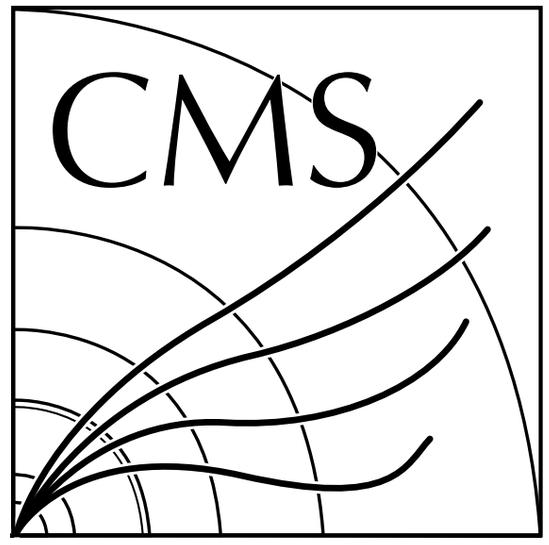
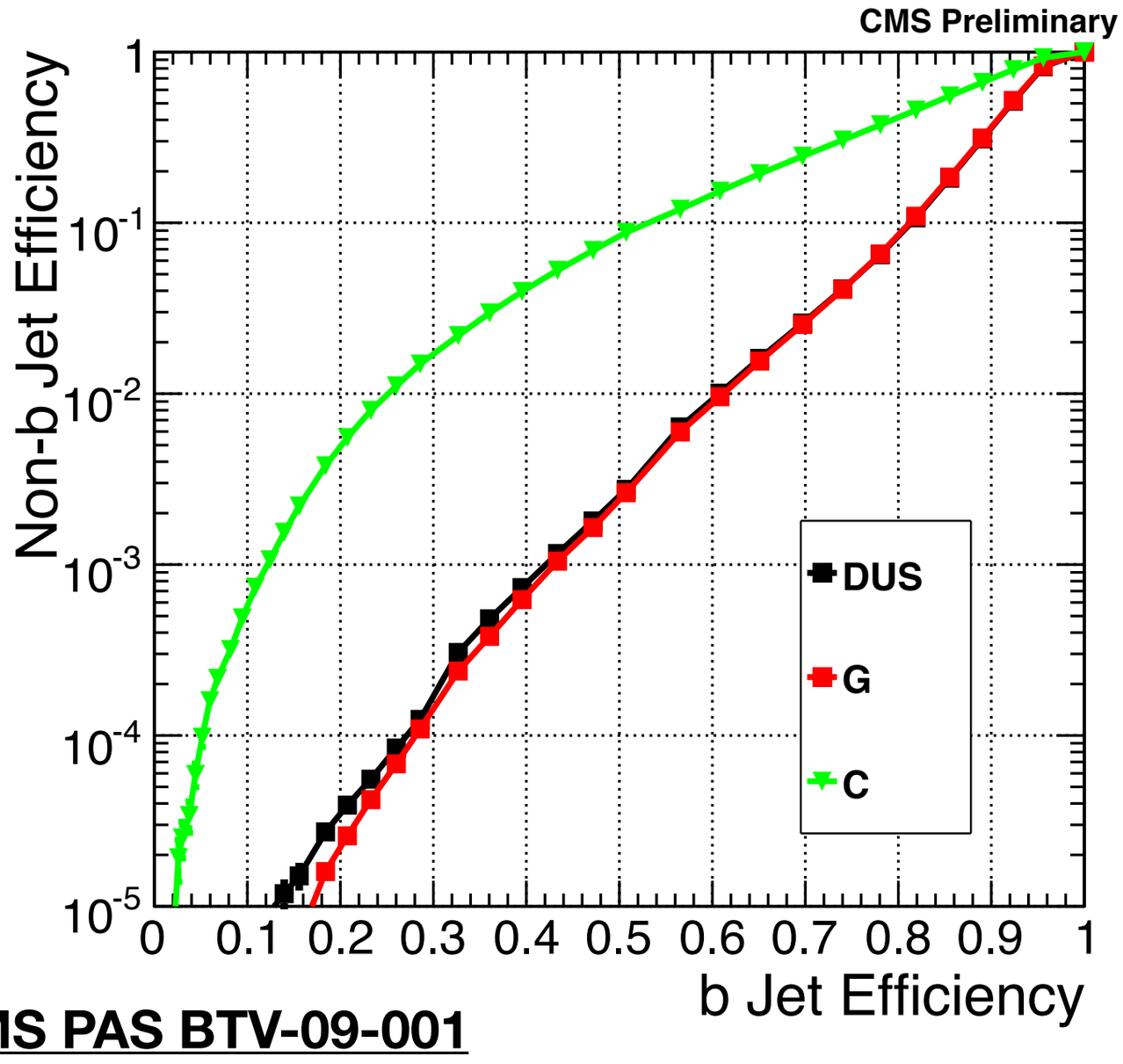
Machine learning for jets in CMS

Mauro Verzetti
on behalf of the CMS Collaboration



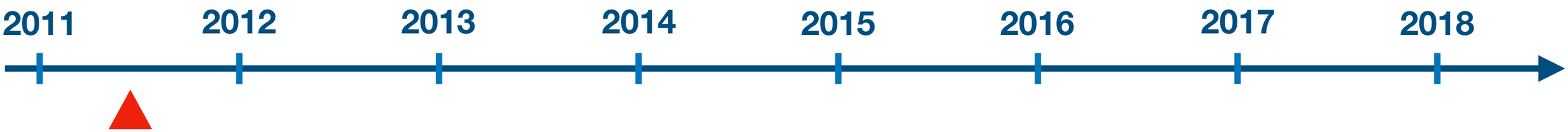
Jet Probability

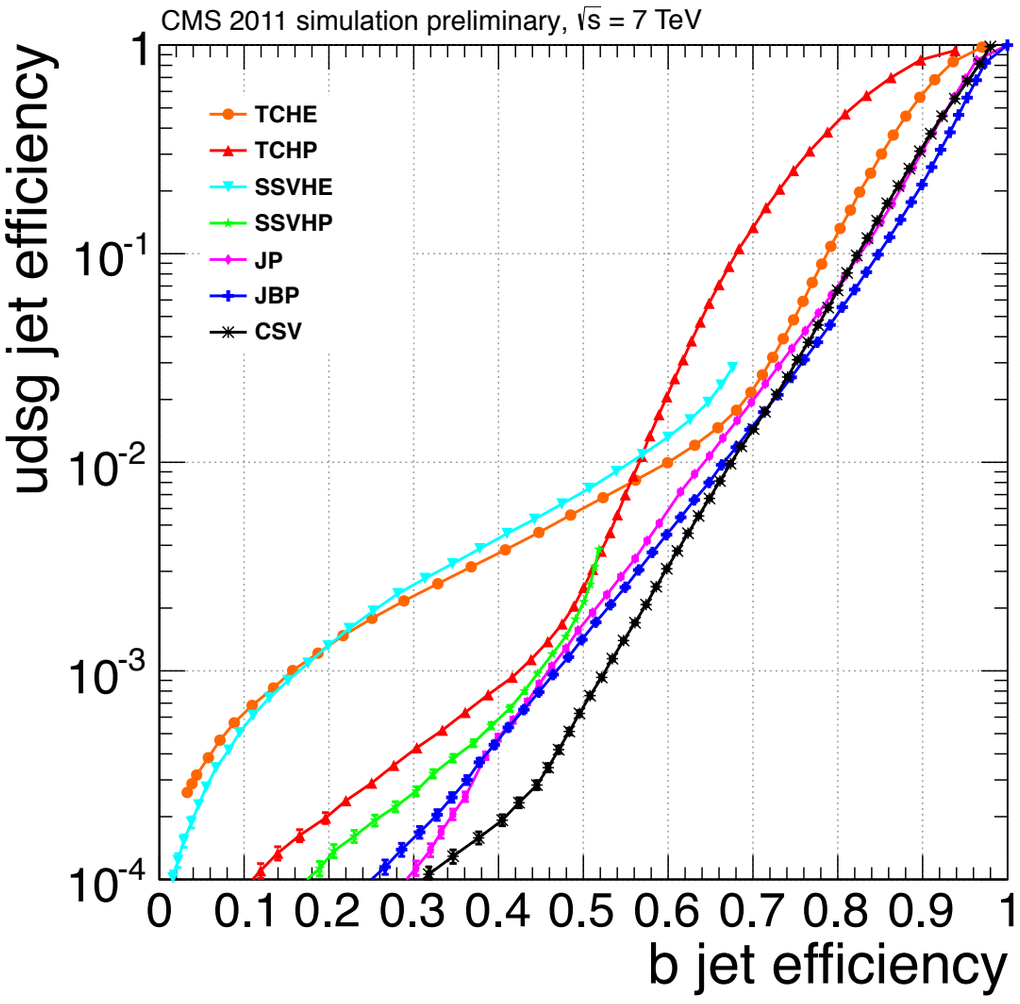
2011 2012 2013 2014 2015 2016 2017 2018

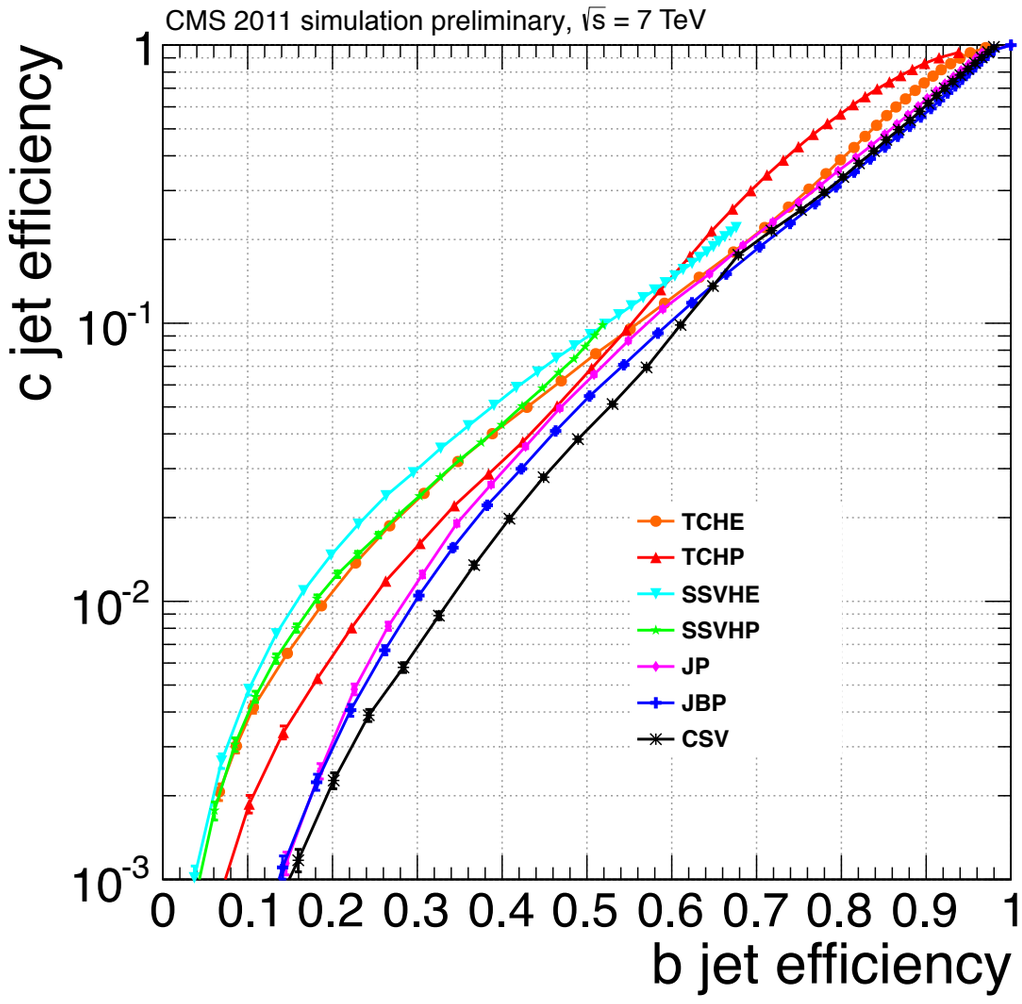


CMS PAS BTV-09-001

Combined Secondary Vertex

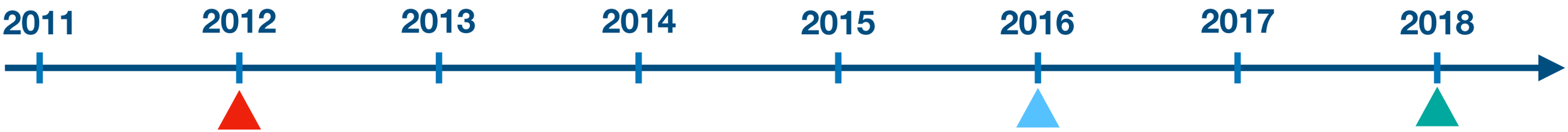






CMS PAS BTV-11-004

b jet energy regression



Jet Kinematics
(p_T, η,
hadronic fraction…)

Flavour content
(soft leptons, SV)

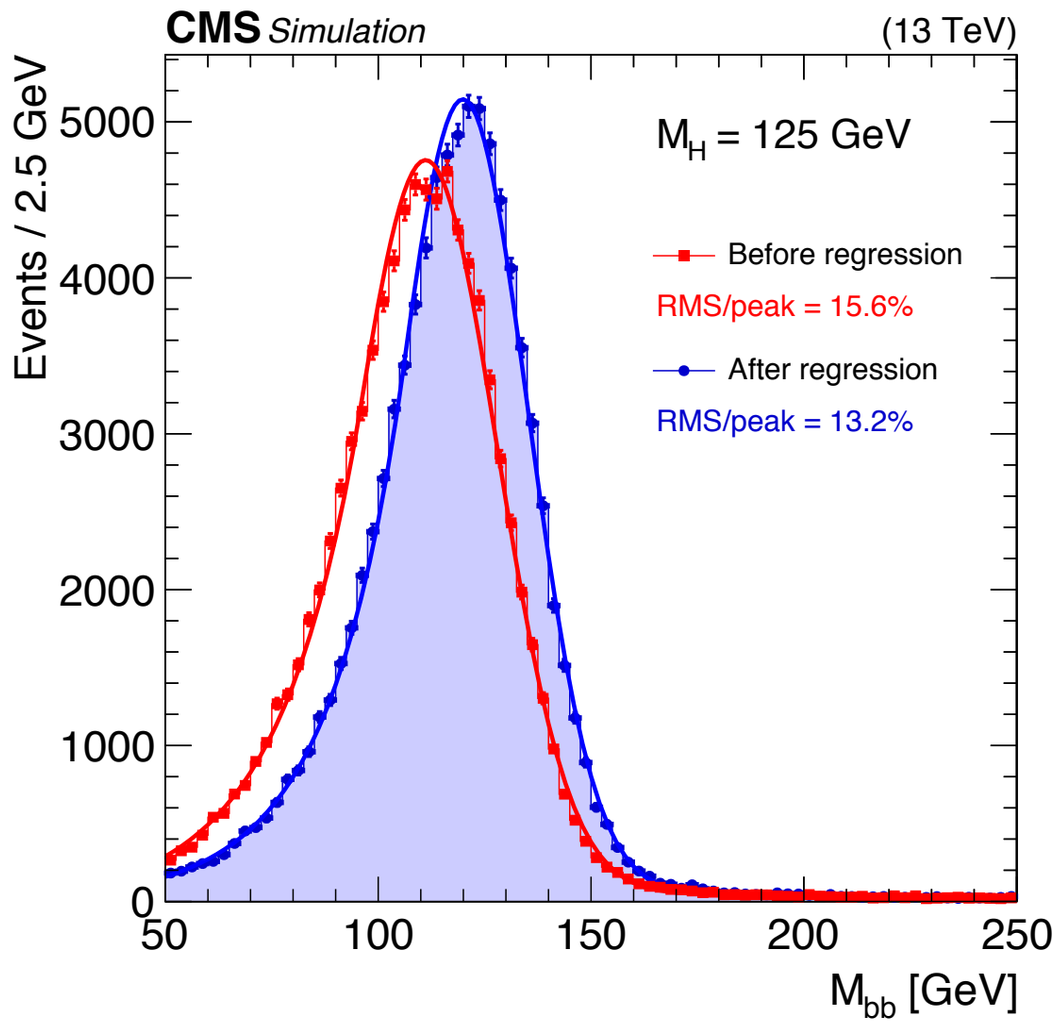
BDT

Textbook application of a BDT

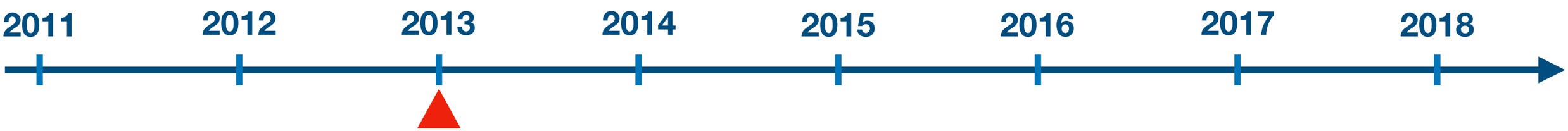
H→bb discovery paper uses a more advanced deep-learning approach.

Public results coming soon!

CMS-PAS-HIG-16-044

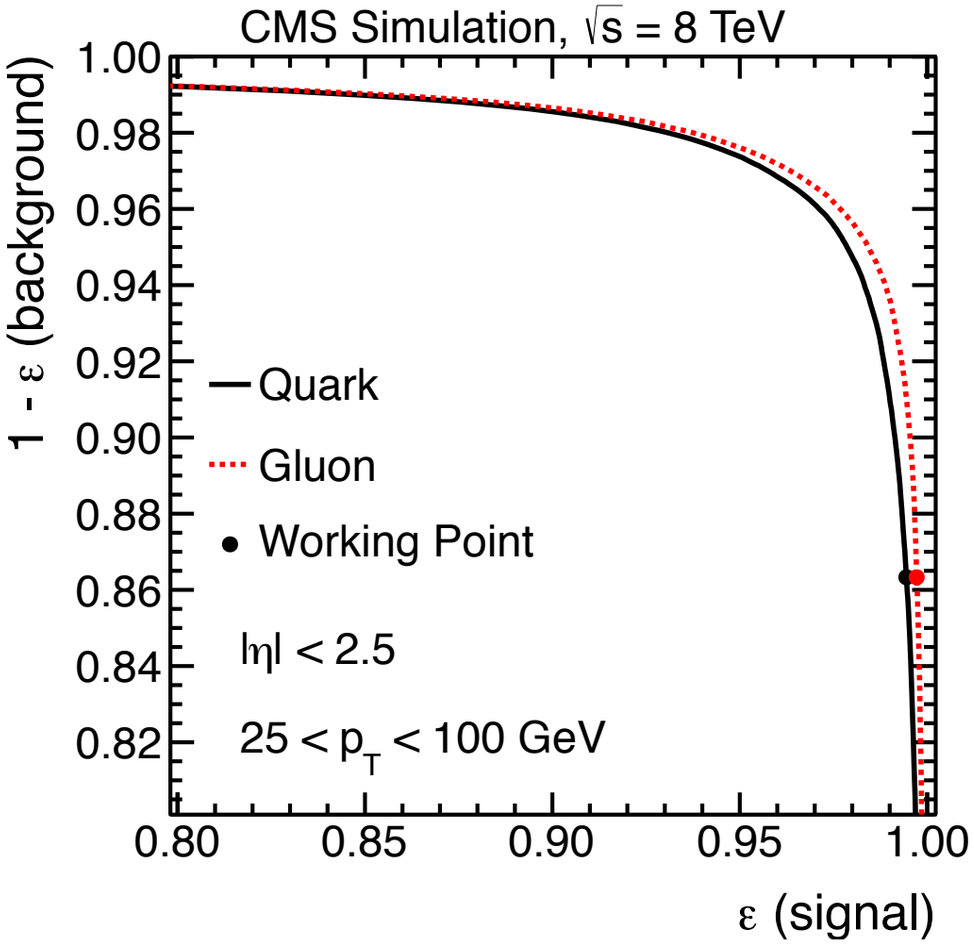


Pile Up Jet ID

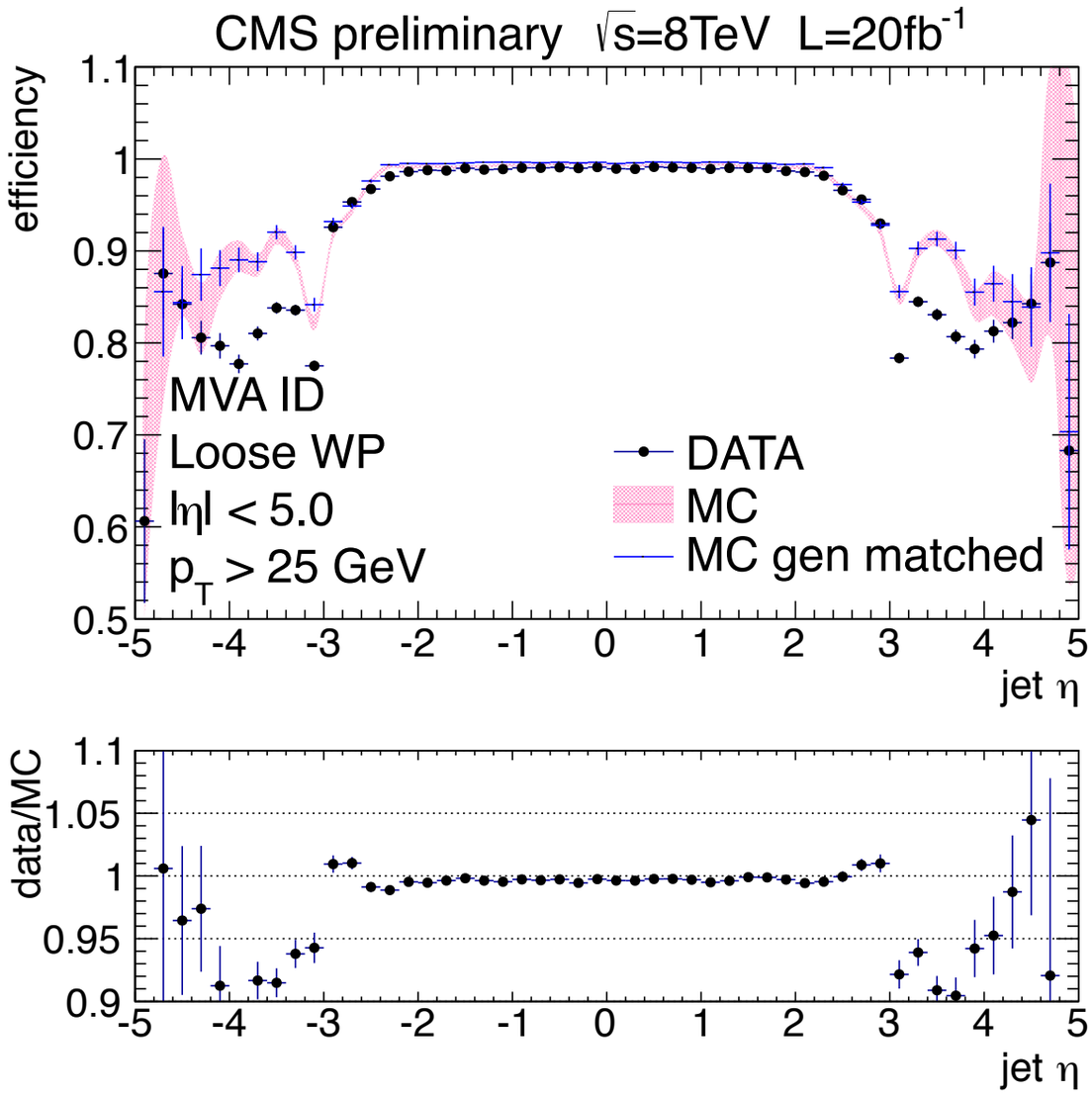


A BDT to reject pile-up jets from jet shape and track information.

Training and validation in Z→μμ in separate η regions.



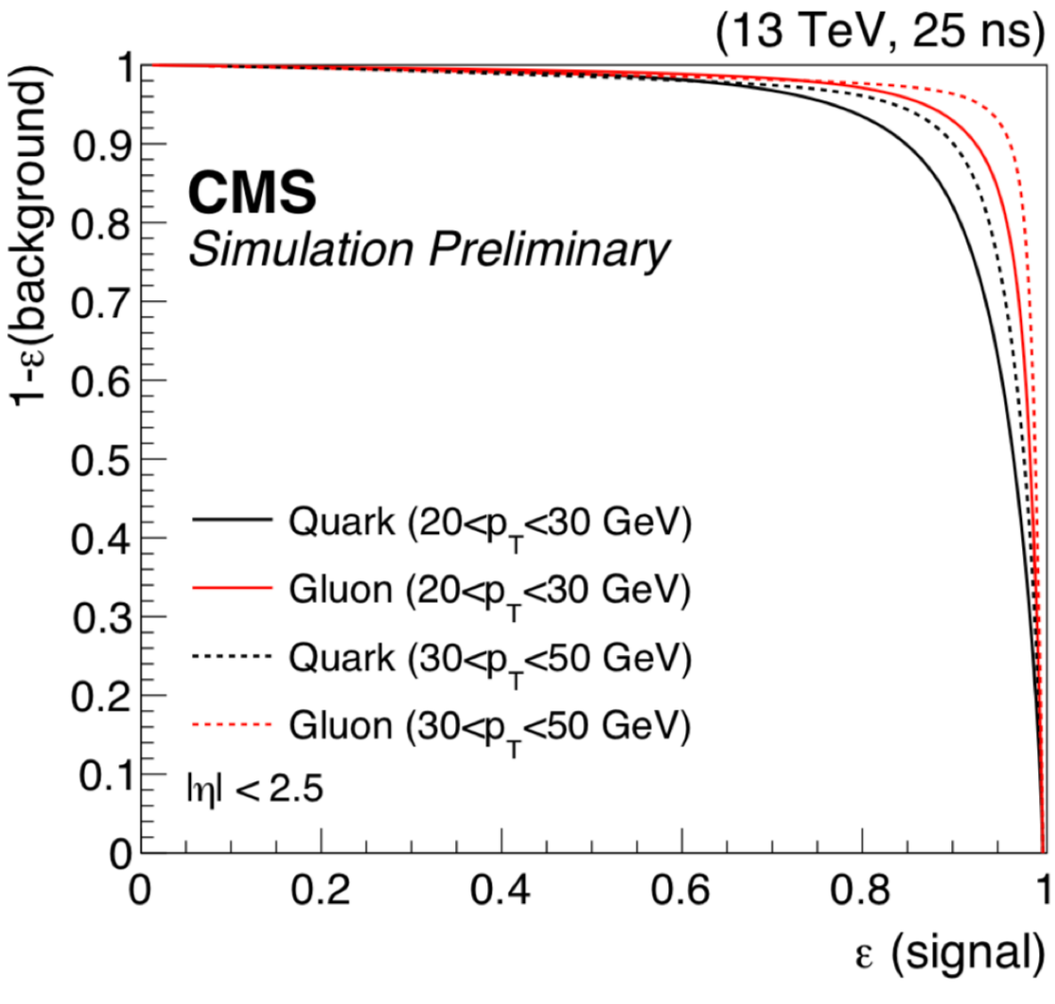
JME-13-005



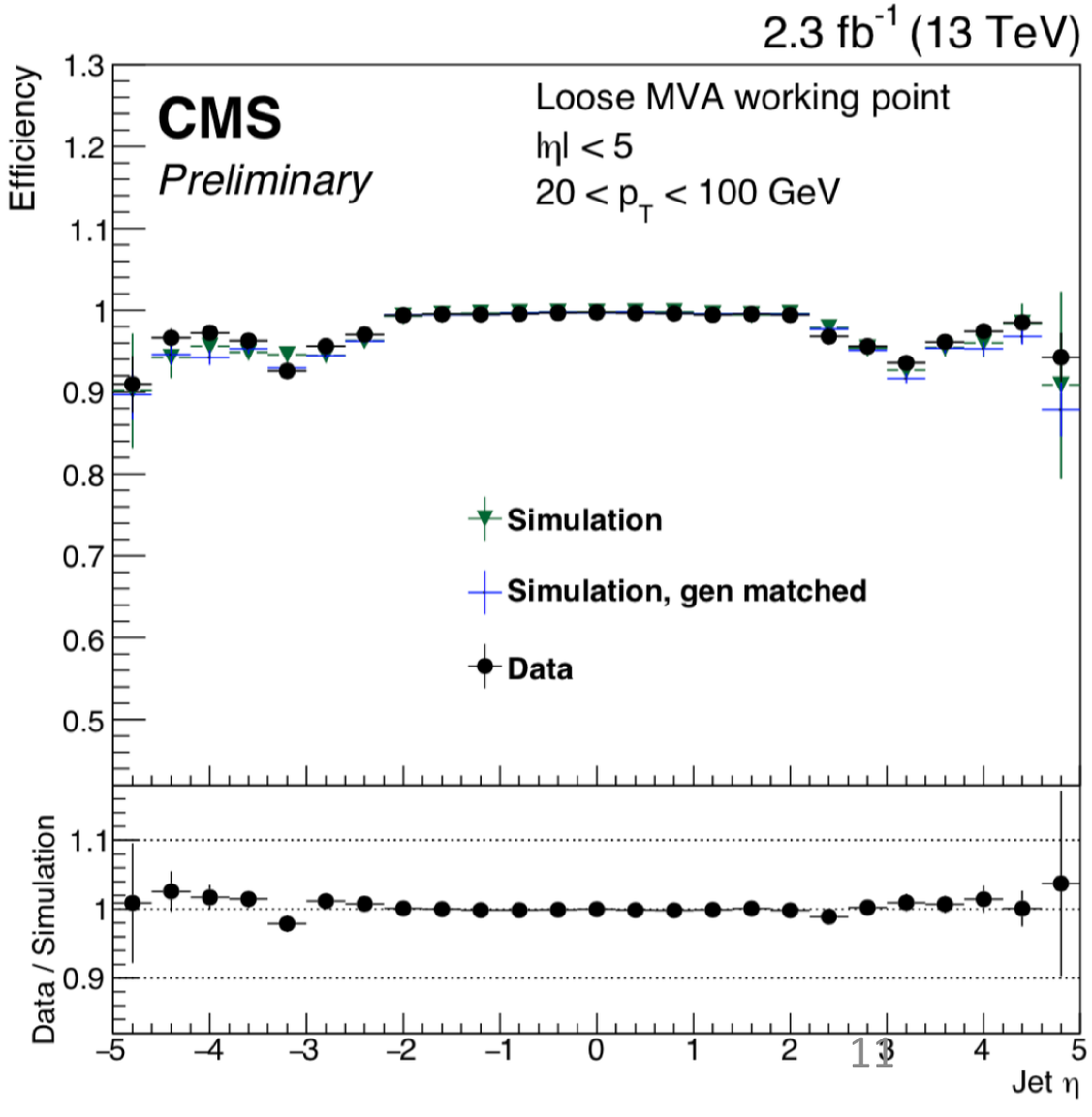
Pile Up Jet ID

A BDT to reject pile-up jets from jet shape and track information.

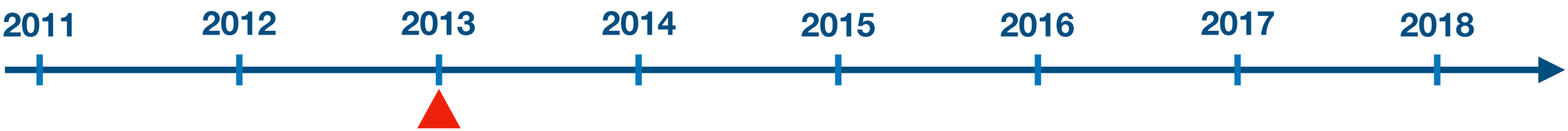
Training and validation in Z→μμ in separate η regions.

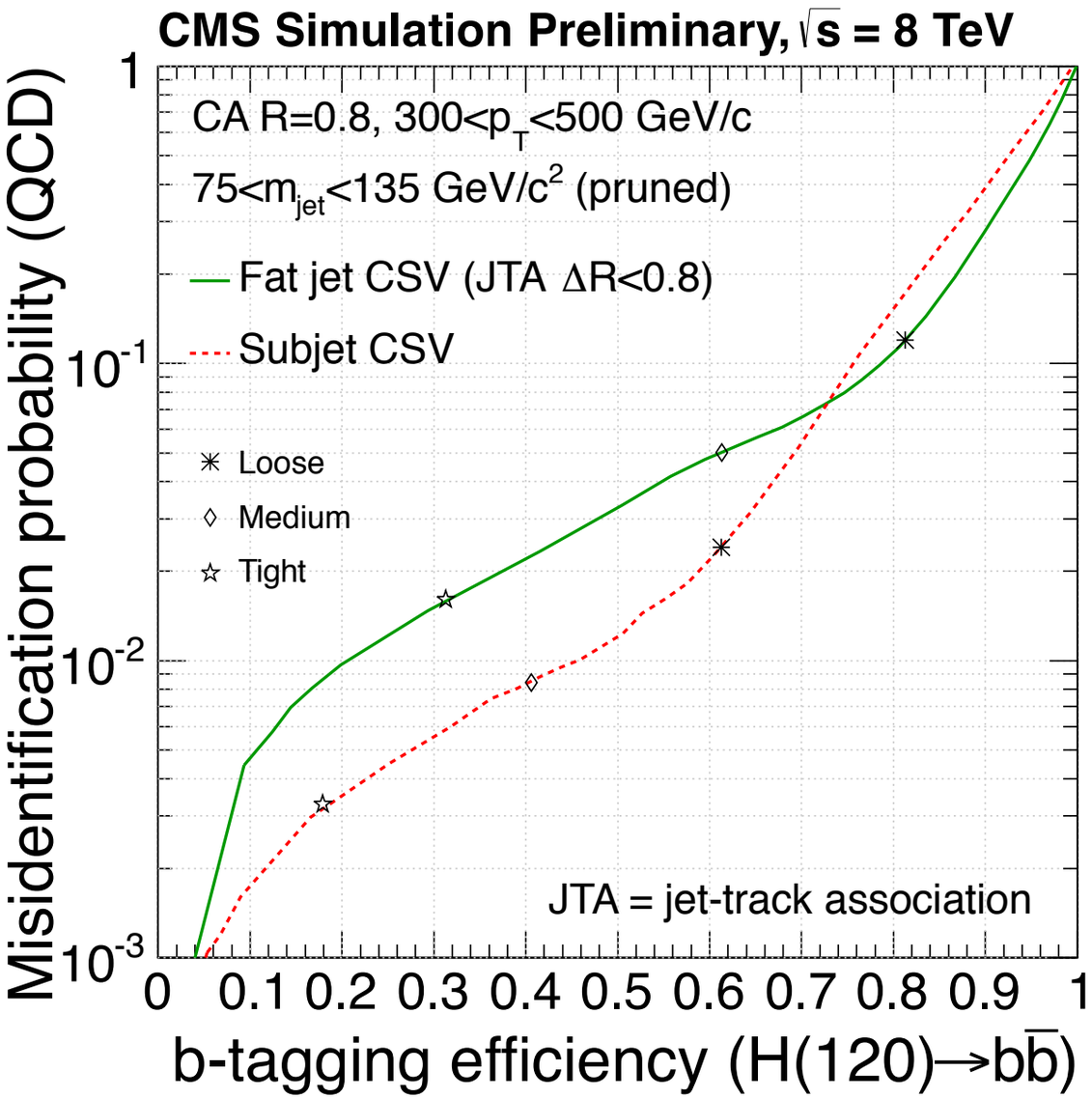


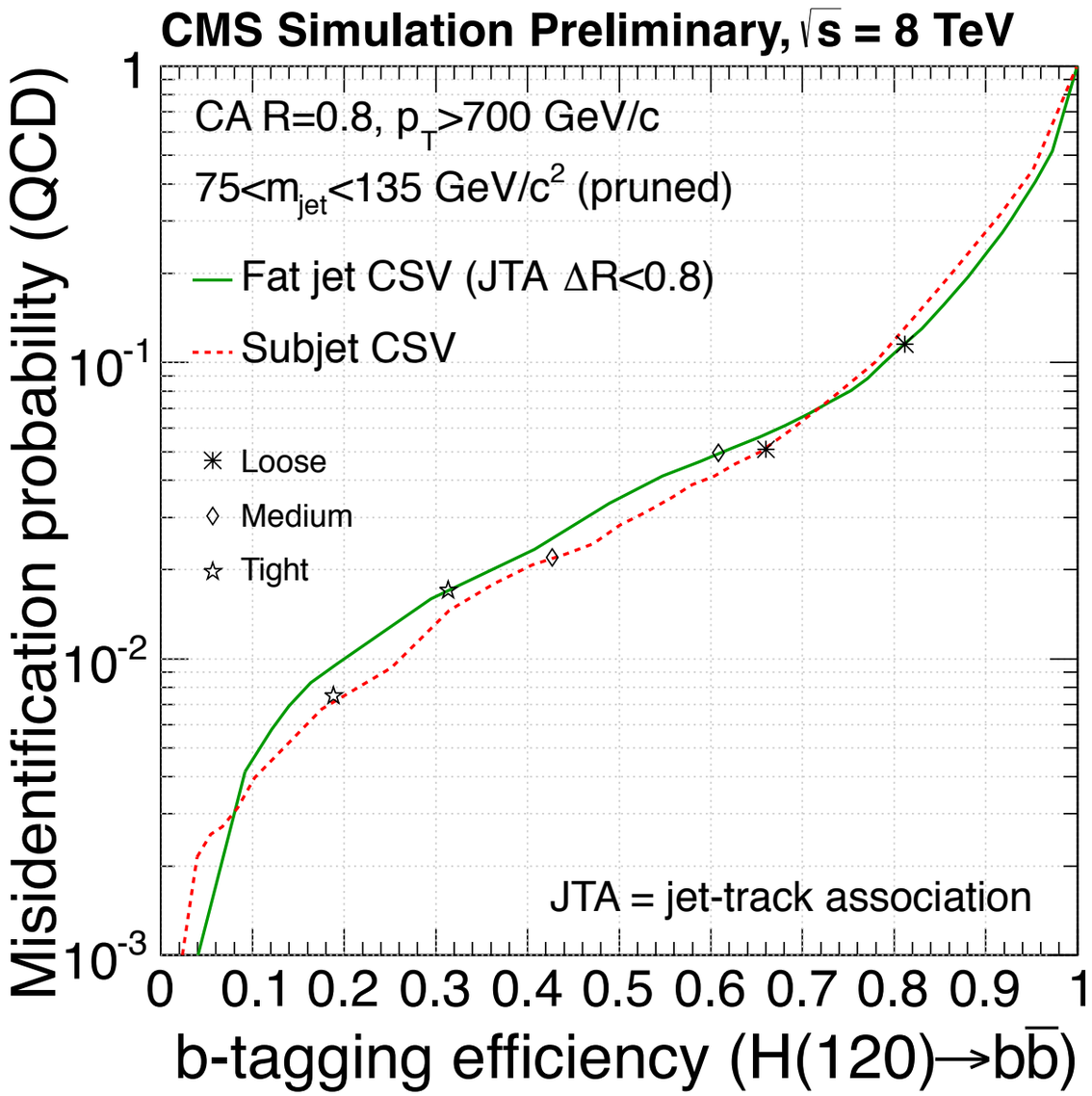
JME-16-003



Boosted HF tagging

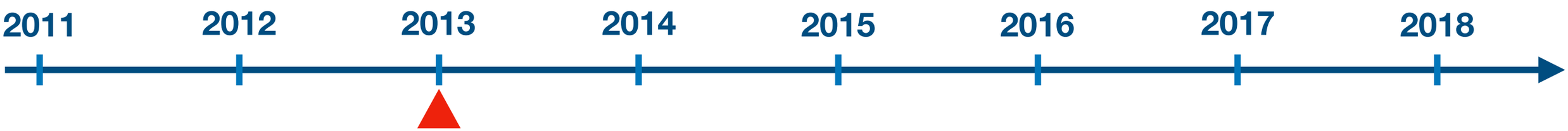






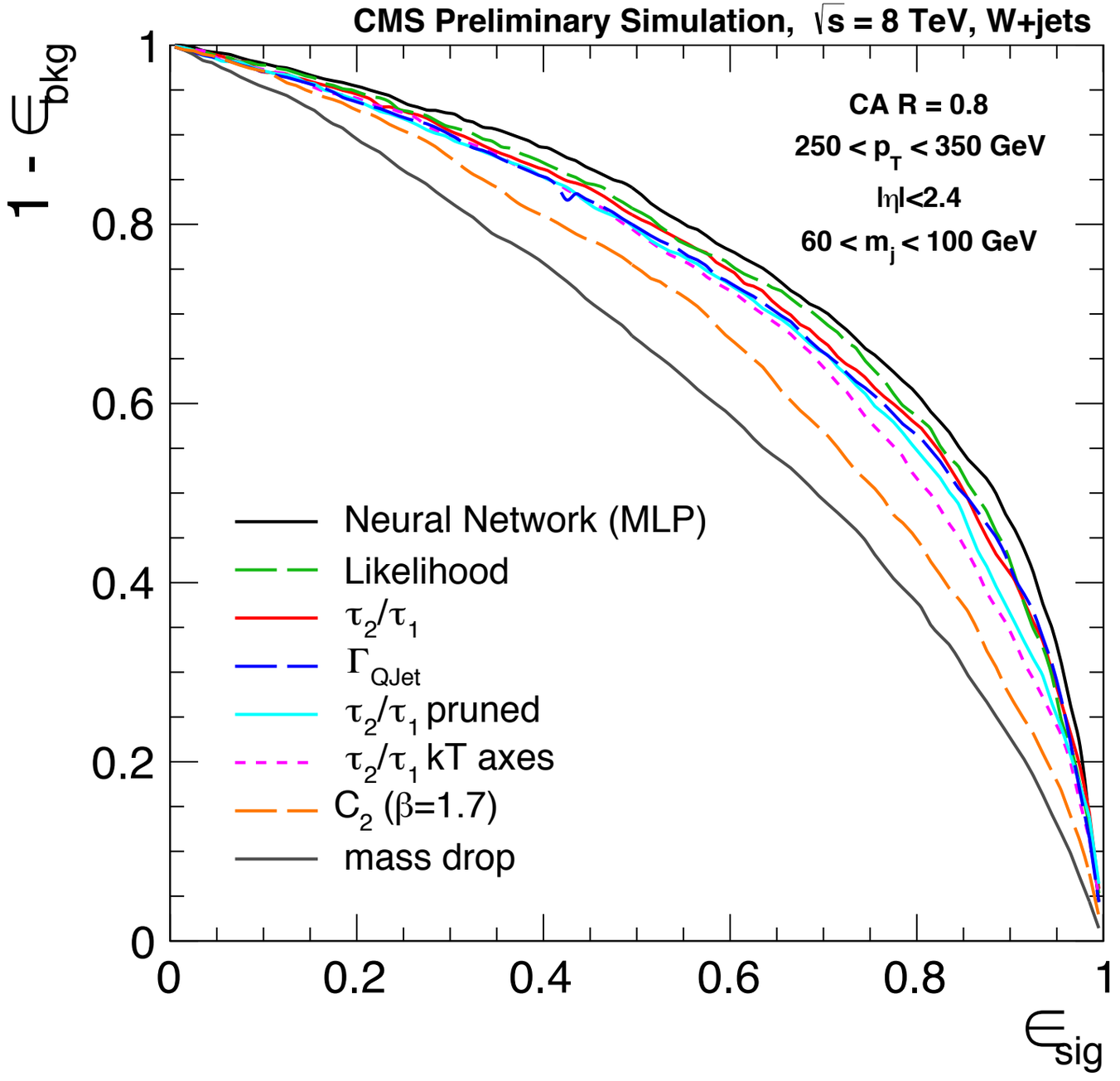
BTV-13-001

W tagging with MLP

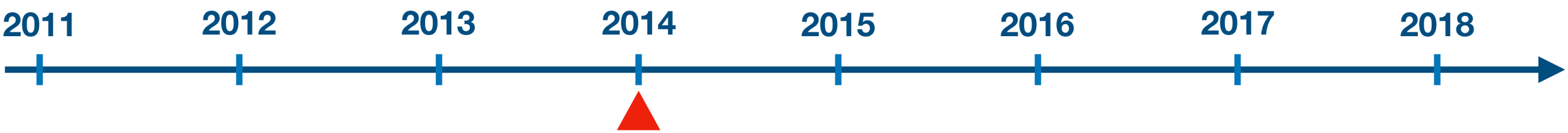


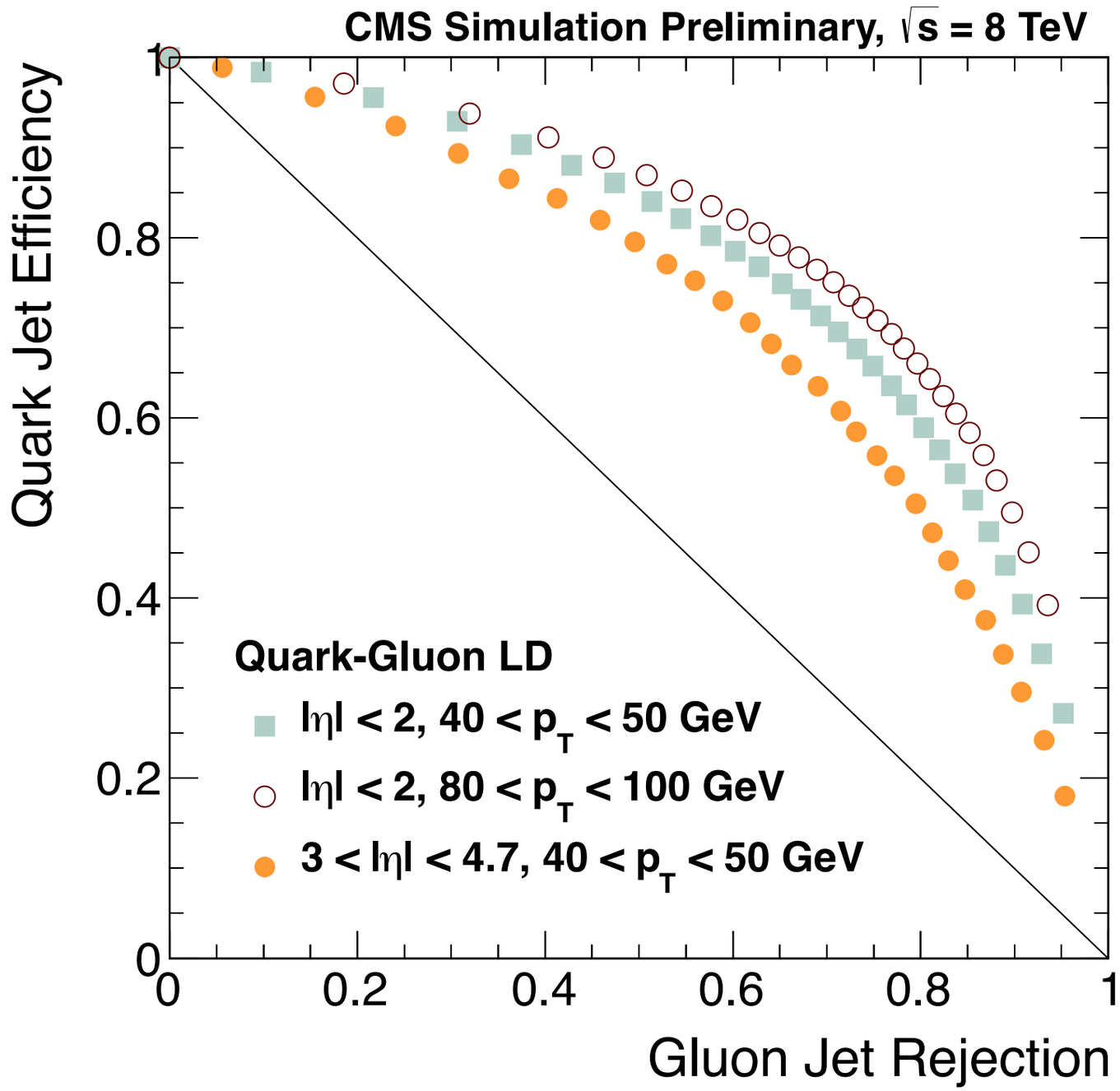
Several high-level features fed to a dense NN

JHEP 12 (2014) 017



Quark / gluon classification



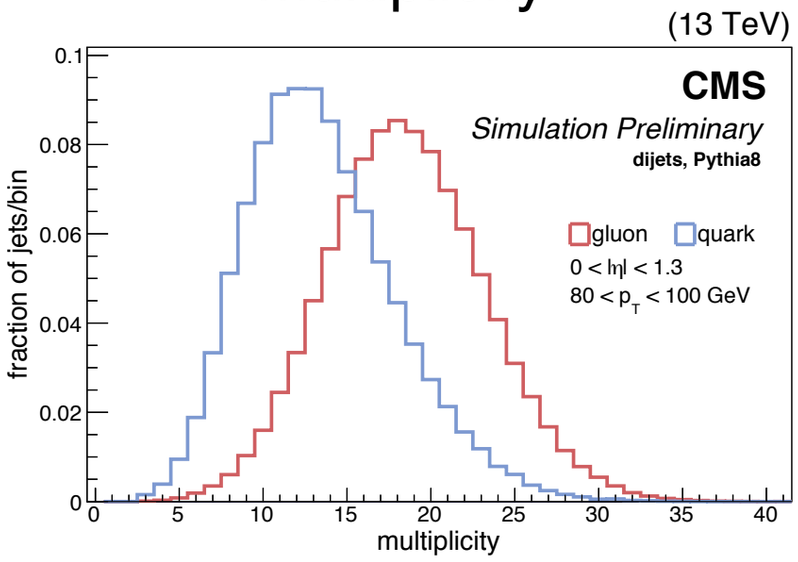


CMS PAS JME-13-002

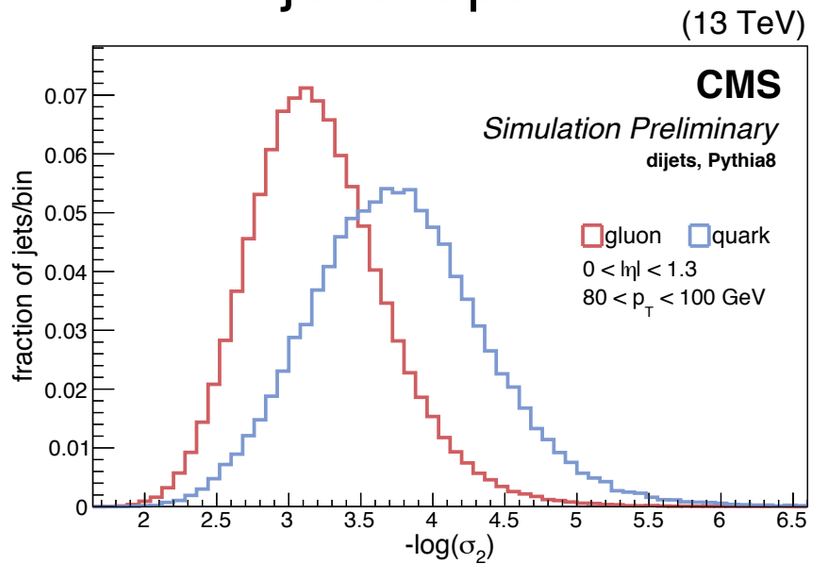
Quark / gluon classification

2011 2012 2013 2014 2015 2016 2017 2018

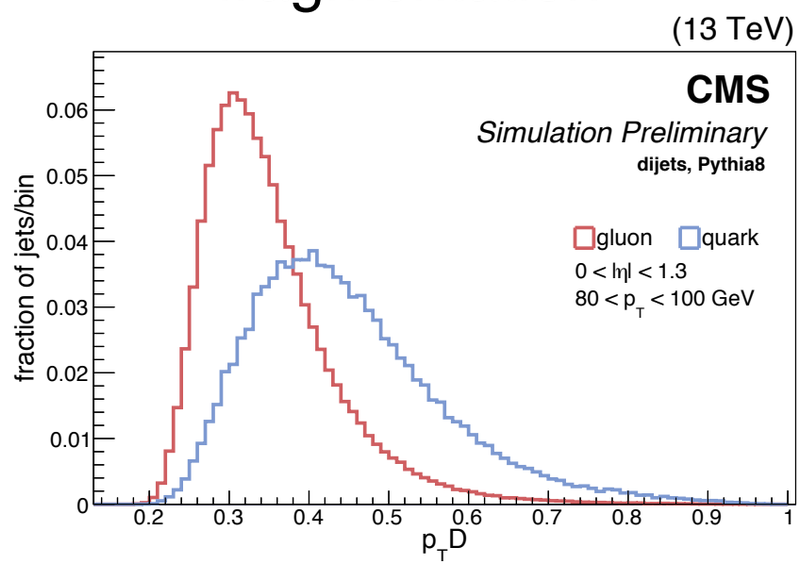
multiplicity

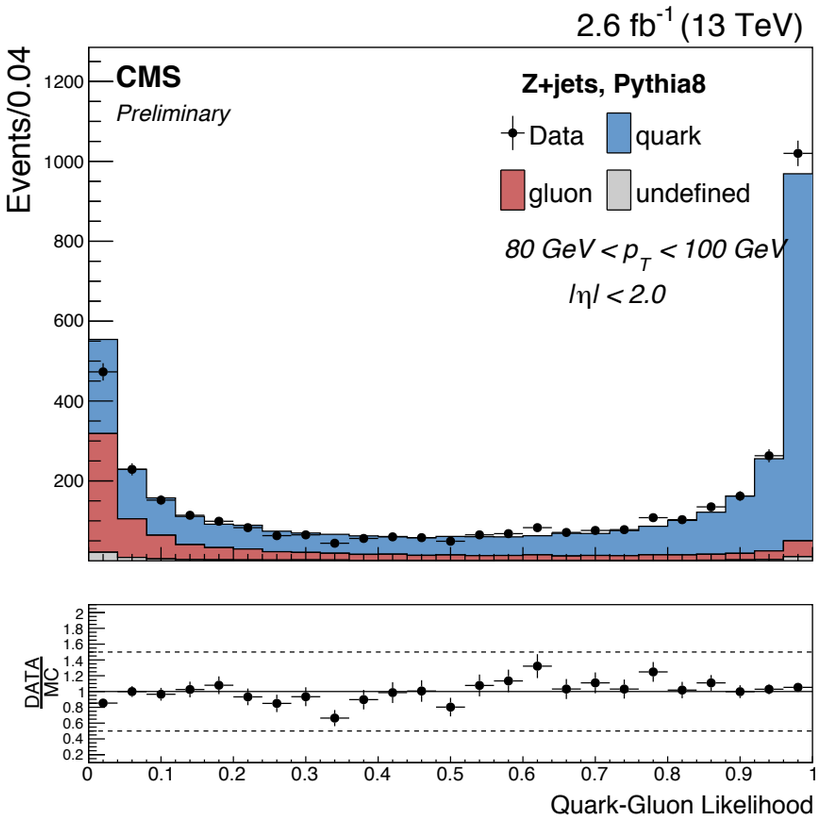


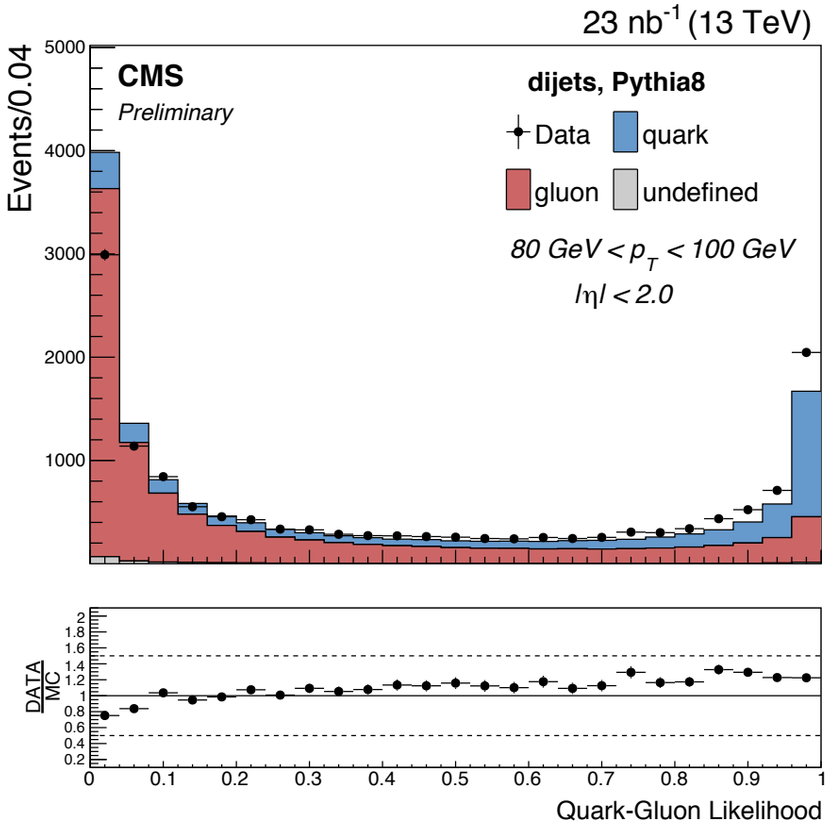
jet shape



fragmentation

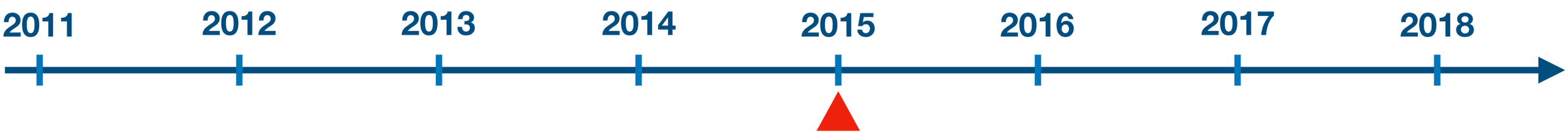


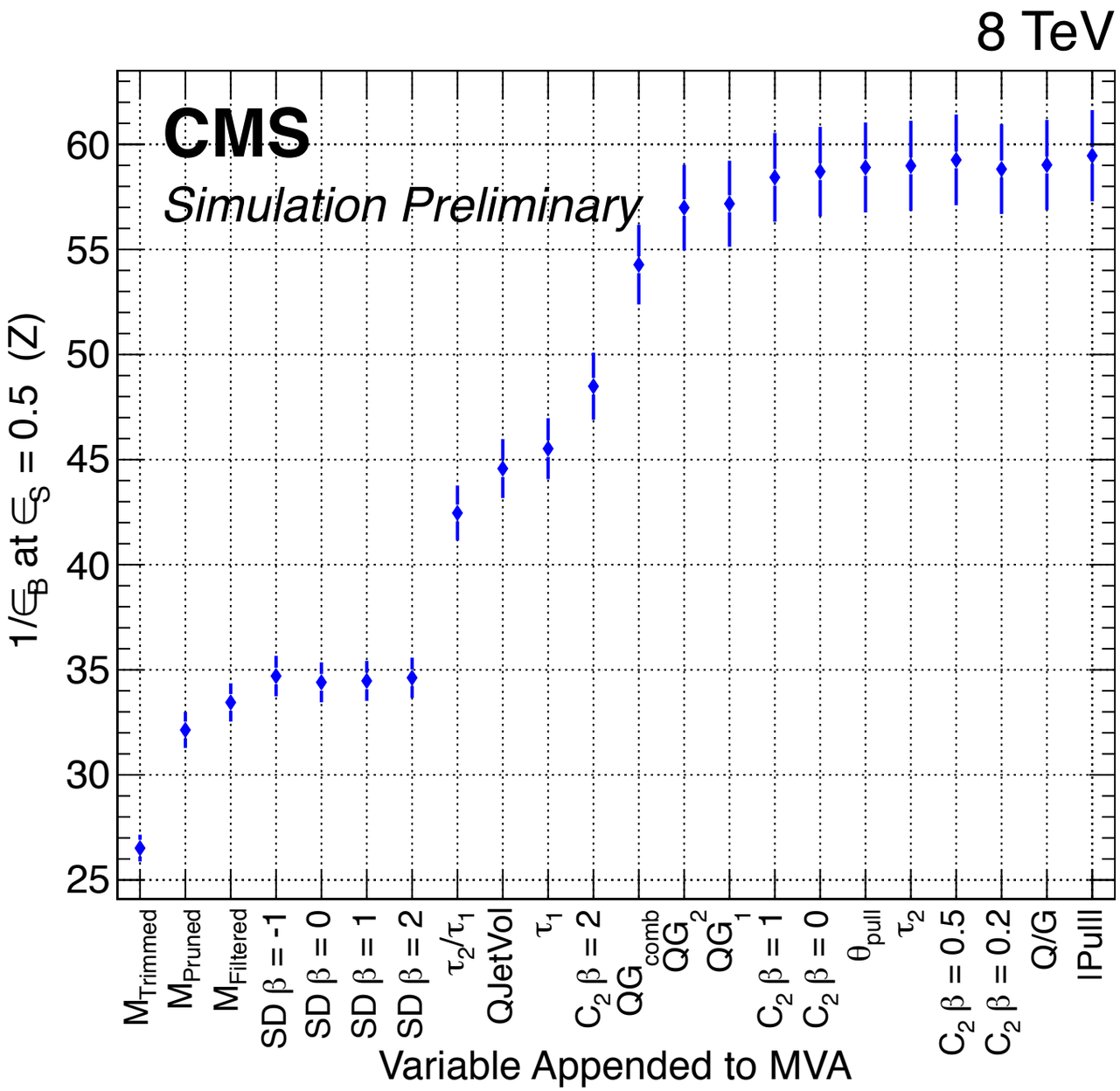




CMS-JME-16-003

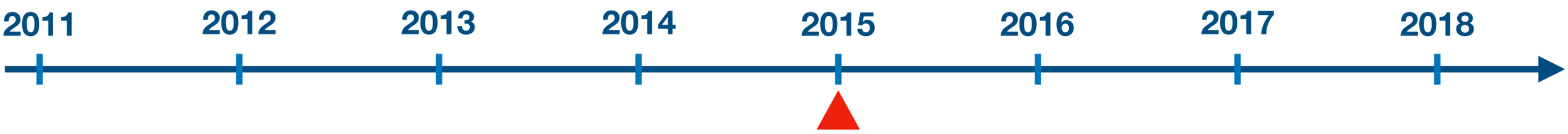
W-tagging with BDTs

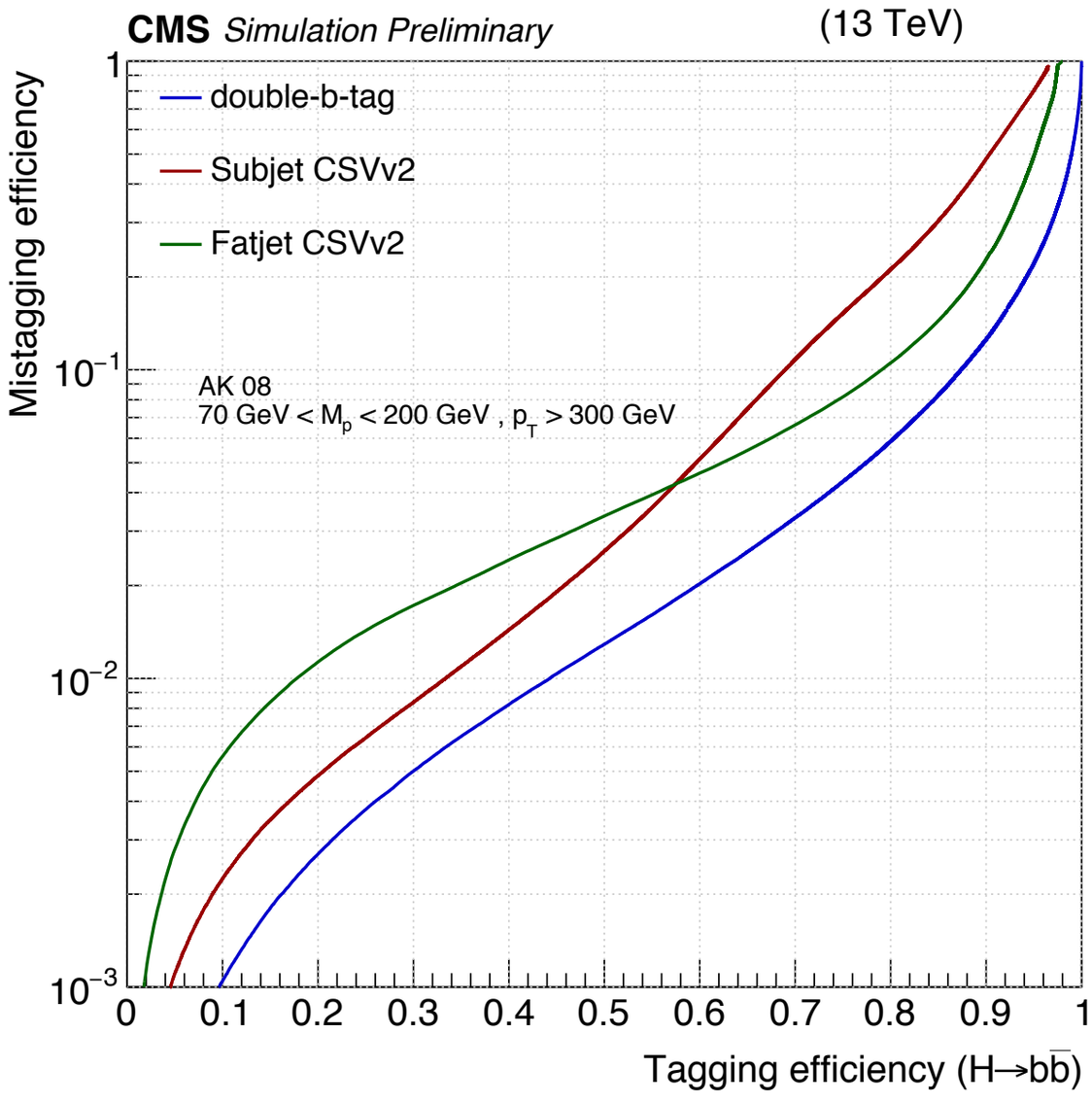


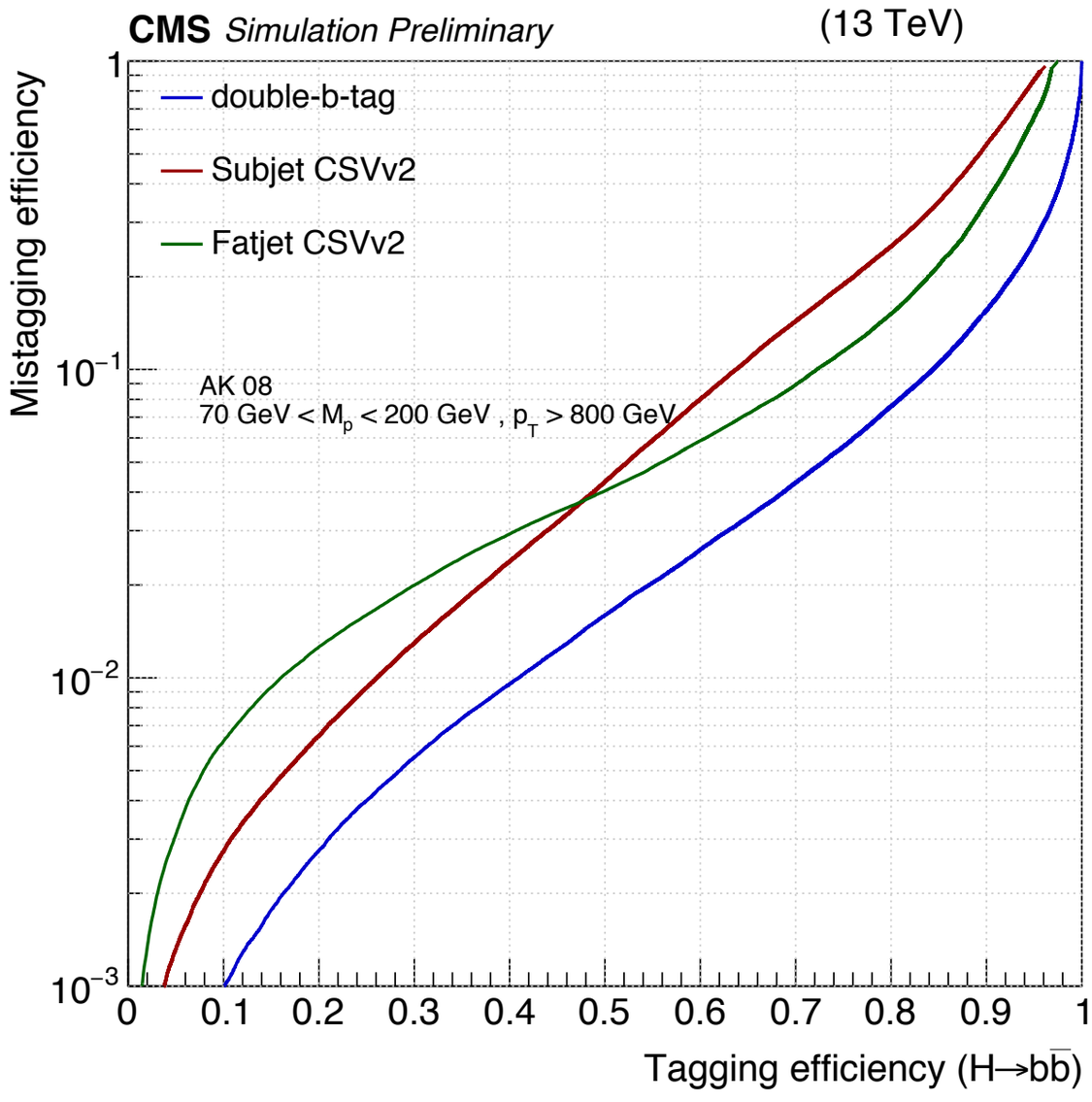


JME-14-002

DoubleB



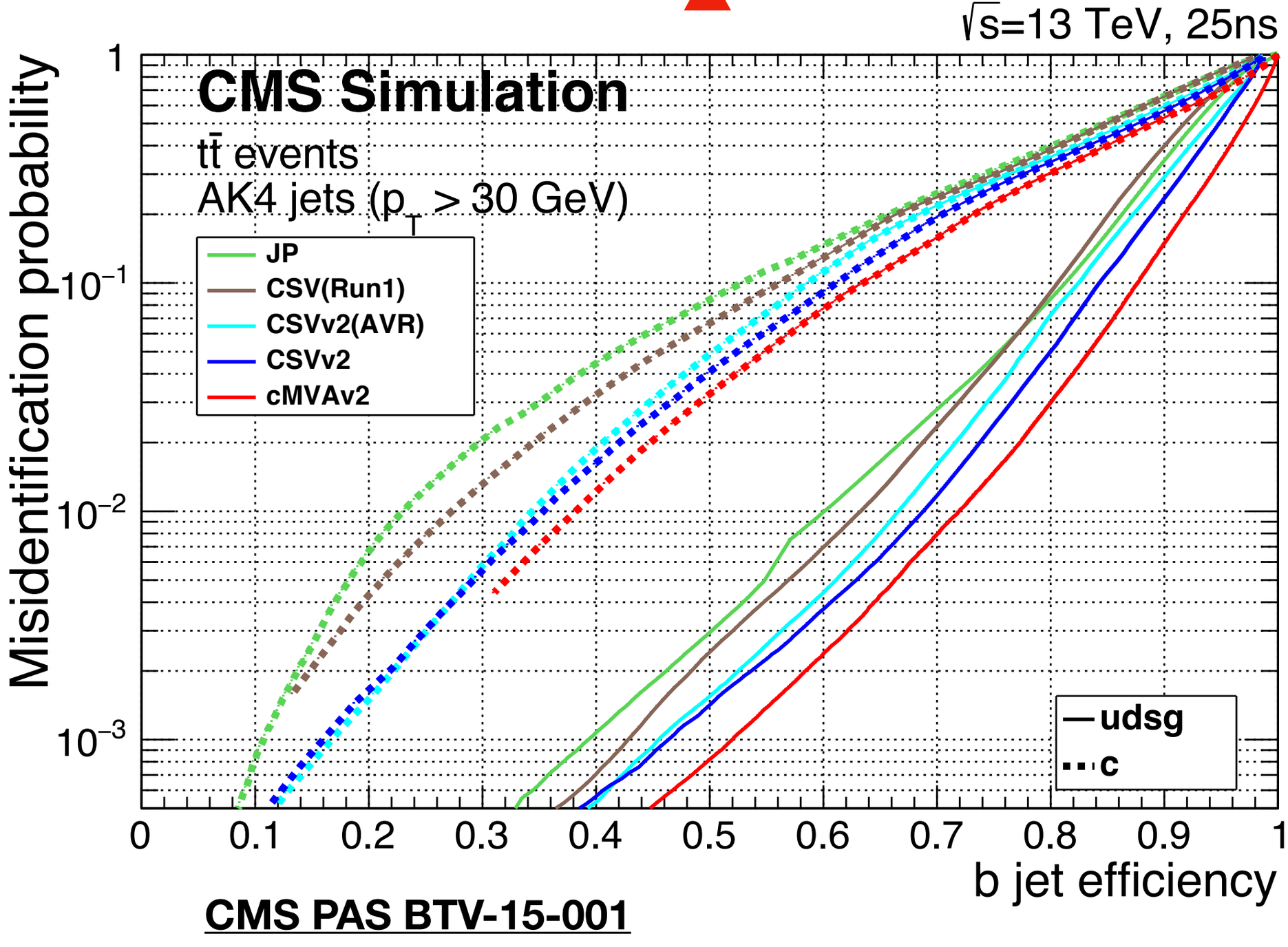




CMS DP-2015/038

cMVAv2 and CSVv2

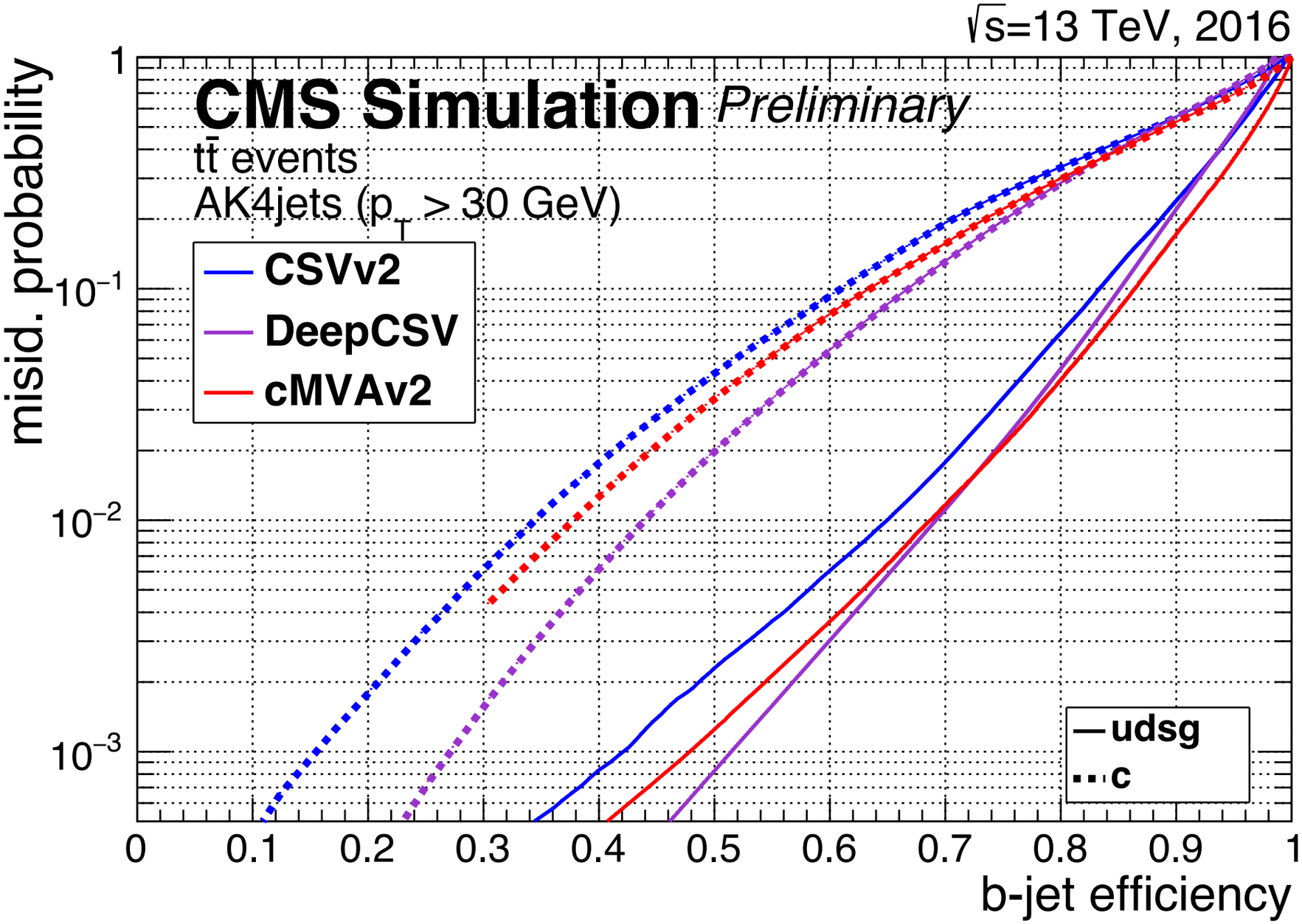
2011 2012 2013 2014 2015 2016 2017 2018



CMS PAS BTV-15-001

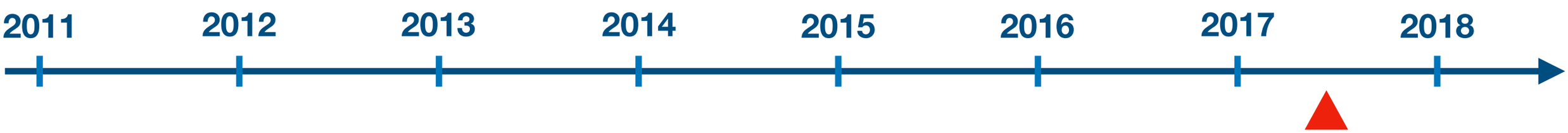
DeepCSV

2011 2012 2013 2014 2015 2016 2017 2018



CMS DP-2017/005

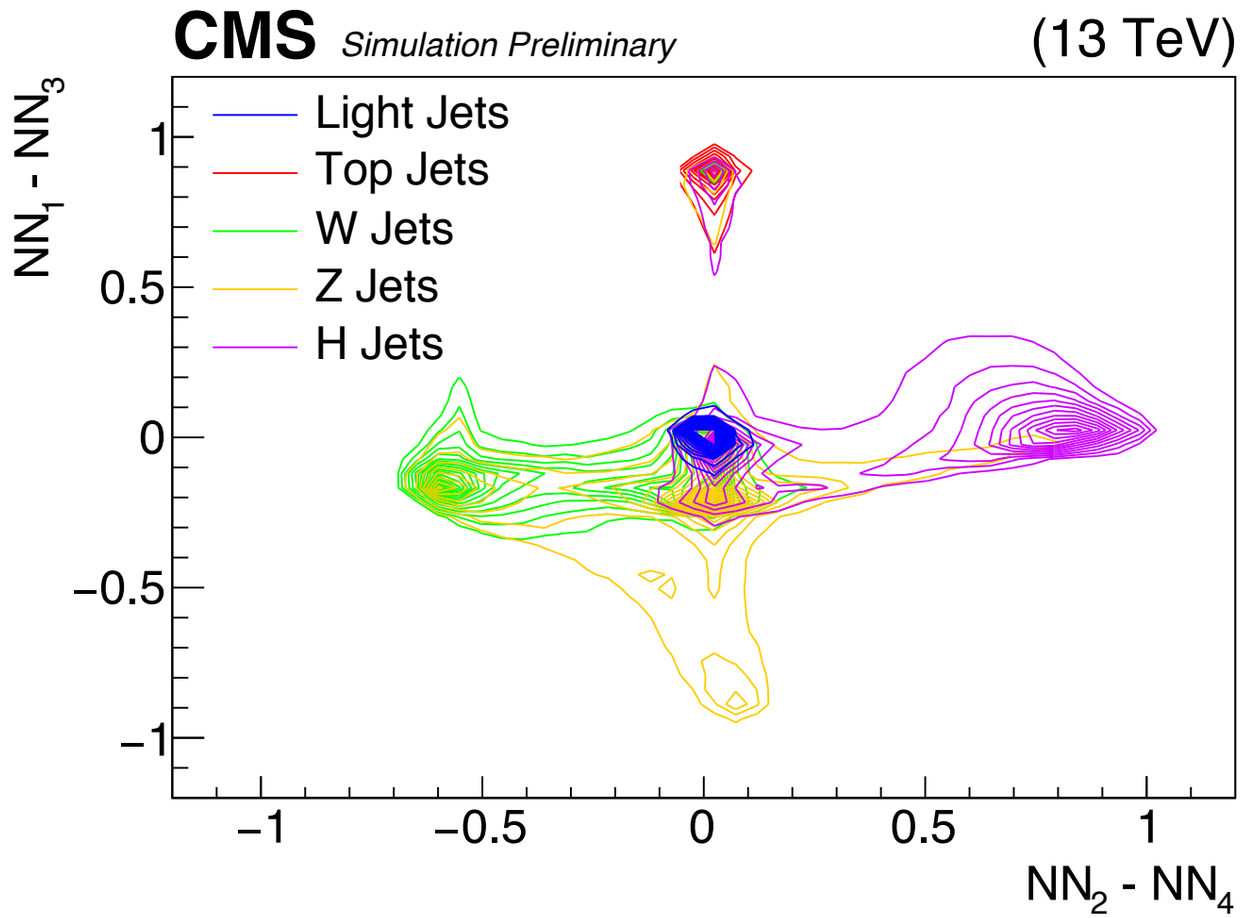
Boosted Event Shape Tagger



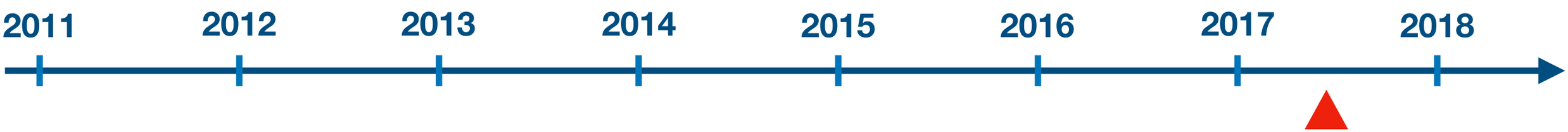
CMS-DP-2017-027

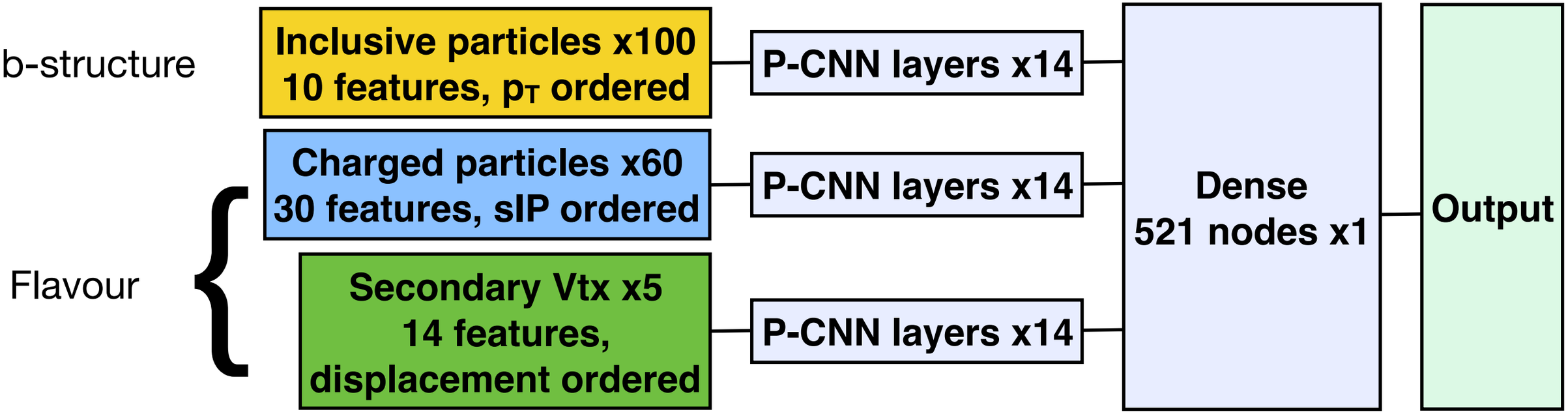
Classifies the boosted jet by providing jet shape features based on different particle hypotheses

Dense deep NN, 41 inputs, 2 hidden layers, 5 outputs.

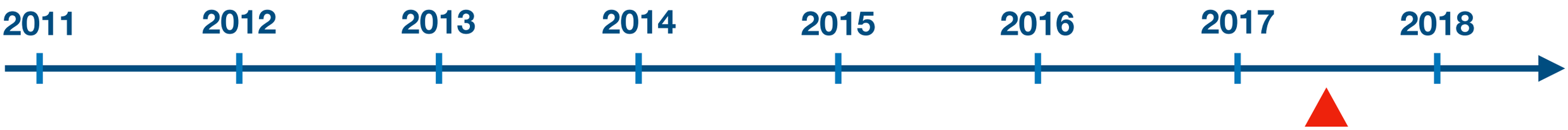


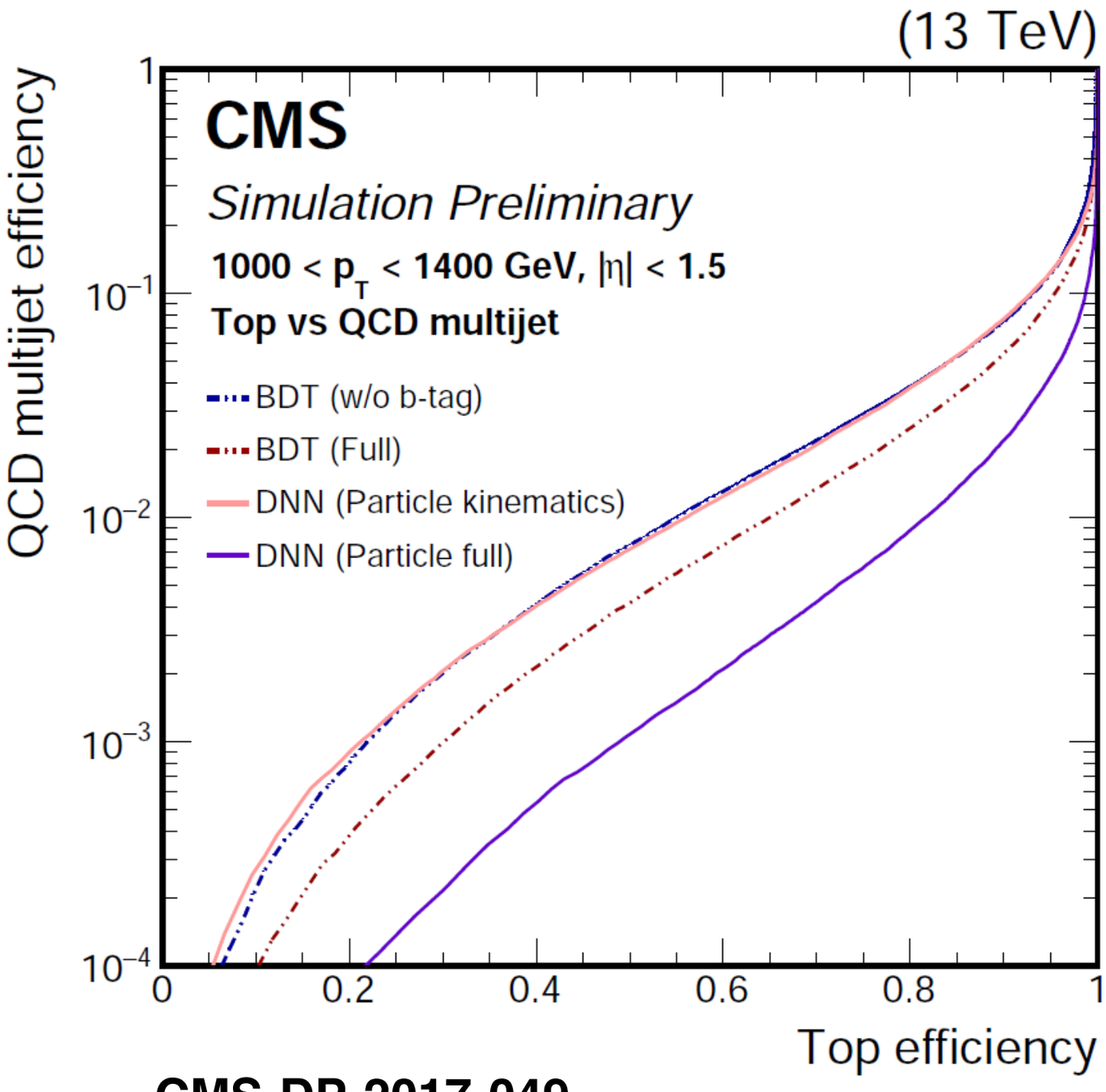
DeepAK8


2011
2012
2013
2014
2015
2016
2017
2018


Sub-structure
Flavour
Inclusive particles x100
10 features, p_T ordered
Charged particles x60
30 features, sIP ordered
Secondary Vtx x5
14 features,
displacement ordered
P-CNN layers x14
P-CNN layers x14
P-CNN layers x14
Dense
521 nodes x1
Output

DeepAK8



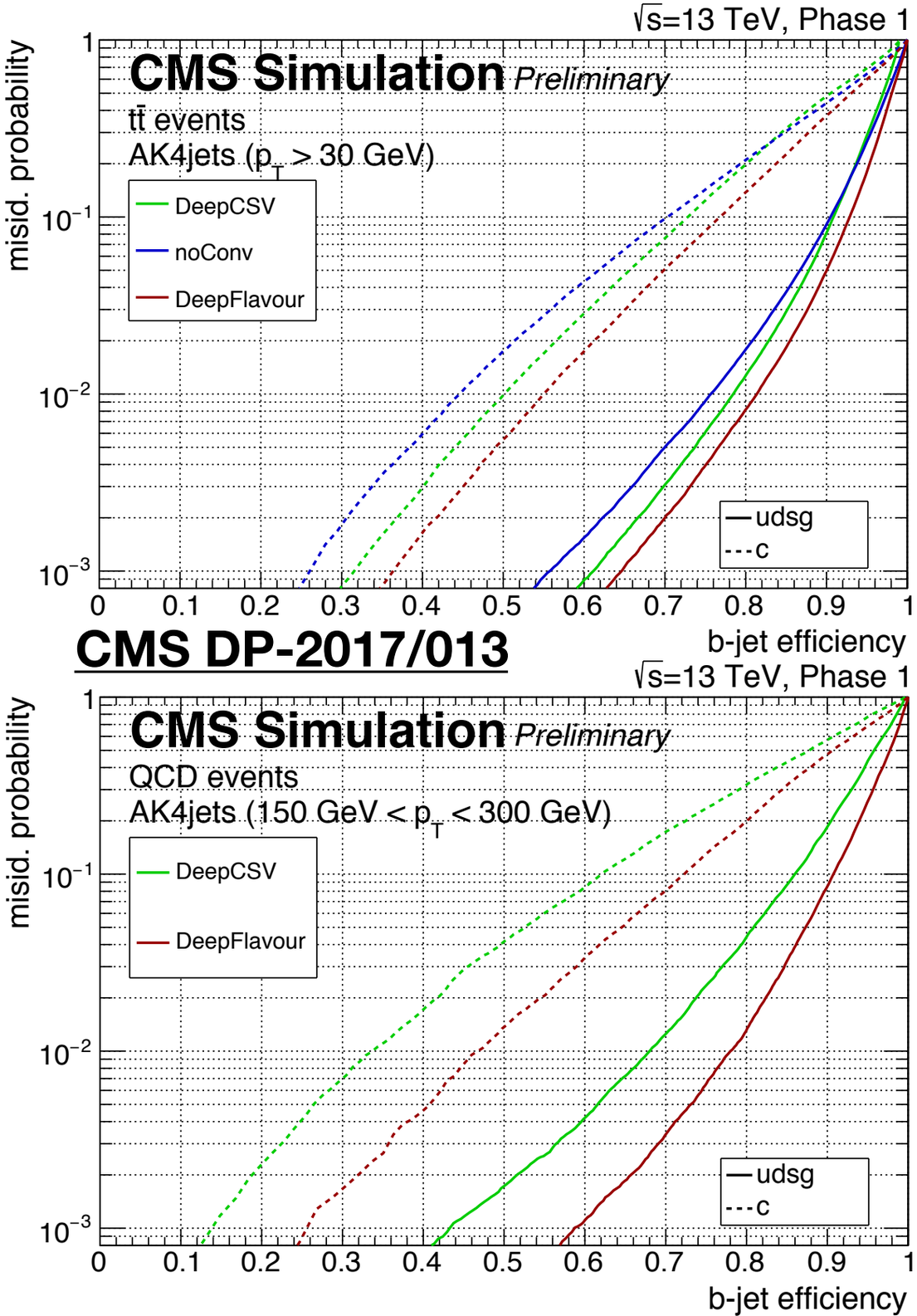


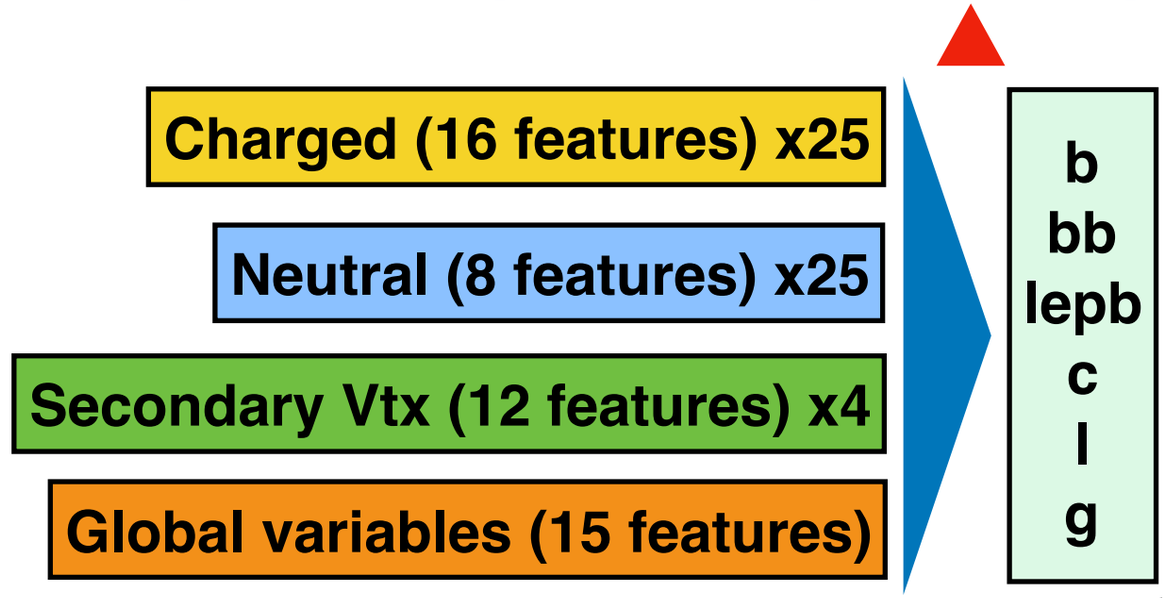
CMS-DP-2017-049

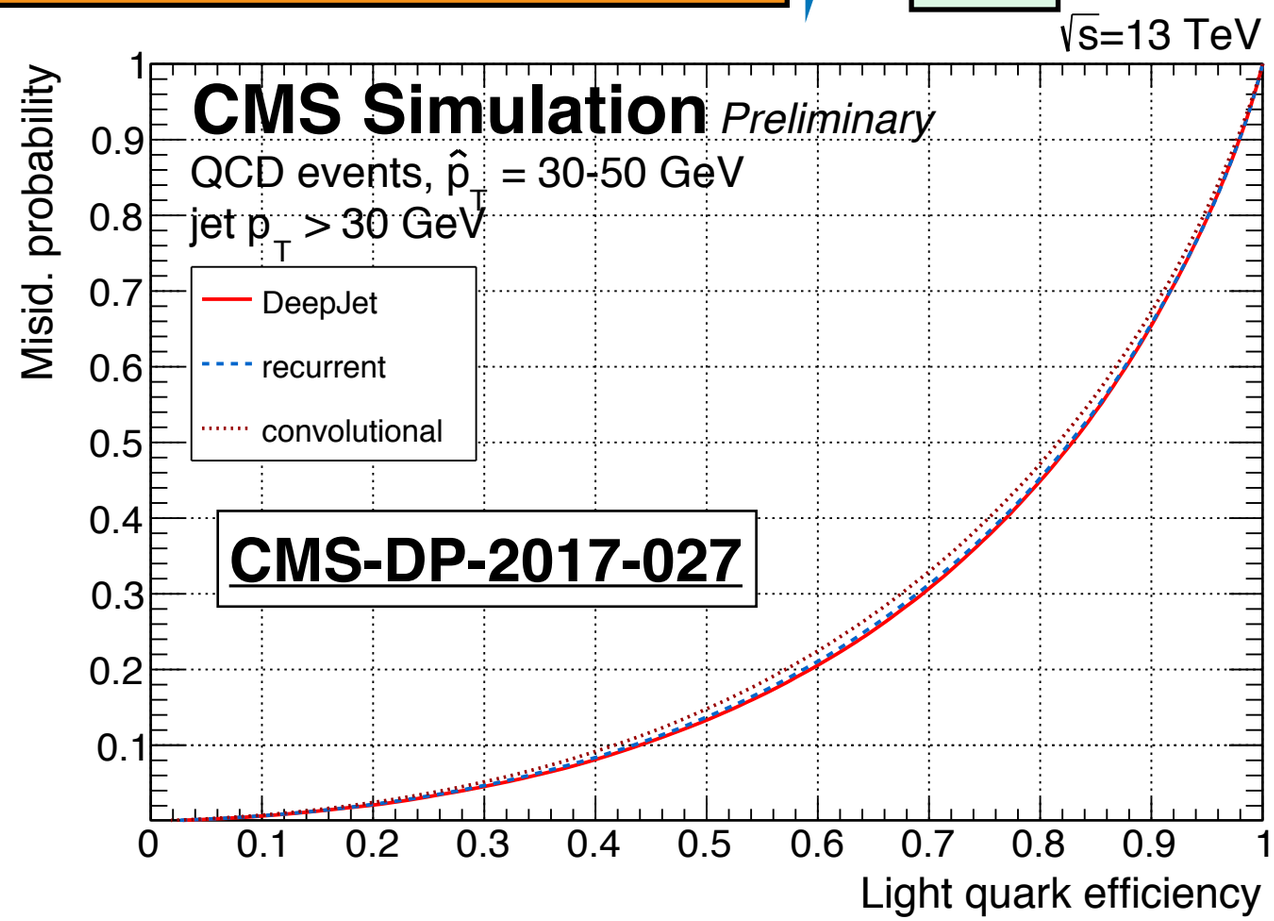
Category	Label
Higgs	H (bb)
	H (cc)
	H (VV*→qqqq)
Top	top (bcq)
	top (bqq)
	top (bc)
	top (bq)
W	W (cq)
	W (qq)
Z	Z (bb)
	Z (cc)
	Z (qq)
QCD	QCD (bb)
	QCD (cc)
	QCD (b)
	QCD (c)
	QCD (others)

DeepJet

2011 2012 2013 2014 2015 2016 2017 2018

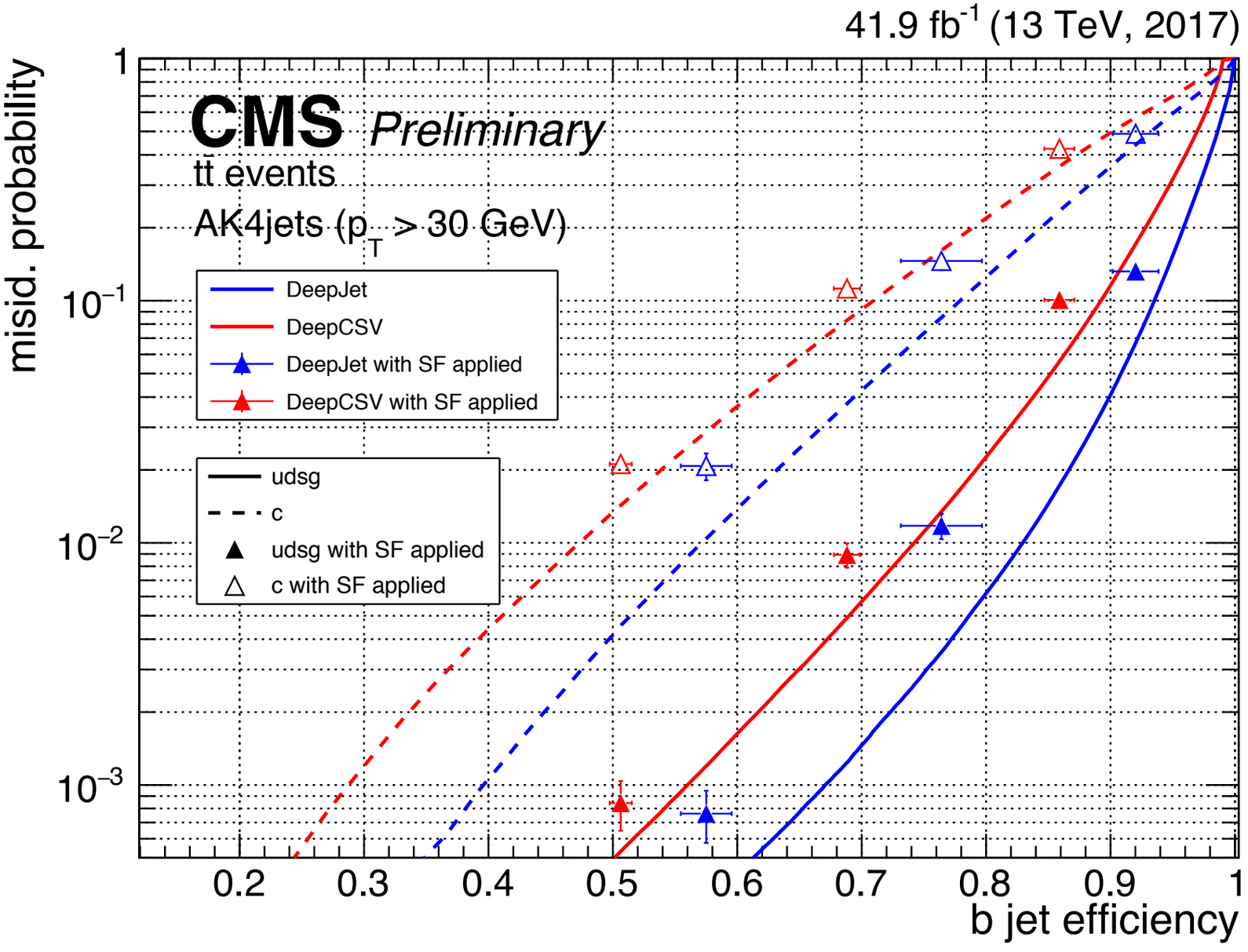






DeepJet

2011 2012 2013 2014 2015 2016 2017 2018

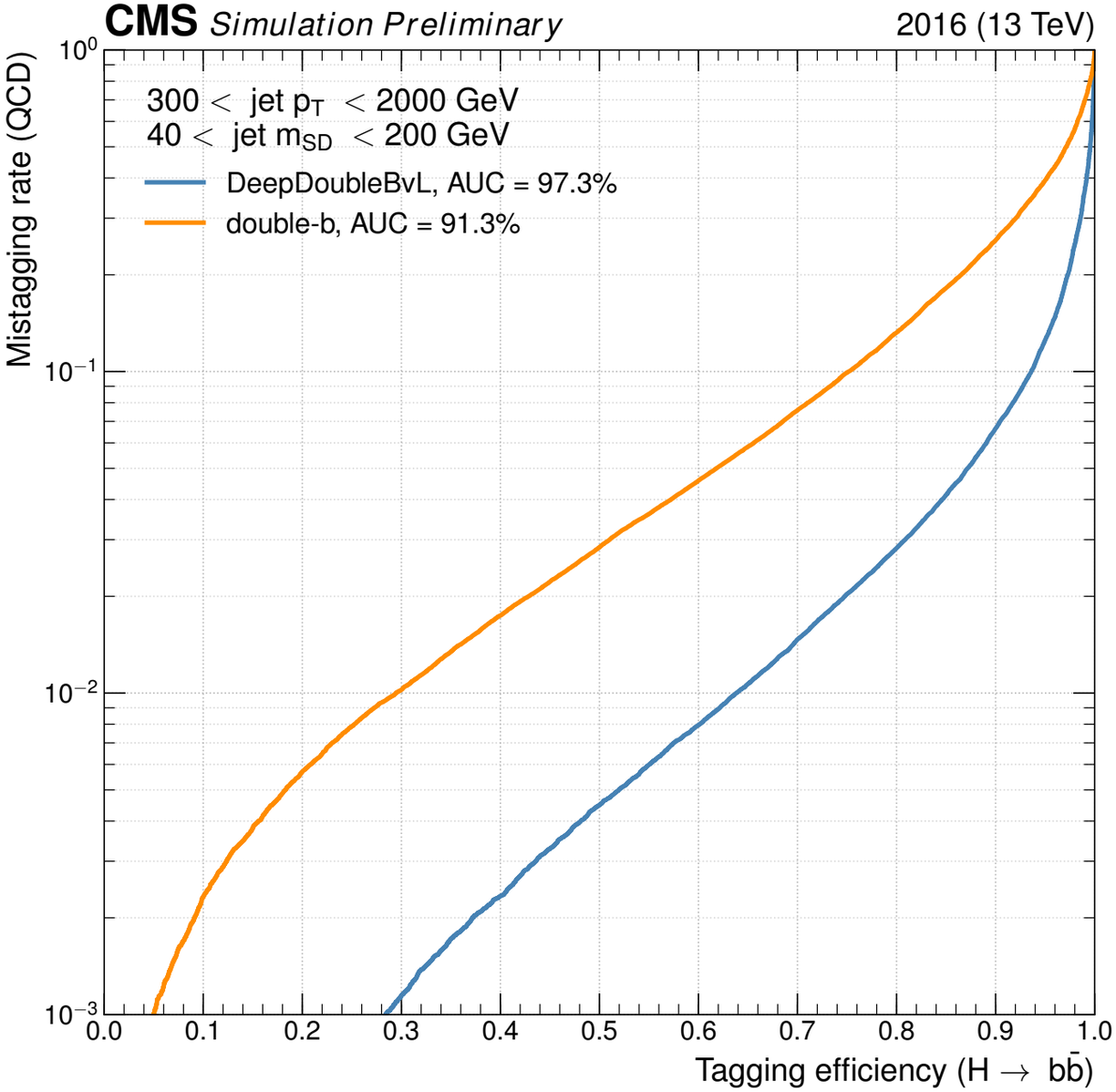


CMS DP-2018/058

More details in the Emil's presentations

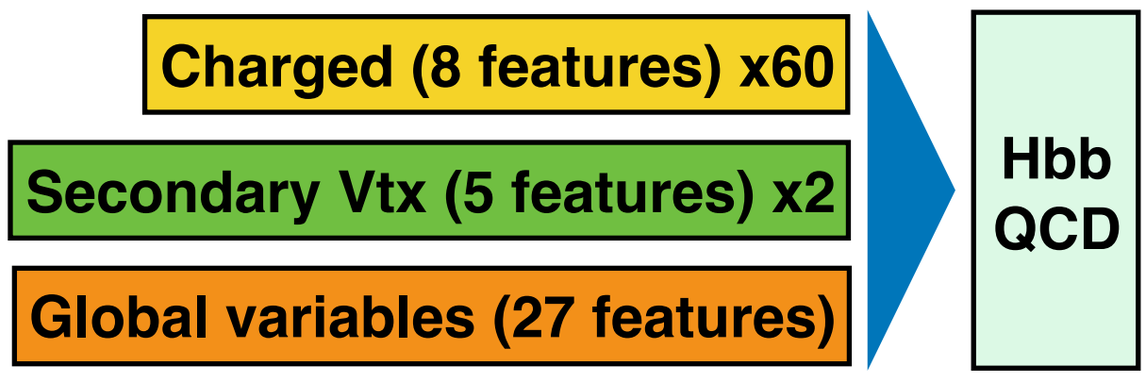
DeepDoubleB/C

2011 2012 2013 2014 2015 2016 2017 2018



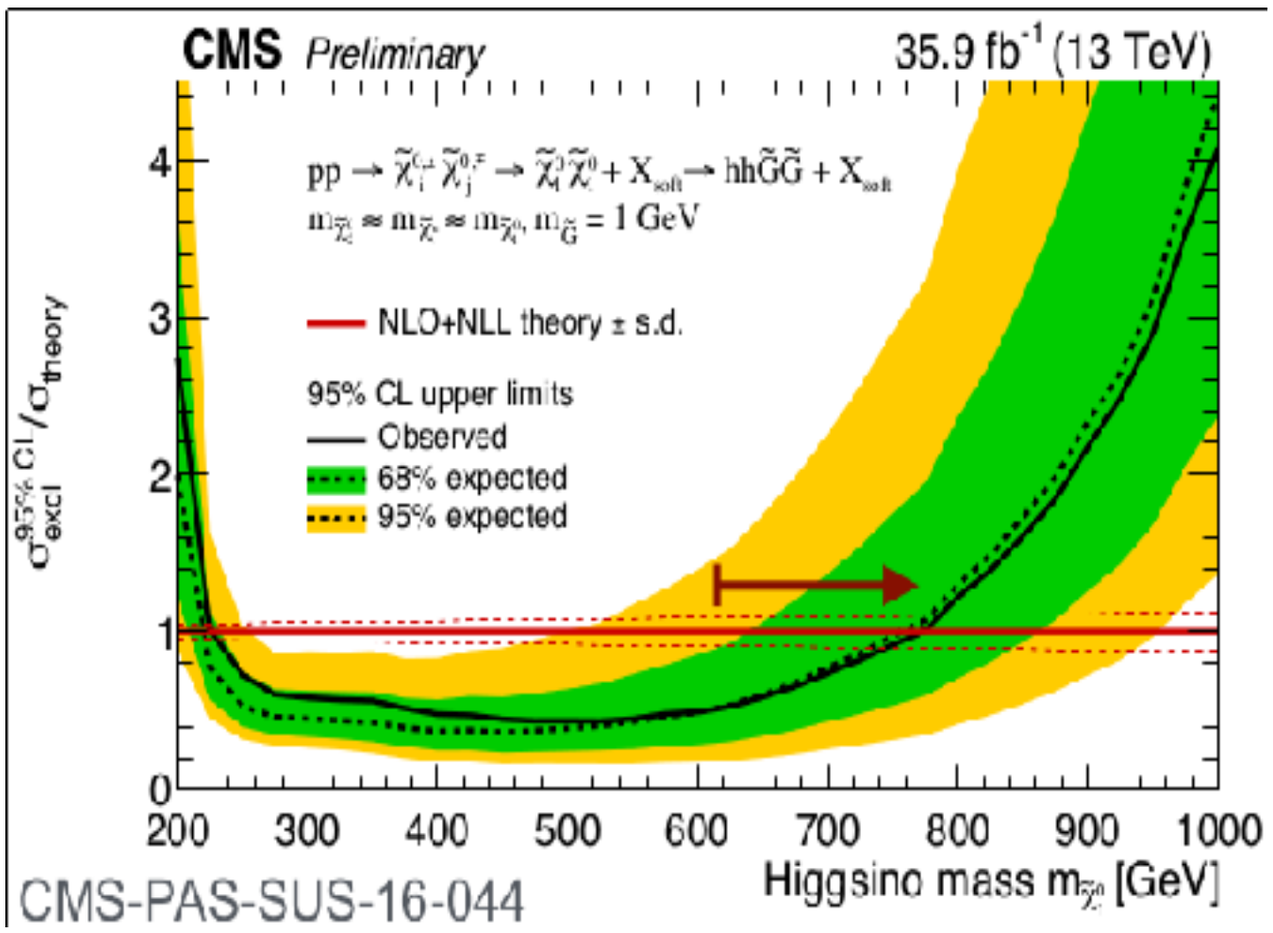
CMS DP-2018/046

More details in the Javier's presentations

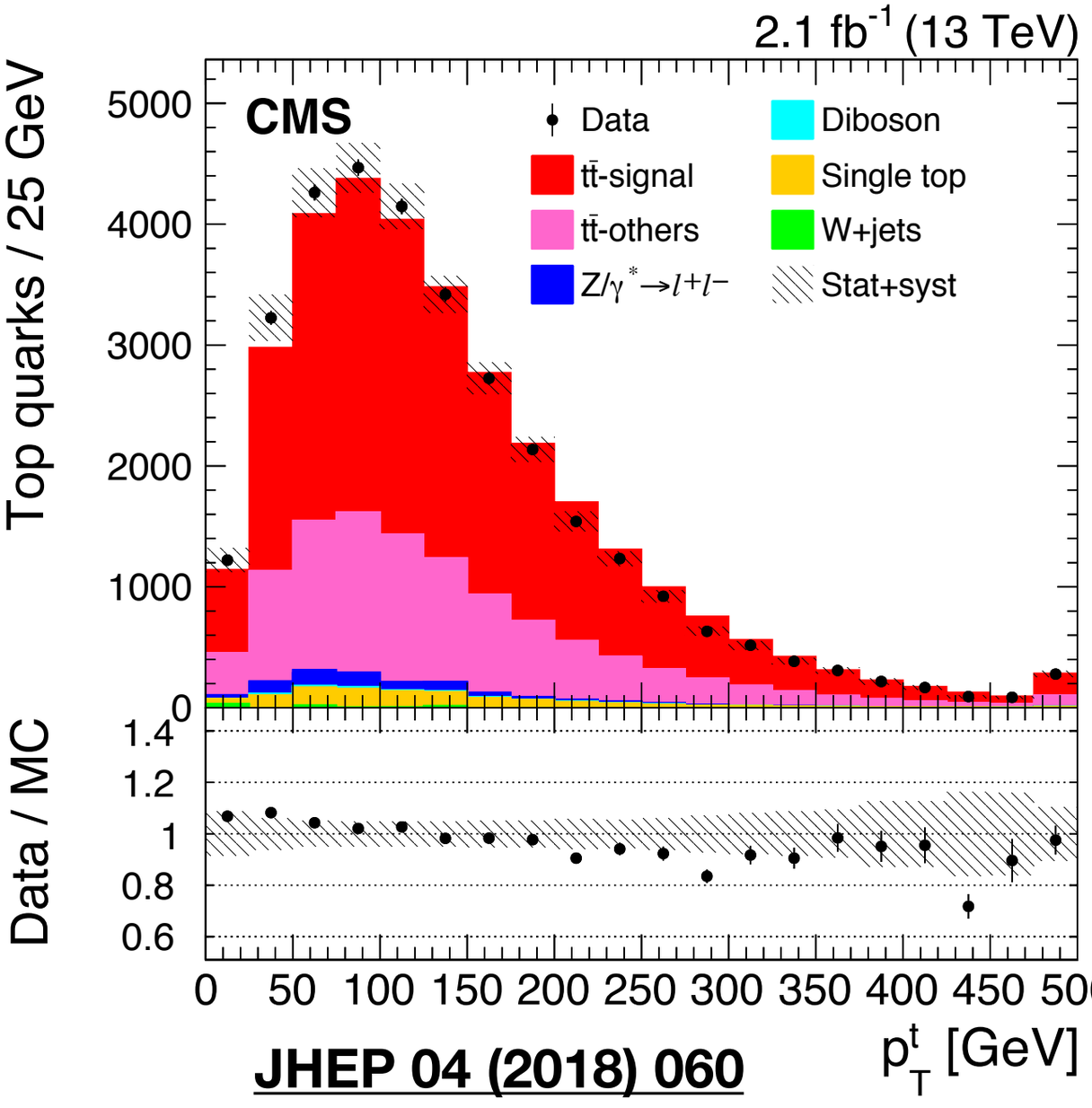


One tagger to rule them all?

What is true performance?



10.1103/PhysRevD.97.032007



JHEP 04 (2018) 060

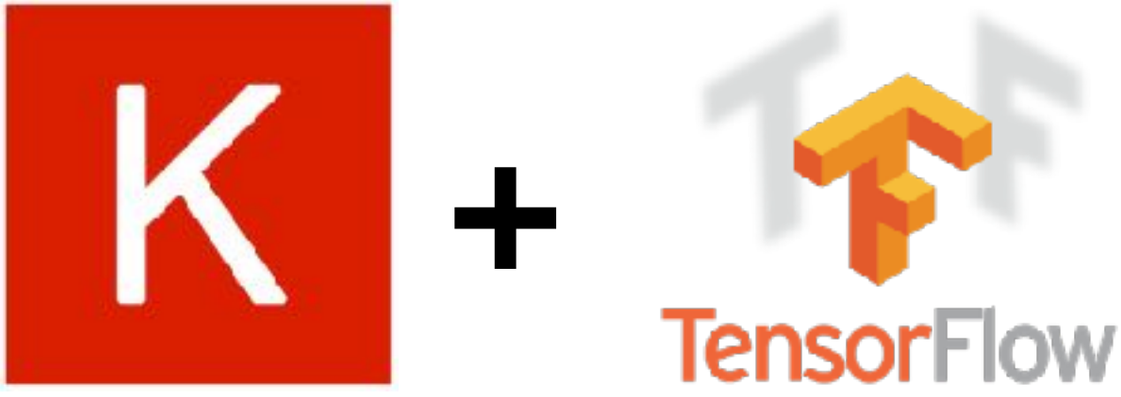
The next challenges ahead lie in a better calibration of these new taggers

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

This repository Search Pull requests Issues Marketplace Explore

cms-sw / cmssw

Watch 73 Star 534 Fork 2,521

Code Issues 330 Pull requests 136 Projects 0 Wiki Insights

Tensorflow-based integration of new DeepFlavour tagger #19893

Merged cmsbuild merged 150 commits into `cms-sw:master` from `pablodecm:deep_flavour_tf_rebased_2007` on 25 Jan

Conversation 830 Commits 150 Files changed 54 +6,293 −29

pablodecm commented on 25 Jul 2017 • edited Contributor

This pull request integrates the new DeepFlavour tagger, using the library CMSSW-DNN by **@riga** (the required part is also included) and adds it to the standard sequences. You can find an overview of the reason and design behind this PR in this BTV WG presentation.

PAT vs reference training framework (latest version)

Here are some checks of compatibility of CMSSW pat-based discriminators computed using the producers develop for this PR with the output from the training framework (DeepJet) as 2D histograms

Reviewers

makortel

riga

mverzett

Dr15Jones

smuzaffar

slava77

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 json 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

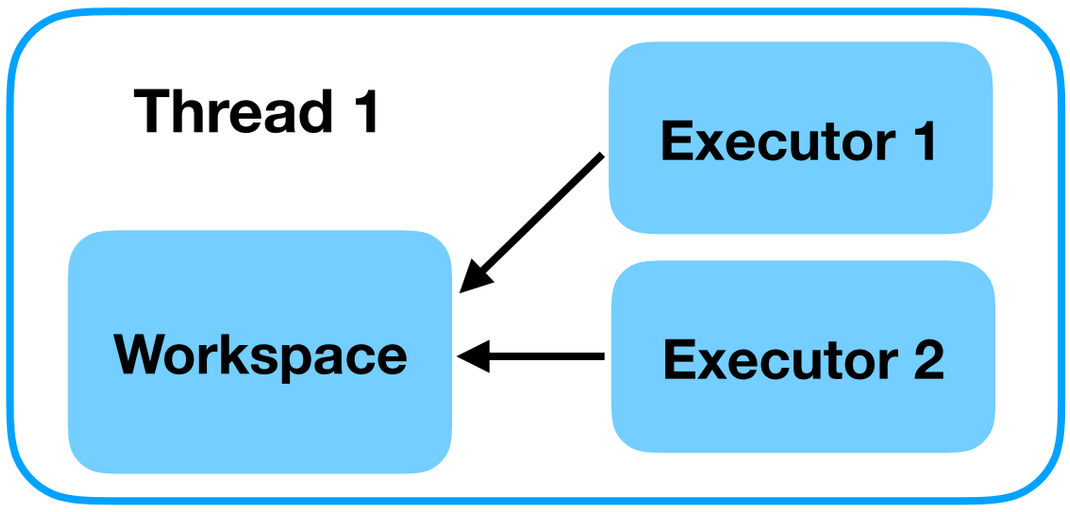
MXNet issues - thread safety

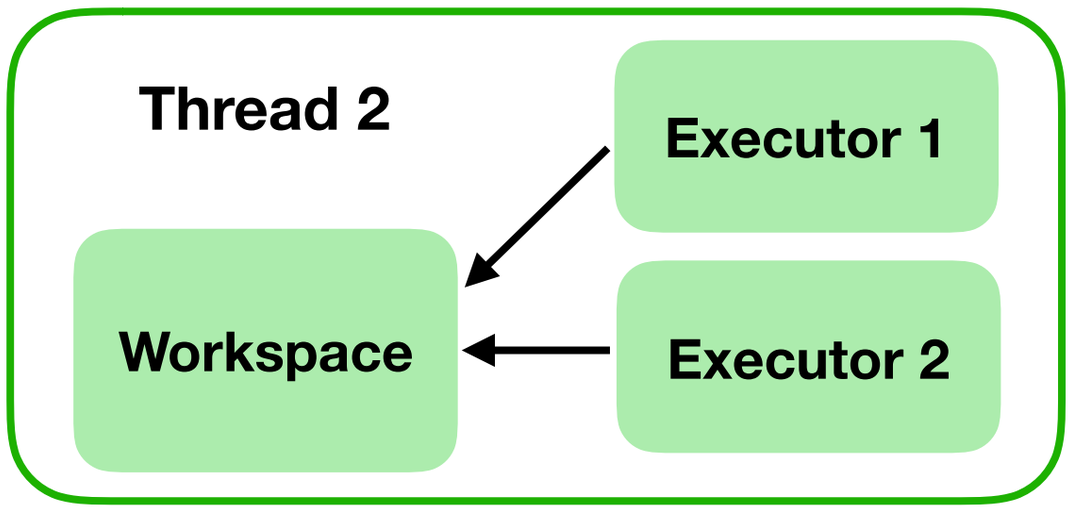
Problem: once the previous problem was fixed, the resulting output was still sometimes inconsistent — race condition

- 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)





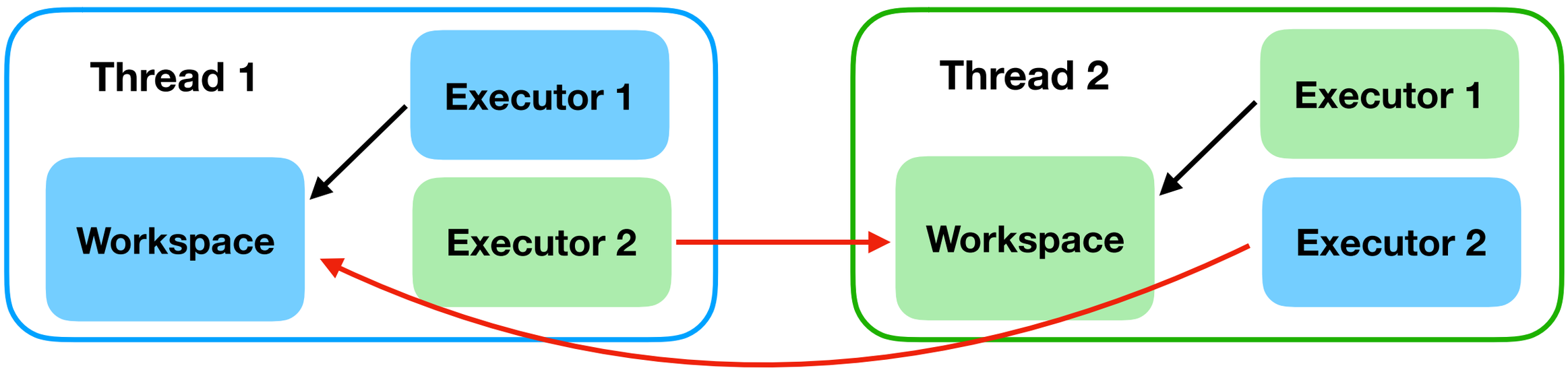
MXNet issues - thread safety

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

- ML has always been a fundamental component of jet physics for classification and regression

- As the datasets grew larger the techniques followed bringing more and more complex solutions
  - New taggers are more performing but also bring significant issues for labelling and calibration
  - Often the knowledge is shared across multiple working groups

- New tools are not physics oriented, we are no longer *the* big data, we have to adapt our mindset, and our code