TensorFlow

TFX Tutorial

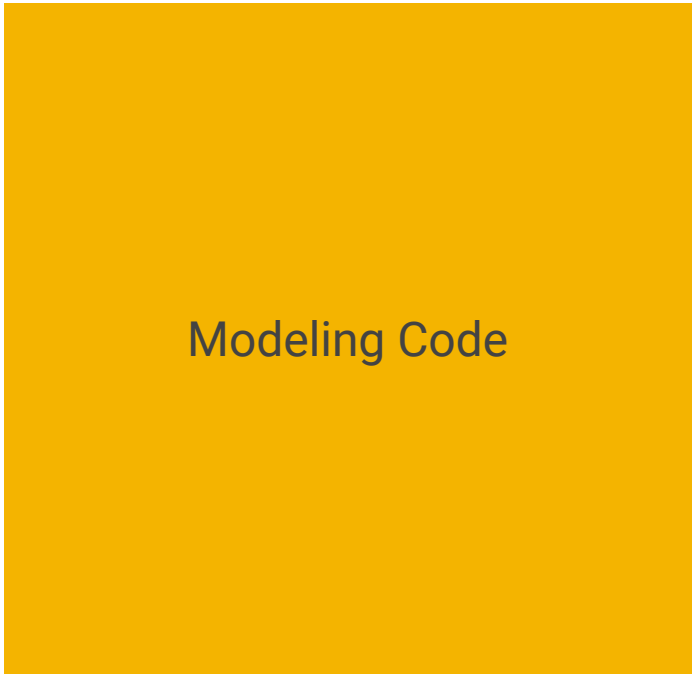
Developing Production ML Pipelines

Aurélien Géron
Consultant
@aureliengeron

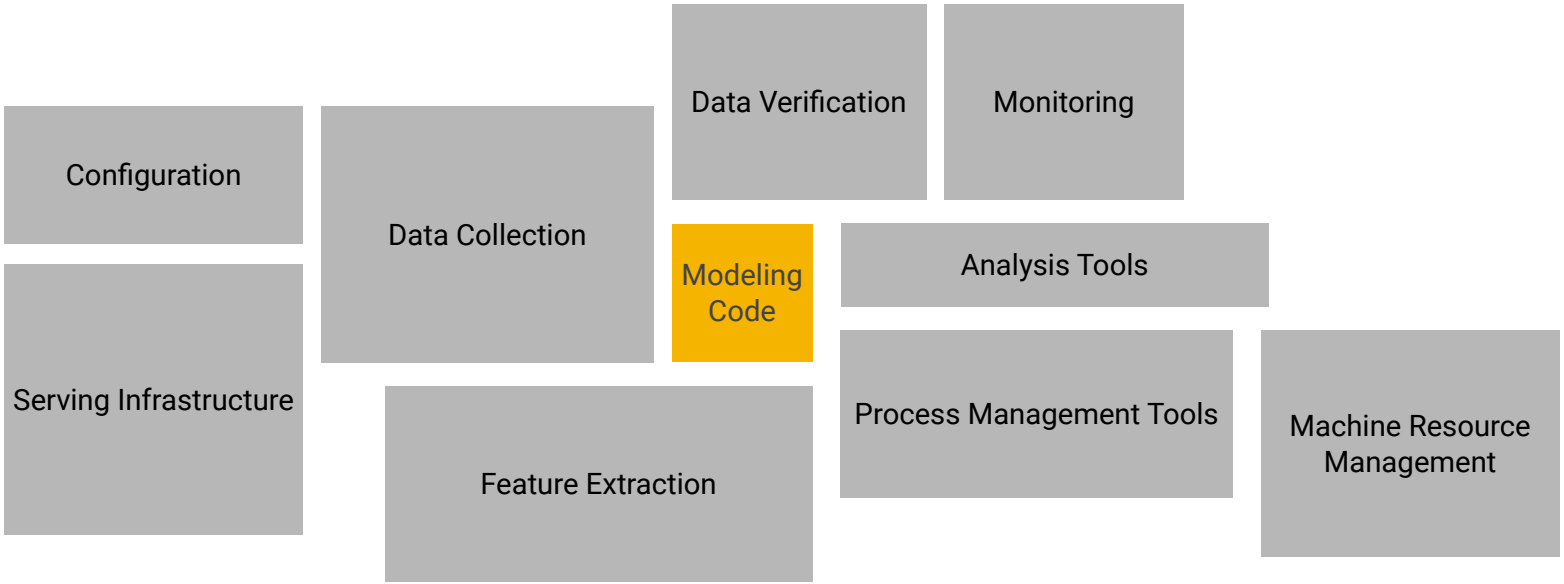
What are we doing here?

What does it all mean?

In addition to training an amazing model ...



… a production solution requires so much more



Production Machine Learning

“Hidden Technical Debt in Machine Learning Systems”

NIPS 2015

http://bit.ly/ml-techdebt

Production Machine Learning

Machine Learning Development

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

Production Machine Learning

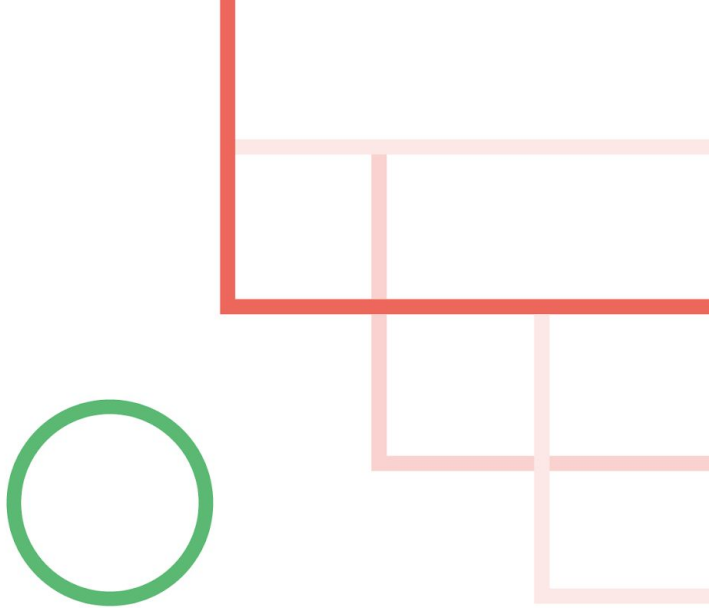
Machine Learning Development

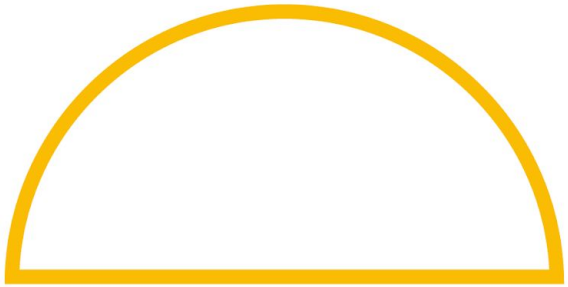
- Labeled data
- Feature space coverage
- Minimal dimensionality
- Maximum predictive data
- Fairness
- Rare conditions
- Data lifecycle management

Modern Software Development

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

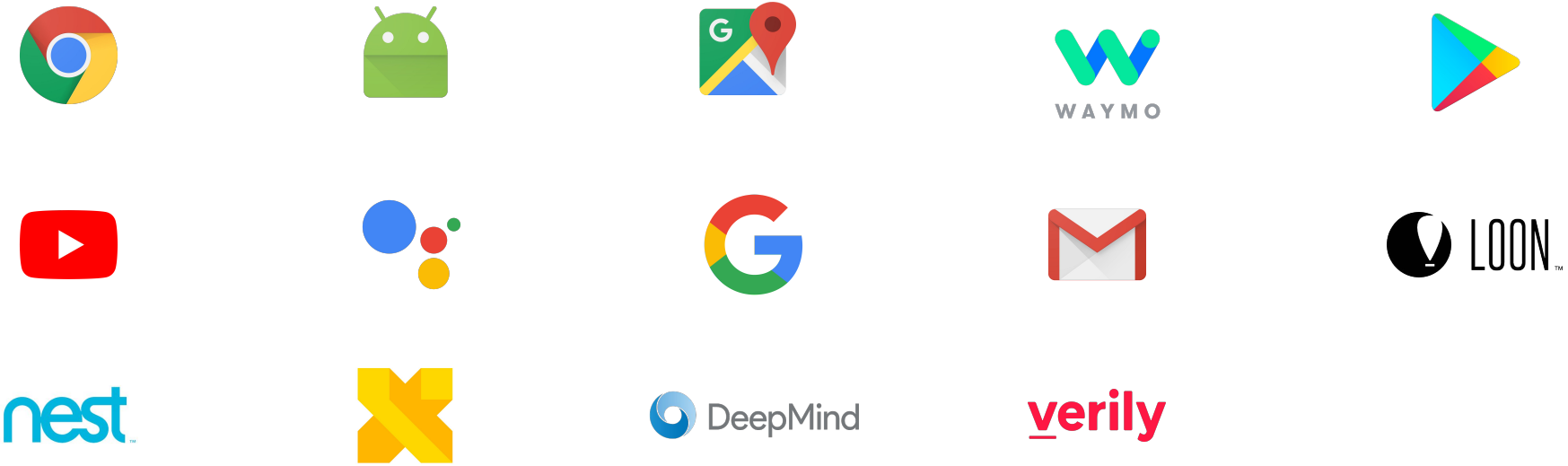
TensorFlow Extended (TFX)



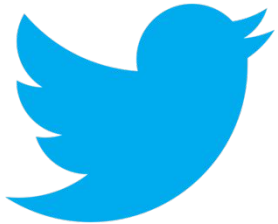


TensorFlow Extended (TFX)

Powers Alphabet's most important bets and products



... and some of Google's most important partners.



"... we have re-tooled our machine learning platform to use TensorFlow. This yielded significant productivity gains while positioning ourselves to take advantage of the latest industry research."

Ranking Tweets with TensorFlow - Twitter blog post

When to use TFX

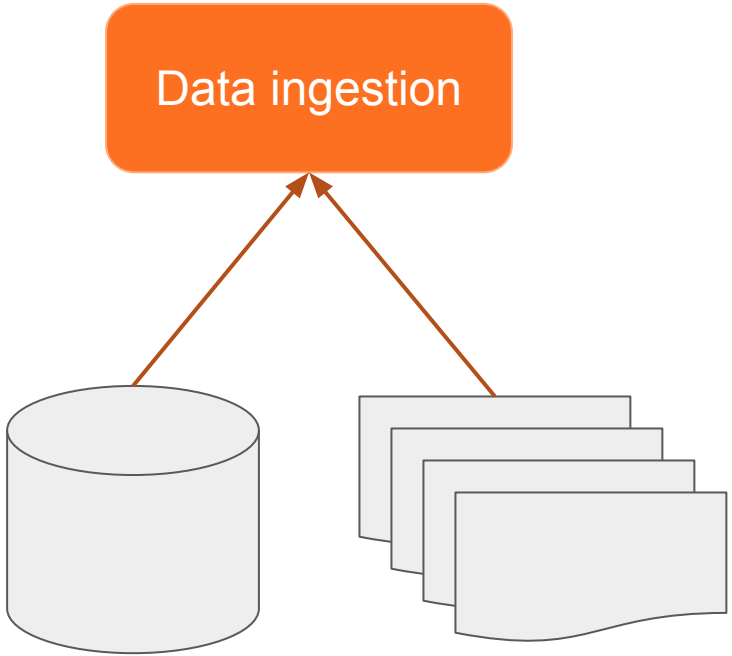
We're learning how to create an ML pipeline using TFX

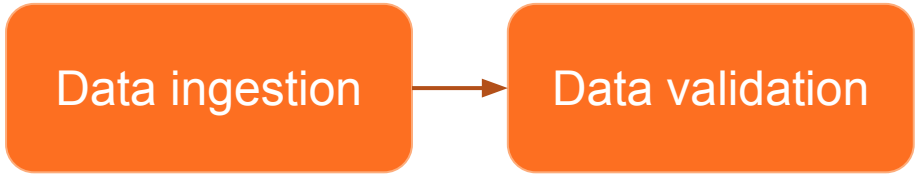
- TFX pipelines are appropriate when:
  - datasets are large, or may someday get large
  - training/serving consistency is important
  - version management for inference is important
- Google uses TFX pipelines for everything from single-node to large-scale ML training and inference

What we're doing

We're following a typical ML development process

- Understanding our data
- Feature engineering
- Training
- Analyze model performance
- Lather, rinse, repeat
- Ready to deploy to production


Data ingestion

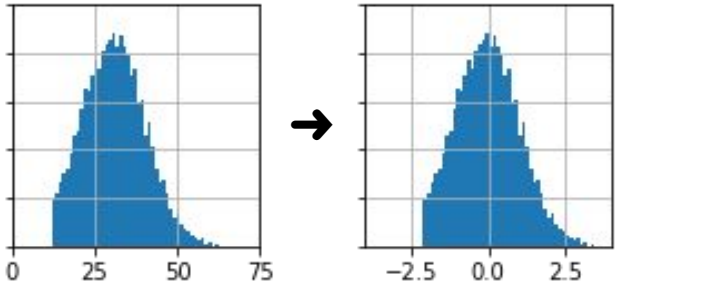


`age` is missing

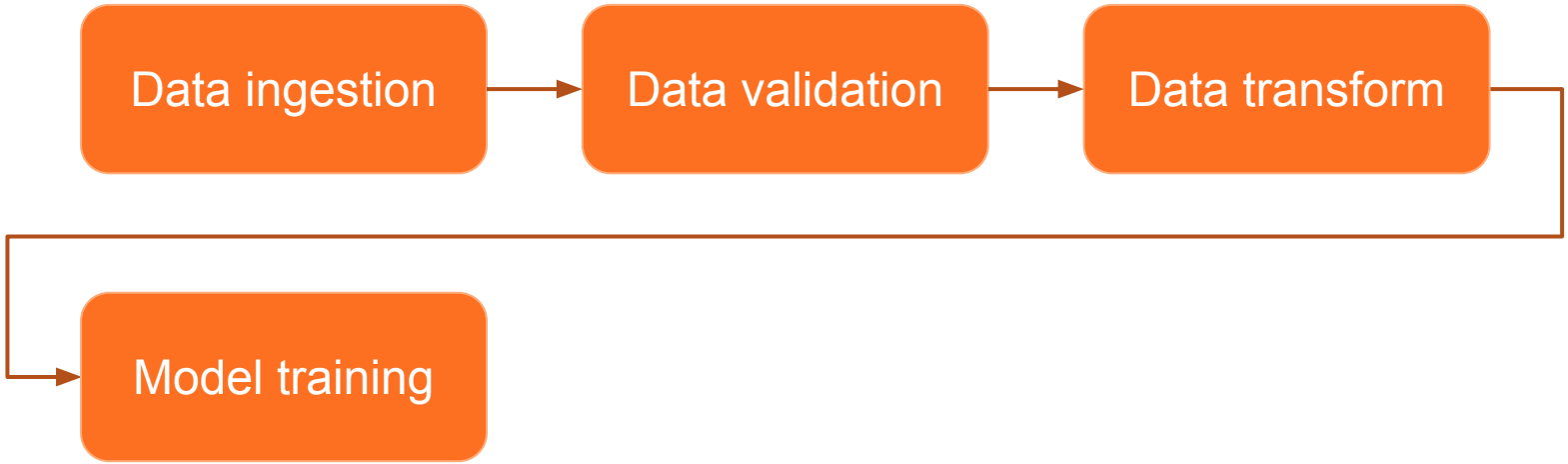
`country` not in:

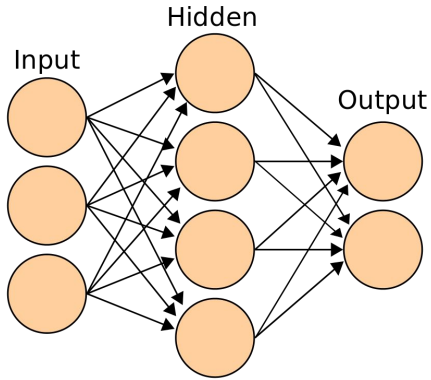
- China
- India
- USA

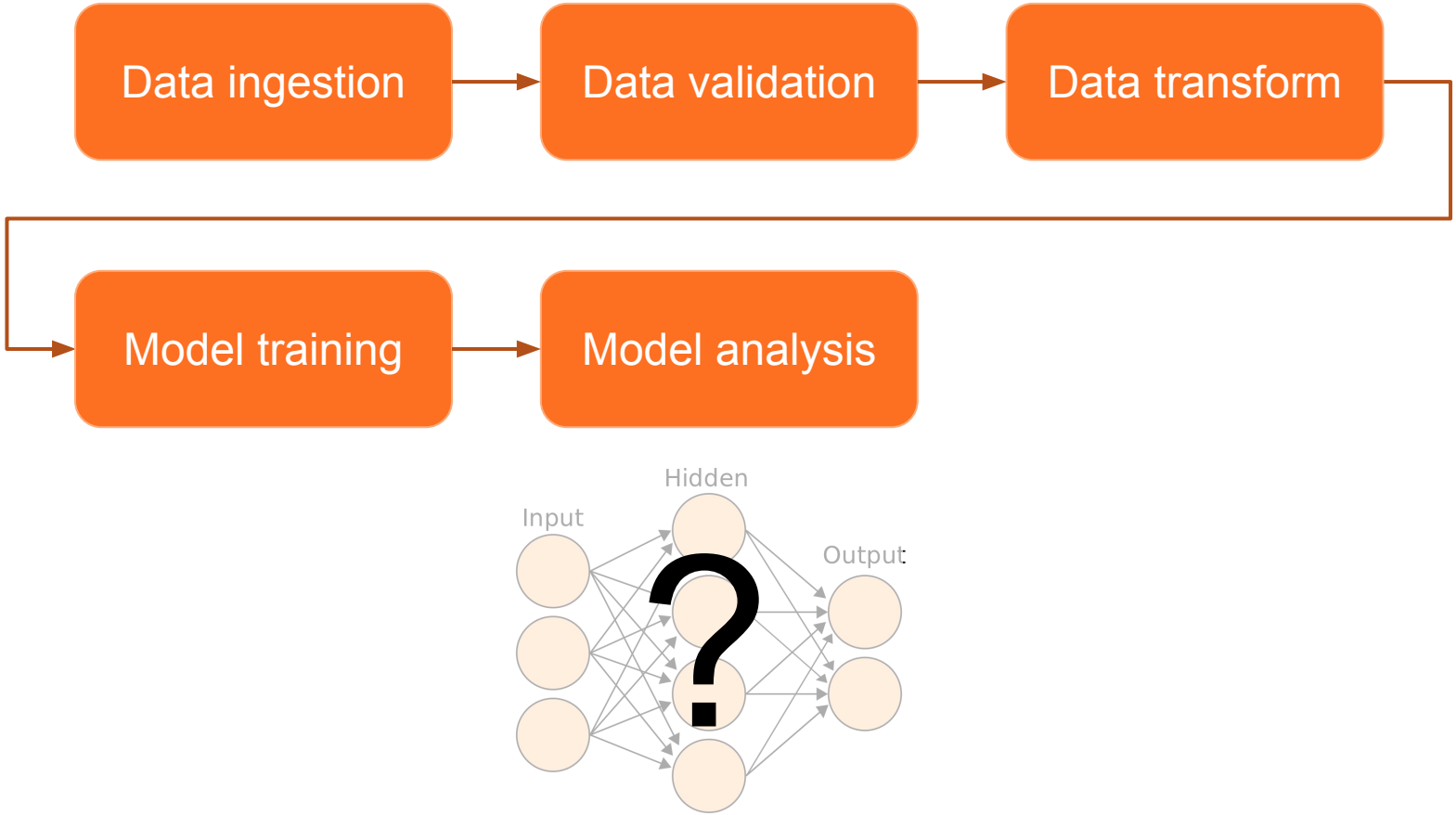
Data ingestion → Data validation → Data transform

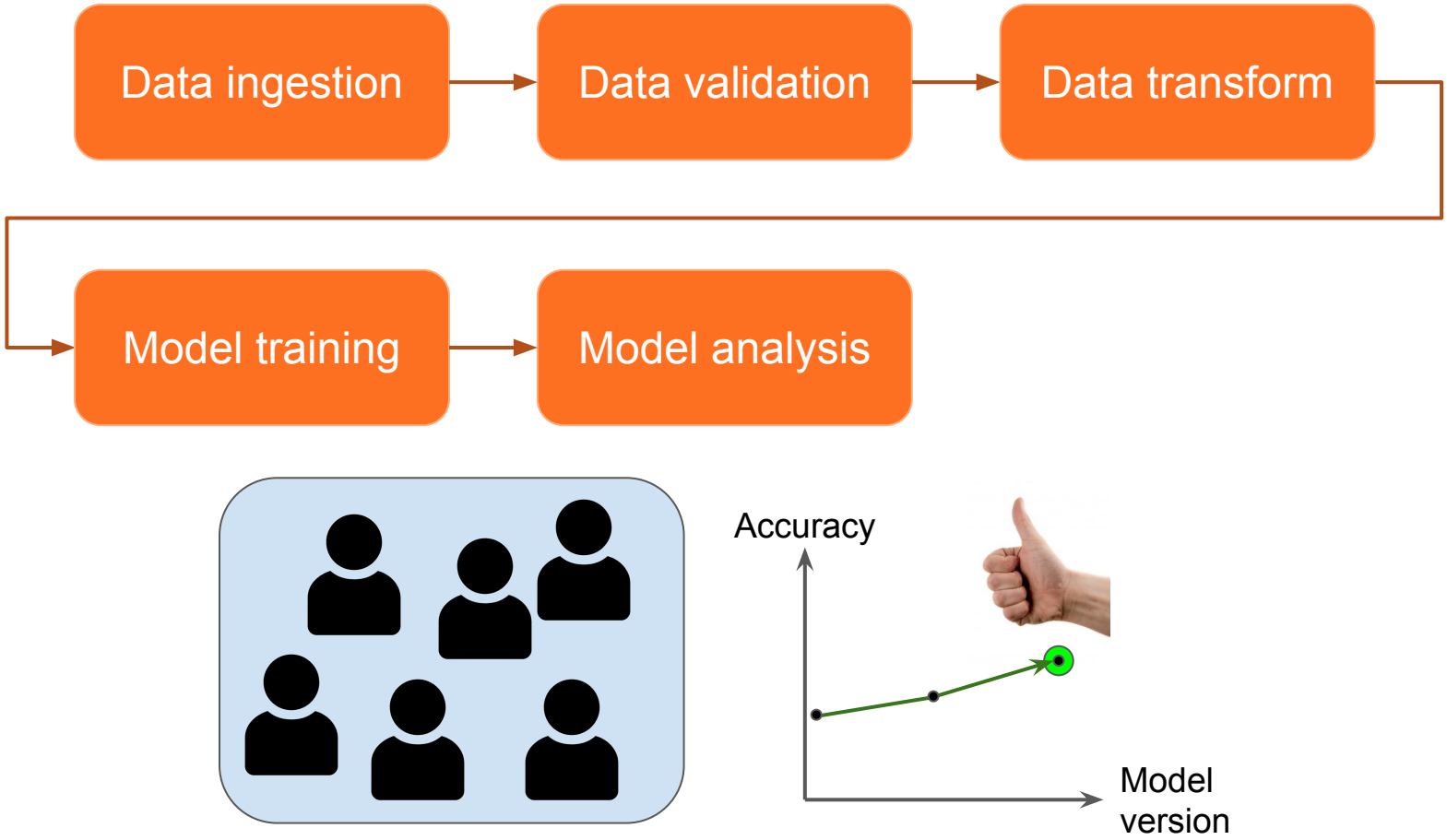


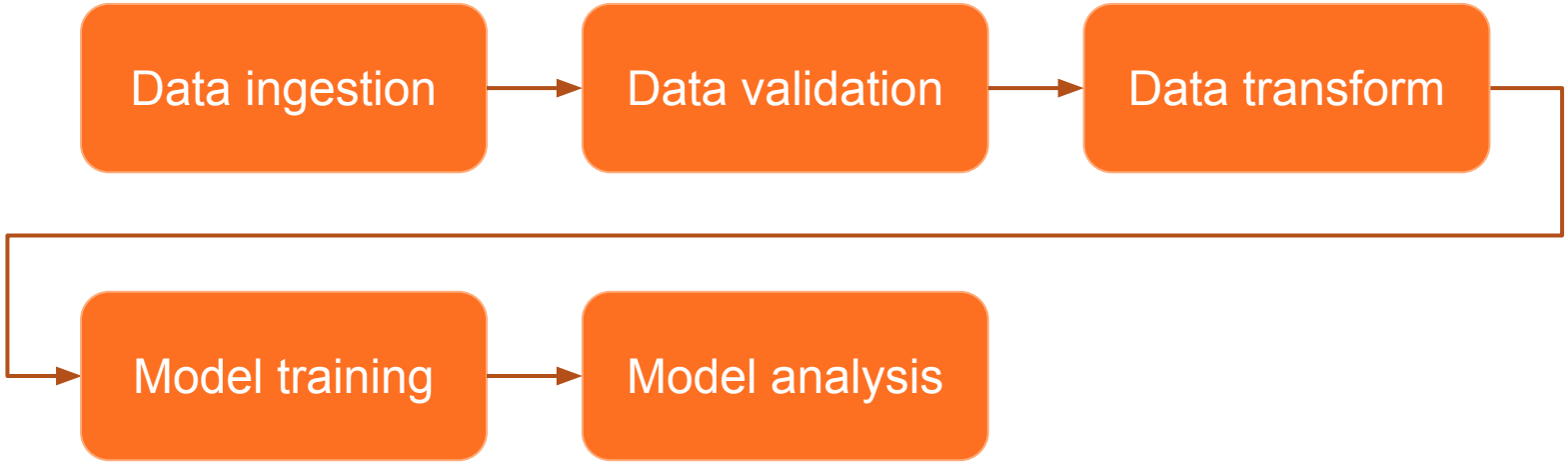
China ➜ [1, 0, 0]
India ➜ [0, 1, 0]
USA ➜ [0, 0, 1]

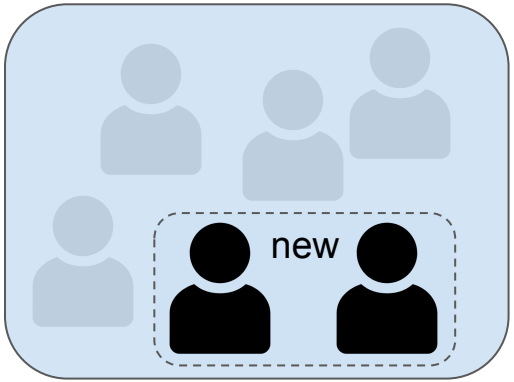

Data ingestion
Data validation
Data transform
Model training

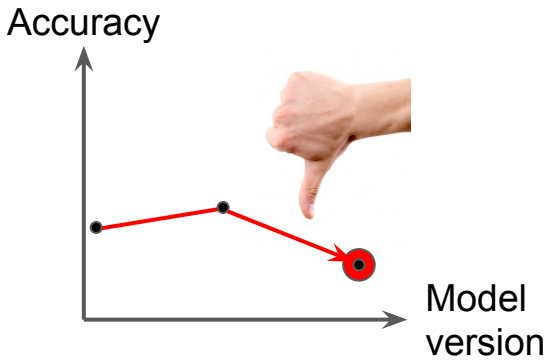

Input
Hidden
Output

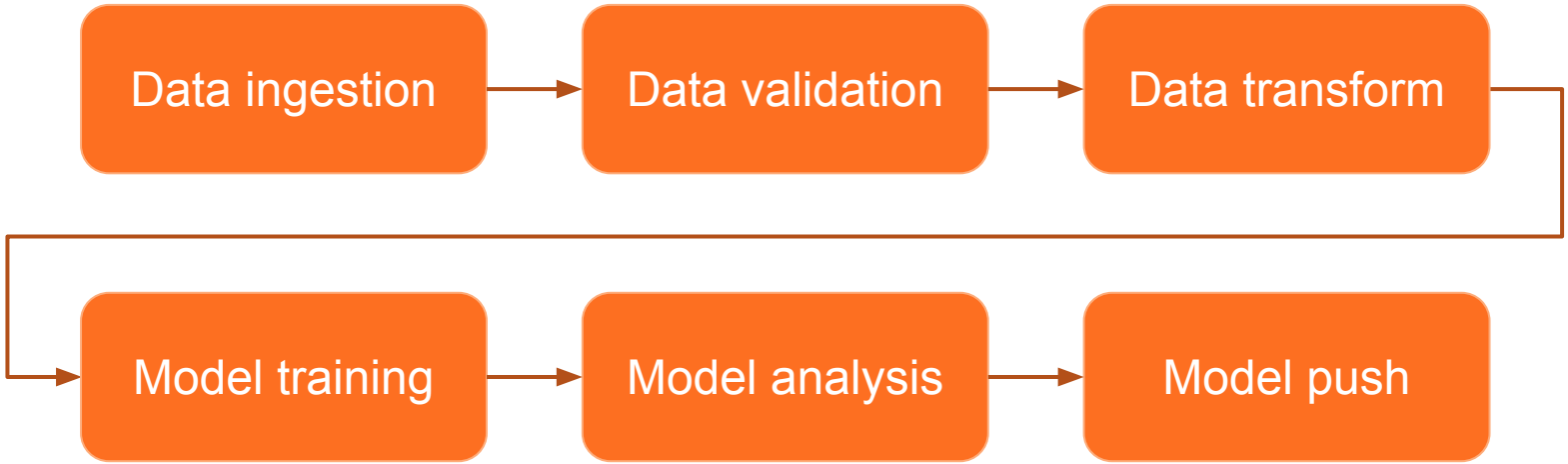

Data ingestion
Data validation
Data transform
Model training
Model analysis
Hidden
Input
Output
?

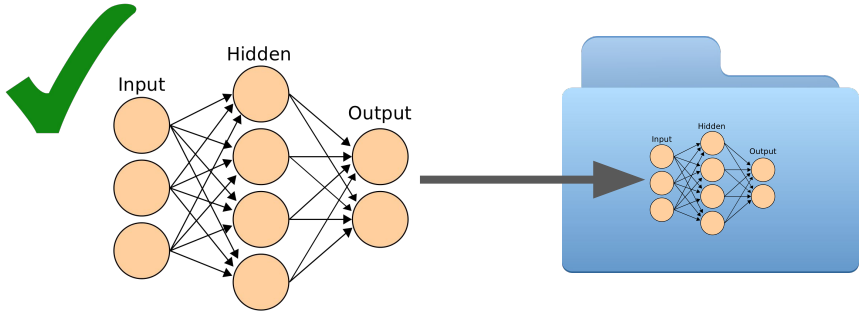

Data ingestion
Data validation
Data transform
Model training
Model analysis
Accuracy
Model version

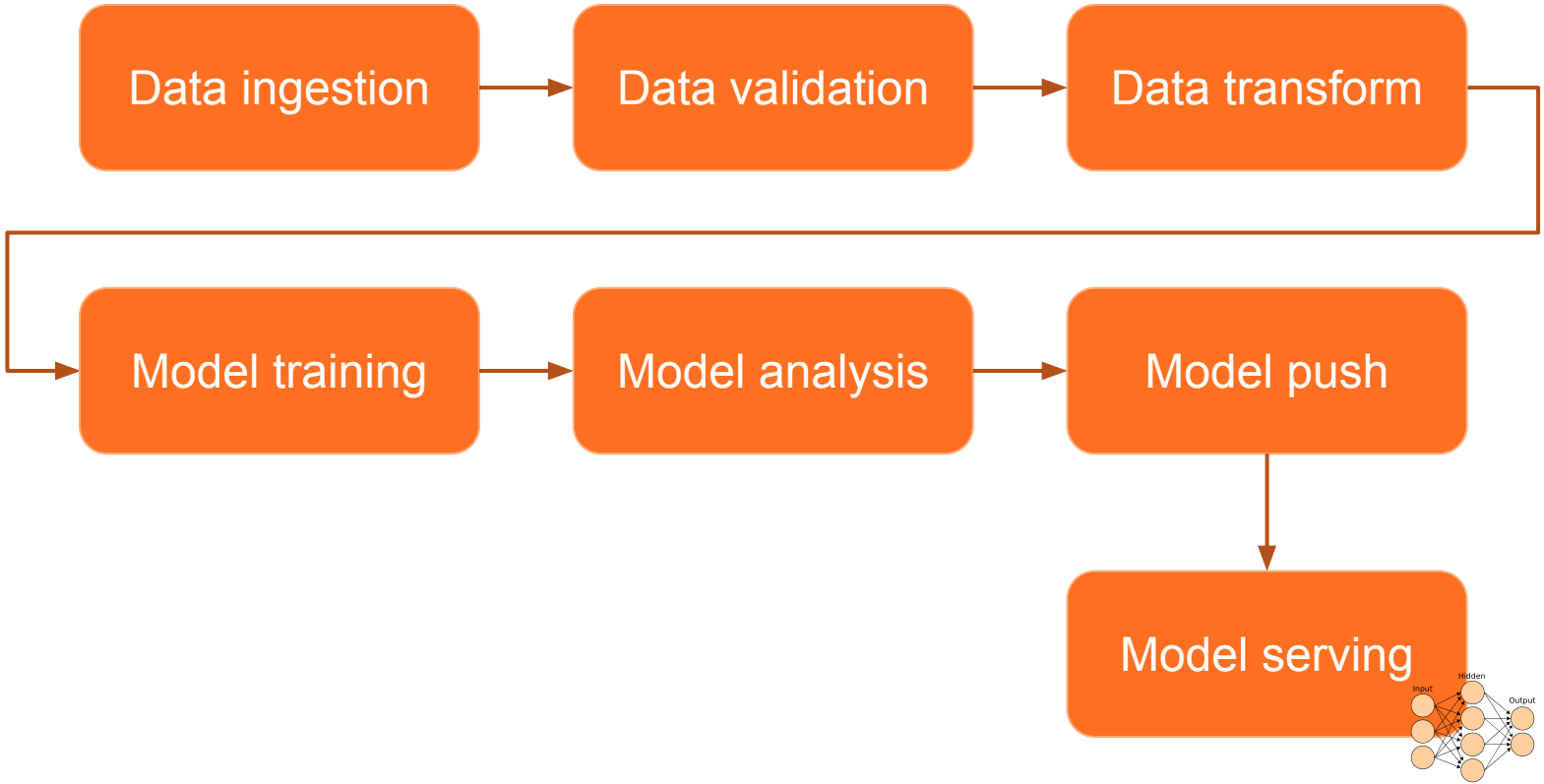

Data ingestion
Data validation
Data transform
Model training
Model analysis

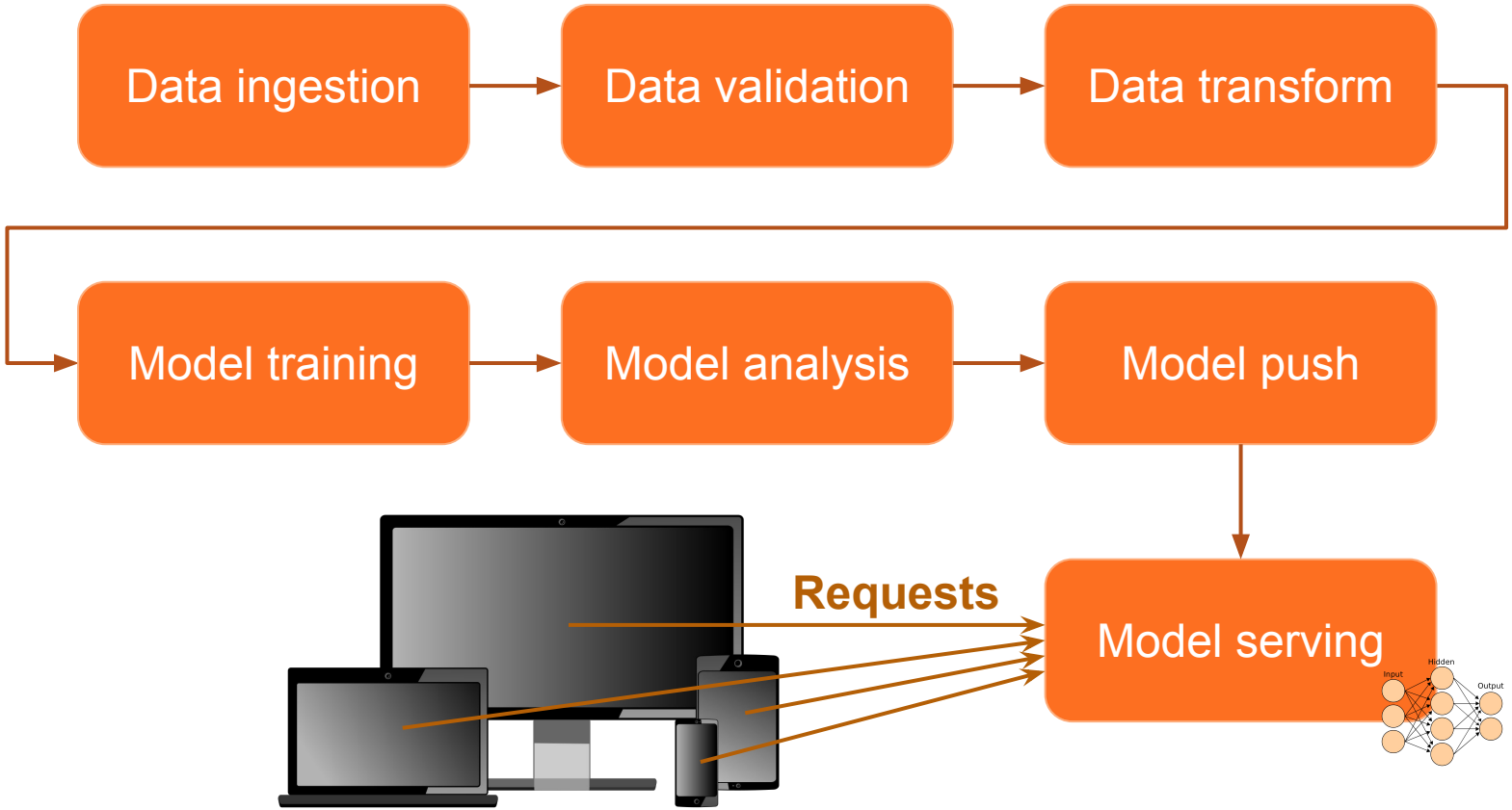

new

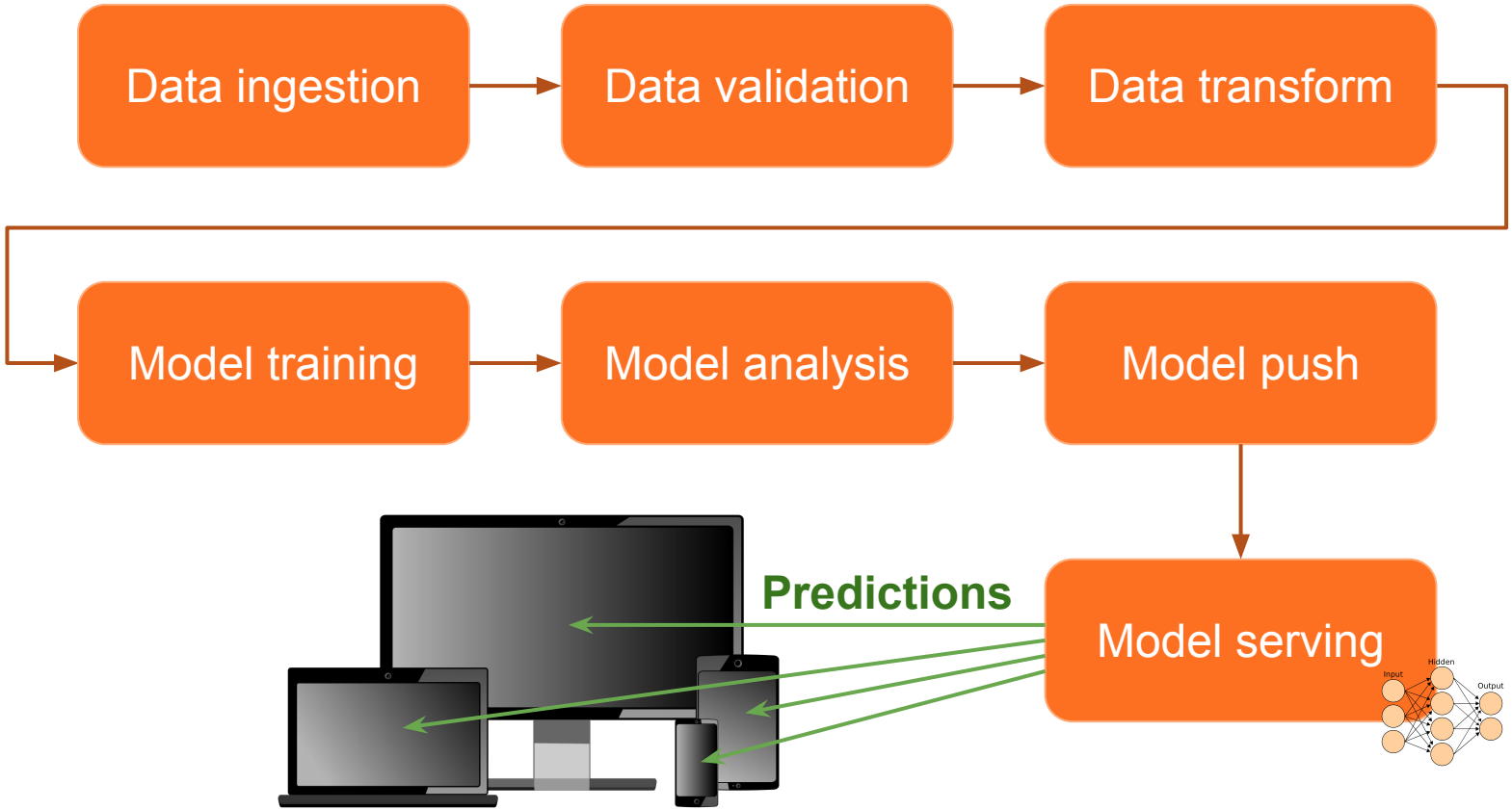

Accuracy
Model version

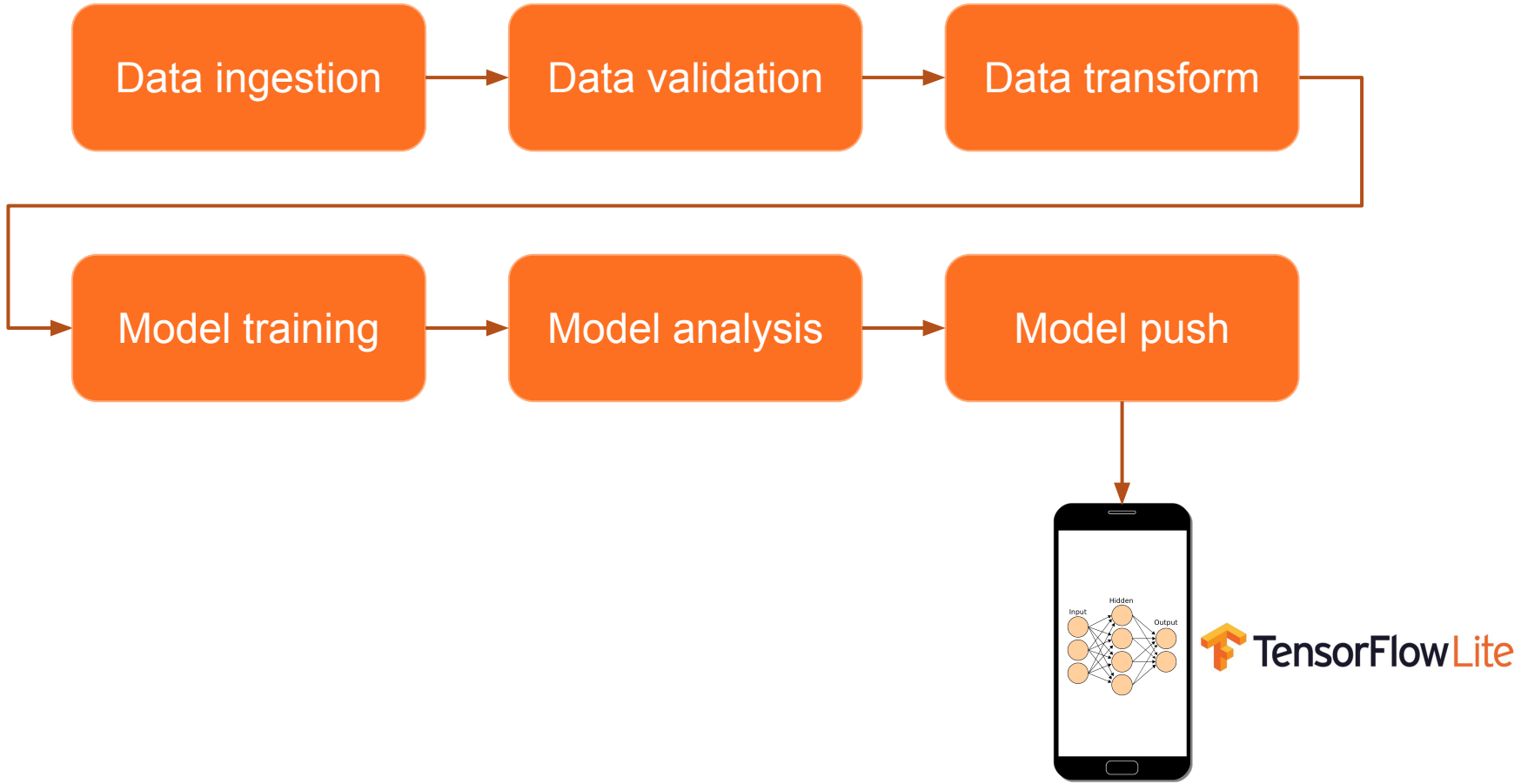

Data ingestion
Data validation
Data transform
Model training
Model analysis
Model push


Input
Hidden
Output
Input
Hidden
Output


Data ingestion
Data validation
Data transform
Model training
Model analysis
Model push
Model serving
Input
Hidden
Output


Data ingestion
Data validation
Data transform
Model training
Model analysis
Model push
Requests
Model serving
Input
Hidden
Output


Data ingestion
Data validation
Data transform
Model training
Model analysis
Model push
Predictions
Model serving
Input
Hidden
Output


Data ingestion
Data validation
Data transform
Model training
Model analysis
Model push
Input
Hidden
Output
TensorFlow Lite

Data ingestion

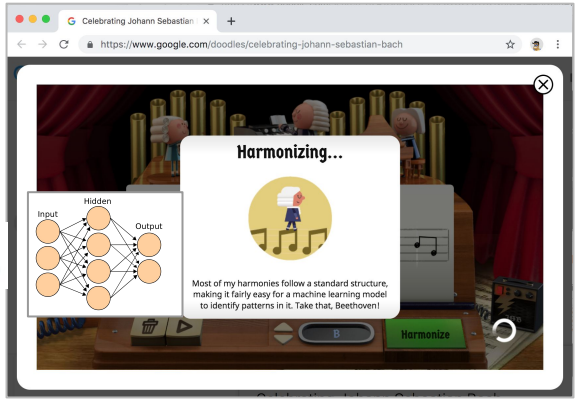
Data validation

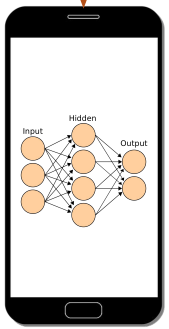
Data transform

Model training

Model analysis

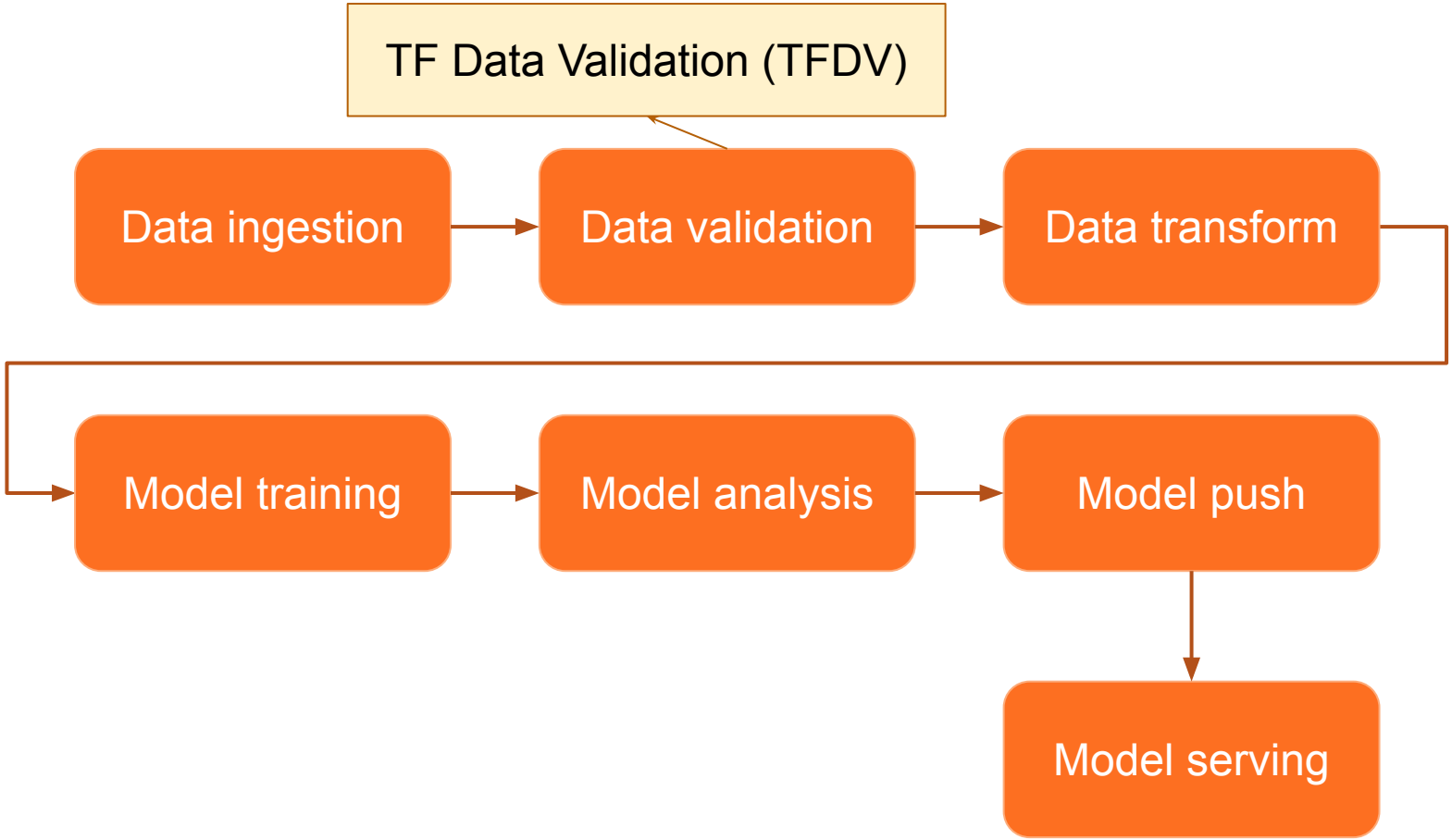
Model push

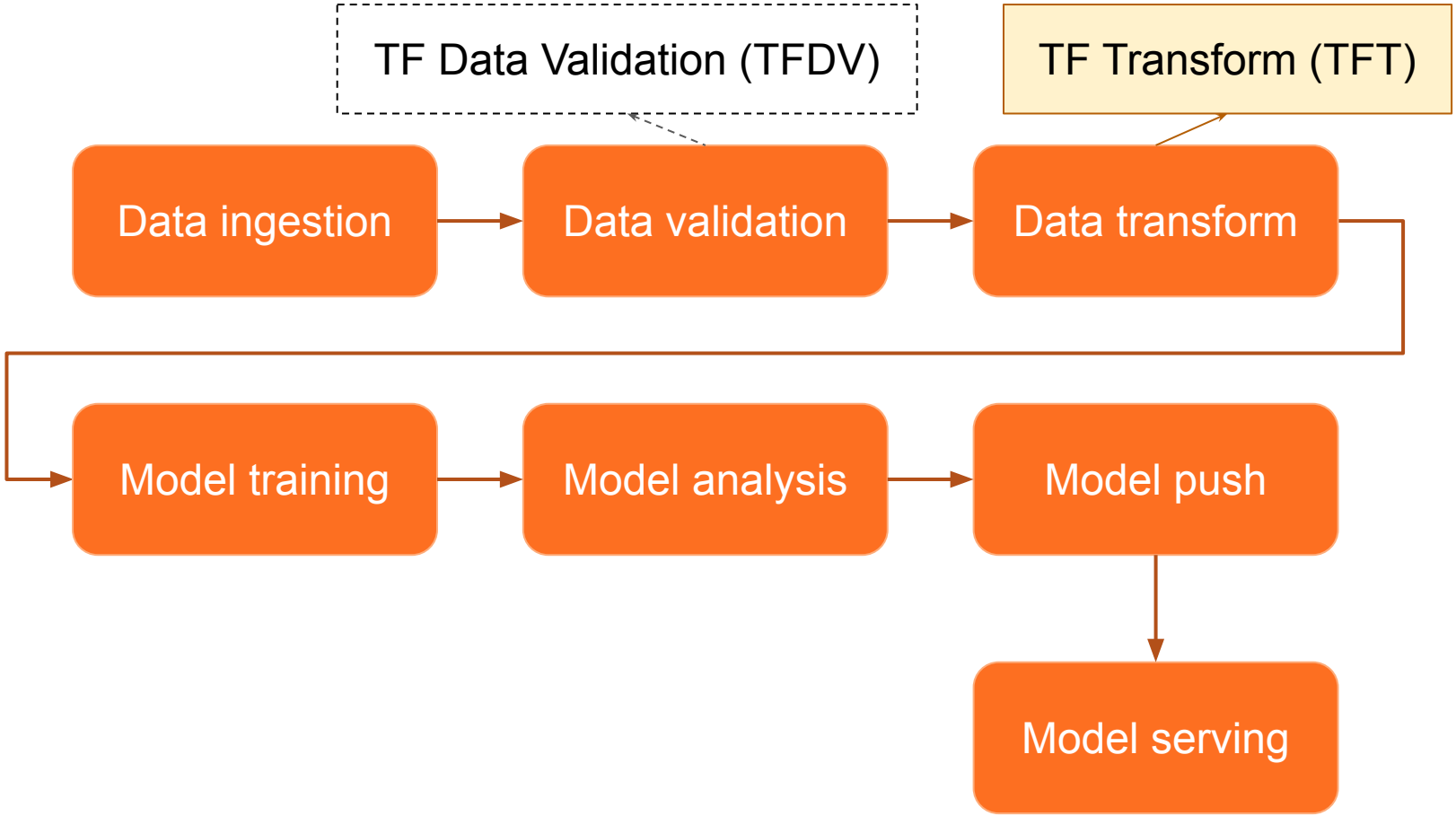


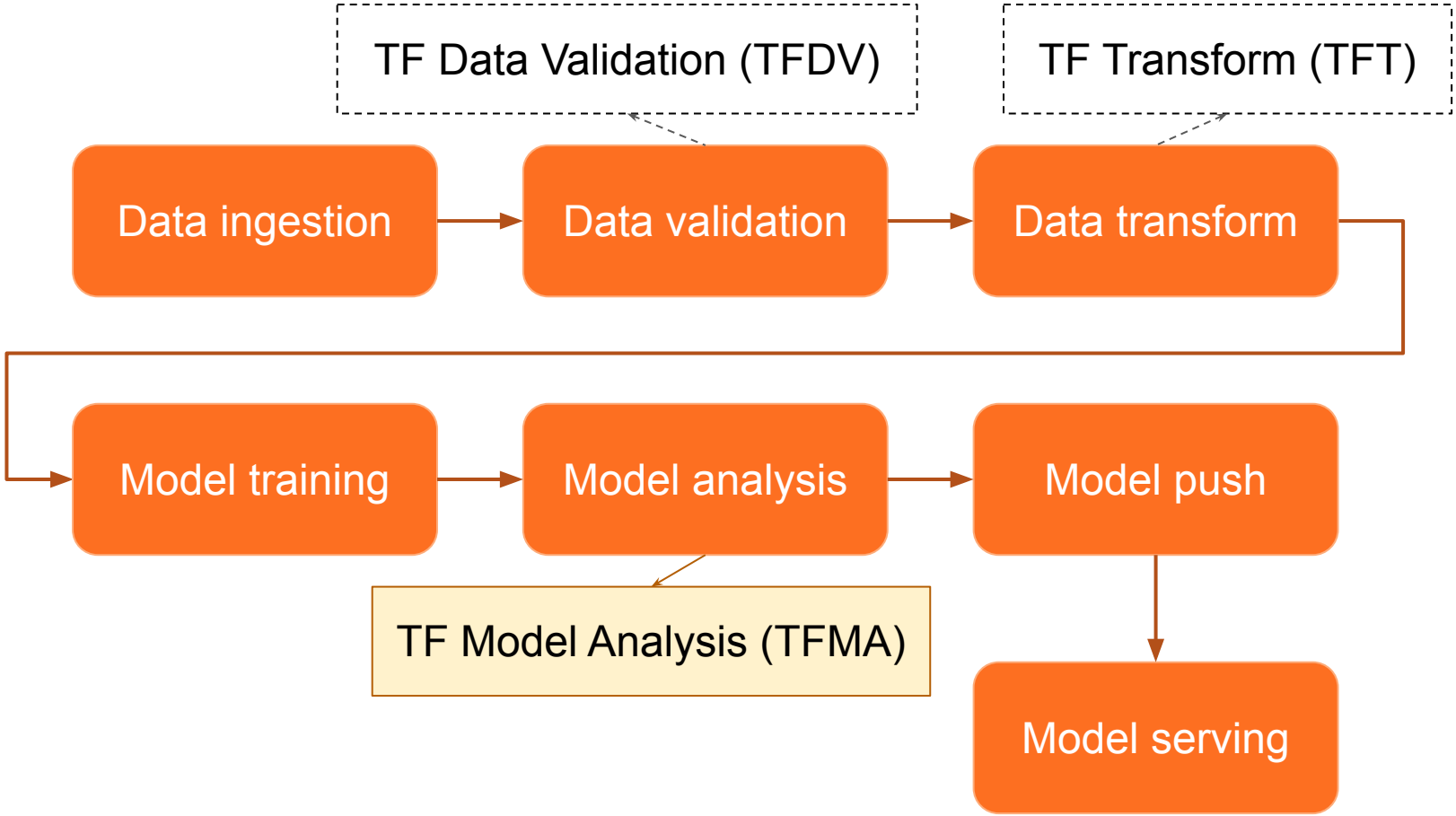


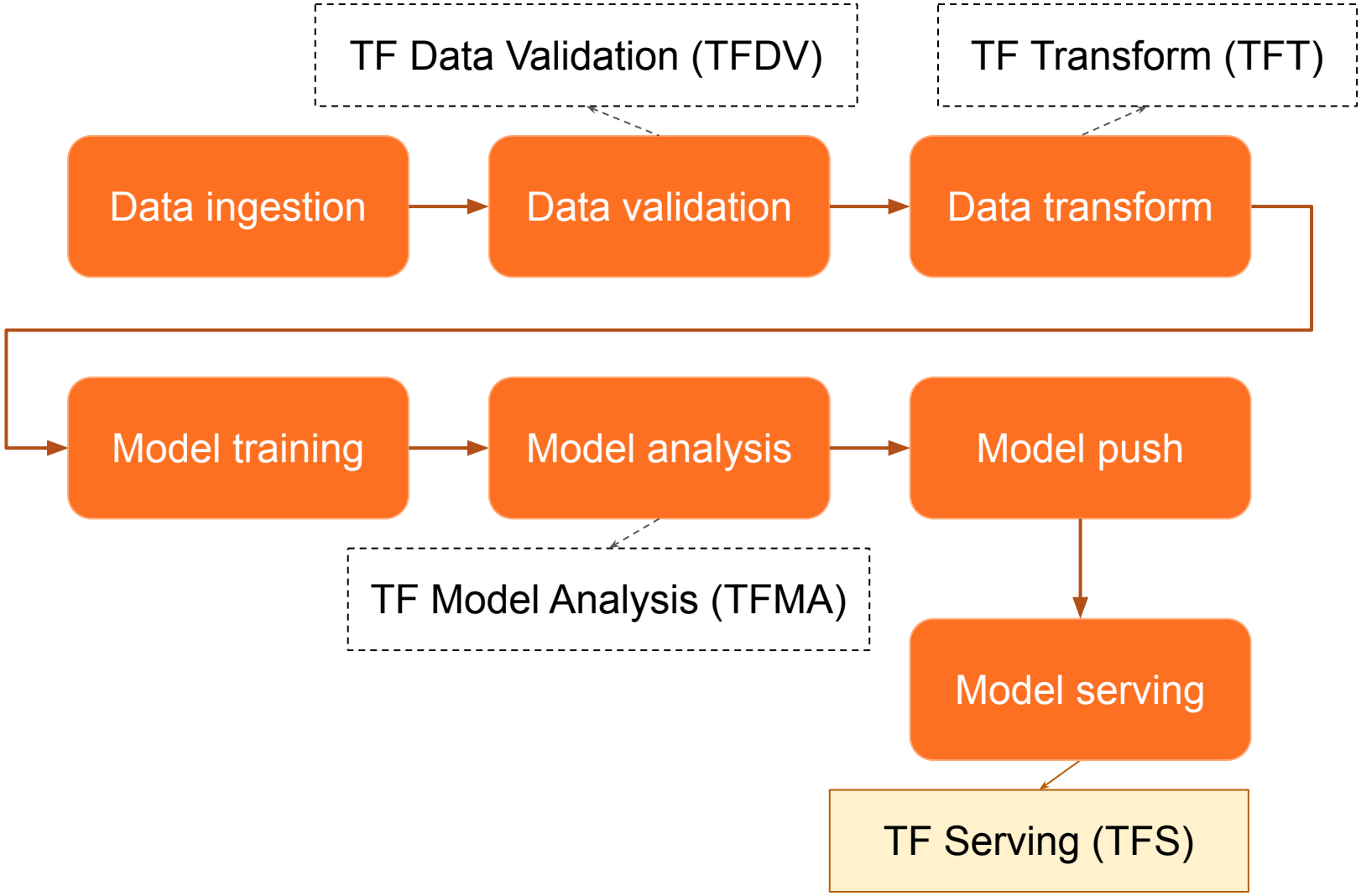
TensorFlow.js

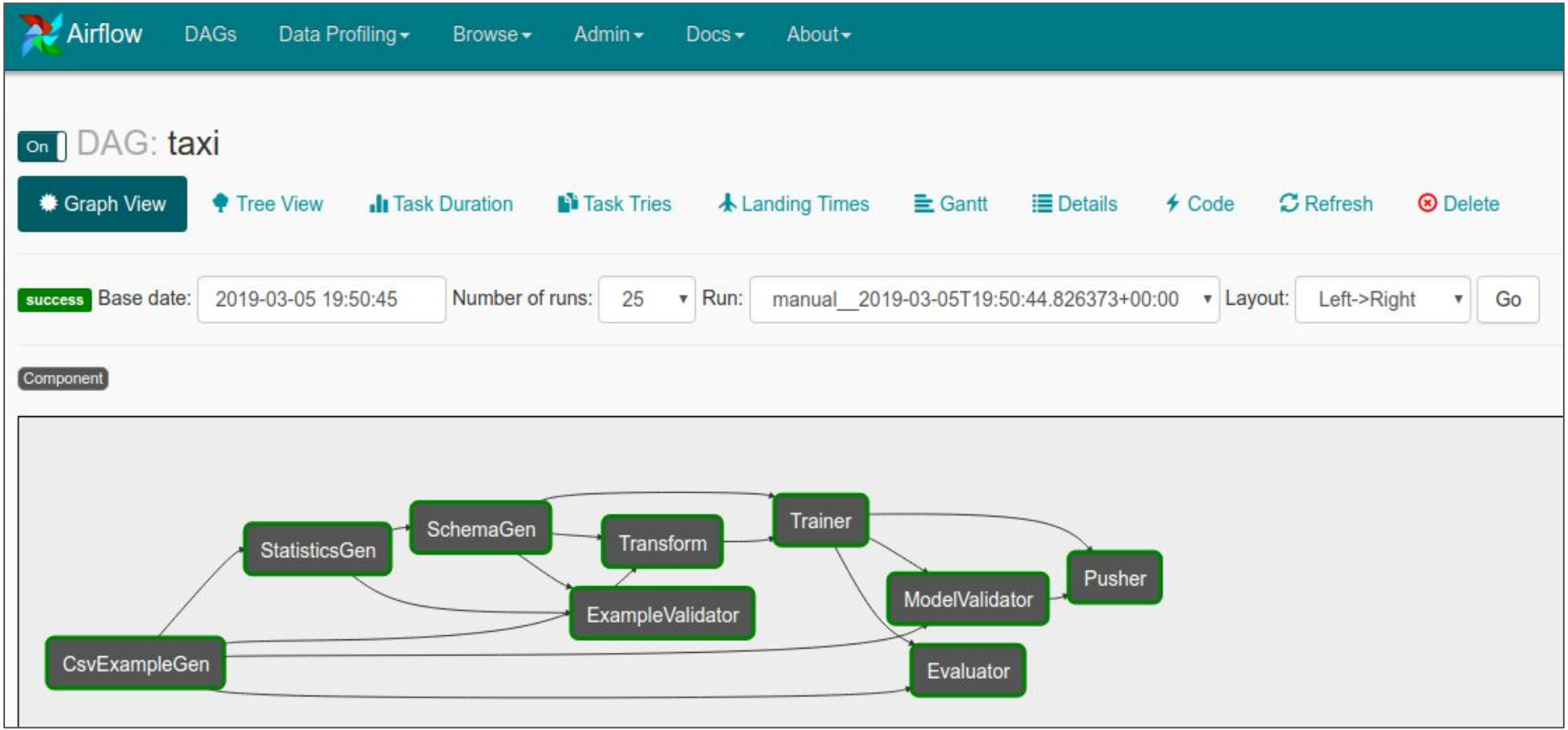
TensorFlow Lite

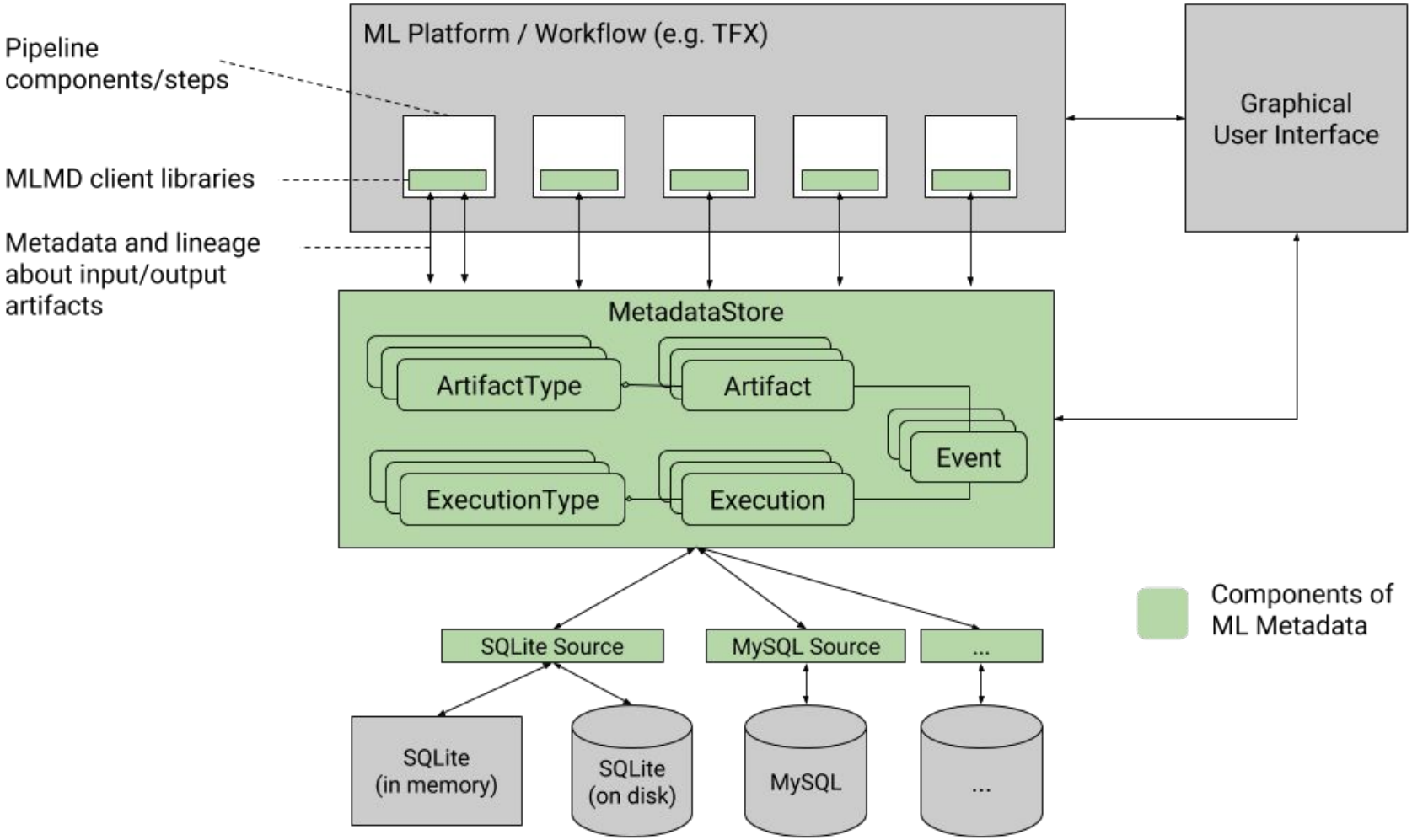

TF Data Validation (TFDV)
Data ingestion
Data validation
Data transform
Model training
Model analysis
Model push
Model serving

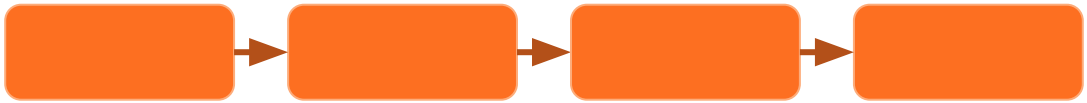

TF Data Validation (TFDV)
TF Transform (TFT)
Data ingestion
Data validation
Data transform
Model training
Model analysis
Model push
Model serving

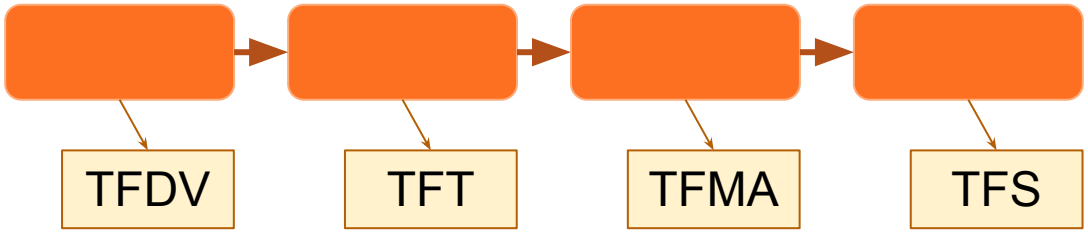

TF Data Validation (TFDV)
TF Transform (TFT)
Data ingestion
Data validation
Data transform
Model training
Model analysis
Model push
TF Model Analysis (TFMA)
Model serving


TF Data Validation (TFDV)
TF Transform (TFT)
Data ingestion
Data validation
Data transform
Model training
Model analysis
Model push
TF Model Analysis (TFMA)
Model serving
TF Serving (TFS)

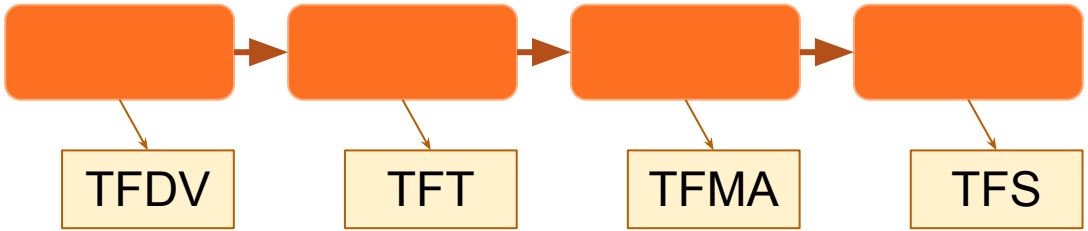

Airflow
DAGs
Data Profiling
Browse
Admin
Docs
About
On
DAG: taxi
Graph View
Tree View
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
Refresh
Delete
success
Base date:
2019-03-05 19:50:45
Number of runs:
25
Run:
manual__2019-03-05T19:50:44.826373+00:00
Layout:
Left->Right
Go
Component
CsvExampleGen
StatisticsGen
SchemaGen
Transform
ExampleValidator
Trainer
ModelValidator
Pusher
Evaluator

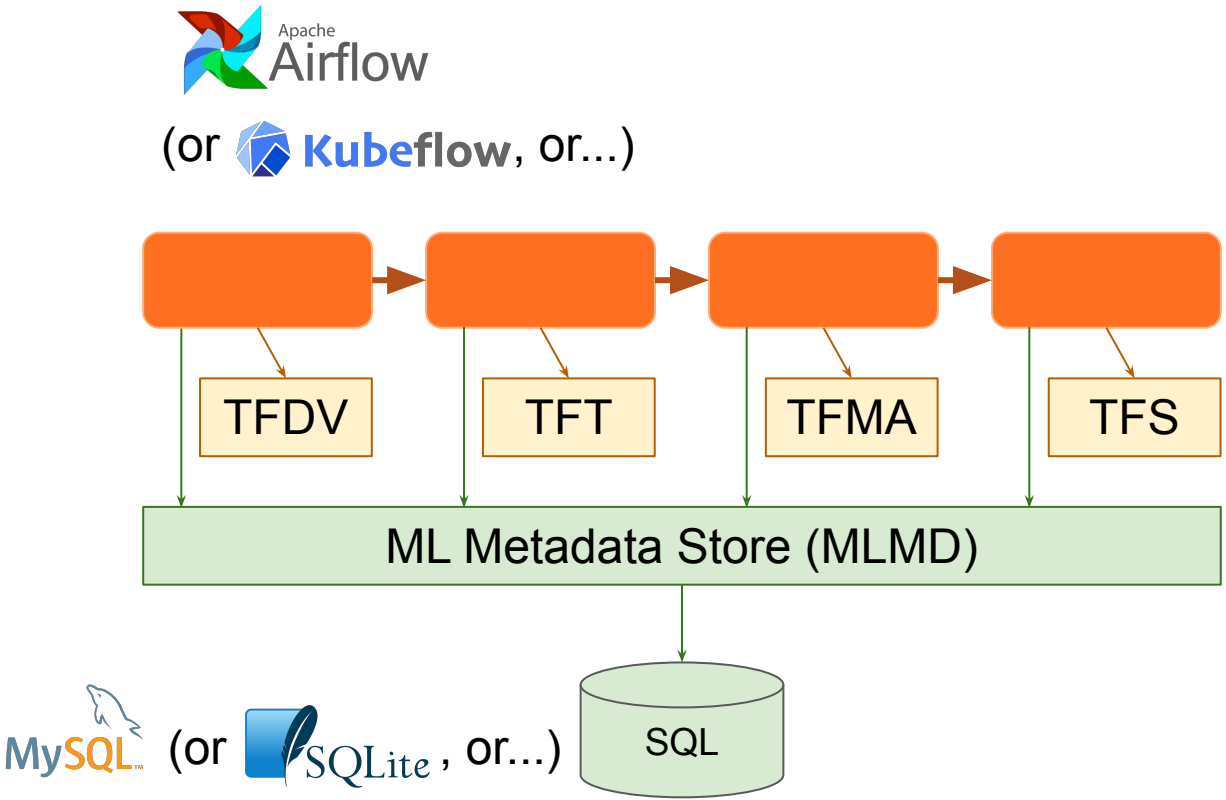

Pipeline components/steps
MLMD client libraries
Metadata and lineage about input/output artifacts
ML Platform / Workflow (e.g. TFX)
Graphical User Interface
MetadataStore
ArtifactType
Artifact
Event
ExecutionType
Execution
SQLite Source
MySQL Source
...
SQLite (in memory)
SQLite (on disk)
MySQL
...
Components of ML Metadata

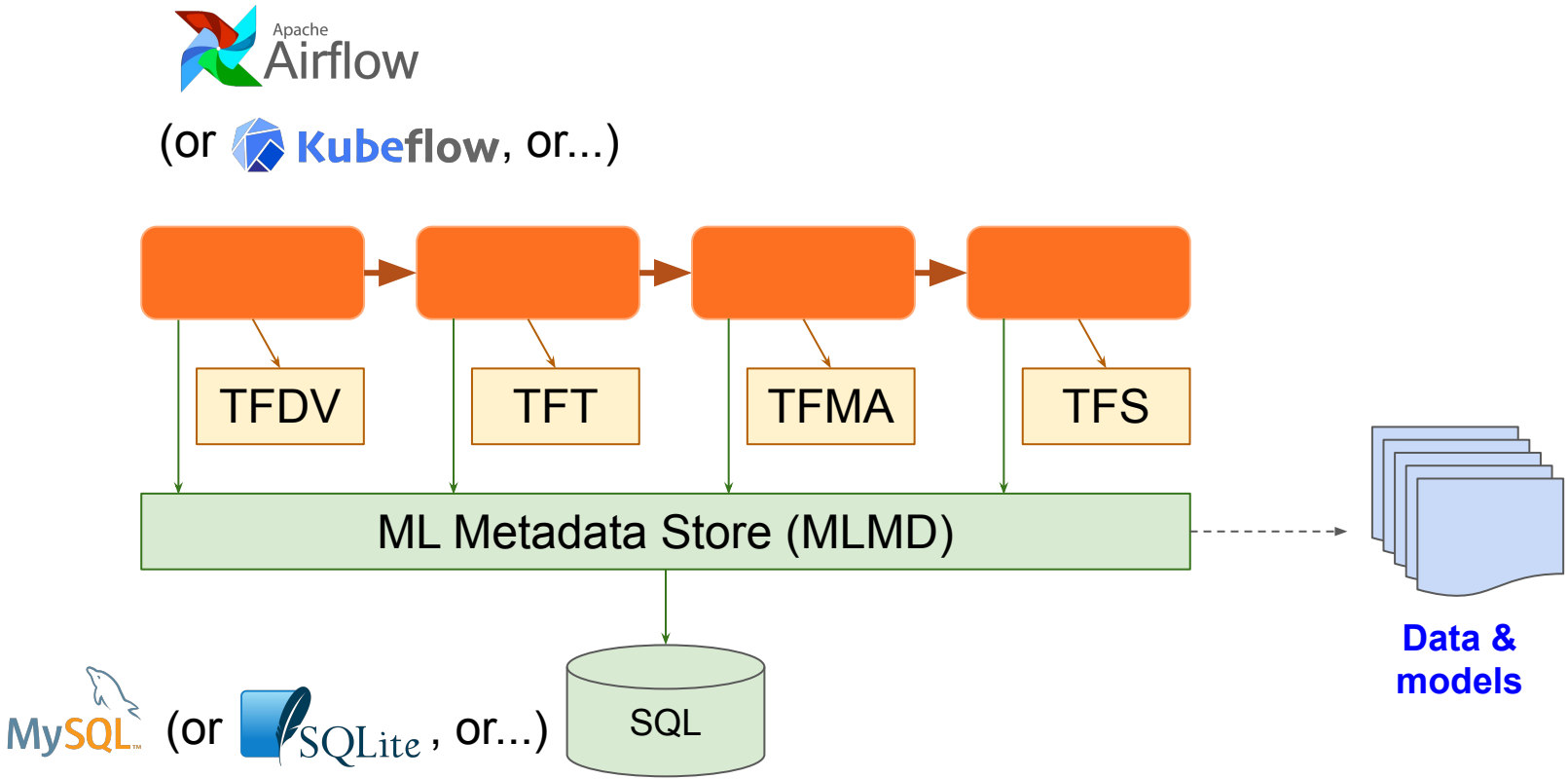


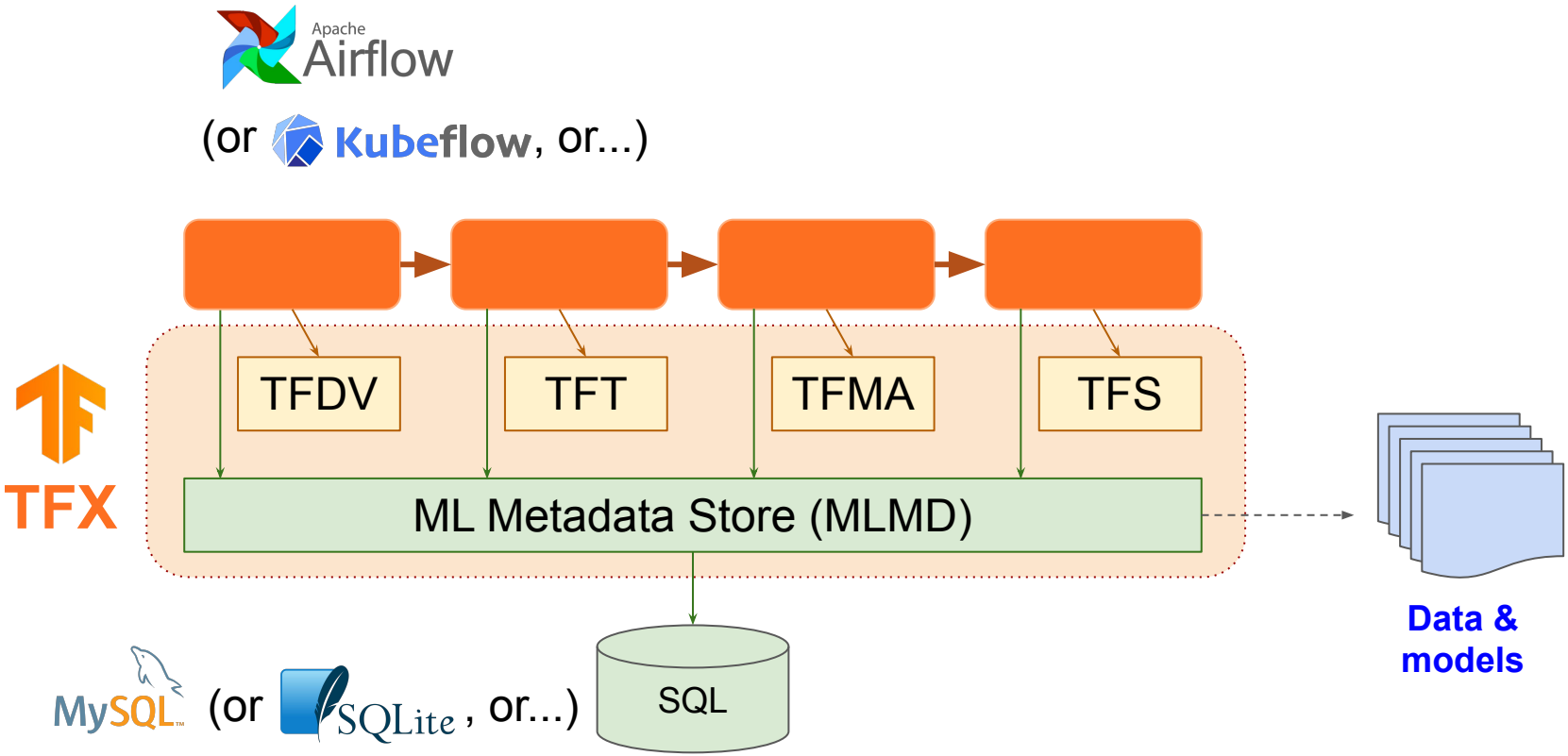

TFDV
TFT
TFMA
TFS

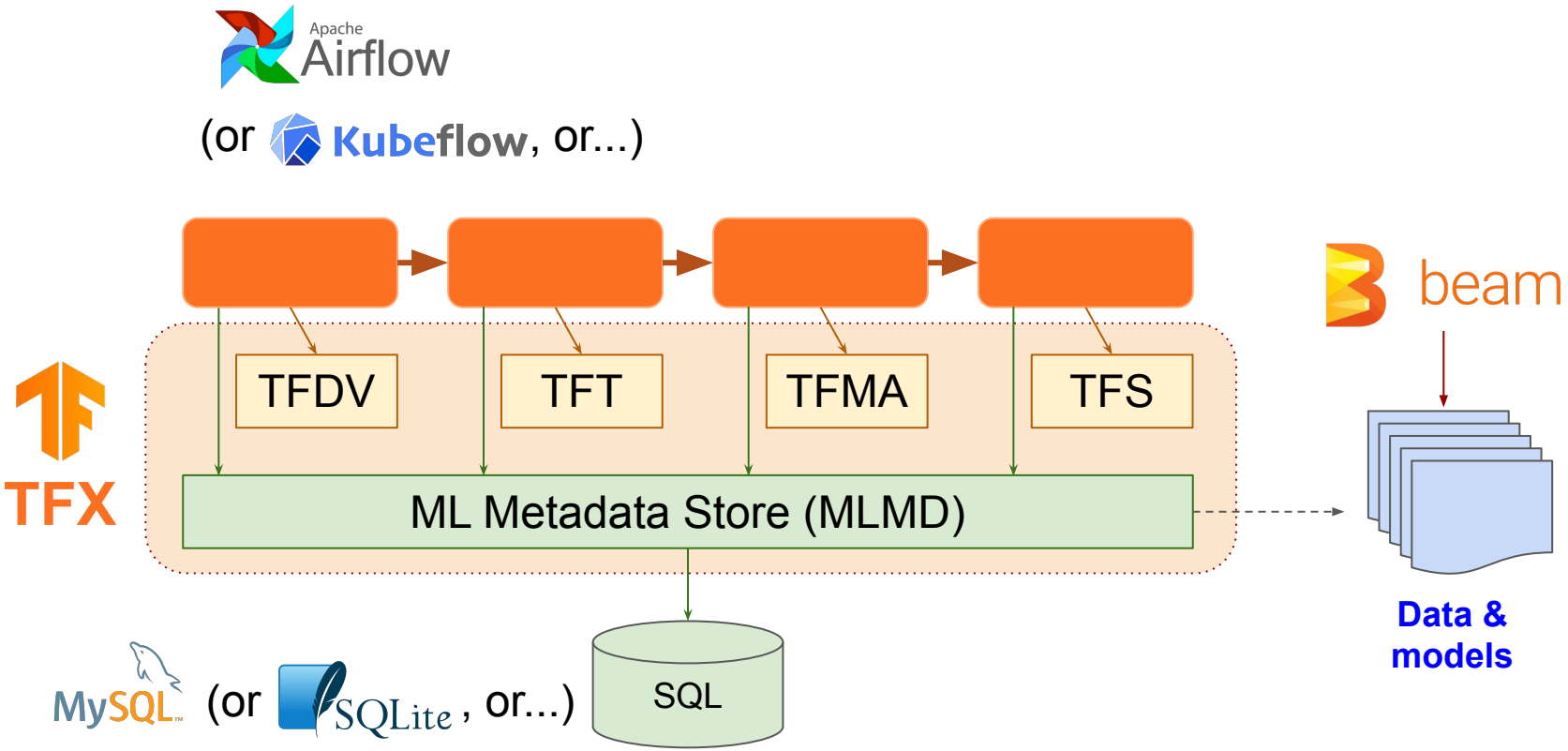
(or  Kubeflow, or...)

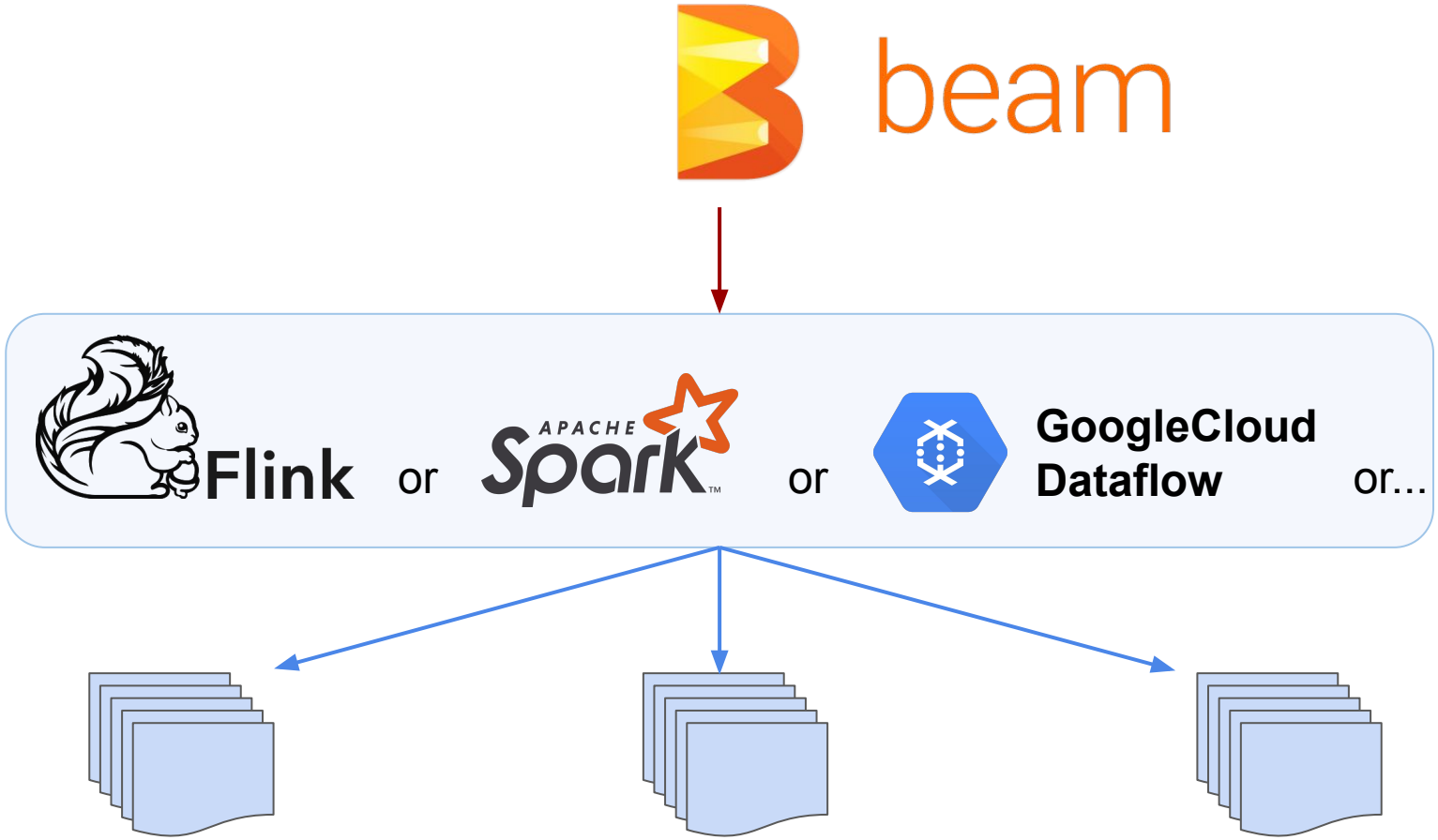



Apache Airflow
(or Kubeflow, or...)
TFDV
TFT
TFMA
TFS
ML Metadata Store (MLMD)
SQL
MySQL
(or SQLite , or...)


Apache Airflow
(or Kubeflow, or...)
TFDV
TFT
TFMA
TFS
ML Metadata Store (MLMD)
Data & models
MySQL
(or SQLite, or...)
SQL


Apache Airflow
(or Kubeflow, or...)
TFDV
TFT
TFMA
TFS
TFX
ML Metadata Store (MLMD)
Data & models
MySQL
(or SQLite, or...)
SQL


Apache Airflow
(or Kubeflow, or...)
TFDV
TFT
TFMA
TFS
TFX
ML Metadata Store (MLMD)
beam
Data & models
MySQL
(or SQLite, or...)
SQL


beam
Flink
or
APACHE
Spark
or
GoogleCloud
Dataflow
or...

Orchestration:   ...

Metadata Store:   ...

Processing API: 

Beam Runner:    ...

Orchestration: Manual w/ `InteractiveContext`

Metadata Store: 

Processing API: 

Beam Runner: 

Exercise 1

Prerequisites

- Linux / MacOS
- Python 3.6
- Virtualenv
- Git

Step 1: Setup your environment

```
% sudo apt-get update
% sudo apt-get install -y python3-dev python3-pip virtualenv
% sudo apt-get install -y build-essential libssl-dev libffi-dev
% sudo apt-get install -y libxml2-dev libxslt1-dev zlib1g-dev
% sudo apt-get install -y git-core
```

Step 1: Setup your environment

```
% cd
% virtualenv -p python3.6 tfx_env
% source tfx_env/bin/activate
(tfx_env) mkdir tfx; cd tfx
(tfx_env) pip install tensorflow==2.0.0
(tfx_env) pip install tfx==0.14.0
```

TFX End-to-End Example

Predicting Online News Popularity

TFX End-to-End Example

Online News Popularity Dataset

Features

Categorical Features	Date Features	Text Features	Numerical Features
data_channel (Lifestyle, Tech…)	publication_date	slug (e.g., snow-dogs)	n_unique_tokens
weekday (Monday, Tuesday…)			n_hrefs
			n_imgs
			global_subjectivity
			kw_avg_max (best kw avg shares)
			self_reference_avg_shares
			global_sentiment_polarity
			…

Label = n_shares

	Parses	Transforms	Expects Label
train_input_fn	No	No	Yes

	Parses	Transforms	Expects Label
train_input_fn	No	No	Yes
eval_input_fn	No	No	Yes

	Parses	Transforms	Expects Label
train_input_fn	No	No	Yes
eval_input_fn	No	No	Yes
serving_receiver_fn	Yes	Yes	No

	Parses	Transforms	Expects Label
train_input_fn	No	No	Yes
eval_input_fn	No	No	Yes
serving_receiver_fn	Yes	Yes	No
receiver_fn (TFMA)	Yes	Yes	Yes

	Parses	Transforms	Expects Label
train_input_fn	No	No	Yes
eval_input_fn	No	No	Yes
serving_receiver_fn	Yes	Yes	No
receiver_fn (TFMA)	Yes	Yes	Yes

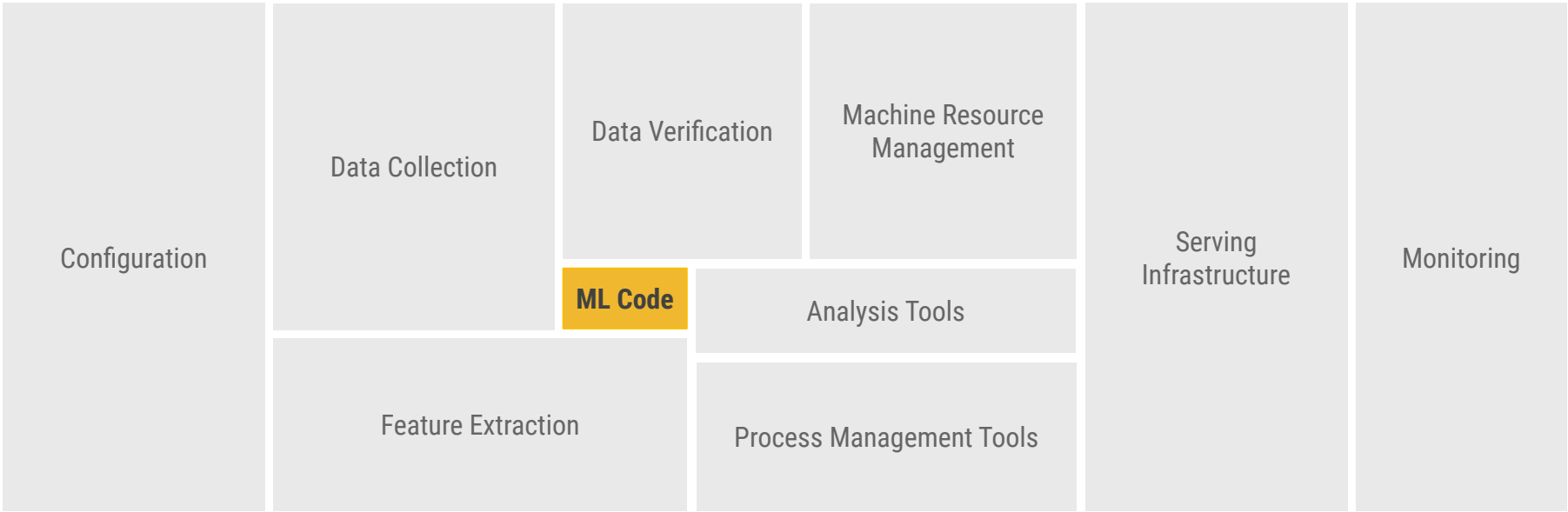
receiver_fn (TFMA) must return both the raw features and the transformed features

Lab 1

Running a simple TFX pipeline manually in a Colab Notebook

ML Coding vs ML Engineering

ML Coding vs ML engineering



Adapted from: Sculley et al.: Hidden Technical Debt in Machine Learning Systems

Writing Software (Programming)

Programming **in the small** (Coding)	Programming **in the large** (Engineering)
Monolithic code	Modular design and implementation
Non-reusable code	Libraries for reuse (ideally across languages)
Undocumented code	Well documented contracts and abstractions
Untested code	Well tested code (exhaustively and at scale)
Unbenchmarked or hack-optimized once code	Continuously benchmarked and optimized code
Unverified code	Reviewed and peer verified code
Undebuggable code or adhoc tooling	Debuggable code and debug tooling
Uninstrumented code	Instrumentable and instrumented code
...	...

Writing ML Software (The “Code” view)

ML Programming **in the small** (Coding)	**ML** Programming **in the large** (Engineering)
Monolithic code	Modular design and implementation
Non-reusable code	Libraries for reuse (ideally across languages)
Undocumented code	Well documented contracts and abstractions
Untested code	Well tested code (exhaustively and at scale)
Unbenchmarked or hack-optimized once code	Continuously benchmarked and optimized code
Unverified code	Reviewed and peer verified code
Undebuggable code or adhoc tooling	Debuggable code and debug tooling
Uninstrumented code	Instrumentable and instrumented code

This slide is, not surprisingly, the same as the previous one however it is **only half** the story :)

Engineering

```
// Strong Contracts.
Output Program(Inputs) {
... Human authored and peer reviewed code ...
}

// Exhaustive testing of Contracts and Performance.
TestProgramCommonCase1...N {
  ...
}

TestProgramEdgeCase1...N() {
  EXPECT_EQ(..., Program(...))
}

BenchmarkProgramWorstCase1...N {
  ...
}
```

Engineering vs ML Engineering

```
// Strong Contracts.
Output Program(Inputs) {
... Human authored and peer reviewed code ...
}

// Exhaustive testing of Contracts and Performance.
TestProgramCommonCase1...N {
  ...
}

TestProgramEdgeCase1...N() {
  EXPECT_EQ(..., Program(...))
}

BenchmarkProgramWorstCase1...N {
  ...
}
```

```
// Subjective representations and unclear objectives
LearnedProgram Learning(ProblemStatement, Data) {
... Human authored and peer reviewed ML pipelines ...
}

// Unclear Contracts
Output LearnedProgram(Inputs) {
... Opaque “Program” (aka Model) ...
}

// Peer Reviewing of ProblemStatement

// Data Validation
Expectations for Data on
  ● Shape, Invariants, Distribution(s), ...

// Model Validation
Expectations for LearnedProgram “average” behavior on:
  ● “Metrics” {Quality, Fairness, Perf, ...}
  Cross Product
  ● “Data Slices” {Global, UserChosen, AutoChosen,
    ...}
```

Writing ML Software (The “Data and other Artifacts” view)

ML Programming **in the small** (Coding)	ML Programming **in the large** (Engineering)
~~Monolithic code~~ Fixed Datasets	Evolving Datasets (Data, Features, ...) and Objectives
~~Non-reusable code~~ Unmergeable Artifacts	Reusable Models aka Modules, Mergeable Statistics ...
~~Undocumented code~~ No Problem Statements	Problem Statements, Discoverable Artifacts
~~Untested code~~ Non-validated Datasets, Models	Expectations, Data Validation, Model Validation ...
Unbenchmarked or hack-optimized once ~~code~~ Models	Quality and Performance Benchmarked Models ...
~~Unverified code~~ Biased Datasets / Artifacts	{Data, Model} x {Understanding, Fairness}
~~Undebuggable code or adhoc tooling~~	Visualizations, Summarizations, Understanding ...
~~Uninstrumented code~~	Full Artifact Lineage

This is the **remaining half!**

Introduction to 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: **Local, Apache Flink**, Apache Spark, Apache Gearpump, Apache Samza, Apache Hadoop, **Google Cloud Dataflow**, ...

Apache Beam

Java

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

Python

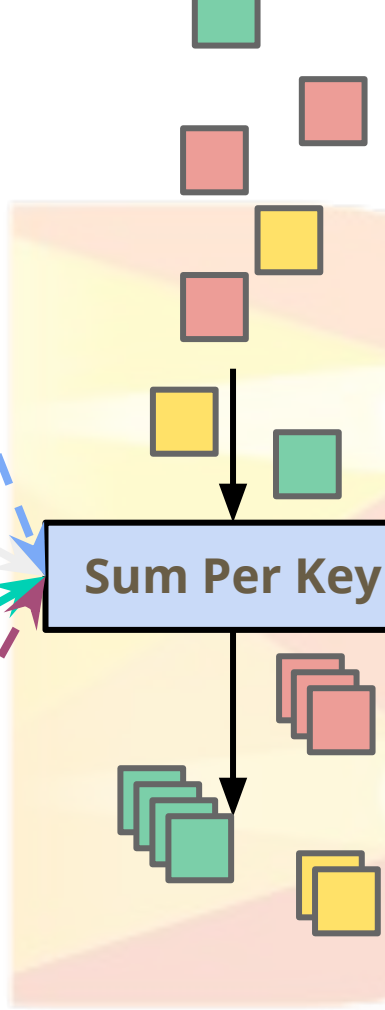
```
input | Sum.PerKey()
```

Go

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

SQL

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



- Cloud Dataflow
- Apache Flink
- Apache Spark
- Apache Apex
- Gearpump
- IBM Streams
- Apache Samza
- Apache Nemo (incubating)
- ⋮

Beam Portability Framework

- **Currently most runners support the Java SDK only**
- Portability framework (https://beam.apache.org/roadmap/portability/) aims to provide full interoperability across the Beam ecosystem
- Portability API
  - Protobufs and gRPC for broad language support
  - Job submission and management: The Runner API
  - Job execution: The SDK harness
- **Python Flink and Spark runners use Portability Framework**

Beam Portability Support Matrix

Apache Beam Portability Support Matrix			https://s.apache.org/apache-beam-portability-support-table					
			Flink (master)		**Spark (master)**		**Dataflow**	
			Python		Python		Python	
FEATURE			Batch	Streaming	Batch	Streaming	Batch	Streaming
	Impulse							
	ParDo							
		w/ side input						
		w/ multiple output						
		w/ user state					BEAM-2902	BEAM-2902
		w/ user timers						
		w/ user metrics						
	Flatten							
		w/ explicit flatten						
	Combine							
		w/ first-class rep						
		w/ lifting						
	SDF							
		w/ liquid sharding						
	GBK							
	CoGBK							
	WindowInto							
		w/ sessions						
		w/ custom windowfn						
Legend								
	Works, based on manual verification. Test desirable.							
BEAM-xxx	Partially works. Cell should contain JIRA.							
BEAM-xxx	Does not work. Cell should contain JIRA.							
	No information. To be evaluated.							

Hello World Example

```
pipeline = beam.Pipeline()

lines = (pipeline
         | "Create" >> beam.Create(["Hello", "World", "!!!"])
         | "Print" >> beam.ParDo(print))

result = pipeline.run()

result.state
```

Hello World Example

```
with beam.Pipeline() as pipeline:
  lines = (pipeline
           | "Create" >> beam.Create(["Hello", "World", "!!!"])
           | "Print" >> beam.ParDo(print))
```

Concepts

- Pipeline
- PCollection
- PTransform
- I/O transforms

Pipeline

- A Pipeline encapsulates your entire data processing task
- This includes reading input data, transforming that data, and writing output data.
- All Beam driver programs must create a Pipeline.
- You can specify the execution options when creating the Pipeline to tell it where and how to run.

Pipeline

```
pipeline = beam.Pipeline()

lines = (pipeline
         | "Create" >> beam.Create(["Hello", "World", "!!!"])
         | "Print" >> beam.ParDo(print))

result = pipeline.run()

result.state
```

PCollection

- A distributed dataset your Beam pipeline operates on.
- The dataset can be bounded (from fixed source) or unbounded (from a continuously updating source).
- The pipeline typically creates a source PCollection by reading data from an external data source
  - But you can also create a PCollection from in-memory data within your driver program.
- From there, PCollections are the inputs and outputs for each step in your pipeline.

PCollection

```
pipeline = beam.Pipeline()
lines = (pipeline
         | "Create" >> beam.Create(["Hello", "World", "!!!"])
         | "Print" >> beam.ParDo(print))
result = pipeline.run()
result.state
```

PTransform

- A PTransform represents a data processing operation, or a step, in your pipeline.
- Every PTransform takes one or more PCollection objects as input
- It performs a processing function that you provide on the elements of that PCollection.
- It produces zero or more output PCollection objects.

PTransform

```
pipeline = beam.Pipeline()
lines = (pipeline
         | "Create" >> beam.Create(["Hello", "World", "!!!"])
         | "Print" >> beam.ParDo(print))
result = pipeline.run()
result.state
```

output

```
Hello
World
!!!
```

I/O Transforms

- Beam comes with a number of “IOs” library PTransforms.
- They read or write data to various external storage systems.

I/O Transforms

```
with beam.Pipeline() as pipeline:
  lines = (pipeline
           | beam.io.ReadFromTFRecord("test_in.tfrecord")
           | beam.Map(lambda line: line + b' processed')
           | beam.io.WriteToTFRecord("test_out.tfrecord"))
```

Lab 2

Introduction to Apache Beam

TFX's Beam Orchestrator

Orchestration:

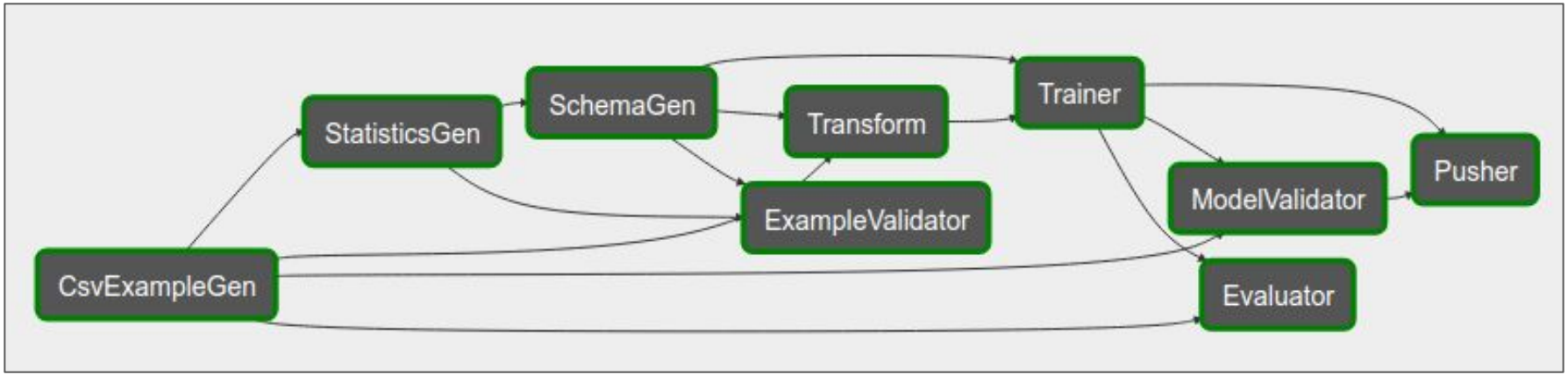
Metadata Store:

Processing API:

Beam Runner:

Exercise 3

```
from tfx.orchestration import pipeline


pipeline.Pipeline(
      pipeline_name=pipeline_name,
      pipeline_root=pipeline_root,
      components=[
          example_gen, statistics_gen, infer_schema, validate_stats,
          transform, trainer, model_analyzer, model_validator, pusher
      ],
      enable_cache=True,
      metadata_connection_config=sqlite_metadata_connection_config(
          metadata_path),
      additional_pipeline_args={},
  )
```


CsvExampleGen
StatisticsGen
SchemaGen
Transform
Trainer
ExampleValidator
ModelValidator
Pusher
Evaluator

Lab 3

On-Prem with Beam Orchestrator

TensorFlow Data Validation

Data Exploration & Cleanup

The first task in any data science or ML project is to understand and clean the data

- Understand the data types for each feature
- Look for anomalies and missing values
- Understand the distributions for each feature

```
import tensorflow_data_validation as tfdv

train_stats = tfdv.generate_statistics_from_csv(
    data_location=_train_data_filepath)

tfdv.visualize_statistics(train_stats)
```

Sort by

Feature order

Reverse order

Feature search (regex enabled)

Features: int(5) float(5) variable-length floats(5) string(1) fixed-length strings(1) variable-length strings(1)

Numeric Features (15)

Chart to show

Standard

log expand

	count	missing	mean	std dev	zeros	min	median	max
fare	10.1k	0%	11.73	12.18	0.17%	0	7.85	700.07
trip_start_hour	10.1k	0%	13.62	6.59	3.97%	0	15	23
dropoff_census_tract	10.1k	0%	17.0B	330k	0%	17.0B	17.0B	17.0B
trip_start_timestamp	10.1k	0%	1.41B	29.0M	0%	1.36B	1.41B	1.48B
pickup_longitude	10.1k	0%	-87.66	0.07	0%	-87.91	-87.63	-87.57
trip_start_month	10.1k	0%	6.61	3.4	0%	1	7	12

```
tfdv.visualize_statistics(
  lhs_statistics=eval_stats,
  rhs_statistics=train_stats,
  lhs_name='EVAL_DATASET',
  rhs_name='TRAIN_DATASET')
```

Sort by

Feature order

Reverse order

Feature search (regex enabled)

Features: int(1) float(47) string(3)

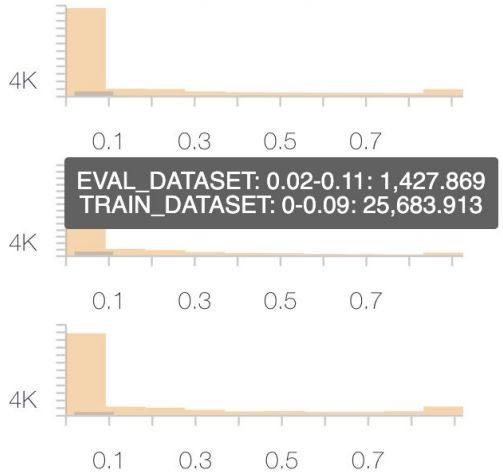
EVAL_DATASET TRAIN_DATASET

Numeric Features (48)

Chart to show: Standard

log expand percentages

	count	missing	mean	std dev	zeros	min	median	max
LDA_00								
	1,930	0%	0.16	0.25	0%	0.02	0.03	0.92
	36.0k	0%	0.19	0.27	0%	0	0.03	0.93
LDA_01								
	1,930	0%	0.14	0.22	0%	0.02	0.03	0.92
	36.0k	0%	0.14	0.22	0%	0	0.03	0.93
LDA_02								
	1,930	0%	0.27	0.31	0%	0.02	0.07	0.92
	36.0k	0%	0.21	0.28	0%	0	0.04	0.92



```
schema = tfdv.infer_schema(statistics=train_stats)
tfdv.display_schema(schema=schema)
```

Feature name	Type	Presence	Valency	Domain
'data_channel'	STRING	required		'data_channel'
'slug'	BYTES	required		-
'date'	BYTES	required		-
'n_hrefs'	FLOAT	required		-
'kw_max_avg'	FLOAT	required		-
'n_imgs'	FLOAT	required		-
'n_non_stop_unique_tokens'	FLOAT	required		-
'kw_min_max'	FLOAT	required		-
'self_reference_max_shares'	FLOAT	required		-

```
anomalies = tfdv.validate_statistics(
  statistics=eval_stats,
  schema=schema)

tfdv.display_anomalies(anomalies)
```

Feature name	Anomaly short description	Anomaly long description
'data_channel'	Unexpected string values	Examples contain values missing from the schema: Fun (<1%).

```
# Relax the minimum fraction of values that must come from
# the domain for feature company.
company = tfdv.get_feature(schema, 'company')
company.distribution_constraints.min_domain_mass = 0.9

# Add new value to the domain of feature payment_type.
payment_type = tfdv.get_domain(schema, 'payment_type')
payment_type.value.append('bitcoin')
```

```
# All features are by default in both TRAINING
# and SERVING environments.
schema.default_environment.append('TRAINING')
schema.default_environment.append('SERVING')

# Specify that 'tips' feature is not in SERVING
# environment.
n_shares_feature = tfdv.get_feature(schema, 'n_shares')
n_shares_feature.not_in_environment.append('SERVING')

serving_anomalies_with_env = tfdv.validate_statistics(
  serving_stats, schema, environment='SERVING')
```

```
# Add skew comparator for 'weekday' feature.
weekday = tfdv.get_feature(schema, 'weekday')
weekday.skew_comparator.infinity_norm.threshold = 0.01

# Add drift comparator for 'title_subjectivity' feature.
title_subjectivity = tfdv.get_feature(schema, 'title_subjectivity')
title_subjectivity.drift_comparator.infinity_norm.threshold = 0.001

skew_anomalies = tfdv.validate_statistics(
  train_stats,
  schema,
  previous_statistics=eval_stats,
  serving_statistics=serving_stats)
```

Lab 4

TensorFlow Data Validation (TFDV)

TensorFlow Transform

Data Preprocessing

The raw data usually needs to be prepared before being fed to a Machine Learning model. This may involve several transformations:

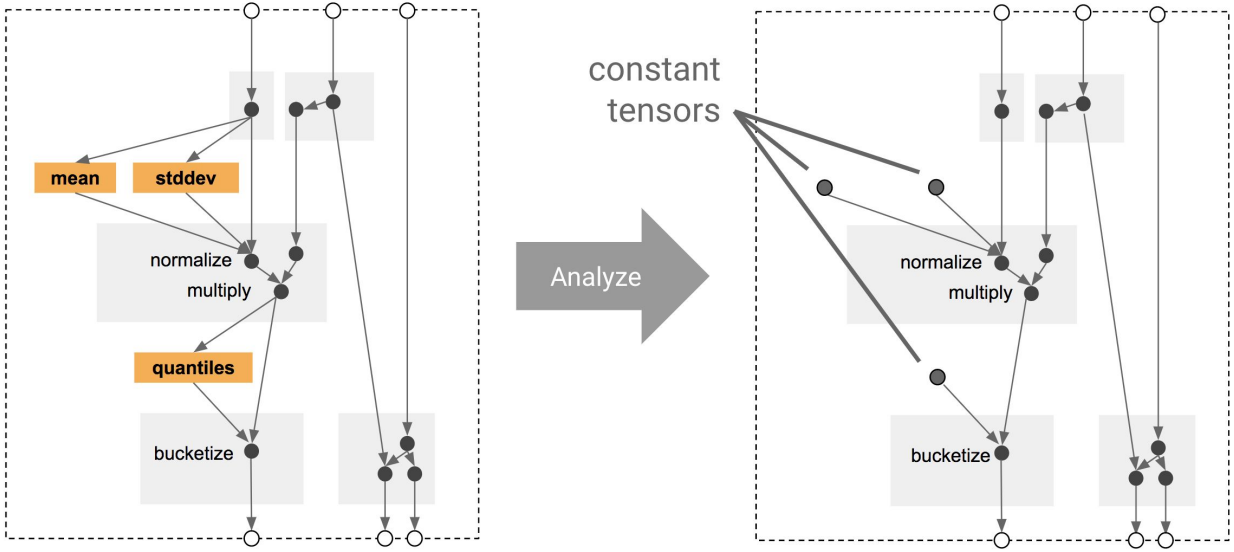
- Fill in missing values
- Normalize features
- Bucketize features
- Zoom/Crop images
- Augment images
- Feature crosses
- Vocabularies
- Embeddings
- PCA
- Categorical encoding

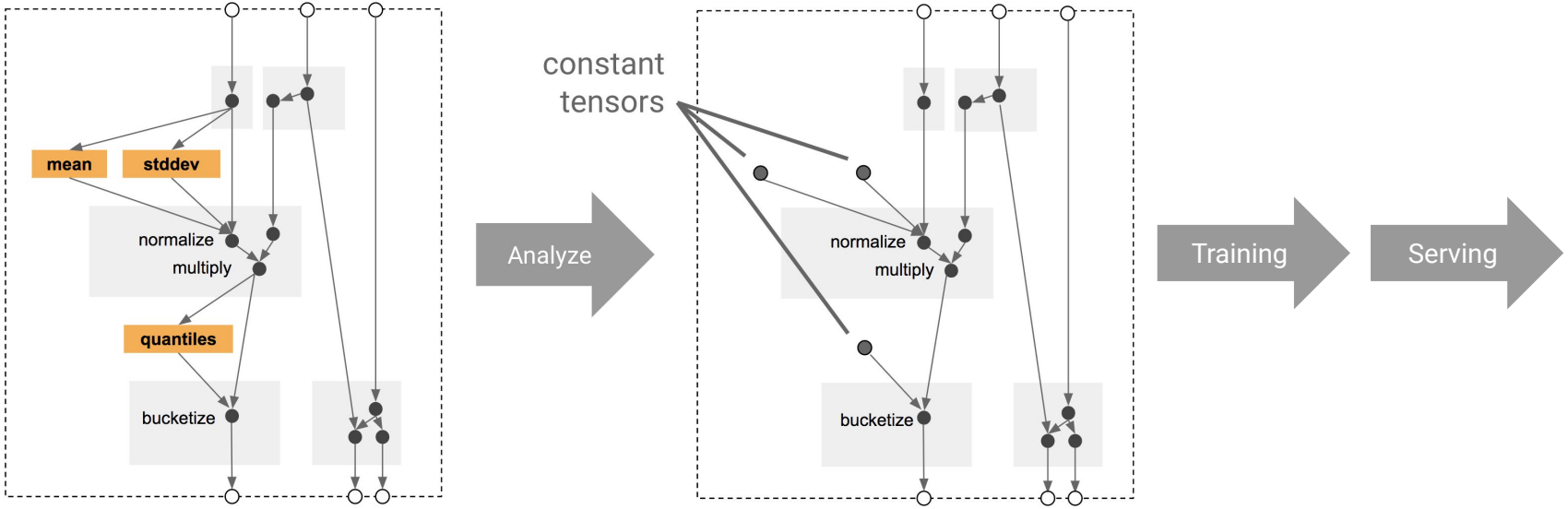
Training/Serving Skew

- Preprocessing data before training
- Same preprocessing required at serving time
- Possibly with multiple serving environments
- Risk of discrepancy

In-model preprocessing

- If we include the preprocessing steps in the TensorFlow graph, the problem is solved
- Except training is slow
  - Preprocessing runs once per epoch instead of just once


mean
stddev
normalize
multiply
quantiles
bucketize
Analyze
constant
tensors
normalize
multiply
bucketize


mean
stddev
normalize
multiply
quantiles
bucketize
Analyze
constant
tensors
normalize
multiply
bucketize
Training
Serving

```
RAW_DATA_FEATURE_SPEC = {
    "name": tf.io.FixedLenFeature([], tf.string)
}
```

```
RAW_DATA_FEATURE_SPEC = {
    "name": tf.io.FixedLenFeature([], tf.string)
}

RAW_DATA_METADATA = dataset_metadata.DatasetMetadata(
  schema_utils.schema_from_feature_spec(RAW_DATA_FEATURE_SPEC))
```

```
RAW_DATA_FEATURE_SPEC = {
    "name": tf.io.FixedLenFeature([], tf.string)
}

RAW_DATA_METADATA = dataset_metadata.DatasetMetadata(
  schema_utils.schema_from_feature_spec(RAW_DATA_FEATURE_SPEC))
```

```
{
  '_schema': feature {
    name: "name"
    type: BYTES
    presence {
      min_fraction: 1.0
    }
    shape {}
  }
}
```

```
data_coder = tft.coders.ExampleProtoCoder(
    RAW_DATA_METADATA.schema)
encoded = data_coder.encode({"name": "café"})
```

```
data_coder = tft.coders.ExampleProtoCoder(
    RAW_DATA_METADATA.schema)
encoded = data_coder.encode({"name": "café"})
```

```
b'\n\x13\n\x11\n\x04name\x12\t\n\x07\n\x05caf\xc3\xa9'
```

```
data_coder = tft.coders.ExampleProtoCoder(
    RAW_DATA_METADATA.schema)
encoded = data_coder.encode({"name": "café"})
```

```
b'\n\x13\n\x11\n\x04name\x12\t\n\x07\n\x05caf\xc3\xa9'
```

```
decoded = data_coder.decode(encoded)
```

```
data_coder = tft.coders.ExampleProtoCoder(
    RAW_DATA_METADATA.schema)
encoded = data_coder.encode({"name": "café"})
```

```
b'\n\x13\n\x11\n\x04name\x12\t\n\x07\n\x05caf\xc3\xa9'
```

```
decoded = data_coder.decode(encoded)
```

```
{'name': b'caf\xc3\xa9'}
```

```
tmp_dir = tempfile.mkdtemp(prefix="tft-data")
train_path = os.path.join(tmp_dir, "train.tfrecord")

with beam.Pipeline() as pipeline:
    _ = (pipeline
      | "Create" >> beam.Create(["Alice", "Bob", "Cathy", "Alice"])
      | "ToDict" >> beam.Map(lambda name: {"name": name})
      | "Encode" >> beam.Map(data_coder.encode)
      | "Write" >> beam.io.WriteToTFRecord(train_path)
    )
```

```
tmp_dir = tempfile.mkdtemp(prefix="tft-data")
train_path = os.path.join(tmp_dir, "train.tfrecord")

with beam.Pipeline() as pipeline:
    _ = (pipeline
      | "Create" >> beam.Create(["Alice", "Bob", "Cathy", "Alice"])
      | "ToDict" >> beam.Map(lambda name: {"name": name})
      | "Encode" >> beam.Map(data_coder.encode)
      | "Write" >> beam.io.WriteToTFRecord(train_path)
    )
```

```
/tmp/tft-datac1z2ichz/train.tfrecord-00000-of-00001
```

```
eval_path = os.path.join(tmp_dir, "eval.tfrecord")

with beam.Pipeline() as pipeline:
    _ = (pipeline
      | "Create" >> beam.Create(["Denis", "Alice"])
      | "ToDict" >> beam.Map(lambda name: {"name": name})
      | "Encode" >> beam.Map(data_coder.encode)
      | "Write" >> beam.io.WriteToTFRecord(eval_path)
    )
```

```
/tmp/tft-datac1z2ichz/eval.tfrecord-00000-of-00001
```

```
with beam.Pipeline() as pipeline:
    _ = (pipeline
      | "Read" >> beam.io.ReadFromTFRecord(f"{train_path}*")
      | "Decode" >> beam.Map(data_coder.decode)
      | "Print" >> beam.Map(print)
    )
```

```
with beam.Pipeline() as pipeline:
    _ = (pipeline
      | "Read" >> beam.io.ReadFromTFRecord(f"{train_path}*")
      | "Decode" >> beam.Map(data_coder.decode)
      | "Print" >> beam.Map(print)
    )
```

```
{'name': b'Alice'}
{'name': b'Bob'}
{'name': b'Cathy'}
{'name': b'Alice'}
```

```
with beam.Pipeline() as pipeline:
    _ = (pipeline
      | "Read" >> beam.io.ReadFromTFRecord(f"{eval_path}*")
      | "Decode" >> beam.Map(data_coder.decode)
      | "Print" >> beam.Map(print)
    )
```

```
{'name': b'Denis'}
{'name': b'Alice'}
```

```
def preprocessing_fn(inputs):
  outputs = {}
  lower = tf.strings.lower(inputs["name"])
  outputs["name_xf"] = tft.compute_and_apply_vocabulary(lower)
  return outputs
```

https://www.tensorflow.org/tfx/transform/api_docs

Buckets

- apply_buckets()
- apply_buckets_with_interpolation()
- bucketize()
- bucketize_per_key()

Text & Categories

- apply_vocabulary()
- bag_of_words()
- compute_and_apply_vocabulary()
- hash_strings()
- ngrams()
- vocabulary()
- word_count()
- tfidf()

Apply arbitrary transformations

- apply_function_with_checkpoint()
- apply_pyfunc()
- apply_saved_model()
- ptransform_analyzer()

Math

- covariance()
- max()
- mean()
- min()
- pca()
- quantiles()
- scale_by_min_max()
- scale_by_min_max_per_key()
- scale_to_0_1()
- scale_to_0_1_per_key()
- scale_to_z_score()
- scale_to_z_score_per_key()
- size()
- sum()
- var()

Misc

- deduplicate_tensor_per_row()
- get_analyze_input_columns()
- get_transform_input_columns()
- segment_indices()
- sparse_tensor_to_dense_with_shape()

```
with beam.Pipeline() as pipeline:

    train_data = (pipeline
      | "ReadTrain" >> beam.io.ReadFromTFRecord(f"{train_path}*")
      | "DecodeTrain" >> beam.Map(data_coder.decode)
    )
```

```
with beam.Pipeline() as pipeline:

    train_data = (pipeline
      | "ReadTrain" >> beam.io.ReadFromTFRecord(f"{train_path}*")
      | "DecodeTrain" >> beam.Map(data_coder.decode)
    )

    train_dataset = (train_data, RAW_DATA_METADATA)
    train_dataset_xf, transform_fn = (train_dataset
      | tft.beam.AnalyzeAndTransformDataset(preprocessing_fn))
    train_data_xf, metadata_xf = train_dataset_xf
```

```
with beam.Pipeline() as pipeline:
  with tft.beam.Context(temp_dir=tmp_dir):
    train_data = (pipeline
      | "ReadTrain" >> beam.io.ReadFromTFRecord(f"{train_path}*")
      | "DecodeTrain" >> beam.Map(data_coder.decode)
    )

    train_dataset = (train_data, RAW_DATA_METADATA)
    train_dataset_xf, transform_fn = (train_dataset
      | tft.beam.AnalyzeAndTransformDataset(preprocessing_fn))
    train_data_xf, metadata_xf = train_dataset_xf
```

```
with beam.Pipeline() as pipeline:
  with tft.beam.Context(temp_dir=tmp_dir):
    train_data = (pipeline
      | "ReadTrain" >> beam.io.ReadFromTFRecord(f"{train_path}*")
      | "DecodeTrain" >> beam.Map(data_coder.decode)
    )

    train_dataset = (train_data, RAW_DATA_METADATA)
    train_dataset_xf, transform_fn = (train_dataset
      | tft.beam.AnalyzeAndTransformDataset(preprocessing_fn))
    train_data_xf, metadata_xf = train_dataset_xf

    data_xf_coder = tft.coders.ExampleProtoCoder(metadata_xf.schema)
    _ = (train_xf_data
      | 'EncodeTrainData' >> beam.Map(data_xf_coder.encode)
      | 'WriteTrainData' >> beam.io.WriteToTFRecord(train_xf_path)
    )
```

```
with beam.Pipeline() as pipeline:
  with tft.beam.Context(temp_dir=tmp_dir):
    [...]
    eval_data = (pipeline
      | "ReadEval" >> beam.io.ReadFromTFRecord(f"{eval_path}*")
      | "DecodeEval" >> beam.Map(data_coder.decode)
    )
```

```
with beam.Pipeline() as pipeline:
  with tft.beam.Context(temp_dir=tmp_dir):
    [...]
    eval_data = (pipeline
      | "ReadEval" >> beam.io.ReadFromTFRecord(f"{eval_path}*")
      | "DecodeEval" >> beam.Map(data_coder.decode)
    )
    eval_dataset = (eval_data, RAW_DATA_METADATA)
    eval_dataset_xf = ((eval_dataset, transform_fn)
      | tft.beam.TransformDataset())
    eval_data_xf, _ = eval_dataset_xf
```

```
with beam.Pipeline() as pipeline:
  with tft.beam.Context(temp_dir=tmp_dir):
    [...]
    eval_data = (pipeline
      | "ReadEval" >> beam.io.ReadFromTFRecord(f"{eval_path}*")
      | "DecodeEval" >> beam.Map(data_coder.decode)
    )
    eval_dataset = (eval_data, RAW_DATA_METADATA)
    eval_dataset_xf = ((eval_dataset, transform_fn)
      | tft.beam.TransformDataset())
    eval_data_xf, _ = eval_dataset_xf

    _ = (eval_data_xf
      | 'EncodeEvalData' >> beam.Map(data_xf_coder.encode)
      | 'WriteEvalData' >> beam.io.WriteToTFRecord(eval_xf_path)
    )
```

```
with beam.Pipeline() as pipeline:
  with tft.beam.Context(temp_dir=tmp_dir):
    [...]
    _ = (transform_fn
      | 'WriteTransformFn' >> tft.beam.WriteTransformFn(graph_dir))
```

```
with beam.Pipeline() as pipeline:
  _ = (pipeline
    | "Read" >> beam.io.ReadFromTFRecord(f"{train_xf_path}*")
    | "Decode" >> beam.Map(data_xf_coder.decode)
    | "Print" >> beam.ParDo(print)
  )
```

```
with beam.Pipeline() as pipeline:
  _ = (pipeline
    | "Read" >> beam.io.ReadFromTFRecord(f"{train_xf_path}*")
    | "Decode" >> beam.Map(data_xf_coder.decode)
    | "Print" >> beam.ParDo(print)
  )
```

```
{'name_xf': 0}
{'name_xf': 2}
{'name_xf': 1}
{'name_xf': 0}
```

```
with beam.Pipeline() as pipeline:
  _ = (pipeline
    | "Read" >> beam.io.ReadFromTFRecord(f"{eval_xf_path}*")
    | "Decode" >> beam.Map(data_xf_coder.decode)
    | "Print" >> beam.ParDo(print)
  )
```

```
{'name_xf': -1}
{'name_xf': 0}
```

metadata_xf.schema

```
feature {
  name: "name_xf"
  type: INT
  int_domain {
    is_categorical: true
  }
  presence {
    min_fraction: 1.0
  }
  shape {}
}
```

```
/tmp/tft-data0o6lwwt0/graph/
    transform_fn/
        assets/
            vocab_compute_and_apply_vocabulary_vocabulary
        variables
        saved_model.pb
    transformed_metadata/
        schema.pbtxt
```

```
/tmp/tft-data0o6lwwt0/graph/
    transform_fn/
        assets/
            vocab_compute_and_apply_vocabulary_vocabulary
        variables
        saved_model.pb
    transformed_metadata/
        schema.pbtxt
```

```
alice
cathy
bob
```

```
/tmp/tft-data0o6lwwt0/graph/
    transform_fn/
        assets/
            vocab_compute_and_apply_vocabulary_vocabulary
        variables/
        saved_model.pb
    transformed_metadata/
        schema.pbtxt
```

```
/tmp/tft-data0o6lwwt0/graph/
    transform_fn/
        assets/
            vocab_compute_and_apply_vocabulary_vocabulary
        variables/
        saved_model.pb
    transformed_metadata/
        schema.pbtxt
```

```
tft_output = tft.TFTransformOutput(graph_dir)

@tf.function
def transform_raw_features(example):
  return tft_output.transform_raw_features(example)

example = {"name": tf.constant(["Alice", "Bob", "Alice", "Suzy"])}
example_xf = transform_raw_features(example)
```

```
tft_output = tft.TFTransformOutput(graph_dir)

@tf.function
def transform_raw_features(example):
  return tft_output.transform_raw_features(example)

example = {"name": tf.constant(["Alice", "Bob", "Alice", "Suzy"])}
example_xf = transform_raw_features(example)
```

```
{'name_xf': <tf.Tensor: [...] numpy=array([ 0,  2,  0, -1])>}
```

```
transform = Transform(
    input_data=example_gen.outputs['examples'],
    schema=infer_schema.outputs['output'],
    module_file='my_transform.py')
context.run(transform)
```

```
transform.outputs
```

```
{
    'transform_output': Channel(
        type_name: TransformPath
        artifacts: [Artifact([...])]),
    'transformed_examples': Channel(
        type_name: ExamplesPath
        artifacts: [Artifact([...], split: train),
                    Artifact([...], split: eval)])
}
```

```
transform = Transform(
    input_data=example_gen.outputs['examples'],
    schema=infer_schema.outputs['output'],
    module_file='my_transform.py')
context.run(transform)
```

my_transform.py

```
import tensorflow  as tf
import tensorflow_transform as tft

def preprocessing_fn(inputs):
  outputs = {}
  outputs["name_xf"] = tft.compute_[...](inputs["name"])
  [...]
  return outputs
```

Lab 5

Preprocessing Data with TF Transform (TFT)

Analyzing Model Results

Understanding more than just the top level metrics

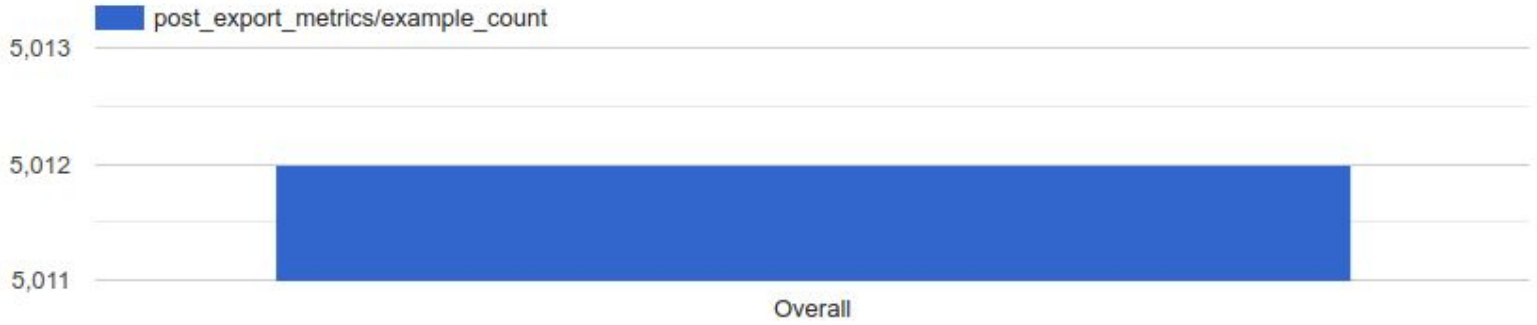
- Users experience model performance for their queries only
- Poor performance on slices of data can be hidden by top level metrics
- Model fairness is important
- Often key subsets of users or data are very important, and may be small
    - Performance in critical but unusual conditions
    - Performance for key audiences such as influencers

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
Overall	0.80188	0.77694	0.94034	0.70763	0.33329

```
eval_model = tfma.default_eval_shared_model(
    eval_saved_model_path='eval/run0/eval_model/0')

slices = [tfma.slicer.SingleSliceSpec(columns=['trip_start_hour']),
          tfma.slicer.SingleSliceSpec(columns=['trip_start_day'])]
```

```
eval_model = tfma.default_eval_shared_model(
    eval_saved_model_path='eval/run0/eval_model/0')

slices = [tfma.slicer.SingleSliceSpec(columns=['trip_start_hour']),
          tfma.slicer.SingleSliceSpec(columns=['trip_start_day'])]


eval_result = tfma.run_model_analysis(
    eval_shared_model=eval_model,
    data_location='data.tfrecord',
    file_format='tfrecords',
    slice_spec=slices,
    output_path='output/run0')
```

```
eval_model = tfma.default_eval_shared_model(
    eval_saved_model_path='eval/run0/eval_model/0')

slices = [tfma.slicer.SingleSliceSpec(columns=['trip_start_hour']),
          tfma.slicer.SingleSliceSpec(columns=['trip_start_day'])]


eval_result = tfma.run_model_analysis(
    eval_shared_model=eval_model,
    data_location='data.tfrecord',
    file_format='tfrecords',
    slice_spec=slices,
    output_path='output/run0')

tfma.view.render_slicing_metrics(
    eval_result,
    slicing_spec=slices[0])
```

```
eval_model = tfma.default_eval_shared_model(
    eval_saved_model_path='eval/run0/eval_model/0')

slices = [tfma.slicer.SingleSliceSpec(
              columns=['trip_start_day'],
              features=[('trip_start_hour', 12)]]

eval_result = tfma.run_model_analysis(
    eval_shared_model=eval_model,
    data_location='data.tfrecord',
    file_format='tfrecords',
    slice_spec=slices,
    output_path='output/run0')

tfma.view.render_slicing_metrics(
    eval_result,
    slicing_spec=slices[0])
```

```
eval_model = tfma.default_eval_shared_model(
    eval_saved_model_path='eval/run0/eval_model/0')

slices = [tfma.slicer.SingleSliceSpec(
              columns=['trip_start_day', 'trip_start_hour'])]


eval_result = tfma.run_model_analysis(
    eval_shared_model=eval_model,
    data_location='data.tfrecord',
    file_format='tfrecords',
    slice_spec=slices,
    output_path='output/run0')

tfma.view.render_slicing_metrics(
    eval_result,
    slicing_spec=slices[0])
```

```
output_dirs = [os.path.join("output", run_name)
               for run_name in ("run_0", "run_1", "run_2")]

eval_results_from_disk = tfma.load_eval_results(
    output_dirs, tfma.constants.MODEL_CENTRIC_MODE)
```

```
output_dirs = [os.path.join("output", run_name)
               for run_name in ("run_0", "run_1", "run_2")]

eval_results_from_disk = tfma.load_eval_results(
    output_dirs, tfma.constants.MODEL_CENTRIC_MODE)
```

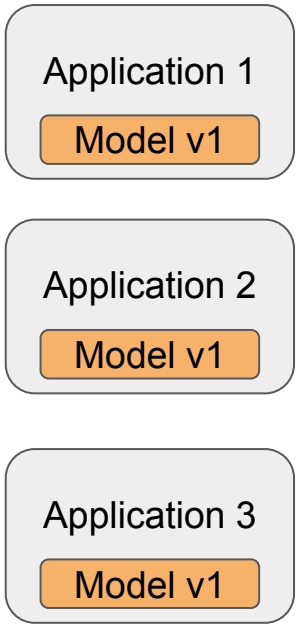
- **MODEL_CENTRIC_MODE**: main axis = model id
- **DATA_CENTRIC_MODE**: main axis = last data span

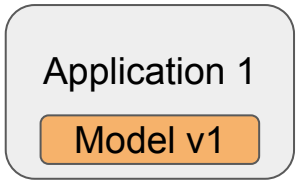
```
output_dirs = [os.path.join("output", run_name)
               for run_name in ("run_0", "run_1", "run_2")]

eval_results_from_disk = tfma.load_eval_results(
    output_dirs, tfma.constants.MODEL_CENTRIC_MODE)


tfma.view.render_time_series(
    eval_results_from_disk,
    slices[0])
```

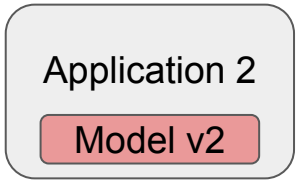
Lab 6

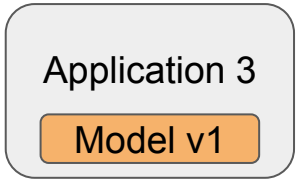
TensorFlow Model Analysis (TFMA)

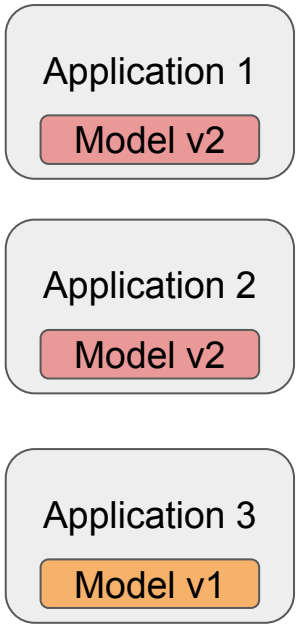
TensorFlow Serving

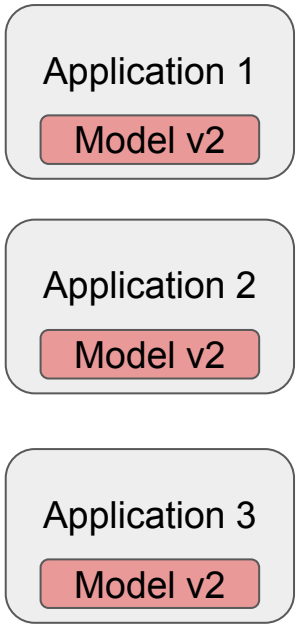

Application 1
Model v1
Application 2
Model v1
Application 3
Model v1

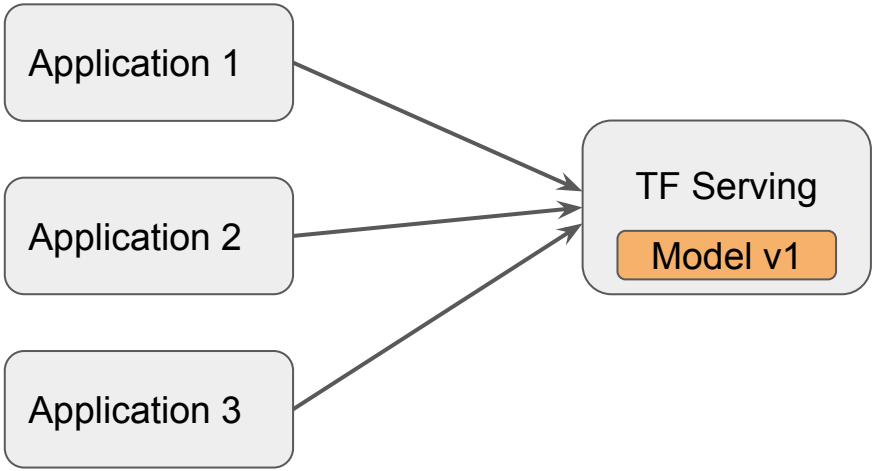

Application 1
Model v1

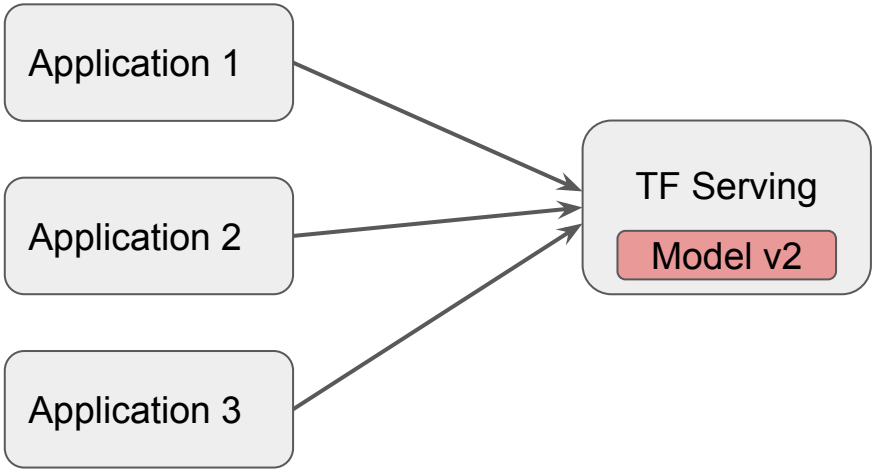

Application 2
Model v2

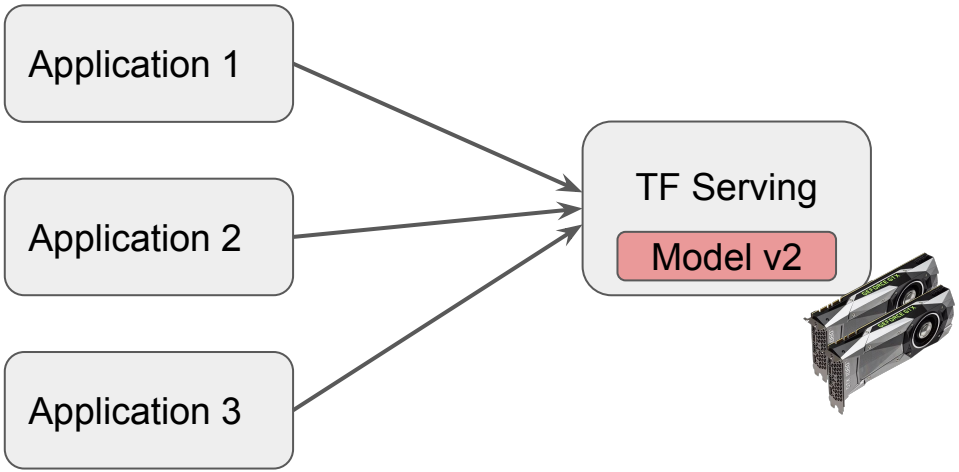

Application 3
Model v1


Application 1
Model v2
Application 2
Model v2
Application 3
Model v1


Application 1
Model v2
Application 2
Model v2
Application 3
Model v2


Application 1
Application 2
Application 3
TF Serving
Model v1


Application 1
Application 2
Application 3
TF Serving
Model v2


Application 1
Application 2
Application 3
TF Serving
Model v2

TensorFlow

Fairness

Lab 7

Fairness

```
from tfx.types import ComponentSpec
from tfx.types.component_spec import ChannelParameter
from tfx.types.component_spec import ExecutionParameter
from tfx.types.standard_artifacts import Examples

class DataAugmentationComponentSpec(ComponentSpec):
  PARAMETERS = {
      'max_rotation_angle': ExecutionParameter(type=float)
  }
  INPUTS = {
      'input_data': ChannelParameter(type=Examples)
  }
  OUTPUTS = {
      'augmented_data': ChannelParameter(type=Examples)
  }
```

```
from tfx.components.base.base_executor import BaseExecutor
from tfx.types.artifact_utils import get_split_uri


class DataAugmentationExecutor(BaseExecutor):
  def Do(self, input_dict, output_dict, exec_properties):
      input_examples_uri = get_split_uri(
          input_dict['input_data'], 'train')
      output_examples_uri = get_split_uri(
          output_dict['augmented_data'], 'train')
      max_rotation_angle = exec_properties['max_rotation_angle']
      [...]
```

```
[...]

decoder = tfdv.TFExampleDecoder()
with beam.Pipeline() as pipeline:
    _ = (pipeline
         | 'ReadTrainData' >> beam.io.ReadFromTFRecord(input_examples_uri)
         | 'ParseExample' >> beam.Map(decoder.decode)
         | 'Augmentation' >> beam.ParDo(_augment_image, **exec_properties)
         | 'DictToExample' >> beam.Map(_dict_to_example)
         | 'SerializeExample' >> beam.Map(lambda x: x.SerializeToString())
         | 'WriteAugmentedData' >> beam.io.WriteToTFRecord(
           os.path.join(output_examples_uri, "data_tfrecord"),
           file_name_suffix='.gz'))

[...]
```

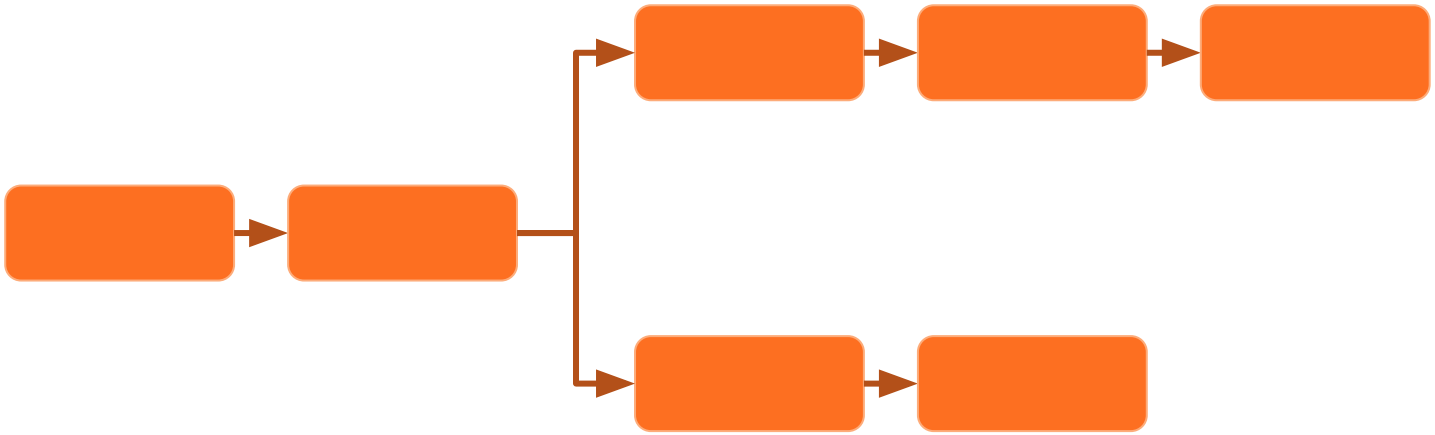
```
from tfx.components.base.base_component import BaseComponent
from tfx.components.base.executor_spec import ExecutorClassSpec


class DataAugmentationComponent(BaseComponent):
  SPEC_CLASS = DataAugmentationComponentSpec
  EXECUTOR_SPEC = ExecutorClassSpec(DataAugmentationExecutor)

  def __init__(self, input_data, max_rotation_angle=10.,
               augmented_data=None, instance_name=None):
    augmented_data = [...]
    spec = DataAugmentationComponentSpec(
        input_data=input_data,
        max_rotation_angle=max_rotation_angle,
        augmented_data=augmented_data)
    super().__init__(spec=spec, instance_name=instance_name)
```

```
augmented_data = augmented_data or tfx.types.Channel(
    type=Examples,
    artifacts=[Examples(split="train"), Examples(split="eval")])
```

```
class MyCustomArtifact(tfx.types.artifact.Artifact):
  TYPE_NAME = 'MyCustomArtifactPath'
```

Lab 8

Custom TFX Components



```
transform = Transform(...)

trainer1 = Trainer(
    trainer_fn='trainer.trainer_fn1',
    transformed_examples=transform.outputs.transformed_examples,
    [...])


trainer2 = Trainer(
    trainer_fn='trainer.trainer_fn2',
    transformed_examples=transform.outputs.transformed_examples,
    [...])
```

```
transform = Transform(...)

trainer1 = Trainer(
    trainer_fn='trainer.trainer_fn1',
    transformed_examples=transform.outputs.transformed_examples,
    [...])


trainer2 = Trainer(
    trainer_fn='trainer.trainer_fn2',
    transformed_examples=transform.outputs.transformed_examples,
    [...])
```

```
transform = Transform(...)

trainer1 = Trainer(
    trainer_fn='trainer.trainer_fn1',
    transformed_examples=transform.outputs.transformed_examples,
    instance_name='Trainer1',
    [...])

trainer2 = Trainer(
    trainer_fn='trainer.trainer_fn2',
    transformed_examples=transform.outputs.transformed_examples,
    instance_name='Trainer2',
    [...])
```

```
transform = Transform(...)

trainer1 = Trainer(
    trainer_fn='trainer.trainer_fn1',
    transformed_examples=transform.outputs.transformed_examples,
    instance_name='Trainer1',
    [...])

trainer2 = Trainer(
    trainer_fn='trainer.trainer_fn2',
    transformed_examples=transform.outputs.transformed_examples,
    instance_name='Trainer2',
    [...])

components = […, transform, trainer1, trainer2,…]
pipeline = Pipeline(components=components, …)
```

Lab 9

Alternate Pipeline Architectures

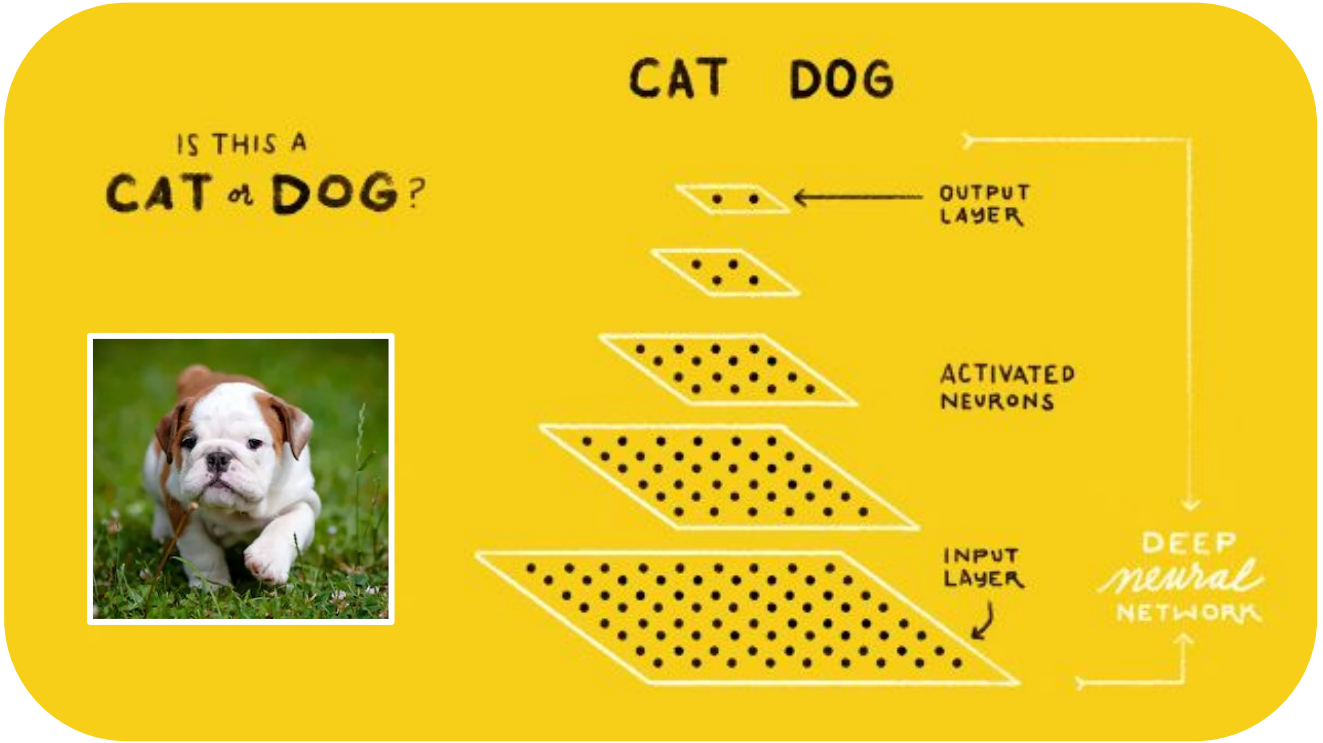
O'REILLY®
TensorFlow World
PRESENTED WITH TensorFlow

Neural Structured Learning

Training Neural Networks with Structured Signals

Arjun Gopalan
Software Engineer

How a Typical Neural Net Works



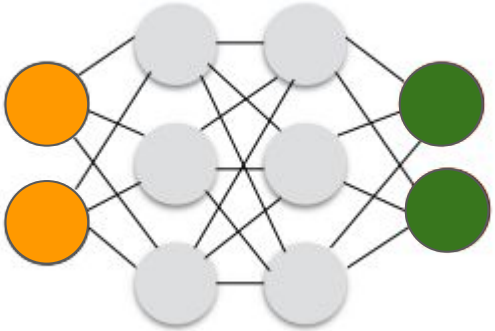
How a Typical Neural Net Works

Labeled data

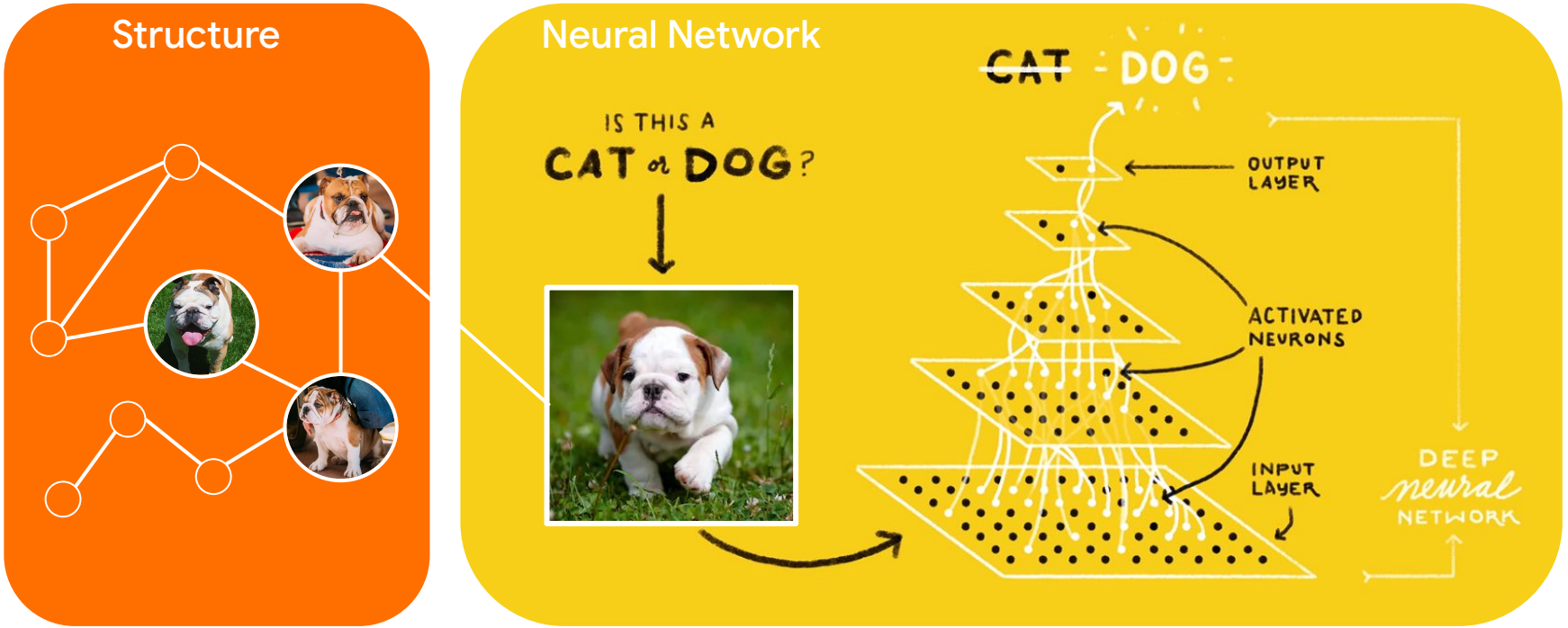
Input	Label
Lots of labeled examples	Cat
	Dog
	...
	Dog
...	...

Train

Neural Network



Neural Structured Learning (NSL)



Neural Structured Learning (NSL)

Concept: train neural net using structure among samples

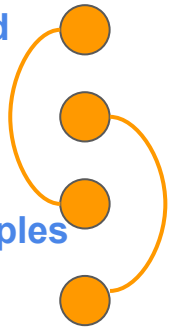
Labeled data

Input

Few labeled examples

\+

Relations between examples



...

Label

Cat

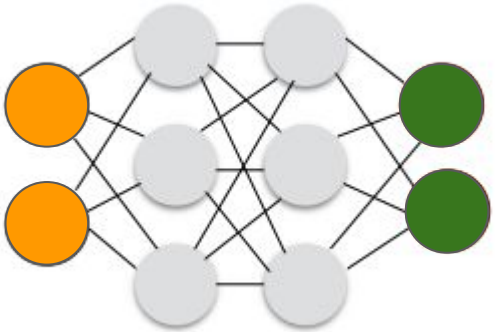
Dog

?

?

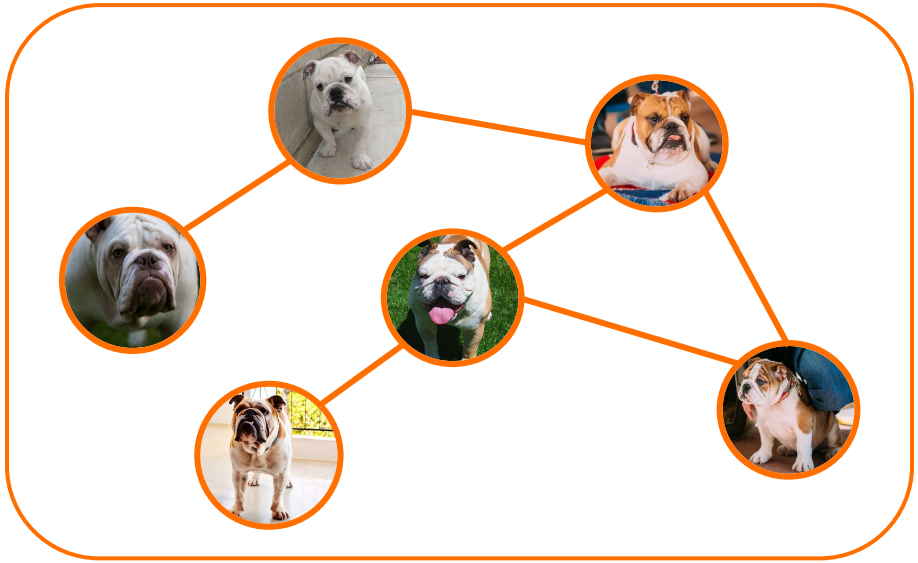
Train

Neural Network



Structure Among Samples

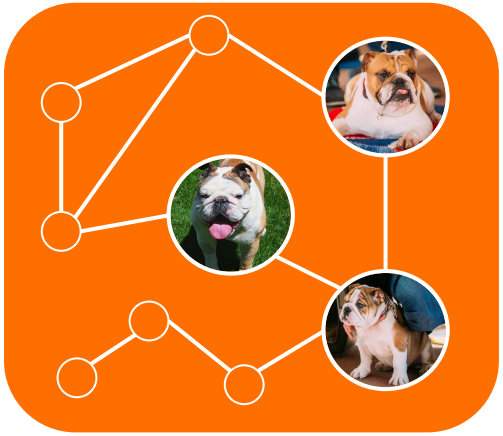
e.g., similar images



[Source: graph concept is from Juan et al., arXiv'19. Original images are from pixabay.com]

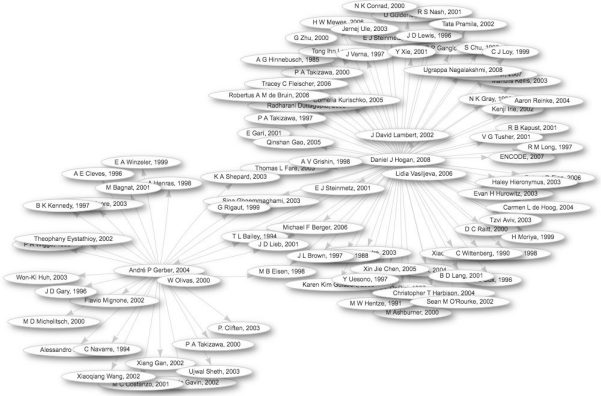
Structure Among Samples

Co-Occurrence Graph



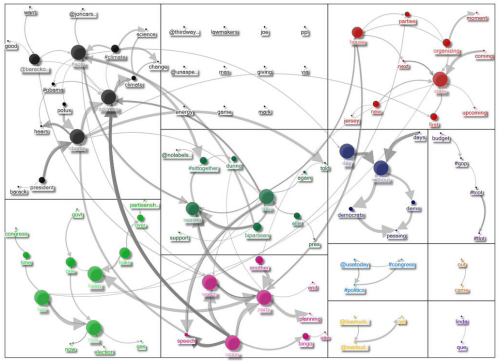
[Source: graph concept is from Juan et al., WSDM'20. Original images are from pixabay.com]

Citation Graph



[Source: https://commons.wikimedia.org/wiki/File:Partial_citation_graph_for_%22A_screen_for_RNA-binding_proteins_in_yeast_indicates_dual_functions_for_many_enzymes%22_as_of_April_12,_2017.png]

Text Graph



[Source: copied without modification from https://www.flickr.com/photos/marc_smith/6705382867/sizes/l/]

NSL: Advantages of Learning with Structure

- Less Labeled Data Required (Neural Graph Learning)
- Robust Model (Adversarial Learning)

Scenario I: Not Enough Labeled Data

Example task:

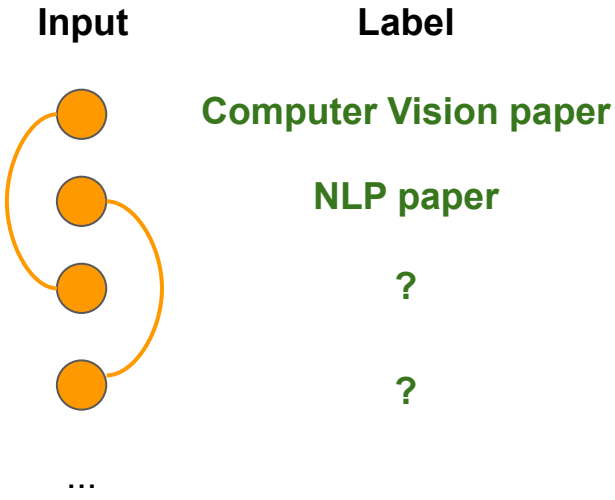
Document Classification

Lots of samples

Not enough labels

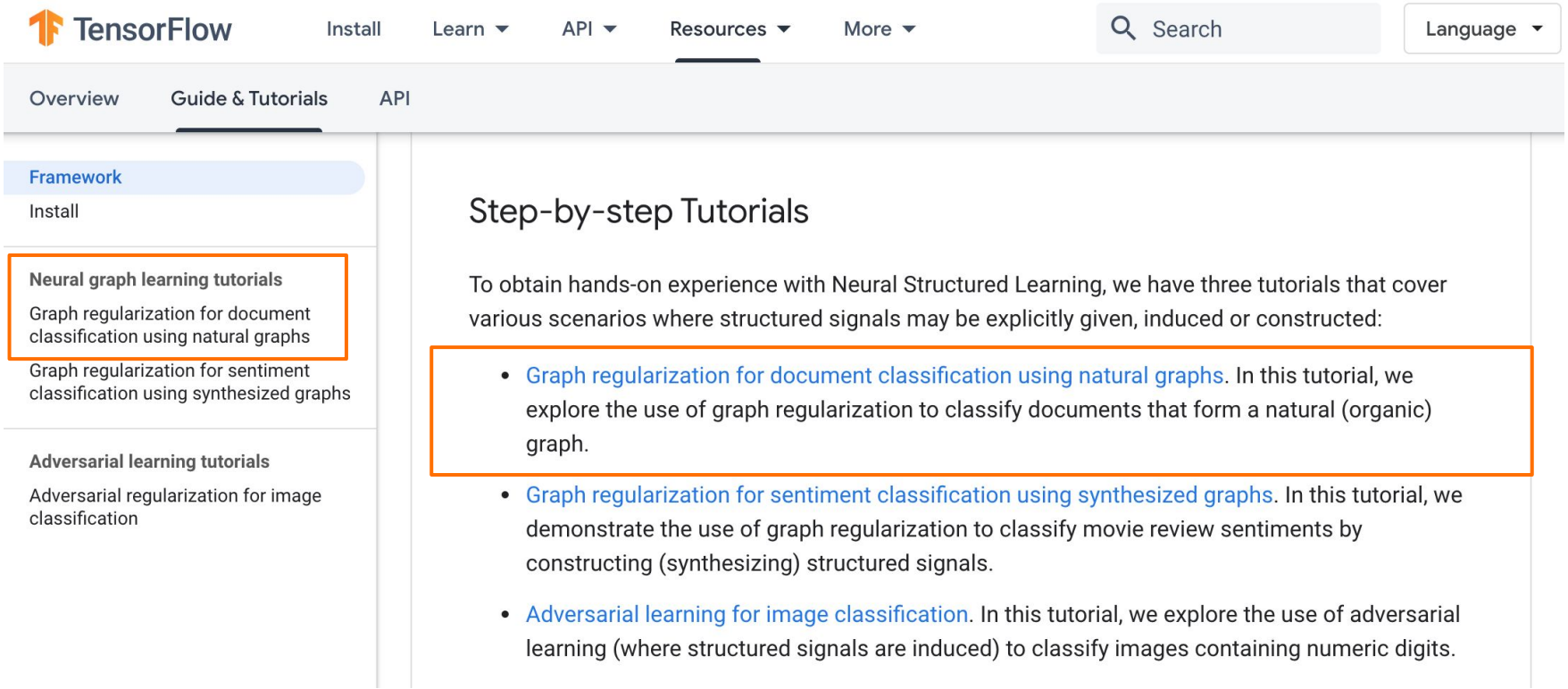
NSL: Advantages of Learning with Structure

➡ **Less Labeled Data Required**



Use relations between examples + few labeled examples

NSL Resource: Tutorials



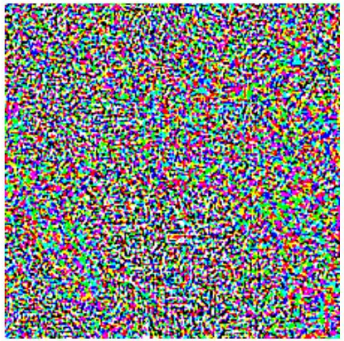
Scenario II: Model Robustness Required

Example task: **Image Classification**

$\boldsymbol{x}$

"panda"

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"

$=$

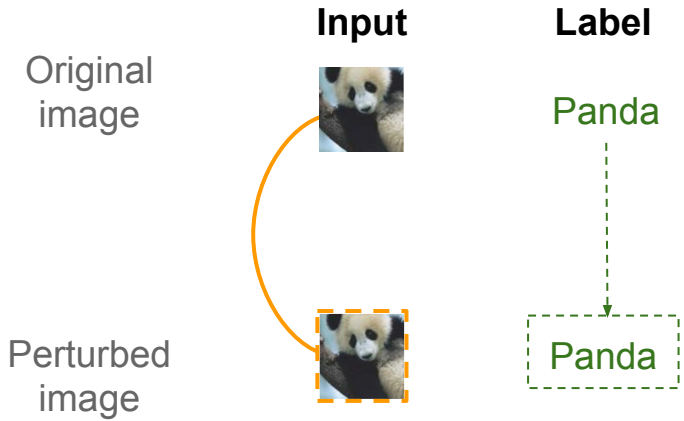
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"

[Source: Goodfellow, et al., ICLR'15]

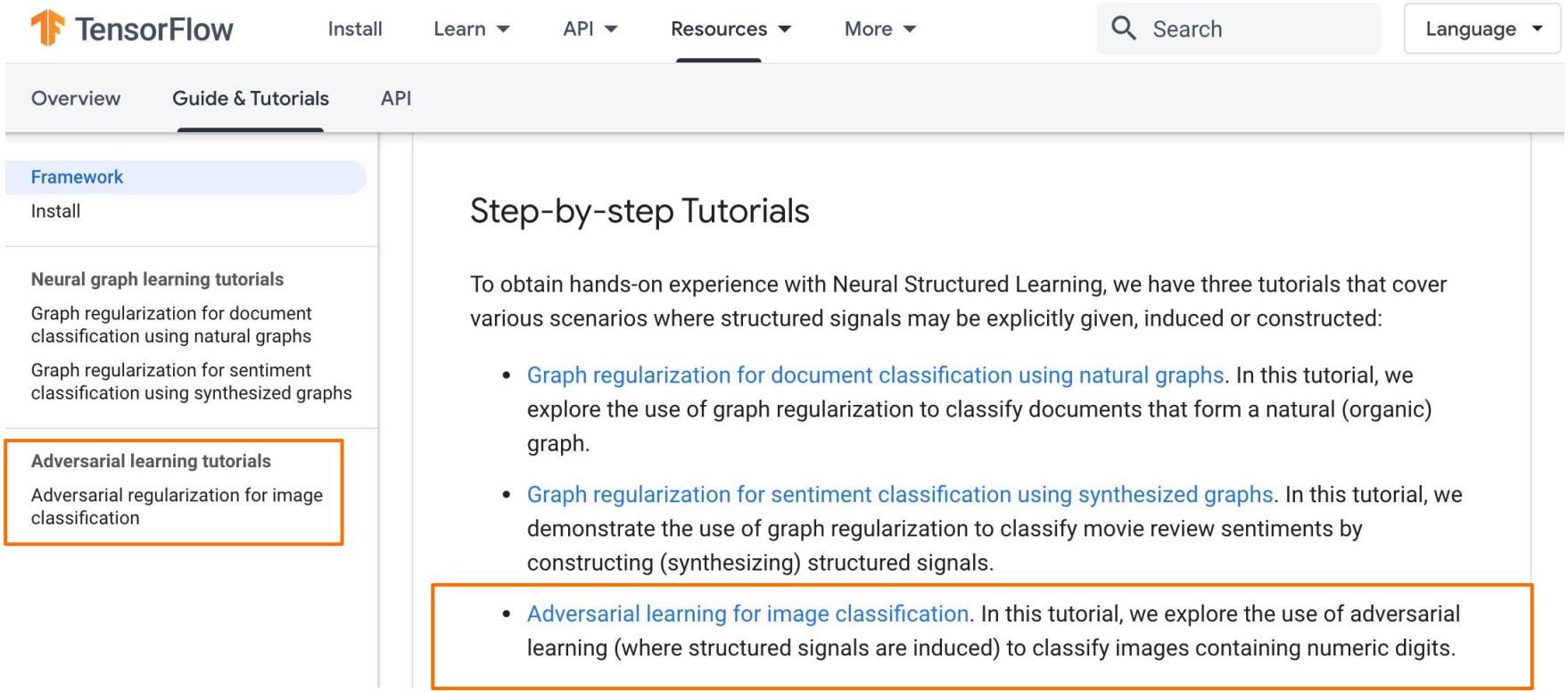
NSL: Advantages of Learning with Structure

➡ **Robust Model**



Use implicit structure derived from "adversarial" examples

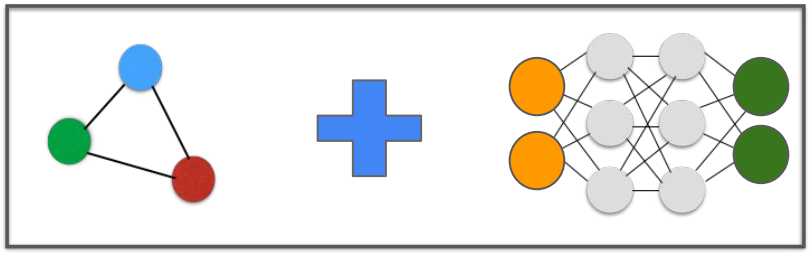
NSL Resource: Tutorials



NSL Framework

NSL: Neural Graph Learning

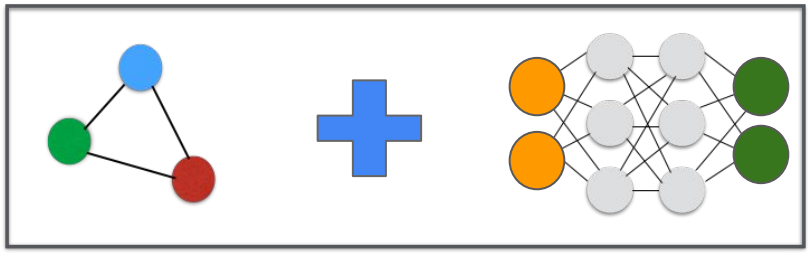
Graph + Neural Net



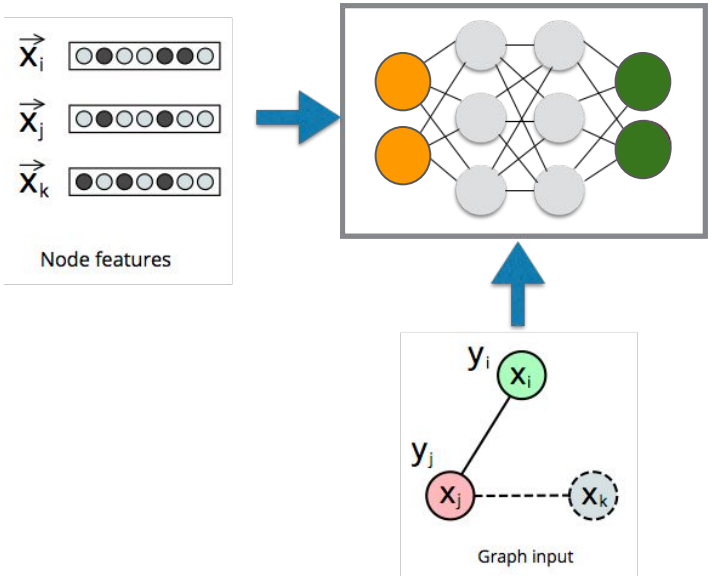
- **Jointly** optimizes both features & structured signals for better models

NSL: Neural Graph Learning

Graph + Neural Net



- **Jointly** optimizes both features & structured signals for better models

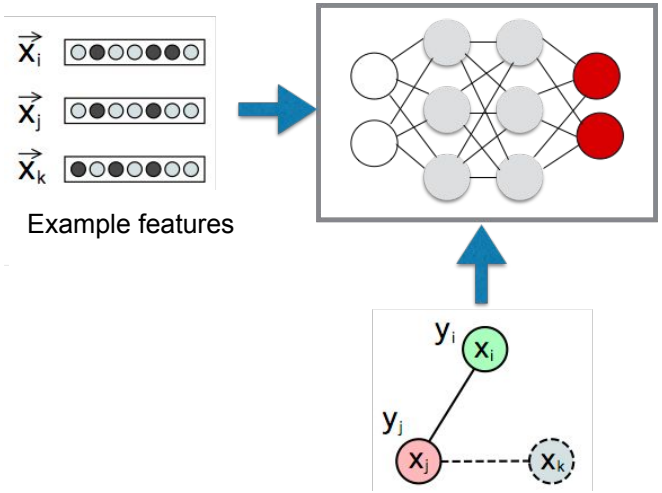


Neural Graph Machines (NGM)

Paper: Bui, Ravi & Ramavajjala [WSDM'18]

NSL: Neural Graph Learning

Joint optimization with label and structured signals:



Optimize: $loss = \sum_{i=1}^{B} \mathcal{L}(y_i, \hat{y}_i) + \alpha \sum_{i=1}^{B} \mathcal{L}_{\mathcal{N}}(y_i, x_i, \mathcal{N}(x_i))$

$x_i \rightarrow f(.) \rightarrow y_i$

Supervised Loss

$$\sum_{i=1}^{B} \mathcal{E}(y_i, g_\theta(x_i))$$

Neighbor Loss

$$\sum_{x_j \in \mathcal{N}(x_i)} w_{ij} \cdot \mathcal{D}(h_\theta(x_i), h_\theta(x_j))$$

$g_\theta(x_i)$: NN output for input x_i

$\mathcal{E}(\cdot)$: Loss function

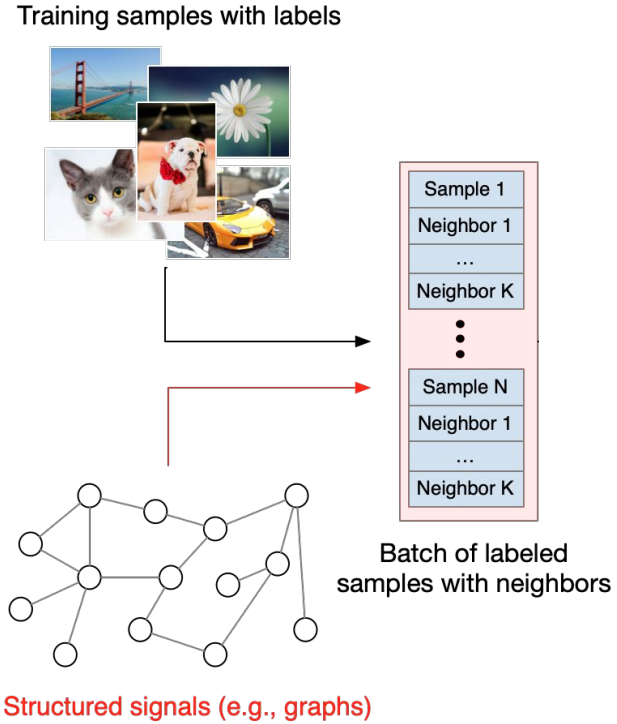
Examples: L2 (for regression)
Cross-Entropy (for classification)

$h_\theta(\cdot)$: Target hidden layer

$\mathcal{D}(\cdot)$: Distance metric

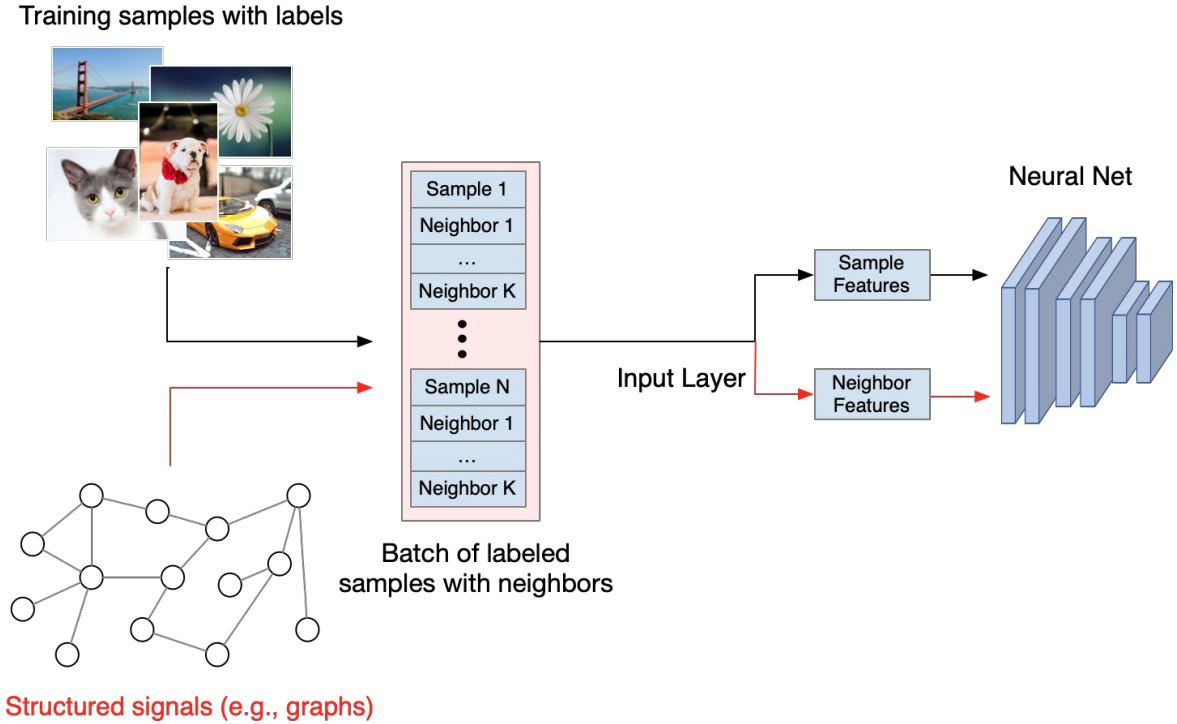
Examples: L1, L2, ...

NSL: Neural Graph Learning Training Workflow

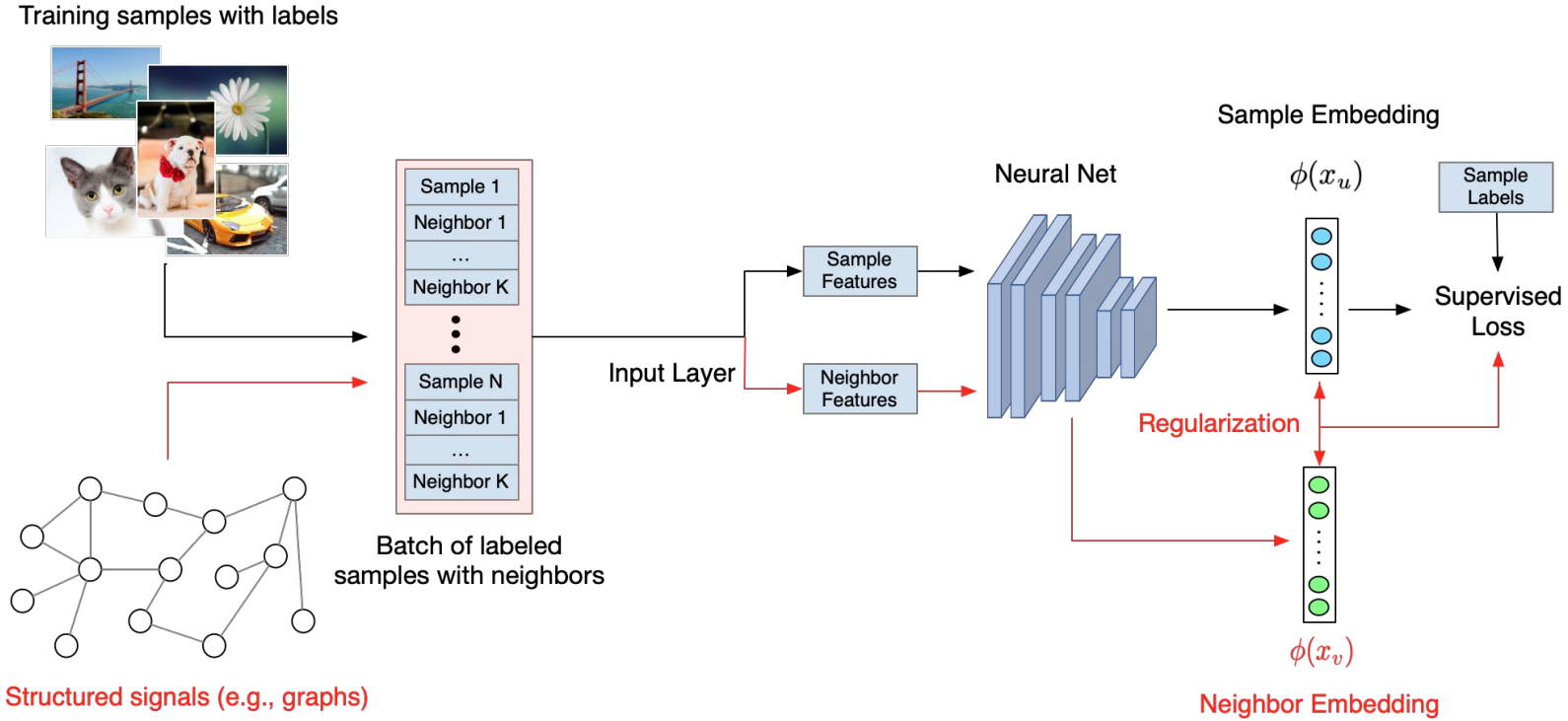


[Source: Juan, et al., WSDM'20]

NSL: Neural Graph Learning Training Workflow



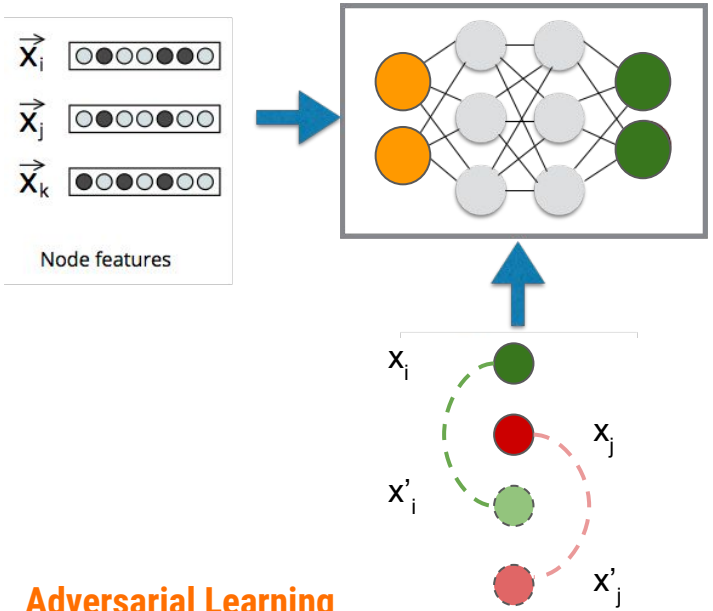
NSL: Neural Graph Learning Training Workflow



NSL: Adversarial Learning

Adversarial + Neural Net

- **Jointly** optimize features from original and “adversarial” examples for more robust models



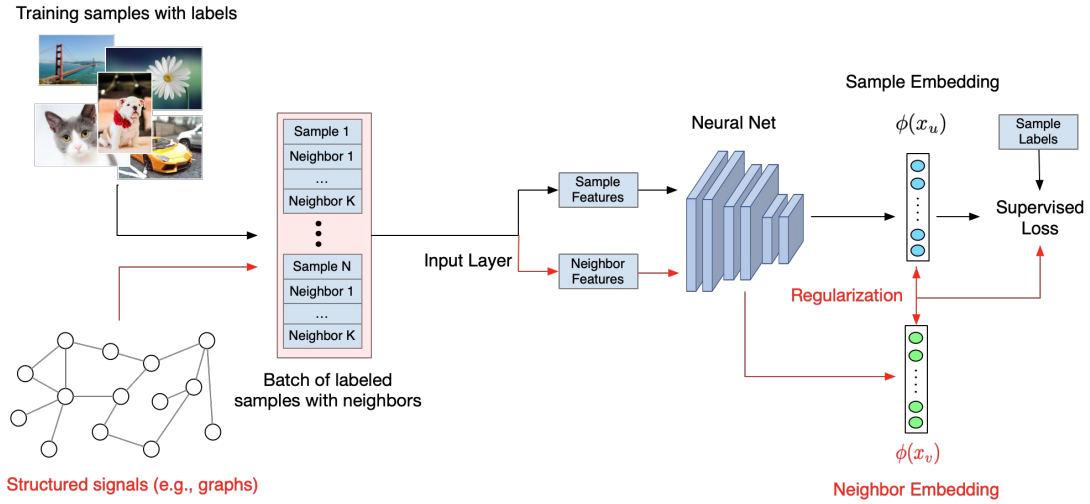
Adversarial Learning

Paper: Goodfellow, et al. [ICLR’15]

Libraries
Tools
Trainers

Libraries, Tools, and Trainers

Red segments represent NSL additions



Libraries, Tools, and Trainers

Standalone Tool

`build_graph`

`pack_nbrs`

Graph Functions

`build_graph`

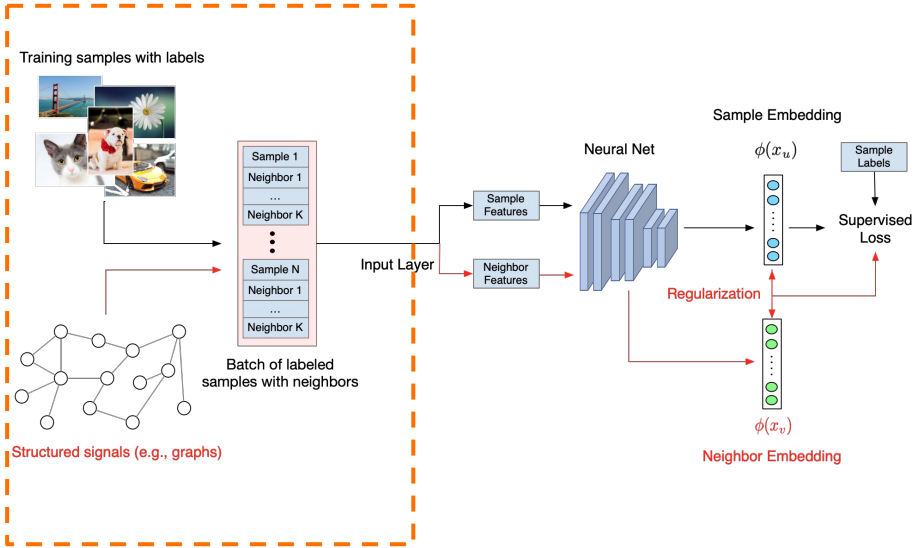
`pack_nbrs`

`read_tsv_graph`

`write_tsv_graph`

`add_edge`

`add_undirected_edges`



Libraries, Tools, and Trainers

Standalone Tool

`build_graph`

`pack_nbrs`

Graph Functions

`build_graph`

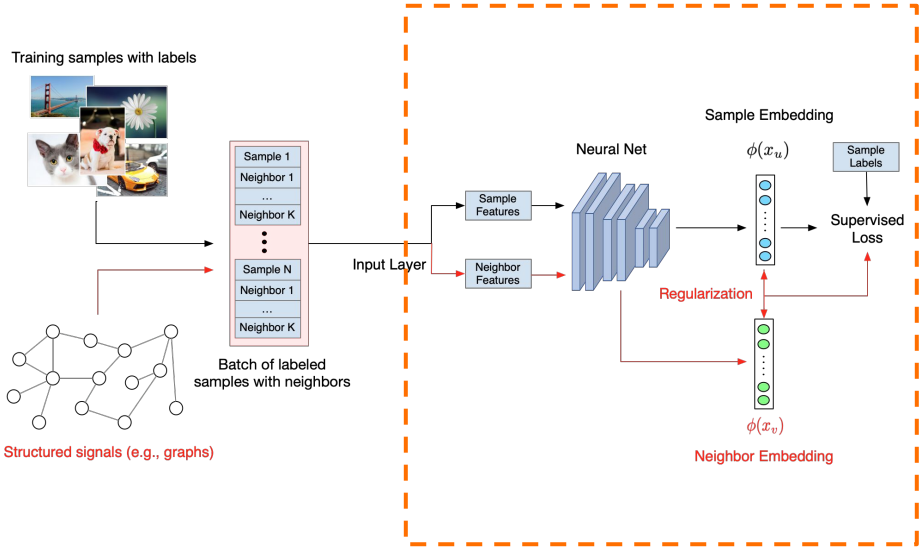
`pack_nbrs`

`read_tsv_graph`

`write_tsv_graph`

`add_edge`

`add_undirected_edges`



Lib

`unpack_neighbor_features`

`adversarial_neighbor`

`replicate_embeddings`

`utils`

Libraries, Tools, and Trainers

Standalone Tool

`build_graph`

`pack_nbrs`

Graph Functions

`build_graph`

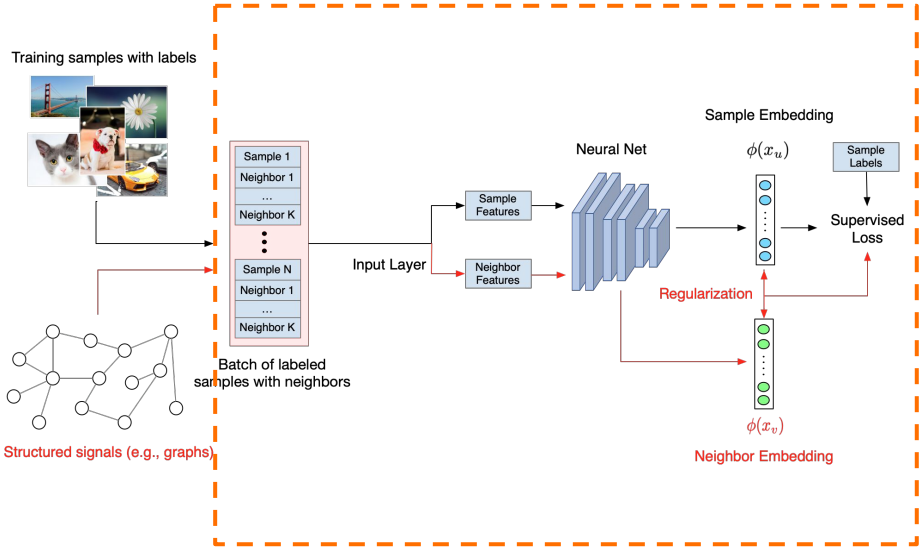
`pack_nbrs`

`read_tsv_graph`

`write_tsv_graph`

`add_edge`

`add_undirected_edges`



Lib

`unpack_neighbor_features`

`adversarial_neighbor`

`replicate_embeddings`

`utils`

Keras

`graph_regularization`

`adversarial_regularization`

`Layers`

Estimator

`add_graph_regularization`

`add_adversarial_regularization`

Libraries, Tools, and Trainers

Standalone Tool

`build_graph`

`pack_nbrs`

Graph Functions

`build_graph`

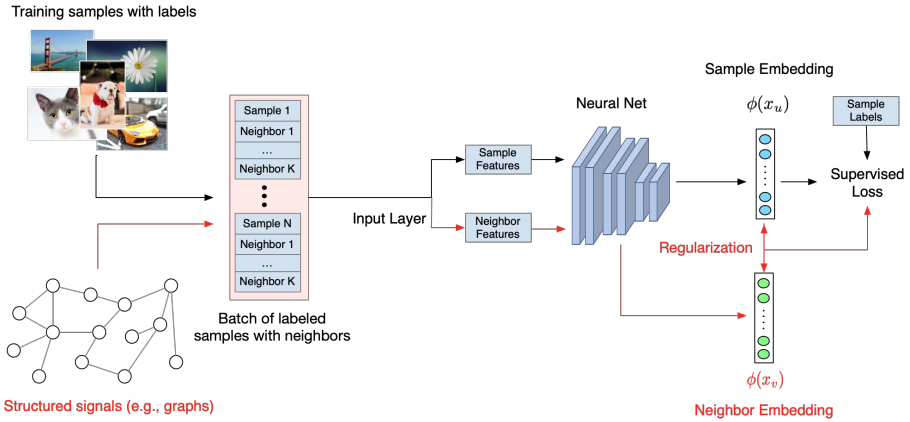
`pack_nbrs`

`read_tsv_graph`

`write_tsv_graph`

`add_edge`

`add_undirected_edges`



Lib

`unpack_neighbor_features`

`adversarial_neighbor`

`replicate_embeddings`

`utils`

Keras

`graph_regularization`

`adversarial_regularization`

`Layers`

Estimator

`add_graph_regularization`

`add_adversarial_regularization`

Web: tensorflow.org/neural_structured_learning

```
pip install neural-structured-learning
```

```
import neural_structured_learning as nsl

# Extract features required for the model from the input.
train_dataset, test_dataset = make_datasets('/tmp/train.tfr',
                                            '/tmp/test.tfr')

# Create a base model -- sequential, functional, or subclass.
base_model = tf.keras.Sequential(...)
```

Read Data

Keras Model

```python
import neural_structured_learning as nsl

# Extract features required for the model from the input.
train_dataset, test_dataset = make_datasets('/tmp/train.tfr',
                                            '/tmp/test.tfr')

# Create a base model -- sequential, functional, or subclass.
base_model = tf.keras.Sequential(...)

# Wrap the base model with graph regularization.
graph_config = nsl.configs.GraphRegConfig(
    neighbor_config=nsl.configs.GraphNeighborConfig(max_neighbors=1))
graph_model = nsl.keras.GraphRegularization(base_model, graph_config)
```

Read Data

Keras Model

Config
Graph Model

```python
import neural_structured_learning as nsl

# Extract features required for the model from the input.
train_dataset, test_dataset = make_datasets('/tmp/train.tfr',
                                            '/tmp/test.tfr')

# Create a base model -- sequential, functional, or subclass.
base_model = tf.keras.Sequential(...)

# Wrap the base model with graph regularization.
graph_config = nsl.configs.GraphRegConfig(
    neighbor_config=nsl.configs.GraphNeighborConfig(max_neighbors=1))
graph_model = nsl.keras.GraphRegularization(base_model, graph_config)

# Compile, train, and evaluate.
graph_model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy'])
graph_model.fit(train_dataset, epochs=5)
graph_model.evaluate(test_dataset)
```

Read Data

Keras Model

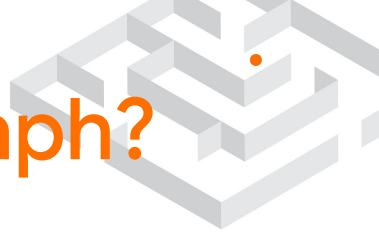
Config
Graph Model

Compile

Fit

Eval

What If No Explicit Structure or Graph?



What If No Explicit Structure or Graph?

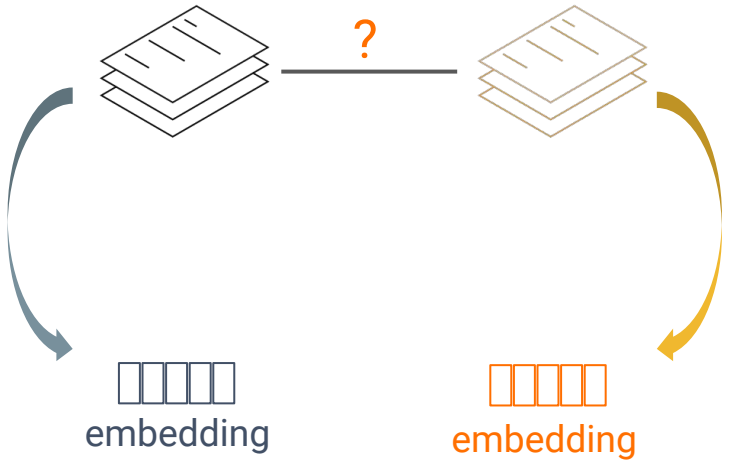
Construct graph via Preprocessing

- Find neighbors using embeddings

What If No Explicit Structure or Graph?

Construct graph via Preprocessing

- Find neighbors using embeddings

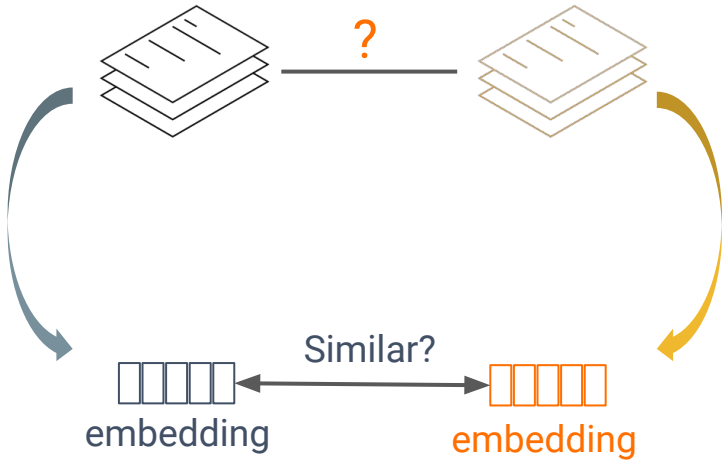


Project documents into embeddings

What If No Explicit Structure or Graph?

Construct graph via Preprocessing

- Find neighbors using embeddings



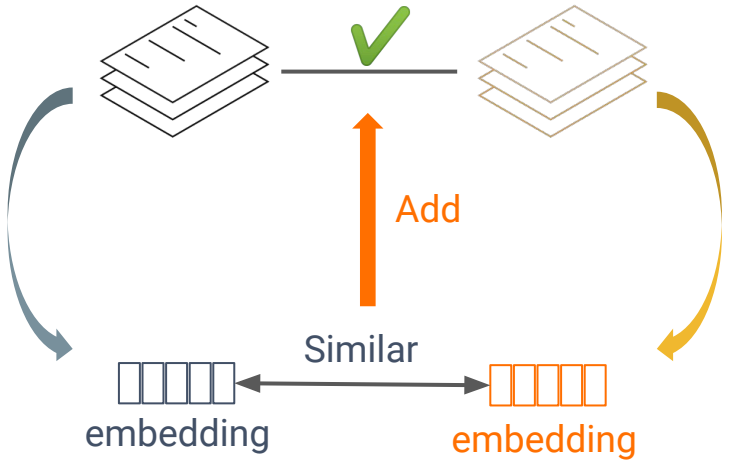
Project documents into embeddings

See if similarity above the threshold

What If No Explicit Structure or Graph?

Construct graph via Preprocessing

- Find neighbors using embeddings

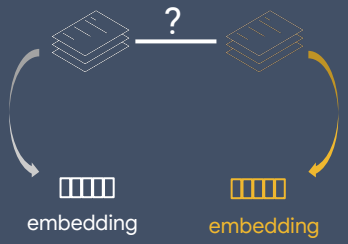


Project documents into embeddings

See if similarity above the threshold

If yes, add an edge making them neighbors

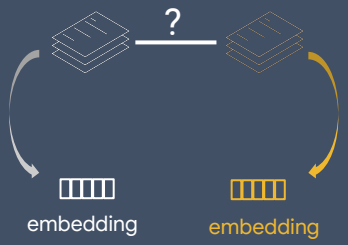
```
"""Generate embeddings."""

import neural_structured_learning as nsl
import tensorflow as tf
import tensorflow_hub as hub

imdb = tf.keras.datasets.imdb
(pp_train_data, pp_train_labels), (pp_test_data, pp_test_labels) = (
    imdb.load_data(num_words=10000))

pretrained_embedding =
'https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1'
hub_layer = hub.KerasLayer(
    pretrained_embedding, input_shape=[], dtype=tf.string,
trainable=True)
```



Load Data

Pre-trained Embedding

```
"""Generate embeddings."""

import neural_structured_learning as nsl
import tensorflow as tf
import tensorflow_hub as hub

imdb = tf.keras.datasets.imdb
(pp_train_data, pp_train_labels), (pp_test_data, pp_test_labels) = (
    imdb.load_data(num_words=10000))

pretrained_embedding =
'https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1'
hub_layer = hub.KerasLayer(
    pretrained_embedding, input_shape=[], dtype=tf.string,
trainable=True)

# Generate embeddings.
record_id = int(0)
with tf.io.TFRecordWriter('/tmp/imdb/embeddings.tfr') as writer:
  for word_vector in pp_train_data:
    text = decode_review(word_vector)
    sentence_embedding = hub_layer(tf.reshape(text, shape=[-1,]))
    sentence_embedding = tf.reshape(sentence_embedding, shape=[-1])
    write_embedding_example(sentence_embedding, record_id)
    record_id += 1
```

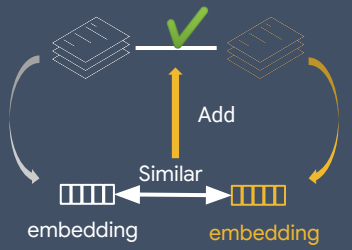


Load Data

Pre-trained Embedding

Text to Embedding Lookup

```
"""Build graph and prepare graph input for NSL."""
```



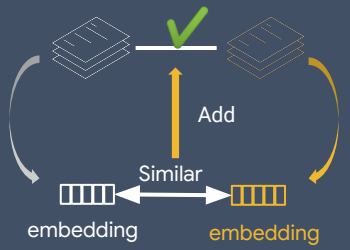

```
# Build a graph from embeddings.
nsl.tools.build_graph(['/tmp/imdb/embeddings.tfr'],
                      '/tmp/imdb/graph_99.tsv',
                      similarity_threshold=0.8)
```

Graph Building

```
# Create example features.
next_record_id = create_examples(pp_train_data, pp_train_labels,
                                 '/tmp/imdb/train_data.tfr',
                                 starting_record_id=0)
create_examples(pp_test_data, pp_test_labels,
                '/tmp/imdb/test_data.tfr',
                starting_record_id=next_record_id)
```

Feature Definition

```
"""Build graph and prepare graph input for NSL."""
```




```
# Build a graph from embeddings.
nsl.tools.build_graph(['/tmp/imdb/embeddings.tfr'],
                      '/tmp/imdb/graph_99.tsv',
                      similarity_threshold=0.8)
```

Graph Building

```
# Create example features.
next_record_id = create_examples(pp_train_data, pp_train_labels,
                                 '/tmp/imdb/train_data.tfr',
                                 starting_record_id=0)
create_examples(pp_test_data, pp_test_labels,
                '/tmp/imdb/test_data.tfr',
                starting_record_id=next_record_id)
```

Feature Definition

```
# Augment training data my merging neighbors into sample features.
nsl.tools.pack_nbrs('/tmp/imdb/train_data.tfr', '',
                    '/tmp/imdb/graph_99.tsv',
                    '/tmp/imdb/nsl_train_data.tfr',
                    add_undirected_edges=True, max_nbrs=3)
```

Training Data Augmentation

```
"""Graph-regularized keras model."""


# Extract features required for the model from the input.
train_ds, test_ds = make_datasets('/tmp/imdb/nsl_train.tfr',
                                  '/tmp/imdb/test.tfr')

# Create a base model -- sequential, functional, or subclass.
base_model = tf.keras.Sequential(...)

# Wrap the base model with graph regularization.
graph_config = nsl.configs.GraphRegConfig(
    neighbor_config=nsl.configs.GraphNeighborConfig(max_neighbors=2))
graph_model = nsl.keras.GraphRegularization(base_model, graph_config)

# Compile, train, and evaluate.
graph_model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy'])
graph_model.fit(train_dataset, epochs=5)
graph_model.evaluate(test_dataset)
```

Read Data

Keras Model

Config
Graph Model

Compile

Fit

Eval

Recap

Training with structured signals is useful!

- Less labeled data required
- Robust model

Neural Structured Learning provides:

- APIs for building Keras and Estimator models
- TF libraries, tools, and tutorials for learning with structured signals
- Support for all kinds of neural nets: feedforward, convolutional, or recurrent

Thank You!

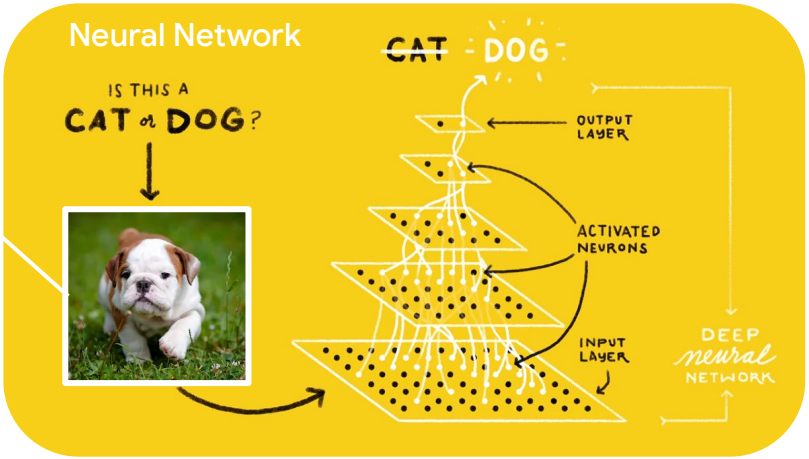
Web: tensorflow.org/neural_structured_learning

Repo: github.com/tensorflow/neural-structured-learning

Survey: cutt.ly/nsl2019

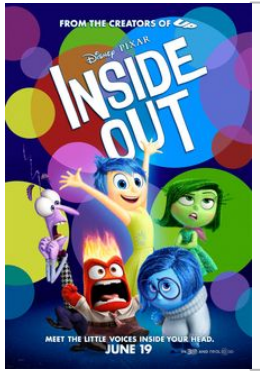
Special acknowledgment:
Google Expander team

Arjun Gopalan
arjung@google.com



Up Next: Hands-on TFX+NSL Tutorial

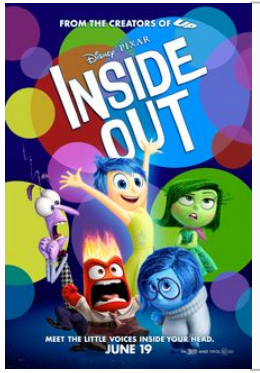
IMDB Reviews



An artistic triumph
21 September 2015

For some reason, I couldn't quite catch this movie in theaters and I managed to watch it on an international flight. And boy, am I glad I did!

IMDB Reviews



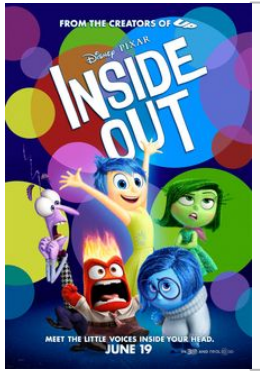
An artistic triumph
21 September 2015

For some reason, I couldn't quite catch this movie in theaters and I managed to watch it on an international flight. And boy, am I glad I did!

POSITIVE ?

IMDB Reviews

Label = True



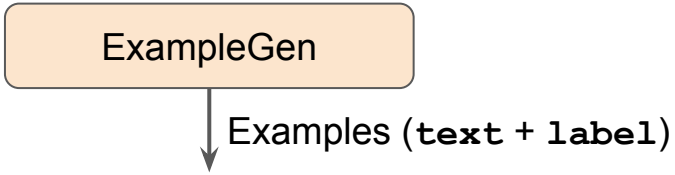
9/10

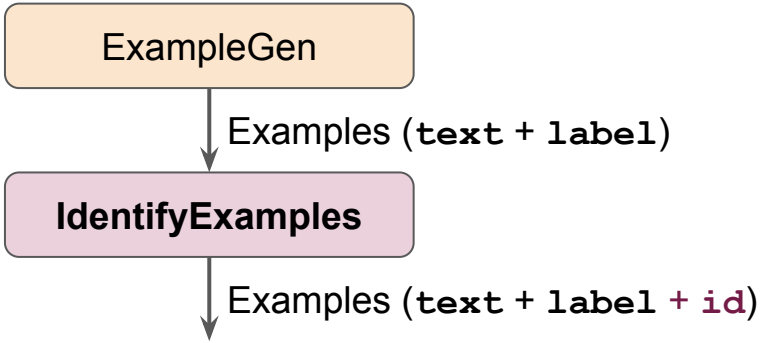
An artistic triumph

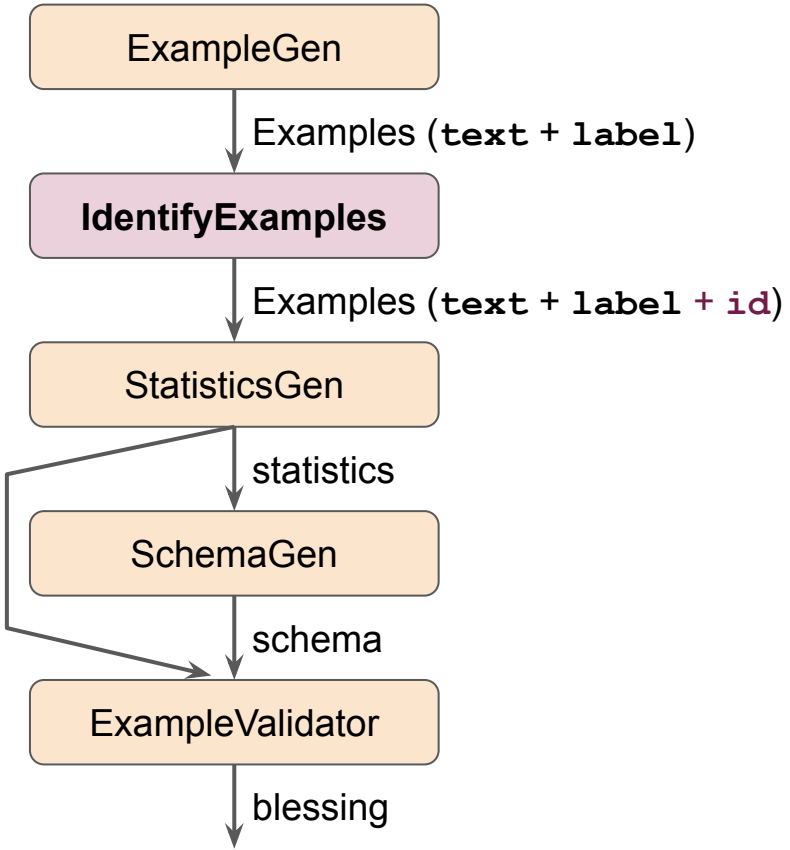
21 September 2015

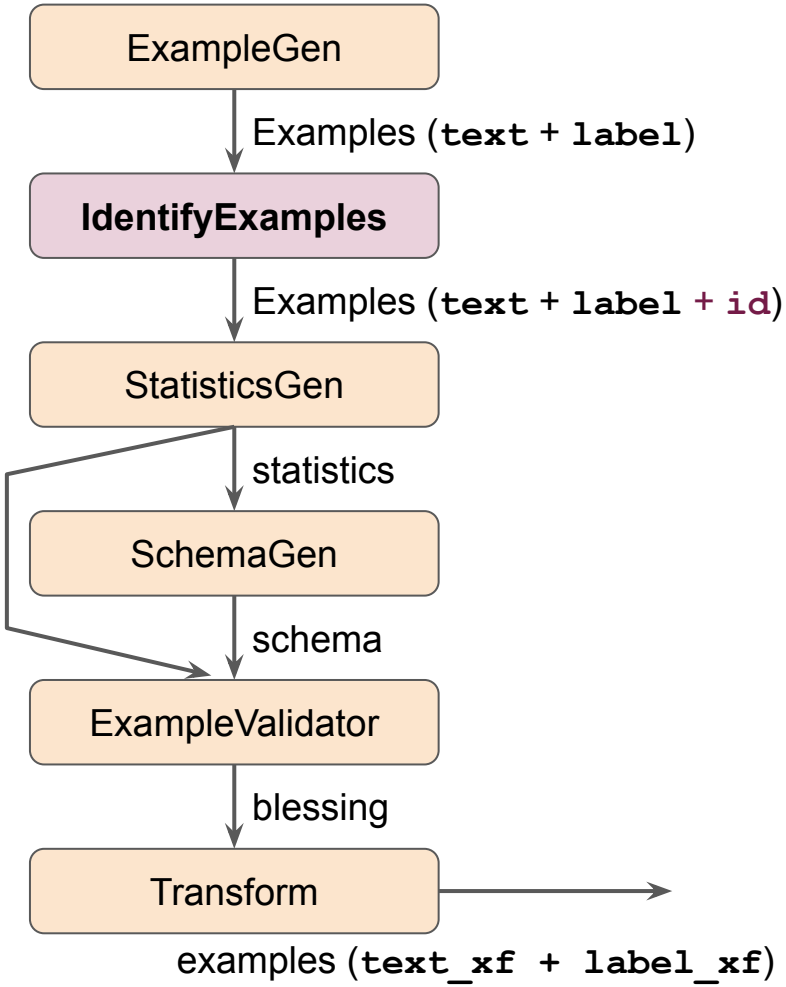
For some reason, I couldn't quite catch this movie in theaters and I managed to watch it on an international flight. And boy, am I glad I did!

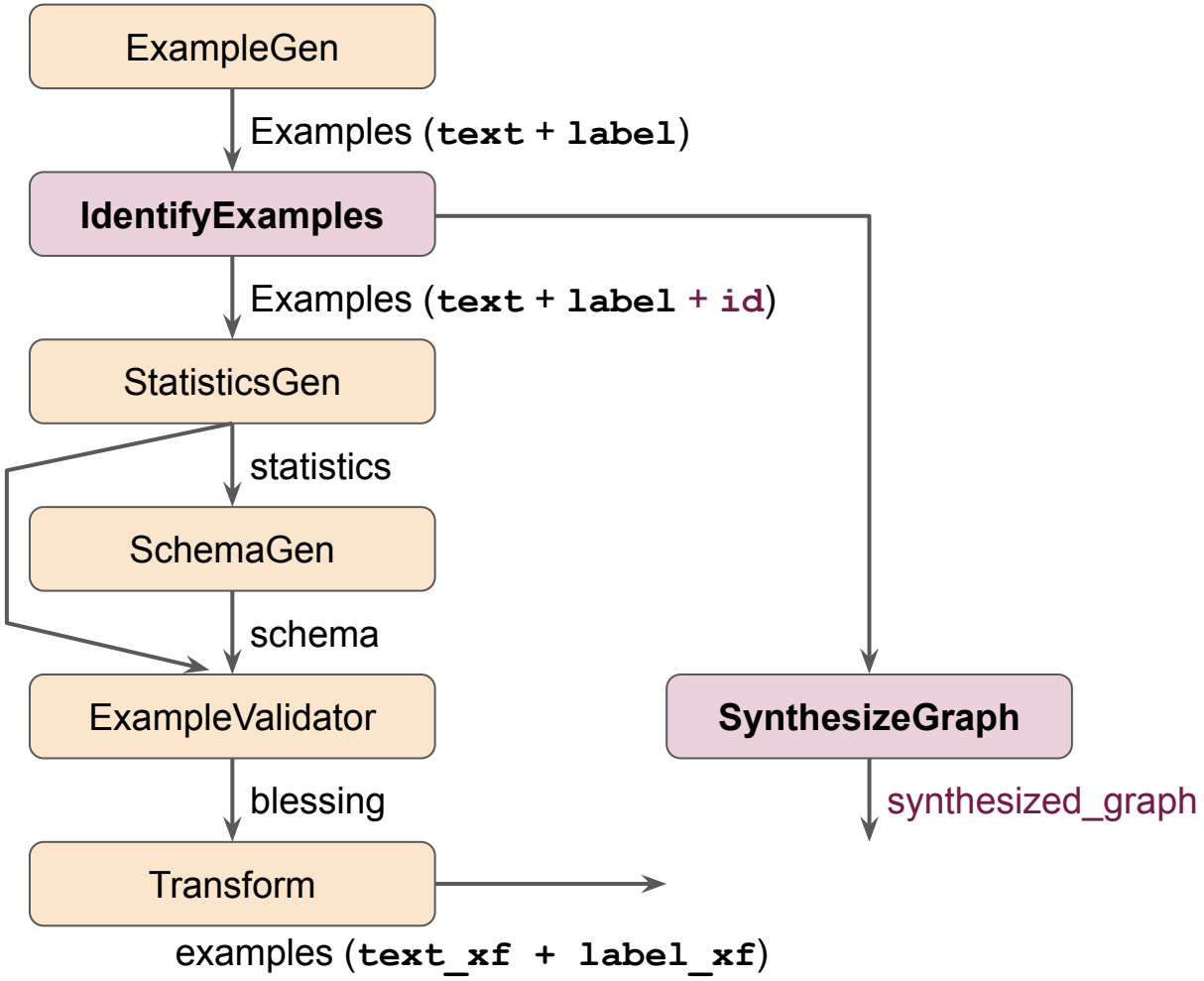
POSITIVE ?

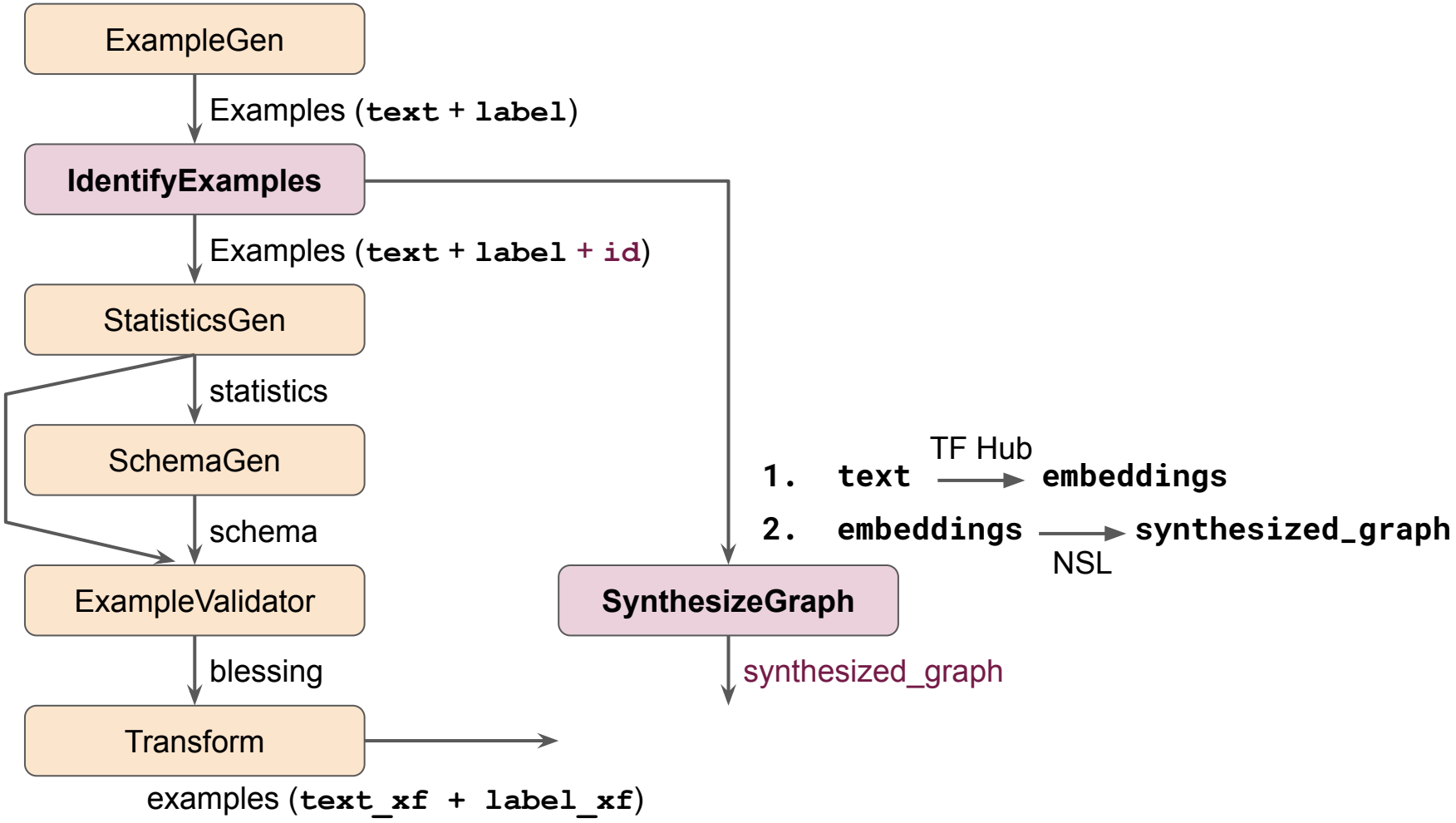

ExampleGen
Examples (text + label)

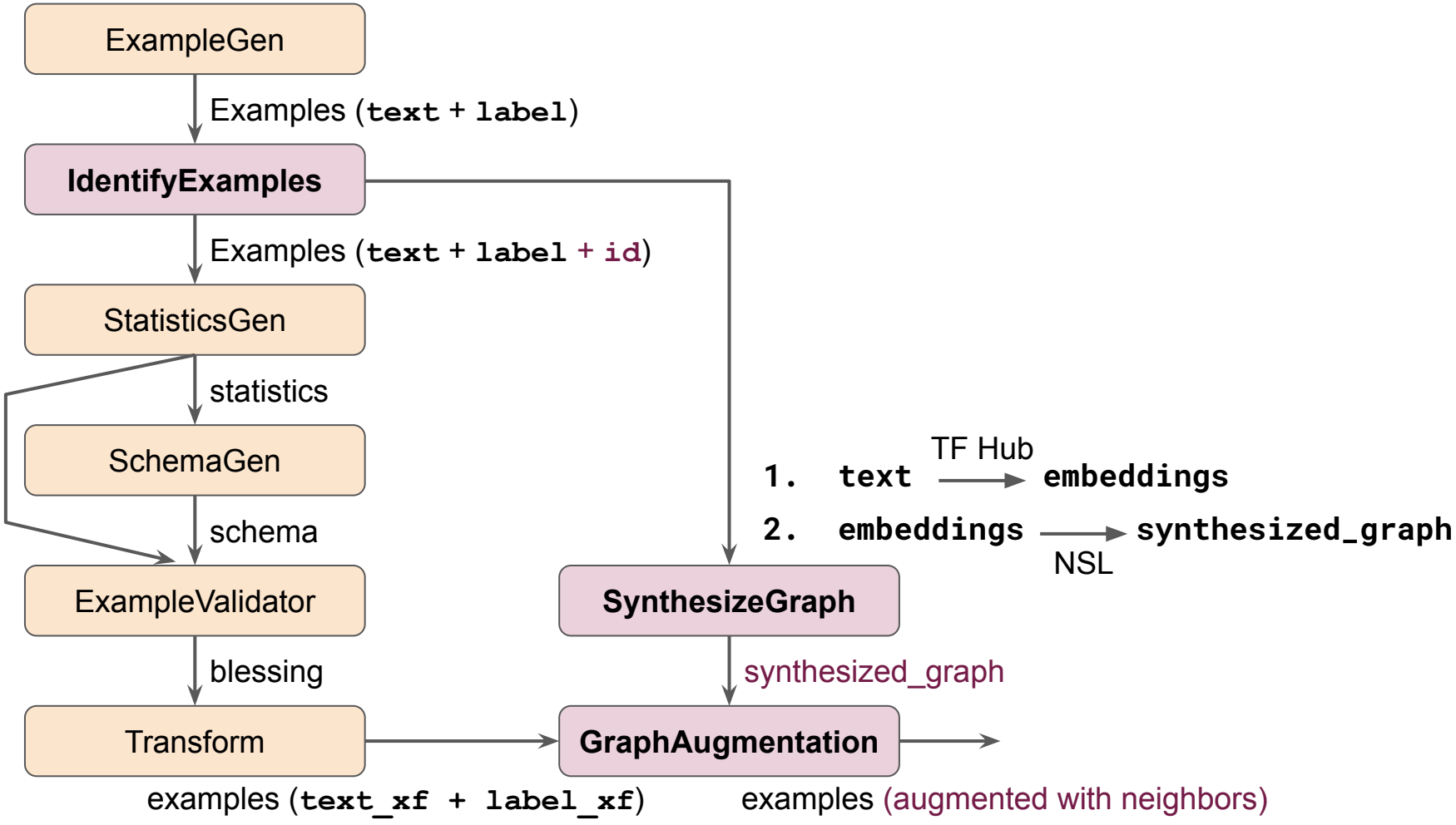

ExampleGen
Examples (text + label)
IdentifyExamples
Examples (text + label + id)

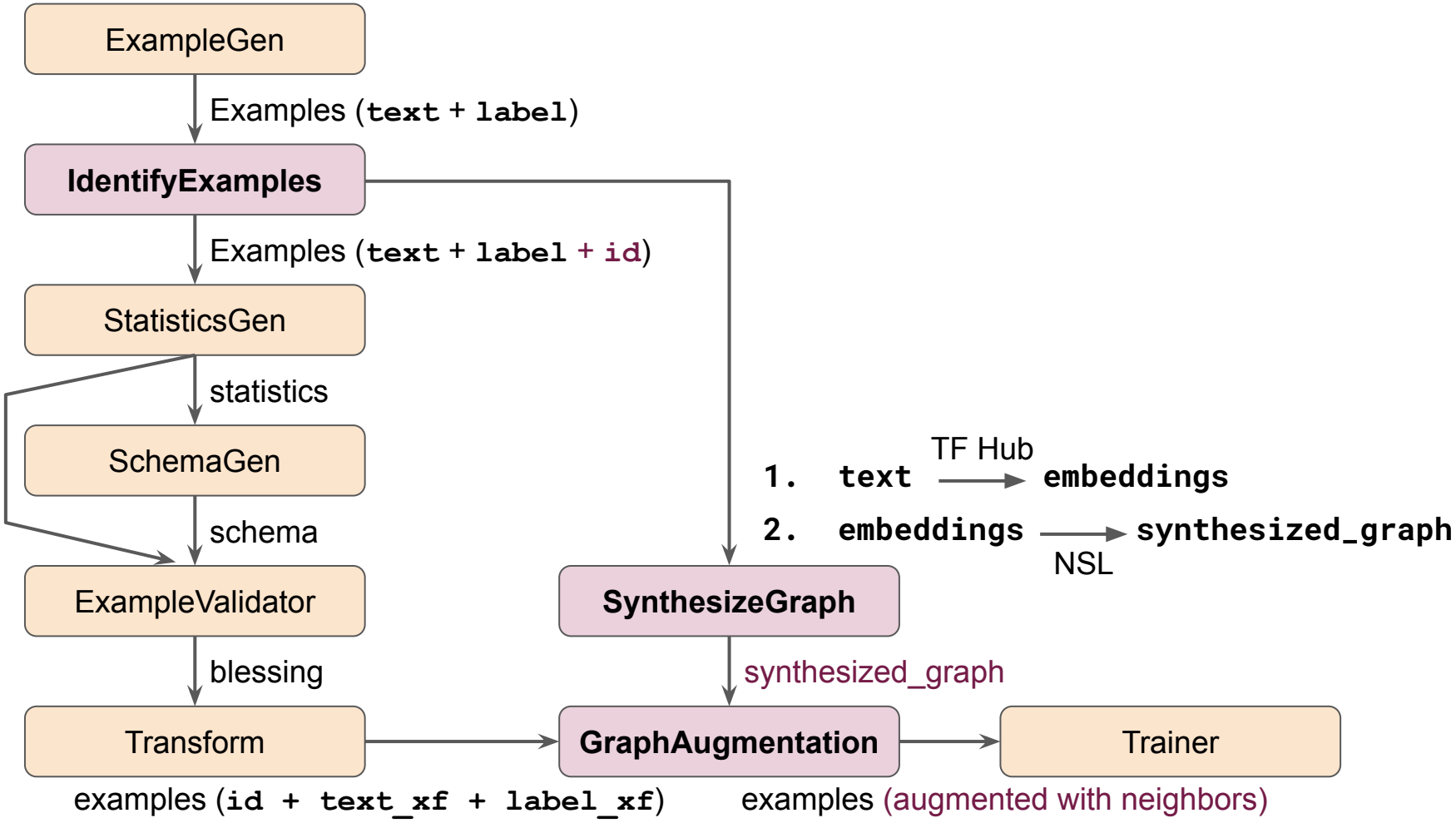

ExampleGen
Examples (text + label)
IdentifyExamples
Examples (text + label + id)
StatisticsGen
statistics
SchemaGen
schema
ExampleValidator
blessing


ExampleGen
Examples (text + label)
IdentifyExamples
Examples (text + label + id)
StatisticsGen
statistics
SchemaGen
schema
ExampleValidator
blessing
Transform
examples (text_xf + label_xf)


ExampleGen
Examples (text + label)
IdentifyExamples
Examples (text + label + id)
StatisticsGen
statistics
SchemaGen
schema
ExampleValidator
blessing
Transform
examples (text_xf + label_xf)
SynthesizeGraph
synthesized_graph


ExampleGen
Examples (text + label)
IdentifyExamples
Examples (text + label + id)
StatisticsGen
statistics
SchemaGen
schema
ExampleValidator
blessing
Transform
examples (text_xf + label_xf)
TF Hub
1. text → embeddings
2. embeddings → synthesized_graph
NSL
SynthesizeGraph
synthesized_graph


ExampleGen
Examples (text + label)
IdentifyExamples
Examples (text + label + id)
StatisticsGen
statistics
SchemaGen
schema
ExampleValidator
blessing
Transform
examples (text_xf + label_xf)
TF Hub
1. text → embeddings
2. embeddings → synthesized_graph
NSL
SynthesizeGraph
synthesized_graph
GraphAugmentation
examples (augmented with neighbors)


ExampleGen
Examples (text + label)
IdentifyExamples
Examples (text + label + id)
StatisticsGen
statistics
SchemaGen
schema
ExampleValidator
blessing
Transform
examples (id + text_xf + label_xf)
TF Hub
1. text → embeddings
2. embeddings → synthesized_graph
NSL
SynthesizeGraph
synthesized_graph
GraphAugmentation
examples (augmented with neighbors)
Trainer

	Parses	Transforms	Expects Label	Expects augmented
train_input_fn	No	No	Yes	Yes

	Parses	Transforms	Expects Label	Expects augmented
train_input_fn	No	No	Yes	Yes
eval_input_fn	No	No	Yes	No

	Parses	Transforms	Expects Label	Expects augmented
train_input_fn	No	No	Yes	Yes
eval_input_fn	No	No	Yes	No
serving_receiver_fn	Yes	Yes	No	No

	Parses	Transforms	Expects Label	Expects augmented
train_input_fn	No	No	Yes	Yes
eval_input_fn	No	No	Yes	No
serving_receiver_fn	Yes	Yes	No	No
receiver_fn (TFMA)	Yes	Yes	Yes	No

	Parses	Transforms	Expects Label	Expects augmented
train_input_fn	No	No	Yes	Yes
eval_input_fn	No	No	Yes	No
serving_receiver_fn	Yes	Yes	No	No
receiver_fn (TFMA)	Yes	Yes	Yes	No

receiver_fn (TFMA) must return both the raw features and the transformed features

	Parses	Transforms	Expects Label	Expects augmented
train_input_fn	No	No	Yes	Yes
eval_input_fn	No	No	Yes	Yes
serving_receiver_fn	Yes	Yes	No	Yes
receiver_fn (TFMA)	Yes	Yes	Yes	Yes

receiver_fn (TFMA) must return both the raw features and the transformed features

Lab 10

NSL in TFX

TensorFlow

Thank You!