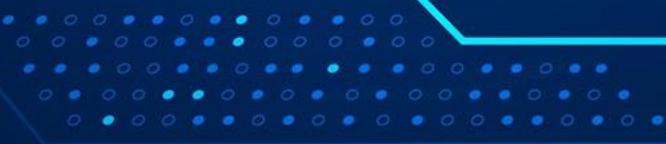


Deploying ML models in Production

Part 3: Model Management and Delivery

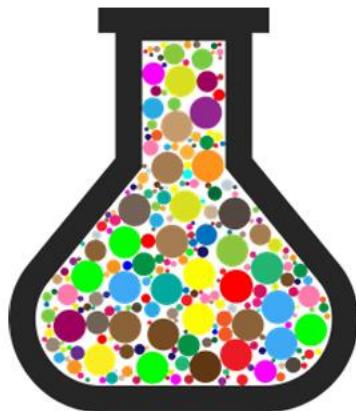
Ramin Toosi





Why experiment tracking?

- ▶ ML projects have far more branching and experimentation
- ▶ Debugging in ML is difficult and time consuming
- ▶ Small changes can lead to drastic changes in a model's performance and resource requirements
- ▶ Running experiments can be time consuming and expensive





What does it mean to track experiments?

- ▶ Enable you to duplicate a result
- ▶ Enable you to meaningfully compare experiments
- ▶ Manage code/data versions, hyperparameters, environment, metrics
- ▶ Organize them in a meaningful way
- ▶ Make them available to access and collaborate on within your organization



Simple Experiments with Notebooks

- ▶ Notebooks are great tools
- ▶ Notebook code is usually not promoted to production
- ▶ Tools for managing notebook code
 - ▶ nbconvert (.ipynb -> .py conversion)
 - ▶ nbdime (diffing)
 - ▶ jupytext (conversion+versioning)
 - ▶ neptune-notebooks (versioning+diffing+sharing)

```
+ Code + Text Copy to Drive  
Import packages  
We import necessary packages,  
[ ] import os  
import pprint  
import tempfile  
import urllib  
  
import absl  
import tensorflow as tf  
import tensorflow_model_research as tfr  
tf.get_logger().propagate = 0  
pp = pprint.PrettyPrinter()
```



Smoke testing for Notebooks

```
jupyter nbconvert --to script train_model.ipynb python  
train_model.py;  
  
python train_model.py
```



Not Just One Big File

- ▶ Modular code, not monolithic
- ▶ Collections of interdependent and versioned files
- ▶ Directory hierarchies or monorepos
- ▶ Code repositories and commits





Tracking Runtime Parameters

Config files

```
data:  
  train_path:  
    '/path/to/my/train.csv'  
  valid_path:  
    '/path/to/my/valid.csv'  
  
model:  
  objective:  
    'binary'  metric:  
    'auc'  
  learning_rate: 0.1  
  num_boost_round: 200  
  num_leaves: 60  
  feature_fraction: 0.2
```

Command line

```
python train_evaluate.py \  
  --train_path '/path/to/my/train.csv' \  
  --valid_path '/path/to/my/valid.csv' \  
  --objective 'binary' \  
  --metric 'auc' \  
  --learning_rate 0.1 \  
  --num_boost_round 200 \  
  --num_leaves 60 \  
  --feature_fraction 0.2
```



Log Runtime Parameters

```
parser = argparse.ArgumentParser()
parser.add_argument('--number_trees')
parser.add_argument('--learning_rate')
args = parser.parse_args()

neptune.create_experiment(params=vars(args))
...
# experiment logic
...
```



Data Versioning

- ▶ Data reflects the world, and the world changes
- ▶ Experimental changes include changes in data
- ▶ Tracking, understanding, comparing, and duplicating experiments includes data





Tools for Data Versioning

- ▶ Neptune
- ▶ Pachyderm
- ▶ Delta Lake
- ▶ Git LFS
- ▶ Dolt
- ▶ LakeFS
- ▶ DVC
- ▶ ML-Metadata



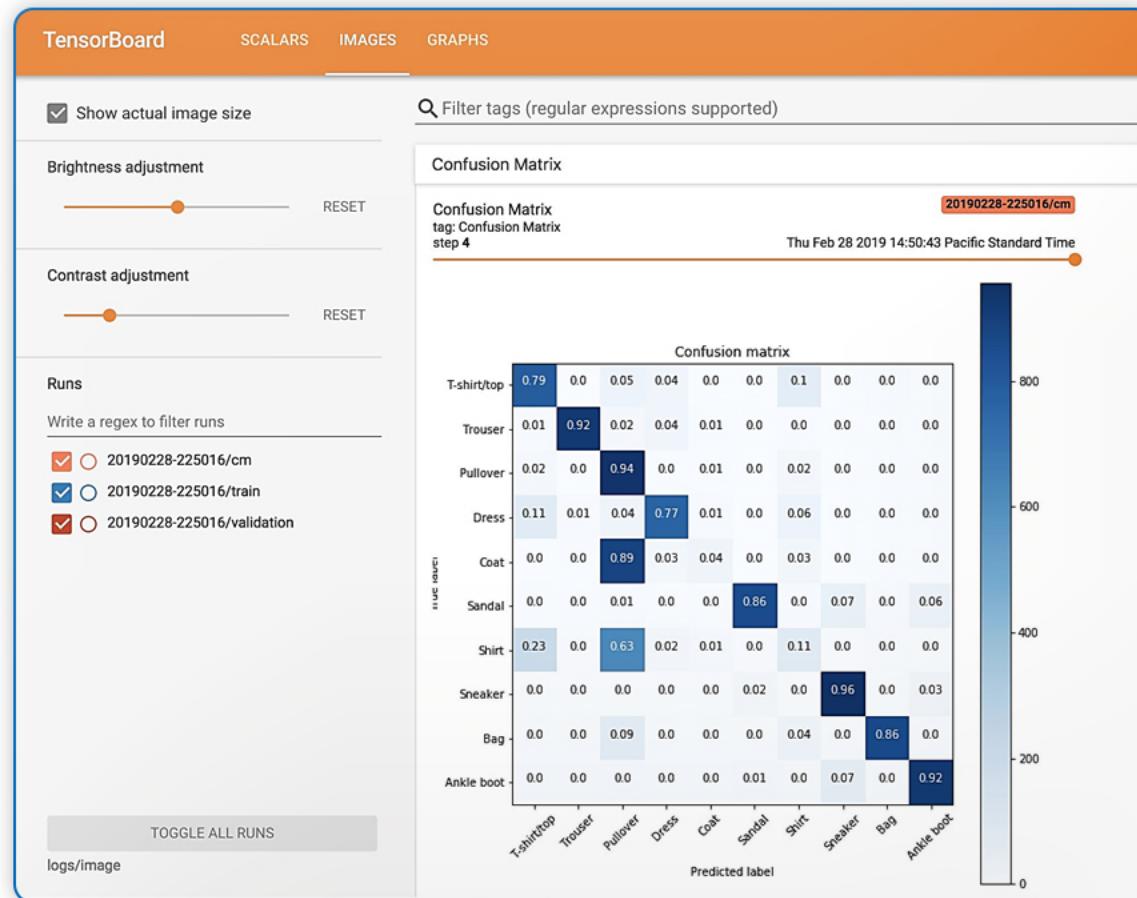


Experiment tracking to compare results

	Name (50 visualized)	Tags	acc	Sweep	optimizer	epoch	batch_size	n_train	n_valid	n_conv_lay	loss	GPU
-	batch 64 4 GPU	4GPU b_64_c	0.4305	-	rmsprop	49	64	5000	800	1	1.632	-
-	batch 64 (V2, 5K train)	2GPU b_64_c	0.4343	-	rmsprop	49	64	5000	800	1	1.63	-
□	50K examples (b 64)		0.4042	-	rmsprop	49	64	50000	8000	1	1.76	-
-	batch 32 4 GPU	4GPU b_32_c	0.4032	-	rmsprop	49	32	5000	800	1	1.714	-
-	batch 64 1 GPU	1GPU GCP	0.4465	-	rmsprop	49	64	5000	800	5	1.615	1
-	batch 128 (5K train)	2GPU b_128	0.4181	-	rmsprop	49	128	5000	800	1	1.658	-
-	batch 256 4 GPU	4GPU keras	0.3882	-	rmsprop	49	256	5000	800	1	1.751	-



Example: Logging metrics using TensorBoard





Organizing model development

- ▶ Search through & visualize all experiments
- ▶ Organize into something digestible
- ▶ Make data shareable and accessible
- ▶ Tag and add notes that will be meaningful to your team





Tooling for Teams

Display mode: Mosaic One in a row Filter charts: iter

valid_iter_auc

The chart displays the validation AUC (Area Under the Curve) over 800 iterations for several CRED models. The y-axis ranges from 0.600 to 0.750. Most models reach a plateau quickly, with CRED-70 reaching the highest AUC of approximately 0.775 by iteration 100. CRED-93 reaches a plateau slightly later at approximately 0.765.

Model	Iteration	valid_iter_auc
CRED-70	0	0.750
CRED-70	100	0.775
CRED-70	200	0.775
CRED-70	400	0.775
CRED-70	800	0.775
CRED-80	0	0.750
CRED-80	100	0.775
CRED-80	200	0.775
CRED-80	400	0.775
CRED-80	800	0.775
CRED-83	0	0.750
CRED-83	100	0.775
CRED-83	200	0.775
CRED-83	400	0.775
CRED-83	800	0.775
CRED-85	0	0.750
CRED-85	100	0.775
CRED-85	200	0.775
CRED-85	400	0.775
CRED-85	800	0.775
CRED-89	0	0.750
CRED-89	100	0.775
CRED-89	200	0.775
CRED-89	400	0.775
CRED-89	800	0.775
CRED-91	0	0.750
CRED-91	100	0.775
CRED-91	200	0.775
CRED-91	400	0.775
CRED-91	800	0.775
CRED-92	0	0.750
CRED-92	100	0.775
CRED-92	200	0.775
CRED-92	400	0.775
CRED-92	800	0.775
CRED-93	0	0.750
CRED-93	100	0.775
CRED-93	200	0.775
CRED-93	400	0.775
CRED-93	800	0.775

Short ID Owner Tags valid_auc train_auc features... train_spli... valid_spli...

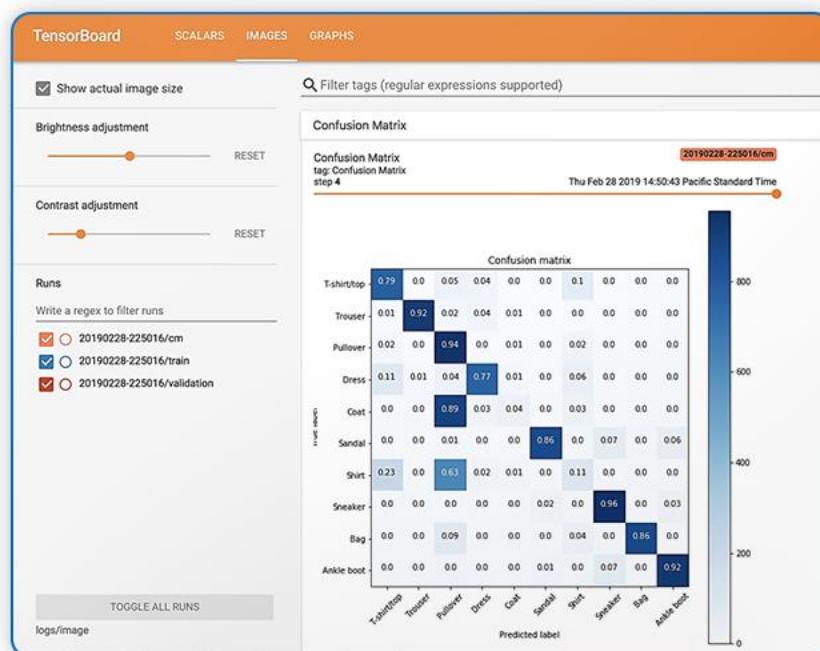
Short ID	Owner	Tags	valid_auc	train_auc	features...	train_spli...	valid_spli...
CRED-93	kamil	lgbm features_v1	0.773536	0.807047	e1f5a473fe...	41bf231b7...	4b61d05c0...
CRED-92	kamil	lgbm features_v1	0.765647	0.790213	e1f5a473fe...	41bf231b7...	4b61d05c0...
CRED-91	kamil	lgbm features_v1	0.767254	0.78022	e1f5a473fe...	41bf231b7...	4b61d05c0...
CRED-89	kamil	lgbm features_v1	0.772704	0.788251	e1f5a473fe...	41bf231b7...	4b61d05c0...
CRED-85	jakub-czakon	lgbm features_v1	0.775187	0.797085	e1f5a473fe...	41bf231b7...	4b61d05c0...



Tooling for Teams

► Vertex TensorBoard

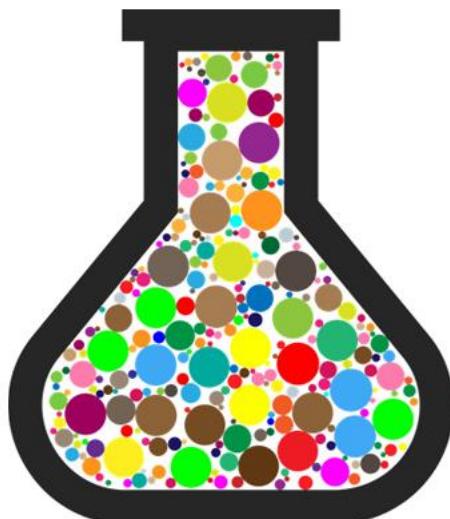
- Managed service with enterprise-grade security, privacy, and compliance
- Persistent, shareable link to your experiment dashboard
- Searchable list of all experiments in a project





Experiments are iterative in nature

- ▶ Creative iterations for ML experimentation
- ▶ Define a baseline approach
- ▶ Develop, implement, and evaluate to get metrics
- ▶ Assess the results, and decide on next steps
- ▶ Latency, cost, fairness, etc.

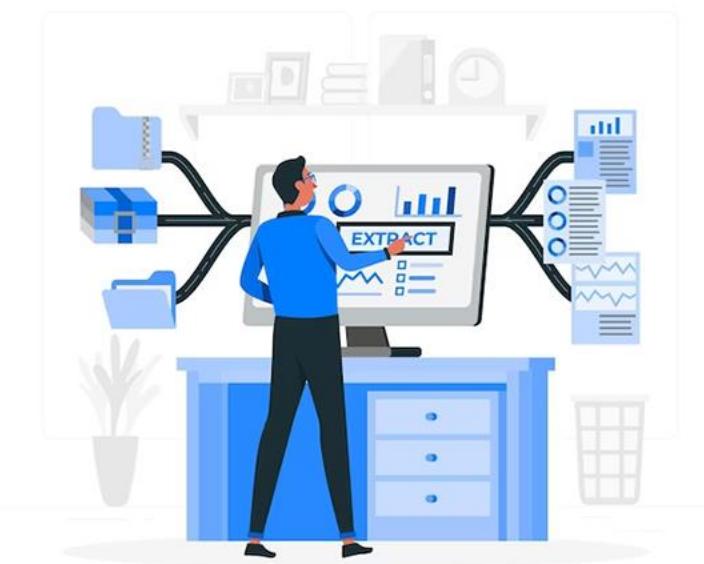




Data Scientists vs. Software Engineers

▶ Data Scientists

- ▶ Often work on fixed datasets
- ▶ Focused on model metrics
- ▶ Prototyping on Jupyter notebooks
- ▶ Expert in modeling techniques and feature engineering
- ▶ Model size, cost, latency, and fairness are often ignored





Data Scientists vs. Software Engineers

► Software Engineers

- ▶ Build a product
- ▶ Concerned about cost, performance, stability, schedule
- ▶ Identify quality through customer satisfaction
- ▶ Must scale solution, handle large amounts of data
- ▶ Detect and handle error conditions, preferably automatically
- ▶ Consider requirements for security, safety, fairness
- ▶ Maintain, evolve, and extend the product over long periods





Growing Need for ML in Products and Services

- ▶ Large datasets
- ▶ Inexpensive on-demand compute resources
- ▶ Increasingly powerful accelerators for ML
- ▶ Rapid advances in many ML research fields
(such as computer vision, natural language understanding, and recommendations systems)
- ▶ Businesses are investing in their data science teams and ML capabilities to develop predictive models that can deliver business value to their customers



Key problems affecting ML efforts today

► We've been here before

- ▶ In the 90s, Software Engineering was siloed
- ▶ Weak version control, CI/CD didn't exist
- ▶ Software was slow to ship; now it ships in minutes
- ▶ Is that ML today?

► Today's perspective

- ▶ Models blocked before deployment
- ▶ Slow to market
- ▶ Manual tracking
- ▶ No reproducibility or provenance
- ▶ Inefficient collaboration
- ▶ Unmonitored models

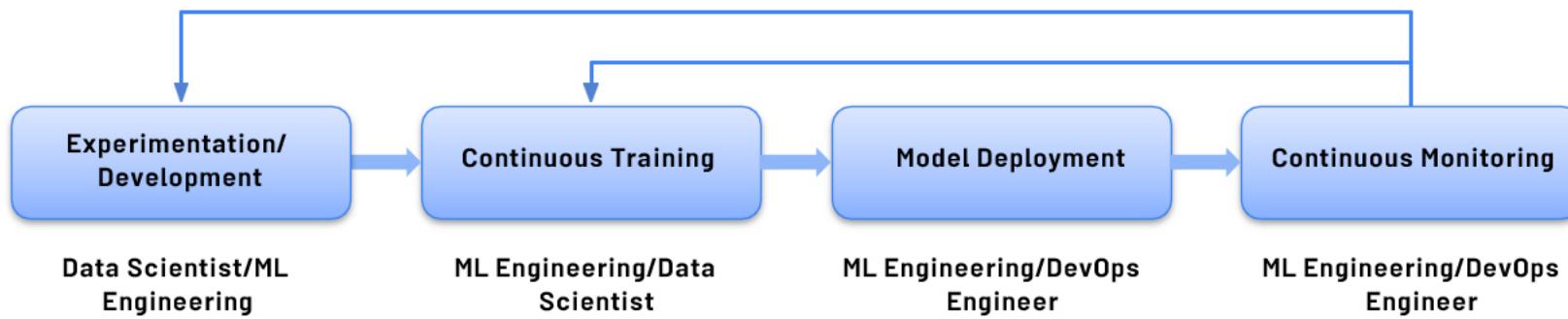


Bridging ML and IT with MLOps

- ▶ **Continuous Integration (CI):** Testing and validating code, components, data, data schemas, and models
- ▶ **Continuous Delivery (CD):** Not only about deploying a single software package or a service, but a system which automatically deploys another service (model prediction service)
- ▶ **Continuous Training (CT):** A new process, unique to ML systems, that automatically retrains candidate models for testing and serving
- ▶ **Continuous Monitoring (CM):** Catching errors in production systems, and monitoring production inference data and model performance metrics tied to business outcomes



ML Solution Lifecycle



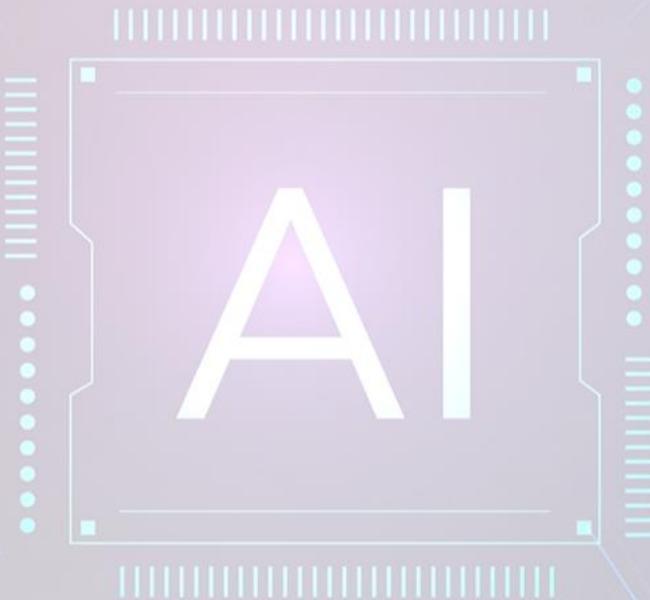


Standardizing ML processes with MLOps

- ▶ **ML Lifecycle Management**
- ▶ **Model Versioning & Iteration**
- ▶ **Model Monitoring and Management**
- ▶ **Model Governance**
- ▶ **Model Security**
- ▶ **Model Discovery**



MLOps level 0





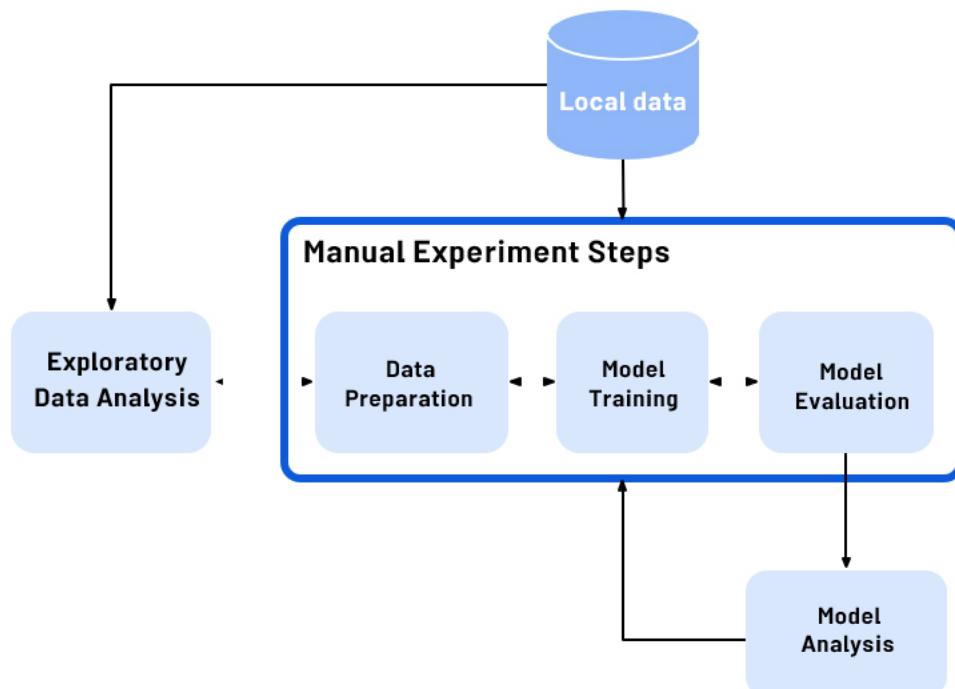
What defines an MLOps process' maturity?

- ▶ The level of **automation** of ML pipelines determines the maturity of the MLOps process
- ▶ As maturity increases, the available **velocity** for the training and deployment of new models also increases
- ▶ Goal is to automate training and deployment of ML models into the core software system, and provide monitoring



MLOps level 0: Manual process

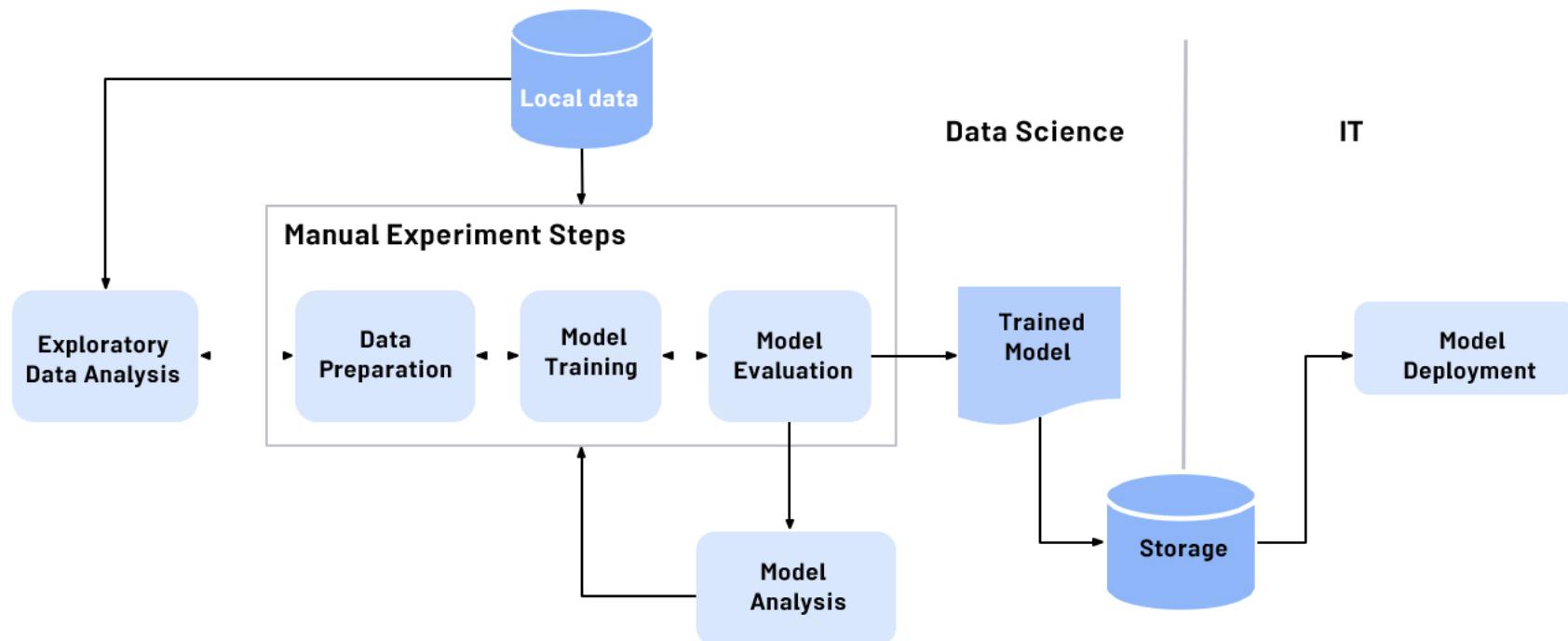
- ▶ Manual, script-driven, interactive





MLOps level 0: Manual process

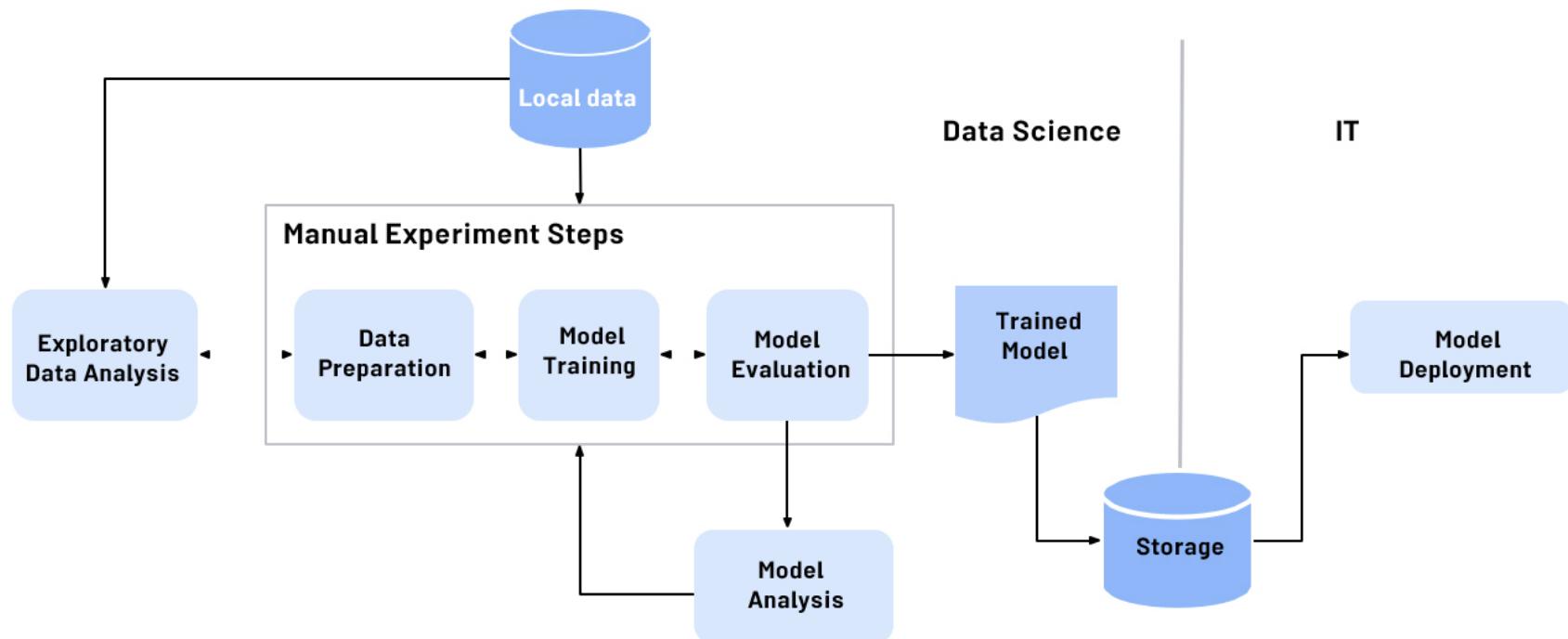
► Disconnection between ML and operations





MLOps level 0: Manual process

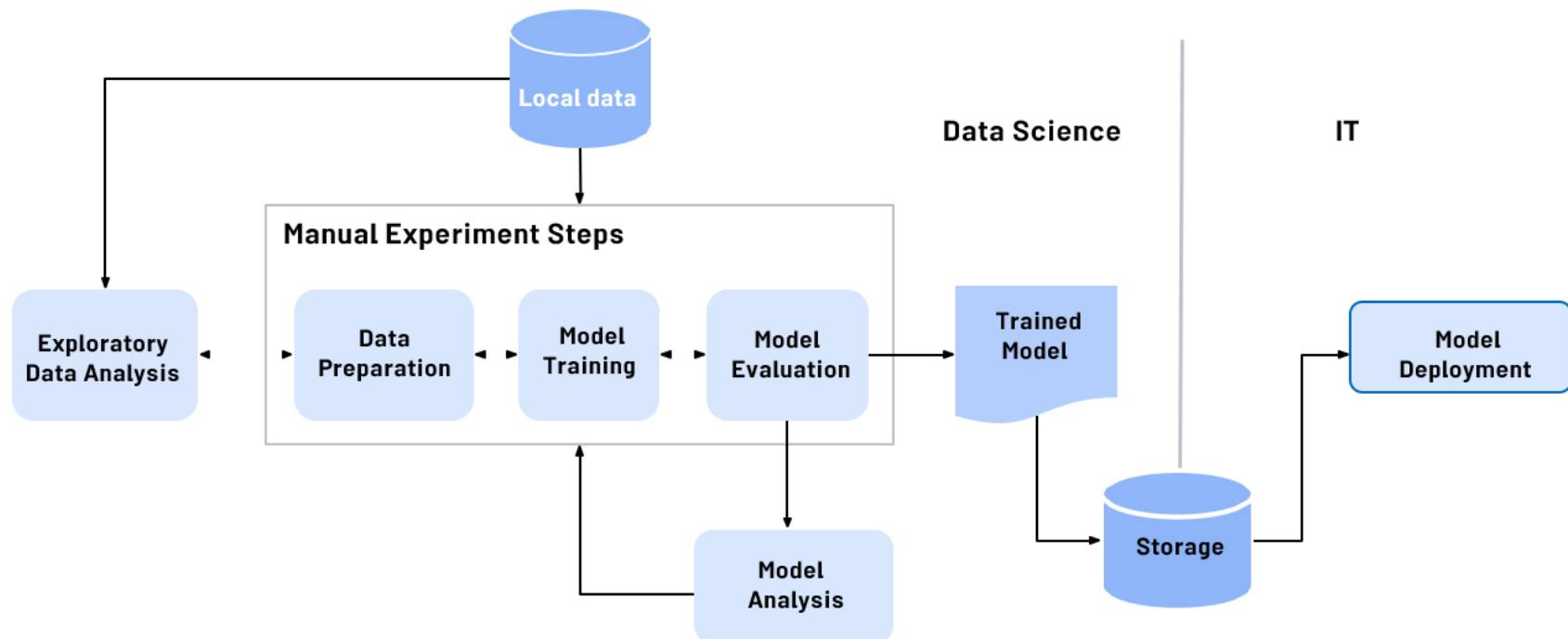
- Less frequent releases, so no CI/CD





How do you scale?

► Deployment and lack of active performance monitoring



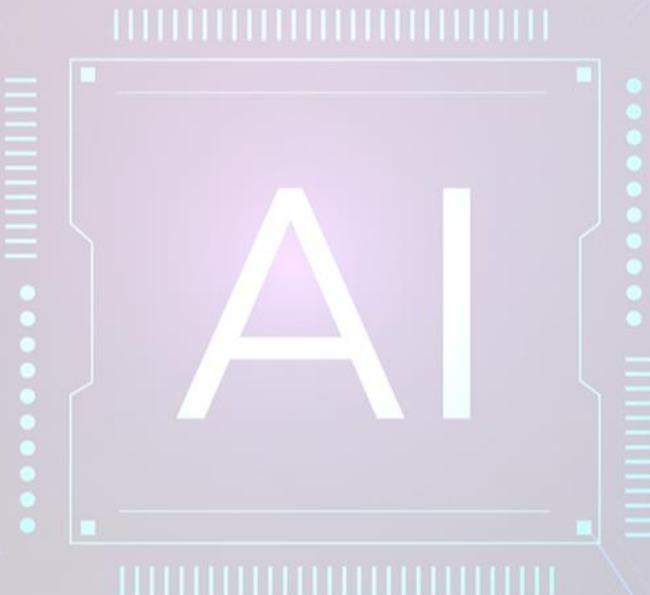


Challenges for MLOps level 0

- ▶ Need for actively monitoring the quality of your model in production
- ▶ Retraining your production models with new data
- ▶ Continuously experimenting with new implementations to improve the data and model



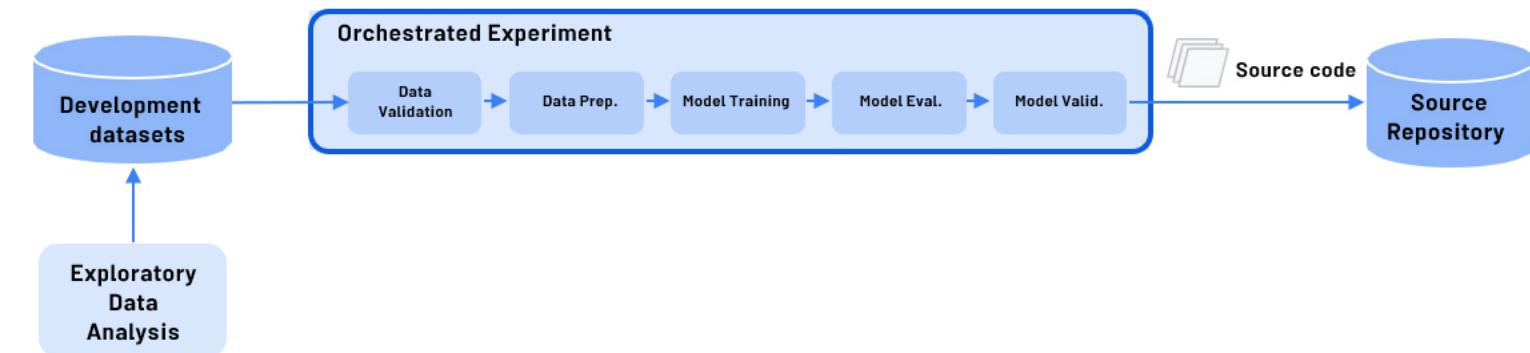
MLOps levels 1 and 2





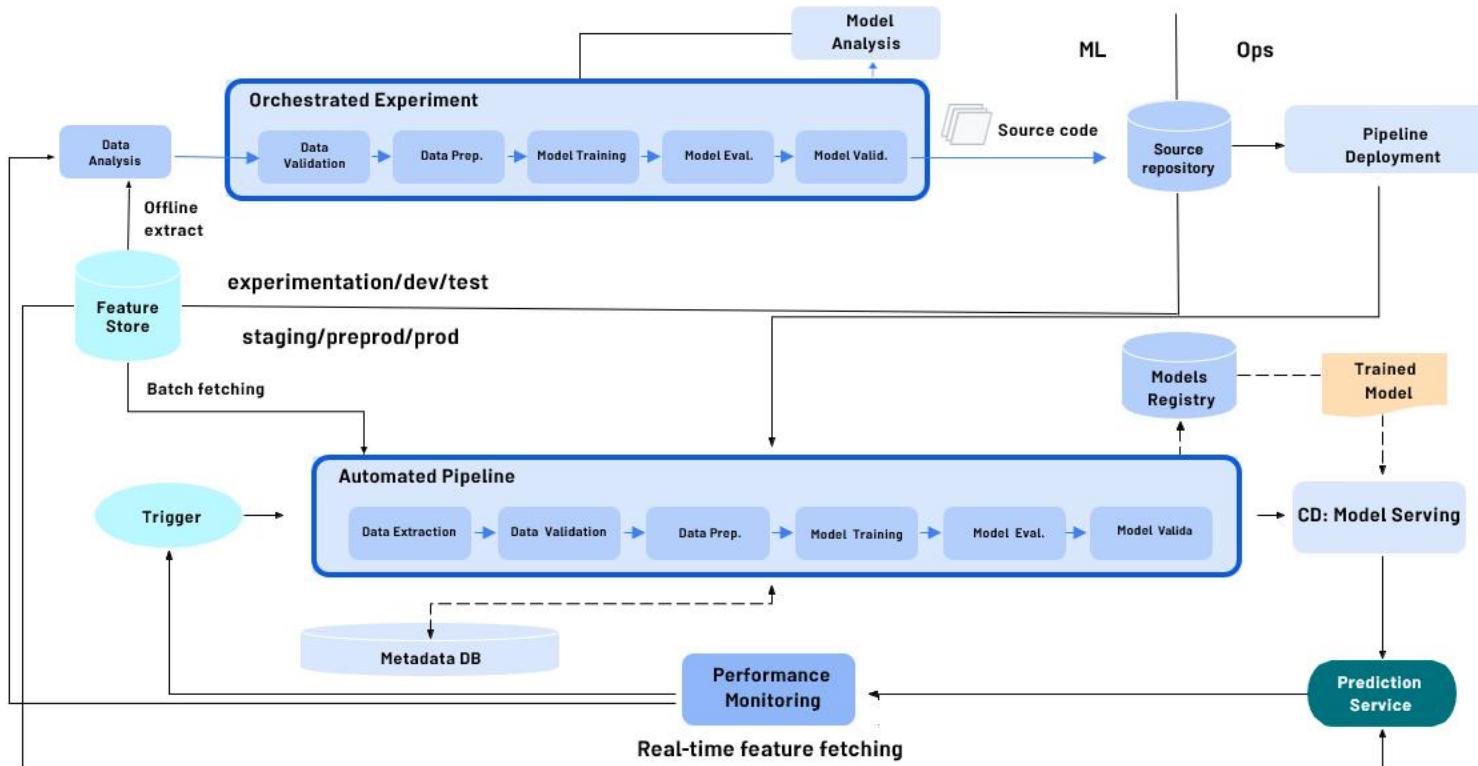
MLOps level 1: ML pipeline automation

► Rapid experimentation



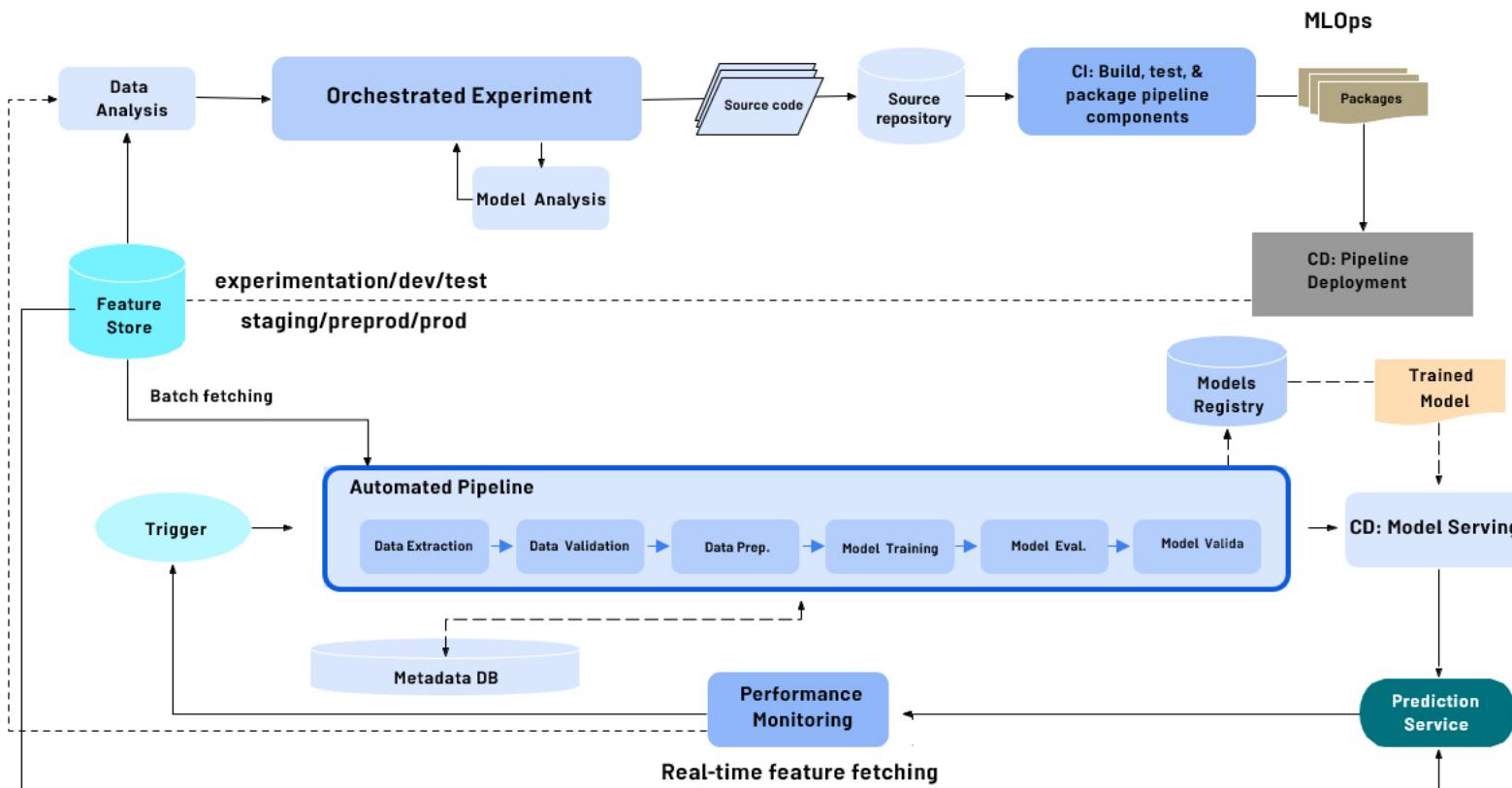


Reusable, composable, and shareable components



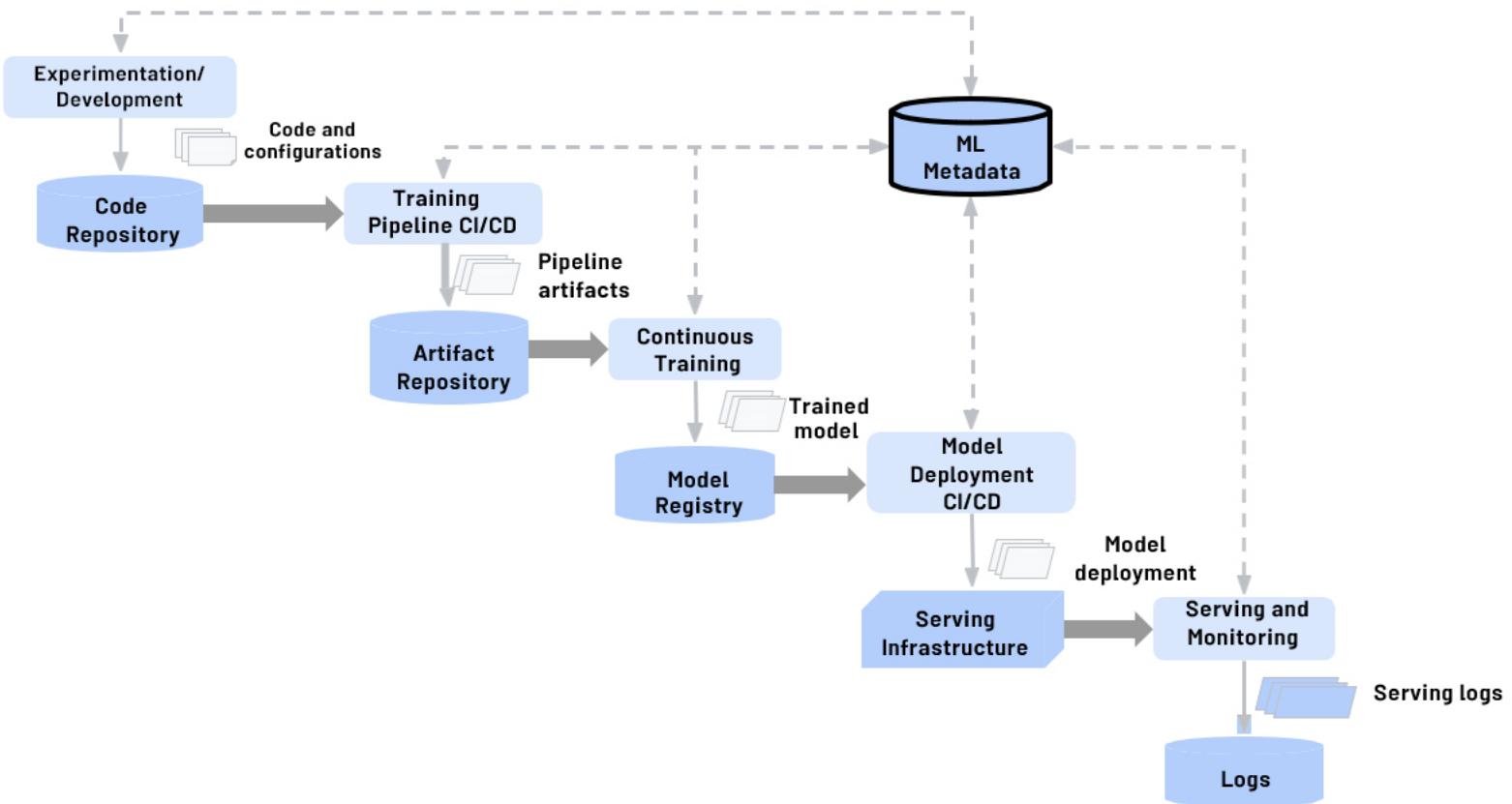


MLOps level 2: CI/CD pipeline automation





Putting all together



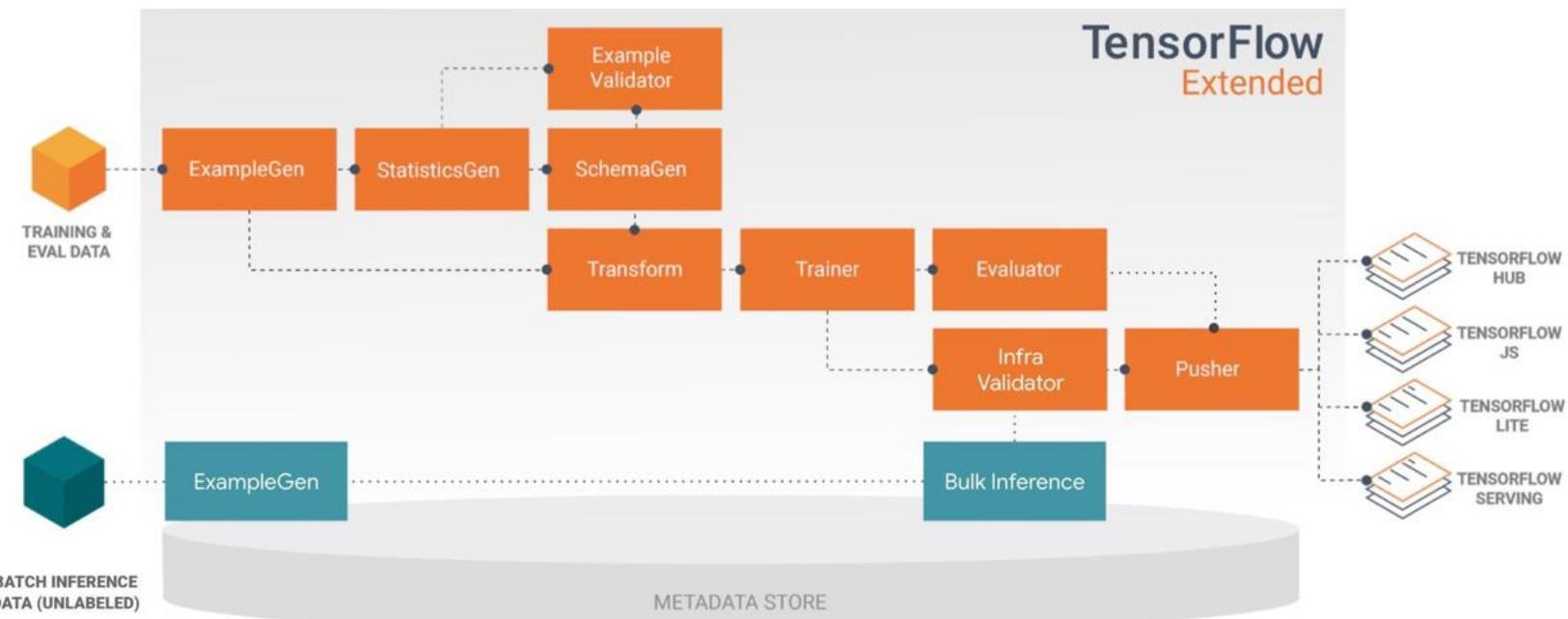


Orchestrate your ML workflows with TFX

- ▶ Pre-built and standard components, and 3 styles of custom components
- ▶ Components can also be containerized
- ▶ Examples of things you can do with TFX components:
 - ▶ Data augmentation, upsampling, or downsampling
 - ▶ Anomaly detection based on confidence intervals or autoencoder reproduction error
 - ▶ Interfacing with external systems like help desks for alerting and monitoring
 - ▶ ... and more!



Hello TFX





Anatomy of a TFX Component

▶ **Component Specification**

- ▶ The component's input and output contract

▶ **Executor Class**

- ▶ Implementation of the component's processing

▶ **Component Class**

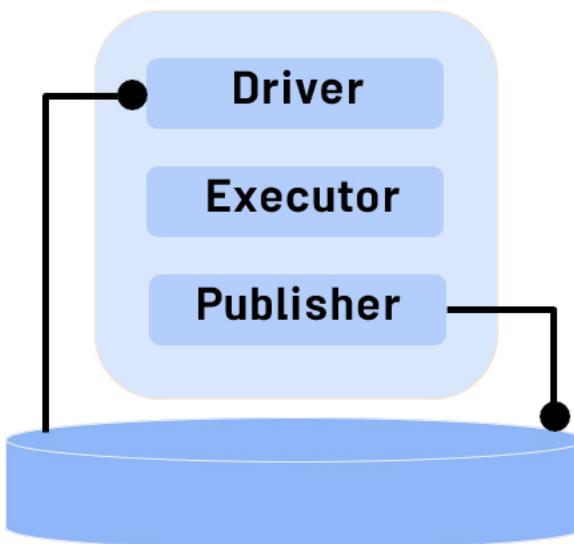
- ▶ Combines the specification with the executor
to create a TFX component



TFX components at runtime

▶ Types of custom components

- ▶ Fully custom components combine the specification with the executor
- ▶ Python function-based components use a decorator and argument annotations
- ▶ Container-based components wrap the component inside a Docker container





Python function-based components

```
@component
def MyValidationComponent(
    model:
        InputArtifact[Model],
    blessing: OutputArtifact[Model],
    accuracy_threshold: Parameter[int] =
        10,
) -> OutputDict(accuracy=float):
    '''My simple custom model validation
    component.'''
    accuracy = evaluate_model(model)
    if accuracy >=
        accuracy_threshold:
        write_output_blessing(blessing)

    return {
        'accuracy': accuracy
    }
```



Container-based components

```
from tfx.dsl.component.experimental import
container_component from tfx.dsl.component.experimental
import placeholders
from tfx.types import standard_artifacts

grep_component =
    container_component.create_container_component(
        name='FilterWithGrep',
        inputs={'text': standard_artifacts.ExternalArtifact},
        outputs={'filtered_text':
            standard_artifacts.ExternalArtifact}, parameters={'pattern': str},
        ...
    )
```



Container-based components

```
grep_component = container_component.create_container_component(  
    ...  
    image='google/cloud-sdk:278.0.0',  
    command=['sh', '-exc', ...  
        ...  
        ...,  
        '--pattern', placeholders.InputValuePlaceholder('pattern'),  
        '--text', placeholders.InputUriPlaceholder('text'),  
        '--filtered-text',  
        placeholders.OutputUriPlaceholder('filtered_text'),  
    ],  
)
```



Fully custom components

- ▶ Define custom component spec, executor class, and component class
- ▶ Component reusability
 - ▶ Reuse a component spec and implement a new executor that derives from an existing component





Defining input and output specifications

```
class HelloComponentSpec(types.ComponentSpec):

    INPUTS = {
        # This will be a dictionary with input artifacts, including URIs
        'input_data':
            ChannelParameter(type=standard_artifacts.Examples),
    }

    OUTPUTS = {
        # This will be a dictionary which this component will populate
        'output_data':
            ChannelParameter(type=standard_artifacts.Examples),
    }

    PARAMETERS = {
        # These are parameters that will be passed in the call to create an instance of this
        # component
        'name': ExecutionParameter(type=Text),
    }
```



Implement the executor

```
class Executor(base_executor.BaseExecutor):
    def Do(self, input_dict: Dict[Text,
        List[types.Artifact]], output_dict: Dict[Text,
        List[types.Artifact]], exec_properties:
        Dict[Text, Any]) -> None:
        ...
    ...
```



Implement the executor

```
class Executor(base_executor.BaseExecutor):  
    ...  
    split_to_instance = {}  
    for artifact in input_dict['input_data']:  
        for split in json.loads(artifact.split_names):  
            uri = os.path.join(artifact.uri, split)  
            split_to_instance[split] = uri  
    for split, instance in split_to_instance.items():  
        input_dir = instance  
        output_dir = artifact_utils.get_split_uri(  
            output_dict['output_data'], split)  
        for filename in tf.io.gfile.listdir(input_dir):  
            input_uri = os.path.join(input_dir, filename)  
            output_uri = os.path.join(output_dir, filename)  
            io_utils.copy_file(src=input_uri, dst=output_uri, overwrite=True)
```



Make the component pipeline-compatible

```
from tfx.types import standard_artifacts
from hello_component import executor

class HelloComponent(base_component.BaseComponent):
    SPEC_CLASS = HelloComponentSpec
    EXECUTOR_SPEC = ExecutorClassSpec(executor.Executor)
```



Completing the component class

```
class HelloComponent(base_component.BaseComponent):

    ...

    def __init__(self,
                 input_data: types.Channel = None,
                 output_data: types.Channel =
                     None, name: Optional[Text] =
                     None):

        if not output_data:
            examples_artifact = standard_artifacts.Examples()
            examples_artifact.split_names =
            input_data.get()[0].split_names
            output_data =
            channel_utils.as_channel([examples_artifact])

        spec = HelloComponentSpec(input_data=input_data, output_data=output_data,
                                   name=name)
        super(HelloComponent, self).__init__(spec=spec)
```



Assemble into a TFX pipeline

```
def _create_pipeline():
    ...
    example_gen = CsvExampleGen(input_base=examples)

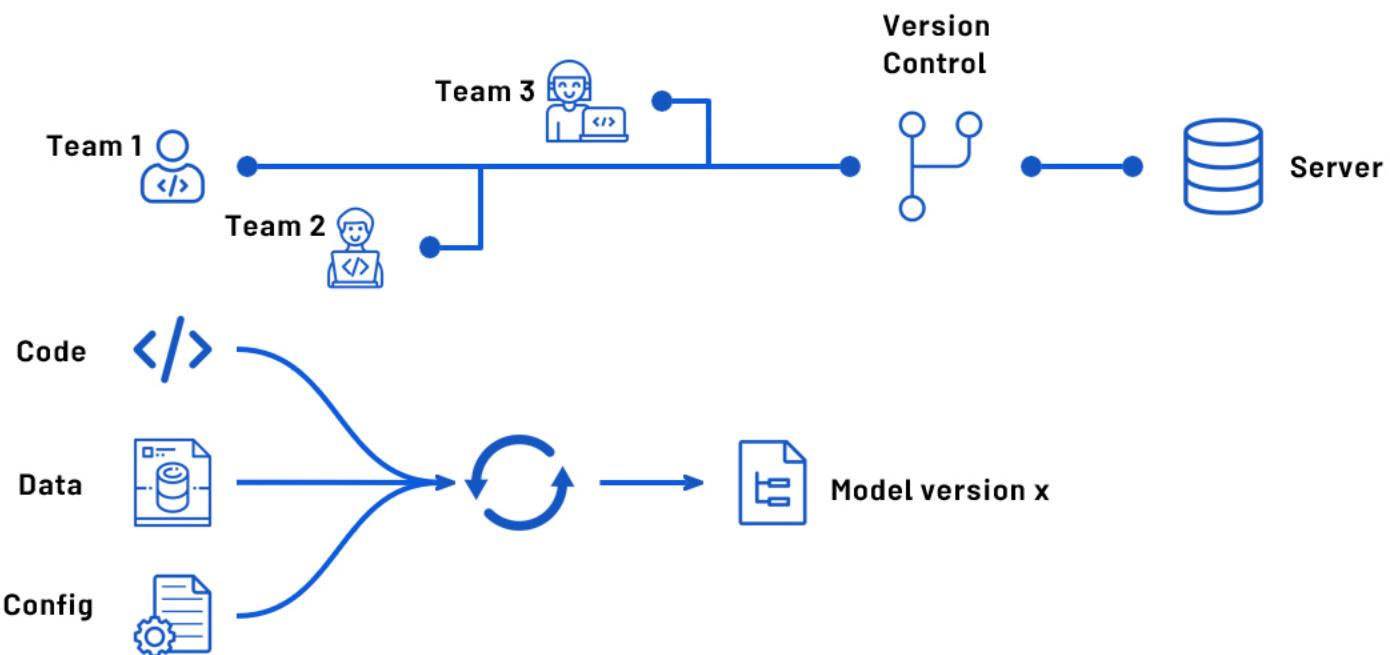
    hello = component>HelloComponent(
        input_data=example_gen.outputs['examples'],
        name='HelloWorld')

    statistics_gen =
    StatisticsGen(examples=hello.outputs['output_data'])

    ...
    return pipeline.Pipeline(
        ...
        components=[example_gen, hello, statistics_gen,
        ...],
        ...
    )
```



Why versioning ML Models?





How ML Models are versioned?

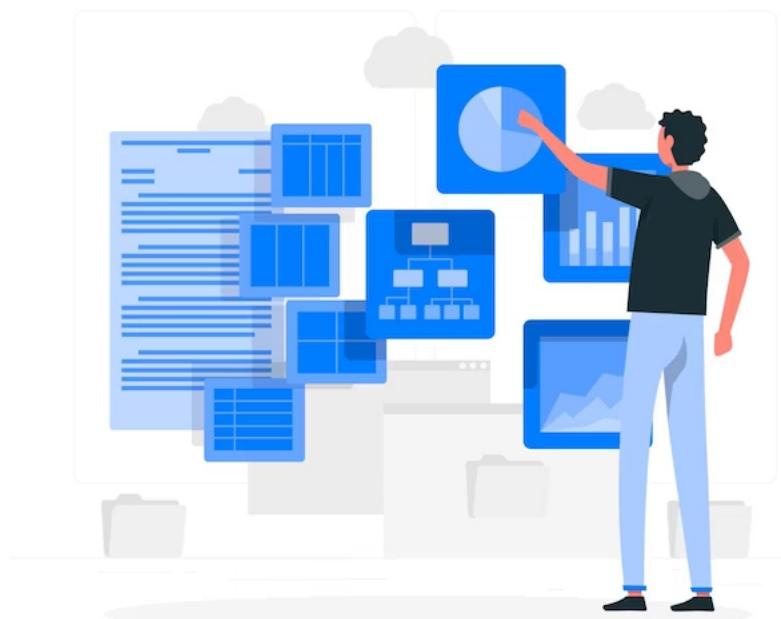
- ▶ How software is versioned?
- ▶ Version: MAJOR.MINOR.PATCH
 - ▶ MAJOR: Contains incompatible API changes
 - ▶ MINOR: Adds functionality in a backwards compatible manner
 - ▶ PATCH: Makes backwards compatible bug fixes
- ▶ ML Models versioning
 - ▶ No uniform standard accepted yet
 - ▶ Different organizations have different meanings and conventions



A Model Versioning Proposal

▶ Version: MAJOR.MINOR.PIPELINE

- ▶ MAJOR: Incompatibility in data or target variable
- ▶ MINOR: Model performance is improved
- ▶ PIPELINE: Pipeline of model training is changed





Retrieving older models

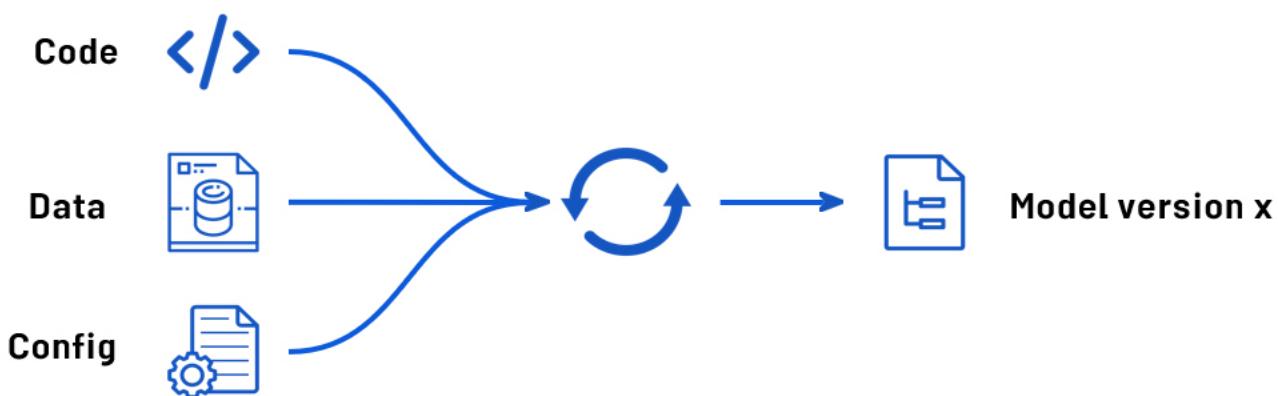
- ▶ Can ML framework be leveraged to retrieve previously trained models?
- ▶ ML framework may internally be versioning models





What is model lineage?

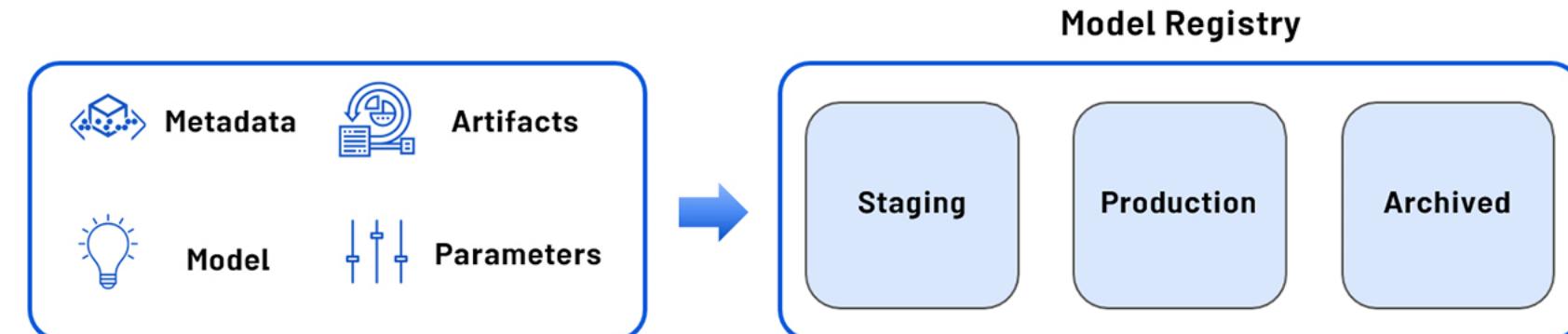
- ▶ Artifacts: information needed to preprocess data and generate result (code, data, config, model)
- ▶ ML orchestration frameworks may store operations and data artifacts to recreate model
- ▶ Post training artifacts and operations are usually not part of lineage





What is a model registry?

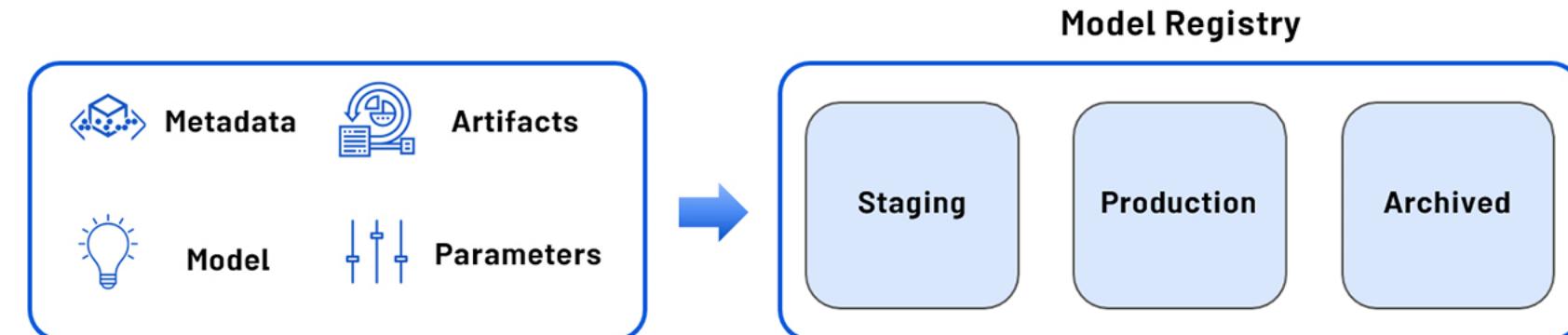
- ▶ Central repository for storing trained ML models
- ▶ Provides various operations of ML model development lifecycle
- ▶ Promotes model discovery, model understanding, and model reuse
- ▶ Integrated into OSS and commercial ML platforms





Metadata stored by model registry

- ▶ Model versions
- ▶ Model serialized artifacts
- ▶ Free text annotations and structured properties
- ▶ Links to other ML artifact and metadata stores





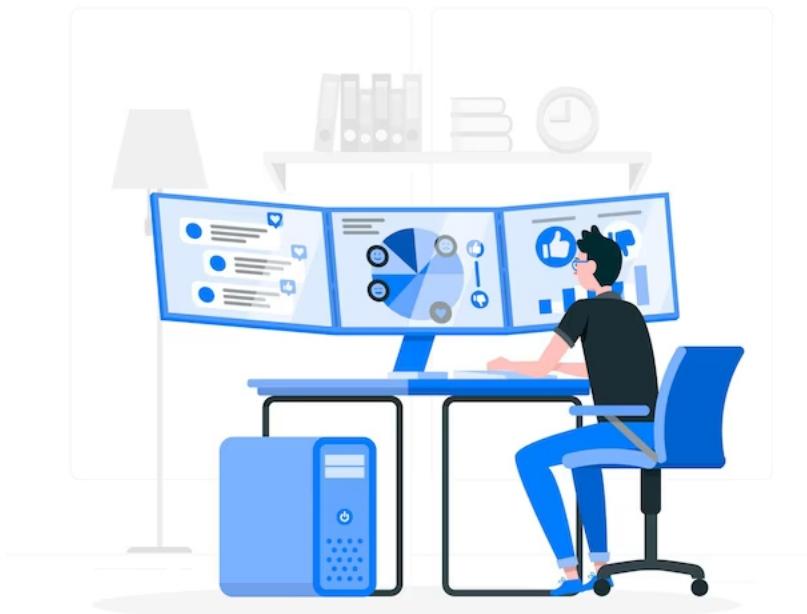
Capabilities Enabled by Model Registries

- ▶ Model search/discovery and understanding
- ▶ Approval/Governance
- ▶ Collaboration/Discussion
- ▶ Streamlined deployments
- ▶ Continuous evaluation and monitoring
- ▶ Staging and promotions



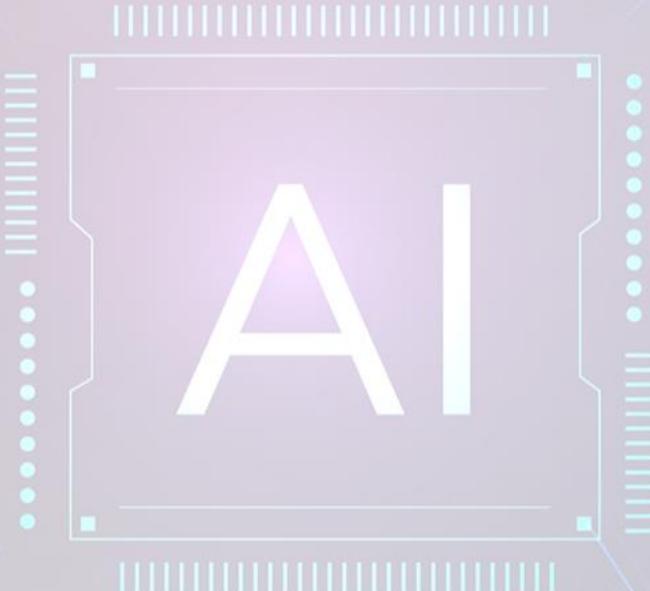
Examples of Model Registries

- ▶ Azure ML Model Registry
- ▶ SAS Model Manager
- ▶ Mlflow Model Registry
- ▶ Google AI Platform
- ▶ Algorithmia



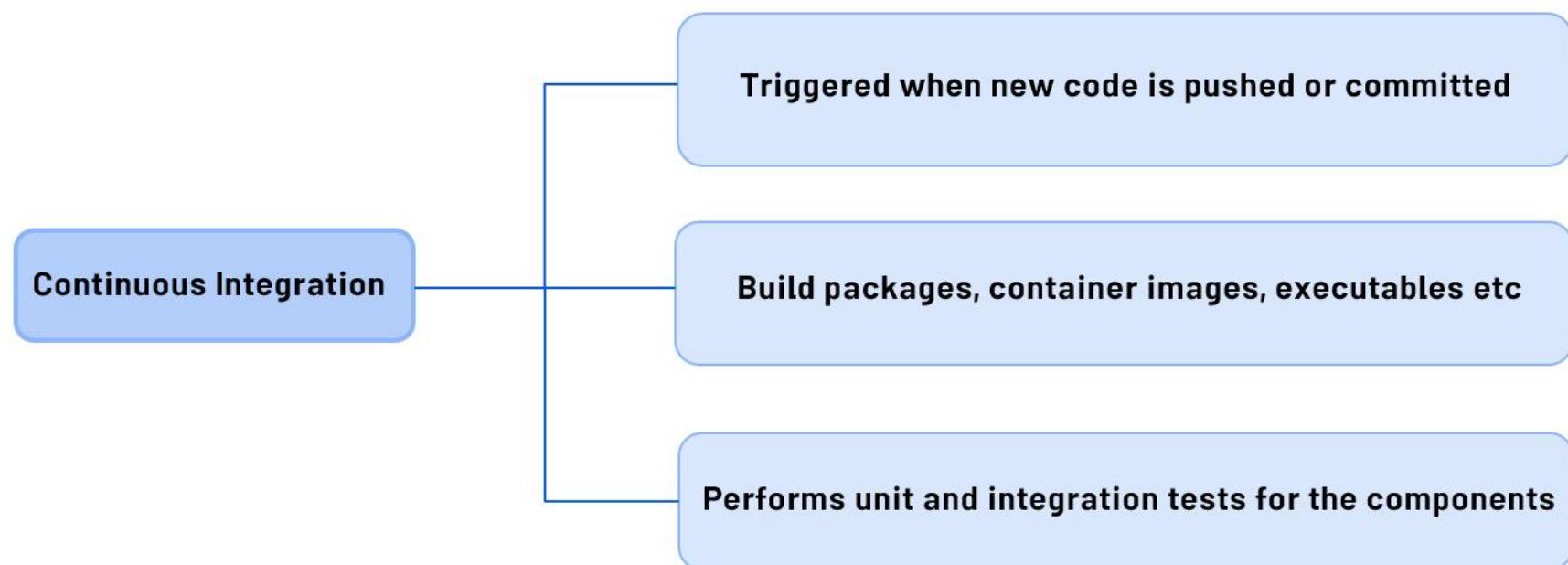


Continuous Delivery





What is Continuous Integration (CI)





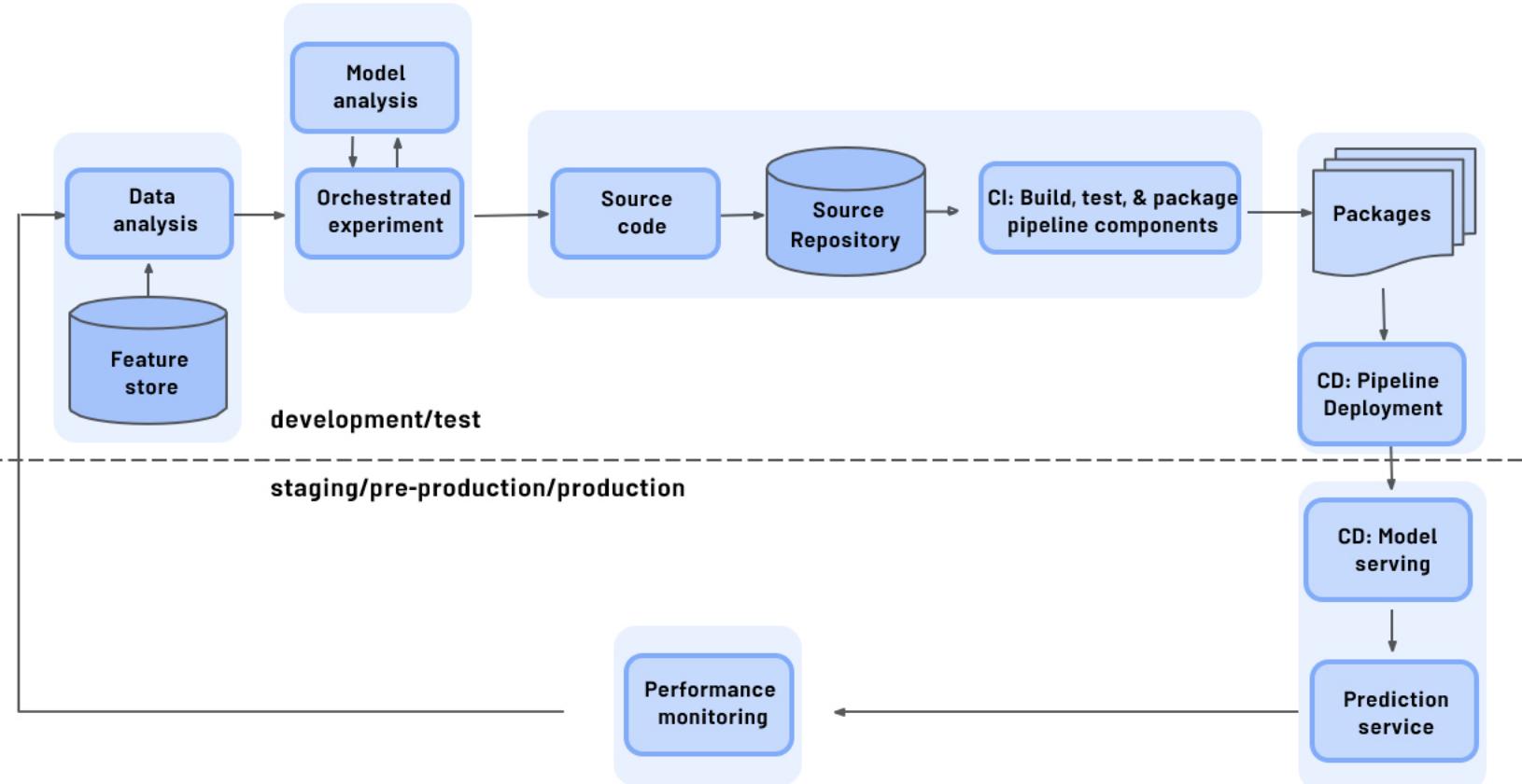
What is Continuous Integration (CI)

Continuous Delivery

- Deploys new code and trained models to the target environment
- Ensures compatibility of code and models with the target environment
- Checks the prediction service performance of the model before deploying



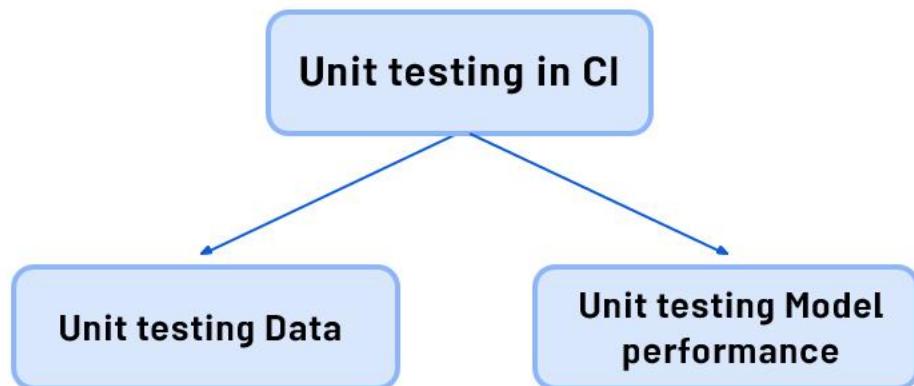
CI/CD Infrastructure





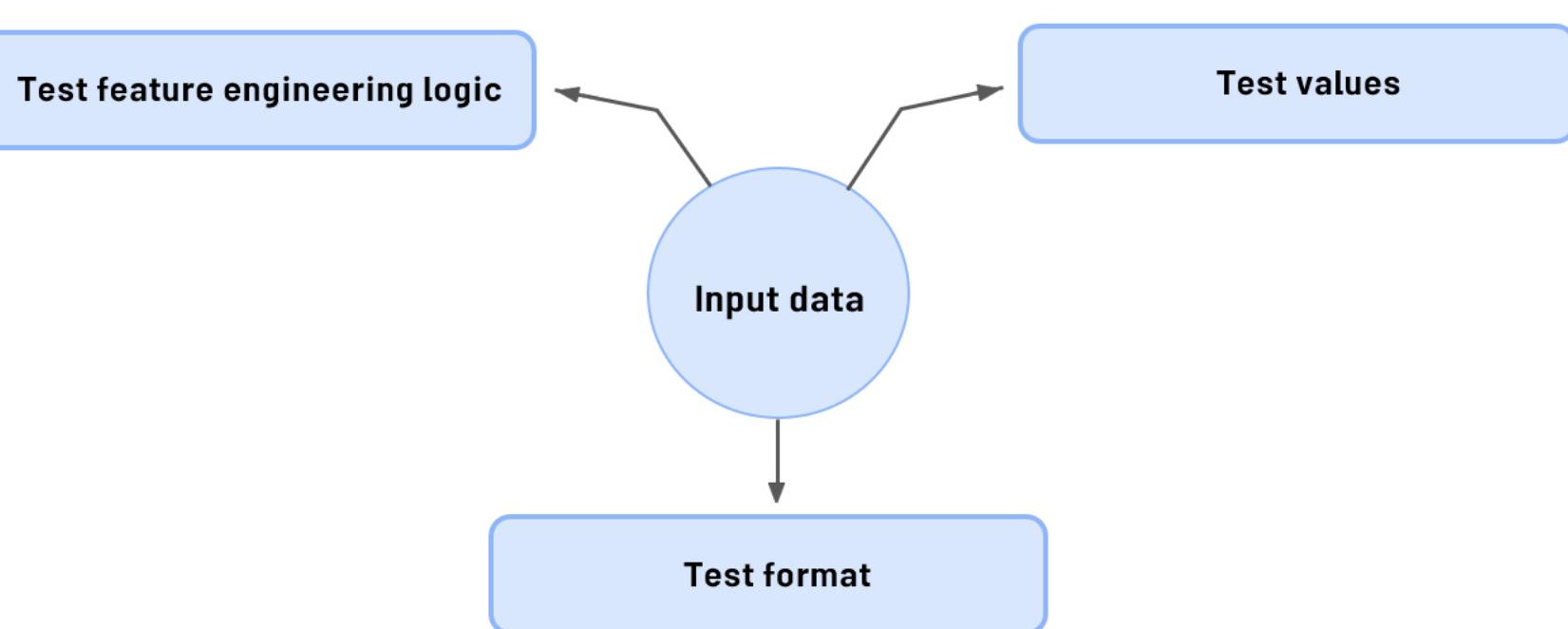
Unit Testing in CI

- ▶ Testing that each component in the pipeline produces the expected artifacts.



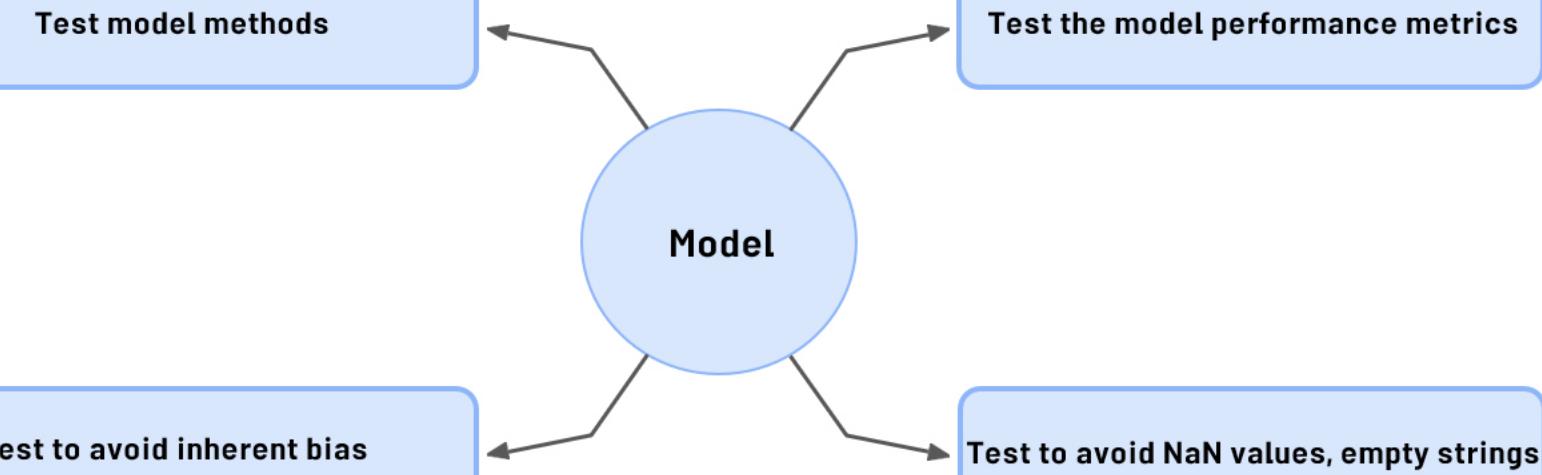


Unit Testing Input Data





Unit Testing Model Performance





ML Unit Testing Considerations

Mocking

Mocks of datasets are especially important for ML. They should cover edge and corner cases.

Data Coverage

Your mocks should sparsely cover the same space as your data, but with a much smaller dataset.

Code Coverage

Use code coverage libraries to make sure that you are not missing unit tests for any part of our code.



Infrastructure validation

► When to apply infrastructure validation

- Before starting CI/CD as part of model training
- Can also occur as part of CI/CD as a last check to verify that the model is deployable to the serving infrastructure

► TFX InfraValidator

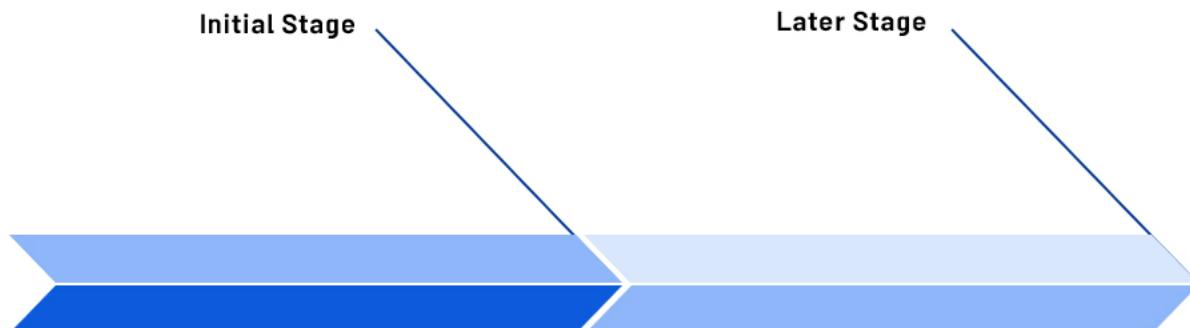
- TFX InfraValidator takes the model, launches a sand-boxed model server with the model, and sees if it can be successfully loaded and optionally queried
- InfraValidator is using the same model server binary, same resources, and same server configuration as production



Progressive Delivery

► Progressive Delivery is essentially an improvement over Continuous Delivery

- Decrease deployment risk
- Faster deployment
- Gradual rollout and ownership



Deliver changes first to
small, low risk audiences

Expand to larger
riskier audience



Complex Model Deployment Scenarios

- ▶ You can deploy multiple models performing same task
- ▶ Deploying competing models, as in A/B testing
- ▶ Deploying as shadow models, as in Canary testing





Live Experimentation: A/B Testing

