Log metrics, parameters and files with MLflow

Article • 10/12/2022 • 10 minutes to read

APPLIES TO: Python SDK azure-ai-ml **v2 (current)**

Select the version of Azure Machine Learning Python SDK you are using:

Azure Machine Learning supports logging and tracking experiments using MLflow Tracking . You can log models, metrics, parameters, and artifacts with MLflow as it supports local mode to cloud portability.

> **Important**
>
> Unlike the Azure Machine Learning SDK v1, there is no logging functionality in the Azure Machine Learning SDK for Python (v2). See this guidance to learn how to log with MLflow. If you were using Azure Machine Learning SDK v1 before, we recommend you to start leveraging MLflow for tracking experiments. See **Migrate logging from SDK v1 to MLflow** for specific guidance.

Logs can help you diagnose errors and warnings, or track performance metrics like parameters and model performance. In this article, you learn how to enable logging in the following scenarios:

- Log metrics, parameters and models when submitting jobs.
- Tracking runs when training interactively.
- Viewing diagnostic information about training.

> **Tip**
>
> This article shows you how to monitor the model training process. If you're interested in monitoring resource usage and events from Azure Machine learning, such as quotas, completed training jobs, or completed model deployments, see **Monitoring Azure Machine Learning**.

> **Tip**

For information on logging metrics in Azure Machine Learning designer, see **How to log metrics in the designer**.

Prerequisites

- You must have an Azure Machine Learning workspace. Create one if you don't have any.
- You must have `mlflow`, and `azureml-mlflow` packages installed. If you don't, use the following command to install them in your development environment:

  Bash

  ```
  pip install mlflow azureml-mlflow
  ```

- If you are doing remote tracking (tracking experiments running outside Azure Machine Learning), configure MLflow to track experiments using Azure Machine Learning. See Setup your tracking environment for more details.

Getting started

To log metrics, parameters, artifacts and models in your experiments in Azure Machine Learning using MLflow, just import MLflow in your training script:

Python

```
import mlflow
```

Configuring experiments

MLflow organizes the information in experiments and runs (in Azure Machine Learning, runs are called Jobs). There are some differences in how to configure them depending on how you are running your code:

Training interactively

When training interactively, such as in a Jupyter Notebook, use the following

pattern:

1. Create or set the active experiment.
2. Start the job.
3. Use logging methods to log metrics and other information.
4. End the job.

For example, the following code snippet demonstrates configuring the experiment, and then logging during a job:

Python

```
import mlflow
# Set the experiment
mlflow.set_experiment("mlflow-experiment")

# Start the run
mlflow_run = mlflow.start_run()
# Log metrics or other information
mlflow.log_metric('mymetric', 1)
# End run
mlflow.end_run()
```

Tip

Technically you don't have to call `start_run()` as a new run is created if one doesn't exist and you call a logging API. In that case, you can use `mlflow.active_run()` to retrieve the run once currently being used. For more information, see **mlflow.active_run()** .

You can also use the context manager paradigm:

Python

```
import mlflow
mlflow.set_experiment("mlflow-experiment")

# Start the run, log metrics, end the run
with mlflow.start_run() as run:
    # Run started when context manager is entered, and ended when con-
text manager exits
    mlflow.log_metric('mymetric', 1)
    mlflow.log_metric('anothermetric',1)
```

```
    pass
```

When you start a new run with `mlflow.start_run`, it may be useful to indicate the parameter `run_name` which will then translate to the name of the run in Azure Machine Learning user interface and help you identify the run quicker:

Python

```
with mlflow.start_run(run_name="iris-classifier-random-forest") as run:
    mlflow.log_metric('mymetric', 1)
    mlflow.log_metric('anothermetric',1)
```

For more information on MLflow logging APIs, see the MLflow reference .

Logging parameters

MLflow supports the logging parameters used by your experiments. Parameters can be of any type, and can be logged using the following syntax:

Python

```
mlflow.log_param("num_epochs", 20)
```

MLflow also offers a convenient way to log multiple parameters by indicating all of them using a dictionary. Several frameworks can also pass parameters to models using dictionaries and hence this is a convenient way to log them in the experiment.

Python

```
params = {
    "num_epochs": 20,
    "dropout_rate": .6,
    "objective": "binary_crossentropy"
}

mlflow.log_params(params)
```

Note

Azure ML SDK v1 logging can't log parameters. We recommend the use of MLflow for tracking experiments as it offers a superior set of features.

Logging metrics

Metrics, as opposite to parameters, are always numeric. The following table describes how to log specific numeric types:

Logged Value	Example code	Notes
Log a numeric value (int or float)	`mlflow.log_metric("my_metric", 1)`	
Log a numeric value (int or float) over time	`mlflow.log_metric("my_metric", 1, step=1)`	Use parameter `step` to indicate the step at which you are logging the metric value. It can be any integer number. It defaults to zero.
Log a boolean value	`mlflow.log_metric("my_metric", 0)`	0 = True, 1 = False

ⓘ Important

Performance considerations: If you need to log multiple metrics (or multiple values for the same metric) avoid making calls to `mlflow.log_metric` in loops. Better performance can be achieved by logging batch of metrics. Use the method `mlflow.log_metrics` which accepts a dictionary with all the metrics you want to log at once or use `mlflow.log_batch` which accepts multiple type of elements for logging.

Logging curves or list of values

Curves (or list of numeric values) can be logged with MLflow by logging the same metric multiple times. The following example shows how to do it:

Python

```
list_to_log = [1, 2, 3, 2, 1, 2, 3, 2, 1]
from mlflow.entities import Metric
from mlflow.tracking import MlflowClient
import time

client = MlflowClient()
client.log_batch(mlflow.active_run().info.run_id,
                 metrics=[Metric(key="sample_list", value=val, time-
stamp=int(time.time() * 1000), step=0) for val in list_to_log])
```

Logging images

MLflow supports two ways of logging images:

Logged Value	Example code	Notes
Log numpy metrics or PIL image objects	`mlflow.log_image(img, "figure.png")`	`img` should be an instance of `numpy.ndarray` or `PIL.Image.Image`. `figure.png` is the name of the artifact that will be generated inside of the run. It doesn't have to be an existing file.
Log matlotlib plot or image file	`mlflow.log_figure(fig, "figure.png")`	`figure.png` is the name of the artifact that will be generated inside of the run. It doesn't have to be an existing file.

Logging other types of data

Logged Value	Example code	Notes
Log text in a text file	`mlflow.log_text("text string", "notes.txt")`	Text is persisted inside of the run in a text file with name `notes.txt`.
Log dictionaries as `JSON` and `YAML` files	`mlflow.log_dict(dictionary, "file.yaml"`	`dictionary` is a dictionary object containing all the structure that you want to persist as `JSON` or `YAML` file.
Log a trivial file already existing	`mlflow.log_artifact("path/to/file.pkl")`	Files are always logged in the root of the run. If `artifact_path` is provided, then the file is logged in a folder as indicated in that parameter.

Logged Value	Example code	Notes
Log all the artifacts in an existing folder	`mlflow.log_artifacts("path/to/folder")`	Folder structure is copied to the run, but the root folder indicated is not included.

Logging models

MLflow introduces the concept of "models" as a way to package all the artifacts required for a given model to function. Models in MLflow are always a folder with an arbitrary number of files, depending on the framework used to generate the model. Logging models has the advantage of tracking all the elements of the model as a single entity that can be **registered** and then **deployed**. On top of that, MLflow models enjoy the benefit of no-code deployment and can be used with the Responsible AI dashboard in studio. Read the article From artifacts to models in MLflow for more information.

To save the model from a training run, use the `log_model()` API for the framework you're working with. For example, mlflow.sklearn.log_model() . For more details about how to log MLflow models see Logging MLflow models For migrating existing models to MLflow, see Convert custom models to MLflow.

Automatic logging

With Azure Machine Learning and MLflow, users can log metrics, model parameters and model artifacts automatically when training a model. Each framework decides what to track automatically for you. A variety of popular machine learning libraries are supported. Learn more about Automatic logging with MLflow .

To enable automatic logging insert the following code before your training code:

Python

```
mlflow.autolog()
```

Tip

You can control what gets automatically logged wit autolog. For instance, if you indicate `mlflow.autolog(log_models=False)`, MLflow will log everything but models

for you. Such control is useful in cases where you want to log models manually but still enjoy automatic logging of metrics and parameters. Also notice that some frameworks may disable automatic logging of models if the trained model goes behond specific boundaries. Such behavior depends on the flavor used and we recommend you to view they documentation if this is your case.

View jobs/runs information with MLflow

You can view the logged information using MLflow through the MLflow.entities.Run object:

Python

```
import mlflow

run = mlflow.get_run(run_id="<RUN_ID>")
```

You can view the metrics, parameters, and tags for the run in the data field of the run object.

Python

```
metrics = run.data.metrics
params = run.data.params
tags = run.data.tags
```

Note

The metrics dictionary returned by `mlflow.get_run` or `mlflow.seach_runs` only returns the most recently logged value for a given metric name. For example, if you log a metric called `iteration` multiple times with values, `1`, then `2`, then `3`, then `4`, only `4` is returned when calling `run.data.metrics['iteration']`.

To get all metrics logged for a particular metric name, you can use `MlFlowClient.get_metric_history()` as explained in the example **Getting params and metrics from a run**.

> **Tip**
>
> MLflow can retrieve metrics and parameters from multiple runs at the same time, allowing for quick comparisons across multiple trials. Learn about this in **Query & compare experiments and runs with MLflow**.

Any artifact logged by a run can be queried by MLflow. Artifacts can't be accessed using the run object itself and the MLflow client should be used instead:

Python

```
client = mlflow.tracking.MlflowClient()
client.list_artifacts("<RUN_ID>")
```

The method above will list all the artifacts logged in the run, but they will remain stored in the artifacts store (Azure ML storage). To download any of them, use the method `download_artifact`:

Python

```
file_path = client.download_artifacts("<RUN_ID>", path="feature_impor-
tance_weight.png")
```

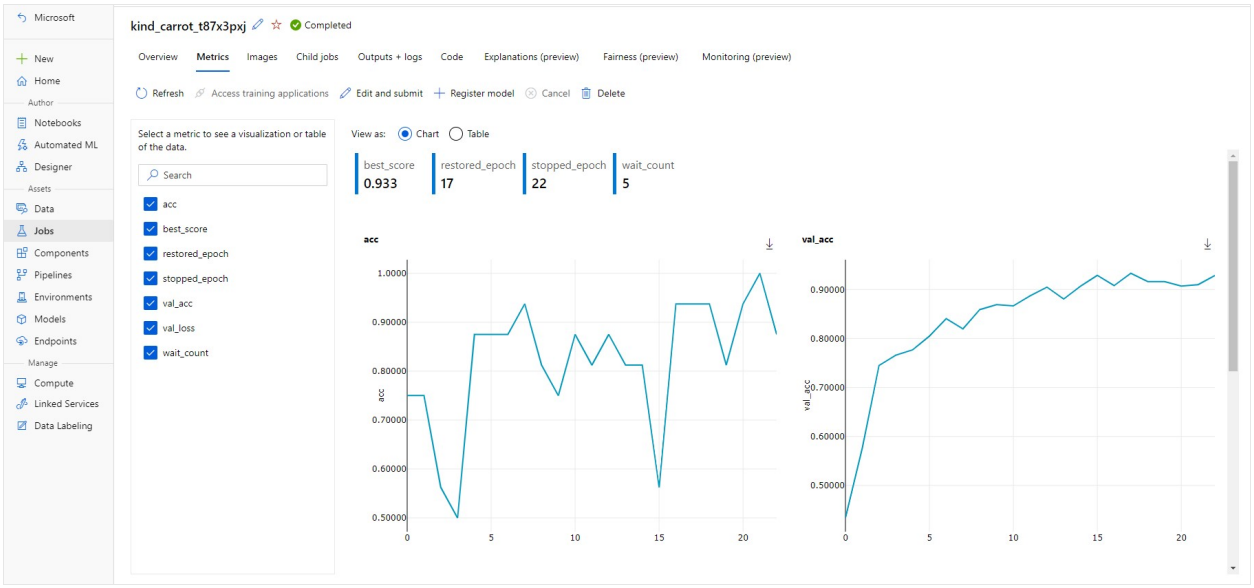
For more information please refer to Getting metrics, parameters, artifacts and models.

View jobs/runs information in the studio

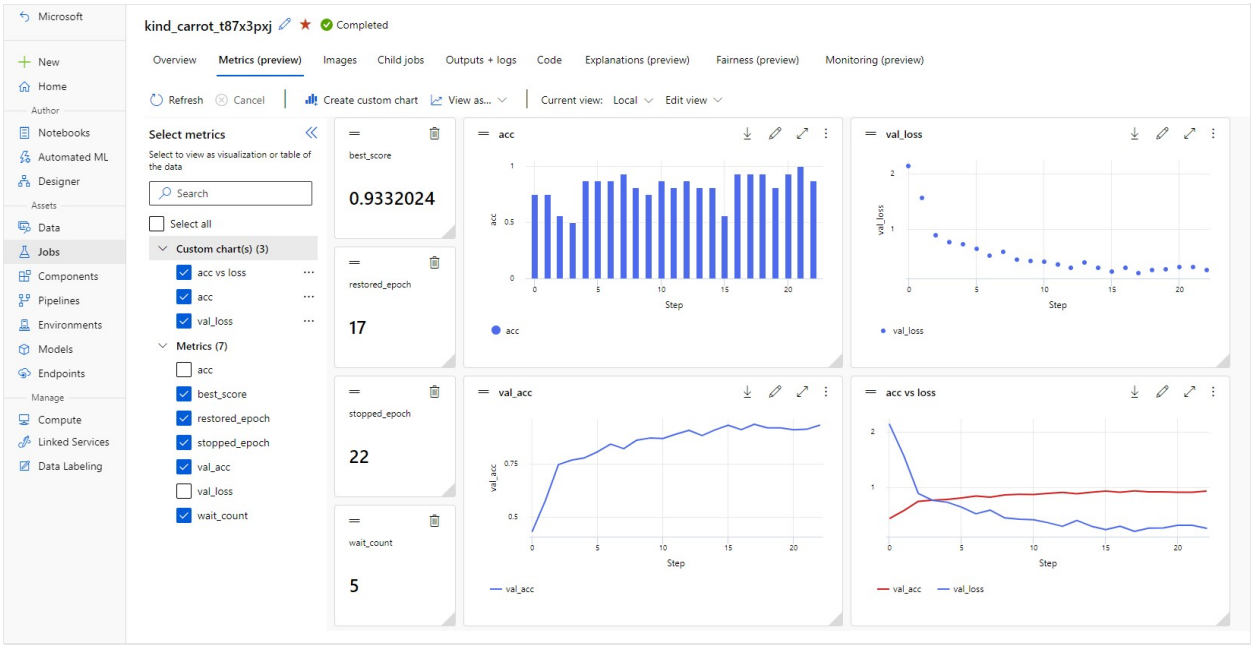
You can browse completed job records, including logged metrics, in the Azure Machine Learning studio .

Navigate to the **Jobs** tab. To view all your jobs in your Workspace across Experiments, select the **All jobs** tab. You can drill down on jobs for specific Experiments by applying the Experiment filter in the top menu bar. Click on the job of interest to enter the details view, and then select the **Metrics** tab.

Select the logged metrics to render charts on the right side.



For a customizable view of your job metrics (preview), use the preview panel to enable the feature. Once enabled, you can add/remove charts and customize them by applying smoothing, changing the color, or plotting multiple metrics on a single graph. You can also resize and rearrange the layout as you wish. Once you have created your desired view, you can save it for future use and share it with your teammates using a direct link.

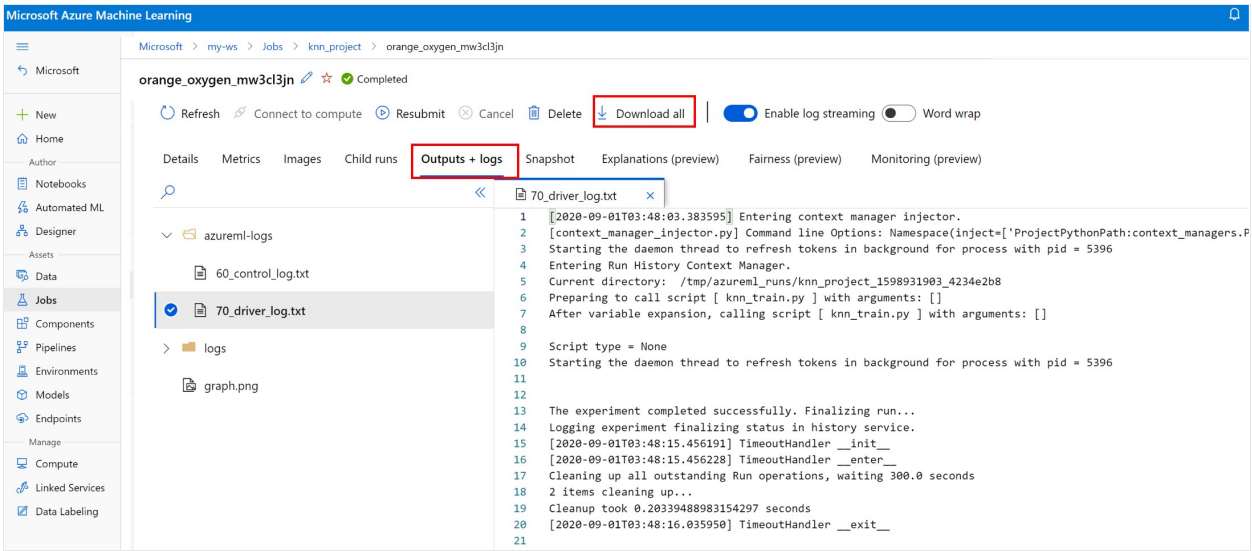


View and download diagnostic logs

Log files are an essential resource for debugging the Azure ML workloads. After submitting a training job, drill down to a specific run to view its logs and outputs:

1. Navigate to the **Jobs** tab.

2. Select the runID for a specific run.
3. Select **Outputs and logs** at the top of the page.
4. Select **Download all** to download all your logs into a zip folder.
5. You can also download individual log files by choosing the log file and selecting **Download**



user_logs folder

This folder contains information about the user generated logs. This folder is open by default, and the **std_log.txt** log is selected. The **std_log.txt** is where your code's logs (for example, print statements) show up. This file contains `stdout` log and `stderr` logs from your control script and training script, one per process. In most cases, you'll monitor the logs here.

system_logs folder

This folder contains the logs generated by Azure Machine Learning and it will be closed by default. The logs generated by the system are grouped into different folders, based on the stage of the job in the runtime.

Other folders

For jobs training on multi-compute clusters, logs are present for each node IP. The structure for each node is the same as single node jobs. There's one more logs folder for overall execution, stderr, and stdout logs.

Azure Machine Learning logs information from various sources during training, such as AutoML or the Docker container that runs the training job. Many of these logs aren't documented. If you encounter problems and contact Microsoft support, they may be able to use these logs during troubleshooting.

Next steps

- Train ML models with MLflow and Azure Machine Learning.
- Migrate from SDK v1 logging to MLflow tracking.