Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle

Andrew Chen, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Clemens Mewald, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Avesh Singh, Fen Xie, Matei Zaharia, Richard Zang, Juntai Zheng, Corey Zumar
Databricks, Inc.

ABSTRACT

MLflow is a popular open source platform for managing ML development, including experiment tracking, reproducibility, and deployment. In this paper, we discuss user feedback collected since MLflow was launched in 2018, as well as three major features we have introduced in response to this feedback: a Model Registry for collaborative model management and review, tools for simplifying ML code instrumentation, and experiment analytics functions for extracting insights from millions of ML experiments.

ACM Reference Format:
Andrew Chen, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Clemens Mewald, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Avesh Singh, Fen Xie, Matei Zaharia, Richard Zang, Juntai Zheng, Corey Zumar . 2020. Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle. In *International Workshop on Data Management for End-to-End Machine Learning (DEEM'20), June 14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3399579.3399867

1 INTRODUCTION

Machine learning development requires solving new problems that are not part of the standard software development lifecycle. While traditional software has a well-defined set of product features to be built, ML development revolves around *experimentation*: ML developers constantly experiment with new datasets, models, software libraries, tuning parameters, etc. to optimize a metric such as model accuracy. Because model performance depends heavily on the input data and training process, *reproducibility* is paramount throughout ML development. Finally, in order to have business impact, ML applications need to be *deployed* to production in an inference-compatible environment; deployments need to be monitored and regularly updated.

Faced with these challenges, some organizations have built *ML platforms* for model development and deployment. Unfortunately, many ML developers are bottlenecked by the limited set of ML libraries and software tools supported by these platforms; for example, Google's TFX [11] is optimized exclusively for TensorFlow [1] models.

In 2018, we introduced MLflow [15]: an open source platform for the ML lifecycle designed to work with any ML library and programming language. MLflow has been widely adopted in industry and the academic community. MLflow powers ML efforts in the energy, biotechnology, and online retail sectors, varying in scale from solo practitioner projects to large organizational initiatives involving hundreds of ML engineers. Additionally, the platform has received contributions from over 170 developers outside of Databricks.

In this paper, we discuss trends in end-to-end machine learning distilled from MLflow adopters' use cases and feedback, as well as three major platform features we have introduced in response: a Model Registry for collaborative model management and review, *autologging* tools that simplify ML code instrumentation, and experiment analytics functions for extracting insights from millions of ML experiments. Finally, we also discuss emerging challenges in ML development that MLflow is well-positioned to address.

2 RELATED WORK

Many software systems aim to simplify ML development. The closest to our work are the end-to-end ML platforms at large web companies. For example, Facebook's FBLearner [6] lets users create reusable ML workflows that operate on warehoused data, and Google's TFX [11] provides data preparation and serving tools. Though these platforms significantly accelerate ML development, they restrict users to a specific

DEEM '20, June 14, 2020, Portland, OR

ACM ISBN 978-1-4503-8023-2/20/06…$15.00
https://doi.org/10.1145/3399579.3399867

set of ML libraries. In contrast, MLflow enables users to easily incorporate their own tools and software throughout the machine learning lifecycle, including custom training steps, inference code, and lineage information.

Other systems also tackle specific problems within the ML lifecycle; MLflow combines concepts from these systems with new ones to create a unified platform. TensorBoard [8] and Sacred [9] are tools for tracking ML experiments; MLflow Tracking provides these capabilities via an open interface design that enables developers to record, store, and query results using tools and infrastructure of their choosing, as discussed in subsection 3.2. Binder [3] and CodaLab [7] enable reproducible software runs; similarly, MLflow Projects facilitates reproducible runs via a standard packaging format. ONNX [12] is a cross-library model serialization format; the MLflow Model format bundles serialized models with additional dependency information and introduces the concept of *flavors*, which enable users to load and evaluate models across multiple ML frameworks and levels of abstraction. Clipper [5] and Amazon SageMaker [13] are real-time serving solutions for ML models; MLflow includes utilities to package MLflow Models as Docker containers for deployment to these services. Finally, ModelDB [14] catalogues models along with lineage information; the MLflow Model Registry augments these features with novel tools for reviewing models and transitioning them through configurable deployment lifecycles, as discussed in subsection 4.1.

3 PLATFORM OVERVIEW

MLflow defines components that are designed to address fundamental challenges in each phase of the machine learning lifecycle, from model development through productionization. Each component is built around an *open interface* philosophy, providing general abstractions for its functionality that enable the platform to operate at varying scales and achieve compatibility with the large ecosystem of programming languages, ML libraries, and deployment environments.

3.1 Components

The MLflow platform defines four components that structure the ML development process:

- **MLflow Tracking** is an API for recording experiment runs, including code used, parameters, input data, metrics, and output files. These runs can be viewed, compared, and searched using an API and UI.
- **MLflow Models** is a generic format for packaging models, including code and data dependencies, that is compatible with diverse deployment environments. Each MLflow Model defines a set of *flavors* that can be used to evaluate it across multiple ML environments.
- **MLflow Projects** is a format for packaging code into reusable projects. Using a YAML configuration file, each project defines its dependencies, code to run, and parameters for programmatic execution.
- **MLflow Model Registry** is a collaborative hub for cataloguing models and managing their deployment lifecycles. The Model Registry is MLflow's newest component and is discussed further in subsection 4.1.

3.2 Architecture

MLflow is built around an *open interface* philosophy. The platform defines general abstractions for each of its four components and includes implementations of these component interfaces for a variety of standard tools and infrastructure, many of which have been contributed by MLflow community members. For instance, MLflow Tracking defines a *Backend Store* interface for recording experiment run metadata and an *Artifact Store* interface for recording model files, providing implementations for the UNIX and NTFS filesystems, as well as several industry-grade relational databases.

This open interface approach enables the platform to satisfy heavyweight research or enterprise requirements by affording developers the ability to run MLflow on proprietary infrastructure and incorporate additional features, such as role-based access controls. Simultaneously, MLflow's default component implementations reduce setup overhead and deliver out-of-the-box support for common ML tools and workflows.

4 TRENDS AND PLATFORM ADDITIONS

Through analyses of MLflow adopters' use cases, we have identified several trends in end-to-end machine learning development that have spurred advancements in the platform.

4.1 Model management

While initially architecting MLflow, we identified the organization and management of model training sessions (experiment runs) as a significant challenge for ML practitioners, particularly for those working in large-scale, collaborative environments. As developers increasingly leveraged MLflow Tracking to streamline their model training efforts, they began to encounter similar challenges with managing and sharing their models. In particular, several adopters in the IoT domain described use cases wherein a unique model is built for each independent device or entity. These training processes frequently produce tens of thousands or even hundreds of thousands of models. ML practitioners and deployment engineers must vet each of these models, map them to specific production applications, and deploy them systematically to guard against regressions and breakages.

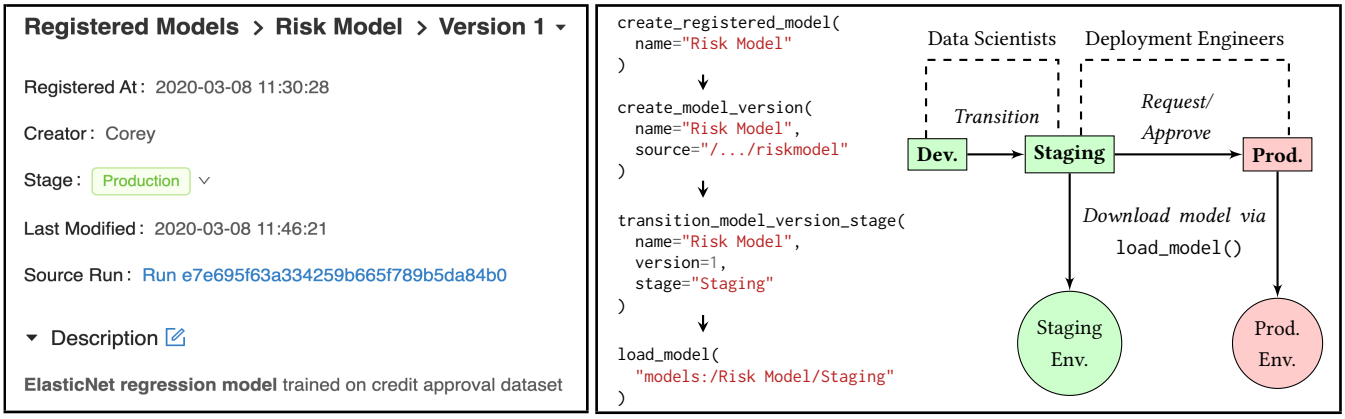

Figure 1: Left: The Model Registry UI displays author and lineage information for each model; users can transition model versions through predefined lifecycle stages and add markdown descriptions. Right: A diagram depicting an example collaborative Model Registry workflow where models are deployed to staging and production environments; a sequence of API calls for the "Data Scientists" portion of the workflow is also displayed.

Operating effectively at this scale necessitates platform functionality beyond the model cataloguing and lineage tracking capabilities of preexisting model management systems; organizations also require tools for the collaborative review and structured deployment of models.

To address these needs, we introduced the MLflow Model Registry: a collaborative hub for managing the model deployment lifecycle. In addition to providing cataloguing and lineage tracking capabilities, the Model Registry standardizes the model deployment workflow by enabling ML developers and deployment engineers to version their models and transition them through four logical stages: "Development," "Staging," "Production," and "Archived." Further, organizations can restrict access to particular stages on a per-user or per-role basis, and the Model Registry enables users to request stage transitions from colleagues. These stages structure the deployment process and provide a model review framework, guarding against common pitfalls, such as the deployment of broken or inferior models to production.

Finally, we observed that many organizations rely on automated frameworks to periodically test and update model deployments. The Model Registry integrates with these *continuous integration and deployment (CI/CD)* tools by providing APIs for fetching models by version and stage that developers can incorporate into their existing CI/CD workflows.

To date, users are managing millions of models with the Model Registry; applications range from demand prediction in transportation logistics to forecasting home energy usage. Figure 1 displays the Model Registry's Model Version UI and an example workflow where data scientists develop models, deployment engineers evaluate transition requests for production deployment, and CI/CD tools fetch model updates via the `load_model()` API.

4.2 Ease of instrumentation

We observe that ML libraries and frameworks are continuing to abstract and simplify the interfaces used to define and train models. These simplified workflows serve as an impetus for providing more convenient mechanisms to extract and record training metadata (e.g., hyperparameters and performance metrics). To reduce the burden of instrumenting ML code in MLflow, we have incorporated *autologging* utilities for specific ML libraries that automatically record relevant training metadata to MLflow Tracking and serialize model graphs in the MLflow Model format. These convenience APIs reduce the overhead of instrumentation to a single line of code. Further, they provide a contract ensuring that relevant metadata is collected as a given ML library evolves. Figure 2 displays an example of MLflow's Python autologging API for TensorFlow [1] training sessions, demonstrating its advantages over preexisting instrumentation methods. Since the release of MLflow's first autologging API, community members have contributed autologging integrations with several prominent ML libraries, such as XGBoost [4].

4.3 Experiment analytics

Organizations are increasingly applying machine learning to extract insights from data and inform critical decisions. Accordingly, information produced by ML methods is often represented and consumed in the form of recurring reports or performance dashboards. Adopters have described these patterns in the context of Business Intelligence (BI) as well as project management: ML developers are leveraging Tracking to catalogue key performance metrics as outputs of their experiment runs, and experiments themselves are serving as a gauge of an ML initiative's progress or effectiveness over time. These developers then employ data science tools and

visualization libraries to analyze their experiment data, generating informative reports and dashboards. To facilitate this process, MLflow introduced APIs for exporting experiment data in tabular format via Spark [16], Pandas [10], and SQL.

```python
# Without autologging
import mlflow
mlflow.log_param(
  "layers", layers
)
model = train_model()
mlflow.log_metric(
  "mse", model.mse()
)
mlflow.log_artifact(
  "plot", plot(model)
)
mlflow.tensorflow.log_model(
  model
)
```

```python
# With autologging
import mlflow


mlflow.tensorflow.autolog()
model = train_model()
```

Figure 2: MLflow's autologging APIs simplify ML code instrumentation. In this Python example, the `mlflow.tensorflow.autolog()` invocation highlighted in the right code snippet replaces the verbose instrumentation logic highlighted in the left code snippet.

5 FUTURE WORK

In addition to the platform improvements discussed previously, adopters' use cases and feedback have illuminated several other natural extension points for MLflow.

We identify data reproducibility as a significant challenge for many organizations. Because training, validation, and test datasets are fundamental to model behavior and to the measurement of model performance, many ML practitioners have expressed the need to systematically version their ML datasets and incorporate them into the lineage information associated with each model.

Further, as the scale of model deployments increases for batch and real-time scoring, we observe that ML developers require *model telemetry* solutions for capturing statistical performance insights from the inputs and outputs of deployed models. As such tools emerge in the context of specific deployment systems, we contend that ML practitioners would benefit from a framework-agnostic structure for production model monitoring that provides compatibility with a broad set of deployment environments.

Finally, we observe that many ML training and inference processes consist of multiple subtasks. Representing and efficiently executing these *multi-step* workflows presents significant challenges for ML practitioners: for example, many such workflows rely on implicit input-output behavior at each step, and alterations to a single step may cause regressions in the broader workflow. We posit that a general representation format for multi-step ML workflows with explicit input-output interfaces would mitigate these failure modes and enable workflow execution systems, such as Apache Airflow [2], to exploit the parallelism of independent subtasks.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*, Vol. 16. 265–283.

[2] Apache Airflow. 2020. Apache Airflow. https://airflow.apache.org/

[3] Binder 2020. Binder. https://mybinder.org.

[4] T. Chen and C. Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[5] D. Crankshaw, X. Wang, G. Zhou, M.J. Franklin, J.E. Gonzalez, and I. Stoica. 2017. Clipper: A Low-latency Online Prediction Serving System. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI'17)*. USENIX Association, Berkeley, CA, USA, 613–627. http://dl.acm.org/citation.cfm?id=3154630.3154681

[6] J. Dunn. 2016. Introducing FBLearner Flow: Facebook's AI backbone. https://code.fb.com/core-data/introducing-fblearner-flow-facebook-s-ai-backbone

[7] Percy Liang et al. 2020. CodaLab. https://worksheets.codalab.org.

[8] Google. 2020. TensorBoard: Visualizing Learning. https://www.tensorflow.org/guide/summaries_and_tensorboard.

[9] K. Greff, A. Klein, M. Chovanec, F. Hutter, and J. Schmidhuber. 2017. The Sacred Infrastructure for Computational Research. In *Proceedings of the 16th Python in Science Conference*, K. Huff, D. Lippa, D. Niederhut, and M. Pacer (Eds.). 49 – 56. https://doi.org/10.25080/shinma-7f4c6e7-008

[10] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). 51 – 56.

[11] A.N. Modi, C.Y. Koo, C.Y. Foo, C. Mewald, D.M. Baylor, E. Breck, H. Cheng, J. Wilkiewicz, L. Koc, L. Lew, M.A. Zinkevich, M. Wicke, M. Ispir, N. Polyzotis, N. Fiedel, S.E. Haykal, S. Whang, S. Roy, S. Ramesh, V. Jain, X. Zhang, and Z. Haque. 2017. TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. In *KDD 2017*.

[12] ONNX Group. 2020. ONNX. https://onnx.ai.

[13] SageMaker 2020. Amazon SageMaker. https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html.

[14] M. Vartak, H. Subramanyam, W. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia. 2016. ModelDB: A System for Machine Learning Model Management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA '16)*. Association for Computing Machinery, New York, NY, USA, Article 14, 3 pages. https://doi.org/10.1145/2939502.2939516

[15] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S.A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar. 2018. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Engineering Bulletin* 41(4) (2018).

[16] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataramen, M.J. Franklin, A. Ghodsi, J.E. Gonzalez, S. Shenker, and I. Stoica. 2016. Apache Spark: a unified engine for big data processing. *Commun. ACM* 59, 11 (2016), 56–65. https://doi.org/10.1145/2934664