A-DARTS: Stable Model Selection for Data Repair in Time Series

Mourad Khayati
Department of Computer Science
University of Fribourg
Fribourg, Switzerland
mourad.khayati@unifr.ch

Guillaume Chacun
Department of Computer Science
HES-SO Yverdon-Les-Bains
Yverdon-Les-Bains, Switzerland
guillaume.chacun@heig-vd.ch

Zakhar Tymchenko
Department of Computer Science
University of Fribourg
Fribourg, Switzerland
zakhar.tymchenko@unifr.ch

Philippe Cudré-Mauroux
Department of Computer Science
University of Fribourg
Fribourg, Switzerland
pcm@unifr.ch

***Abstract*—Time series often present gaps in the data. This phenomenon, also called *missing values*, is so prevalent that a cottage industry of missing-value imputation algorithms exists, each with different capabilities and efficacy/efficiency tradeoffs. So far, however, there has been no way to accurately select the most appropriate approach among all algorithms, given a new time series requiring repair.**

In this paper, we introduce a new configuration-free system, A-DARTS (for Automated DAta Repair in Time Series), to automatically select the best repair technique for a given faulty time series. A-DARTS's recommendation engine is trained via an iterative process that carefully learns the behavior of imputation algorithms using an extensive dataset of series that we curated. The selection process is made efficient by several new pruning techniques particularly adjusted to time series data. Applications that manipulate time series can now easily embed A-DARTS's recommendation engine and repair data on the fly. Our experiments show that our system picks, on average, the best imputation algorithm 20% more frequently than the best-in-class AutoML technique. Moreover, it produces stable recommendations across datasets by incurring 2.5x less error variance, eliminating the stability issue observed in all state-of-the-art methods we tested.

***Index Terms*—time series, data repair, imputation, missing values, model selection, feature extraction.**

I. Introduction

Time series data is the lingua franca of IoT devices in particular and sensors in general. Processing data from these devices is at the core of several applications, such as anomaly detection [1]–[3], forecasting [4], [5], data mining [6], [7], classification [8], similarity search [9], [10], to cite a few. In practice, IoT devices and sensors may suffer temporary failures in data transfer due to power loss or interference, leaving the resulting time series with a missing block of values. Processing these faulty time series is known to yield suboptimal or wrong results [11], [12].

Over the last decades, several imputation algorithms have been introduced to recover the missing blocks in time series [13]–[31]. These algorithms take a faulty series and, based on various replacement strategies, suggest surrogate values for the missing data portions. The critical aspect of these algorithms is that they aim to produce an imputation suggestion that does not mischaracterize the time series. Suggesting a verisimilar replacement for the missing block restores the time series' ability to be processed correctly.

This quest for verisimilarity is ultimately what fuels the diversity of imputation algorithms. A missing block in one such series should be replaced with values that reflect the series features, and no single algorithm has proven capable of producing good replacement values for a wide variety of cases. Finding the appropriate imputation is of paramount importance since it heavily impacts the quality of downstream tasks [15]. Our experimental results on various datasets show that the careful selection of imputation algorithms drastically improves time series forecasting by up to 80%. To guide the choice of existing algorithms, there exist benchmarks and evaluation studies that determine which algorithm does well and in which circumstances [32]–[34].

Those works, however, are all qualitative as they only provide guidelines for algorithm selection. Instead, we seek a method that, given a new time series with missing blocks, automatically recommends the best imputation algorithm to fix it. Since the incomplete time series is unknown, this problem can intuitively be solved using classifiers. In other words, a classifier can be trained over the results of each imputation algorithm on a large number of time series and then used to recommend the best imputation class for the new series. Unfortunately, this alternative is known to yield subpar results, as the performance of classifiers often varies from one dataset to another depending on the features carried by the series [35], [36]. We will discuss this observation in more detail later.

AutoML methods [37] have recently emerged to cope with the lack of stability of classifiers. Simply put, AutoML methods are expressed as "just add water" frameworks that aim to find the most adequate models, classifiers in our case, and their

hyperparameters for a given task [38]–[41]. These techniques essentially pitch different models against one another, pruning poor-performing ones along the way and eventually selecting the best model. Existing AutoML techniques are instrumental for recommendation problems similar to ours, but they suffer a major drawback: only one model survives the search. As stated above, no known model performs best in all scenarios, thus leading to poor recommendations. Those frameworks cannot find more than one "winning" model for two reasons. First, their search strategy explores a reduced space of model configurations, which does not consider time series features. Second, they lack an adaptive filtering that allows variations of the same model to survive. Due to those challenging aspects, the automated selection of time series repair techniques remains an open problem.

Our paper aims to solve the limitations of existing AutoML solutions using a new method specialized for time series imputation. Our framework, called A-DARTS, achieves efficacy and efficiency thanks to (a) a novel fine-grained navigation strategy to pitch models against one another that allows for more than one winning model, (b) a new feature selection mechanism, and (c) highly-curated training datasets to use as ground truth. We methodically gather 67K real-world time series from various applications that cover a significant range of values for all the needed features.

The A-DARTS framework is systematically more accurate for time series imputation in all the cases we tested, and it presents 2.5× less error variance than the best existing AutoML frameworks available. A-DARTS is also extremely fast and is almost instantaneous concerning the time it takes to make a recommendation. The source code of A-DARTS and the training datasets are open-sourced. Any other application can easily embed the model that results from A-DARTS's training, along with a selection of algorithm implementations, thus acquiring the ability to perform automatic time series imputation. To the best of our knowledge, no other method exists for this task that is as precise and efficient as A-DARTS. In summary, we make the following contributions:

- We propose a time-series-specialized AutoML system called A-DARTS to find the best imputation algorithm.
- We introduce ModelRace, the model selection component of A-DARTS, which utilizes a two-phase pruning approach to identify the optimal model.
- We devise two additional components to characterize incomplete time series and speed up their labeling with imputation algorithms.
- We conduct a comprehensive evaluation of A-DARTS, demonstrating that it significantly outperforms state-of-the-art comparable systems across various datasets, particularly by providing much tighter bounds on the variance of recommendation quality.

II. Background and Motivation

In this section, we discuss the variety of available imputation algorithms and the inherent difficulties of matching them to the use cases where each performs best.

Imputation Algorithms and Dataset Labeling. ImputeBench [32], [42] is the most comprehensive benchmark to date for time series imputation algorithms. It brings together a large variety of the most advanced imputation algorithms using the same code base and presents parameterized versions of each that strike a balance between accuracy and runtime. ImputeBench considers many different patterns for missing blocks. The idea is that different shapes of missing blocks may require different algorithms when occurring on multiple time series. The imputation algorithms we consider in this paper are all covered by the ImputeBench framework [43].

While ImputeBench does not recommend algorithms per se, it can generate labeled data for training a recommendation model. A naive solution would run a large variety of faulty time series through the algorithms suggested by the benchmark and obtain, for each series, how the measured algorithms performed relative to one another. Although inefficient—a large number of series would be necessary to cover the many particular scenarios where missing blocks occur—this annotated dataset is the starting point for our problem analysis. In the latter sections of the paper, we discuss obtaining a similar annotated dataset with a fraction of the effort.

The Model Selection Problem. With such a labeled set, a classifier could be trained to learn the circumstances under which each imputation algorithm performs well. Presumably, given a metric of time series similarity, the best algorithm for the new series should be the one that performed best for the series used in the training set. Many candidates exist for such a model, e.g., kNN classifier [44]. This model would use nearest-neighbor search to match the features of the new faulty series with previously known series features.

We evaluate whether kNN could be an adequate model in this case by testing its efficacy—how precise its predictions are—in distinct scenarios. We compare kNN against two commonly used classifiers, Multi-layer Perceptrons (MLP) [45] and CatBoost [46], by applying a configuration that seems sensible for the three classifiers. We report the average results from six different dataset categories: households' electricity consumption (Power), water quality measurements in rivers (Water), motion sensors in humans performing different activities (Motion), weather phenomena in different Swiss cities (Climate), electromagnetic events associated with storms (Lightning), and human health-related data (Medical). Each category includes multiple datasets with thousands of time series as described in Section VII.

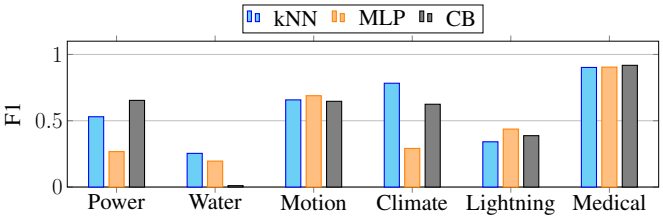


Fig. 1: Classifier Performance on Six Dataset Categories.

TABLE I: Comparison of existing techniques. ✓: Supported. (✓): Requires non-trivial extension. ✕: Not supported.

Technique	Low Resources	Model Configuration			Data Features	
		multiple models	multiple instances	multiple winners	extraction	scaling
FLAML [40]	✓	✓	✕	✕	(✓)	✕
Tune [39]	✓	✕	✕	✕	(✓)	✕
AutoFolio [38]	✓	✕	✕	✕	(✓)	✕
RAHA [47]	✕	✓	(✓)	✕	✓	✕
A-DARTS	✓	✓	✓	✓	✓	✓

The results in Figure 1 show that kNN outperforms the other classifiers for series from water and climate datasets. It does less so for the other profiles. We also observe the same phenomenon with the two other classifiers. No model in this experiment performs consistently better in all cases. At first, we suspected that the classifiers were not adequately configured. There are two aspects to consider in this context: a classifier can be naturally parameterizable, such as defining the 'k' in kNN or the number of branches in a decision tree; and the feature dimensions should be somehow normalized. Regarding the latter, some features are categorical, while some are numeric. Even among numeric features, some range from 0 to 1, while others may range from -4'728 to 6'217. To calculate meaningful Euclidean distances in a feature space, we should assign weights to each dimension and normalize the feature space.

The combination of available classifiers, their potential parameterization, and the myriad of different feature scalers open a considerable search space for model selection. The manual exploration of this large space is prohibitively expensive.

AutoML Techniques. Fortunately, efficient exploration techniques exist. AutoML [37], [48] has recently emerged as an automated solution to find the most suitable model for a given machine learning task. It encompasses various techniques, but in our case, we are interested in the space search mechanism known as Model Selection. Specifically, we aim to discover an effective combination of a classifier, its hyperparameter configuration, and a feature scaler (a weight assigned to each feature). In the AutoML literature, this combination is commonly referred to as a *pipeline*.

In general, automatically performing Model Selection requires three components: (a) a *pipeline synthesizer* that generates candidate pipelines, (b) a *pipeline filtering* strategy that eliminates pipelines that do not meet the desired performance metrics, and (c) a *search strategy* that feedbacks into the synthesizer, guiding it on the types of pipelines to generate next. We apply this approach in our case as follows. A synthesizer generates an initial set of pipelines that the evaluator then tests. The testing involves two phases: training the classifier on a portion of the annotated data, e.g., 80%, and assessing its performance using the remaining data, e.g., 20%, as ground truth. Based on the evaluator's scores, the search strategy might determine that exploring slight variations of kNN, for instance, could be beneficial. It then directs the synthesizer to focus on generating more of those. This cycle continues until a specified time limit or performance threshold is reached.

III. Related Work

Several AutoML frameworks are available that purport to address generic recommendation tasks. They essentially differ in how they approach pipeline synthesis, evaluation, and search strategy. We apply those frameworks to solve our task by feeding them with the labeled data we generate and the time series features we extract. We discuss next what aspects of these frameworks make them unsuitable for our use-case and empirically compare their performance against A-DARTS in Section VII.

Table I provides a comparative summary of the existing techniques we compare against, highlighting how A-DARTS enhances the model selection landscape. The first column "Low Resources' assesses the computational efficiency of each method. The second column "Model Configuration" indicates whether the methods consider various input classifiers, allow multiple instances of the same classifier to survive, and incorporate a voting mechanism to choose among multiple winners. The third column "Data Features" specifies whether the techniques include a feature extractor and feature scaling in the search space. A brief description of each technique is provided below.

FLAML is a lightweight and easy-to-use AutoML framework from Microsoft [40]. It configures multiple classifiers at a time and selects a single winner pipeline. FLAML considers all variations of a given classifier to be the same pipeline. In other words, when it discards a given pipeline, it eliminates the chances of any more helpful variation being selected. Similarly to our system, FLAML generates the configurations on-the-fly by expanding the parameter space. It builds a tree-like representation of configurations and more eagerly explores the branches expected to find better parameters. Each branch represents a different classifier; thus, a unique configuration survives the race. The configurations are iteratively trained on a random sample of the data, and a corresponding cost value that combines error and time is computed. FLAML compares the cost produced by each configuration, and the training sample is resized based on the cost improvement between consecutive iterations.

Tune [39] is a model selection system that configures a single classifier at a time (hand-picked by the user). It pre-generates configurations evaluated using various search algorithms such as Hyperband [49]. The parameter search starts with a large set of randomly generated configurations and iteratively decreases the size of the set. Each iteration uniformly allocates a time budget to each configuration,

evaluates its performance, and discards the worst half until one configuration survives. The surviving configurations from each iteration are compared against one another, and the best-performing one is returned.

AutoFolio [38] is yet another system that configures a single classifier by pre-generating a list of parameter configurations. The parameter exploration begins by generating random seed configurations and iteratively introducing perturbations to only one parameter of the configuration at a time. The updated configurations are evaluated on different data partitions over a dynamic time budget. The configuration that does not improve the performance is discarded, and the one that yields the best average performance across different data partitions is recommended.

RAHA [47] is a model selection system for error detection in relational data. It compares the performance of multiple classifiers and recommends the configuration that best predicts if an unseen tuple is erroneous. RAHA extracts basic statistical features of the data such as mean, variance, etc., and uses them to compute the similarity between the columns of the table. A different classifier is trained for each cluster of similar columns. The training is executed on a fraction of data clusters that are already user-labeled with the best detection technique. To apply RAHA to our problem, we merge RAHA's features with the ones we extract. We also adjust its objective function by using the inverse of the Root Mean Squared Error (RMSE) instead of the F1 score when computing a strategy's score, as the latter needs to be maximized.

A handful of model selection techniques for other time series tasks exists. Unfortunately, none of those techniques can be applied to our case as they operate on a single time series partitioned into smaller subsequences.

Time series forecasting is one of the most common fields where several model selection techniques have been proposed [4], [36], [50], [51]. Those techniques operate in the same vein, using predictors instead of classifiers. AutoAI-TS [4] is a seminal work in this context, which trains the predictors on some time series subsequences and infers their performance for the remaining ones. A linear regression model is applied to produce an upper-bound estimate of the full training performance of the used predictors. The surviving predictor will be used to forecast future values.

Anomaly detection in time series is another application for model selection. In [35], the authors introduce a benchmark for model selection techniques. They train three categories of classifiers on time series subsequences labeled with various anomaly detection techniques. The classifiers are pre-configured with their default values. The authors compare the performance of the classifiers and recommend, for each type of anomaly, the best-preforming one.

Addressing the issues of existing model selection techniques entails navigating significantly larger pipeline search space. We achieve this through novel, specialized search and pruning strategies, implemented within an imputation-centric AutoML framework that we refer to as A-DARTS.

IV. System Overview

A-DARTS's goal is to take a time series with missing blocks and to recommend the best imputation technique to recover them. The two most salient features of the framework were motivated by our preliminary experiments: (a) the recommendation is made by a voting process as opposed to a single classifier, and (b) the pruning performed by the framework is on a pipeline level rather than on a classifier one. Figure 2 presents A-DARTS's main components and how they interact. Our framework undergoes a one-time training phase before it can be used to produce recommendations. We discuss the training process in Section IV-A and the recommendation process in Section IV-B.

A. Training

The training phase revolves around A-DARTS's model selection process, called ModelRace. The process requires having labeled data available that associates time series/missing block configuration with an imputation algorithm. Obtaining this annotated data would be prohibitive in time if applied to each time series in a dataset separately. Instead, we propose to cluster the series according to their similarity and propagate the labels within the clusters (step 1 in Figure 2). The detailed clustering process will be discussed in Section VI. Once the data is labeled, our system extracts the time series features (2) needed to identify the best imputation technique.

Internally, ModelRace implements an iterative process to select the best possible pipelines. The process starts with an initial set of pipelines called seed. For initialization, the seed must contain at least one pipeline per classifier type that needs to be considered. The reason is that ModelRace will try to synthesize and evaluate variations of the seed pipelines, each containing different parameterization and feature scaling configurations. To do so, it uses a component called a synthesizer (3) to generate derived pipelines from existing ones. Each new pipeline is then evaluated, i.e., trained on labeled data, and then scored according to its efficacy by a scoring component in ModelRace (4). The less-performing pipelines are filtered out by a pruning mechanism (5) before the process starts over. We explain the process in more detail in Section V.

B. Inference

The algorithm performing imputation on a faulty time series can be obtained by extracting the series' features. To that end, we use the same feature extractor as used in training (6) and feed that information to the winning pipelines. To select the algorithm, a voting mechanism is applied to aggregate the recommendations of the selected pipelines (7). We experimented with several aggregation functions. We empirically find that using a "soft" voting mechanism provides higher scores with most of the classifiers compared to using a standard majority voting algorithm.

The voting procedure computes a matrix of scores where each entry represents the probability of a given imputation algorithm being chosen by the selected pipelines. It then aggregates results by averaging the probabilities of different

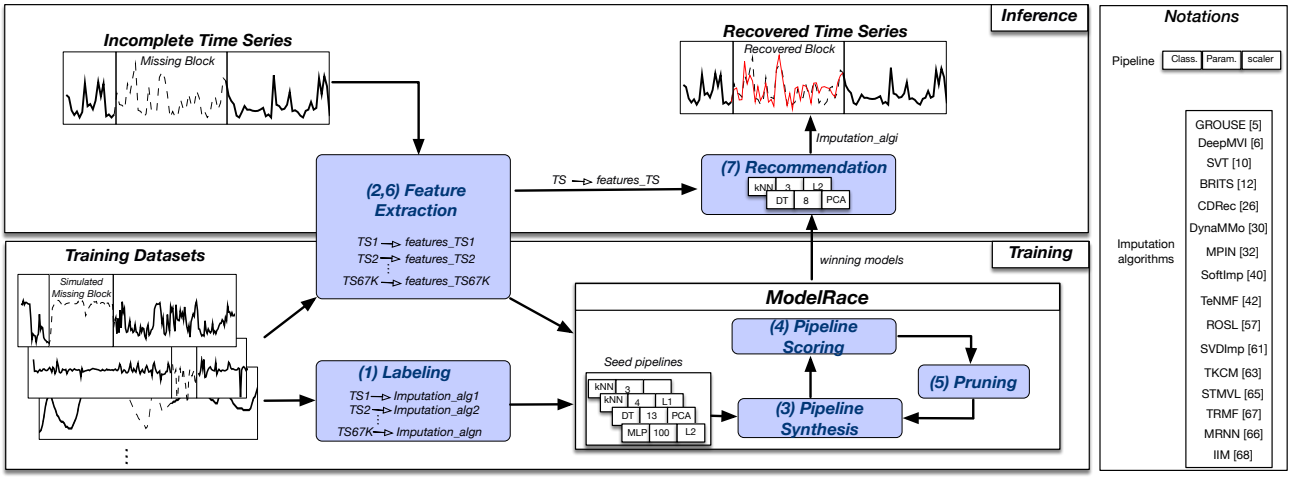


Fig. 2: Overview of A-DARTS, which takes time series with missing blocks as input and recommends the best imputation to repair them. (1) It starts by labeling the time series for training and (2) extracts their features. Then, it executes the model training where it iteratively (3) generates new pipelines, (4) defines the cut-off performance, and (5) prunes the less-performing pipelines. Upon the execution of ModelRace, A-DARTS returns the best-performing pipelines. Next, (6) it feeds the winning pipelines with the features of the new incomplete series and, lastly, (7) recommends the imputation technique.

pipelines for a given imputation class. The class with the highest average is chosen as the imputation to apply to the incomplete series.

V. Model Selection

At the heart of ModelRace lies a process that expands and prunes the number of pipelines being considered. We present the search algorithm that guides this process in Section V-A. Our search algorithm heavily relies on similarity computations between time series. We discuss the time series features we use for such similarity calculations in detail in Section V-B.

A. ModelRace

We start by introducing the intuition behind ModelRace's search for competitive pipelines. ModelRace operates in an iterative manner. It takes a seed of pipelines and synthesizes new pipelines at each iteration. More promising pipelines are further expanded. The synthesis is centered around the existing pipelines such that it introduces only small changes to the parent pipeline by modifying only one parameter at a time. ModelRace also prunes out pipelines it deems inefficient. Pruning is a two-phase procedure. It starts with a small percentage of training data and increases it at each iteration of the algorithm. This allows the pipelines that survive the pruning to continue training on new and unseen data. The candidates (original and derived) are compared pairwise, and the ones performing statistically worse than others are eliminated from the race [52].

We now formally discuss our search and pruning strategy. Algorithm 1 describes the pseudocode of ModelRace. The symbols used in this and the remaining algorithms are introduced in Table II. The ModelRace algorithm takes as input a set of seed pipelines Θ with a unique set of parameter values sampled from the set of all possible pipelines $\mathcal{P}$. Assuming

TABLE II: Notations used in Algorithms 1 and 2.

Algorithm	Symbols	Description
Alg. 1	$\mathcal{P}$	set of all possible pipelines
	Θ	subset of pipelines
	θ_i	a pipeline {classifier, params., scaler}
	$\mathcal{S}$	training set of features and labels
	S_i	subset of training set
	$\mathcal{T}$	test set
	$f1, r3$	F1 and Recall@3 of a pipeline
	$time$	runtime of a pipeline
	α, β, γ	scoring coefficients
Alg. 2	$\mathcal{X}$	set of time series
	X_i	individual time series
	$\tilde{\mathcal{C}}$	set of clusters
	C_i	cluster of X_i
	$\bar{\rho}(C_i)$	average correlation inside C_i

12 classifiers, there are 1650 possible parameterizations and 60 different feature scaling options, leading to 99'000 possible pipelines to consider. The algorithm uses partial training sets $\mathcal{S} = \{S_1, \ldots, S_m\}$, each representing a subset of the whole training set, and $\mathcal{T}$, a test set used during evaluation.

At each iteration, we select a new partial training set S_i (line 2) and expand the set of pipelines by generating new candidate pipelines (line 3). This expansion helps fine-tune the pipelines' parameters and scaling steps. We generate new pipelines based on the current surviving candidates Θ^{elite}, used as seeds.

The evaluation of the pipelines is performed on different stratified k-folds [53] to produce multiple scores per pipeline (line 5). The stratification guarantees the same distribution of samples/classes as in the original dataset to avoid overfitting. Each pipeline is trained on a small portion of the data (line 6) and evaluated on the test set (line 7). We compute a score using a weighted average of performance metrics such as F1-Score and normalized runtime to maximize

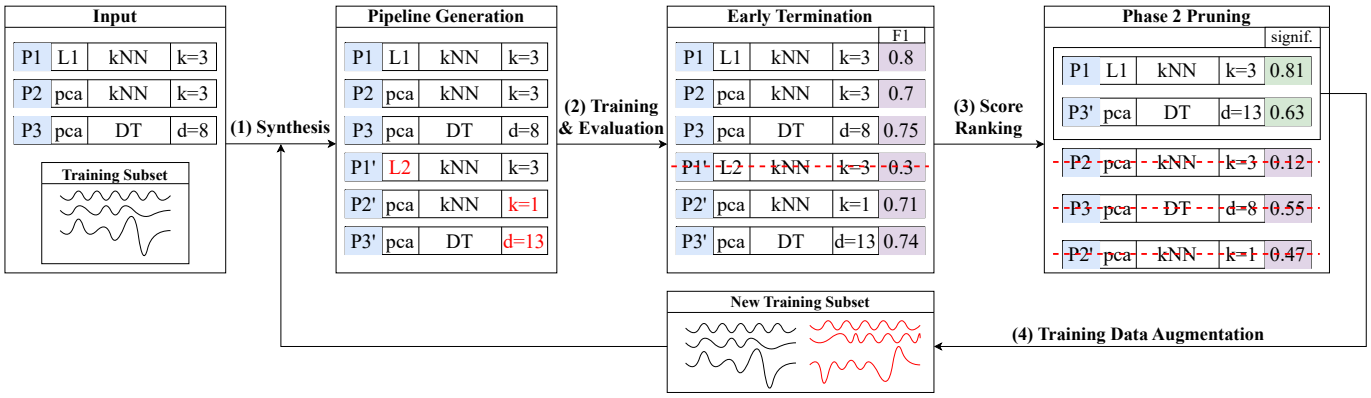


Fig. 3: ModelRace Pipeline Selection.

Algorithm 1: ModelRace

Input : Set of seed Pipelines: $\Theta = \{\theta_1, \ldots, \theta_n\} \sim \mathcal{P}$,
Set of partial training data: $\mathcal{S} = \{S_1, \ldots, S_m\}$,
Test set: T

Output: Θ^{elite}: set of best-performing pipelines

1 $\Theta^{elite} = \Theta$
2 **foreach** $S_i \in \mathcal{S}$ **do**
3 $\quad \Theta^{new} = \text{Synthesize}(\Theta^{elite})$;
$\quad$ ▷ generate new pipelines based on Θ^{elite}
4 $\quad \Theta^{cand} = \Theta^{elite} \cup \Theta^{new}$
5 $\quad$ **foreach** $SF \in \text{StratifyKFold}(S_i)$ **do**
6 $\quad\quad$ **foreach** *pipeline* $\theta \in \Theta^{cand}$ **do**
7 $\quad\quad\quad \theta = \text{Train}(\theta, SF)$;
8 $\quad\quad\quad f1, r3, time = \text{Evaluate}(\theta, T)$;
9 $\quad\quad\quad score_\theta = \frac{(\alpha \cdot f1 + \beta \cdot r3) - (\gamma \cdot time)}{\alpha+\beta+\gamma}$;
10 $\quad\quad\quad scores \leftarrow scores \cup score_\theta$;
11 $\quad\quad\quad$ **if** $\exists \theta'$ *s.t.* $score_\theta \ll score_{\theta'}$ **then**
12 $\quad\quad\quad\quad \Theta^{cand} = \Theta^{cand} \setminus \{\theta\}$;
$\quad\quad\quad\quad$ ▷ Early termination
13 $\quad \Theta^{elite} = \text{Prune}(\Theta^{cand}, scores)$;
$\quad$ ▷ Prune the pipelines based on T-test comparison between the scores of all pairs of pipelines
14 return Θ^{elite}

effectiveness while minimizing execution time (line 8). Once the pipeline evaluation on a given fold is completed, we search for pipelines with a significantly higher score than the current one on the same fold. If the search is not empty, we early terminate the current pipeline (lines 11-12), allowing the training on the remaining folds to complete faster.

The remaining pipelines (Θ^{cand}) can still be large, as only the worst ones are terminated early. We apply a second-phase pruning to reduce its size further while keeping some diversity in the results. At the end of each iteration, pairwise significance t-tests [54] are performed on the pipeline scores (line 13). We compare the score distributions between all pairs of pipelines, and if the t-test considers that they are similar with a high significance, we prune the one with the lower average of scores. More training data is then available for the remaining pipelines to continue training. The algorithm terminates when all partial training sets have been used.

Figure 3 illustrates ModelRace through an example. As introduced above, a pipeline is, in our context, a tuple <classifier, classifier hyperparameters, feature vector scaling strategy>. We consider two input classifiers: k-nearest Neighbor (kNN) and Decision Tree (DT). Their hyperparameters are the number of neighbors (k) and the tree depth (d), respectively. Each classifier may choose to normalize the feature values differently, e.g., through an L1/L2-norm normalizer, a PCA-based dimensionality reduction, etc.

We start by generating new pipelines based on the input (Step 1 in Figure 3). For example, $P1'$ is generated based on $P1$ by changing only the scaling to the L2-norm. Note that this generation process is centered around the existing pipelines such that it introduces only small changes to the parent pipeline by modifying only one parameter at a time.

Next, we perform the training and evaluation step (2) of all the pipelines. Once the results are obtained, we compute the F1 score using the test set and verify if any pipelines perform significantly worse than the others. In our case, $P1'$ has a very low F1 compared to the best result. At this point, we apply early termination on $P1'$, which is eliminated from the race.

The last step is the second phase of pruning (3). Using previously obtained scores, we rank the remaining pipelines according to the average results of pairwise t-tests. Only $P3'$ and $P1$ survive this phase of pruning. Next, we augment the training subset with more time series and repeat the process again, starting with the generation procedure until we exhaust all the training subsets, as described in step (4).

Complexity Analysis. The pipeline selection iterates over $m = |\mathcal{S}|$ folds. Each iteration invokes the training of n pipelines, each over k stratified folds. Since at each iteration only a small fraction of the whole training set is used until it reaches the full size and k is set to a small constant, the worst case time complexity is $O(n \times |\mathcal{S}|)$.

ModelRace keeps in memory the training set, the pipelines, and their evaluation scores. Since the number of pipelines decreases with the iterations, the space complexity is linear with the size of the training set.

B. Feature Extraction for Imputation

As mentioned earlier, A-DARTS provides a configuration-free recommendation for imputing missing values. The best recommendation is obtained by feeding parameterized classifiers with the relevant features. Those features help the classifiers assign the same imputation algorithm to similar time series. Feature extraction is a well-studied problem in some time series tasks such as clustering [55] or forecasting [50]. We build upon this line of research and introduce a new set of features for imputation.

We curate a set of existing statistical features and group them into two representative categories: statistical and topological. The topological features need is evident in early experiments, where we notice that classifiers that rely solely on statistical features perform poorly. Ultimately, the combination of both types of features is what allows our system to accurately identify the time series properties that impact the imputation process.

Statistical Features. We extract a list of statistical features by concatenating various feature extraction tools (e.g., TSFresh, Catch22, or Kats) [56]. We explain those features using a coarse-grained categorization.

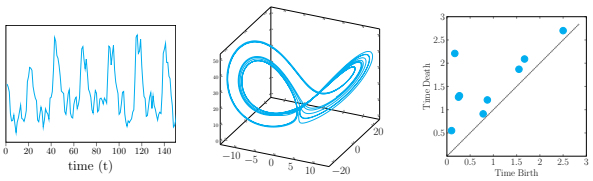
Canonical. This category includes measures summarizing time series basic statistical measures such as mean or variance. They serve as an indicator of the data evolution of over time.

Dependencies. This category encompasses measures that capture dependencies in time series, such as auto-correlation. Those dependencies occur within the time series at different time intervals.

Trends. The values describe the seasonality and frequency of a time series. Linear transformation methods, such as Principal Components (PCA) [57], that help detect the data trend also belong to this group.

Topological Features. Topology pertains to the shape of the data, which is, unfortunately, not covered by any statistical feature. Feeding the classifiers with this property allows them to include the visual resemblance in the similarity computation. Several properties can contribute to the definition of shape in time series. The temporal order of the values is one example that heavily impacts the performance of some imputation techniques. Statistical measures cannot capture the order of the data, as they are time-agnostic. Perturbation is another example of shape-based properties that are not detectable by any statistical extractor. It identifies whether a time series has changed its trend due, for example, to a sensor malfunction.

We built a new topological extractor that maps time series onto a multidimensional space, which captures the shape of the series. Our topological extractor extends Topological Data Analysis (TDA) [58] to time series data. Figure 4 illustrates the main steps of the process. First, we embed the input time series into a time-delay space that captures their non-linear relationships. We achieve this by mapping each time series $X = \{(t_1, v_1), \ldots, (t_m, v_m)\}$ into a sequence of vectors (see Figure 4b). Each new time series will have the form of $v_p^\tau(j) = (v_j, v_{j+\tau}, \ldots, v_{j+(d-1)\tau})$, where τ is the time shift, and d is the embedding dimension.



(a) Time Series (b) Embedding (c) Persistence Diagram

Fig. 4: Topological Feature Extraction.

After embedding the time series, we construct a persistence diagram (cf. Figure 4c) [59]. This diagram captures the birth and death of each pattern in the time series. Those patterns correspond to the "holes" visible in the embedding. More specifically, each point (b_i, d_i) on the diagram represents a pattern where b_i is the time of its birth, d_i is the time of its death, and $d_i - b_i > 0$ is the span of time over which it was alive. Finally, we use the distributions of points in the persistence diagram as topological features.

VI. Data Labeling

The ModelRace process outlined above relies on on access to a large and diverse dataset of labeled time series. To facilitate labeling, we cluster time series and label them at a cluster level. We start this section with an intuitive example of how the clustering algorithm works in Section VI-A. The algorithm has two phases: one in which we construct an initial set of clusters, which we discuss in Section VI-B, and one in which we refine the clusters, described in Section VI-C.

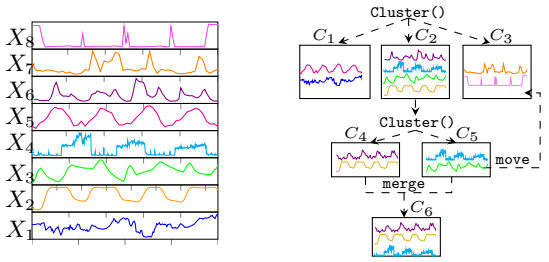
A. Clustering Intuition

Given the vast number of time series we consider—more than 67k—it would be unfeasible to apply all imputation algorithms on each individual series. For instance, the results presented in [32] were derived from just 260 time series and required several days to compute.

Instead, we draw on an existing approach for a similar problem [47] and propose approximating the labeling results. A-DARTS clusters times series and labels a representative series from each cluster, propagating the label to the other series from the same cluster. This labeling process can leverage shape-based clustering [60], which groups time series with similar shapes. A popular method, K-shape [61], utilizes time series cross-correlation to estimate shape similarity, given a predefined number of clusters. However, determining a desirable number of clusters is challenging due to the diverse nature of the time series datasets and their application domains. To address this, we propose an incremental splitting approach to identify the optimal number of clusters.

We illustrate in Figure 5 the two phases of our iterative clustering process, initial and refined clustering, using eight time series and their cross-correlation as a similarity measure.

In the first phase, we cluster the input time series into several clusters based on their similarity. This produces three different



(a) Input Time Series (b) Clustering Procedure

Fig. 5: Example of Incremental Clustering using Eight Time Series.

clusters C1, C2, and C3. The second iteration will further cluster the time series inside C2 into two different clusters, C4 and C5. Therefore, the first phase produces four intermediate clusters: C1, C3, C4, and C5.

The second phase reduces the number of clusters while maintaining a high series similarity. It does so by redistributing some series and merging small clusters. For instance, we move T_3 from C5 to C3, as this will further increase the similarity inside C3. Following the same rationale, we merge C4 with the updated C5. We obtain three clusters at the end of this phase: C1, C3, and C6.

We now present and discuss each phase of the clustering algorithm in more detail.

B. Initial Clustering Phase

Algorithm 2 describes the pseudo-code for our clustering algorithm. The initial phase is described in lines 2-8. It checks whether the time series within each cluster are not highly correlated (line 4). In that case, it dynamically divides the cluster into sub-clusters (line 5). The number of subclusters is the product between the size of the cluster and the estimated percentage of time series in the cluster, p. We empirically set this ratio to 20%. The resulting clusters are added to a pending ensembles' list $\mathbb{S}$ (line 6), which contains the set of time series that do not yet form final clusters. This process is repeated until it reaches the desirable average correlation.

This initial clustering phase yields either many mono-sequences or small clusters, as it tries to maximize the correlation inside clusters. This incurs a high runtime since the imputation benchmark will need to perform more runs to label the data. We address this issue by applying a merging phase that minimizes the number of produced clusters while maintaining a high correlation.

C. Refining Phase

For each small cluster, we evaluate the correlation gain that would result from merging it with another one (line 10). In case of positive gain, we merge the two clusters (lines 11-12). If no such cluster exists, we evaluate each series separately in the same way and move an individual series instead (lines 15-17). In case no cluster yields any positive gain, the sequences remain in their original cluster.

Algorithm 2: Incremental Clustering

Input : Set of time series $\mathcal{X} = \{X_1, \ldots, X_n\}$
Output: List of clusters $\mathcal{C}$

1 $\mathcal{C} \leftarrow \emptyset$;
2 $\mathbb{S} \leftarrow \{\mathcal{X}\}$;
3 **while** $\mathbb{S} \neq \emptyset$ **do**
4 $\quad C_{last}$ =Pop($\mathbb{S}$);
5 $\quad$ **if** $\bar{\rho}(C_{last}) < \delta$ **then**
6 $\quad\quad C_{temp} \leftarrow$ `cluster`$(C_{last}, \max(2, p \times |C_{last}|))$;
7 $\quad\quad \mathbb{S} \leftarrow \mathbb{S} \cup C_{temp}$;
8 $\quad$ **else**
9 $\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup C_{last}$;
10 **foreach** *Cluster* $C_i \in \mathcal{C}$ **do**
11 $\quad C^* = \underset{C_j \neq C_i}{arg\ max}\ \Delta G_{ij}$;
12 $\quad$ **if** $\Delta G_{ij} > 0$ **then**
13 $\quad\quad C^* \leftarrow C^* \cup C_i \triangleright$ `merge clusters`
14 $\quad$ **else**
15 $\quad\quad$ **foreach** *series* $X \in C_i$ **do**
16 $\quad\quad\quad C^* = \underset{C_j \cup \{X\}}{arg\ max}\ \Delta G_{ij}$
17 $\quad\quad\quad$ **if** $\Delta G_{ij} > 0$ **then**
18 $\quad\quad\quad\quad C^* \leftarrow C^* \cup \{X\} \triangleright$ `move into new cluster`
19 **return** $\mathcal{C}$;

We note that our merging procedure reorganizes the clustering using two operations: merge and move. These operations are based on maximizing the correlation gain (CG) between clusters. CG borrows its intuition from the concept of modularity, which is used in the popular Louvain graph clustering algorithm [62], and extends it to time series. We now define the correlation gain formally.

Definition 1: Let C_i and C_j be two clusters in a dataset D, $\bar{\rho}(C_i)$ and $\bar{\rho}(C_j)$ the average correlations between all time series pairs in C_i and C_j, respectively, and m the total number of time series in D. The correlation gain of merging C_i with C_j is defined as:

$$\Delta G_{ij} = \frac{1}{2m}(\bar{\rho}(C_i \cup C_j) - \frac{\bar{\rho}(C_i) \times \bar{\rho}(C_j)}{m}) \quad (1)$$

In case we move one sequence $X \in C_i$ to another cluster, then $\bar{\rho}(C_i \cup C_j)$ is equal to $\bar{\rho}(\{X\} \cup C_j)$.

The correlation gain clusters together the most similar time series by computing the cross-correlation before the move (second term of Eq. 1) and the potential correlation after the move (first term of Eq. 1). The difference between the two terms measures the gain of merging the clusters.

The merging process based on correlation gain has two important properties. First, the gain is monotonic thanks to the greedy maximization of the correlation. Second, the move and merge procedure is guaranteed to terminate since a time series that moves out of a cluster never moves back into it, otherwise, the gain would become negative.

VII. EXPERIMENTS

In this section, we validate our technical contributions based on a series of experiments. The experiments are divided mainly into two sets. The first set aims to contrast A-DARTS with comparable systems. We start by evaluating how representative the datasets we use in the experiments are (Section VII-A). Once that is established, we compare A-DARTS's recommendations efficacy to that of other systems (Sections VII-B and VII-C) as well as their relative running time (Section VII-D). This set of experiments shows that A-DARTS systematically offers better recommendations without incurring additional runtime costs.

In the second experiment set, we assess how different components of A-DARTS contribute to its efficacy and running time. We investigate the importance of our feature selection (Section VII-E1), pipeline scoring (Section VII-E2), and clustering techniques (Section VII-E3). This set of experiments shows that every component of A-DARTS has a measurable impact on the final recommendation and is properly tuned.

In the last set of experiments, we evaluate the downstream impact of choosing the appropriate imputation technique on forecasting (Section VII-F). The downstream experiments reveal that our tool improves not only the upstream analysis but also the downstream one.

Experimental Setup. All the experiments were run on an Ubuntu 18.04 Linux machine with a 2.1 GHz Intel Xeon E5-2620 processor and 128 GB of main memory. The processor comprises six physical cores and carries 15 MB of LLC. The code and data involved in A-DARTS and the experiments are publicly available[1].

Efficacy Metrics. We report our results using standard metrics such as **A**ccuracy, **P**recision, **R**ecall, and **F1**-score (bold letters will be used as acronyms in charts). For each metric, we compute a weighted average to account for the label imbalance between the imputation techniques during the labeling. In addition to those metrics, we use the Mean Reciprocal Rank (MRR) to compute the ranking of the recommended techniques. Formally, let Q be the queries (test set) and $rank_i$ be the position at which the "correct" imputation technique is predicted, then:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

A. Datasets Description

We aimed to gather a diverse set of real-world time series that cover a broad spectrum of characteristics. The data consists of 107 datasets, each containing several hundred time series of different lengths and properties, amounting to a total of 67K series. The data originates from various sources including the TSC repository [63], the UCR repository [64], the UCI repository [65], and ImputeBench [32]. We group the datasets into six categories based on their domain and summarize the key properties of each category below:

- *Power.* This category encompasses household electricity consumption data recorded in different countries. The data is collected using smart meters allowing automated collection at fine-grained time intervals. The electricity time series are periodic, and some are shifted in time.
- *Water.* This category consists of water quality measurements such as discharge, conductivity, oxygen level, and pH value, provided by the BundesAmt Für Umwelt (BAFU) [66], the Swiss Federal Office for the Environment. Water time series contain synchronized trends and sporadic anomalies.
- *Motion.* This category includes time series originating from various motion sensors, such as accelerometers or gyroscopes, that capture body movements. Motion time series contain erratic fluctuations and varying frequency.
- *Climate.* The time series in this group describe various weather phenomena (such as temperature or precipitation) provided by the Swiss Federal Office of Meteorology and Climatology collected from different Swiss cities from 1974 to 2015. Climate time series are periodic and exhibit very high correlation.
- *Lightning.* This category contains electromagnetic events associated with lightning using optical and radio-frequency (RF) instruments. It provides time series collected at a high rate (50 MHz), which have mixed correlation (high/low, positive/negative) and exhibit partial trend similarities.
- *Medical.* This category represents various human health-related data such as ECG and hemodynamics (blood flow such as airway pressure, arterial blood pressure, or central venous pressure). Medical time series are measured at high frequency (250Hz) and present aligned and shifted trends.

The datasets were amassed to ensure sufficient representation of each of the 430 features used by A-DARTS. To that end, we normalize each feature value to the range [0, 1], divide the interval into k buckets, and compute the number of buckets covered by the time series across our 107 datasets. The results of this experiment are shown in Figure 6 with features plotted on the y-axis and the datasets (in no particular order) plotted on the x-axis.

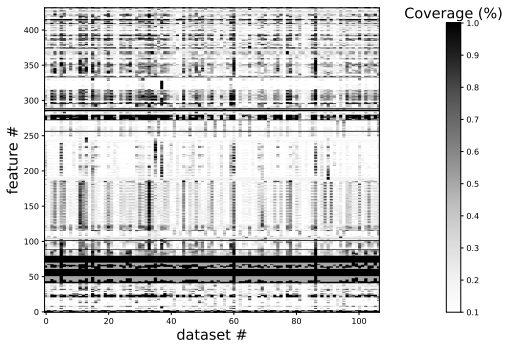


Fig. 6: Feature Coverage Heatmap.

We observe that all the features are covered by at least one

[1]https://github.com/eXascaleInfolab/recimpute

dataset and each time series covers a different combination of features. Some features, such as the ones in lines 63-81, are predominantly present in all the time series. Those features correspond to binary features, such as whether the distribution of values is symmetric, which naturally occur in many time series. We also observe other features, which can be seen in lines 320-330, which are covered by fewer time series. Those features correspond to peculiar features such as sudden changes or noise.

B. Recommendation Efficacy

In the next experiment, we compare our system against the four AutoML frameworks: FLAML [40], Tune [39], AutoFolio [67], and RAHA [47], introduced in Section III. We adapt those systems to apply to the task of recommending imputation techniques for time series. We do so by training all the systems on our labeled data and feeding the ones that do not implement any feature extraction with our set of features. We experiment with various versions of the baselines and report their results under their optimal setup.

We test 12 different classifiers ranging from standard k-nearest-neighbors (kNN) [44], decision trees (DT) [68], and multi-layer perceptrons (MLP) [45] to more recent, sophisticated ones such as CatBoost [46]. We report the recommendation quality using a sample holdout strategy on each category with a 65/35 splitting ratio. We create synthetic missing blocks of varying sizes on each series and compare the recommended imputation technique against the ground truth. Figure 7 shows the average F1 scores (marked as points) and the standard deviation (marked as an interval).

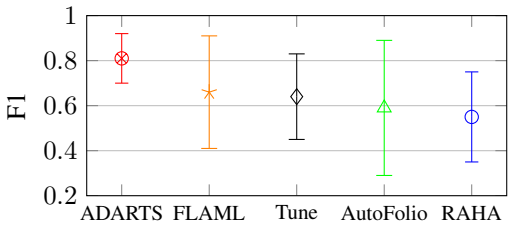


Fig. 7: Average Efficacy Performance.

We observe that A-DARTS yields the highest average F1 and achieves 20% F1 gain compared to its closest competitor, FLAML. Our technique also produces the tightest F1 bounds and is about 2.5x more stable than the second-best technique. This stems from the fact that A-DARTS can easily accommodate any dataset, as we will show next.

C. Recommendation Efficacy Breakdown

We break down in table III the efficacy of each system using all metrics grouped by dataset category. The results show that A-DARTS outperforms all the baselines on all datasets. The results also show that recommendation performance depends on the properties of the time series, and three performance trends emerge.

TABLE III: Efficacy comparison of the recommendation per dataset. The best results are highlighted in bold, and the second-best results are marked by '+'.

Dataset	System	Efficacy Metric				
		A	P	R	F1	MRR
Power	RAHA	0.42	0.48	0.42	0.44	0.58+
	AutoFolio	0.67+	0.63	0.63	0.63	-
	Tune	0.66	0.62	0.66	0.63	-
	FLAML	0.67+	0.64+	0.68+	0.65+	-
	A-DARTS	**0.70**	**0.66**	**0.70**	**0.76**	**0.81**
Water	RAHA	0.64+	0.79+	0.64+	0.69+	0.77+
	AutoFolio	0.20	0.12	0.20	0.14	-
	Tune	0.58	0.62	0.59	0.59	-
	FLAML	0.29	0.33	0.29	0.25	-
	A-DARTS	**0.81**	**0.92**	**0.81**	**0.84**	**0.89**
Motion	RAHA	0.43	0.48	0.43	0.45	0.62+
	AutoFolio	0.73	0.71	0.73	0.71	-
	Tune	0.73	0.71	0.73	0.72	-
	FLAML	0.75+	0.73+	0.75+	0.73+	-
	A-DARTS	**0.78**	**0.77**	**0.78**	**0.77**	**0.78**
Climate	RAHA	0.35	0.49	0.35	0.38	0.52+
	AutoFolio	0.85	0.86	0.85	0.84	-
	Tune	0.85	0.87	0.85	0.85	-
	FLAML	0.88+	0.89+	0.88+	0.88+	-
	A-DARTS	**0.92**	**0.92**	**0.92**	**0.92**	**0.96**
Lightning	RAHA	0.52	0.46	0.52	0.47	0.70+
	AutoFolio	0.32	0.52	0.32	0.32	-
	Tune	0.46	0.43	0.46	0.41	-
	FLAML	0.57+	0.52+	0.57+	0.51+	-
	A-DARTS	**0.68**	**0.63**	**0.68**	**0.66**	**0.81**
Medical	RAHA	0.88	0.92	0.88	0.90	0.91+
	AutoFolio	0.94	0.94+	0.94	0.93	-
	Tune	0.94	0.94+	0.94	0.94+	-
	FLAML	0.95+	0.94+	0.95+	0.94+	-
	A-DARTS	**0.96**	**0.96**	**0.96**	**0.96**	**0.98**

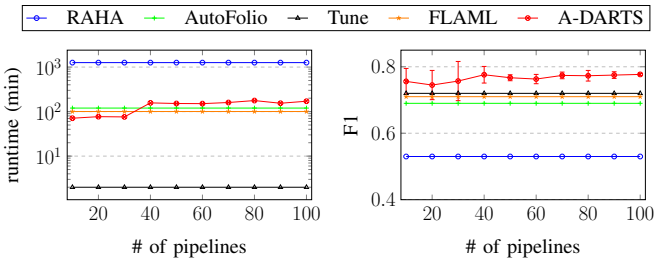
High Variability Datasets. In some categories, the difference between A-DARTS and the baselines is very high. In the Water dataset, for example, A-DARTS achieves an F1 of 0.84, while the second-best technique, RAHA, achieves an F1 of 0.69. FLAML produces a very low F1 of 0.25. As explained earlier, Water is a complex dataset that contains a high number of anomalies. Our system can easily capture those data inconsistencies thanks to the different feature scalers it considers. We observe similar trends in the Lightning dataset, but a smaller gap as this dataset contains fewer data inconsistencies.

Moderate Variability Datasets. In Power and Motion, the performance of the baselines improves, but the performance difference is still noticeable. For instance, in the Power dataset, A-DARTS is 15% more effective than FLAML (0.76 vs. 0.65). Those datasets include features, such as time shifts or varying frequencies, which are difficult to process by imputation techniques [69].

Ranked Results Availability. Only A-DARTS and RAHA can provide a ranked list of recommendations (MRR). Our system accurately recommends the best imputation method and systematically computes the correct ranking of imputation techniques, achieving an average MRR value of 0.87. This means that it finds the correct ranking in 87% of the cases, yielding 28% improvement in MRR compared to RAHA, which returns an MRR of 0.68.

D. Recommendation Efficiency

In addition to effectiveness, we evaluate the elapsed running time (wall clock) to select a model, train it, and produce the recommendation in the previous experiment. We vary the number of seed pipelines for ModelRace, which impacts the total runtime, instead of varying the number or length of the time series. We also report the average F1 values. Figure 8a depicts the results of this experiment.



(a) Average Running Time. (b) Average Efficacy.

Fig. 8: Recommendation Running Time vs. Efficacy.

We observe that up to 30 pipelines, A-DARTS is 1.35x and 1.6x faster than FLAML and AutoFolio, respectively. For such a number of input pipelines, the pruning helps keep a relatively low number of configurations. For a higher number of pipelines, FLAML becomes, on average, 1.3x faster than our system. This is expected as our system explores a larger search space of pipeline configurations. Tune is an order of magnitude faster than the other systems, as it needs to configure only one classifier with no scaling steps. This low running time, however, comes with a low accuracy cost, as shown in Table III.

The impact of the number of pipelines on the recommendation quality, measured as F1 scores, appears in Figure 8b. We observe that increasing the number of pipelines does not only increase F1 but also the stability of our recommendation—the standard deviation is monotonically decreasing. Using more pipelines adds more diversity to the selection process, allowing our system to find better pipelines.

This experiment shows another important property of A-DARTS. Counter-intuitively, our system allows duplicate classifiers to be selected for voting. In this experiment in particular, in some instances, A-DARTS picks three of the available 12 classifiers—and two duplicates. The duplicates allow the same classifier to be configured differently by varying the parameters and scaling steps. It is this diversity that contributes to the low variability we observe in Figure 8b. Our analysis also shows that no particular classifier stands out, and the type of the chosen classifier heavily depends on the properties of the time series data.

E. Ablation Study

We now turn our attention to a series of micro-benchmarks that determine the importance of different A-DARTS components in the results shown above.

1) Effects of Varying Feature Selection: In this experiment, we feed ModelRace with different configurations of feature sets and determine its performance. We consider three configuration sets: i) statistical features only, ii) topological features only, and iii) a combination of both feature sets. Figure 9 depicts the results for each dataset category.

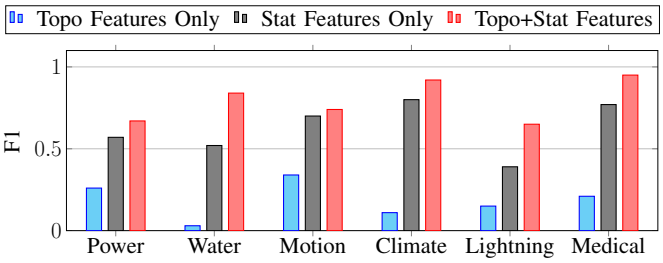
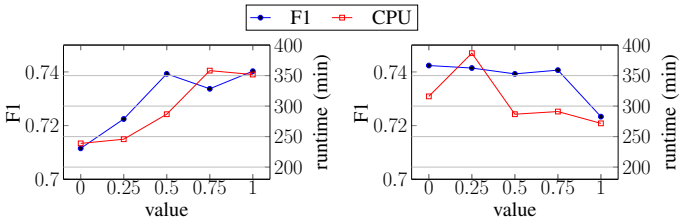


Fig. 9: Feature Analysis.

We observe that in categories with complex properties, such as Water or Lightning, both categories of features are needed to characterize the time series properly. The similarity between time series, in this case, is much harder to compute using statistical measures only, thus the need to leverage the shape of the data. Using only statistical features can be viable occasionally, e.g., in categories with simple properties such as the Motion dataset.

2) Effects of Pipeline Scoring Alternatives: The pruning mechanism relies on a scoring function that assigns a weight to the evaluated pipelines (see Line 9 in Algorithm 1). We evaluate this function by varying its coefficients α and γ through two experiments. When we vary one coefficient, we set the remaining ones to a constant value. Note that we use two y axes: the first one for F1 and the second for runtime. We report the results of these experiments in Figure 10.



(a) varying α. (b) varying γ.

Fig. 10: Score Function.

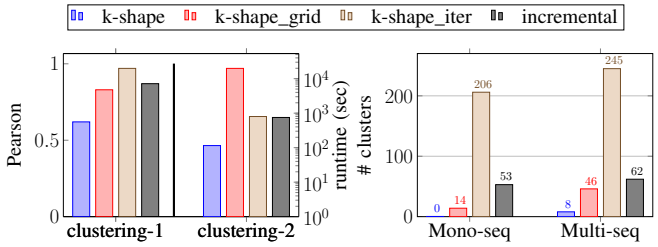
The results shown in Figure 10a show that increasing α has a similar impact on F1 and CPU. This is expected, as this increase gives more weight to the most effective pipelines, which are the more time-consuming ones—they have more parameters to set. Setting α above 0.5 does not significantly increase F1 while incurring significant runtime overhead.

The results of the second experiment in Figure 10b are generally aligned with those in Figure 10a and we identify two main results. First, varying γ between 0 and 0.75 does not substantially affect the effectiveness of the system. At $\gamma = 1$, we notice a significant decrease in F1 due to prioritizing the

runtime of the pipelines too much. The CPU curve follows a downward trend, but the runtime yields no significant decrease beyond 0.75.

The optimal trade-off between selecting effective and efficient pipelines is achieved by setting α and γ to 0.5 and 0.75, respectively. This configuration allows ModelRace to give a slightly higher priority to faster pipelines with no negative impact on F1. Thus, our system can find pipelines with an acceptable quality even if the runtime is given priority. This occurs because many pipelines have similar effectiveness but different runtimes.

3) Effects of Varying the Clustering Techniques: In this section, we evaluate our incremental clustering (used for labeling) and compare it against three variants of K-shape [61]: default ($k = 8$), grid search, and iterative. We report the results using two y-axes. The first y-axis shows the cluster average correlation (the higher, the better), while the second y-axis shows the runtime. Note that we do not report F1 since we cannot compare recommendations using different clusterings. Figure 11 shows the result of this experiment.



(a) Correlation analysis. (b) # of clusters.

Fig. 11: Clustering Performance.

Figure 11a shows that our incremental clustering yields clusters with highly correlated time series (correlation = 0.87) in a reasonable runtime. k-shape performs a very efficient clustering with a relatively low cross-correlation between the time series (correlation = 0.61). Such a low correlation hinders the labeling step, as all time series will be assigned the same imputation technique despite exhibiting different trends. Using grid search improves the correlation of k-shape but incurs a very high runtime.

Finally, we provide a deeper analysis of the clustering results by computing the number of final clusters. The results in Figure 11b show that our solution produces the closest number of clusters to the ground truth obtained through a grid search. The high correlation produced by the iterative solution comes with a very high number of clusters. Such a high number incurs a prohibitive execution time for the labeling step, which will be performed more times than needed. K-shape assigns all the time series to a small number of clusters, which hinders the quality of data labeling.

F. Downstream Analysis

In this last experiment, we evaluate the impact of recommending imputation algorithms on downstream tasks. We focus on forecasting and evaluate the impact using seven forecasting datasets collected from different sources, including the Monash benchmark [70]. We create random blocks at the tip of each time series with the size of 20% and use A-DARTS to recommend the best imputation algorithm. We compare the forecasting results on a horizon of 12 observations of the repaired series with and without A-DARTS. The latter simulates the recommendation introduced in [32] by constructing a binary vector that defines the recommendation axis and computing its dot product with each algorithm's score vector. The algorithm yielding the highest score is then recommended.

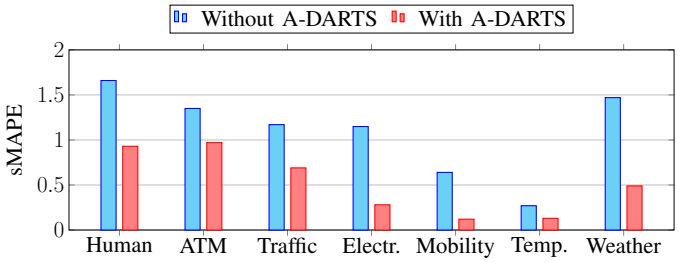


Fig. 12: Impact on Time Series Forecasting.

Figure 12 depicts the Symmetric Mean Absolute Percentage Error (sMAPE) [36] results for each dataset (the lower the better). We observe that A-DARTS substantially improves the forecasting task, on average, by 55%. The gain ranges between 28% (0.97 vs. 1.32) in the ATM dataset and 80% (0.12 vs. 0.64) in the Paris mobility dataset. The highest difference in sMAPE is observed in the Weather dataset (0.49 vs. 1.47). The datasets with the highest gains contain time series with complex features, which require a specialized imputation algorithm. The downstream improvement is explained by the fact that the performance of forecasters heavily depends on how well the data has been repaired and, thus, the choice of the imputation algorithm. A-DARTS restores the ability of forecasters to learn the trends of the time series.

VIII. Conclusion

This paper introduces A-DARTS, a system that automatically selects a suitable repair technique for a new time series. Without A-DARTS, users have to manually identify which imputation technique could be best suited to every new incoming time series. After being adequately trained, A-DARTS can perform such a recommendation automatically, with high accuracy and minimal runtime overhead. It substantially improves F1 by 20% and is 2.5x more stable than the state of the art in model selection. It also drastically improves downstream tasks, such as forecasting, by 55%.

To develop A-DARTS, we proposed several innovative techniques, chief among them ModelRace, which sifts through a vast pool of configurations to efficiently identify the ideal repair technique. Our techniques allow ModelRace to quickly and efficiently learn which are the best classifiers and under which conditions. In the future, we intend to continue this line of work by looking into novel techniques that would automatically detect the types of missing patterns and include them as additional features to the recommendation process.

REFERENCES

[1] J. Paparrizos, P. Boniol, T. Palpanas, R. Tsay, A. J. Elmore, and M. J. Franklin, "Volume under the surface: A new accuracy evaluation measure for time-series anomaly detection," *Proc. VLDB Endow.*, vol. 15, no. 11, pp. 2774–2787, 2022. [Online]. Available: https://www.vldb.org/pvldb/vol15/p2774-paparrizos.pdf

[2] P. Boniol, J. Paparrizos, T. Palpanas, and M. J. Franklin, "SAND in action: Subsequence anomaly detection for streams," *Proc. VLDB Endow.*, vol. 14, no. 12, pp. 2867–2870, 2021. [Online]. Available: http://www.vldb.org/pvldb/vol14/p2867-boniol.pdf

[3] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," *ACM Comput. Surv.*, vol. 54, no. 3, apr 2021. [Online]. Available: https://doi.org/10.1145/3444690

[4] S. Y. Shah, D. Patel, L. Vu, X. Dang, B. Chen, P. Kirchner, H. Samulowitz, D. Wood, G. Bramble, W. M. Gifford, G. Ganapavarapu, R. Vaculín, and P. Zerfos, "Autoai-ts: Autoai for time series forecasting," in *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, G. Li, Z. Li, S. Idreos, and D. Srivastava, Eds. ACM, 2021, pp. 2584–2596. [Online]. Available: https://doi.org/10.1145/3448016.3457557

[5] Y. Cui, K. Zheng, D. Cui, J. Xie, L. Deng, F. Huang, and X. Zhou, "METRO: A generic graph neural network framework for multivariate time series forecasting," *Proc. VLDB Endow.*, vol. 15, no. 2, pp. 224–236, 2021. [Online]. Available: http://www.vldb.org/pvldb/vol15/p224-cui.pdf

[6] P. Esling and C. Agon, "Time-series data mining," *ACM Comput. Surv.*, vol. 45, no. 1, dec 2012. [Online]. Available: https://doi.org/10.1145/2379776.2379788

[7] G. Atluri, A. Karpatne, and V. Kumar, "Spatio-temporal data mining: A survey of problems and methods," *ACM Comput. Surv.*, vol. 51, no. 4, aug 2018. [Online]. Available: https://doi.org/10.1145/3161602

[8] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: A review," *Data Min. Knowl. Discov.*, vol. 33, no. 4, p. 917–963, jul 2019. [Online]. Available: https://doi.org/10.1007/s10618-019-00619-1

[9] K. Echihabi, P. Fatourou, K. Zoumpatianos, T. Palpanas, and H. Benbrahim, "Hercules against data series similarity search," *Proc. VLDB Endow.*, vol. 15, no. 10, pp. 2005–2018, 2022. [Online]. Available: https://www.vldb.org/pvldb/vol15/p2005-echihabi.pdf

[10] A. Gogolou, T. Tsandilas, K. Echihabi, A. Bezerianos, and T. Palpanas, "Data series progressive similarity search with probabilistic quality guarantees," in *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, Eds. ACM, 2020, pp. 1857–1873. [Online]. Available: https://doi.org/10.1145/3318464.3389751

[11] S. Song and A. Zhang, "Iot data quality," in *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, M. d'Aquin, S. Dietze, C. Hauff, E. Curry, and P. Cudré-Mauroux, Eds. ACM, 2020, pp. 3517–3518. [Online]. Available: https://doi.org/10.1145/3340531.3412173

[12] J. Cambronero, J. K. Feser, M. J. Smith, and S. Madden, "Query optimization for dynamic imputation," *Proc. VLDB Endow.*, vol. 10, no. 11, p. 1310–1321, aug 2017. [Online]. Available: https://doi.org/10.14778/3137628.3137641

[13] S. Papadimitriou, J. Sun, and C. Faloutsos, "Streaming pattern discovery in multiple time-series," in *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, K. Böhm, C. S. Jensen, L. M. Haas, M. L. Kersten, P. Larson, and B. C. Ooi, Eds. ACM, 2005, pp. 697–708. [Online]. Available: http://www.vldb.org/archives/website/2005/program/paper/thu/p697-papadimitriou.pdf

[14] A. Zhang, S. Song, Y. Sun, and J. Wang, "Learning individual models for imputation," in *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*. IEEE, 2019, pp. 160–171. [Online]. Available: https://doi.org/10.1109/ICDE.2019.00023

[15] P. Bansal, P. Deshpande, and S. Sarawagi, "Missing value imputation on multidimensional time series," *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 2533–2545, 2021. [Online]. Available: http://www.vldb.org/pvldb/vol14/p2533-bansal.pdf

[16] M. Khayati, P. Cudré-Mauroux, and M. H. Böhlen, "Scalable recovery of missing blocks in time series with high and low cross-correlations," *Knowl. Inf. Syst.*, vol. 62, no. 6, pp. 2257–2280, 2020. [Online]. Available: https://doi.org/10.1007/s10115-019-01421-7

[17] X. Miao, Y. Wu, L. Chen, Y. Gao, J. Wang, and J. Yin, "Efficient and effective data imputation with influence functions," *Proc. VLDB Endow.*, vol. 15, no. 3, pp. 624–632, 2021. [Online]. Available: http://www.vldb.org/pvldb/vol15/p624-miao.pdf

[18] H. Yu, N. Rao, and I. S. Dhillon, "Temporal regularized matrix factorization for high-dimensional time series prediction," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 847–855.

[19] X. Yi, Y. Zheng, J. Zhang, and T. Li, "ST-MVL: filling missing values in geo-sensory time series data," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 2016, pp. 2704–2710.

[20] L. Li, J. McCann, N. S. Pollard, and C. Faloutsos, "Dynammo: mining and summarization of coevolving sequences with missing values," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, 2009, pp. 507–516. [Online]. Available: https://doi.org/10.1145/1557019.1557078

[21] R. Mazumder, T. Hastie, and R. Tibshirani, "Spectral regularization algorithms for learning large incomplete matrices," *J. Mach. Learn. Res.*, vol. 11, pp. 2287–2322, 2010. [Online]. Available: https://dl.acm.org/doi/10.5555/1756006.1859931

[22] O. G. Troyanskaya, M. N. Cantor, G. Sherlock, P. O. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman, "Missing value estimation methods for DNA microarrays," *Bioinform.*, vol. 17, no. 6, pp. 520–525, 2001. [Online]. Available: https://doi.org/10.1093/bioinformatics/17.6.520

[23] X. Li, H. Li, H. Lu, C. S. Jensen, V. Pandey, and V. Mark, "Missing value imputation for multi-attribute sensor data streams via message propagation," in *Proceedings of the VLDB Endowment*, vol. 17, no. 3, 2023, pp. 345–358.

[24] J. Yoon, W. R. Zame, and M. van der Schaar, "Estimating missing data in temporal data streams using multi-directional recurrent neural networks," *IEEE Trans. Biomed. Engineering*, vol. 66, no. 5, pp. 1477–1490, 2019. [Online]. Available: https://doi.org/10.1109/TBME.2018.2874712

[25] K. Wellenzohn, M. H. Böhlen, A. Dignös, J. Gamper, and H. Mitterer, "Continuous imputation of missing values in streams of pattern-determining time series," in *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017.*, 2017, pp. 330–341.

[26] W. Cao, D. Wang, J. Li, H. Zhou, L. Li, and Y. Li, "BRITS: bidirectional recurrent imputation for time series," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 6776–6786. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/734e6bfcd358e25ac1db0a4241b95651-Abstract.html

[27] X. Ren, K. Zhao, P. J. Riddle, K. Taskova, Q. Pan, and L. Li, "DAMR: dynamic adjacency matrix representation learning for multivariate time series imputation," *Proc. ACM Manag. Data*, vol. 1, no. 2, pp. 188:1–188:25, 2023. [Online]. Available: https://doi.org/10.1145/3589333

[28] J. Mei, Y. de Castro, Y. Goude, and G. Hébrail, "Nonnegative matrix factorization for time series recovery from a few temporal aggregates," in *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, 2017, pp. 2382–2390.

[29] L. Balzano, Y. Chi, and Y. M. Lu, "Streaming PCA and subspace tracking: The missing data case," *Proceedings of the IEEE*, vol. 106, no. 8, pp. 1293–1310, 2018. [Online]. Available: https://doi.org/10.1109/JPROC.2018.2847041

[30] X. Shu, F. Porikli, and N. Ahuja, "Robust orthonormal subspace learning: Efficient recovery of corrupted low-rank matrices," in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, 2014, pp. 3874–3881. [Online]. Available: https://doi.org/10.1109/CVPR.2014.495

[31] J. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1956–1982, 2010. [Online]. Available: https://doi.org/10.1137/080738970

[32] M. Khayati, A. Lerner, Z. Tymchenko, and P. Cudré-Mauroux, "Mind the gap: An experimental evaluation of imputation of missing values techniques in time series," *Proc. VLDB Endow.*, vol. 13, no. 5, pp. 768–782, 2020. [Online]. Available: http://www.vldb.org/pvldb/vol13/p768-khayati.pdf

[33] S. Moritz, A. Sardá, T. Bartz-Beielstein, M. Zaefferer, and J. Stork, "Comparison of different methods for univariate time series imputation in R," *CoRR*, vol. abs/1510.03924, 2015. [Online]. Available: http://arxiv.org/abs/1510.03924

[34] S. Jäger, A. Allhorn, and F. Bießmann, "A benchmark for data imputation methods," *Frontiers Big Data*, vol. 4, p. 693674, 2021. [Online]. Available: https://doi.org/10.3389/fdata.2021.693674

[35] E. Sylligardos, P. Boniol, J. Paparrizos, P. Trahanias, and T. Palpanas, "Choose wisely: An extensive evaluation of model selection for anomaly detection in time series," *Proceedings of the VLDB Endowment*, vol. 16, no. 11, pp. 3418–3432, 2023.

[36] Y. Yao, D. Li, H. Jie, L. Chen, T. Li, J. Chen, J. Wang, F. Li, and Y. Gao, "Simplets: An efficient and universal model selection framework for time series forecasting," *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 3741–3753, 2023. [Online]. Available: https://www.vldb.org/pvldb/vol16/p3741-chen.pdf

[37] Y. Li, Z. Wang, Y. Xie, B. Ding, K. Zeng, and C. Zhang, "Automl: From methodology to application," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, ser. CIKM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 4853–4856. [Online]. Available: https://doi.org/10.1145/3459637.3483279

[38] M. Lindauer, H. H. Hoos, F. Hutter, and T. Schaub, "Autofolio: An automatically configured algorithm selector," *J. Artif. Intell. Res.*, vol. 53, pp. 745–778, 2015. [Online]. Available: https://doi.org/10.1613/jair.4726

[39] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[40] C. Wang, Q. Wu, M. Weimer, and E. E. Zhu, "Flaml: A fast and lightweight automl library," in *Fourth Conference on Machine Learning and Systems (MLSys 2021)*, April 2021. [Online]. Available: https://www.microsoft.com/en-us/research/publication/flaml-a-fast-and-lightweight-automl-library/

[41] J. Ding, V. Tarokh, and Y. Yang, "Model selection techniques: An overview," pp. 16–34, 2018. [Online]. Available: https://doi.org/10.1109/MSP.2018.2867638

[42] M. Khayati, Q. Nater, and J. Pasquier, "Imputevis: An interactive evaluator to benchmark imputation techniques for time series data," *Proc. VLDB Endow.*, vol. 17, no. 12, pp. 4329–4332, 2024. [Online]. Available: https://www.vldb.org/pvldb/vol17/p4329-khayati.pdf

[43] M. Khayati, A. Lerner, Z. Tymchenko, and P. Cudré-Mauroux, "Imputebench framework. https://github.com/eXascaleInfolab/bench-vldb20," 2020.

[44] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.

[45] D. W. Ruck, S. K. Rogers, and M. Kabrisky, "Feature selection using a multilayer perceptron," *Journal of neural network computing*, vol. 2, no. 2, pp. 40–48, 1990.

[46] L. O. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 6639–6649. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/14491b756b3a51daac41c24863285549-Abstract.html

[47] M. Mahdavi, Z. Abedjan, R. C. Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, "Raha: A configuration-free error detection system," in *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, Eds. ACM, 2019, pp. 865–882. [Online]. Available: https://doi.org/10.1145/3299869.3324956

[48] T. Mu, H. Wang, S. Zheng, Z. Liang, C. Wang, X. Shao, and Z. Liang, "Tsc-automl: Meta-learning for automatic time series classification algorithm selection," in *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 2023, pp. 1032–1044. [Online]. Available: https://doi.org/10.1109/ICDE55515.2023.00084

[49] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, pp. 185:1–185:52, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-558.html

[50] M. Abdallah, R. A. Rossi, K. Mahadik, S. Kim, H. Zhao, and S. Bagchi, "Autoforecast: Automatic time-series forecasting model selection," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, M. A. Hasan and L. Xiong, Eds. ACM, 2022, pp. 5–14. [Online]. Available: https://doi.org/10.1145/3511808.3557241

[51] O. Shchur, A. C. Türkmen, N. Erickson, H. Shen, A. Shirkov, T. Hu, and Y. Wang, "Autogluon-timeseries: Automl for probabilistic time series forecasting," *CoRR*, vol. abs/2308.05566, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.05566

[52] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214716015300270

[53] M. D. Mullin and R. Sukthankar, "Complete cross-validation for nearest neighbor classifiers." in *ICML*. Citeseer, 2000, pp. 639–646.

[54] J. A. Rice, *Mathematical Statistics and Data Analysis.*, 3rd ed. Belmont, CA: Duxbury Press., 2006.

[55] D. Tiano, A. Bonifati, and R. Ng, "Feature-driven time series clustering," in *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, Y. Velegrakis, D. Zeinalipour-Yazti, P. K. Chrysanthis, and F. Guerra, Eds. OpenProceedings.org, 2021, pp. 349–354. [Online]. Available: https://doi.org/10.5441/002/edbt.2021.33

[56] T. Henderson and B. D. Fulcher, "An empirical evaluation of time-series feature sets," in *2021 International Conference on Data Mining, ICDM 2021 - Workshops, Auckland, New Zealand, December 7-10, 2021*. IEEE, 2021, pp. 1032–1038. [Online]. Available: https://doi.org/10.1109/ICDMW53433.2021.00134

[57] I. Jolliffe, *Principal component analysis*. New York: Springer Verlag, 2002.

[58] F. Chazal and B. Michel, "An introduction to topological data analysis: Fundamental and practical aspects for data scientists," *Frontiers Artif. Intell.*, vol. 4, p. 667963, 2021. [Online]. Available: https://doi.org/10.3389/frai.2021.667963

[59] Y. Zhang, Q. Shi, J. Zhu, J. Peng, and H. Li, "Time series clustering with topological and geometric mixed distance," *Mathematics*, vol. 9, no. 9, 2021. [Online]. Available: https://www.mdpi.com/2227-7390/9/9/1046

[60] W. Meesrikamolkul, V. Niennattrakul, and C. A. Ratanamahatana, "Shape-based clustering for time series data," ser. PAKDD'12. Berlin, Heidelberg: Springer-Verlag, 2012, p. 530–541. [Online]. Available: https://doi.org/10.1007/978-3-642-30217-6_44

[61] J. Paparrizos and L. Gravano, "k-shape: Efficient and accurate clustering of time series," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, T. K. Sellis, S. B. Davidson, and Z. G. Ives, Eds. ACM, 2015, pp. 1855–1870. [Online]. Available: https://doi.org/10.1145/2723372.2737793

[62] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, oct 2008. [Online]. Available: https://doi.org/10.1088/1742-5468/2008/10/p10008

[63] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, pp. 606–660, 2017.

[64] D. H. Anh, K. Eamonn, K. Kaveh, Y. C.-C. Michael, Z. Yan, G. Shaghayegh, R. C. Ann, Yanping, H. Bing, B. Nurjahan, B. Anthony, M. Abdullah, B. Gustavo, and Hexagon-ML, "The ucr time series classification archive," October 2018.

[65] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[66] B. F. U. S. F. O. for the Environment, "hydrology data. https://www.hydrodaten.admin.ch/en," 2022.

[67] M. Lindauer, F. Hutter, H. H. Hoos, and T. Schaub, "Autofolio: An automatically configured algorithm selector (extended abstract)," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August*

19-25, 2017, C. Sierra, Ed. ijcai.org, 2017, pp. 5025–5029. [Online]. Available: https://doi.org/10.24963/ijcai.2017/715

[68] S. Suthaharan, *Decision Tree Learning*. Boston, MA: Springer US, 2016, pp. 237–269. [Online]. Available: https://doi.org/10.1007/978-1-4899-7641-3_10

[69] M. Khayati, I. Arous, Z. Tymchenko, and P. Cudré-Mauroux, "ORBITS: online recovery of missing values in multiple time series streams," *Proc. VLDB Endow.*, vol. 14, no. 3, pp. 294–306, 2020. [Online]. Available: http://www.vldb.org/pvldb/vol14/p294-khayati.pdf

[70] R. Godahewa, C. Bergmeir, G. I. Webb, R. J. Hyndman, and P. Montero-Manso, "Monash time series forecasting archive," in *Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.