

MLflow

Experiment tracking, model registry, and serving for ML and LLM workflows.

Last reviewed 28 Sep 2026 ·

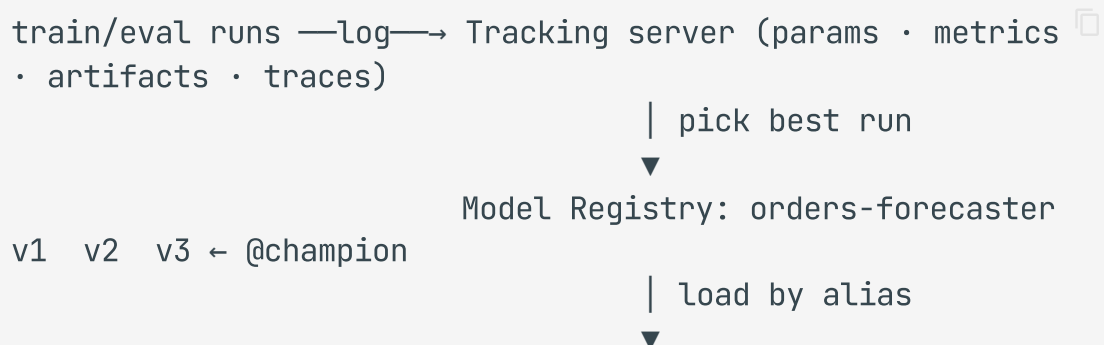
Prerequisites: [Python for DE](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/python-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/python-reference/)

Related: [Databricks](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/data-bricks-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/data-bricks-reference/) · [Fine-Tuning](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/fine-tuning/) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/fine-tuning/) · [Evals](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/eval-and-evals/) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/eval-and-evals/) · [Glossary](https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/) (https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/)

Overview

Challenge: Machine learning work produces many experiments — training runs with different parameters, data versions, and results — spread across notebooks and machines. Without tracking, it becomes impossible to say which run produced the production model, what data it used, or how to reproduce it.

Solution: MLflow is an open-source system of record for ML and LLM work. It records every run (parameters, metrics, code version, artifacts), stores models in a standard format, maintains a *registry* of model versions with aliases such as `@champion`, and serves models as REST endpoints. Recent versions also trace LLM calls and evaluate GenAI applications.



batch scoring job • REST endpoint • Spark
UDF • Databricks serving

Relevance to data engineering: data engineers typically operate the tracking server, integrate model scoring into pipelines (load `@champion`, score the latest partition), record data versions alongside models for lineage, and trigger retraining when data drifts.



On this page

Basic - [What Is MLflow](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#what-is-mlflow) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#what-is-mlflow) - [Setup](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#setup) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#setup) - [Tracking Experiments](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#tracking-experiments) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#tracking-experiments) - [Logging Parameters, Metrics, Artifacts](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#logging-parameters-metrics-artifacts) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#logging-parameters-metrics-artifacts)

Intermediate - [MLflow Projects](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#mlflow-projects) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#mlflow-projects) - [Model Registry](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#model-registry) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#model-registry) - [MLflow with Scikit-learn & PySpark](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#mlflow-with-scikit-learn--pyspark) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#mlflow-with-scikit-learn--pyspark) - [MLflow with LLMs](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#mlflow-with-llms) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#mlflow-with-llms)

Advanced - [Model Serving](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#model-serving) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#model-serving) - [Custom Python Models](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#custom-python-models) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#custom-python-models) - [MLflow in Databricks](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#mlflow-in-databricks) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#mlflow-in-databricks) - [DE Integration Patterns](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#de-integration-patterns) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#de-integration-patterns)

Reference - [Common Pitfalls](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#common-pitfalls) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#common-pitfalls) - [Cheat Sheet](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#cheat-sheet) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#cheat-sheet) - [Interview Questions](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#interview-questions) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#interview-questions) - [Further Reading](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#further-reading) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/mlflow/#further-reading)

What Is MLflow

MLflow is an open-source platform for managing the ML lifecycle:

Component	What it does
Tracking	Log parameters, metrics, code, and artifacts per experiment run
Projects	Package ML code for reproducible execution
Models	Standard format for packaging models for deployment
Registry	Versioned model store; aliases (e.g. <code>@champion</code>) mark which version is live
Serving	REST endpoint for serving models

Typical workflow:

- Train model (multiple runs, different hyperparameters)
- Track each run in MLflow (params, metrics, artifacts)
- Pick the best run
- Register model in the Model Registry
- Point the "champion" alias at the new version
- Serve via MLflow serve or integrate into pipeline

Setup

```
pip install mlflow
```

```
# Start the MLflow UI (local)
```

```
mlflow ui --port 5000
```

```
# Open http://localhost:5000
```

```
# Or point to a remote tracking server
```

```
export MLFLOW_TRACKING_URI=http://my-mlflow-server:5000
```

```
import mlflow
```

```
# Configure tracking URI (default: ./mlruns in current directory)
```

```
mlflow.set_tracking_uri("http://localhost:5000") # remote server
```

```
mlflow.set_tracking_uri("sqlite:///mlflow.db") # local SQLite
```

```
mlflow.set_tracking_uri("file:///path/to/mlruns")      #  
local filesystem  
  
# Set the experiment (creates it if it doesn't exist)  
mlflow.set_experiment("orders-prediction")
```

Tracking Experiments

```
import mlflow  
import numpy as np  
  
mlflow.set_experiment("orders-forecasting")  
  
# Context manager approach (recommended)  
with mlflow.start_run(run_name="xgboost-v1") as run:  
    print(f"Run ID: {run.info.run_id}")  
  
    # ... train model, compute metrics ...  
    mlflow.log_param("learning_rate", 0.01)  
    mlflow.log_metric("rmse", 142.3)  
    mlflow.log_artifact("feature_importance.png")  
  
# Manual approach  
run = mlflow.start_run(run_name="xgboost-v2")  
mlflow.log_param("learning_rate", 0.05)  
mlflow.end_run()  
  
# Nested runs (hyperparameter search)  
with mlflow.start_run(run_name="hyperparam-search") as  
parent:  
    for lr in [0.001, 0.01, 0.1]:  
        with mlflow.start_run(run_name=f"lr={lr}",  
nested=True):  
            mlflow.log_param("learning_rate", lr)  
            rmse = train_and_eval(lr)  
            mlflow.log_metric("rmse", rmse)
```

Logging Parameters, Metrics, Artifacts

```
with mlflow.start_run():  
  
    # — Parameters (hyperparameters, config)  
    

---

  
    mlflow.log_param("model_type", "xgboost")  
    mlflow.log_param("learning_rate", 0.01)  
    mlflow.log_param("max_depth", 6)  
    mlflow.log_param("n_estimators", 100)  
  
    # Log dict of params at once  
    mlflow.log_params({  
        "model_type": "xgboost",  
        "learning_rate": 0.01,  
        "max_depth": 6,  
    })  
  
    # — Metrics (scalar values over time)  
    

---

  
    mlflow.log_metric("train_rmse", 120.4)  
    mlflow.log_metric("val_rmse", 142.3)  
    mlflow.log_metric("r2", 0.87)  
  
    # Metrics over steps (for training curves)  
    for epoch in range(100):  
        train_loss = train_one_epoch()  
        val_loss = validate()  
        mlflow.log_metric("train_loss", train_loss,  
step=epoch)  
        mlflow.log_metric("val_loss", val_loss,  
step=epoch)  
  
    # Log dict of metrics  
    mlflow.log_metrics({"rmse": 142.3, "mae": 98.1, "r2":  
0.87})  
  
    # — Artifacts (files)  
    

---

  
    mlflow.log_artifact("feature_importance.png")  
    # single file  
    mlflow.log_artifacts("./output/")
```

```
# entire directory
mlflow.log_artifact("model_config.yaml", "configs")
# into a subfolder (no trailing slash)

# Log in-memory objects
import json
with open("/tmp/metrics.json", "w") as f:
    json.dump({"rmse": 142.3}, f)
mlflow.log_artifact("/tmp/metrics.json")

# — Tags

mlflow.set_tag("team", "data-science")
mlflow.set_tag("data_version", "2024-03-15")
mlflow.set_tag("git_commit", "abc1234")
mlflow.set_tags({"env": "dev", "validated": "false"})

# — Model

mlflow.sklearn.log_model(model, name="model")
```

MLflow Projects

Package your training code for reproducibility.

```
# MLproject file
name: orders-forecasting

conda_env: conda.yaml # or pip_env: requirements.txt

entry_points:
  main:
    parameters:
      learning_rate: {type: float, default: 0.01}
      max_depth:     {type: int,   default: 6}
      data_path:     {type: str,   default: "s3://my-
bucket/data/"}
```

```
command: "python train.py --lr {learning_rate} --depth {max_depth} --data {data_path}"
```

```
evaluate:
```

```
parameters:
```

```
model_uri: {type: str}
```

```
command: "python evaluate.py --model {model_uri}"
```

```
# Run locally
```

```
mlflow run . -P learning_rate=0.05
```

```
# Run from GitHub
```

```
mlflow run https://github.com/org/project -P  
learning_rate=0.05
```

```
# Run on Databricks
```

```
mlflow run . --backend databricks --backend-config  
cluster.json
```

Model Registry

```
import mlflow
```

```
from mlflow.tracking import MlflowClient
```

```
client = MlflowClient()
```

```
# — Register a model
```

```
# Option 1: register at log time
```

```
with mlflow.start_run():
```

```
    info = mlflow.sklearn.log_model(  
        model,  
        name="model",  
        registered_model_name="orders-forecaster"  
    )
```

```
# Option 2: register a model that is already logged
```

```
mlflow.register_model(model_uri=info.model_uri,
```

```

name="orders-forecaster")

# — List versions


---



for v in client.search_model_versions("name='orders-forecaster'"):
    print(f"Version {v.version}: aliases={v.aliases}, run={v.run_id}")

# — Promote with aliases


---



# Stages (Staging/Production) are deprecated since MLflow 2.9 – use aliases instead.
# An alias is a movable pointer to one version; deployments load by alias.
client.set_registered_model_alias("orders-forecaster", "challenger", version=3)

# After validation, point "champion" at the new version (the old one is simply un-aliased)
client.set_registered_model_alias("orders-forecaster", "champion", version=3)

# Roll back = move the alias back
client.set_registered_model_alias("orders-forecaster", "champion", version=2)

# — Load a registered model


---



# By alias
model = mlflow.pyfunc.load_model("models:/orders-forecaster@champion")

# By version
model = mlflow.pyfunc.load_model("models:/orders-forecaster/3")

# Predict
predictions = model.predict(X_test)

# — Add descriptions and tags

```

```
client.update_registered_model(  
    name="orders-forecaster",  
    description="XGBoost model trained on 3 years of order  
history"  
)  
client.set_model_version_tag("orders-forecaster", "3",  
    "validated_by", "alice")
```

MLflow with Scikit-learn & PySpark

— Scikit-learn autolog 

```
from sklearn.ensemble import GradientBoostingRegressor  
from sklearn.model_selection import train_test_split  
  
mlflow.sklearn.autolog() # logs params, metrics, model  
automatically  
  
with mlflow.start_run():  
    model = GradientBoostingRegressor(n_estimators=100,  
learning_rate=0.05)  
    model.fit(X_train, y_train)  
    # params, metrics, and model logged automatically
```

— PySpark MLlib

```
from pyspark.ml.regression import GBRegressor  
from pyspark.ml import Pipeline  
  
mlflow.spark.autolog()  
  
with mlflow.start_run():  
    gbt = GBRegressor(featuresCol="features",  
labelCol="label")  
    pipeline = Pipeline(stages=[gbt])  
    model = pipeline.fit(train_df)  
    # Spark ML pipeline logged automatically
```

```
# Log Spark model explicitly
mlflow.spark.log_model(model, "spark-model")
loaded = mlflow.spark.load_model(f"runs://{run_id}/spark-model")
```

MLflow with LLMs

Tracing LLM calls (MLflow 2.18+ / 3.x)

```
import mlflow
import anthropic

mlflow.set_experiment("de-rag-assistant")
mlflow.anthropic.autolog()      # every Anthropic SDK call
                                # is traced: prompt, response, tokens, latency

client = anthropic.Anthropic()
response = client.messages.create(
    model="claude-sonnet-5-5",
    max_tokens=1024,
    messages=[{"role": "user", "content": "What does the
orders DAG load?"}],
)
# Traces appear in the MLflow UI under the experiment's
# "Traces" tab.
# Equivalent autologging exists for OpenAI (mlflow.openai),
# LangChain, and LlamaIndex.
```

Logging LLM experiment results

```
# Log LLM experiment results
with mlflow.start_run(run_name="rag-eval-v2"):
    mlflow.log_params({
        "llm_model": "claude-sonnet-5-5",
        "embedding_model": "text-embedding-3-small",
        "chunk_size": 512,
        "chunk_overlap": 50,
        "retrieval_k": 5,
        "reranking": True,
```

```

    })

    mlflow.log_metrics({
        "faithfulness":      0.92,
        "relevance":         0.88,
        "context_precision": 0.85,
        "avg_latency_ms":    840,
        "avg_cost_usd":      0.0021,
    })

    # Log eval results as artifact
    import json
    with open("/tmp/eval_results.json", "w") as f:
        json.dump(eval_results, f, indent=2)
    mlflow.log_artifact("/tmp/eval_results.json", "evals")

    # Tag the experiment
    mlflow.set_tags({
        "rag_version": "v2",
        "knowledge_base_date": "2024-03-15",
    })

    # Compare runs in UI – filter by parameters, sort by
    metrics

```

Model Serving

```

# Serve a registered model (--env-manager local reuses the
current environment;
# the default builds a fresh virtualenv for the model,
which takes longer)
mlflow models serve -m "models:/orders-forecaster@champion"
-p 5001 --env-manager local

# Serve a model by URI from a run
mlflow models serve -m "models:/m-<model_id>" -p 5001 --
env-manager local

# Test it
curl http://localhost:5001/invocations \

```

```
-H "Content-Type: application/json" \  
-d '{"dataframe_records": [{"feature1": 1.0, "feature2":  
2.0}]}'
```

```
# Serve programmatically and call from Python  
import requests  
import pandas as pd  
  
def predict_via_mlflow(features: pd.DataFrame, endpoint:  
str = "http://localhost:5001") -> list:  
    payload = {"dataframe_records":  
features.to_dict(orient="records")}  
    response = requests.post(f"{endpoint}/invocations",  
                             headers={"Content-Type":  
"application/json"},  
                             json=payload)  
    response.raise_for_status()  
    return response.json()["predictions"]
```

Custom Python Models

Package any Python logic as an MLflow model.

```
import mlflow.pyfunc  
  
class RAGModel(mlflow.pyfunc.PythonModel):  
    """Wrap a RAG pipeline as an MLflow model for  
serving."""  
  
    def load_context(self, context):  
        """Called once when the model is loaded."""  
        import pickle  
        with open(context.artifacts["vector_index"], "rb")  
as f:  
            self.index = pickle.load(f)  
        import anthropic  
        self.client = anthropic.Anthropic()  
  
    def predict(self, context, model_input):
```

```

"""
model_input: pd.DataFrame with a 'question' column
returns: pd.Series of answers
"""

import pandas as pd
answers = []
for question in model_input["question"]:
    chunks = self.index.search(question, k=5)
    context_text = "\n".join(c["text"] for c in
chunks)

    resp = self.client.messages.create(
        model="claude-haiku-4-5-20251001",
        max_tokens=512,
        messages=[{"role": "user", "content":
f"Context: {context_text}\n\nQ: {question}"}]
    )
    answers.append(next(b.text for b in
resp.content if b.type == "text"))
return pd.Series(answers)

# Log and register
import pickle
with open("/tmp/vector_index.pkl", "wb") as f:
    pickle.dump(my_index, f)

with mlflow.start_run():
    mlflow.pyfunc.log_model(
        name="rag_model",
        python_model=RAGModel(),
        artifacts={"vector_index":
"/tmp/vector_index.pkl"},
        registered_model_name="de-rag-assistant"
    )

# Load and use
model = mlflow.pyfunc.load_model("models:/de-rag-
assistant@champion")
import pandas as pd
results = model.predict(pd.DataFrame({"question": ["What is
the orders schema?"]})))

```

MLflow in Databricks

Databricks has MLflow built in — no setup required.

```
# In a Databricks notebook
import mlflow

# Tracking server is auto-configured
# Experiments live in /Users/<email>/my-experiment
mlflow.set_experiment("/Users/alice@example.com/orders-forecasting")

with mlflow.start_run():
    mlflow.log_param("model", "xgboost")
    mlflow.log_metric("rmse", 142.3)
    mlflow.sklearn.log_model(model, name="model",

registered_model_name="orders-forecaster")

# Unity Catalog model registry (Databricks Unity Catalog)
mlflow.set_registry_uri("databricks-uc")

mlflow.sklearn.log_model(
    model, name="model",

registered_model_name="main.ml_models.orders_forecaster"
# catalog.schema.model
)

# Load from Unity Catalog
model =
mlflow.pyfunc.load_model("models:/main.ml_models.orders_for
ecaster@champion")
```

DE Integration Patterns

Log pipeline metadata alongside model metrics

```
with mlflow.start_run(run_name="model-training-2024-03-15"):
```

```

# Model metrics
mlflow.log_metrics({"rmse": 142.3, "r2": 0.87})

# Data pipeline metadata – ties the model to the data
it was trained on
mlflow.log_params({
    "training_data_path":
"s3://bucket/gold/orders/2024-03-15/",
    "training_data_rows": 1_523_400,
    "feature_pipeline_ver": "v2.3",
    "data_cutoff_date": "2024-03-14",
})
mlflow.set_tags({
    "airflow_run_id": "scheduled__2024-03-15T02:00:00",
    "dbt_job_id": "1234",
    "data_team": "platform",
})

```

Trigger retraining when data drifts

```

# In Airflow or a monitoring pipeline:
def check_model_drift_and_retrain(**context):
    from scipy import stats
    import mlflow

    client = mlflow.MlflowClient()
    prod_model = client.get_model_version_by_alias("orders-
forecaster", "champion")

    # Compare current data distribution to training data
distribution
    current_data = get_recent_features()
    training_dist =
mlflow.artifacts.load_dict(f"runs://{prod_model.run_id}/feat
ure_stats.json")

    p_value = stats.ks_2samp(current_data["amount"],
training_dist["amount_sample"]).pvalue
    if p_value < 0.05:
        print(f"Data drift detected (p={p_value:.4f}) –
triggering retraining")

```

```
# Trigger Airflow DAG or Databricks job
trigger_retraining_pipeline()
```

Common Pitfalls

Pitfall	Symptom	Fix
Using the default local <code>./mlruns</code> store in a team	Runs scattered across laptops; nothing shared	A tracking server with a database backend and object-storage artifacts (or managed MLflow)
Loading models by stage (<code>models:/name/Production</code>)	Deprecation warnings; unclear promotion history	Aliases (<code>@champion</code> , <code>@challenger</code>) with <code>set_registered_model_alias</code>
Not logging the data version	Can't reproduce or explain a model	Log the dataset path, snapshot/version (Delta/Iceberg), and row counts as params or datasets
No signature or input example on logged models	Serving fails on schema mismatches	<code>infer_signature(X, y)</code> and <code>input_example</code> when logging
Environment not captured	The model loads locally but not in serving	Let MLflow record <code>requirements.txt</code> /conda env; pin versions
Huge artifacts logged every run	Storage costs balloon; the UI slows down	Log only what you need; lifecycle rules on the artifact store
Sklearn tree models fail to log with an untrusted-types error	MLflow 3 saves sklearn models with the <code>skops</code> format by default, which refuses types it has not been told to trust (for example <code>sklearn.tree._tree.Tree</code>)	Review the listed types and pass them as <code>skops_trusted_types=[...]</code> ; <code>serialization_format="cloudpickle"</code> works but can execute code when loaded
Pickled custom models relying on local code	<code>ModuleNotFoundError</code> at load time	Package code with <code>code_paths</code> , or use a models-from-code approach

Pitfall	Symptom	Fix
Scoring with whatever model is newest	Unvalidated models reach production	Promote via alias only after automated evaluation passes

Cheat Sheet

Task	Code
Point to a server	<code>mlflow.set_tracking_uri("http://mlflow:5000")</code>
Choose an experiment	<code>mlflow.set_experiment("orders-forecasting")</code>
Start a run	<code>with mlflow.start_run(run_name="xgb-v3"):</code>
Log params / metrics	<code>mlflow.log_params({...}) · mlflow.log_metric("rmse", 12.3, step=epoch)</code>
Log files	<code>mlflow.log_artifact("report.html") · mlflow.log_dict(d, "stats.json")</code>
Autolog a framework	<code>mlflow.sklearn.autolog() · mlflow.xgboost.autolog() · mlflow.spark.autolog()</code>
Trace LLM calls	<code>mlflow.anthropic.autolog() · mlflow.openai.autolog() · mlflow.langchain.autolog()</code>
Log and register a model	<code>mlflow.sklearn.log_model(model, name="model", registered_model_name="orders-forecaster", signature=sig)</code>
Promote	<code>MlflowClient().set_registered_model_alias("orders-forecaster", "champion", version=3)</code>
Load	<code>mlflow.pyfunc.load_model("models:/orders-forecaster@champion")</code>
Score in Spark	<code>mlflow.pyfunc.spark_udf(spark, "models:/orders-forecaster@champion")</code>
Serve locally	<code>mlflow models serve -m "models:/orders-forecaster@champion" -p 5001 --env-manager local</code>

Task	Code
Find the best run	<code>mlflow.search_runs(experiment_names=["x"], order_by=["metrics.rmse ASC"], max_results=1)</code>
Start a server	<code>mlflow server --backend-store-uri postgresql://... --artifacts-destination s3://bucket/mlflow</code>

Model URIs: `models:/<model_id>` (MLflow 3 logged model, returned as `model_uri` by `log_model`) · `runs:/<run_id>/<artifact_path>` (artifact logged to a run) · `models:/name/3` (version) · `models:/name@champion` (alias) · on Databricks with Unity Catalog: `models:/catalog.schema.name@champion`

Interview Questions

Q: What are the main components of MLflow? A: Tracking (logging runs with parameters, metrics, artifacts, and — in recent versions — LLM traces), Models (a standard packaging format with "flavors" such as sklearn, PyTorch, and a generic `pyfunc` interface), the Model Registry (versioned models with aliases, tags, and descriptions), and serving and deployment tools. Projects package code for reproducible runs, and recent releases add GenAI evaluation and prompt management.

Q: How do you promote a model to production with MLflow? A: Register each candidate as a new version of a registered model, run automated validation (metrics above the current champion's, no regressions on key slices, signature checks), and then move an alias such as `@champion` to the new version. Consumers always load `models:/name@champion`, so promotion and rollback are just moving the alias — no code change. Record who approved it and why with tags. Older MLflow used stages (`Staging / Production`), which are now deprecated.

Q: How would you integrate MLflow into a data pipeline? A: The training pipeline logs parameters, metrics, the data snapshot version, and the model, then registers a new version. A validation step compares it with the champion and moves the alias if it's better. A scoring pipeline (Airflow or Databricks job) loads `@champion`, scores the new partition in batch (for example with `spark_udf`), and writes predictions along with the model version for lineage. Monitoring jobs compare incoming feature distributions with training statistics and trigger retraining on drift.

Q: What is the `pyfunc` flavor and why does it matter? A: `pyfunc` is MLflow's generic Python model interface: any model logged with MLflow can be loaded as a `pyfunc` and called with `.predict()` on a DataFrame, whatever library trained it. That gives serving, batch scoring, and Spark UDFs one consistent API, and custom `PythonModel` classes let you wrap anything — including a RAG chain calling an LLM — in the same format.

Further Reading

- [MLflow documentation](https://mlflow.org/docs/latest/index.html) (https://mlflow.org/docs/latest/index.html)
- [Model Registry and aliases](https://mlflow.org/docs/latest/ml/model-registry/) (https://mlflow.org/docs/latest/ml/model-registry/)
- [MLflow Tracing for GenAI](https://mlflow.org/docs/latest/genai/tracing/) (https://mlflow.org/docs/latest/genai/tracing/)
- [MLflow on Databricks](https://docs.databricks.com/en/mlflow/index.html) (https://docs.databricks.com/en/mlflow/index.html)
- *Designing Machine Learning Systems* — Chip Huyen (O'Reilly). The broader MLOps picture.

Previous: [Evals](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/eval-and-evals/) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/eval-and-evals/) ·

Next: [Claude Code](https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/) (https://sarangambekar1997.github.io/data-engineering-handbook/07-ai/claude-code/) ·

Back to: [Index](https://sarangambekar1997.github.io/data-engineering-handbook/) (https://sarangambekar1997.github.io/data-engineering-handbook/)

 September 28, 2026