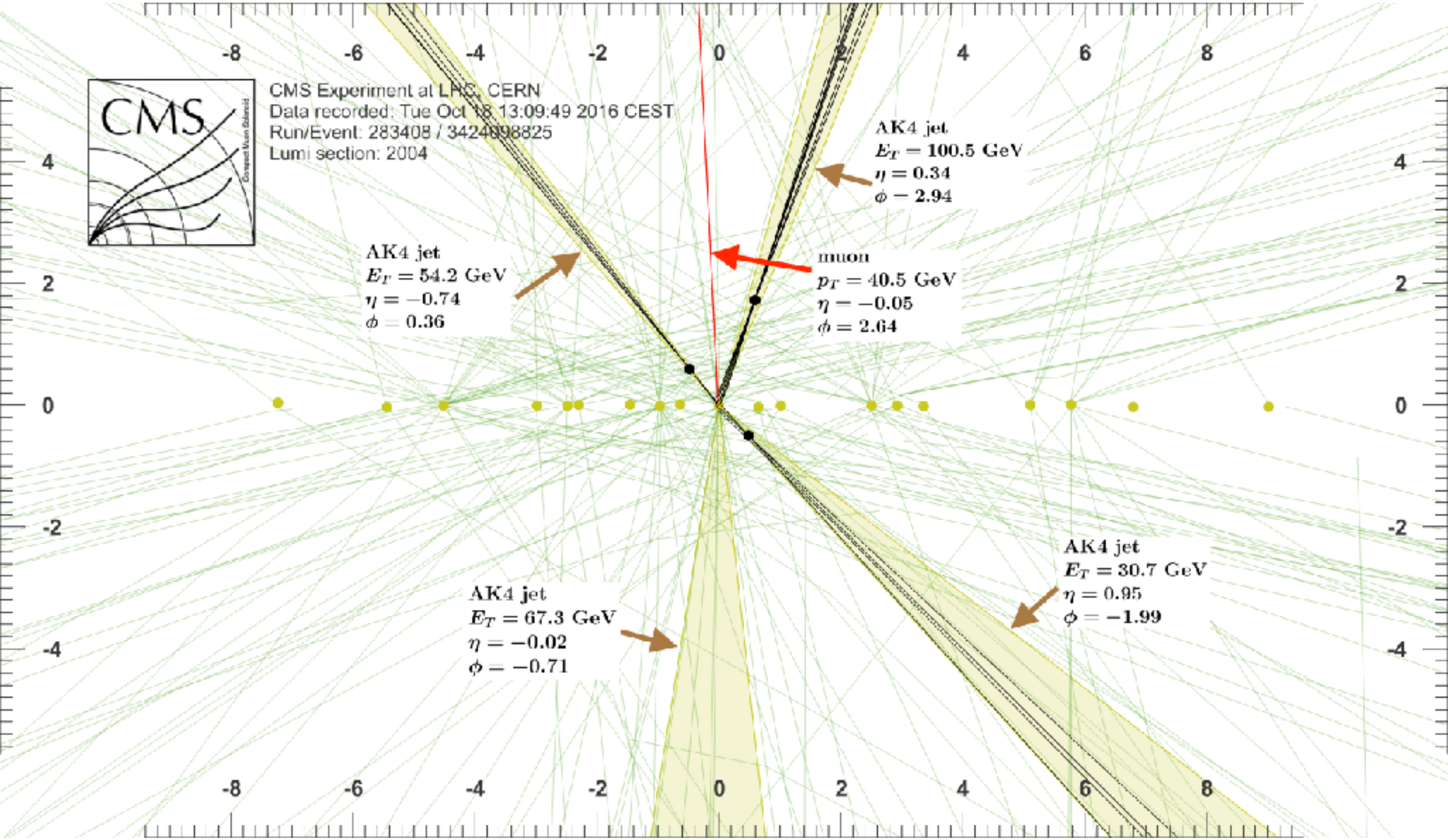
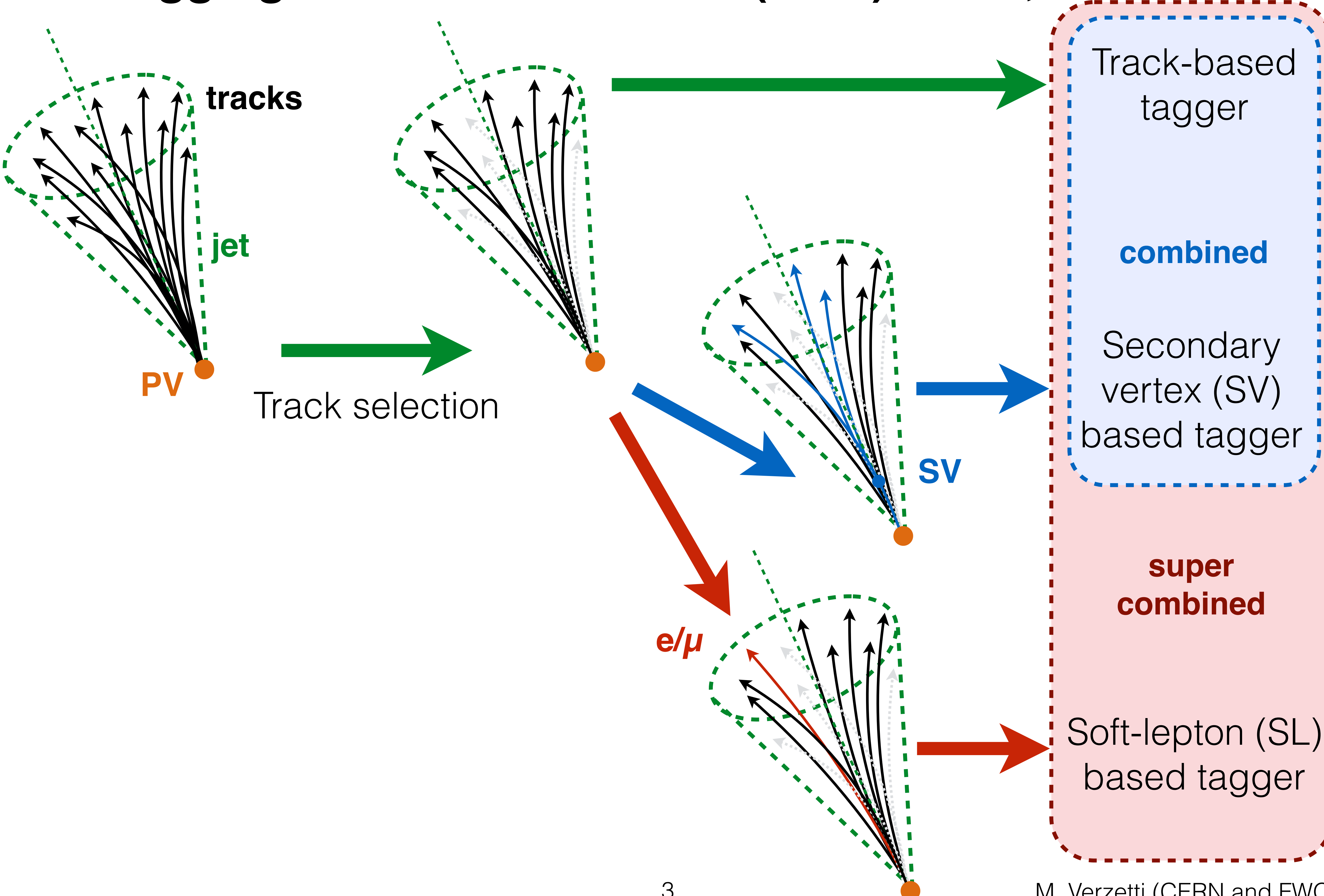
DeepFlavour and Tensorflow in CMSSW

Mauro Verzetti (CERN and FWO)
on behalf of the CMS Collaboration

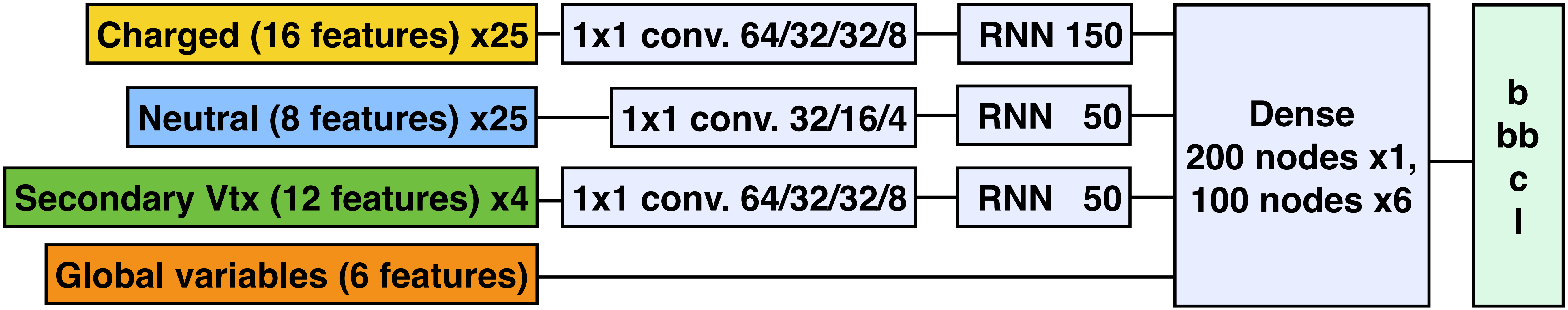
The problem - Jet flavour classification



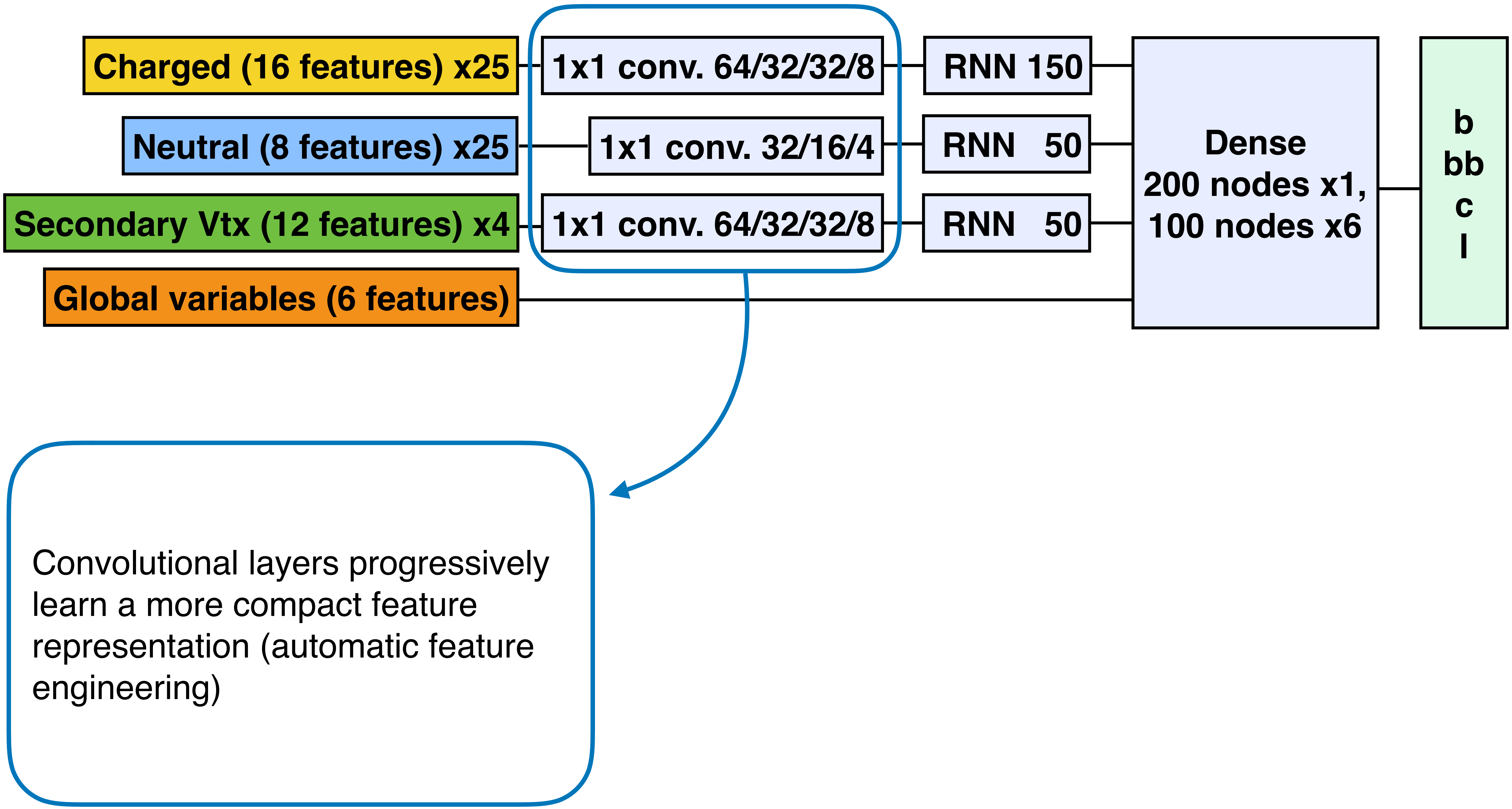
HF tagging @ CMS — JINST 13 (2018) no.05, P05011



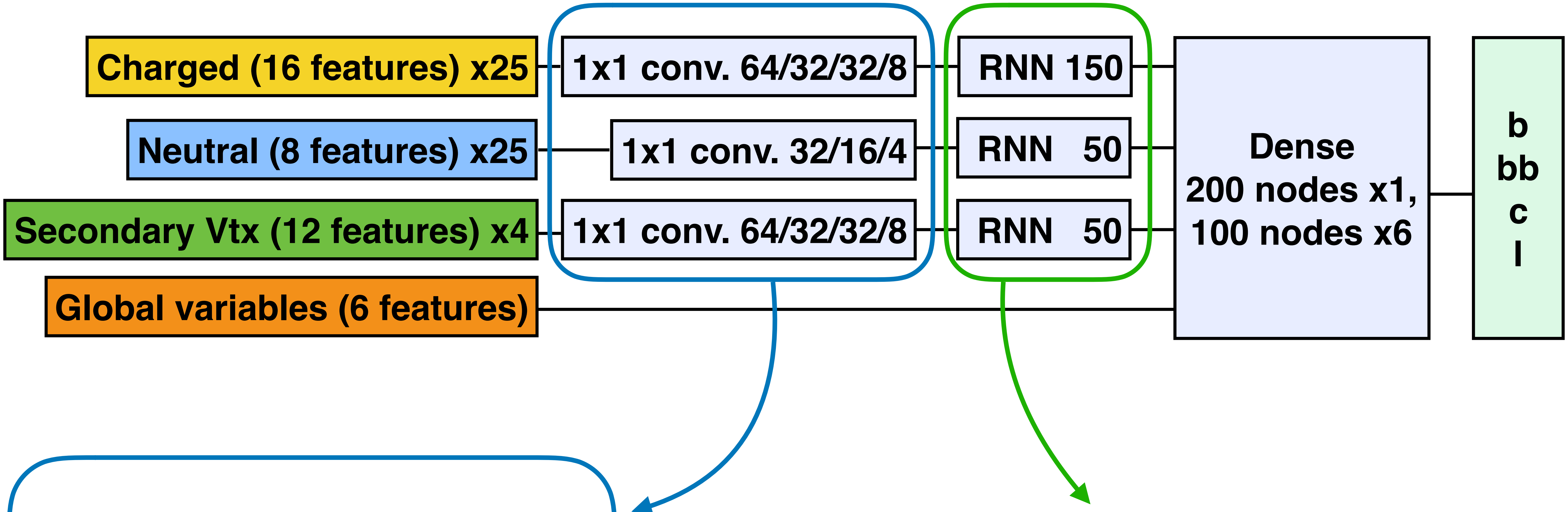
Particle-based NN architecture



Particle-based NN architecture



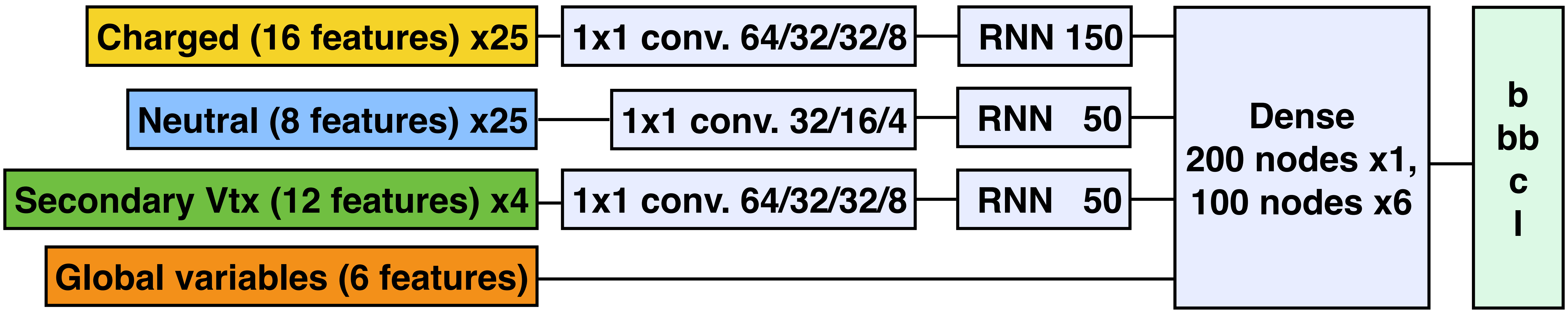
Particle-based NN architecture

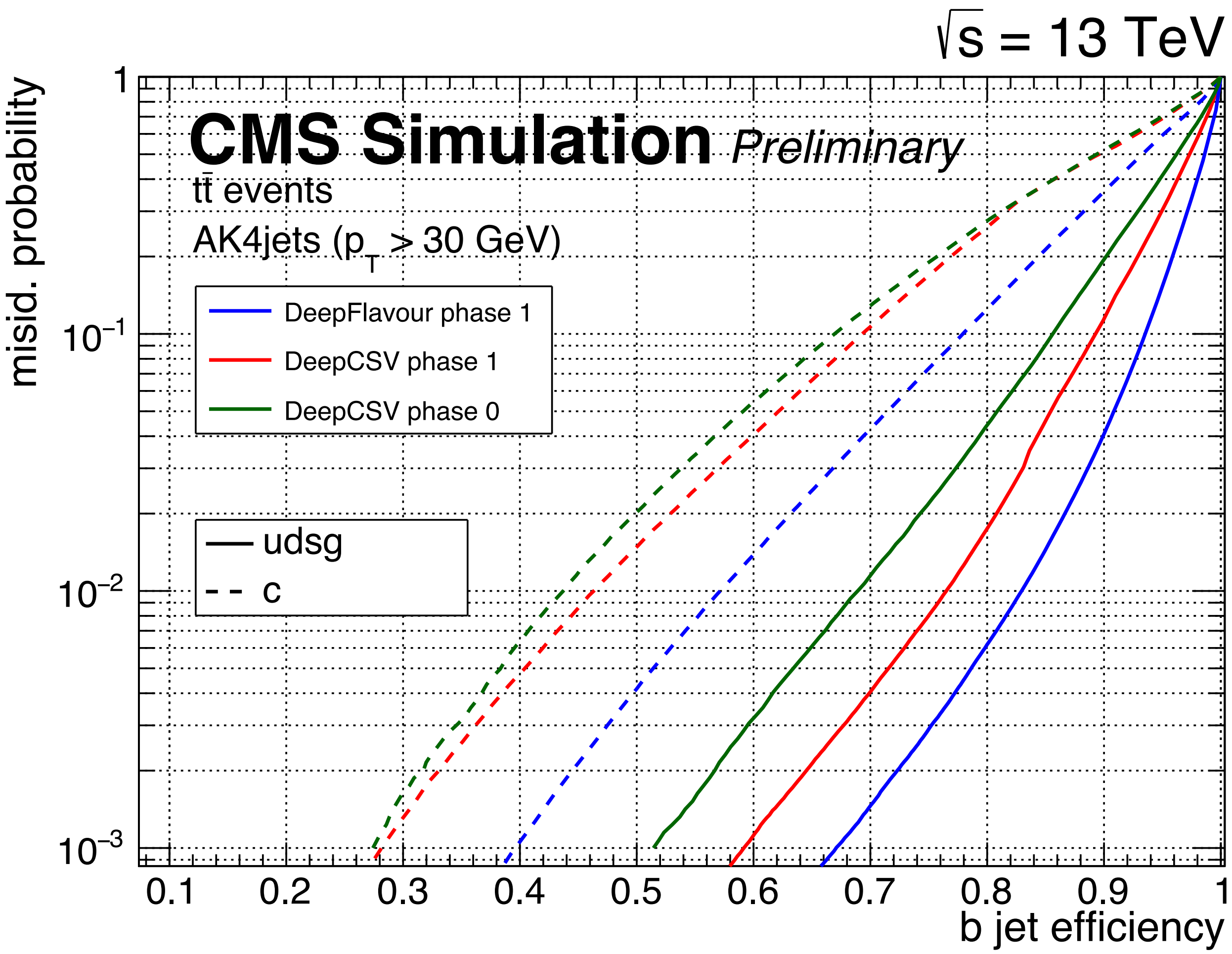


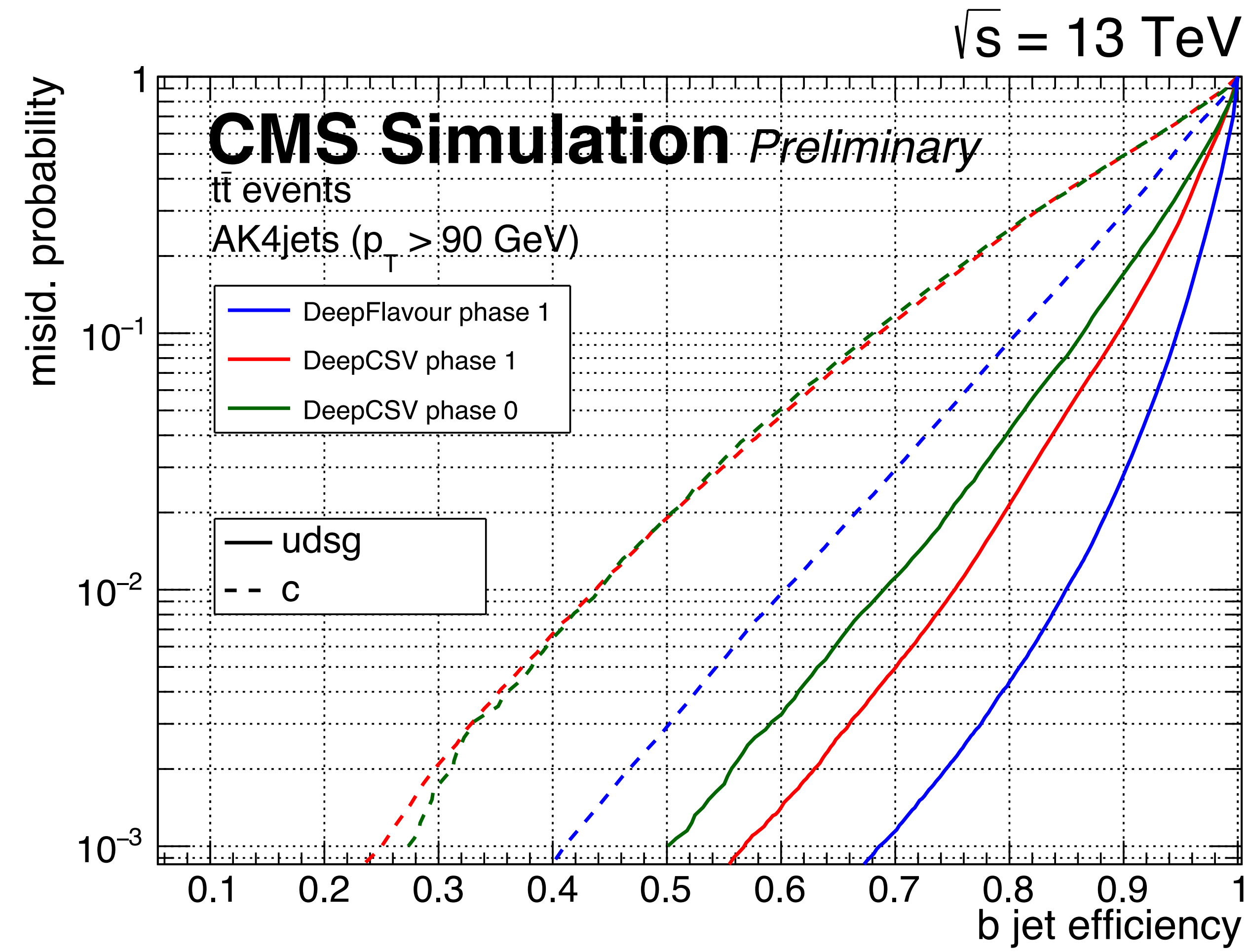
Convolutional layers progressively learn a more compact feature representation (automatic feature engineering)

The recurrent layers (LSTM) builds a "summary" of the information contained in each set of feature types

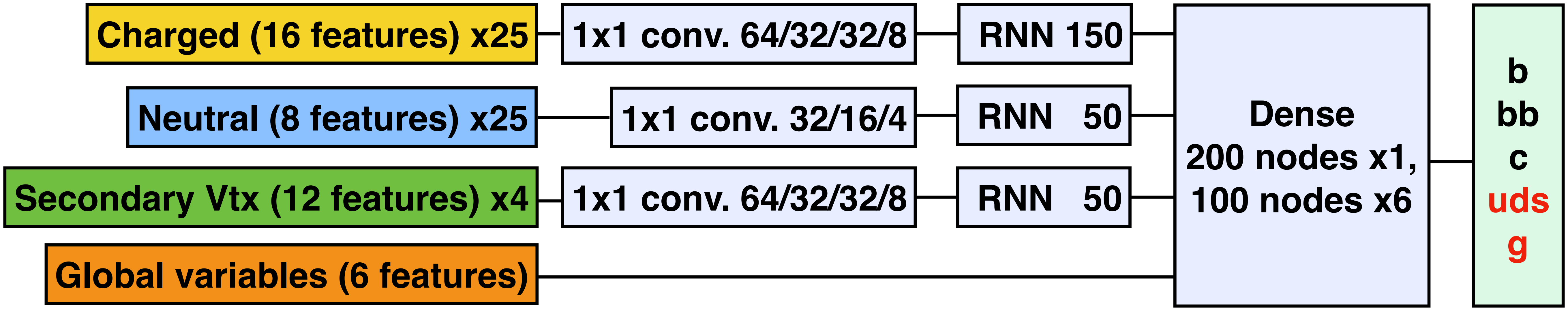
Particle-based NN architecture





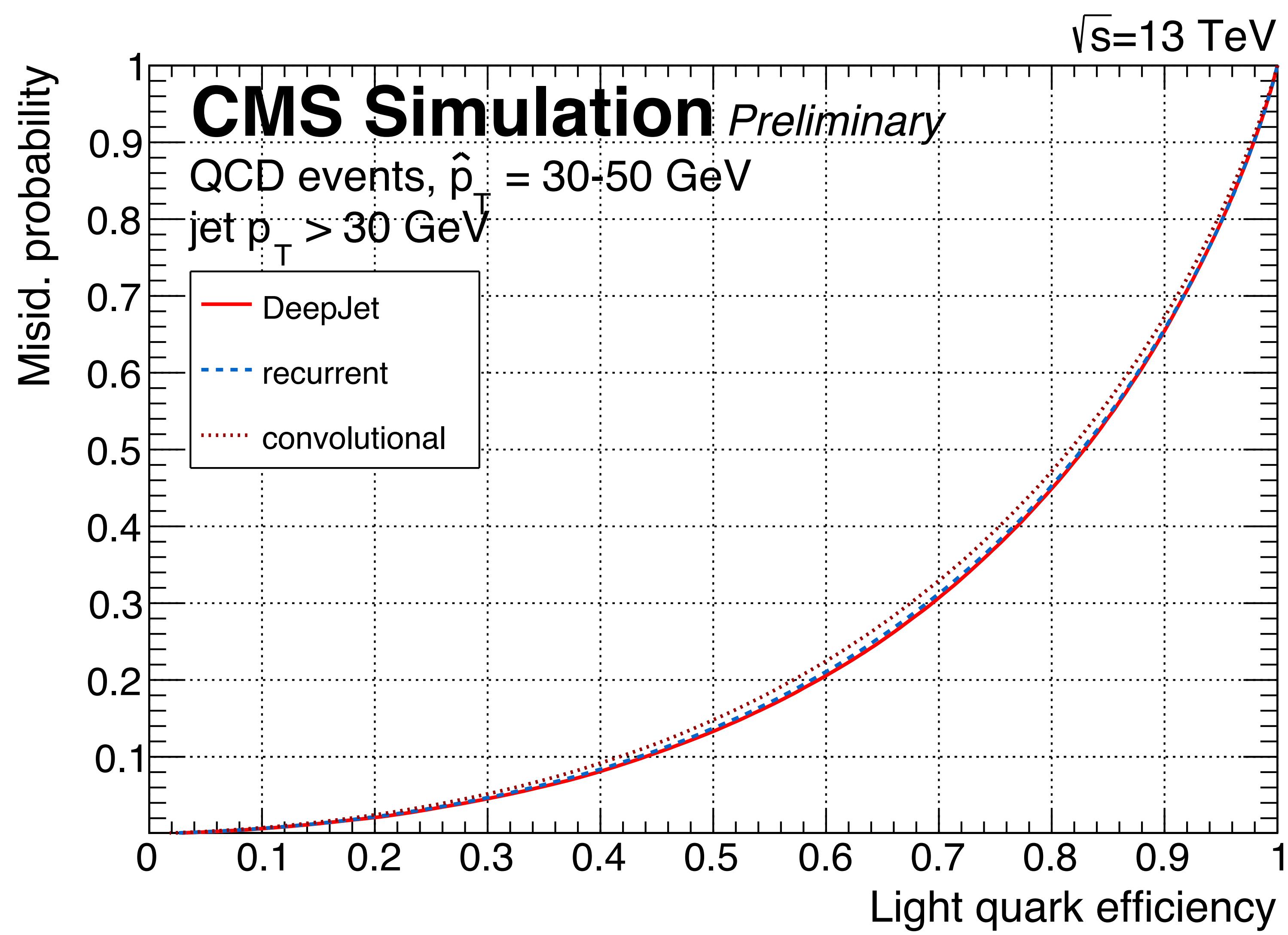


Particle-based NN architecture



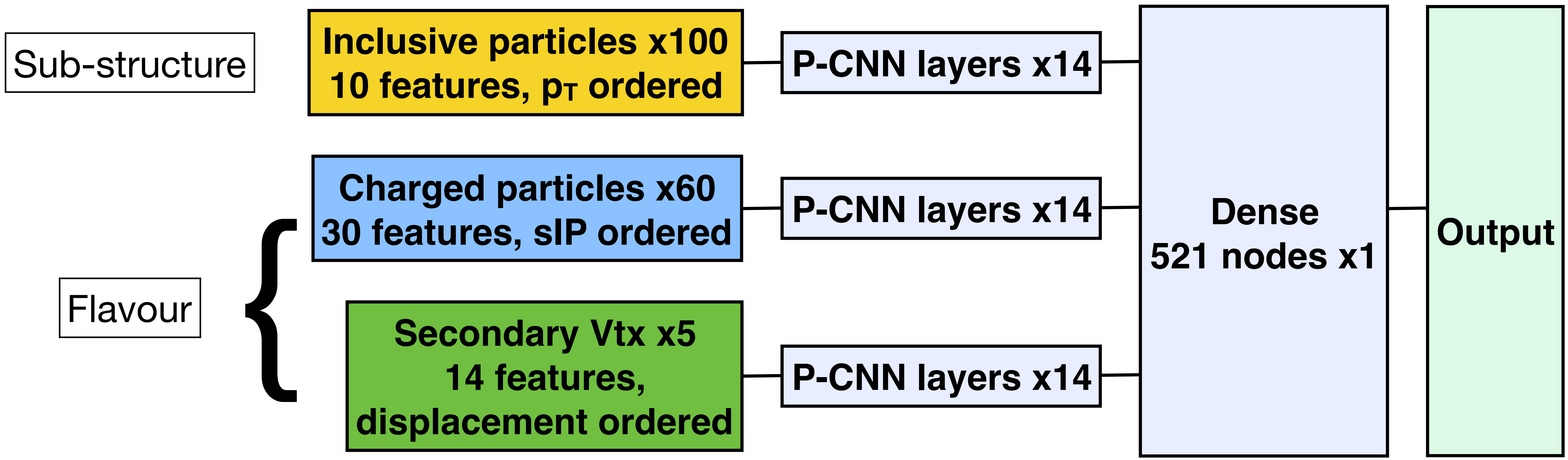
Similar performance to simpler, dedicated binary taggers, but with full multi-class power.

Significantly better performances in given regions with different quark composition



CMS-DP-2017-027

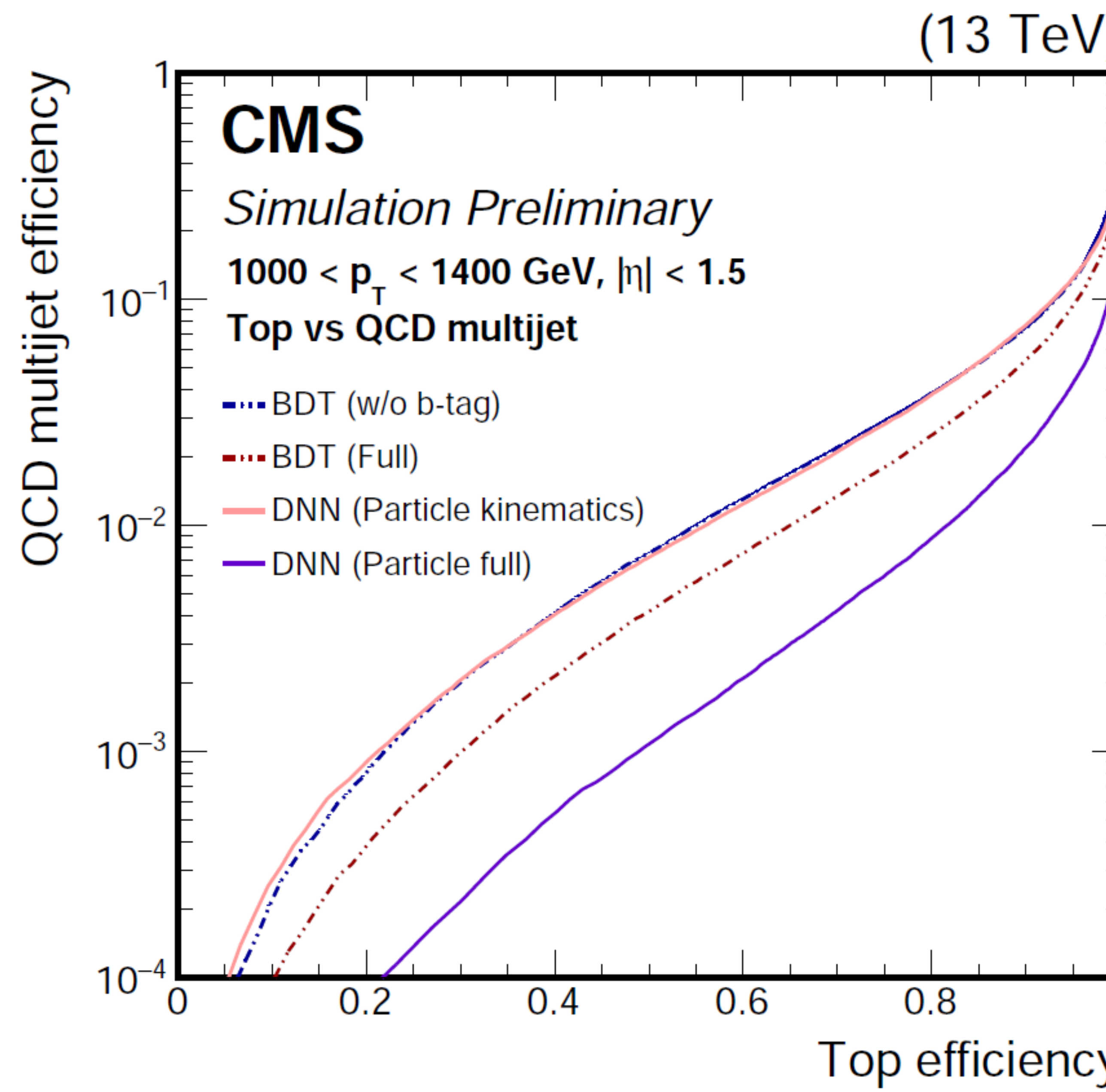
DeepAK8: for boosted resonances



- Significantly larger amount of candidates used to accomodate for 90% of the fat jets
- Need to learn substructure from both charged and neutral candidates
- RNNs become computationally too expensive to train
- Use particle-level convolutional layers (P-CNN) where each feature is treated as a “colour”

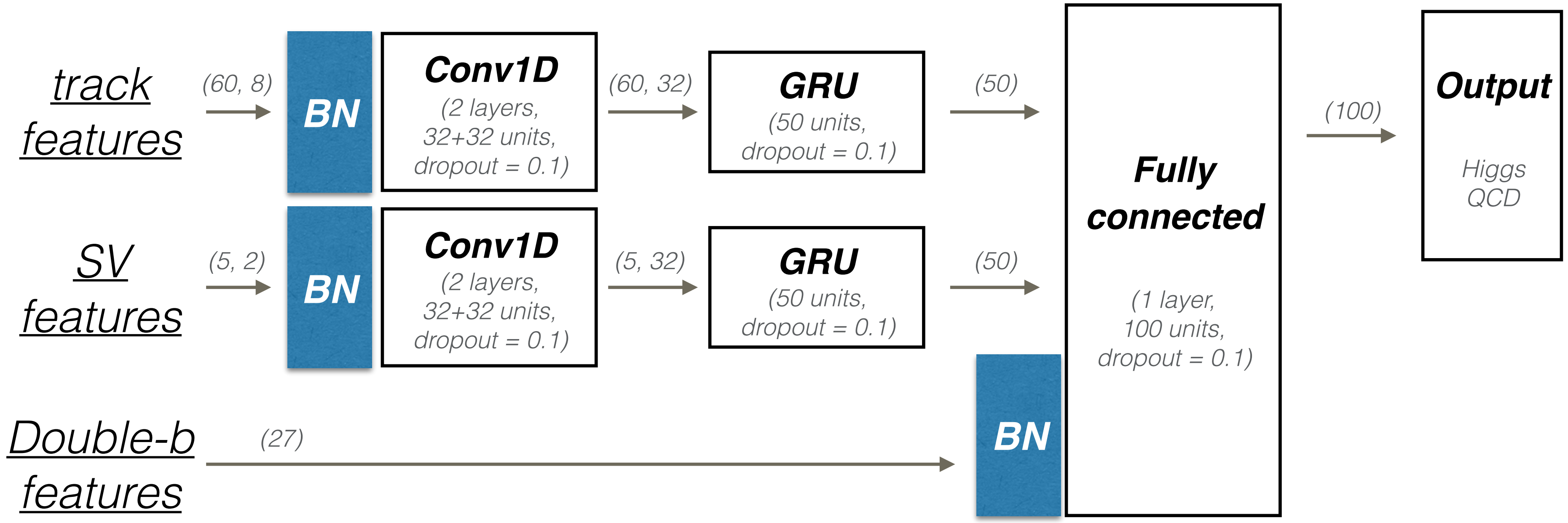
Performance

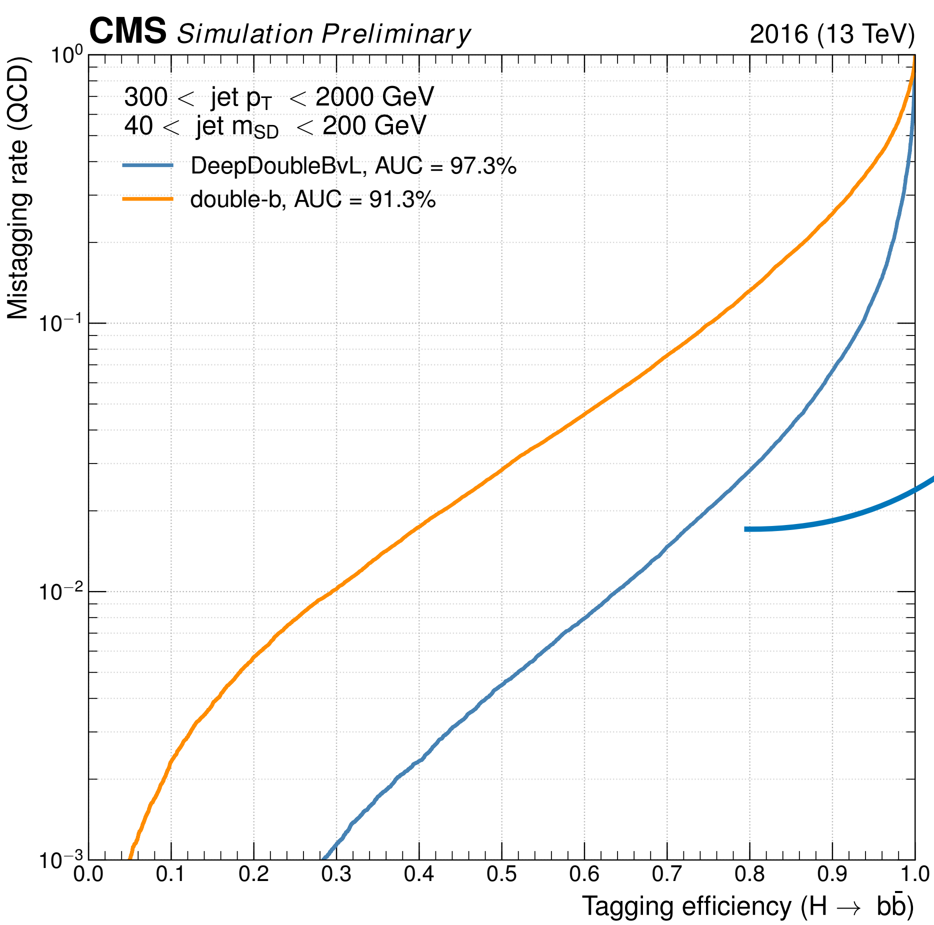
- Flavour information largely improves jet tagging
- Large improvement w.r.t to the BDT approach
- Introduces mass sculpting, not necessarily a bad thing



CMS-DP-2017-049

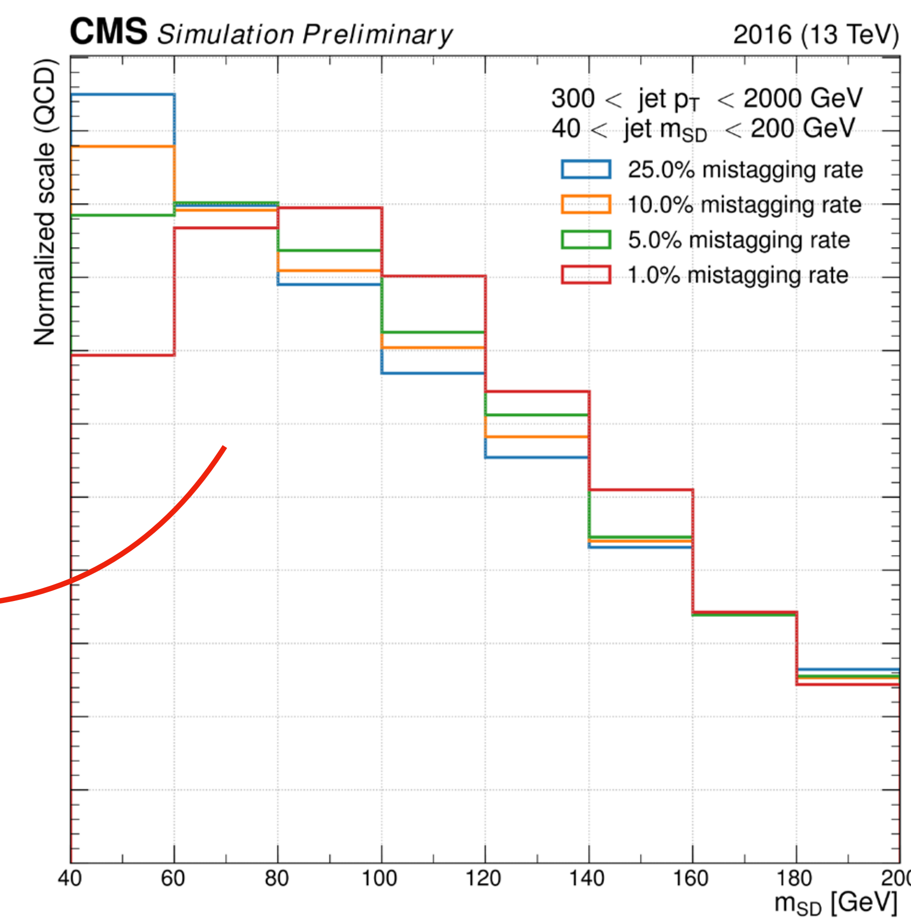
DeepDoubleB





Significantly better than current BDT approach!

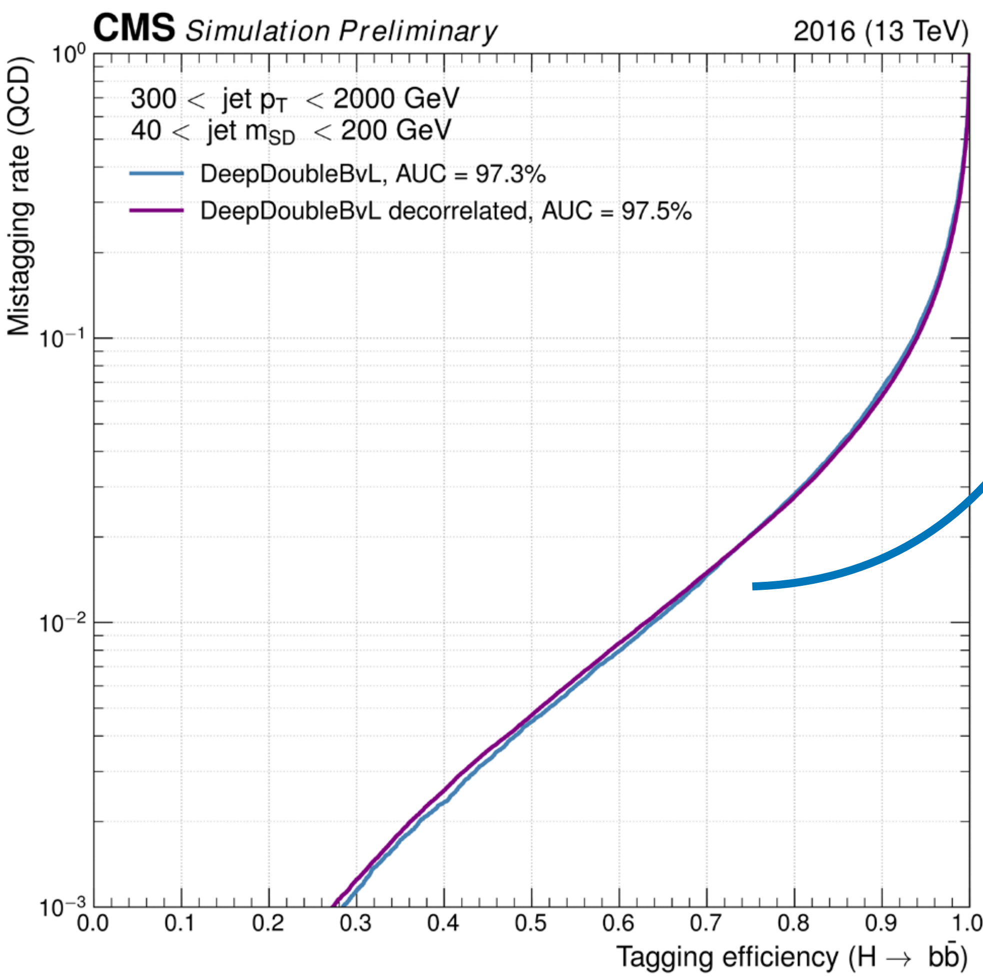
Some mass sculpting



CMS DP-2018/046

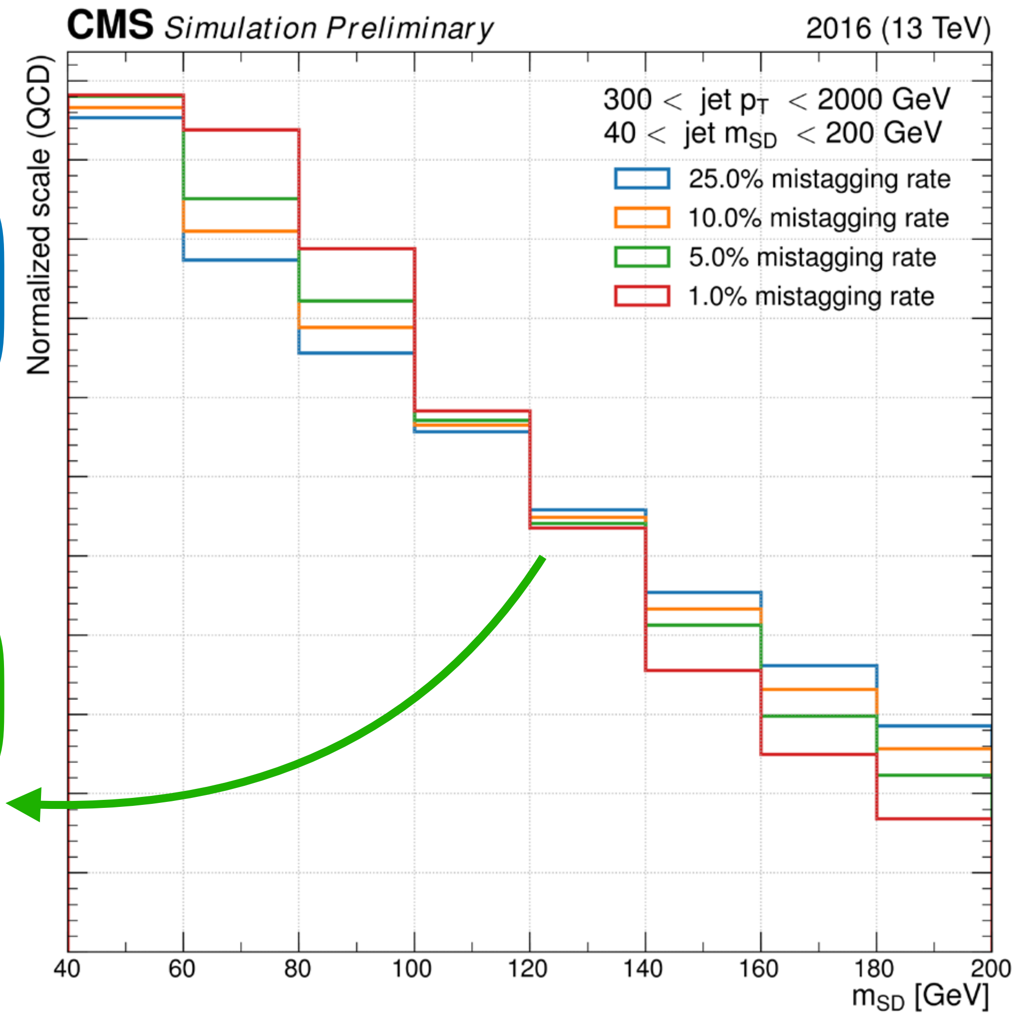
Removing mass correlation

Per-batch penalty term proportional to the Kullback-Liebler (KL) divergence



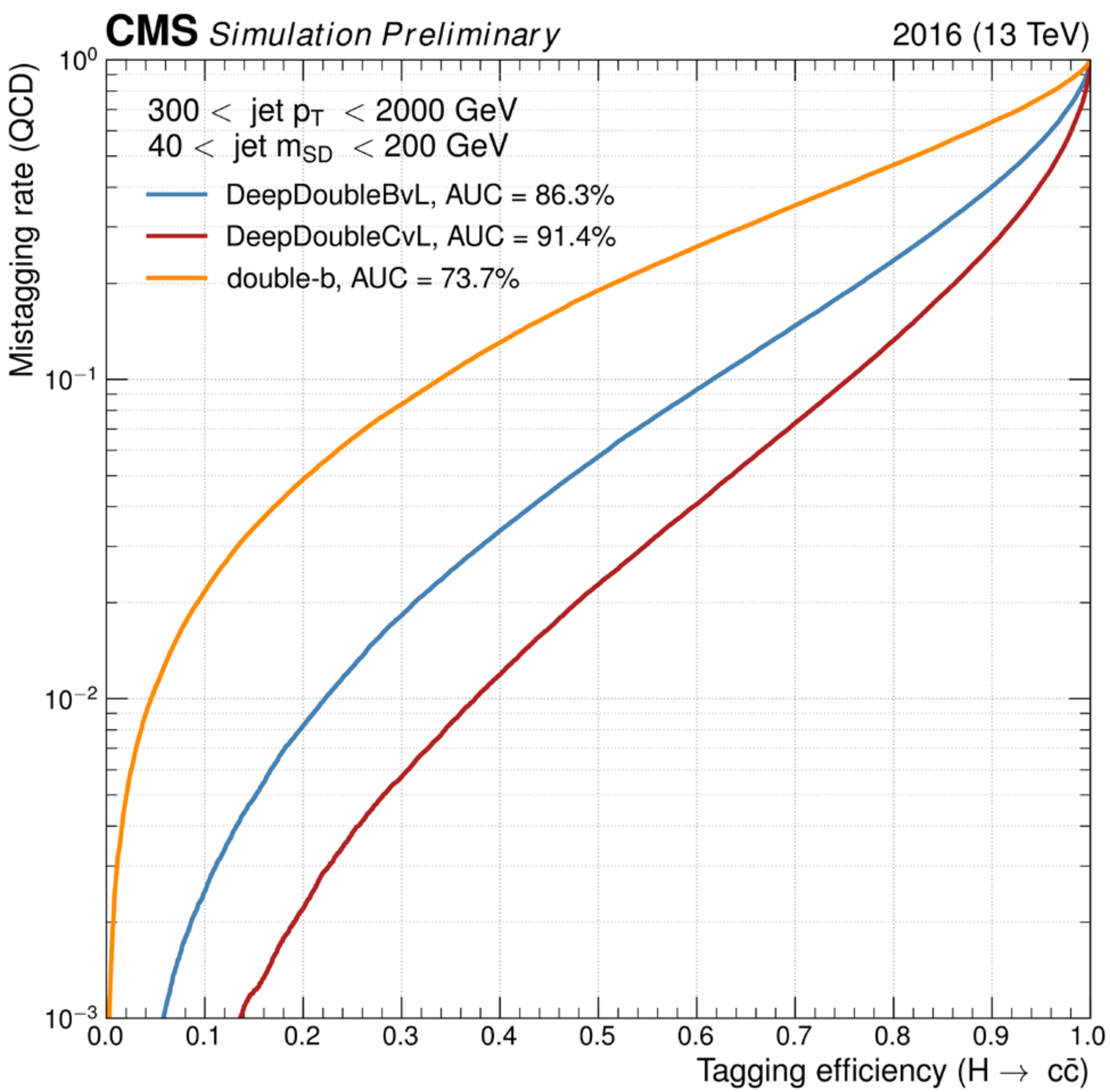
Minimal loss in performance

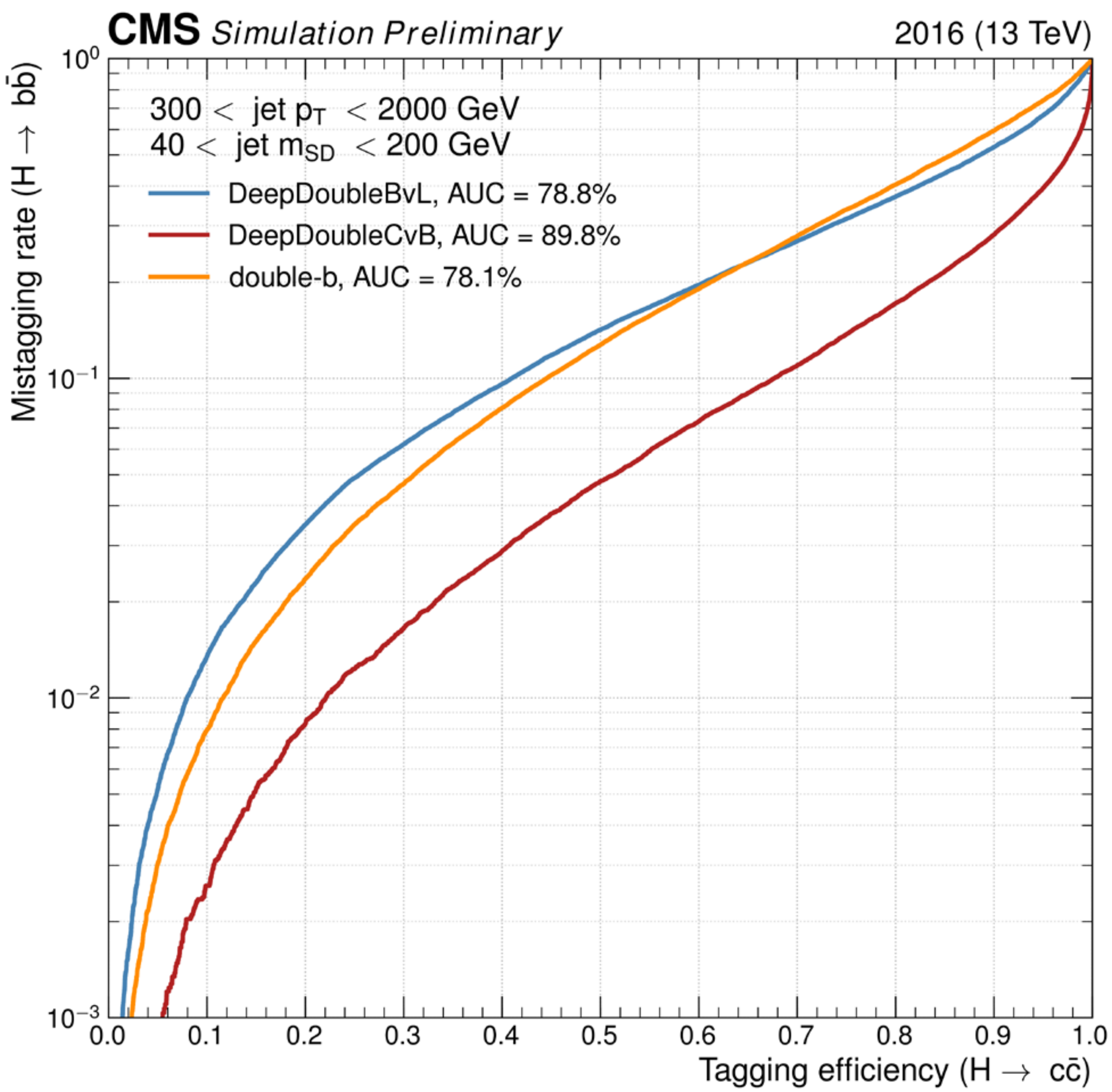
Mass sculpting gone



CMS DP-2018/046

DeepDoubleC!





CMS DP-2018/046

From training to practice

Two worlds colliding

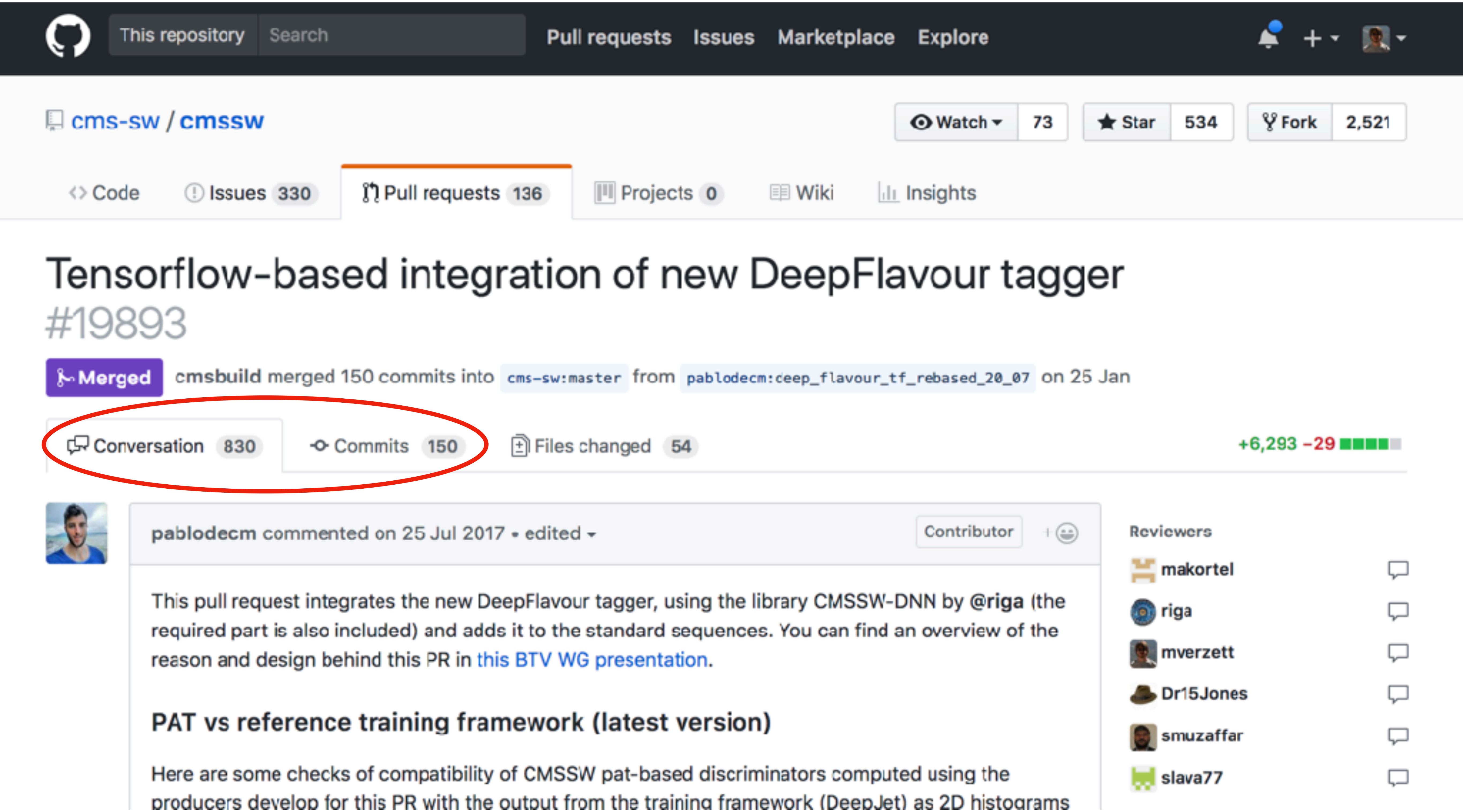
Training / Analysis:

- Keras + TensorFlow
- Python-based
- Private productions
- Minimal interaction with ROOT
- Few processes, single threads
- Little memory constraints
- Expendable jobs

Production:

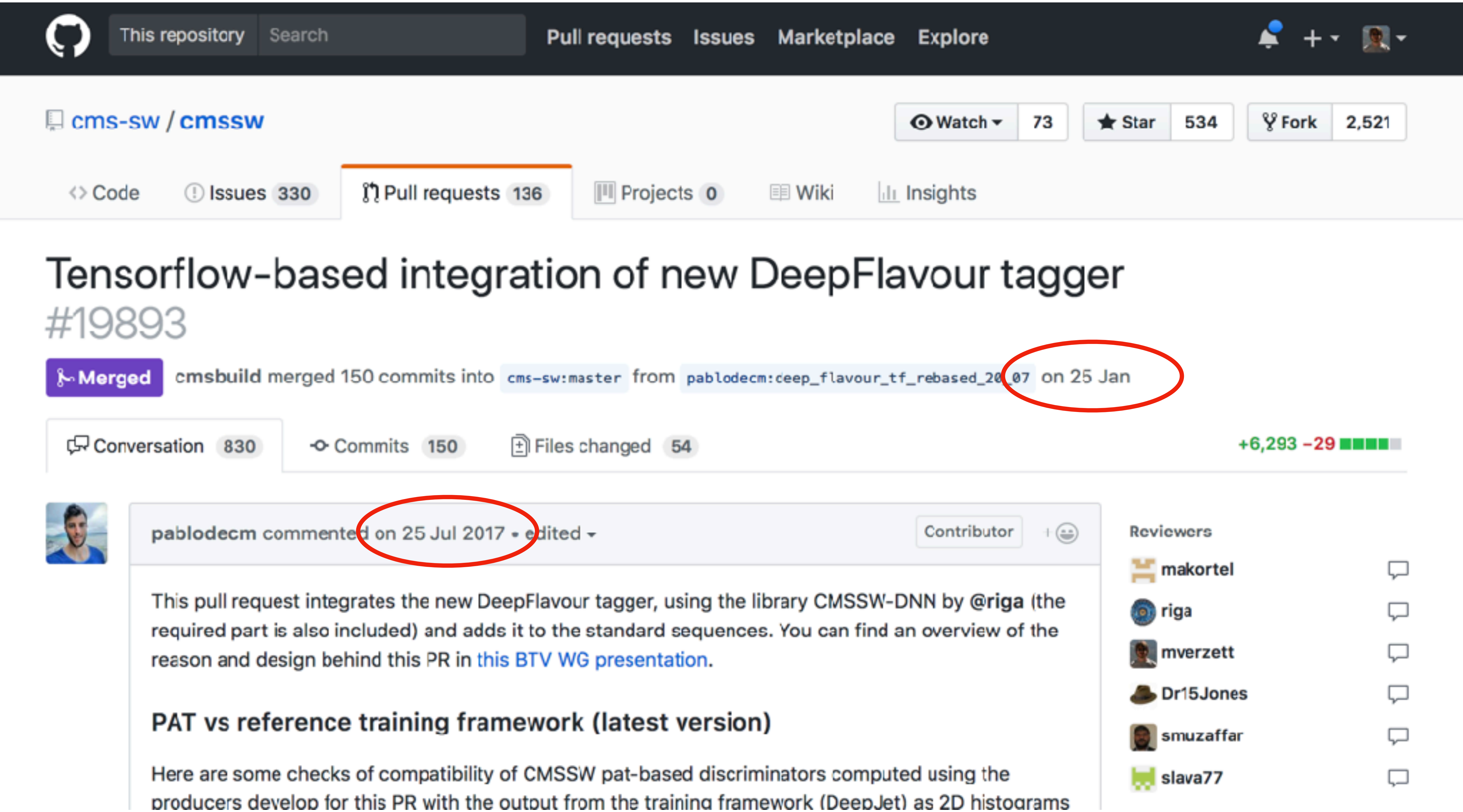
- Custom framework
- C++ based (speed!)
- Mostly ROOT-centric (at least I/O)
- Many processes, multiple threads
- Many other concurrent activities → memory constraints
- Processes cannot die (e.g. trigger)

Integration of DeepFlavour into CMSSW. PR #19893



M. Verzetti (CERN and FWO)

Integration of DeepJet (AK4) into CMSSW. PR #19893



Backend choice

✗ Interface based on TF python API:

- Uses python C API and a pre-built TF package
- Large overhead and no handle on memory/threading

✗ Interface based on TF C API:

- Low level and not very convenient
- Lots of customisations and ad-hoc handling needed

✓ Interface based on TF C++ API:

- Access to all the needed internals for production usage with minimal need for custom code
- Shallow interface to connect TF to the CMSSW internals (e.g. logging)

Issue 1: Multithreading

- TF starts **lots** of threads in its own thread pool to:
  - Faster loading of data
  - Parallelism **between** operations (`inter_op_parallelism_threads`)
  - Parallelism **within** operations (`intra_op_parallelism_threads`)
- Normally a good thing, has a critical impact on memory consumption in HEP frameworks, which have their own thread schemes/pools (CMSSW uses TBB)
- Solved with the implementation of two custom sessions:
  - **Without** any threading (NTSession)
  - **Sharing** the thread pool with the rest of the framework (TBBSession)

```
import os
import psutil
import tensorflow as tf

p = psutil.Process(os.getpid())
print(p.num_threads())          → 2

sess = tf.Session()

print(p.num_threads())          → 10
```

Issue 2: Memory footprint

- Initially DeepJet graph was **large** (~150MB)
  - Not feasible for production operations
  - Weights stored as *Variables*, which need more memory then *Constants*
  - By default Keras stores a lot of ancillary information on top of the model (operations and tensors used for training, optimiser status etc.)
- Reduction of O(10-100) by removing things not needed for inference and converting to constants
- Further reduction: one single computation graph loaded and shared across threads, multiple sessions computing inference
- In the future: AOT compilation?

A new kid in town: MXNet

- Used to train DeepAK8
- Found to be 2-3x faster to train that type of model
- Model exported with by the gluon API
  - no post-processing needed
  - outputs a son describing the network architecture and a binary containing the weights
- “Simple” inference engine: one .h + one .so file

MXNet integration

Include MXNet as an external #21314

Open makortel opened this issue on 15 Nov 2017 · 17 comments

makortel commented on 15 Nov 2017 Contributor

Add MXNet prediction tool #4167

Merged cmsbuild merged 2 commits into `cms-sw:IB/CMSSW_10_2_X/gcc630` from `hqucms:mxnet/IB/CMSSW_10_2_X/gcc630` on 11 Jul

Conversation 26 · Commits 2 · Checks 0 · Files changed 3

hqucms commented on 3 Jul Contributor

Reviewers

No reviews

DeepAK8 tagger integration #23768

Merged cmsbuild merged 29 commits into `cms-sw:master` from `hqucms:deep-boosted-jets` on 11 Sep

Conversation 195 · Commits 29 · Checks 0 · Files changed 57

hqucms commented on 9 Jul Contributor

MXNet issues - thread safety

Problem: the simple "NaiveEngine" is not thread safe. Meant to run as a static singleton and only allows for synchronous operations.

- The other engine type "ThreadedEngine" is thread safe, but spawns its own thread pool

Solution: make the engine "thread_local"

- Only 2 lines of code changed
- Not without potential dangers

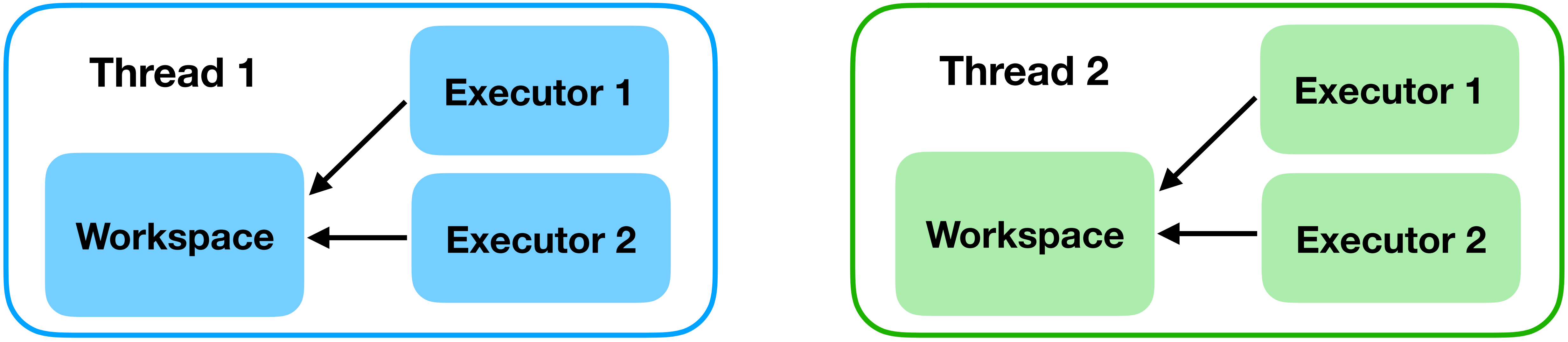
MXNet issues - thread safety

Problem: once the previous problem was fixed, the resulting output was still sometimes inconsistent — race condition

- For each thread, mxnet creates creates a workspace to store temporary variables (partially computed results).
- If multiple graphs are run simultaneously there might be a race condition, but this does should not happen since each thread runs one graph at a time…
- …unless the object computing the graph is reattached to a different thread every time, in a worker-pool design!

Solution: re-attach the workspace every time before running the computation graph (mxnet patch)

- Only 2 lines of code changed
- Additional mutex added when initialising the Executor as helgrind reported race conditions in initialisation (minimal cost)



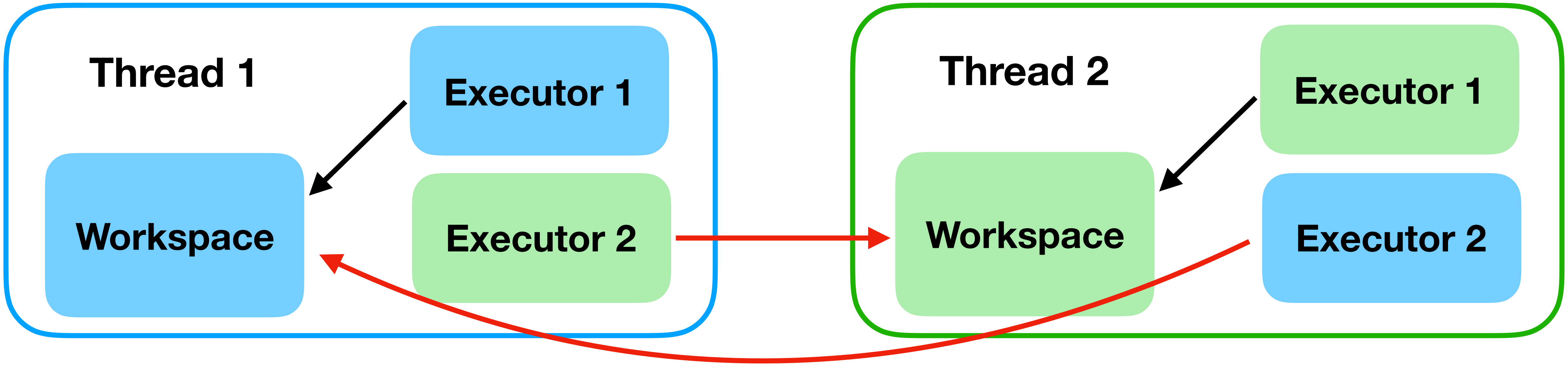
MXNet issues - thread safety

Problem: once the previous problem was fixed, the resulting output was still sometimes inconsistent — race condition

- For each thread, mxnet creates creates a workspace to store temporary variables (partially computed results).
- If multiple graphs are run simultaneously there might be a race condition, but this does should not happen since each thread runs one graph at a time…
- …unless the object computing the graph is reattached to a different thread every time, in a worker-pool design!

Solution: re-attach the workspace every time before running the computation graph (mxnet patch)

- Only 2 lines of code changed
- Additional mutex added when initialising the Executor as helgrind reported race conditions in initialisation (minimal cost)



MXNet vs. TensorFlow

MXNet

- Lighter dependencies (only a BLAS library)
- Simple model export for inference
- Official ONNX integration
- **New:** Keras compatible

TensorFlow

- Better thread safety
- Significantly better operators support and coverage (always cutting edge)
- Native Keras support

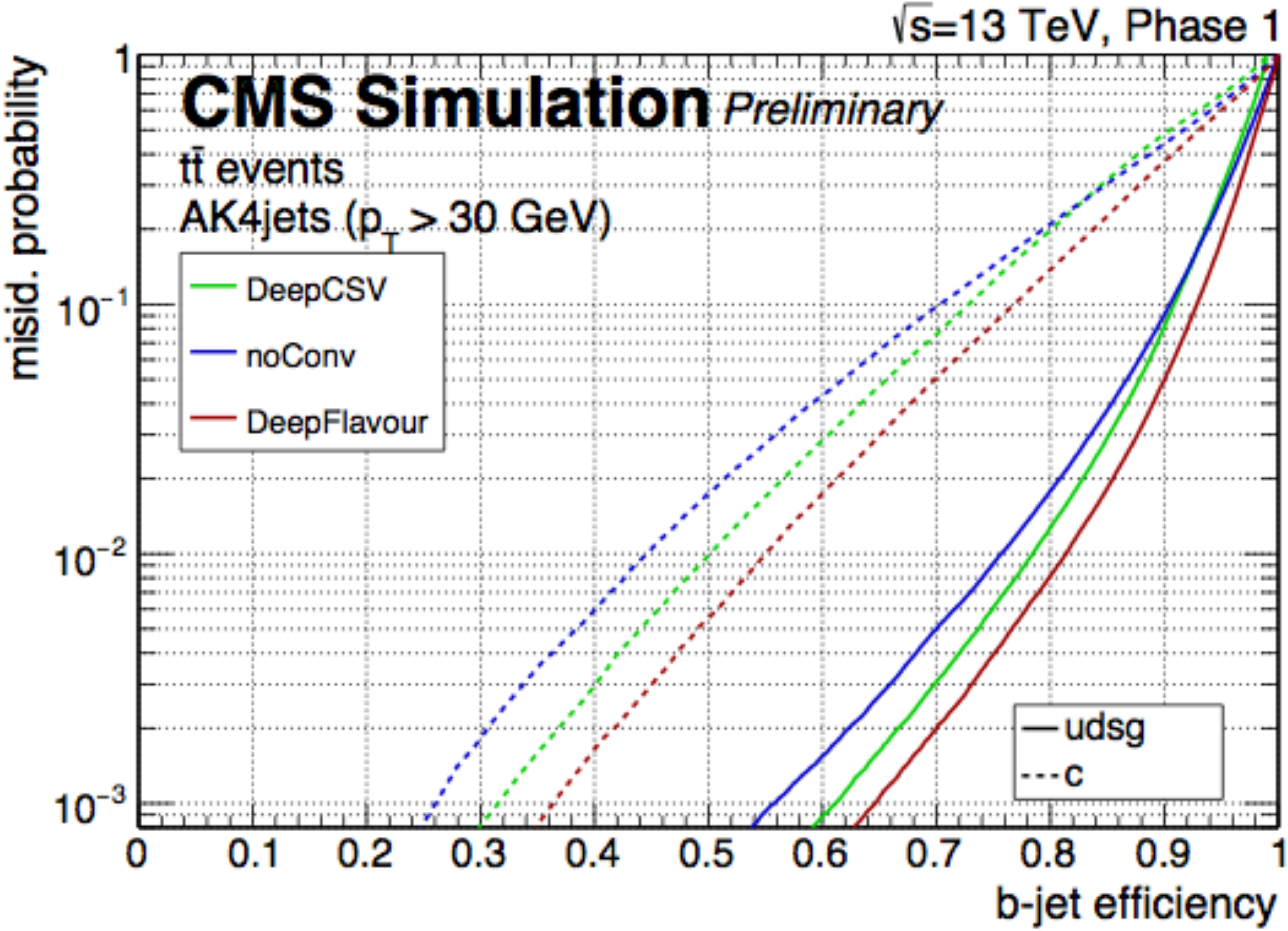
WO)

Summary

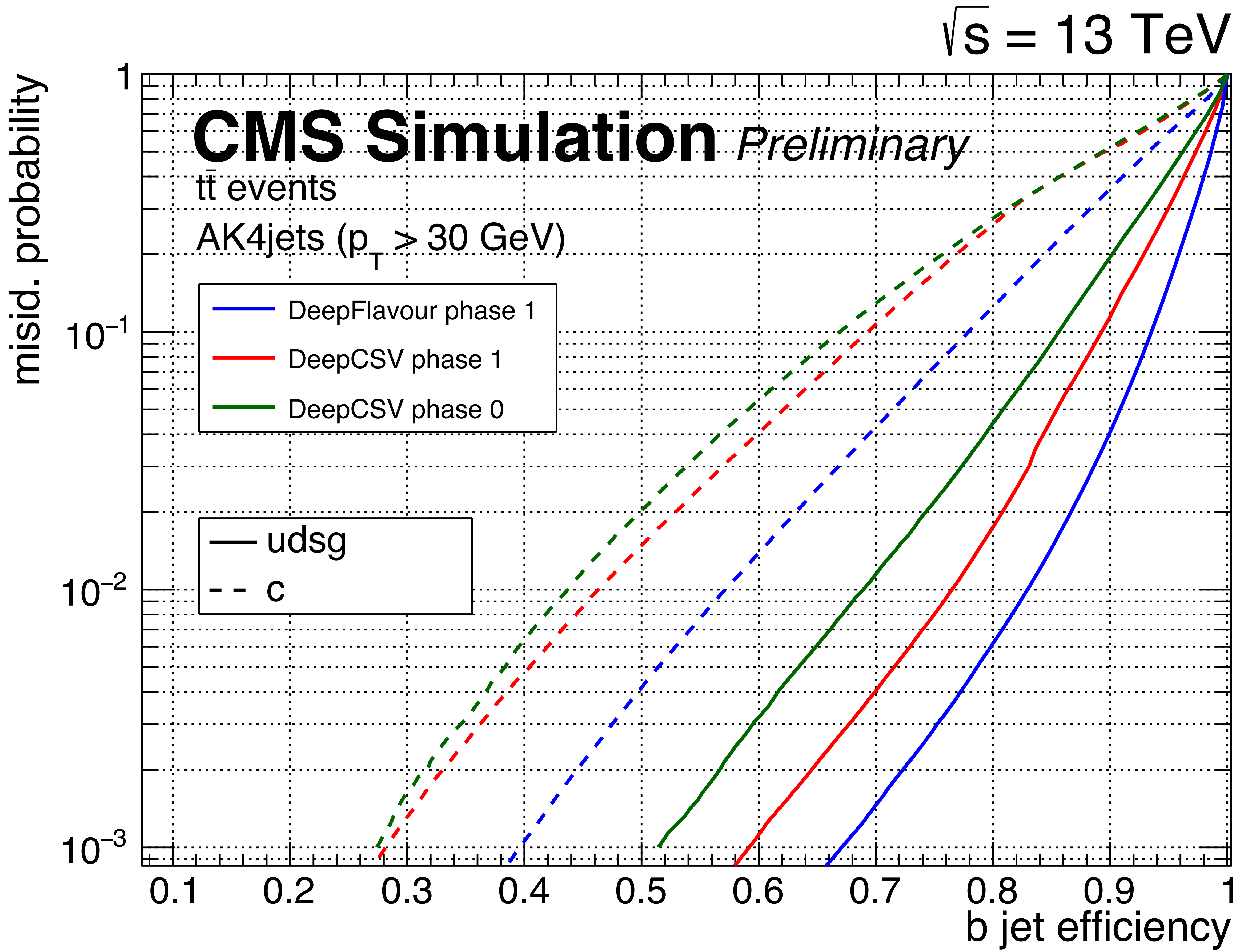
- Jet tagging is of paramount importance for the CMS Physics program
- Lots of development in the last ~1.5 years to apply modern machine learning techniques to this field
  - Large improvements in performance
  - Still some room for new developments, especially in the boosted regime
- Flavour tagging is not only fancy algorithms, but solid and performing computing infrastructures as well
  - Model deployment can take a significant amount of time

Backup

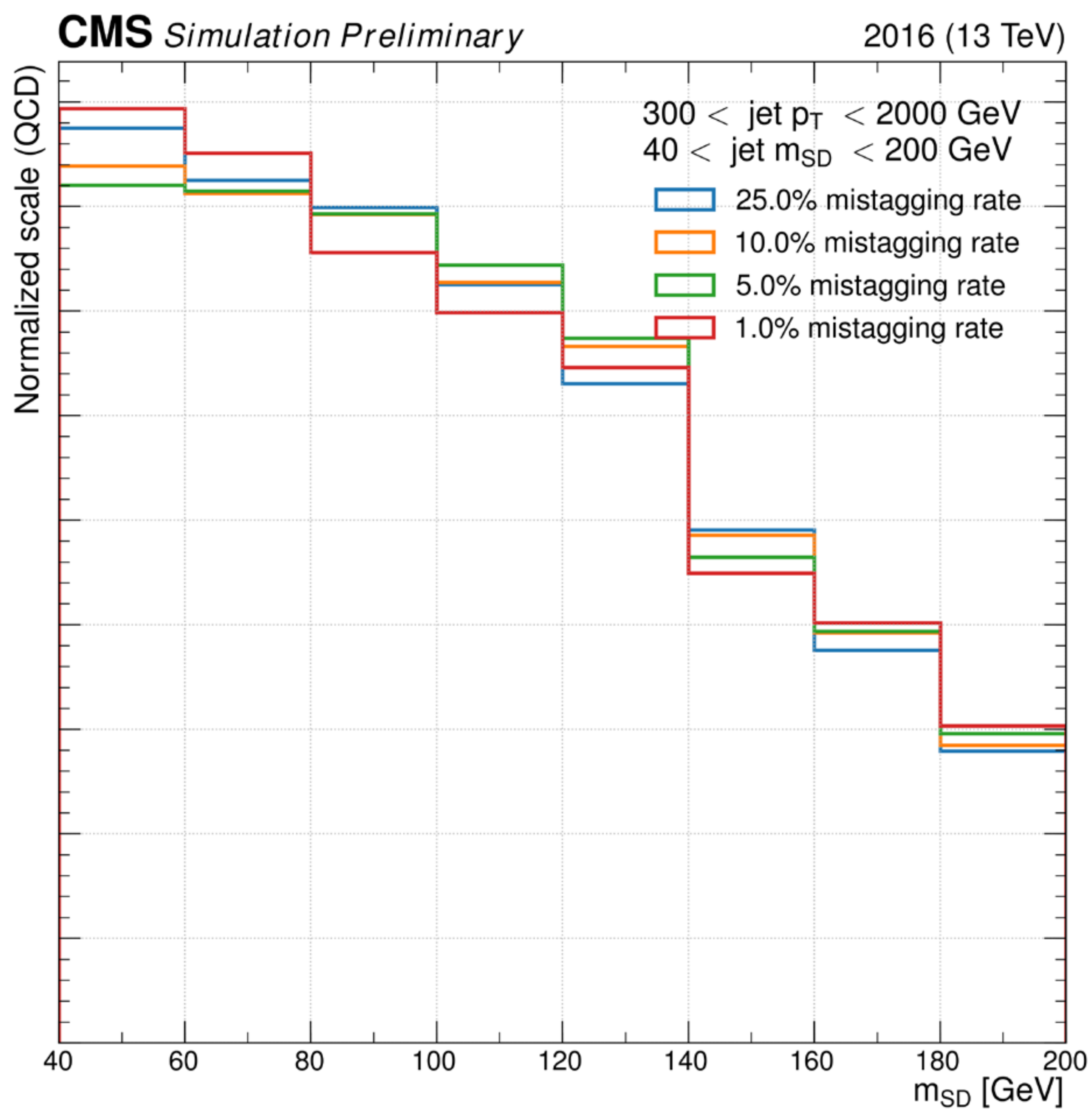
Optimistic MC simulation



Realistic MC simulation

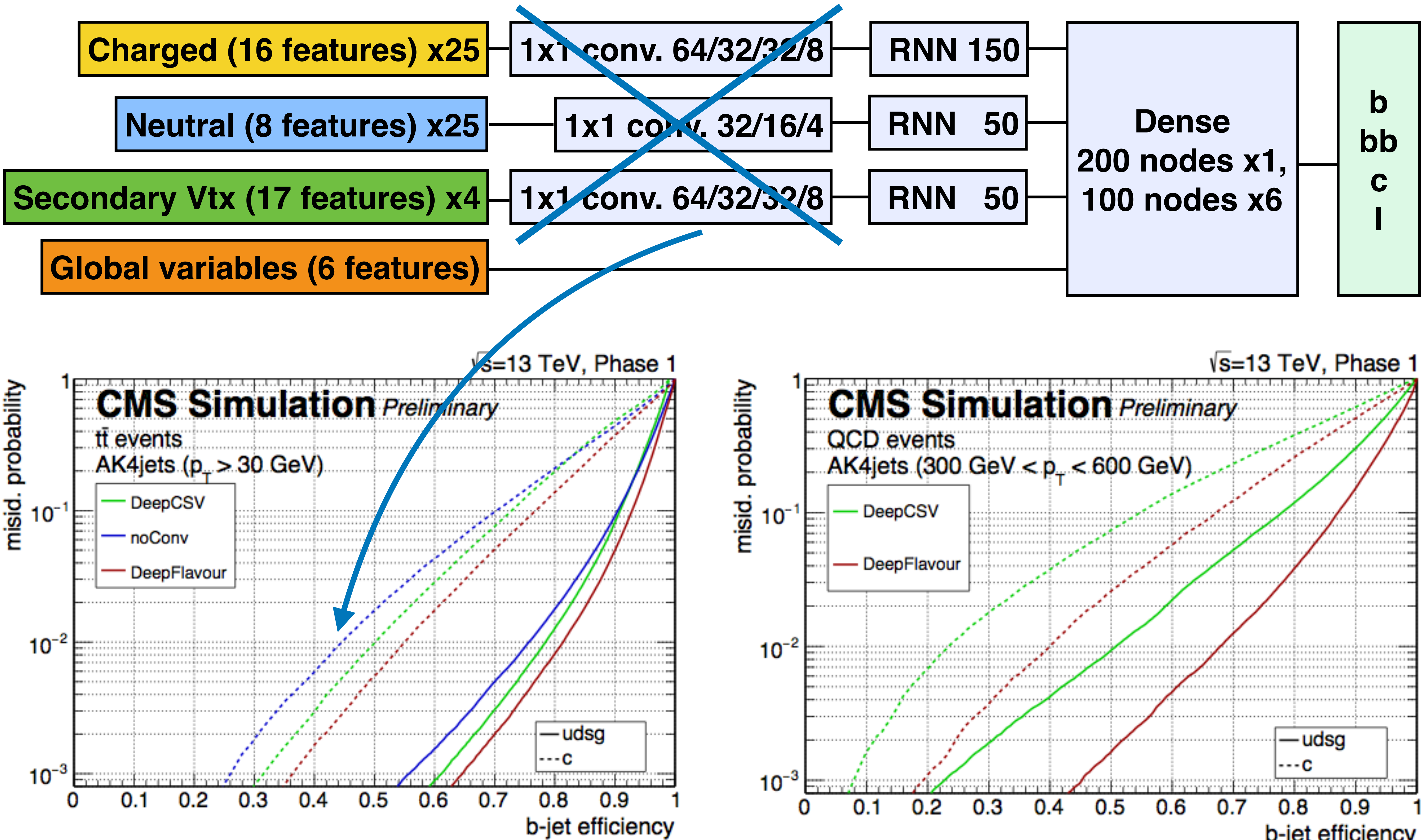


DeepDoubleC - mass sculpting



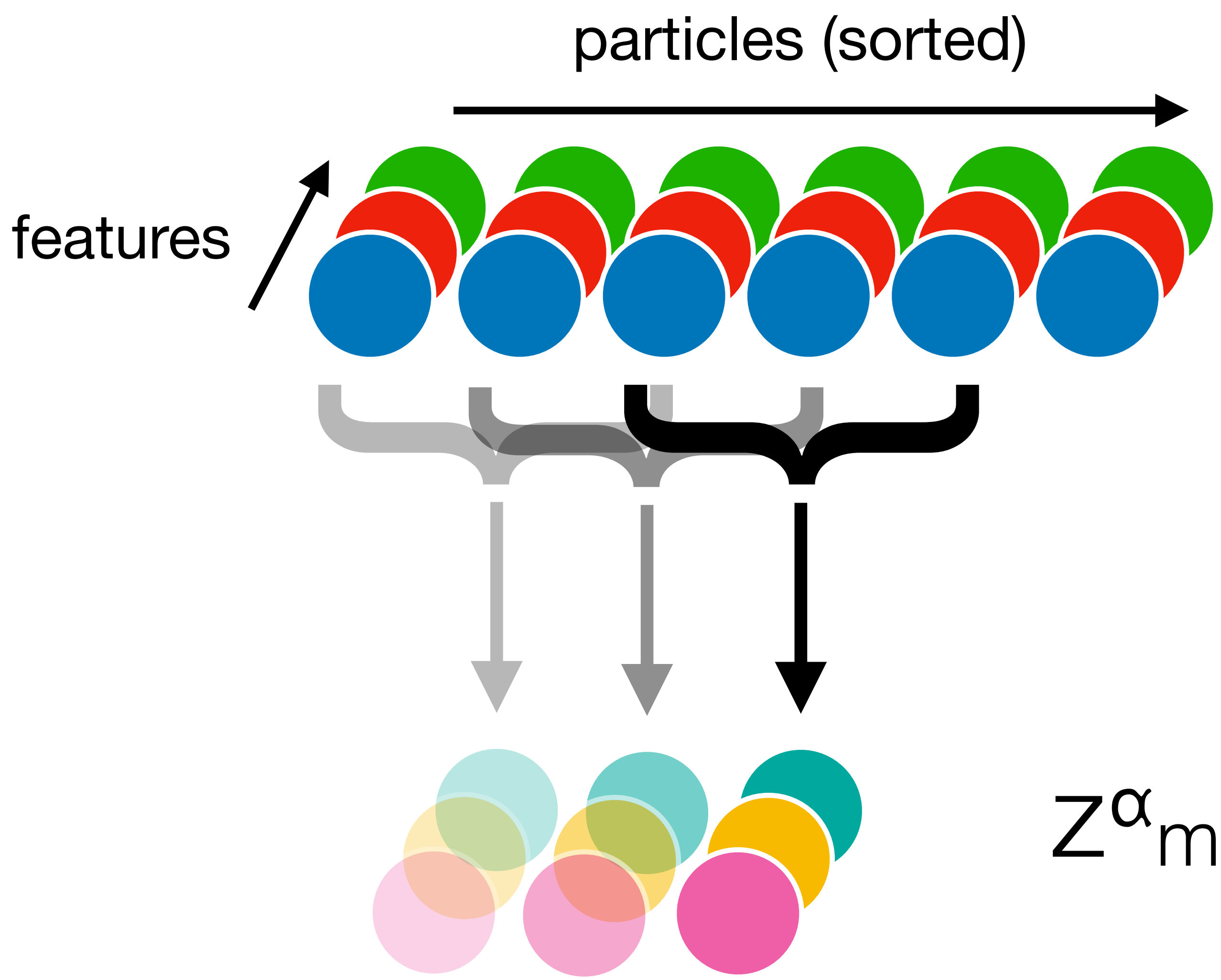
CMS-DP-2018-XXX

Particle-based NN architecture



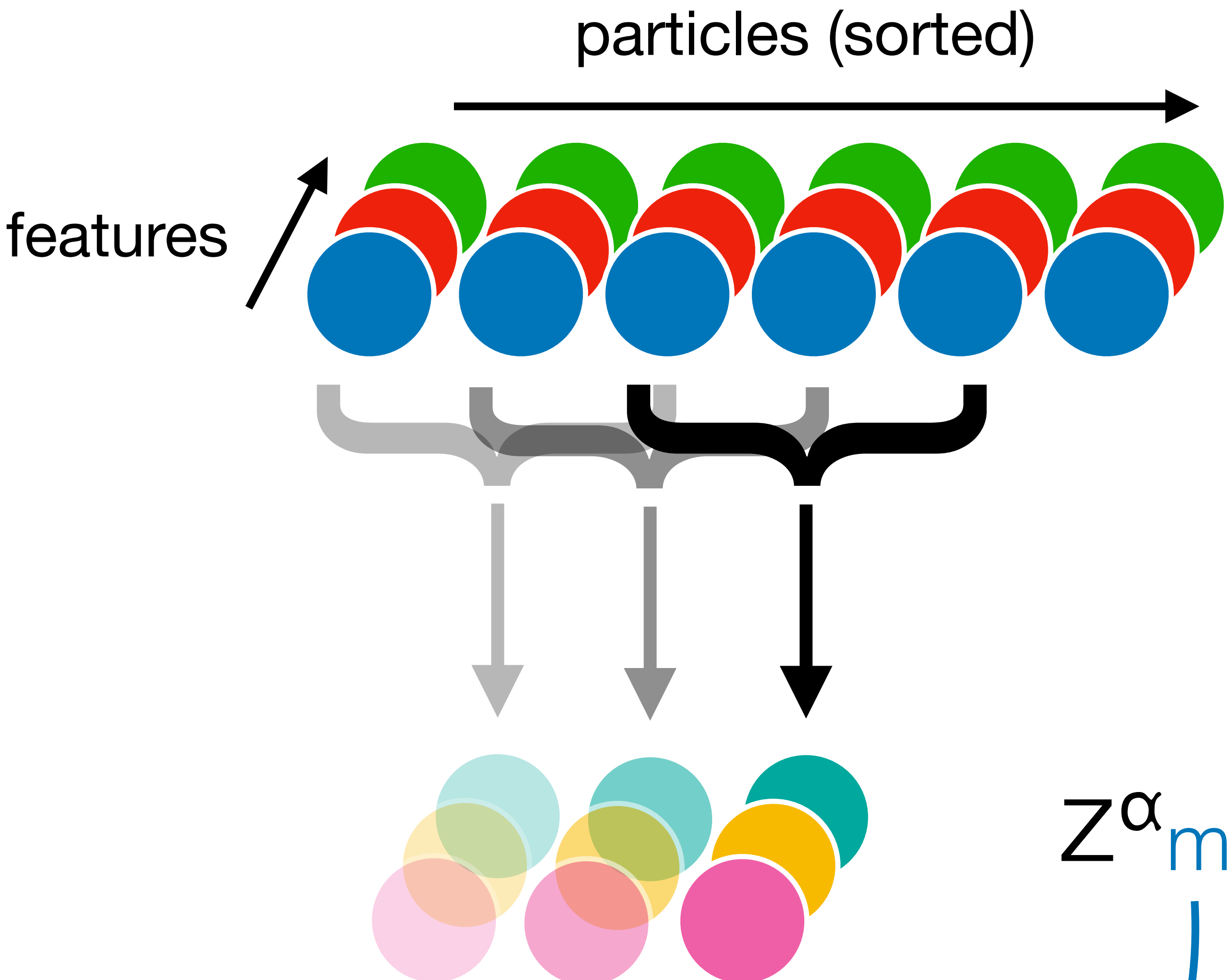
CMS-DP-2017-013

P-CNNs



$$z^{\alpha}_{m} = \Sigma_a \Sigma_j \, k^{\alpha}_{a,j} \, x_{a,(m+j-1)}$$

P-CNNs

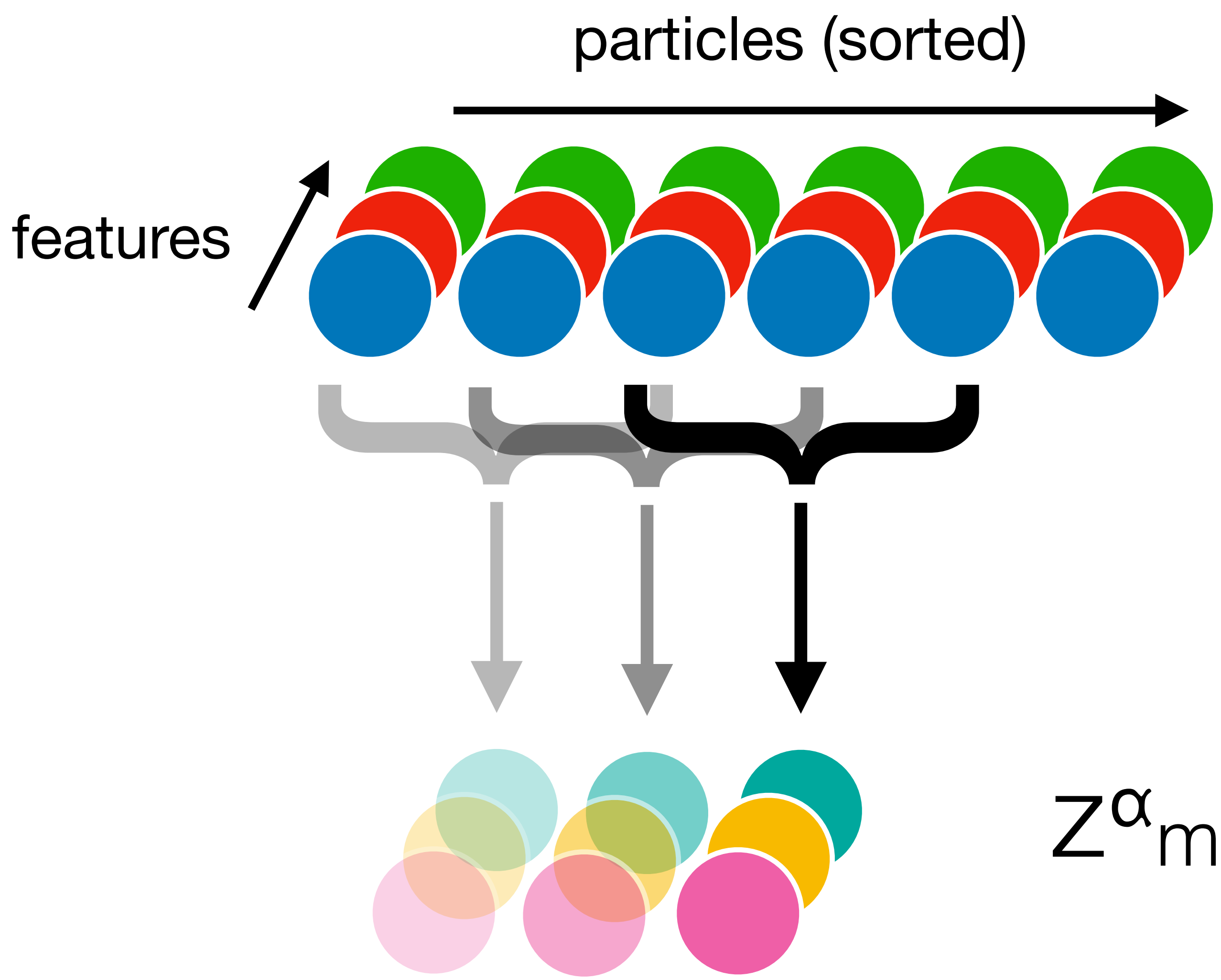


$$z^{\alpha}_{m} = \Sigma_a \Sigma_j \, k^{\alpha}_{a,j} \, x_{a,(m+j-1)}$$

Sweep over the elements

Loop over contiguous elements of the kernel

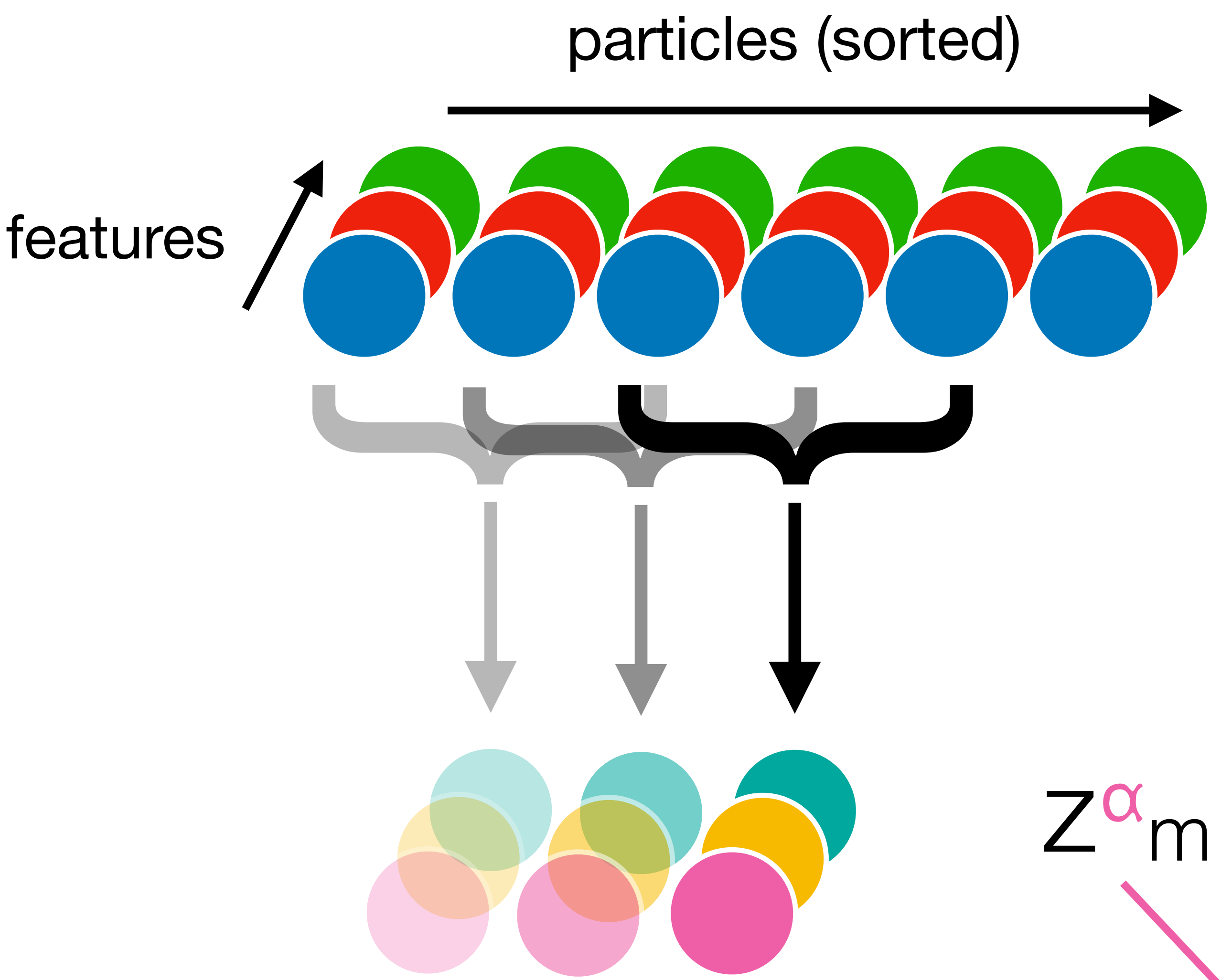
P-CNNs



$$z^{\alpha}_{m} = \sum_a \sum_j k^{\alpha}_{a,j}\ x_{a,(m+j-1)}$$

Multiple features ("colours") are accounted computing the transformation

P-CNNs



$$z^{\alpha}_{m} = \sum_a \sum_j k^{\alpha}_{a,j} \, x_{a,(m+j-1)}$$

Different filters/kernels learn different transformations