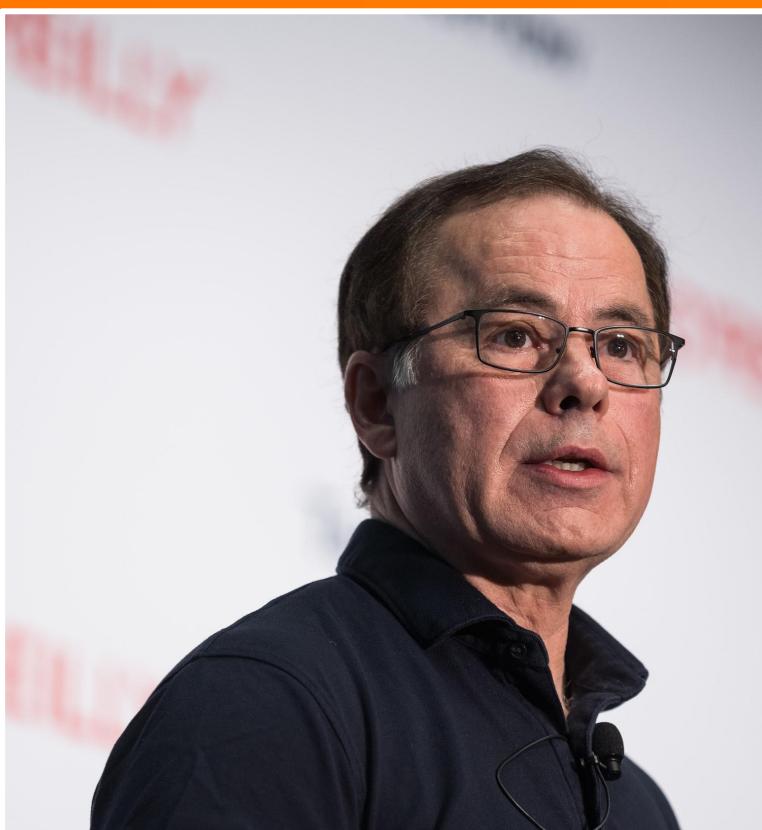


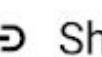
Production Machine Learning



Motivations, Requirements, and Processes



Robert Crowe
TensorFlow Developer Engineer
 @robert_crowe

 beginner.ipynb 

File Edit View Insert Runtime Tools Help

 Table of contents 

 Copyright 2019 The TensorFlow Authors.

TensorFlow 2 quickstart for beginners

 Section

 **TensorFlow 2 quickstart for beginners**

 [View on TensorFlow.org](#)  [Run in Google Colab](#)  [View source on GitHub](#)  [Download notebook](#)

This short introduction uses [Keras](#) to:

1. Build a neural network that classifies images.
2. Train this neural network.
3. And, finally, evaluate the accuracy of the model.

This is a [Google Colaboratory](#) notebook file. Python programs are run directly in the browser—a great way to learn and use TensorFlow. To follow this tutorial, run the notebook in Google Colab by clicking the button at the top of this page.

1. In Colab, connect to a Python runtime: At the top-right of the menu bar, select *CONNECT*.
2. Run all the notebook code cells: Select *Runtime > Run all*.

Download and install TensorFlow 2. Import TensorFlow into your program:

Note: Upgrade `pip` to install the TensorFlow 2 package. See the [install guide](#) for details.

```
[ ] import tensorflow as tf
```

Load and prepare the [MNIST dataset](#). Convert the samples from integers to floating-point numbers:

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova

Google AI Language

{jacobdevlin, mingweichang, kentonl, kristout}@google.com

Abstract

We introduce a new language representation model called **BERT**, which stands for **Bidirectional Encoder Representations from Transformers**. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer.

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-



Production Machine Learning

Machine Learning Development

- Labeled data
- Feature space coverage
- Minimal dimensionality
- Maximum predictive data
- Fairness
- Rare conditions
- Data lifecycle management



Modern Software Development

- Scalability
- Extensibility
- Configuration
- Consistency & Reproducibility
- Modularity
- Best Practices
- Testability
- Monitoring
- Safety & Security



We need MLOps

“MLOps is an ML engineering culture and practice that aims at unifying ML system development (Dev) and ML system operation (Ops). Practicing MLOps means that you advocate for automation and monitoring at all steps of ML system construction, including integration, testing, releasing, deployment, and infrastructure management.” - <http://goo.gle/mlops-levels>



Continuous Integration, Deployment, and Training

Continuous Integration

- Testing and validating code and components, but also testing and validating data, data schemas, and models



Continuous Integration, Deployment, and Training

Continuous Deployment

- No longer about a single software package or service, but a system (an ML training pipeline) that should automatically deploy another service (model prediction service)



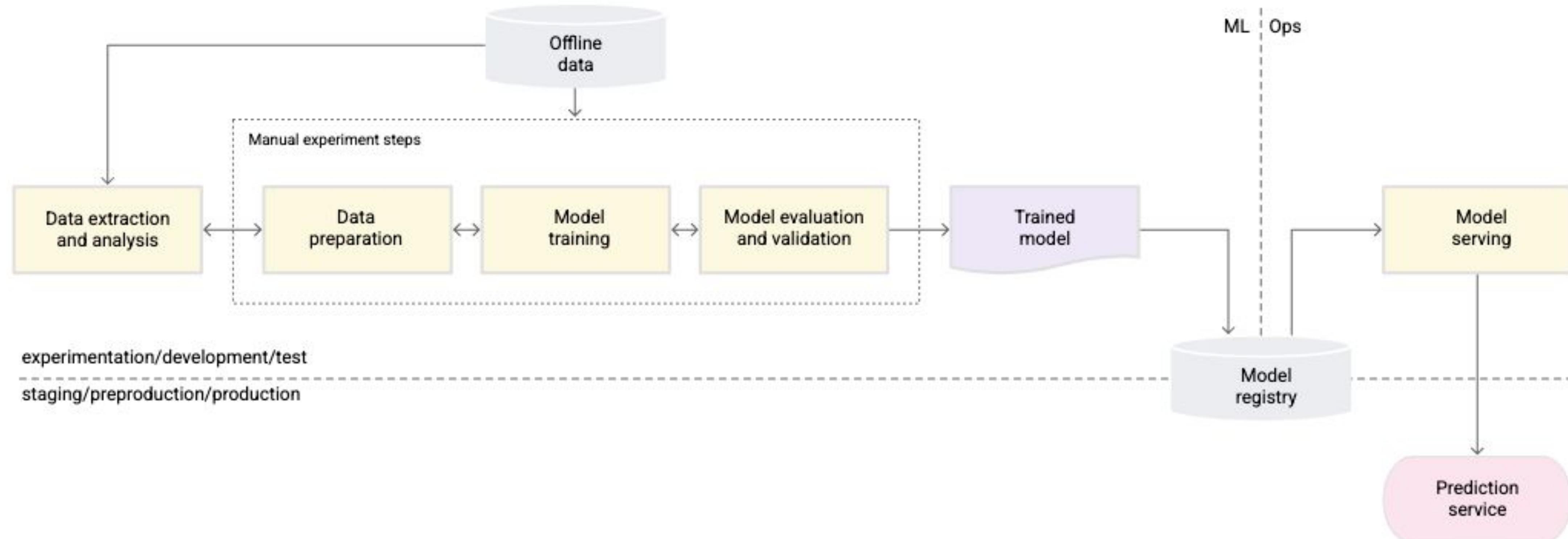
Continuous Integration, Deployment, and Training

Continuous Training

- CT is a new process, unique to ML systems, that's concerned with automatically gathering and labeling new data, retraining, and serving new models



MLOps Level 0: Manual Process



Why isn't one model good enough?

You're an Online Retailer Selling Shoes ...

Your model predicts
click-through rates (CTR),
helping you decide how much
inventory to order

REPLACE PHOTO

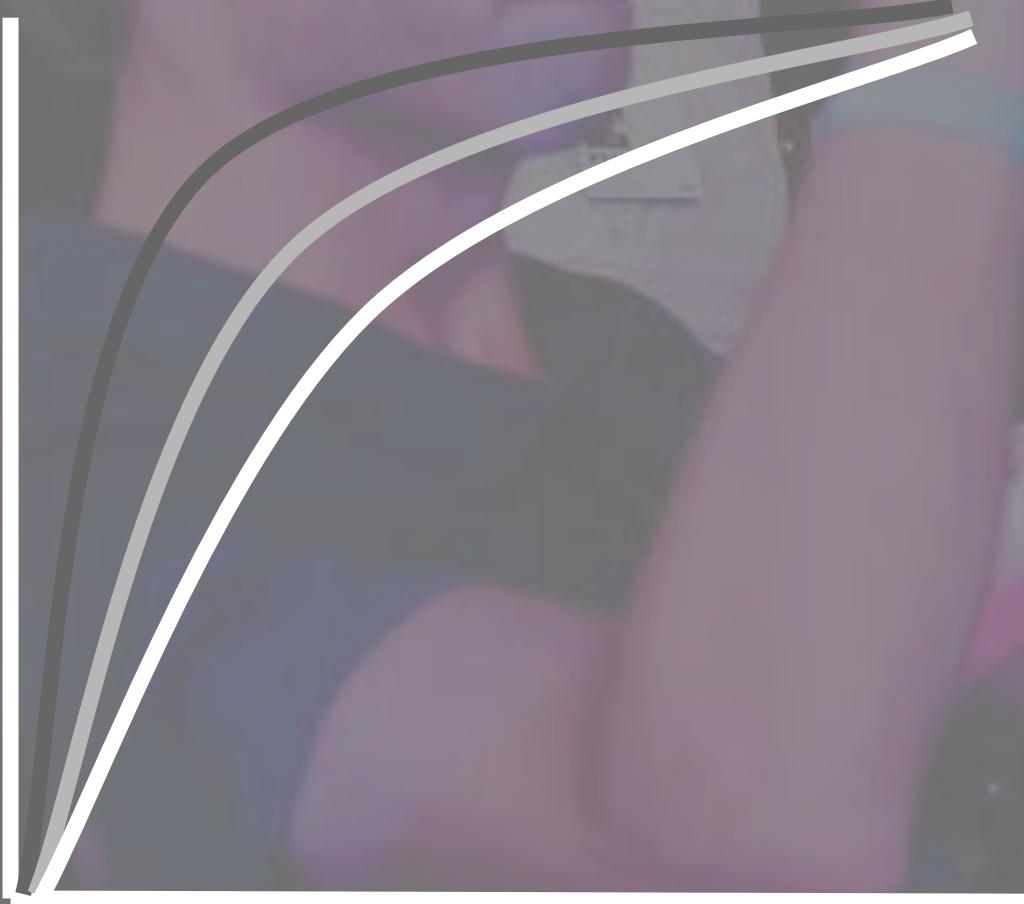




When suddenly

Your AUC and prediction accuracy
have dropped on men's dress shoes!

REPLACE PHOTO



Why?

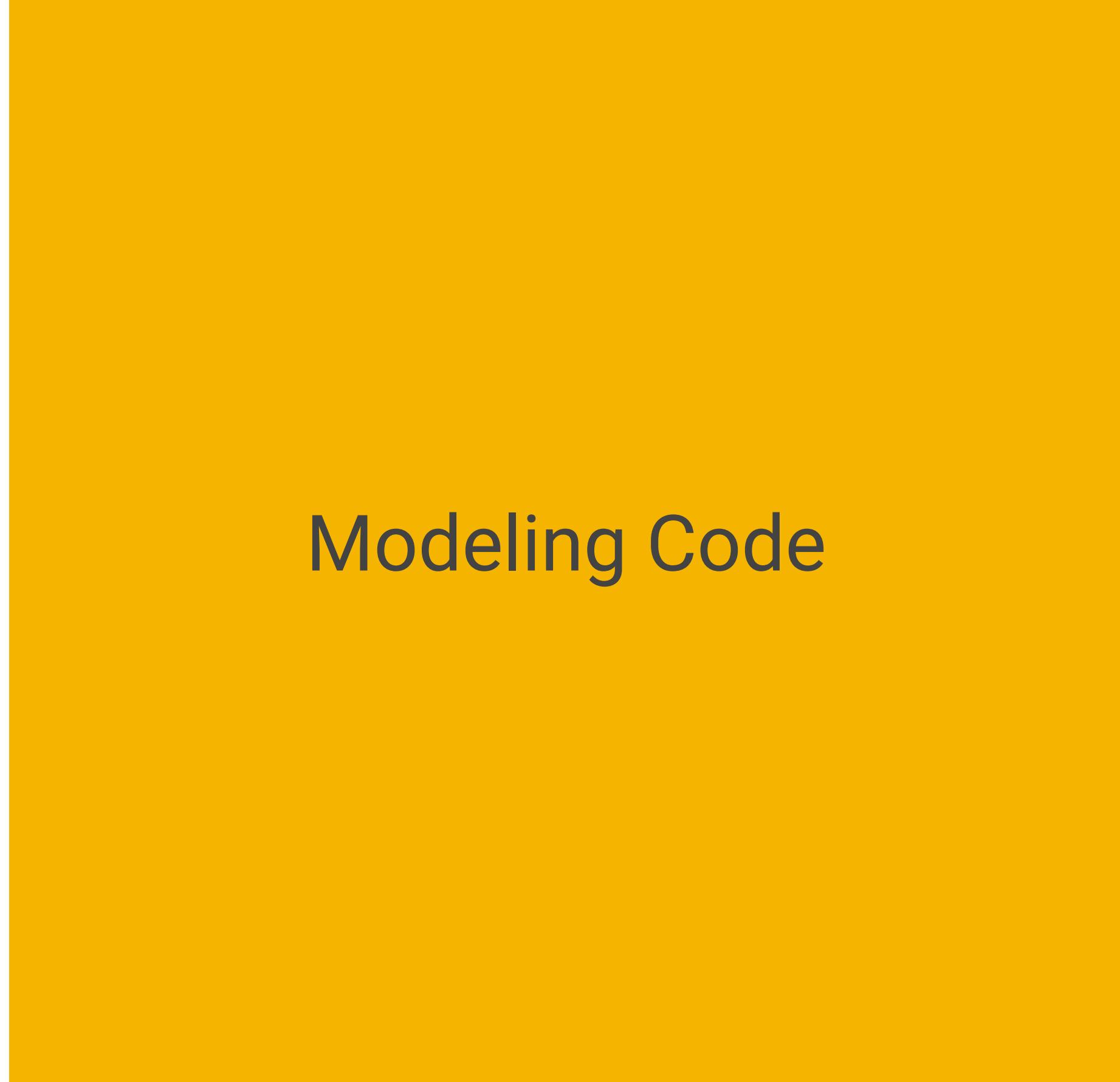
How do we know that we
have a problem?

Why?

Change

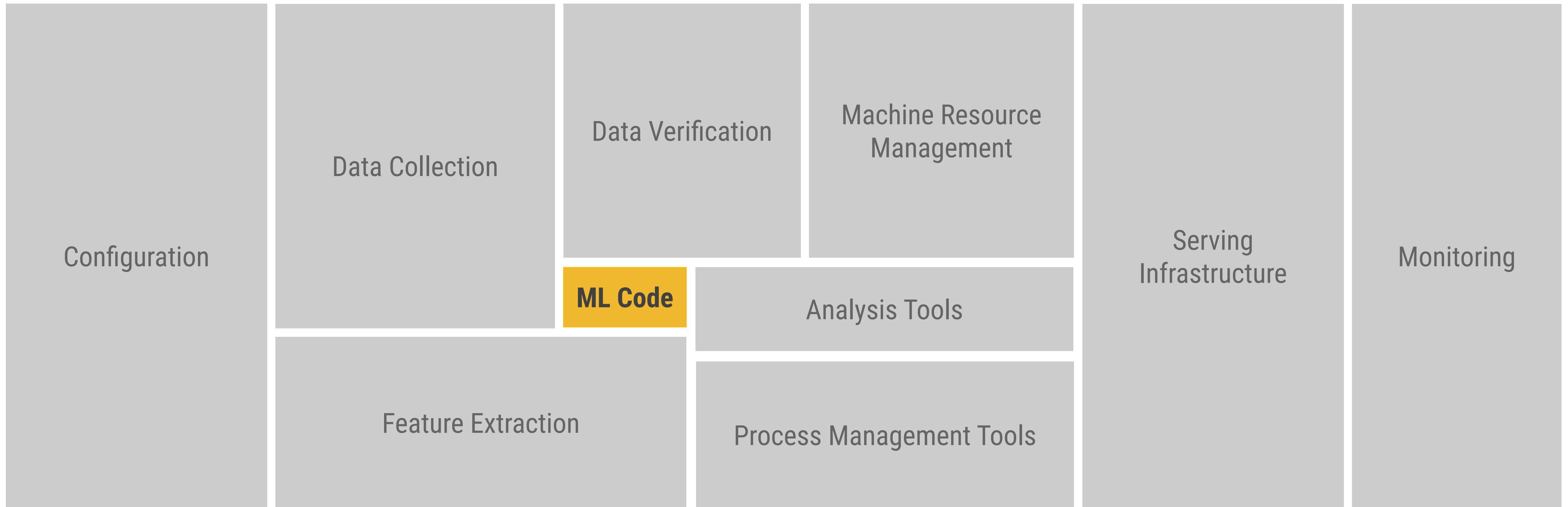
Production Machine Learning

In addition to training an amazing model ...



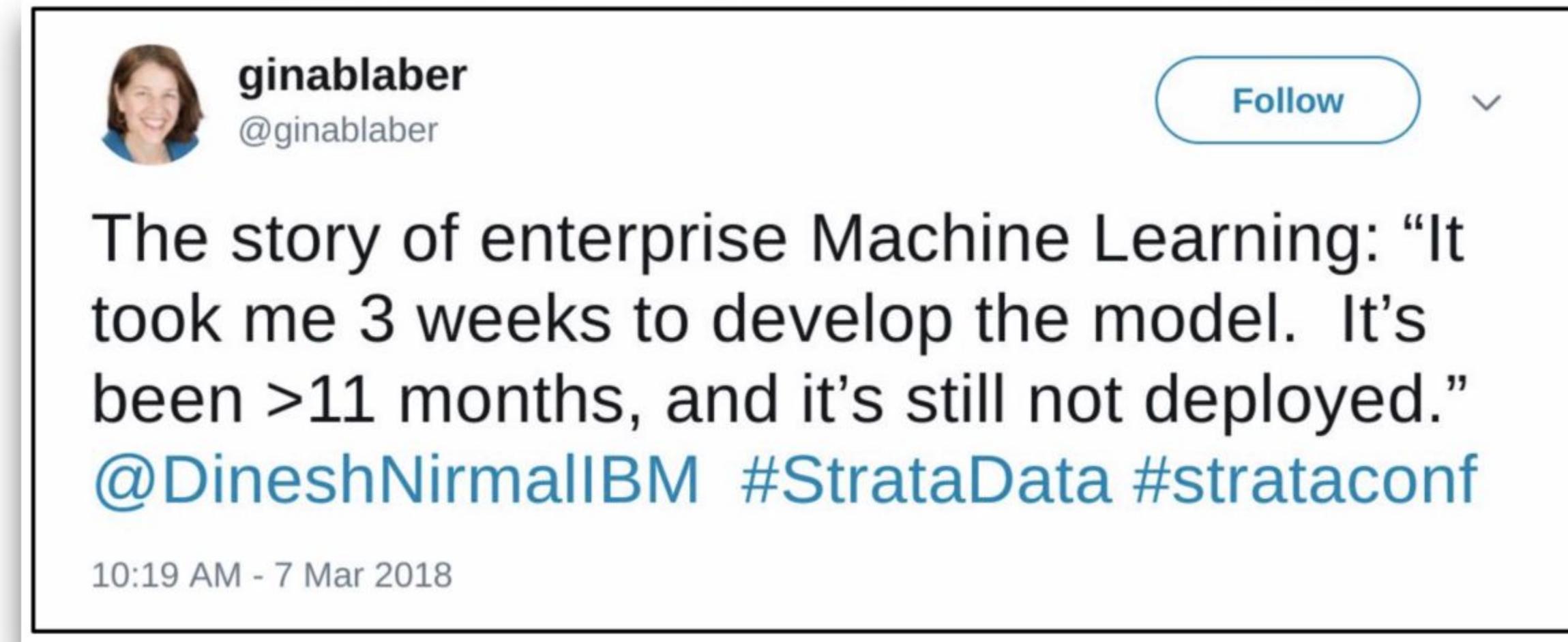
Modeling Code

... a production solution requires so much more





Tales From The Trenches



A screenshot of a Twitter post from user **ginablaber**. The post features a profile picture of a woman with short brown hair, smiling. The text of the tweet reads: "The story of enterprise Machine Learning: “It took me 3 weeks to develop the model. It’s been >11 months, and it’s still not deployed.” @DineshNirmalIBM #StrataData #strataconf". Below the tweet, the timestamp "10:19 AM - 7 Mar 2018" is visible. To the right of the tweet, there is a "Follow" button and a dropdown arrow.

<https://twitter.com/ginablaber/status/971450218095943681>



Production Machine Learning

“Hidden Technical Debt in Machine Learning Systems”

NIPS 2015

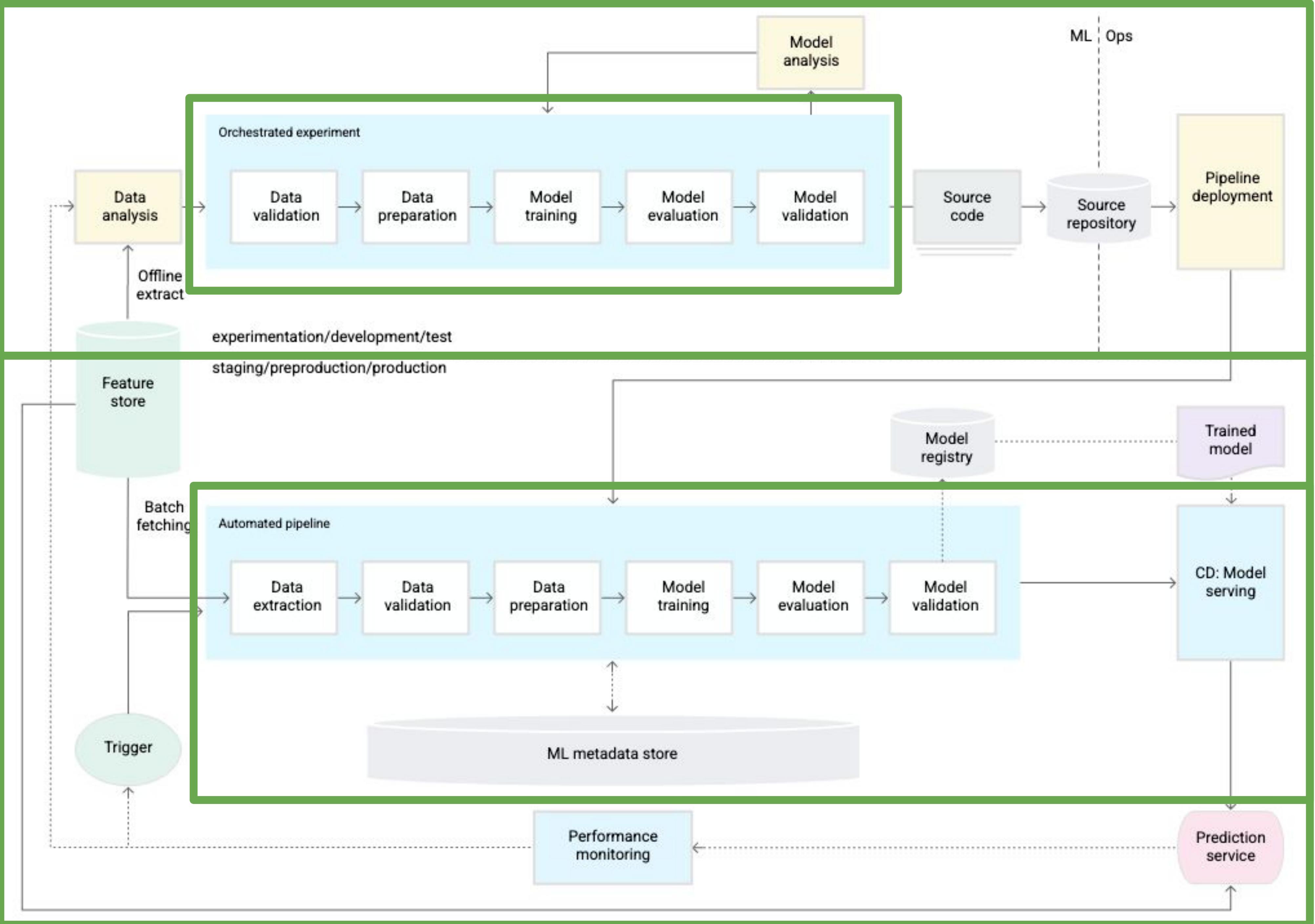
<http://bit.ly/ml-techdebt>



What now?

MLOps

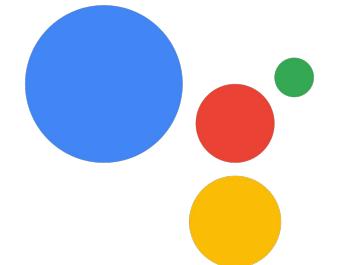
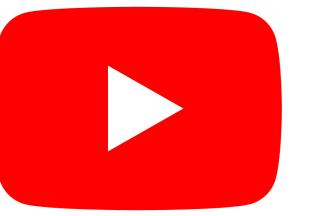
Level 1: ML pipeline automation



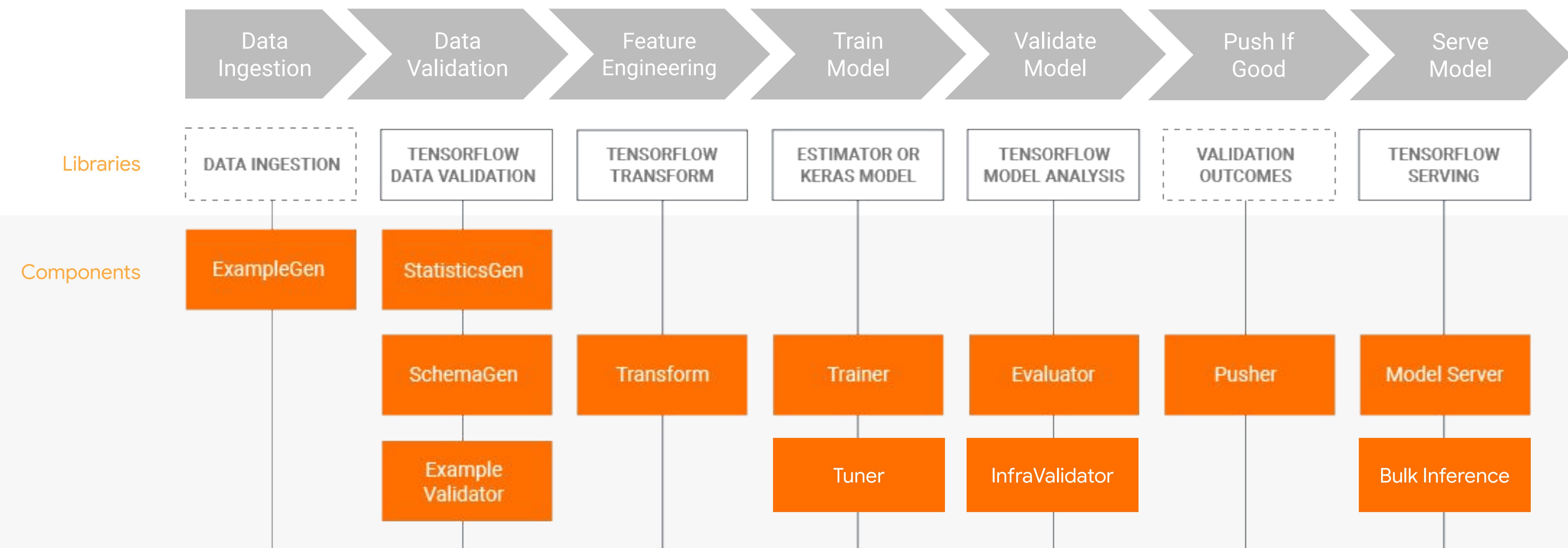
TensorFlow Extended (TFX)

Tensorflow Extended (TFX)

Powers Alphabet's most important bets and products

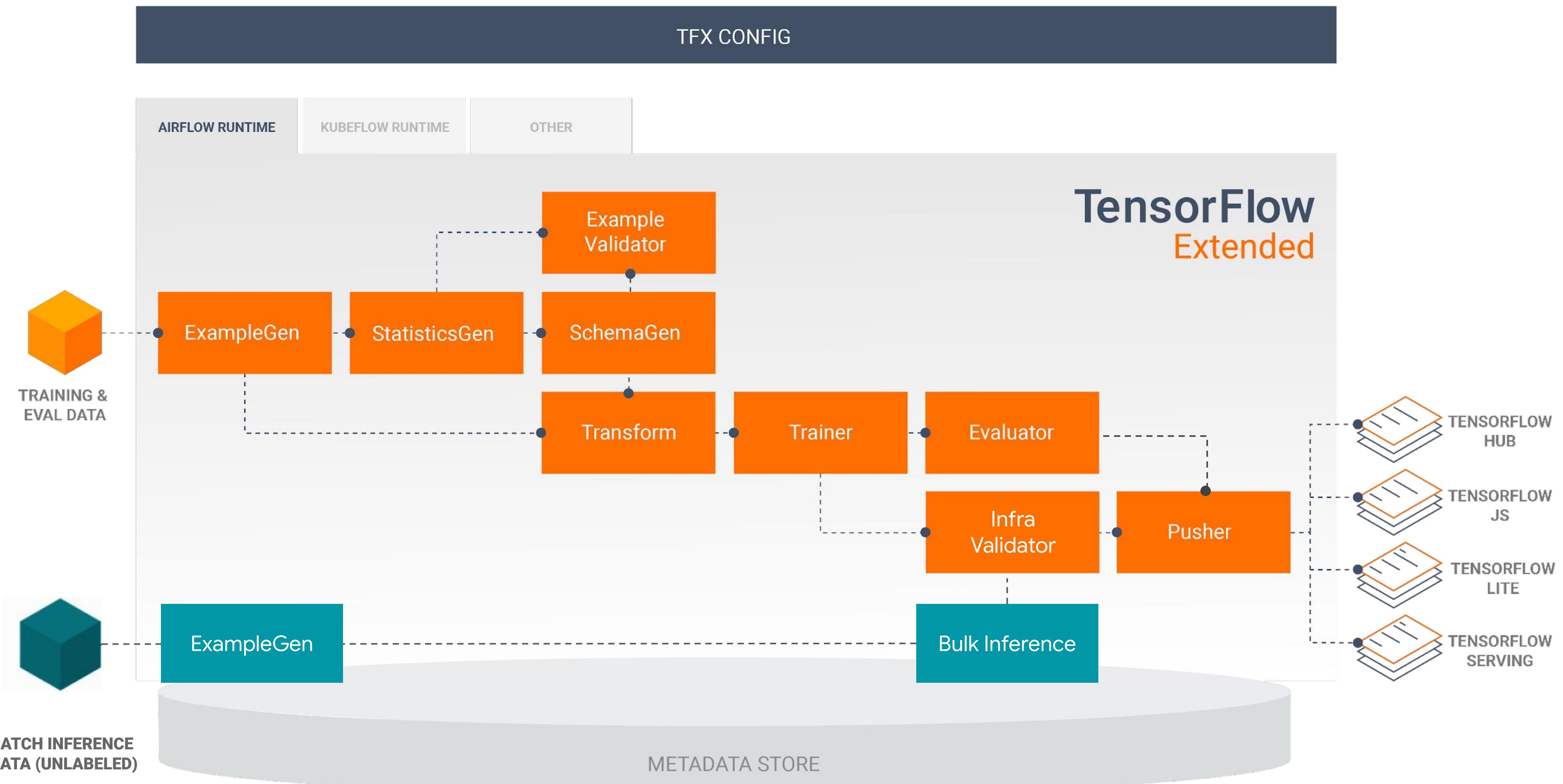


TFX Production Components

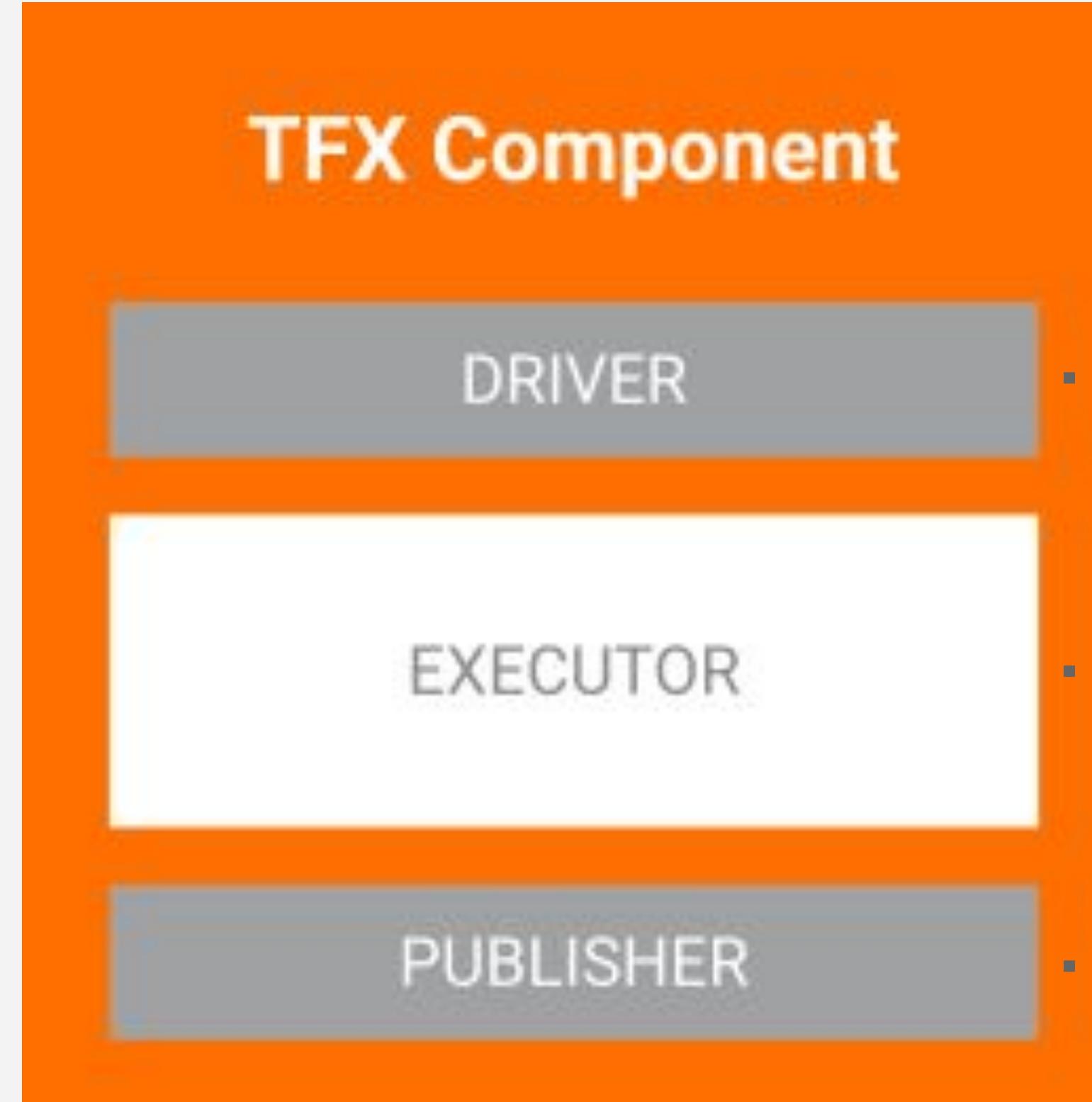




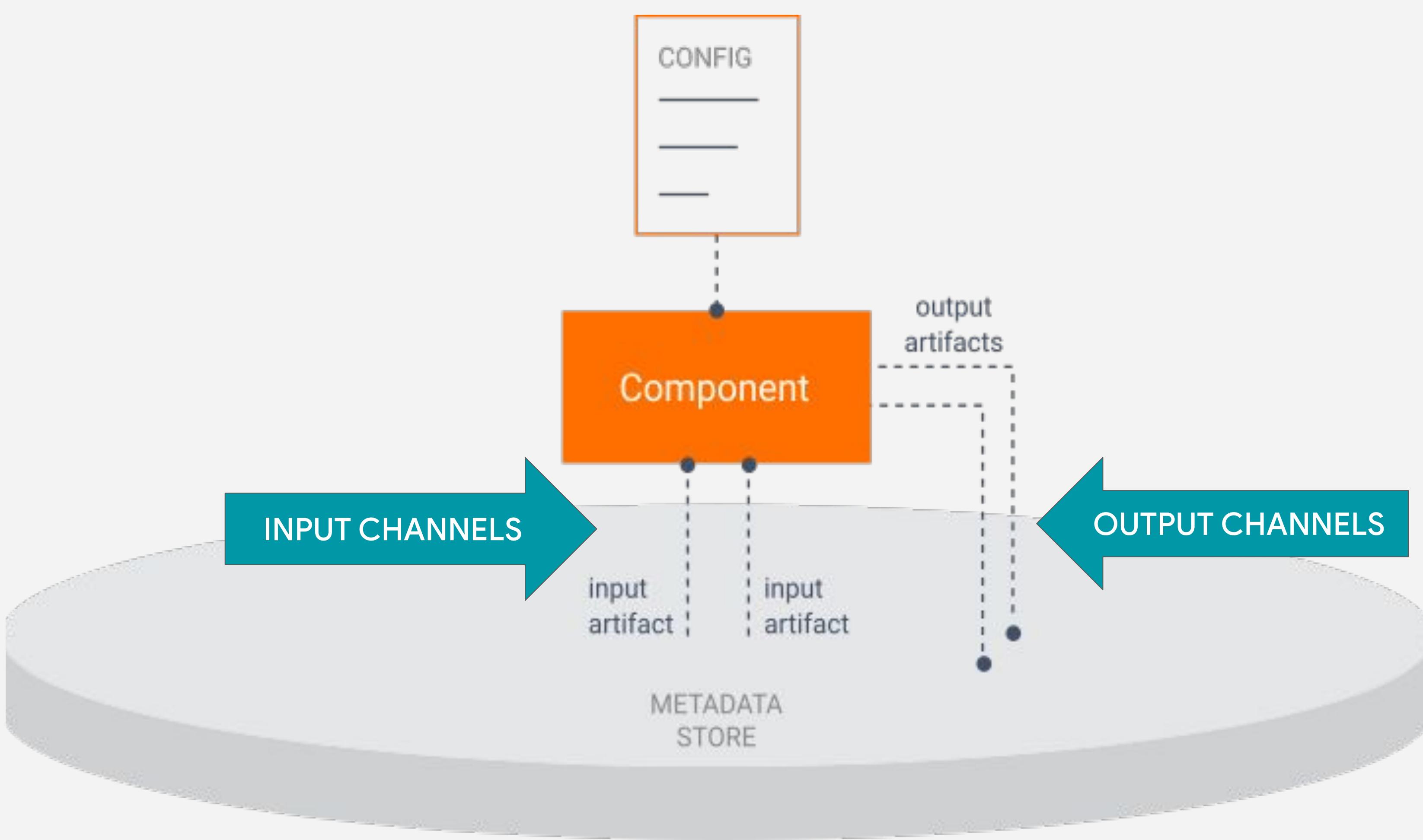
Hello TFX



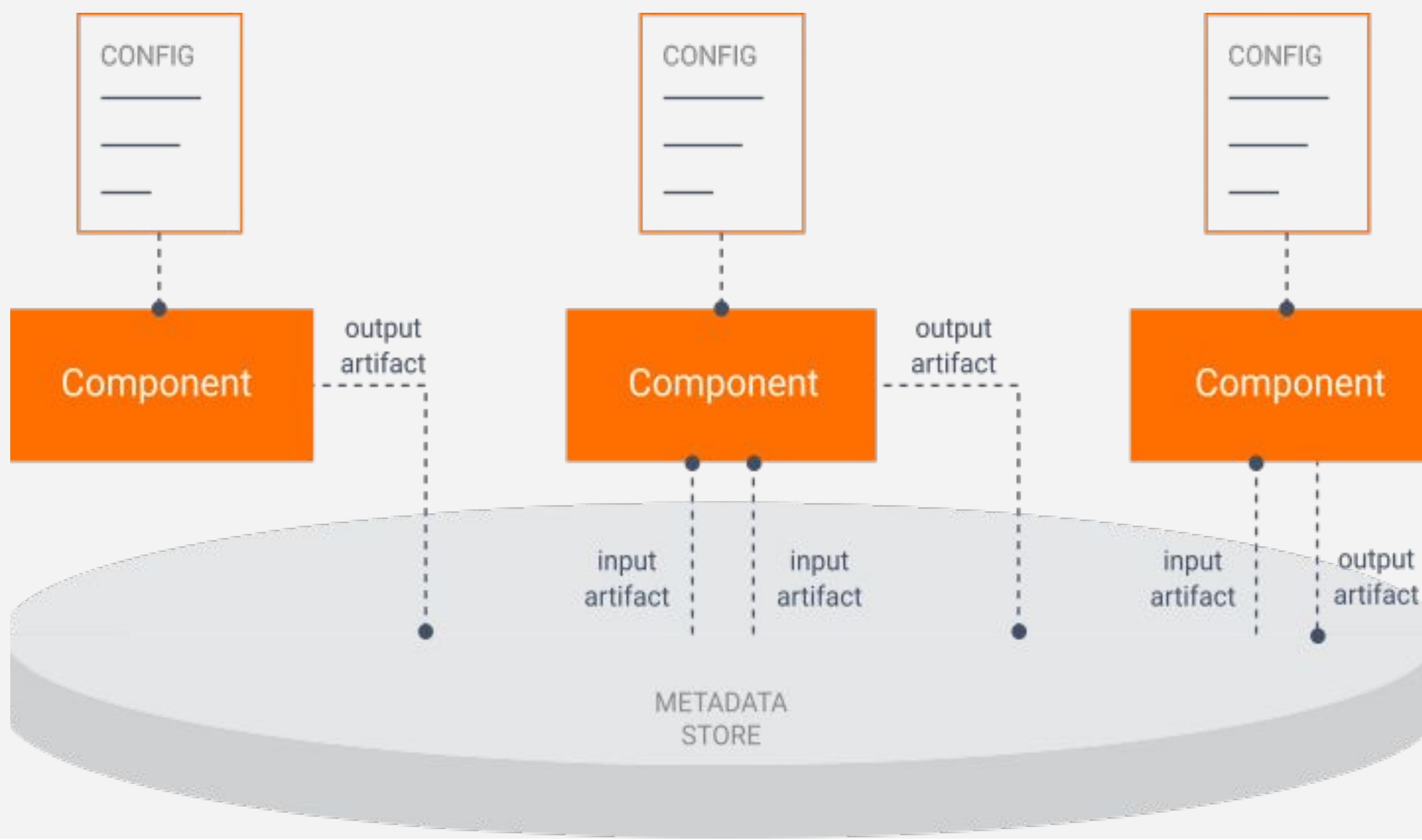
What is a TFX component?



What makes a Component



What makes a Component?



What makes a Component?

TFX Orchestration



What do we mean by portability?

TFX runs just about anywhere

Execution



Orchestration



Processing





TFX Orchestration in a Notebook

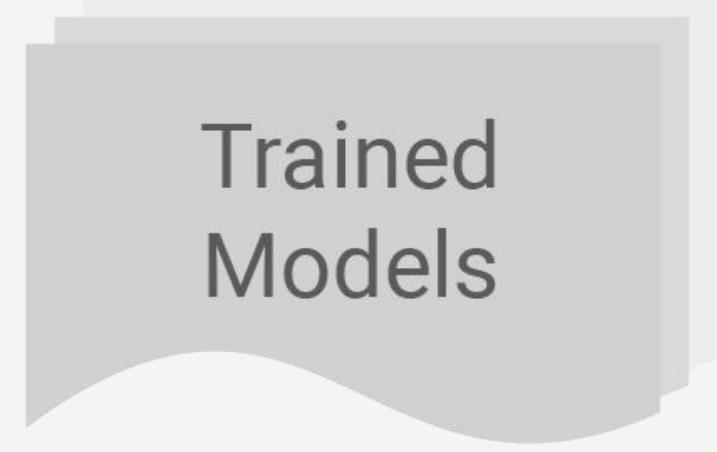
- Environment for experimental and iterative development
- Build up your pipeline iteratively in a Jupyter / Colab notebook and export to production with minimal changes
- InteractiveContext object handles component execution, metadata management, and artifact visualization
- In production, you would use Airflow, Kubeflow, or something similar

```
context = InteractiveContext()

component = MyComponent(...)
context.run(component)
context.show(component.outputs['my_output'])
```

Metadata Store

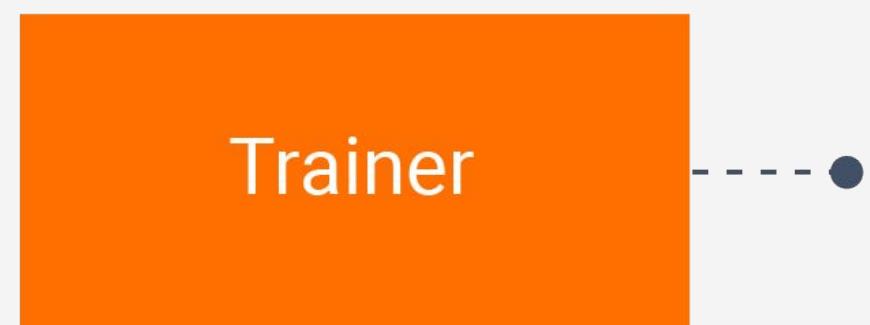
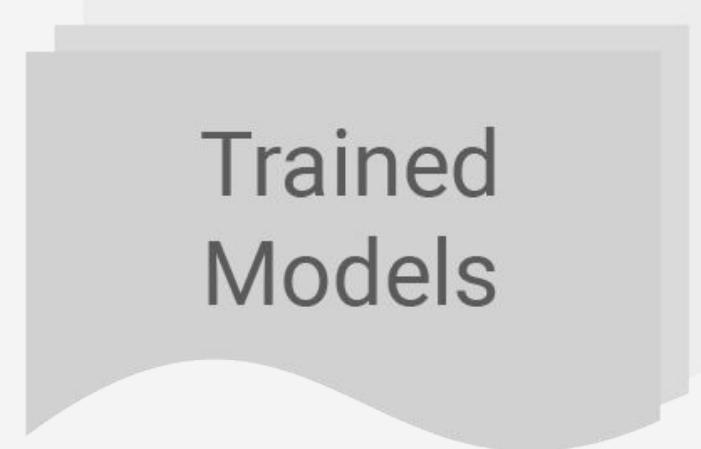
What is in Metadata Store?



Trained
Models

Type definitions of Artifacts and their Properties

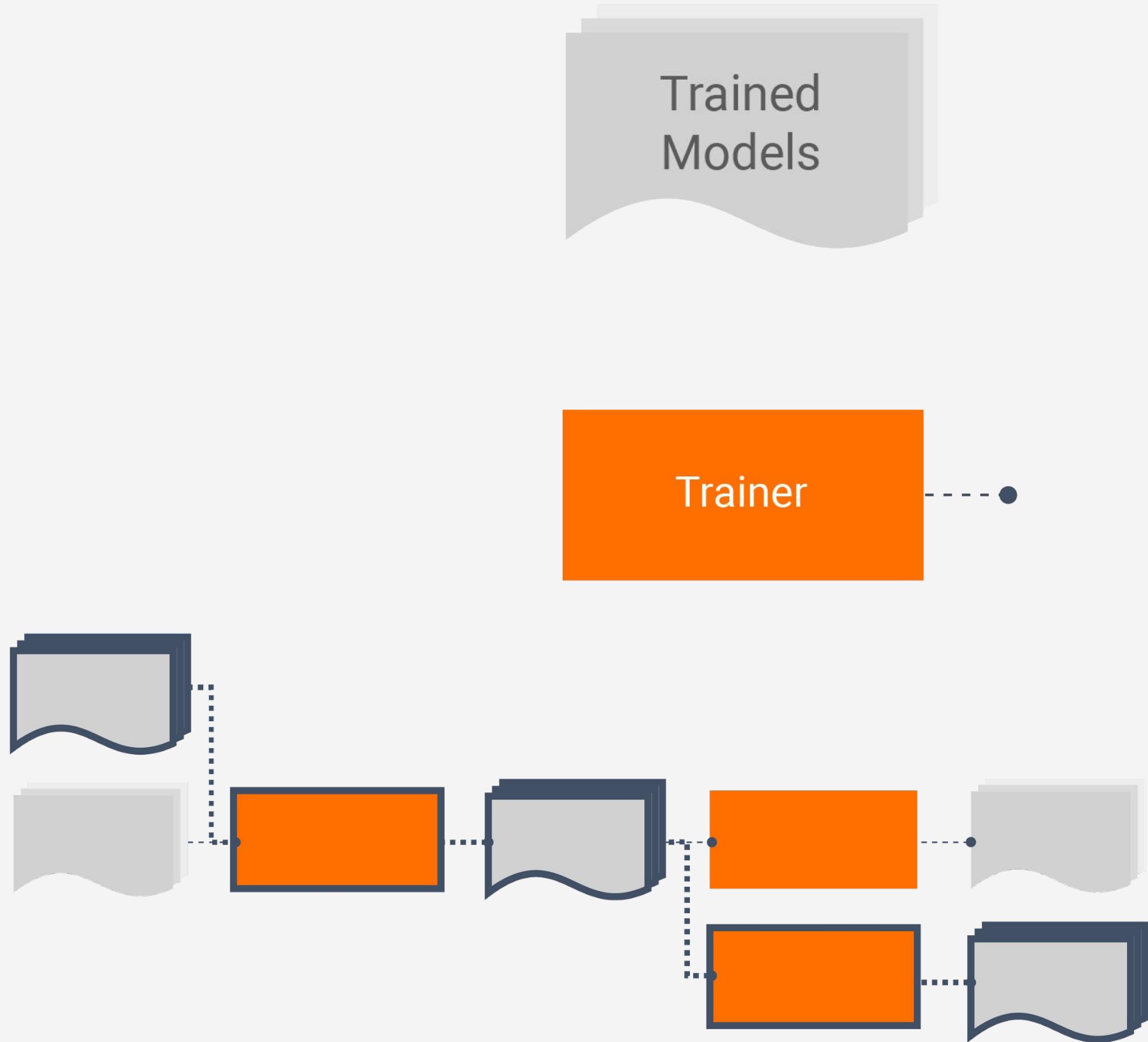
What is in Metadata Store?



Type definitions of Artifacts and their Properties

Execution Records (Runs) of Components

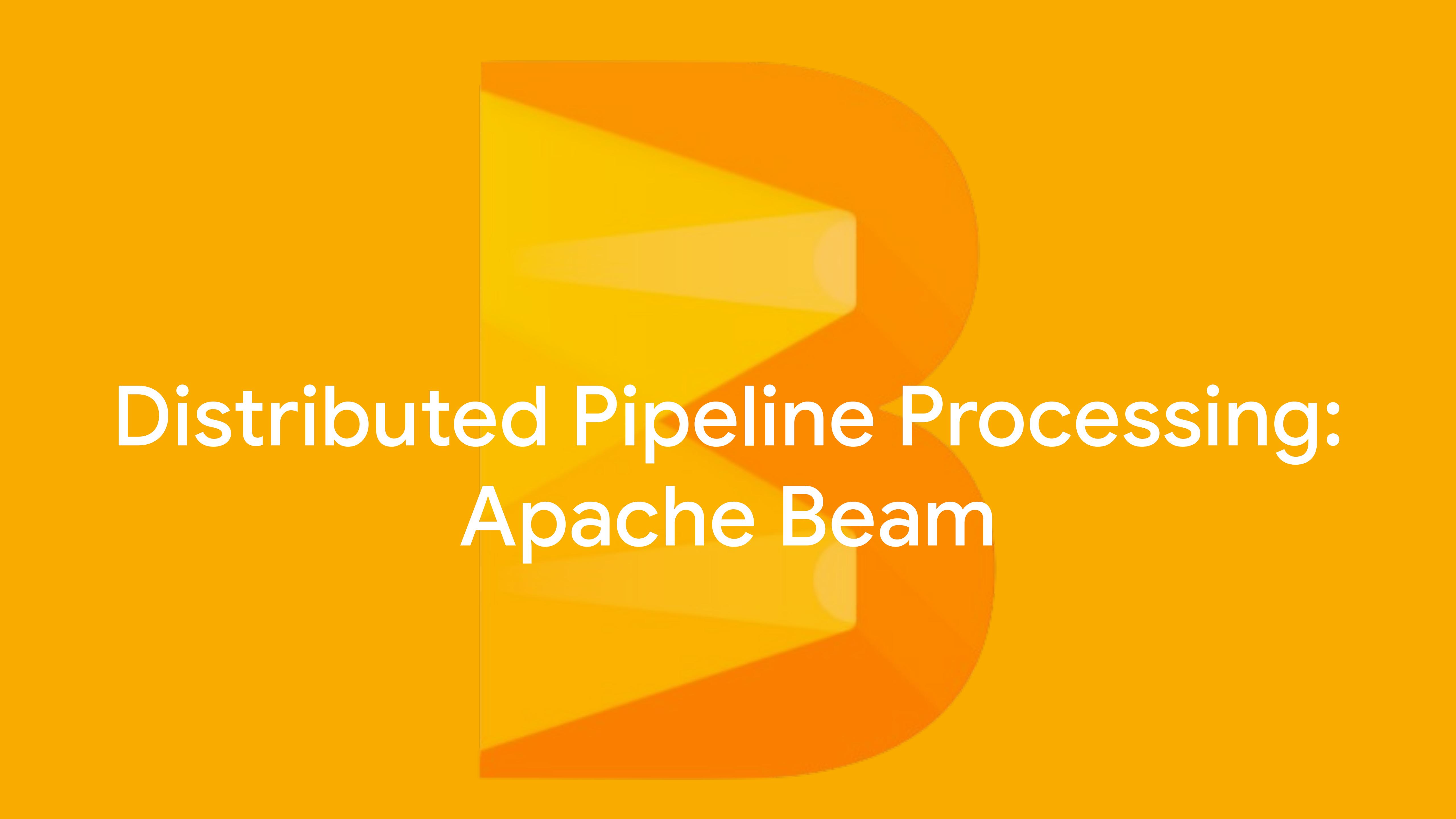
What is in Metadata Store?



Type definitions of Artifacts and their Properties

Execution Records (Runs) of Components

Data Provenance Across All Executions



Distributed Pipeline Processing: Apache Beam



What is Apache Beam?

- A unified **batch** and stream distributed processing API
- A set of **SDK frontends**: Java, **Python**, Go, Scala, SQL
- A set of **Runners** which can execute Beam jobs into various backends: **DirectRunner**, **Apache Flink**, **Apache Spark**, Apache Gearpump, **Apache Samza**, Apache Hadoop, **Google Cloud Dataflow**, ...



Apache Beam

Java

```
input.apply(  
    Sum.integersPerKey())
```

Python

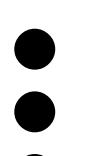
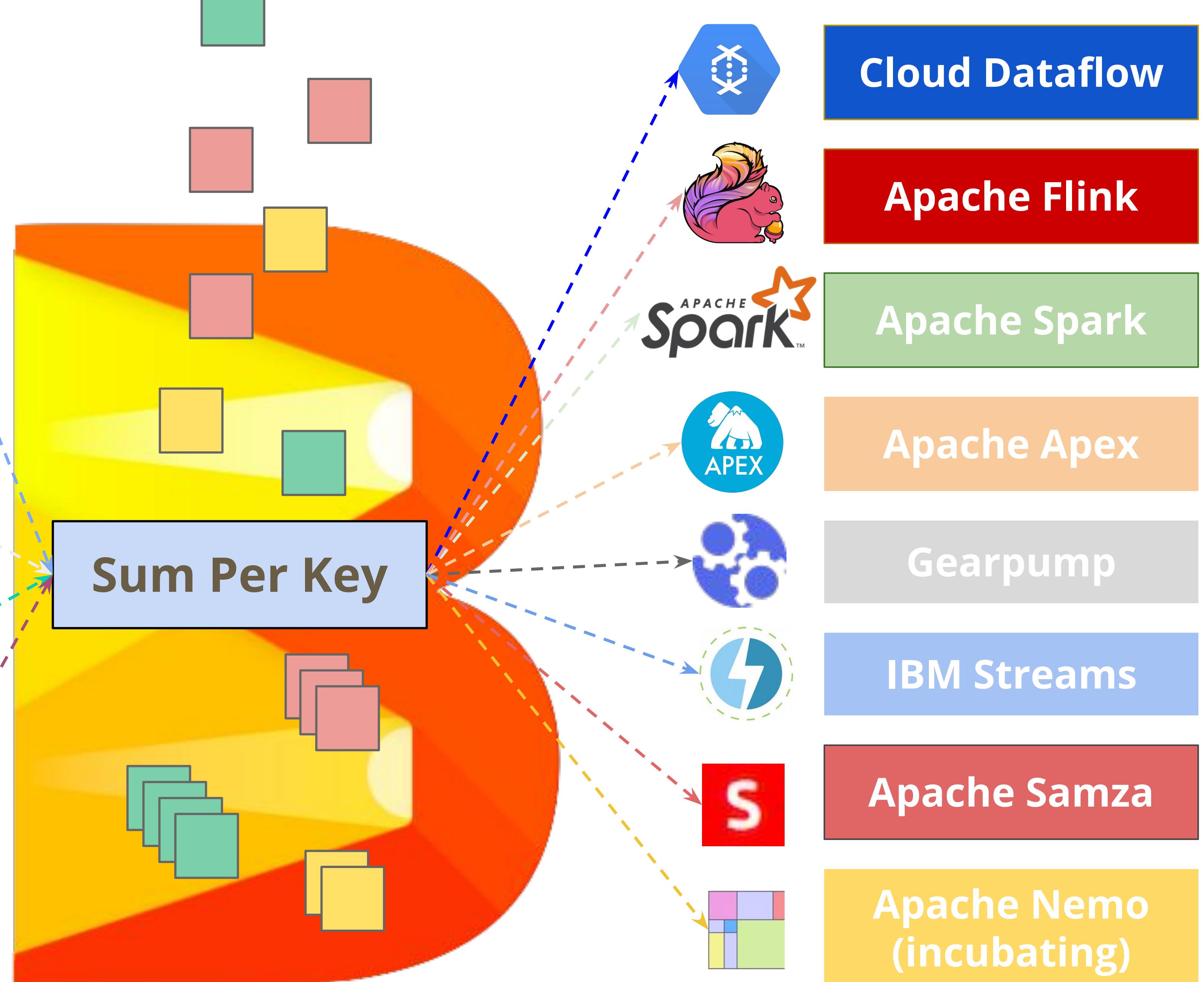
```
input | Sum.PerKey()
```

Go

```
stats.Sum(s, input)
```

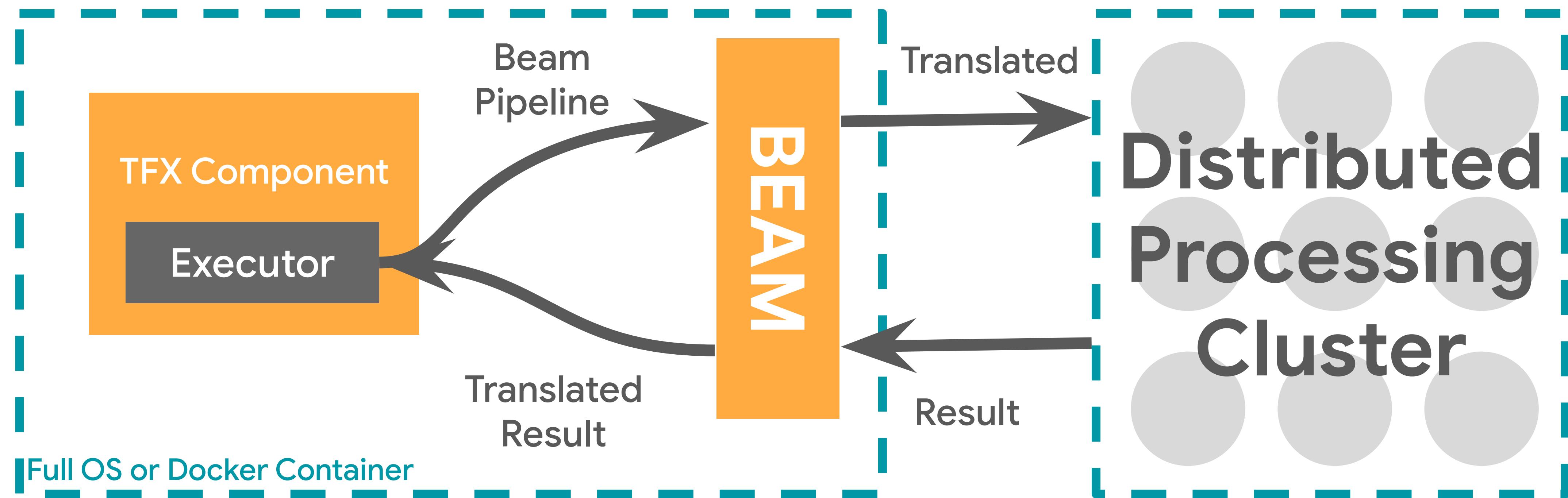
SQL

```
SELECT key, SUM(value)  
FROM input GROUP BY key
```





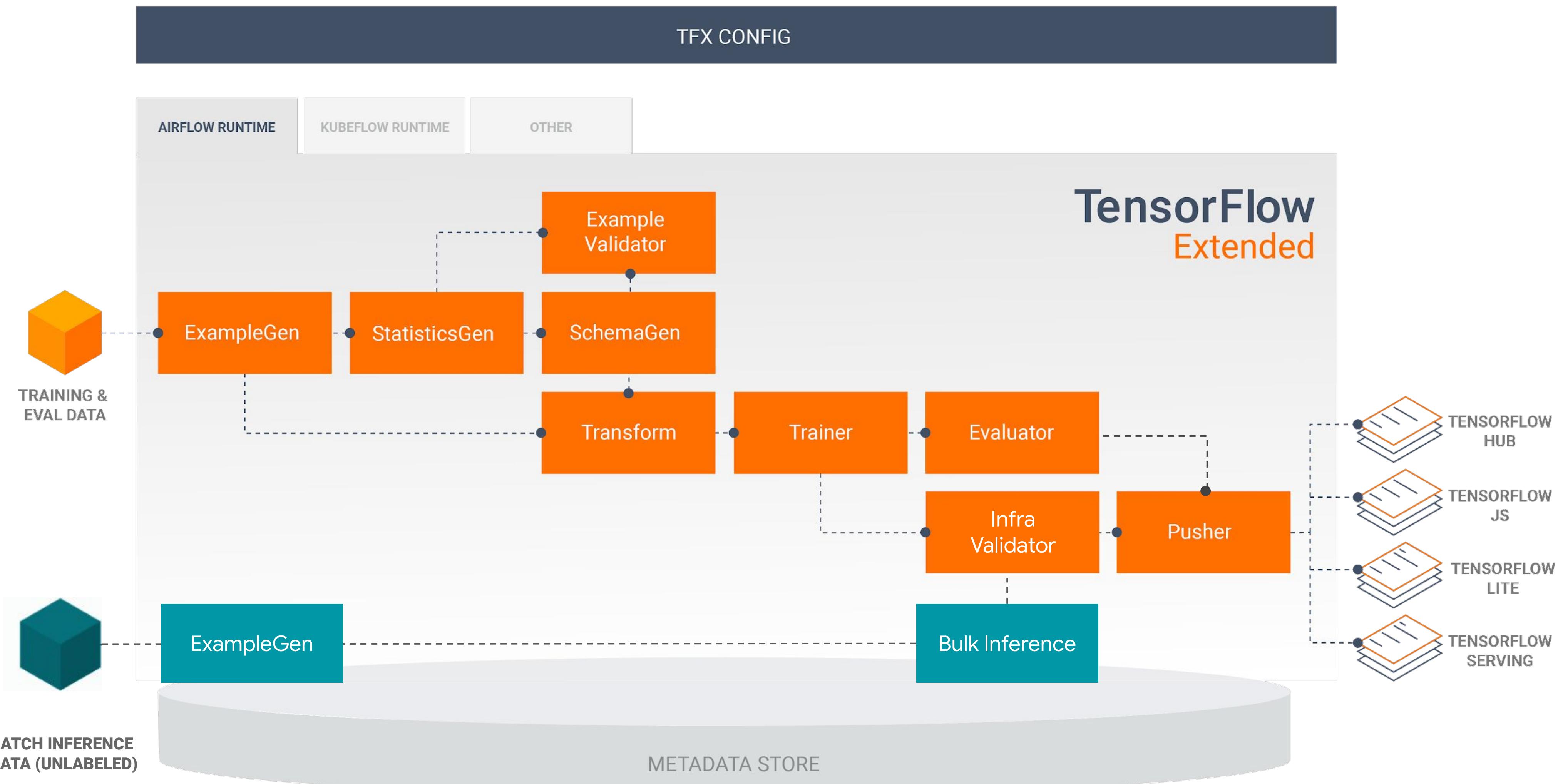
How TFX Components Use Beam



TFX Standard Components



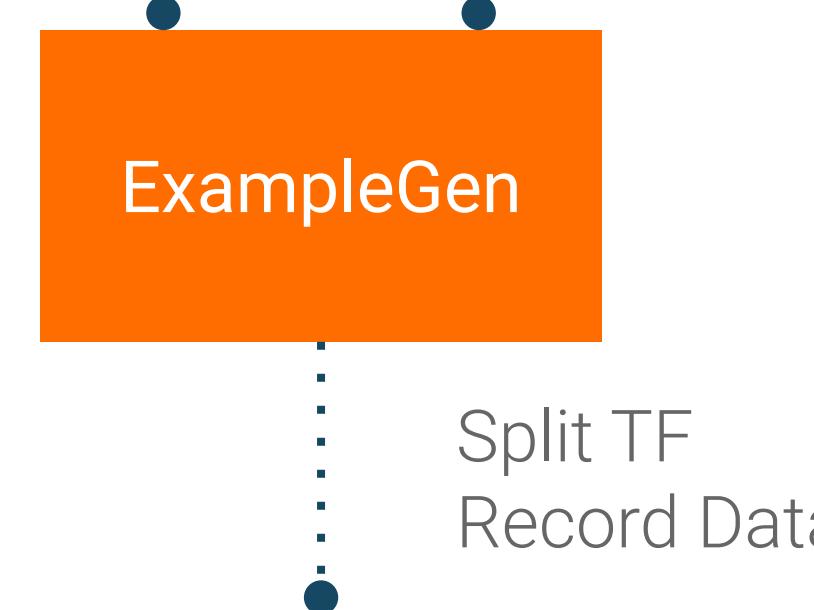
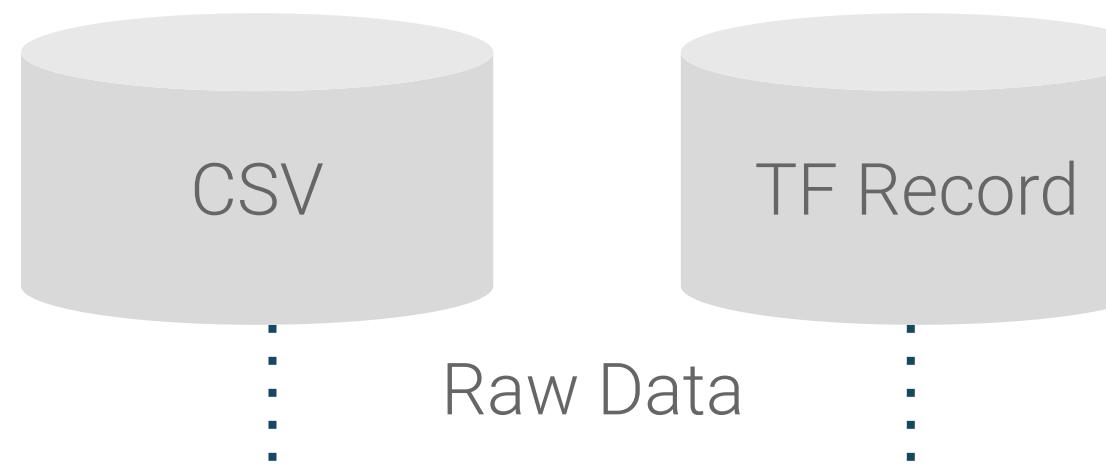
Hello TFX





Component: ExampleGen

Inputs and Outputs



Configuration

```
examples = csv_input(os.path.join(base_dir, 'data/simple'))  
example_gen = CsvExampleGen(input=examples)
```

Standard Formats

- CSV
- tf.Record
- BigQuery

Custom Formats

- Avro
- Parquet
- Presto

Example 1

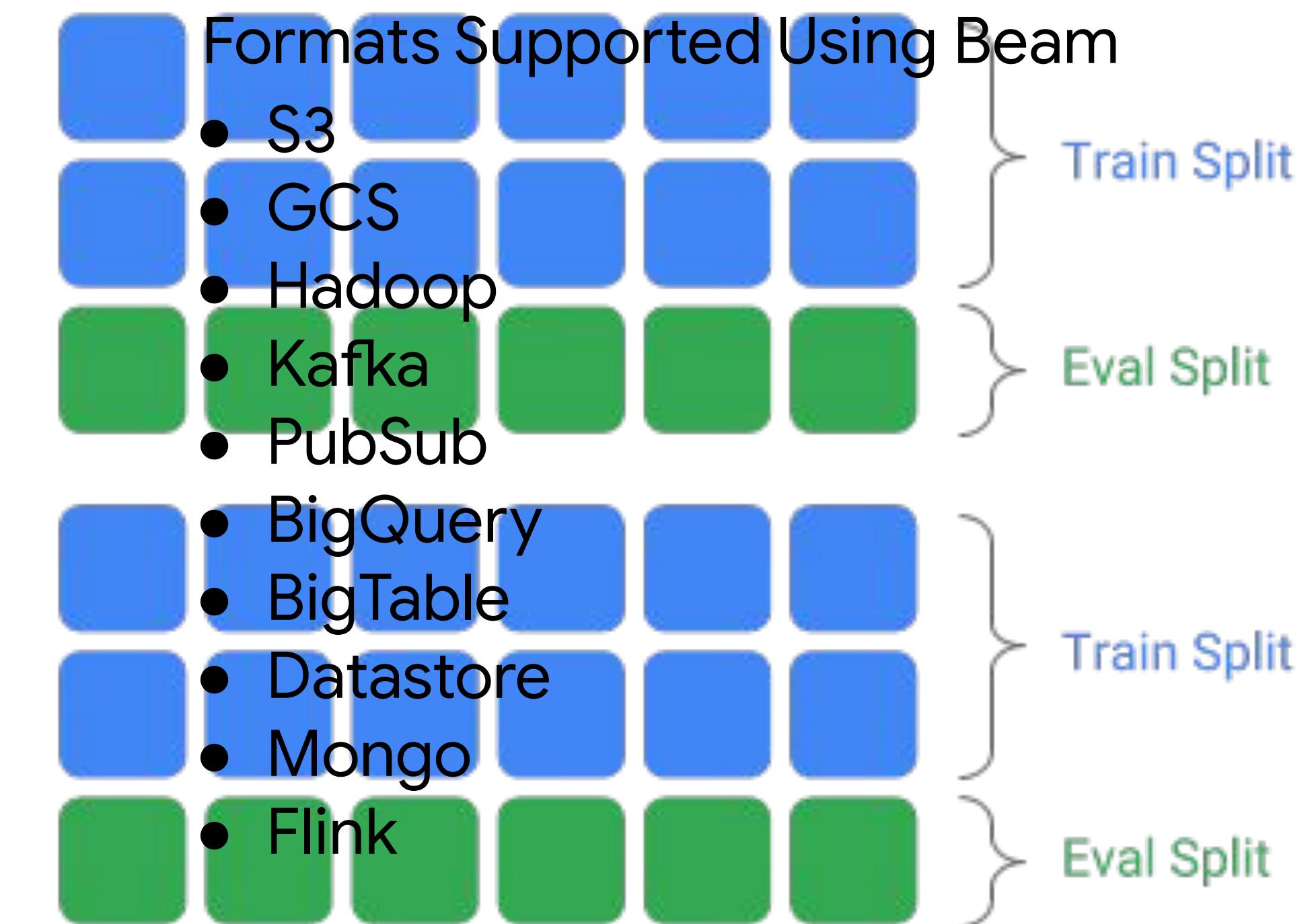
Example 2

Example 3

Example 4

Example 5

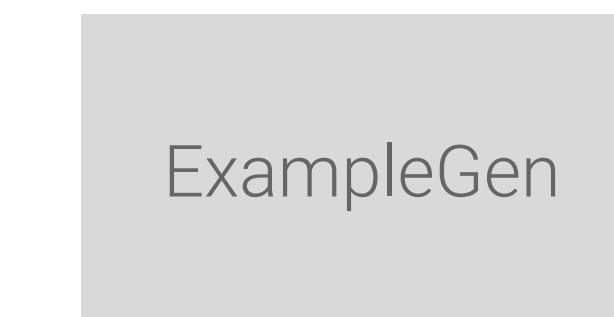
Example 6





Component: StatisticsGen

Inputs and Outputs



Data



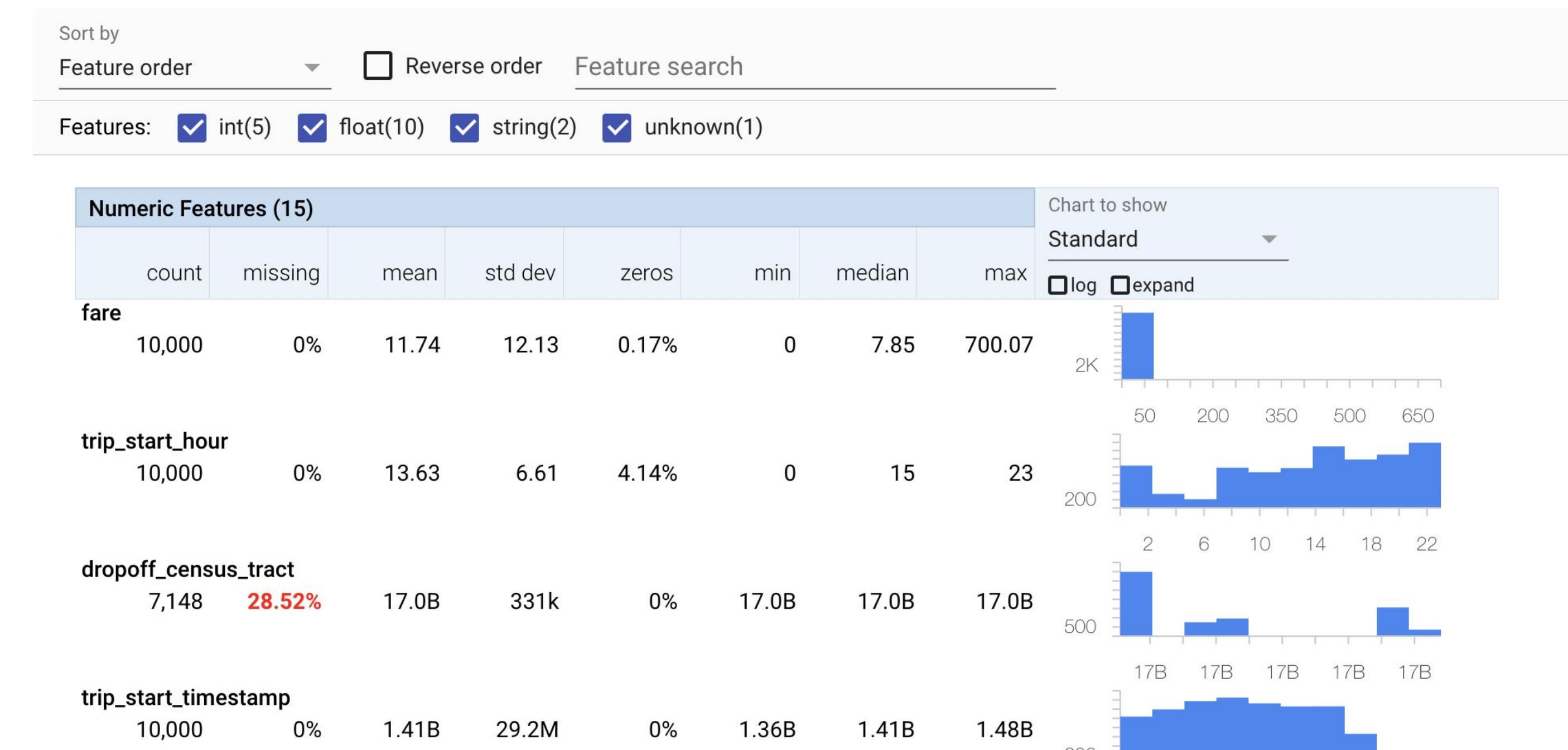
Statistics



Configuration

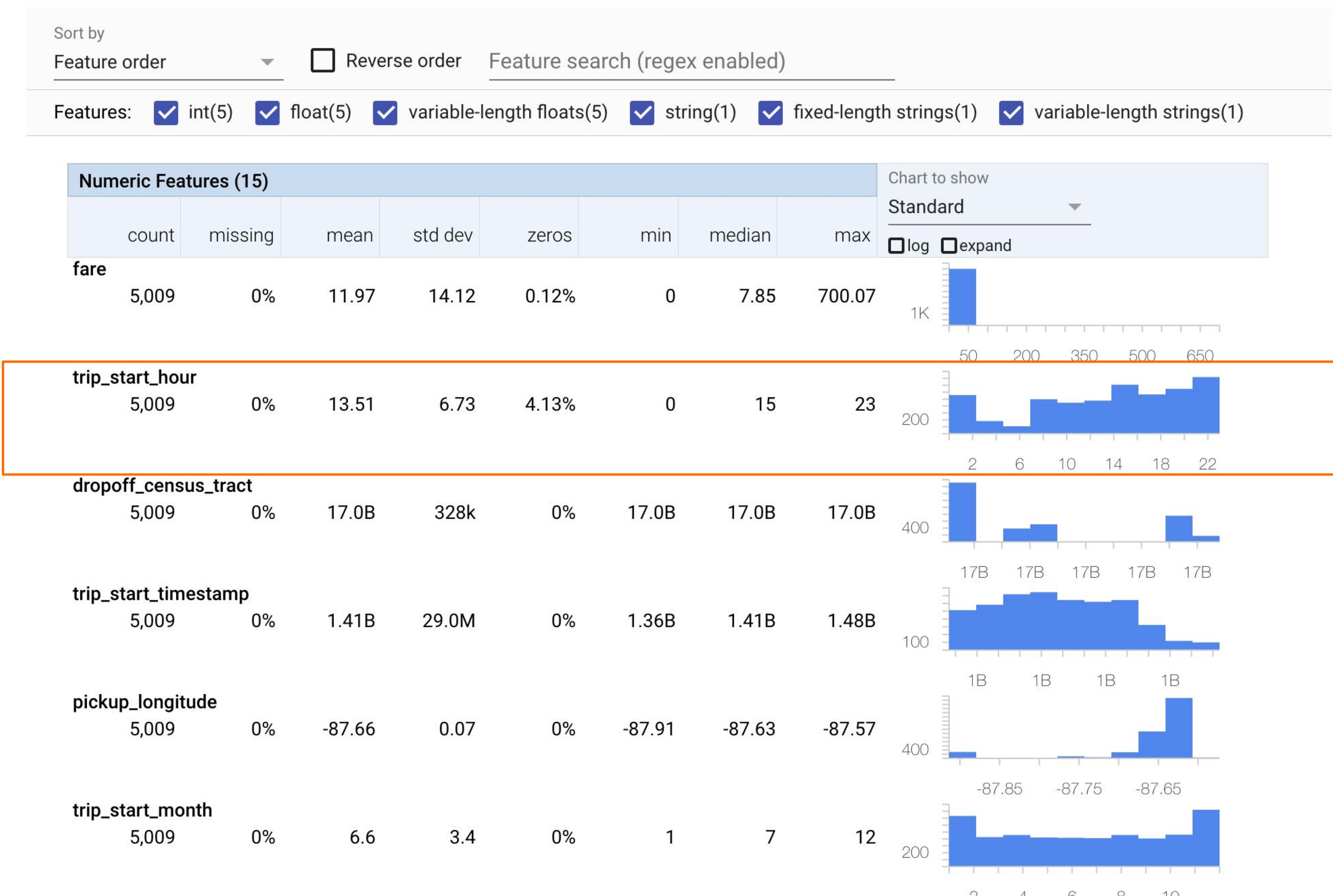
```
statistics_gen = StatisticsGen(  
    input_data=example_gen.outputs['examples'])
```

Visualization





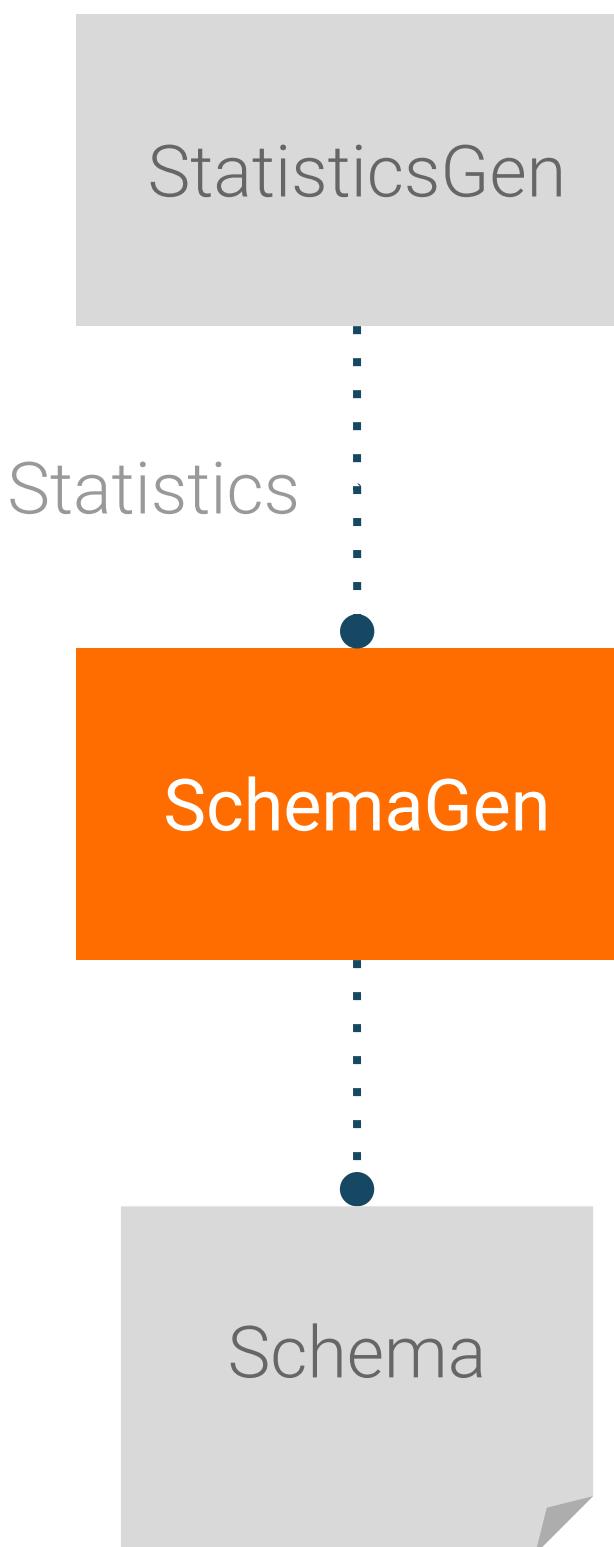
Analyzing Data with TensorFlow Data Validation





Component: SchemaGen

Inputs and Outputs



Configuration

```
infer_schema =  
    SchemaGen(statistics=statistics_gen.outputs['statistics'],  
               infer_feature_shape=False)
```

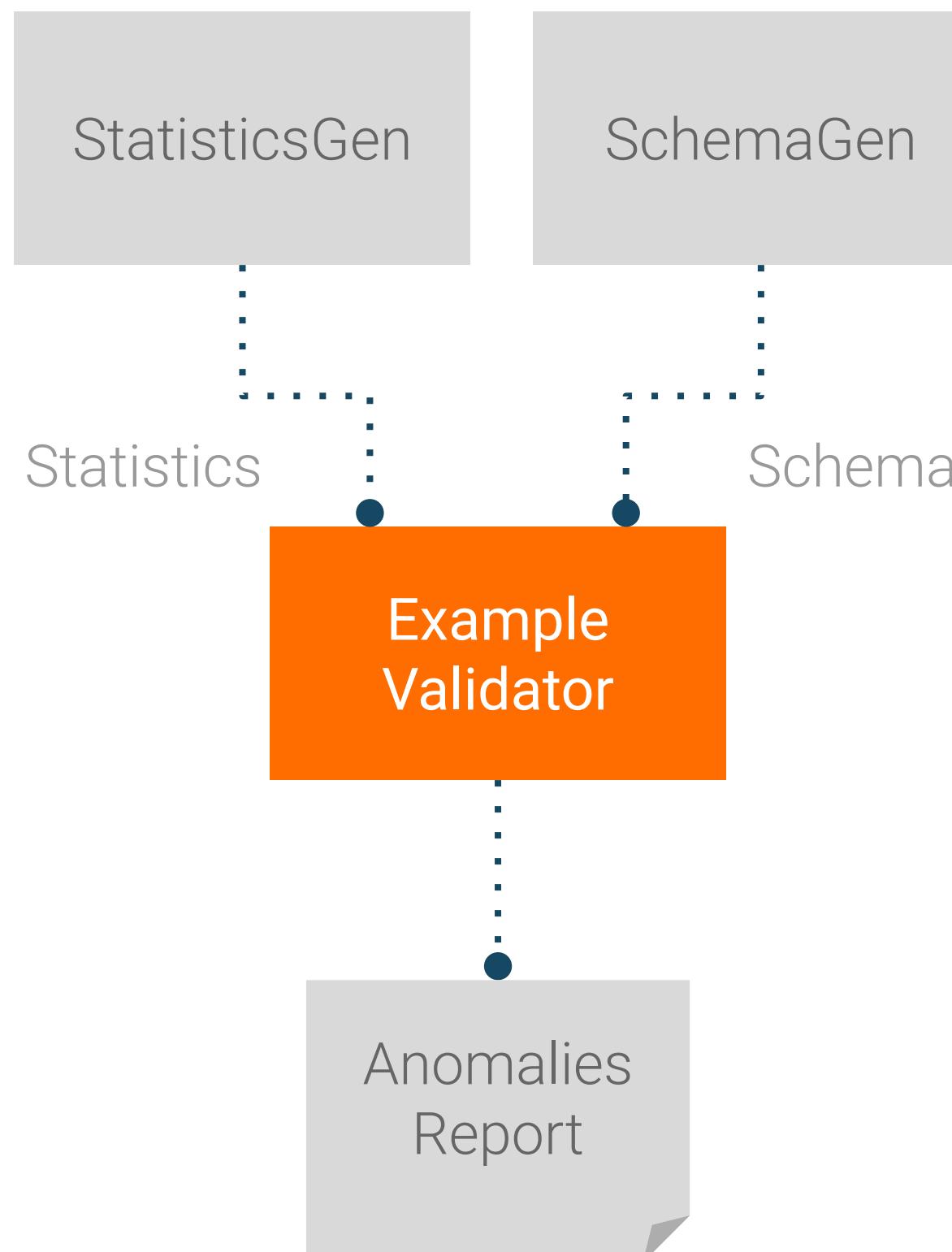
Visualization

Feature name	Type	Presence	Valency	Domain
'fare'	FLOAT	required	single	-
'trip_start_hour'	INT	required	single	-
'pickup_census_tract'	BYTES	optional	-	-
'dropoff_census_tract'	FLOAT	optional	single	-
'company'	STRING	optional	single	'company'



Component: ExampleValidator

Inputs and Outputs



Configuration

```
validate_stats = ExampleValidator(  
    statistics=statistics_gen.outputs['statistics'],  
    schema=infer_schema.outputs['schema'])
```

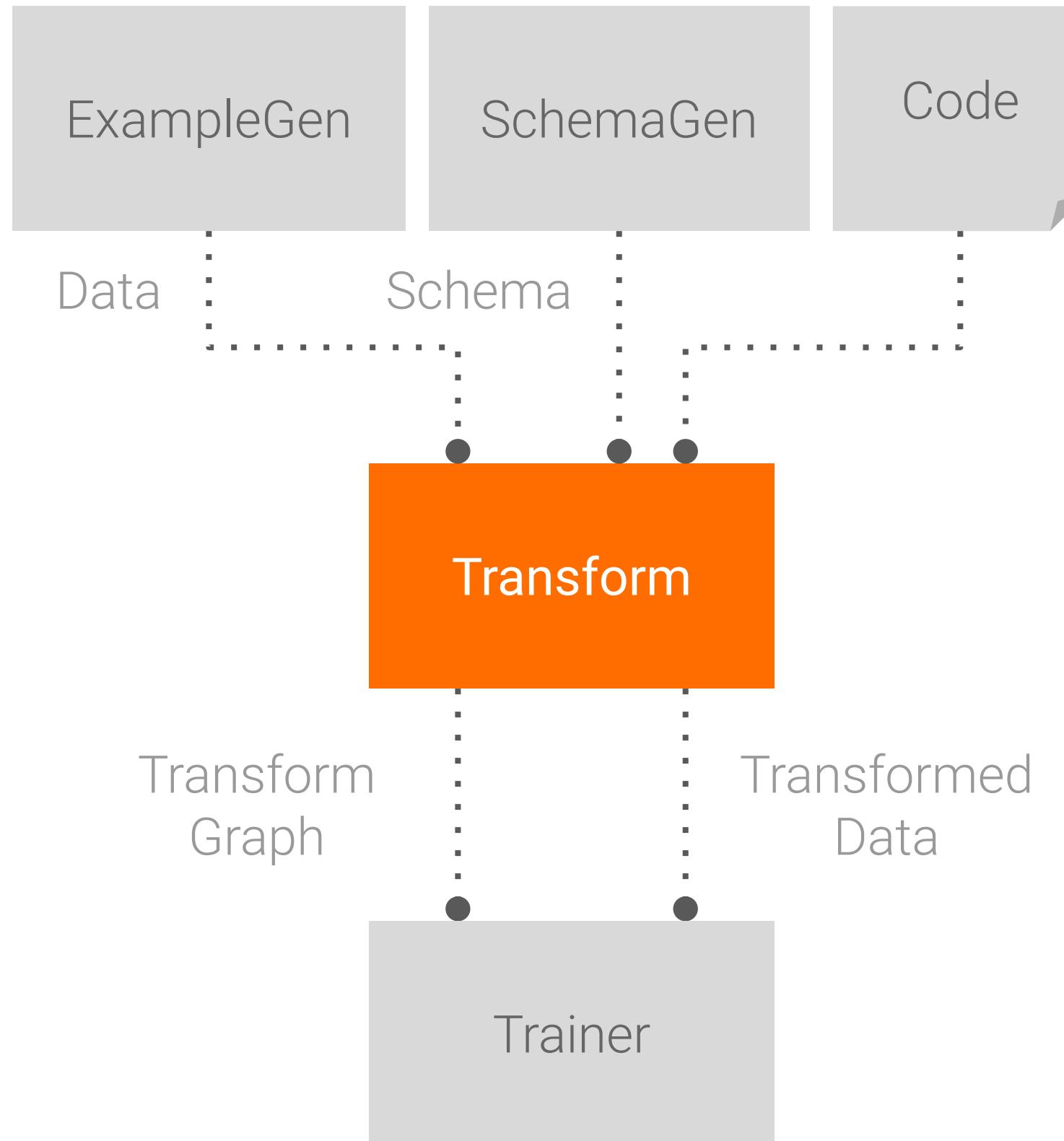
Visualization

Feature name	Anomaly short description	Anomaly long description
'payment_type'	Unexpected string values	Examples contain values missing from the schema: Prcard (<1%).
'company'	Unexpected string values	Examples contain values missing from the schema: 2092 - 61288 Sbeih company (<1%), 2192 - 73487 Zeymane Corp (<1%), 2192 - Zeymane Corp (<1%), 2823 - 73307 Seung Lee (<1%), 3094 - 24059 G.L.B. Cab Co (<1%), 3319 - CD Cab Co (<1%), 3385 - Eman Cab (<1%), 3897 - 57856 Ilie Malec (<1%), 4053 - 40193 Adwar H. Nikola (<1%), 4197 - Royal Star (<1%), 585 - 88805 Valley Cab Co (<1%), 5874 - Sergey Cab Corp. (<1%), 6057 - 24657 Richard Addo (<1%), 6574 - Babylon Express Inc. (<1%), 6742 - 83735 Tasha ride inc (<1%).



Component: Transform

Inputs and Outputs



Configuration

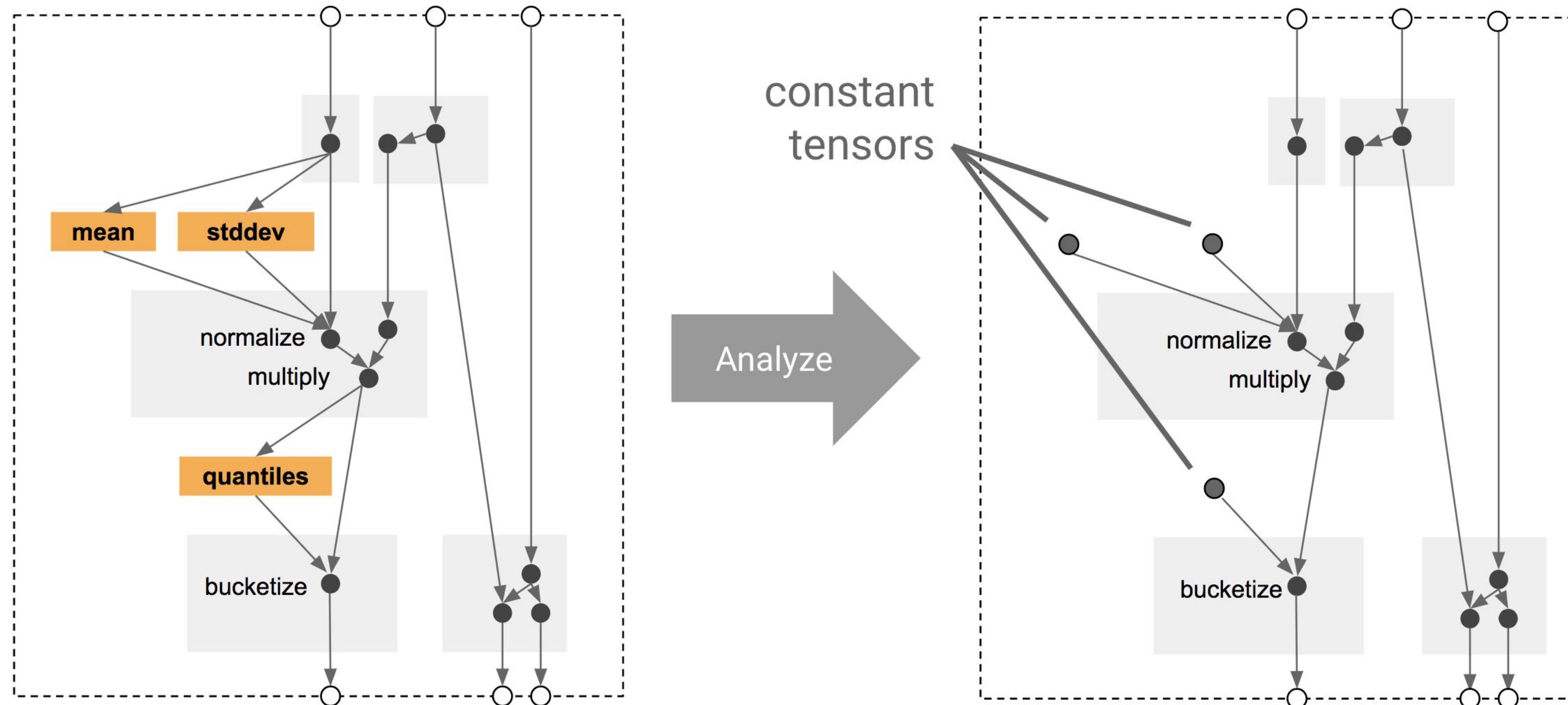
```
transform = Transform(  
    examples=example_gen.outputs['output_data'],  
    schema=infer_schema.outputs['schema'],  
    module_file=module_file)
```

Code

```
for key in _DENSE_FLOAT_FEATURE_KEYS:  
    outputs[_transformed_name(key)] = transform.scale_to_z_score(  
        _fill_in_missing(inputs[key]))  
    # ...  
  
outputs[_transformed_name(_LABEL_KEY)] = tf.where(  
    tf.is_nan(taxi_fare),  
    tf.cast(tf.zeros_like(taxi_fare), tf.int64),  
    # Test if the tip was > 20% of the fare.  
    tf.cast(  
        tf.greater(tips, tf.multiply(taxi_fare, tf.constant(0.2))), tf.int64))  
    # ...
```

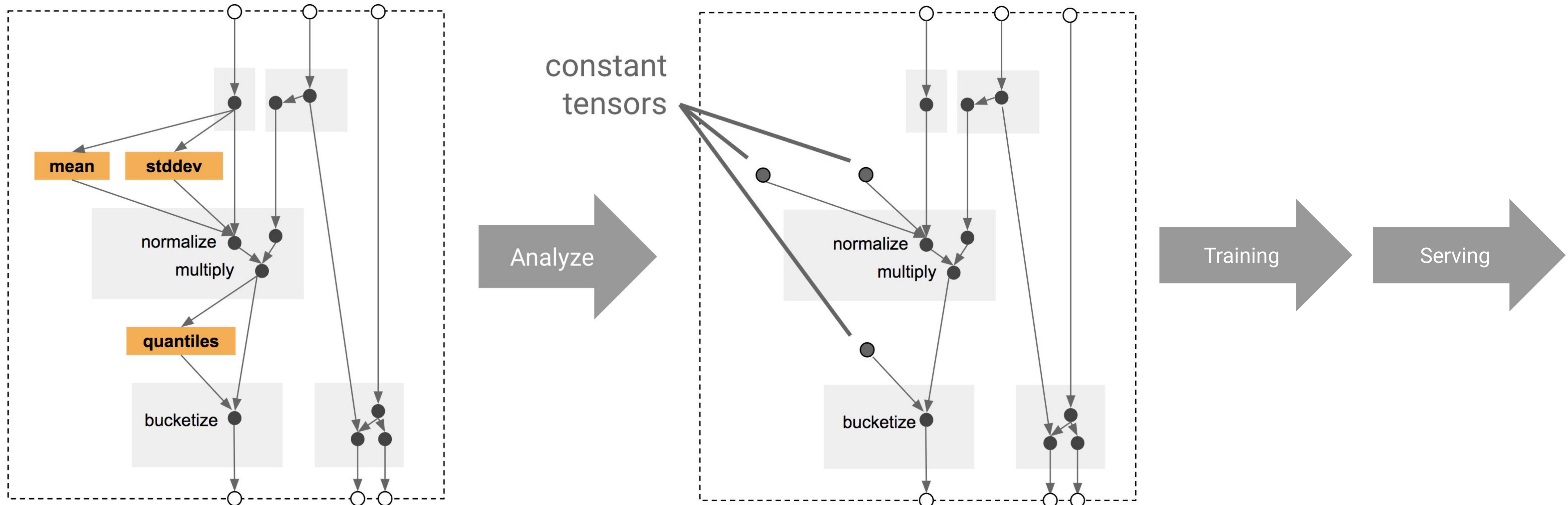


Using TensorFlow Transform for Feature Engineering



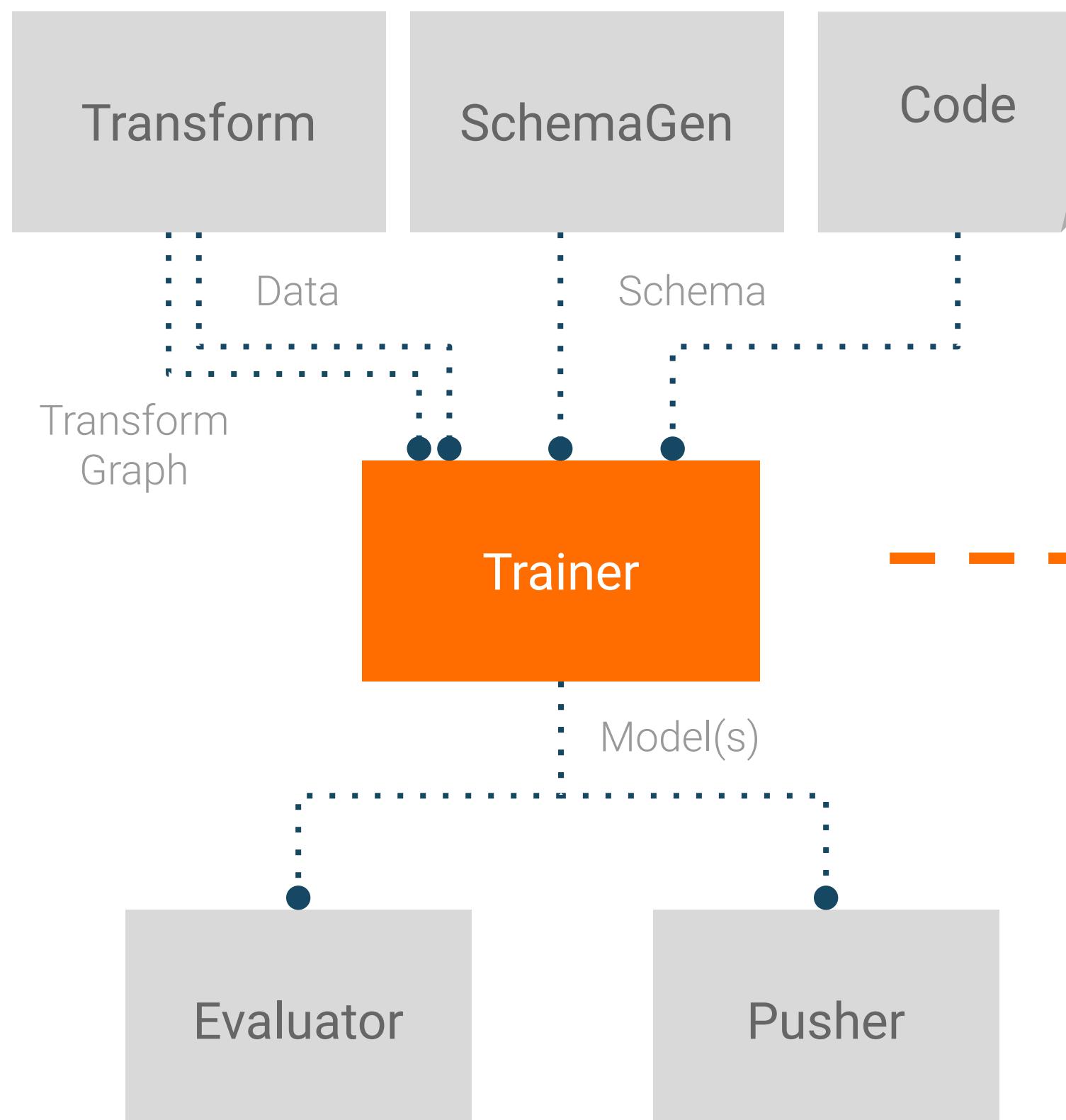


Using TensorFlow Transform for Feature Engineering



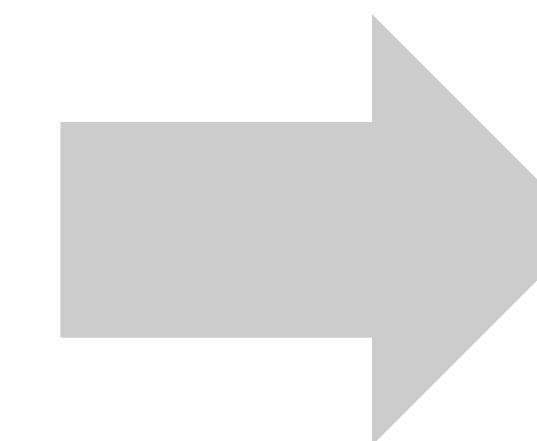
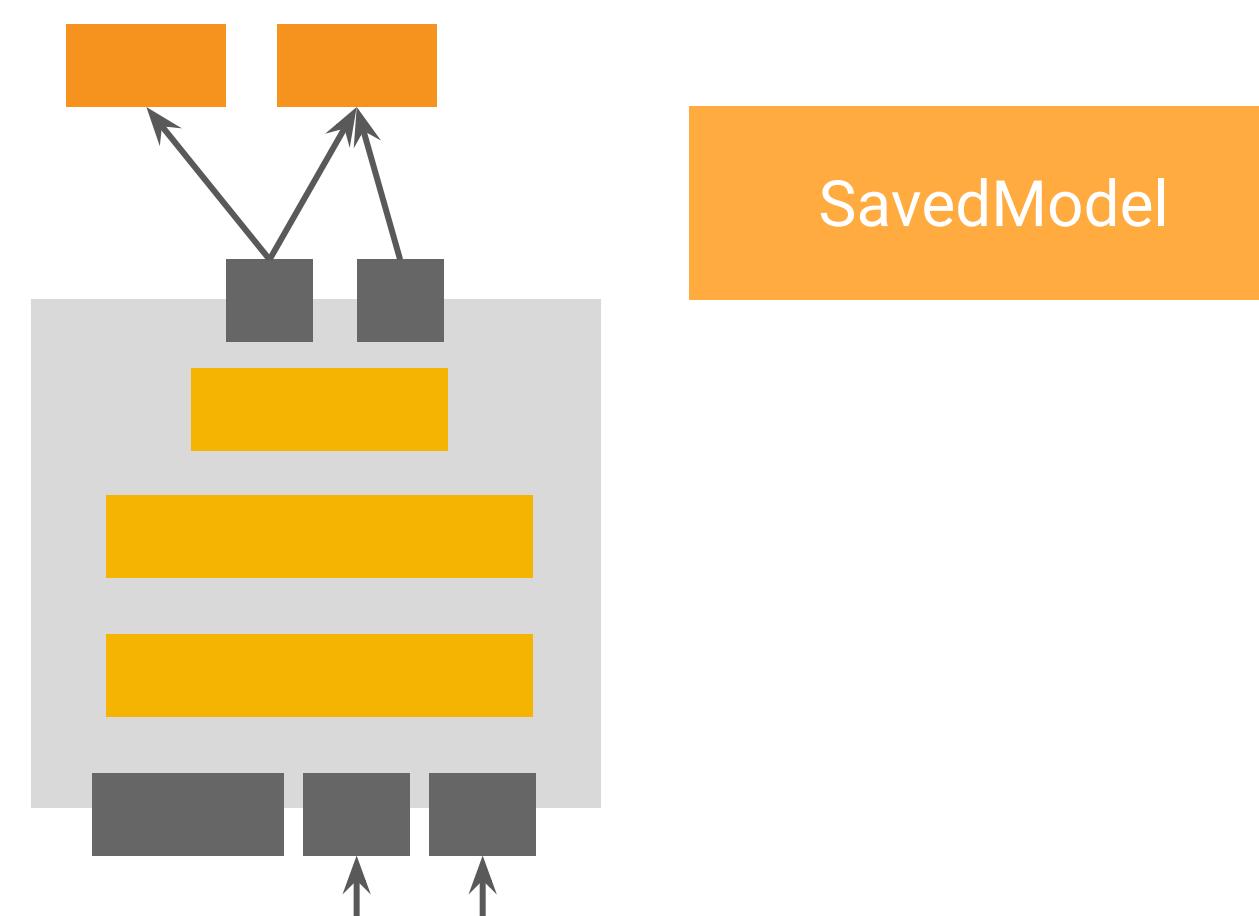
Component: Trainer

Inputs and Outputs



Highlight: SavedModel Format

Train, Eval, and Inference Graphs

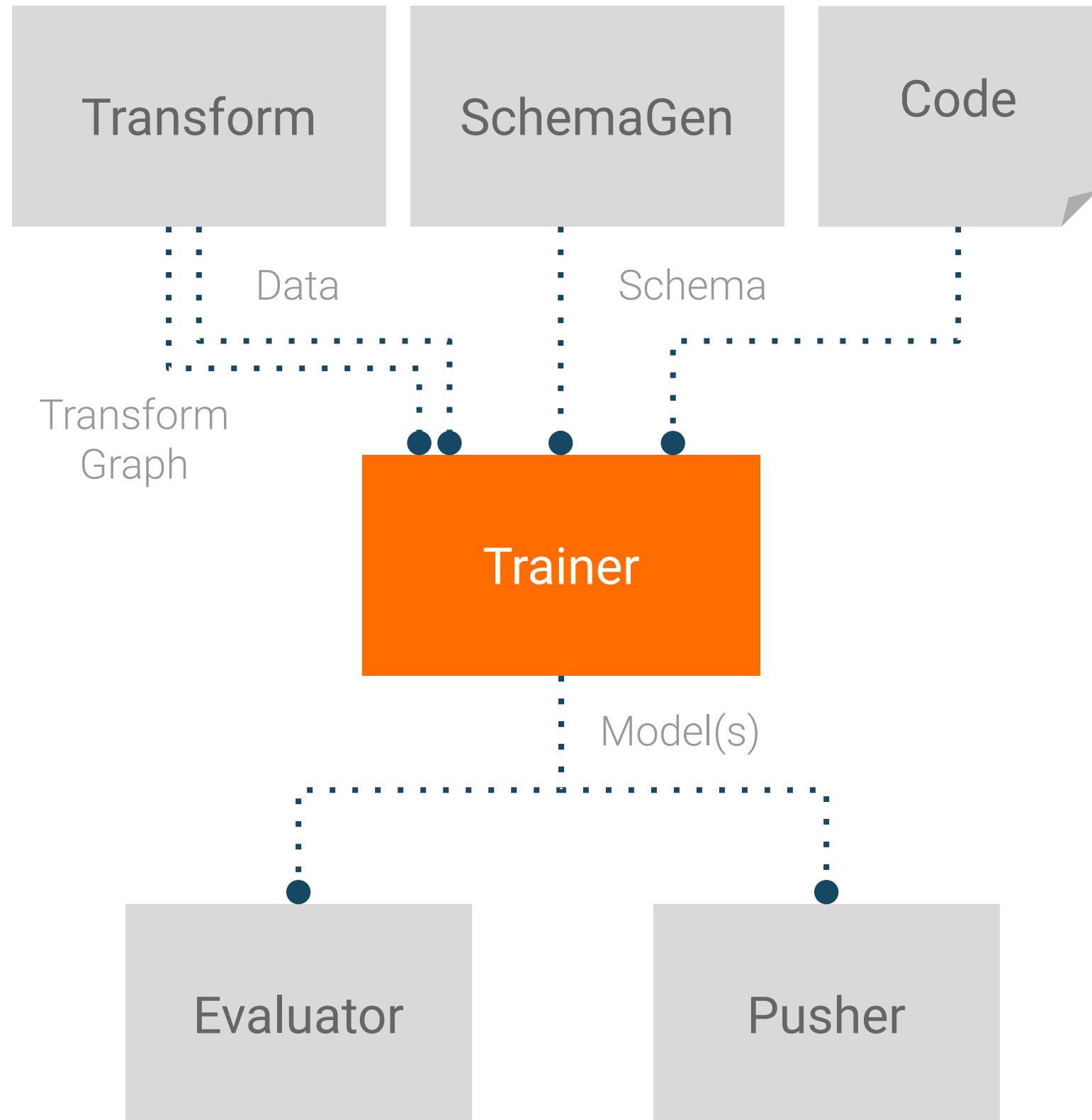


TensorFlow
Model Analysis

TensorFlow
Serving

Component: Trainer

Inputs and Outputs



Configuration

```
trainer = Trainer(  
    module_file=module_file,  
    transformed_examples=  
        transform.outputs['transformed_examples'],  
    schema=infer_schema.outputs['schema'],  
    transform_graph=transform.outputs['transform_graph'],  
    train_args=trainer_pb2.TrainArgs(num_steps=10000),  
    eval_args=trainer_pb2.EvalArgs(num_steps=5000))
```

Code

Your model

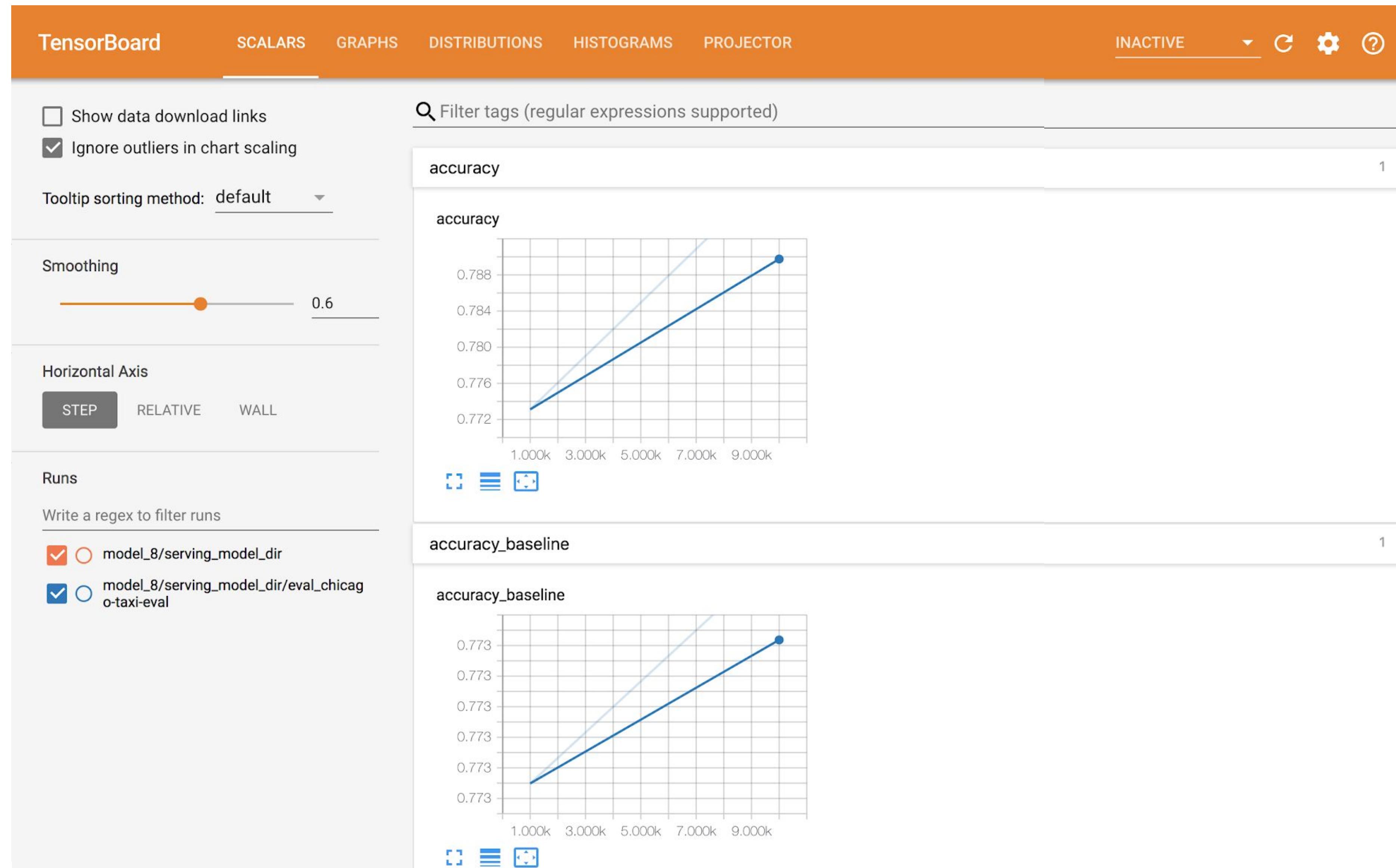
- TensorFlow (Keras or Estimator)
- Sci-Kit Learn
- With Cloud AI Platform
 - XGBoost
 - PyTorch

Advantages to using TensorFlow



```
# Open up Tensorboard for model_id.  
print(display_tensorboard(model_id))
```

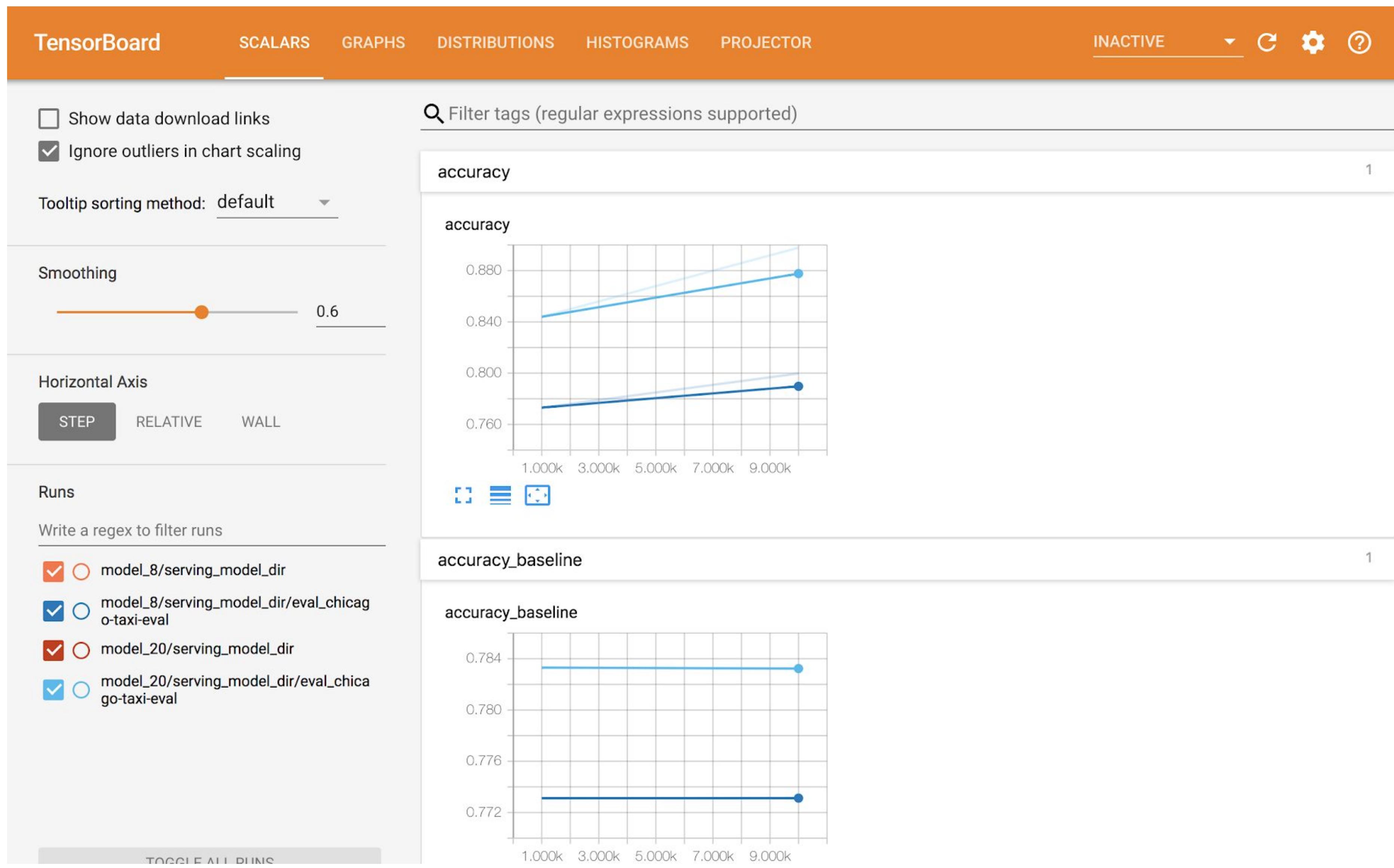
<http://your.host.name:53143>





```
# Compare Tensorboard metrics for different models.  
if num_models > 1:  
    print(display_tensorboard(model_id, other_model_id=other_model_id))
```

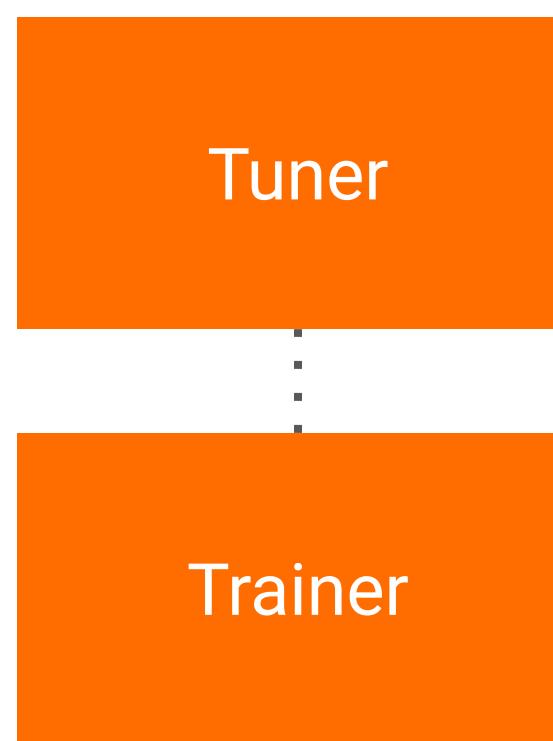
<http://your.host.name:53230>





Component: Tuner

Tuner works with Trainer



Configuration

```
tuner = Tuner(  
    module_file=module_file, # Contains `tuner_fn`.  
    examples=transform.outputs['transformed_examples'],  
    transform_graph=transform.outputs['transform_graph'],  
    train_args=trainer_pb2.TrainArgs(num_steps=20),  
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

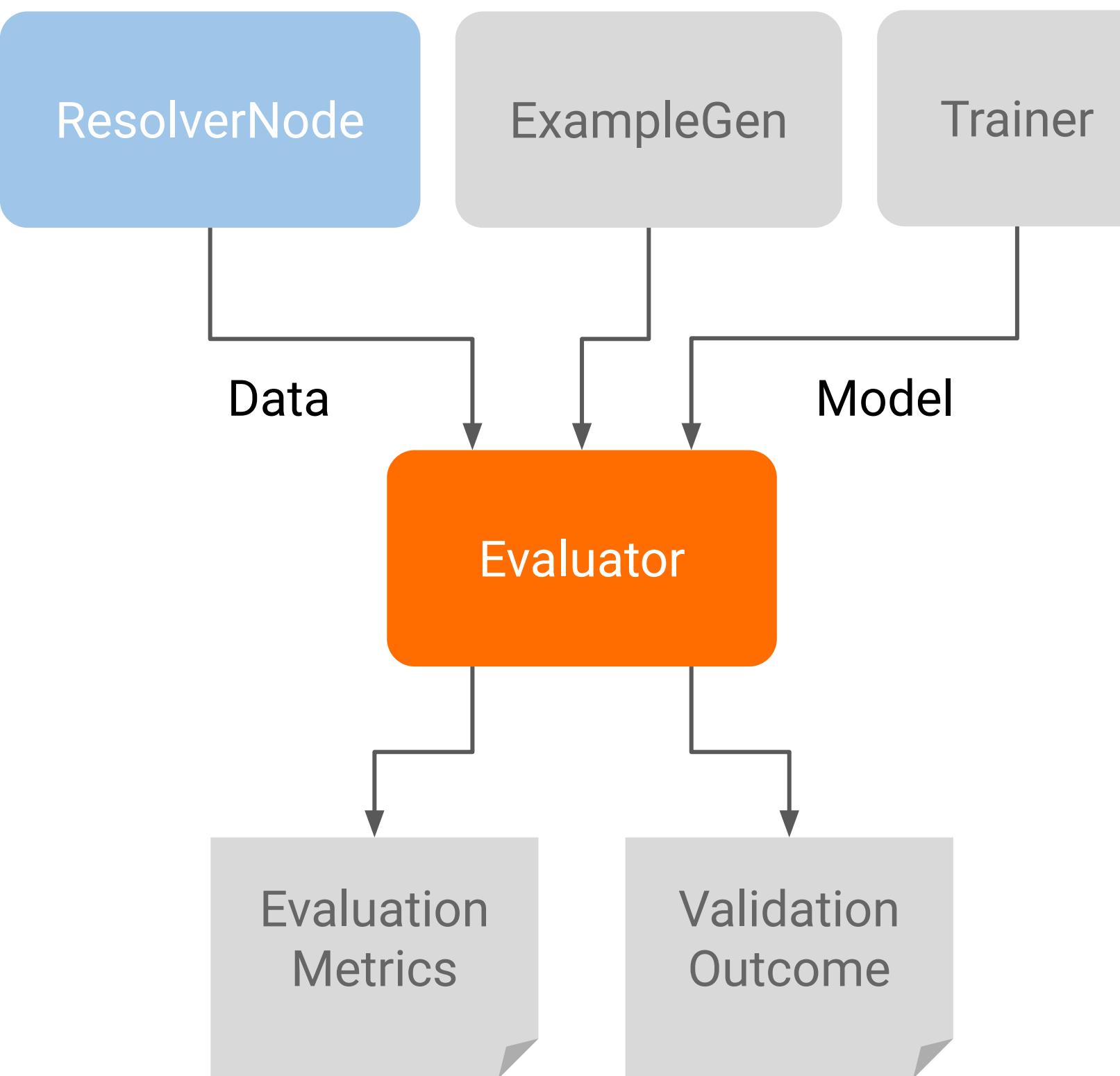
Adding to Pipeline

```
trainer = Trainer(  
    module_file=module_file, # Contains `run_fn`.  
    custom_executor_spec=executor_spec.  
        ExecutorClassSpec(GenericExecutor),  
    examples=transform.outputs['transformed_examples'],  
    transform_graph=transform.outputs['transform_graph'],  
    schema=schema_gen.outputs['schema'],  
    # This will be passed to `run_fn`.  
    hyperparameters=tuner.outputs['best_hyperparameters'],  
    train_args=trainer_pb2.TrainArgs(num_steps=100),  
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```



Component: Evaluator

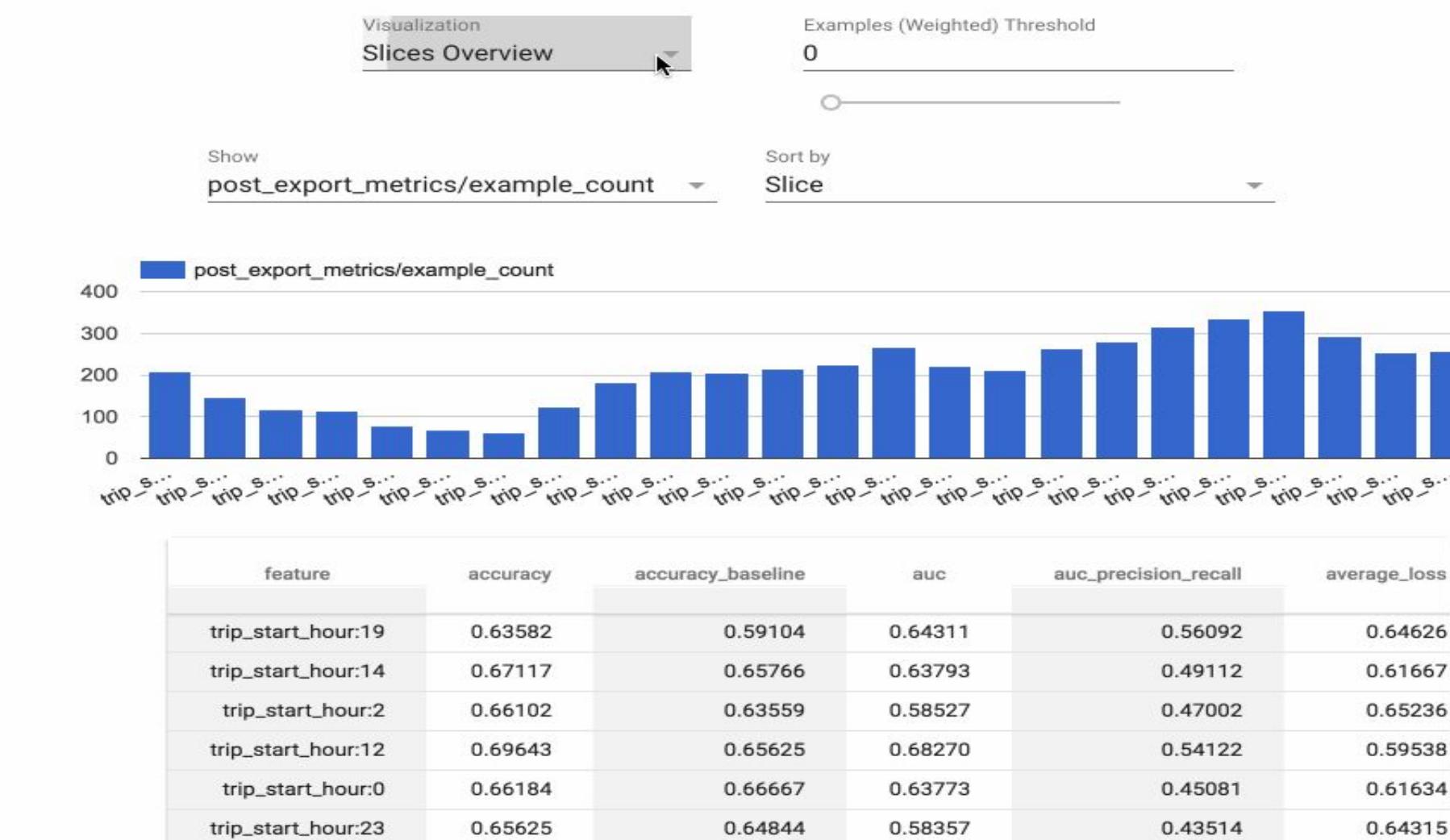
Inputs and Outputs



Configuration

```
evaluator = Evaluator(  
    examples=example_gen.outputs['examples'],  
    model=trainer.outputs['model'],  
    baseline_model=model_resolver.outputs['model'],  
    eval_config=eval_config)
```

Visualization





Visualization

Slices Overview

Examples (Weighted) Threshold

0

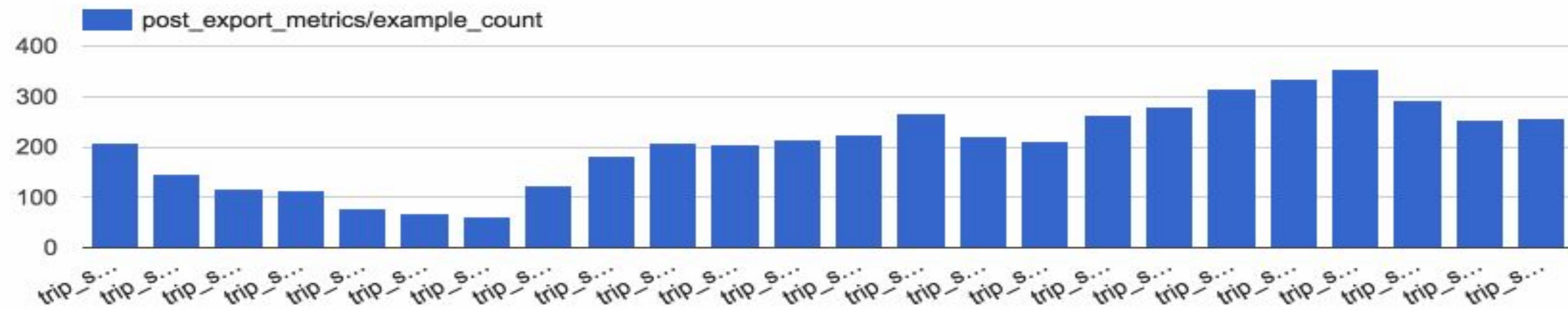


Show

post_export_metrics/example_count

Sort by

Slice

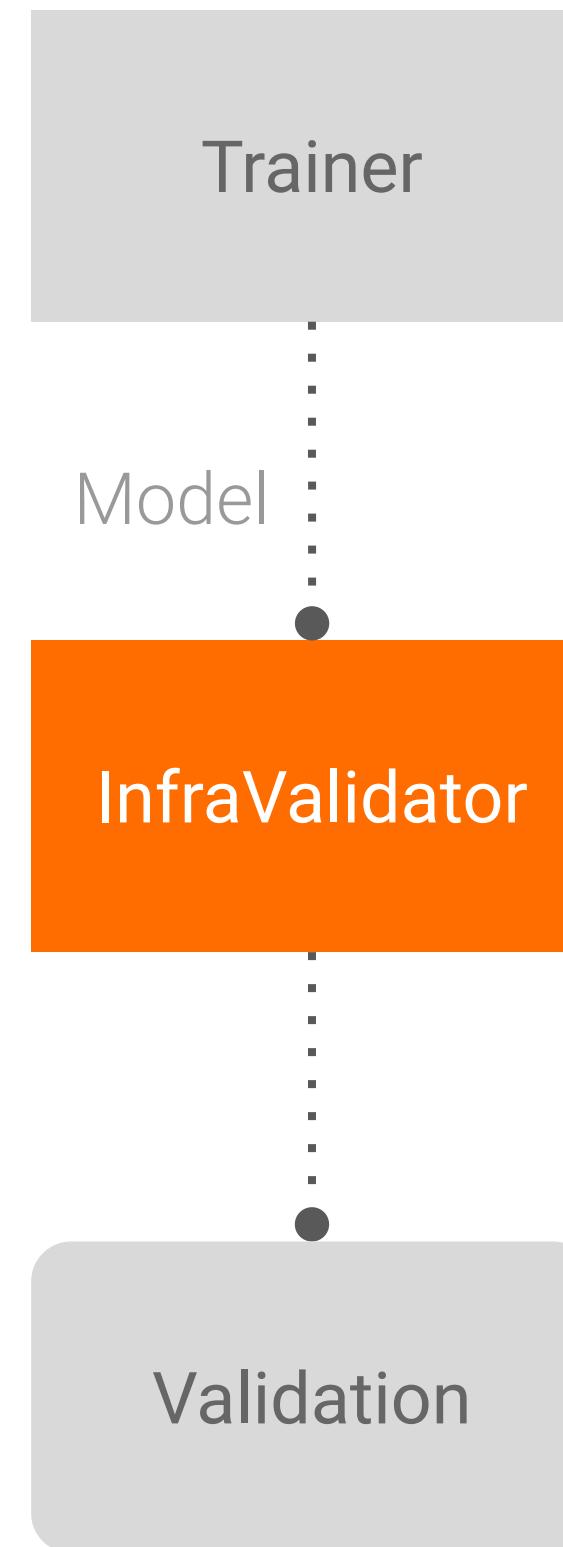


feature	accuracy	accuracy_baseline	auc	auc_precision_recall	average_loss
trip_start_hour:19	0.63582	0.59104	0.64311	0.56092	0.64626
trip_start_hour:14	0.67117	0.65766	0.63793	0.49112	0.61667
trip_start_hour:2	0.66102	0.63559	0.58527	0.47002	0.65236
trip_start_hour:12	0.69643	0.65625	0.68270	0.54122	0.59538
trip_start_hour:0	0.66184	0.66667	0.63773	0.45081	0.61634
trip_start_hour:23	0.65625	0.64844	0.58357	0.43514	0.64315



Component: InfraValidator

Inputs and Outputs



Configuration

```
infra_validator = InfraValidator(  
    model=trainer.outputs['model'],  
    serving_spec=ServingSpec(...),  
    validation_spec=ValidationSpec(...),  
    request_spec=RequestSpec(...)  
)
```

Configuration Options

ServingSpec: Type of model server and infrastructure to test with

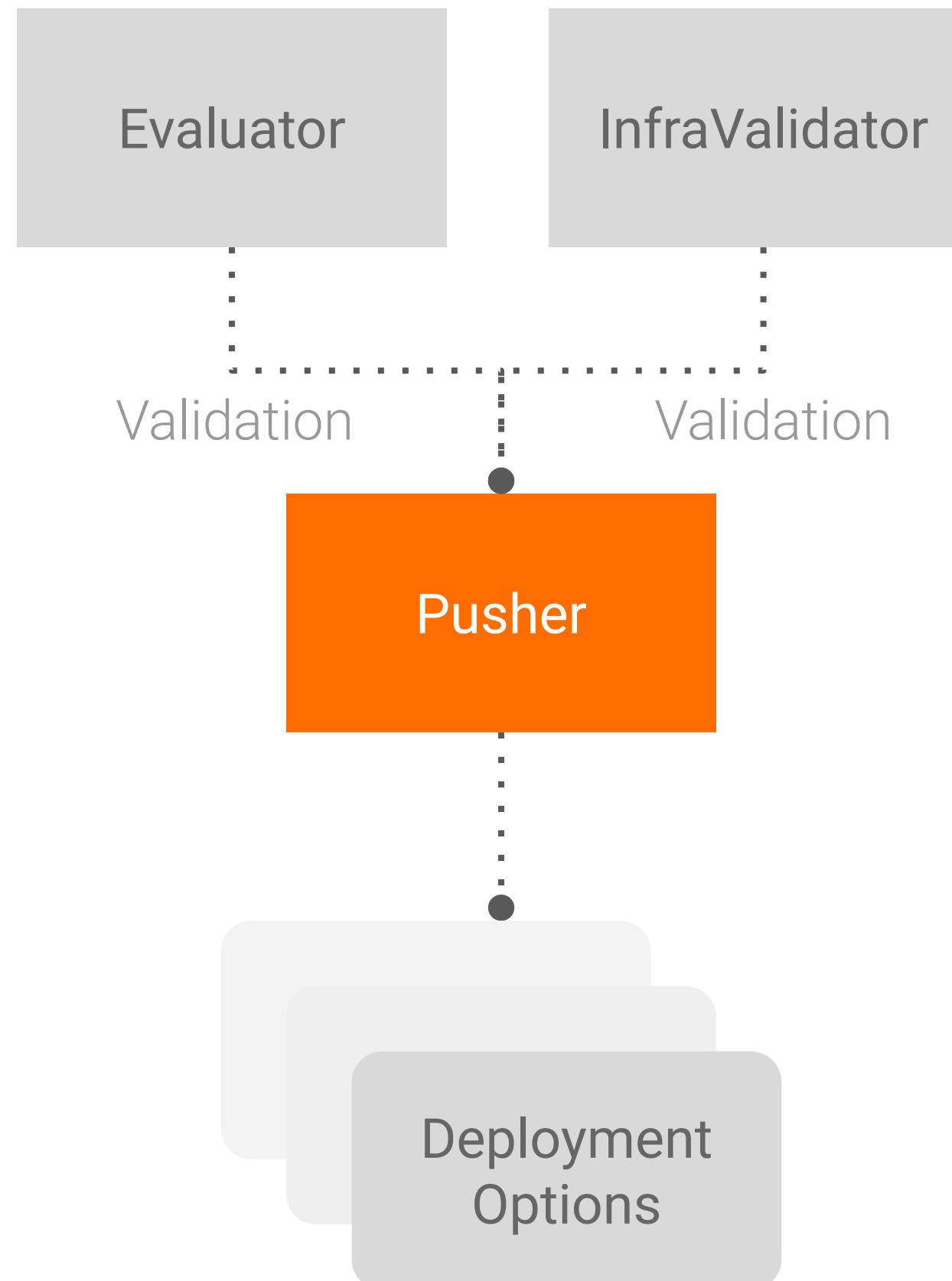
ValidationSpec: Adjusts the infra validation criteria or workflow

RequestSpec: Which model signature, how many examples to test



Component: Pusher

Inputs and Outputs



Configuration

```
pusher = Pusher(  
    model=trainer.outputs['model'],  
    model_blessing=evaluator.outputs['blessing'],  
    infra_blessing=infra_validator.outputs['blessing'],  
    push_destination=pusher_pb2.PushDestination(  
        filesystem=pusher_pb2.PushDestination.Filesystem(  
            base_directory=_serving_model_dir)))
```

Block push on validation outcome

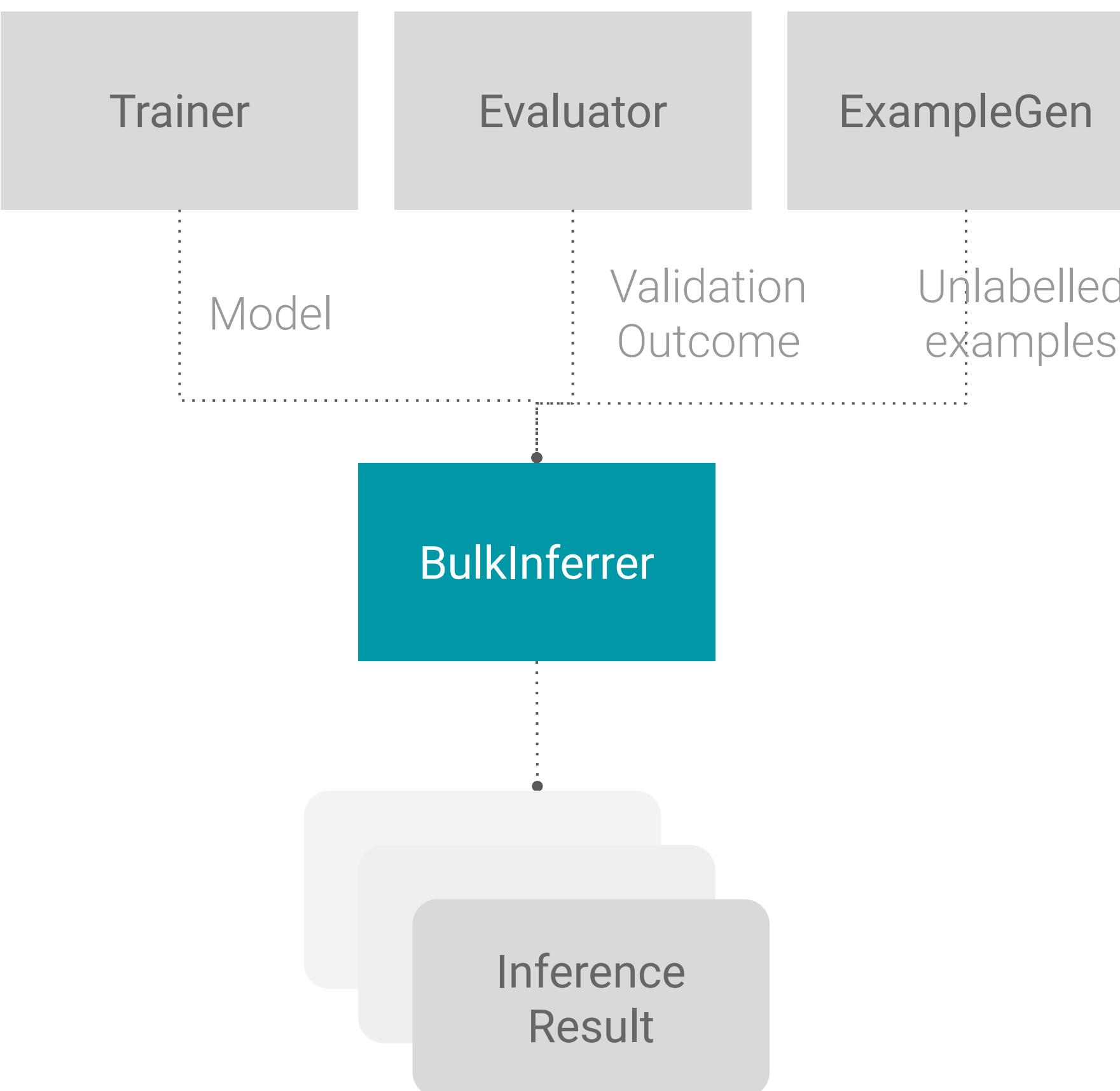
Push destinations supported today

- Filesystem (TensorFlow Lite, TensorFlow JS)
- TensorFlow Hub
- TensorFlow Serving
- Cloud AI Pipelines
- etc



Component: BulkInferencer

Inputs and Outputs



Configuration

```
bulk_inferrer = BulkInferencer(  
    examples=inference_example_gen.outputs['examples'],  
    model_export=trainer.outputs['output'],  
    model_blessing=evaluator.outputs['blessing'],  
    data_spec=bulk_inferrer_pb2.DataSpec(  
        example_splits=['unlabelled']),  
    model_spec=bulk_inferrer_pb2.ModelSpec())
```

Configuration Options

- Block batch inference on a successful model validation.
- Choose the inference examples from example gen's output.
- Choose the signatures and tags of inference model.

Inference Result

Contains features and predictions.

TFX Pipeline Nodes



What are Pipeline Nodes?

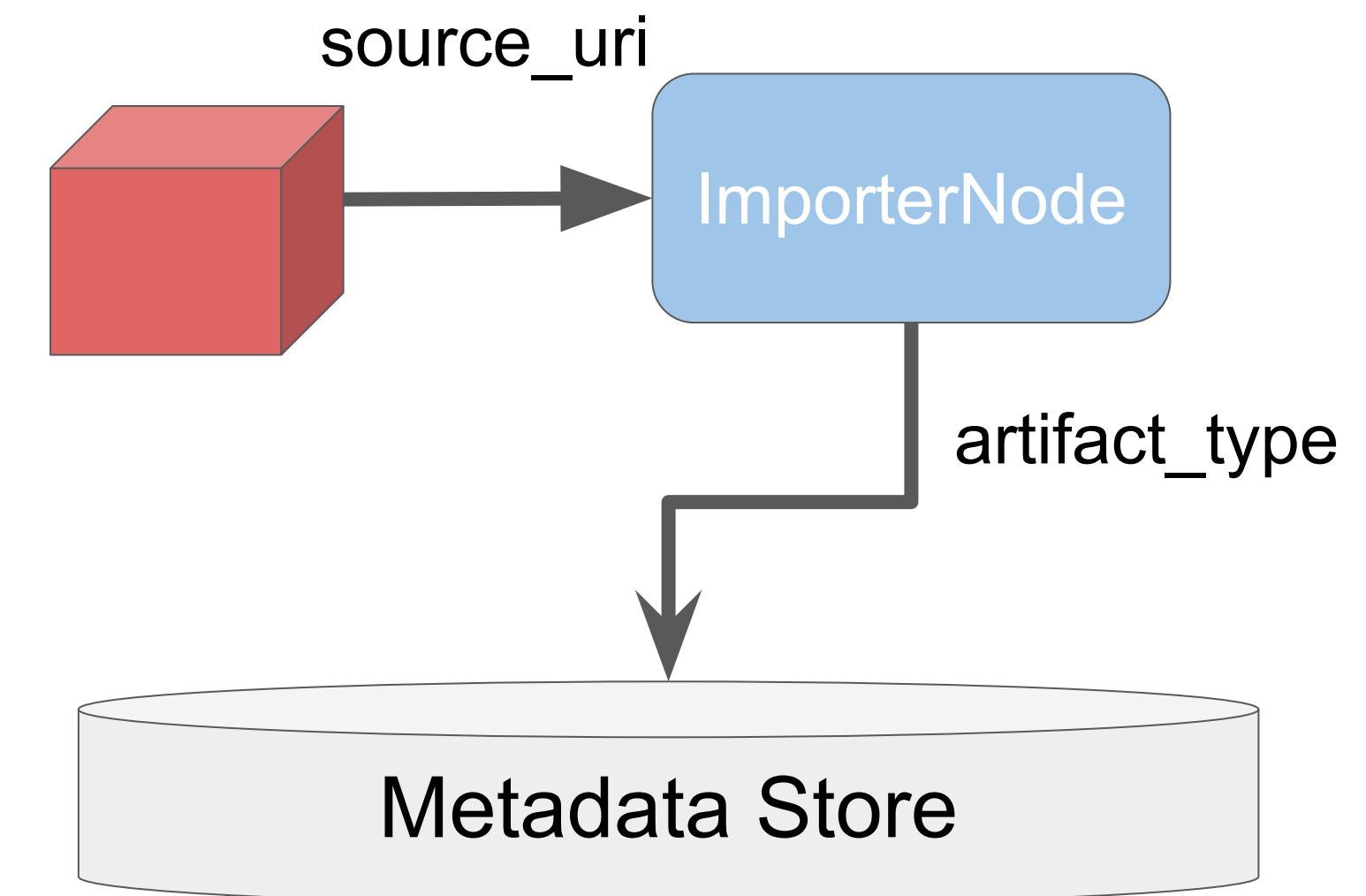
Pipeline nodes are special-purpose classes for performing advanced metadata operations

- Import external artifacts into ML-Metadata
- Perform queries of current ML Metadata based on artifact properties and history



ImporterNode

ImporterNode imports an external data object into the metadata store



- Key requirements: source_uri, artifact_type

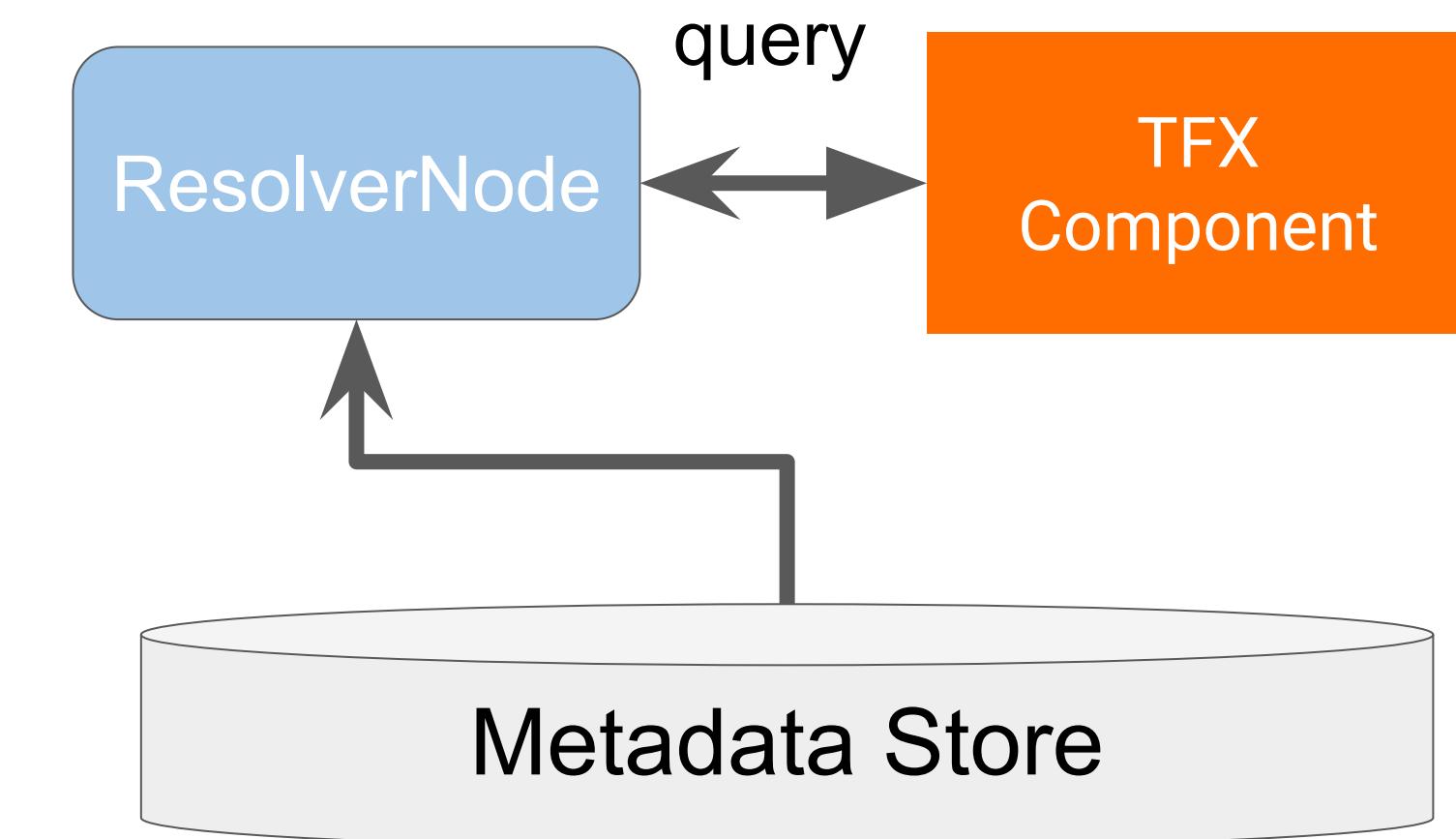
```
importer = ImporterNode(  
    instance_name='import_schema',  
    source_uri='uri/to/schema'  
    artifact_type=standard_artifacts.Schema,  
    reimport=False)
```



ResolverNode

ResolverNode is used to perform metadata queries

- Key requirements: Class, Query config, Channel



```
latest_five_examples_resolver = ResolverNode(  
    instance_name='latest_five_examples_resolver',  
    resolver_class=latest_artifacts_resolver.LatestArtifactsResolver, # Class  
    resolver_config={'desired_num_of_artifacts' : 5}, # Query config  
    examples=example_gen.outputs['examples']) # Channel
```



Current ResolverNodes

- **LatestArtifactsResolver**
 - Resolver that returns the latest N artifacts in a given channel
- **LatestBlessedModelResolver**
 - Returns the latest blessed model

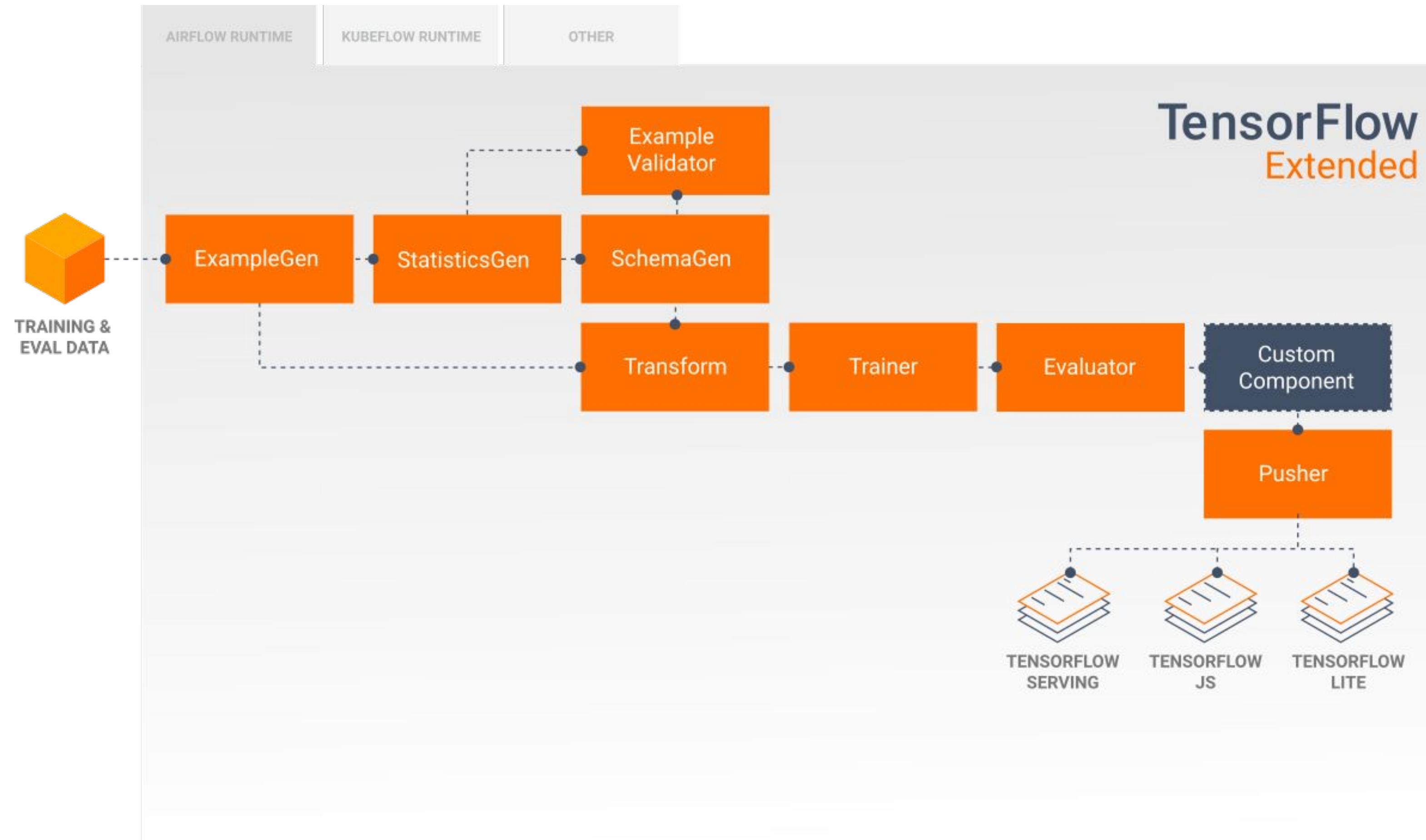
TFX Custom Components



Types of Custom Components

3 ways to create a custom component

- Python function with decorator and annotations
- Components using containers
- Extending existing component classes



Build your own component

Create your own **components** to run within a TFX pipeline while still providing the benefits of metadata management, lineage, and pipeline monitoring.



Python function-based component

- Decorate a Python function
- Use Python annotations to declare inputs and outputs

```
@component
def MyComponent(
    model: InputArtifact[Model],
    blessing: OutputArtifact[Model],
    accuracy_threshold: Parameter[int] = 10,
) -> OutputDict(accuracy=float):

    # Component code here

    return {
        'something': something
    }
```



Container-based component

- Configure inputs, outputs, parameters, and the image
- Wrap with `container_component.create_container_component`

```
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='...',
    inputs={ 'text': standard_artifacts.ExternalArtifact },
    outputs={ 'filtered_text': standard_artifacts.ExternalArtifact },
    parameters={ ... },

    image='google/cloud-sdk:278.0.0', # Image to run
    command=[ # Command to run inside container
        'sh', '-exc',
        ...
        # Body of command
        '',
    ],
)
```

TFX-Addons

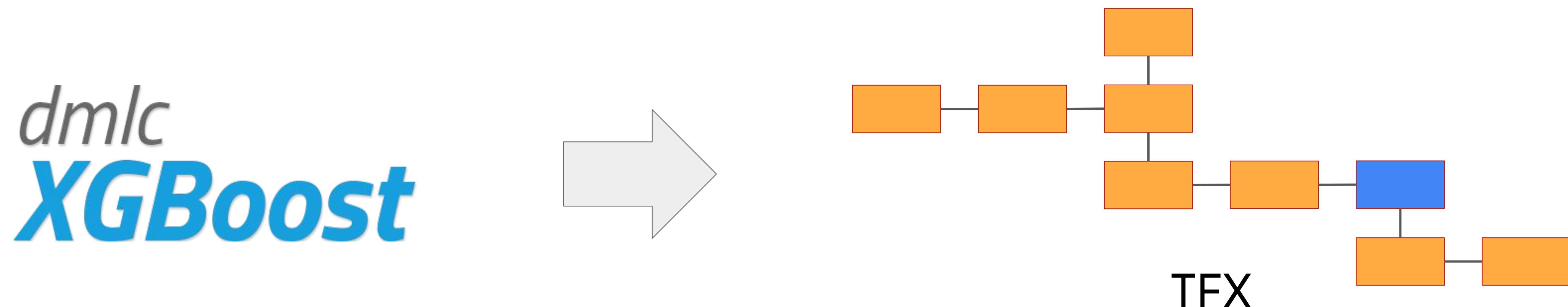
Community contributed components, tools,
libraries, examples, and visualizations

Join at <http://goo.gle/tfx-addons-group>

XGBoost Evaluator

SIG TFX-Addons Community Project

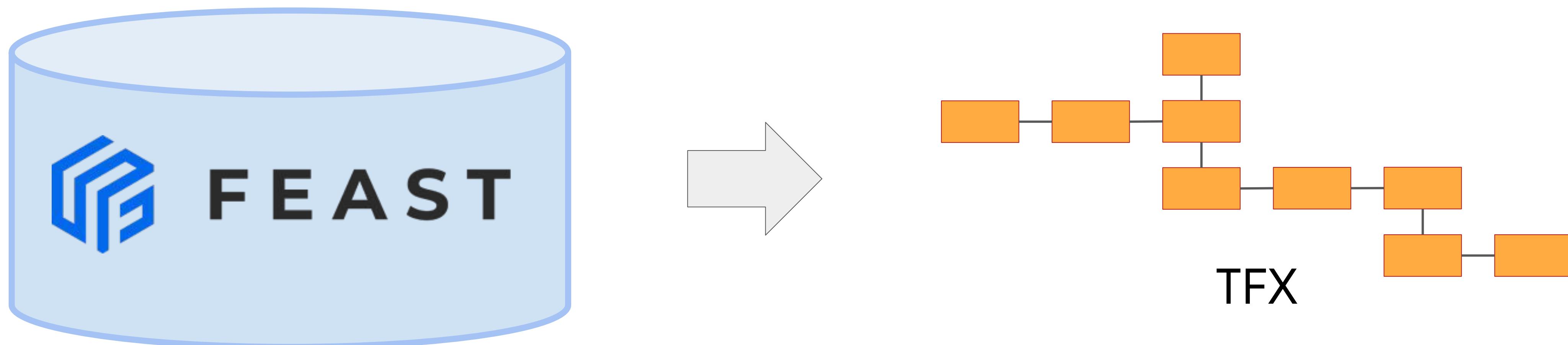
- Adds support for evaluating XGBoost models in TFX pipelines



Feast Feature Store

SIG TFX-Addons Community Project

- TFX-Addon component allows TFX pipelines to ingest Feast datasets



Coursera 4-Course Specialization

Production ML Engineering specialization
with Andrew Ng



TensorFlow Extended (TFX)

Standard components for your production model needs

Flexible orchestration and metadata

Extensible with custom components



Install

Learn ▾

API ▾

Resources ▾

Community

Why TensorFlow ▾

Search

GitHub

For Production

[Overview](#) [Tutorials](#) [Guide](#) [API](#)

Apply to speak at TensorFlow World. Deadline April 23rd.

[Propose talk](#)

**TensorFlow Extended (TFX) is
an end-to-end platform for
deploying production ML
pipelines**

When you're ready to move your models from research to production, use TFX to create and manage a production



<https://www.tensorflow.org/tfx>

Thank you!



More Resources

Web	https://tensorflow.org/tfx
Repo	https://github.com/tensorflow/tfx
YouTube	https://goo.gle/tfx-youtube
SIG	https://github.com/tensorflow/tfx-addons
Community	https://discuss.tensorflow.org/tag/tfx