Accelerating Machine Learning Predictions with NVIDIA TensorRT and Apache Beam

Shubham Krishna
ML Engineer, ML6

BEAM SUMMIT
NYC 2023

Who is ML6?

Machine Learning services company.

We help our clients build machine learning applications using technologies such as Apache Beam.

Agenda

- Motivation
- Solution
  - **Beam RunInference:** Seamless integration of ML in a Beam pipeline for semantic enrichment
  - **Nvidia TensorRT**: Accelerated + Optimized ML Inference
- Example

Motivation

- Semantic Enrichment: ML models provide semantic information.
- Increasing scale and hardware requirements of ML models.

Semantic Enrichment of Data

- Categorise: Add specific label
- Summarize
- Sentiment Analysis
- Translate
- Extract important keywords
- Image Annotation
- Image Captioning
- Speech Recognition
- .....

Problem

Seamlessly integrate ML models in a Beam pipeline for semantic enrichment of data.

Increasing scale (longer inference times) and hardware requirements of models.

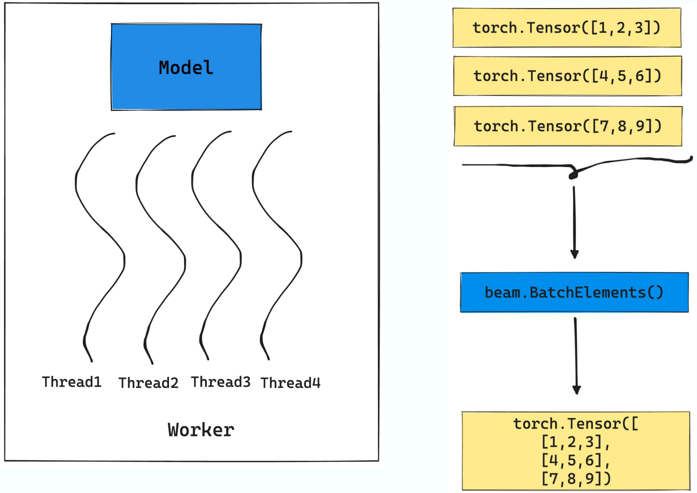
Solution

RunInference API = Inference with ML model in batch and streaming pipelines, without needing lots of boilerplate code.

Nvidia TensorRT = optimized + accelerated ML inference

RunInference >> Custom DoFn

Seamlessly integrate ML model in a Beam pipeline for semantic enrichment of data.



RunInference supports popular ML frameworks

PyTorch

TensorFlow

scikit learn

ONNX

NVIDIA TENSORRT

XGBoost

How to use RunInference?

```python
from apache_beam.ml.inference.base import RunInference
with pipeline as p:
 predictions = ( p | beam.ReadFromSource('a_source')
                   | RunInference(ModelHandler)
               )
```

ModelHandlers

```
from apache_beam.ml.inference.sklearn_inference import SklearnModelHandlerNumpy
from apache_beam.ml.inference.sklearn_inference import SklearnModelHandlerPandas
from apache_beam.ml.inference.pytorch_inference import PytorchModelHandlerTensor
from apache_beam.ml.inference.pytorch_inference import
PytorchModelHandlerKeyedTensor
model_handler = SklearnModelHandlerNumpy(model_uri='model.pkl',
 model_file_type=ModelFileType.JOBLIB)

model_handler = PytorchModelHandlerTensor(state_dict_path='model.pth',
 model_class=PytorchLinearRegression,
 model_params={'input_dim': 1, 'output_dim': 1})
```

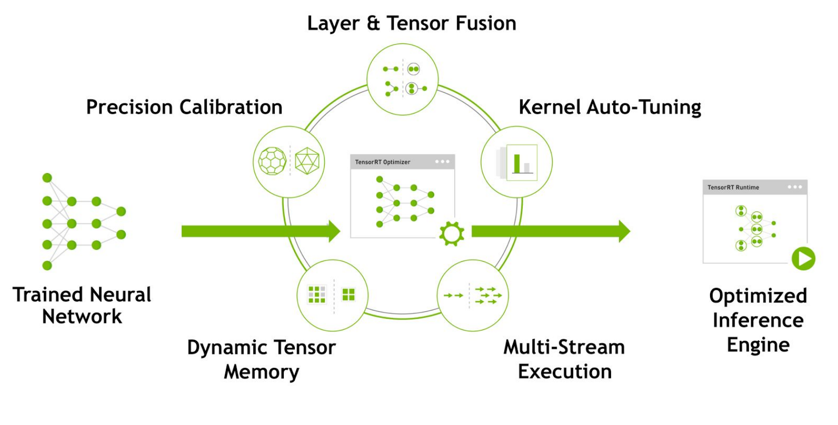
KeyedModelHandler

```
from apache_beam.ml.inference.base import
KeyedModelHandler
keyed_model_handler = \
KeyedModelHandler(PytorchModelHandlerTensor(...))

with pipeline as p:
 data = p | beam.Create([
 ('img1', np.array[[1,2,3],[4,5,6],...]),
 ('img2', np.array[[1,2,3],[4,5,6],...]),
 ('img3', np.array[[1,2,3],[4,5,6],...]),
 ])

 predictions = data | RunInference(keyed_model_handler)
```

Nvidia TensorRT

Flexible: An SDK designed to work with ONNX, TensorFlow, PyTorch, and others.

Optimizes a neural network for faster inference on NVIDIA GPUs, while preserving model accuracy.



Simplified and Accelerated Inference Pipelines

beam

RunInference API

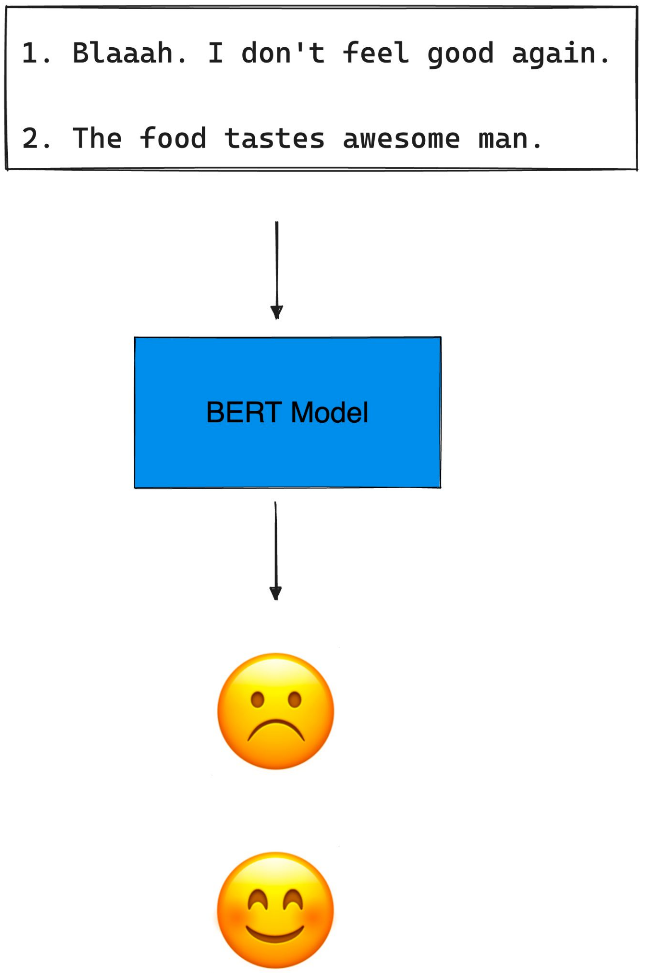
+

NVIDIA

TENSORRT

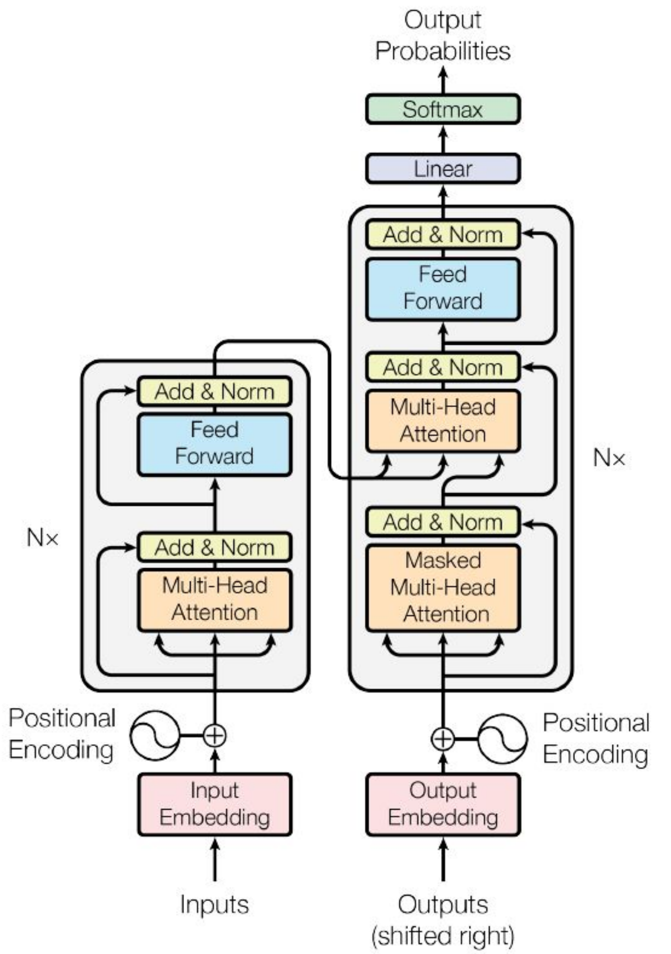
Example

Using a trained BERT-based (Transformer) text classification model for sentiment analysis in a Beam pipeline.

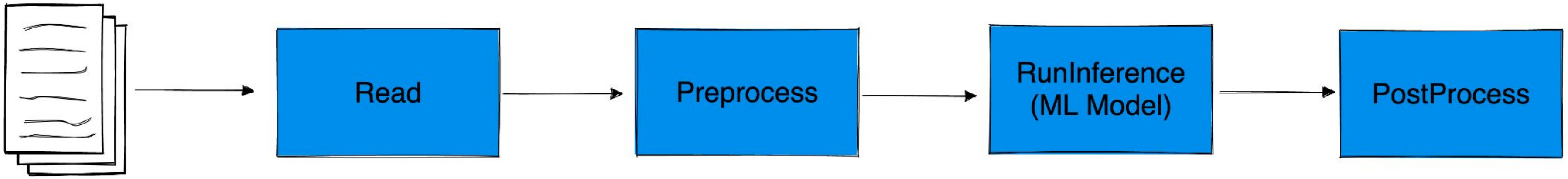


BERT

- A state-of-the-art (NLP) language model, Google.
- Can be fine-tuned for NLP tasks: text classification, named entity recognition, question answering, etc.
- textattack/bert-base-uncased-SST-2 finetuned on SST-2 for sentiment analysis.



ML Inference Pipeline in Beam as a DAG

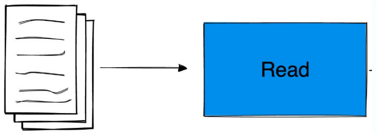


[File]

```python
with beam.Pipeline(options=pipeline_options) as pipeline:
  _ = (
      pipeline
      | "ReadSentences" >> beam.io.ReadFromText(known_args.input)
      | "Preprocess" >> beam.ParDo(Preprocess(tokenizer=tokenizer))
      | "RunInference" >> RunInference(model_handler=model_handler)
      | "PostProcess" >> beam.ParDo(Postprocess(tokenizer=tokenizer))
  )
```

Tutorial Link: [Apache Beam Documentation]()

Read Texts from File



[File]

1. Blaaah. I don't feel good again.

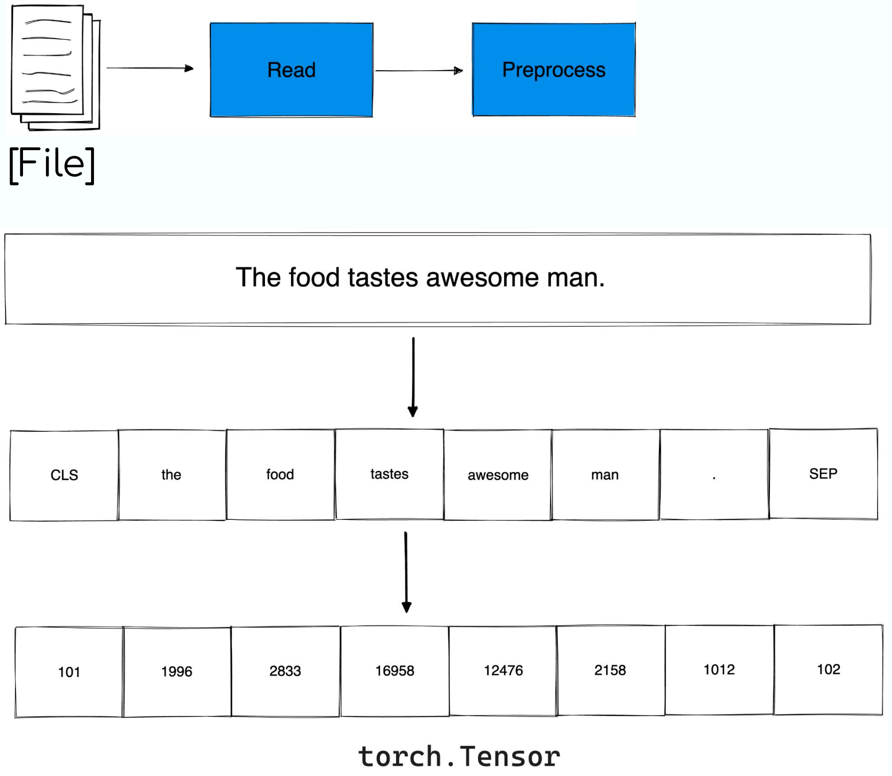
2. The food tastes awesome man.

.

.

```
with beam.Pipeline(options=pipeline_options) as pipeline:
  _ = (
      pipeline
      | "ReadSentences" >> beam.io.ReadFromText(known_args.input)
```

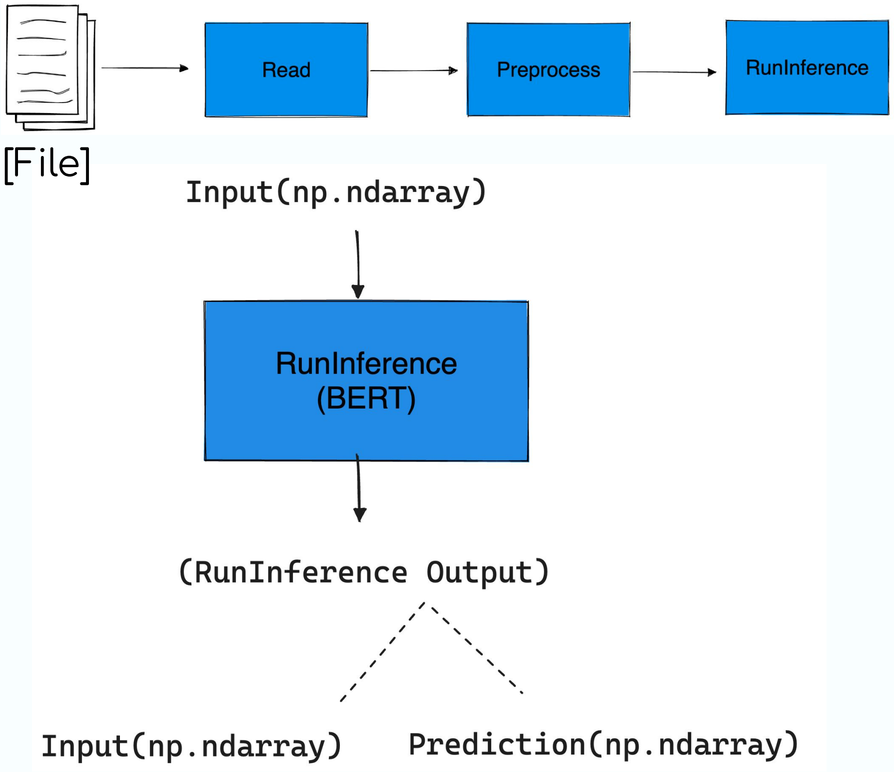
Preprocess(Tokenization)




```python
class Preprocess(beam.DoFn):
  def __init__(self, tokenizer: AutoTokenizer):
    self._tokenizer = tokenizer

  def process(self, element):
    inputs = self._tokenizer(
      element, return_tensors="np",
      padding="max_length",
      max_length=128)
    return inputs.input_ids

model_id = "textattack/bert-base-uncased-SST-2"
tokenizer = AutoTokenizer.from_pretrained(model_id)


| "Preprocess" >> beam.ParDo(Preprocess(tokenizer=tokenizer))
```

Hugging Face

TensorRT and RunInference



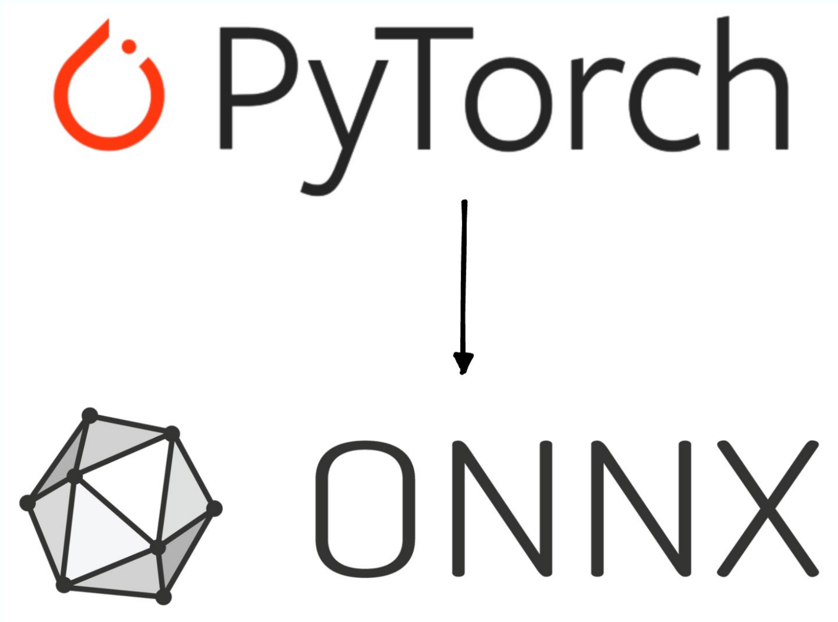

```
model_handler = TensorRTEngineHandlerNumPy(
    min_batch_size=1,
    max_batch_size=1,
    engine_path=known_args.trt_model_path,
)


| "RunInference" >> RunInference(model_handler=model_handler)
```

Hugging Face

Convert PyTorch model to a TensorRT Engine File

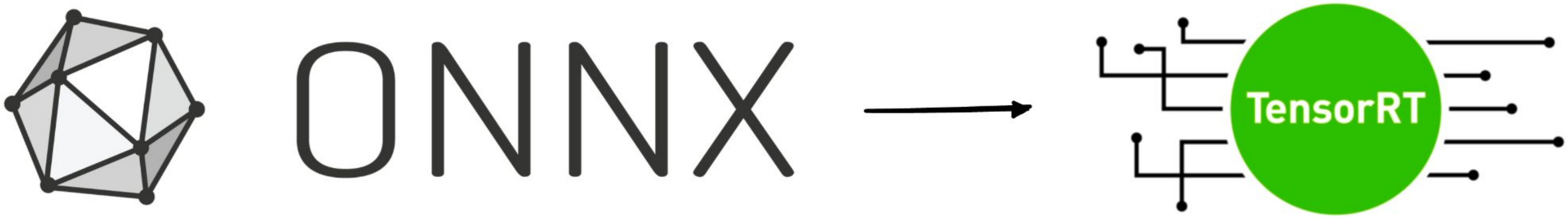
A common way to convert PyTorch model to TensorRT

PyTorch to ONNX




```python
from pathlib import Path
import transformers
from transformers.onnx import FeaturesManager
from transformers import AutoConfig
from transformers import AutoTokenizer
from transformers import AutoModelForMaskedLM
from transformers import AutoModelForSequenceClassification


# load model and tokenizer
model_id = "textattack/bert-base-uncased-SST-2"
feature = "sequence-classification"
model = AutoModelForSequenceClassification.from_pretrained(model_id)
tokenizer = AutoTokenizer.from_pretrained(model_id)

# load config
model_kind, model_onnx_config = FeaturesManager.check_supported_model_or_raise(model,
feature=feature)
onnx_config = model_onnx_config(model.config)

# export
onnx_inputs, onnx_outputs = transformers.onnx.export(
        preprocessor=tokenizer,
        model=model,
        config=onnx_config,
        opset=12,
        output=Path("bert-sst2-model.onnx")
)
```

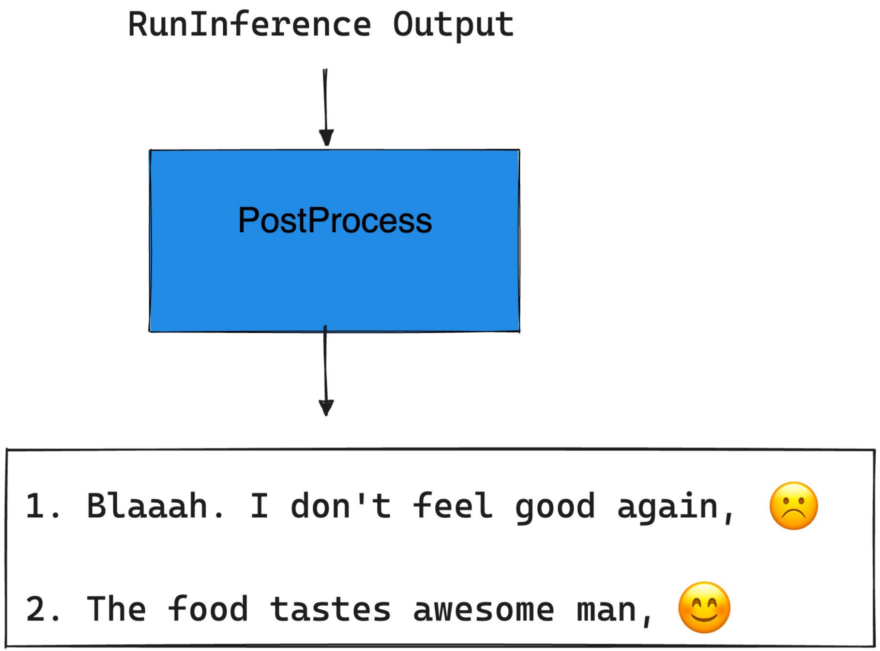
ONNX to TensorRT




```
trtexec --onnx=<path to onnx model> --saveEngine=<path to save TensorRT engine> --useCudaGraph --
verbose
```

trtexec - a command-line tool for Onnx to TensorRT Engine conversion

PostProcess RunInference Output




```python
class Postprocess(beam.DoFn):
  def __init__(self, tokenizer: AutoTokenizer):
    self._tokenizer = tokenizer

  def process(self, element):
    decoded_input = self._tokenizer.decode(
        element.example, skip_special_tokens=True)
    logits = element.inference[0]
    argmax = np.argmax(logits)
    output = "Positive" if argmax == 1 else "Negative"
    yield decoded_input, output


| "PostProcess" >> beam.ParDo(Postprocess(tokenizer=tokenizer))
```

TensorRT is 4.1x faster than PyTorch

Model	Mean Inference batch Latency (in microseconds)
PyTorch	15,176
TensorRT	3,685

Mean Inference batch Latency: Average time to perform the inference on a batch of examples.

GPU: T4, Batch-size = 1 to mimic streaming setup

Takeaways

- RunInference transform eliminates the need for extensive boilerplate code in pipelines with machine learning models.

- Beam and Nvidia TensorRT integration: Enhancing inference pipeline with improved GPU utilization, reduced production cost, and superior latency and throughput.

Code and Tutorial Link

Code: [GitHub Link]

Tutorial: [Apache Beam Documentation Link]

Slides: [GitHub Link]

Shubham Krishna

QUESTIONS?

in shubham-krishna-998922108

shub-kris

BEAM SUMMIT
NYC 2023