SoftwareX 34 (2026) 102596

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

ObServML: Deployable Python application for compact and modular systems monitoring

Ádám Ipkovich, János Abonyi, Alex Kummer*

HUN-REN-PE Complex Systems Monitoring Research Group, University of Pannonia, Veszprém, H-8200, Hungary

ARTICLE INFO

Keywords:
MLOps
Monitoring
Machine learning
Deep learning
Fault detection

ABSTRACT

ObservML enables the combination of training and deploying ML monitoring models within a single microservices-based system. Its application focuses on monitoring problems that can be solved with fault detection and isolation (FDI), time series analysis, and process mining through an operator-friendly and adaptable framework based on MLOps practices. The framework is developed to connect to RabbitMQ for real-time data communication and MLflow for model versioning. It supports a wide range of machine learning techniques, including decision trees, autoencoders, and time series models, providing a robust toolkit for anomaly detection and predictive maintenance, and can be extended as required.

Metadata

Code metadata (mandatory)

Nr.	Code metadata description	Metadata
C1	Current code version	v1.0.0
C2	Permanent link to code/repository used for this code version	https://github.com/adamipkovich/observml.git
C3	Permanent link to Reproducible Capsule	https://hub.docker.com/r/ipkovichadam/observml
C4	Legal Code License	MIT
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python, Docker, RabbitMQ, Mlflow, FastAPI, Streamlit
C7	Compilation requirements, operating environments & dependencies	Provided in requirements.txt / pyproject.toml
C8	If available Link to developer documentation/manual	https://adamipkovich.github.io/observml/
C9	Support email for questions	ipkovichadam@gmail.com

* Corresponding author.
Email address: kummer.alex@mk.uni-pannon.hu (A. Kummer).

https://doi.org/10.1016/j.softx.2026.102596
Received 30 November 2025; Received in revised form 27 February 2026; Accepted 3 March 2026
Available online 6 March 2026

Software metadata (optional)

Nr.	(Executable) software metadata description	
S1	Current software version	1.0.0
S2	Permanent link to executables of this version	https://github.com/adamipkovich/observml.git
S3	Permanent link to Reproducible Capsule	https://hub.docker.com/r/ipkovichadam/observml
S4	Legal Software License	MIT
S5	Computing platforms/Operating Systems	Containerized
S6	Installation requirements	Docker, for local development, python dependencies are provided in the pyproject.toml
S7	If available, link to user manual - if formally published include a reference to the publication in the reference list	https://adamipkovich.github.io/observml/
S8	Support email for questions	ipkovichadam@gmail.com

1. Motivation and significance

The contribution of fault detection and isolation (FDI) tools in modern industrial systems is critical to ensure state-of-the-art reliability, safety, and operational efficiency, so the industry increasingly relies on data-driven approaches. The demand for adjustable, efficient, and user-friendly tools to handle complex monitoring tasks has increased significantly [1]. There are several monitoring tools, *e.g.,* advanced fault diagnosis analysis library in MatLab for large-scale models that have a Python implementation [2].

The BibMon package for example provides several implemented monitoring tools [3], however, it provides no deployment and MLOps features. Some solutions target very specific problems, such as digital twins to simulate sensors [4], system analysis to understand model behavior [5]. While there are innovative solutions to existing problems, such as monitoring earthquakes whose models can be adjusted [6].

As opposed to Kubernetes-based orchestrators, such as Flyte and Kubeflow, our project focuses on smaller applications. It is not computationally and cost efficient to use Kubernetes on such a scale. It also requires proper Kubernetes Operators to properly monitor the containers that could drive up the costs. Modern end-to-end ML solutions, such as Dataiku, often focus on cloud deployment, which could significantly increase the costs for developers/researchers who are unable to afford their own private cloud systems. Thus, we aim to provide a self-deployable pre-defined system for monitoring, while its modularity can reduce costs and improve the flexibility of ML model use. For more thorough comparison, please see Table 1.

In summary, existing monitoring libraries lack operationalization and MLOps support, while general-purpose MLOps platforms lack domain-specific industrial monitoring capabilities and impose significant deployment overhead. As a result, a unified, lightweight framework that integrates industrial monitoring models, real-time data streaming, and MLOps practices for small- to medium-scale applications remains missing.

Moreover, increasing resource constraints—such as limited availability and rising costs of memory and compute resources in shared and cloud-based environments—further emphasize the need for efficient, resource-aware end-to-end pipelines. Addressing both functional and infrastructural limitations, this work contributes a modular and self-deployable framework designed to minimize memory footprint and deployment overhead while maintaining full monitoring and MLOps capabilities.

We adhere to MLOps principles as predictive systems can provide more harm than benefit without proper specifications. As such, we define the following requirements to assure quality workflow in our package:

- Loosely Coupled Architecture (Modularity) – A flexible, microservices-based architecture.
- Reproducibility – Versioning, tracking and storing ML models.
- Orchestrated – Configuration files to build and deploy ML models.
- Continuous Testing – Implement custom metrics, and KPIs.
- Continuous Delivery – A system that is capable of exposing/deploying custom predictive functionalities.
- Continuous Training – Retraining capabilities affected by real-time data.
- Continuous Monitoring – Performance monitoring of ML models.

Motivated by this requirement specification, we developed a Python package for modular monitoring applications tailored to fault detection and isolation workflows. Our system is built on a microservices architecture leveraging Docker for containerization, RabbitMQ for efficient data handling, and MLflow [10] for experiment tracking. It supports diverse machine learning models, including decision trees, autoencoders, ARIMA, LSTM, *etc*. The modularity of the package allows for straightforward model substitution and customization, enabling users to fine-tune configurations to suit specific industrial datasets and requirements. The models are trained based on one set of instructions (Hydra/YAML configuration files [15]) that the operators provide so that they do not have to manually provide code for specific setups. These features ensure that operators deploy models and monitor their systems in real-time.

Table 1
Comparison of industrial ML tools and ObservML.

Tool / Platform	Main Focus	Key Limitation Compared to ObservML
BibMon [3]	Process monitoring, fault detection, soft sensing	Lacks integrated deployment, real-time streaming, and MLOps lifecycle support
ProgPy [7]	Prognostics and remaining useful life estimation	Focused on model development; no end-to-end operationalization
PM4Py [8]	Process mining from event logs	Not designed for real-time monitoring or microservice-based deployment
PyOD [9]	Generic anomaly and outlier detection	No native support for industrial data streaming or model lifecycle management
MLflow [10]	Experiment tracking and model versioning	Provides MLOps infrastructure but no domain-specific FDI or monitoring models
Fledge [11]	Industrial IoT edge data pipelines and inference	Infrastructure-oriented; lacks built-in monitoring and FDI model toolkit
Dataiku [12]	Enterprise end-to-end ML platform	Cloud-centric and heavyweight; limited suitability for lightweight industrial monitoring pipelines
Flyte [13]	Workflow automation and orchestration	No native industrial monitoring or real-time anomaly detection capabilities
Kubeflow [14]	Kubernetes-native ML pipelines	Requires substantial infrastructure overhead; lacks domain-specific industrial models
ObservML	Integrated industrial monitoring with MLOps	Unified framework combining FDI, time series analysis, real-time streaming, and resource-aware deployment

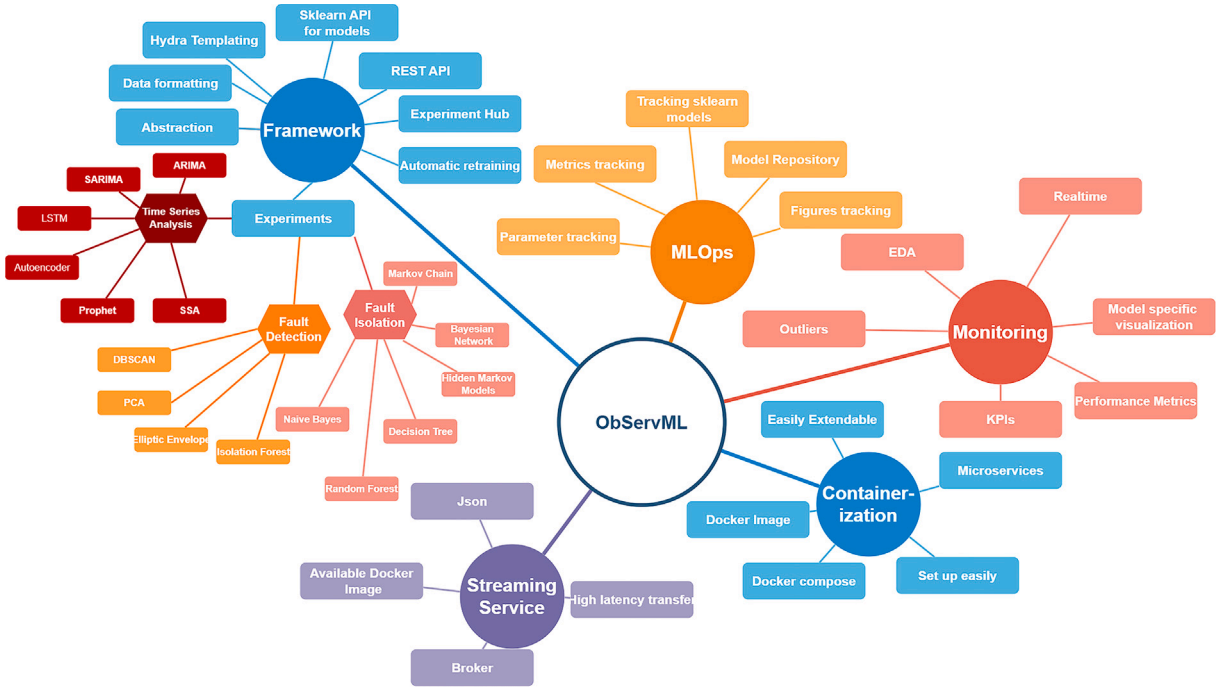

Fig. 1. Concept map of the developed ObservML python package. The colored circles denote the microservices, whilst rectangles present a feature of the service. For the Experiments features, more concrete information is depicted in forms of the problems it can solve (fault detection, fault isolation and time series analysis). Process mining is not on the concept map as it is only an experimental feature.

Fig. 1 presents the main concepts of the package.

Adopting Internet of Things (IoT) technologies and Industry 4.0 practices can significantly enhance industrial performance [16]. However, the presence of sensors does not inherently provide additional value; what really contributes to performance is the analysis of this data. A key focus in manufacturing remains cost reduction, which can be achieved through approaches such as operator behavior analysis [17], manufacturing process optimization [18], alarm management [19], and predictive maintenance [1].

Industry 4.0 faces several challenges, including handling anomalies and noisy data, addressing the lack of labeled datasets, and managing efficiency with large-scale data [20]. Proper infrastructure, such as RabbitMQ and cloud services, can address the challenges of big data. For typical data-driven predictive maintenance tasks, simpler models like Principal Component Analysis (PCA) can effectively tackle the problem of the lack of labeled data [20].

To further reduce the strain on engineers, we have developed the ObservML package for predictive maintenance. This package includes infrastructure for real-time monitoring with fault detection approaches, fault isolation for understanding machine states, time-series analysis to forecast sensor values and detect deviations from expected outcomes. We also provide some options for process mining to analyze machine and operator logs.

The paper presents the architecture, framework, and use of the package for predictive maintenance. By combining modularity with an operator-friendly design, the system aims to provide a straightforward predictive maintenance application. We demonstrate its utility through fault detection and isolation, showcasing its ability to transform monitoring and operational efficiency in an industrial environment. The contribution of the package is as follows:

- A flexible, microservices-based architecture leveraging Docker (Compose), RabbitMQ, and MLflow for containerization, data handling, and model tracking.
- A modular system enabling seamless configuration and substitution of machine learning models from a wide range of packages for diverse datasets and tasks so that operators do not have to write code themselves.
- Easily extendable framework for both experiments and models.
- Ease of integration with data collectors and databases through REST API-based deployment of the models.
- Adherence to MLOps principles, ensuring scalability, reproducibility, and ease of integration into industrial workflows.
- Real-time monitoring frontend application made in Streamlit.

2. Software description

The ObServML package provides a modular framework for monitoring and deploying ML for realtime applications. The methodology for this framework is structured into three key stages: Architecture, Framework, and Use. Section 2.1 outlines the modular design of the system, built on a microservices-based approach leveraging containerization, message brokering, and experiment tracking. This design ensures scalability, reproducibility, and easy integration with additional services. Section 2.2 details the functionality and modularity of the framework of ExperimentHub Service.

Throughout the paper, we highlight the use of the framework with the use of DBSCAN. Problem-specific experiments are introduced for fault detection, fault isolation, time series analysis, and process mining.

2.1. Software architecture

The architecture of the system is built on a modular, microservices-based framework to support fault detection and isolation workflows. Key components include Docker for containerization, RabbitMQ for asynchronous data handling, and MLflow for tracking and versioning models. The backend manages model training, configuration, and data transfer, while the frontend, built with Streamlit, provides a user-friendly interface for real-time visualization and interaction. Communication between components is streamlined through APIs, ensuring scalability, reproducibility, and integration with diverse industrial environments.

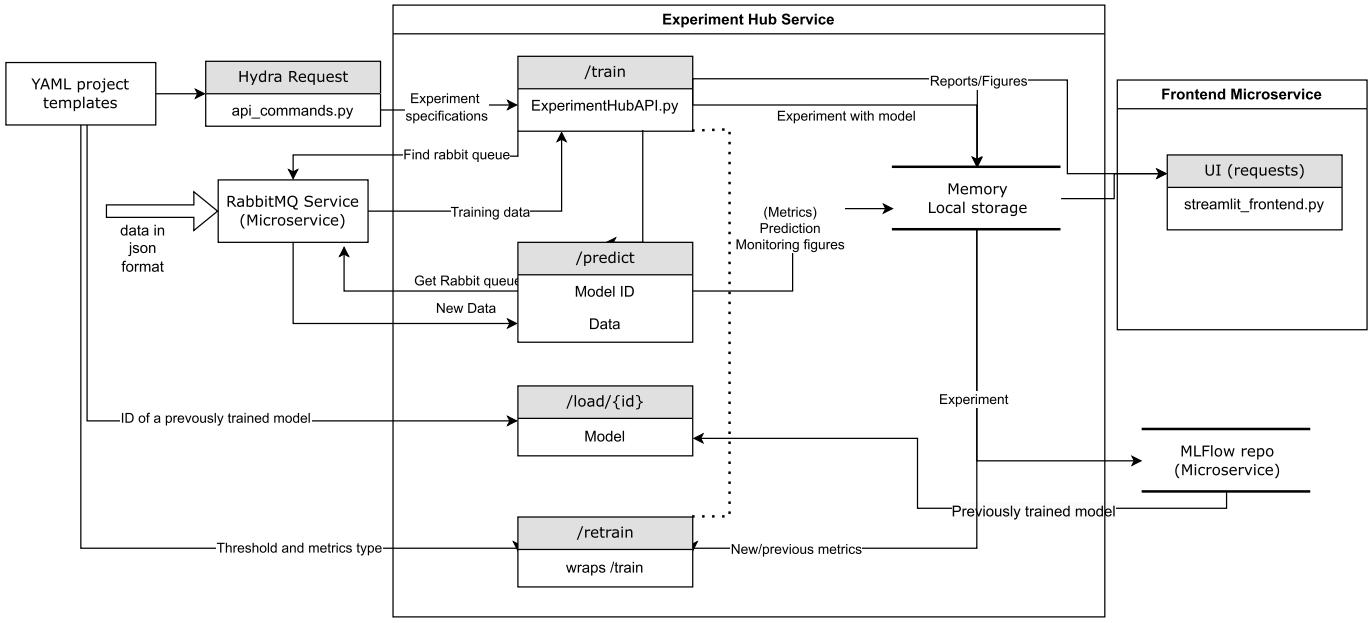

Fig. 2. System Architecture.

Fig. 2 illustrates the current microservice architecture of the package. To use the package, data must first be sent to the RabbitMQ queue. This approach improves security by forcing users to possess valid RabbitMQ credentials, preventing unauthorized access to resource-intensive training or prediction operations, and mitigating potential denial-of-service attacks. The package includes client code examples to demonstrate how to establish a connection to RabbitMQ. The codes contain commands abstracting away the REST API of the ExperimentHub Service. The connection between RabbitMQ and ExperimentHub Service is configured during startup via Docker Compose, ensuring a simple setup process. Data sent to RabbitMQ must be JSON serializable as it transfers.json files from one microservice to another.

An MLflow repository is **required** to execute the REST API commands of the ExperimentHub. This microservice contains the Experiment names (queue names of the RabbitMQ service), under which several run IDs are accessible. Individual models are saved in a standardized format with a run ID under the corresponding queue name. As such, any instance of ExperimentHub will be able to reload models.

When data is sent to ExperimentHub, REST API commands can be used to access specific processes. For a new model, the process begins by sending a training dataset to the RabbitMQ queue and then submitting a training configuration. If the training completes without exceptions, the model is saved to the experiment in MLflow for future use and provides information about the training process on the frontend to check whether the model is viable.

Additional commands enhance functionality, such as saving to and loading from MLflow, or retraining to replace the current model in the ExperimentHub. Automated retraining can also be configured during the training phase, enabling continuous model updates based on new data or predefined conditions.

The config.yaml file, in particular, plays a central role by coordinating multiple experiments. The config.yaml is used to compose a configuration file with relevant information to send to the ExperimentHub, which upon retrieval builds the models. It dynamically pulls data from different RabbitMQ queues (keys in the config.yaml) for each experiment, allowing for the execution of several tasks with different configurations. This setup enables users to train and test various models using diverse datasets, streamlining the management of complex workflows and facilitating scalable experimentation, without redeploying the Docker image. The documentation of the Python package provides more details on the inner working of the system.

It is important to note that models retain their inherent properties as this system provides an infrastructure for data and use rather than reimplementing these methods.

2.2. Software functionalities

While MLflow is designed to be framework-agnostic, its reliance on flavors introduces some framework-specific dependencies. This can result in a difference in the approach taken to integrate models from certain frameworks. Of course, the MLflow contains pyfunc general approach to models, however it lacks convenience functions, and this limitation creates challenges when integrating models. Therefore, the framework limits the use of models to custom sklearn models, which are derived from *BaseEstimator* class.

The Experiment class serves as the core of the ExperimentHub system, managing the lifecycle of machine learning experiments, including tasks such as training, saving, loading, and predicting. This class holds essential metadata about each experiment, *e.g.,* its configuration, plots, and the corresponding RabbitMQ queue name. Fig. 3 presents the interaction between the experiments and the models and general methods that need to be implemented.

Derived classes extend the functionality of the Experiment class by providing specialized behaviors tailored to specific types of experiments or workflows. These subclasses implement custom methods for model training processes, prediction, or experiment management tasks. The *setup* function handles the data formatting, *create_model* uses a predefined model training process, along with *predict* that infers new results from the model. Both functions require a string-like input that is most often a read JSON from the RabbitMQ. These are also included as keywords in the configuration files.

The framework includes default implementations of the Experiment class to address various use cases. Fault Detection is designed for anomaly detection, aiming to identify errors in production, such as incorrect packaging selections or failed resource placements. As a multivariate approach, it can group production inputs and provide insights into which sensors might be contributing to the problem. It supports multiple sensors and does not require labeled output, making it versatile for detecting anomalies in complex systems.

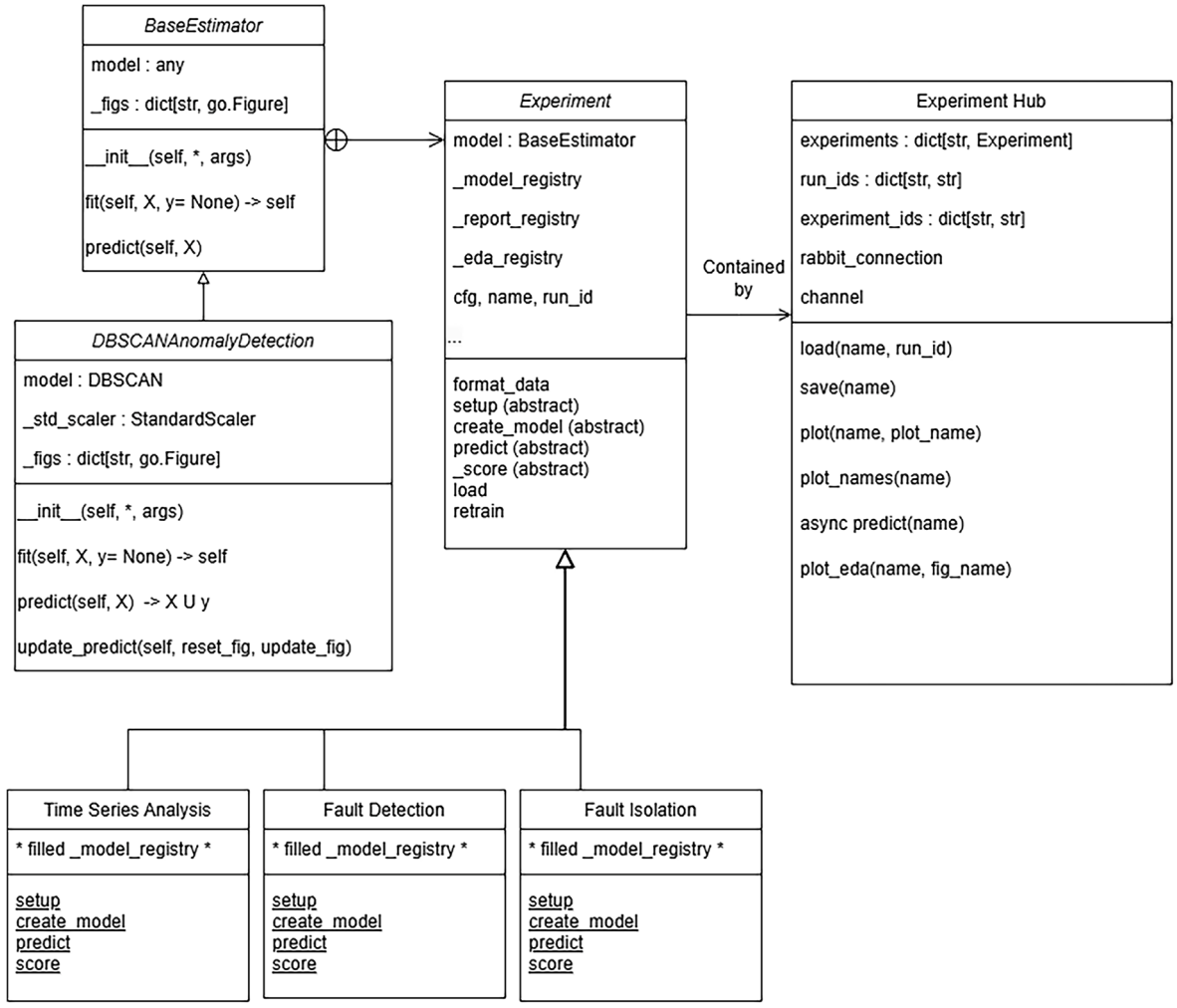

Fig. 3. Class interactions. Empty-headed arrows show that an object implements a target abstract class.

The Fault Isolation class serves as a broader wrapper for classification problems where labeled data is available. It is particularly useful for root cause analysis, identifying sensor contributions to anomaly detection, and analyzing machine states. This facilitates a deeper understanding of the system and supports targeted troubleshooting and optimization.

Time Series Analysis class is designed for monitoring sensor data over time, offering insights into the behavior of individual sensors across their lifecycle. It is ideal for applications like predictive maintenance and real-time monitoring, enabling proactive system management and minimizing unexpected downtime.

If the data is not tabular or nominal, the process mining may help analyze otherwise unusable data. Process Mining class is a straightforward implementation of process mining techniques, which can also find causal connections between sequences and states. These approaches can be applied to analyze operator behavior and machine logs, offering valuable insights into system workflows and identifying potential areas for improvement.

During the definition of the class, only models with similar data input requirements can be used. Some functions may be reconfigurable, but it is difficult to debug them properly. Table 2 contains the list of models implemented in the framework.

Table 2
Models included in the Framework. The columns focus on the implemented models (from BaseEstimator class), experiment types, and associated packages.

Model Name	Experiment Type	Package
Decision Tree	Fault Isolation	scikit-learn [21]
Random Forest	Fault Isolation	scikit-learn [21]
Naive Bayes	Fault Isolation	scikit-learn [21]
Hidden Markov Models	Fault Isolation	hmmlearn [22]
Bayesian Network	Fault Isolation	pgmpy [23]
Markov Chain	Fault Isolation	statsmodels [24]
PCA	Fault Detection	pca [25]
DBSCAN	Fault Detection	scikit-learn [21]
Elliptic Envelope	Fault Detection	scikit-learn [21]
Isolation Forest	Fault Detection	scikit-learn [21]
ARIMA/SARIMA	Time Series Analysis	statsmodels [21]
LSTM	Time Series Analysis	TensorFlow [26]
Autoencoder	Time Series Analysis	TensorFlow [26]
Prophet	Time Series Analysis	Prophet [27]
SSA	Time Series Analysis	pySSA [28]
Heuristics Miner	Process Mining	pm4py [8]
Top-K Rules	Process Mining	SPMF [29]
Apriori Association Rules	Process Mining	SPMF [29]
CMSPAM	Process Mining	SPMF [29]

3. Illustrative examples

In order for the application to work properly, one needs to set up the Docker environment. Listing 1 provides commands to pull the application from DockerHub:

```
docker pull rabbitmq:3.12.14-management-alpine
docker pull ipkovichadam/observml:mlflow
docker pull ipkovichadam/observml:backend
docker pull ipkovichadam/observml:frontend
docker compose up
```

Listing 1. Docker Compose up without installing dependencies.

```
poetry install
mlflow run
docker run --publish 5672:5672 rabbitmq:3.12.14-management-alpine
python ExperimentHubAPI.py
streamlit run streamlit_frontend.py
```

Listing 2. Local use of the application (*e.g.,* for development). Use each command on a different terminal.

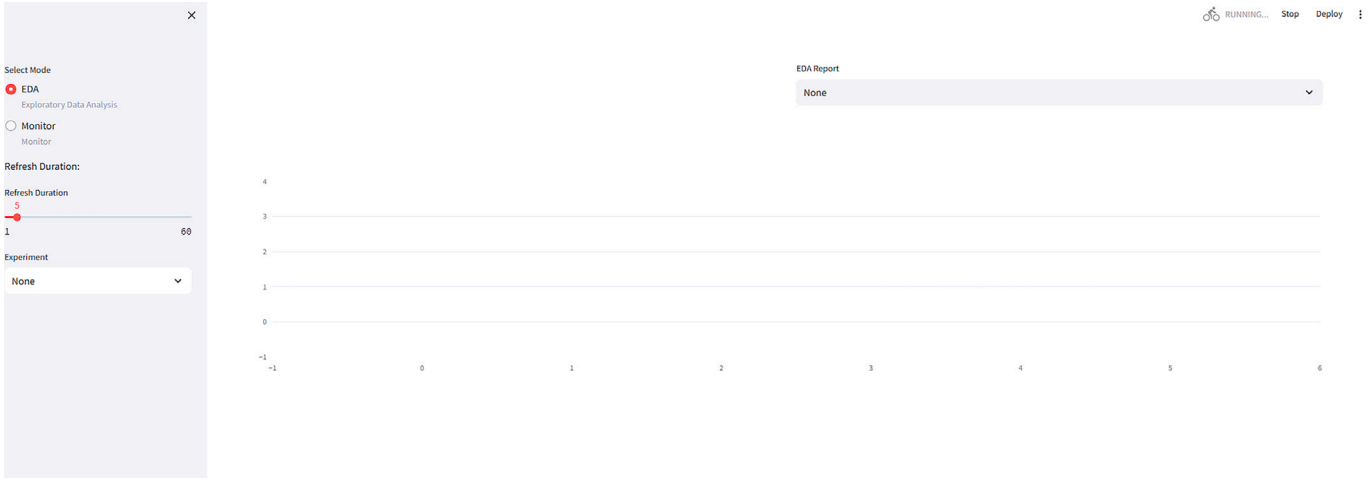

Fig. 4. User Interface for an empty ExperimentHub.

```yaml
## This is the configuration file FaultDetectionExperiment,
##specifying a PCA instance.
load_object : ## this is a must
##- should not remove -
##unique dynamic import of class
##(dynamic package download is not yet supported)
  module: framework.FaultDetection
  name: FaultDetectionExperiment
setup: # from this key, it is arbitrary
##-> the key name is a function
##in the experiment, and the value is the
## parameter of the function.
    # MUST NOT HAVE A 'data' KEYWORD!
    datetime_column : "ds"
    datetime_format : "ms"
#eda: # Commenting out disables the feature
create_model :
    model : "pca" #"ee" #"dbscan" #"pca"
    # If model is none, then compare model is used.
    params:
```

Listing 3. Configuration file for Decision Tree-based fault isolation.

In case of customizing and improving the framework, it might be in one's best interest to set up local services without having to create a Docker image. Listing 2 presents a local approach that helps you develop new experiments and models.

Fig. 4 presents the default user interface for the Experiment Hub. On the sidebar, two modes are possible; one for presenting the exploratory data analysis, and one for monitoring (training and prediction results). A slider is available to set the refresh duration, which can reduce the

Fig. 5. Variance of the Principal components from the pump dataset.

strain on the front-end application. The Experiment tab expands upon new models, currently there is no model to select. On the right-hand side the relevant EDA figures can be selected, which are also updated if new models are trained to show the figures of the selected experiment. This is only an example UI, which provides code on how to communicate with the backend API and can be replaced by custom UIs such as Grafana or Prometheus.

To train models, we are required to post a train request, and push a dataset to the rabbit. Some example codes are provided.

3.1. Fault detection - PCA

The pump dataset contains 50 sensors, without any labeled one to use fault isolation Experiment on [30]. Therefore, we use the fault detection experiment to find anomalies that could present possible malfunction. In this case, principal component analysis (PCA) is applied with Hotelling's T-squared and SPE/DmodX tests to find the outliers [25]. Listing 3 presents the configuration file sent to the Experiment Hub service. In this case, we did not enable EDA function, which was done by commenting out the keyword in the configuration file.

The PCA model provides a performance figure showing the variances for the PCs. This is presented in Fig. 5. The first two PCs account for 60% of the variance.

We also provide a dynamically updating figure called "outliers" to adjust for incoming data. Fig. 6 presents the anomalies with time on the x-axis, and PC on the y-axis. The blue dots are considered outliers by the Hotelling T-squared tests, while the purple dots indicate if a score is an outlier for both tests.

Fig. 6. Outliers in the Principal components from the pump dataset.

4. Impact

The ObServML package opens up new possibilities in the field of real-time monitoring, fault detection, and predictive maintenance in compact and modular industrial environments. Its adherence to MLOps principles, modular microservice-based architecture, and flexible model integration allow it to have a significant impact across research, practice, and industry. For further information on the capabilities of the tool, see the full documentation here.

ObServML enables research into adaptive monitoring strategies, data-driven fault isolation without labeled data, and automated retraining mechanisms in environments with limited computational resources. The support for both traditional models (e.g., PCA, ARIMA) and advanced methods (e.g., LSTM, Autoencoders) in a plug-and-play configuration invites comparative studies on model performance across industrial scenarios. Furthermore, its support for experimental process mining methods introduces new possibilities for studying operator behavior and machine log sequences, areas typically under-explored due to a lack of integrative tooling.

The software significantly reduces the overhead for researchers conducting fault detection and isolation (FDI) by automating the model training, deployment, and retraining pipeline within a reproducible framework. Compared to traditional pipelines that require manual scripting and orchestration, ObServML enables faster experimentation and deployment through a unified API and configuration management via Hydra. Prior works like BibMon [3] offer similar analytics capabilities, but lack the deployment-ready MLOps integration that ObServML provides. By decoupling model design from data engineering tasks, the framework streamlines the pursuit of predictive maintenance research as emphasized in Zonta et al. [1].

The package was specifically designed with small- and medium-sized industrial enterprises in mind, focusing on real-time process monitoring and predictive maintenance. It is already being tested and developed in industrial settings, ensuring that its functionalities address practical operational needs. This will be presented by the authors future separate work. Although widespread adoption is still in progress, the software is openly available via GitHub and DockerHub, and early use within academic-industrial collaborations confirms its usability in both research and production environments.

5. Conclusions

This paper presented a Python package designed for compact and modular monitoring applications, addressing key challenges in fault detection and isolation (FDI). By integrating MLOps principles, the framework ensures reproducibility, scalability, and adaptability, making it a versatile solution for industrial environments. Its modular design simplifies the customization of models to suit diverse datasets and operational needs, while its operator-friendly interface streamlines real-time monitoring and deployment processes.

The integration of Docker, RabbitMQ, and MLflow reinforces the application through micro-service architecture, enabling seamless workflow management and efficient communication between components.

Future work includes expanding model options, enhancing automation features, and integrating advanced visualization tools to further optimize usability. This framework sets a strong foundation for scalable, modular, and user-focused FDI solutions, bridging the gap between technical complexity and operational simplicity in industrial systems.

CRediT authorship contribution statement

Ádám Ipkovich: Writing – original draft, Visualization, Software, Methodology, Formal analysis, Data curation, Conceptualization. **János Abonyi:** Supervision, Methodology, Conceptualization. **Alex Kummer:** Writing – review & editing, Software, Funding acquisition, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the the PIACI-KFI research development program of the National Research, Development and Innovation Office (2020-1.1.2-PIACI-KFI-2020-00062). The research presented in the article was carried out within the framework of the Széchenyi Plan Plus program with the support of the RRF-2.3.1-21-2022-00008 project. This project was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences (B0/00439/25/6).

References

[1] Zonta T, da Costa CA, da Rosa Righi R, de Lima MJ, da Trindade ES, Li GP. Predictive maintenance in the industry 4.0: a systematic literature review. Comput Ind Eng 2020;150:106889. https://doi.org/10.1016/j.cie.2020.106889
[2] Frisk E, Krysander M, Jung D. A toolbox for analysis and design of model based diagnosis systems for large scale models. IFAC-Pap 20th IFAC World Congress.2017;50(1):3287–93. https://doi.org/10.1016/j.ifacol.2017.08.504
[3] Melo A, Lemos TS, Soares RM, Spina D, Clavijo N, de O. Campos LF, Câmara MM, Feital T, Anzai TK, Thompson PH, Diehl FC, Pinto JC. BibMon: an open source Python package for process monitoring, soft sensing, and fault diagnosis. Digit Chem Eng 2024;13:100182. https://doi.org/10.1016/j.dche.2024.100182
[4] Bonney MS, de Angelis M, Dal Borgo M, Andrade L, Beregi S, Jamia N, Wagg DJ. Development of a digital twin operational platform using Python flask. Data-Centric Eng 2022;3:e1. https://doi.org/10.1017/dce.2022.1
[5] Schummer F, Hyba M. An approach for system analysis with model-based systems engineering and graph data engineering. Data Cent Eng 2022;3:e33. https://doi.org/10.1017/dce.2022.33
[6] Li W, Chakraborty M, Cartaya CQ, Köhler J, Faber J, Meier M-A, Rümpker G, Srivastava N. Saipy: a Python package for single-station earthquake monitoring using deep learning. Comput Geosci 2024;192:105686. https://doi.org/10.1016/j.cageo.2024.105686
[7] NASA. ProgPy: a Python prognostics library; 2023. https://github.com/nasa/progpy.
[8] Berti A, van Zelst SJ, van der Aalst WMP. Process mining for python (pm4py): Bridging the gap between process- and data science, CoRR abs/1905.06169. arXiv:1905.06169 [arXiv preprint]. 2019.
[9] Zhao Y, Nasrullah Z, Li Z. PyOD: a Python toolbox for scalable outlier detection. J Mach Learn Res 2019;20(96):1–7.
[10] Chen A, Chow A, Davidson A, DCunha A, Ghodsi A, Hong SA, Konwinski A, Mewald C, Murching S, Nykodym T, Ogilvie P, Parkhe M, Singh A, Xie F, Zaharia M, Zang R, Zheng J, Zumar C. Developments in mlflow: a system to accelerate the machine learning lifecycle. In: Proceedings of the fourth international workshop on data management for End-to-End machine learning, DEEM '20. New York, NY, USA: Association for Computing Machinery; 2020. https://doi.org/10.1145/3399579.3399867
[11] Fledge: open source industrial edge platform. 2023. https://lfedge.org/projects/fledge/.
[12] Dataiku: everyday AI, extraordinary people. 2023. https://www.dataiku.com.
[13] Flyte: scalable and reproducible ML workflows. 2023. https://flyte.org.
[14] Kubeflow: machine learning toolkit for kubernetes. 2023. https://www.kubeflow.org.
[15] Yadan O. Hydra - a framework for elegantly configuring complex applications. Github; 2019. https://github.com/facebookresearch/hydra.
[16] Büchi G, Cugno M, Castagnoli R. Smart factory performance and industry 4.0. Technol Forecast Soc Change 2020;150:119790. https://doi.org/10.1016/j.techfore.2019.119790
[17] Gladysz B, anh Tran T, Romero D, van Erp T, Abonyi J, Ruppert T. Current development on the operator 4.0 and transition towards the operator 5.0: a systematic literature review in light of industry 5.0. J Manuf Syst 2023;70:160–85. https://doi.org/10.1016/j.jmsy.2023.07.008. https://www.sciencedirect.com/science/article/pii/S0278612523001401.
[18] Soori M, Arezoo B, Dastres R. Digital twin for smart manufacturing, a review. Sustain Manuf Serv Econ 2023;2:100017. https://doi.org/10.1016/j.smse.2023.100017
[19] Bántay L, Sas N, Dörgő G, Abonyi J. Sequence compression and alignment-based process alarm prediction. Ind Eng Chem Res 2023;62(27):10577–86. https://doi.org/10.1021/acs.iecr.3c00935
[20] Nunes P, Santos J, Rocha E. Challenges in predictive maintenance – a review. CIRP J Manuf Sci Technol 2023;40:53–67. https://doi.org/10.1016/j.cirpj.2022.11.004
[21] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E. Scikit-learn: machine learning in Python. J Mach Learn Res 2011;12:2825–30.
[22] Cournapeau D, Pedregosa F, Varoquaux G, Lebedev S, Lee A, Danielson M. Hmmlearn: hidden markov models in Python; 2024. https://pypi.org/project/hmmlearn/ [accessed 8 December 2024].

[23] Ankan A, Panda A. Probabilistic graphical models using Python. In: Proceedings of the Python in science conference. SciPy, SciPy; 2015. https://doi.org/10.25080/majora-7b98e3ed-001.
[24] Seabold S, Perktold J. Statsmodels: econometric and statistical modeling with Python. In: 9th Python in science conference; 2010.
[25] Taskesen E. PCA: a Python package for principal component analysis. Oct 2020. https://erdogant.github.io/pca.
[26] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X. TensorFlow: large-scale machine learning on heterogeneous systems, software available from tensorflow.org 2015. https://www.tensorflow.org/.
[27] Taylor SJ, Letham B. Forecasting at scale. Am Stat 2018;72(1):37–45. https://doi.org/10.7287/peerj.preprints.3190v2
[28] Kishore D, Chandrasekaran S. PySSA: singular spectrum analysis in Python; 2023. https://pypi.org/project/pyssa/.
[29] Fournier-Viger P, Lin C, Gomariz A, Gueniche T, Soltani A, Deng Z, Lam H. The spmf open-source data mining library version 2. In: Proc. 19th european conference on principles of data mining and knowledge discovery (PKDD 2016) part III, springer LNCS 9853. Springer; 2016. p. 36–40.
[30] Phantawee N. Pump sensor data; 2019. https://www.kaggle.com/datasets/nphantawee/pump-sensor-data [accessed 25 January 2025].