown in Figure 2, these modules contain the following processing steps:

(1) The **data generation** (Section 4.2) module generates a diverse set of synthetic training time series by (i) analyzing the input time series for dominant periods and snippets that form representative subsequences, (ii) extracting different coherent regimes as base behaviors, (iii) cleaning the regimes to remove potentially anomalous subsequences, and (iv) injecting (potentially multiple) different anomaly configurations to produce labeled training time series. This leads to a set of up to 120 diverse training time series.

(2) The **algorithm optimization** (Section 4.3) uses the prelabeled training time series to create a pool of optimized, possibly diverse algorithm configurations by (i) seeding the optimization process with promising hyperparameter settings, (ii) applying Bayesian optimization (BO) for the hyperparameter and algorithm search, (iii) pruning unpromising or already fully optimized algorithm types, and (iv) pruning redundant training time series. The optimization module's output is a relatively small set of algorithm instances that together successfully solve all time series in the training corpus. The hyperparameter optimization process is an optional step in AutoTSAD (cf. dashed steps in Figure 2) because our default values already provide good results in practice.

(3) The **scoring ensembling** (Section 4.4) ensembles the final ranking by (i) executing the algorithm instances on the input time series, (ii) ranking the best algorithm instances, and (iii) aggregating their anomaly scorings to show them to the user.

AutoTSAD exposes various configuration options for expert users to tailor its behavior to specific requirements. Configuration options are, for example, the list of internal algorithms, the types of injected anomalies, the optimization metric, pruning switches, and limits for maximum training time series lengths or maximum number of optimization trials. For our evaluation and general usage of AutoTSAD, we, however, suggest a conservative default configuration that provides reliably good results (cf. Section 5).

Many steps of AutoTSAD can be executed in parallel. For these steps, AutoTSAD uses a (cached) pool of processes, distributes the tasks evenly across them, and collects all results in the main process when the tasks are finished. Figure 2 shows parallel steps, such as the *dataset analysis*, *anomaly injection*, or *algorithm instance execution*, as staggered boxes.

4.2 Data Generation

The first modules' goal is to produce, based on the input time series, a diverse set of training time series that can be used to optimize and select algorithm instances. To capture the variance of the input time series, we analyze the dataset (Section 4.2.1) and extract multiple representative regions from it (Section 4.2.2). After a thorough cleaning step (Section 4.2.3), we assume that the extracted regions consist of only normal subsequences. Because AutoTSAD does not know the properties of the anomalies in the input dataset, it injects (potentially multiple) different synthetic anomalies into the base time series (Section 4.2.4). In this way, the resulting training time series represent different motifs of the input time series and cover a wide variety of precisely labeled *potential* anomalies. AutoTSAD's training data generation approach is similar to data augmentation for self-supervised learning methods [36, 37]. However, AutoTSAD does not only inject synthetic anomalies, but also extracts and cleans different regimes of the input time series.

4.2.1 Dataset Analysis. Often, time series consist of multiple regions with different characteristics. For example, jet engine measurements differ at takeoff, climb, cruise, descend, landing, and taxi. The first step of the training data generation, therefore, reads the input time series, extracts the dominant period length, and discovers time series snippets, each based on prior methods. The period length is used in multiple places throughout the AutoTSAD system to improve the effectiveness and the efficiency of the processes. We use the period length (i) to set the window size parameter of the snippet discovery algorithm, (ii) to fix gaps in regimes and remove undersized parts of regimes, (iii) to set the window size parameter of the cleaning algorithms, (iv) to avoid creating too small training time series, (v) to set the length of the injected anomalies, and (vi) to initialize window size hyperparameters during HPO.

If the input time series consists of multiple regions with different characteristics, we need to generate training time series that cover all those regions. Hence, we use a snippet discovery algorithm to discover distinct representative subsequences in the input time series. If there are multiple different regions in the time series, we discover multiple snippets and use them to extract the different regions. In the following paragraphs, we explain the period detection and snippet discovery process in detail.

Period Detection. To estimate the dominant period length in the input time series, we use a simple, yet effective, feature extraction method from *tsfresh* [18] called `number_peaks`. This method counts the number of points in the input time series that have a larger value as their n left and right neighbors [18]. We use a conservative

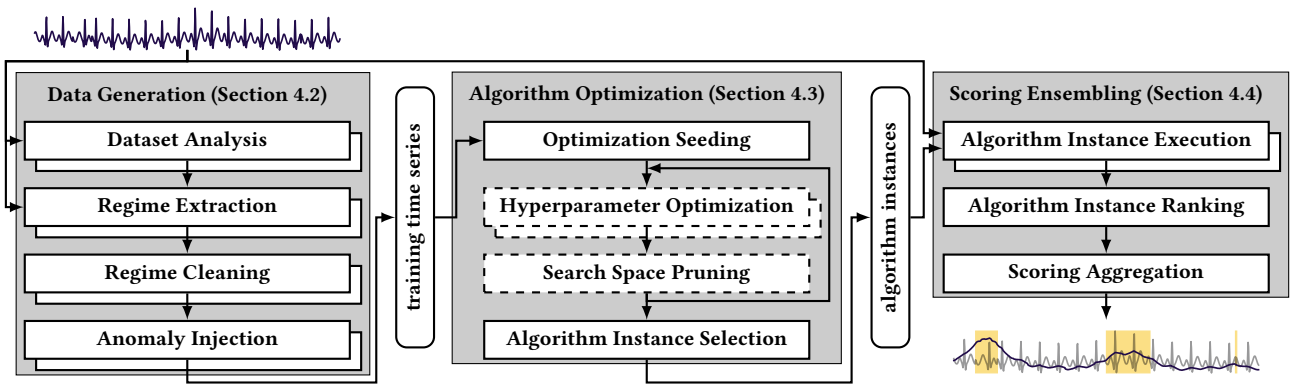

Figure 2: Overview of the AutoTSAD process with the three modules *Data Generation*, *Algorithm Optimization*, and *Scoring Ensembling*. Parallel steps are shown with staggered boxes and optional steps with dashed lines. The output of AutoTSAD is an aggregated anomaly scoring and optionally a ranking of anomaly scorings.

support of $n = 100$. To estimate the dominant period length, we devide the time series length by the number of peaks. If the time series has no significant periodicity, `number_peaks` detects very few to no peaks and the computed period size explodes. Because current TSAD algorithms cannot deal with very large window sizes, we limit the period size to below 600 data points, and ignore larger periods. AutoTSAD deals with this edge case by using fallback mechanisms in all places where a period length would be used. For example, we fall back to sampled fixed-length subsequences for the regime extraction process (cf. Section 4.2.2) and, as related work [61], use a default value of 100 for the window size parameters.

Snippet Detection. Time series *snippets* are distinct representative subsequences in a time series that balance *fidelity* (low distance to a part of the time series) and *coverage* (representative of large regions of the time series) [34]. Because they are the ideal tool to detect and extract potentially diverse regions, AutoTSAD extracts the k most representative snippets from our input time series. The optimal k in this extraction depends on the input time series and period length, and is determined automatically in the snippet detection process. For the implementation of this step, AutoTSAD uses the snippet discovery algorithm Snippet-Finder [34]. This algorithm executes a classic greedy search strategy to identify the k most representative snippets in the input time series (with a conservative $k_{max} = \delta_{max\_snippets} = 5$ because time series rarely have more than one snipped per dominant period length). Snippet-Finder leverages the following two data structures to guide the search: the *snippet profile* and the *snippet profile area*.

The *snippet profile* is a new (shorter) time series that captures the distance between a subsequence (the snippet) and all (sliding window) subsequences of the time series:

Definition 4.1 (snippet profile cf. [34]). Given a snippet $T_{i,m}$ as a subsequence of T, the *snippet profile* $P^{i,m} = \left[P_0^{i,m}, p_1^{i,m}, \ldots, p_{n-m}^{i,m}\right]$ is a sequence of distance values $P_j^{i,m} = MPdist(T_{i,m}, T_{j,m})$ computed between the snippet $T_{i,m}$ and all subsequences $\mathcal{T}_m$ of T using the *MPdist* distance measure.

The distances are computed using the robust *MPdist* distance measure [28]. For example, given the time series of Figure 3 (A), then Figure 3 (B) shows the top-5 snippet profiles with window size 100. The corresponding snippets are listed in the legend. The smaller the distance in the snipped profile is, the better its snipped describes the time series at the considered point. Hence, the snippet with the smallest distance (ideally close to zero) in some region describes that region best. The more snippets we consider, the better the set of snippets can approximate every segment in the time series. Because AutoTSAD requires only the most significant snippets, a selection is needed. Snippet-Finder considers the *snippet profile area*, which is the area under the element-wise minimum of all snippet profiles, to find a possibly small, but well enough snippet set:

Definition 4.2 (profile area cf. [34]). For a given list of snippets $TT = \left[T_{i,m} | 0 \leq i \leq n-m, 3 < m \ll n\right]$ of length $l = |TT|$ for time series T and their respective snippet profiles $PP = [P^{i,m} \; \forall T_{i,m} \in TT]$ also of length l, we can compute a new curve M as the element-wise minimum of all snippet profiles: $M = [M_0, M_1, \ldots, M_{n-m}]$, where $M_j = min\left(\left[P_j^{i,m} \; \forall P^{i,m} \in PP\right]\right)$. The *profile area* A is the (discrete) area under the curve M: $A = \Sigma_{j=0}^{n-m} M_j$.

With increasing k, the snippet profile area shrinks constantly. To determine a small number of snippets k that also possibly minimizes the snippet profile area, we execute Snippet-Finder with all $k \in \{1, ..., k_{max}\}$ and use the knee-finding algorithm on the change in the profile area over k [34, Sect. 3B]. Figure 3 (C) displays the change in the profile area when considering the top-k snippets compared to the top-$(k-1)$ snippets. The best k has the highest change in the profile area, thus, $k = 4$ is optimal for this example. The snipped detection finally forwards the top-k snippets with their profiles to the regime extraction.

4.2.2 Regime Extraction. After AutoTSAD analyzed the input time series, it extracts certain subsequences that correspond to potentially different base behaviors, such as takeoff, cruise, and landing of a plane. These *regimes* will later serve as the foundation for generating training time series. By default, we use the previously detected snippets and their profiles to extract various regimes from

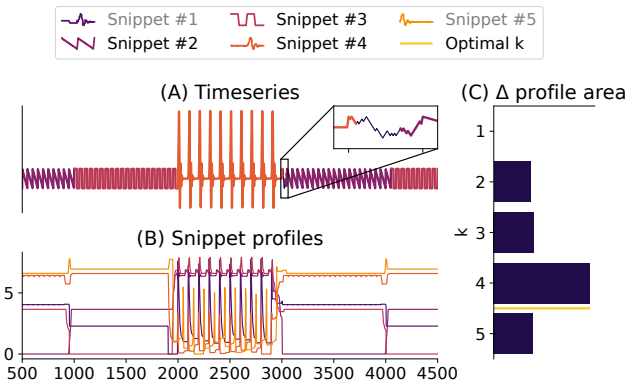

Figure 3: Regime extraction process using snippets of length $m = 100$ and $k_{max} = 5$: (A) Input time series partitioned into snippet-based regimes with a zoom-in on a subsequence that is not covered by the selected regimes. (B) Distance profile for each snippet. (C) Change in the profile area when comparing the top-k area to the top-$(k-1)$ area.

the input time series with a specific *snippet-based extraction* technique. However, if we could not extract any snippets or if there is only a single snippet, AutoTSAD falls back to a *sampling-based* strategy. In the following, we describe both the snippet-based and the sampling-based extraction in more detail.

Snippet-based Extraction. The snippet-based extraction uses the snippet profiles to identify subsequences of the time series that are covered by a specific snippet. We call all subsequences of the same snippet a *regime*. Given the snippet profiles PP for a period length m, we can easily calculate the profile area curve M as described in Definition 4.2. The element-wise minima in M partition the time series into up to k different snippet types. A coherent subsequence of a specific snippet might be longer or shorter. AutoTSAD extracts all these subsequences, groups them by snippet type, and concatenates the grouped sets to retrieve one possibly long regime for each snipped type. To make the described regime extraction process work in practice, we have to add a few additional considerations that we describe in the following.

Because the concatenation leaves sharp *cut points* between the concatenated subsequences that anomaly detection algorithms will identify as anomalous, AutoTSAD needs to track all cut points and deal with them appropriately: (i) When fetching an actual training time series from a regime, we choose a concatenated subsequence with a minimal number of cut points, (ii) when injecting anomalies into the regimes, we ensure an anomaly-free margin around the cut points, and (iii) when optimizing the TSAD algorithms, we mask the cut points, such that the algorithms do not see them.

If certain snippets were chosen badly or if the dataset is particularly chaotic, the regime extraction sometimes produces highly *fragmented regimes* with many cut points. To reduce regime fragmentation, AutoTSAD first identifies subsequences, which are shorter than one period length (technically $0.95 \cdot m$ or shorter). If such a short subsequence lies between two sufficiently long subsequences of another regime in the profile area curve M, AutoTSAD resolves the short subsequence by assigning it to the surrounding regime. The remaining, not re-assignable short subsequences, such as the zoomed-in subsequence in Figure 3 (A), are simply dropped, as the cut-point masking would ignore them anyway.

Despite the short subsequence removal, regimes might still be highly fragmented. To measure the degree of fragmentation in a regime, AutoTSAD measures every regimes' median number of consecutive periods. If the median is less than five consecutive periods ($5 \cdot m$), we consider the regime as highly fragmented. The threshold $5 \cdot m$ works well as a static decision criterion, because fragmentation is usually either very high (1-2 periods) or very low (hundreds of periods). AutoTSAD removes every highly fragmented regime and corresponding snippet. It, then, restarts the entire regime extraction with the reduced number of snippets. Snippet #1 in Figure 3 (B), for example, generates only one tiny regime at $T_{1900,100}$. Because the regime's size of 100 is below the $5 \cdot m = 500$ threshold, it is excluded entirely.

Sampling-based Extraction. If the entire time series consists of only a single normal behavior, AutoTSAD cannot do any better than taking a random subsequence for generating training time series. This *simple random sampling (SRS)* strategy, which has shown to be competitive in many domains [34], is also the fallback strategy if either the period detection or the snipped extraction produced empty results. SRS, then, randomly cuts at most two continuous regimes without overlap from the time series. The length of the extracted regimes is chosen 10% larger than the desired training data length (cf. Section 4.2.3) to allow some further reduction in the subsequent cleaning step. If the input time series is shorter than this length, AutoTSAD selects the entire time series without sampling.

4.2.3 Regime Cleaning. To generate well labeled training data from the regimes, AutoTSAD tries to avoid the real anomalies in the input dataset – otherwise, we might train the algorithms to ignore them. The short subsequence removal in the previous step has already removed some potential anomalies, but others may still prevail, especially when SRS was used. Thus, AutoTSAD employs an additional, aggressive cleaning step that removes possibly many anomalous points from our regimes.

For the cleaning, AutoTSAD executes our internal TSAD algorithms with their default parameters and the window sizes m and $\frac{m}{2}$ on each extracted regime. Then, it removes from every regime all points, for which a stable majority of at least 75% of the algorithm instances (with default parameters) agree that they are anomalous. An algorithm defines a point of a regime as anomalous by applying a threshold on its anomaly scoring to produce binary predictions; all points above the threshold are anomalous. Initially, AutoTSAD sets this threshold to the 90-th percentile of the scoring. To alleviate the impact of unreliable results that produce highly fragmented predictions with many short anomalous subsequences, AutoTSAD dynamically increases the threshold for some algorithms: It counts the predicted anomalies, which are contiguous subsequences of anomalously predicted points, and increases the threshold until the algorithm does not report more than half of the number of periods in the regime as anomalous. This dynamic threshold is purposefully aggressive but retains sufficiently large portions of the regimes for training. Wherever AutoTSAD removes points from a regime, it introduces new cut points.

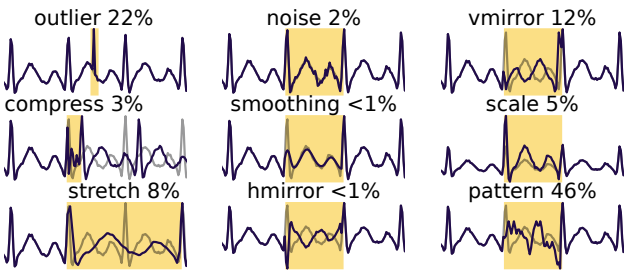

Figure 4: The synthetic anomaly types that can be injected into the regimes and their occurrences in the benchmark datasets; the original regimes are shown in light blue and the resulting training time series in dark blue.

What remains from the regimes is now trimmed to a maximum length l of 2000 or $10 \cdot m$ points, whichever is larger. For a useful trimming result, AutoTSAD extracts the subsequences of length l from each regime, such that it contains the *least number of cut points*. The overall length restriction is necessary, because AutoTSAD's algorithm optimization step is an iterative process and would starve on extremely long regimes, i. e., training time series.

4.2.4 Anomaly Injection. The final step in the data generation module is the injection of synthetic anomalies into the extracted, cleaned, and trimmed regimes. For each regime, AutoTSAD applies different anomaly injection configurations, which results in a rich set of training time series with different anomalies and base regimes.

The injected anomalies cover nine different types (see Figure 4): (i) deviation of a single point (*outlier*), (ii) reduction of the resolution (*compress*), (iii) increase in the resolution (*stretch*), (iv) addition of Gaussian or white noise (*noise*), (v) removal of small deviations (*smoothing*), (vi) reversal in the time axis (*vmirror*), (vii) mirroring on the mean value (horizontal axis) (*hmirror*), (viii) amplitude magnification or reduction by some factor (*scale*), and (ix) interference with one of ten different signals generated with, in our case, the GutenTAG time series generator [73] (*pattern*). These types cover all anomalies in our evaluation datasets and they also cover all anomaly types discussed in related works [29, 36, 37, 52, 73]. The injection process transforms the respective regimes and may slightly alter their lengths (e. g., for compress or stretch). The process configuration is also customizable, but we recommend using our default values, which have empirically shown to produce reliable and robust results; AutoTSAD's documentation explains the parameters and their default values in more detail. The default configuration uses all anomaly types and four representative anomaly lengths. Besides the configuration parameters, AutoTSAD determines the remaining anomaly properties, such as the anomaly positions, their strengths, and their order, via random sampling. The placement chooses random positions biased towards the middle of the time series, while enforcing margins around the anomalies and cut points. If the injection process cannot insert all desired anomalies due to a lack of viable positions, the current configuration is ignored. Because AutoTSAD injects for each anomaly type and each anomaly length one anomaly into each regime, the process generates up to 36 training time series per base behavior, of which up to $\delta_{max\_snippets}$ have been extracted from the input time series.

Table 1: All unsupervised TSAD algorithms in AutoTSAD with their method family, research area, supported dimensionality (*uni*- or *multi*variate), and programming language.

Algorithm	Area [61]	Family [61]	Dim.	Lang.
STOMP [81]	Data Mining	distance	uni	Python
k-Means [77]	Classic ML	distance	multi	Python
Sub-KNN [56]	Classic ML	distance	uni	Python
Sub-LOF [12]	Outlier Det.	distance	uni	Python
Sub-IF [48]	Outlier Det.	trees	uni	Python
GrammarViz [63]	Data Mining	encoding	uni	Java
Torsk [31]	Deep L.	forecasting	multi	Python
DWT-MLEAD [70]	Signal A.	distribution	uni	Python

4.3 Algorithm Optimization

The second module's goal is to build a set of promising algorithm instances that can successfully detect the synthetic anomalies in the diverse training time series, and by proxy work well on the input dataset. AutoTSAD is an ensembling system that builds on a carefully chosen set of eight base TSAD algorithms that perform well and have unique strengths. Table 1 lists the algorithms with their research area and method family. Note, though, that AutoTSAD allows the addition of more or different algorithms.

STOMP [81] efficiently computes the matrix profile, a vector representing the Euclidean distance between all z-normalized subsequences and their nearest neighbors. The distances in the matrix profile can be used as anomaly scores. *k-Means [77]*, *Sub-KNN [56]*, *Sub-LOF [12]*, and *Sub-IF [48]* slide a fixed-length window over the time series to generate subsequences. k-Means clusters these subsequences, and uses the distance to their nearest cluster center as anomaly scores, Sub-KNN computes the distance to the subsequences' k^{th} nearest neighbor, and Sub-LOF computes the local outlier factor (LOF) (a measure for how isolated a subsequence is from its local neighborhood). Sub-IF builds an ensemble of trees isolating the subsequences from each other; anomalies are easier to isolate and, thus, are close to the root. The reciprocal of the average path lengths from each subsequence to the tree roots are used as anomaly scores. *GrammarViz [63]* is based on symbolic discretization and grammar inference. It discovers variable-length time series patterns and uses the grammar rule coverages as anomaly scores. *Torsk [31]* is a deep learning algorithm that uses echo state networks to forecast time series' points; anomalies are harder to forecast, so their predictions deviate significantly from the observed values. The prediction errors are, therefore, used as anomaly scores. *DWT-MLEAD [70]* computes the discrete wavelet transform (DWT) over many levels and utilizes maximum likelihood estimation (MLE) to fit Gaussian distributions on windows of each level. For each point, the log-likelihoods of its windows in the different levels are aggregated to form anomaly scores.

To ensure that AutoTSAD can build a capable ensemble of optimized algorithm instances, we need to optimize the hyperparameters of the base algorithms on the training time series, choose the

best candidates, and measure their strengths for the ensembling. This is implemented in an efficient four-step HPO process: First, we initialize the search with intelligently chosen hyperparameter values (Section 4.3.1). Then, we run the hyperparameter search for a fixed number of trials (Section 4.3.2), prune the search space (Section 4.3.3), and repeat this step until convergence or the maximum number of trials is reached. At the end, we select the best algorithm instance for each training time series and compute its performance on all training time series as a *proxy* for its real performance on the actual input dataset (Section 4.3.4). Because we already have good heuristics to set the hyperparameters of our current selection of base algorithms (from TimeEval), HPO and search space pruning are optional (dashed steps in Figure 2).

4.3.1 Optimization Seeding. We initialize the hyperparameters of each algorithm with the manually optimized values from the TimeEval benchmark (including window size heuristics based on AutoTSAD's detected period sizes) [61, 73]. If we lack TimeEval heuristics for an algorithm, we use its default hyperparameter values.

4.3.2 Hyperparameter Optimization. AutoTSAD optimizes the hyperparameters of every base algorithm for every training time series separately and relies on different pruning techniques to reduce the search space (cf. Section 4.3.3). We use Bayesian optimization (BO) [54] to efficiently explore the search space and RANGE-PR-AUC as the optimization criterion.

AutoTSAD uses the Optuna library [3] for the implementation of the BO process with the CMA-ES (Covariance Matrix Adaptation Evolution Strategy) [30] for real-valued hyperparameters and the TPE (Tree-structured Parzen Estimator) [6] as a fallback for categorical and integer hyperparameters. AutoTSAD configures both sampling strategies to use 100 random guesses to explore the hyperparameter search space before exploiting existing trials to provide informed suggestions. Every combination of base algorithm and training time series is one independent Optuna study in AutoTSAD, for which the system performs a maximum of 800 trials. AutoTSAD stops the optimization process every 80 trials to prune non-promising studies. The first round is allowed 160 trials to have a better decision basis for pruning. AutoTSAD uses two stop conditions that are evaluated after each trial: A study is stopped when either the maximum number of trials (800) is reached, or when AutoTSAD found ten hyperparameter settings with a quality ≥ 0.95, which is sufficiently good considering that our training data also only approximates the real anomalies. The chosen BO configuration (100 guesses, 800 trails, 160 warm-ups, 10 stopping) is a fair compromise between runtime and effectiveness.

4.3.3 Search Space Pruning. Despite the initial optimization seeding step, all studies are processed independently of each other. We can, however, leverage information from already executed studies to prune entire studies from our search space. Algorithm configurations of pruned studies are not optimized, but they are still considered in the subsequent processing steps. After every round of optimization, AutoTSAD evaluates two pruning rules: *Algorithm pruning* stops the study of the worst performing algorithm for each dataset if still more than one algorithm per dataset remains. *Dataset pruning* stops all but one representative study for a group of similar datasets that are successfully solved by the same algorithm with

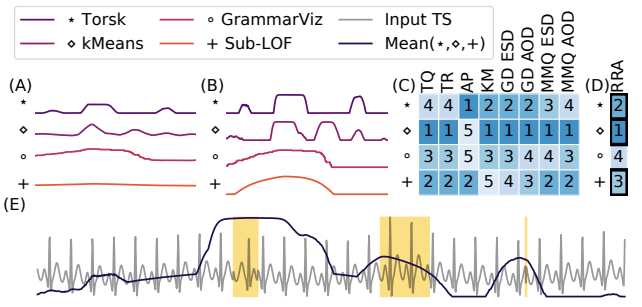

Figure 5: Scoring ensembling process with $k = 3$: (A) → (B) *Gaussian* scoring normalization, (B) → (C) algorithm instance ranking, (C) → (D) Robust Rank Aggregation, and, (B, D) → (E) *Mean* scoring aggregation.

the same hyperparameter configuration; at the end of the optimization process, AutoTSAD then considers the best hyperparameter configuration of the representative study as the best configuration for all datasets in the entire group if the configuration leads to a better detection quality than the initial configuration.

4.3.4 Algorithm Instance Selection. After all studies have been processed (or simply skipped), AutoTSAD builds a set of well-performing algorithm instances by selecting the best algorithm instance for each training time series. In this way, AutoTSAD translates the variance in the input time series and the variety of potential anomalies into a broad, but still concise ensemble of algorithm configurations. To increase the variance further and ensure that every algorithm is present at least once, AutoTSAD adds each algorithm with its default hyperparameter values if it is not yet present in the set. After selecting the best algorithm instances, AutoTSAD executes these algorithm instances on all training time series to compute proxy performance metrics. If an algorithm instance was already executed on a training time series during the previous optimization step, AutoTSAD uses the existing results. As proxy metrics, we compute for each algorithm instance the *mean quality* (RANGE-PR-AUC per default) over all training time series to capture the algorithm instance's absolute anomaly detection effectiveness and the *number of datasets*, for which this algorithm instance performed best, to capture the algorithm instance's relative effectiveness. AutoTSAD, finally, stores the set of top-performing algorithm instances and their proxy metrics for the ensembling.

4.4 Scoring Ensembling

The third module's goal is to compute the final anomaly scoring of AutoTSAD. This process is visualized in Figure 5 (A-E). AutoTSAD, first, executes the best-performing algorithm instances from the previous step on the actual input time series to retrieve their anomaly scorings (A) and normalize the scoring ranges (B) (Section 4.4.1). Then, it runs different algorithm selection and ranking methods that use the proxy performance and scoring diversity of the algorithm instances to create interesting score rankings (C). The rankings are, subsequently, aggregated into a single ranking (D) of up to size k (Section 4.4.2). In a last step, AutoTSAD computes a single final ensemble scoring S (E) by aggregating the scorings of

the k selected algorithm instances (Section 4.4.3). Because this process precomputes the scorings for all suggested ranking methods, it allows the user to further explore and tune the results interactively.

4.4.1 Algorithm Instance Execution. The first step takes all algorithm instances that survived the selection step and executes them on the entire input time series. The resulting anomaly scorings are, then, used in the next steps to capture diversity in the rankings.

The executed algorithms differ in the way they compute their anomaly scorings. For example, distance methods (STOMP, k-Means, Sub-KNN, and Sub-LOF) use the distance to a normal model as the anomaly score, while distribution methods (DWT-MLEAD) use a probability or likelihood as the anomaly score. These differences foster diversity in the score rankings, but they also necessitate that we calibrate and unify the score ranges before computing any diversity measures or aggregated scores [42]. To scale all scorings to be in $[0, 1]$, AutoTSAD uses *Gaussian normalization*, which is the preferred normalization method for outlier/anomaly scores [42] and also performed best in our evaluations. It converts any scoring S to probabilities S^* assuming a Gaussian distribution by transforming every score $S_i \in S$ to $S_i^* = \text{erf}\left(\frac{S_i - \mu_S}{\sqrt{2}\sigma_S}\right)$, where μ_S is the scoring's mean, σ_S is its standard deviation, and $\text{erf}()$ is the error function.

4.4.2 Algorithm Instance Ranking. The second step of the ensembling takes as input (i) the proxy metrics *mean quality* and *number of datasets* from the *Algorithm Instance Selection* step and (ii) the normalized scorings of the algorithm instances on the input time series from the *Algorithm Instance Execution* step. Because we found many interesting ranking methods for the anomaly scorings (and their respective algorithm instances), we take the following approach to create a single, possibly robust and diverse ranking: We first let AutoTSAD create multiple individual rankings using different ranking heuristics. Afterward, AutoTSAD aggregates all rankings via *Robust Rank Aggregation (RRA)* [29] into a single ranking. From this ranking, the top-k algorithm instances are selected for the final ensemble. We use $k = 6$ as default to potentially cover all algorithm families in one ranking. The ranking step in AutoTSAD covers six ranking methods that each implement a unique ranking heuristic:

(1) **Training Quality (TQ)** simply ranks algorithm instances based on their *mean quality* proxy scores. The algorithm instance with the highest mean quality on the training data is ranked first.

(2) **Training Result (TR)** ranks algorithm instances based on their normalized mean of both proxy metrics, *mean quality* and *number of datasets*. Again, high scores are ranked first.

(3) **K-Medoids Clustering (KM)** uses *k-medoids clustering* [53] with a scoring distance metric to create k diverse clusters of algorithm instances. AutoTSAD chooses k to equal the maximum number of algorithm instances in the ensemble and selects the cluster medoids as ensemble candidates. The medoids are, then, sorted descending by their proxy metrics.

(4) **Affinity Propagation Clustering (AP)** uses *affinity propagation clustering* [26] with a scoring distance metric to create diverse clusters of algorithm instances. Affinity propagation clustering determines the optimal number of clusters automatically, and AutoTSAD uses the algorithm instances belonging to the cluster centers as ensemble candidates. The selected algorithm instances are again sorted descending by their proxy metrics.

(5) **Greedy (GD)** is a ranking strategy comparable to *Farthest Point Sampling (FPS)* in point cloud analysis [45, 49] or image sampling [23] and, hence, tries to maximize the diversity in the ranked scorings. For initialization, the greedy strategy selects the scoring of the algorithm instance with the highest *number of datasets* proxy metric value as the first element in the ranking. Then, it iteratively adds those scorings to the ranking that maximize the scoring distance to all already contained scorings in the ranking. The process ends when all scorings are part of the ranking.

(6) **Maximal Marginal Quality (MMQ)** is comparable to the *Maximal Marginal Relevance (MMR)* criterion from information retrieval [14]. It tries to balance the quality and diversity of the scorings by also building the candidate list greedily. AutoTSAD uses *training-quality* as the quality measure and a scoring distance as the diversity measure. As the first element in the ranking, MMQ selects the algorithm instance with the highest quality. For all following elements, it iteratively chooses the algorithm instance that maximizes a weighted score of quality and maximum distance to the existing algorithm instances. For AutoTSAD, we propose a weighting of 30% quality to 70% diversity. The MMQ strategy finishes when all algorithm instances have been added to the ranking.

The different rankings are all biased towards specific metrics and diversity-creating selection strategies. AutoTSAD, therefore, uses RRA to combine the (noisy) rankings of all ranking methods into a single scoring ranking. More specifically, we use the *Minimum Influence Metric (MIM)* method [29, Section A.7.2], which works well for algorithm selection use cases [29]. It computes the empirical influence of each algorithm instance on an all-encompassing Borda ranking. The algorithm instances are, then, ranked with increasing influence: High influence indicates "bad" rankings, while low influence indicates "good" rankings. The RRA method is the default ranking technique in AutoTSAD, but AutoTSAD's user interface allows exploring the results of all ranking strategies.

The attentive reader might have noticed that the ranking strategies (3) to (6) rely on a *scoring distance*, which is a distance measure for two anomaly scorings. This distance can be calculated in various ways. In AutoTSAD, we utilize the two scoring distances, *Euclidean Scoring Distance* and *Annotation Overlap Distance*.

The *Euclidean Scoring Distance (ESD)* is computed directly on the point-wise anomaly scorings, which is possible because the scorings from different algorithm instances have the same length. After normalization in the previous step, the values are also in the same range and comparable to each other. Thus, we can simply apply the Euclidean distance function to each pair of scorings in the ensemble: $d_{ESD}(S^i, S^j) = \sqrt{(S^i - S^j)^2}$. The Euclidean distance captures small variations in the scorings and is sensitive to large differences in single points. This emphasizes point outlier detections in the rankings. However, also small differences for many points sum up over the time series, which might lead to the inclusion of non-optimal scorings in the rankings.

The *Annotation Overlap Distance (AOD)* uses the Jaccard index on the binary predictions received after applying 2σ-thresholding (cf. Section 3) on each anomaly scoring. We, first, compute the binary predictions P^j for all scorings S^j of the candidate algorithm

instances by $P_i^j = \mathbb{1}\left(S_i^j \geq \theta_{2\sigma}(S^j)\right)$ for all points $S_i^j \in S^j$. Then, we compute the Jaccard distance for a candidate pair of algorithm instances i and j using $d_{AOD}(P^i, P^j) = \frac{|P^i \cap P^j|}{|P^i \cup P^j|}$. The AOD is more robust to small changes in the scores compared to the ESD, and allows us to include algorithm instances in the ensemble that consider different parts of the time series as anomalous.

The ranking strategies (3) and (4) use ESD. For the ranking strategies (5) and (6), AutoTSAD considers both scoring distance metrics; thus, they actually produce four rankings. This leads to overall eight different ranking strategies plus their RRA.

4.4.3 Scoring Aggregation. Because rank-wise aggregation looses a lot of information [1], especially the temporal context, AutoTSAD aggregates the results of the algorithm instances score-wise to produce a final scoring S. Let $\mathcal{S}$ be the ranking of chosen scorings for the final aggregation with $|\mathcal{S}| = k$. All $S^i \in \mathcal{S}$ are already in $[0, 1]$ because the aggregation is performed after the unification of the anomaly scorings (cf. Section 4.4.1). This allows AutoTSAD to use either of two traditional aggregation methods: *Max* or *Mean*.

Max aggregation calculates the element-wise maximum anomaly score over all scorings. Hence, we aggregate each index i in the scorings as $S_i = max\left(S_i^0, \ldots, S_i^j, \ldots, S_i^k\right)$ for all $S^j \in \mathcal{S}$. While Max is bias-reducing, it can be unstable [1], and tends to produce scorings, where large regions have high scores and few scorings dominate the overall result.

Mean aggregation calculates the element-wise mean anomaly score over all scorings. For each index i in the scorings, we take the average of all algorithm instance's scorings $S^j \in \mathcal{S}$: $S_i = mean\left(S_i^0, \ldots, S_i^j, \ldots, S_i^k\right)$. Mean is variance-reducing [1], and we show in Section 5.4 that it is superior to Max aggregation.

5 EXPERIMENTAL EVALUATION

In this section, we evaluate AutoTSAD's performance in different settings and on various datasets. More specifically, we first evaluate the relevance of the hyperparameter optimization (Section 5.1), afterward, we compare AutoTSAD's detection quality to six state-of-the-art baselines (Section 5.2), then we evaluate the effectiveness of the data generation module (Section 5.3), and finally, we assess the differences among AutoTSAD's ensembling strategies (Section 5.4).

Hardware and Software. We perform all experiments in a Slurm-managed HPC lab. The compute nodes are equipped with Intel Xeon Gold 5220S or AMD EPYC 7742 CPUs and run Ubuntu Linux. We assign a single CPU, 20 GB of main memory, and a time limit of 12 hours to every job/execution, and disable AutoTSAD's parallelism. AutoTSAD is implemented in Python version 3.8.15 and uses OpenJDK version 11.0.20 to execute GrammarViz.

Baseline Algorithms. We compare AutoTSAD's anomaly detection accuracy to six baselines: *k-Means*, *Oracle*, *SELECT Vertical*, *SELECT Horizontal*, *CAE-Ensemble*, and *TSADAMS*. The *Oracle* baseline is a perfect selection algorithm that "magically" selects the best performing algorithm for every time series based on 71 carefully optimized TimeEval algorithms [61]; because Oracle uses ground truth information for the selection, it creates an upper bound for the anomaly detection quality. k-Means [77] is a TSAD algorithm with one of the overall best detection scores in the TimeEval study and the best performing base algorithm in AutoTSAD. All baseline algorithms and AutoTSAD's internal algorithms use the manually and dynamically optimized TimeEval hyperparameter settings (window sizes, for example, are set dynamically according to the inputs' period lengths). For its own hyperparameters, AutoTSAD uses the proposed static default settings. *SELECT* [58] is an unsupervised outlier ensembling technique that selects and aggregates the results from its base components using two different selection strategies: *vertical* and *horizontal*. We ported the author's Matlab implementation to Python and use AutoTSAD's base algorithms as the base components for the SELECT ensemble. *CAE-Ensemble* is a deep-learning based ensemble of convolutional autoencoders with a diversity-driven self-supervised learning scheme. We adapt the author's implementation to be able to use the test time series (without labels) during training and automatically execute the unsupervised hyperparameter selection process with ten parameter settings. CAE-Ensemble uses an NVIDIA TITAN X GPU. *TSADAMS* [29] is a state-of-the-art method selection technique for TSAD. It selects the possibly best model using RRA and various surrogate metrics. We use the author's implementation, including the proposed semi-supervised base algorithms. Because TSADAMS requires *forecasting approaches* as base algorithms, it cannot use AutoTSAD's base algorithms that cover all algorithm families.

Datasets. Our evaluation uses 106 univariate time series from 12 different dataset collections with varying characteristics that are available in our repository: SAND [10], GutenTAG [73], IOPS [59], KDD-TSAD [39, 75], MGAB [69], NAB [2], NASA-MSL [33], NASA-SMAP [33], WebscopeS5 [43], TSB-UAD-synthetic [52], TSB-UAD-artificial [52], and NormA [9]. For the SAND datasets, we select 11 time series from the different SAND categories and include time series, for which SAND performed particularly well and poor. For the remaining collections, we use the preprocessed datasets from TimeEval and sample 10 time series from each collection. We include all time series that (i) have at least one anomaly, (ii) have a contamination < 0.1, and (iii) could be solved by at least one algorithm with a RANGE-PR-AUC ≥ 0.6 in the TimeEval benchmark [61]. Some (filtered) collections contain fewer than 10 time series, so we use all remaining time series.

5.1 Hyperparameter Optimization

In Section 4.3, we claim that the use of pre-optimized hyperparameter values (e. g. TimeEval hyperparameter values) justifies skipping HPO and its large runtime overhead. To support this claim, we first exemplary show that the *Optimization* module, despite our runtime reduction efforts, makes up most of AutoTSAD's runtime. Then, we compare the anomaly detection quality of AutoTSAD with and without HPO for different default hyperparameter settings.

In Figure 6 *(left)*, we compare the runtime of AutoTSAD' major steps with HPO turned on ($+O$) and off ($-O$) for three different datasets. The measurements show that executing AutoTSAD without HPO can be orders of magnitude faster than running the full HPO process. With HPO, the HPO steps in *Algorithm Optimization* clearly dominate the runtime of the system, which is due to the many algorithm executions caused by the Bayesian optimization; the *Algorithm Optimization* causes some overhead even without

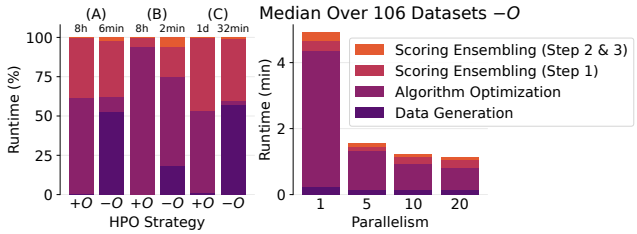

Figure 6: *(left)* runtime breakdown with optimization (+O) and without optimization (−O) for the datasets (A) `TSB-UAD-artificial-69_2_0.02_15`, (B) `KDD-TSAD-022`, and (C) `SAND-SED`; *(right)* median runtime over all datasets without optimization for different numbers of parallelism.

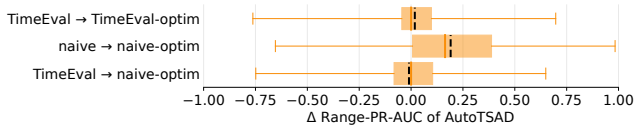

Figure 7: Improvement in RANGE-PR-AUC with and without HPO over all datasets. We compare four versions of AutoTSAD: *TimeEval* and *TimeEval-optim* use the default parameter values of AutoTSAD with the TimeEval heuristics; *naive* and *naive-optim* use random default values.

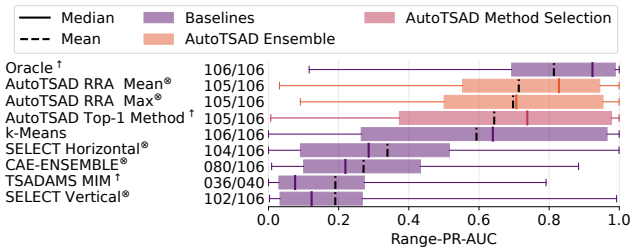

Figure 8: Anomaly detection quality (RANGE-PR-AUC) of AutoTSAD compared to the baselines over all successfully processed datasets (after method name); ↑ marks method selection approaches and ⊗ marks ensembling approaches.

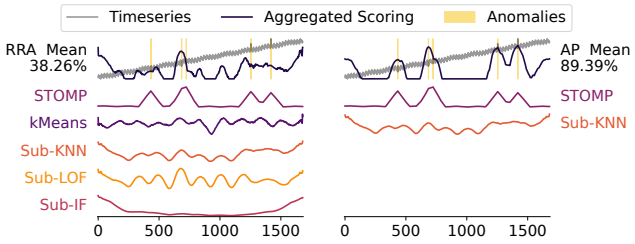

Figure 9: Two different rankings from AutoTSAD on the `A4Benchmark-2` dataset from the WebscopeS5 collection.

HPO due to the calculation of the proxy metrics. The *Algorithm Instance Execution* step, i. e., *Scoring Ensembling (Step 1)*, takes up the bulk of the remaining runtime. Note that this step also takes longer with HPO because HPO generates more algorithm configurations than just the TimeEval configurations. The *Algorithm Instance Execution* time with HPO turned off is the inevitable core activity of any ensemble (automatic or manual). Figure 6 *(right)* plots the median runtime of AutoTSAD over all datasets for different degrees of parallelism. The plot shows that AutoTSAD scales decently well with increasing parallelism, although the more expensive algorithms in the ensemble eventually cap the gains of additional processes.

Because the hyperparameter optimization is the by far most expensive step, we analyze its relevance. Figure 7 shows the improvement in anomaly detection quality achieved via AutoTSAD parameter optimization when using either the already well optimized TimeEval parameterization strategy or naive default values. The plots show that if we use TimeEval values, the mean RANGE-PR-AUC gain is negligible. For this reason and because executing AutoTSAD with HPO is about 40× slower than without HPO (mean runtime over all datasets: 67,840s > 1,712s), we can skip the optimization steps if the internal algorithms are executed with effective parameter selection heuristics. For naive default hyperparameters, though, HPO can significantly improve the detection quality. The plot also shows that the quality achieved with HPO-optimized naive values is on a par with non-HPO-optimized TimeEval values. Hence, we recommend executing AutoTSAD with the optimization steps, if new, not pre-optimized TSAD algorithms are used, and skip the step otherwise (as we do for our remaining experiments).

5.2 Anomaly Detection Quality

For the results shown in Figure 8, we executed all baseline algorithms and AutoTSAD (without HPO) on all 106 datasets, and measured the runs' RANGE-PR-AUC anomaly detection quality. Because TSADAMS's base algorithms require anomaly-free training data, the approach was executed on only 40 datasets that offer such training data. The box plots show the results for each algorithm over all successfully processed datasets, including the median (vertical line), the mean (vertical dashed line), and the minima and maxima (whiskers).

The baseline that always uses the best algorithm for each dataset (*Oracle*) solves the discovery task best, i. e., with a mean RANGE-PR-AUC > 0.8, but under unrealistic assumptions. By using the best base TSAD algorithm (k-Means) of AutoTSAD, we observe on average rather low scores compared to the optimal results. This is due to the algorithms' focus on specific types of anomalies. The SELECT ensembling baselines are (on average and by mean) worse than k-Means, despite k-Means being one of the ensemble's base algorithms. This is because SELECT considers each individual data point in the time series independently and, thus, looses the time context during score selection and aggregation. Despite the increased resource usage, CAE-Ensemble cannot achive competitive results.

AutoTSAD, which is shown with the two score aggregation methods Mean and Max, achieves significantly better results than the realistic baselines. Both variants have a mean and average RANGE-PR-AUC > 0.7. This demonstrates that AutoTSAD's ranking and aggregation techniques, which are based on the individual scores gathered from the semi-synthetic regimes, work well in

practice. Overall, Mean is superior to Max aggregation, confirming related work [1]. We note that, for a few time series, individual AutoTSAD ranking strategies can still not reliably detect the anomalies (whiskers reaching as low as 0.08) because the anomalies in these time series are particularly hard to detect. We display one such time series in Figure 9. Alternative AutoTSAD ranking strategies can usually detect these anomalies, though. Because users can create and explore rankings in AutoTSAD efficiently, namely without re-configuring and re-running TSAD algorithms, user-guided AutoTSAD runs can achieve even higher performances.

For comparison, we also show two method selection approaches: TSADAMS and *AutoTSAD Top-1 Method*, which simply uses the top-ranked algorithm instance from our RRA ranking. TSADAMS generally performs poorly because its base algorithms are very similar and often miss the anomalies in our datasets, despite that we share $\frac{1}{4}$ of the datasets with the original paper [29]. *AutoTSAD Top-1 Method* is significantly better than TSADAMS. AutoTSAD's aggregation techniques, however, show to be even more effective than selecting the top algorithm instance from the rankings, especially in the mean and minimum quality because in this way the system combines the strengths of different TSAD algorithms. Thus, we suggest an ensembling instead of a selection strategy.

5.3 Data Generation

To assess AutoTSAD's data generation process, we first evaluate the *Regime Cleaning* step by measuring how many of the real anomalous points it removes from a time series. On average, the cleaning step achieves a recall of 0.63 (median 0.87) and a precision of 0.12, which is in line with average TSAD performances. The cleaning step purposefully favors recall over precision because removing more points than necessary is not harmful.

To evaluate the regiming process, we executed AutoTSAD with *Dataset Analysis*, *Regime Extraction*, and *Regime Cleaning* turned on and off. When the steps are turned off, the average RANGE-PR-AUC decreased only from 0.71 to 0.70. This is because almost all datasets in existing benchmark collections exhibit only a single base oscillation. Hence, we also consider three specific datasets with actual regime shifts, from [73], [10], [39] respectively, and measure much higher RANGE-PR-AUC scores with the regiming steps (0.76, 0.63, 0.99) than without these steps (0.03, 0.43, 0.84).

To demonstrate the generalizability of the *Anomaly Injection* step, the next experiment samples 69 additional random datasets from our collections. With the additional dataset, AutoTSAD's average RANGE-PR-AUC decreases slightly from 0.71 (original 106 datasets) to 0.70 (all 175 datasets) while keeping the same relative improvement over its competitor approaches.

5.4 Ensembling Strategies

The next set of experiments investigates AutoTSAD's different ensembling strategies. The proposed eight ranking methods and two aggregation methods result in 16 different ensembling strategies. In addition, AutoTSAD uses RRA to combine all rankings into a *single ranking* for the two aggregation methods. We show all 18 ensembling strategies of AutoTSAD in Figure 10. The RRA strategies are highlighted in a darker color (see also Figure 8).

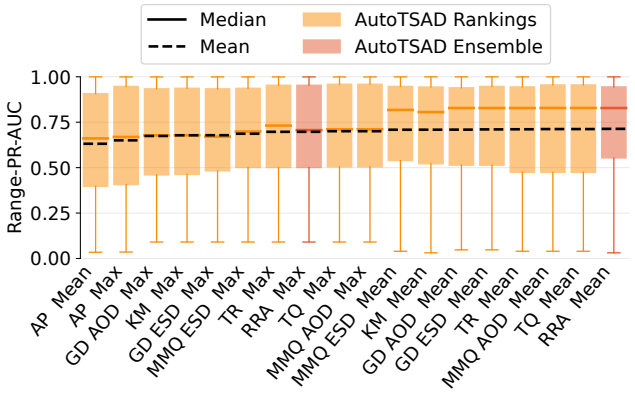

Figure 10: Comparison of the different ensembling strategies supported by AutoTSAD over all datasets.

The results confirm that Mean aggregation is on average and for most rankings superior to Max aggregation [1]. Mean aggregation, in general, has a significantly higher median performance than Max aggregation. In agreement with related work [29, 58], RRA Mean is better in mean and median over all datasets than the individual rankings that it combines. It, therefore, serves as our default setting.

We can see that some ensembling methods have specific strengths for some datasets, but overall, their performance is comparable. In Figure 9, we show a ranking of AutoTSAD RRA Mean that produces poor results and AutoTSAD AP Mean that produces much better results. Note that AP automatically determines the number of algorithm instances based on scoring similarities and, thus, chooses only two representatives. Because ranking strategies produce scorings of varying quality, their presence in AutoTSAD is an important feature for the user to find relevant anomalies. AutoTSAD makes this easy by providing an interactive user interface that displays not only the final scoring but also the ranked individual scores.

6 CONCLUSION

AutoTSAD is an unsupervised anomaly detection ensembling system that does not require labeled training data. It still offers various configuration and exploration options that either let the user trade precision for runtime or provide means to interactively ensemble the ranked scoring results. Due to the regime extraction and anomaly injection, AutoTSAD can deal with different motifs and anomaly types in the same input time series. The internal ensemble of TSAD algorithms covers an effective selection of anomaly detection approaches, which can easily be extended with additional algorithms, including future ones. We tested AutoTSAD for univariate time series with excellent results and plan to extend it for multivariate time series in future work. The most challenging part for this extension is the non-trivial extraction of regimes in multivariate data; with *correlation anomalies* [72] and multivariate TSAD algorithms [27, 71], the remaining steps work very similar.

ACKNOWLEDGMENTS

This work was funded by the German Federated Ministry for Economic Affairs and Climate Action (grant number 020E-100511083).

REFERENCES

[1] Charu C. Aggarwal and Saket Sathe. 2017. *Outlier Ensembles*. Springer International Publishing. ISBN: 978-3-319-54764-0. DOI: 10.1007/978-3-319-54765-7.

[2] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. 2017. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262, 134–147. DOI: 10.1016/j.neucom.2017.04.070.

[3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: a next-generation hyperparameter optimization framework. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

[4] Sardar Ansari, Negar Farzaneh, Marlena Duda, Kelsey Horan, Hedvig B. Andersson, Zachary D. Goldberger, Brahmajee K. Nallamothu, and Kayvan Najarian. 2017. A Review of Automated Methods for Detection of Myocardial Ischemia and Infarction Using Electrocardiogram and Electronic Health Records. *IEEE Reviews in Biomedical Engineering*, 10, 264–298. DOI: 10.1109/RBME.2017.2757953.

[5] Luis Basora, Xavier Olive, and Thomas Dubot. 2019. Recent Advances in Anomaly Detection Methods Applied to Aviation. *Aerospace*, 6, 11, 117. DOI: 10.3390/aerospace6110117.

[6] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[7] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research (JMLR)*, 13, 1.

[8] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A. Lozano. A review on outlier/anomaly detection in time series data. (2020). arXiv: 2002.04236 [cs, stat].

[9] Paul Boniol, Michele Linardi, Federico Roncallo, Themis Palpanas, Mohammed Meftah, and Emmanuel Remy. 2021. Unsupervised and Scalable Subsequence Anomaly Detection in Large Data Series. *The VLDB Journal*. DOI: 10.1007/s00778-021-00655-8.

[10] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J Franklin. 2021. SAND: Streaming Subsequence Anomaly Detection. *Proceedings of the VLDB Endowment (PVLDB)*, 14, 10, 1717–1729. DOI: 10.14778/3467861.3467863.

[11] Mohammad Braei and Sebastian Wagner. Anomaly Detection in Univariate Time-series: A Survey on the State-of-the-Art. (2020). arXiv: 2004.00433 [cs, stat].

[12] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 93–104. DOI: 10.1145/342009.335388.

[13] David Campos, Tung Kieu, Chenjuan Guo, Feiteng Huang, Kai Zheng, Bin Yang, and Christian S. Jensen. 2021. Unsupervised Time Series Outlier Detection with Diversity-Driven Convolutional Ensembles. *Proceedings of the VLDB Endowment (PVLDB)*, 15, 3, 611–623. DOI: 10.14778/3494124.3494142.

[14] Jaime Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the International Conference on Research and Development in Information Retrieval (SIGIR)*, 335–336. DOI: 10.1145/290941.291025.

[15] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Computing Surveys*, 41, 3, 1–58. DOI: 10.1145/1541880.1541882.

[16] Haibin Cheng, Pang-Ning Tan, Christopher Potter, and Steven Klooster. 2009. Detection and Characterization of Anomalies in Multivariate Time Series. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 413–424. DOI: 10.1137/1.9781611972795.36.

[17] Zhangyu Cheng, Chengming Zou, and Jianwei Dong. 2019. Outlier detection using isolation forest and local outlier factor. In *Proceedings of the Conference on Research in Adaptive and Convergent Systems (RACS)*, 161–168. DOI: 10.1145/3338840.3355641.

[18] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. 2018. Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package). *Neurocomputing*, 307, 72–77. DOI: 10.1016/j.neucom.2018.03.067.

[19] Andrew A. Cook, Goksel Misirli, and Zhong Fan. 2020. Anomaly Detection for IoT Time-Series Data: A Survey. *IEEE Internet of Things Journal*, 7, 7, 6481–6494. DOI: 10.1109/JIOT.2019.2958185.

[20] Jessamyn Dahmen and Diane J Cook. 2021. Indirectly supervised anomaly detection of clinically meaningful health events from smart home data. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 12, 2.

[21] Jesse Davis and Mark Goadrich. 2006. The relationship between Precision-Recall and ROC curves. In *Proceedings of the International Conference on Machine Learning (ICML)*, 233–240. DOI: 10.1145/1143844.1143874.

[22] Theekshana Dissanayake, Tharindu Fernando, Simon Denman, Sridha Sridharan, Houman Ghaemmaghami, and Clinton Fookes. 2021. A robust interpretable deep learning classifier for heart anomaly detection without segmentation. *IEEE Journal of Biomedical and Health Informatics*, 25, 6, 2162–2171. DOI: 10.1109/JBHI.2020.3027910.

[23] Y. Eldar, M. Lindenbaum, M. Porat, and Y.Y. Zeevi. 1997. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6, 9, 1305–1315. DOI: 10.1109/83.623193.

[24] 2019. *Hyperparameter optimization. Automatic Machine Learning: Methods, Systems, Challenges*. Springer Berlin Heidelberg.

[25] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2755–2763.

[26] Brendan J. Frey and Delbert Dueck. 2007. Clustering by Passing Messages Between Data Points. *Science*, 315, 5814, 972–976. DOI: 10.1126/science.1136800.

[27] Astha Garg, Wenyu Zhang, Jules Samaran, Ramasamy Savitha, and Chuan-Sheng Foo. 2022. An Evaluation of Anomaly Detection and Diagnosis in Multivariate Time Series. *IEEE Transactions on Neural Networks and Learning Systems*, 33, 6, 2508–2517. DOI: 10.1109/TNNLS.2021.3105827.

[28] Shaghayegh Gharghabi, Shima Imani, Anthony Bagnall, Amirali Darvishzadeh, and Eamonn Keogh. 2020. An ultra-fast time series distance measure to allow data mining in more complex real-world deployments. *Data Mining and Knowledge Discovery*, 34, 4, 1104–1135. DOI: 10.1007/s10618-020-00695-8.

[29] Mononito Goswami, Cristian Challu, Laurent Callot, Lenon Minorics, and Andrey Kan. 2023. Unsupervised Model Selection for Time-series Anomaly Detection. In *Proceedings of the International Conference on Learning Representations (ICLR)*. arXiv: 2210.01078 [cs].

[30] Nikolaus Hansen. 2023. The CMA Evolution Strategy: A Tutorial. arXiv: 1604.00772 [cs, stat]. Retrieved 09/13/2023 from http://arxiv.org/abs/1604.00772. preprint.

[31] Niklas Heim and James E. Avery. Adaptive Anomaly Detection in Chaotic Time Series with a Spatially Aware Echo State Network. (2019). arXiv: 1909.01709 [cs, stat].

[32] 2012. *A practical guide to training restricted boltzmann machines. Neural networks: Tricks of the trade*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-35289-8_32.

[33] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. 2018. Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 387–395. DOI: 10.1145/3219819.3219845.

[34] Shima Imani, Frank Madrid, Wei Ding, Scott Crouter, and Eamonn Keogh. 2018. Matrix Profile XIII: Time Series Snippets: A New Primitive for Time Series Data Mining. In *Proceedings of the International Conference on Big Knowledge (ICBK)*, 382–389. DOI: 10.1109/ICBK.2018.00058.

[35] Vincent Jacob, Fei Song, Arnaud Stiegler, Bijan Rad, Yanlei Diao, and Nesime Tatbul. 2021. Exathlon: A Benchmark for Explainable Anomaly Detection over Time Series. *Proceedings of the VLDB Endowment (PVLDB)*, 14, 2613–2626. DOI: 10.14778/3476249.3476307.

[36] Yungi Jeong, Eunseok Yang, Jung Hyun Ryu, Imseong Park, and Myungjoo Kang. 2023. AnomalyBERT: Self-Supervised Transformer for Time Series Anomaly Detection using Data Degradation Scheme. arXiv: 2305.04468 [cs]. Retrieved 04/03/2024 from http://arxiv.org/abs/2305.04468. preprint.

[37] Sheo Yon Jhin, Jaehoon Lee, and Noseong Park. 2023. Precursor-of-Anomaly Detection for Irregular Time Series. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 917–929. DOI: 10.1145/3580305.3599469.

[38] E. Keogh, J. Lin, and A. Fu. 2005. HOT SAX: efficiently finding the most unusual time series subsequence. In *Proceedings of the International Conference on Data Mining (ICDM)*. DOI: 10.1109/ICDM.2005.79.

[39] Eamonn Keogh, T Dutta Roy, U Naik, and Ankit Agrawal. 2021. Multi-dataset Time-Series Anomaly Detection Competition. (2021).

[40] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. 2019. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2, 20. DOI: 10.1186/s42400-019-0038-7.

[41] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2019. Outlier Detection for Time Series with Recurrent Autoencoder Ensembles. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2725–2732. DOI: 10.24963/ijcai.2019/378.

[42] Hans-Peter Kriegel, Peer Kroger, Erich Schubert, and Arthur Zimek. 2011. Interpreting and Unifying Outlier Scores. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 13–24. DOI: 10.1137/1.9781611972818.2.

[43] N Laptev, S Amizadeh, and Y Billawala. 2015. S5 - A labeled anomaly detection dataset, version 1.0 (16M). Yahoo, (2015).

[44] 2012. *Efficient backprop. Neural networks: Tricks of the trade*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-35289-8_3.

[45] Jingtao Li, Jian Zhou, Yan Xiong, Xing Chen, and Chaitali Chakrabarti. 2022. An Adjustable Farthest Point Sampling Method for Approximately-sorted Point Cloud Data. In *Proceedings of the IEEE Workshop on Signal Processing Systems (SiPS)*, 1–6. DOI: 10.1109/SiPS55645.2022.9919246.

[46] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. 2020. COPOD: Copula-Based Outlier Detection. In *Proceedings of the International Conference on Data Mining (ICDM)*.

[47] Dapeng Liu, Youjian Zhao, Haowen Xu, Yongqian Sun, Dan Pei, Jiao Luo, Xiaowei Jing, and Mei Feng. 2015. Opprentice: towards practical and automatic anomaly detection through machine learning. In *Proceedings of the Internet Measurement Conference (IMC)*.

[48] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *Proceedings of the International Conference on Data Mining (ICDM)*, 413–422. DOI: 10.1109/ICDM.2008.17.

[49] Carsten Moenning and Neil A Dodgson. 2003. Fast Marching Farthest Point Sampling. 562. University of Cambridge.

[50] Antonios Ntroumpogiannis, Michail Giannoulis, Nikolaos Myrtakis, Vassilis Christophides, Eric Simon, and Ioannis Tsamardinos. 2023. A meta-level analysis of online anomaly detectors. *The VLDB Journal*, 32, 4, 845–886. DOI: 10.1007/s00778-022-00773-x.

[51] John Paparrizos, Paul Boniol, Themis Palpanas, Ruey S. Tsay, Aaron Elmore, and Michael J. Franklin. 2022. Volume under the surface: a new accuracy evaluation measure for time-series anomaly detection. *Proceedings of the VLDB Endowment (PVLDB)*, 15, 11, 2774–2787. DOI: 10.14778/3551793.3551830.

[52] John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S Tsay, Themis Palpanas, and Michael J Franklin. 2022. TSB-UAD: An End-to-End Benchmark Suite for Univariate Time-Series Anomaly Detection. In *Proceedings of the VLDB Endowment (PVLDB)*, 2150–8097. DOI: 10.14778/3529337.3529354.

[53] Hae-Sang Park and Chi-Hyuck Jun. 2009. A simple and fast algorithm for K-medoids clustering. *Expert Systems with Applications*, 36, 2, 3336–3341. DOI: 10.1016/j.eswa.2008.01.039.

[54] Martin Pelikan, David E Goldberg, Erick Cantú-Paz, et al. 1999. Boa: the bayesian optimization algorithm. In *Proceedings the Genetic and Evolutionary Computation Conference (GECCO)*.

[55] Vijay Raghavan, Peter Bollmann, and Gwang S. Jung. 1989. A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems*, 7, 3, 205–229. DOI: 10.1145/65943.65945.

[56] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. 2000. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 427–438. DOI: 10.1145/342009.335437.

[57] Faraz Rasheed, Peter Peng, Reda Alhajj, and Jon Rokne. 2009. Fourier transform based spatial outlier mining. In *Proceedings of the International Conference on Intelligent Data Engineering and Automated Learning (IDEAL)*, 317–324.

[58] Shebuti Rayana and Leman Akoglu. 2016. Less is More: Building Selective Anomaly Ensembles. *ACM Transactions on Knowledge Discovery from Data*, 10, 4, 1–33. DOI: 10.1145/2890508.

[59] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. 2019. Time-Series Anomaly Detection Service at Microsoft. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 3009–3017. DOI: 10.1145/3292500.3330680.

[60] John R. Rice. 1976. The algorithm selection problem. In Advances in Computers. Volume 15. Morris Rubinoff and Marshall C. Yovits, editors. Elsevier, 65–118. DOI: https://doi.org/10.1016/S0065-2458(08)60520-3.

[61] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. 2022. Anomaly Detection in Time Series: A Comprehensive Evaluation. *Proceedings of the VLDB Endowment (PVLDB)*, 15, 9, 1779–1797. DOI: 10.14778/3538598.3538602.

[62] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. 2023. HYPEX: Hyperparameter Optimization in Time Series Anomaly Detection. *Proceedings of the Conference Datenbanksysteme in Business, Technologie und Web (BTW)*, 461–483. DOI: 10.18420/BTW2023-22.

[63] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P. Boedihardjo, Crystal Chen, and Susan Frankenstein. Time series anomaly discovery with grammar-based compression. OpenProceedings.org. DOI: 10.5441/002/edbt.2015.42.

[64] ChangMin Seong, YoungRok Song, Jiwung Hyun, and Yun-Gyung Cheong. 2022. Towards building intrusion detection systems for multivariate time-series data. In *Silicon Valley Cybersecurity Conference (SVCC)*.

[65] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: a review of bayesian optimization. In *Proceedings of the IEEE*.

[66] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[67] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan Pei. 2019. Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2828–2837. DOI: 10.1145/3292500.3330672.

[68] Emmanouil Sylligardos, Paul Boniol, John Paparrizos, Panos Trahanias, and Themis Palpanas. 2023. Choose Wisely: An Extensive Evaluation of Model Selection for Anomaly Detection in Time Series. *Proceedings of the VLDB Endowment*, 16, 11, 3418–3432. DOI: 10.14778/3611479.3611536.

[69] Markus Thill, Wolfgang Konen, and Thomas Bäck. MarkusThill / MGAB: The Mackey-Glass Anomaly Benchmark. Version v1.0.1. Zenodo. DOI: 10.5281/ZENODO.3762385.

[70] Markus Thill, Wolfgang Konen, and Thomas Bäck. 2017. Time Series Anomaly Detection with Discrete Wavelet Transforms and Maximum Likelihood Estimation. In *Proceedings of the International Conference on Time Series (ITISE)*.

[71] Dennis Wagner, Tobias Michels, Florian C F Schulz, Arjun Nair, Maja Rudolph, and Marius Kloft. 2023. TimeSeAD: Benchmarking Deep Multivariate Time-Series Anomaly Detection. *Transactions on Machine Learning Research (TMLR)*.

[72] Phillip Wenig, Sebastian Schmidl, and Thorsten Papenbrock. 2024. Anomaly Detectors for Multivariate Time Series: The Proof of the Pudding is in the Eating. In *Proceedings of the International Conference on Data Engineering Workshops (ICDEW)*. DOI: 10.1109/ICDEW61823.2024.00018.

[73] Phillip Wenig, Sebastian Schmidl, and Thorsten Papenbrock. 2022. TimeEval: a benchmarking toolkit for time series anomaly detection algorithms. *Proceedings of the VLDB Endowment (PVLDB)*, 15, 12, 3678–3681. DOI: 10.14778/3554821.3554873.

[74] Mark Woike, Ali Abdul-Aziz, and Michelle Clem. 2014. Structural health monitoring on turbine engines using microwave blade tip clearance sensors. In *Proceedings of the International Conference on Smart Sensor Phenomena, Technology, Networks, and Systems Integration (SPIE)*. DOI: 10.1117/12.2044967.

[75] Renjie Wu and Eamonn J. Keogh. Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress. (2020). arXiv: 2009.13807 [cs, stat].

[76] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. 2018. Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications. In *Proceedings of the International Conference on World Wide Web (WWW)*. International World Wide Web Conferences Steering Committee, 187–196. DOI: 10.1145/3178876.3185996.

[77] Takehisa Yairi, Yoshikiyo Kato, and Koichi Hori. 2001. Fault detection by mining association rules from house-keeping data. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (SAIRAS)*.

[78] Yuanxiang Ying, Juanyong Duan, Chunlei Wang, Yujing Wang, Congrui Huang, and Bixiong Xu. 2020. Automated Model Selection for Time-Series Anomaly Detection. arXiv: 2009.04395 [cs, eess]. Retrieved 07/17/2023 from http://arxiv.org/abs/2009.04395. preprint.

[79] Aoqian Zhang, Shuqing Deng, Dongping Cui, Ye Yuan, and Guoren Wang. 2024. An Experimental Evaluation of Anomaly Detection in Time Series. *Proceedings of the VLDB Endowment (PVLDB)*, 17, 3, 483–496. DOI: 10.14778/3632093.3632110.

[80] Peiyi Zhang, Xiaodong Jiang, Ginger Holt, Nikolay Pavlovich Laptev, Caner Komurlu, Peng Gao, and Yang Yu. 2021. Self-supervised learning for fast and scalable time series hyper-parameter tuning. *CoRR*, abs/2102.05740. ArXiv: 2102.05740.

[81] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk, and Eamonn Keogh. 2016. Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. In *Proceedings of the International Conference on Data Mining (ICDM)*, 739–748. DOI: 10.1109/ICDM.2016.0085.