



I have a model TAR in S3 bucket as a result of Model Training pipeline in SageMaker. How do I convert the TAR ball trained model to MLFlow + Databricks without rerunning the model training pipeline.

You can wrap the existing SageMaker artifact as an MLflow model and then register it on Databricks; you don't need to retrain, but you do need to write a one-time "adapter" that knows how to load and invoke the TAR contents.^[1] ^[2]

Below is the typical pattern.

1. Understand what's inside the TAR

First you need to know what framework the SageMaker artifact uses and how it expects inputs.

- Download and inspect the TAR locally or in a throwaway notebook (untar and look for model files like `model.pkl`, `xgboost-model`, `pytorch_model.bin`, `model.joblib`, `tensorflow/saved_model.pb`, or a custom `inference.py`).
- Identify: framework (`sklearn` / `XGBoost` / `PyTorch` / `TF` / custom), Python version, and how inference is done today (SageMaker `inference.py` or default container entrypoint).

This will drive whether you can use a built-in MLflow flavor (`sklearn`, `xgboost`, `pytorch`, `tensorflow`) or need a custom `pyfunc` wrapper.^[3] ^[1]

2. Implement an MLflow `pyfunc` wrapper around the TAR

You create a small Python project that:

- Downloads the SageMaker TAR from S3.
- Extracts it to a temp directory.
- Loads the underlying model (e.g., with `joblib.load`, `xgboost.Booster().load_model`, `torch.load`, or by importing the SageMaker `inference.py`).
- Implements a `predict` method that takes a pandas DataFrame or numpy array and returns predictions.

Example skeleton:

```
import os
import tarfile
```

```

import tempfile
import boto3
import mlflow
import pandas as pd
from mlflow.pyfunc import PythonModel

S3_BUCKET = "your-bucket"
S3_KEY = "path/to/model.tar.gz"

class SageMakerTarModel(PythonModel):
    def load_context(self, context):
        local_dir = tempfile.mkdtemp()
        local_tar = os.path.join(local_dir, "model.tar.gz")

        s3 = boto3.client("s3")
        s3.download_file(S3_BUCKET, S3_KEY, local_tar)

        with tarfile.open(local_tar, "r:gz") as tar:
            tar.extractall(local_dir)

        # TODO: inspect extracted files and load your real model here
        # Example for sklearn:
        # import joblib
        # self.model = joblib.load(os.path.join(local_dir, "model.joblib"))
        self.model = ... # load real model

    def predict(self, context, model_input: pd.DataFrame):
        # Adapt to your model's expected input
        return self.model.predict(model_input)

```

This class is the “bridge” that lets MLflow treat your existing TAR as a normal MLflow model.^[2]
[\[3\]](#)

3. Log the wrapped model to MLflow (pointing at Databricks)

Run the wrapper once in an environment that can talk to Databricks MLflow tracking (can be a local script, ECS task, or a Databricks notebook).

Configure MLflow to use Databricks:

```

export MLFLOW_TRACKING_URI=databricks
export DATABRICKS_HOST="https://<your-databricks-workspace-url>"
export DATABRICKS_TOKEN="<personal-access-token>"

```

Then in Python:

```

import mlflow
import mlflow.pyfunc

mlflow.set_experiment("/Shared/sagemaker-imports") # or a UC experiment in Databricks

with mlflow.start_run() as run:
    mlflow.pyfunc.log_model(

```

```

        artifact_path="model",
        python_model=SageMakerTarModel(),
        registered_model_name=None # we'll register explicitly after
    )

    run_id = run.info.run_id
    print("Logged run:", run_id)

```

This will upload a full MLflow model artifact to Databricks' MLflow store (the artifact will still pull the real weights from S3 at load time unless you choose to re-upload them into the MLflow artifact).[\[4\]](#) [\[1\]](#)

If you prefer the TAR content to live inside Databricks instead of S3, you can modify `load_context` to read from `context.artifacts` and log the TAR as an MLflow artifact once.

4. Register the model in Databricks MLflow / Unity Catalog

Once the run has the logged MLflow model, you register it in the Databricks Model Registry (ideally in Unity Catalog).

Programmatically:

```

from mlflow import MlflowClient

client = MlflowClient()

model_uri = f"runs:{run_id}/model" # artifact_path from log_model
mv = mlflow.register_model(
    model_uri=model_uri,
    name="sagemaker_imported_model" # if using UC: "catalog.schema.sagemaker_imported_mc"
)

print("Registered:", mv.name, "version:", mv.version)

```

Or via UI: open the run in Databricks, expand Artifacts → model, click “Register model”, and create a new name.[\[1\]](#) [\[2\]](#)

If your workspace is already using Unity Catalog for models, use the fully qualified UC name (for example `main.ml_models.sagemaker_imported_model`) so it's governed and shareable.[\[5\]](#) [\[6\]](#)

5. Serve or batch-run from Databricks

Once registered, you can:

- Use Databricks Model Serving to deploy a REST endpoint backed by this MLflow model.
- Use Databricks Jobs to run batch inference notebooks that load the model via
`mlflow.pyfunc.load_model("models:/name/Production")`.

From your Java/SpringBoot ecosystem, you'd typically call the Databricks Served Model endpoint over HTTPS (similar to how you would call a SageMaker endpoint), rather than embedding the MLflow model directly in Java.[\[7\]](#) [\[1\]](#)

6. Notes on limitations and tradeoffs

- You're not "converting" the original SageMaker model format; you're wrapping it so MLflow can manage it and Databricks can serve it. The underlying binaries stay the same.
- If your SageMaker artifact has custom preprocessing logic in `inference.py`, you should port that into the `predict` method (or import and call it) so behavior matches your existing pipeline.
- Later, if you retrain models directly in Databricks, you can drop the TAR indirection and log native MLflow models instead, but this bridge lets you migrate gradually.

If you tell me: (a) what framework is inside the TAR (sklearn / XGBoost / PyTorch / TF / something else) and (b) what the extracted file structure looks like, I can give you a concrete `SageMakerTarModel.load_context` and `predict` implementation tailored to your artifact.

*
*

1. <https://learn.microsoft.com/en-us/azure/databricks/mlflow/models>
2. <https://mlflow.org/docs/latest/ml/model-registry/>
3. <https://mlflow.org/docs/latest/ml/>
4. <https://stackoverflow.com/questions/77681583/how-to-register-a-model-in-databricks-mlflow-registry-from-outside-of-the-databr>
5. <https://learn.microsoft.com/en-us/azure/databricks/machine-learning/manage-model-lifecycle/>
6. <https://docs.databricks.com/aws/en/machine-learning/manage-model-lifecycle/>
7. <https://docs.databricks.com/aws/en/machine-learning/mlops/mlops-workflow>
8. https://github.com/aws-samples/sagemaker-studio-mlflow-integration/blob/main/lab/1_mlflow-admin-lab.ipynb
9. https://sagemaker-examples.readthedocs.io/en/latest/sagemaker-mlflow/sagemaker_deployment_mlflow.html
10. <https://www.youtube.com/watch?v=DqVvUmRgB5k>
11. <https://stackoverflow.com/questions/76259858/getting-artifacts-from-a-registered-model-in-mlflow>
12. <https://mlflow.org/docs/latest/ml/deployment/deploy-model-to-sagemaker/>
13. <https://notes.kodekloud.com/docs/Fundamentals-of-MLOps/Model-Development-and-Training/Demo-MLflow-Model-Artifact-and-Versioning>
14. <https://community.databricks.com/t5/machine-learning/register-model-mounted-in-s3/td-p/73638>
15. <https://mlflow.org/docs/latest/deployment/deploy-model-to-sagemaker/>
16. <https://docs.aws.amazon.com/sagemaker/latest/dg/mlflow-track-experiments-model-registration.html>
17. <https://docs.aws.amazon.com/sagemaker/latest/dg/mlflow-track-experiments-model-deployment.html>
18. <https://stackoverflow.com/questions/71237899/register-sagemaker-model-in-mlflow>