Machine Learning Pipelines at Scale with Apache Spark

Example of an End-To-End ML pipeline for High Energy Physics

Matteo Migliorini
University of Padua, CERN IT-DB

High Level Goals

- Investigate and develop solutions integrating:
  - Data Engineering/Big Data tools
  - Machine learning tools
  - Data analytics platform
- Use Industry standard tools
  - Well known and maintained by a large community

Use case

- Topology classification with deep learning to improve real time event selection at the LHC [https://arxiv.org/abs/1807.00083]
- Improve the purity of data samples selected in real time at the Large Hadron Collider
    - Triggers are designed to maximize efficiency (TP rate)
    - **Inclusive** selection rules: more than one topology selected by the same requirements (e.g. isolated lepton triggers)
    - This trigger selects events containing **W** and **$t\bar{t}$** but also **QCD**
    - Classify different event topology at trigger level

Datasets: simulated sample

- Each event of the **simulated sample** consists of a list of Particle-Flow candidates.

```
|    |    |-- Phi: float (nullable = true)
|-- Jet: array (nullable = true)
|    |-- element: struct (containsNull = true)
|    |    |-- fUniqueID: integer (nullable = true)
|    |    |-- fBits: integer (nullable = true)
|    |    |-- PT: float (nullable = true)
|    |    |-- Eta: float (nullable = true)
|    |    |-- Phi: float (nullable = true)
|    |    |-- T: float (nullable = true)
|    |    |-- Mass: float (nullable = true)
|    |    |-- DeltaEta: float (nullable = true)
|    |    |-- DeltaPhi: float (nullable = true)
|    |    |-- Flavor: integer (nullable = true)
|    |    |-- FlavorAlgo: integer (nullable = true)
|    |    |-- FlavorPhys: integer (nullable = true)
|    |    |-- BTag: integer (nullable = true)
|    |    |-- BTagAlgo: integer (nullable = true)
|    |    |-- BTagPhys: integer (nullable = true)
|    |    |-- TauTag: integer (nullable = true)
|    |    |-- Charge: integer (nullable = true)
|    |    |-- EhadOverEem: float (nullable = true)
|    |    |-- NCharged: integer (nullable = true)
|    |    |-- NNeutrals: integer (nullable = true)
|    |    |-- Beta: float (nullable = true)
|    |    |-- BetaStar: float (nullable = true)
|    |    |-- MeanSqDeltaR: float (nullable = true)
|    |    |-- PTD: float (nullable = true)
|    |    |-- FracPt: array (nullable = true)
|    |    |    |-- element: float (containsNull = true)
|    |    |-- Tau: array (nullable = true)
|    |    |    |-- element: float (containsNull = true)
|    |    |-- TrimmedP4: array (nullable = true)
|    |    |    |-- element: struct (containsNull = true)
|    |    |    |    |-- TObject: struct (nullable = true)
|    |    |    |    |    |-- fUniqueID: integer (nullable = true)
|    |    |    |    |    |-- fBits: integer (nullable = true)
|    |    |    |    |-- fP: struct (nullable = true)
|    |    |    |    |    |-- TObject: struct (nullable = true)
|    |    |    |    |    |    |-- fUniqueID: integer (nullable = true)
|    |    |    |    |    |    |-- fBits: integer (nullable = true)
|    |    |    |    |    |-- fX: double (nullable = true)
|    |    |    |    |    |-- fY: double (nullable = true)
|    |    |    |    |    |-- fZ: double (nullable = true)
|    |    |    |    |-- fE: double (nullable = true)
|    |    |-- PrunedP4: array (nullable = true)
|    |    |    |-- element: struct (containsNull = true)
|    |    |    |    |-- TObject: struct (nullable = true)
|    |    |    |    |    |-- fUniqueID: integer (nullable = true)
|    |    |    |    |    |-- fBits: integer (nullable = true)
|    |    |    |    |-- fP: struct (nullable = true)
|    |    |    |    |    |-- TObject: struct (nullable = true)
|    |    |    |    |    |    |-- fUniqueID: integer (nullable = true)
|    |    |    |    |    |    |-- fBits: integer (nullable = true)
|    |    |    |    |    |-- fX: double (nullable = true)
|    |    |    |    |    |-- fY: double (nullable = true)
|    |    |    |    |    |-- fZ: double (nullable = true)
```

```
|-- EFlowNeutralHadron: array (nullable = true)
|    |-- element: struct (containsNull = true)
|    |    |-- fUniqueID: integer (nullable = true)
|    |    |-- fBits: integer (nullable = true)
|    |    |-- ET: float (nullable = true)
|    |    |-- Eta: float (nullable = true)
|    |    |-- Phi: float (nullable = true)
|    |    |-- E: float (nullable = true)
|    |    |-- T: float (nullable = true)
|    |    |-- NTimeHits: integer (nullable = true)
|    |    |-- Eem: float (nullable = true)
|    |    |-- Ehad: float (nullable = true)
|    |    |-- Edges: array (nullable = true)
|    |    |    |-- element: float (containsNull = true)
|-- EFlowPhoton: array (nullable = true)
|    |-- element: struct (containsNull = true)
|    |    |-- fUniqueID: integer (nullable = true)
|    |    |-- fBits: integer (nullable = true)
|    |    |-- ET: float (nullable = true)
|    |    |-- Eta: float (nullable = true)
|    |    |-- Phi: float (nullable = true)
|    |    |-- E: float (nullable = true)
|    |    |-- T: float (nullable = true)
|    |    |-- NTimeHits: integer (nullable = true)
|    |    |-- Eem: float (nullable = true)
|    |    |-- Ehad: float (nullable = true)
|    |    |-- Edges: array (nullable = true)
|    |    |    |-- element: float (containsNull = true)
|-- Electron: array (nullable = true)
|    |-- element: struct (containsNull = true)
|    |    |-- fUniqueID: integer (nullable = true)
|    |    |-- fBits: integer (nullable = true)
|    |    |-- PT: float (nullable = true)
|    |    |-- Eta: float (nullable = true)
|    |    |-- Phi: float (nullable = true)
|    |    |-- T: float (nullable = true)
|    |    |-- Charge: integer (nullable = true)
|    |    |-- EhadOverEem: float (nullable = true)
|    |    |-- Electron_Particle: struct (nullable = true)
|    |    |    |-- TObject: struct (nullable = true)
|    |    |    |    |-- fUniqueID: integer (nullable = true)
|    |    |    |    |-- fBits: integer (nullable = true)
|    |    |-- IsolationVar: float (nullable = true)
|    |    |-- IsolationVarRhoCorr: float (nullable = true)
|    |    |-- SumPtCharged: float (nullable = true)
|    |    |-- SumPtNeutral: float (nullable = true)
|    |    |-- SumPtChargedPU: float (nullable = true)
|    |    |-- SumPt: float (nullable = true)
```

HLF & LLF datasets

- Each event of the simulated sample consists of a list of Particle-Flow candidates.
- The trigger selection is emulated by requiring events to include one isolated electron/muon with **p_T>23GeV** and particle based **isolation<0.45**
- All particles are ranked in decreasing order of p_T, where the isolated lepton is the first particle of the list
    - **Low Level Feature dataset**: First 801 particles of this list, each described by 19 features (four-momentum, origin, …)
- **High Level Feature dataset**: List of 14 physics-motivated features computed from the LLF dataset

Models

- **HLF classifier**: fully connected DNN taking as input the 14 high level features. It consists of 3 hidden layers with 50, 20, 10 nodes and an output with 3 units.

- **Particle-sequence classifier**: RNN taking as input the list of 801 particles. Particles are sorted by a decreasing distance ΔR from the isolated lepton. Gated recurrent unit are used to aggregate the input sequence and the width of the recurrent layer was 50.

- **Inclusive classifier:** In this model some physics knowledge is injected into the Particle-sequence classifier by concatenating the 14 HLF to the output of the GRU.

Machine Learning Pipeline

The goals of this work are:

- Produce an example of a ML pipeline using Spark
- Test the performances of Spark at each stage

Machine Learning Pipeline

The goals of this work are:

- Produce an example of a ML pipeline using Spark
- Test the performances of Spark at each stage

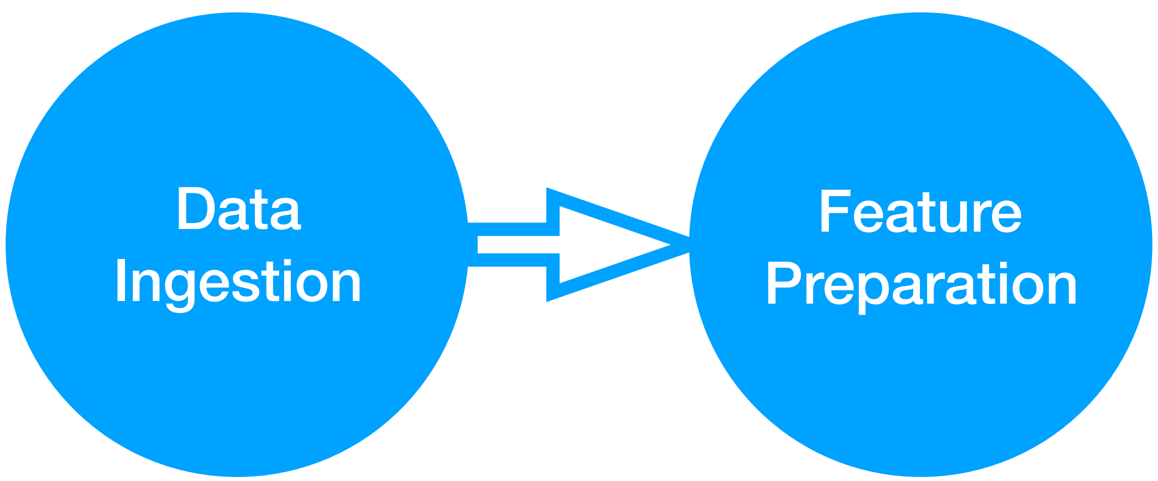
Data Ingestion

- Read Root Files from EOS
- Produce HLF and LLF datasets

Machine Learning Pipeline

The goals of this work are:

- Produce an example of a ML pipeline using Spark
- Test the performances of Spark at each stage



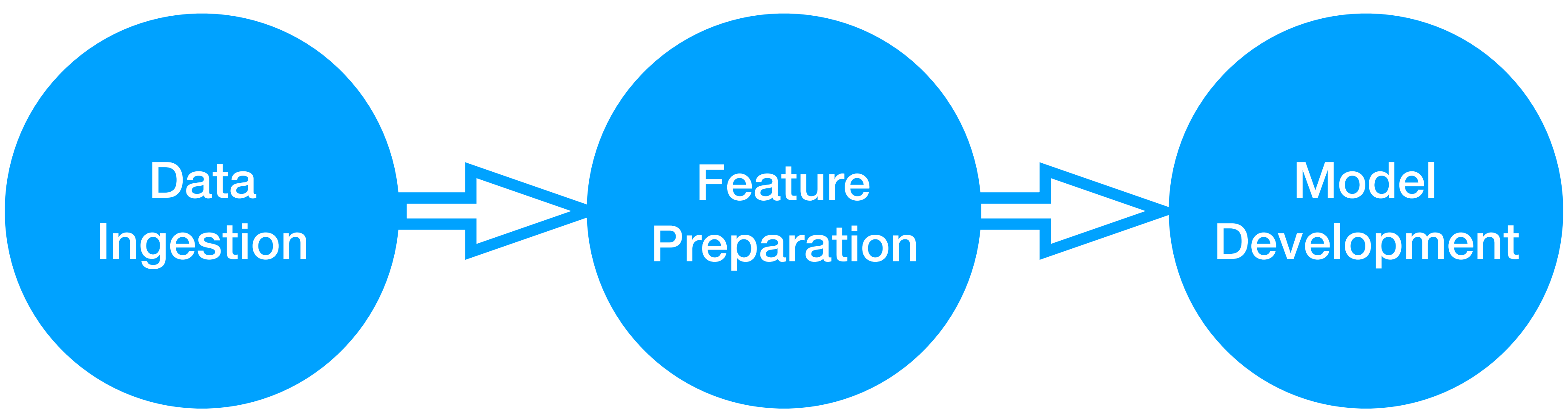
- Read Root Files from EOS
- Produce HLF and LLF datasets

- Produce the input for each classifier

Machine Learning Pipeline

The goals of this work are:

- Produce an example of a ML pipeline using Spark
- Test the performances of Spark at each stage



Data Ingestion
- Read Root Files from EOS
- Produce HLF and LLF datasets

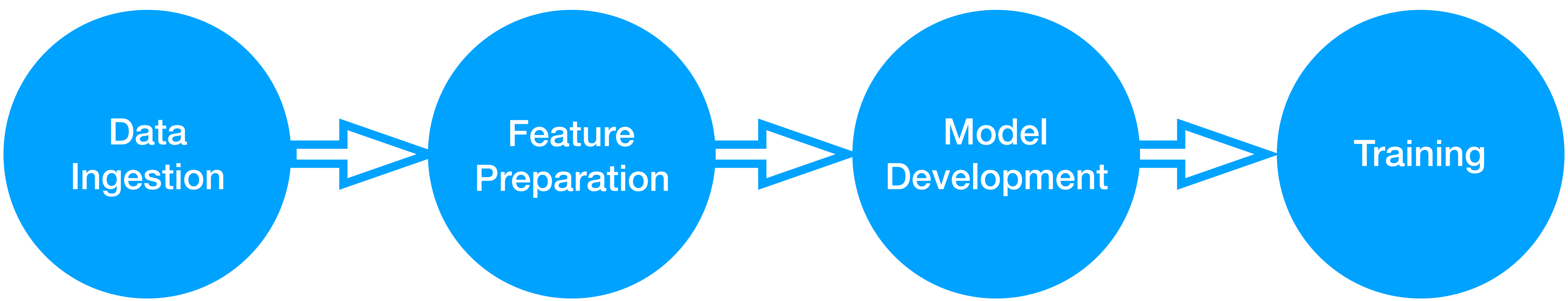
Feature Preparation
- Produce the input for each classifier

Model Development
- Find the best model using Grid/Random search

Machine Learning Pipeline

The goals of this work are:

- Produce an example of a ML pipeline using Spark
- Test the performances of Spark at each stage

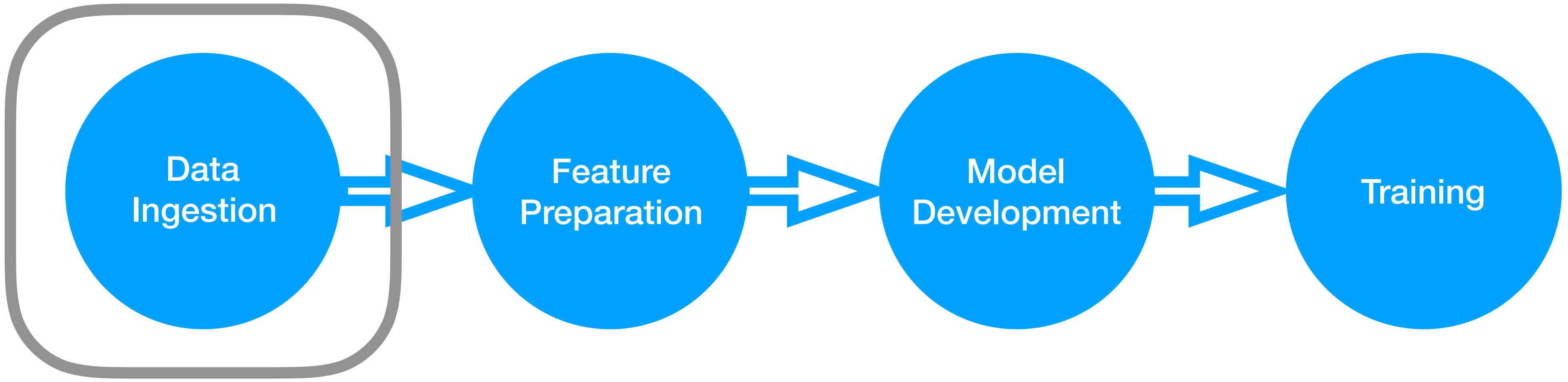


Data Ingestion
- Read Root Files from EOS
- Produce HLF and LLF datasets

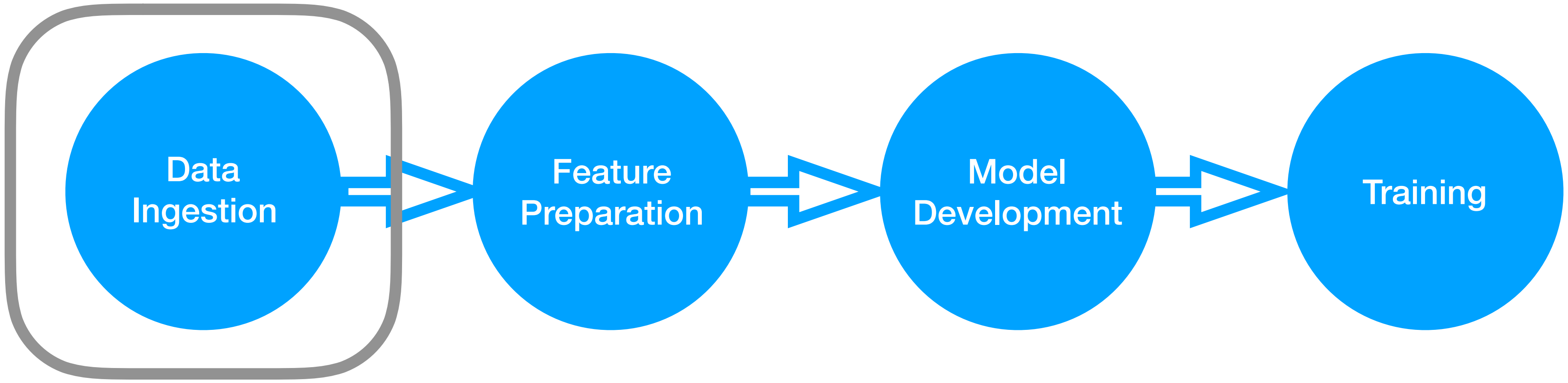
Feature Preparation
- Produce the input for each classifier

Model Development
- Find the best model using Grid/Random search

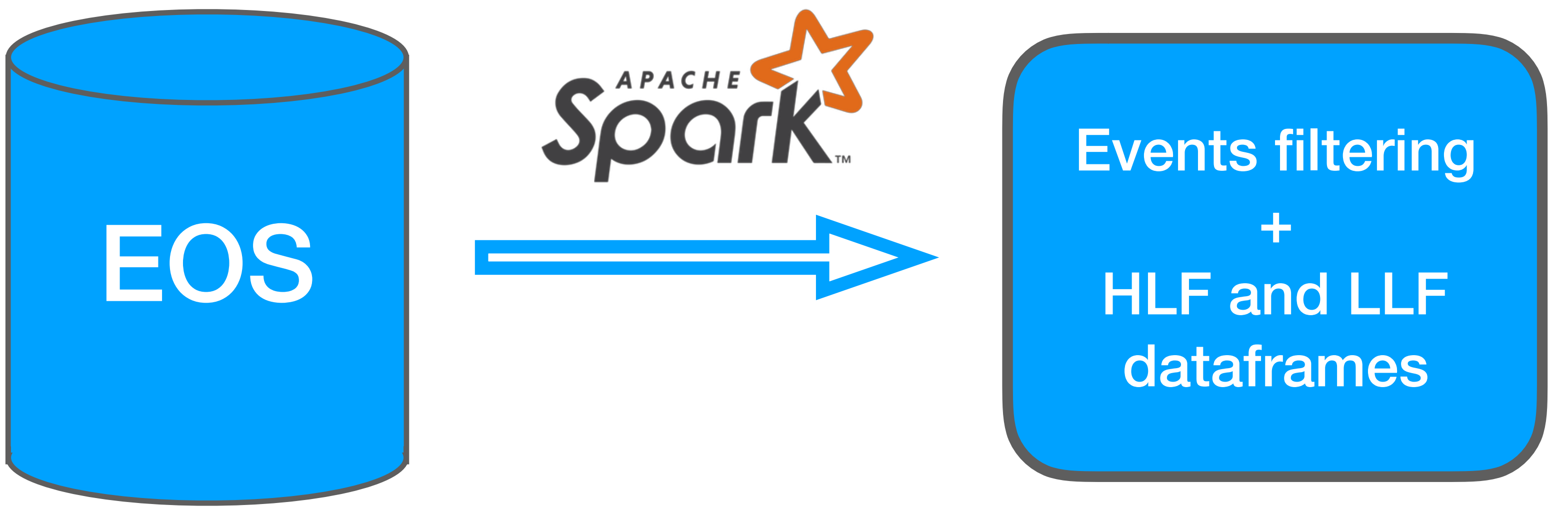
Training
- Train the best model on the entire dataset



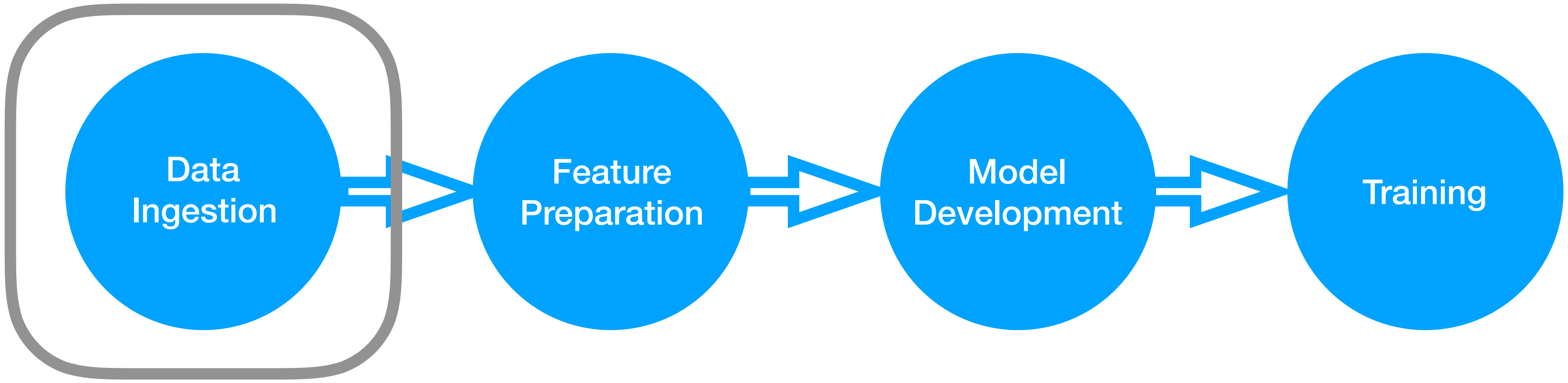
Input Size: ~10 TBs,
50M events



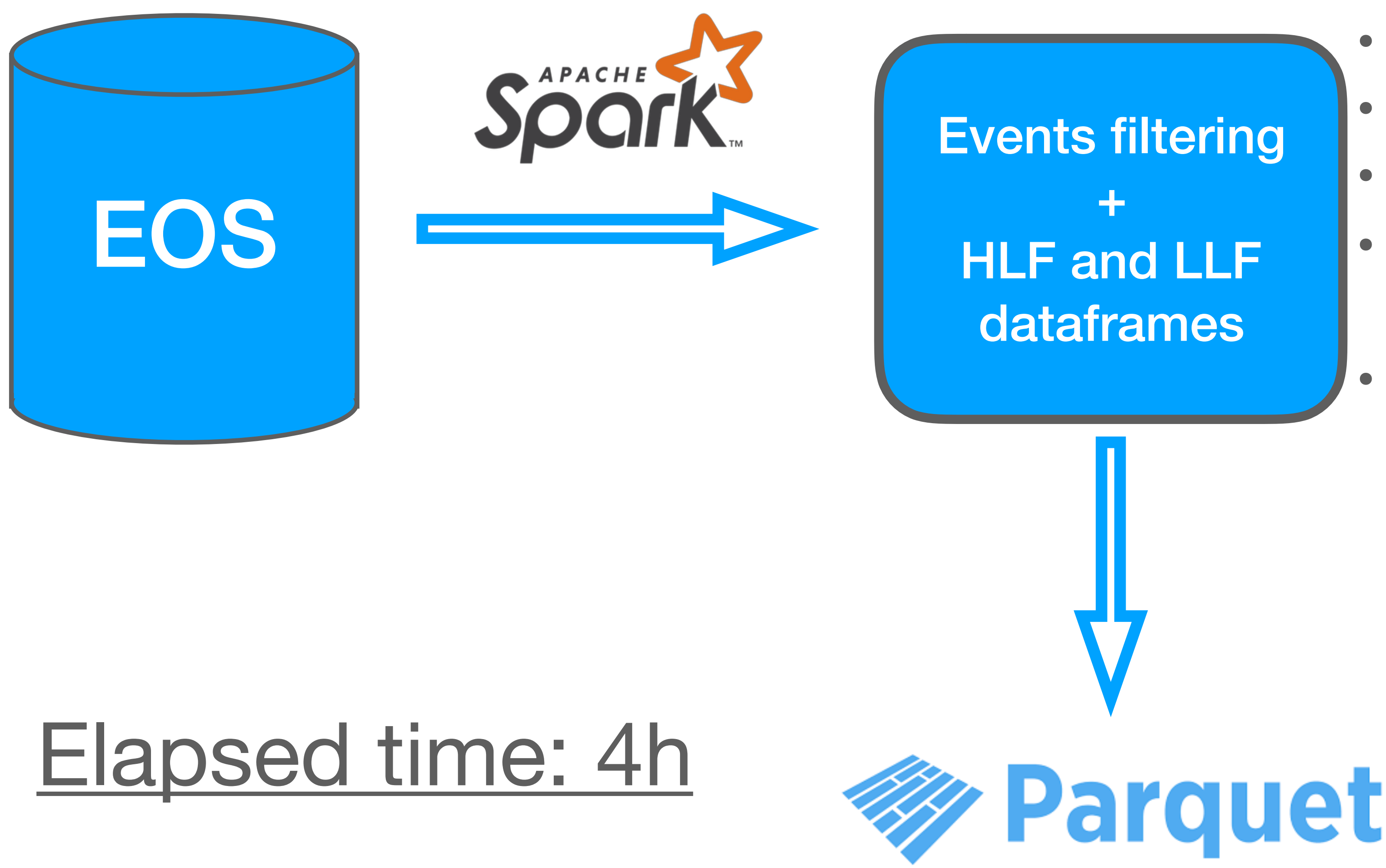
Input Size: ~10 TBs,
50M events



- Electron or Muon with p_T>23 GeV
- Particle-based isolation<0.45
- …
- LLF: list of 801 particles, each characterised by 19 features
- 14 HLF features



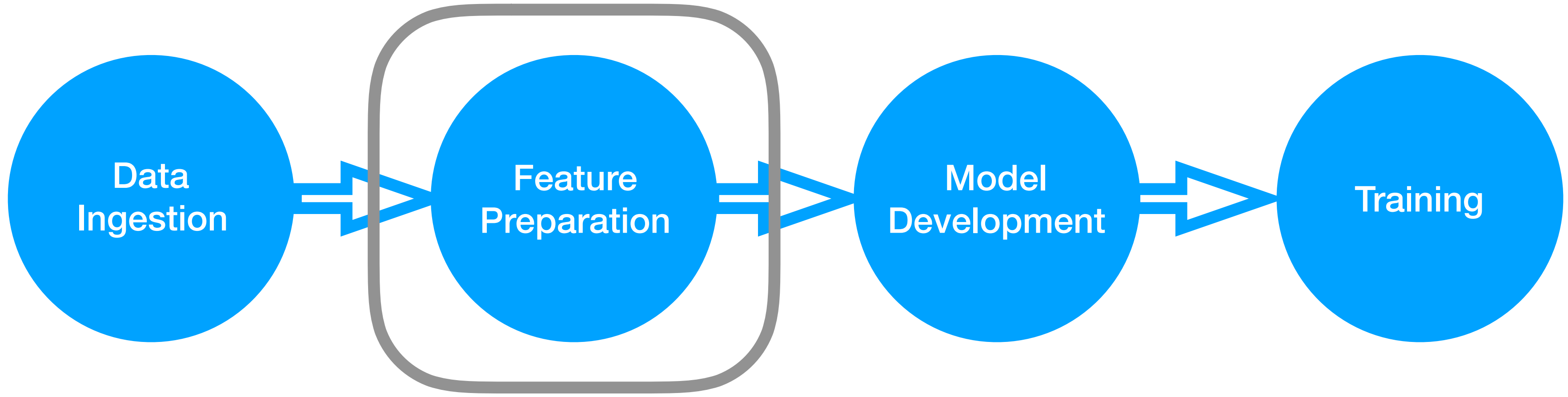
Input Size: ~10 TBs,
50M events



- Electron or Muon with p_T>23 GeV
- Particle-based isolation<0.45
- …
- LLF: list of 801 particles, each characterised by 19 features
- 14 HLF features

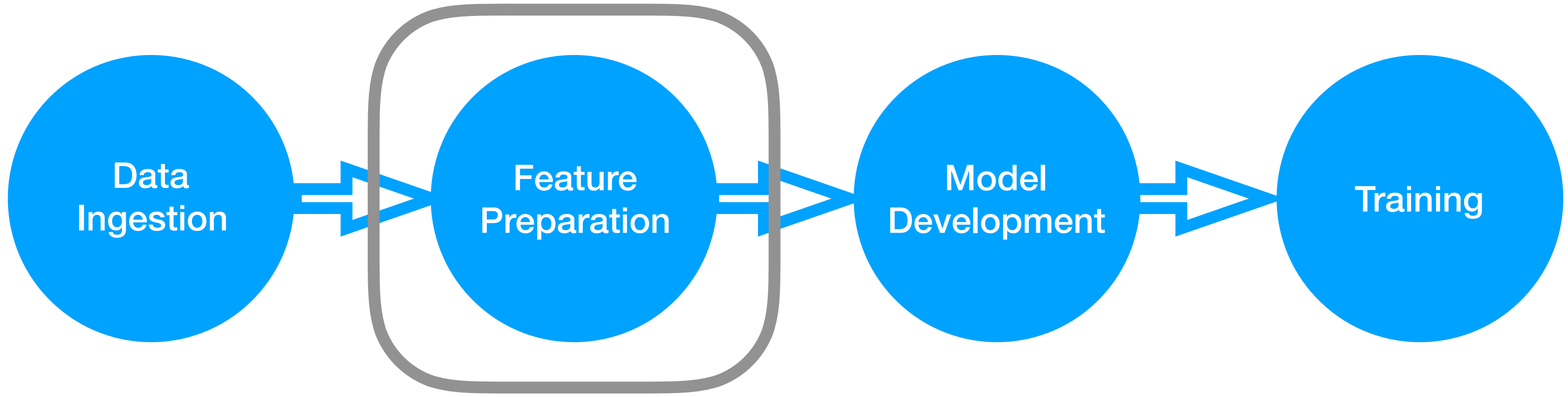
<u>Elapsed time: 4h</u>

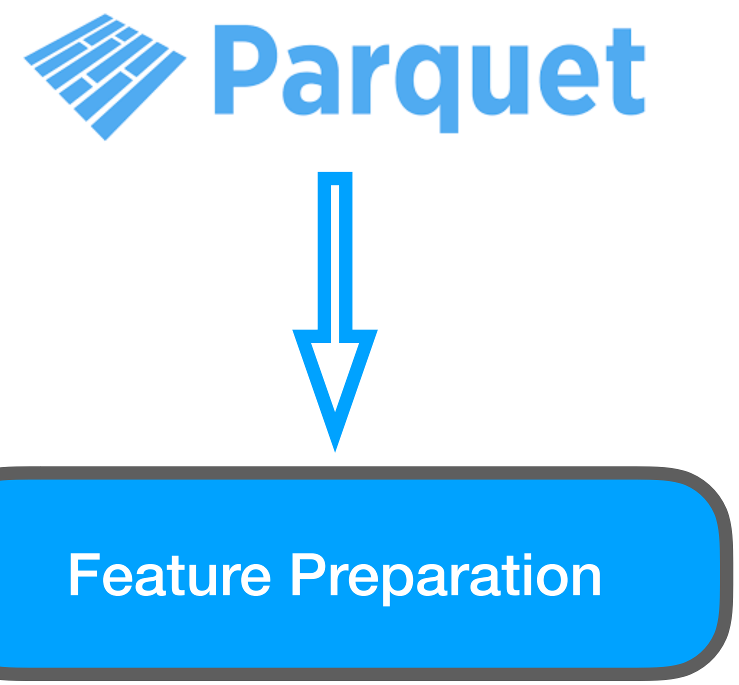
Output Size: 750 GBs



Parquet

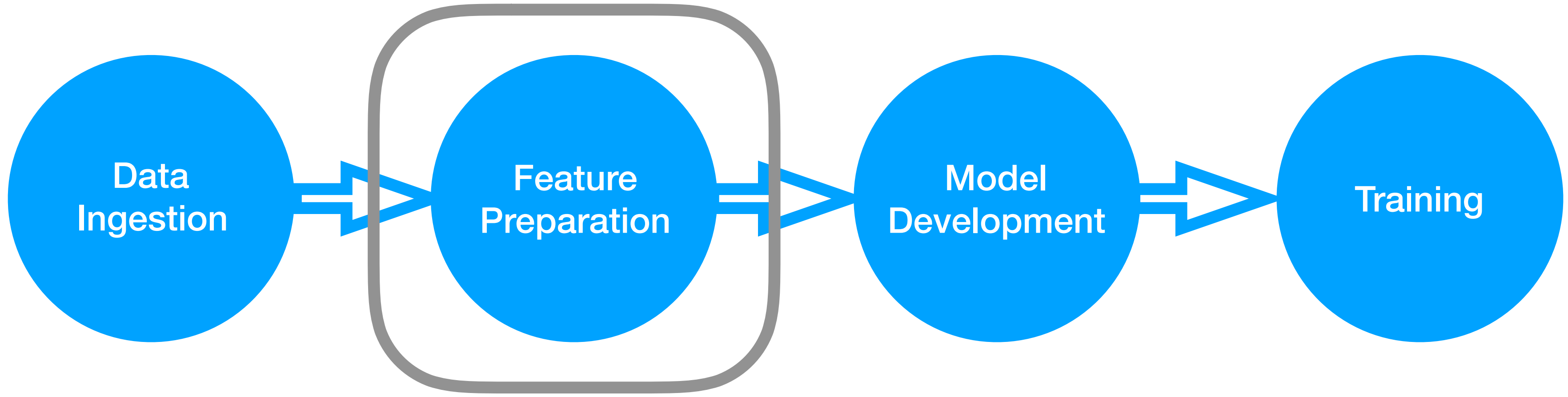
Start from the output of the previous stage

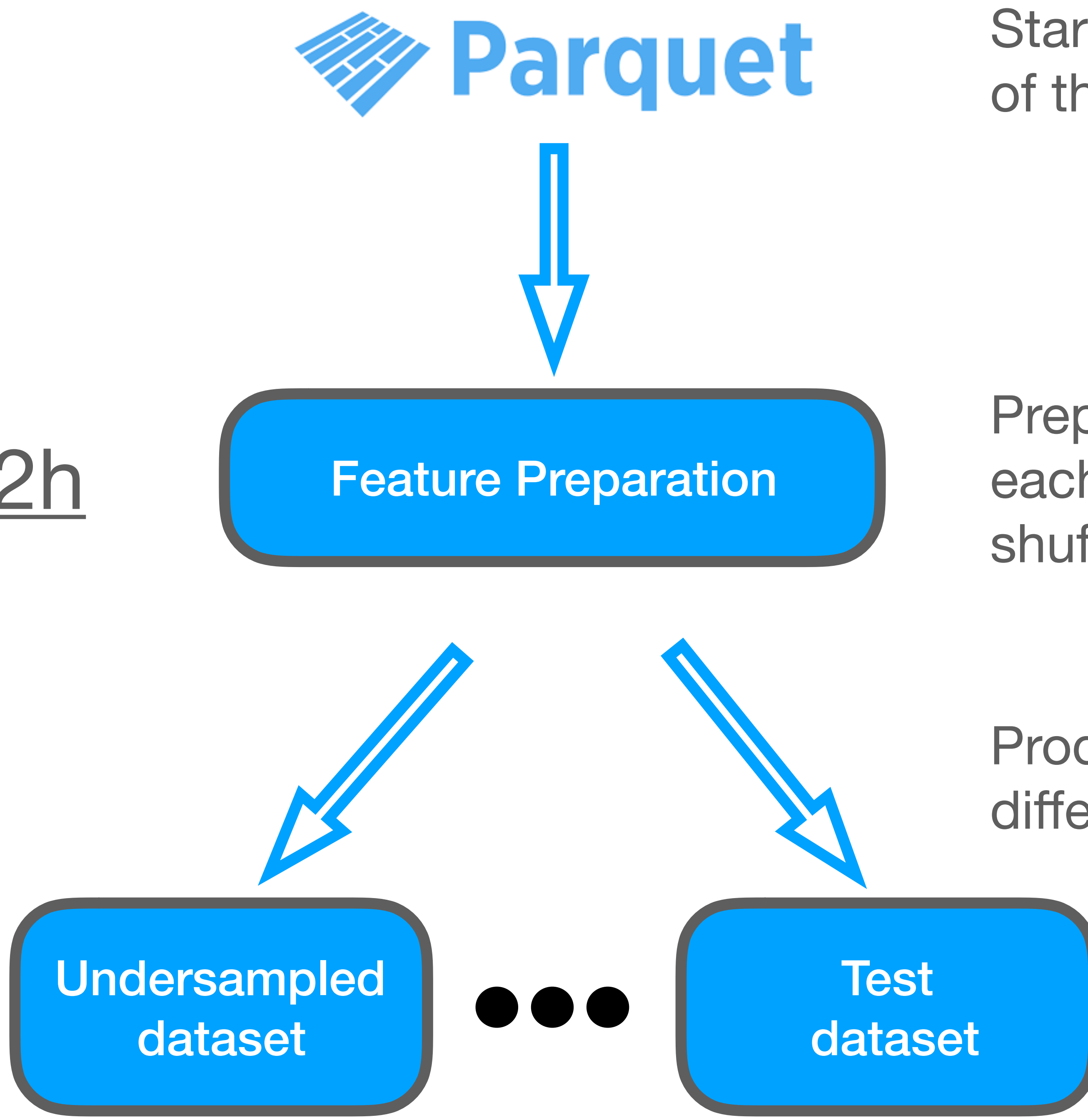




Start from the output of the previous stage

Prepare the input for each classifier and shuffle the dataframe





Start from the output of the previous stage

Elapsed time: 2h

Prepare the input for each classifier and shuffle the dataframe

Produce samples of different sizes

Comments on Spark for Data Engineering at Scale

- Easy to do feature preparation at scale!
- Spark scalability:
  - Allows to develop code locally and then deploy the same code on a arbitrary large cluster
  - We can connect a notebook to the cluster and performe an interactive analysis

Feature preparation

Elements of the `hfeatures` column are list, hence we need to convert them into `Vectors.Dense`

In [10]:
```
from pyspark.ml.linalg import Vectors, VectorUDT
from pyspark.sql.functions import udf

vector_dense_udf = udf(lambda r : Vectors.dense(r),VectorUDT())
data = data.withColumn('hfeatures_dense',vector_dense_udf('hfeatures'))
```

Now we can build the pipeline to scale HLF and encode the labels

In [11]:
```
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoderEstimator
from pyspark.ml.feature import MinMaxScaler

## One-Hot-Encode
encoder = OneHotEncoderEstimator(inputCols=["label"],
                                 outputCols=["encoded_label"],
                                 dropLast=False)

## Scale feature vector
scaler = MinMaxScaler(inputCol="hfeatures_dense",
                      outputCol="HLF_input")

pipeline = Pipeline(stages=[encoder, scaler])

%time fitted_pipeline = pipeline.fit(data)
```

```
CPU times: user 294 ms, sys: 293 ms, total: 587 ms
Wall time: 1min 34s
```

In [12]:
```
data = fitted_pipeline.transform(data)
```

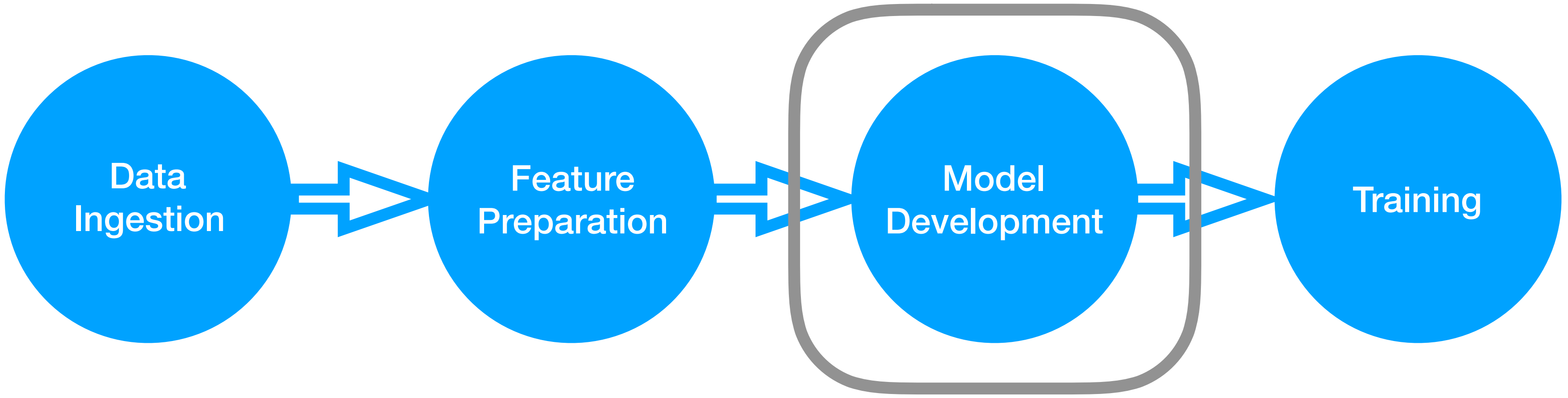
Now, for the particle-sequence classifier, we need to sort the particles in each event by decreasing ΔR distance from the isolated lepton, where

$$\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2}$$

From the production of low level we know that the isolated lepton is the first particle and the 19 features (foreach particle) are:

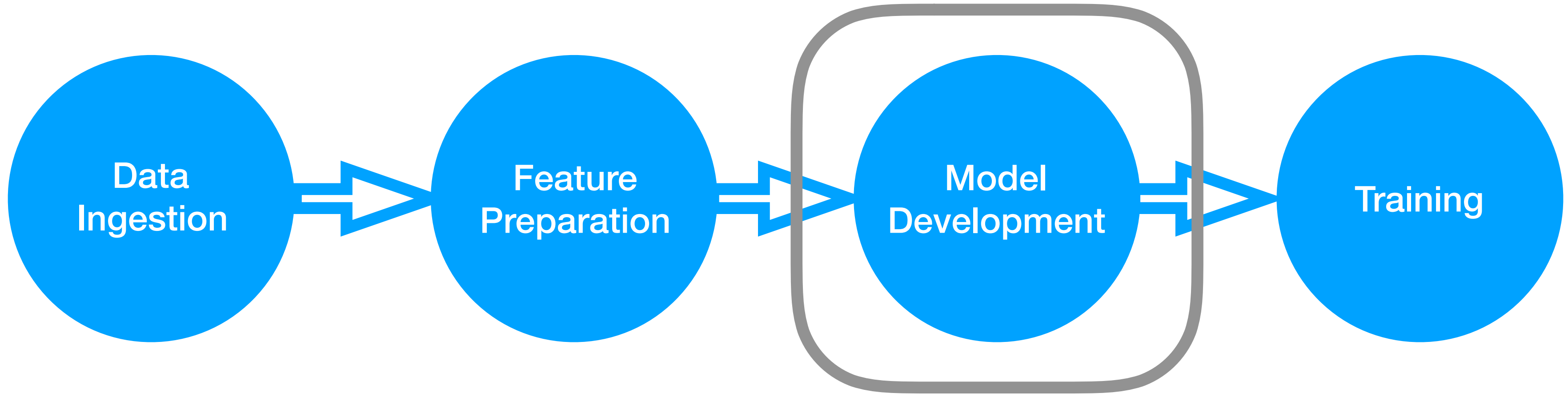
['Energy', 'Px', 'Py', 'Pz', 'Pt', 'Eta', 'Phi', 'vtxX', 'vtxY', 'vtxZ', 'ChPFIso', 'GammaPFIso', 'NeuPFIso', 'isChHad', 'isNeuHad', 'isGamma', 'isEle', 'isMu', 'Charge']

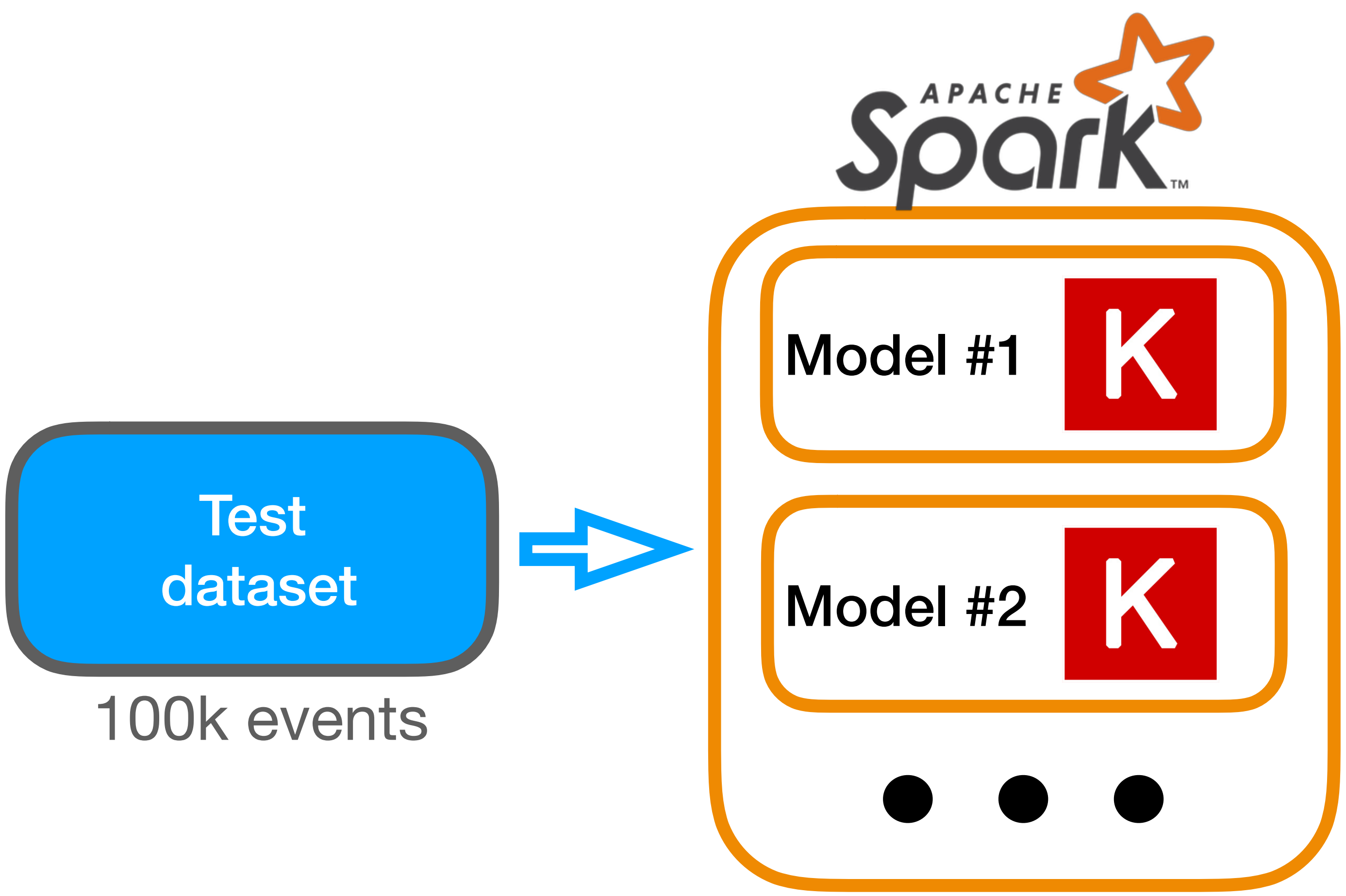
hence we need feature 5 (η) and 6 (ϕ) to compute ΔR.

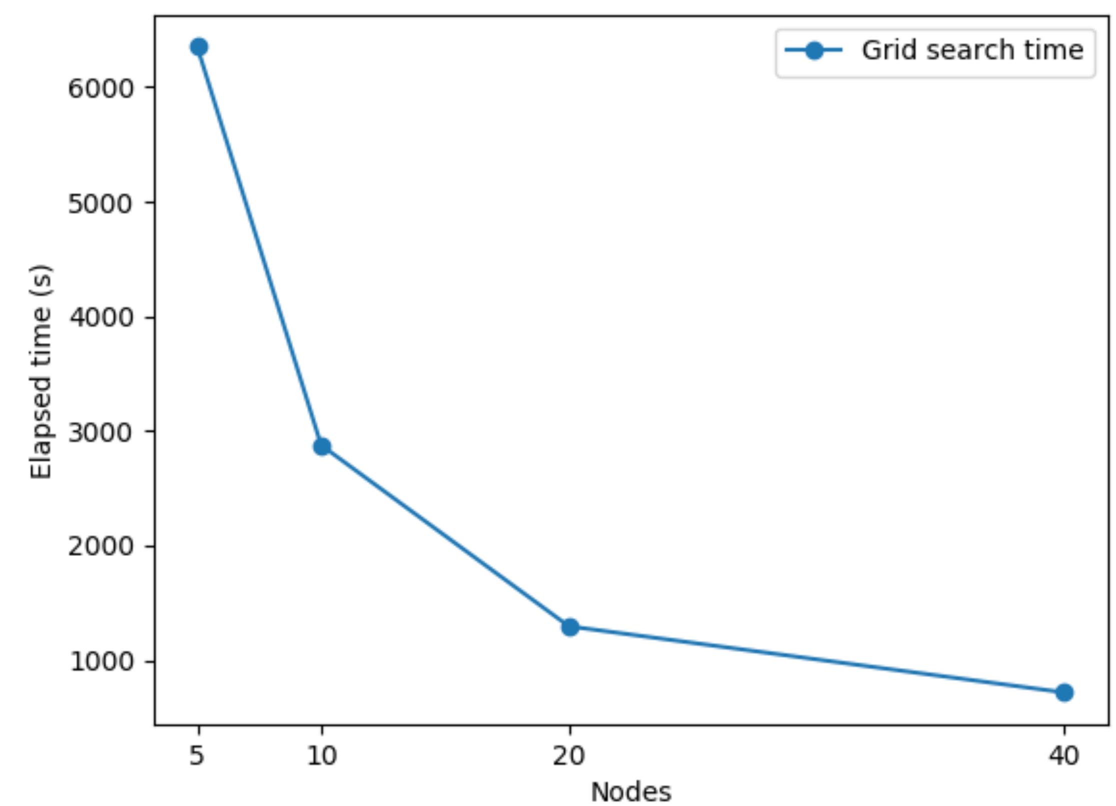

Data
Ingestion
Feature
Preparation
Model
Development
Training


Test
dataset

100k events



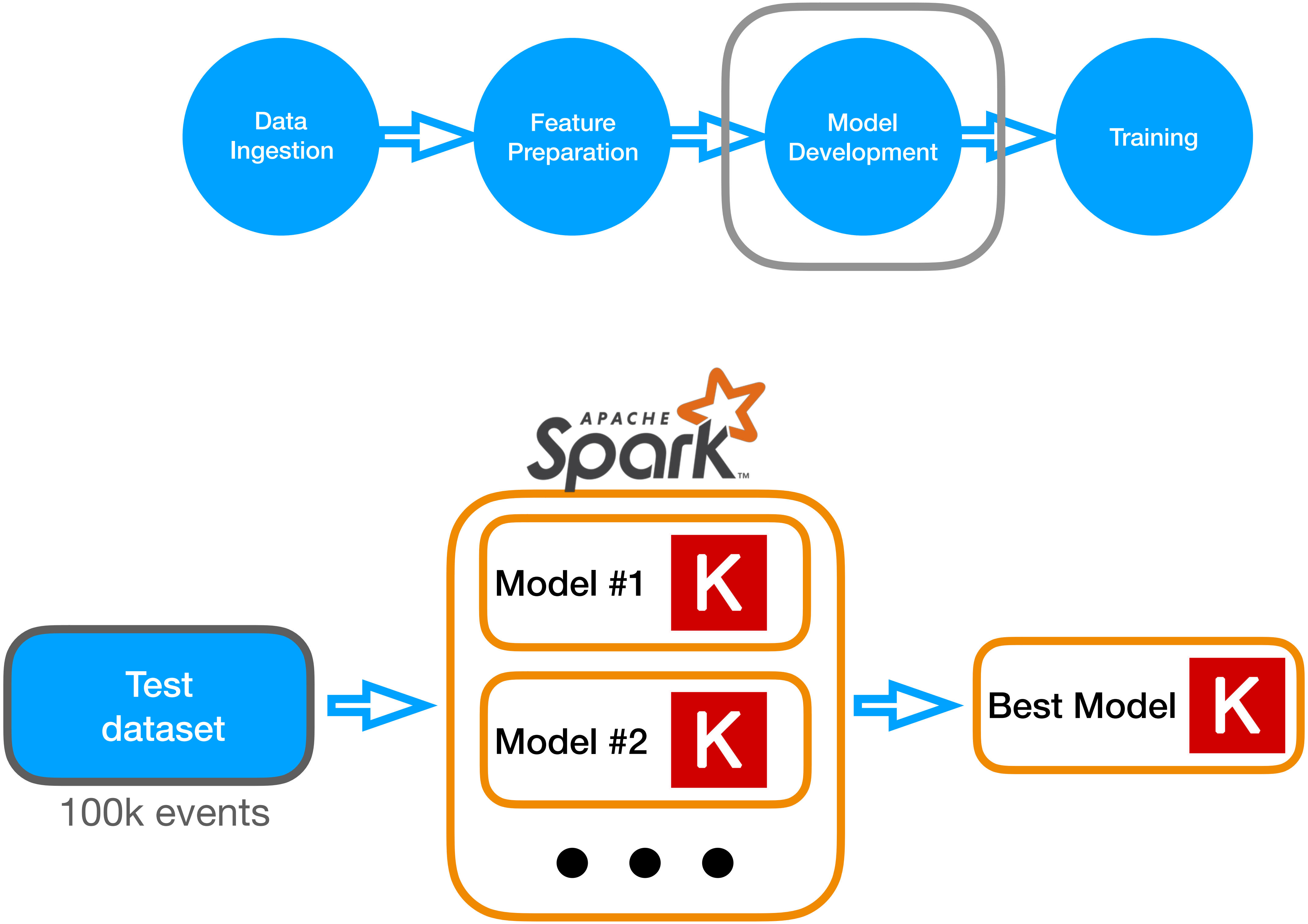


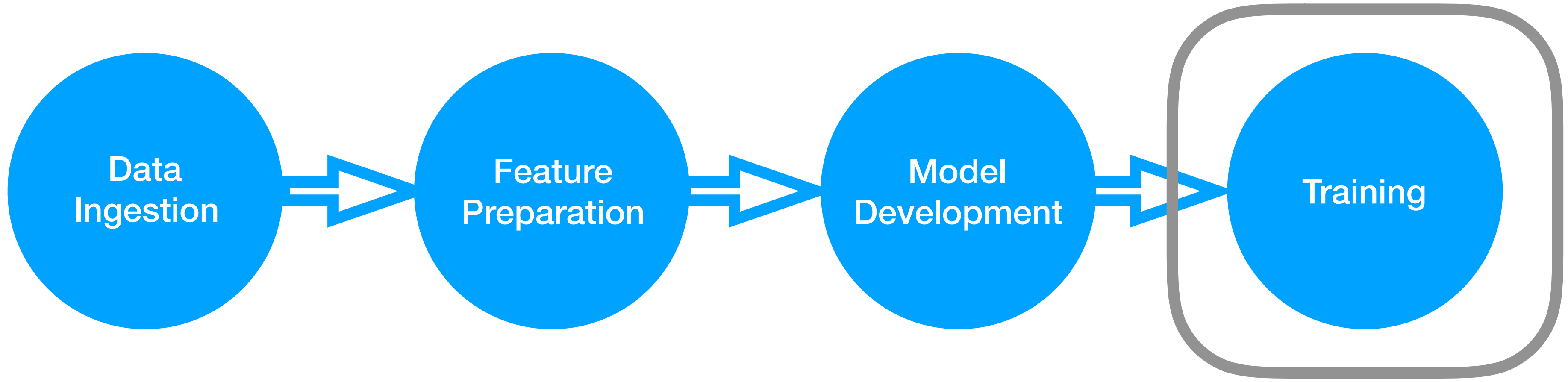


Tests made with the HLF classifier:

- Trained 162 different models changing topology and training parameters
- 3-fold cross validation

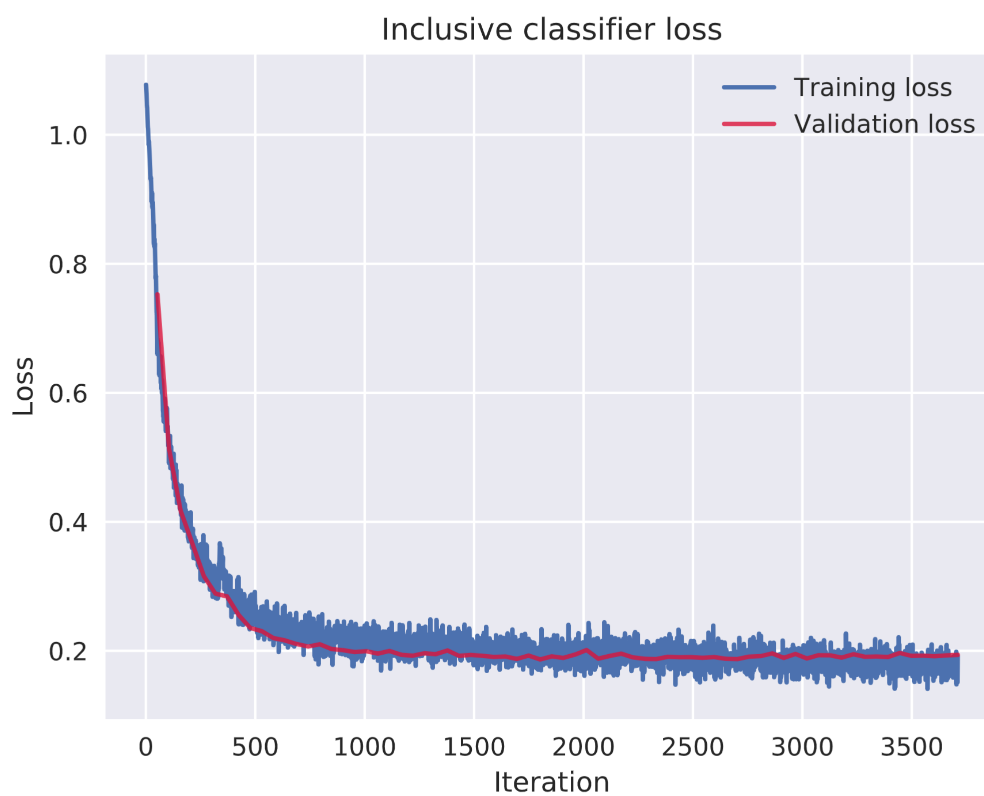
Each node (executor) has two cores


Data Ingestion
Feature Preparation
Model Development
Training
APACHE Spark
Test dataset
100k events
Model #1
K
Model #2
K
Best Model
K



Once the best model is found we can train it on the full dataset

Different tools that can be used to train the best model



Training

- Three models:
  i. High Level Feature (HLF) classifier
  ii. Particle-sequence classifier
  iii. Inclusive classifier

- Hardware and configs (at present) available for the training:

Single machine with 24 physical cores and 500GBs of RAM

+

Keras

GPU NVidia GeForce GTX1080

+

Keras

Yarn Cluster used with 22 executors, 6 cores each

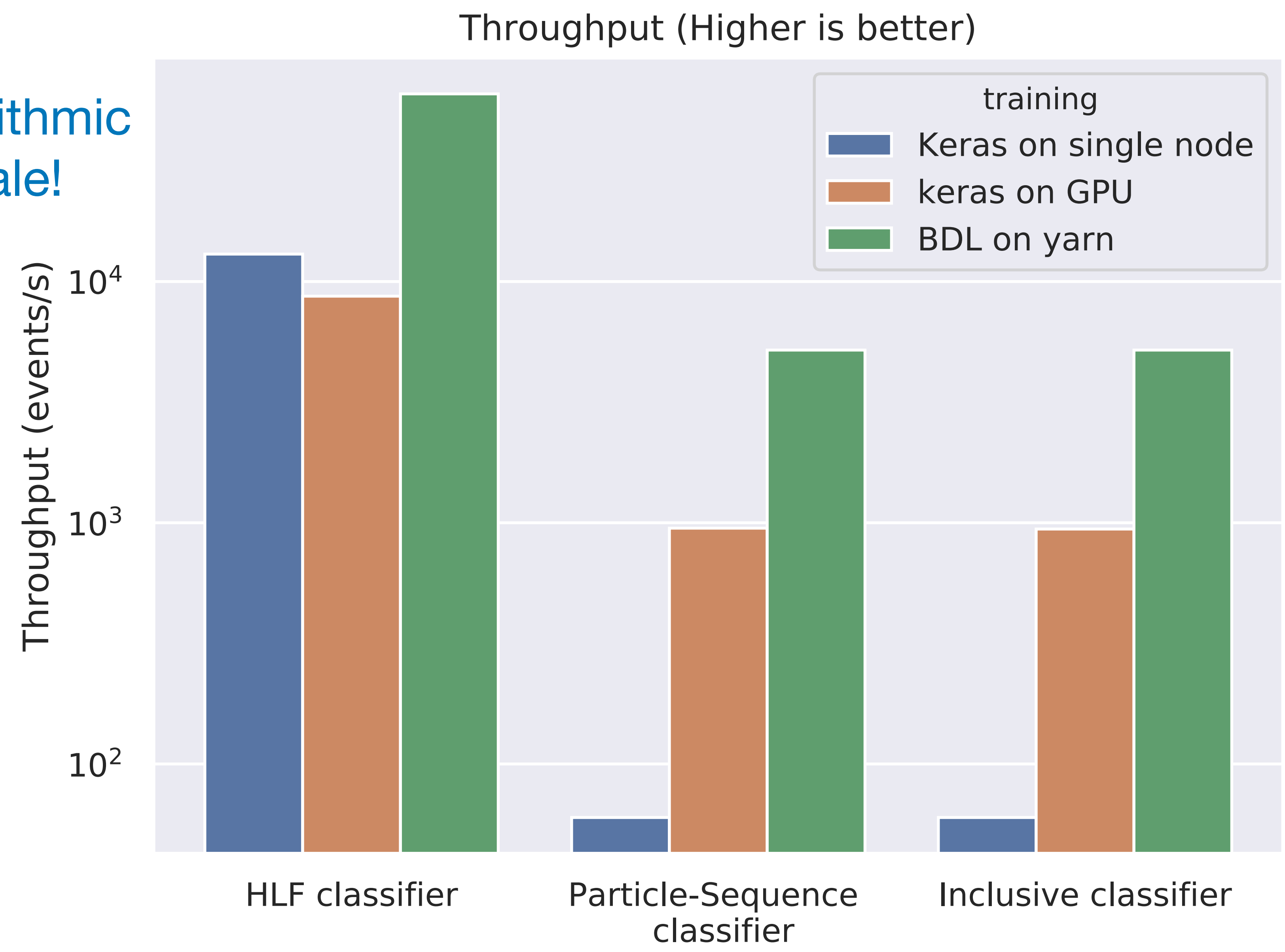
+

BigDL

Throughput Measurements

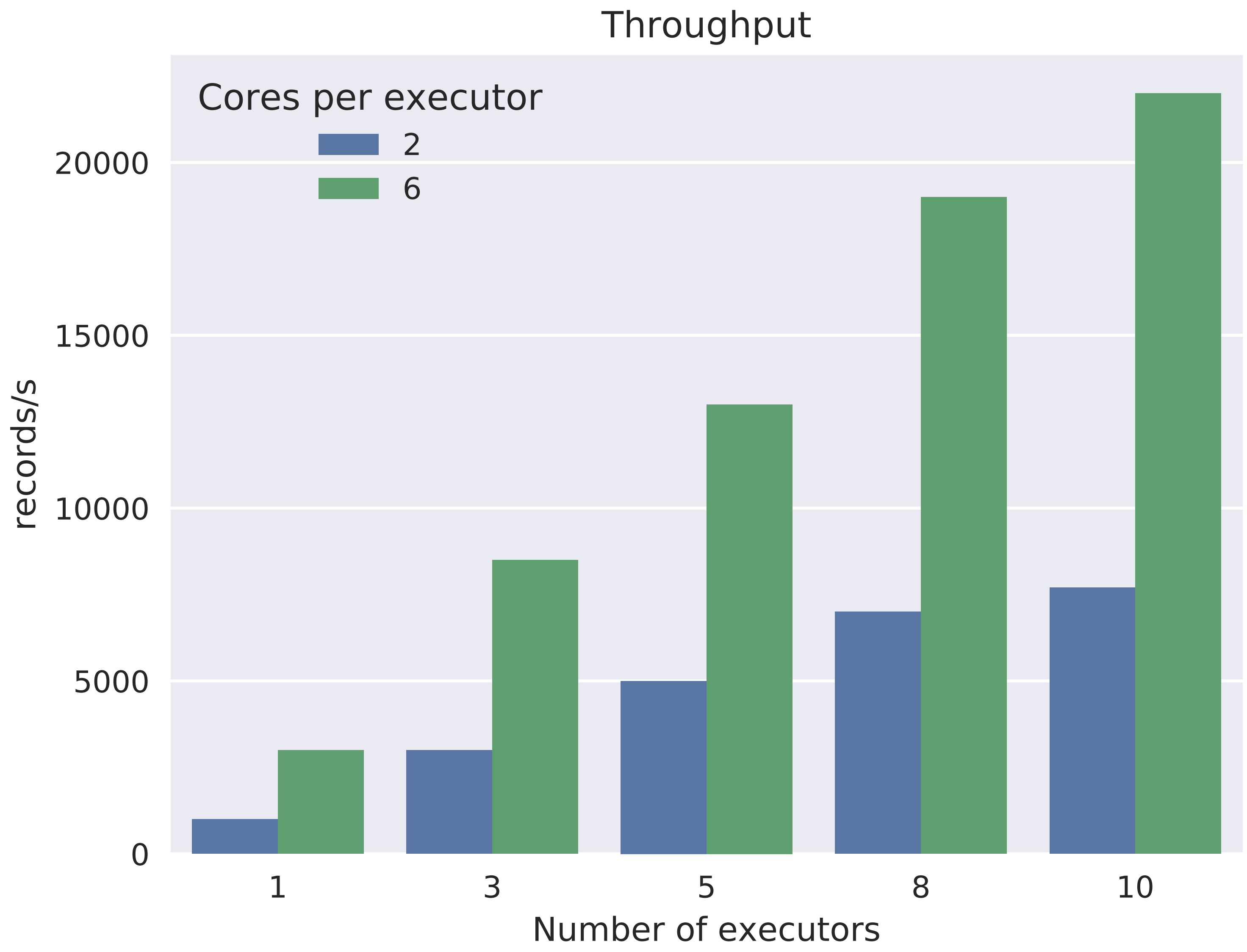
- BigDL + Spark on CPU performs and scales well for recurrent NN and deep NN.

Logarithmic scale!



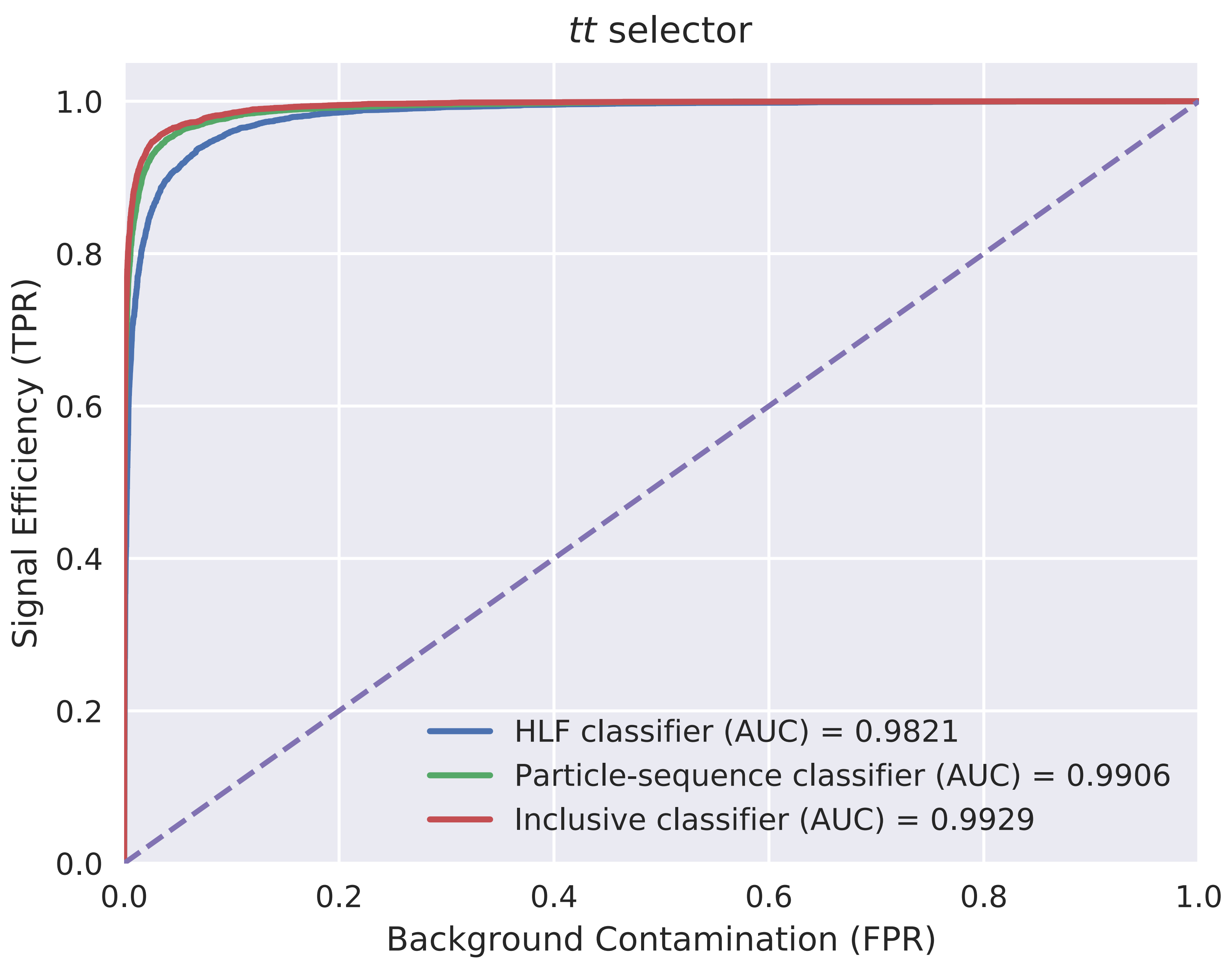
BigDL Scales

- Experiments changing the number of executors and cores per executor (HLF classifier)



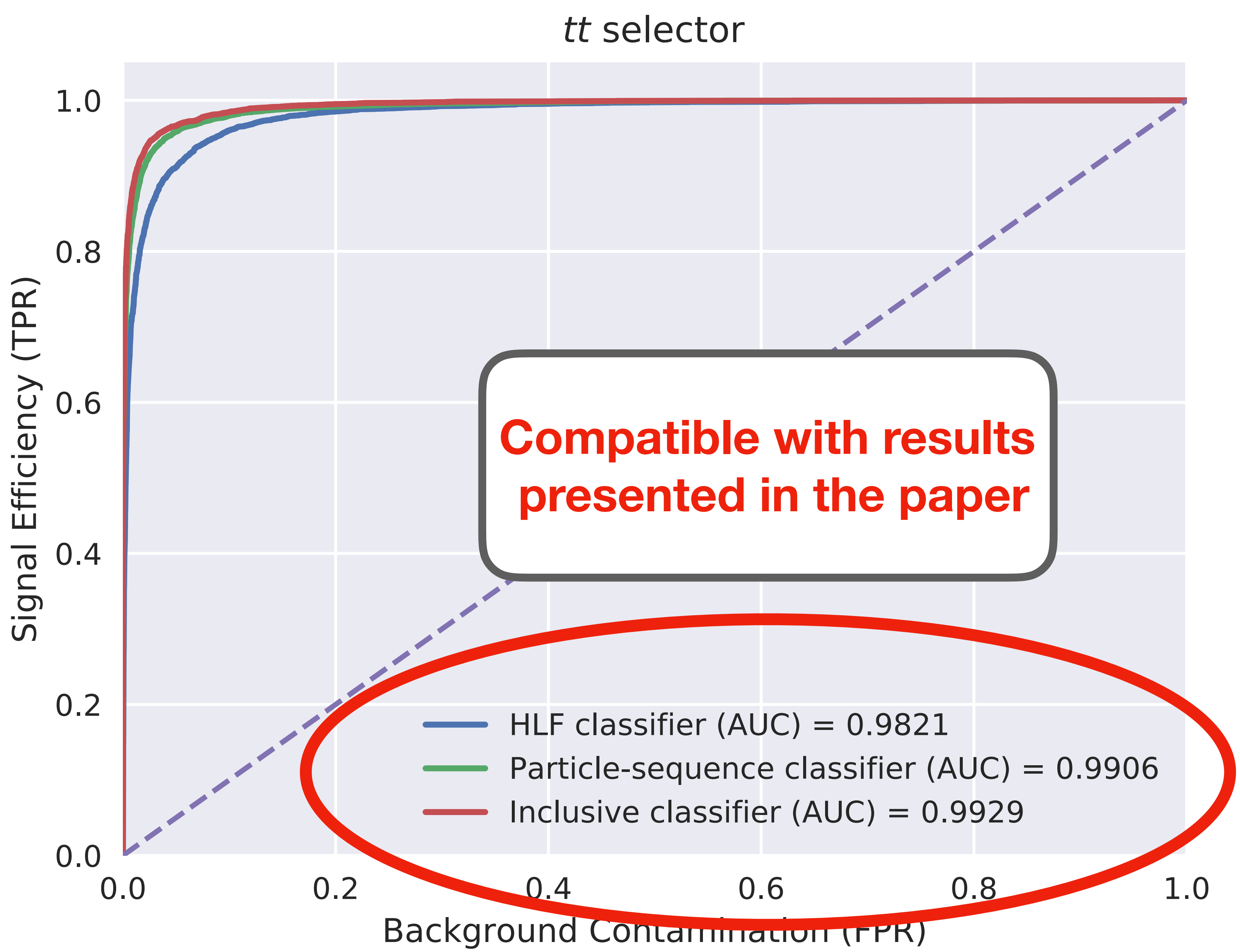
Results

- Trained models with BigDL on the Undersampled dataset (Equal number of events for each class) ~ 4M events



Results

- Trained models with BigDL on the Undersampled dataset (Equal number of events for each class) ~ 4M events



Conclusions

- Created an End-to-End scalable machine learning pipeline using Apache Spark and industry standard tools
    - Python & Spark allow to distribute computation in a simple way
    - BigDL easy to use, API similar to Keras
    - Interactive analysis using Notebooks connected to Spark
    - Easy to share and collaborate

Further work

- Spark works well for **Data Ingestion** and **Feature Preparation**
    - There is still room for improvement: with simple changes to the code it is possible to halve the time required for the feature preparation
- Test different tools and frameworks for the **Training**
    - multiple GPUs
    - Distributed Tensorflow, Kuberflow etc.
- The next step is the **Model Serving** stage:
    - After training the model we can use it to do inference on streaming data

Acknowledgement

- My supervisors Luca Canali, Marco Zanetti
- Viktor Khristenko for the help with the data ingestion stage
- Maurizio Pierini for providing the dataset and use case
- IT-DB-SAS section for providing and maintaining the clusters used in this work
- CERN Openlab
- CMS Big Data Project

Backup Slides

Training Time

Classifier	Keras Throughput (records/s)	GPU throughput (records /s)	BDL Throughput (records / s)	Time to train one epoch with BDL (s)
HLF	17500	8700	60000	66
Particle-sequence	60	950	5200	770
Inclusive	60	942	5200	770