ImputePilot: A Graphical Model Selection Toolkit for Time Series Imputation

Yuanyuan Yao*
National University of Singapore
Singapore
yy.yao@nus.edu.sg

Zhexin Jin*
Zhejiang University
China
zhexinjin@zju.edu.cn

Lu Chen
Zhejiang University
China
luchen@zju.edu.cn

Anthony K. H. Tung
National University of Singapore
Singapore
atung@comp.nus.edu.sg

Mourad Khayati
University of Fribourg
Switzerland
mourad.khayati@unifr.ch

ABSTRACT

Time series recorded by IoT sensors are frequently disrupted by device failures, resulting in missing blocks of consecutive values that hinder downstream analysis. While several imputation algorithms exist to recover such gaps, their effectiveness varies greatly depending on time series features and the size and position of the missing blocks. Identifying the most optimal algorithm for a given incomplete time series remains a difficult and time-consuming task, requiring significant benchmarking effort.

In this demonstration, we present ImputePilot, an interactive tool that automatically steers users toward the most suitable imputation algorithm for their time series data. Powered by a user-friendly AutoML engine trained on a large and diverse collection of real-world series, ImputePilot delivers stable and accurate algorithm recommendations without requiring any manual tuning. Through its intuitive web-based interface, users can load their own time series, simulate synthetic missing blocks mimicking real-world sensor faults, instantly receive an algorithm recommendation, and visually compare imputation outcomes across multiple algorithms.

PVLDB Reference Format:
Yuanyuan Yao, Zhexin Jin, Lu Chen, Anthony K. H. Tung, and Mourad Khayati. ImputePilot: A Graphical Model Selection Toolkit for Time Series Imputation. PVLDB, 19(12): 4742 - 4745, 2026.
doi:10.14778/3827998.3828111

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Marblue0609/ImputePilot/.

1 INTRODUCTION

The Internet of Things (IoT) has transformed data collection across various applications, connecting billions of smart sensors that continuously record time series data in real time. In practice, these devices are prone to temporary failures caused by power loss, interference, or hardware malfunctions, leaving the resulting time series with missing blocks of consecutive values. Processing such incomplete data is known to degrade the quality of downstream analytics tasks, including forecasting, anomaly detection, and classification, and can yield substantially wrong results.

A rich body of imputation algorithms has emerged over the years to recover missing blocks in time series. These algorithms adopt fundamentally different strategies, ranging from matrix completion and pattern-based techniques to deep neural networks and LLMs, and consequently exhibit widely varying performance tradeoffs depending on the properties of the data. No single algorithm dominates across all settings: a technique that excels on periodic, highly correlated series may perform poorly on data with erratic fluctuations or complex topological structure. Identifying the most suitable algorithm for a given faulty time series, therefore, remains a persistent and consequential challenge for practitioners.

To address this challenge, we have previously proposed ImputeBench [2], a comprehensive benchmark for evaluating time series imputation algorithms, and ImputeVIS [3], an interactive graphical tool that exposes ImputeBench's features through a visual dashboard. These tools have advanced the state of the art in benchmarking, but they leave a fundamental question unanswered: given a new, unseen time series, which imputation algorithm should one use in the first place? ImputeVIS and ImputeBench can evaluate algorithms after the fact, but they do not automatically recommend the best one before the repair is performed.

To fill this gap, we recently introduced A-DARTS [1], a user-friendly AutoML system for automatic algorithm selection of time series imputation. A-DARTS learns the behavior of imputation algorithms over a curated set of real-world time series spanning various domains. Its recommendation engine, ModelRace, employs a two-phase pruning strategy that allows multiple classifier pipelines to survive the selection process, combining both statistical and topological time series features to achieve accurate and stable recommendations. Experimentally, A-DARTS selects the best imputation algorithm 20% more frequently than the closest AutoML baseline and produces recommendations with 2.5 x less error variance, eliminating the stability problem that affects all existing model selection frameworks [4–6]. Furthermore, using A-DARTS to select the imputation algorithm before forecasting improves prediction accuracy by up to 80% on complex datasets.

*These authors contributed equally to this work.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828111

In this paper, we present ImputePilot, a graphical interface that makes the automatic repair capabilities of A-DARTS accessible to practitioners through an end-to-end interactive pipeline. Starting from the upload of a faulty time series and visualization of its missing block structure, ImputePilot guides users through algorithm recommendation, repair execution, and result inspection. Crucially, by exposing the underlying mechanisms of A-DARTS, including feature extraction and candidate algorithm ranking, ImputePilot fosters interpretability, allowing users to understand the rationale behind each recommendation while relieving them of the burden of manual algorithm selection and benchmarking.

The rest of this paper is organized as follows. Section 2 provides background on A-DARTS. Section 3 presents the ImputePilot architecture and its features. Section 4 introduces the demonstration scenarios that attendees will experience.

2 BACKGROUND

A-DARTS addresses the problem of automatic imputation algorithm selection through three tightly integrated components. ModelRace is the model selection engine that searches over a large space of classifier pipelines and retains winners for ensemble voting. A topological feature extractor complements standard statistical features by capturing the shape and temporal structure of time series in a geometrically meaningful way. Finally, an incremental clustering algorithm enables efficient labeling of large training corpora by propagating imputation labels across groups of similar series.

2.1 ModelRace

The core contribution of A-DARTS is ModelRace, an iterative pipeline selection algorithm that searches over a space of over 100K candidate pipelines, each defined as a tuple of a classifier type, its hyperparameters, and a feature scaling strategy.

The algorithm takes a seed set containing at least one pipeline per classifier type under consideration. At each iteration, a synthesizer generates new candidate pipelines by modifying exactly one parameter of an existing pipeline at a time, either changing a hyperparameter or substituting a different feature scaler, ensuring that the search remains anchored to known good configurations while incrementally exploring nearby ones. All candidates are then trained on partitions of an incrementally growing subset of the labeled training data, and evaluated against a held-out test set.

Pruning proceeds in two phases. In the first phase, early termination is applied within a fold: if a pipeline's score on the current fold is significantly worse than the best score observed so far on the same fold, it is immediately discarded without completing training on the remaining folds. In the second phase, pairwise t-tests are conducted over the score distributions of all surviving pipelines. If two pipelines are statistically indistinguishable in performance, the one with the lower mean score is eliminated. This test is applied at the pipeline level rather than the classifier level, such that two differently configured instances of the same classifier type can both survive if their score distributions are sufficiently distinct. This is a deliberate departure from existing AutoML systems, which treat all variants of a classifier as a single candidate and eliminate them together. The training subset is expanded after each iteration, giving surviving pipelines progressive access to more training data.

Each pipeline is scored using a weighted combination of F1-score, Recall@3, and normalized runtime:

$$\text{score}_\theta = \frac{(\alpha \cdot f_1 + \beta \cdot r_3) - (\gamma \cdot \text{time})}{\alpha + \beta + \gamma} \tag{1}$$

where α, β, and γ are weighting coefficients. The weights are tuned so that accuracy is prioritized without incurring high computational overhead. At inference time, the surviving pipelines each produce a probability distribution over imputation algorithms for a given query series, and a soft voting mechanism averages these distributions. The algorithm with the highest average probability is recommended.

2.2 Topological Feature Extractor

A-DARTS introduces a topological feature extractor designed to capture the shape of a time series in ways that statistical measures cannot. The process begins by embedding the input series into a time-delay space: each time step is mapped to a vector of the form

$$v_p^\tau(j) = (v_j,\, v_{j+\tau},\, \ldots,\, v_{j+(d-1)\tau}) \tag{2}$$

where τ is the time shift and d is the embedding dimension. This transforms the one-dimensional series into a point cloud in a higher-dimensional space, making non-linear temporal relationships and cyclic structures geometrically visible.

A persistence diagram is then constructed from this point cloud. The diagram tracks topological features, referred to as "holes" in the embedding, by recording the time at which each pattern is born and the time at which it disappears. Each point (b_i, d_i) in the diagram represents one such pattern, with its lifespan $d_i - b_i$ indicating its significance. Patterns with long lifespans correspond to prominent, stable structures in the series, such as periodicity or sustained trends, while short-lived ones correspond to noise. The distribution of points in the persistence diagram is then summarized into a fixed-length feature vector, which is concatenated with the statistical features before being passed to the classifiers.

2.3 Incremental Clustering

To generate labeled training data efficiently across all training datasets, A-DARTS clusters time series and propagates labels at the cluster level. The algorithm operates in two phases. In the initial phase, the full set of series is recursively split: any cluster whose average pairwise cross-correlation falls below a user-defined threshold is subdivided into a number of sub-clusters proportional to its size. This continues until all clusters meet the correlation threshold, producing a set of tight but potentially numerous clusters.

The second phase reduces the number of clusters through merge and move operations, both guided by a correlation gain criterion. If merging two clusters yields a positive gain, they are combined; otherwise, the algorithm attempts to move individual series from small clusters into others where they would increase internal correlation. The process is guaranteed to terminate because no time series can return to a cluster it has left without producing a negative gain. The result is a compact set of highly correlated clusters, each labeled with a single imputation algorithm that is then propagated to all cluster member time series.

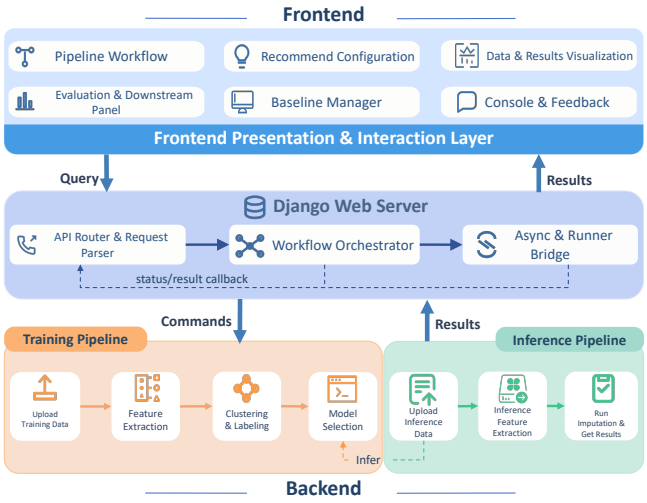

Figure 1: Architecture Overview.

3 IMPUTEPILOT OVERVIEW

Figure 1 provides an overview of the ImputePilot architecture, which consists of (i) an interactive frontend for configuring workflows, managing baselines, visualizing data and results, inspecting evaluation and downstream analysis, and collecting console logs and user feedback, (ii) a backend for training and inference, and (iii) a Django-based web server layer, which acts as the central coordination layer, receiving user queries from the presentation layer and returning execution results.

Frontend. The frontend serves as the presentation and interaction layer of ImputePilot. It integrates several functional modules, including *Pipeline Workflow*, *Recommend Configuration*, *Baseline Manager*, *Data & Results Visualization*, *Evaluation & Downstream Panel*, and *Console & Feedback*. Through these modules, users can define execution processes, select or recommend configurations, manage baseline methods, inspect intermediate and final results, evaluate downstream performance, and monitor runtime feedback.

Backend. The backend serves as the core computational layer and is organized into two main components. The first is the training pipeline, which handles offline model construction. When training data is uploaded, the system extracts the key time series features that influence imputation quality. These features drive a clustering step that speeds up, by orders of magnitude, the labeling of each time series with its most effective imputation algorithm. Once labeling is complete, classifiers are trained to identify the most suitable pipeline configuration based on the characteristics of the training datasets. The second component is the inference pipeline, which processes new time series for imputation. Its workflow covers data upload, feature extraction, and execution of the recommended imputation algorithm. By drawing on the model selection results produced during training, the system automatically matches the most appropriate model to the current input, performs missing value imputation, and generates the final recovery outputs.

4 DEMONSTRATION PLAN

We proceed to present the implementation details of ImputePilot, including its system setup and the main demonstration scenarios supported by the system.

4.1 System Setup

ImputePilot takes the burden of benchmarking imputation algorithms and understanding their technical quirks off the shoulders of time series practitioners. To this end, it provides an interface for managing and uploading training datasets, and ships with 107 curated datasets spanning approximately 67K time series from diverse real-world applications. This collection offers broad coverage of the data characteristics commonly encountered in imputation tasks, and supports both univariate and multivariate time series with regularly sampled timestamps.

Users can choose from several advanced imputation algorithms spanning various families. Beyond these built-in options, ImputePilot provides seamless connection with the ImputeGAP library[1], allowing further integration of algorithms and datasets.

The system also supports simulation of diverse missingness patterns by varying the size and position of contiguous missing blocks, with additional patterns importable from ImputeGAP.

4.2 Demonstration Scenarios

ImputePilot supports three core functionalities: offline pipeline training, online algorithm recommendation, and missing value imputation evaluation. Together, these steps form an end-to-end data repair workflow, from data preprocessing to repairing incomplete series and assessing both the upstream and downstream impact of missing value imputation (cf. Figure 2).

4.2.1 Scenario 1: Offline Model Pipeline Training. In this scenario, users can build and train imputation pipelines. They begin by uploading training datasets and labeling the data, before moving on to the training (as illustrated in the upper part of Figure 2).

Feature Extraction. ImputePilot extracts hundreds of features from the input time series using two popular statistical feature extractors: Catch22 and TSFresh, and a new topological extractor (1). The features capture statistical characteristics, temporal dependencies, and shape-based structural patterns. Using this functionality, users can visually compare the values of the extracted features, as well as the variations across training datasets.

Clustering and Labeling. Once the features are established, ImputePilot allows for organizing time series into coherent groups through an incremental clustering strategy (2). Users can fully control the clustering process through two key parameters: correlation threshold and split ratio. The correlation threshold defines the desired similarity within each cluster, triggering further splitting when the average cross-correlation falls below a user-defined threshold, while the split ratio determines the number of sub-clusters generated during each split.

The clustering process is iteratively refined through recursive splitting and optimization, and the interface visualizes the resulting cluster distributions, intra-cluster shape similarity, and clustering

[1] https://imputegap.readthedocs.io/

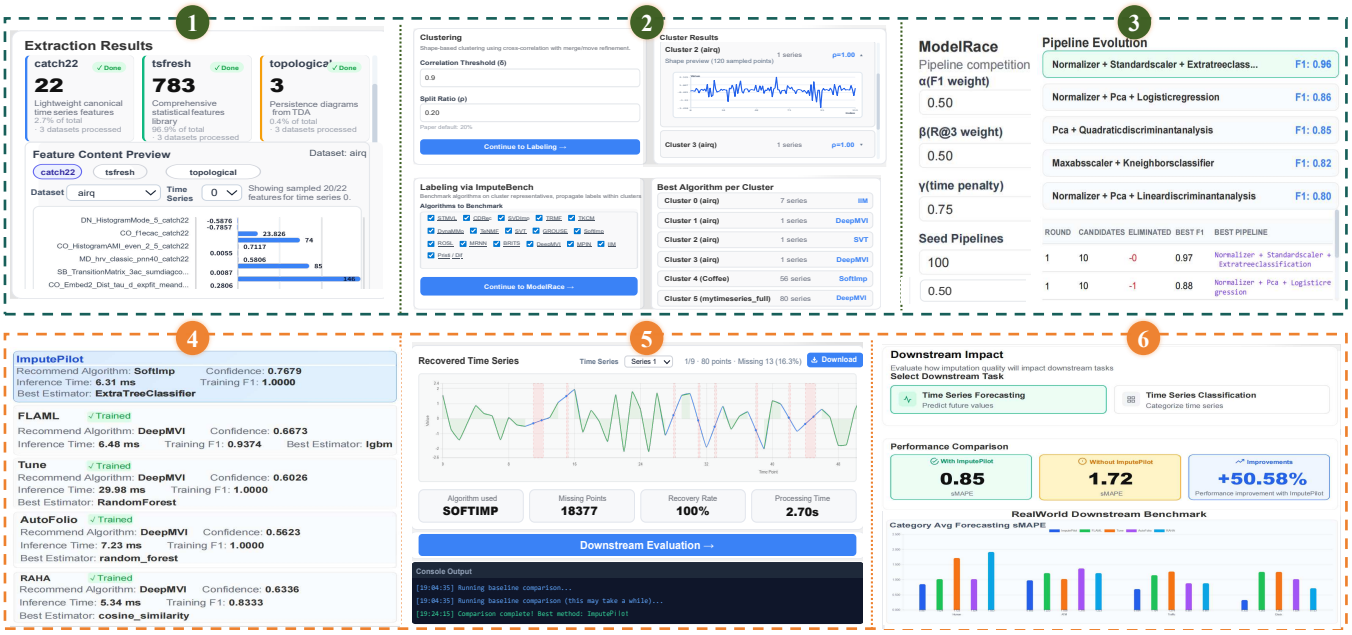

Figure 2: Demonstration UI Overview.

evolution. Once clustering is complete, users can visually trace the label propagation process: a representative series from each cluster is evaluated against all candidate imputation algorithms, and the best-performing algorithm is assigned as the cluster label and propagated to the remaining series within the cluster.

Imputation Pipeline Training. Here, users can train a recommendation model via ModelRace (3). They can vary the parameters of the search process, R@3 Rate, Time Penalty, Seed Pipelines, and T-test value, and inspect their impact. Starting from a set of seed pipelines, the system iteratively generates, evaluates, and prunes candidate pipelines, each composed of a classifier together with feature processing and parameter settings. The best-performing pipelines are combined into an ensemble model for subsequent online imputation recommendation. The interface visualizes the voting results and the final selected pipeline, helping users understand the performance evolution of pipelines and how the different steps of the two-phase pruning intertwine.

4.2.2 Scenario 2: Online Model Recommendation. In the second scenario, attendees can upload an incomplete time series and use ImputePilot to extract its features, apply the trained ensemble model, and receive personalized imputation recommendations (4). The recommendation process is implemented using a soft-voting mechanism, in which multiple candidate pipelines collaboratively vote to identify the most suitable imputation algorithm. Participants can then compare the single best strategy produced by conventional AutoML solutions against ImputePilot's multi-winner pipeline selection strategy. To facilitate this comparison, ImputePilot displays for each AutoML system the recommended algorithm alongside its chosen pipeline, including the classifier names, incurred error, inference time, and confidence in the results.

4.2.3 Scenario 3: Imputation Analysis. In this scenario, users are prompted to apply the recommended algorithm to fill in missing values and compare the reconstructed series (5). They can also examine imputation quality across different missingness rates and patterns, along with the corresponding runtime. Beyond the upstream analysis, ImputePilot allows downstream evaluation to quantify the impact of the repair on subsequent downstream tasks (6). Specifically, the system implements forecasting and classification tasks, with support for different downstream models and metrics.

ACKNOWLEDGEMENT

This work was supported in part by the NSFC under Grant No.62472377. Lu Chen is the corresponding author.

REFERENCES

[1] Mourad Khayati, Guillaume Chacun, Zakhar Tymchenko, and Philippe Cudré-Mauroux. 2025. A-DARTS: Stable Model Selection for Data Repair in Time Series. In *41st IEEE International Conference on Data Engineering, ICDE 2025, Hong Kong, May 19-23, 2025*. IEEE, 2009–2023. https://doi.org/10.1109/ICDE65448.2025.00153

[2] Mourad Khayati, Alberto Lerner, Zakhar Tymchenko, and Philippe Cudré-Mauroux. 2020. Mind the Gap: An Experimental Evaluation of Imputation of Missing Values Techniques in Time Series. *Proc. VLDB Endow.* 13, 5 (2020), 768–782. https://doi.org/10.14778/3377369.3377383

[3] Mourad Khayati, Quentin Nater, and Jacques Pasquier. 2024. ImputeVIS: An Interactive Evaluator to Benchmark Imputation Techniques for Time Series Data. *Proc. VLDB Endow.* 17, 12 (2024), 4329–4332. https://doi.org/10.14778/3685800.3685867

[4] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. 2018. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint arXiv:1807.05118* (2018).

[5] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A Configuration-Free Error Detection System. In *Proc. of the 2019 International Conference on Management of Data, SIGMOD, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 865–882. https://doi.org/10.1145/3299869.3324956

[6] Chi Wang, Qingyun Wu, Markus Weimer, and Erkang Zhu. 2021. FLAML: A Fast and Lightweight AutoML Library. In *Proc. of MLSys 2021, virtual, April 5-9, 2021.*