Journal of Machine Learning Research 23 (2022) 1-7 Submitted 1/21; Revised 5/22; Published 9/22

PyPOTS: A Python Toolkit for Machine Learning on Partially-Observed Time Series

Wenjie Du WDU@TIME-SERIES.AI
PyPOTS Research

Yiyuan Yang YIYUAN.YANG@CS.OX.AC.UK
PyPOTS Research & University of Oxford

Linglong Qian LINGLONG.QIAN@KCL.AC.UK
PyPOTS Research & King's College London

Jun Wang JWANGFX@CONNECT.UST.HK
PyPOTS Research & Hong Kong University of Science and Technology

Qingsong Wen QINGSONGEDU@GMAIL.COM
Squirrel Ai Learning

Editor: My editor

Abstract

`PyPOTS` is an open-source Python library dedicated to data mining and analysis on multivariate partially-observed time series with missing values. Particularly, it provides easy access to diverse algorithms categorized into five tasks: imputation, forecasting, anomaly detection, classification, and clustering. The included models represent a diverse set of methodological paradigms, offering a unified and well-documented interface suitable for both academic research and practical applications. With robustness and scalability in its design philosophy, best practices of software construction, for example, unit testing, continuous integration and continuous delivery, code coverage, maintainability evaluation, interactive tutorials, and parallelization, are carried out as principles during the development of `PyPOTS`. The toolbox is available on PyPI, Anaconda, and Docker. `PyPOTS` is open source and publicly available on GitHub `https://github.com/WenjieDu/PyPOTS`.

Keywords: time series, data science, machine learning, neural network, Python

1 Introduction

Missing data is a pervasive challenge in real-world time series due to sensor errors, communication failures, and other unexpected malfunctions. This issue leads to partially-observed time series (POTS), which can hinder advanced data analysis and modeling Wang et al. (2025). Effective POTS algorithms are therefore highly valuable in many domains, including health monitoring Silva et al. (2012), network failure detection Du et al. (2021), urban traffic forecasting Chen and Sun (2021), and gene expression analysis Bar-Joseph et al. (2003).

To handle missing data in time series, several libraries offer imputation tools, as shown in Table 1, Python's `ImputeBench` Khayati et al. (2020), `Impyute` Law (2021), `Autoimpute` Kearney (2022), and `ImputeGAP` Nater et al. (2025); and R's `Mice` Van Buuren and Groothuis-Oudshoorn (2011) and `ImputeTS` Moritz and Bartz-Beielstein (2017). While a two-step

Table 1: A comparison of available partially-observed time series (POTS) libraries. ✓ means implemented, (✓) indicates ongoing, and ✗ is unimplemented.

Library	Modelling Task					Number of	Number of	Execution
	Imputation	Forecasting	Detection	Classification	Clustering	Algorithms	Datasets	Environment
Mice	✓	✗	✗	✗	✗	1	20	R
ImputeTS	✓	✗	✗	✗	✗	10	6	R
Impyute	✓	✗	✗	✗	✗	10	1	Python
Autoimpute	✓	✗	✗	✗	✗	8	1	Python
ImputeBench	✓	✗	✗	✗	✗	20	10	Python
ImputeGAP	✓	(✓)	✗	✗	✗	34	17	Python
PyPOTS	✓	✓	✓	✓	✓	**52**	**172**	Python

approach, imputing missing values followed by downstream modelling, can be effective, it may underperform in complex tasks. End-to-end methods that directly operate on POTS often yield better results. PyPOTS accommodates both strategies by supporting modular pipelines and integrated models Du et al. (2024).

Despite the importance of modeling partially-observed time series, there has been no dedicated library to support a wide range of data mining tasks on POTS, even in a community as vast as Python. PyPOTS was developed as a comprehensive Python toolbox to fill this gap. It provides end-to-end solutions for various tasks on POTS, moving beyond imputation-only methods to enable more reliable analysis of time series with missing data.

PyPOTS has the following evident advantages compared to existing packages: **1).** It contains 52 algorithms and 172 datasets that cover five modelling tasks on partially-observed time series; **2).** It provides a unified interface, detailed documentation, and interactive tutorials across all algorithms for ease of development and usage; **3).** It utilizes several automation services to measure, track, and ensure library quality, including cross-platform continuous integration with unit testing covering all algorithms, code coverage measurement, and maintainability evaluation; **4).** It employs optimization instruments whenever possible to enhance the scalability of the library. PyPOTS is capable of training models on large datasets but with limited computational resources, parallelly running a model across multiple GPU devices, and enabling all algorithms to train once and run anywhere.

2 Project Concentration

Robustness Guarantee. GitHub actions are leveraged to automatically conduct unit testing with various versions of Python and different operating systems, including *Linux (Ubuntu distribution)*, *macOS*, and *Windows*. There are CI (continuous integration) workflows that automatically run all tests daily and when a pull request is raised to ensure everything is good in the code base. When a new version with a batch of new features is released on GitHub, the CD (continuous delivery) workflows will automatically run the building pipelines to publish the new version on PyPI (Python Package Index), Anaconda, and Docker for delivering to users seamlessly.

Code Quality Assurance. PyPOTS project follows PEP 8 Python code style guide. Code coverage is published on and tracked by *Coveralls*, a web-based code coverage service. And code maintainability is evaluated by *SonarQube Cloud*, an automated tool for software quality assurance. PyPOTS currently has 85% overall code coverage and 95% maintainability (rated rank **A**). The code standards and these measurements are conducted to ensure the code quality and also to serve as safeguards for all pull requests from the open-source community.

Documentation and Tutorials. The comprehensive documentation is developed with *Sphinx*, hosted on *Read the Docs*, and available on https://docs.pypots.com. It keeps the same docstring and rendering style as NumPy Harris et al. (2020). The documentation contains thorough installation instructions, quick-start examples, detailed API references, and an FAQ list. In addition to the documentation, `PyPOTS`' interactive tutorials in Jupyter Notebooks are released in the GitHub repository https://github.com/WenjieDu/BrewPOTS, and a simplified tutorial on *Google Colab* is provided for users to instantly run and explore.

Open-Source Community. As a practical machine learning library, `PyPOTS` receives extensive application within the scientific research community and has been recognized as a component of the `PyTorch Ecosystem`. The code of `PyPOTS` is hosted on GitHub to be completely open source and to encourage collaborations from the community. We have community groups on instant message platforms (e.g., Slack and WeChat) for the community to promptly discuss and provide feedback. A dozen contributors are helping develop the framework, and others have contributed in the way of reporting bugs and requesting features.

3 Design and Implementation

Base Frame. `PyPOTS` is built on top of common libraries like `NumPy` Harris et al. (2020), `Scikit-learn` Pedregosa et al. (2011), `SciPy` Virtanen et al. (2020), and `PyTorch` Paszke et al. (2019) for modeling and computation, and uses `Pandas` and `H5py` Collette et al. (2017) for data handling. Inspired by the API design of `Scikit-learn` Buitinck et al. (2013), it adopts a unified and task-oriented interface for imputation, forecasting, anomaly detection, classification, and clustering. `fit()` processes the training procedure and selects the best model checkpoint. `impute()`, `forecast()`, `detect()`, `classify()`, and `cluster()`, corresponding to each task, run inferences and return results. Unlike conventional libraries that rely on a generic `predict()` method, `PyPOTS` uses task-specific methods to clearly indicate model capabilities. Its modular design, leveraging inheritance and polymorphism, facilitates easy integration of new models.

Scalability Enhancement. To strengthen the scalability of `PyPOTS`, three key features are designed and implemented: 1). Data lazy-loading: Industrial time-series datasets are often large and memory-intensive. To relieve this, `PyPOTS` provides a lazy-loading strategy in addition to full loading for small-sized datasets. Preprocessed data can be stored in HDF5 files, and only the necessary data batches are read into memory on demand during training and inference, significantly reducing memory usage; 2). Multi-device parallel acceleration: Leveraging `PyTorch`, `PyPOTS` enables seamless GPU acceleration. For large datasets and compute-heavy models, users can scale across multiple GPUs simply by specifying device indices (e.g., `["cuda:0", "cuda:1"]`), making parallel training efficient and accessible; 3). Unified model serialization interface: `PyPOTS` offers a consistent, model-agnostic, and device-agnostic interface for saving and loading models. This facilitates transferring trained models across devices and environments, supporting the "train once, run anywhere" paradigm and improving deployment flexibility.

Hyperparameter Optimization. Deep learning models rely heavily on hyperparameter choices, which significantly impact performance Feurer and Hutter (2019). Given that most `PyPOTS` algorithms are neural networks, hyperparameter optimization is essential. This need is amplified by the diversity of time-series datasets across domains, where optimal settings for one dataset may perform poorly on another. To address this, `PyPOTS` integrates Microsoft `NNI` Microsoft (2021), an AutoML toolkit for efficient hyperparameter tuning. Users only need to prepare two configuration files: one for defining the hyperparameter search space and the other for `NNI`'s tuning settings. Once launched, `PyPOTS` will communicate with NNI and provide a web interface to monitor the tuning process and view results, which can be sorted by performance metrics to identify the best configuration.

A Showcase. The code snippet displays how to train a BRITS model to classify the PhysioNet-2012 dataset Goldberger et al. (2000). With a prepared dataset, users only need to write a few lines of code with `PyPOTS` to train a model and produce results on new data.

```python
# Our ecosystem kit will download and preprocess the dataset
from benchpots.datasets import preprocess_physionet2012
data = preprocess_physionet2012(subset='set-a',rate=0.1)
train_set = {"X": data["train_X"], "y": data["train_y"]}
val_set = {"X": data["val_X"], "y": data["val_y"]}
test_set, test_labels = {"X": data["test_X"]}, data["test_y"]

# Initialize and train the BRITS model
from pypots.classification import BRITS
from pypots.nn.functional import calc_binary_classification_metrics
model = BRITS(n_steps=data["n_steps"],
              n_features=data["n_features"],
              n_classes=data["n_classes"],
              rnn_hidden_size=256,
              epochs=5)
model.fit(train_set, val_set)

# Make predictions and evaluate
preds = model.classify(test_set)
metrics = calc_binary_classification_metrics(preds, test_labels)
```

4 Conclusion

This paper presents `PyPOTS`, a comprehensive Python toolkit for modeling partially-observed time series. It contains 52 algorithms and supports five tasks: imputation, classification, clustering, anomaly detection, and forecasting. Being committed to becoming a handy library, `PyPOTS` still has a long way to go in the future. At the time point of writing, most of the algorithms in `PyPOTS` are state-of-the-art methodologies based on neural networks, which can achieve outstanding results, but lacking explainability makes them not applicable in some sensitive fields, e.g., finance and marketing. We plan to include more models, especially ones with explainability, such as probabilistic methods and graph models. Besides, we will pay more attention to spatiotemporal data, which is also a common kind of time series, and implement models that work well with it.

References

Ziv Bar-Joseph, Georg K Gerber, David K Gifford, Tommi S Jaakkola, and Itamar Simon. Continuous representations of time-series gene expression data. *Journal of Computational Biology*, 10(3-4):341–356, 2003.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, et al. API design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

Xinyu Chen and Lijun Sun. Bayesian Temporal Factorization for Multidimensional Time Series Prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2021. ISSN 0162-8828, 2160-9292, 1939-3539. doi: 10.1109/TPAMI.2021.3066551. URL `http://arxiv.org/abs/1910.06366`.

Andrew Collette, James Tocknell, Thomas A Caswell, Darren Dale, Ulrik Kofoed Pedersen, Aleksandar Jelenak, Andrea Bedini, Martin Raspaud, Jialin Lei, Laurence Hole, Matthieu Brucher, Martin Teichmann, Ghislain Antony Vaillant, Jakirkham, Konrad Hinsen, Pierre De Buyl, Axel Huebl, Florian Rathgeber, Toon Verstraelen, Spaghetti Sort, Simon Gregor Ebner, Smutch, Matthew Zwier, Antony Lee, Matthew Brett, Joseph Kleinhenz, Jonah Bernhard, John Tyree, Antoine Pitrou, and Andy Salnikov. H5py/h5py: 2.7.0, March 2017. URL `https://doi.org/10.5281/zenodo.594310`.

Wenjie Du, David Cote, Chris Barber, and Yan Liu. Forecasting loss of signal in optical networks with machine learning. *J. Opt. Commun. Netw.*, 13(10):E109–E121, Oct 2021. doi: 10.1364/JOCN.423667. URL `http://opg.optica.org/jocn/abstract.cfm?URI=jocn-13-10-E109`.

Wenjie Du, Jun Wang, Linglong Qian, Yiyuan Yang, Zina Ibrahim, Fanxing Liu, Zepu Wang, Haoxin Liu, Zhiyuan Zhao, Yingjie Zhou, et al. Tsi-bench: Benchmarking time series imputation. *arXiv preprint arXiv:2406.12747*, 2024.

Matthias Feurer and Frank Hutter. Hyperparameter optimization. *Automated machine learning: Methods, systems, challenges*, pages 3–33, 2019.

A. Goldberger, L. Amaral, L. Glass, Jeffrey M. Hausdorff, P. Ivanov, R. Mark, J. Mietus, G. Moody, C. Peng, and H. Stanley. Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *Circulation*, 101 23:E215–20, 2000.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

Joe Kearney. Autoimpute. `https://github.com/kearnz/autoimpute`, 2022.

Mourad Khayati, Alberto Lerner, Zakhar Tymchenko, and Philippe Cudré-Mauroux. Mind the gap: an experimental evaluation of imputation of missing values techniques in time series. In *VLDB*, 2020.

Elton Law. Impyute. `https://github.com/eltonlaw/impyute`, 2021.

Microsoft. NNI: Neural Network Intelligence, 1 2021. URL `https://github.com/microsoft/nni`.

Steffen Moritz and Thomas Bartz-Beielstein. imputeTS: Time Series Missing Value Imputation in R. *The R Journal*, 9(1):207–218, 2017. doi: 10.32614/RJ-2017-009.

Quentin Nater, Mourad Khayati, and Jacques Pasquier. Imputegap: A comprehensive library for time series imputation. 2025. URL `https://arxiv.org/abs/2503.15250`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Ikaro Silva, George Moody, Daniel J Scott, Leo A Celi, and Roger G Mark. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. *Computing in cardiology*, 39:245, 2012.

Stef Van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of statistical software*, 45:1–67, 2011.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Jun Wang, Wenjie Du, Yiyuan Yang, Linglong Qian, Wei Cao, Keli Zhang, Wenjia Wang, Yuxuan Liang, and Qingsong Wen. Deep learning for multivariate time series imputation: A survey. In *the 34th International Joint Conference on Artificial Intelligence (IJCAI)*, 2025.