STREAMER 3.0: Towards Online Monitoring and Distributed Learning

Baudouin Naline, Sandra Garcia-Rodriguez, Karine Zeitouni

STREAMER 3.0: Towards Online Monitoring and Distributed Learning

Baudouin Naline
Université Paris-Saclay, CEA, List
F-91120, Palaiseau, France
baudouin.naline@cea.fr

Sandra Garcia-Rodriguez
Université Paris-Saclay, CEA, List
F-91120, Palaiseau, France
sandra.garciarodriguez@cea.fr

Karine Zeitouni
Université Paris-Saclay, UVSQ, DAVID Lab
F-78035, Versailles, France
karine.zeitouni@uvsq.fr

ABSTRACT

Applications that generate continuous data have proliferated in recent years, and thus the challenge of processing those data streams has emerged. This requires Data Stream Processing frameworks with monitoring capabilities able to detect and react to any non-desired situation. Many streaming use cases deal with distributed sources of data which, for privacy and communication saving purposes, need to be tackled in a distributed manner. Based on the mentioned challenges, this paper presents STREAMER 3.0, an improvement on the former data stream framework with two new modules: (i) a monitoring manager with detection algorithms, alert raising and automatic model updater; and (ii) a distributed learning module relying on federated learning. We showcase these new functionalities with an example of remaining useful life estimation of turbofan engines using an LSTM.

CCS CONCEPTS

• **Computer systems organization** → **Real-time system architecture**; • **Software and its engineering** → **Integrated and visual development environments**; • **Computing methodologies** → **Online learning settings**.

KEYWORDS

Streaming Framework, Data Stream, Distributed Machine Learning, LSTM, RUL Estimation, Federated Learning

ACM Reference Format:
Baudouin Naline, Sandra Garcia-Rodriguez, and Karine Zeitouni. 2023. STREAMER 3.0: Towards Online Monitoring and Distributed Learning. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3583780.3614755

1 INTRODUCTION

With the proliferation of data generated in a continuous manner, data stream processing has become key in research in the recent years. Therefore, there emerged a need for dedicated tools (*Data Stream Processing Systems* - DSPS) able to learn from streams, monitor them and automatically react according to their evolution. But real-world problems may present co-located or distributed sources of data. In the first case, a classical non-distributed solution may be proposed. However, when data come from several sources, non-distributed approaches imply gathering data in the same node which generally involves high communication costs. Moreover, it may be subjected to data confidentiality and security issues. To tackle such privacy and communication-saving purposes, distributed learning methods can be used. Among these methods in the state of the art, Federated Learning (FL) [12] seems to be the best candidate by its popularity and community.

Nowadays, existing DSPS are designed with non-distributed architectures. Among them, we can cite *Avalanche* [11] , *River* [13], STREAMER 1.0 [6], and others already analyzed in [6]. Regarding FL tools, *Flower* [4], *FATE* [10], and *TensorFlow Federated* [16] are the most popular ones. However, any of these frameworks guarantees the direct use of distributed learning in realistic streaming environments (with continuously generated data). Therefore, combining DSPS and FL is needed. This motivated **STREAMER 3.0** release, the contribution of this paper, which enables performing distributed learning for exploration and monitoring purposes.

As previously presented [6], STREAMER is a DSPS for integrating and testing machine learning algorithms in realistic streaming operational contexts. In other words, it implements the whole streaming environment so that data scientists can use it for their own experimentation. The design and creation of this tool pursues three main objectives:

(1) **Control**: an experimenter is required to facilitate the integration and testing of machine learning algorithms into realistic streaming operational environments. Users must be able to define the context and learning configurations.
(2) **Easiness**: it must be easy and quick to install and deploy on any computer. Similarly, all processes must be effortless to follow visually, without consulting the code.
(3) **Reactivity**: a powerful monitoring and detection system is needed to react to an unexpected situation, by giving an alert and updating the model. For this task, the framework must be able to process and incrementally learn from continuous data streams with different contexts.

Since its first publication, STREAMER has been used in several scientific research projects, such as [1, 19], which allowed identifying new challenges and adding new capabilities. Therefore, after continuous improvements, **STREAMER 3.0** is now available [17]. Its main achievements are:

- **Distributed Learning**: it enables performing distributed learning. This way STREAMER can now work with federated learning algorithms.

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3614755

- **Monitoring module**: it allows users to follow (visually and through log files) the evolution of the system while learning/performing the inference or when STREAMER uses detection algorithms to launch the model update.

STREAMER keeps its previous advantages as it can be deployed in any operating system (Windows, Linux, MacOS); accepts the integration of algorithms programmed in a wide variety of programming languages (Java, Python, R, etc.); provides a graphical interface (with Kibana technologies), and is free and accessible for everybody. STREAMER also counts with 2 user guides (a quick start and a detailed one), an HTML documentation (javadoc), and a set of machine learning algorithms, metrics, and use cases that were developed within several projects. STREAMER is open source (https://github.com/streamer-framework/streamer) under the GNU3 license and has an official website: https://streamer-framework.github.io.

2 ARCHITECTURE & FUNCTIONALITY EVOLUTION

STREAMER is a DSPS framework for integrating, testing and monitoring machine learning algorithms in realistic streaming operational contexts. Its modular design makes it flexible, robust and easy to extend. Since STREAMER main architecture was already described in [6], we focus here on its new modules.

2.1 Distributed learning Module

The distributed learning module enables STREAMER to operate in distributed environments. Instead of performing a traditional training process, a STREAMER instance can take part in a distributed learning phase, in which systems collaborate in a network with the aim of reaching a global model.

As mentioned in the introduction, Federated Learning was selected as a distributed learning method. In this context, a *server* orchestrates a network in which *clients* can participate in a collaborative process using their own data source. First, each client, represented by the STREAMER instances, trains its model locally from the server model parameters. Then, clients send their own updated model parameters to the server which computes a global model by aggregating those client parameters. Finally, this process is repeated until the new global model is satisfactory. Since existing FL tools are not suitable for streaming data, we decided to implement our own module. Its development in JAVA was inspired by the Python FL framework "Flower" [4]. Moreover, communication between server and clients is managed by the framework gRPC which uses the protocol HTTP/2 for the transport.

The activation of this module implies that the server takes the lead in the training phase. This way, a new event-based training process will replace module M3 of STREAMER instances [6]. Note that STREAMER can be used for training, inference, or both. If training is disabled, the STREAMER instance is a passive participant. It only retrieves the updated model and uses it for inference. When training is enabled, the STREAMER instance becomes an active participant. Whether learning is performed *online* (in data streams context) or *offline* (with batch data), the client participates in the distributed training process by following the instruction of the server. The only difference between offline and online training is that, in the first case, the process will be directly invoked without using the rest of STREAMER modules.

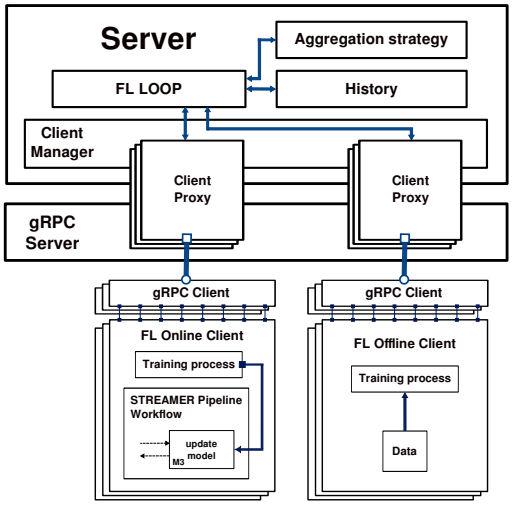

Figure 1: Distributed module architecture.

The module is composed of two new components: the **server**, which purpose is to orchestrate the global training process, and the **client**(s), which is a distributed layer built on top of the STREAMER instance(s) that connects with the server. The architecture overview and the interaction of these components are illustrated in Fig. 1.

Server component . The server component launches two processes in parallel. The *server process*, which will orchestrate the FL loop and activate the difference phases (initialization, training, evaluation), and the *gRPC server process* which is a gRPC interface that manages the client message processing and the server message sending. Several objects are involved during the server process:

- the **FL loop**, which determines the action performed in each round (model parameters initialization, fitting step, evaluation step, server evaluation step).
- the **ClientManager**, which controls the management of the client proxies. It creates, registers, and unregisters client proxies, samples clients, and manages the message waiting.
- the **aggregation strategy** class, which defines the FL strategy to follow. So far just the FedAvg [12] algorithm is implemented, but other strategies can be easily implemented.
- the **history** of the FL experience results, which are then saved in log files.

For further details, clients proxy are defined at the server level as abstract clients with which the server can interact. It can send them requests, receive results from them or disconnect them from the network. Each client proxy represents one single client that could be solicited by the server during the different phases.

Client component. On the other side, the client component is built on top of the STREAMER instance. A new process is opened in parallel to this instance in which a gRPC connection is made with the server. The training and evaluation steps will be now controlled by the server. This component contains a gRPC interface for the server that manages the server message processing as well as the client message sending.

To configure the FL scenario, as to define the streaming context, its parameters can also be set up in the properties files [6]. For instance, users can configure the server address to launch an experiment on several machines, the number of rounds, the minimum number of clients to start a training step, and the maximum quantity of clients to sample during a training step.

2.2 Automatic model update & Monitoring Capabilities

The online monitoring module performs the following tasks:

(1) Monitoring the evolution of the results provided by a model when performing inference and online learning through the use of certain metrics.
(2) Automatically detecting when performance drops below expectations and raise an alert.
(3) Updating the model
(4) Showing graphically and in logs the whole monitoring process.

Users can define the monitoring context through the setup properties files *monitoring.props* and consult at any time the evaluation metric values or monitoring messages in the log files which are updated in real time.

Detection and Alert raising. STREAMER enables monitoring [1...N] metrics with different priorities. It can detect when expected results are dropping to, therefore, raise an alarm and call the model update. For this, a **detection algorithm** was implemented based on two **deviation detection criteria**:

(1) **CUSUM** [2] detector: it sequentially calculates the positive and negative cumulative sum S across data. When the value of S exceeds a certain positive and negative threshold value, a change is detected.
(2) **Threshold** detector: it follows the evolution of the metric values and notifies a deviation when one of them exceeds a specific $threshold$.

The detection algorithm is called each time a new bunch of data is evaluated. It observes each metric separately and raises an alert when a deviation is notified by the detection criteria for more than a certain number of iterations in a row ($robustness$). In the same way, the alert is removed when the detection criteria did not detect any deviation during consecutive $robustness$ iterations. Thus, using the parameter $robustness$ gives stability to the detection algorithm since it avoids constant changes.

When an **alert** is raised, the detection method launches a detection event with format $[id, metric, Start, Stop, Touch]$; where id is the identifier of the use case, $metric$ is the monitored metric, $Start$ is the first iteration the alert was raised, $Stop$ is the iteration in which the alert was stopped, and $Touch$ is the last iteration a deviation was identified.

Based on the previous specifications, we can observe the following constraints:

- Initial values: $Start = Stop = Touch$
- if $Start = Stop \neq Touch$ then alert is launched and ongoing
- if $(Touch + robustness) = Stop$ then alert is deactivated

For each iteration (a new bunch of data processed), if the alert is still activated, STREAMER calls the routine in charge of updating the model with the new data. Once the new model is generated, STREAMER removes the alert but does not stop the detection event. This way, if a deviation is still detected during the next iteration, the alert will be raised again.

Interface and Logs. In order to facilitate the understanding of the monitoring system, STREAMER provides a dynamic graphical interface (figure 2) which shows in real time the metrics evaluations, the alerts, the historical charts, etc.

Users can also follow the evolution of the process (errors, warnings, evaluation metrics across time, monitoring detections, alerts, retraining phases, etc.) at any time by just checking the log files STREAMER generates. Placed in the log folder, the users can find:

- Logs reporting the execution of the algorithms as *displayLogTrain*, *displayLogTest*, *errorLog*, *infoLog* (also shows alerts and detections), *temp_[id]* (temporary log while algorithm is running), etc.
- *metricsLog* where all the evaluation metrics are stored in time.
- The rest of the logs are already described in [6].

3 DEMONSTRATION: REMAINING USEFUL LIFE ESTIMATION OF TURBOFAN ENGINE

Estimating a system Remaining Useful Life (RUL) is one of the most important challenges in the predictive maintenance research domain. Most of the current machine learning methods are based on non-distributed learning processes. When the condition monitoring data of different users are gathered in the same place, data security and privacy issues can appear. To address these challenges, some work combining FL and prognostics has already begun in an attempt to make a distributed approach realistic. They usually apply their new contributions to prognostic benchmarking problems, as for example the NASA's turbofan engine degradation one [15] with the C-MAPSS dataset. In the continuity of existing work combining FL and this dataset [3, 5, 8, 14], this demonstration showcases the new functionalities of our system which not only applies the FL paradigm, but also takes into account the streaming nature of the data. The video, available here: https://www.youtube.com/watch?v=2A4I0aCMQyE, shows the steps involved in setting up a distributed learning process and the application of the new online monitoring functionalities in STREAMER 3.0.

The **C-MAPSS dataset** is composed of four subsets, see Table 1 in [9], containing multivariate temporal data of 21 sensors per turbofan engine. These sensors measure information about the physical state of the engine, such as temperature, pressure, and speed values. They all permit to monitor the health state of the engines and can be used to estimate their RUL. However, differences between subsets come from the variety of operating conditions and fault modes. For this demonstration, we focus on the subset 'FD001' which describes the sensor time series evolution of 100 engines for the training set and 100 other engines for the testing set.

For evaluating the performance of the RUL estimation task, two metrics are usually used. *Root Mean Square Error* (RMSE) is selected as it is a classic metric for regression tasks. *Scoring Function* [7] is another metric specifically created to apply a harder penalization to larger RUL deviations.

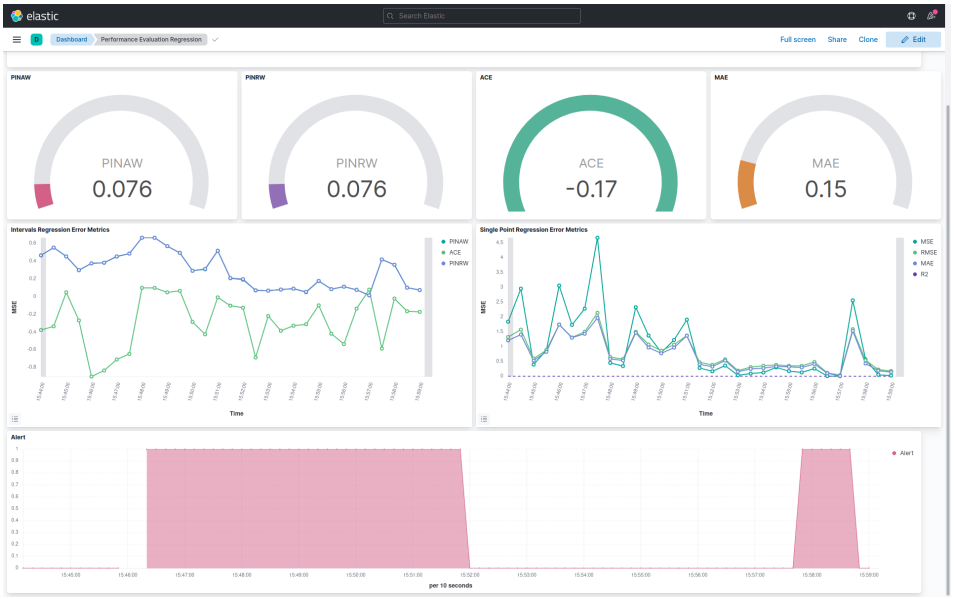

Figure 2: Example of the Monitoring interface

3.1 Scenario Components

This demo performs a distributed learning process by using the new STREAMER module presented in sec. 2.1. Here, a machine learning model is trained offline and then, in parallel, the model is used by an online client to perform RUL estimation inferences.

Because of their strong ability to predict a value from sequential data, we rely on a Long-Short Term Memory (LSTM) [18] for solving this problem. LSTM is a recurrent neural network that does not have long-term time dependency issues by using input gates, forget gates, and output gates that control its information flow. The chosen LSTM is built using one hidden layer with 8 neurons and combined with a linear output layer that estimates the remaining useful life.

The whole process, in common with the RUL estimation, is monitored through the new monitoring module using the *Threshold detector* described in sec. 2.2. There are 5 STREAMER instances that make this demonstration of RUL estimation possible: 1 server, 3 offline clients in charge of the distributed model training, and 1 online client that performs the inference and monitors the incoming data stream. We describe below the tasks performed by each kind:

Server. Initially, a server is configured, launched, and set ready to run a specified number of rounds (see component 'FL loop' in 2.1) while waiting for the right number of clients to connect. The *FedAvg* strategy is selected to aggregate the client model parameters.

Clients - Training process. Then, the training set is split into three subsets of equivalent size which are assigned to three clients. The separation has been achieved by assigning the data from different engines to the clients (33 engines for the first subset, 33 for the second, and 34 for the last one). Each engine has its own sensor data (21 sensors in total). 7 sensors are removed from all the input data because they contain constant values or information that are not representative of the engine degradation. Then, the time series of the 14 remaining sensors are normalized and sliced into sequences of size 31. It constitutes therefore a batch of thousand sequences per client (again grouped by their allocated engines), which correspond to the correct input data format for this simple neural network. The three clients are connected to the server and follow its training process directions. This way, in every FL round, each client trains its model in an offline way with their own associated subset.

Client - Online inference. At the same time, another STREAMER instance is launched and connected to the server in order to predict the RUL based on the incoming data (online inference). It uses a model which is being updated by the server. The data of this fourth client belongs to the testing dataset and is not related to the data used for training. They arrive continuously in blocks of 31 values, grouped by engines, and are subject to the same transformation as the training process, i.e. sensors selection, normalization, and sequences generation. Inference results will be analyzed through the monitoring dashboard and the generated logs. The interface shows a timeline of the engines being evaluated, the evaluations of the model (metrics), and the alerts raised by the system.

4 CONCLUSION

This paper presents **STREAMER 3.0**, a data stream processing framework with new monitoring and distributed learning capabilities. Since preserving the security and the confidentiality of condition monitoring data is key in predictive maintenance, we applied STREAMER 3.0 to create a distributed realistic approach to estimate the RUL of turbofan jet engines. The demo video illustrates a distributed batch training of an LSTM model which is used, in parallel, to estimate the RUL in new incoming streams.

5 ACKNOWLEDGMENTS

This work has been supported by the French government under the "France 2030" program, as part of the SystemX Technological Research Institute within the "Confiance.ai" project.

REFERENCES

[1] Mohammad Alshaer, Sandra Garcia-Rodriguez, and Cedric Gouy-Pailler. 2020. Detecting Anomalies from Streaming Time Series using Matrix Profile and Shapelets Learning. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 376–383.

[2] George A Barnard. 1959. Control charts and stochastic processes. *Journal of the Royal Statistical Society: Series B (Methodological)* 21, 2 (1959), 239–257.

[3] Ali Bemani and Niclas Björsell. 2022. Aggregation Strategy on Federated Machine Learning Algorithm for Collaborative Predictive Maintenance. *Sensors* 22, 16 (Aug. 2022), 6252. https://doi.org/10.3390/s22166252

[4] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. 2022. Flower: A Friendly Federated Learning Research Framework. *arXiv:2007.14390 [cs, stat]* (March 2022).

[5] Maharshi Dhada, Adria Salvador Palau, and Ajith Kumar Parlikad. 2019. Federated Learning for Collaborative Prognosis. In *International Conference on Precision, Meso, Micro and Nano Engineering (COPEN 2019), IIT Indore*. 6.

[6] Sandra Garcia-Rodriguez, Mohammad Alshaer, and Cedric Gouy-Pailler. 2020. STREAMER: A Powerful Framework for Continuous Learning in Data Streams. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 3385–3388.

[7] Felix O. Heimes. 2008. Recurrent neural networks for remaining useful life estimation. In *2008 International Conference on Prognostics and Health Management*. 1–6. https://doi.org/10.1109/PHM.2008.4711422

[8] Sayaka Kamei and Sharareh Taghipour. 2023. A Comparison Study of Centralized and Decentralized Federated Learning Approaches Utilizing the Transformer Architecture for Estimating Remaining Useful Life. *Reliability Engineering & System Safety* 233 (May 2023), 109130. https://doi.org/10.1016/j.ress.2023.109130

[9] Xiang Li, Qian Ding, and Jian-Qiao Sun. 2018. Remaining Useful Life Estimation in Prognostics Using Deep Convolution Neural Networks. *Reliability Engineering & System Safety* 172 (April 2018), 1–11. https://doi.org/10.1016/j.ress.2017.11.021

[10] Yang Liu, Tao Fan, Tianjian Chen, Qian Xu, and Qiang Yang. [n. d.]. FATE: An Industrial Grade Platform for Collaborative Learning With Data Protection. ([n. d.]), 6.

[11] Vincenzo Lomonaco, Lorenzo Pellegrini, Andrea Cossu, Antonio Carta, Gabriele Graffieti, Tyler L. Hayes, Matthias De Lange, Marc Masana, Jary Pomponi, Gido van de Ven, Martin Mundt, Qi She, Keiland Cooper, Jeremy Forest, Eden Belouadah, Simone Calderara, German I. Parisi, Fabio Cuzzolin, Andreas Tolias, Simone Scardapane, Luca Antiga, Subutai Amhad, Adrian Popescu, Christopher Kanan, Joost van de Weijer, Tinne Tuytelaars, Davide Bacciu, and Davide Maltoni. 2021. Avalanche: An End-to-End Library for Continual Learning. arXiv:2104.00405 [cs]

[12] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. *arXiv:1602.05629 [cs]* (Feb. 2017). arXiv:1602.05629 [cs]

[13] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. 2020. River: Machine Learning for Streaming Data in Python. arXiv:2012.04740 [cs]

[14] Raul Llasag Rosero, Catarina Silva, and Bernardete Ribeiro. 2020. Remaining Useful Life Estimation in Aircraft Components with Federated Learning. *PHM Society European Conference* 5, 1 (2020), 9.

[15] Abhinav Saxena, Kai Goebel, Don Simon, and Neil Eklund. 2008. Damage Propagation Modeling for Aircraft Engine Run-to-Failure Simulation. In *2008 International Conference on Prognostics and Health Management*. IEEE, Denver, CO, USA, 1–9. https://doi.org/10.1109/PHM.2008.4711414

[16] https://github.com/tensorflow/federated. 2023.

[17] https://streamer-framework.github.io. 2023.

[18] Shuai Zheng, Kosta Ristovski, Ahmed Farahat, and Chetan Gupta. 2017. Long Short-Term Memory Network for Remaining Useful Life Estimation. In *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*. 88–95. https://doi.org/10.1109/ICPHM.2017.7998311

[19] Jingwei Zuo, Karine Zeitouni, Yehia Taher, and Sandra Garcia-Rodriguez. 2023. Graph convolutional networks for traffic forecasting with missing values. *Data Mining and Knowledge Discovery* 37, 2 (2023), 913–947.