Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2021, 8(11):134-145**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Databricks MLflow

Ravi Shankar Koppula

Email: ravikoppula100@gmail.com

Abstract This paper examines MLflow, an open-source platform specifically designed to simplify the management of the machine learning lifecycle. It covers various aspects, such as experiment tracking, code packaging, and sharing and deployment of models. The paper focuses on the integration of MLflow with Databricks, emphasizing how this collaboration enhances automatic experiment tracking and provides easier access to data and models. This integration ultimately leads to more efficient and reproducible machine learning workflows. The paper thoroughly explores the four main components of MLflow: MLflow Tracking, MLflow Projects, MLflow Models, and MLflow Model Registry. It highlights the platform's ability to address common challenges in machine learning projects, including experiment management, reproducibility, and model deployment across different environments. Furthermore, it delves into the crucial roles of MLflow Models and MLflow Model Registry in enabling flexible deployment options, such as batch, streaming, real-time, and edge deployments. These components also ensure centralized model management, compliance through audit logs, and facilitate collaboration among team members. In conclusion, the paper states that MLflow greatly contributes to overcoming the complexities of deploying real-time machine learning systems. It achieves this by offering streamlined workflows, centralized management, and comprehensive model deployment strategies. As a result, MLflow is seen as an invaluable resource for data scientists, machine learning practitioners, and researchers who are eager to enhance the efficiency and reproducibility of their machine learning projects.

Keywords MLflow, Model Registry,MLflow models, MLflow tracking , Model serving, Databricks MLflow

1. Introduction

MLflow is an open-source platform to manage the ML lifecycle. It tackles four primary functions: it offers tracking experiments to record and compare parameters and results. It offers a project component, packaging code and its dependencies to reproduce runs on any platform. It offers a central model store to collaboratively manage the staged models. Lastly, it offers tools to simplify deployment of models to new data environments. MLflow is simply designed to integrate with existing ML applied codes. It is linked to a data science forum which shares the system to test how MLflow is helpful. As a result, it has improved over 500 existing ML applied codes.

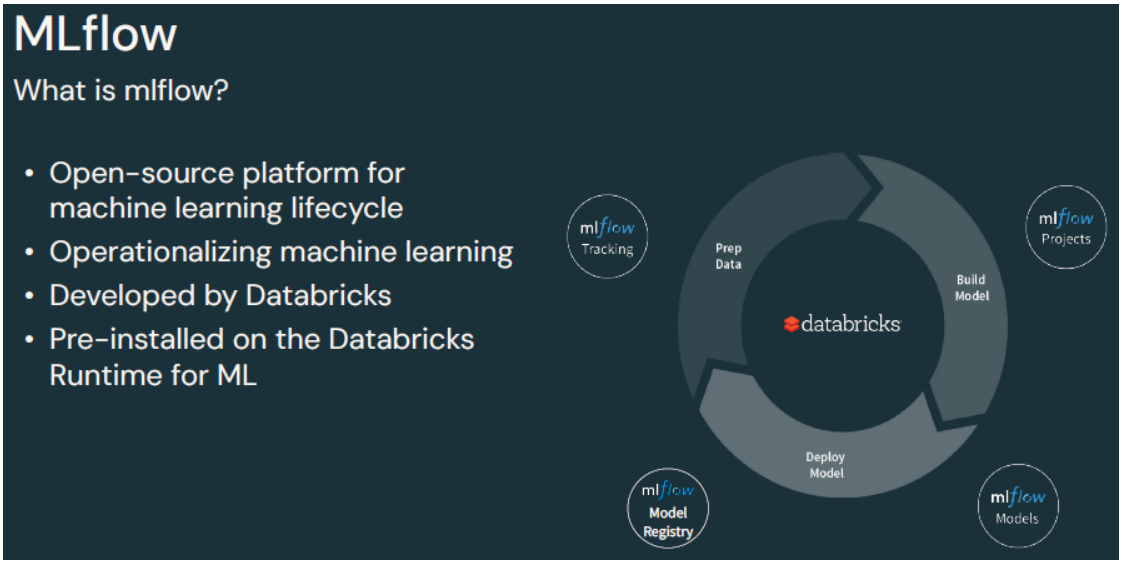
MLflow simplifies tracking experiments because it is difficult to keep a report when the data scientist is enhancing and changing the model. Part of this is due to not having a standard record keeping. Often data scientists will record their experiments via Excel, Jupyter notebook, or perhaps on a piece of paper. This record is likely to be lost as programming codes typically change rapidly during an ML project. Tracking can be said as an extension of recording experiments. Good tracking will not only record what event or experiment took place but it can track the entire lineage of a model. With MLflow, data scientists can tag and navigate to previous experiments which allows them to track the history of a model compared to a present model. Easier navigation

for tracking to identify the best model is an everyday problem for a machine learner and it is said that MLflow is directly attacking this problem.[1]

1.1. Overview

MLFlow is a platform that will not only centralize efforts to track experiments and their results but also allow the next steps to be seamless. With tracking, MLFlow logs all the parameters, code, and data to be used in a run, be it on a local machine or cloud server. This ensures that the logged information can be used to reproduce the run in another environment if needed. This goes hand in hand with the packaging steps as whatever has been logged can be used to package the model in an environment and then use the same environment to reproduce the model if needed. Sharing will be easy as all runs and packages are stored in directories and can be sent to other data scientists who can then continue from where the last left off, no matter if it's a short-term collaboration or a full model production with colleagues in the same team. This works wonders with organizations whose machine learning models are spread around and are unsure of the most cost-effective manner to deploy machine learning on a large scale.

MLFlow is an open-source platform for the complete machine learning lifecycle. It tackles three primary functions which are tracking experiments, packaging code into reproducible runs, and sharing and deploying models so that organizations can utilize the models. Create a single module to take the model from experimentation into the full production with a workflow. MLFlow allows diverse tools to be incorporated at each step of the lifecycle, providing ease of use for all data scientists who currently have their own stacks.[2]

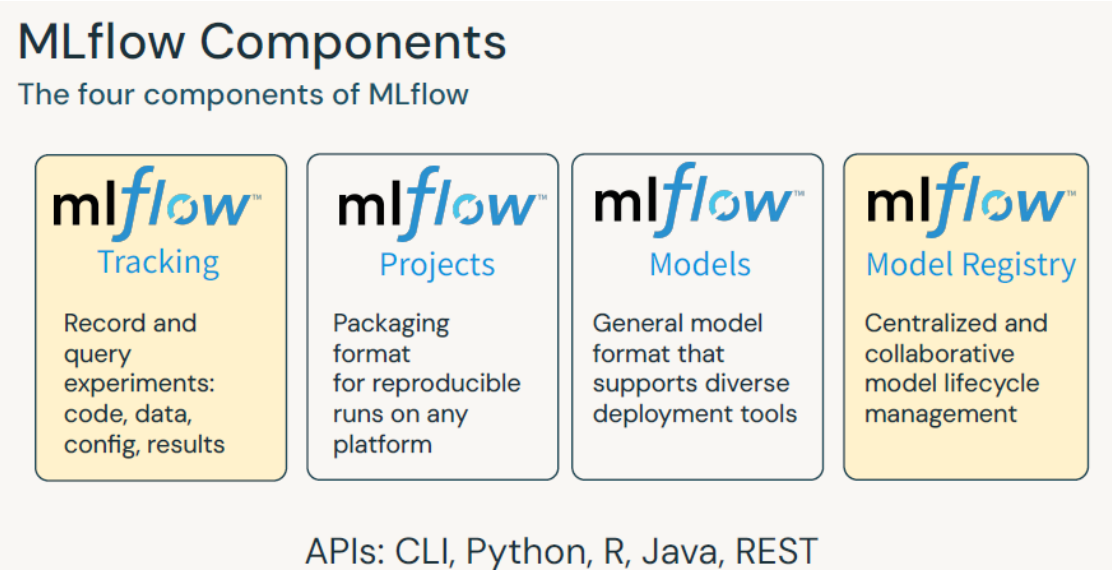


Core Machine Learning Issues

Modern machine learning (ML) lifecycle is accompanied by an array of formidable challenges. These include but are not limited to the intricate task of meticulously maintaining an organized record of experiments and model developments. Furthermore, attaining code reproducibility is a crucial hurdle to overcome, as it is essential for ensuring the reliability and robustness of ML algorithms. Another crucial challenge lies in the constant need for model comparison, as practitioners strive to identify the most optimal solution. Lastly, standardization of packaging and deploying models emerges as a pressing concern, demanding careful attention and meticulous implementation. These formidable challenges underscore the complexity and multidimensional nature of the modern ML lifecycle. [3]

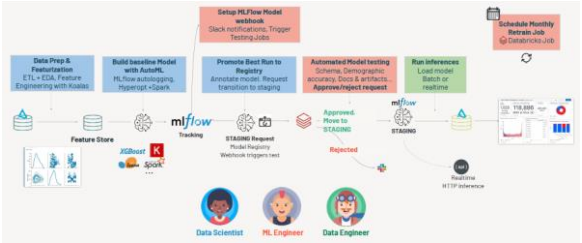
1.2. Benefits

Databricks MLflow provides a platform to manage the complete machine learning lifecycle. It provides four main components:



- MLflow Tracking is an API and UI for logging parameters, code, and results. It is designed to support the most mainstream machine learning libraries.
- MLflow Projects provides a standard format for packaging machine learning code in a shareable and reproducible way. A project is a directory of code or a git repository.
- MLflow Models is a general model packaging format that supports all models. It does not bind the user to any specific ML library or framework.
- MLflow Registry is a place to store and manage models. It is a central repository for keeping track of all the models. It ensures that models are well-managed and can be used by others in the future.[4]

When an organization adopts a machine learning model, there are different people who are interested in its different aspects. A software engineer may be interested in its end-to-end application in the product, while a data scientist may be interested in its statistical aspects and model building, and a business analyst may be interested in its effectiveness, which can be in terms of return on investments or how it improves user engagement. Considering such a broad audience, it is required that a machine learning platform must provide tools to cater to the needs of all the above-mentioned people.



1.3. Use Cases

MLflow introduced in the previous section is a platform for the complete machine learning lifecycle. With its RESTful APIs for native platform coupling, Databrick's collaboration with MLflow gives their already robust big data platform a way to excel in organizing and replicating the machine learning pipeline. The rest of this document will focus on how MLflow fits the DatabricksCtrl platform and the different intents of the MLflow platform to the platform.

A very powerful feature of the Databricks environment is the scalability and optimization it provides to parallel processing. It's no mistake that MLlib and now Spark MLlib have been a successful machine learning platform

with their ability to bring the algorithm to the data and the linear scalability in training models. Spark's MLLib model is one form of the model representation that can be loaded into the MLflow tracking API. With MLflow aiming to organize, version and productionaize machine learning models, Databricks is in a unique position to provide a platform where the community that is developing Spark ML models can effectively track it through its life cycle.

Building machine learning pipelines with Spark is composed of a DataFrame related algorithm, multiple feature engineering steps and finally the model itself. With its integration with Apache Hive and Apache Parquet, Databricks is typically used as a platform to bring structure to large data sets stored in HDFS clusters. Due to the potentially high computational cost of doing large scale machine learning in a distributed environment and the abilities of machine learning pipeline to address it, often times it is desirable to run the model itself in a smaller non-distributed environment for testing and predictions. One of MLflow's goals is to provide an experience where the steps to an experiment are saved, the data is logged, the code is saved and the reproducibility and progress is query-able. In the case of attempting to bring the machine learning to a smaller cluster or a different environment, one would ideally like to take that experiment and reproduce it, maybe with different parameters and a portion of the data, in its different cluster or environment. The ability to reproduce the experiment in a different context is a strong point where MLflow would benefit a typical DatabricksCtrl user.

2. Getting Started

Setting up a Databricks Workspace

MLflow is especially easy to use with Databricks, because it supports running MLflow projects directly on Databricks. When you run an MLflow project on Databricks, MLflow uses a Databricks Job to start a Databricks Workspace that runs the project code. Databricks Jobs are known for their ability to create reproducible data workflows with any Spark code. These jobs are directly linked to MLflow Project, making it possible to run Spark projects in any language. This means that you can treat your data and training code as a unified project and easily execute it on Databricks. The Databricks runs of the project automatically log information in relation to the run such as source code version, configuration, and results, which makes management of data science projects easier. Databricks also provides easy visualization of any artifacts like a Scikit learn model, making it easier to manage the results of your experiments. Databricks features automatic logging to a Databricks File System (DBFS) or an external storage service. DBFS is especially useful for MLflow projects because you can use it to store run data, while tracking run and experiment information in the MLflow Tracking Server. This system is ideal for maintaining organized record of your experiments while using an event driven process.

By default, run output, code, and any files written with log artifacts methods from the API will be stored in a directory called mlruns in your current working directory. If you would like to store these files in a different location, set a MLFLOW_EXPERIMENT_LOCATION environment variable to the desired location of the experiment.

After installing the package, make sure you have an environment variable called MLFLOW_HOME that is set to the desired location of your MLflow server. This location determines where tracking data and files are stored. If you restart the server with this location changed, then you will have a clean instance with no recorded runs. This setup is useful if you want separate tracking server instances for different users, or if you want to periodically archive tracking data and restart with a fresh slate.

To install from source, first clone the MLflow git repository and then within the repository, run pip install -e .. If you would like to install a specific branch or pull request, then checkout that branch or pull request first.

To install MLflow from PyPI via pip, run:

pip install mlflow

You can install MLflow using pip, its source tarball, or simply clone the git repository and install from the project root. We recommend installing nightly if you want to try out the latest features and bug fixes, but please note that nightly does not receive the same level of testing as release.

The easiest way to use MLflow is to install the Python package, as described in the MLflow quickstart. If you are integrating MLflow into a project that does not yet use it, we generally recommend using pip or conda to install MLflow globally (i.e., in a shared location rather than a project-specific location). This approach makes it easy to use the full MLflow suite in the future—particularly the tracking component, which requires invoking mlflow functions in your training code.

2.1. Installation

Next, we need to install MLflow on a server or VM. MLflow works with Databricks, but also runs on other cloud platforms or an on-prem server. It's easiest to start with MLflow on Databricks, then run it locally and finally set up a cloud server. A cloud deployment typically just requires the proper configuration for S3 or Azure blob storage. Continuing with the example, below are instructions for a local install. See here for more information on the Databricks deployment and the server deployment. MLflow can be installed on various distributions of Linux via pip, on macOS with Homebrew, and on Windows with pip.

Please ensure you are using Python ≥ 3.5. We also recommend using virtualenv as detailed here to avoid permission issues.The simplest way to install MLflow is using pip. With a command line or terminal, type: pip install mlflow

2.2. Setting up a Databricks Workspace

Step 1. Log into Databricks. If your organization has not signed up for Databricks, each account user can access a free version of the Databricks workspace.

Step 2. Choose the best pricing option. Once you log into the Databricks account, the fastest way to get to the Admin Console is to click the drop down in the workspace picker in the upper right-hand corner and click on the "Admin Console". This link will take the user to the system-internal interface for the administrative account. All features of the Databricks workspace can be configured from here.

To switch to a paid version, just type in the access code from the trial version into the "Plan Management" tab. There are a few different pricing options, so choose the one best for your organization.

Step 3. Create a new Databricks workspace. If your organization has the trial version and created a new workspace, that workspace will still be active once the organization switches to a paid version. A new workspace can be created from the Admin Console by typing a new name into the "Workspace Name" field and clicking "Create Workspace". If an organization has multiple workspaces and wants to delete some, the admin just has to choose the corresponding workspace from the drop down, and delete it.

Step 4. Adjust the settings for the workspace.

To the left of the drop down to select a workspace, there is a link called "Settings" with a gear icon. Click this link to adjust the settings for the entire workspace. This includes adding users, handling security, AWS integration and metadata retention among others. Check here to verify that the settings are appropriate for the organization, and make adjustments as needed.

Step 5: Start a Community Edition workspace.

The Community Edition has a few different settings from the standard editions, and there are still different settings depending on if the workspace is a new version or a version that was upgraded from a previous free trial. Check the link for details on what the Community Edition offers, and click the link at the bottom of the page that says "Please create a new Databricks Community Edition workspace for me" to start up the Community Edition.

2.3. Configuring MLFlow

s"s3cmd --configure --access_key=$token --secret_key=$token".!

%sh export MLFLOW_EXTRA_PYPI_INDEX_URL=https://pypi.mirrors.ustc.edu.cn/simple

You may also install a private mlflow PyPI mirror to use a specific MLflow version when conducting mlflow runs or launching projects on Databricks. This can be specified with the MLFLOW_EXTRA_PYPI_INDEX_URL.

MLFLOW_EXTRA_PYPI_INDEX_URL is an experimental environment variable that changes the behavior of MLflow by redirecting all pip install commands to the given additional index URL. Note that artifacts from

mlflow runs and launches are always resolved from the tracking server and never from MLFLOW_EXTRA_PYPI_INDEX_URL.

Model serving does not take place in an experimental mode. This may change for future versions of the product so for use of a specific model uri after model serving becomes generally available.

MLflow can be configured to work against a remote tracking URI, which can be specified with the MLFLOW_TRACKING_URI environment variable. MLflow will store experiment and run information as well as code and model binaries to this URI. In Databricks, you can set this to an S3 bucket. MLflow will use the default credentials available on the cluster to read and write from the S3 bucket.

For example, you can add the following for Scala/Java in the notebook to set the environment variable:

```
import com.databricks.dbutils_v1.DBUtilsHolder.dbutils
val token = dbutils.notebook.getContext.notebookEnvironment.json.get("token_hash")
import scala.sys.process._
```

3. MLflow Components

MLflow is an excellent framework that consists of four main components. These components are essential for implementing efficient machine learning workflows and managing the entire lifecycle of models. Let's take a closer look at each of these components:

a. **MLflow Tracking**: As the name suggests, this component allows you to seamlessly record and query various experiments. You can conveniently store code, data, configuration details, and results related to your experiments. With MLflow Tracking, you can easily keep track of different iterations, compare their performances, and make well-informed decisions based on the obtained results.

b. **MLflow Projects**: This component introduces a convenient packaging format that ensures reproducible runs across any platform. By packaging your code, data, and configuration into an MLflow Project, you can ensure that your models can be easily replicated and deployed on any infrastructure. This helps in eliminating compatibility issues and streamlining the deployment process.

c. **MLflow Models**: Another crucial component is MLflow Models, which offers a unified and flexible model format. This format supports various deployment tools, allowing you to seamlessly transition your models to production environments. With MLflow Models, you can easily convert your trained models into a portable format and deploy them using tools like TensorFlow Serving, Microsoft Azure ML, or even on edge devices.

d. **MLflow Model Registry**: The final piece of the puzzle is the MLflow Model Registry. This centralized and collaborative component enables efficient model lifecycle management. You can register, manage, and version your models in a structured manner. The Model Registry allows easy collaboration between team members, ensuring smooth transitions between different stages of the model lifecycle.

4. MLFlow Tracking

Enabling MLflow on Databricks automatically records runs as you issue spark commands. Alternatively, you can use the MLflow tracking API to log runs from Python code. Runid is the unique identifier for a run and is generated by MLflow. Databricks also allows you to log the results of sparkML models, though the syntax to do this is slightly different from what is mentioned in the sparkML section of this document. If you are using Databricks, it is recommended to use DBFS to store the data to avoid issues with local filepaths. See these notebooks for further detail on logging from sparkML and TensorFlow to MLflow on Databricks.

MLflow Tracking is organized around the concept of an experiment. An experiment is a user-created collection of runs. For example, if you are comparing support vector machines and logistic regression, you can create a Databricks MLflow experiment to record runs from each one. Each run is a training process that produces a model. An MLflow run records the details of the experiment (parameters, code, and data), and the output (metrics, the model itself, files, and artifacts).

4.1. Experiment Tracking

Experiment tracking records information about each experiment run, enabling you to reproduce results and compare the effects of different parameters. All components of MLflow can log information to the same place,

so you can use it with any MLflow tracking server. Tracking is organized into runs where each run has a versioned set of parameters, metrics, and artifacts. The MLflowClient class provides an interface to the tracking server and supports recording and querying runs. See the API documentation for the complete interface. Tracking supports automatic logging from MLflow training integrations and manual logging from other libraries and tools. Automatic logging is demonstrated in later sections.

By default, each run records to a new directory within the currently active experiment as defined by mlflow.set_experiment. Previous runs are read only unless the default experiment is changed for your MLflowClient. You can create a new run via the start_run method and close out the current run by starting another or using end_run. All subsequent logging API calls will write to the most recently created run until you create a new one. use_run allows you to specify which run the client is interacting with by setting it as the active run. Artifacts and other files written by a run are associated with it and tracked to it through the file path and run id.[5]

4.2. Logging Parameters and Metrics

Once we have a sequence of runs for experiments you want to compare, MLflow allows you to log parameters, results, and models during these runs. This will then allow you to visualize and compare the results and quickly understand which parameters are leading to which results. With mlflow.logging in Python, use log_param to log a key-value pair on the run. This code logs the parameter alpha with 0.5. Use log_metric to log a single metric for the run. This code logs the loss.

Finally, use log_artifact to record a file or a directory, which will be associated with a run. Logged artifacts can be any file format, and are intended for large data like images, videos, machine learning models, etc. Their size and content need not be known in advance.

All of the above logging APIs record their information for the currently active run (if any). If there is no active run (for example, if your script is not yet calling start_run), each of these functions will start a new run before logging the information. This means that you can write a script to log parameters and metrics without changing your logic when runs are or are not already started. This script will always record its information to new runs. Note that mlflow.start_run can also take a nested with statement to activate a run for a contained block, automatically ending the run even in case of exceptions within the block.

4.3. Visualizing Experiment Results

What if we have complex machine learning with lots of parameters and we want to determine what is the best set of parameters that gives the best result for the model? It is hard to move back and forth changing the set of parameters, then run the model to see the result. Say no more, MLflow has provided the feature to do hyperparameter tuning. Currently, MLflow only supports grid search, but it should be good enough for a start.

Once we have lots of runs on the same set of parameters, we can try to determine what is the best set of parameters by visualizing the run results. For example, if we are doing a logistic regression, we can try to visualize the test accuracy result of the run by drawing a graph of test accuracy against the number of iterations. Then we can select the set of parameters which gives the best result for the model. To do this, you can use the Databricks Notebook to query the params and test accuracy result, then use the Databricks visualization to draw the graph.

4.4. Managing Runs and Artifacts

MLflow will automatically save the execution of each trial, and these results can be fetched through an API. Each run has a runID, which comes in handy when one needs to query results for an individual trial. Users who would like a list of all trials will be able to get some metadata about each by querying the experiment runs. This can be in the form of a pandas dataframe for ease of access. Logging is nice, but it may become expensive or unwieldy to log certain data. On Databricks, they have developed a nice display UI for the artifacts (attachments) you have from each run. In the future, there are hopes of implementing automatic artifact logging so users do not need to specify each individual artifact. Artifacts from the UI can be saved and loaded as a URL. This URL can then be shared with co-workers for easier collaboration. All these features combined allow for very easy storage and access to trial results. This can help greatly in speeding up the iterative process involved in machine learning.

5. MLFlow Projects

The important thing to understand is that the run of a project is recorded and results in an artifact, the code that produced the run. This means all the runs of a project can be collected to form an experiment which encapsulates the total code lifecycle of the project. This is not possible without an existing system and is typically difficult and time-consuming to set up for a single project. This is an advantage for those who do not wish to invest time to learn the intricacies of MLflow.

Environment: The project's code will usually have an important connection to an environment that it was tested or developed in. To maintain the repeatability of some code, an environment is a configuration specifying a set of hardware parameters and software packages. MLflow will enable users to run code in any specific environment configuration by providing the required environment information in a YAML file or as a Docker image.

Dependencies: Dependencies are any required language-specific packages or environment requirements needed to run the project. These often include Python packages such as numpy or pandas. Dependency information can be written into a conda.yaml file for projects using conda, or a requirements.txt file for all other projects. MLflow will then use this information to enact the appropriate environment before running the code.

Entry Points: An entry point is a place in the code where MLflow will use to trigger its run. This is typically a main function or a train script at the root of the projects directory. By providing entry points, these scripts become reusable by triggering them with the run command.

The key idea behind projects is to be able to write a project that describes its code using a project file and can easily be run by others without having to become experts on the code. To do this, the code must specify the following:

MLflow Projects are a standard format for packaging machine learning code in a reusable and reproducible way. Each project is simply a directory with code or a Git repository, and uses a descriptor file to define its dependencies and how to run the code. This lets you take any MLflow project and run it in a consistent way on any platform, whether that's your laptop or a cloud service, without translating code between development envionments, and it lets others run your code at the click of a button.

5.1. Packaging and Deploying Models

MLflow provides a convention for a machine learning project to package dependencies, and the model itself in a format that can be reproduced by other data scientists. This makes it possible to run the project in a consistent environment, and to compare results across runs. Consider the following project directory, which has the training code and a conda.yaml file:

```
project/
├── MLproject
├── conda.yaml
├── train.py
```

The MLproject file is a file that describes the project. It has a name, and a reference to the code that will run the project and produce the model. It can also have descriptions of parameters, and references to other packages that are part of the project.

Here is an example MLproject file:

```
name:example
entry_points:
main:
parameters:
data_file: string
regularize: {type: boolean, default: true}
alpha: {type: float, default: 0.1}
command: "python train.py -d {params.data_file} -r {params.regularize} -a {params.alpha}"
```

In this project, the default environment is a Conda environment. The conda.yaml file describes the Conda environment and all of the dependencies, including the runtime, of this project. When these files are present, running mlflow run <project> from the project's directory will run the project in a new conda environment with those dependencies. If the current state of the model is not logged, MLflow will record the parameters, metrics, and a summary of the execution of any project run invoked from the tracking URI, so it's easier to compare results across the projects and reproduce the results. Any artifact written to runs:/ will also be collected, and easily accessible as part of the run. Finally, MLflow will infer the environment in which the project ran and store the result in a file call MLmodel. This is a file that describes the project and the id of the run that produced the current version of the model. For conventions and examples of packaging other types of projects, see the documentation about the project format.[6]

5.2. Reproducible Model Training

MLFlow is solving a crucial problem by providing a reproducible machine learning solution. Direct integration with GitHub has made it possible to put ML code, data, and environment in one place. This can ensure that we can run the same code and get the same result in the future. MLFlow makes tracking experiments as easy as Keras. Represented is a simple decision tree model fit on iris data. A script is written to test various parameters and store the results. MLFlow can automatically infer dependencies. But to be specific, creating a conda.yaml file can ensure that the environment is reproducible.

Although this is a new project, I can see myself using this regularly in the future. In fact, other experiments that I have done in the past are sometimes too cumbersome to rerun when switching between projects. This is mainly because it can be difficult to remember or record in thorough detail all the steps taken to get to a certain result with machine learning, and previous tracking tools have often resulted in muddled or incomplete records. Simulation of data can also be time-consuming, so if we can guarantee that changing only one variable in code will not change the result, it can save time in some cases.

5.3. Integrating with Databricks Notebooks

The easiest way to integrate an MLflow project with Databricks notebooks is to use the MLflow Tracking API. With both the Databricks notebook and the MLflow project open, call mlflow.set_tracking_uri to point the MLflow client of the notebook at the experiment where the project's runs are being recorded. Then access the run by its run_id or other attributes.

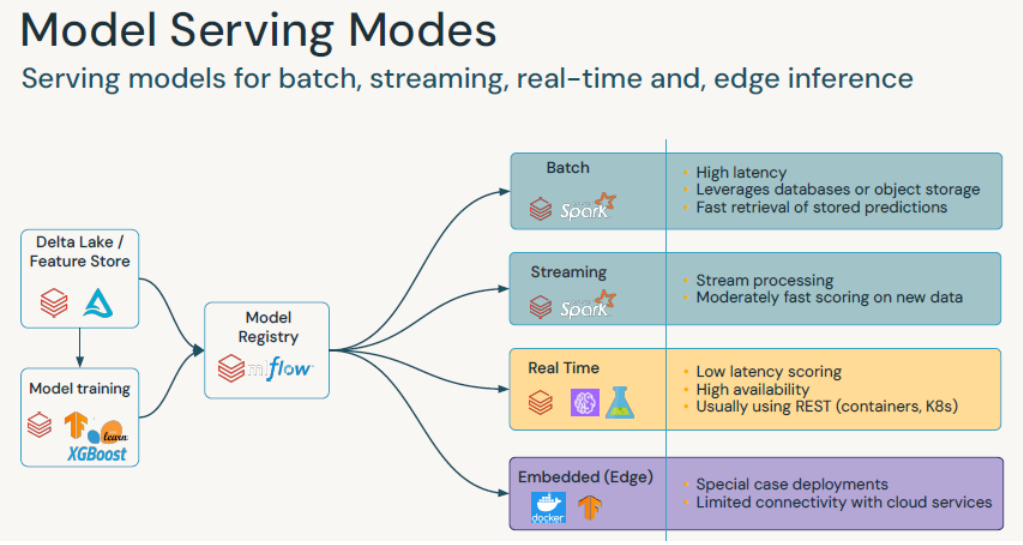
For example, use mlflow.search_runs to find and load runs from your notebook into a Pandas DataFrame, enabling further analysis or visualization. Also, structured data in MLflow runs, models, and artifacts can be easily loaded into Databricks tables, enabling a variety of advanced analytics. See Databricks File System (DBFS) and databases documentation for more information on importing data into Databricks.

The Databricks REST API has endpoints for Databricks Workspace and DBFS, enabling programmatic, auditable management of notebooks and libraries. Notebook workflows can also be orchestrated programmatically using Jobs and the REST API. With notebook and library management in source control with an MLflow project, that automation could be part of the project source code. For example, use dbfs cp to copy the project code and notebooks to/from the existing version control system into the Databricks Workspace. Easier still, depending on properties of the existing automation and the version control system, the Workspace can mount the VCS directly. That would facilitate a seamless integration between MLflow, notebooks, libraries, and a wide variety of source code available from the Databricks Workspace.[7]

This last section contains a boilerplate example specifying how to package a Python function into a library using setup tools, installing on a cluster, and running it in a notebook, which easily translates to doing the same with MLflow project code. An MLproject with train and predict entry points might be prototyped in a notebook, then quickly moved to cluster execution during iterative development. The MLflow project also acts as an advertisement for code portability, as it can be executed on a variety of environments. This is an appealing feature for users wanting to leverage their work in a variety of clusters and Databricks Runtime versions.

MLflow Models: Flexible deployment at any scaleModel Serving Modes: Serving models for batch, streaming, real-time, edge inference, and more Batch scoring: One-click deployment of models from the Model Registry to scalable compute clusters for batch scoring, streamlining your workflow and increasing efficiencyOnline

scoring: One-click deployment of models to REST endpoints for auto-scaling low latency scoring, allowing for real-time predictions and instantaneous results. With MLflow Models, you have the power to deploy and serve your models in a variety of ways, making it easier than ever to put your machine learning projects into production. Whether you need to serve models for batch processing, streaming data, real-time applications, or even edge devices, MLflow Models has you covered. And with the ability to deploy models with just one click, you can save time and effort, enabling faster iteration and deployment. So why wait? Start using MLflow Models today and take your machine learning deployment to new heights.



6. Challenges with building Real-time ML Systems

Most machine learning models don't make it to production. Machine learning infrastructure is incredibly difficult: Real-time ML systems necessitate rapid and scalable serving infrastructure, which can be expensive to construct and uphold.

Deploying real time models needs disparate tools: Data teams use diverse tools to develop models. These tools encompass a wide range of functionalities, such as data preparation, feature engineering, and model training. Additionally, customers often utilize separate platforms for managing data, implementing machine learning algorithms, and serving the models in production. While this approach allows for flexibility and specialization, it also introduces added complexity and cost. By adopting an integrated platform that consolidates these tasks, data teams can streamline their workflows, reduce overhead expenses, and enhance collaboration across the entire model development and deployment process. Such a unified solution simplifies the overall data pipeline, improves efficiency, and enables teams to focus on delivering valuable insights and impactful results.

Operating production ML requires expert resources:Steep learning curve of deployment tools has become a significant challenge for organizations in various industries. The complexity and intricacy associated with deploying models have resulted in a bottlenecked process that hampers the overall efficiency and effectiveness of model deployment. This bottleneck is primarily fueled by the limited engineering resources available, which severely restrict the scalability and expansion potential of deployment capabilities.

The process of model deployment requires a deep understanding of various tools and technologies which often necessitates a steep learning curve. As organizations strive to embrace cutting-edge technologies and innovative solutions, the complexity of these tools continues to grow exponentially. Consequently, engineers and data scientists often face numerous challenges, spending a significant amount of time learning and mastering these tools before they can efficiently deploy models.

The limitation of engineering resources further exacerbates the situation, hindering the ability to scale effectively. With a shortage of skilled professionals in this domain, organizations struggle to allocate the

necessary personnel to handle model deployment, leading to delays and reduced scalability. As a result, valuable time and effort are wasted, and the potential for growth and advancement is significantly diminished.

Addressing the bottleneck in model deployment requires a comprehensive approach that focuses on both reducing the steep learning curve and finding solutions to the limited engineering resources. Organizations must invest in training programs and initiatives that equip their personnel with the necessary skills and knowledge to navigate the intricacies of deployment tools. By providing adequate resources and support, organizations can empower their teams to become proficient and efficient model deployers.

Furthermore, addressing the limited engineering resources requires a strategic and proactive approach. Organizations can consider various options such as outsourcing certain aspects of model deployment, collaborating with external experts, or optimizing their existing workforce to maximize productivity. By expanding the pool of available resources, organizations can alleviate the strain on their engineering teams and create a more scalable and flexible system that supports growth and innovation.

MLflow Model Registry: Collaborative, centralized model hub that serves as a seamless platform for managing and organizing machine learning artifacts. With its advanced features, MLflow Model Registry enables the versioning of ML artifacts, streamlining the process of experimentation, testing, and production. By effortlessly integrating with approval and governance workflows, this model hub empowers teams to maintain control and ensure compliance in their projects.

Furthermore, MLflow Model Registry offers an audit log of stage transitions and requests, providing comprehensive visibility into the model's lifecycle. Alongside this, it facilitates an approval workflow for stage transitions, promoting a structured and collaborative environment. With its robust automation capabilities through CI/CD integration, MLflow Model Registry becomes an indispensable tool in the machine learning pipeline.[8]

7. Conclusion

In conclusion, MLflow, when used in combination with Databricks, emerges as a pivotal tool for augmenting the manageability and efficiency of machine learning (ML) lifecycle processes. The incorporation of MLflow's extensive features for tracking experiments, deploying models, facilitating collaboration, and centrally managing ML models directly addresses the significant challenges encountered throughout the ML lifecycle. This paper has demonstrated how MLflow on Databricks not only simplifies the tracking of experiments and deployment of ML models, but also ensures reproducibility and enhances collaboration by automating logging, standardizing project packaging, and establishing a robust model registry. Additionally, MLflow's flexibility in deployment across various environments, such as real-time and edge inference, underscores its applicability in a wide array of production scenarios. Despite obstacles like the high costs associated with infrastructure and the complexity of integrating diverse tools, MLflow's integration with the Databricks REST API and its project management capabilities offer effective solutions that streamline the iterative development and deployment of ML projects. The comprehensive overview provided by this research emphasizes the potential of MLflow on Databricks to significantly enhance the automation, collaboration, and overall effectiveness of ML workflows, rendering it an invaluable asset for data scientists and machine learning engineers seeking to navigate the intricacies of modern ML systems development and deployment.

References

[1]. "MLflow: The Complete Guide," www.run.ai. https://www.run.ai/guides/machine-learning-operations/mlflow#:~:text=MLflow%20is%20an%20open%20source

[2]. "MLflow Projects — MLflow 2.12.1 documentation," mlflow.org. https://mlflow.org/docs/latest/projects.html#overview

[3]. C. Yang et al., "MLife: a lite framework for machine learning lifecycle initialization," Machine Learning, Oct. 2021, doi: https://doi.org/10.1007/s10994-021-06052-0.

[4]. "What is MLflow? — MLflow 2.11.1 documentation," mlflow.org. https://mlflow.org/docs/latest/introduction/index.html

[5]. "Announcing Databricks Autologging for Automated ML Experiment Tracking," Databricks, 2021. https://www.databricks.com/blog/2021/08/27/announcing-databricks-autologging-for-automated-ml-experiment-tracking.html

[6]. "Automate Deployment and Testing with Databricks Notebook + MLflow," Databricks, 2020. https://www.databricks.com/blog/2020/01/16/automate-deployment-and-testing-with-databricks-notebook-mlflow.html

[7]. "How to Quickly Deploy, Test & Manage ML Models as REST Endpoints with Databricks," Databricks, 2020. https://www.databricks.com/blog/2020/11/02/quickly-deploy-test-and-manage-ml-models-as-rest-endpoints-with-mlflow-model-serving-on-databricks.html

[8]. "How to Share and Control ML Model Access with MLflow Model Registry," Databricks, Apr. 15, 2020. https://www.databricks.com/blog/2020/04/15/databricks-extends-mlflow-model-registry-with-enterprise-features.html