Streaming NLP infrastructure on Dataflow

By Alex Chan and Angus Neilson

BEAM SUMMIT Austin, 2022

Introductions

Who are we

Your presenters

- Alex Chan
  - Senior ML Engineer, ML Platform, Trustpilot
  - Background in data science, ML,
  - Likes Whisky
- Angus Neilson
  - Senior Data Engineer, Data Platform , Trustpilot
  - Data Engineer with a background in building scalable Data pipelines in many sectors. Working currently with Beam Kafka BigQuery Python Java
  - Likes Whisky

Agenda

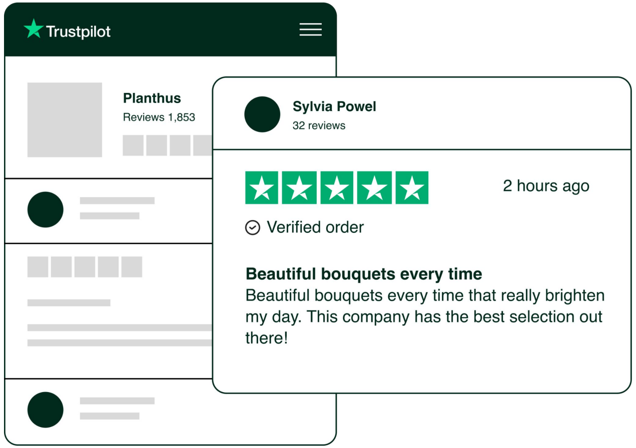
1. Trustpilot Data Platform
2. Beam Programming Model
3. GPUs on Dataflow
4. Beam for MLOps

Our mission
is to become a universal symbol of trust

Trustpilot what we do

What we do is bring consumers and companies together to continuously share, collaborate, and improve through our reviews platform.

- 46.7 M reviews were written on the Trustpilot platform globally in 2021
- Thats 21% increase compared to previous year.
- We work hard to make sure you're reading reviews based on real experiences.
- 2.7m fake reviews were removed in 2021



Trustpilot

Trustpilot Data Platform

Where we started

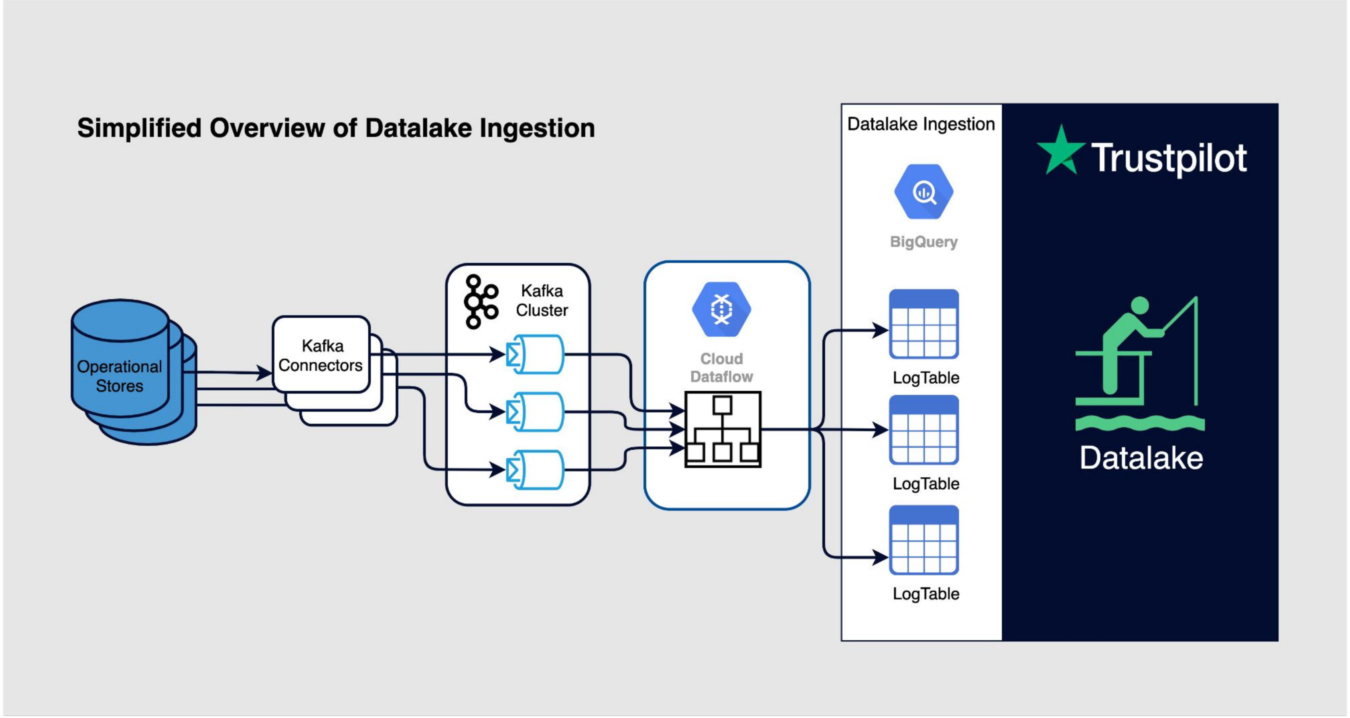
With and existing pipeline that had a few issues

- Slow to backfill
- English Language only (we were founded in Denmark 🇩🇰)
- Pre Transformer Era 🤖 (no not that one)

Where we wanted to go

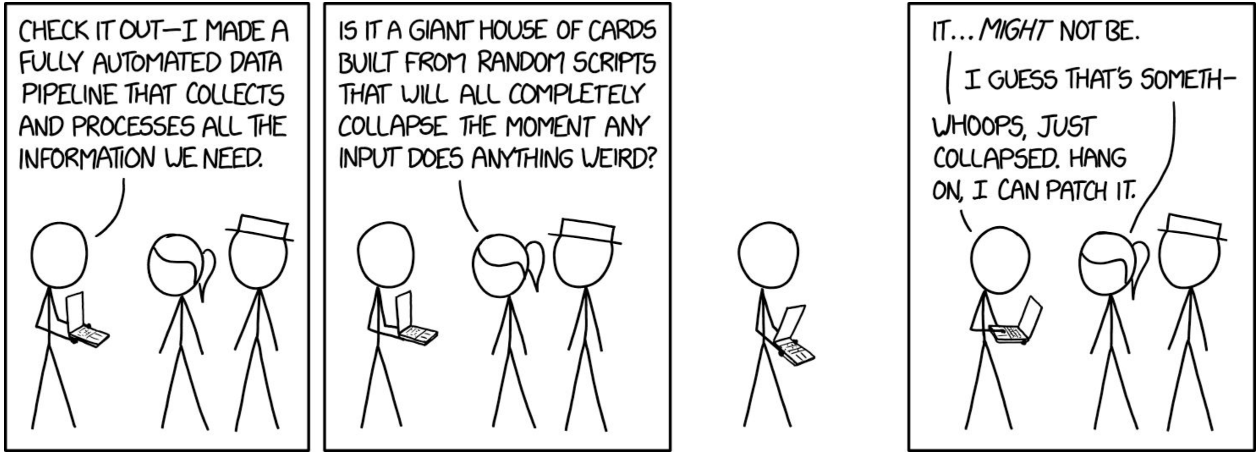
- Faster Turnaround of our models
- Ability to extend quickly - enrichments for example
- Integrate in the Datalake to support our Data Science teams

Existing Infrastructure



Data Mesh

As a Data platform Team we provide the environment enable our different contexts to easily manage and add value to our data



xkcd.com/2054

Trustpilot

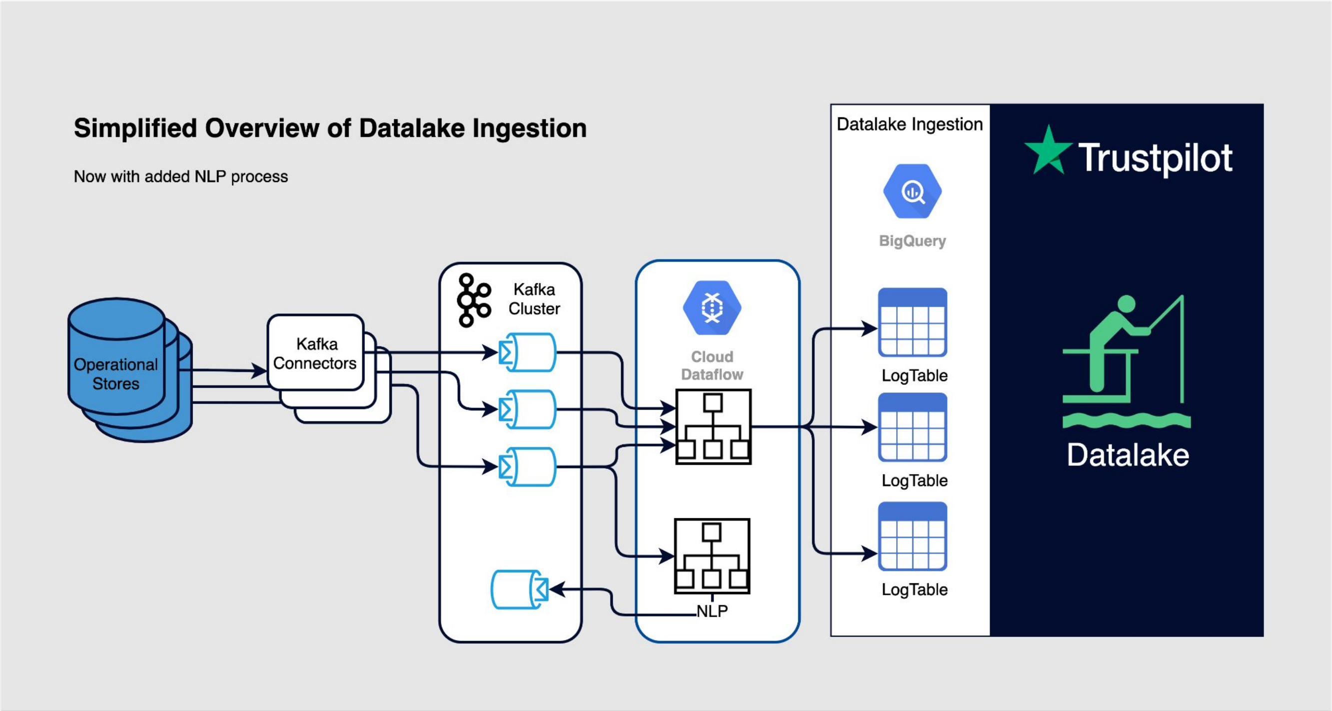
Data Mesh

“a type of data platform architecture that embraces the ubiquity of data in the enterprise by leveraging a domain-oriented, self-serve design”

– Zhamak Dehghani

Trustpilot ML Platform

Again Simplified



So never as simple as you think

Some issues we had

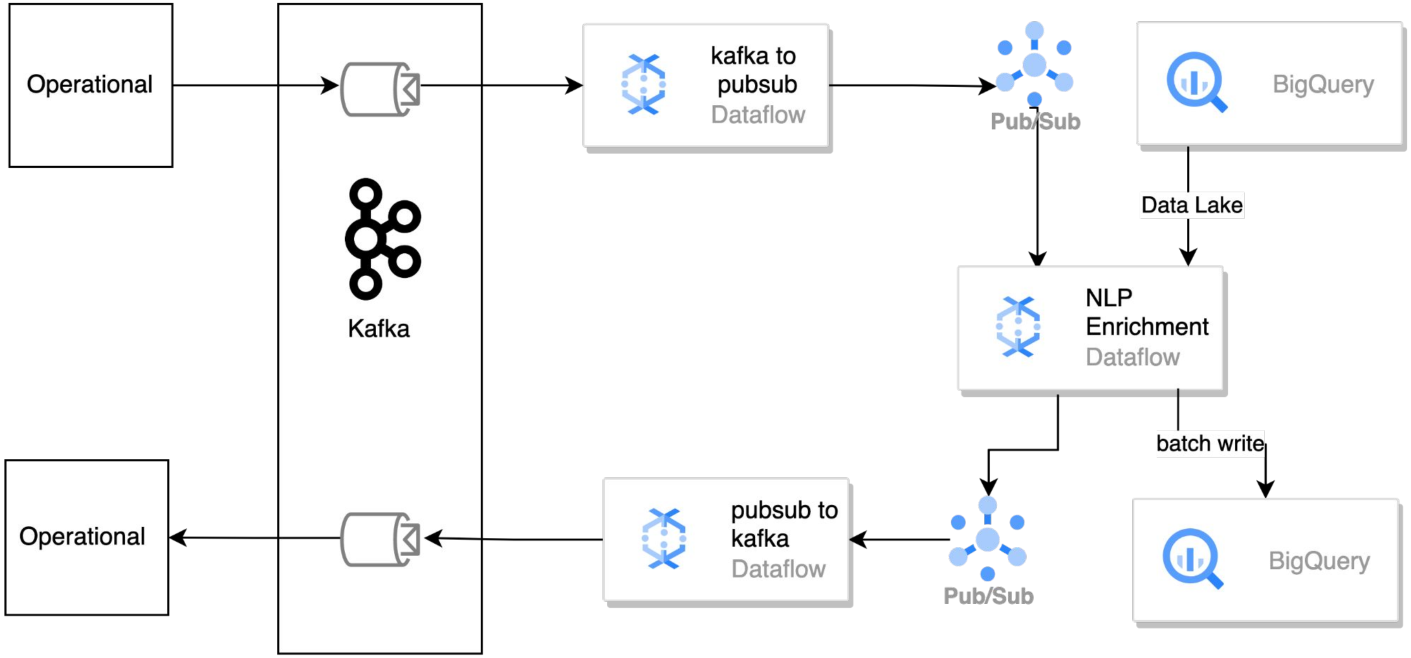
- **`KafkaIO`** - Python
- Reading the **Kafka** metadata
- Handling **Kafka** Tombstone messages (BEAM-10529)

So we altered our design

- Added a **Kafka** to **Pub/Sub** **Beam** job

Trustpilot

So we ended up here



Beam's unified programming model

Advantages of using Beam

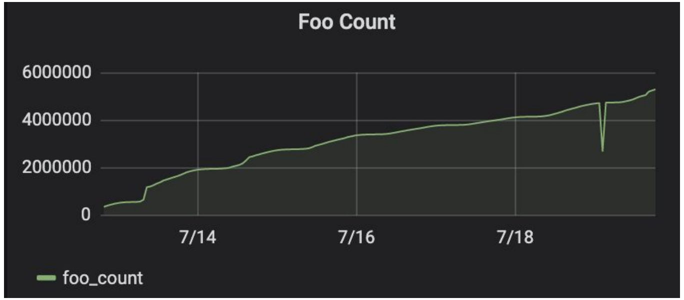
- The unified model gives flexibility for backfilling data
- Streaming
- Easy for us to use Batch for short term for the whole process.

Advantages of using Beam

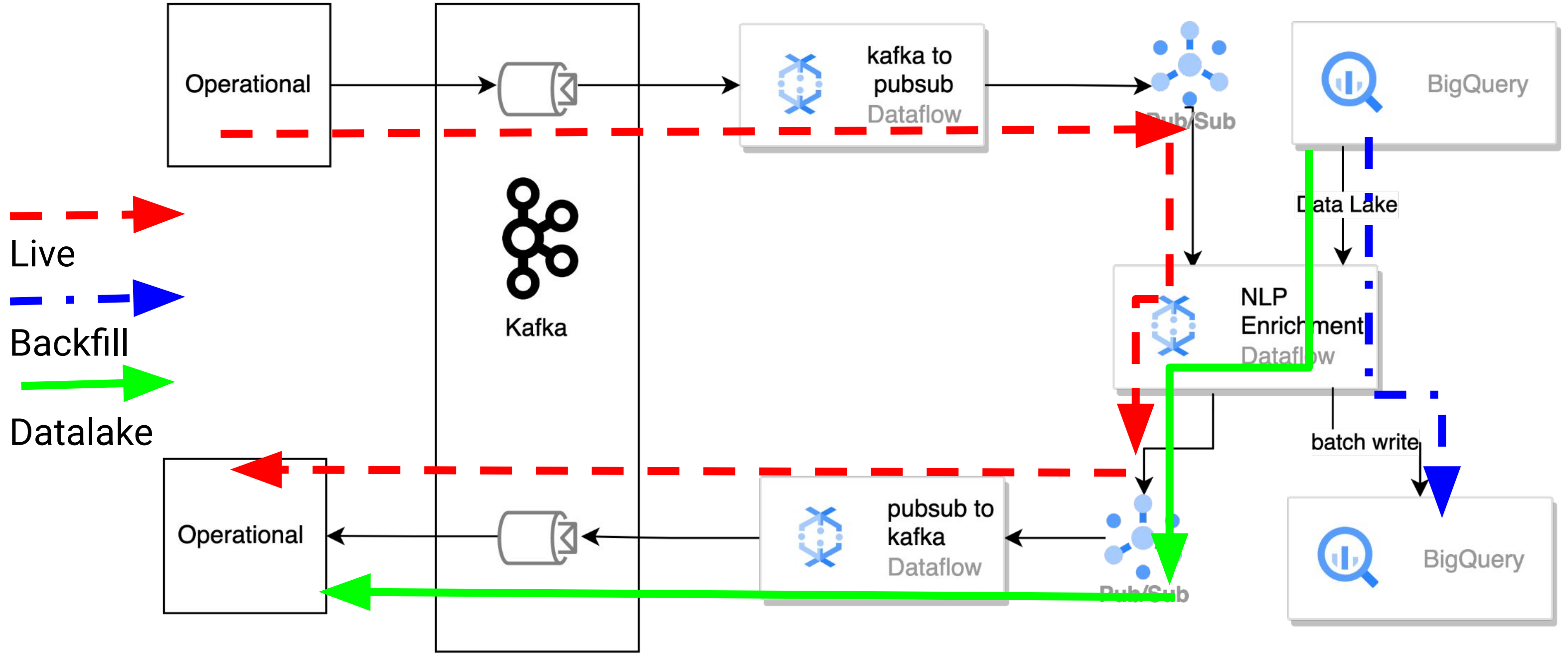
- Portable, it runs locally using the same code
- `Google Cloud Dataflow` our chosen method
- Open Source, We like open source
- Custom metrics very simple to add

```
self.foo_count = Metrics.counter("foo", "foo_count")

self.foo_count.inc()
```



The current architecture



GPUs on Dataflow

Operating GPUs on Dataflow

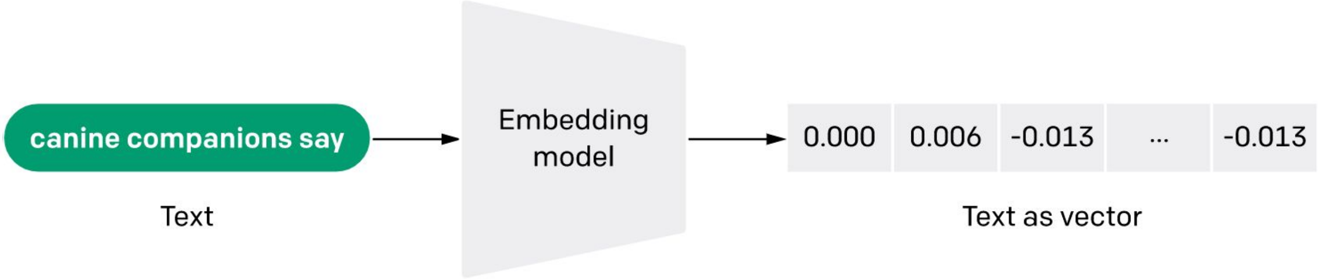
Overview

- When to use
- Our setup
- Some pitfalls

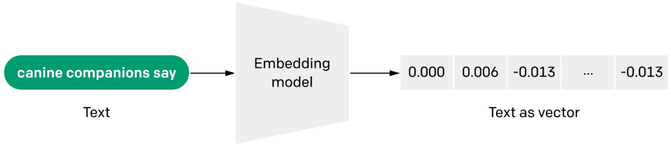
BEAM SUMMIT

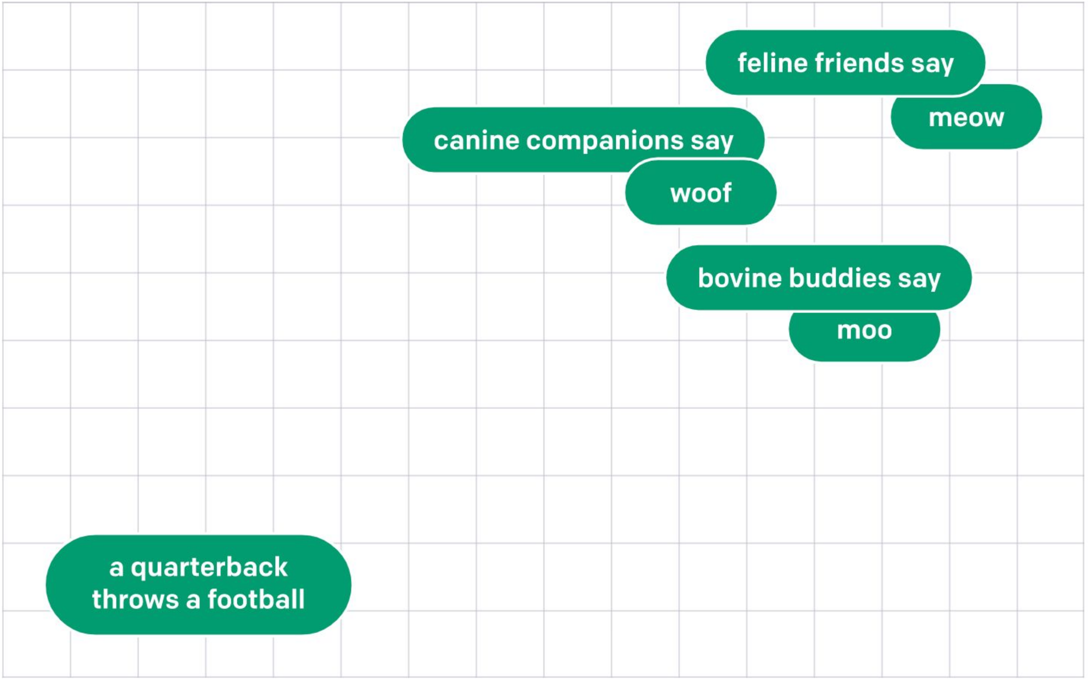
GPUs: Motivation

- Large transformer embedder model from 🤗



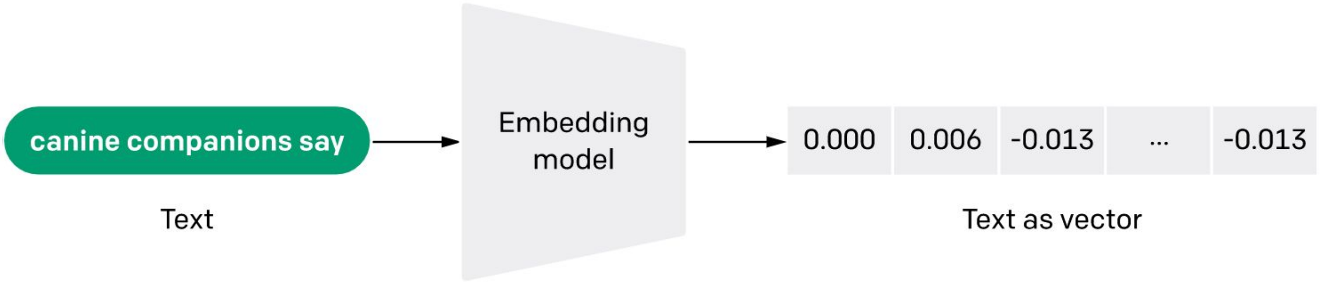
GPUs: Motivation





GPUs: Motivation

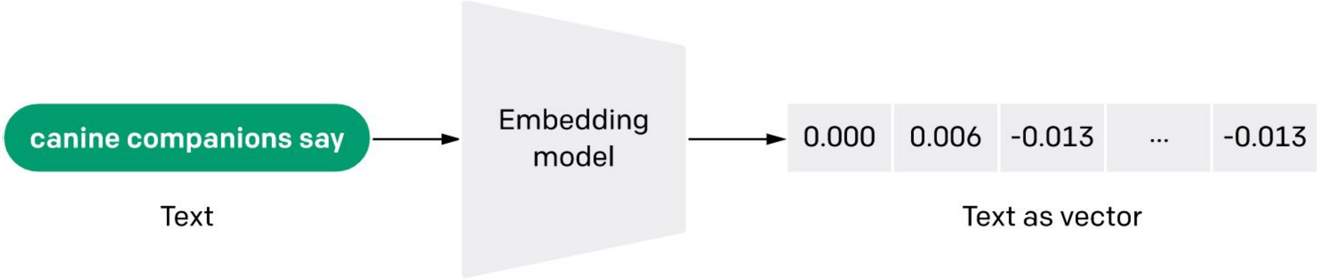
- Large transformer embedder model from 🤗



- Achieve low latency in streaming pipeline
- Enable quick model releases with batch pipelines

GPUs: Motivation

- Large transformer embedder model from 🤗



- Achieve low latency in streaming pipeline
- Enable quick model releases with batch pipelines
- Doing local inference vs remote service call

GPUs: Common pitfalls

- Conflicts with Beam's parallelism:
    - Worker 😌
    - vCPU/Docker runtime 😬
    - Thread 😱

GPUs: Common pitfalls

```python
from apache_beam.utils import shared


(p
| ...
| beam.ParDo(Embed(shared.Shared()))
| ...
)
```

- Use a shared API to set an object to be shared

GPUs: Common pitfalls

```python
class Embed(beam.DoFn):
    def load_model(self)

        def initialize_model():
            model = Transformer(self.model_path)
            return WeakRefModel(model)

        self.model = self.shared_handle.acquire(initialize_model)
```

- Use a shared API to set an object to be shared

GPUs: Common pitfalls

- Multiple Docker runtimes on a worker
- Use custom instance sizes to limit to 1x vCPU
- Set `--no_use_multiple_sdk_containers` `experiment` flag

GPUs: Common pitfalls

```
model = Transformer(self.model_path)

# model is "lazy-loaded" to CPU only at this point, model is
# only placed on GPU when running the first inference
_ = model.encode(
    [
        "IN PRINCIPIO ERAT VERBUM",
        "ET VERBUM CARO FACTUM EST",
        "ET HABITAVIT IN NOBIS",
    ]
    * 3
)
```

- Run a dummy inference to place onto GPU

GPUs: Common pitfalls

```python
import threading
lock = threading.Lock()
...

class Embed(beam.DoFn):
    def __init__(self, lock):
        self.lock = lock

    def process(self, elem):
        ...
        with self.lock:
            return self._weakRef.model.encode(
                sentences=sentences,
            ).tolist()
```

- Use a thread lock to limit access to shared resource

GPUs: Common pitfalls

Worker 😌

- safe

vCPU/Docker runtime 😌

- shared objects
- set **`--no_use_multiple_sdk_containers`** experiment flag

Thread 😌

- shared objects
- set **`--number_of_worker_harness_threads=1`** pipeline option
- dummy model initialisation
- lock

GPUs: Checklist

- Custom image
  - debug on GCP with VM instance
- Model loading/inference
  - beware of when parallelism can cause problems
- Run benchmarks
  - find optimal batch size, memory, device
- Flex templates
  - build custom image, `CI/CD` with `Cloud Build`
- Store model artifacts in GCS
  - avoid hitting `HuggingFace` public repository limits

Beam for MLOps

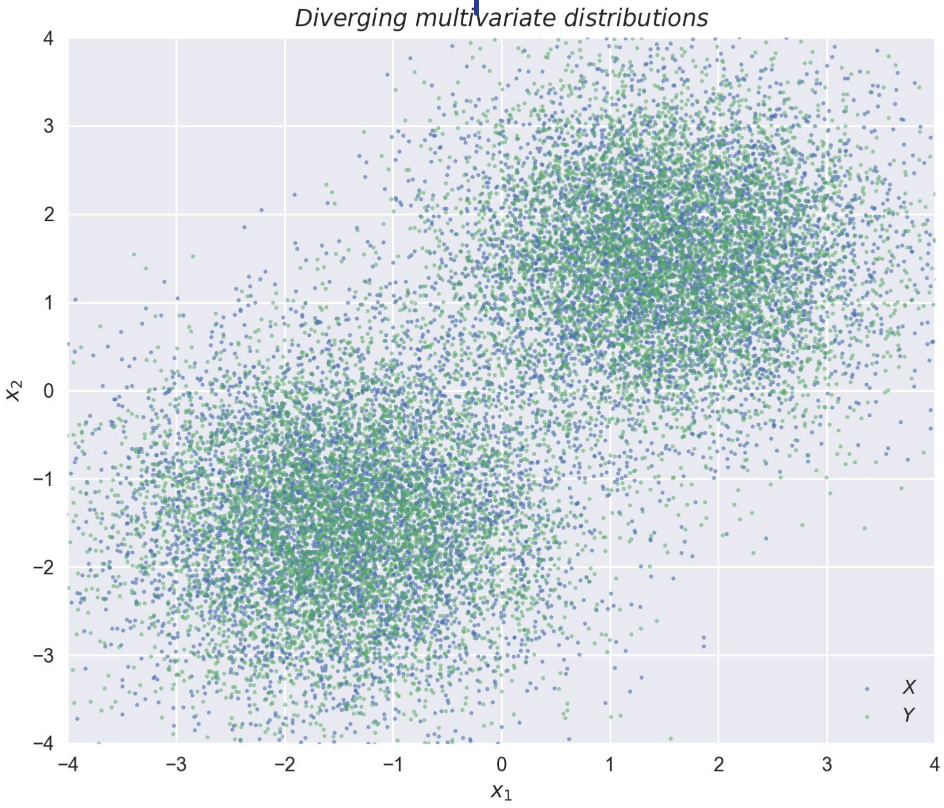
Drift detection with Beam

Overview

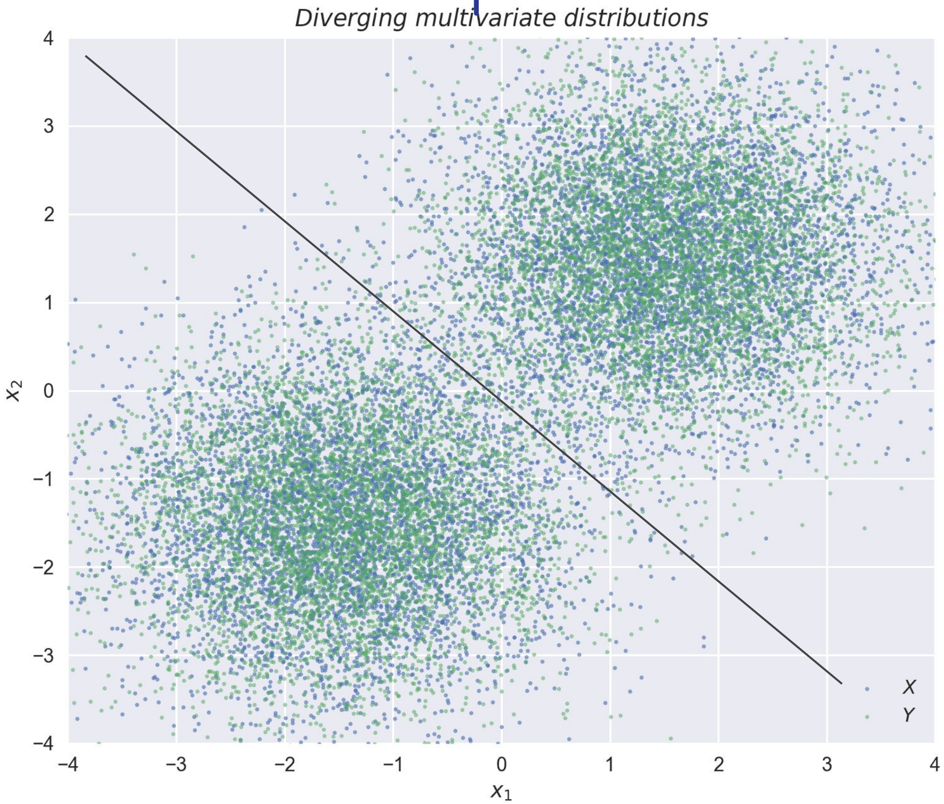
- Drift detection
- Math(s)
- Accelerating with JAX
- Results

BEAM SUMMIT

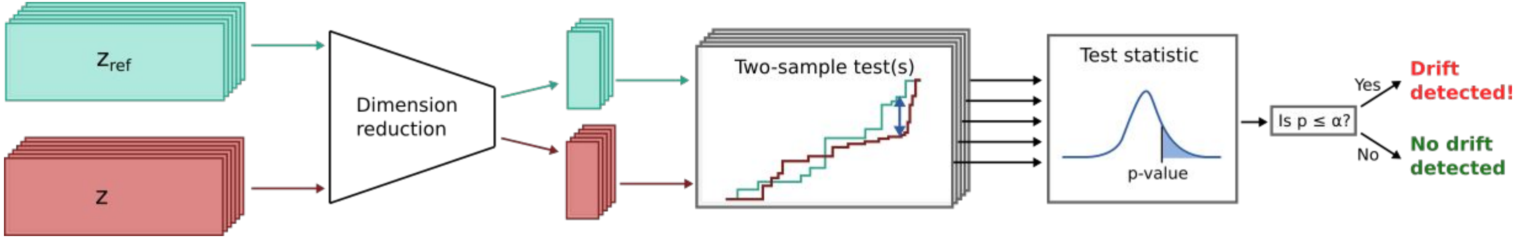
Drift detection: Example



Drift detection: Example



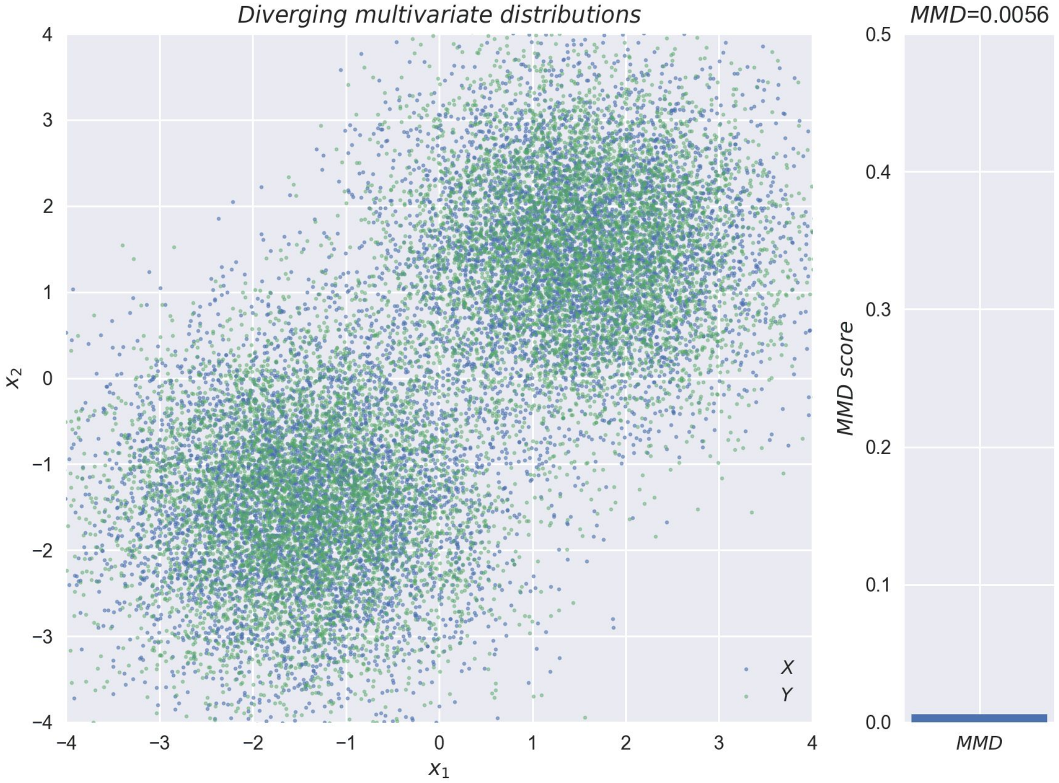
Drift detection: Approach



Max. Mean Discrepancy: Introduction

- A two-sample test statistic
- Multivariate data
- Measure pairwise distance between two data points
- Kernel method — Pick an appropriate kernel function
    - e.g. kernels for strings, image, audio, graph, tree data
- Evaluate generative models
    - measure distribution of generated data to real data

MMD: Example


Diverging multivariate distributions
MMD=0.0056
4
3
2
1
0
−1
−2
−3
−4
−4 −3 −2 −1 0 1 2 3 4
x_1
x_2
X
Y
0.5
0.4
0.3
0.2
0.1
0.0
MMD score
MMD

MMD: Challenges

① Scale of data

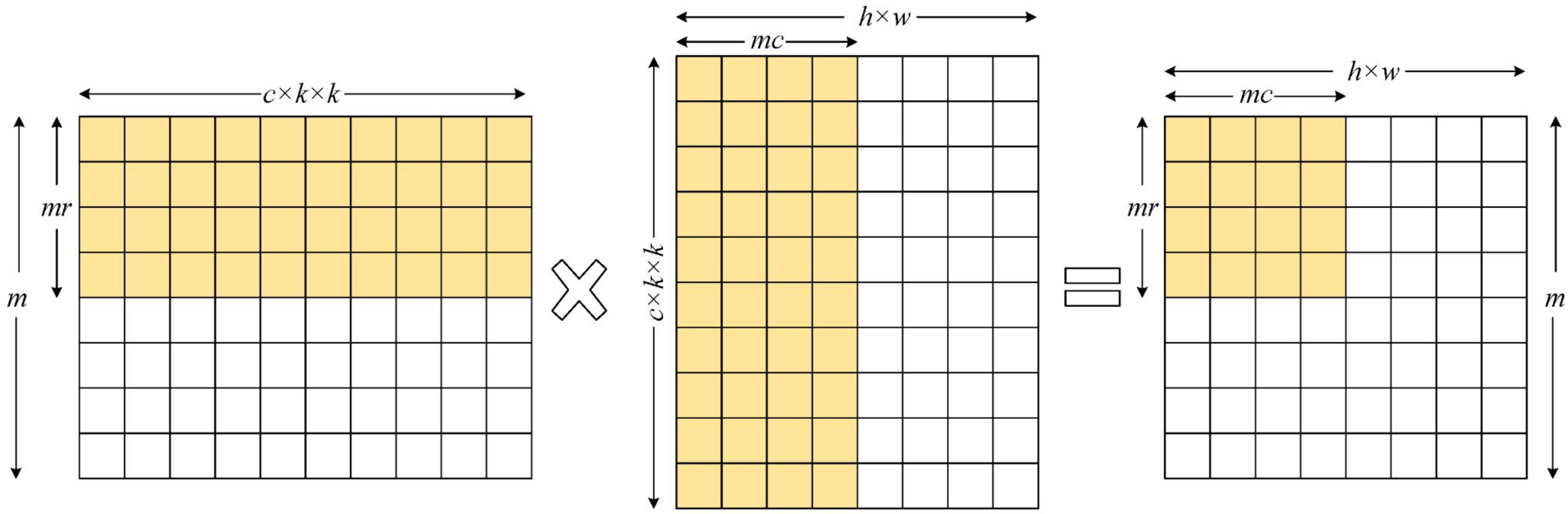
② Matrix operations slow, scale poorly

③ Hyperparameter/Kernel search

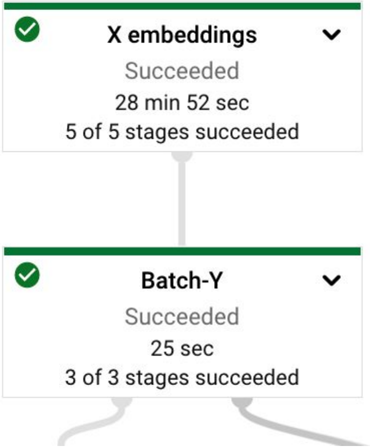
MMD: ① Scale of data

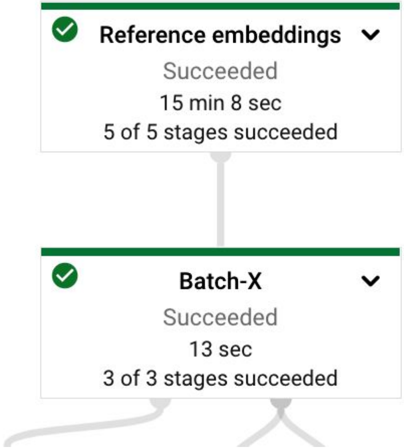
- Take advantage of **Beam** parallelism
- Implement parallel linear algebra
- Chunk matrix into submatrices

MMD: ① Scale of data

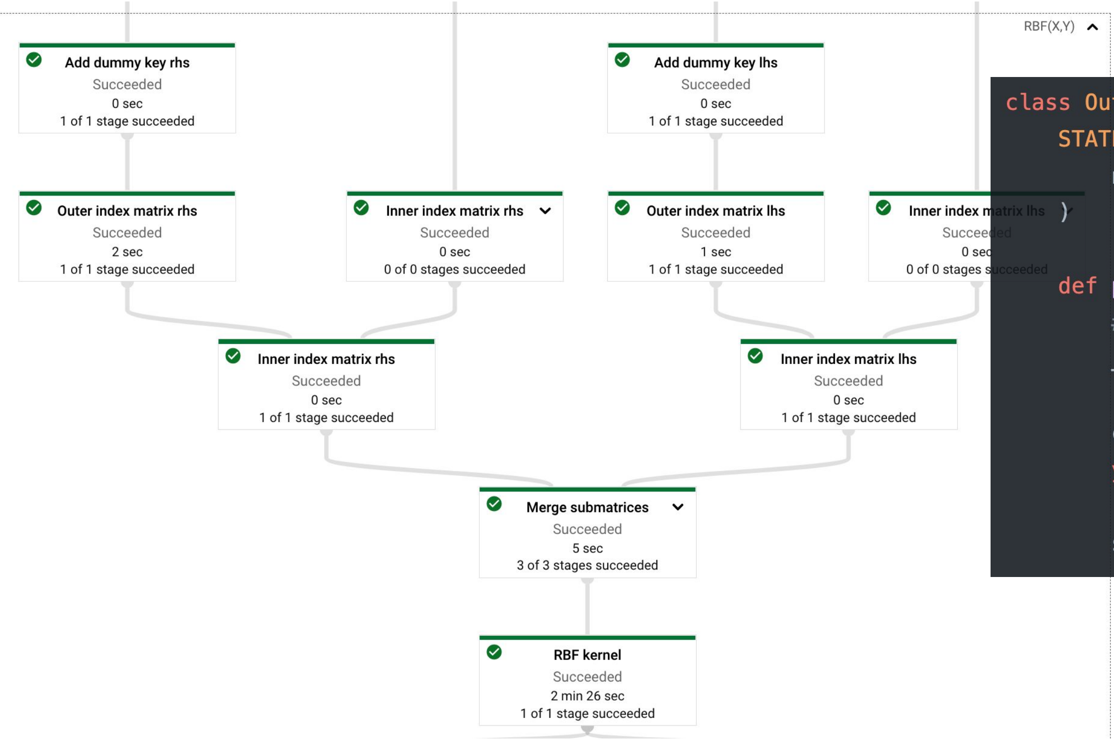


MMD: ① Scale of data





MMD: ① Scale of data




```python
class Outer(beam.DoFn):
    STATE_SPEC = beam.transforms.userstate.ReadModifyWriteStateSpec(
        name="count", coder=beam.coders.VarIntCoder()
    )

    def process(self, elem, state=beam.DoFn.StateParam(STATE_SPEC)):
        # discard dummy key
        _, elem = elem

        count = state.read() or 0
        yield count, np.vstack(elem)

        state.write(count + 1)
```

MMD: ② Matrix operations scale poorly

sklearn.metrics.pairwise.rbf_kernel

```
sklearn.metrics.pairwise.rbf_kernel(X, Y=None, gamma=None)
```

Compute the rbf (gaussian) kernel between X and Y:

```
K(x, y) = exp(-gamma ||x-y||^2)
```

for each pair of rows x in X and y in Y.

- 1M to 1M comparison would take 3.5h
- **JAX**

JAX: Overview

- Numerical computation library in **Python**
  - compile to an intermediate representation for **XLA**
  - Drop-in replacement for **NumPy**
  - accelerate vector operations on G/TPU
- Autodiff. for native **Python** functions
- Excellent package for scientific computing
  - working with arrays, matrices, linear algebra, gradients
- We are using it to help scale drift detection computations within **Beam**

JAX: vs NumPy

```python
import numpy as np

def euclidean_dist_np(X, Y):
    squared_diffs =  np.power(X[:,None] - Y, 2)
    summed  =  np.sum(squared_diffs, axis=-1)
    return  np.sqrt(summed)
```

JAX: vs NumPy

```
from jax import numpy as jnp

def euclidean_dist_ (X,Y):
    squared_diffs = jnp.power(X[:,None] - Y, 2)
    summed  = jnp.sum(squared_diffs, axis=-1)
    return jnp.sqrt(summed)

euclidean_dist_jax = jit(euclidean_dist_)
```

JAX: vs NumPy

```
X = np.random.normal(0,1,(int(1e4),5))
Y = np.random.normal(0,1,(int(1e4),5))

%%timeit
euclidean_dist_np(X,Y)
```

numpy:

JAX: vs NumPy

```
X = np.random.normal(0,1,(int(1e4),5))
Y = np.random.normal(0,1,(int(1e4),5))
```

```
        %%timeit
numpy:  euclidean_dist_np(X,Y)
        10 loops, best of 5: 8.02 s per loop
```

JAX: vs NumPy

```
X = np.random.normal(0,1,(int(1e4),5))
Y = np.random.normal(0,1,(int(1e4),5))
```

```
        %%timeit
numpy:  euclidean_dist_np(X,Y)
        10 loops, best of 5: 8.02 s per loop
```

```
        %%timeit
  jax:  euclidean_dist_jax(X,Y)
```

JAX: vs NumPy

```
X = np.random.normal(0,1,(int(1e4),5))
Y = np.random.normal(0,1,(int(1e4),5))

        %%timeit
numpy:  euclidean_dist_np(X,Y)
        10 loops, best of 5: 8.02 s per loop

        %%timeit
  jax:  euclidean_dist_jax(X,Y)
        100 loops, best of 5: 4.71 ms per loop
```

JAX: vs NumPy

```
X = np.random.normal(0,1,(int(1e4),5))
Y = np.random.normal(0,1,(int(1e4),5))
```

numpy:

```
%%timeit
euclidean_dist_np(X,Y)
10 loops, best of 5: 8.02 s per loop
```

1700x!

jax:

```
%%timeit
euclidean_dist_jax(X,Y)
100 loops, best of 5: 4.71 ms per loop
```

JAX: vs NumPy

```
X = np.random.normal(0,1,(int(1e4),5))
Y = np.random.normal(0,1,(int(1e4),5))
```

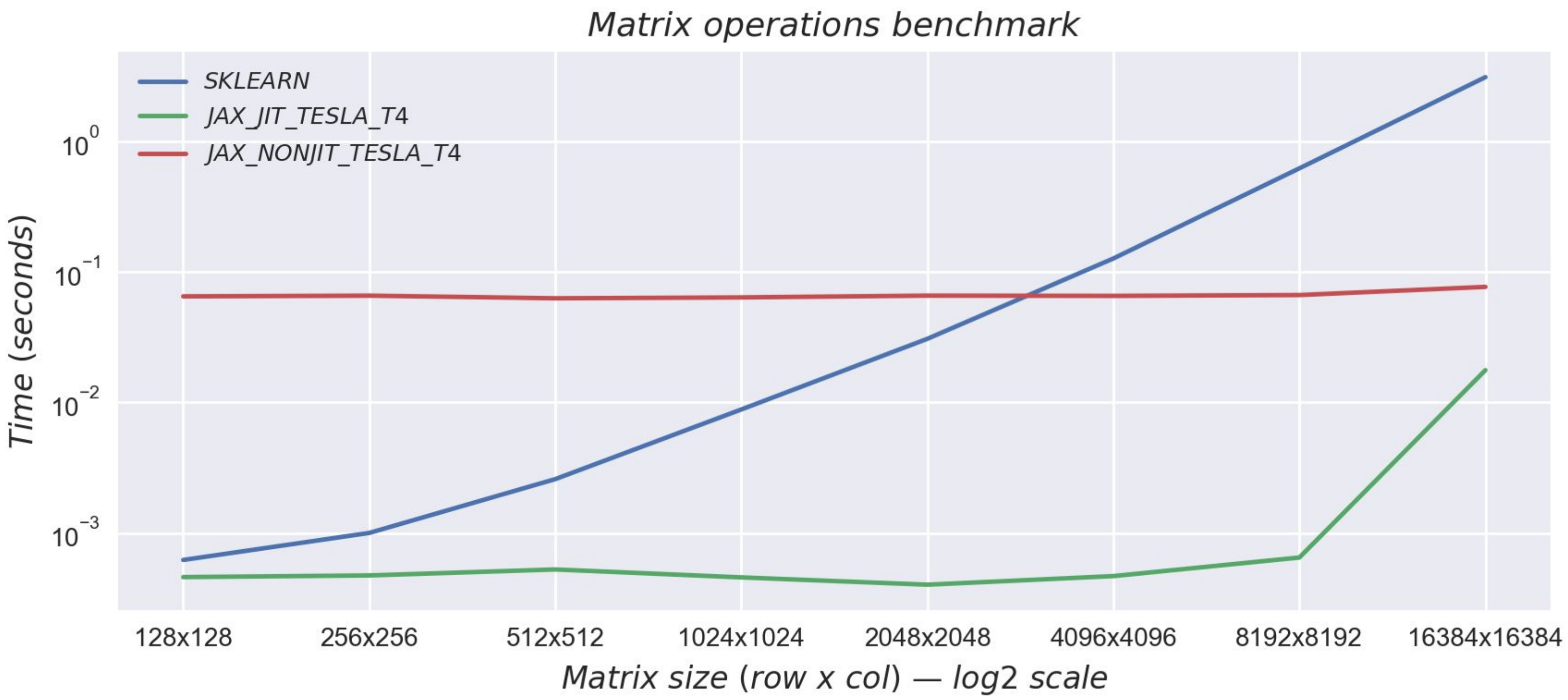
```
        %%timeit
numpy:  euclidean_dist_np(X,Y)
        10 loops, best of 5: 8.02 s per loop
```

```
        %%timeit
  jax:  euclidean_dist_jax(X,Y)
        100 loops, best of 5: 4.71 ms per loop
```

1700x!

30x (cpu)

JAX: Benchmark



JAX: **vmap** and **pmap**

- Automatic vectorisation with **vmap**
- Device parallelism with **pmap**
    - Easily dispatch operations to multiple accelerator devices
    - Dataflow supports multiple GPUs per worker

JAX: in Beam

- Define a pure function

```
class RBFKernel(beam.DoFn):


    @staticmethod
    def rbf_(X, Y, gamma):
        def distance(X, Y):
            return
jnp.sqrt(jnp.sum(jnp.power(X[:, None] - Y, 2),
axis=-1))

        d = distance(X, Y)
        return jnp.exp(-gamma * d**2)
```

JAX: in Beam

- Define a pure function
- **JIT** Compile in **DoFn** `__init__` method

```
class RBFKernel(beam.DoFn):
    def __init__(self):
        self.rbf_jax = jit(self.rbf_)

    @staticmethod
    def rbf_(X, Y, gamma):
        def distance(X, Y):
            return jnp.sqrt(jnp.sum(jnp.power(X[:, None] - Y, 2), axis=-1))

        d = distance(X, Y)
        return jnp.exp(-gamma * d**2)
```

JAX: in Beam

- Define a pure function
- **JIT** Compile in **DoFn** `__init__` method
- Dispatch to GPU for speedup

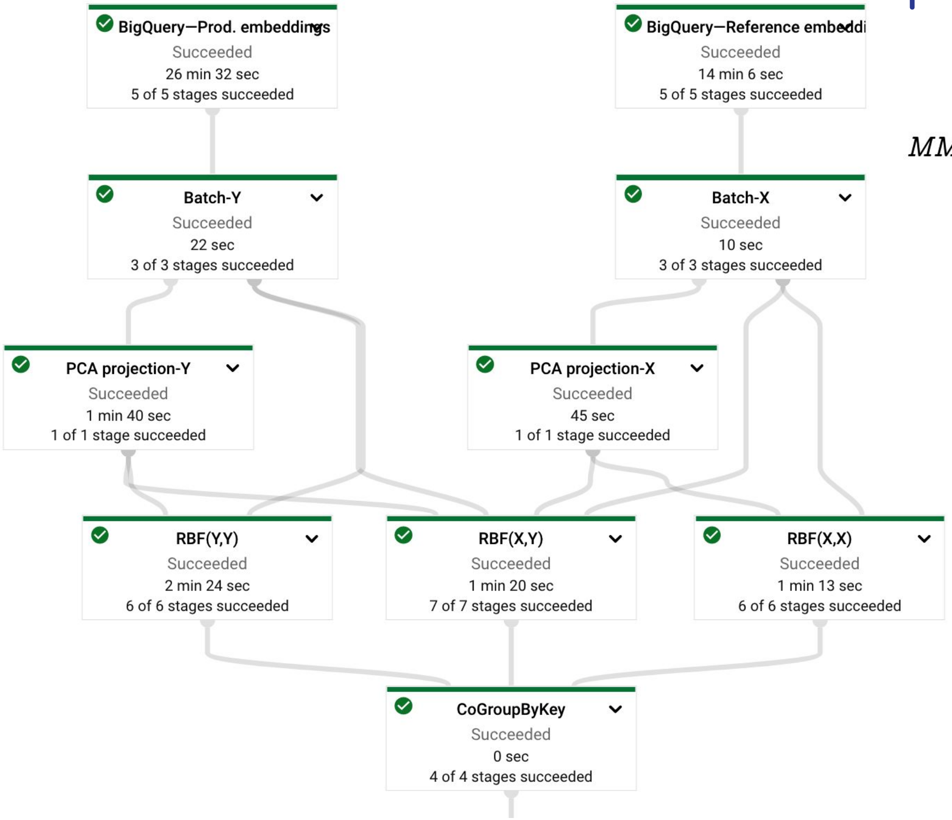
```
class RBFKernel(beam.DoFn):
    def __init__(self):
        self.rbf_jax = jit(self.rbf_)

    @staticmethod
    def rbf_(X, Y, gamma):
        def distance(X, Y):
            return
jnp.sqrt(jnp.sum(jnp.power(X[:, None] - Y, 2),
axis=-1))

        d = distance(X, Y)
        return jnp.exp(-gamma * d**2)

    def process(self, elem):
        ...
        yield key, self.rbf_jax(X, Y, gamma)
```

Drift detection: Beam pipeline

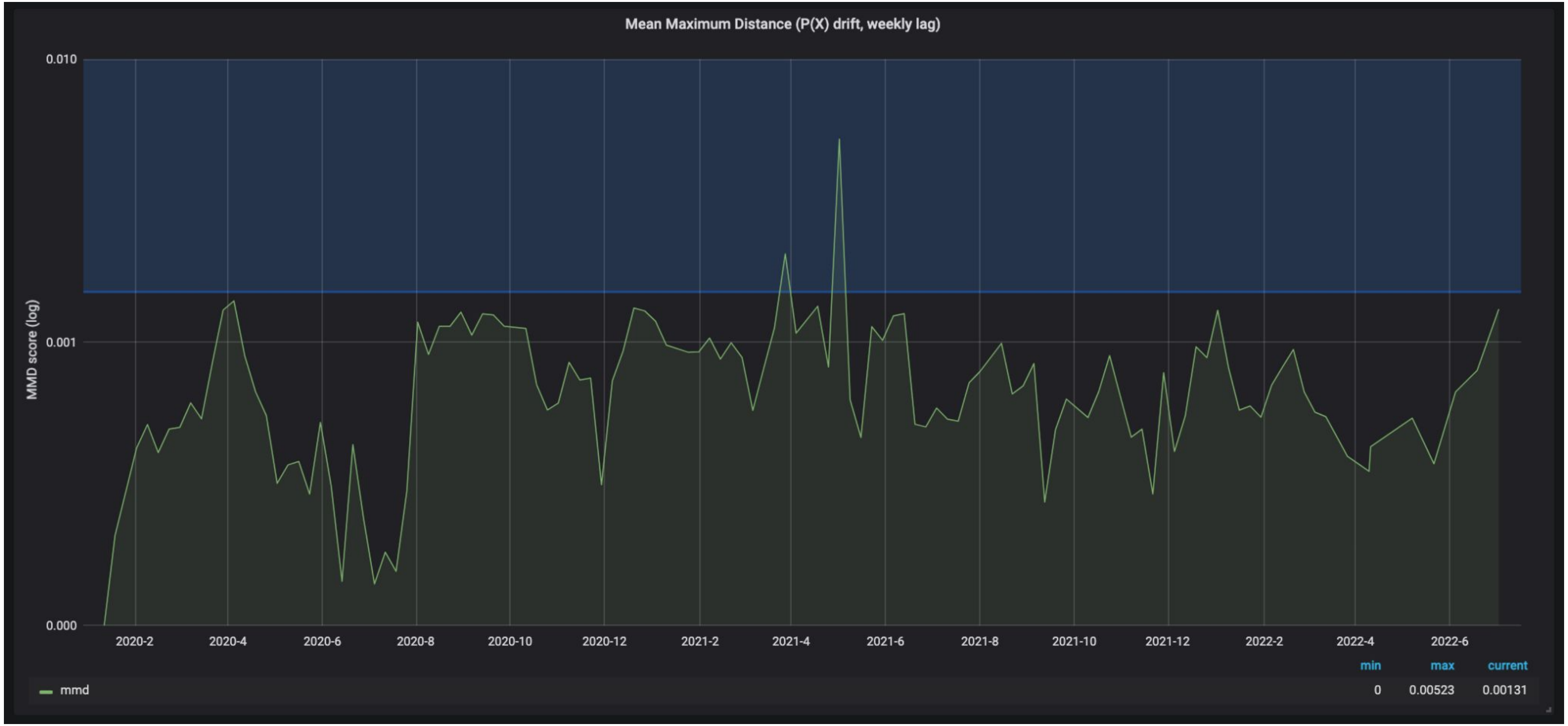


$$
\begin{aligned}
MMD^2(P,Q) &= \|\mu_P - \mu_Q\|_{\mathcal{F}}^2 \\
&= \langle \mu_P, \mu_P \rangle_{\mathcal{F}} + \langle \mu_Q, \mu_Q \rangle_{\mathcal{F}} - 2\langle \mu_P, \mu_Q \rangle_{\mathcal{F}} \\
&= \underbrace{\mathbf{E}_P k(X, X')}_{(a)} + \underbrace{\mathbf{E}_Q k(Y, Y')}_{(a)} - \underbrace{2\mathbf{E}_{P,Q} k(X, Y)}_{(b)}
\end{aligned}
$$

Drift detection: Beam pipeline

Step name	Status	Wall time	
Reference embeddings	Succeeded	13 minutes	I/O
Batch-X	Succeeded	9 seconds	
X embeddings	Succeeded	24 minutes	I/O
Batch-Y	Succeeded	17 seconds	
PCA projection-Y	Succeeded	1 minute	Matrix multiplication
PCA projection-X	Succeeded	40 seconds	Matrix multiplication
RBF(X,Y)	Succeeded	1 minute	Kernel function
RBF(Y,Y)	Succeeded	1 minute	Kernel function
RBF(X,X)	Succeeded	55 seconds	Kernel function
CoGroupByKey	Succeeded	0 seconds	
ParDo(MaximumMeanDiscrepancy)	Succeeded	0 seconds	
WriteMetrics	Succeeded	1 second	I/O

Drift detection: Results



Drift detection: Monitoring

Drift detection: Next steps

- Hypothesis testing
- Alternative parallel matrix algorithms
- Online drift

Conclusion

- Python + Kafka
- Batch + Streaming
- GPUs on Dataflow
- Statistical drift detection on Beam

Trustpilot

We are recruiting !!

business.trustpilot.com/jobs

BEAM SUMMIT

Questions?

Maybe some contact info here?
| @AngusNeilson1
trustpilot.com/jobs

Further reading

- Trustpilot Transparency Report 2022 trustpilot-transparency-report-uk-2022.pdf
- Data Mesh https://martinfowler.com/articles/data-mesh-principles.html (Zhamak Dehghani)
- Domain Oriented https://martinfowler.com/bliki/BoundedContext.html (Martin Fowler)

Trustpilot