# CESNET TS-Zoo: A Library for Reproducible Analysis of Network Traffic Time Series

Milan Kureš[1], Josef Koumar[1,2], and Karel Hynek[1,2]

[1]Czech Technical University in Prague, Prague, Czech Republic
[2]CESNET, a.l.e., Prague, Czech Republic
Email: {kuresmil, josef.koumar, karel.hynek}@fit.cvut.cz

***Abstract*—Time Series Analysis (TSA) is an essential tool in computer networking, supporting tasks such as traffic forecasting, capacity planning, load balancing, quality of service monitoring, behavior profiling, and anomaly detection. Despite its widespread use, the community was limited by the lack of sufficient datasets. Our recent dataset, CESNET-TimeSeries24, finally fills this gap. However, its substantial size presents significant challenges for practical use in research. Therefore, inspired by the other machine learning communities that often develop supportive tools and benchmarks to accelerate research, we introduced a CESNET TS-Zoo library. It is designed to streamline dataset management, experiment setting, and reproducibility in the TSA of network traffic. TS-Zoo provides a standardized API for accessing the CESNET-TimeSeries24 dataset and includes methods for time series preprocessing, multiple dataset partitioning, and data loading for experiments. Furthermore, the preprocessing steps can be exported and imported, enabling reproducible experiments. Therefore, the TS-Zoo library simplifies TSA experiments and enables reproducibility of TSA research applied in computer networking.**

***Index Terms*—Network traffic, Time Series, Time Series Analysis, Machine learning, Open datasets, Open-world evaluation, Library, Toolset, Reproducible evaluation**

## I. Introduction and Background

Network traffic monitoring is a crucial task for network management and overall computer security [1]. In network management, it supports a wide range of key functions including performance monitoring, network troubleshooting, enforcement of Quality of Service and Quality of Experience policies, and strategic planning and optimization of network resources [2]. Moreover, traffic monitoring systems help protect infrastructure by identifying user misconfiguration, policy violations, anomalies, and insider threats [3], [4]. Progress across all these research tasks relies heavily on the availability of high-quality, real-world data.

While various types of real-world network data are available, time series remain one of the most widely used traffic representations. Network traffic time series can be segmented based on temporal granularity into short-term (minutes), mid-to-long-term (hours), and long-term (days and beyond) components. Short-term time series are usually packet time series capturing behaviour of the connection [5], [6]. Mid-to-long-term time series are usually multi-flow time series capturing long-term dependency [7], [8]. Both short-term and mid-to-long-term time series are usually used for network traffic classification and threat detection [5], [6], [8]–[10]. Long-term time series are usually uniform time series created by the aggregation of network traffic of a device or a whole network to capture their behaviour [11], [12]. They are primarily used for network traffic forecasting [11], [12], capacity planning [13], and network digital twins research tasks [14], [15].


However, long-term time series datasets were missing for a long time. Researchers used non-optimal data for most tasks, such as the MAWILab samples [16]. The CESNET-TimeSeries24 dataset [17] addresses this issue and brings a big-data time series dataset from a real-world ISP network. It brings the possibility for research acceleration. Nevertheless, the acceleration is slowed down by the demands of working with big data datasets. In different research domains, like computer vision or natural language processing, researchers solve this problem by developing numerous supportive tools and benchmarks to accelerate the development [18]–[20]. Some libraries focus on working with time series. For example, the *pytsbe*[1] does not contain any preprocessing options, and time series are downloaded with the library without the possibility of choosing a directory. The *datasetsforecast*[2] loads the whole dataset into memory and contains no configurable preprocessing options. The *Time-Series-Library*[3] focuses on benchmarking state-of-the-art models on time series datasets rather than supporting TSA on the time series datasets. More-

This research was partially funded by the Ministry of Interior of the Czech Republic in project "Flow-Based Encrypted Traffic Analysis" (VJ02010024), by the Ministry of Education, Youth and Sports of the Czech Republic in project "e-Infrastructure CZ" (LM2023054), and also by the Grant Agency of the CTU in Prague, grant No. SGS23/207/OHK3/3T/18 funded by the MEYS of the Czech Republic.

[1]https://github.com/ITMO-NSS-team/pytsbe
[2]https://github.com/Nixtla/datasetsforecast
[3]https://github.com/thuml/Time-Series-Library

978-3-903176-75-1 ©2025 IFIP

over, it is not inside PyPi, bash scripts run the benchmarks, and users must manually download datasets from the Google Drive. Since the existing tools are not sufficient for TSA of big data time series datasets, we decided to create a novel supportive tool called CESNET TS-Zoo.

The TS-Zoo is a Python library that speeds up experiments utilizing network traffic time-series-based datasets. TS-Zoo enables downloading datasets from our storage in an optimal format. The library contains state-of-the-art methods for time-series preprocessing and returns time series in several formats, for example, a Pandas DataFrame or a PyTorch dataloader. Therefore, users can focus on modelling and do not waste time on preparing a dataset. Moreover, TS-Zoo enables users to share settings via configuration files, which enables easy reproducibility. The configuration files can be added to the library as standardized benchmarks.

## II. CESNET TS-Zoo Library

The library focuses on providing full support for experiments with network time series datasets. Users can install TS-Zoo from the Python Package Index (PyPi) for Python 3.10 or newer. The source codes are available at GitHub[4] with several tutorials. Moreover, complete library documentation is available using GitHub pages[5].

### A. Datasets

While writing this paper, the library supports two network traffic time series datasets—CESNET-TimeSeries24 and CESNET-AGG23. Both datasets were created by long-term network traffic capturing in the academic ISP network in the Czech Republic operated by the association CESNET.

**CESNET-TimeSeries24 dataset** [17] captures 40 weeks of network traffic from the CESNET ISP network. The dataset offers multivariate time series created through traffic aggregation at three distinct intervals: 10 minutes, 1 hour, and 1 day. Each time series contains 12 different metrics. The dataset contains time series across 283 institutions, 548 institutional subnets, and over 270,000 individual IP addresses.

**CESNET-AGG23 dataset** [21] captures two months of network traffic from the CESNET ISP network. The dataset offers one multivariate time series with 44 different metrics.

### B. Providing Datasets

Usually, datasets are one or multiple CSV files published in data repositories such as Zenodo. Nevertheless, since we aim to process large datasets, CSV files are not suitable. A single file would become excessively large, and splitting it into many smaller files poses challenges for the filesystem and significantly degrades read performance. Therefore, the library provides datasets in the HDF5[6] format. This file format can work effectively with large datasets and has several advantages, such as allowing direct access to data, loading only part of the dataset, allowing read data using multiple processes from one file, and allowing for more efficient data storage with an integrity check.

The HDF5 files are available for download from our S3 storage. We opted to use a dedicated storage service, since standard repositories such as Zenodo or Hugging Face often limit the maximum file sizes. TS-Zoo contains methods to download the dataset after the first demand for the dataset in the code, as shown in Fig. 1.

```python
from cesnet_tszoo.datasets import CESNET_TimeSeries24

dataset = CESNET_TimeSeries24.get_dataset(
    "<path-to-datasets>",
    source_type="institutions",
    aggregation="1_hour",
    dataset_type="time_based"
)
```

Fig. 1: Example of initializing the dataset using TS-Zoo. In the example, the CESNET-TimeSeries24 dataset is used with an institutional aggregation type and a one-hour aggregation window. The dataset is downloaded and stored in the filesystem at the selected location (< path-to-datasets >).

### C. Time Series Loading

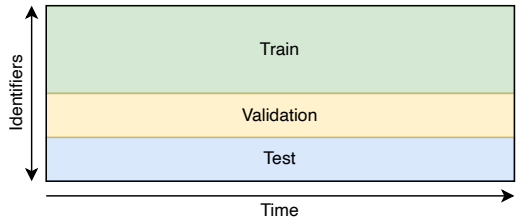
Time series are loaded from the downloaded HDF5 file based on predefined properties, including selection of identifiers, selection of time series metrics, decision on whether to include the exact time of datapoints, and splitting into train, optional validation, and test sets. A particularly important step in this process is the data splitting procedure, which has a significant impact on the final evaluation results. The library provides multiple splitting strategies to accommodate different TSA tasks, such as forecasting, classification, and similarity search.

**Series-based** splitting procedure splits time series based on the different time series identifiers into train, optional validation, and test sets as shown in Fig. 2. The series-based splitting is valuable, for example, for classification based on time series behavior or similarity detection in the same time frame.

**Time-based** approach splits each time series separately based on the time axis, as shown in Fig. 3. The times of splits into train, optional validation, and test sets can be selected in multiple ways, for example, exact timestamp or classical percentage split (i.e., 60:20:20). The train set is always before the validation and test set, and the validation set is always before the test set. This splitting approach is practical, for example, for forecasting or anomaly detection, where we need to predict future data from historical data.

**Time-based splitting with disjoint identifiers** was implemented to support better generalization of algorithms. Time series are split into train, optional validation, and test sets not only by time but simultaneously by identifiers as shown in Fig. 4. This approach allows robust evaluation of a model trained and validated on different time series in a different time span.

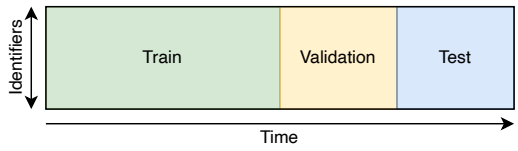
[4]https://github.com/CESNET/cesnet-tszoo
[5]https://cesnet.github.io/cesnet-tszoo/
[6]https://github.com/HDFGroup/hdf5




```python
from cesnet_tszoo.configs import SeriesBasedConfig

config = SeriesBasedConfig(
    time_period='all',
    train_ts=0.6,
    val_ts=0.2,
    test_ts=0.2,
    ...
)
```

Fig. 2: Series-based approach splits time series based on identifiers in the selected time frame



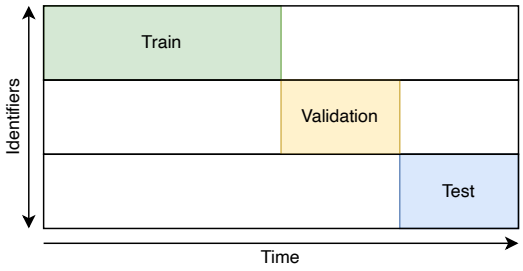

```python
from cesnet_tszoo.configs import TimeBasedConfig

config = TimeBasedConfig(
    ts_ids=1.0,
    train_time_period=0.5,
    val_time_period=0.25,
    test_time_period=0.25,
    ...
)
```

Fig. 3: Time-based approach splits each time series separately based on time.

**Sliding window** approach was also implemented in the TS-Zoo library, since it is extremely common in TSA. The sliding window approach is specified using a window size, which defines the size of historical datapoints (lookback window), a prediction window size, which defines the size of future datapoints, and a window step, which defines the size of the shift in time between windows. The diagram of the sliding window approach with code snippet is shown in Fig. 5. Notably, for the validation and test sets, the first prediction window starts at the beginning of the respective split. Consequently, in this case, the loopback windows are still taken from the training set (for validation) or from the validation set (for testing), and the windowing mechanism will gradually shift forward within the respective split.

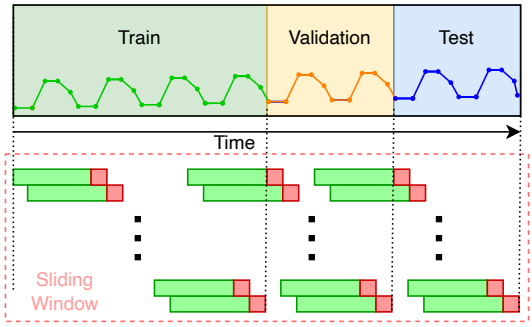
**Obtaining time series data** is possible after the selected configuration is applied for the settings to take effect. The library then generates the prepared training, optional validation, and test sets, which can be returned as a Pandas DataFrame,




```python
from cesnet_tszoo.configs import DisjointTimeBasedConfig

config = DisjointTimeBasedConfig(
    train_ts=0.34,
    val_ts=0.33,
    test_ts=0.33,
    train_time_period=0.5,
    val_time_period=0.25,
    test_time_period=0.25,
    ...
)
```

Fig. 4: Time-based with disjoint identifiers split allows evaluation of the generalization of algorithms




```python
config = TimeBasedConfig(
    ...
    sliding_window_size=168,
    sliding_window_step=1,
    sliding_window_prediction_size=24,
)
```

Fig. 5: Sliding window approach enables model training on pairs of historical and future datapoints

NumPy array, or PyTorch DataLoader as shown in Fig. 6.

### *D. Time Series Preprocessing*

We implemented common time series preprocessing approaches, including filtering, handling anomaly values in the training set, gap filling, and scaling time series metrics. These approaches are applied after the train, validation, and test split, and before transforming the time series into the return format. The sequence of preprocessing steps is shown in Fig. 7.

**Filtering** is the first preprocessing step. The library allows users to select time series based on their identifiers and the percentage of missing values. Additionally, users can specify

```python
dataset.set_dataset_config_and_initialize(config)
# Get data in one of the following ways
train_df = dataset.get_train_df()
train_array = dataset.get_train_numpy()
train_dataloader = dataset.get_train_dataloader()
```

Fig. 6: Apply the config on the dataset and get time series data in the selected format

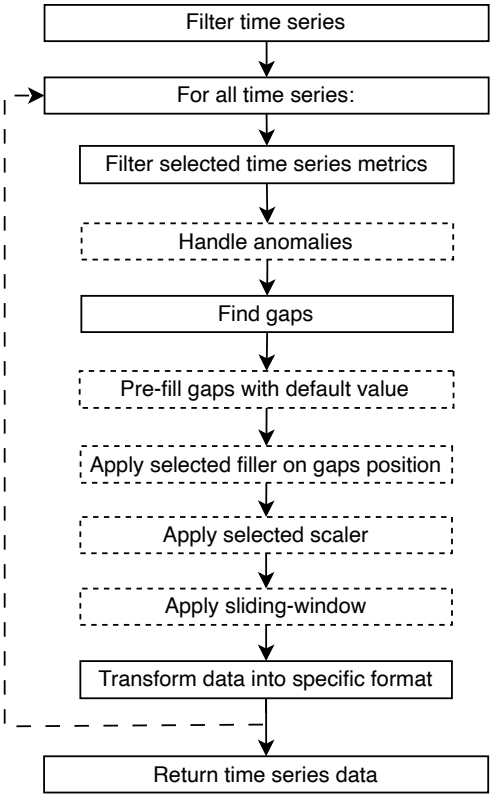



Fig. 7: Sequence of preprocessing steps

the metrics of interest to exclude irrelevant or unusable ones from the resulting time series.

```python
config = TimeBasedConfig(
    ...
    nan_threshold=0.75,
    features_to_take=[  "n_packets",
                        "n_bytes"    ]
)
```

Fig. 8: Filtering by metrics and percentage of gaps

**Anomaly handling** is a process to substitute anomalies for some threshold values to minimize their negative influence during the training process. The library already implements common handling of anomaly values in the training set listed in Table I or allows the user to specify a custom anomaly handling procedure.

**Gaps filling** is another implemented preprocessing method that handles missing time series datapoints. We implemented several state-of-the-art filling methods listed in Table II or allow the user to specify a custom filling procedure.

**Transforming** is another crucial step in modeling that has a large impact on the performance. The library supports common but also sophisticated state-of-the-art transforming methods listed in Table III or allows the user to specify a custom transforming procedure.

TABLE I: Implemented methods of handling anomaly values in the training set

| Method | Description |
|---|---|
| Z-score | Based on mean and standard deviation |
| Interquartile Range | Based on Q1 and Q3 |
| Custom AnomalyHandling class | User can for handling anomaly values define a class that inherits from the AnomalyHandling class |

```python
config = TimeBasedConfig(
    ...
    handle_anomalies_with="z-score",
)
```

TABLE II: Implemented methods of gaps filling

| Method | Description |
|---|---|
| Default value | A predefined value for each metric. |
| Mean | An average value of previous values |
| Forward | A previous value |
| Linear interpolation | A value on a straight line between the previous and next value |
| Custom Filler class | User can for gap filling define a class that inherits from the Filler class |

```python
config = TimeBasedConfig(
    ...
    default_values=0,
    fill_missing_with="forward_filler",
)
```

TABLE III: Implemented methods of transforming

| Method | Description |
|---|---|
| Min Max Scaler | Scales to [0, 1] range |
| Standard Scaler | Zero mean, unit variance |
| Max Abs Scaler | Scales by max absolute value |
| Log Scaler | Log transform of values |
| L2 Normalizer | Unit norm per sample |
| Robust Scaler | Median and IQR scaling |
| Power Transformer | Gaussian-like transformation |
| Quantile Transformer | Uniform or normal mapping |
| Custom Transformer class | User can for transforming define a class that inherits from the Transformer class |

```python
config = TimeBasedConfig(
    ...
    transform_with="standard_scaler",
)
```

### *E. Annotation Support*

The library supports three types of annotations: I) whole time series, for example, device type, II) time point or time

span across all time series, and III) time point or time span in one specific time series. The library supports creating multiple annotation groups, such as types of anomalies. Annotations can be created using the library and exported or imported from a file. Moreover, a GitHub Pull Request can add annotations to the library for other users.

## III. Reproducibility and Benchmarks

Reproducible experiments are essential for building trust in scientific results. However, replicating many existing approaches is often infeasible due to limited methodological descriptions in papers, the absence of published source code, and unclear data handling practices. To address this challenge, we implement functionality for exporting and importing dataset configurations.

Exporting the config to a file is possible, as shown in Fig. 9, and can be distributed using, for example, a GitHub repository or the Zenodo platform. The config file can be imported for reproducing the approach or comparing on the same data, as shown in Fig. 10.

```
dataset.save_config(identifier="<id>")
```

Fig. 9: Export config from TS-Zoo to file

```
dataset.import_config(identifier="<id>")
```

Fig. 10: Import config from file to TS-Zoo

Moreover, the exporting and importing of dataset configs facilitate the creation of community benchmarks distributed within the library. We implement a method to load a predefined benchmark, as shown in Fig. 11. The library now contains several benchmarks for network traffic forecasting, and more can be added in the future using GitHub Pull Requests.

```
from cesnet_tszoo.benchmarks import load_benchmark

benchmark = load_benchmark(
    "<benchmark_hash>",
    "<path-to-datasets>"
)
dataset = benchmark.get_initialized_dataset()
```

Fig. 11: Example of using a benchmark

## IV. Conclusions

We introduced the TS-Zoo library to streamline dataset management, experiment setting, and reproducibility in the TSA of the network traffic field. TS-Zoo provides an API for working with time series datasets, mainly the CESNET-TimeSeries24. Moreover, it includes methods for time series preprocessing, multiple dataset partitioning, and data loading. It also enables configuration sharing to address reproducibility.

As future work, we plan to focus on anomaly detection. We have identified a lack of standardized, annotated benchmarks for outlier and anomaly detection. The CESNET TS-Zoo methods, together with sufficiently complex datasets, provide a strong foundation for creating such benchmarks, including annotations, making it an ideal platform for this purpose.

## References


[1] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia, and P. Casas, "A survey on big data for network traffic monitoring and analysis," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 800–813, 2019.

[2] S. Lee, K. Levanti, and H. S. Kim, "Network monitoring: Present and future," *Computer Networks*, pp. 84–98, 2014.

[3] G. Fernandes, J. J. Rodrigues, L. F. Carvalho, J. F. Al-Muhtadi, and M. L. Proença, "A comprehensive survey on network anomaly detection," *Telecommunication Systems*, vol. 70, pp. 447–489, 2019.

[4] T. Yi, X. Chen, Y. Zhu, W. Ge, and Z. Han, "Review on the application of deep learning in network attack detection," *Journal of Network and Computer Applications*, 2023.

[5] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Distiller: Encrypted traffic classification via multimodal multitask deep learning," *Journal of Network and Computer Applications*, vol. 183, 2021.

[6] M. Shen, Y. Liu, L. Zhu, K. Xu, X. Du, and N. Guizani, "Optimizing feature selection for efficient encrypted traffic classification: A systematic approach," *IEEE Network*, vol. 34, no. 4, pp. 20–27, 2020.

[7] J. Koumar and T. Čejka, "Unevenly spaced time series from network traffic," in *2023 7th Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 2023, pp. 1–4.

[8] J. Koumar and T. Cejka, "Network traffic classification based on periodic behavior detection," in *2022 18th International Conference on Network and Service Management (CNSM)*. IEEE, 2022, pp. 359–363.

[9] J. Koumar, K. Hynek, J. Pešek, and T. Čejka, "Nettisa: Extended ip flow with time-series features for universal bandwidth-constrained high-speed network traffic classification," *Computer Networks*, vol. 240, 2024.

[10] T.-L. Huoh, Y. Luo, P. Li, and T. Zhang, "Flow-based encrypted network traffic classification with graph neural networks," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1224–1237, 2022.

[11] R. Madan and P. S. Mangipudi, "Predicting computer network traffic: a time series forecasting approach using dwt, arima and rnn," in *2018 Eleventh International Conference on Contemporary Computing (IC3)*. IEEE, 2018, pp. 1–5.

[12] T. H. Aldhyani, M. Alrasheedi, A. A. Alqarni, M. Y. Alzahrani, and A. M. Bamhdi, "Intelligent hybrid model to enhance time series models for predicting network traffic," *IEEE Access*, vol. 8, 2020.

[13] A. Mahmood, M. L. M. Kiah, M. R. Z'aba, A. N. Qureshi, M. S. S. Kassim, Z. H. A. Hasan, J. Kakarla, I. S. Amiri, and S. R. Azzuhri, "Capacity and frequency optimization of wireless backhaul network using traffic forecasting," *IEEE Access*, vol. 8, pp. 23 264–23 276, 2020.

[14] R. Verdecchia, L. Scommegna, B. Picano, M. Becattini, and E. Vicario, "Network digital twins: A systematic review," *IEEE Access*, 2024.

[15] L. Hui, M. Wang, L. Zhang, L. Lu, and Y. Cui, "Digital twin for networking: A data-driven performance modeling perspective," *IEEE Network*, vol. 37, no. 3, pp. 202–209, 2022.

[16] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking," in *ACM CoNEXT '10*, Philadelphia, PA, December 2010.

[17] J. Koumar, K. Hynek, T. Čejka, and P. Šiška, "Cesnet-timeseries24: Time series dataset for network traffic anomaly detection and forecasting," *Scientific Data*, vol. 12, no. 1, p. 338, 2025.

[18] B. E. Moore and J. J. Corso, "Fiftyone," *GitHub. Note: https://github.com/voxel51/fiftyone*, 2020.

[19] Dongxu Li et al., "Lavis: A library for language-vision intelligence," *arXiv preprint arXiv:2209.09019*, 2022.

[20] Q. Lhoest, A. V. Del Moral, Y. Jernite, A. Thakur, P. Von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall *et al.*, "Datasets: A community library for natural language processing," *arXiv preprint arXiv:2109.02846*, 2021.

[21] T. Benes, J. Pesek, and T. Cejka, "Look at my network: An insight into the isp backbone traffic," in *2023 19th International Conference on Network and Service Management (CNSM)*. IEEE, 2023, pp. 1–7.