Software Engineering Best Practices for Using Machine Learning in the Oil and Gas Industry

Rodrigo da S. Cunha[1], Ismael H. F. Santos[5], Rodrigo A. Barreira[6], Edson S. Gomi[2], Eduardo A. Tannuri[3], Anna H. R. Costa[4]

Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, 380, 05508-010, São Paulo-SP, Brazil
[1]*rodrigo.cunha@usp.br, orcid 0000-0002-9374-9568,* [2]*gomi@usp.br, orcid 0000-0003-1267-9519,* [3]*eduat@usp.br, orcid 0000-0001-7040-413X,* [4]*anna.reali@usp.br, orcid 0000-0001-7309-4528,*

Petrobras, Rio de Janeiro, Brazil
[5]*ismaelh@petrobras.com.br,* [6]*barreira@petrobras.com.br*

Abstract.

The widespread adoption of Machine Learning (ML), due to the increased availability of data and the fast evolution of computing power and software techniques experienced in the last decade, has fundamentally changed our world. The implementation of ML models has become fast and cheap, allowing state-of-the-art discoveries to become widely accessible. That context enabled innovation in several industries and businesses, with problems hitherto untouched by science being addressed, therefore requiring a reduction in the gap between scientific research and its application in real-world issues. However, ML models have proven to be expensive to maintain and scale. New challenges emerge as ML models are deployed and monitored, driving a rising concern about best practices for building reliable ML systems. Despite the increasing popularity of this subject and the consolidation of specialized literature, applying the best practices is not a simple task. Depending on time or team experience and size, such practices can represent a technical overhead, making the enforcement of such practices an arduous task in many situations. This paper proposes a strategy for adopting software engineering best practices by committing to a set of principles from the beginning of an ML project. To illustrate our strategy, we will use mooring line failure detection in the floating production storage and offloading unit (FPSO) mooring system, providing an example of our strategy applied in the Oil and Gas industry.

Keywords: Software Engineering, Best Practices, Machine Learning, Mooring Failure Detection

1 Introduction

Over the last decade, the widespread adoption of Machine Learning (ML) in different areas has fundamentally changed industries, businesses, and science in many ways. Relying intensely on mathematics and software engineering, ML algorithms can learn relations from data, allowing complex problems to be tackled without programming a deterministic solution. Although the implementation of ML models has indeed become relatively fast and cheap, allowing state-of-the-art discoveries to become widely accessible, they have proven to be expensive to maintain and scale. The learning capabilities of ML algorithms impose a new software development paradigm, creating new challenges concerning the development and deployment of ML models, as acknowledged by academic literature [1–4]. Those challenges have driven a rising concern regarding best practices for the development of trustworthy ML systems. There are works in the recent academic literature that aim to determine the state-of-the-art in how teams apply software engineering best practices in ML development, including systematic literature reviews on academic and gray literature, as well as surveys among ML practitioners [5–7].

However, applying the best software engineering practices in ML is still a challenging task despite the consolidation of academic literature. The application of such principles may depend on experience or the size of the team, making the application of best practices impracticable for many teams, especially those outside the niche of

tech companies [7]. In this work, we propose a strategy to apply software engineering best practices with minimal technical overhead. The smooth application of the best practices from the beginning of an ML project can reduce the gap between research and engineering, helping the ML community to build better solutions in general. To illustrate our strategy, the problem of detecting line failure in the mooring system of a floating production storage and offloading (FPSO) unit will be used as a case study, providing an example of our strategy applied in the Oil and Gas (O&G) industry.

2 Related Work

The ML software development life cycle poses several engineering challenges. There are many particularities regarding how ML software solutions are developed and deployed, originating problems unaddressed by traditional software engineering best practices. That is a fact acknowledged by the academic literature. Sculley et al. [1] explore the concept of technical debt to evaluate risk factors of ML systems. They argue that ML systems are more likely to be subjected to technical debt because such systems have all the maintenance and development problems associated with traditional software, as well as a new set of issues specific to ML software. Following the rising concern about ML-specific challenges among engineers and researchers, many works in the academic literature emerged proposing practices to address different stages of the ML development process. Breck et al. [2] introduced 28 test and monitoring practices, as well as a model to score best practices adoption and measure the technical debt. Lwakatare et al. [3] propose a taxonomy to identify and classify software engineering challenges faced in systems that incorporate ML components. Recent works in the academic literature are focusing on the compilation of the state-of-the-art practices through systematic literature reviews and surveys, such as the work by Wan et al. [5], Giray et al. [4], Washizaki et al. [6], and Serban et al. [7].

Nevertheless, despite the consolidation of the academic literature, applying the practices to overcome the ML challenges is not a simple task. In this context, our strategy aims to contribute to the popularization of such practices in the O&G industry.

3 Applying Software Engineering Best Practices in ML

The consolidation of the literature on the best practices for ML software development facilitates the commitment of engineers and researchers in producing trustworthy ML systems. Nonetheless, there is a wide variety of practices to address all the challenges faced in the ML life cycle. Hence we propose a strategy to apply the best practices in ML development, taking the point of view of a practitioner from the beginning of an ML project. In this section, we present our strategy, structured into a set of ideas. By using a real-world problem, we will give examples of how to apply them.

3.1 Understand the Problem

The first step of the proposed strategy is to build a solid understanding of the problem. Knowing the motivations, challenges, and solutions available in the literature is critical to designing an ML solution. We exemplify this step of the strategy below with our problem of detecting a fault in mooring lines in FPSO units.

The problem of fault detection in FPSO mooring lines. The exploration of offshore oil fields in deep water poses challenges regarding station-keeping. To overcome that problem, the O&G industry developed FPSO platforms, which have been used to explore offshore oil fields successfully. These floating platforms are kept stable by mooring lines anchored to the seafloor. Mooring systems are a critical component to ensure position stability and safe platform operations. Hence, monitoring the integrity of mooring systems is essential to maintain the FPSO integrity. However, current approaches used to monitor mooring lines are inefficient, relying on on-site inspections with remote operated vehicles or sensors that are expensive to install and maintain [8]. Recently, researchers in the field are trying to adopt ML techniques to solve this problem using data such as metocean data and motion data – from the Differential Global Positioning System (DGPS) and Inertial Measurement Unit (IMU), resulting in the variables that constitute the 6 degrees of freedom (6DoF) of the platform – and the draft, or loading, of the platform. However, metocean data and FPSO loading data may not be available for a given platform. Although motion data is available for most platforms, acquiring line-breaking motion data needed to train the ML models is challenging. Such details will be fundamental for the design of our ML solution, which will be based on the state-of-the-art presented by the academic literature.

State-of-the-art solutions. Gumley, Henry, and Potts [9] used multi-layer perceptron (MLP) and kriging methods to predict the mooring state using GPS data and metocean data. The models were trained in real data without line breakage and then tested with line breakage. The error between the datasets was then used to detect line failure. Similarly, Prislin and Maroju [10] proposed a method named Position Response Learning System, which uses at its core MLP models to detect line failure using GPS data and metocean data. Chung et al. [11] used simulations to obtain data with line breakage, providing an alternative for the cases where GPS data with line breakage is not available. The approach using simulated data was also adopted by Saad et al. [12]. They used simulated data to train a model to predict platform motion. The error between predictions and measurements was then inputted into a classification model to detect line failure.

By completing these steps, the developer builds a clear understanding of the problem. Now it is time to set a clear goal, the next step in our strategy.

3.2 Define a Clear Objective

The efficient development of an ML project depends on a clear definition of its objectives. Software engineering best practices are adopted in ML applications to produce systems capable of resisting real-world challenges. Therefore, defining the goals of the project and methods to achieve them is essential. In this section, we illustrate this idea using the mooring fault detection problem in FPSO units.

Mooring Line Fault Detection in FPSO units. The goal of the case study is straightforward: detecting mooring line fault in FPSO units in real-time. However, multiple approaches are available in the literature. Based on recent academic literature on mooring line fault detection, using motion data for training ML models with supervised learning is a common approach as this data is available on most platforms. Supervised Learning (SL) consists in learning a function that maps an input to an output based on example input-output pairs, and it is widely used for training ML models. On the other hand, motion data with broken mooring lines may not be available, which is circumvented in the literature by using simulated data. Therefore, we will follow the approach proposed by Saad et al. [12], which uses simulations to generate line-faulted and non-line-faulted data. The simulation requires a hydrodynamic model of the platform and data on the environmental conditions of the platform's installation site. As an output, it generates time series corresponding to the platform's 6DoF. Neural network-based models, trained with SL using non-line-faulted data, are then used to produce motion forecasts. These forecasts are compared with measured data to generate an error signal, which is input to the classifier that detects the line failure, trained with SL using non-line-faulted and line-faulted data. Thus, our case-study project aims to develop a mooring line failure detection system by using time series forecasting models and classifiers models, as illustrated in Fig.1.

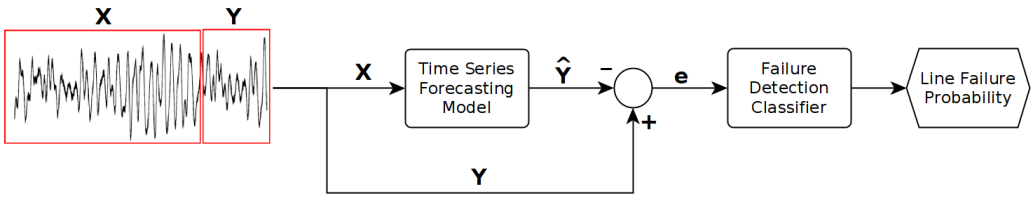

Figure 1. Method proposed by Saad et. al [12] for mooring line failure detection using time series forecasting models.

By defining the objectives of the project, it is necessary to plan how to achieve them. In the next steps of our strategy, we will define the path to achieve these objectives.

3.3 Choose Good Metrics

At the heart of ML models based on SL is the training process - also called the learning process - that allows ML models to learn from the data. The training process consists of minimizing (or maximizing) a loss function. Finally, at the end of the training process, the trained model is evaluated to determine if the model performs well enough. During the development of the ML components, this process is repeated many times throughout the experiments needed to fine-tune the ML models. Therefore, defining a metric that is easy to measure and understand is critical as it determines model assessment. The designer can use multiple metrics to get different

points of view, but the most important is to have metrics that capture the model goal efficiently.

Metrics for the mooring fault detection problem. We chose to detect mooring line failure based on time series forecasting models, which generate the error signal, and a classifier model, which detects line failures from the generated error. Therefore, it is also necessary to define two metrics: one for the forecasting models and one for the classifier. For the time series forecasting model, we will use the MASE metric, proposed by Hyndman and Koehler [13]. This metric is symmetric and scaled, providing good properties for evaluation. For the classifier metrics, we will use simple accuracy, precision, and recall metrics.

3.4 Adopt an Architectural Pattern

With the project's goals and methods defined, the focus naturally shifts to model development. The development of ML models involves several tasks: first, the model requirements must be defined. Then data must be collected, processed, and properly stored to be used on the training and testing routines that produce new models. After that, these models also need to be deployed for inference and monitored. These steps are well known by the ML community, defined as the ML workflow by Amershi et al. [14]. On the other hand, applications that rely on ML models, such as a system for detecting mooring line failure, have several requirements that are not contemplated by the ML workflow architecture, yet are common to traditional software. Firstly, the ML workflow itself needs to be managed, depending on some business logic. Besides, the communication between the ML-based application and its users - or maybe other applications - requires specific software to be implemented. While multi-layer architectural patterns, such as client/server architecture and three-layer architecture [15], are widely adopted to address such challenges, the particularities of ML systems are not fully addressed. This lack of design often induces technical debts such as "pipeline jungles" [1], affecting the operational stability of the ML system.

An architecture for the mooring line fault detection system. In our case study, the goal is to design a system capable of monitoring the mooring line integrity of FPSO units in real-time using motion data. Beyond processing data and training new models, the system has several other attributions, such as communication, inference, data collection, and storage. Yokoyama and Haruki [16] proposed an architectural pattern for ML systems designed to improve operational stability. The architecture decouples business logic and other aspects of the application from the ML components, as depicted by Fig. 2. The author argues that such architecture can make troubleshooting more accessible and enhance system stability. Following this idea, we will adopt this pattern at the beginning of the project, aiming to enable developers to clearly understand the system as a whole, not only the ML components.

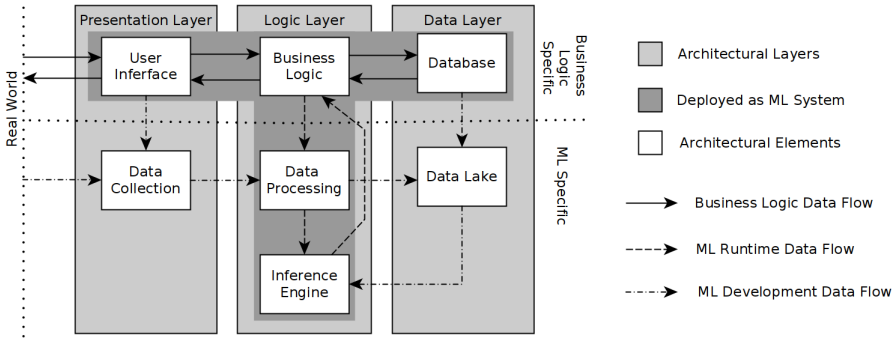

Figure 2. ML architectural pattern proposed by Yokoyama and Haruki [16].

3.5 Enable Teamwork

As mentioned before, the development of ML systems involves several components that are dependent on each other, requiring a multidisciplinary and cooperative team to overcome all the associated challenges. A team member might be working on data collection and data processing components, while other members work with training and testing routines. Not to mention the management of the infrastructure necessary to store data and train new models. Therefore, encouraging measures to enhance team communication and cooperation is indispensable

in an ML project.

Making teamwork easier. Considering our case study, teamwork is essential to manage all the challenges associated with developing the complete system. There are several methods and tools to enable teamwork, but the strategy we propose is simple, and it is based on three simple aspects: a communication channel, a shared backlog, and regular meetings. We choose those as they are easy to implement and accessible to most developers. Platforms such as Slack [1] provides features for team communication, as well as integration with many other tools, making it useful also for alerts and automation. With collaborative development platforms, such as Trello[2] and Jira[3], it is possible to manage tasks in a Kanban-style board, using a shared backlog among team members. With video-conference tools such as Zoom [4] and Google Meet[5] it is easy to conduct regular meetings, aligning team members with the project's objectives and fomenting discussion around the project's challenges. Hence, we do not claim that our strategy is enough to handle all the challenges regarding teamwork, but rather it guarantees minimal conditions for team management.

3.6 Enforce Code Quality

Enforcing code quality is often tricky for ML teams, as many ML practitioners have little or no software engineering expertise. Therefore, implementing tests and a helpful directory structure, versioning data, models and training scripts, and many other standard practices in traditional software development might be beyond the reach of many ML practitioners. Such measures are crucial to ensure code quality, essential for project development and maintenance. As this issue is a common problem among ML practitioners, the ML community developed frameworks to boost overall code quality. Using a framework can save much effort in implementing such standard practices in an ML project, making those practices more accessible to the ML community. Such frameworks allow team members to focus on fine-tuning the ML models, not to mention that the large community of practitioners using those frameworks makes the development less troublesome.

Making code quality accessible. To ensure code quality in our case study with minimal technical overhead, we use the framework Kedro [6]. According to its documentation, Kedro is an open-source framework for creating reproducible, maintainable, and modular ML code. It borrows concepts from traditional software engineering best practices, such as separation of concerns, modularity, and version control. Kedro also provides comprehensive documentation and tutorials, making it easy for any ML practitioner to adopt it. Therefore, we suggest the adoption of Kedro as a framework for enhancing overall code quality.

3.7 Enforce Strict Control over Data Engineering Processes

Data is the raw material for an ML application, accordingly needing to be collected, cleaned, and labeled before serving as input for training new ML models. The activities associated with data processing steps are known as data engineering, and in order to guarantee data quality, such processes must be strictly controlled. Errors and malfunctions in the data engineering processes can incur bugs that are difficult to detect, potentially causing the designer to draw wrong conclusions about the model's performance. Performing data checks on data sources, verifying data completeness, integrity, balancing, and distribution, and using reusable scripts for data cleansing and transformation are essential practices that help overcome data issues. In addition, it is also essential to keep data on a shared infrastructure, from where data is served to ML components and team members. Data processing is often expensive and time-consuming, so centralizing data engineering processes and data serving is crucial for fast and efficient development.

Making data quality accessible. In order to train the models for our approach to detecting mooring line failures, centralizing the data engineering processes is imperative given the challenges associated with data simulation. By centralizing the data engineering process, all team members and ML components can access data ready for ML training and inference. This is also important because, in the inference runtime, the input data must be sent to the same data engineering pipeline as the data used in the training runtime.

[1] https://slack.com
[2] https://trello.com
[3] https://www.atlassian.com/software/jira
[4] https://zoom.com
[5] https://meet.google.com/
[6] https://kedro.readthedocs.io/

3.8 Know Your Data

Despite its capabilities to learn patterns directly from data, ML models must be designed to address a particular problem. In other words, ML developers must explore data as best as possible in order to adapt the ML models better. In these terms, a strategy for enhancing the overall knowledge about the data enables efficient analytics. In order to achieve that, it is crucial to maintain data in a ready to be analyzed form, saving time that would be spent on data engineering tasks. It is also crucial to work on good visualization for data and metrics, which allows for a more insightful understanding of how to better train models. From the point of view of our case study, data is composed of a set of multivariate time series that represent the motion of an FPSO platform. Therefore, connecting data to the dynamics of vessel motion through data visualization is essential to enhance the comprehension of the problem, directly reflecting on developers' capabilities to design robust ML applications.

3.9 Enable Efficient Experimentation

Enabling experimentation within the framework of the system is deeply entangled with the commitment with the best practices for coding, presented in subsection 3.6. Such practices enhance code flexibility and maintainability, making it more accessible for the ML developer to experiment with different configurations and later incorporating the results back into the system. However, in addition to committing to coding best practices, it is necessary to build software that enables the experimentation to be tracked, connecting all the components used on the experiment, such as data, training scripts, model hyperparameters, and other important information needed to guarantee the results reproducibility. Nonetheless, implementing and debugging such software can be arduous, requiring time, effort, and experience that may not be available. Fortunately, there are open-source platforms designed to address such ML experimentation particularities. One that is worth mentioning is the MLFlow [7] project, an open source platform for managing the ML lifecycle, providing functionalities for tracking experiments, code packaging, and model deployment and storage. The MLFlow is growing popular among ML practitioners and is constantly being improved, as observed in work by Chen et al. [17]. By using the Kedro framework, previously cited in subsection 3.6, the system can be easily extended with the MLFlow functionalities, making the experiment tracking relatively easy to be implemented. Therefore, using such open source projects is of great help in implementing the best practices for ML development, as they provide code that is implemented and debugged by a large community of users.

Using MLFlow to implement the experiment structure and the framework Kedro enables easy experimentation and the automatic tracking of data, parameters, scripts, and results. Such structure enables ML developers to focus on experimentation, drastically reducing the amount of work necessary to apply the best practices for ML development.

4 Discussion

The development of ML applications is non-deterministic, depending directly on the experimentation required to fine tune the models. This characteristic entails several challenges concerning software development, as the system is constantly changing. Consequently, ML developers strive to incorporate experiment results into the application, not only because the experimentation is often done outside the framework of the system but also because the experiments are not fully reproducible. Therefore, enabling an efficient experimentation pipeline is critical to enable straightforward development.

Although based on the specialized literature, the strategy presented here was built out of the authors' experience with developing applications with ML components. Also, we mention that the problem of mooring system failure detection, adopted as a case study, was chosen because of the authors' familiarity with the subject, but the proposed strategy could be applied in any other ML application. Therefore, despite the limitations of our point of view, we share our experience in enforcing best practices in ML development, aiming to make its application more accessible for ML developers in general and encourage discussion of the subject within the community working on the application of ML solutions in the O&G industry.

5 Conclusions

We presented a strategy for applying best practices of software engineering for ML with minimal technical overhead. The strategy comprises nine main topics, which altogether propose a method for applying most of the

[7] https://mlflow.org/

state of the art best practices of software engineering in ML. The strategy does not aim to address all the issues related to ML-based applications but rather to cover the essential topics straightforwardly.

Acknowledgements. This work was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES Finance Code 001), Brazil, and ANP/PETROBRAS, Brazil (project N. 21721-6). We also gratefully acknowledge partial support from CNPq (grants 310085/2020-9, and 310127/2020-3).

Authorship statement. The authors hereby confirm that they are the sole liable persons responsible for the authorship of this work, and that all material that has been herein included as part of the present paper is either the property (and authorship) of the authors, or has the permission of the owners to be included here.

References

[1] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pp. 2503–2511, Cambridge, MA, USA. MIT Press, 2015.

[2] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley. The ml test score: A rubric for ml production readiness and technical debt reduction. In *Proceedings of IEEE Big Data*, 2017.

[3] L. E. Lwakatare, A. Raj, J. Bosch, H. H. Olsson, and I. Crnkovic. A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation. In P. Kruchten, S. Fraser, and F. Coallier, eds, *Agile Processes in Software Engineering and Extreme Programming*, volume 355, pp. 227–243. Springer International Publishing, Cham. Series Title: Lecture Notes in Business Information Processing, 2019.

[4] G. Giray. A software engineering perspective on engineering machine learning systems: State of the art and challenges. *Journal of Systems and Software*, vol. 180, pp. 111031, 2021.

[5] Z. Wan, X. Xia, D. Lo, and G. C. Murphy. How does machine learning change software development practices? *IEEE Transactions on Software Engineering*, vol. , pp. 1–1, 2019.

[6] H. Washizaki, H. Uchida, F. Khomh, and Y.-G. Guéhéneuc. Studying software engineering patterns for designing machine learning systems. In *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, pp. 49–495, 2019.

[7] A. Serban, van der K. Blom, H. Hoos, and J. Visser. Adoption and effects of software engineering best practices in machine learning. *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, vol. , 2020.

[8] *Mooring System Life Extension Using Subsea Inspection Technologies*, volume All Days of *OTC Offshore Technology Conference*. OTC-24184-MS, 2013.

[9] *A Novel Method for Predicting the Motion of Moored Floating Bodies*, volume Volume 3: Structures, Safety and Reliability of *International Conference on Offshore Mechanics and Arctic Engineering*. V003T02A056, 2016.

[10] *Mooring Integrity and Machine Learning*, volume Day 3 Wed, May 03, 2017 of *OTC Offshore Technology Conference*. D031S034R007, 2017.

[11] M. Chung, S. Kim, K. Lee, and D. H. Shin. Detection of damaged mooring line based on deep neural networks. *Ocean Engineering*, vol. 209, pp. 107522, 2020.

[12] A. M. Saad, F. Schopp, R. A. Barreira, I. H. F. Santos, E. A. Tannuri, E. S. Gomi, and A. H. R. Costa. Using neural network approaches to detect mooring line failure. *IEEE Access*, vol. 9, pp. 27678–27695, 2021.

[13] R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, vol. 22, n. 4, pp. 679–688, 2006.

[14] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 291–300, 2019.

[15] M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee, and R. Stafford. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[16] H. Yokoyama. Machine learning system architectural pattern for improving operational stability. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 267–274, 2019.

[17] A. Chen, A. Chow, A. Davidson, A. DCunha, A. Ghodsi, S. A. Hong, A. Konwinski, C. Mewald, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, A. Singh, F. Xie, M. Zaharia, R. Zang, J. Zheng, and C. Zumar. 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.