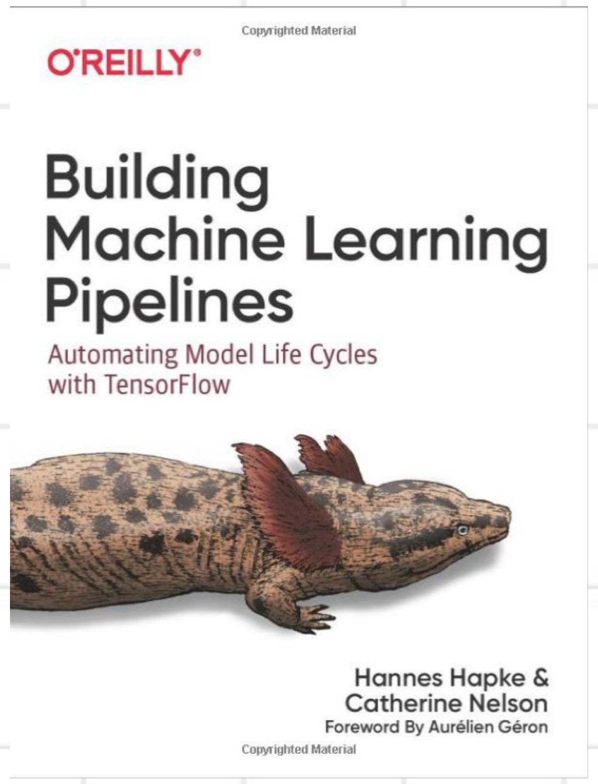
Build an ML pipeline for BERT models with TensorFlow Extended

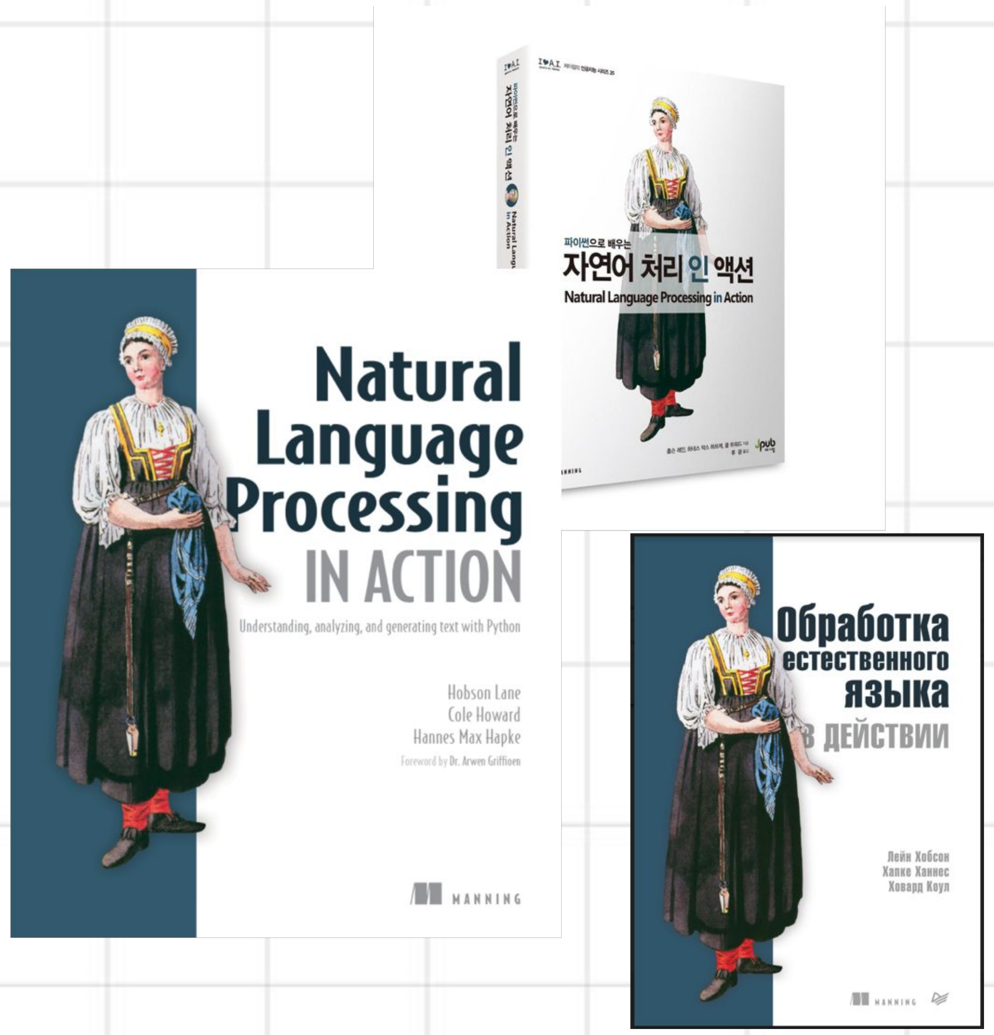
Hannes Hapke
@hanneshapke

Hi, I am Hannes

Machine Learning Enthusiast

- Senior Machine Learning Engineer at SAP
- Google Developer Expert
- Co-Author of ML Publications: “Building Machine Learning Pipelines” (O’Reilly Media) and “NLP in Action” (Manning Publishing)





Things we will be discussing

Presentation (25-30 min)

- What is ML Engineering?
- What is BERT?
- What are ML Pipelines?
- Brief overview of the implementation (demo model, pipeline setup, etc.)

Break (5 min)

Workshop (50 min)

- Walk through an end-to-end pipeline implementation

Q&A

Takeaways from this session

How to ....

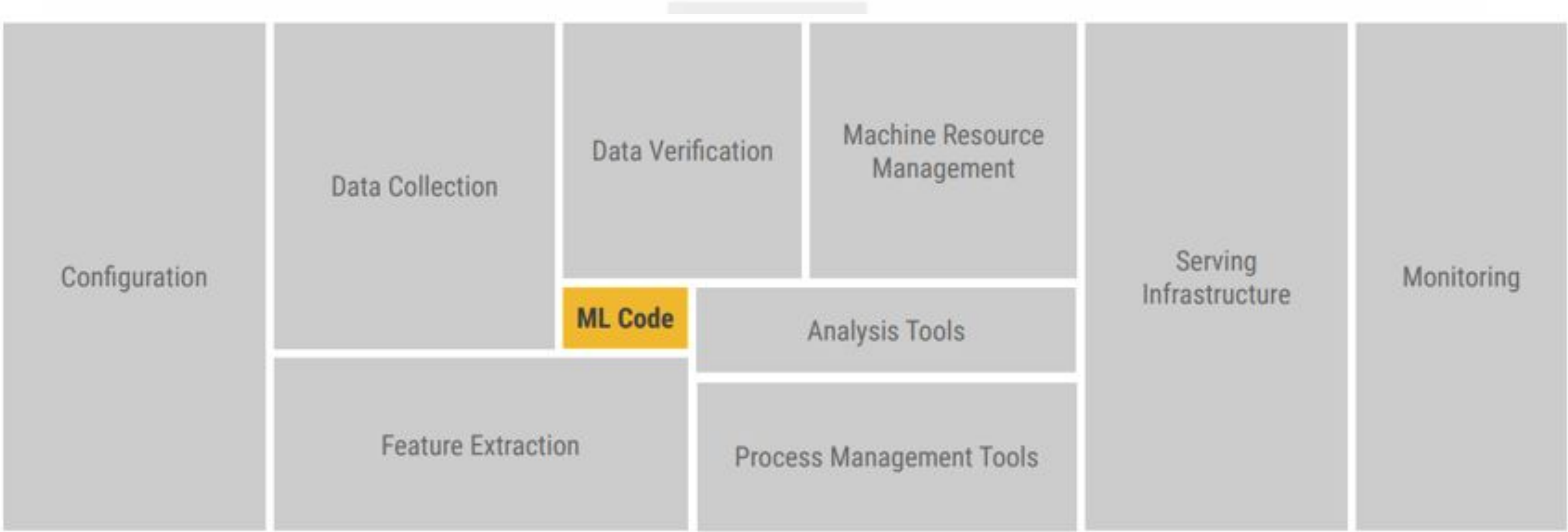
- Build reproducible ML pipelines
- Use TF Extended
- Incorporate TF Text and how to use tf.raggedTensors
- Perform feature engineering with TF Transform
- Validate your project data
- Validate and evaluate your ML model
- Deploy your ML model

ML Engineering

What is ML Engineering?

ML Code

What is ML Engineering?



Original image: https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

Why we need ML Engineering?

- Integrate models in Real world scenarios
- Focus on reproducibility
- Provide traceability via audit trails
- Reduce burden for data scientists

Model Experiments → **Production Models**

Models produced via Jupyter / colab notebooks

Models produced via Orchestrated Pipelines

What happens to trained models?

Most models don’t get deployed


ginablaber
@ginablaber

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

10:19 AM · Mar 7, 2018 · TweetDeck

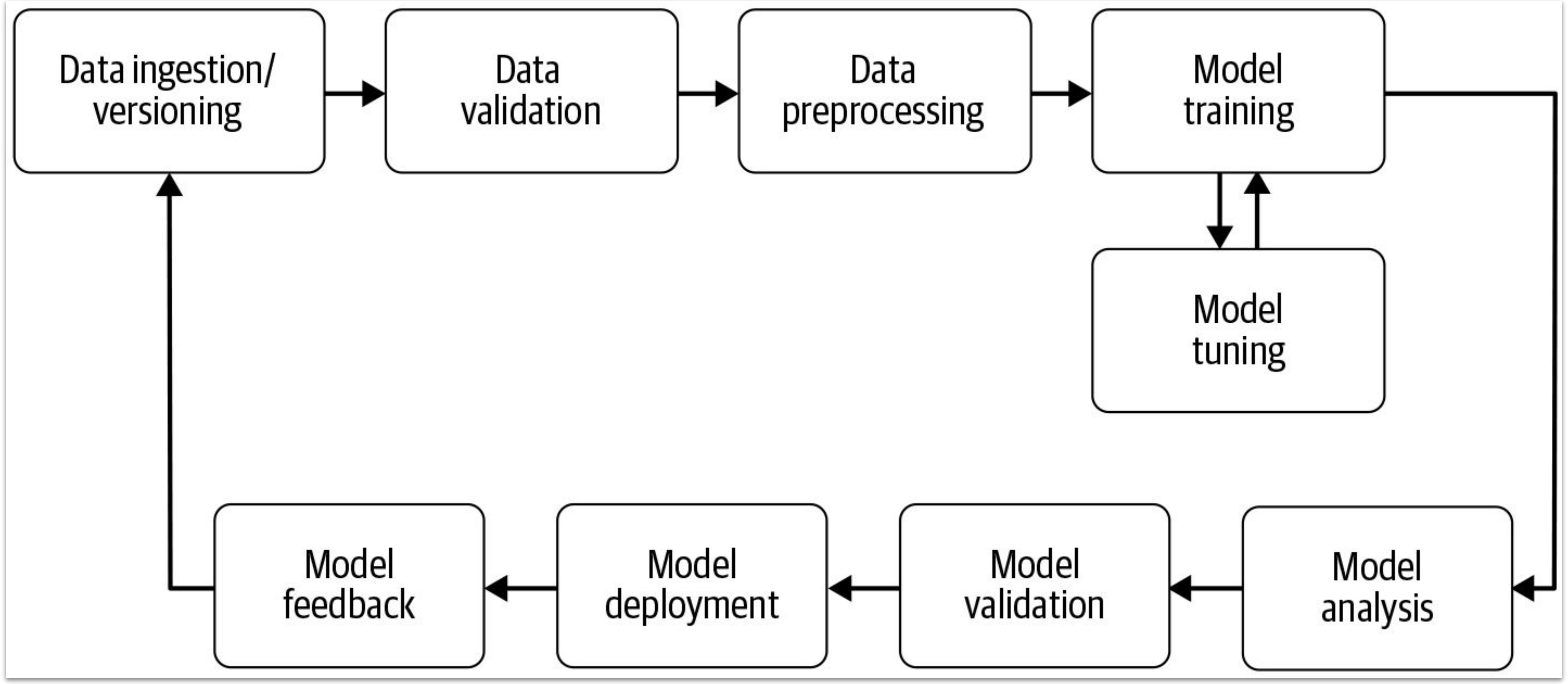
And if they get deployed ...

They experience ...

- Data drift
- Changing data schemas
- Training-serving skews
- Changing preprocessing steps
- Complicated retraining processes
- High prediction latencies

ML Model Life Cycle

What is a ML Model Life Cycle?

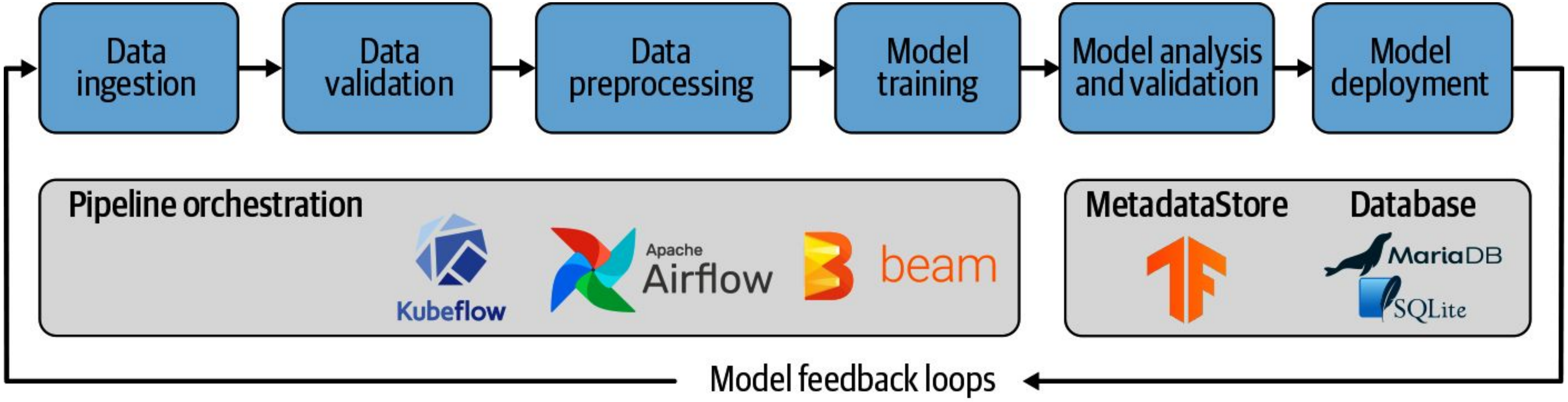


Original image: “Building Machine Learning Pipelines” - https://learning.oreilly.com/library/view/building-machine-learning/9781492053187/

Components of
ML Systems are entangled.

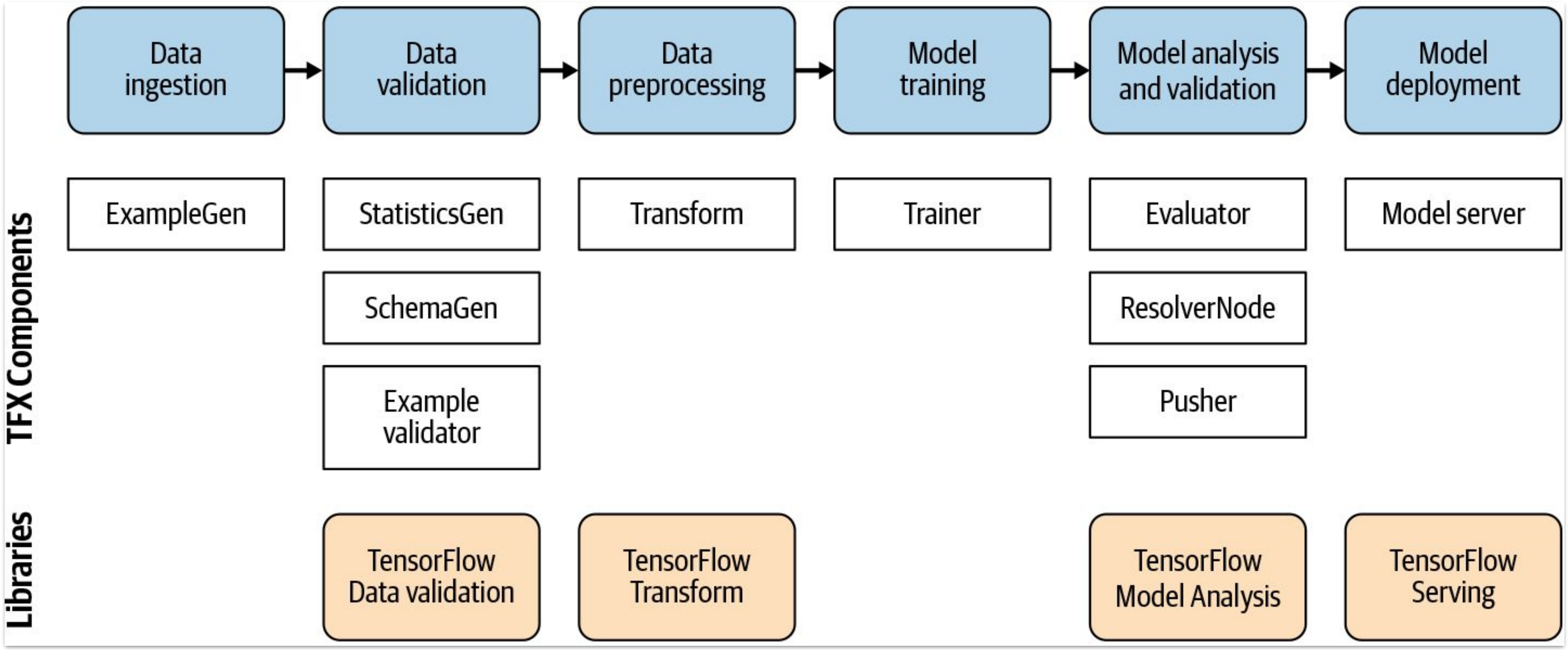
TFX is here to help!

TFX components



Original image: "Building Machine Learning Pipelines" - https://learning.oreilly.com/library/view/building-machine-learning/9781492053187/

TFX components



Original image: “Building Machine Learning Pipelines” - https://learning.oreilly.com/library/view/building-machine-learning/9781492053187/

TFX Pipeline Orchestration

jupyter

Apache Beam

Apache Airflow

Kubeflow

Transformer Models

What is BERT?

- “**B**idirectional **E**ncoder **R**epresentations from **T**ransformers”
- Transformer model
- Attention based
- Pre-trained, open source model
- Multi-language support

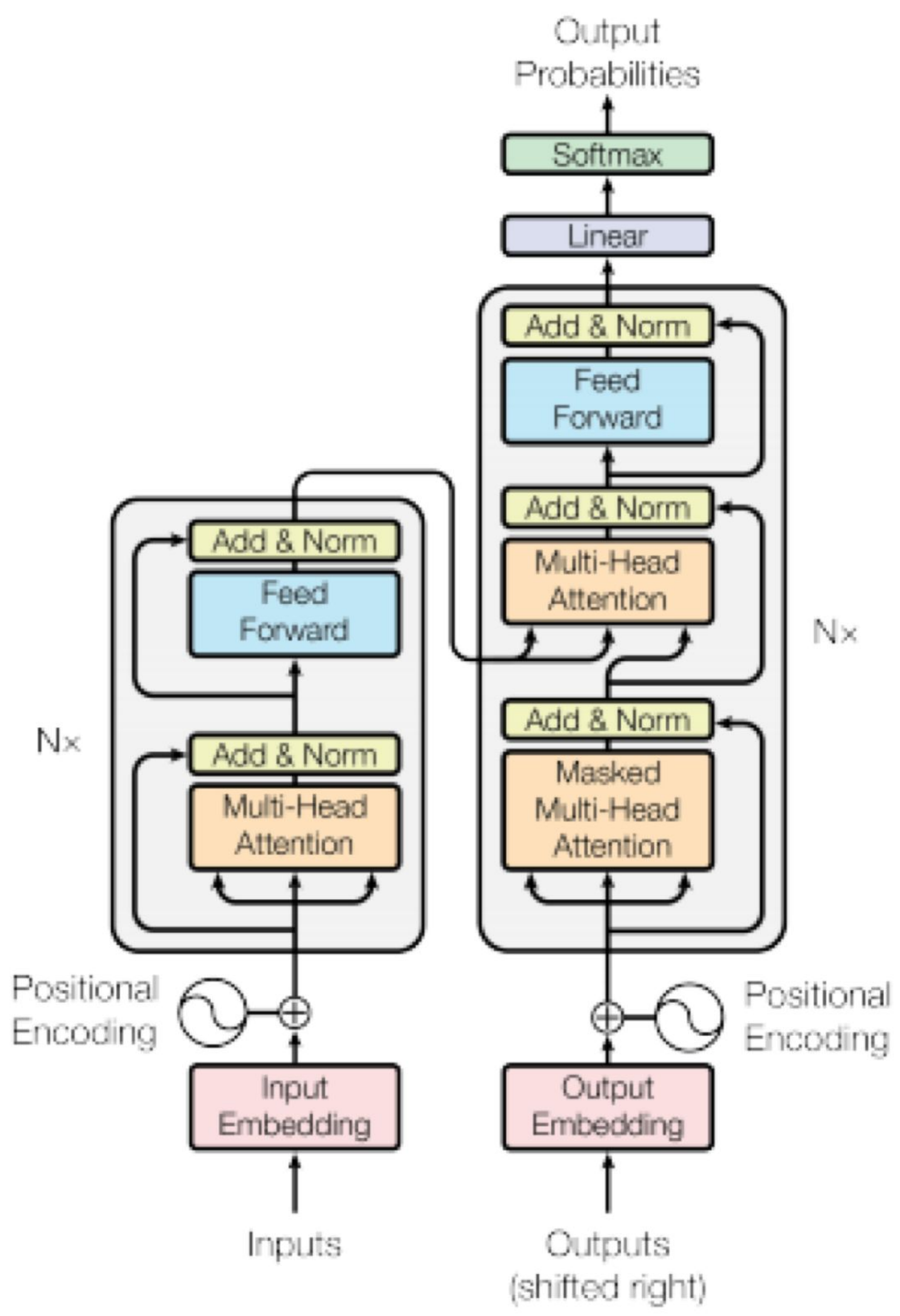


Figure 1: The Transformer - model architecture.

What is BERT?

- “**B**idirectional **E**ncoder **R**epresentations from **T**ransformers”
- Transformer model
- Attention based
- Pre-trained, open source model
- Multi-language support

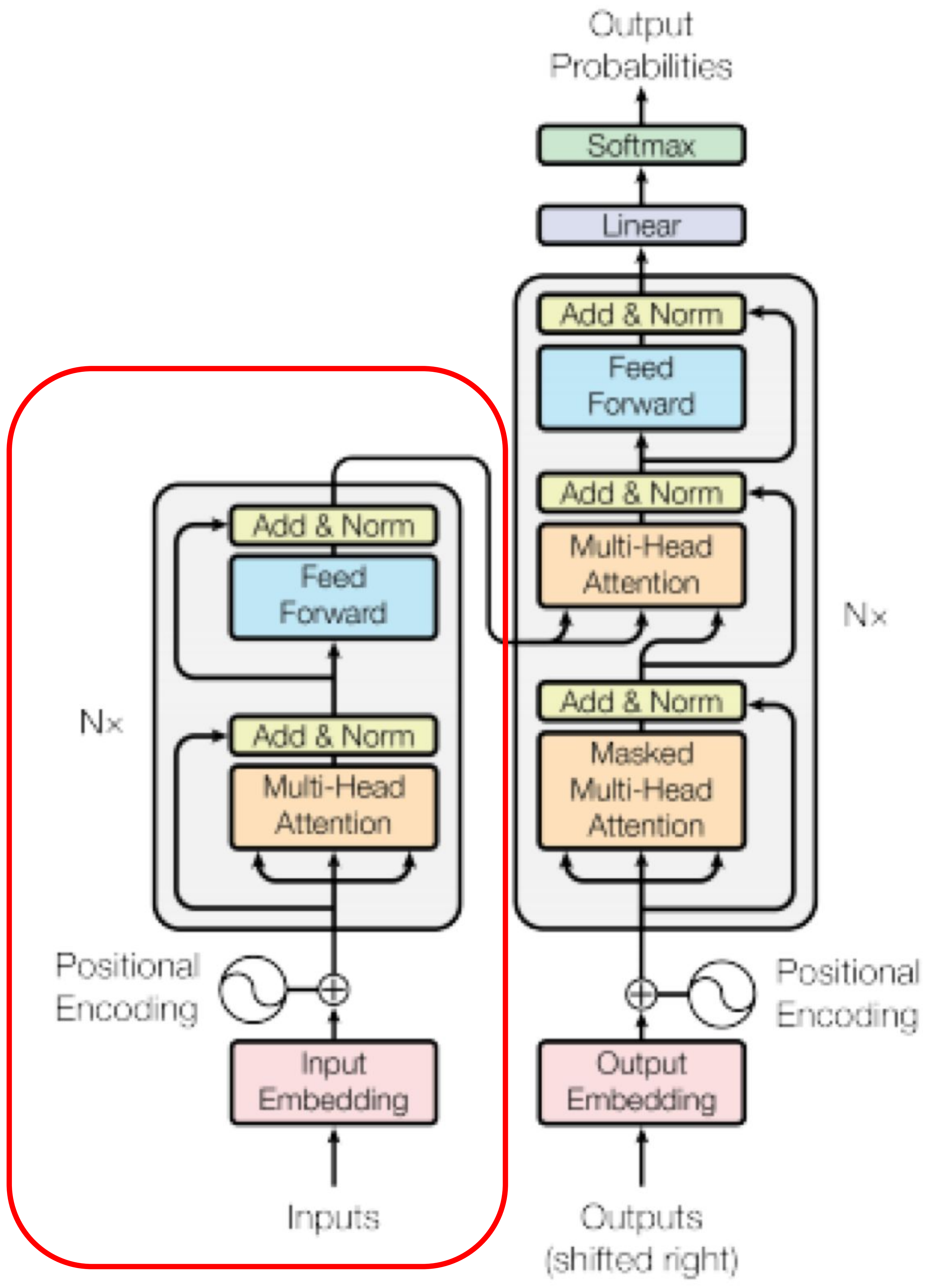
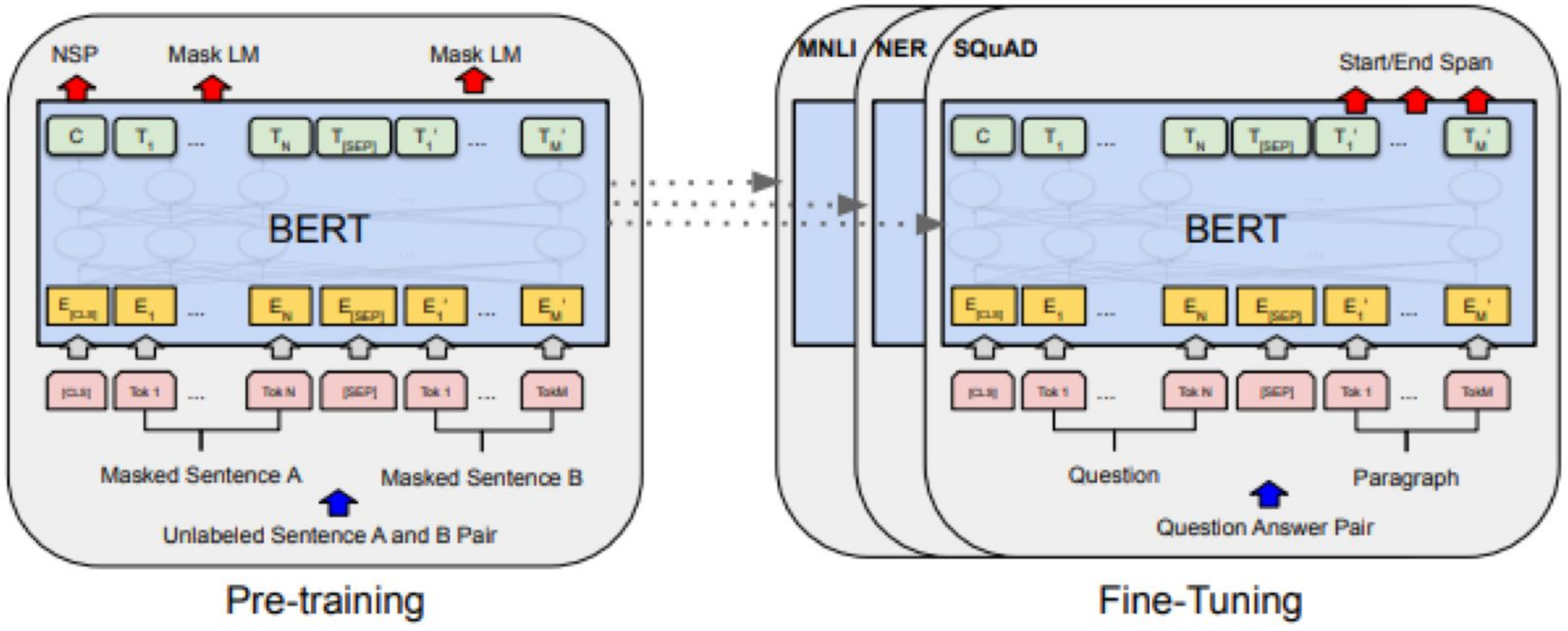


Figure 1: The Transformer - model architecture.

How is BERT trained?

- Trained on 2 tasks
  - Masked Language Model
  - Next Sentence Prediction

- Fine tuned on other NLP tasks
  - Questions-Answers
  - Named Entity Recognition
  - Classifications



Tokenization

- “Traditional” Approach
- Split on white spaces, commas, periods, etc.

  Tim manufactured tractors -> [‘Tim’, ‘manufactured’, ‘tractors’]

- One word, one token
- Language specific vocabularies
- Large vocabularies
- UNK tokens

Subword Tokenization

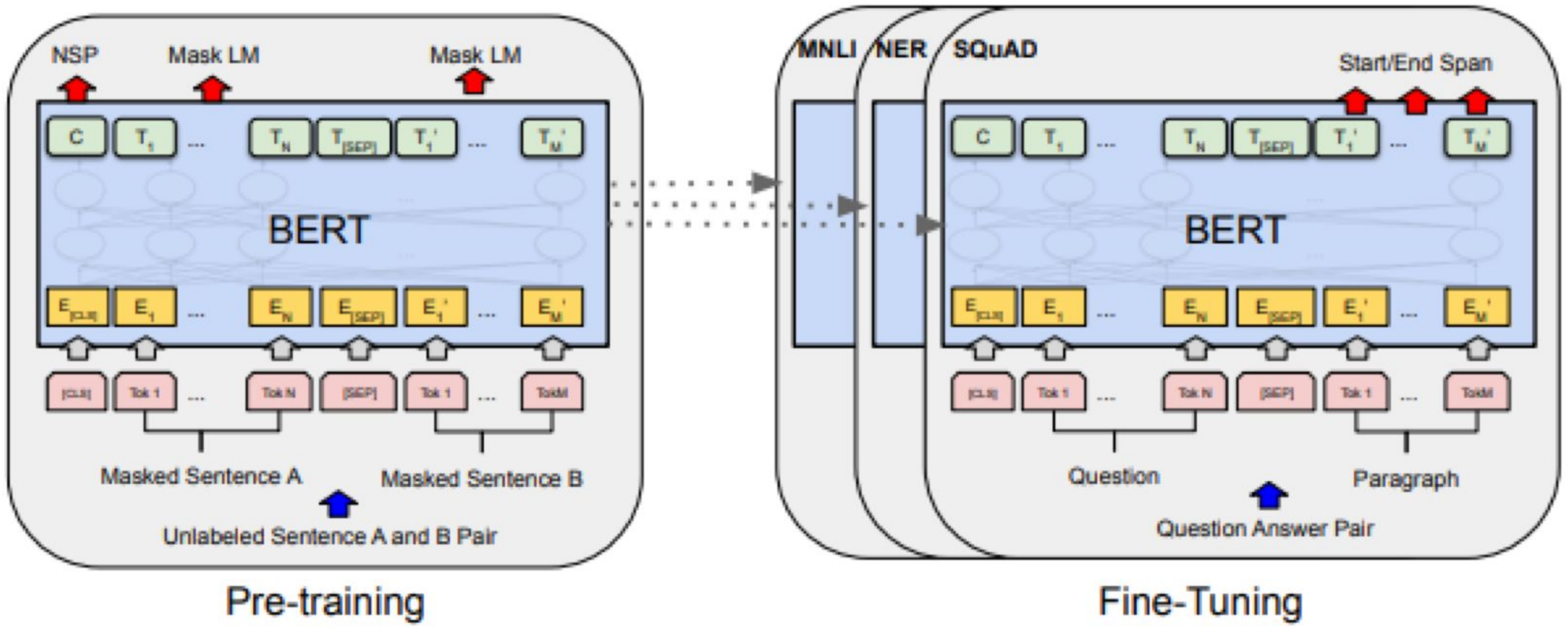
- Started with Facebook's FastText
- Split on most common character "blocks"

  Tim manufactured tractors ->
  [['Tim'], ['ma', '##k', '##nu', '##fa', '##cture', '##s'], ['tra', '##ctors']]

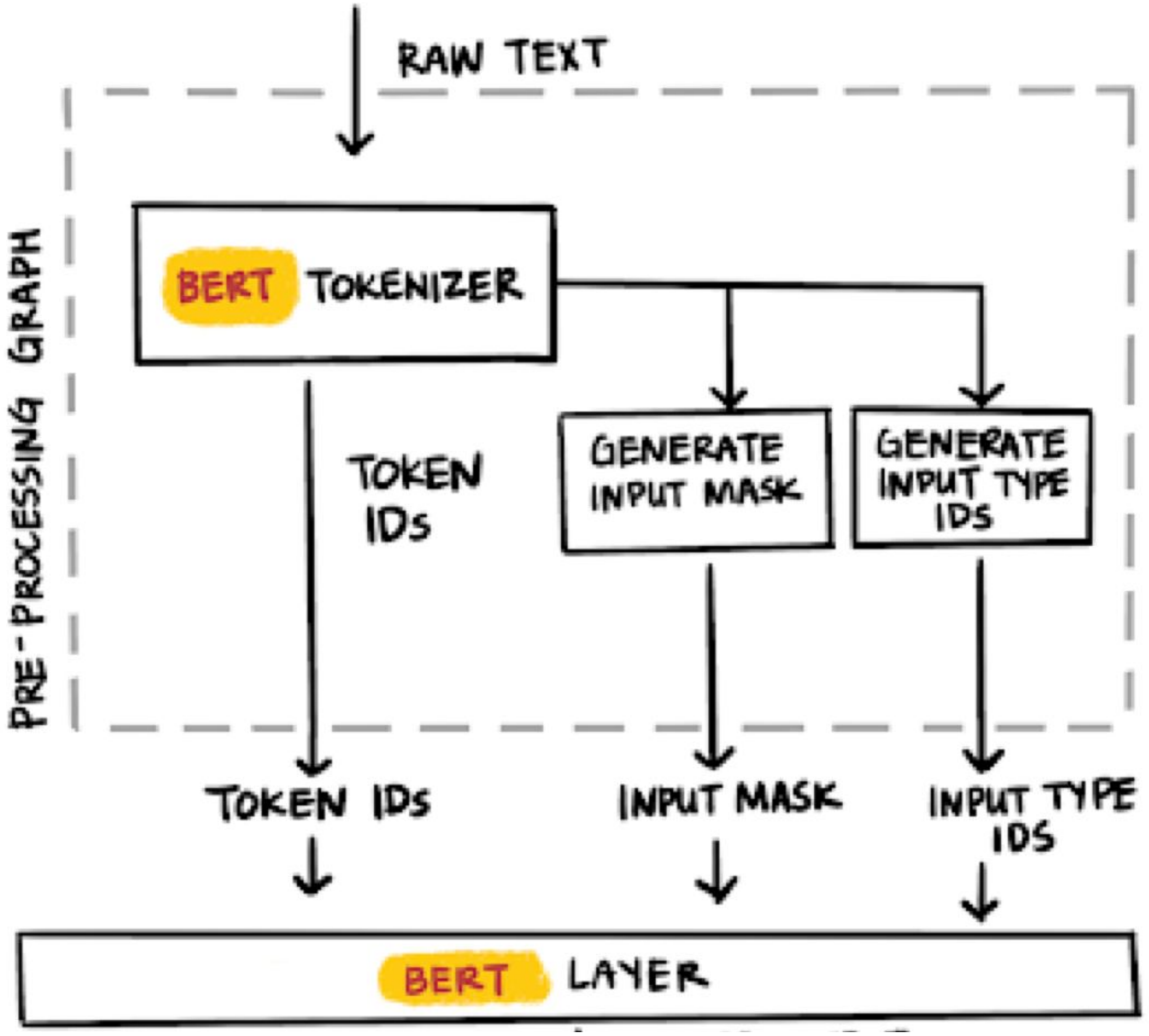
- Vocabularies aren't language specific anymore
- Smaller vocabularies
- Wordpiece vs. SentencePiece Tokenization

BERT Data Structures

- Model specific tokens: CLS, SEP
- Sequence vectors vs pooled vectors
- Inputs are based on
  - Token ids
  - Segment ids
  - Input mask
- Input/Output limitations (max 512 tokens)



BERT Input Preprocessing



BERT Input Preprocessing II

- Token IDs (Combination of question and context)
  [101, 1129, 387, ..., 102, 6830, 9983, 8983, ... 102, 0, 0, ...]
- Input Mask
  [1, 1, 1, 1, 1, 1, 1, 1, ... 1, 0, 0, 0, ...]
- Input Type IDs
  [0, 0, 0, 0, 0, 0, 0, ..., 0, 1, 1, 1, ... 1, 0, 0, 0, ...]

BERT Input Preprocessing III

Raw Text

Clara is playing the piano

BERT Deployment Complexities

- Tokenization
- Model inputs require preprocessing beyond tokenization
- Large memory footprint
- GPU requirements (batching requirements)
- Potential high prediction latencies

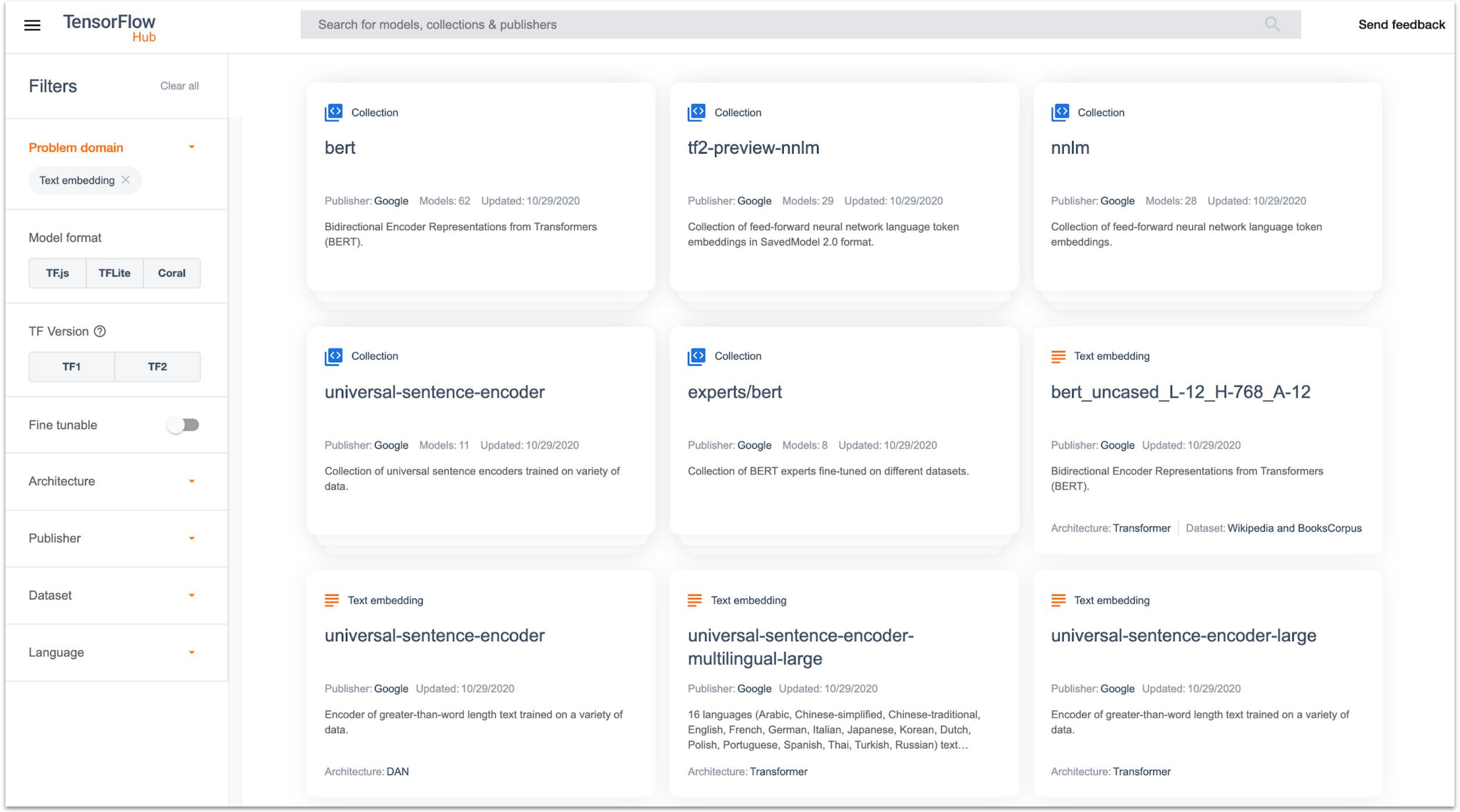
TensorFlow Ecosystem

TensorFlow Text

- Allows string operations with TensorFlow
- Python's string.lower() isn't an option
- Library provides string ops and tokenizers
- Concept of Ragged Tensors

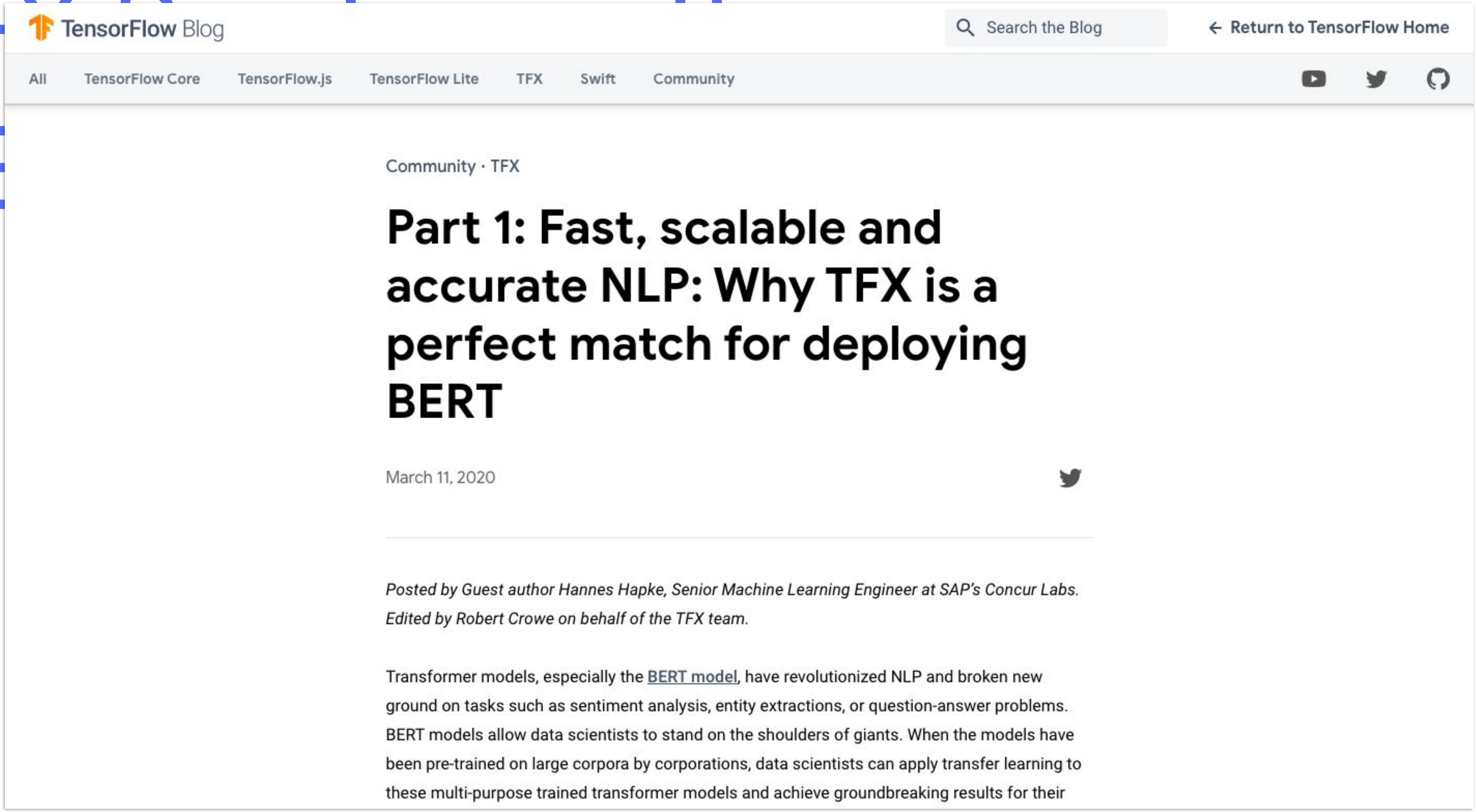
TensorFlow Hub

- Pre-trained models can be incorporated in your models



TFX in Action!

TF
BE



TensorFlow Blog

Search the Blog

← Return to TensorFlow Home

All TensorFlow Core TensorFlow.js TensorFlow Lite TFX Swift Community

Community · TFX

Part 1: Fast, scalable and accurate NLP: Why TFX is a perfect match for deploying BERT

March 11, 2020

Posted by Guest author Hannes Hapke, Senior Machine Learning Engineer at SAP's Concur Labs. Edited by Robert Crowe on behalf of the TFX team.

Transformer models, especially the BERT model, have revolutionized NLP and broken new ground on tasks such as sentiment analysis, entity extractions, or question-answer problems. BERT models allow data scientists to stand on the shoulders of giants. When the models have been pre-trained on large corpora by corporations, data scientists can apply transfer learning to these multi-purpose trained transformer models and achieve groundbreaking results for their

Data Ingestion

Define splits and spans

- Split the data effectively
- Define the split ratios

```
output = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(
            name='train', hash_buckets=3),
        example_gen_pb2.SplitConfig.Split(
            name='eval', hash_buckets=1)
    ])
)

examples = external_input("/path/to/data/")
example_gen = ImportExampleGen(
    input=examples, output_config=output)
```

Data Ingestion

Define splits and spans

- Split the data effectively
- Define the split ratios

```
output = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(
            name='train', hash_buckets=3),
        example_gen_pb2.SplitConfig.Split(
            name='eval', hash_buckets=1)
    ])
)

examples = external_input("/path/to/data/")
example_gen = ImportExampleGen(
    input=examples, output_config=output)
```

Experts

Data Ingestion

Define splits and spans

- Split the data effectively
- Define the split ratios

```
output = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(
            name='train', hash_buckets=3),
        example_gen_pb2.SplitConfig.Split(
            name='eval', hash_buckets=1)
    ])
)

examples = external_input("/path/to/data/")
example_gen = ImportExampleGen(
    input=examples, output_config=output)
```

Experts

Data Validation

Data statistics and schema

- Create data statistics
- Create data schema

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

schema_gen = SchemaGen(
    statistics=statistics_gen.outputs['statistics'],
    infer_feature_shape=True)
```

Data Validation

Data statistics and schema

- Create data statistics
- Create data schema

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

schema_gen = SchemaGen(
    statistics=statistics_gen.outputs['statistics'],
    infer_feature_shape=True)
```

Data Validation

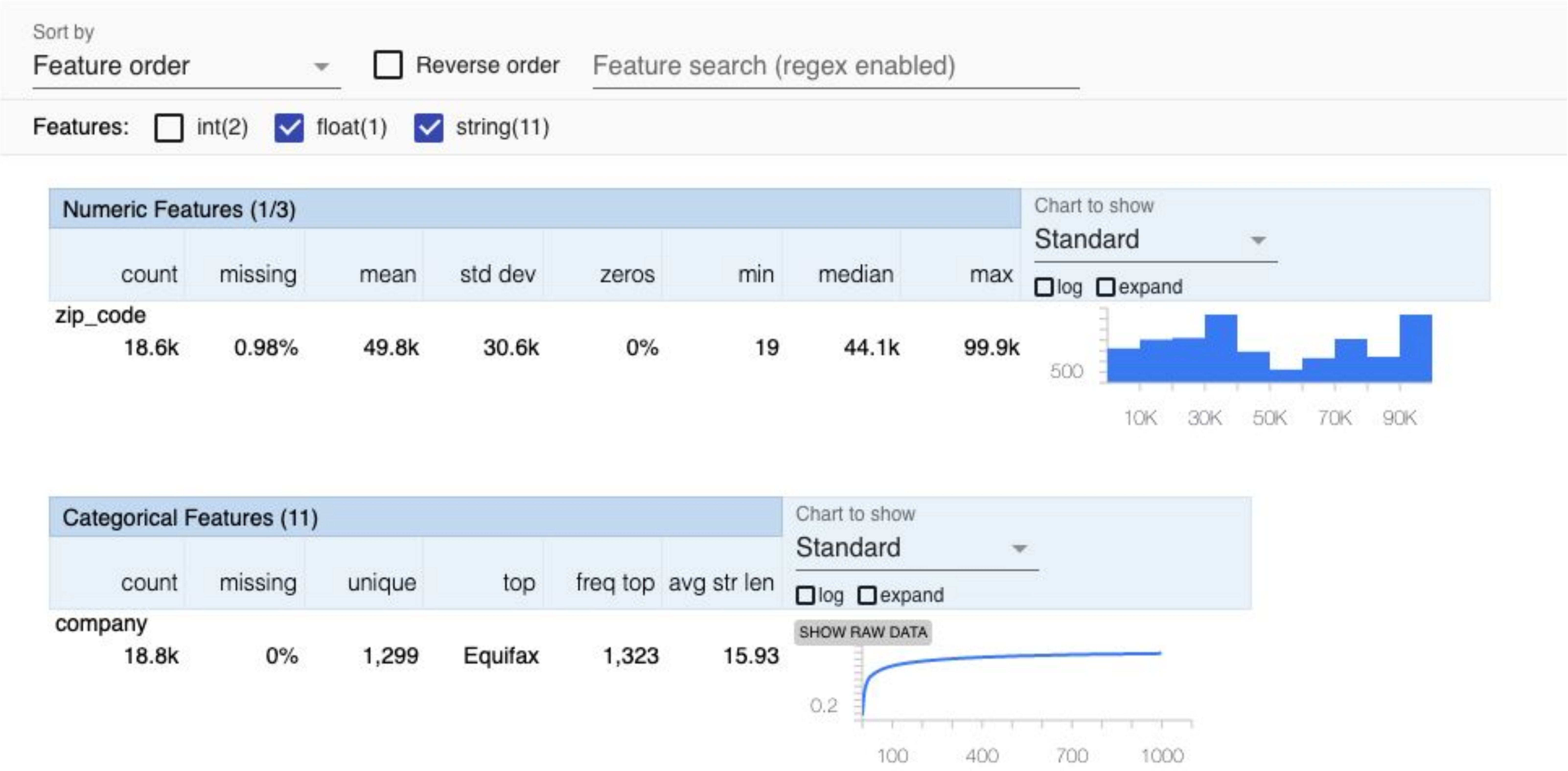
Data statistics and schema

- Create data statistics
- Create data schema

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

schema_gen = SchemaGen(
    statistics=statistics_gen.outputs['statistics'],
    infer_feature_shape=True)
```

Experts

Data Validation



Original image: “Building Machine Learning Pipelines” - https://learning.oreilly.com/library/view/building-machine-learning/9781492053187/

Data Transformation

Consistent data preprocessing

- Create powerful preprocessing
- Provides consistent preprocessing graph

Experts

```
# transform.py

def preprocessing_fn(inputs):
    ...
    bert_tokenizer = text.BertTokenizer(...)
    ...
    input_word_ids, input_mask, input_type_ids = \
        preprocess_bert_input(
            tf.squeeze(inputs['text'], axis=1))

    return {
        'input_word_ids': input_word_ids,
        'input_mask': input_mask,
        'input_type_ids': input_type_ids,
        'label': inputs['label']
    }

# load preprocessing steps in pipeline

transform = Transform(
    examples=example_gen.outputs['examples'],
    schema=schema_gen.outputs['schema'],
    module_file=os.path.abspath("transform.py"))
```

Data Transformation

Consistent data preprocessing

- Create powerful preprocessing
- Provides consistent preprocessing graph

```
# transform.py

def preprocessing_fn(inputs):
    ...
    bert_tokenizer = text.BertTokenizer(...)
    ...
    input_word_ids, input_mask, input_type_ids = \
        preprocess_bert_input(
            tf.squeeze(inputs['text'], axis=1))

    return {
        'input_word_ids': input_word_ids,
        'input_mask': input_mask,
        'input_type_ids': input_type_ids,
        'label': inputs['label']
    }

# load preprocessing steps in pipeline

transform = Transform(
    examples=example_gen.outputs['examples'],
    schema=schema_gen.outputs['schema'],
    module_file=os.path.abspath("transform.py"))
```

Model Training

Automate the model training

- Very similar to “manual” training
- Mirrored Distribution Strategy can be used
- Exported model includes the preprocessing steps

```python
# trainer.py

def run_fn(fn_args: TrainerFnArgs):
    tf_transform_output = tft.TFTransformOutput(...)
    train_dataset = _input_fn(...)
    ...
    with mirrored_strategy.scope():
        model = get_model(
            tf_transform_output=tf_transform_output)

    model.fit(train_dataset, ...)
    model.save(fn_args.serving_model_dir, ...)

# load preprocessing steps in pipeline

trainer = Trainer(
    module_file=os.path.abspath("trainer.py"),
    ...
)
```

Model Training

Automate the model training

- Very similar to “manual” training
- Mirrored Distribution Strategy can be used
- Exported model includes the preprocessing steps

```
# trainer.py

def run_fn(fn_args: TrainerFnArgs):
    tf_transform_output = tft.TFTransformOutput(...)
    train_dataset = _input_fn(...)
    ...
    with mirrored_strategy.scope():
        model = get_model(
            tf_transform_output=tf_transform_output)

    model.fit(train_dataset, ...)
    model.save(fn_args.serving_model_dir, ...)

# load preprocessing steps in pipeline

trainer = Trainer(
    module_file=os.path.abspath("trainer.py"),
    ...
)
```

Model Training

Automate the model training

- Very similar to “manual” training
- Mirrored Distribution Strategy can be used
- Exported model includes the preprocessing steps

```
# trainer.py

def run_fn(fn_args: TrainerFnArgs):
    tf_transform_output = tft.TFTransformOutput(...)
    train_dataset = _input_fn(...)
    ...
    with mirrored_strategy.scope():
        model = get_model(
            tf_transform_output=tf_transform_output)

    model.fit(train_dataset, ...)
    model.save(fn_args.serving_model_dir, ...)

# load preprocessing steps in pipeline

trainer = Trainer(
    module_file=os.path.abspath("trainer.py"),
    ...
)
```

Model Training

Automate the model training

- Very similar to “manual” training
- Mirrored Distribution Strategy can be used
- Exported model includes the preprocessing steps

```
# trainer.py

def run_fn(fn_args: TrainerFnArgs):
    tf_transform_output = tft.TFTransformOutput(...)
    train_dataset = _input_fn(...)
    ...
    with mirrored_strategy.scope():
        model = get_model(
            tf_transform_output=tf_transform_output)

    model.fit(train_dataset, ...)
    model.save(fn_args.serving_model_dir, ...)

# load preprocessing steps in pipeline

trainer = Trainer(
    module_file=os.path.abspath("trainer.py"),
    ...
)
```

Experts

Model Training

Automate the model training

- Very similar to “manual” training
- Mirrored Distribution Strategy can be used
- Exported model includes the preprocessing steps

Experts

```
# trainer.py

def run_fn(fn_args: TrainerFnArgs):
    tf_transform_output = tft.TFTransformOutput(...)
    train_dataset = _input_fn(...)
    ...
    with mirrored_strategy.scope():
        model = get_model(
            tf_transform_output=tf_transform_output)

    model.fit(train_dataset, ...)
    model.save(fn_args.serving_model_dir, ...)

# load preprocessing steps in pipeline

trainer = Trainer(
    module_file=os.path.abspath("trainer.py"),
    ...
)
```

Model Training

Automate the model training

- Very similar to "manual" training
- Mirrored Distribution Strategy can be used
- Exported model includes the preprocessing steps

Experts

```
# trainer.py

def run_fn(fn_args: TrainerFnArgs):
    tf_transform_output = tft.TFTransformOutput(...)
    train_dataset = _input_fn(...)
    ...
    with mirrored_strategy.scope():
        model = get_model(
            tf_transform_output=tf_transform_output)

    model.fit(train_dataset, ...)
    model.save(fn_args.serving_model_dir, ...)

# load preprocessing steps in pipeline

trainer = Trainer(
    module_file=os.path.abspath("trainer.py"),
    ...
)
```

Model Analysis

Analysis and Validation

- Define criteria for model performance improvements
- Compare with previous models

```
eval_config = tfma.EvalConfig(
    model_specs=[
        tfma.ModelSpec(label_key='label')
    ],
    metrics_specs=[
        ...
    ],
    slicing_specs=[
        ...
    ])

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

Experts

Model Analysis

Analysis and Validation

- Define criteria for model performance improvements
- Compare with previous models

```
eval_config = tfma.EvalConfig(
    model_specs=[
        tfma.ModelSpec(label_key='label')
    ],
    metrics_specs=[
        ...
    ],
    slicing_specs=[
        ...
    ])

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

Model Analysis

Analysis and Validation

- Define criteria for model performance improvements
- Compare with previous models

```
eval_config = tfma.EvalConfig(
    model_specs=[
        tfma.ModelSpec(label_key='label')
    ],
    metrics_specs=[
        ...
    ],
    slicing_specs=[
        ...
    ])

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

Model Analysis

Analysis and Validation

- Define criteria for model performance improvements
- Compare with previous models

```
eval_config = tfma.EvalConfig(
    model_specs=[
        tfma.ModelSpec(label_key='label')
    ],
    metrics_specs=[
        ...
    ],
    slicing_specs=[
        ...
    ])

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

Experts

Model Analysis

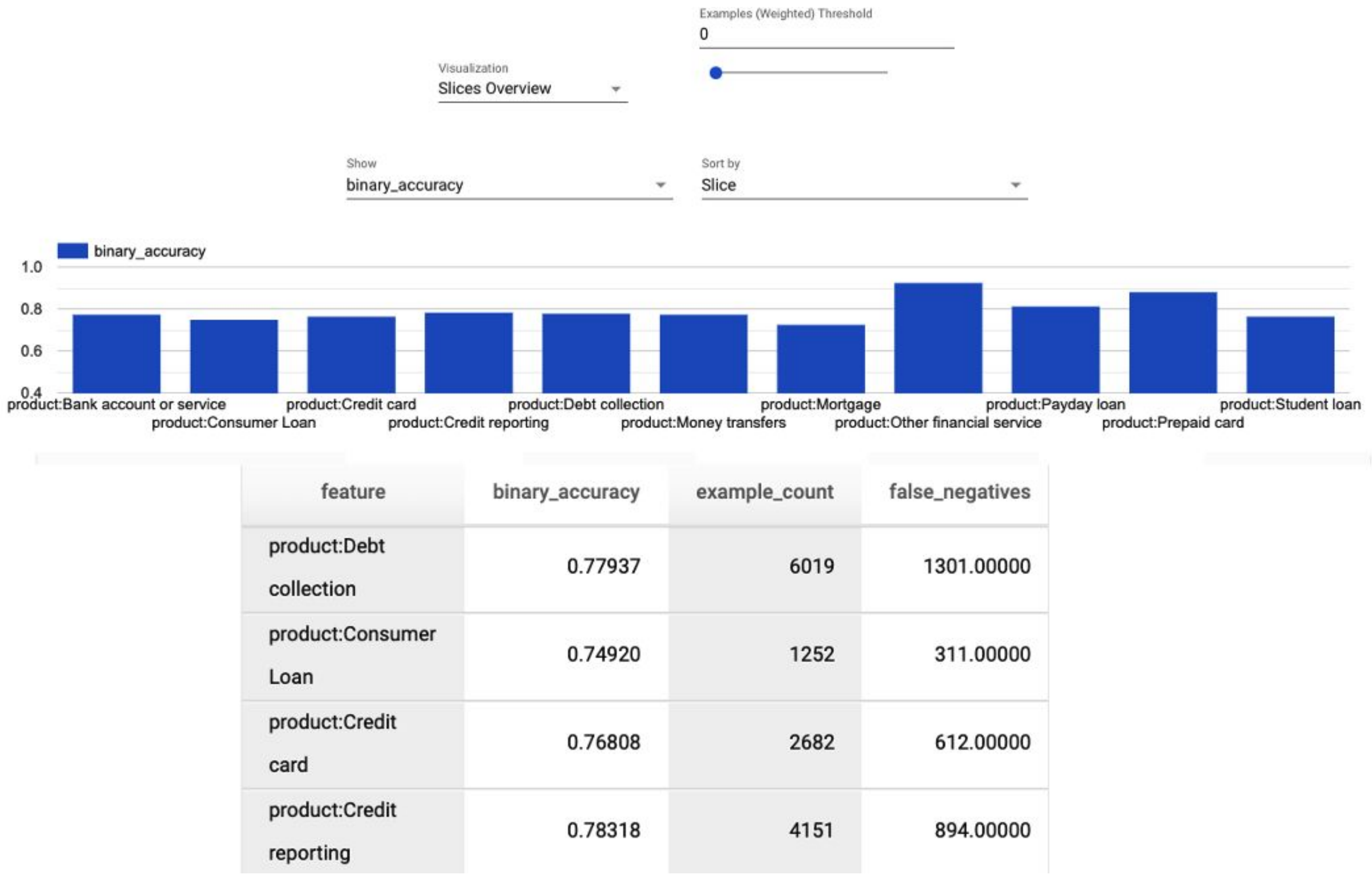
Analysis and Validation

- Define criteria for model performance improvements
- Compare with previous models

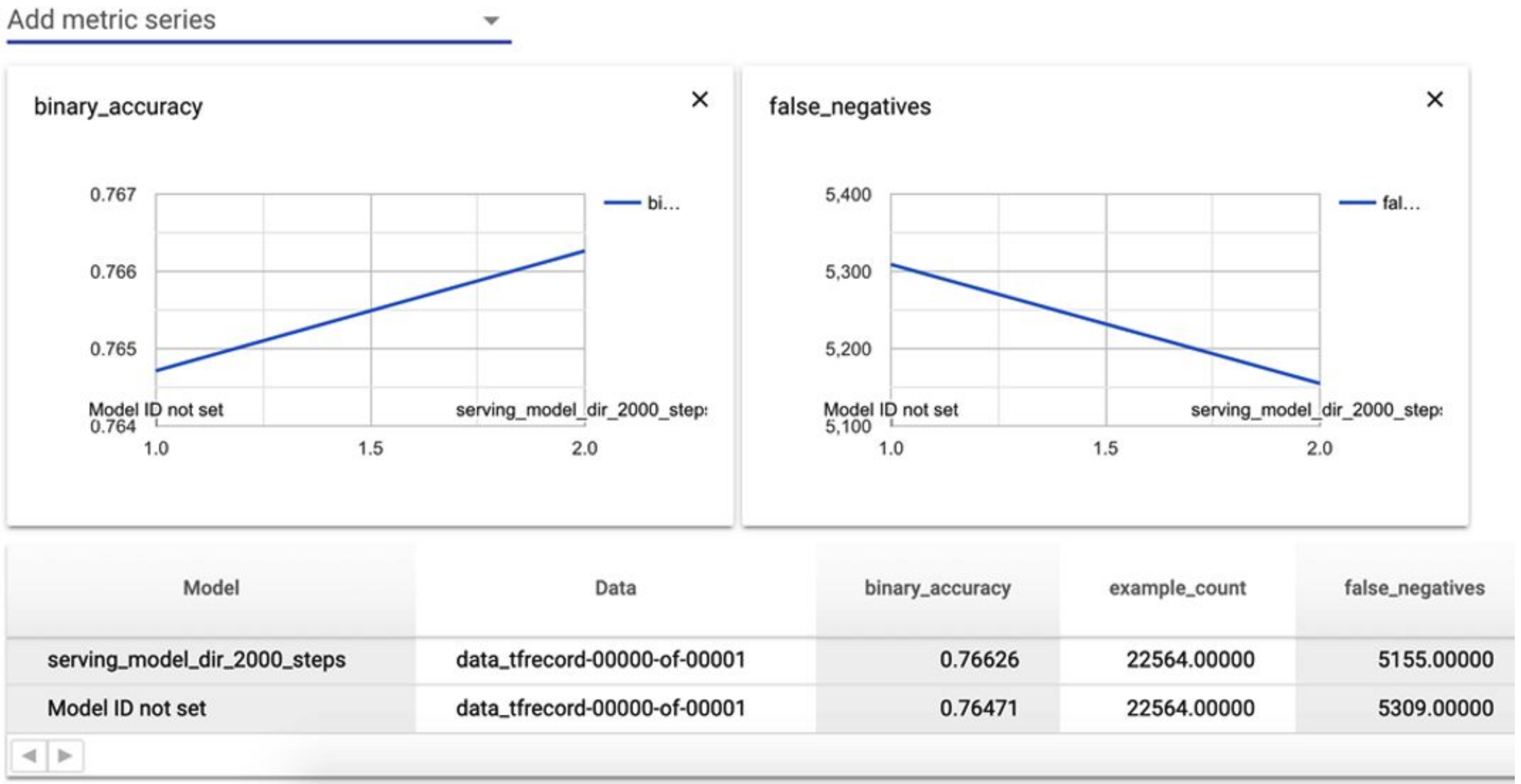
```
eval_config = tfma.EvalConfig(
    model_specs=[
        tfma.ModelSpec(label_key='label')
    ],
    metrics_specs=[
        ...
    ],
    slicing_specs=[
        ...
    ])

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

Experts

Screenshot



feature	binary_accuracy	example_count	false_negatives
product:Debt collection	0.77937	6019	1301.00000
product:Consumer Loan	0.74920	1252	311.00000
product:Credit card	0.76808	2682	612.00000
product:Credit reporting	0.78318	4151	894.00000



Model	Data	binary_accuracy	example_count	false_negatives
serving_model_dir_2000_steps	data_tfrecord-00000-of-00001	0.76626	22564.00000	5155.00000
Model ID not set	data_tfrecord-00000-of-00001	0.76471	22564.00000	5309.00000

Original image: “Building Machine Learning Pipelines” - https://learning.oreilly.com/library/view/building-machine-learning/9781492053187/

Model Deployment

Push models

- Push models to file system
- Deploy to GCP's AI Platform or AWS Sagemaker

Experts

```
serving_model_dir = "/export/path/for/the/model"

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

Execute your TFX Pipeline

jupyter

Apache Beam

Apache Airflow

Kubeflow

Execute the Pipeline

Execution for various platforms

- KubeflowDagRunner produces Argo configuration
- Airflow & Beam runner execute directly

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

runner_config = \
    kubeflow_dag_runner.KubeflowDagRunnerConfig(
        kubeflow_metadata_config=metadata_config,
        ...
    )

kubeflow_dag_runner.KubeflowDagRunner(
    config=runner_config
).run(tfx_pipeline)
```

Experts

Execute the Pipeline

Execution for various platforms

- KubeflowDagRunner produces Argo configuration
- Airflow & Beam runner execute directly

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

runner_config = \
    kubeflow_dag_runner.KubeflowDagRunnerConfig(
        kubeflow_metadata_config=metadata_config,
        ...
    )

kubeflow_dag_runner.KubeflowDagRunner(
    config=runner_config
).run(tfx_pipeline)
```

Experts

Execute the Pipeline

Execution for various platforms

- KubeflowDagRunner produces Argo configuration
- Airflow & Beam runner execute directly

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

runner_config = \
    kubeflow_dag_runner.KubeflowDagRunnerConfig(
        kubeflow_metadata_config=metadata_config,
        ...
    )

kubeflow_dag_runner.KubeflowDagRunner(
    config=runner_config
).run(tfx_pipeline)
```

Execute the Pipeline

Execution for various platforms

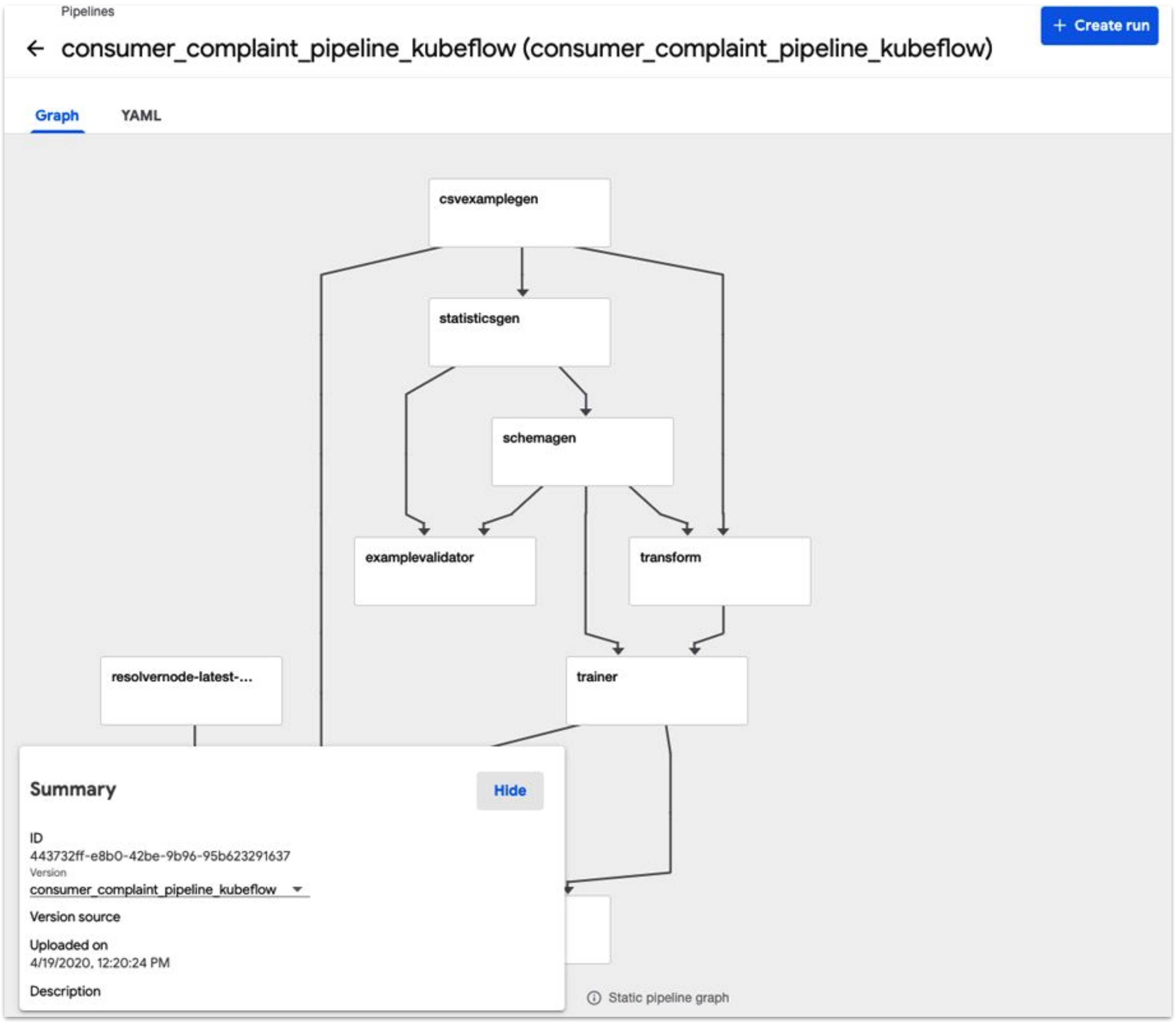
- KubeflowDagRunner produces Argo configuration
- Airflow & Beam runner execute directly

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

runner_config = \
    kubeflow_dag_runner.KubeflowDagRunnerConfig(
        kubeflow_metadata_config=metadata_config,
        ...
    )

kubeflow_dag_runner.KubeflowDagRunner(
    config=runner_config
).run(tfx_pipeline)
```

Experts

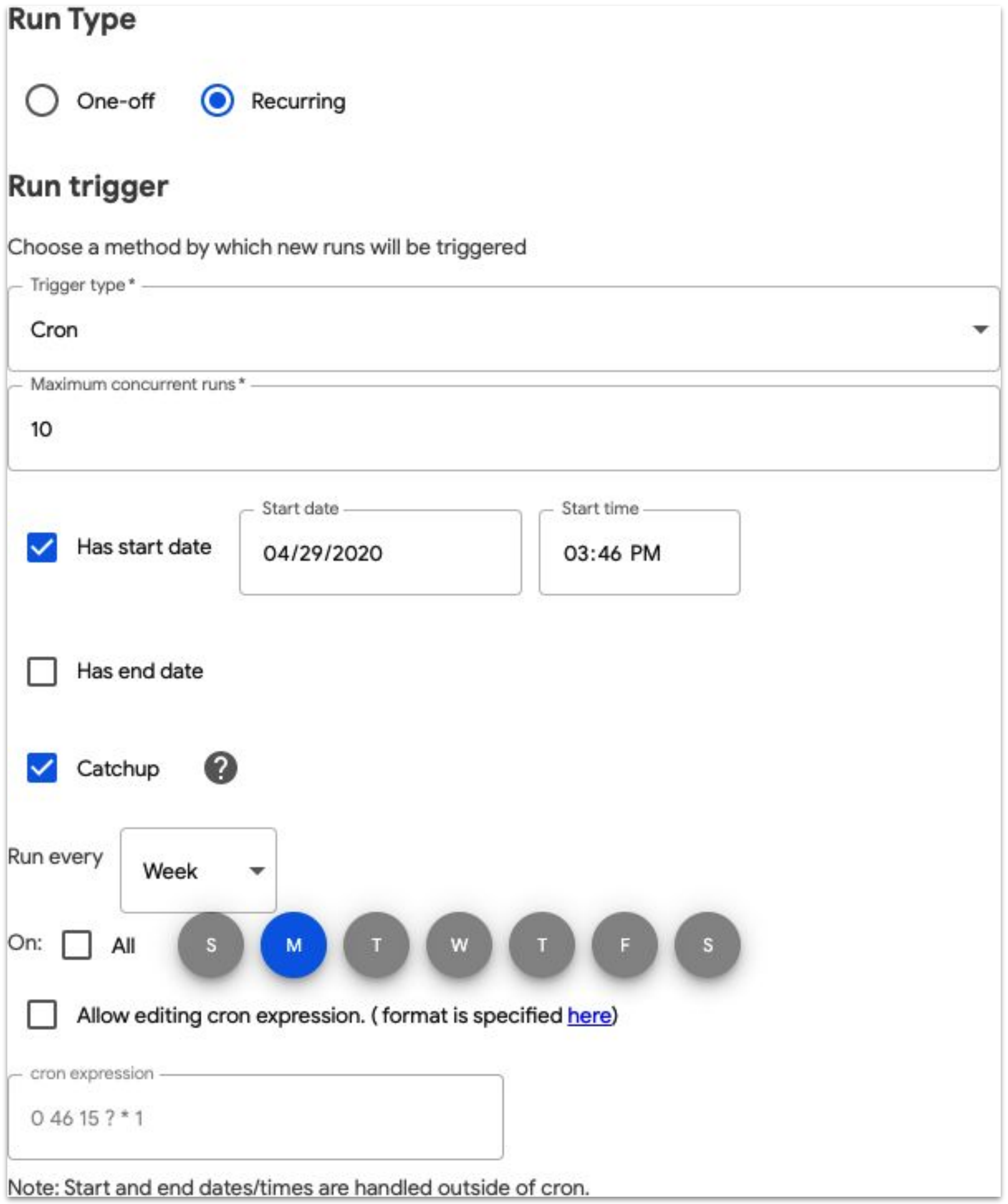
Metadata changes everything!

Kubeflow Pipelines



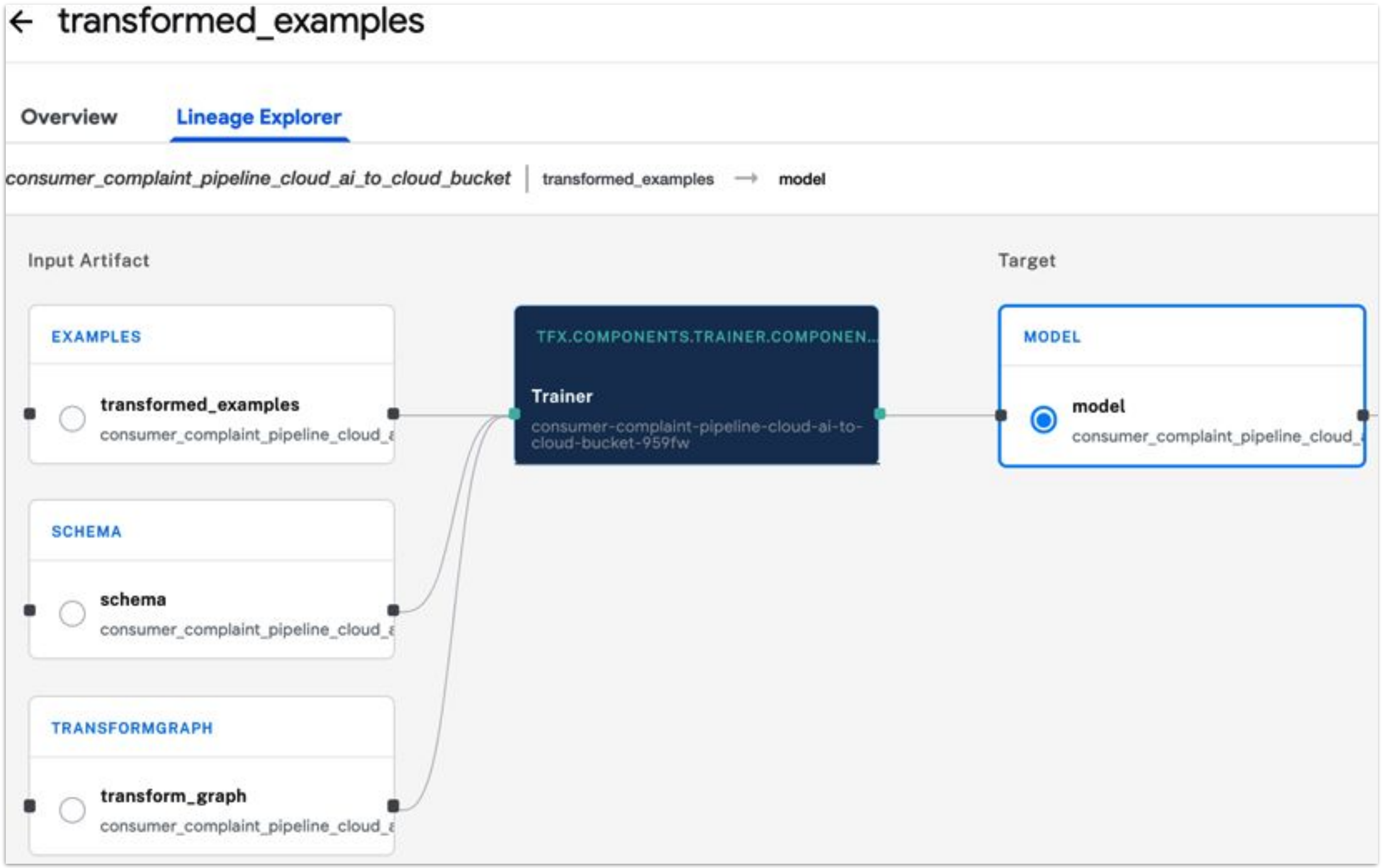
Original image: “Building Machine Learning Pipelines” - https://learning.oreilly.com/library/view/building-machine-learning/9781492053187/

Kubeflow Pipelines



Original image: “Building Machine Learning Pipelines” - https://learning.oreilly.com/library/view/building-machine-learning/9781492053187/

Kubeflow Pipelines



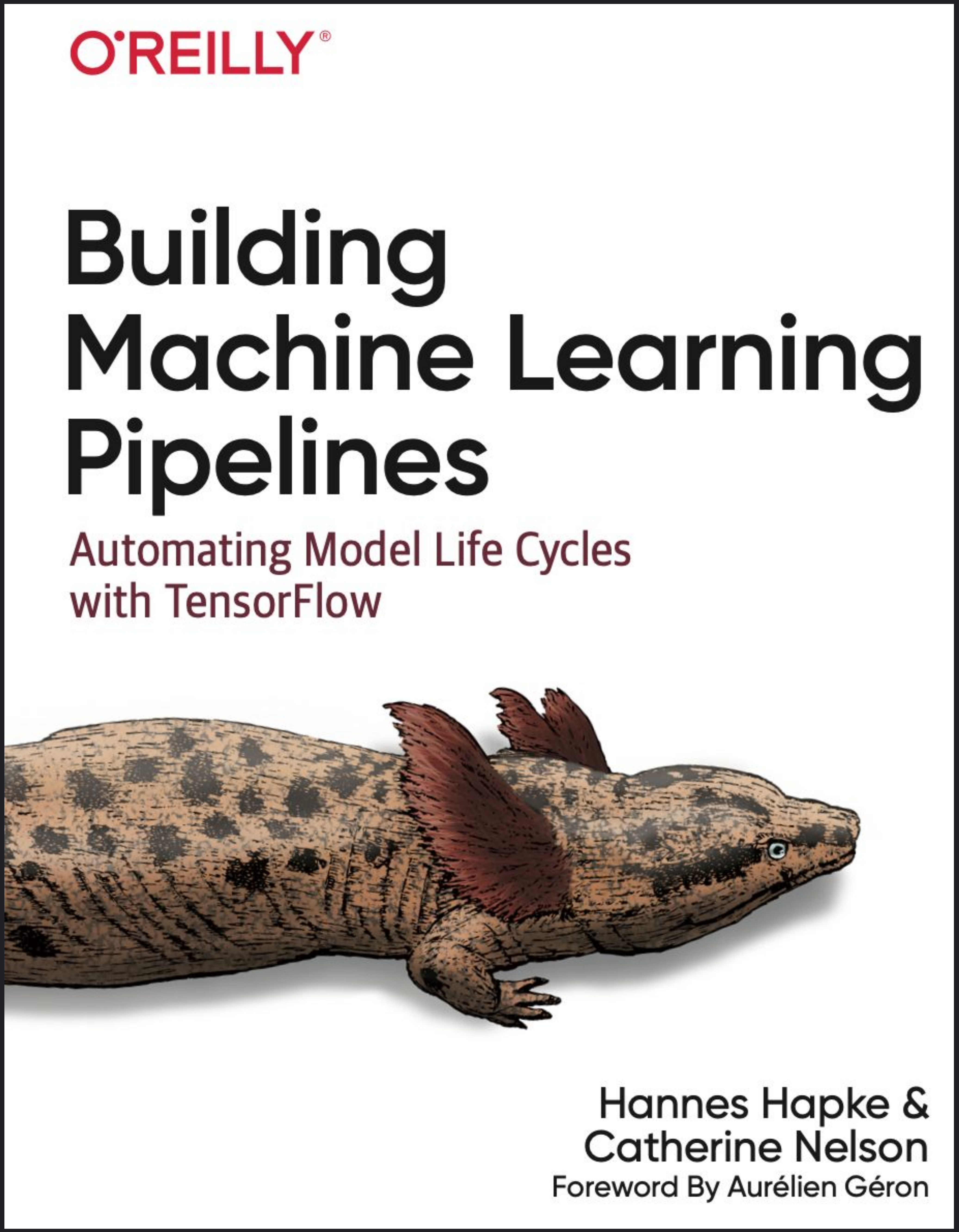
Original image: “Building Machine Learning Pipelines” - https://learning.oreilly.com/library/view/building-machine-learning/9781492053187/

Building Machine Learning Pipelines

O'Reilly Publication
available now

Incl. GCP & AWS examples

buildingmlpipelines.com



TFX Pipelines with BERT

TensorFlow Blog | Search the Blog | ← Return to TensorFlow Home

All | TensorFlow Core | TensorFlow.js | TensorFlow Lite | TFX | Swift | Community

Community · TFX

Part 1: Fast, scalable and accurate NLP: Why TFX is a perfect match for deploying BERT

March 11, 2020

Posted by Guest author Hannes Hapke, Senior Machine Learning Engineer at SAP's Concur Labs. Edited by Robert Crowe on behalf of the TFX team.

Transformer models, especially the BERT model, have revolutionized NLP and broken new ground on tasks such as sentiment analysis, entity extractions, or question-answer problems. BERT models allow data scientists to stand on the shoulders of giants. When the models have been pre-trained on large corpora by corporations, data scientists can apply transfer learning to these multi-purpose trained transformer models and achieve groundbreaking results for their

Quick Break (5 min)

Buildingmlpipelines.com

@hanneshapke

Workshop

bit.ly/ODSC-BERT-TFX

Workshop

bit.ly/ODSC-BERT-TFX

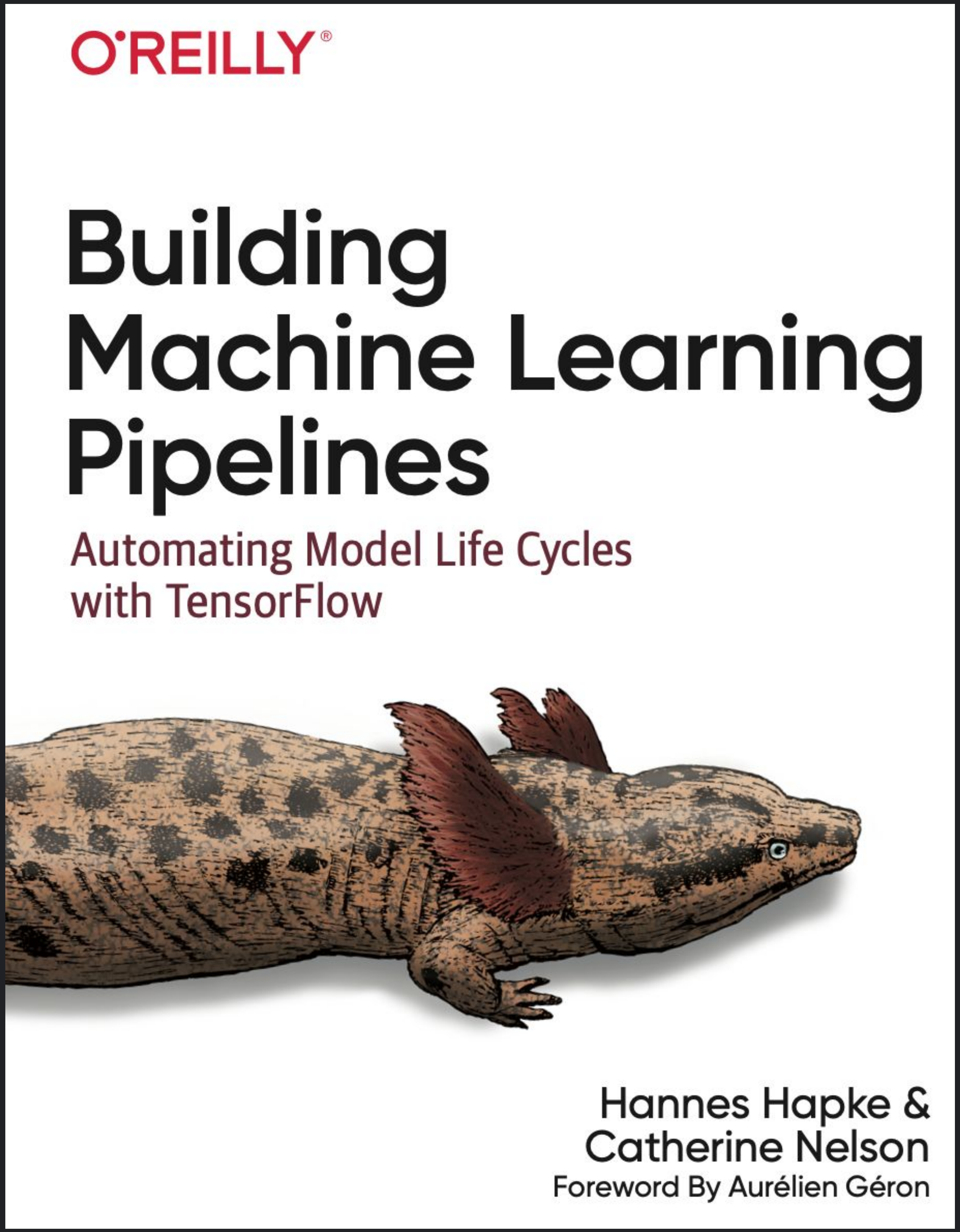
@hanneshapke

Building Machine Learning Pipelines

O'Reilly Publication
available now

Incl. GCP & AWS examples

buildingmlpipelines.com



TFX Pipelines with BERT

TensorFlow Blog

Search the Blog

← Return to TensorFlow Home

All TensorFlow Core TensorFlow.js TensorFlow Lite TFX Swift Community

Community · TFX

Part 1: Fast, scalable and accurate NLP: Why TFX is a perfect match for deploying BERT

March 11, 2020

Posted by Guest author Hannes Hapke, Senior Machine Learning Engineer at SAP's Concur Labs. Edited by Robert Crowe on behalf of the TFX team.

Transformer models, especially the BERT model, have revolutionized NLP and broken new ground on tasks such as sentiment analysis, entity extractions, or question-answer problems. BERT models allow data scientists to stand on the shoulders of giants. When the models have been pre-trained on large corpora by corporations, data scientists can apply transfer learning to these multi-purpose trained transformer models and achieve groundbreaking results for their

Q&A

Buildingmlpipelines.com
@hanneshapke

bit.ly/ODSC-BERT-TFX