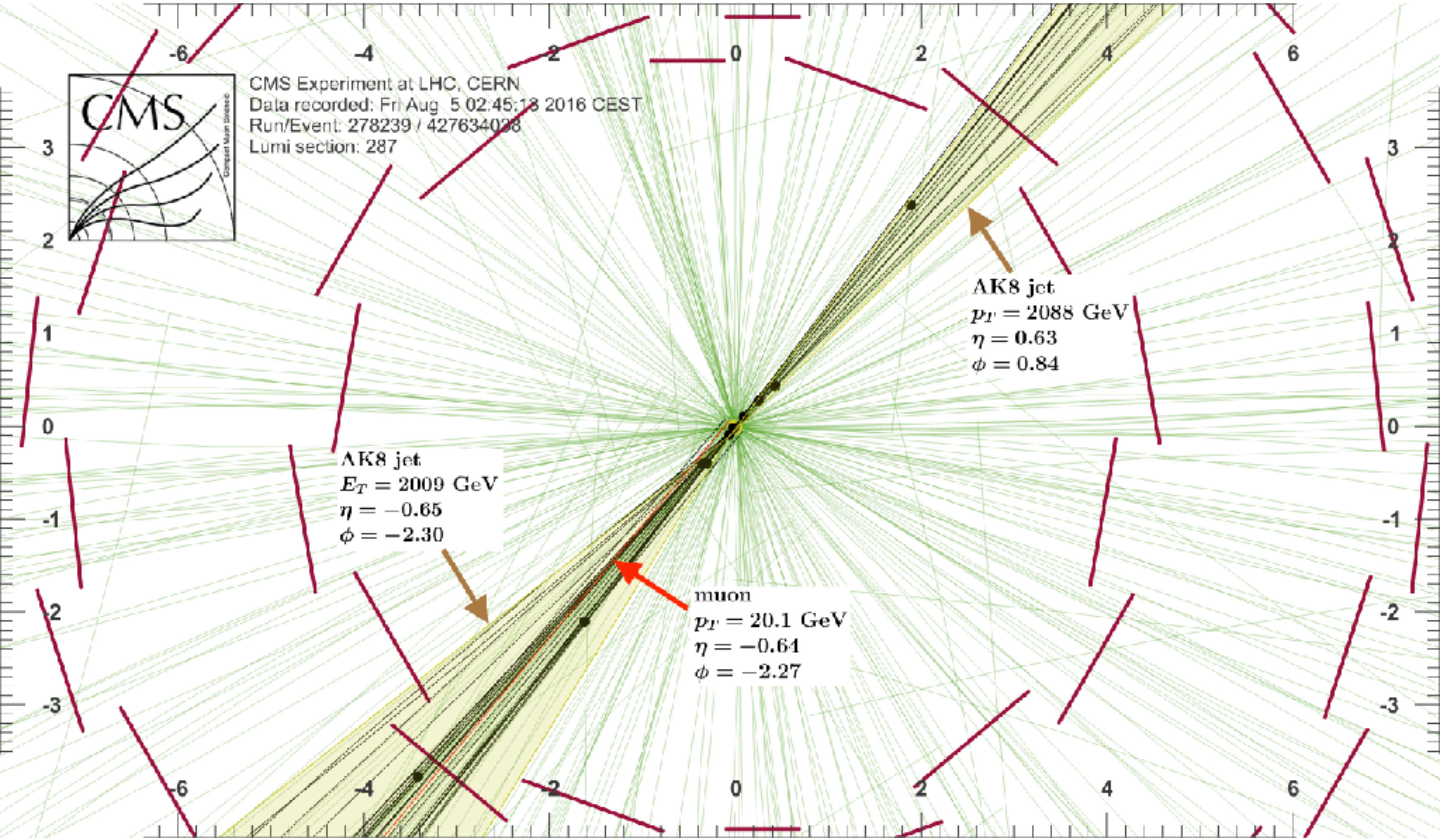
Machine learning applications to jet tagging in CMS

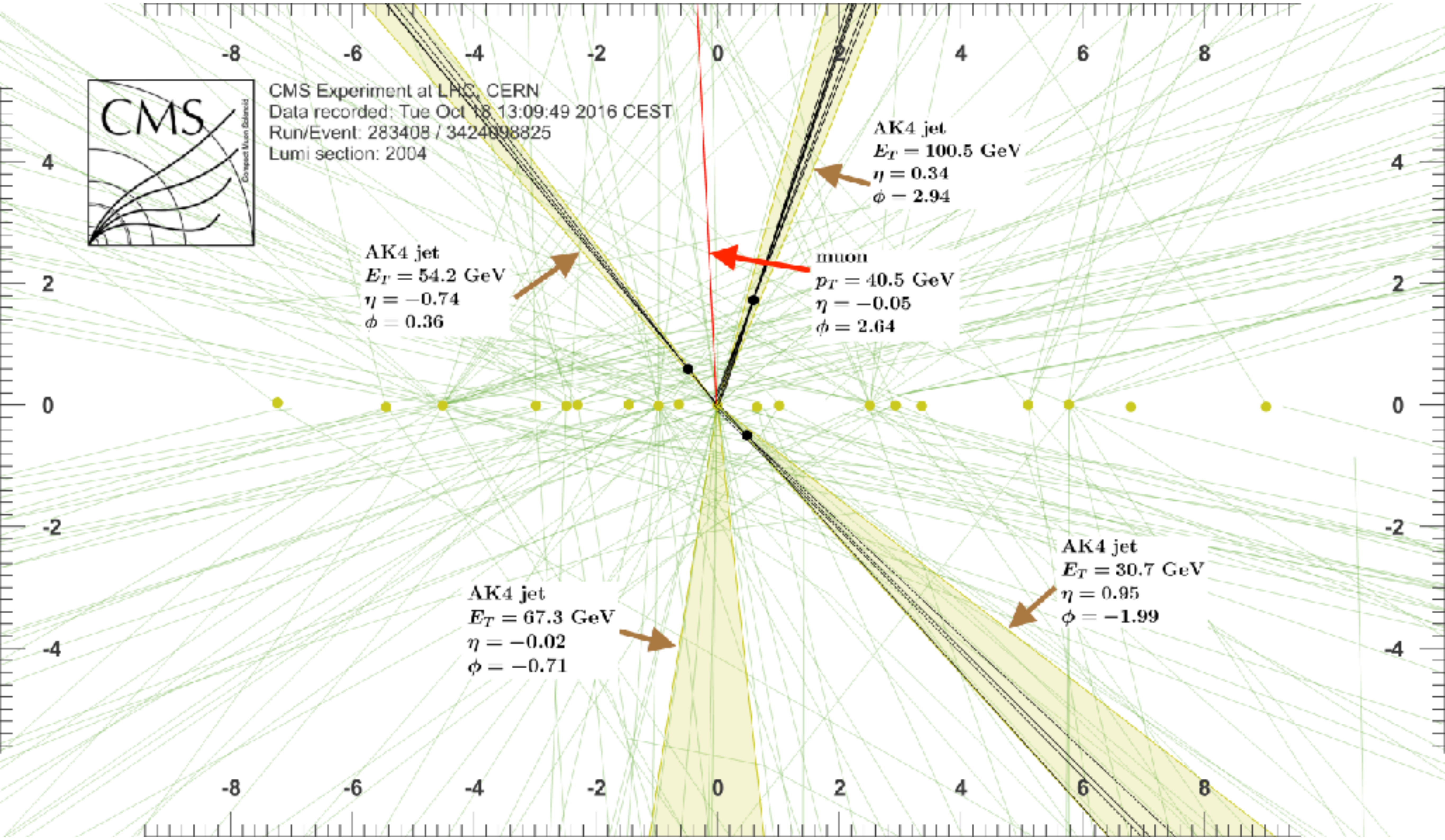
Mauro Verzetti (CERN and FWO)
on behalf of the CMS Collaboration

The problem



CMS Experiment at LHC, CERN
Data recorded: Fri Aug 5 02:45:18 2016 CEST
Run/Event: 278239 / 427634028
Lumi section: 287
AK8 jet
$p_T = 2088$ GeV
$\eta = 0.63$
$\phi = 0.84$
AK8 jet
$E_T = 2009$ GeV
$\eta = -0.65$
$\phi = -2.30$
muon
$p_T = 20.1$ GeV
$\eta = -0.64$
$\phi = -2.27$

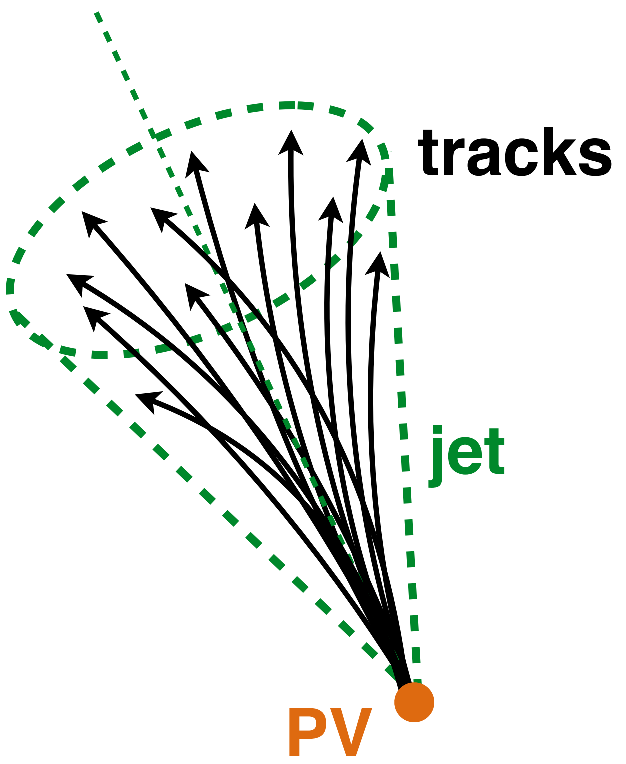
The problem



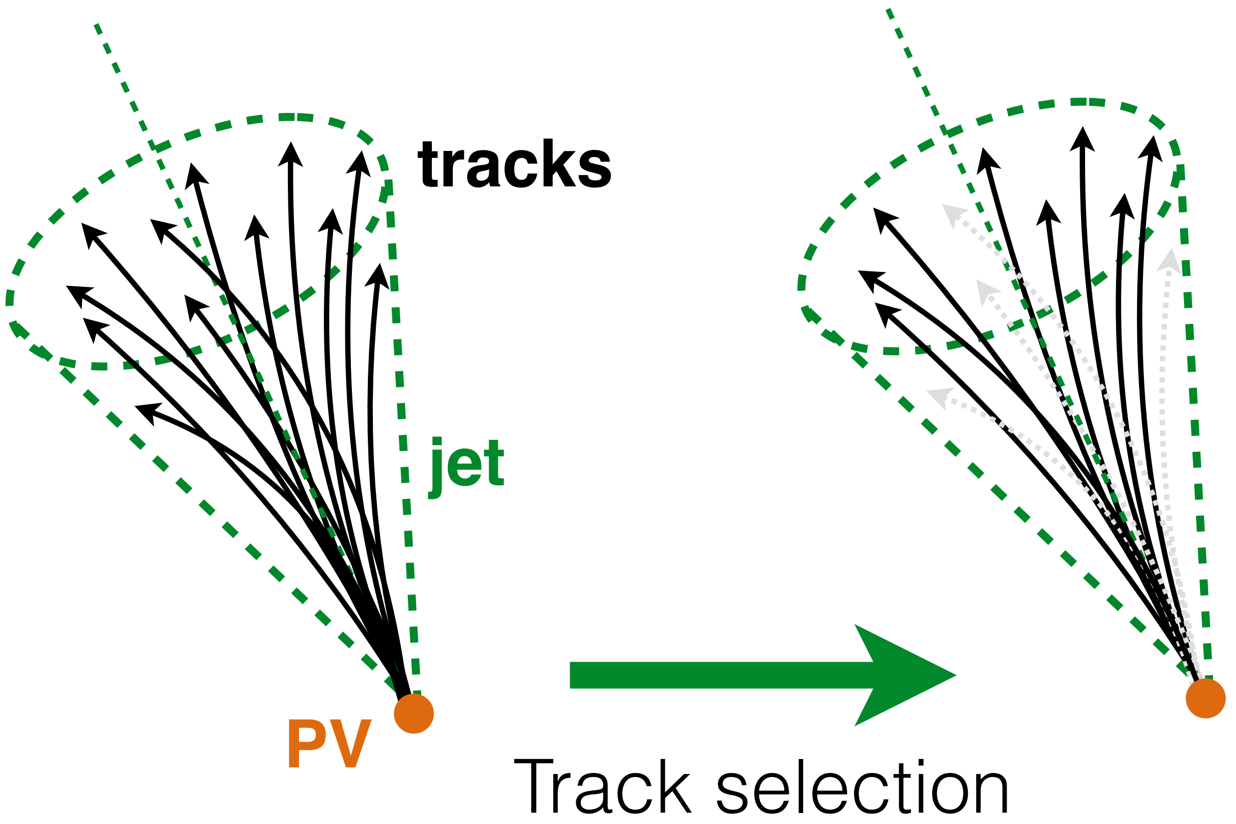
CMS Experiment at LHC, CERN
Data recorded: Tue Oct 18 13:09:49 2016 CEST
Run/Event: 283408 / 3424098825
Lumi section: 2004
AK4 jet
$E_T = 100.5$ GeV
$\eta = 0.34$
$\phi = 2.94$
AK4 jet
$E_T = 54.2$ GeV
$\eta = -0.74$
$\phi = 0.36$
muon
$p_T = 40.5$ GeV
$\eta = -0.05$
$\phi = 2.64$
AK4 jet
$E_T = 30.7$ GeV
$\eta = 0.95$
$\phi = -1.99$
AK4 jet
$E_T = 67.3$ GeV
$\eta = -0.02$
$\phi = -0.71$

“resolved” AK4

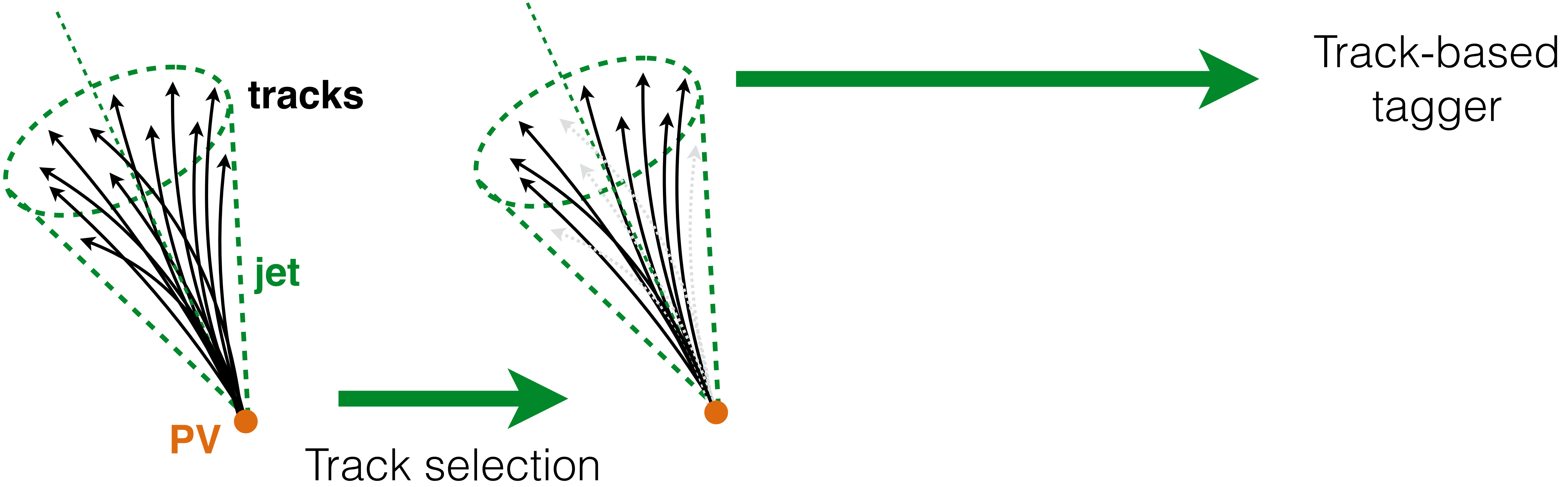
HF tagging @ CMS — arXiv:1712.07158



HF tagging @ CMS — arXiv:1712.07158



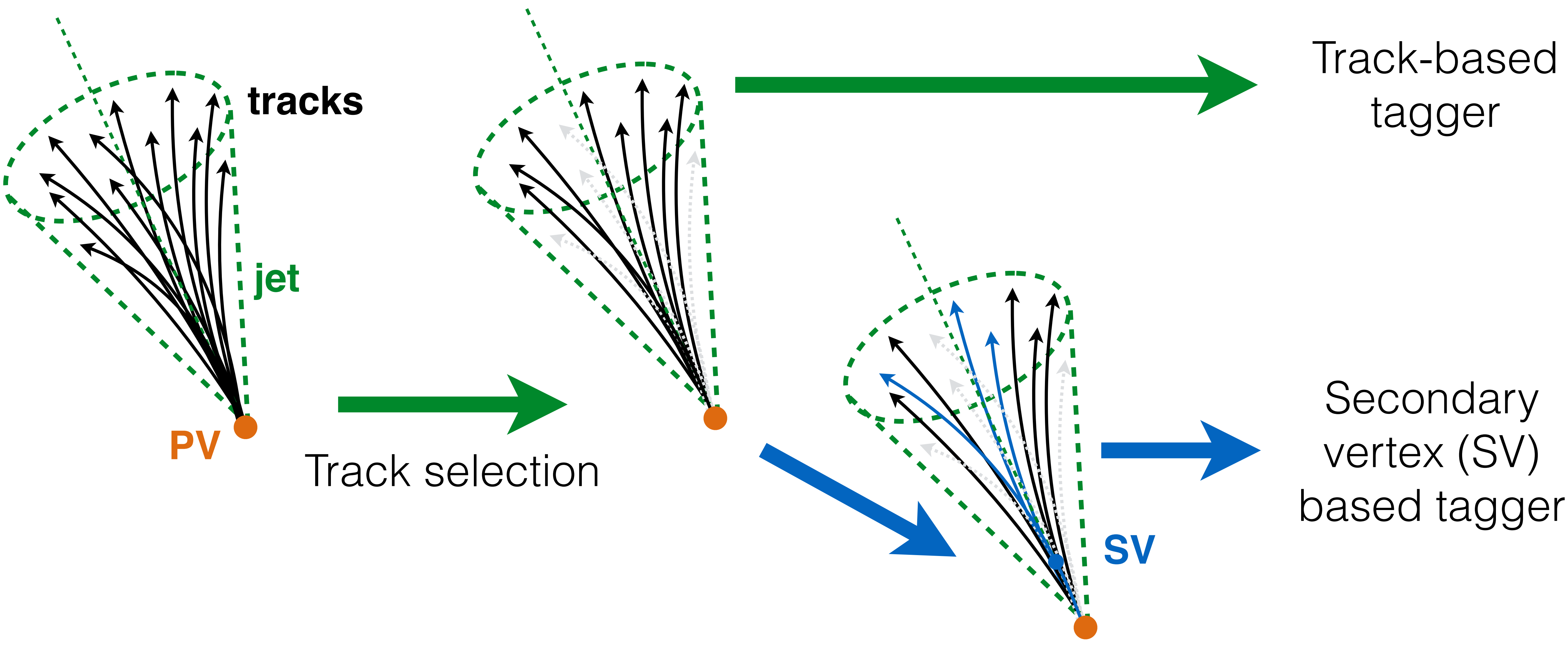
HF tagging @ CMS — arXiv:1712.07158



e.g.: **J**et **P**robability (**JP**)

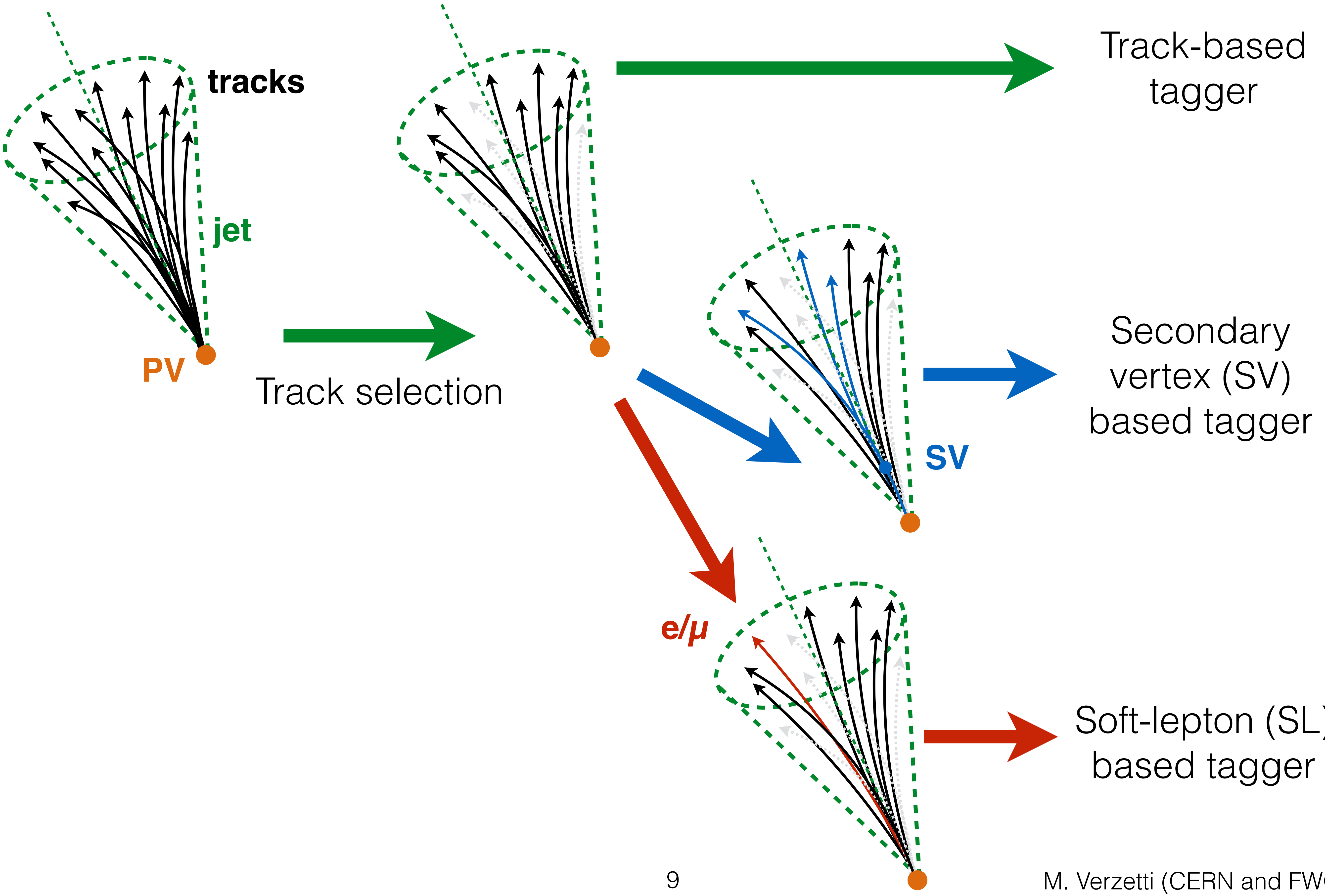
Few marking features → binned likelihood discriminant

HF tagging @ CMS — arXiv:1712.07158

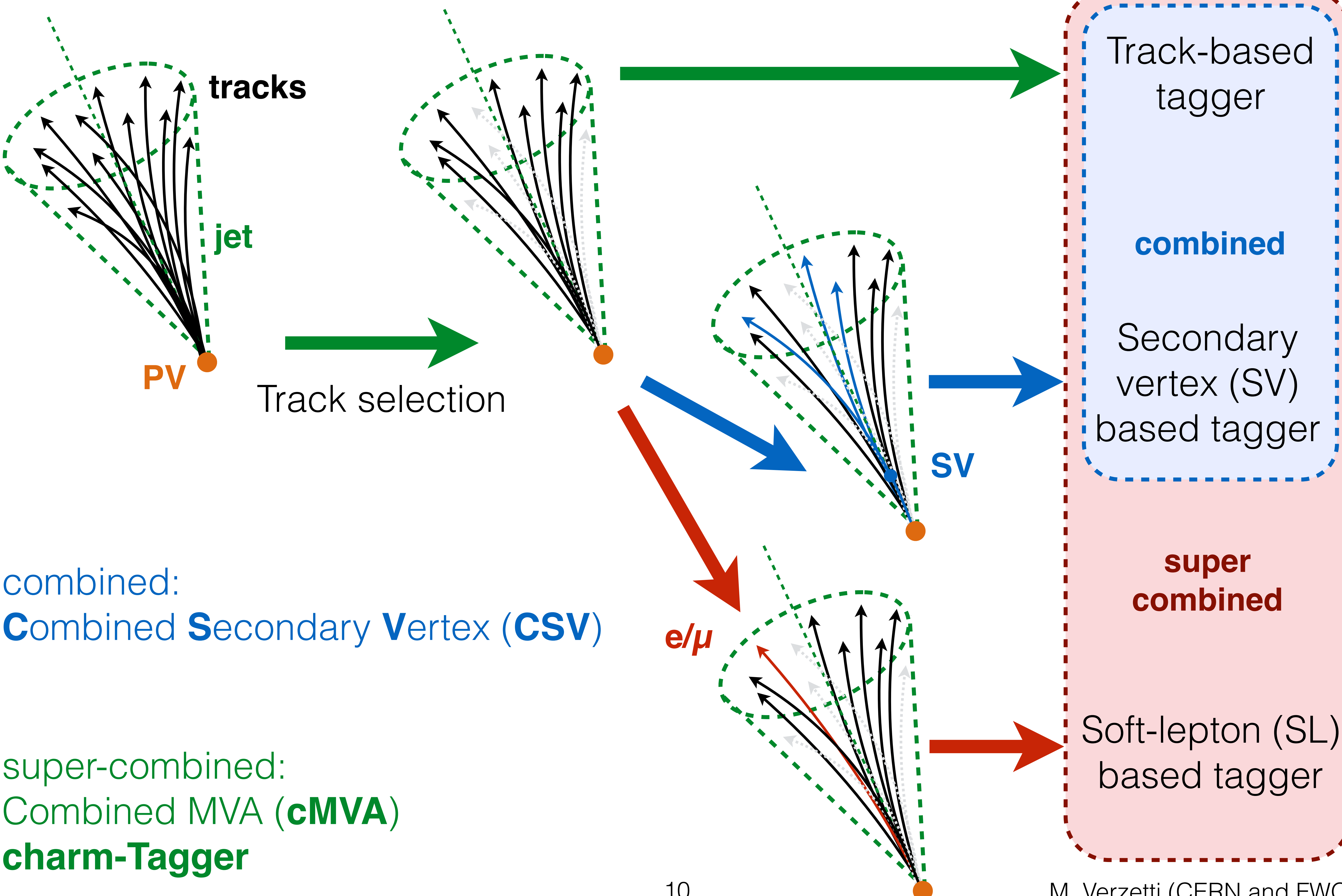


- **A**daptive **V**ertex **R**econstruction **(AVR)**: applied on tracks associated to the jet
- **I**nclusive **V**ertex **F**itter **(IVF)**: on the full set of tracks recorded in the event (SV ΔR-matched to jet).
  Current reconstruction default

HF tagging @ CMS — arXiv:1712.07158



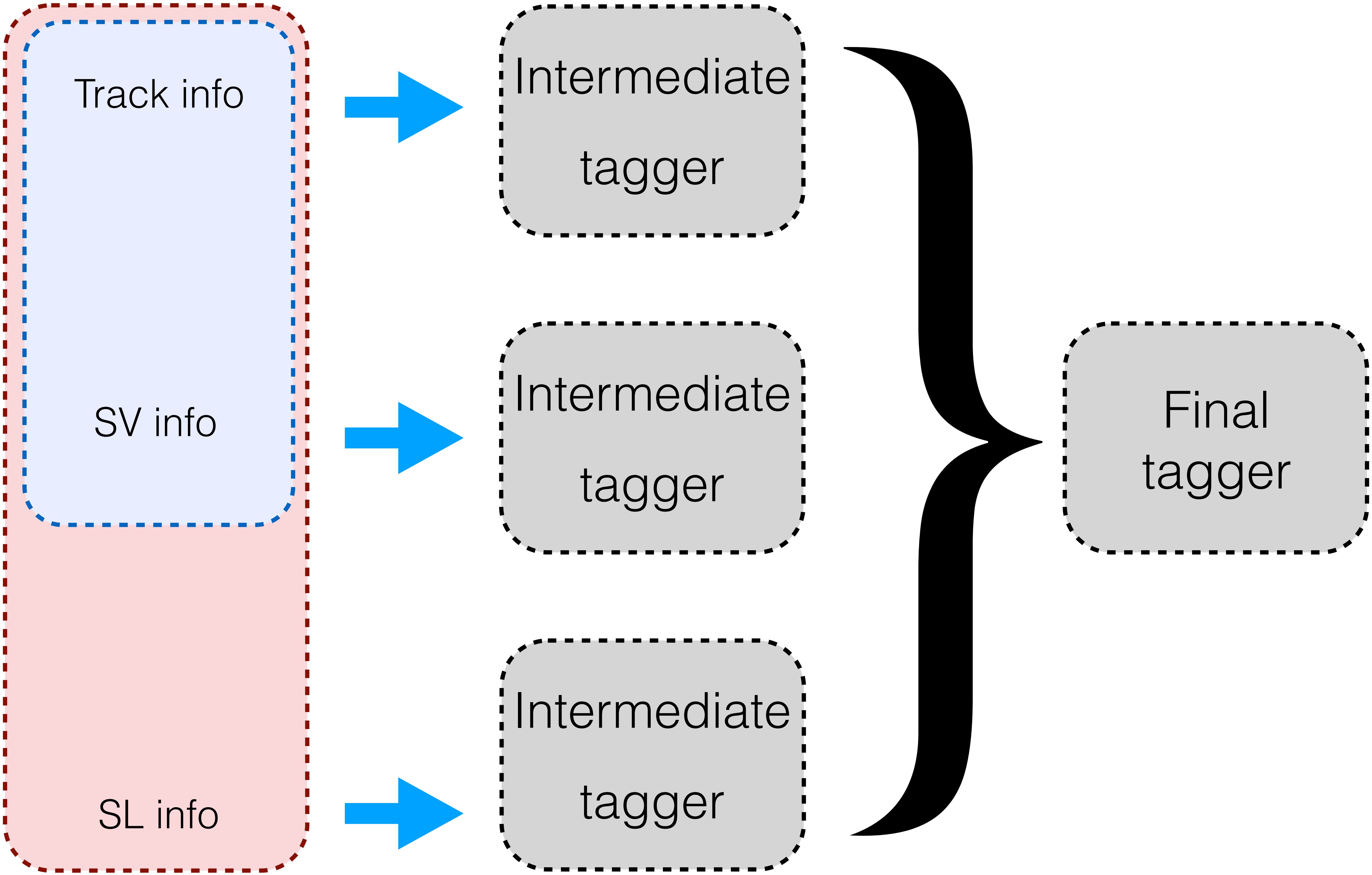
HF tagging @ CMS — arXiv:1712.07158



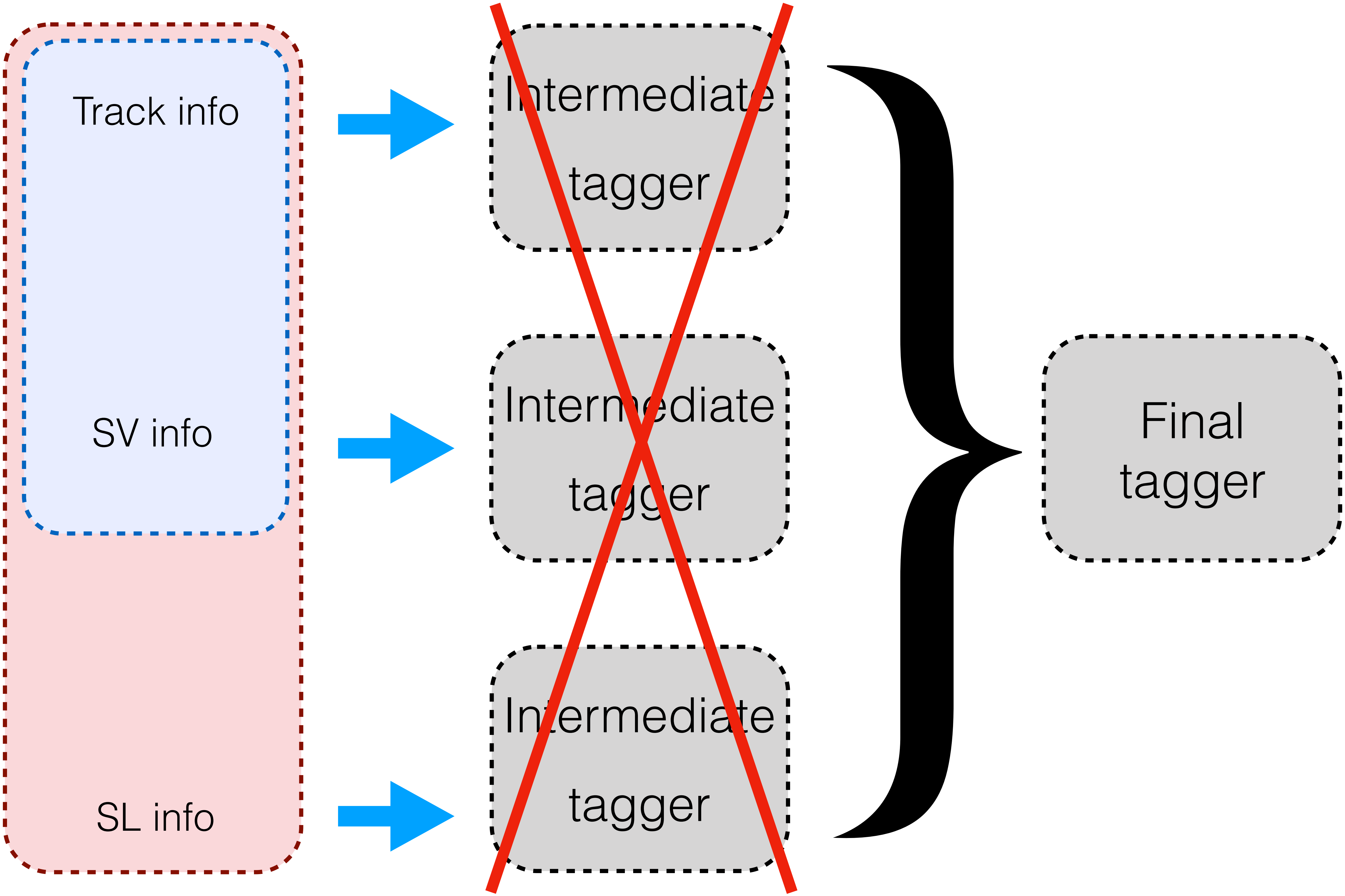
combined:
Combined **S**econdary **V**ertex (**CSV**)

super-combined:
Combined MVA (**cMVA**)
charm-Tagger

From features to taggers: different approaches



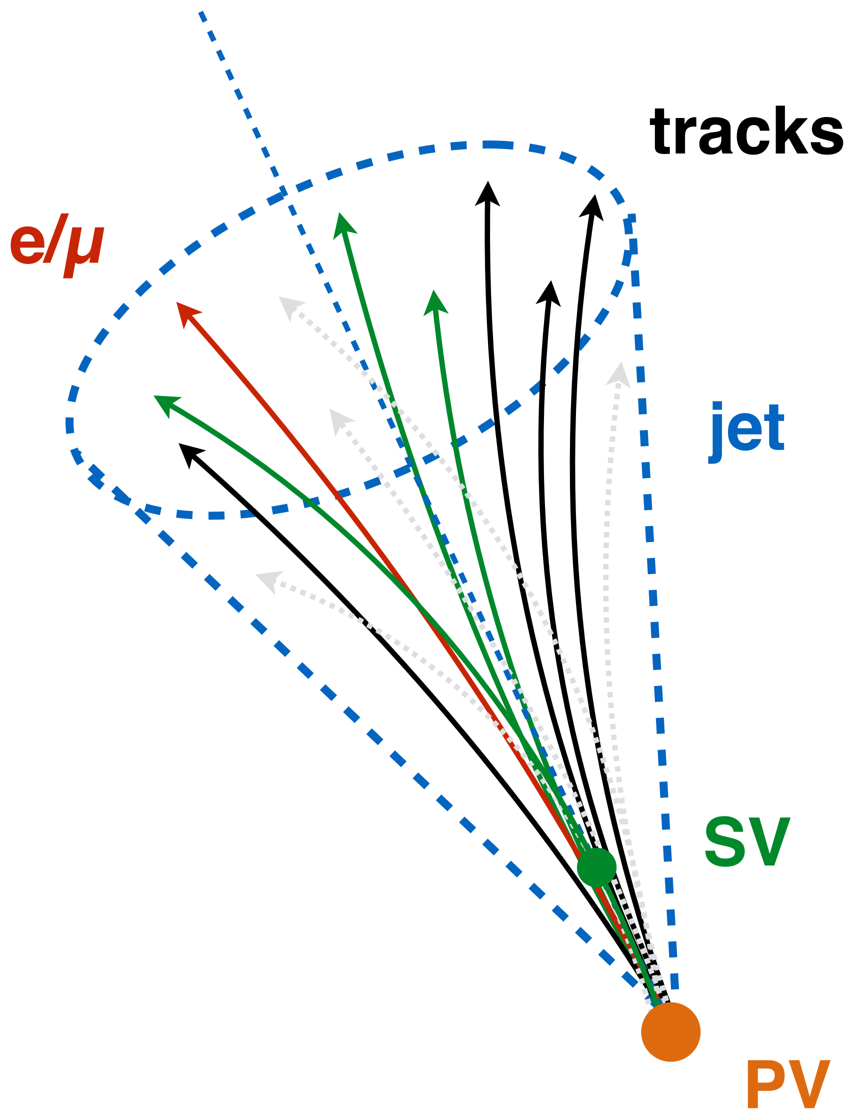
From features to taggers: different approaches

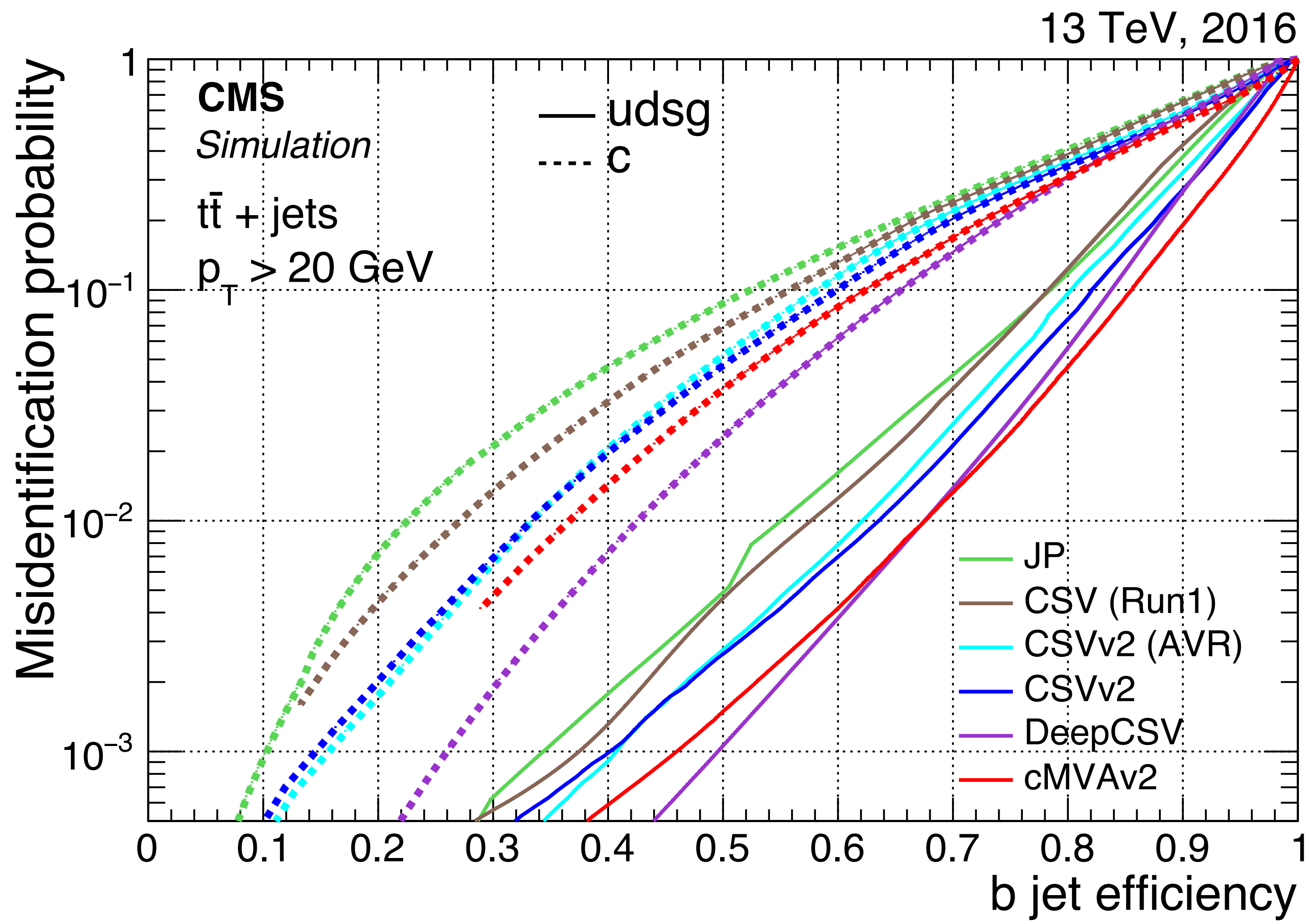


The four main CMS taggers

- **CSVv2:** three shallow NNs, depending on the vertex information, combined with a likelihood method
- **cMVAv2:** meta-tagger that combines the outputs of CSVv2 and other simpler taggers in a single BDT discriminator
- **C-tagger:** set of two BDT classifiers to identify charm jets
- **DeepCSV:** deep DNN that retains the simplicity of CSV, with a different ML approach

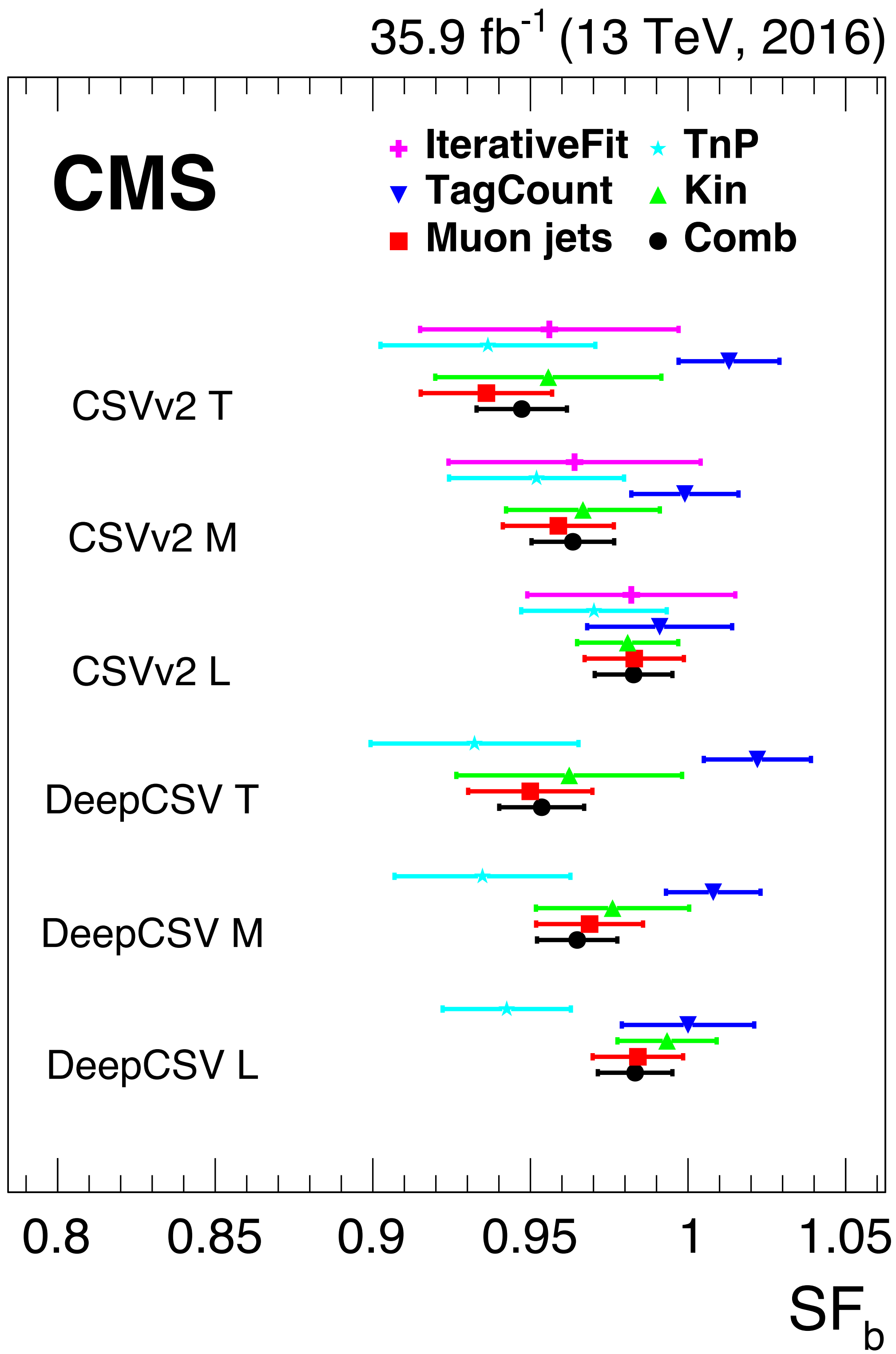
HF tagging @ CMS — arXiv:1712.07158

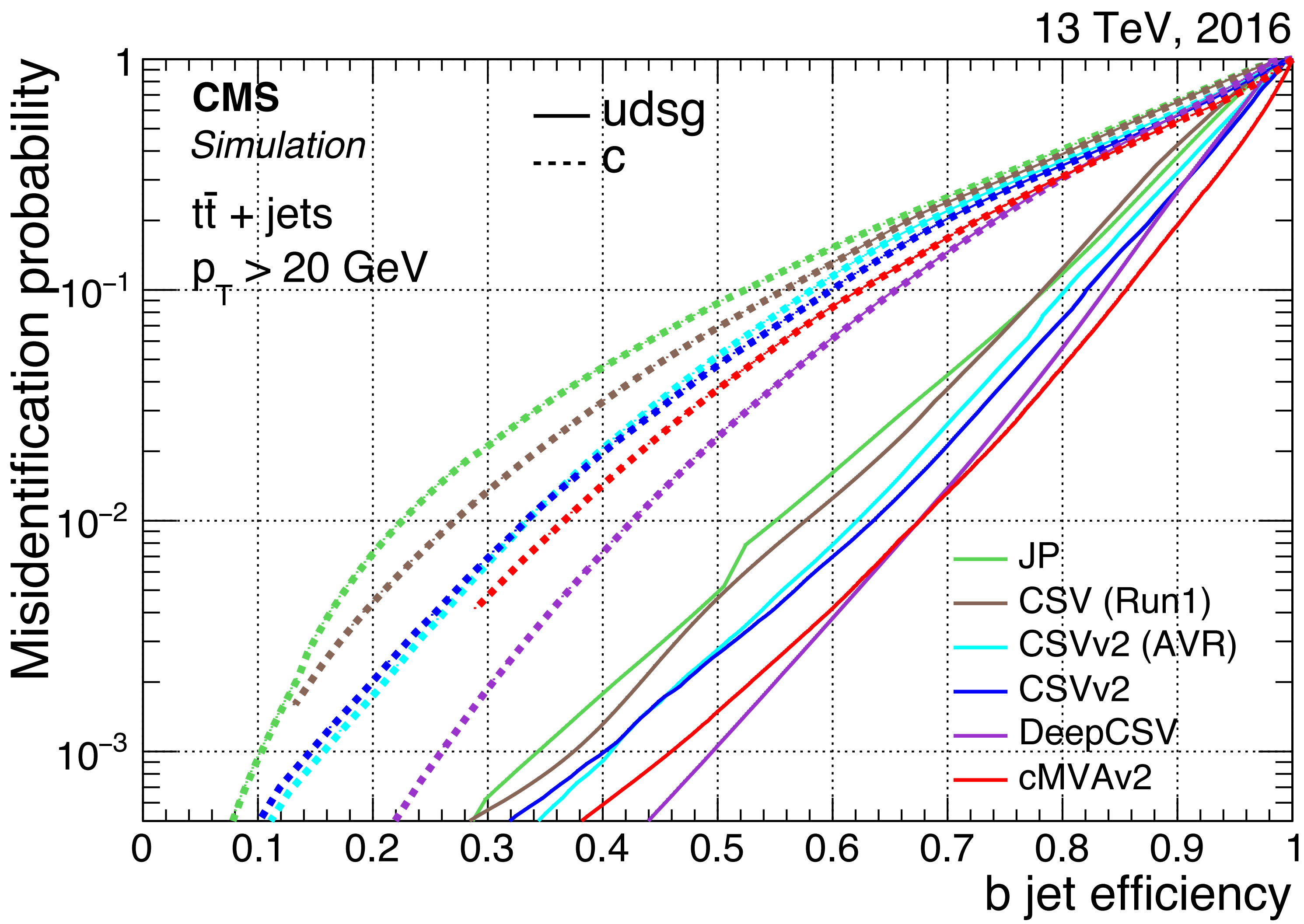




DeepNN shows best performance

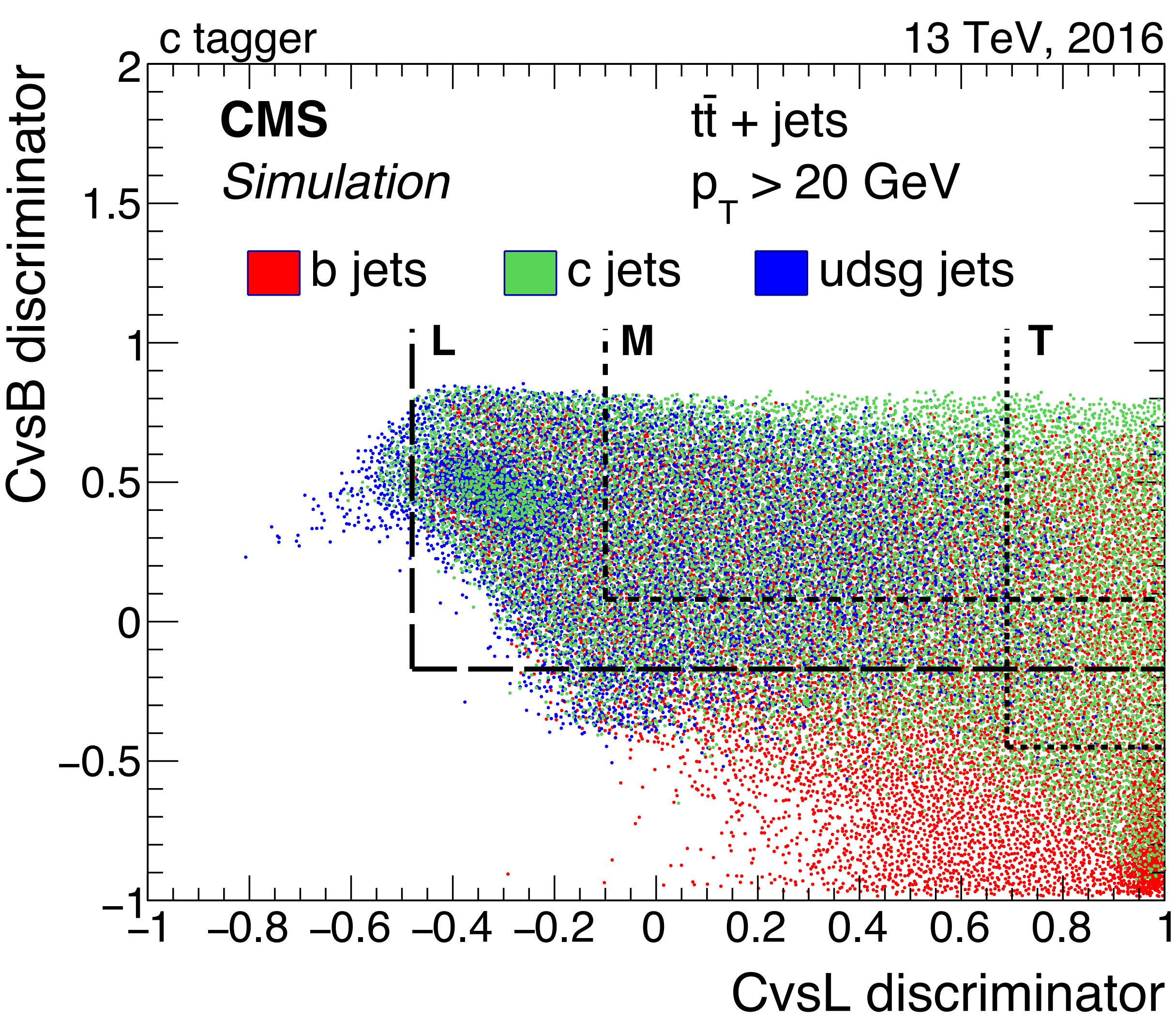
HF tagging @ CMS — arXiv:1712.07158

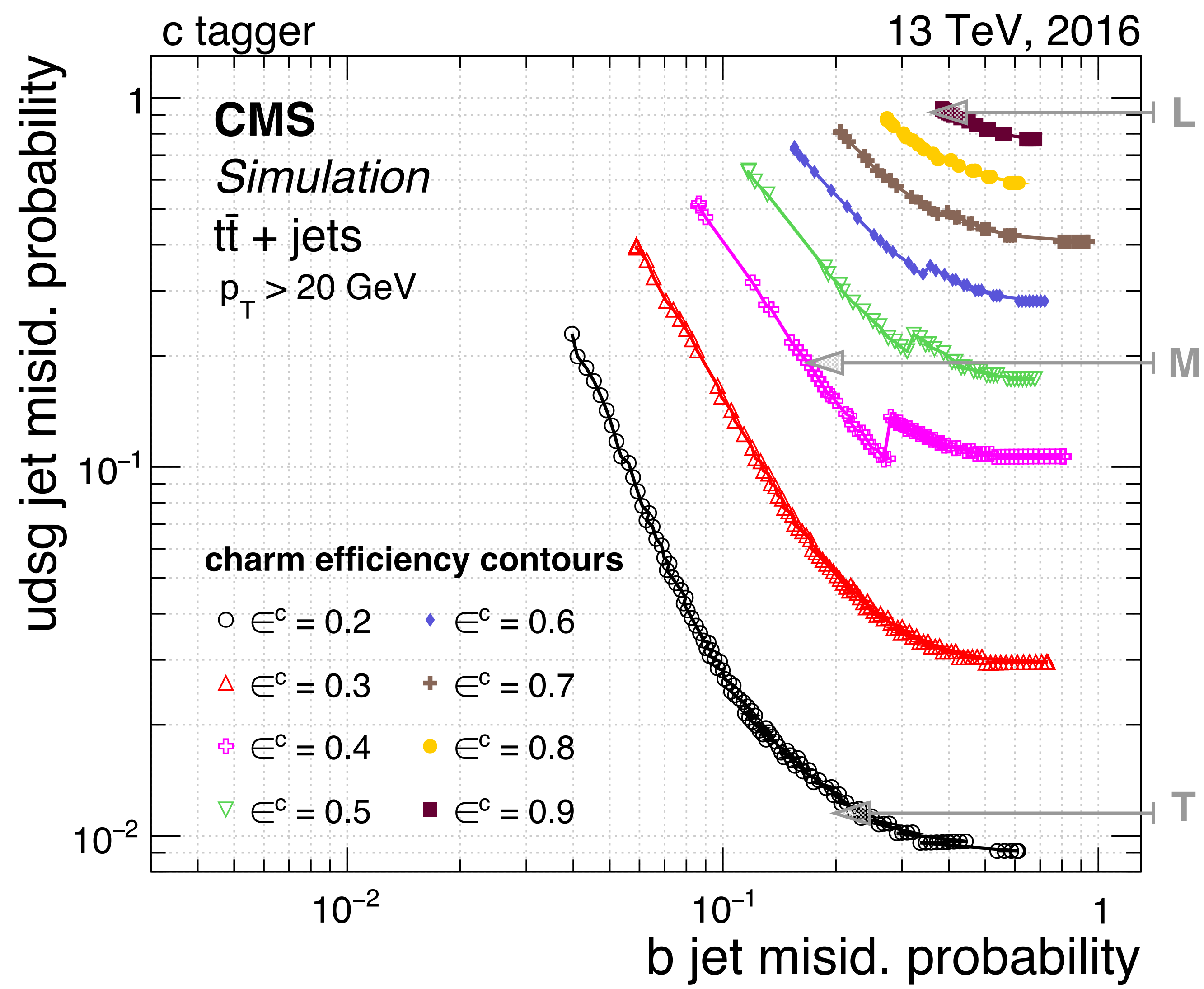




Charm tagging

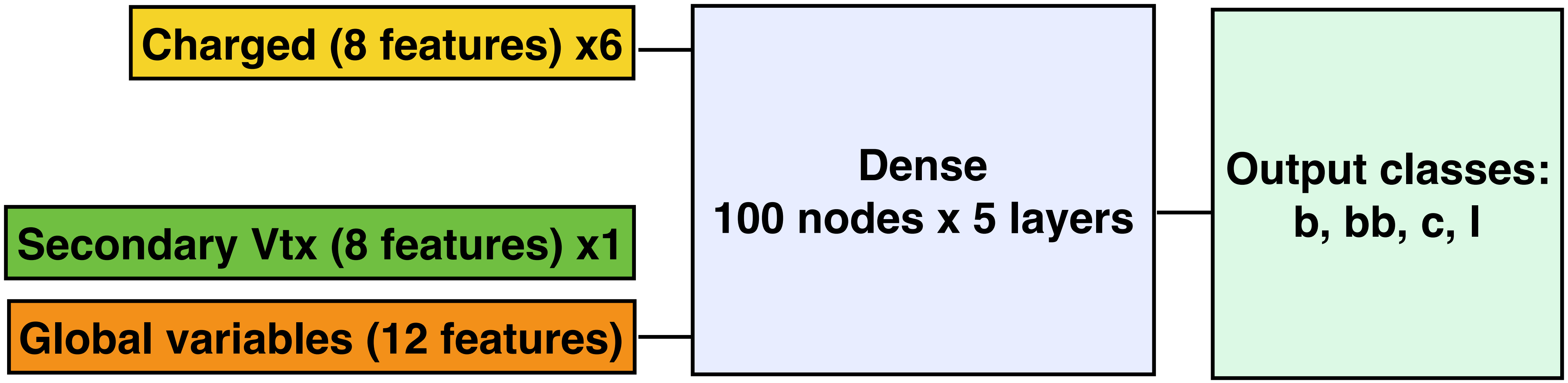
Use two BDT's to discriminate the charm from the light and B components

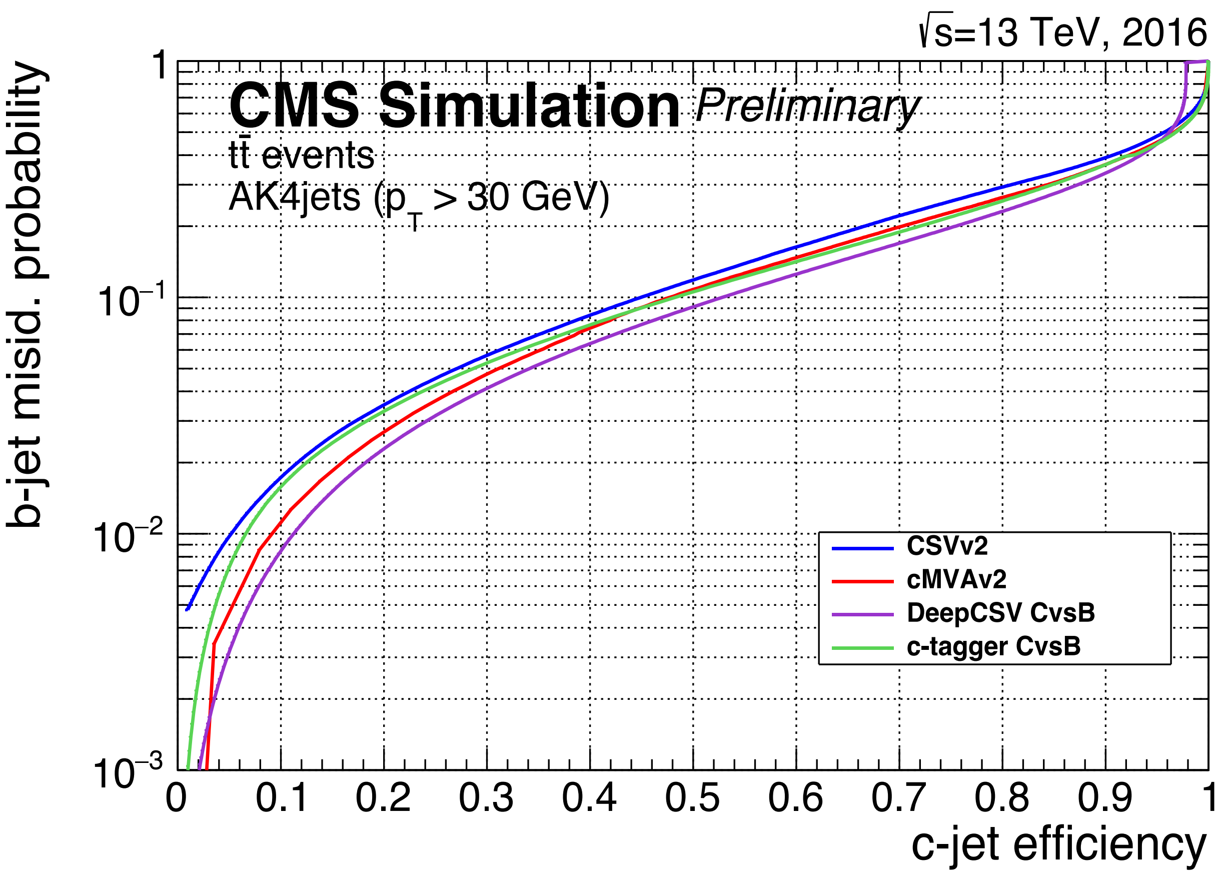


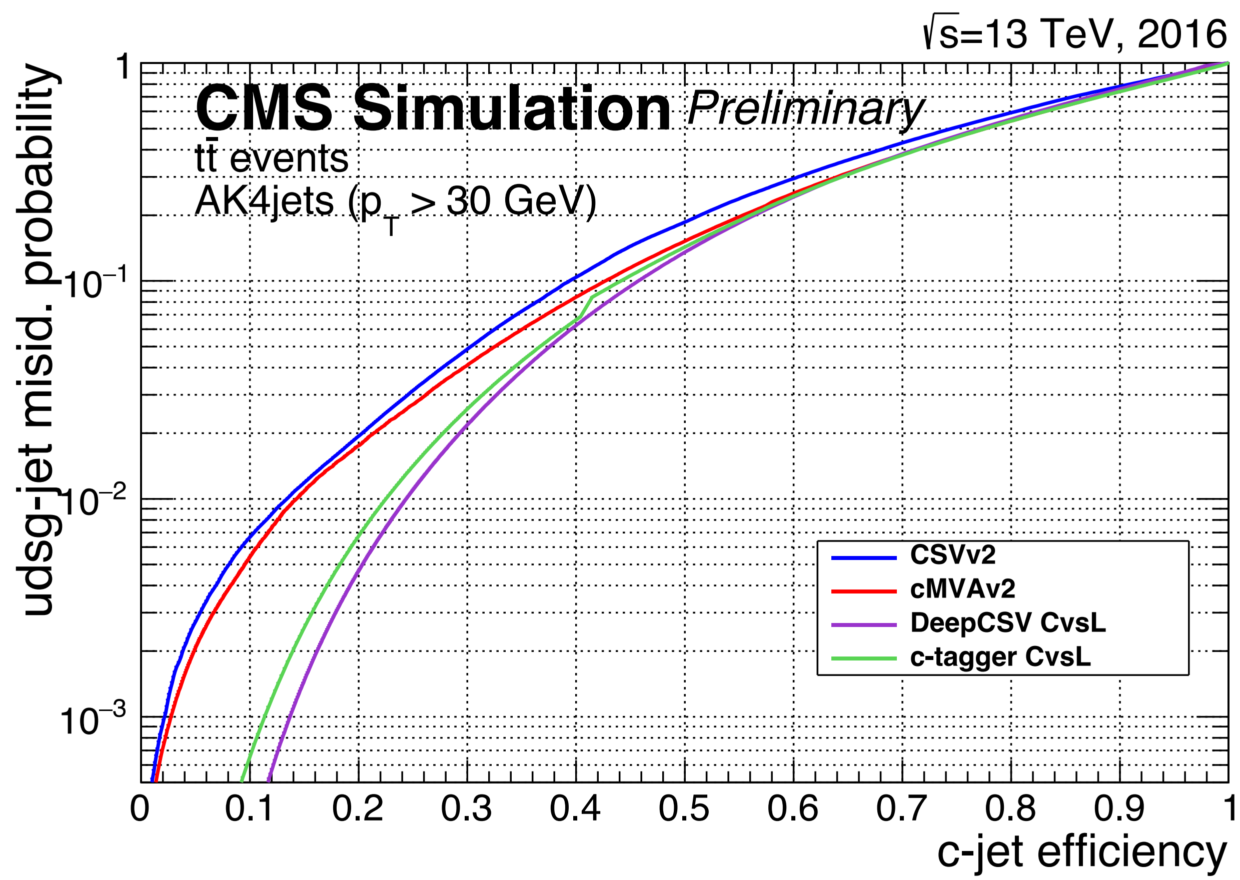


arXiv:1712.07158

Heavy flavor jets with DNN — DeepCSV



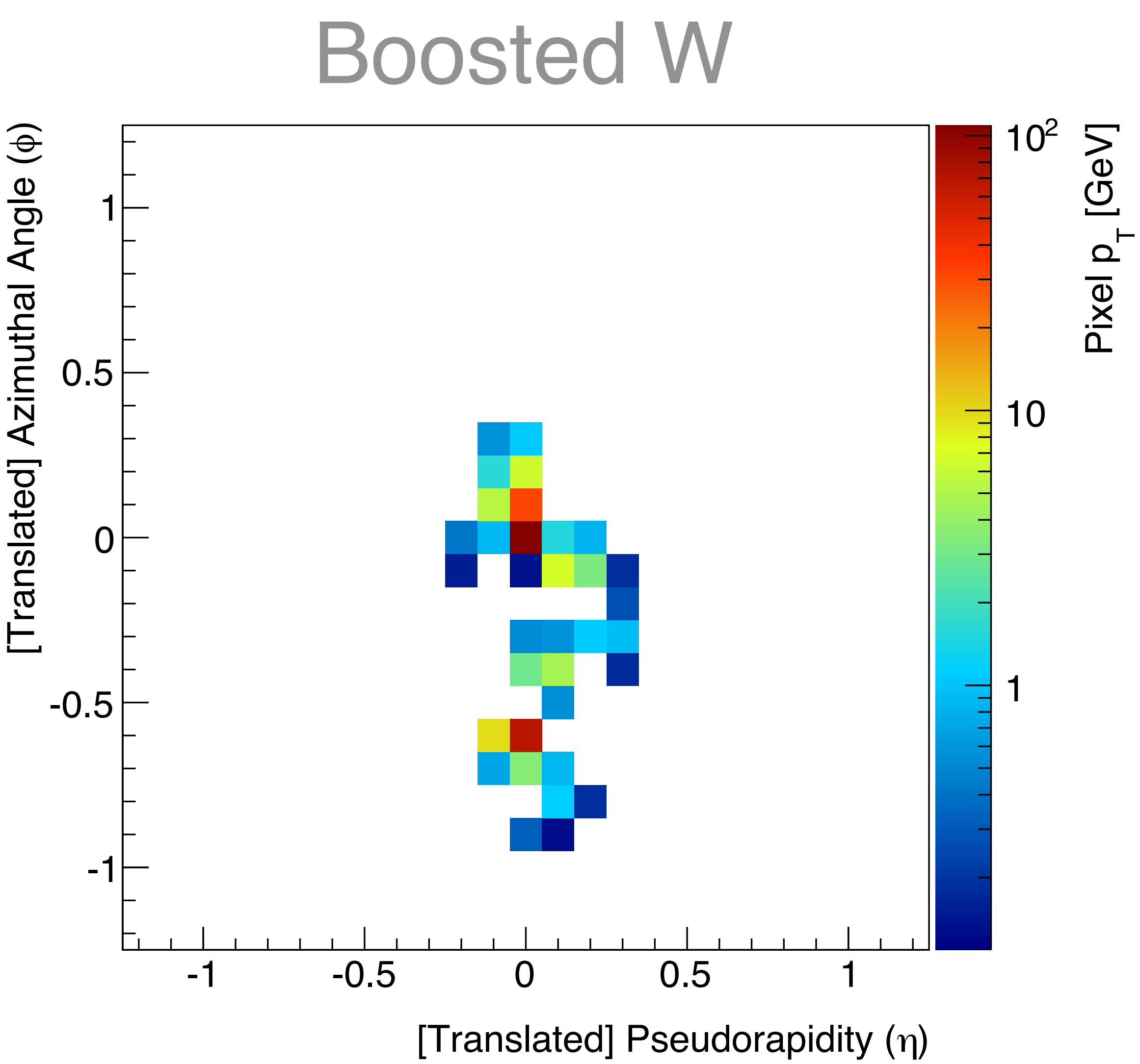


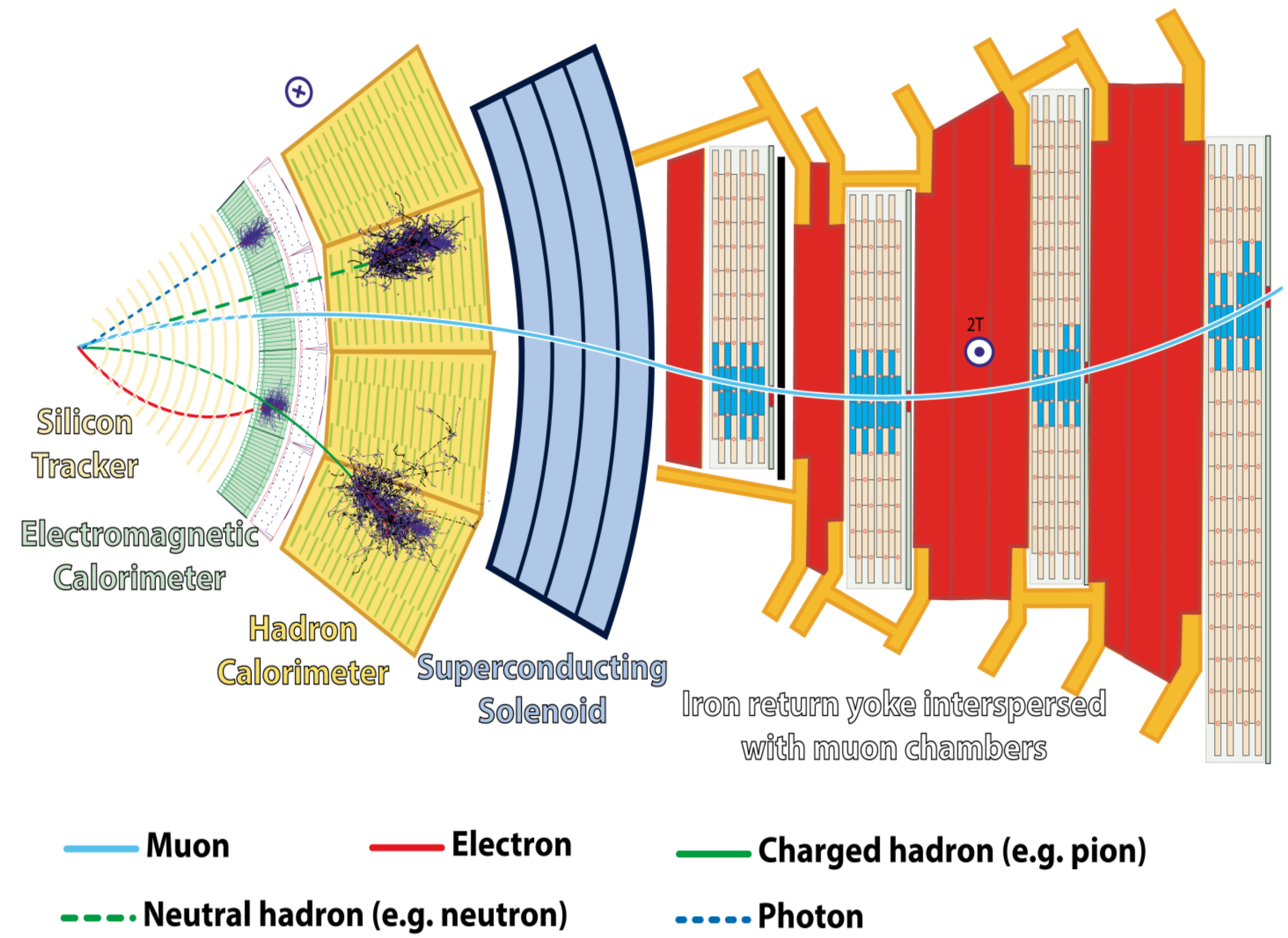


arXiv:1712.07158

Trying more complex architectures

- Convolutional NN successfully applied in neutrino physics and image recognition
- Some proposals to treat jets as images

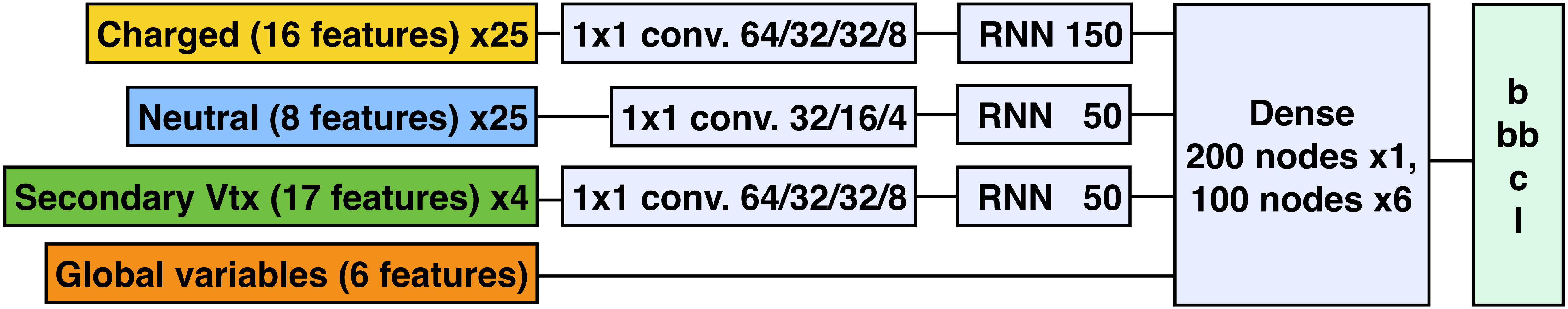




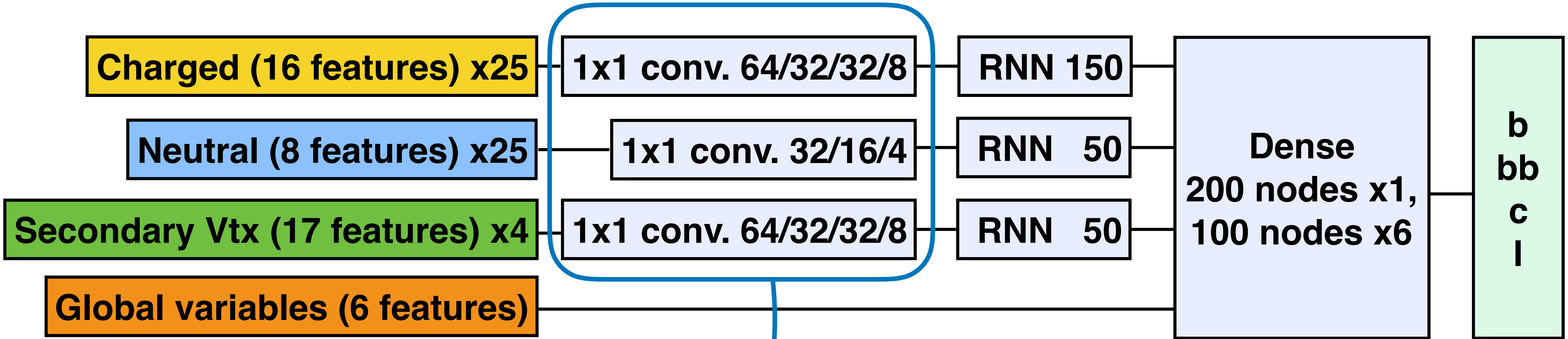
… but

- Jets do not look like normal images!
- CMS events are way more complex and bring more information than a flat image (e.g. tracking information)

Particle-based NN architecture

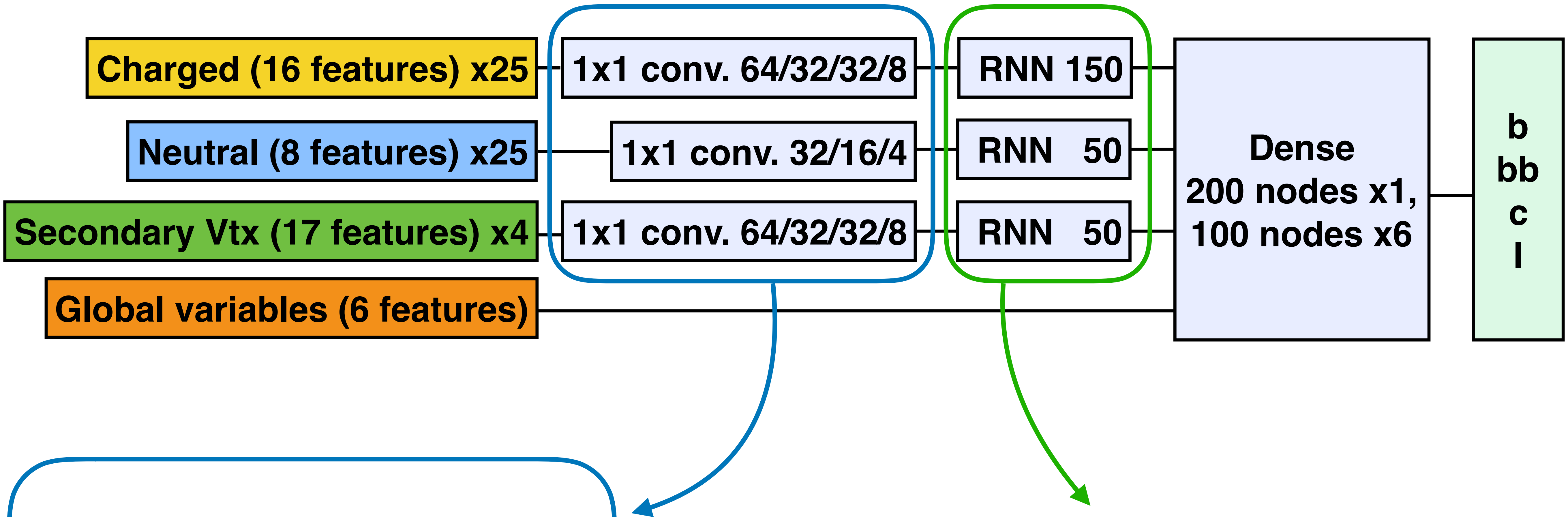


Particle-based NN architecture



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

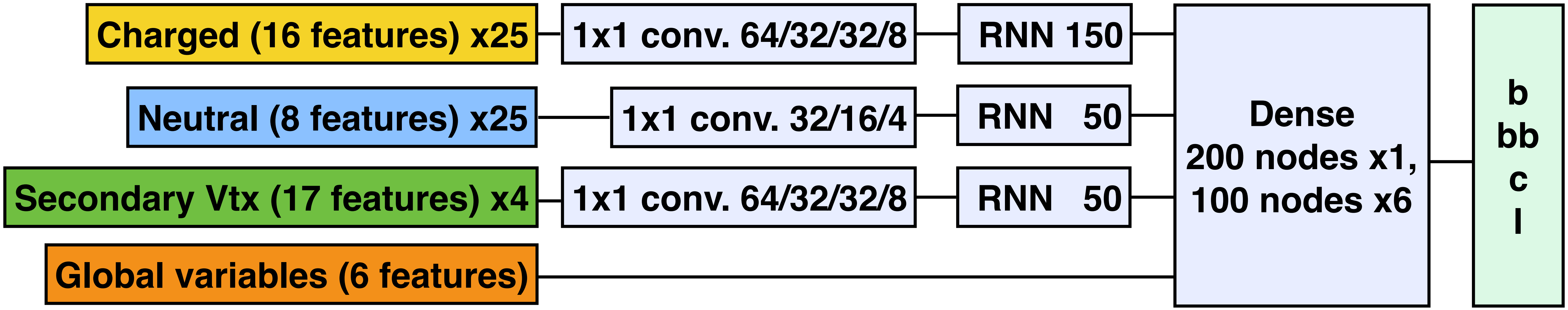
Particle-based NN architecture

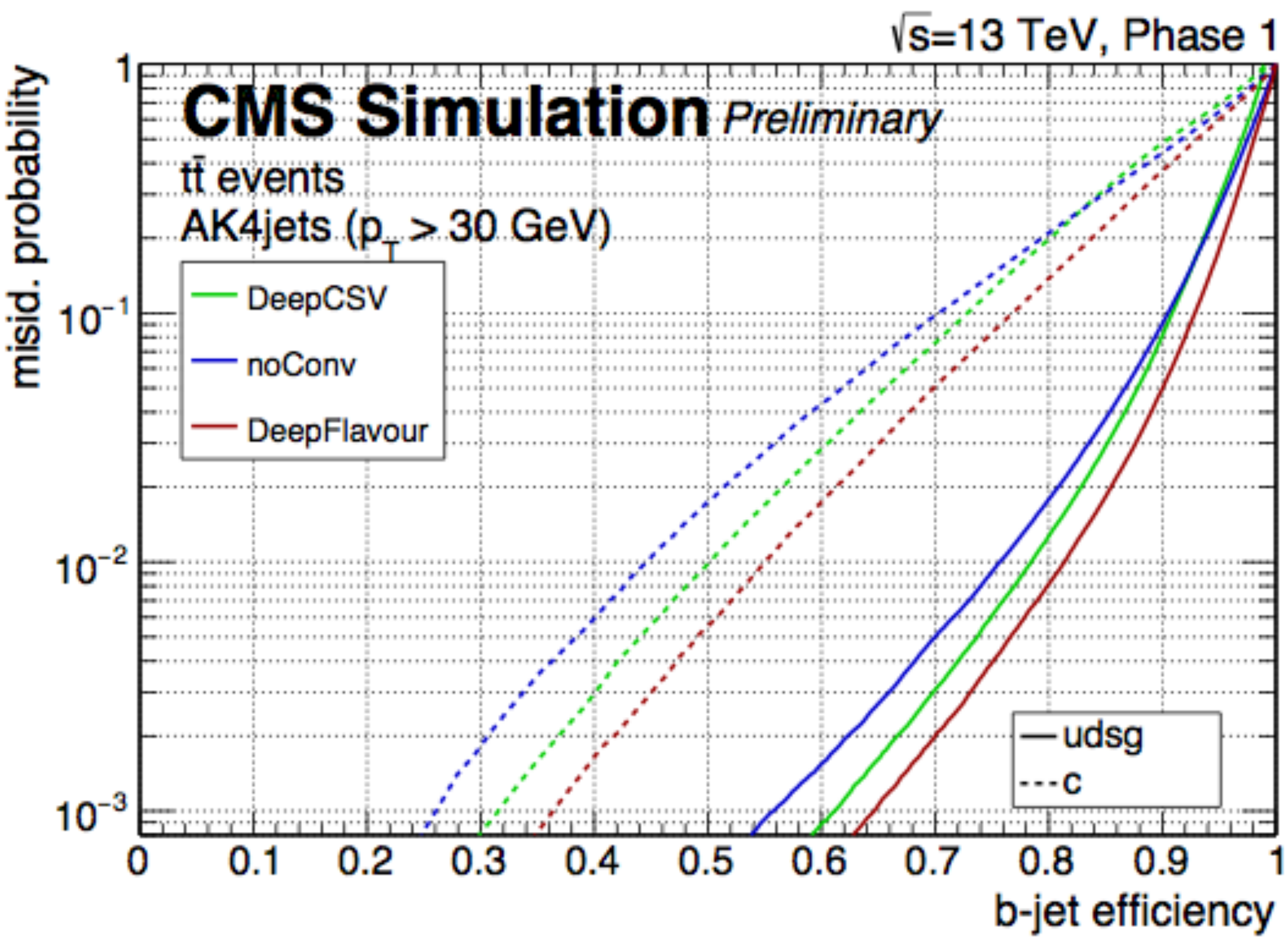


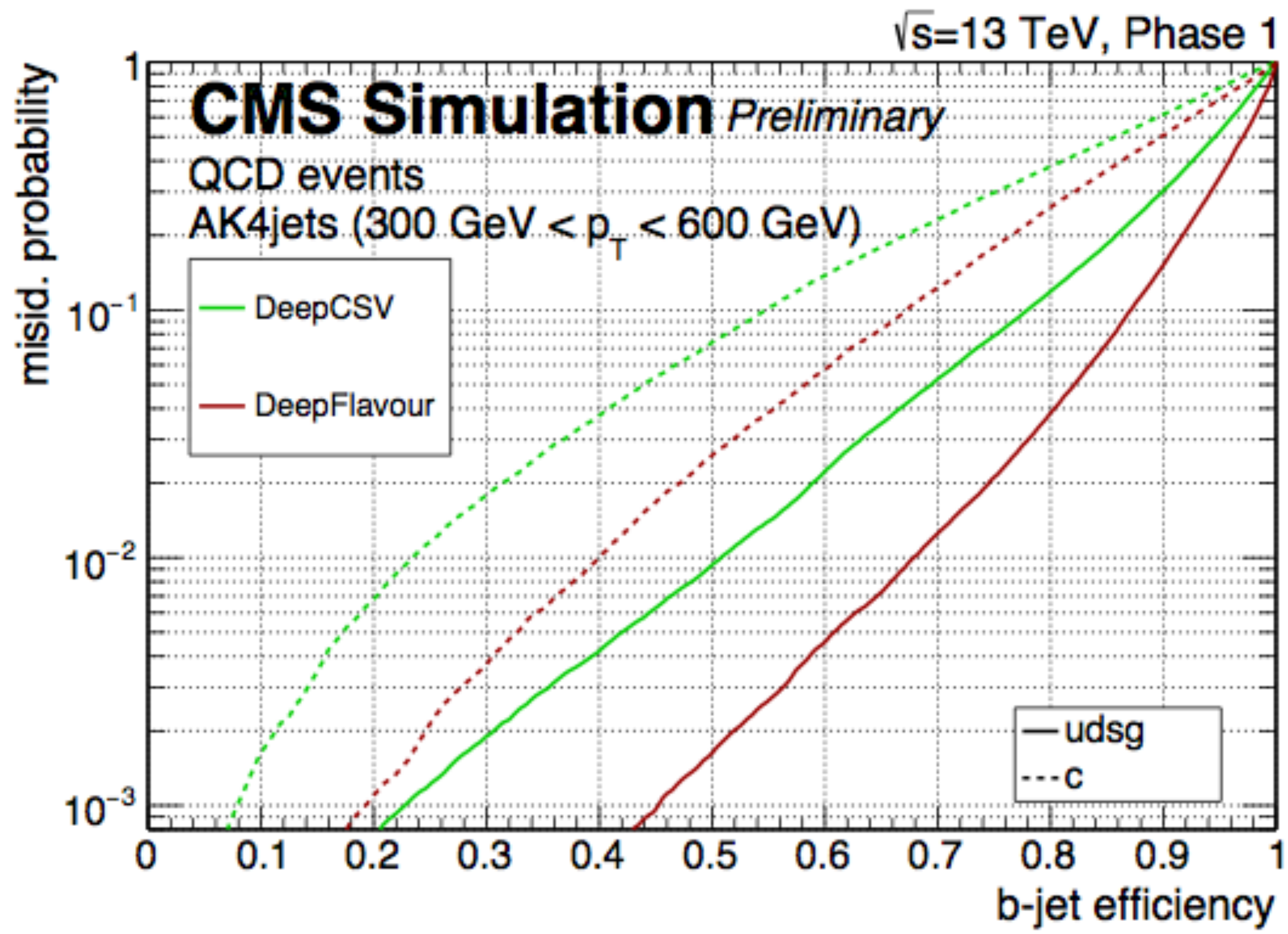
Convolutional layers progressively learn a more compact feature representation (automatic feature engineering)

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

Particle-based NN architecture

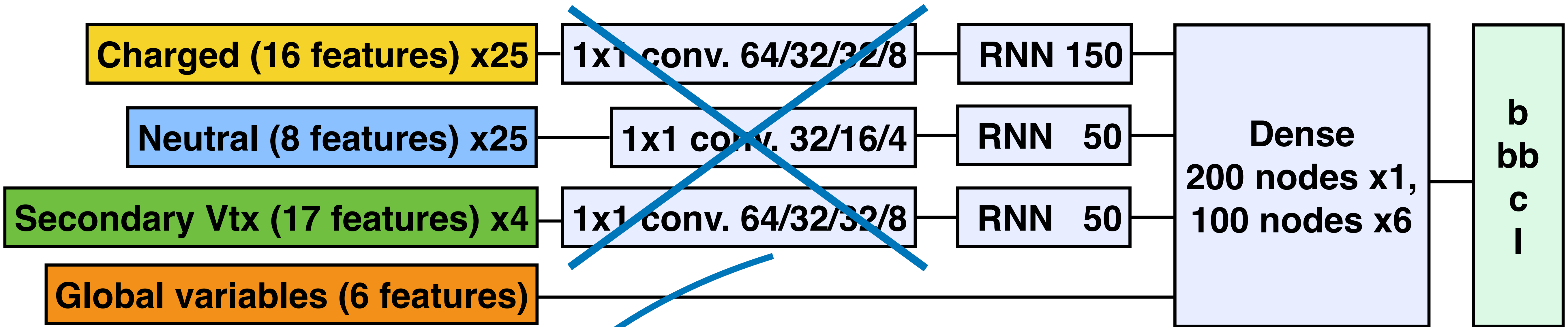


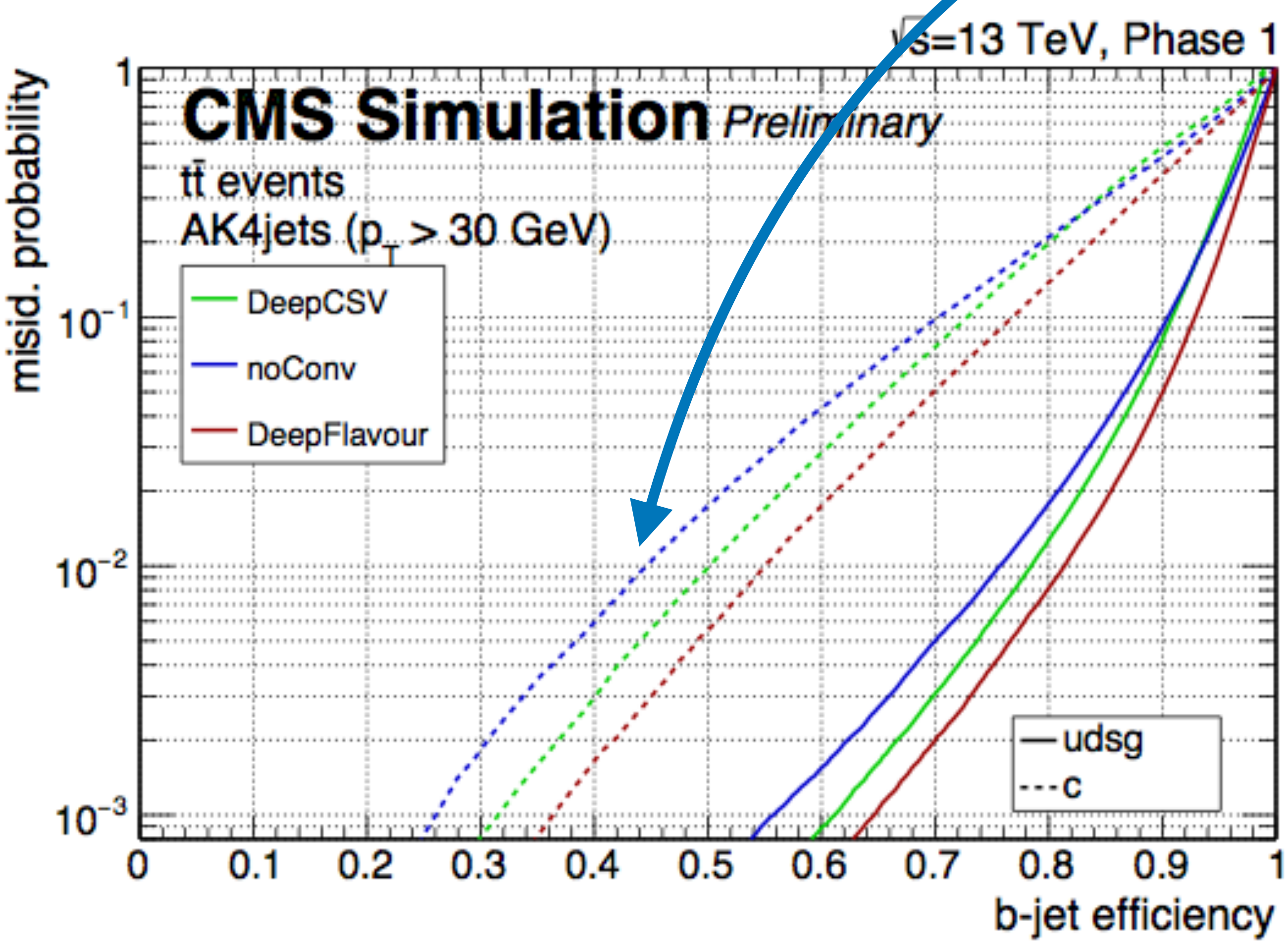


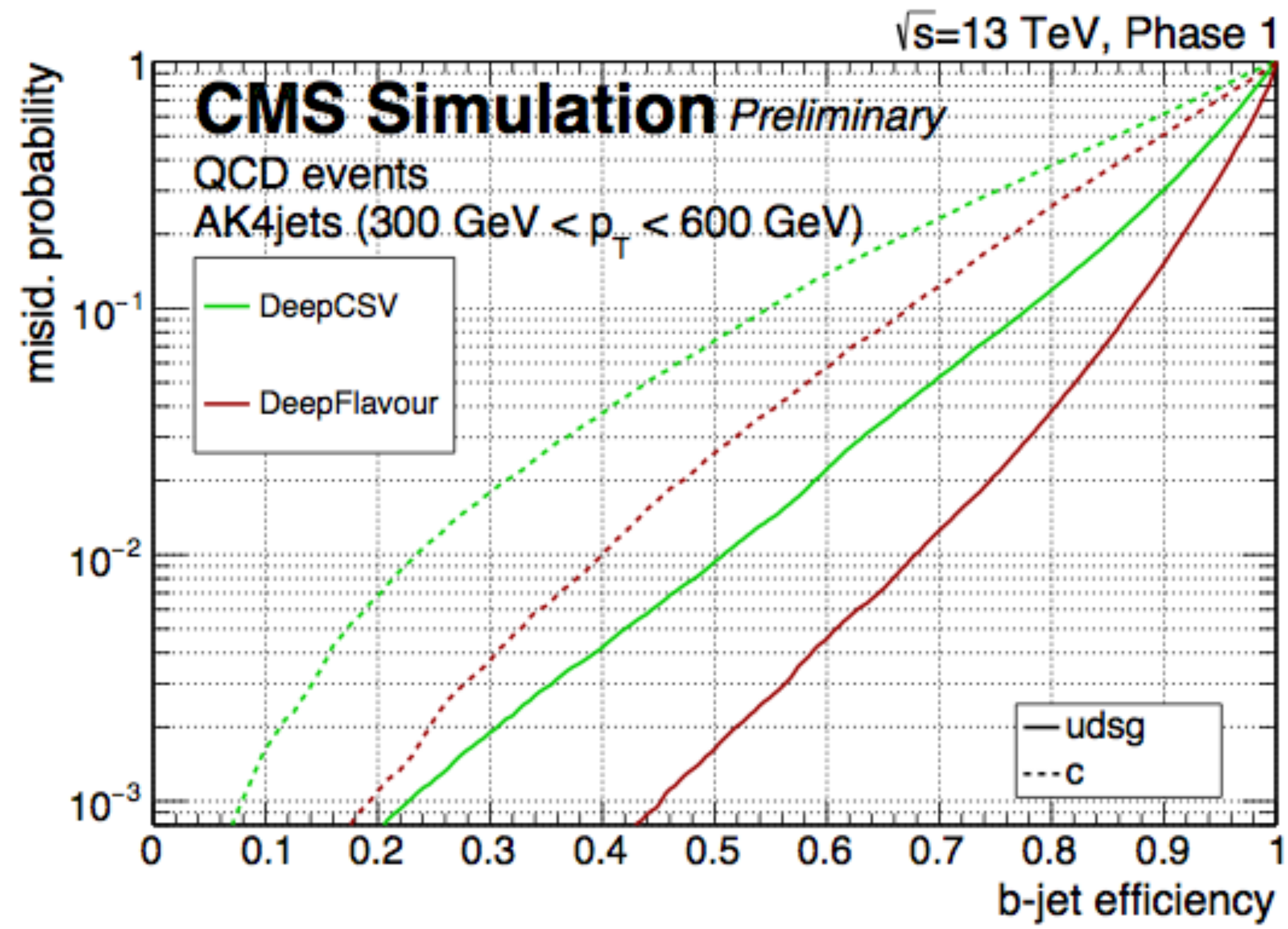


CMS-DP-2017-013

Particle-based NN architecture

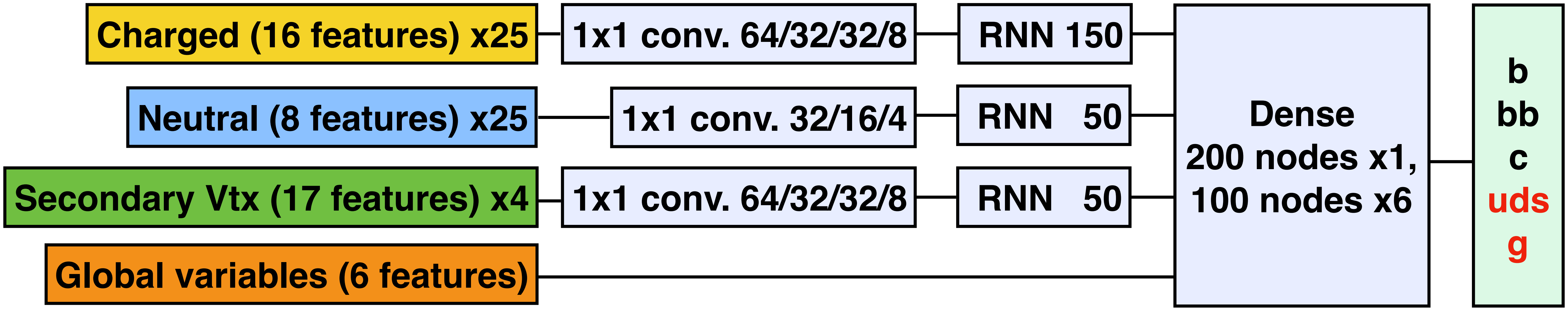






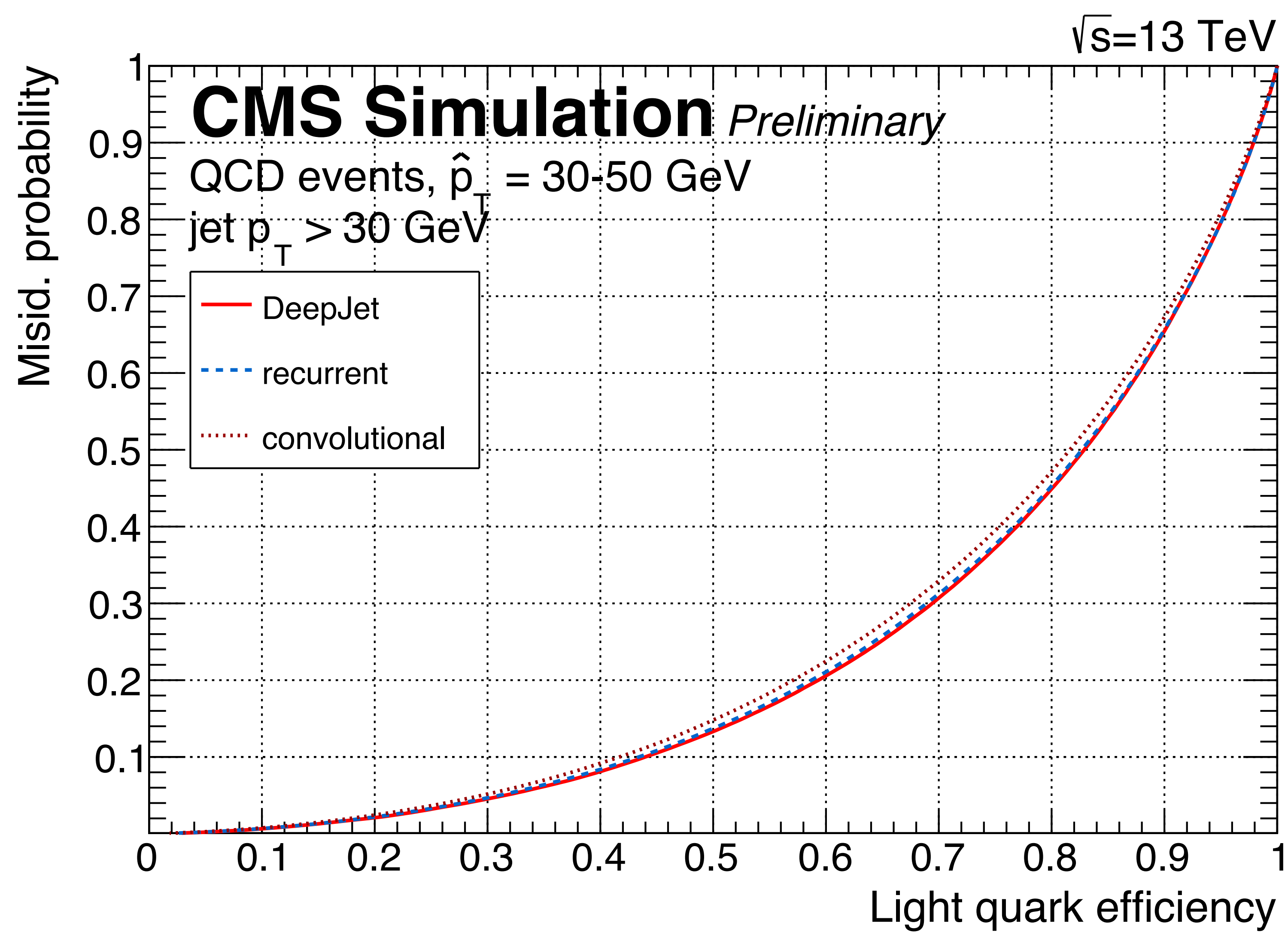
CMS-DP-2017-013

Particle-based NN architecture



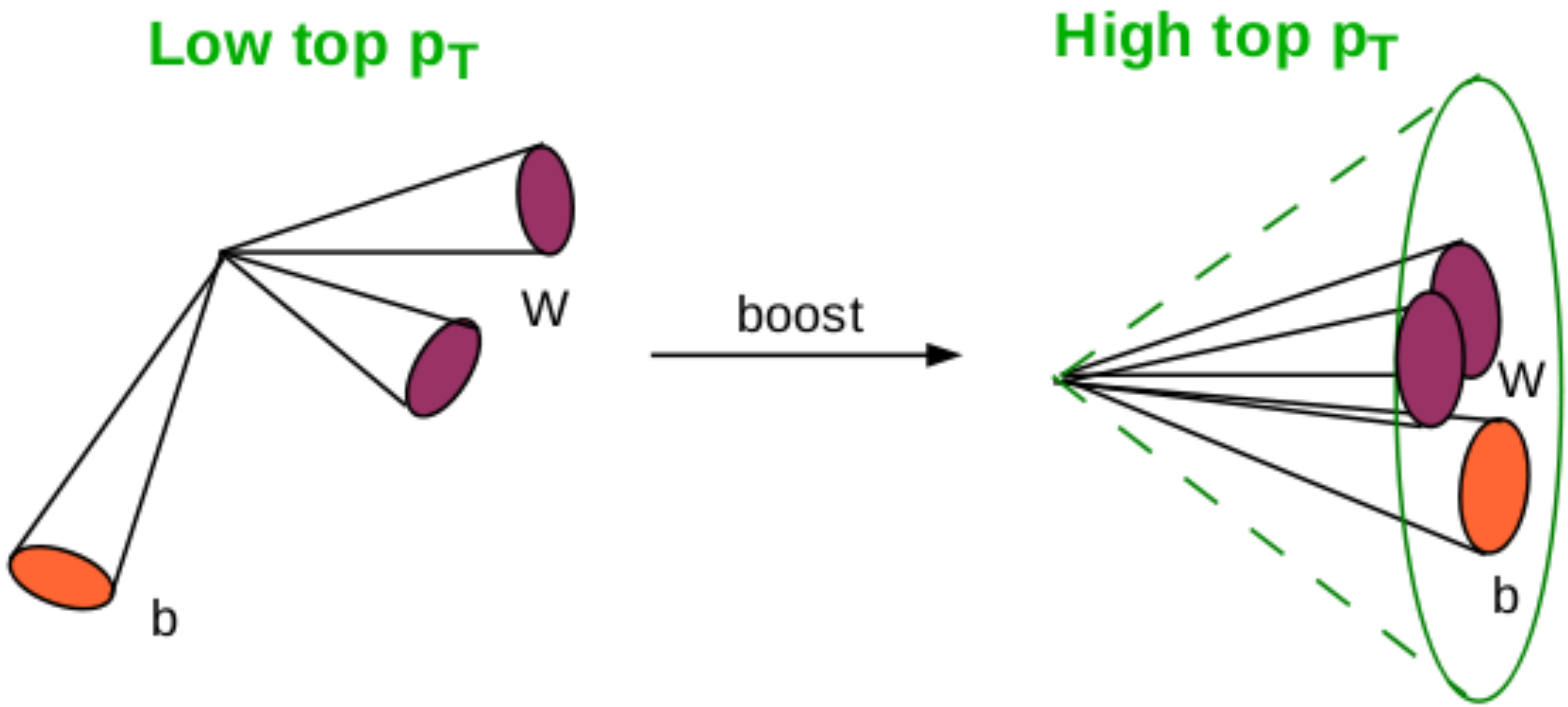
Similar performance to simpler, dedicated binary taggers, but with full multi-class power.

Significantly better performances in given regions with different quark composition

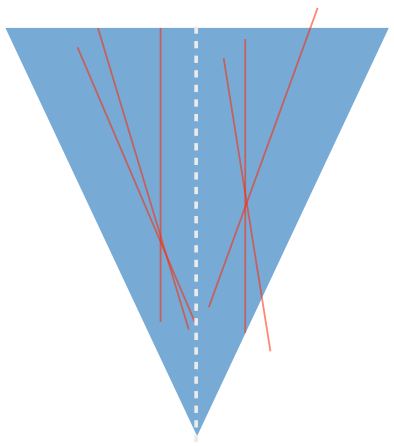


CMS-DP-2017-027

Boosted objects AK8

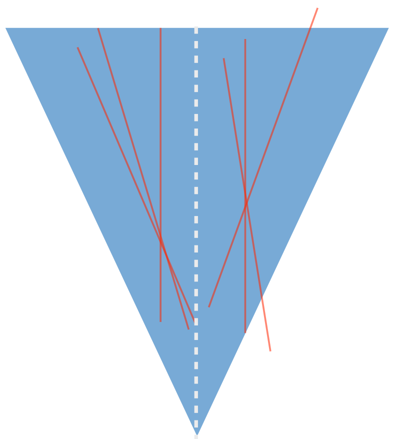


Boosted tagging @ CMS



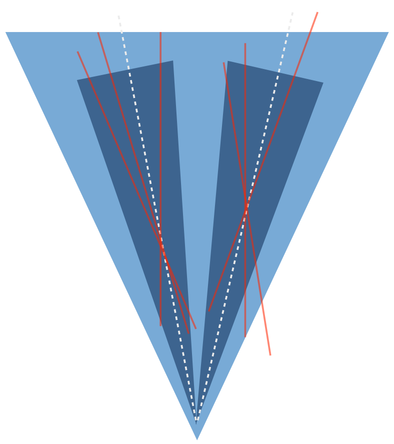
FatJet: CSVv2 w/o retraining. Custom (relaxed) track and SV association directly on anti-k_T 0.8

Boosted tagging @ CMS



fatjet

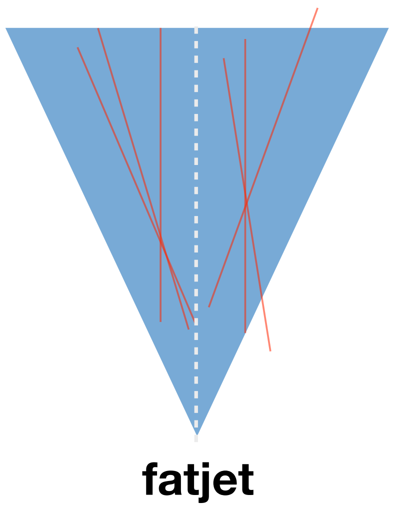
FatJet: CSVv2 w/o retraining. Custom (relaxed) track and SV association directly on anti-k_T 0.8



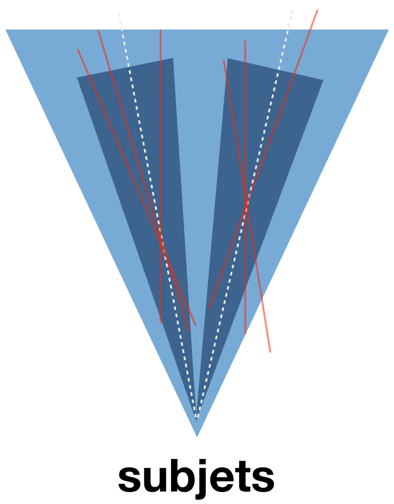
subjets

Sub-jet: CSVv2 w/o retraining applied to sub-jets (soft drop, pruned, etc…)

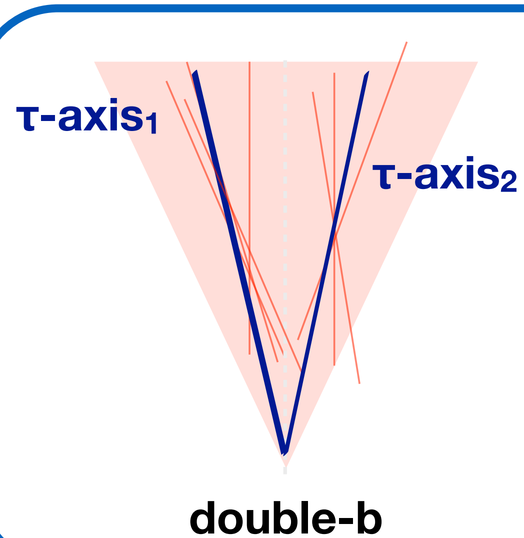
Boosted tagging @ CMS



FatJet: CSVv2 w/o retraining. Custom (relaxed) track and SV association directly on anti-k_T 0.8



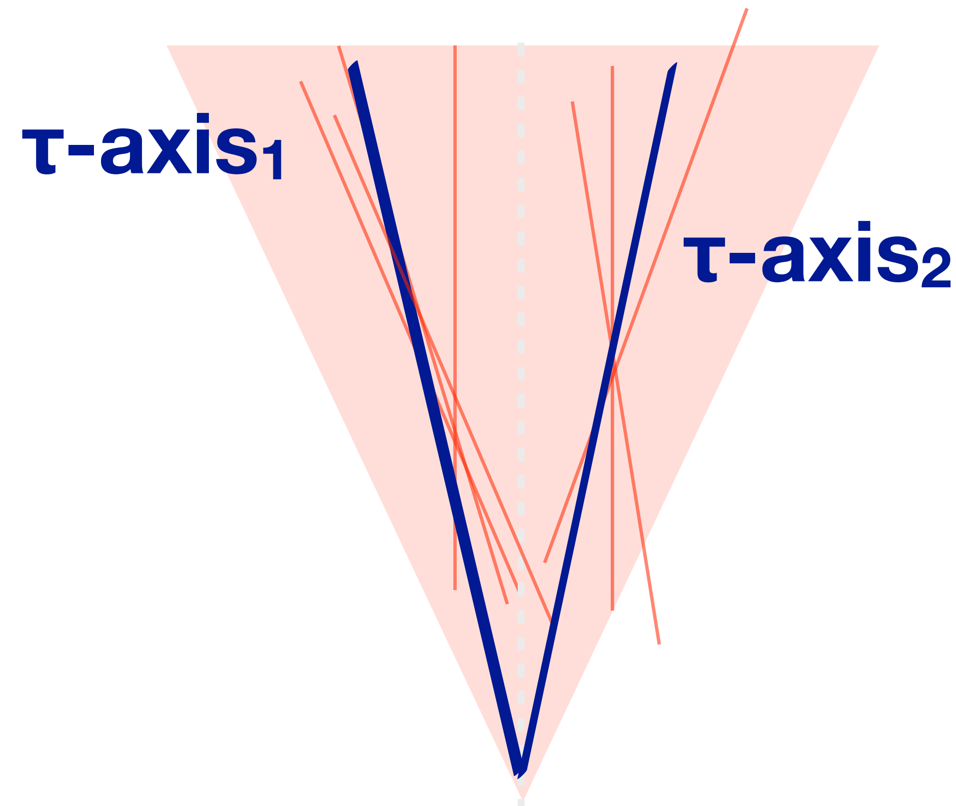
Sub-jet: CSVv2 w/o retraining applied to sub-jets (soft drop, pruned, etc…). Used for boosted top



Double b: dedicated training targeting boosted resonances X→bb

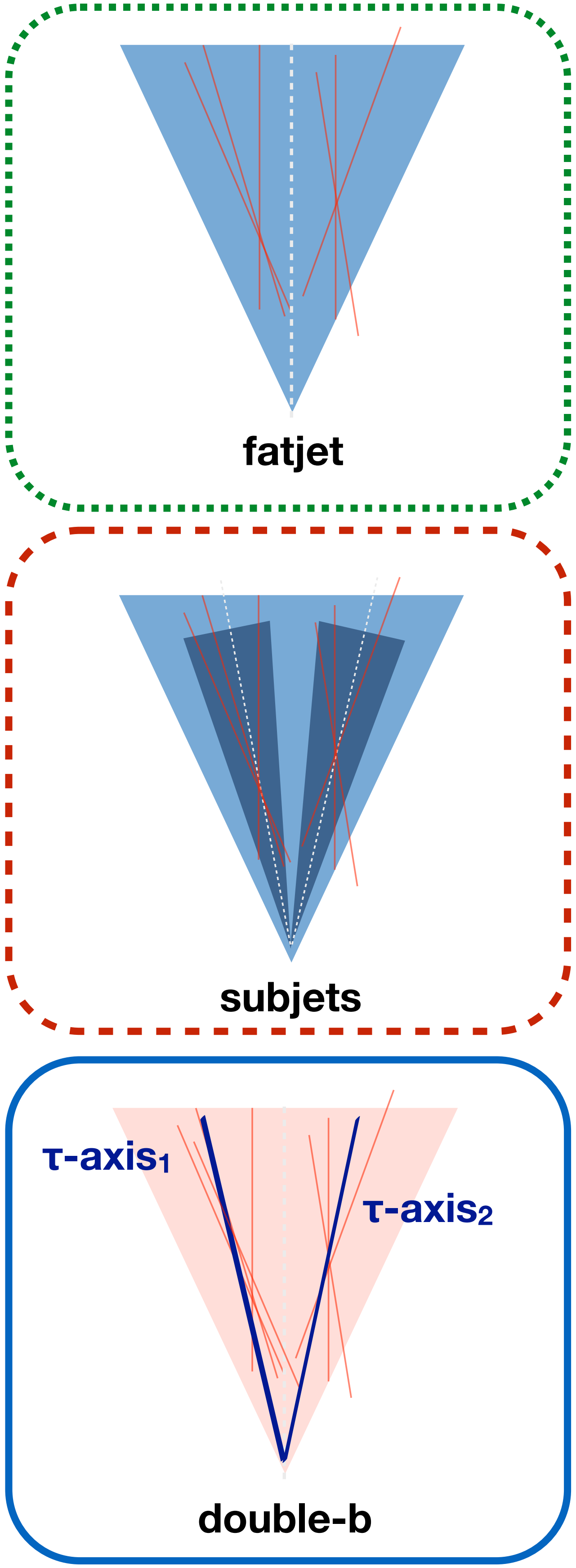
Double-b tagger

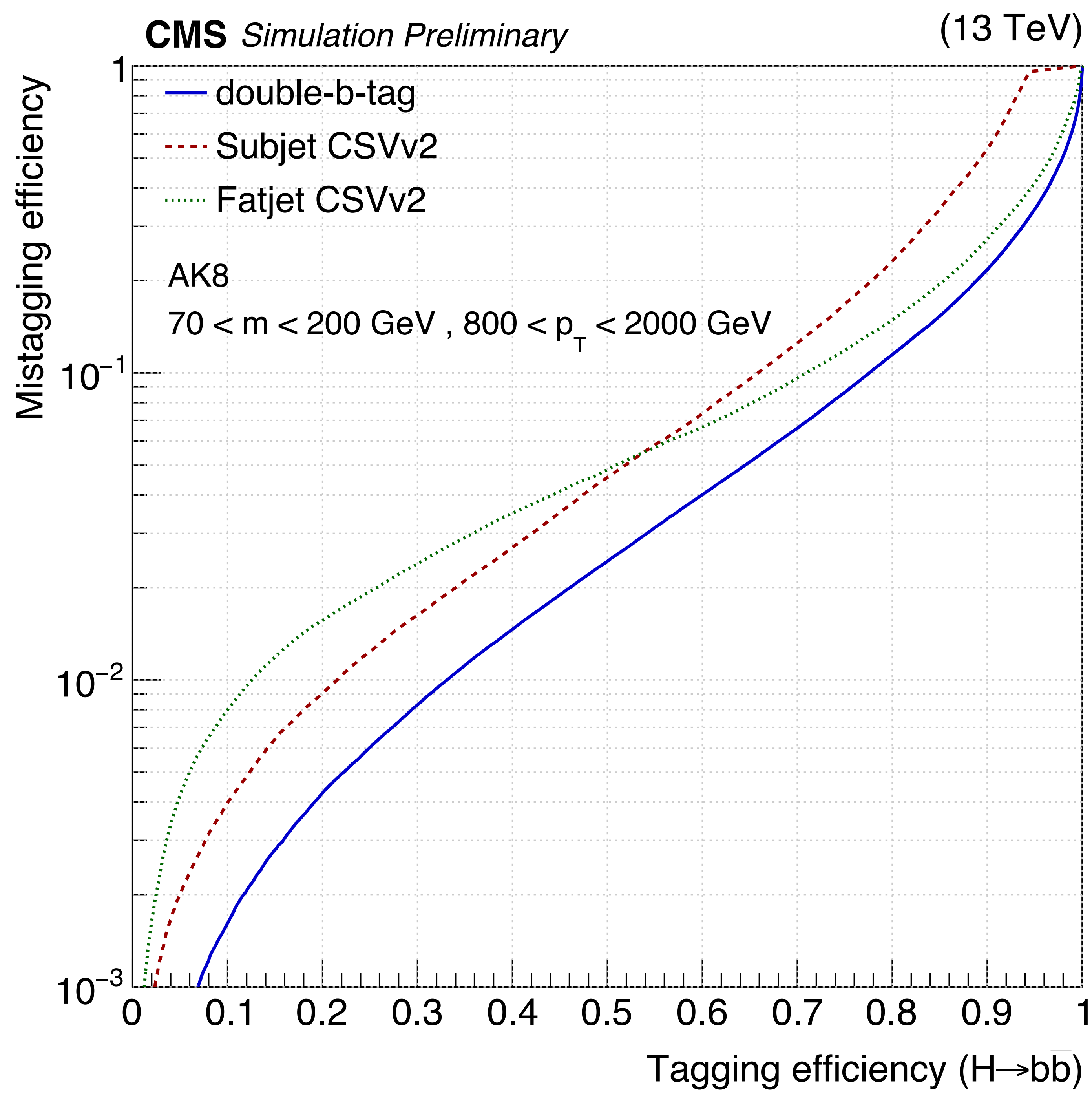
- All input features ~duplicated to account for the two sub-jet axes
- The input features are checked to be ~independent from the jet p_T and mass to ease background estimation in the analyses
- A total of 27 input features combined in a BDT



double-b

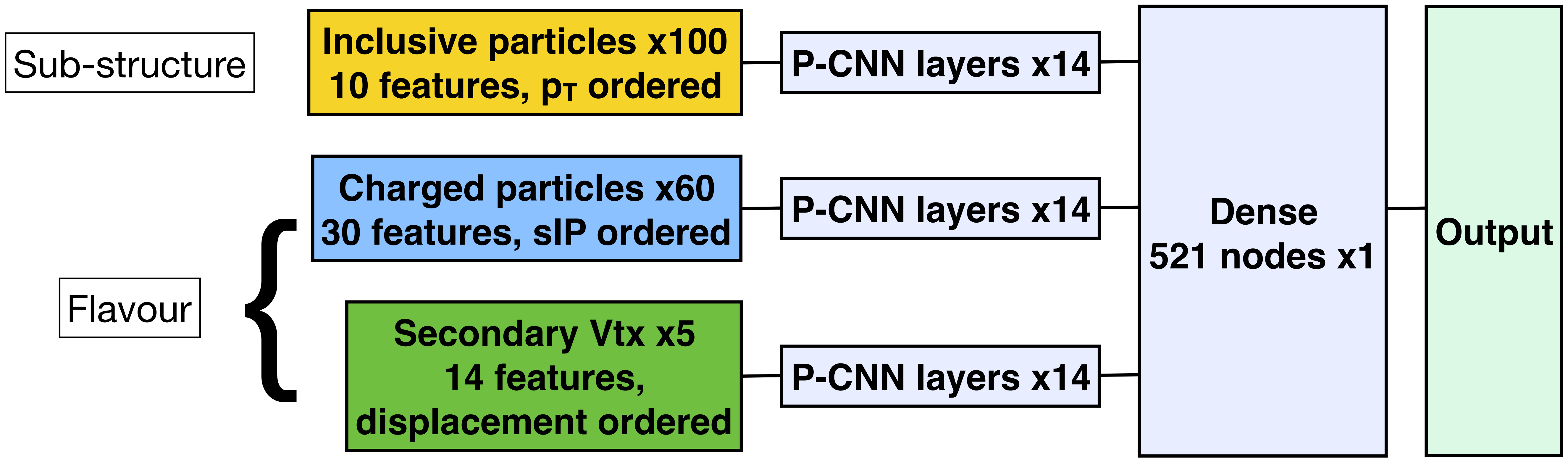
Boosted tagging @ CMS





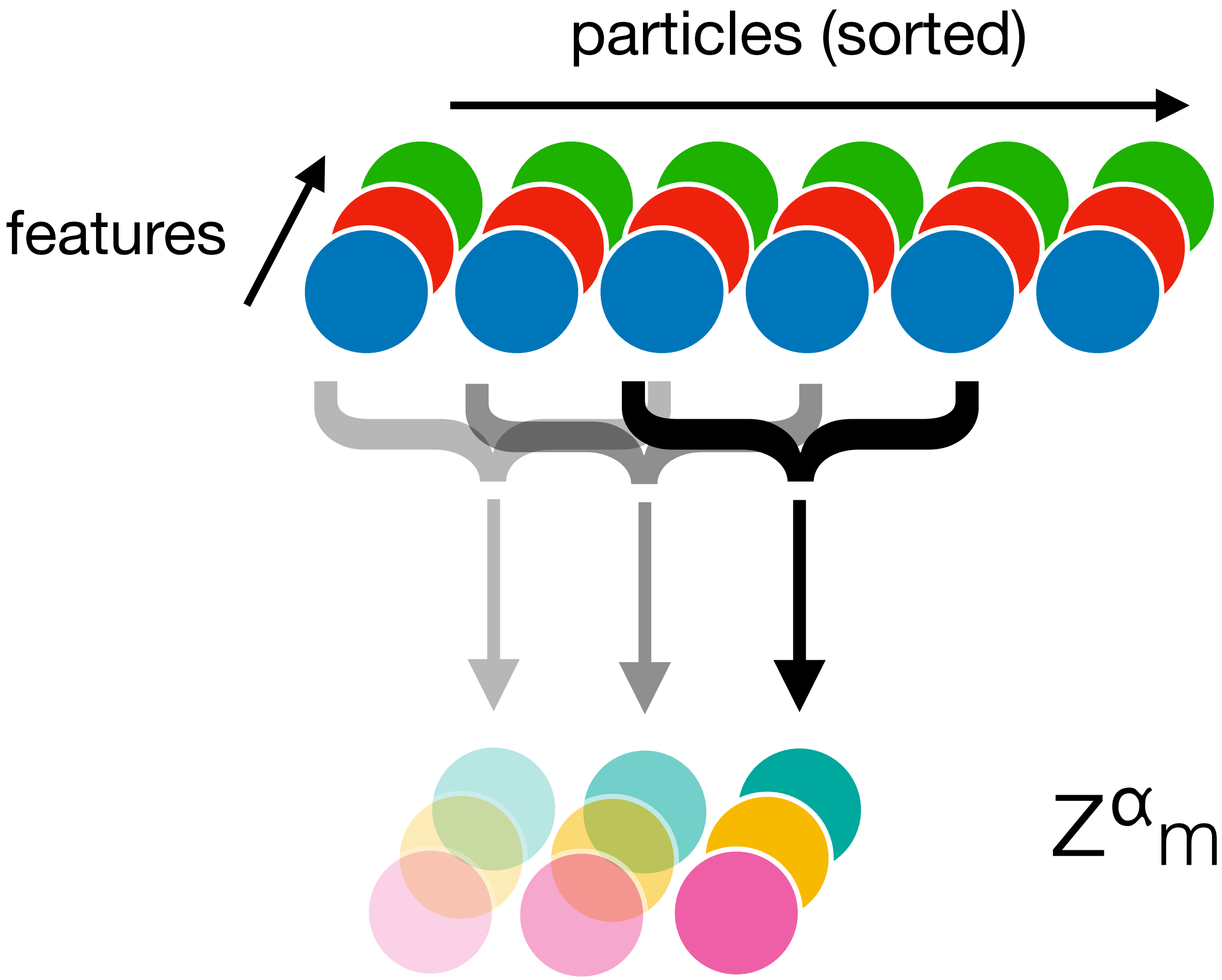
BTV-13-001

DeepJet for boosted resonances



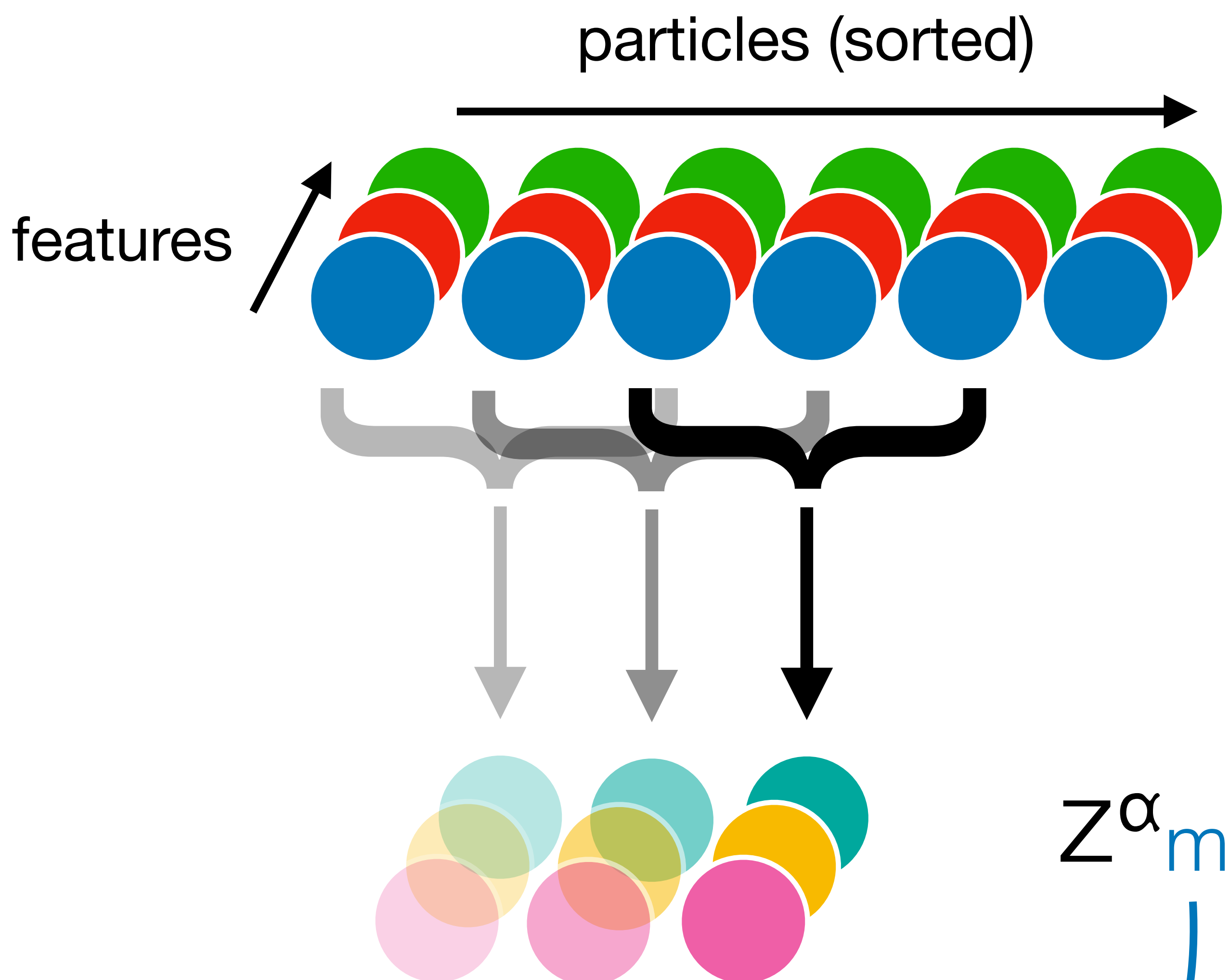
- Significantly larger amount of candidates used to accomodate for 90% of the fat jets
- Need to learn substructure from both charged and neutral candidates
- RNNs become computationally too expensive to train
- Use particle-level convolutional layers (P-CNN) where each feature is treated as a “colour”

P-CNNs



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

P-CNNs

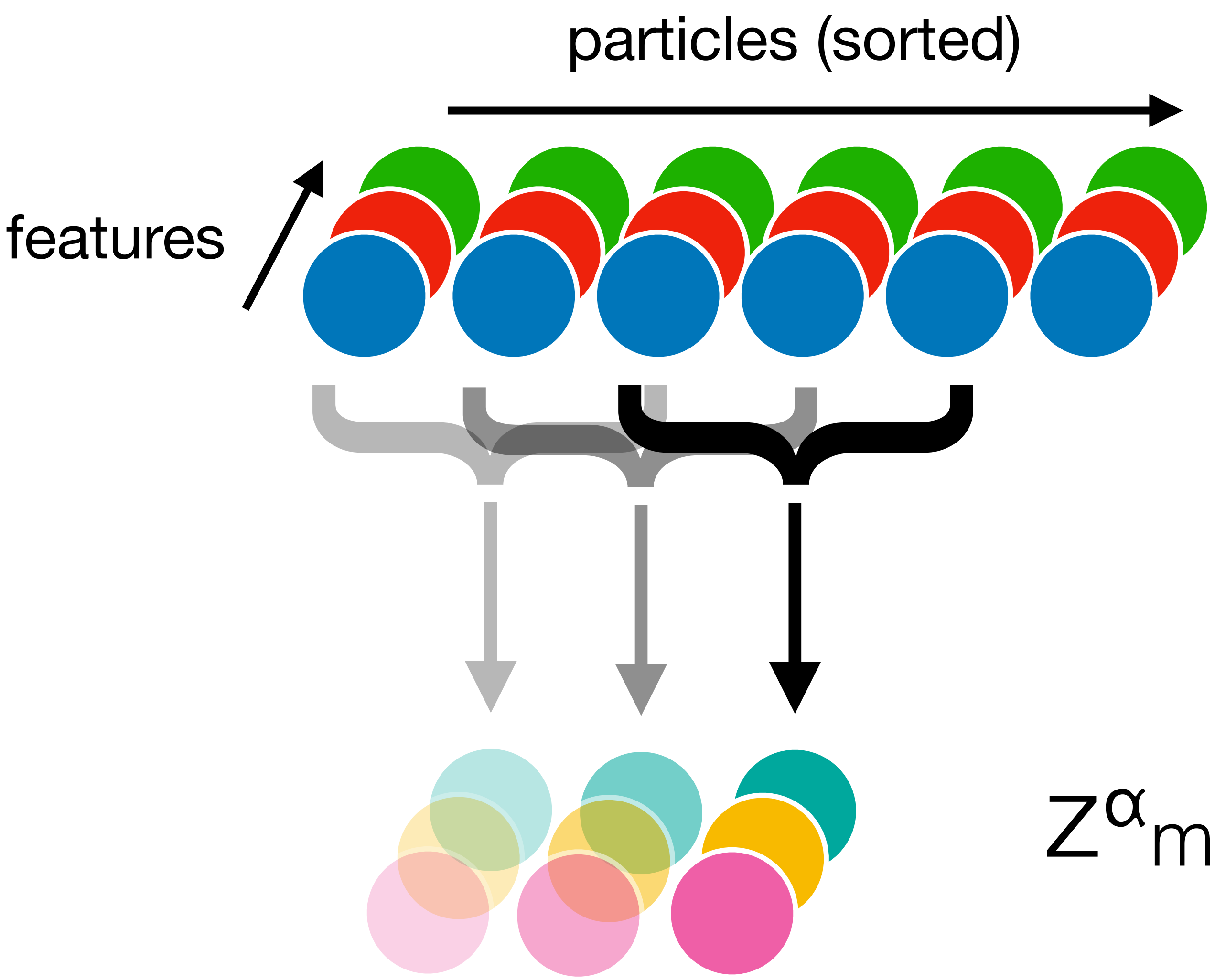


$$z^{\alpha}_{m} = \Sigma_a \Sigma_j \, k^{\alpha}_{a,j} \, x_{a,(m+j-1)}$$

Sweep over the elements

Loop over contiguous elements of the kernel

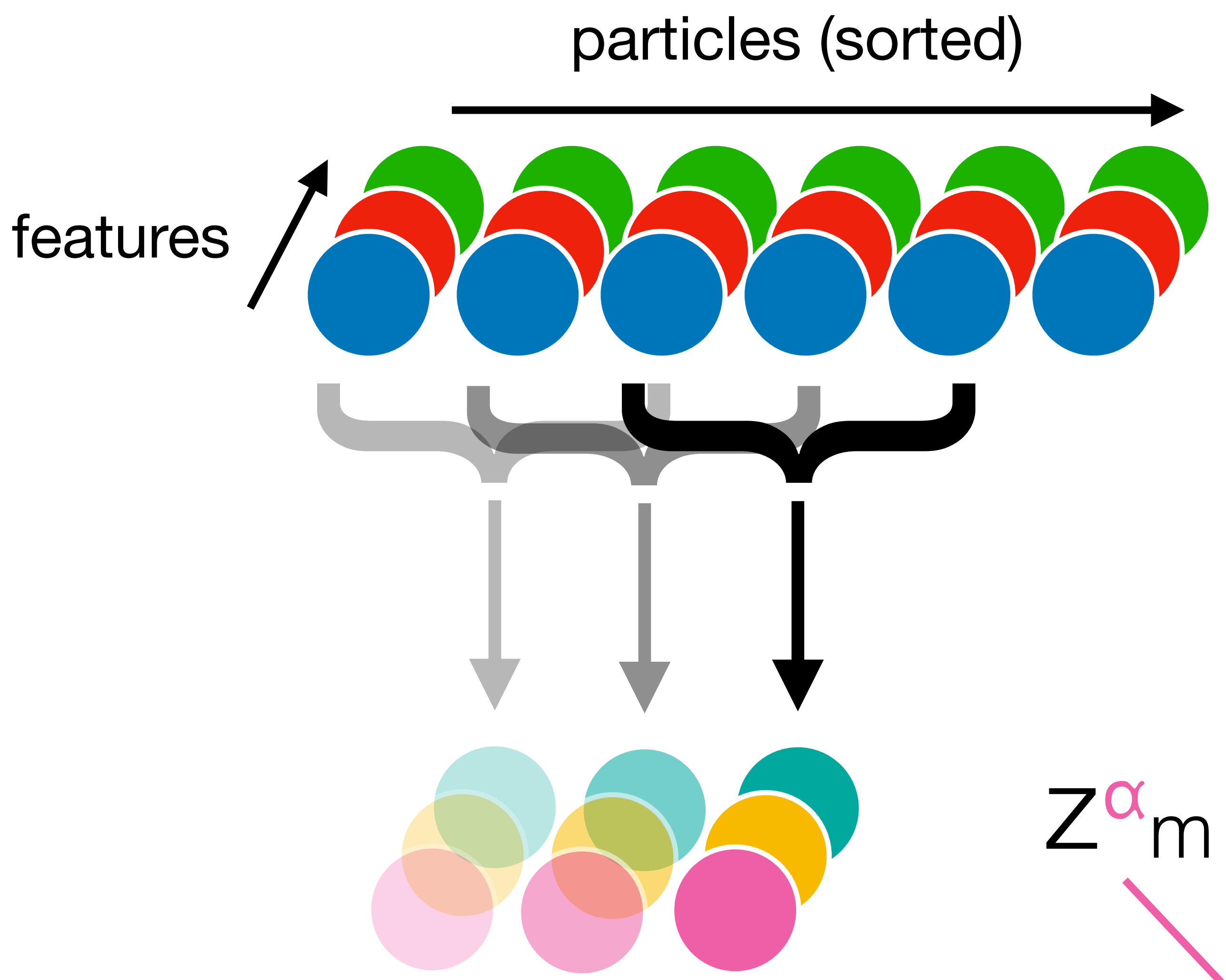
P-CNNs



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

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

P-CNNs

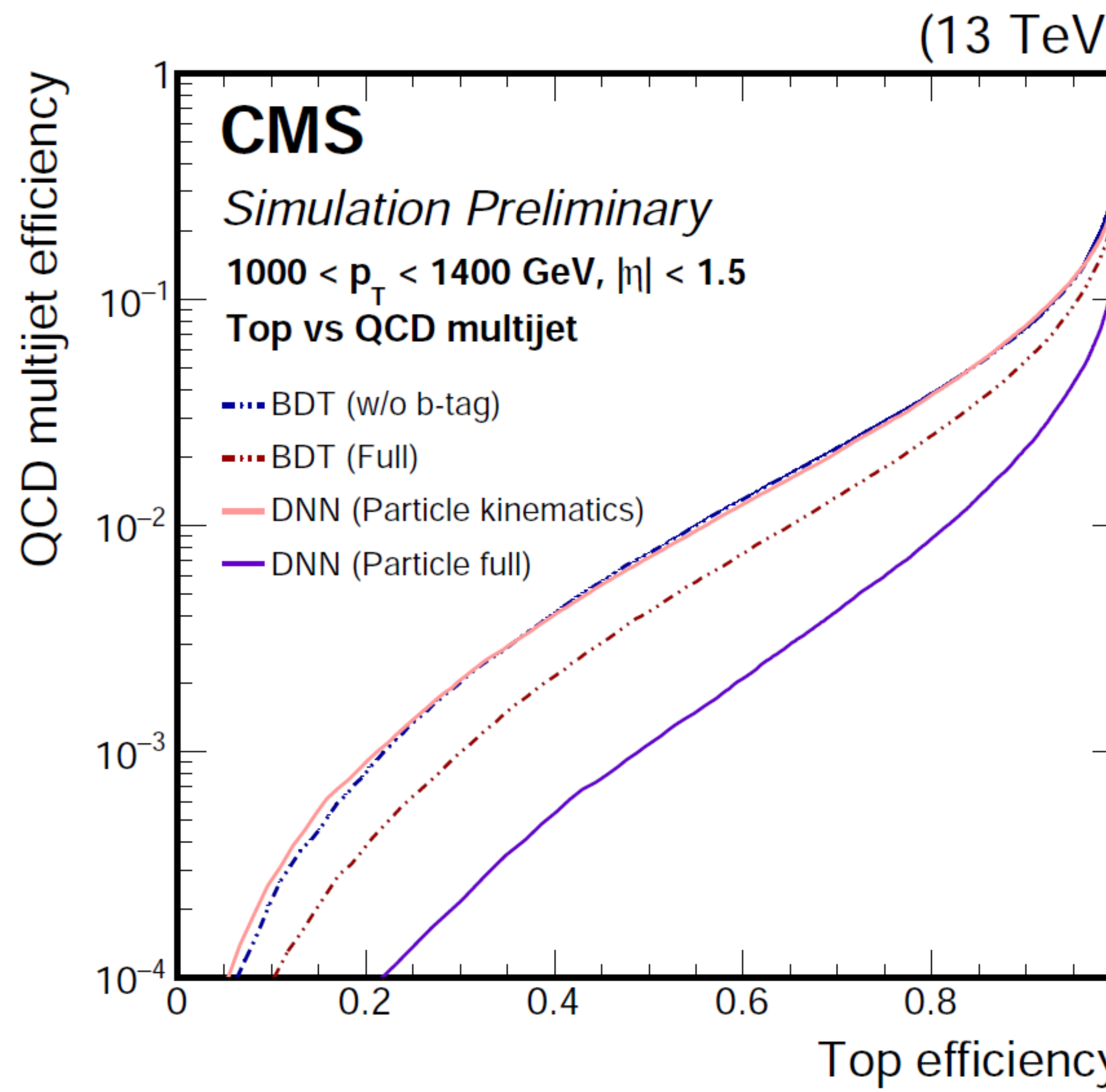


$$z^{\alpha}_{m} = \Sigma_a \Sigma_j \, k^{\alpha}_{a,j} \, x_{a,(m+j-1)}$$

Different filters/kernels learn
different transformations

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

From training to practice

Two worlds colliding

Training / Analysis:

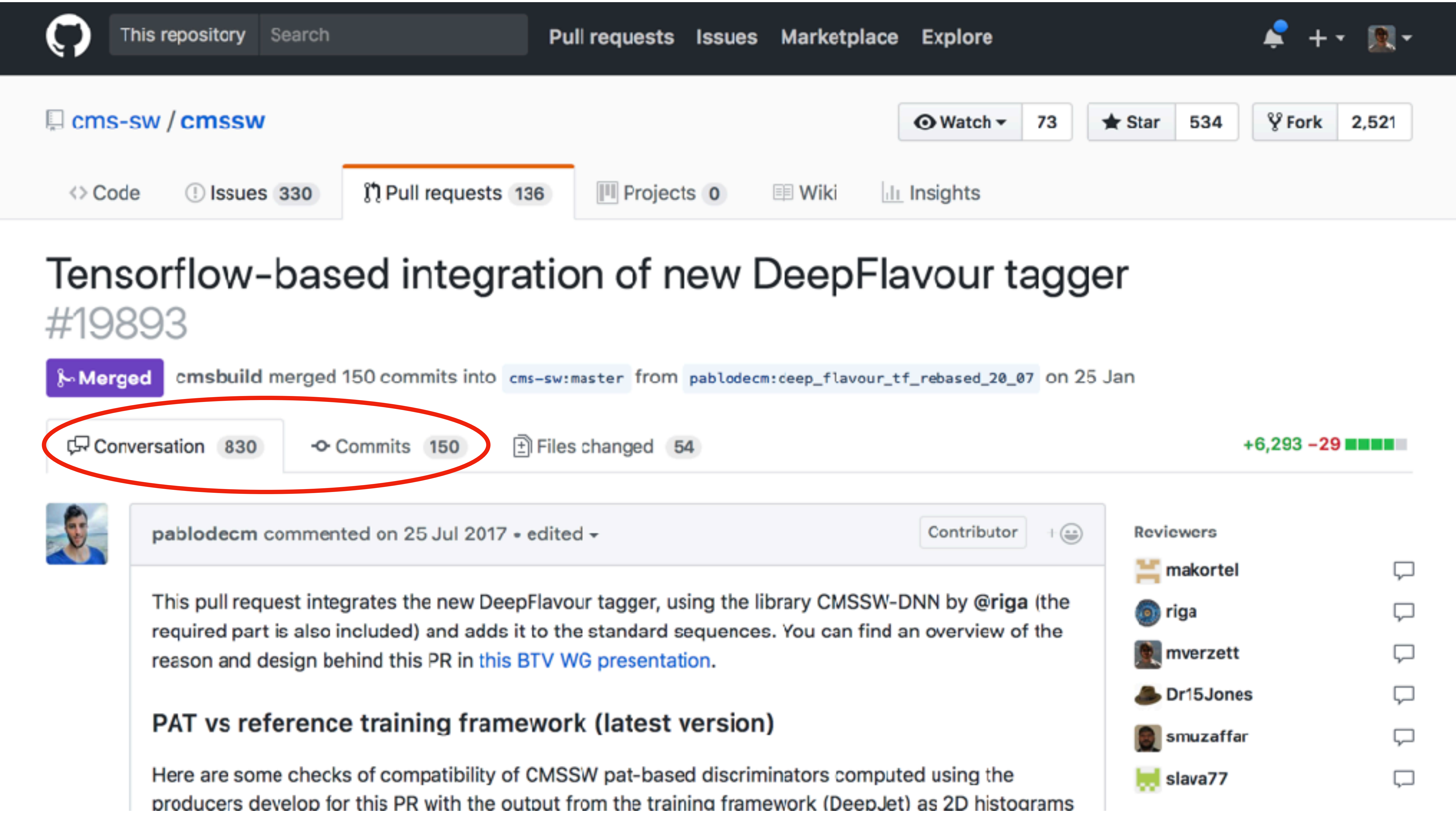
- Keras + TensorFlow
- Python-based
- Private productions
- Minimal interaction with ROOT
- Few processes, single threads
- Little memory constraints
- Expendable jobs

+

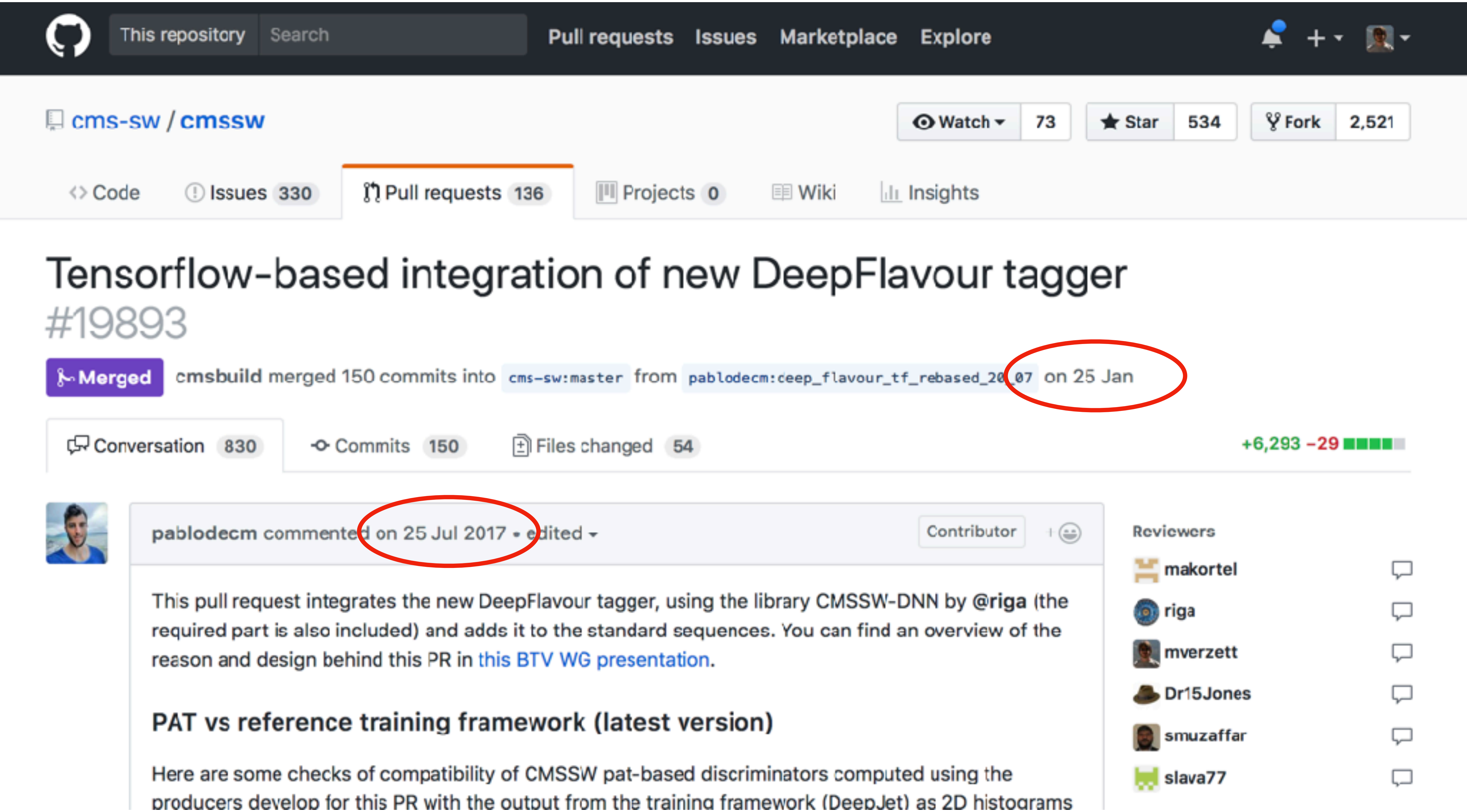
Production:

- 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 DeepJet (AK4) into CMSSW. PR #19893



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



M. Verzetti (CERN and FWO)

Backend choice

x Interface based on TF python API:

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

x 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?

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