# BEAM
SUMMIT

# Optimizing ML Workloads on Dataflow

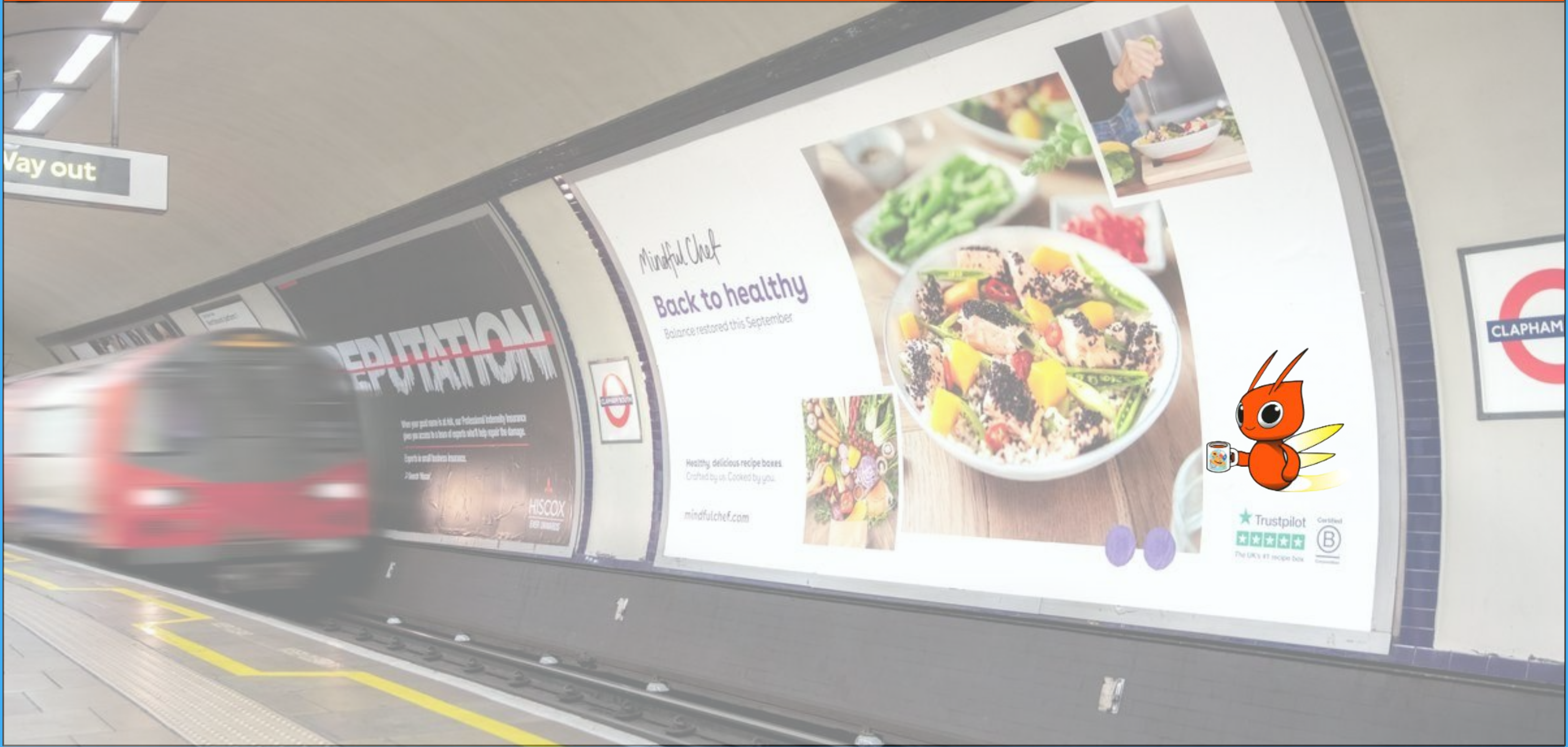
Alex Y. Chan
Trustpilot

BEAM SUMMIT
NYC 2023

# Trustpilot

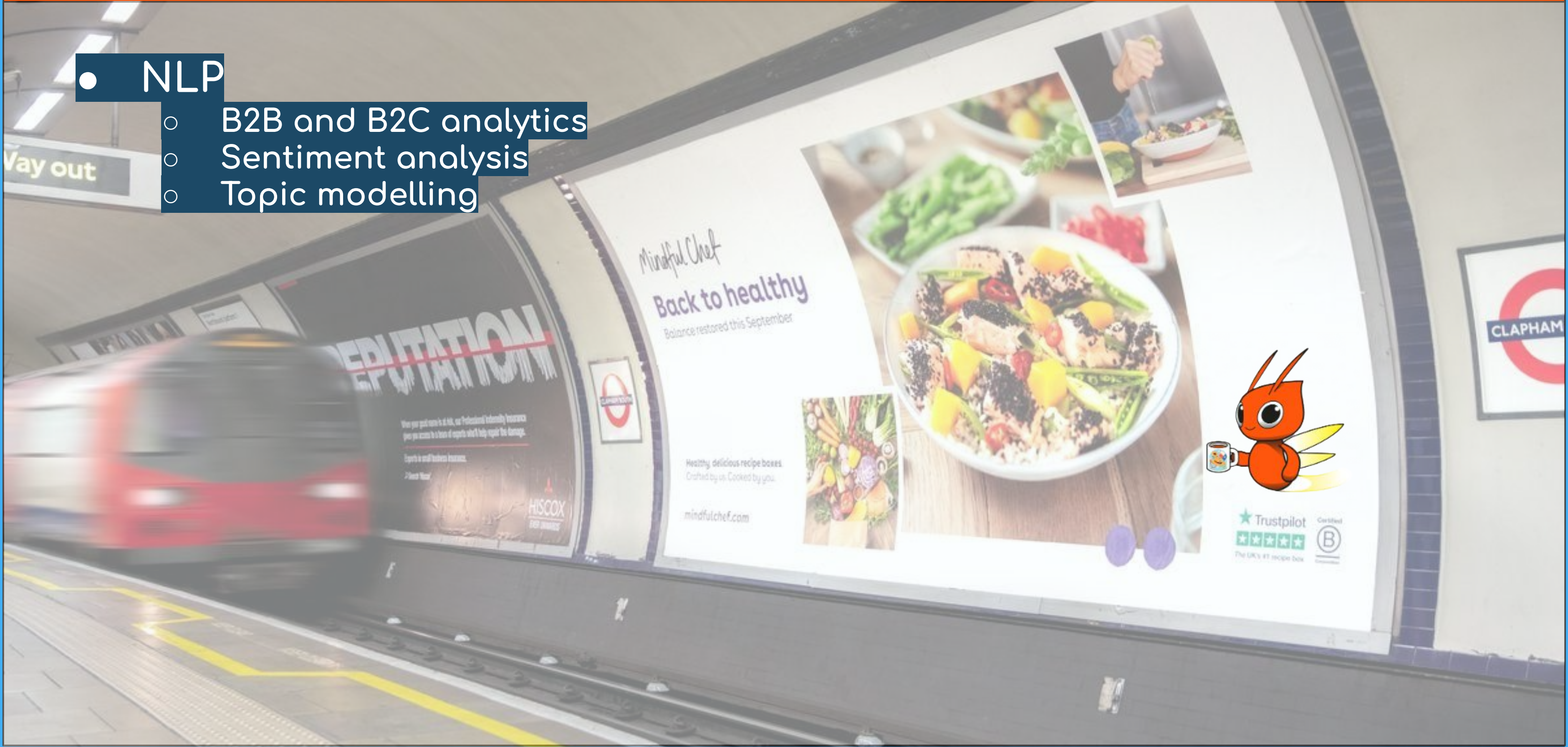
Way out
Mindful Chef
Back to healthy
Balance restored this September
Healthy, delicious recipe boxes.
Crafted by us. Cooked by you.
mindfulchef.com
Trustpilot
Certified
B
HISCOX
CLAPHAM

# Apache Beam at Trustpilot



# Apache Beam at Trustpilot

- NLP
  - B2B and B2C analytics
  - Sentiment analysis
  - Topic modelling



# Apache Beam at Trustpilot

- NLP
  - B2B and B2C analytics
  - Sentiment analysis
  - Topic modelling
- Platform integrity
  - Scam/spam
  - Fake reviews
  - Bad actors

# Apache Beam at Trustpilot


- NLP
  - B2B and B2C analytics
  - Sentiment analysis
  - Topic modelling
- Platform integrity
  - Scam/spam
  - Fake reviews
  - Bad actors
- Feature Store
  - Online store (real time stream)
  - Offline store (batch)
  - User events, review activity

# Apache Beam at Trustpilot

- NLP
    - B2B and B2C analytics
    - Sentiment analysis
    - Topic modelling
- Platform integrity
    - Scam/spam
    - Fake reviews
    - Bad actors
- Feature Store
    - Online store (real time stream)
    - Offline store (batch)
    - User events, review activity
- All on GCP Dataflow

# Agenda

- Dataflow Prime cost savings
    - Saving **40%** on our large batch ML jobs

# Agenda

- Dataflow Prime cost savings
  - **Saving 40%** on our large batch ML jobs
- Multi-model GPU sharing
  - Multiple embedding vectors for **LLM applications**

# Agenda

- Dataflow Prime cost savings
  - **Saving 40%** on our large batch ML jobs
- Multi-model GPU sharing
  - Multiple embedding vectors for **LLM applications**
- JAX for lin. alg. speedup
  - **Speeding up linear algebra operations** in Beam

# Agenda

- Dataflow Prime cost savings
  - Saving **40%** on our large batch ML jobs
- Multi-model GPU sharing
  - Multiple embedding vectors for **LLM applications**
- JAX for lin. alg. speedup
  - **Speeding up linear algebra operations** in Beam
- **`RunInference` for ML inference**
  - Improving **codebase maintainability**

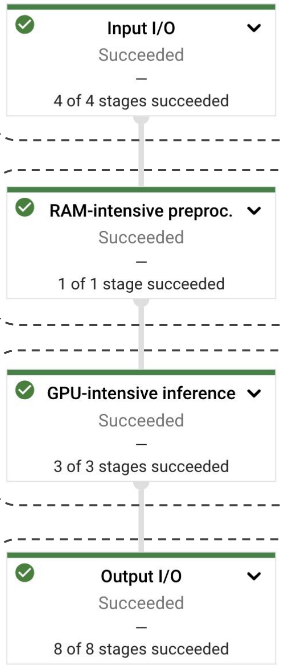
# Dataflow Prime

- Enable pipeline step-level resource specification

# Dataflow Prime

- Enable pipeline step-level resource specification
- Example ML pipeline

# Example pipeline




- Prepare data in BigQuery for processing

- Text preprocessing
- Feature engineering

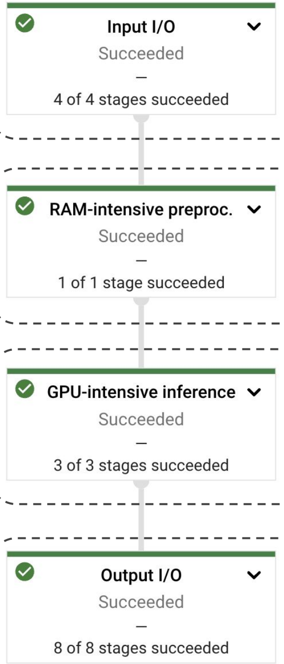
- ML model inference

- Launch load of data into BigQuery

## Dataflow Prime

- Enable pipeline step-level resource specification
- Example ML pipeline
- Imbalanced resource requirements
    - Each step has differing resource requirements

## Example pipeline




- Prepare data in BigQuery for processing
- *Low CPU, low RAM*

- Text preprocessing
- Feature engineering
- *High CPU, high RAM*

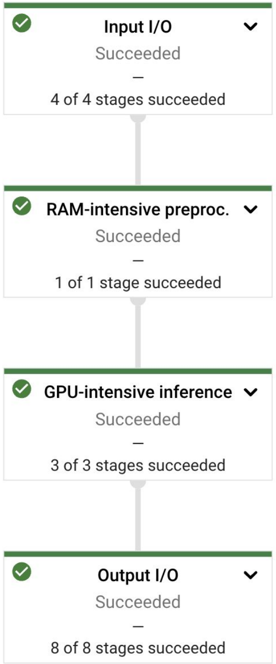
- ML model inference
- *Mid CPU, mid RAM, GPU*

- Launch load of data into BigQuery
- *Low CPU, low RAM*

# Dataflow Prime

- Enable pipeline step-level resource specification
- Example ML pipeline
- **Imbalanced resource requirements**
    - **Each step has differing resource requirements**

# Example pipeline




- Prepare data in BigQuery for processing
- ~~Low CPU, low RAM~~

- Text preprocessing
- Feature engineering
- *High CPU, high RAM*

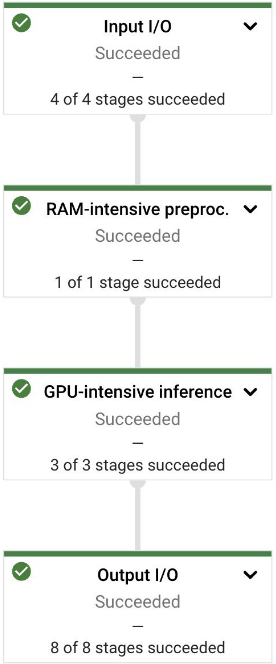
- ML model inference
- ~~Mid CPU~~, ~~mid RAM~~, GPU

- Launch load of data into BigQuery
- ~~Low CPU, low RAM~~

# Dataflow Prime

- Enable pipeline step-level resource specification
- Example ML pipeline
- Imbalanced resource requirements
    - Each step has differing resource requirements
- **Forced to take the "argmax" over each resource type**

# Uniform worker pool




- Prepare data in BigQuery for processing
- ~~Low CPU, low RAM~~

- Text preprocessing
- Feature engineering
- *High CPU*, *high RAM*

- ML model inference
- ~~Mid CPU~~, ~~mid RAM~~, *GPU*

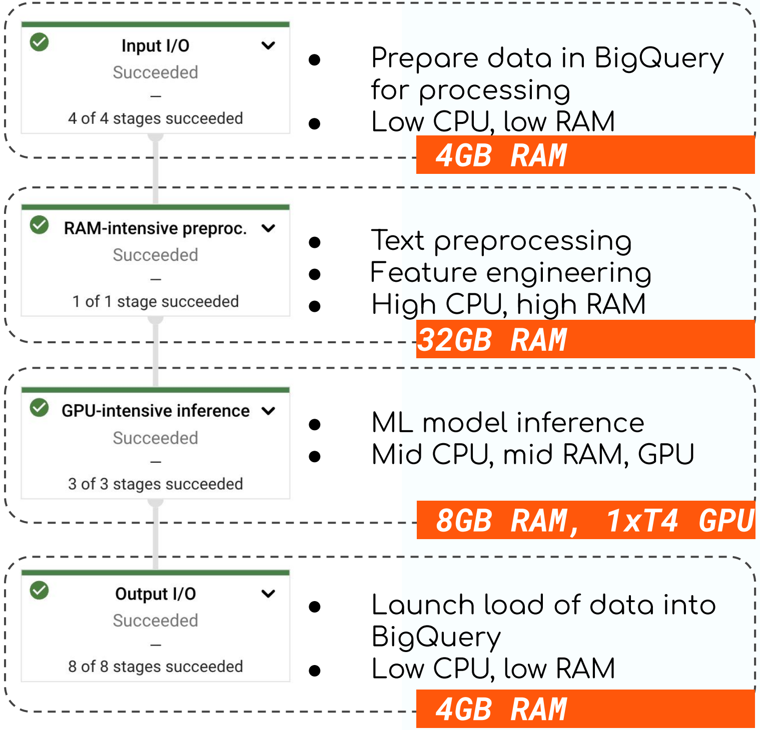
- Launch load of data into BigQuery
- ~~Low CPU, low RAM~~

***custom-4-32000-ext 1x T4 GPU***

# Dataflow Prime

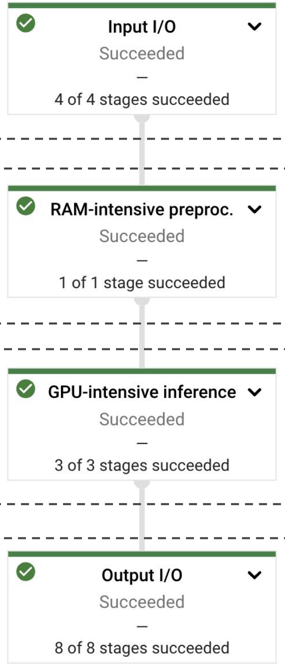
- Enable pipeline step-level resource specification
- Example ML pipeline
- Imbalanced resource requirements
    - Each step has differing resource requirements
- Forced to take the "argmax" over each resource type

## With resource hint specification



```python
with beam.Pipeline(options=PipelineOptions(

)) as pipeline:
  _ = (
     pipeline
     |inputPTransform()

     |preprocPTransform()

     |inferencePTransform()

     |inferencePTransform()
  )
```

# With resource hint specification




- Prepare data in BigQuery for processing
- Low CPU, low RAM

**4GB RAM**

- Text preprocessing
- Feature engineering
- High CPU, high RAM

**32GB RAM**

- ML model inference
- Mid CPU, mid RAM, GPU

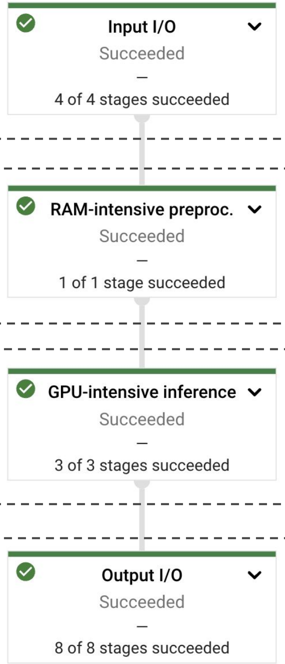
**8GB RAM, 1xT4 GPU**

- Launch load of data into BigQuery
- Low CPU, low RAM

**4GB RAM**

```python
with beam.Pipeline(options=PipelineOptions(
    dataflow_service_options=["enable_prime"]
)) as pipeline:
  _ = (
    pipeline
    |inputPTransform().with_resource_hints(
        min_ram="4GB", accelerator=None
    )
    |preprocPTransform().with_resource_hints(
        min_ram="32GB", accelerator=None
    )
    |inferencePTransform().with_resource_hints(
        min_ram="8GB",
        accelerator="type:nvidia-tesla-t4;
        count:1;install-nvidia-driver"
    )
    |inferencePTransform().with_resource_hints(
        min_ram="4GB", accelerator=None
    )
  )
```

# With resource hint specification




- Prepare data in BigQuery for processing
- Low CPU, low RAM

***4GB RAM***

- Text preprocessing
- Feature engineering
- High CPU, high RAM

***32GB RAM***

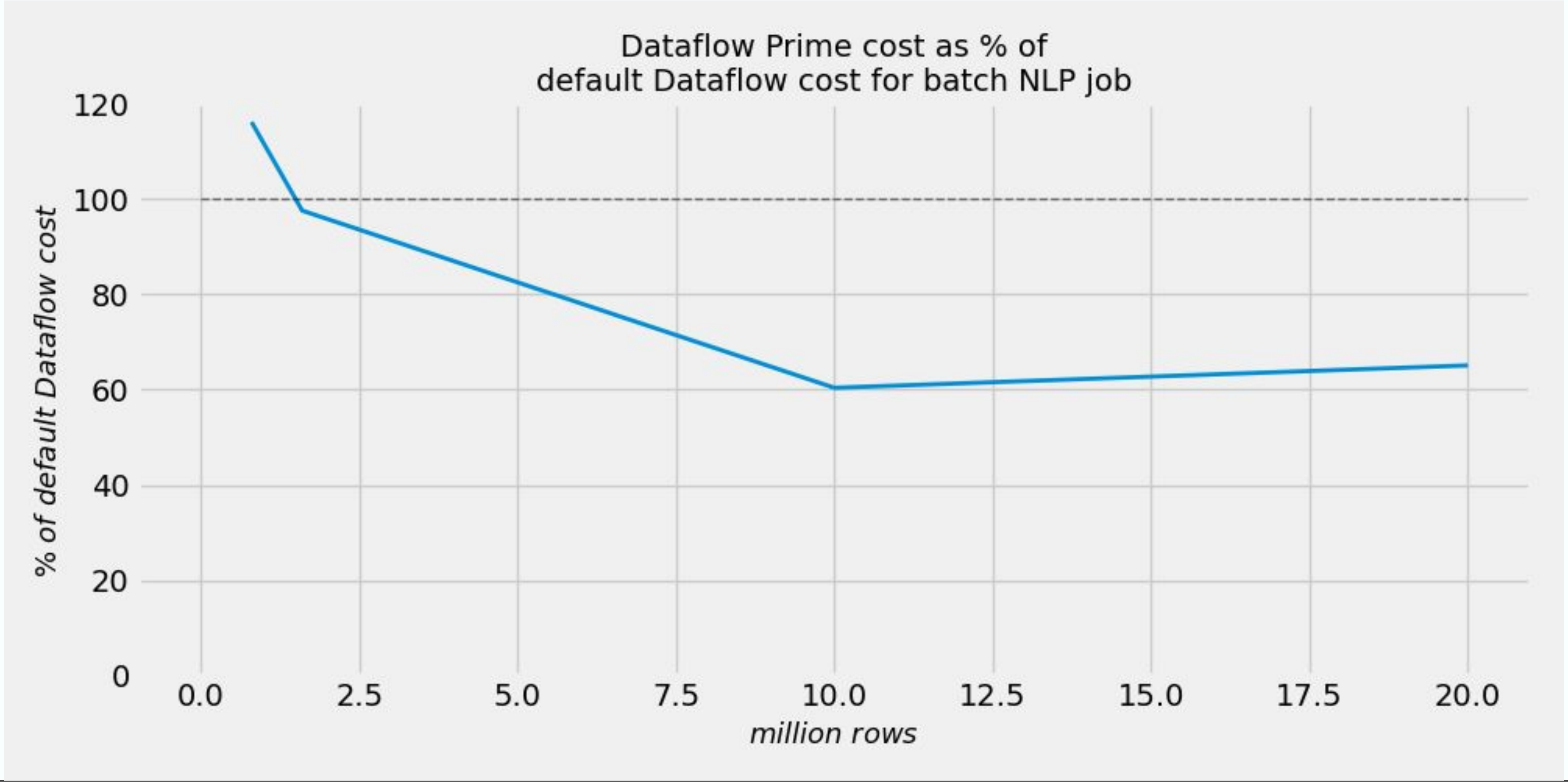
- ML model inference
- Mid CPU, mid RAM, GPU

***8GB RAM, 1xT4 GPU***

- Launch load of data into BigQuery
- Low CPU, low RAM

***4GB RAM***

# Dataflow Prime cost savings



# Review

- I/O can take a lot of time
  - In particular the **`WriteToBigQuery`** **`PTransform`** is I/O-bound
  - Can take a long time:
    - **`TriggerLoadJobs DoFn`**
    - **`WriteRecordsToFile DoFn`**
  - Don't need special resources for these steps
  - With *Prime*, just specify to run with lower resources

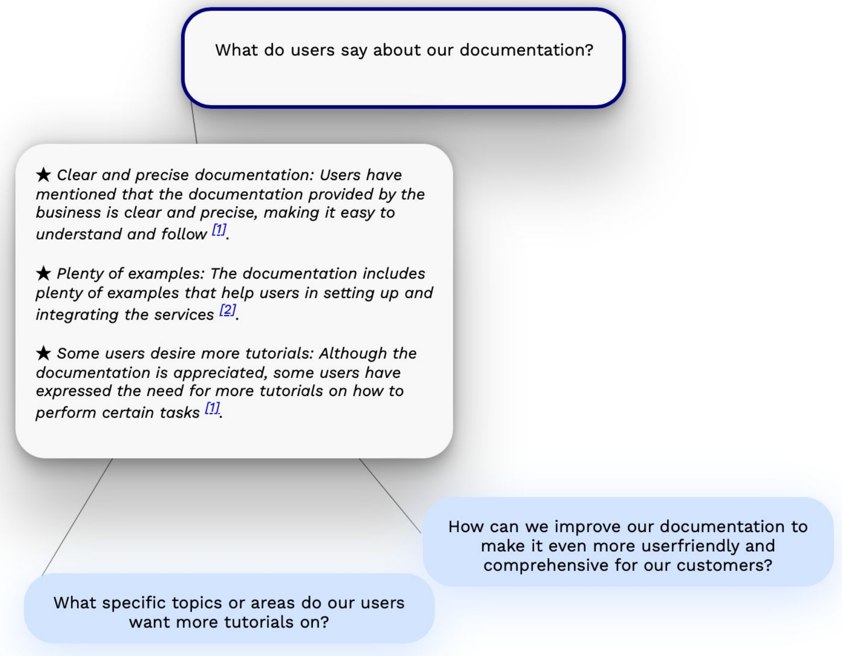
- GPU steps are dependent on CPU-bound steps
  - Separating these steps enables more efficient use of GPU
  - Ensures higher batch size can be realized == higher throughput
  - Breaking the dependency ensures higher GPU utililization

# Review

- Benchmark your pipelines to measure expected savings
    - It may not be the case that you will end up better off
- Break fusion to discretize worker pools
    - Dataflow will sometimes fuse together steps you want to keep separate
    - Insert a Reshuffle PTransform

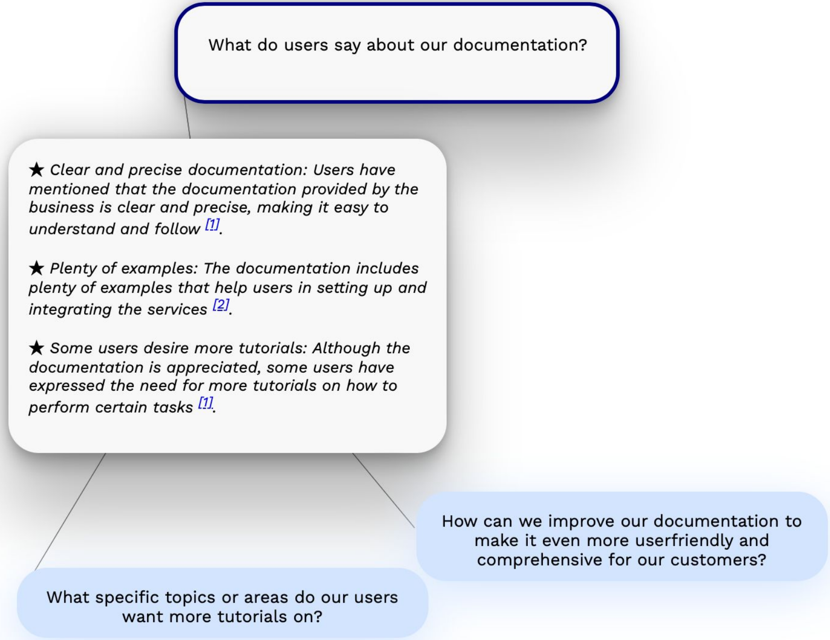
# Multi-model GPU sharing

- LLM-based products and applications
    - chat, agents
- Retrieval augmentation
    - add relevant context to prompt to reduce hallucinations
- Embed data for similarity lookup
    - vector store



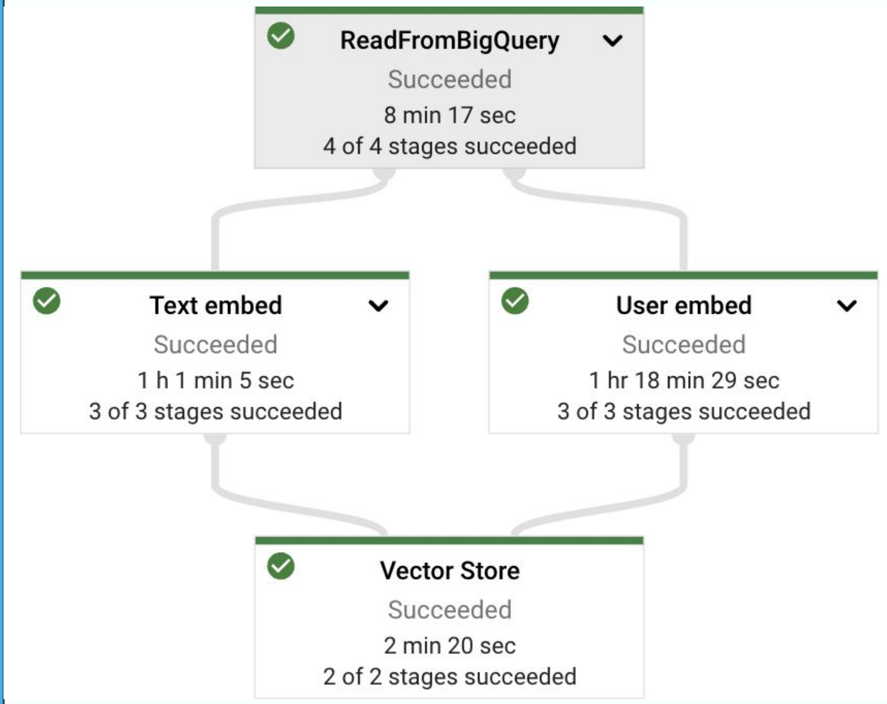
# Multi-model GPU sharing

- LLM-based products and applications
  - chat, agents
- Retrieval augmentation
  - add relevant context to prompt to reduce hallucinations
- Embed data for similarity lookup
  - vector store
- Multiple ways to embed data
  - semantic
  - fraud detection
  - recommendations



- Currently: Difficult to share GPU
- Forthcoming: Load multiple models onto GPU
- Sink to vector store
    - Multi-embedding index

## Example pipeline



# Speeding up linalg operations

- Linear algebra operations
  - Matrix multiplication
  - PCA
  - Distance metrics
  - Broadcasted vector operations
  - Even neural networks

# Numpy example

```
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 example

*drop-in library*

```python
from jax import numpy as jnp

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

    return jnp.sqrt(summed)

# JIT compile
euclidean_dist_jax = jit(_euclidean_dist)
```

*compile to fuse operations together*

## Set up JAX in Beam

① Define a pure function

- **`@staticmethod`**
- Linear algebra operations

## Code

```
class RBFKernel(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)
```

①

## Set up JAX in Beam

① Define a pure function

- **@staticmethod**
- Linear algebra operations

② JIT compile

- In **__init__**
- Compile once

## Code

```
class RBFKernel(DoFn):
    def __init__(self):
        self.rbf_jit = jit(self._rbf)  ②

    @staticmethod
    def _rbf(X, Y, gamma):      ①
        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)
```

## Set up JAX in Beam

① Define a pure function

- **@staticmethod**
- Linear algebra operations

② JIT compile

- In **__init__**
- Compile once

③ Use the compiled function

## Code

```
class RBFKernel(DoFn):
    def __init__(self):
        self.rbf_jit = jit(self._rbf)   ②

    @staticmethod
    def _rbf(X, Y, gamma):      ①
        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):        ③
        ...
        yield key, self.rbf_jit(X, Y, gamma)
```

## Set up JAX in Beam

① Define a pure function

- **@staticmethod**
- Linear algebra operations

② JIT compile

- In **__init__**
- Compile once

③ Use the compiled function

- PCA + distance metric faster by ~10x compared to **sklearn** implementation (CPU)

## Code

```
class RBFKernel(DoFn):
    def __init__(self):
        self.rbf_jit = jit(self._rbf)   ②

    @staticmethod
    def _rbf(X, Y, gamma):       ①
        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):        ③
        ...
        yield key, self.rbf_jit(X, Y, gamma)
```

# Adopting `RunInference`

- **`RunInference`**
  - **`PTransform`** in the Python SDK for running ML inference

| Custom inference (previous) | | RunInference |
|---|---|---|
| - Create weak references to model object | → | - Choose a **`ModelHandler`** and provide model URI |
| - Set shared handle on model init. func. | → | - Model object sharing is handled inside **`RunInference`** |
| - Manual device/**`CUDA`** management | → | - Device set as part of **`ModelHandler`** args (**`PyTorch`**) |
| - Dealing with different APIs for each ML library | → | - Pick among many supported **`ModelHandler`**s |

# "Benchmarking" codebase improvements

```
~/go/bin/scc src/ml/sentiment.py
```

(Composite **PTransform** with custom inference)

| Language | Files | Lines | Blanks | Comments | Code | Complexity |
|---|---|---|---|---|---|---|
| Python | 1 | 212 | 32 | 68 | 112 | 14 |
| Total | 1 | 212 | 32 | 68 | 112 | 14 |

```
Estimated Cost to Develop (organic) $2,711
Estimated Schedule Effort (organic) 1.46 months
Estimated People Required (organic) 0.17

Processed 7007 bytes, 0.007 megabytes (SI)
```

GitHub: boyter/scc
GitHub: boyter/scc—Complexity
Wikipedia: Cyclomatic complexity

# "Benchmarking" codebase improvements

```
~/go/bin/scc ./v2/experimental/transforms/ml/sentiment.py
───────────────────────────────────────────────────────────────────────────────
Language                 Files     Lines   Blanks  Comments     Code Complexity
───────────────────────────────────────────────────────────────────────────────
Python                       1       122       16        52       54          2
───────────────────────────────────────────────────────────────────────────────
Total                        1       122       16        52       54          2
───────────────────────────────────────────────────────────────────────────────
Estimated Cost to Develop (organic) $1,260
Estimated Schedule Effort (organic) 1.09 months
Estimated People Required (organic) 0.10
───────────────────────────────────────────────────────────────────────────────
Processed 3972 bytes, 0.004 megabytes (SI)
───────────────────────────────────────────────────────────────────────────────
```

(Composite PTransform with **RunInference**)

[GitHub: boyter/scc](#)
[GitHub: boyter/scc—Complexity](#)
[Wikipedia: Cyclomatic complexity](#)

# Recap

- Use case 1: Large batch pipelines with imbalanced resources
  - Dataflow Prime cost saving 40%
- Use case 2: Running pipelines with multiple embedding models
  - Forthcoming on Dataflow
- Use case 3: Speeding up linear algebra operations in Beam
  - 
- Use case 4: Improve codebase maintainability
  - Replace custom inference implementations with **`RunInference`**

# QUESTIONS?

# Resources

- https://cloud.google.com/dataflow/docs/guides/enable-dataflow-prime
- https://beam.apache.org/documentation/runtime/resource-hints/

- Qdrant-Storing multiple vectors per object in Qdrant

- https://github.com/google/jax
- https://jax.readthedocs.io/en/latest/notebooks/thinking_in_jax.html

- https://beam.apache.org/documentation/transforms/python/elementwise/runinference/
- GitHub: boyter/scc
- GitHub: boyter/scc—Complexity
- Wikipedia: Cyclomatic complexity