On the Challenges of Implementing MLOps for Stream Learning Algorithms

Miguel G. Rodrigues, Eduardo K. Viegas, Fabrício Enembreck, Altair O. Santin, Juliano S. Langaro, and Adilson G. Filho

Abstract Machine Learning Operations (MLOps) are essential for the efficient management of the Machine Learning (ML) lifecycle, ensuring scalability, adaptability, and operational performance across deployments. However, there is limited implementation of MLOps architectures specifically designed for stream learning scenarios, which involve continuous data flows and frequent model updates. This paper provides a comprehensive overview of existing MLOps architectures with a focus on their support for streaming data processing, model versioning, and real-time updates. Our analysis identifies critical gaps related to scalability, adaptability, and efficient handling of frequent updates in dynamic stream learning environments. We highlight the open gaps to pave the way for architectures that provide real-time model versioning and update mechanisms to improve stream learning performance across diverse application domains.

Miguel G. Rodrigues
Pontifícia Universidade Católica do Paraná, PUC-PR e-mail: miguel.rodrigues@ppgia.pucpr.br

Eduardo K. Viegas
Pontifícia Universidade Católica do Paraná, PUC-PR e-mail: eduardo.viegas@ppgia.pucpr.br

Fabricio Enembreck
Pontifícia Universidade Católica do Paraná, PUC-PR e-mail: fabricio.enembreck@ppgia.pucpr.br

Altair O. Santin
Pontifícia Universidade Católica do Paraná, PUC-PR e-mail: santin@ppgia.pucpr.br

Juliano S. Langaro
Pontifícia Universidade Católica do Paraná, PUC-PR e-mail: juliano.langaro@ppgia.pucpr.br

Adilson G. Filho
Pontifícia Universidade Católica do Paraná, PUC-PR e-mail: adilson.filho@ppgia.pucpr.br

1 Introduction

Machine Learning (ML) focuses on developing computational models capable of learning and making predictions or decisions from historical data [32]. ML methods are primarily differentiated by their training paradigms. On the one hand, batch learning approaches rely on static datasets, processing the entire volume of information at once to obtain the models [35]. On the other hand, stream learning algorithms rely on incremental methods to continuously update models by incorporating new data as it arrives in a constant stream. This adaptability makes stream learning particularly well-suited for dynamic and real-time scenarios [1].

The effective application of ML models extends beyond training, encompassing an iterative and interdisciplinary lifecycle that includes data collection, preprocessing, model training, validation, deployment, performance monitoring, and periodic updates to maintain predictive accuracy [9]. To address the complexities of managing this lifecycle, Machine Learning Operations (MLOps) has emerged as a collaborative framework that combines DevOps principles with ML development. It streamlines workflows through automation and orchestration, facilitates model versioning, promotes code standardization and reuse, and efficiently scales systems [26].

Over the past years, several frameworks have been proposed to support the development of ML pipelines aligned with MLOps practices. While these architectures effectively address the demands of traditional batch-oriented ML scenarios, where data patterns remain static, they often fall short of meeting the requirements of stream learning environments. This limitation arises because conventional MLOps practices are not designed to accommodate incremental model updates or the real-time deployment of updated models within very short timeframes [42].

Stream learning architectures impose unique demands, particularly during the inference phase, which must handle high volumes of requests through parallel processing techniques. While this requirement aligns with traditional MLOps applications, stream learning introduces the added complexity of frequently replacing stored models in memory with updated versions. This challenge becomes significant in scenarios with numerous endpoints (applications encapsulating the model) or when model replacement occurs at a high frequency [38]. Notwithstanding. the model versioning task is not easily achieved in stream learning environments, given the requirement for real-time loading and serialization to ensure the timely availability of updated models on endpoints. This situation requires that the versioning module can efficiently manage the substantial volume of versions generated by frequent updates [44].

In practice, conducting model updates in stream learning is the most challenging task in MLOps [39]. Unlike traditional MLOps, where updates are performed using batch or mini-batch approaches, stream learning necessitates frequent, ideally incremental, updates. However, implementing incremental updates in production environments presents significant obstacles, as each update requires endpoints to reload the updated model into memory. This process is resource-intensive and can often lead to interruptions to the inference service [20]. Given these complexities, real-time model update solutions tailored for stream learning within the MLOps

framework are often overlooked. Traditional approaches, such as those relying on periodic batch updates or concept drift detection, are often employed but may compromise the system's accuracy. This is because, by the time an updated model is deployed, it may already be misaligned with current data distributions, undermining its effectiveness due to the inherent delay in generating and deploying the new model [1].

In light of this, this paper addresses these challenges by reviewing the current architectures available in both the commercial use and the research literature for implementing stream learning in alignment with MLOps practices. Beyond detailing the characteristics of these architectures, we examine their suitability for stream learning scenarios, emphasizing their limitations and gaps. The goal is to offer a comprehensive analysis of the difficulties and challenges in deploying systems that demand frequent and near-real-time updates, thereby contributing to the development of solutions that fulfill the requirements of stream learning environments while maintaining scalability, efficiency, and predictive accuracy.

2 Preliminaries

2.1 MLOps

Machine Learning Operations (MLOps) is an approach designed to streamline the development, deployment, monitoring, and management of ML systems in production environments. It represents an extension of DevOps (Development and Operations), adapted specifically to address the requirements of the ML model lifecycle. The primary goal of MLOps is to ensure the successful deployment of ML models into production while maintaining their predictive performance and reliability over time [26].

To achieve such a goal, MLOps combines a set of practices and tools specifically designed to address the unique lifecycle needs of ML models, ensuring that these systems remain robust, reliable, and efficient throughout their operational life. Among these tasks, continuous monitoring allows for the proactive detection of performance issues or concept drift [33]. Similarly, the model versioning enhances traceability by allowing teams to monitor changes over time and providing the ability to revert to previous versions when necessary. This approach not only boosts operational efficiency but also strengthens security and ensures compliance with regulatory standards. [23].

Deploying models into production is one of the most critical steps in MLOps, as it involves making ML models available for user inference. Deployment can take various forms, such as through web services (containerized applications), local applications hosted on on-premise servers, or even on IoT devices via edge computing [21, 16]. However, successful deployment alone is not sufficient—continuous monitoring is essential to ensure sustained performance and the early detection of

any deviations in data patterns or model behavior. Consequently, MLOps emphasizes periodic model updates and retraining based on insights and data obtained from monitoring models in production [33].

2.2 Stream Learning

In a stream learning environment, data is made available in a continuous and potentially infinite stream, transmitted at very high speeds. In such scenarios, data is typically accessed only once or retained for a brief period. Consequently, each step of the stream learning process (also known as online learning or ML stream) must be quick and computationally efficient, ensuring minimal resource consumption [10].

One of the primary challenges in stream learning is the continuous change of concepts within the data. Unlike traditional ML scenarios, where model creation and updates rely on static historical data, stream learning operates in a highly dynamic environment. Data patterns shift in response to environmental changes, leading to alterations in data distributions—a phenomenon known as concept drift. Such changes can significantly affect the predictive performance of deployed models. Therefore, frequent model updates are crucial in a stream learning context, with incremental updates being the ideal approach to ensure adaptability while minimizing computational overhead [18].

3 MLOps Architectures

This section examines the main MLOps architectures currently available, covering both commercial solutions and open-source approaches discussed in the scientific literature. In particular, we focus on the challenges such as supporting incremental updates, ensuring efficient versioning in high model turnover environments, and facilitating real-time integration.

3.1 Commercial Solutions

Commercial MLOps solutions are platforms developed by leading technology companies such as Amazon, Google, Microsoft, and Databricks, designed to streamline the operationalization of ML models. The following provides an overview of these platforms, along with a concise analysis of their capabilities and limitations in supporting stream learning environments.

Amazon SageMaker is a managed platform designed to streamline the development, training, deployment, and monitoring of ML models, offering tools for automating pipelines and seamless integration with other AWS services. Although

SageMaker excels in traditional ML scenarios, it faces limitations in stream learning environments, particularly due to its lack of native support for continuous incremental updates and efficient model management under high-frequency update requirements. While adapting SageMaker for real-time streams is feasible through the integration of other AWS continuous integration services, this approach often demands significant additional effort, complicating the implementation of pipelines in stream learning environments.

Azure Machine Learning, Microsoft's platform for developing, training, deploying, and monitoring ML models, provides advanced features such as managed pipelines, traceable experimentation, and automated ML (AutoML), along with seamless integration with other Azure services. While robust for traditional ML scenarios, the platform exhibits notable limitations in stream learning environments. It lacks native support for continuous incremental model updates, relying instead on periodic retraining pipelines that require manual configuration, which is inefficient for high-frequency update demands. Additionally, its management and versioning capabilities for multiple models in continuous data streams are not optimized, posing challenges for effective application in stream learning scenarios.

Databricks, a commercial platform for managing the ML model lifecycle, provides robust tools for experimentation, traceability, versioning, and pipeline automation. It excels in traditional ML scenarios, particularly for executing large-scale pipelines and integrating seamlessly with frameworks like TensorFlow [19], PyTorch [36] and Scikit-learn [34]. However, in stream learning environments, the platform reveals certain limitations. While it supports periodic model updates, it lacks native functionality for continuous incremental updates, which are critical in dynamic data scenarios. Additionally, its management of multiple models generated in these environments is not fully optimized, posing challenges for efficient versioning and organization in high-turnover scenarios.

Vertex AI, offered by Google Cloud Platform, is a comprehensive solution for developing, training, and deploying ML models, featuring tools such as AutoML and seamless integration with other Google Cloud services. Its capabilities include managing large-scale data and automating ML workflows, making it a strong contender in traditional ML scenarios. However, Vertex AI falls short in stream learning contexts, as it does not natively support continuous incremental model updates. Although real-time data processing can be achieved through integration with other tools, this requires additional setup and configuration. Furthermore, the platform's model management is not optimized for environments with high update frequencies, which can hinder operational efficiency in dynamic stream learning applications.

3.2 Open Source Architectures

Open source MLOps solutions such as MLFlow [30], Airflow [40], Kubeflow [41], and TFX [12] have gained widespread adoption due to their flexibility, adaptability, and strong community support. These platforms provide a broad range of function-

alities tailored to the needs of the ML lifecycle, including experimentation tracking, pipeline orchestration, and deployment management. However, their effectiveness in stream learning scenarios is limited, as they lack native support for incremental updates, efficient versioning for high-frequency model updates, and optimized real-time integration. Below, we summarize their key features and limitations in addressing the demands of stream learning environments.

Apache Airflow is a widely adopted open-source workflow orchestration tool, particularly effective for managing data pipelines and ML workflows. Its Directed Acyclic Graph (DAG)-based architecture facilitates the coordination of tasks such as data collection, transformation, and analysis, making it well-suited for traditional ML processes. However, in stream learning scenarios, Airflow encounters significant limitations. It does not natively support continuous incremental updates or the rapid adaptation required for real-time data streams. Additionally, the static nature of its DAGs complicates automatic adjustments to frequent changes, reducing its efficiency in dynamic environments where high adaptability is crucial.

Kubeflow is an open-source platform designed for orchestrating and managing ML workflows within Kubernetes environments, featuring capabilities such as pipeline automation, monitoring, and model versioning. Although its model management tools are robust, they are not optimized for the intensive versioning demands arising from real-time updates. Furthermore, the platform lacks native support for continuous update processes, which restricts its effectiveness in scenarios that require seamless integration of continuous learning workflows.

MLflow is an open-source platform designed to streamline the ML lifecycle by offering tools for the development, tracking, management, and deployment of ML models. Its focus on experiment traceability, reproducibility, and continuous integration makes it a popular choice among ML practitioners. Key features include model versioning, which facilitates the organized storage and recording of hyperparameters, metrics, and execution artifacts. However, despite its versatility, MLflow lacks native support for the specific demands of online or incremental model updates, a critical requirement in stream learning scenarios where models must adapt continuously to evolving data streams.

3.3 Scientific Literature

A comprehensive literature review was conducted to identify MLOps architectures applicable to stream learning scenarios. The review focused on understanding how these architectures address the unique challenges of stream learning environments, including the need for continuous model updates, efficient versioning, practical applicability, and the specific components and tools that constitute them.

Several architectures identified in our research incorporate specialized stream learning tools or frameworks. One example is StreamDM, introduced by Albert Bifet *et al.*[13]. It is an open-source library built on top of Spark Streaming[6] and developed in Scala at Huawei Noah's Ark Lab. Designed for both academic and

practical applications, StreamDM leverages the Hadoop open-source ecosystem [2] to enable distributed data processing. Despite its advantages, such as distributed computation, StreamDM processes data using mini-batches, which can introduce latency of several seconds, thereby limiting its real-world applicability.

Another library in the context of stream learning is SOLMA, introduced by W. Jamil *et al.*[22]. SOLMA is built upon the Apache Flink ecosystem[7], leveraging distributed data processing resources to ensure scalability and fault tolerance. It supports real-time inference and parallelism during model updates, which enhances its operational capabilities. However, SOLMA has significant limitations: it does not support incremental model updates, has high configuration complexity, and offers limited algorithm availability. These factors make its practical application more challenging, particularly in dynamic or resource-constrained environments.

Following this concept, Donna Xu *et al.*[43] propose an architecture designed to provide real-time analysis services. In their approach, models are initially trained in batches but can be continuously updated through incoming data streams using the concept of mini-batches. The architecture leverages a microservices-based design, which provides scalability and system flexibility. Additionally, it enables the integration of powerful frameworks such as Apache Spark [5] and the MLlib library [8], among others. This design emphasizes modularity and adaptability, allowing the system to efficiently process streaming data while maintaining real-time model updates.

Several studies also present ML management platforms that offer resources for deploying models for real-time inference, though with model updates restricted to batch learning. One such example is the Looper tool, introduced by Igor L. Markov *et al.* [28]. Looper is a comprehensive platform for managing the ML model lifecycle, designed to support all stages of the ML pipeline for decision-making in software products. Looper provides automation capabilities, allowing repetitive tasks to be streamlined, which saves time and resources and enhances the user experience. While Looper enables real-time inference, it relies solely on batch learning for model updates, limiting its ability to support continuous or incremental model adaptation in real-time streaming scenarios.

Architectures dedicated to specific use cases for implementing stream learning were also identified, such as Lambda Learner, developed by Rohan Ramanath *et al.* [37]. It is specifically designed to predict the click-through rate on advertisements for a well-known business social network. It combines batch processing (offline learning) to build a robust initial model with updates using mini-batches for real-time adaptation. Lambda Learner employs a lambda architecture, dividing the learning process into two components: a fast component that updates the model with new data in real-time, and a slow component that periodically retrains the entire model to ensure long-term performance and quality. However, the system is tailored to this specific use case and does not provide a generalized platform suitable for broader or alternative stream learning scenarios.

Another architecture designed only to a specific application is the Scalable Real-time Fraud Finder (SCARFF), introduced by Fabrizio Carcillo *et al.* [14]. SCARFF is an open-source platform designed for processing and analyzing credit card trans-

action data, with the primary goal of delivering reliable fraud alerts in near real-time. The system integrates ML with Big Data tools, enabling efficient analysis and scalability. SCARFF employs sliding window techniques (mini-batches) for model updates, allowing it to adapt to new patterns in transaction data. While this method enhances its scalability by supporting horizontal scaling through distributed processing, SCARFF's use is limited by its design. The architecture is highly specialized for fraud detection in credit card transactions, and its structure lacks the flexibility for other use cases. Consequently, its application remains exclusive and highly restrictive due to these constraints.

The Big Data Engine architecture, presented by Mikołaj Komisarek *et al.*[24], integrates several backend tools, including Apache Spark, Apache Kafka[4], Elasticsearch [15], and HDFS [3]. Designed specifically for detecting network activities and patterns indicative of malicious or suspicious behaviors, such as cyberattacks, the Big Data Engine supports both batch and real-time processing of network data. This enables the immediate detection of anomalies. However, model updates rely on the use of mini-batches, which may limit real-time adaptability in rapidly changing environments.

Another module-based framework is STREAMER, introduced by Sandra Garcia Rodriguez *et al.* [17]. STREAMER is a stream processing system designed to run on any operating system while supporting the integration of ML algorithms written in various programming languages. Similar to other frameworks previously discussed, STREAMER relies on third-party tools for its underlying functionality. One of its main limitations is the lack of support for incremental model updates. This is because model updates can only be performed through mini-batches of data or by processing the entire previously stored dataset, which limits its ability to adapt efficiently to real-time changes in streaming data.

Tools that use the low-code approach (allowing the creation of applications with little or no programming) were also found, such as the ClowdFlows platform presented by Janez Kranjc *et al.* [25]. It runs as a Web application and supports data mining through a graphical interface that has as its main component for creating workflows a processing unit called a widget. Although it was initially designed to work only with batch processing, special daemons were implemented that execute workflows at a fixed time interval, which makes it possible to perform inferences using a data flow; however, training continues to be performed offline.

Kafka-ML, introduced by Cristian Martín *et al.* [29], is an open-source framework designed for managing ML pipelines using data streams. It supports popular ML frameworks such as TensorFlow and PyTorch, enabling compatibility with widely used ML models. Kafka-ML offers a user-friendly web interface and runs all its components as Docker containers, which enhances portability and allows seamless orchestration and monitoring via Kubernetes [27]. While Kafka-ML provides real-time inference capabilities using streaming data, its model training process does not support incremental updates. Instead, training is performed in batches, which may limit its adaptability in highly dynamic stream learning scenarios.

StreamMLOps, presented by Mariam Barry *et al.*[11], represents an online learning architecture geared toward the continuous deployment and learning of ML mod-

Table 1: Summary of architectures and their characteristics.

Source	Architecture	Generic Application	Incremental Update	Mini-Batch Update	Batch Update	Model Versioning
Open	TensorFlow Ext.	✓	×	✓	✓	✓
	Airflow	✓	×	×	×	×
	KubeFlow	✓	×	×	×	×
	MLFlow	✓	×	✓	✓	✓
Private	Databricks	✓	×	×	×	✓
	Amazon SageMaker	✓	×	×	×	✓
	Vertex AI	✓	×	×	×	✓
	Azure ML	✓	×	×	×	✓
Research	StreamDM	✓	×	✓	×	×
	SOLMA	✓	×	✓	×	×
	Donna Xu *et al.*	✓	×	✓	×	×
	Looper	×	×	×	✓	×
	MLPacker	✓	×	×	×	×
	Lambda Learner	×	×	✓	×	×
	SCARFF	×	×	×	×	✓
	Big Data Engine	×	×	×	×	✓
	Spring XD	✓	×	✓	×	×
	STREAMER	✓	×	✓	×	✓
	ClowdFlows	×	×	×	×	✓
	Kafka-ML	✓	×	✓	×	✓
	Clipper	✓	×	✓	×	×
	Striim	✓	×	✓	×	×
	StreamMLOps	×	✓	×	×	✓
	StreamAI	✓	✓	×	×	✓

els in real-time applications. StreamMLOps incorporates a variety of open-source tools, including Apache Kafka, Apache Flink, MLFlow, and River[31], to provide a flexible and extensible solution. While the architecture is broadly applicable across domains, its adaptation for specialized applications may require fine-tuning.

4 Open challenges

Table 1 presents a summary of the MLOps approaches discussed in this paper, highlighting their deployment characteristics in production environments, including support for continuous updates, generic applicability, and model versioning systems. These features are essential for assessing the ability of existing solutions to meet the specific requirements of stream learning, particularly in scenarios demanding high predictive performance and rapid adaptability. Additionally, the analysis identifies gaps that need to be addressed by future research and development efforts.

When evaluating the presented architectures, it is observed that only a subset of the analyzed solutions supports real-time data streams, which indicates a significant limitation in their scope of application. Support for continuous updates is similarly

limited, with few approaches implementing it, often without optimizations for scenarios with high-frequency changes. Furthermore, the limited domain applicability of these solutions restricts their adoption in broader contexts, such as heterogeneous and general-purpose environments. On the other hand, some approaches demonstrate greater flexibility and adaptability, making them more suitable for varied and dynamic scenarios.

Although versioning systems are present in several solutions, most were not designed to handle the high turnover and large volume of updates typical of stream learning environments. The lack of scalability in these systems underscores the need for technical advancements to enable efficient real-time version management without compromising operational flow performance. This challenge represents an opportunity for innovation, aiming to create more robust solutions that meet the requirements of modern systems.

5 Conclusion

Despite advances in MLOps, significant challenges remain to address the demands of stream learning systems. One key challenge is the integration of streaming architectures with scalable and efficient versioning systems, enabling the handling of high update frequencies and frequent model turnover in production. Furthermore, the limited support for continuous updates in dynamic data streams highlights a critical gap in current solutions, reducing flexibility and adaptability.

Future research should focus on developing more robust, scalable, and adaptable solutions capable of addressing the challenges inherent in stream learning environments.

Acknowledgment

This work was partially sponsored by the Brazilian National Council for Scientific and Technological Development (CNPq), grants nº 304990/2021-3, 407879/2023-4, and 302937/2023-4.

References

1. Agrahari, S., Singh, A.K.: Concept drift detection in data stream mining: A literature review. Journal of King Saud University - Computer and Information Sciences 34(10), 9523–9540 (Nov 2022)
2. Apache Software Foundation: Apache hadoop (2006), https://hadoop.apache.org/
3. Apache Software Foundation: Hadoop distributed file system (hdfs) (2006), https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html

4. Apache Software Foundation: Apache kafka (2011), https://kafka.apache.org/
5. Apache Software Foundation: Apache spark (2013), https://spark.apache.org/
6. Apache Software Foundation: Spark streaming (2013), https://spark.apache.org/docs/latest/streaming-programming-guide.html
7. Apache Software Foundation: Apache flink: Stateful computations over data streams (2014), https://flink.apache.org/
8. Apache Software Foundation: Mllib: Machine learning library (2014), https://spark.apache.org/mllib/
9. Ashmore, R., Calinescu, R., Paterson, C.: Assuring the machine learning lifecycle: Desiderata, methods, and challenges. ACM Computing Surveys 54(5), 1–39 (May 2021)
10. Barddal, J.P., Gomes, H.M., Enembreck, F., Pfahringer, B.: A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. Journal of Systems and Software 127, 278–294 (May 2017)
11. Barry, M., Montiel, J., Bifet, A., Wadkar, S., Manchev, N., Halford, M., Chiky, R., Jaouhari, S.E., Shakman, K.B., Fehaily, J.A., Le Deit, F., Tran, V.T., Guerizec, E.: Streammlops: Operationalizing online learning for big data streaming amp; real-time applications. In: 2023 IEEE 39th International Conference on Data Engineering (ICDE). p. 3508–3521. IEEE (Apr 2023)
12. Baylor, D., Breck, E., Cheng, H.T., Fiedel, N., Foo, C.Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., Koo, C.Y., Lew, L., Mewald, C., Modi, A.N., Polyzotis, N., Ramesh, S., Roy, S., Whang, S.E., Wicke, M., Wilkiewicz, J., Zhang, X., Zinkevich, M.: Tfx: A tensorflow-based production-scale machine learning platform. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Disc. and Data Mining. KDD '17, ACM (Aug.)
13. Bifet, A., Maniu, S., Qian, J., Tian, G., He, C., Fan, W.: Streamdm: Advanced data mining in spark streaming. In: 2015 IEEE International Conference on Data Mining Workshop (ICDMW). IEEE (Nov 2015)
14. Carcillo, F., Dal Pozzolo, A., Le Borgne, Y.A., Caelen, O., Mazzer, Y., Bontempi, G.: Scarff: A scalable framework for streaming credit card fraud detection with spark. Information Fusion 41, 182–194 (May 2018)
15. Elastic: Elasticsearch (2010), https://www.elastic.co/elasticsearch/
16. Espindola, A., Viegas, E.K., Traleski, A., Pellenz, M.E., Santin, A.O.: A deep autoencoder and rnn model for indoor localization with variable propagation loss. In: 2021 17th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob). IEEE (Oct 2021)
17. Garcia-Rodriguez, S., Alshaer, M., Gouy-Pailler, C.: Streamer: A powerful framework for continuous learning in data streams. In: Proceedings of the 29th ACM International Conference on Information amp; Knowledge Management. CIKM '20, ACM (Oct 2020)
18. Gonçalves, P.M., de Carvalho Santos, S.G., Barros, R.S., Vieira, D.C.: A comparative study on concept drift detectors. Expert Systems with Applications 41(18), 8144–8156 (Dec 2014)
19. Google Brain Team: Tensorflow: An open-source machine learning framework (2015), https://www.tensorflow.org/
20. Horchulhack, P., Viegas, E.K., Santin, A.O.: Detection of service provider hardware overcommitment in container orchestration environments. In: GLOBECOM 2022 - 2022 IEEE Global Communications Conference. p. 6354–6359. IEEE (Dec 2022)
21. Horchulhack, P., Viegas, E.K., Santin, A.O., Ramos, F.V., Tedeschi, P.: Detection of quality of service degradation on multi-tenant containerized services. Journal of Network and Computer Applications 224, 103839 (Apr 2024), http://dx.doi.org/10.1016/j.jnca.2024.103839
22. Jamil, W., Duong, N.C., Wang, W., Mansouri, C., Mohamad, S., Bouchachia, A.: Scalable online learning for flink: Solma library. In: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings. ECSA '18, vol. 58, p. 1–4. ACM (Sep 2018)
23. John, M.M., Olsson, H.H., Bosch, J., Gillblad, D.: Exploring trade-offs in mlops adoption. In: 2023 30th Asia-Pacific Software Engineering Conference (APSEC). p. 369–375. IEEE (Dec.)
24. Komisarek, M., Choraś, M., Kozik, R., Pawlicki, M.: Real-time stream processing tool for detecting suspicious network patterns using machine learning. In: Proceedings of the 15th International Conference on Availability, Reliability and Security. ARES 2020, ACM (Aug 2020)

25. Kranjc, J., Orač, R., Podpečan, V., Lavrač, N., Robnik-Šikonja, M.: Clowdflows: Online workflows for distributed big data mining. Future Generation Computer Systems 68, 38–58 (Mar 2017)
26. Kreuzberger, D., Kühl, N., Hirschl, S.: Machine learning operations (mlops): Overview, definition, and architecture. IEEE Access 11, 31866–31879 (2023)
27. Kubernetes Authors: Kubernetes: Open-source container orchestration platform (2014), https://kubernetes.io/
28. Markov, I.L., Wang, H., Kasturi, N.S., Singh, S., Garrard, M.R., Huang, Y., Yuen, S.W.C., Tran, S., Wang, Z., Glotov, I., Gupta, T., Chen, P., Huang, B., Xie, X., Belkin, M., Uryasev, S., Howie, S., Bakshy, E., Zhou, N.: Looper: An end-to-end ml platform for product decisions. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. KDD '22, vol. 28, p. 3513–3523. ACM (Aug 2022)
29. Martín, C., Langendoerfer, P., Zarrin, P.S., Díaz, M., Rubio, B.: Kafka-ml: Connecting the data stream with ml/ai frameworks. Future Generation Computer Systems 126, 15–33 (Jan 2022)
30. MLflow Project: An open source platform for the machine learning lifecycle (2018), https://mlflow.org/
31. Montiel, Jacob and Halford, Max and Mastelini, Saulo Martiello and Bolmier, Geoffrey and Sourty, Raphael and Vaysse, Robin and Zouitine, Adil and Gomes, Heitor Murilo and Read, Jesse and Abdessalem, Talel and others: River: machine learning for streaming data in python (2021), https://riverml.xyz/
32. Neto, E.C.P., Dadkhah, S., Sadeghi, S., Molyneaux, H., Ghorbani, A.A.: A review of machine learning (ml)-based iot security in healthcare: A dataset perspective. Computer Communications 213, 61–77 (Jan 2024)
33. Paleyes, A., Urma, R.G., Lawrence, N.D.: Challenges in deploying machine learning: A survey of case studies. ACM Computing Surveys 55(6), 1–29 (Dec 2022)
34. 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. Journal of Machine Learning Research 12, 2825–2830 (2011)
35. Pruneski, J.A., Williams, R.J., Nwachukwu, B.U., Ramkumar, P.N., Kiapour, A.M., Martin, R.K., Karlsson, J., Pareek, A.: The development and deployment of machine learning models. Knee Surgery, Sports Traumatology, Arthroscopy 30(12), 3917–3923 (Sep 2022)
36. PyTorch Foundation: Pytorch: An open-source machine learning library (2016), https://pytorch.org/
37. Ramanath, R., Salomatin, K., Gee, J.D., Talanine, K., Dalal, O., Polatkan, G., Smoot, S., Kumar, D.: Lambda learner: Fast incremental learning on data streams. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery amp; Data Mining. KDD '21, ACM (Aug 2021)
38. Ribeiro, R., Santin, A., Abreu, V., Marynowski, J., Viegas, E.: Providing security and privacy in smart house through mobile cloud computing. In: 2016 8th IEEE Latin-American Conference on Communications (LATINCOM). p. 1–6. IEEE (Nov 2016)
39. dos Santos, R.R., Viegas, E.K., Santin, A.O., Tedeschi, P.: Federated learning for reliable model updates in network-based intrusion detection. Computers and Security 133 (Oct 2023)
40. The Apache Software Foundation: Apache airflow (2014), https://airflow.apache.org/
41. The Kubeflow Authors: Kubeflow (2018), https://www.kubeflow.org/
42. Viegas, E., Santin, A., Neves, N., Bessani, A., Abreu, V.: A resilient stream learning intrusion detection mechanism for real-time analysis of network traffic. In: GLOBECOM 2017 - 2017 IEEE Global Communications Conference. p. 1–6. IEEE (Dec 2017)
43. Xu, D., Wu, D., Xu, X., Zhu, L., Bass, L.: Making real time data analytics available as a service. In: Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures. p. 73–82. CompArch '15, ACM (May 2015)
44. Zaharia, M.A., Chen, A., Davidson, A., Ghodsi, A., Hong, S.A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Xie, F., Zumar, C.: Accelerating the machine learning lifecycle with mlflow. IEEE Data Eng. Bull. 41, 39–45 (2018)