ELoader: A Web Application for Event Log Selection and Preparation for Neural Networks

Henryk Mustroph*, Michel Kunkler and Stefanie Rinderle-Ma

Technical University of Munich, TUM School of Computation, Information and Technology, Garching, Germany

Abstract

One essential step in building reproducible and comparable results in deep learning, while also supporting custom neural network designs, is unified data selection and preparation. For neural network–based process monitoring applications, where the underlying data are primarily event logs, only a few Python libraries support unified event-log selection and preparation. However, there is no practical tool that provides these functionalities in an intuitive manner. Therefore, we present *ELoader*, a prototype web application that enables users to select an event log, prepare it, and download the resulting training, validation, and test sets as Python pickle files bundled in a `.zip` package for direct use in neural networks.

Keywords

Event Log, Data Selection and Preparation, Neural Network, Web Application

Metadata description	Value
Tool name	ELoader
Current version	1.0
Legal code license	GNU Lesser General Public License v3.0
Languages (libraries) used	Frontend: JavaScript (React.js), Backend: Python (FastAPI, PyTorch)
Web App URL	https://power.bpm.cit.tum.de/eloader/
Source code repository	https://github.com/ProbabilisticSuffixPredictionLab/eloader_webapp_pub
Screencast video	https://lehre.bpm.in.tum.de/~kunkler/icpm_2025_demo.mp4

1. Introduction

In deep learning applications, reproducibility of results and fair comparison can be ensured only when the same data are used and the data preparation process is consistent. This also applies to process monitoring tasks, such as predictive process monitoring (PPM), where custom neural networks (NNs) trained on event logs are compared against each other [1]. Therefore, [1] recognize that one of the future research goals of PPM is to focus not only on standard evaluation metrics but also on standardized event log selection and preparation. In this context, data selection refers to using the same event logs across comparable approaches. Data preparation is the umbrella term for preprocessing (e.g., cleaning, feature extraction, normalization), encoding

ICPM Doctoral Consortium and Demo Track 2025, October 20-24, 2025, Montevideo, Uruguay

*Corresponding author.

henryk.mustroph@tum.de (H. Mustroph); michel.kunkler@tum.de (M. Kunkler); stefanie.rinderle-ma@tum.de (S. Rinderle-Ma)

0009-0005-1946-1979 (H. Mustroph); 0000-0002-1920-7322 (M. Kunkler); 0000-0001-5656-6108 (S. Rinderle-Ma)

© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-4088/paper_251.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

(e.g., numerical and categorical), and splitting into training, validation, and test sets. Consistent data preparation is essential for evaluating an NN's performance across different approaches. For example, while most PPM approaches rely on common event logs, such as those from the Business Process Intelligence Challenges (BPICs), some still use other logs, including private or non-shareable ones [2, 3]. This variety increases bias and complicates fair comparison across methods [4]. Existing work has addressed unifying event log selection and preparation in process monitoring by developing Python libraries or extensions. One example is VERONA [4], a Python library designed to support reproducibility and comparability in NN–based process monitoring, which provides users with event log selection and preprocessing functionalities. Additionally, there is a Scikit-learn library extension for process mining [5]. However, while Python libraries offer flexibility and customization for data preparation, they require time to read the documentation and careful environment setup to ensure function calls work correctly. In contrast, practical tools such as web applications hide this complexity and require less effort to set up. Nirdizati [6] is a web-based toolkit for PPM that lets users upload, prepare, and directly train machine learning models on event logs. It targets practitioners building low-code prediction models and dashboards for decision making, rather than selecting open-source event logs and downloading the prepared datasets for custom NN input.

At present, there is no practical tool that enables easy event log selection and preparation. Such a tool would allow custom NNs to be implemented, trained, and tested without the need to develop a dedicated data preparation pipeline, and it would also support the use of standardized datasets for comparing different process monitoring approaches, such as in PPM. This work presents *ELoader*, a web application prototype that simplifies the preparation of open-source event logs for training NNs. Users can select an event log and define various preprocessing parameters, e.g., which event attributes should be encoded, or which event attributes should additionally be obtained via feature engineering. Thereafter, the event log is encoded and split into a training, validation, and test set. The user then receives a single `.zip` package that contains the training, validation, and test sets, each stored as a pickled PyTorch dataset. The tool is intended for practitioners and researchers who aim to build custom or reimplement existing NNs for process monitoring applications without the need to implement their own data preparation pipeline.

2. ELoader Web Application

This section starts with the functionalities of *ELoader*, followed by its architecture, implementation details, design, and a description of the user interface.

2.1. Functionality

The functionality of *ELoader* is split into event log selection, preparation, and splitting.
Selection: The application is linked to a directory containing all event logs for *data selection*. We aim to provide a web application that converts openly accessible and widely used event logs for NN-based process monitoring tasks, thereby simplifying and unifying event log selection.
Preparation: For *data preparation*, we use the same procedure as described in [7]. We apply *feature engineering* on the timestamp value by introducing a `case elapsed time` attribute,

representing the time elapsed since the first event in the case, an `event elapsed time` attribute, representing the time since the last event within the same case (with the value set to 0 for the first event), a `day of the week` attribute, and a `time of day` attribute. The latter two features are incorporated due to the potential influence of periodic trends on the future course of a process. For example, in a company that operates only on weekdays, when an activity is completed on Friday evening, the next activity is unlikely to occur before Monday. Then, we apply *standard scaling* to all continuous event attributes, except for the raw timestamp, and encode missing values as 0. Following [3], we also apply *input padding* to facilitate batch training. Each case is padded with zeros at the beginning to a fixed length, the so-called window size, determined by the maximum case length in the event log, excluding the top 1.5% of the longest cases plus the minimum suffix size. For every categorical event attribute with K unique category classes, we add an additional NA (not available) class for missing values and an unknown class. For the event label attribute we added an end of sequence token (EOS) category class. We then apply for each categorical event attribute *index encoding*. The user can specify a minimum suffix size (i.e., the length of the target event sequence). Each case is then transformed into a list of prefix–suffix samples stored as one concatenated tensor list. Starting with a prefix of length one and increasing until the prefix length reaches case length. The corresponding suffix consists of the remaining events of the case, followed by EOS tokens as needed to ensure it is at least the minimum suffix size. The suffix length decreases from case length $-$ 1 down to 0, with EOS tokens used to pad the suffix if the actual number of remaining events is smaller than the minimum suffix size. For all other event attributes, categorical (other than the event label) and continuous, the values from the last valid event in the suffix are copied forward.

Suppose a case consists of the events $[A, B, C, D]$, and the minimum suffix size is set to 2. The resulting prefix-suffix samples are: prefix $[A]$ with suffix $[B, C, D]$, prefix $[A, B]$ with suffix $[C, D]$, prefix $[A, B, C]$ with suffix $[D, EOS]$, and prefix $[A, B, C, D]$ with suffix $[EOS, EOS]$. This type of case encoding can be used for full-sequence training (i.e., suffix prediction) as in [7, 3], as well as for next-activity training (i.e., next-activity prediction) as in [8].

Split: The data are then split into training, validation, and test sets according to user-defined percentages, ensuring a random yet balanced distribution. When the same event log and split percentages are used, the resulting sets remain identical.

2.2. Architecture and Implementation

Figure 1 depicts the architecture of the *ELoader* web application. Users access the user interface (UI), which is implemented in the frontend using the React.js JavaScript library and Material UI. Communication between the frontend and backend is handled via a REST API. The backend is implemented in Python and is organized into two packages: the `Communication` package, which manages communication with the frontend using the FastAPI library, and the `EL_Functionality` package, which handles all event log preparation functionality using PyTorch. Each `.zip` package returned by the frontend contains a train, validation and test set as a Python pickle file [1] (`.pkl`). Pickle files are encoded byte streams of Python objects [2]) and can be easily stored, read, and decoded using just a few lines of Python code.

[1] https://docs.python.org/3/library/pickle.html, accessed on 2025-09-23
[2] https://docs.pytorch.org/docs/stable/tensors.html, accessed on 2025-09-23

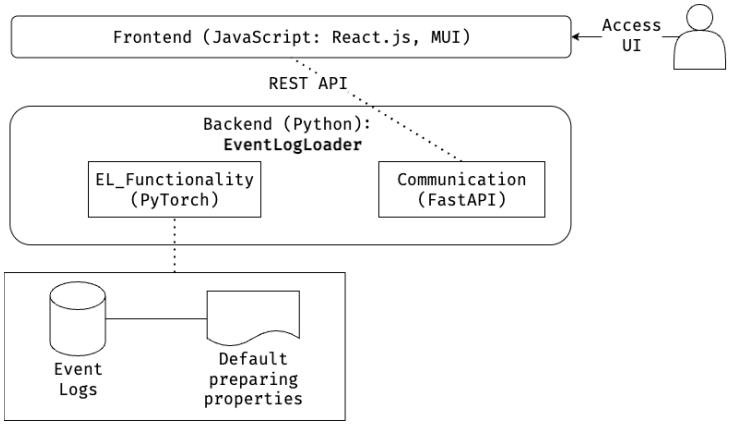

Figure 1: ELoader architecture.

Each set is represented as an object of our custom *EventLogDataset* class, which provides three main attributes: *all_categories*, *encoder_decoder*, and *tensor_list*. The *all_categories* attribute stores a tuple of two lists with the same structure, one for categorical event attributes and one for continuous event attributes. Each list contains multiple tuples, one per event attribute. Each tuple consists of: A string with the attribute name, an integer specifying the number of labels (for continuous attributes this is always 1), and, a dictionary mapping each category class to its corresponding index ID (for continuous attributes the dictionary is empty). The *encoder_decoder* attribute contains configuration information such as the window size, minimum suffix size, and the parameters for standardization and de-standardization of continuous event attributes which can be accessed, for example, via `dataset.encoder_decoder.continuous_encoders['case_elapsed_time']` for the `case elapsed time`. The *tensor_list* attribute contains a tuple of three elements: the first holds the tensors for categorical event attribute values, the second holds the tensors for continuous event attribute values, and the third stores the case IDs corresponding to those tensors. Each event attribute has its own tensor. Each tensor is a matrix with shape, number of samples $\times$ window size, while the list of case IDs has length equal to the number of samples.

The encoded data or the decoded plain data can both be accessed by calling the *EventLogDataset* class that is serialized in the pickle files. An example is provided in the `main.py` that is included in every `.zip` package.

2.3. Design and User Interface

Figure 2 shows the main page of *ELoader*. In the first step, the user can select an event log from the list to be prepared. Once an event log is chosen, the boxes for step two, data preparation, open. The first box displays all individual event log–specific case, event label, timestamp, date format, and time feature attribute names. These values are required for the data preparation functionalities and cannot be changed by the user. The second box contains all custom input fields, which are pre-filled with default values but can be modified by the user. Here, the user

can set the validation set size and the test set size (values between 0 and 1). Additionally, the user can choose the minimum suffix size (a value between 1 and 10). Finally, the user can select which categorical and continuous event attributes should be included in the train validation and test sets. Here only valid event attribute names are allowed. When the user has filled in all data preparation fields, the *Start Encoding* button can be pressed to begin the preprocessing, encoding, and splitting. When the process is finished, the training, validation, and test sets are downloaded in a `.zip` package.

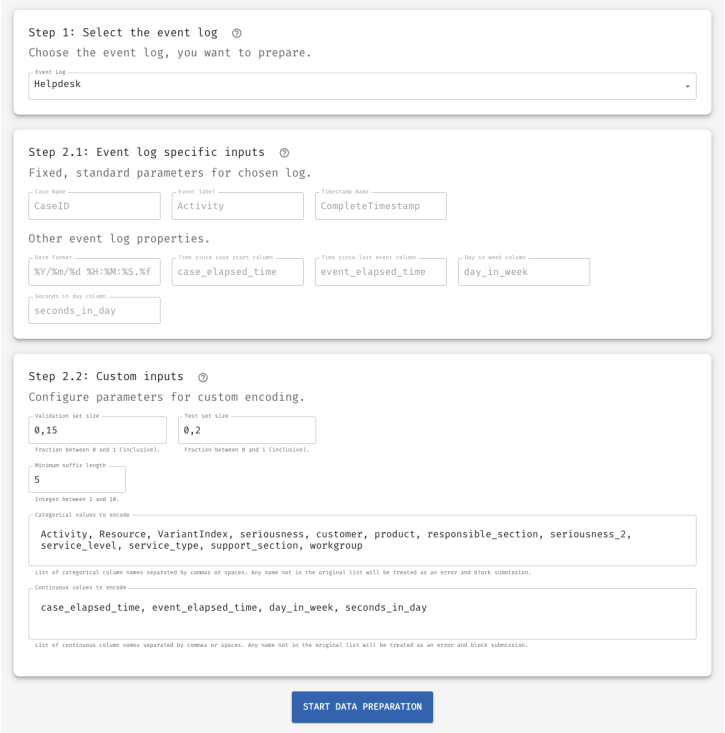

Figure 2: Event log selection, preparation and loading: Main screen.

3. Discussion and Future Work

We tested the functionality of *ELoader* on multiple datasets, including several BPIC event logs, the Helpdesk and the Sepsis event log. As expected, the computation time and resource consumption for data preparation increase with the log size, which also affects the size of the resulting `.zip` package. Furthermore, we used the underlying *ELoader* data preparation functionality in our ICPM 2025 paper *Probabilistic Suffix Prediction of Business Processes*, where it was applied to three different models, our own suffix prediction model [7], the next-activity/suffix prediction model of [8], and the remaining-time prediction model of [9]. These experiments demonstrate that *ELoader* is sufficiently generic to be applied to various PPM tasks. However, *ELoader* is not yet complete, and several additional functionalities are planned for future development, such as

temporal data splitting as described in [10]. Moreover, we intend to gather further requirements from practitioners, researchers, and potential users during the ICPM demonstration to guide the extension and improvement of the tool. Nevertheless, the first prototype already enables easy and efficient event log selection, preparation, and loading for direct input into NNs for PPM. This supports more reproducible and comparable evaluations of different NN–based process monitoring methods.

Declaration on Generative AI

During the preparation of this work, we used ChatGPT in order to: Grammar and spelling check, paraphrase, and reword. After using this tool, we reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] P. Ceravolo, M. Comuzzi, J. De Weerdt, C. Di Francescomarino, F. M. Maggi, Predictive process monitoring: concepts, challenges, and future research directions, Process Science 1 (2024) 2.

[2] N. Mehdiyev, M. Majlatow, P. Fettke, Augmenting post-hoc explanations for predictive process monitoring with uncertainty quantification via conformalized monte carlo dropout, Data Knowl. Eng. 156 (2025) 102402. doi:10.1016/J.DATAK.2024.102402.

[3] B. Wuyts, S. K. L. M. vanden Broucke, J. D. Weerdt, Sutran: an encoder-decoder transformer for full-context-aware suffix prediction of business processes, in: ICPM, 2024, pp. 17–24. doi:10.1109/ICPM63005.2024.10680671.

[4] P. Gamallo-Fernandez, E. Rama-Maneiro, J. C. Vidal, M. Lama, VERONA: A python library for benchmarking deep learning in business process monitoring, SoftwareX 26 (2024) 101734. doi:10.1016/J.SOFTX.2024.101734.

[5] R. S. Oyamada, G. M. Tavares, S. B. Junior, P. Ceravolo, A scikit-learn extension dedicated to process mining purposes, in: Demonstration Track CoopIS, 2023, pp. 11–15. URL: https://ceur-ws.org/Vol-3552/paper-3.pdf.

[6] W. Rizzi, C. D. Francescomarino, C. Ghidini, F. M. Maggi, Nirdizati: an advanced predictive process monitoring toolkit, J. Intell. Inf. Syst. 63 (2025) 259–291. doi:10.1007/S10844-024-00890-9.

[7] H. Mustroph, M. Kunkler, S. Rinderle-Ma, An uncertainty-aware ED-LSTM for probabilistic suffix prediction, CoRR abs/2505.21339 (2025). doi:10.48550/ARXIV.2505.21339.

[8] M. Camargo, M. Dumas, O. G. Rojas, Learning accurate LSTM models of business processes, in: BPM, 2019, pp. 286–302. doi:10.1007/978-3-030-26619-6_19.

[9] H. Weytjens, J. D. Weerdt, Learning uncertainty with artificial neural networks for predictive process monitoring, Appl. Soft Comput. 125 (2022) 109134. doi:10.1016/J.ASOC.2022.109134.

[10] H. Weytjens, J. D. Weerdt, Creating unbiased public benchmark datasets with data leakage prevention for predictive process monitoring, in: BPM Workshops, volume 436, 2021, pp. 18–29. doi:10.1007/978-3-030-94343-1_2.