Machine Learning Inference in CMSSW

Huilin Qu

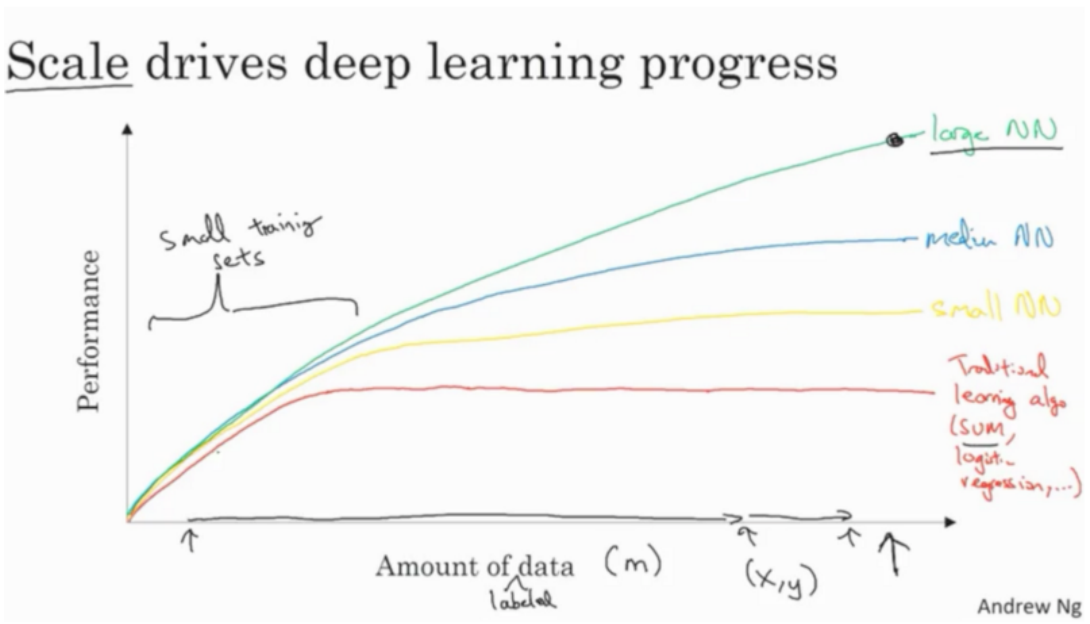
on behalf of the CMS collaboration

ATLAS Machine Learning Workshop

November 15, 2019

Introduction

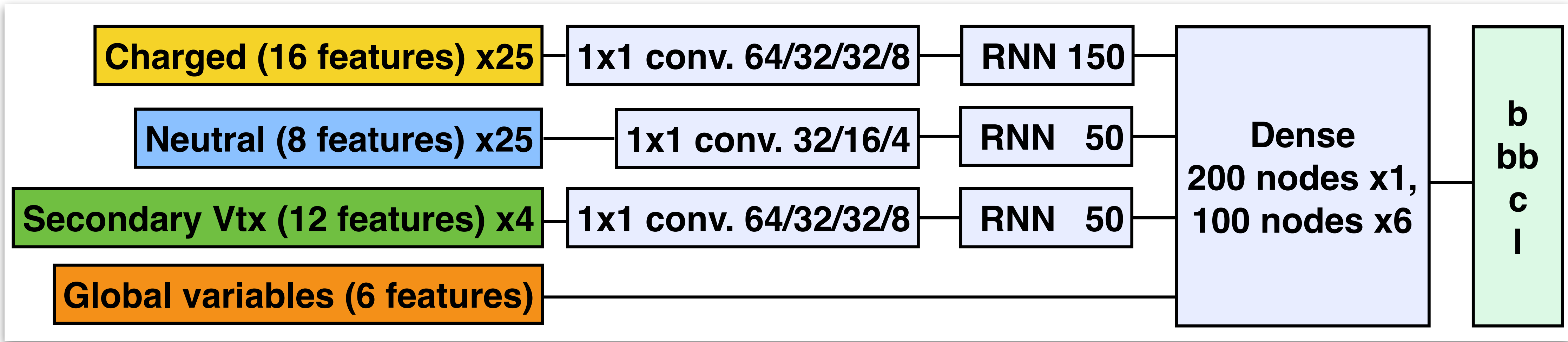
- Machine learning (ML) can provide powerful tools for particle physics experiments
- Trend in recent years: deep learning (DL) + low-level inputs

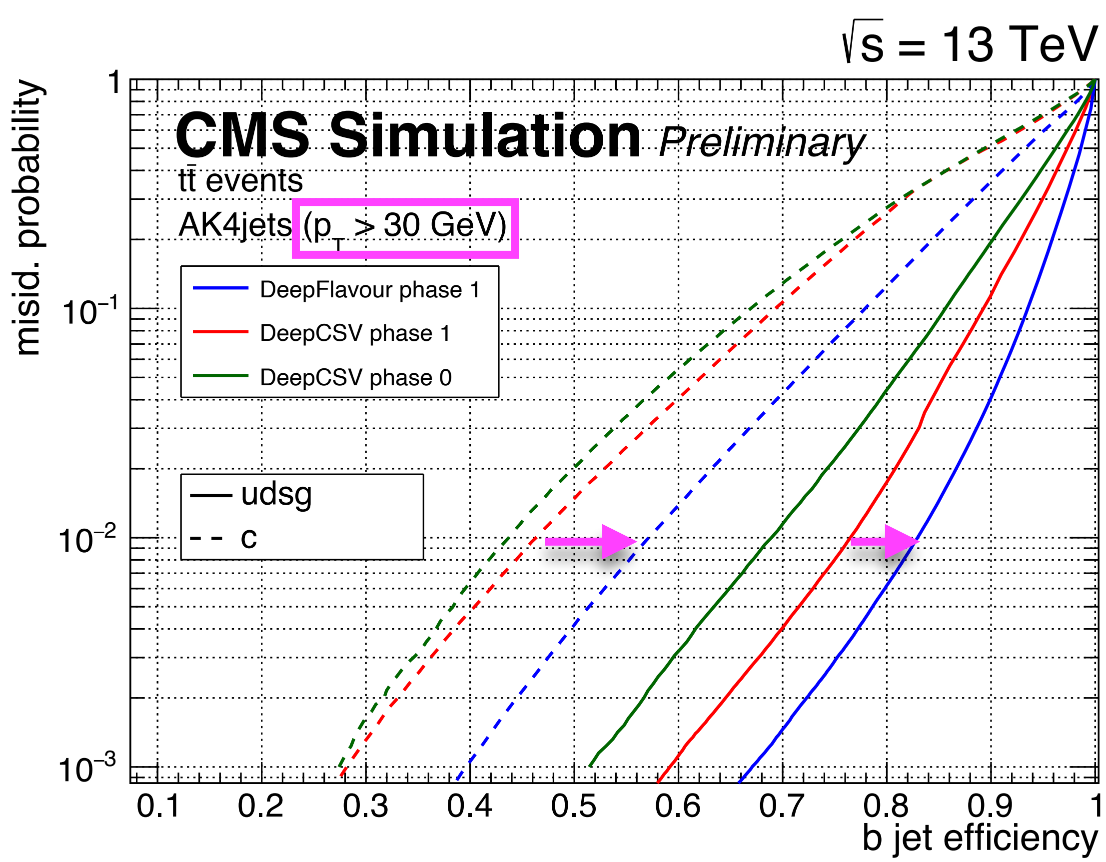


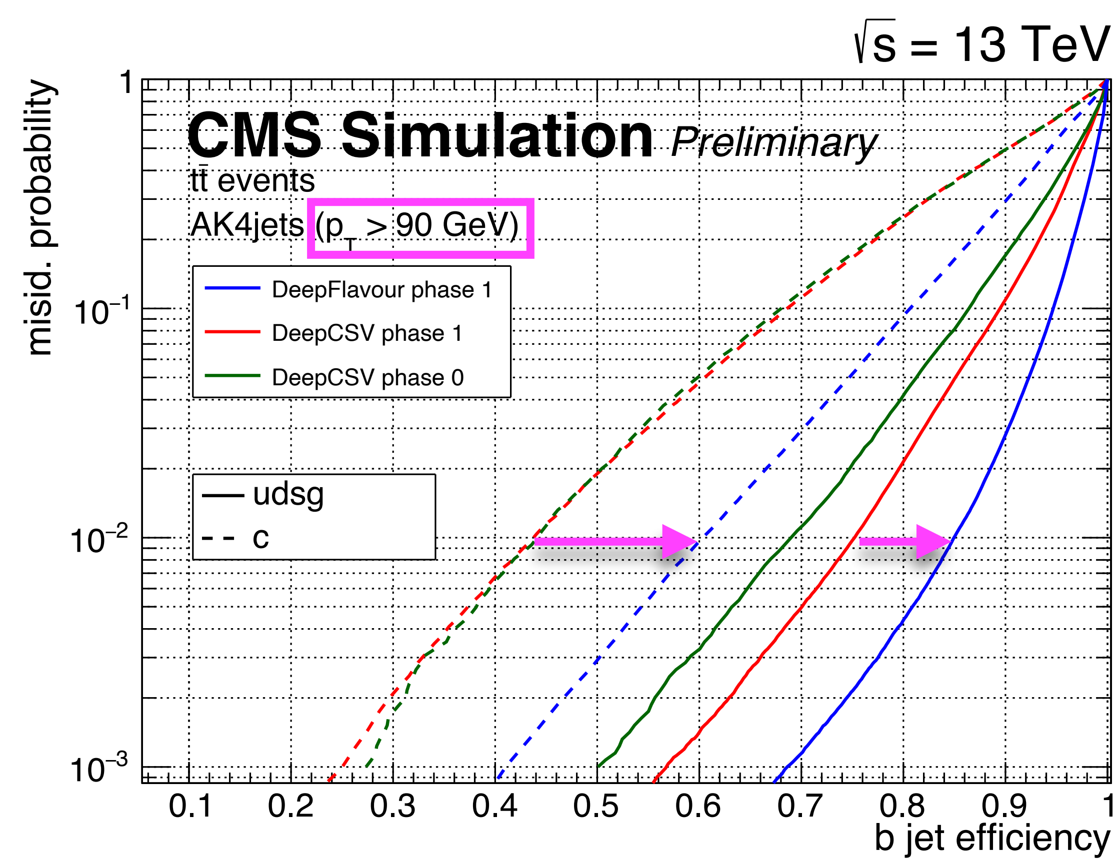
- A variety of new DL algorithms have been developed in CMS
  - b-tagging
  - boosted jet tagging
  - tau identification
  - b-jet energy regression
  - …

DEEPJET (DEEPFLAVOUR)

- AK4 jet flavour tagger



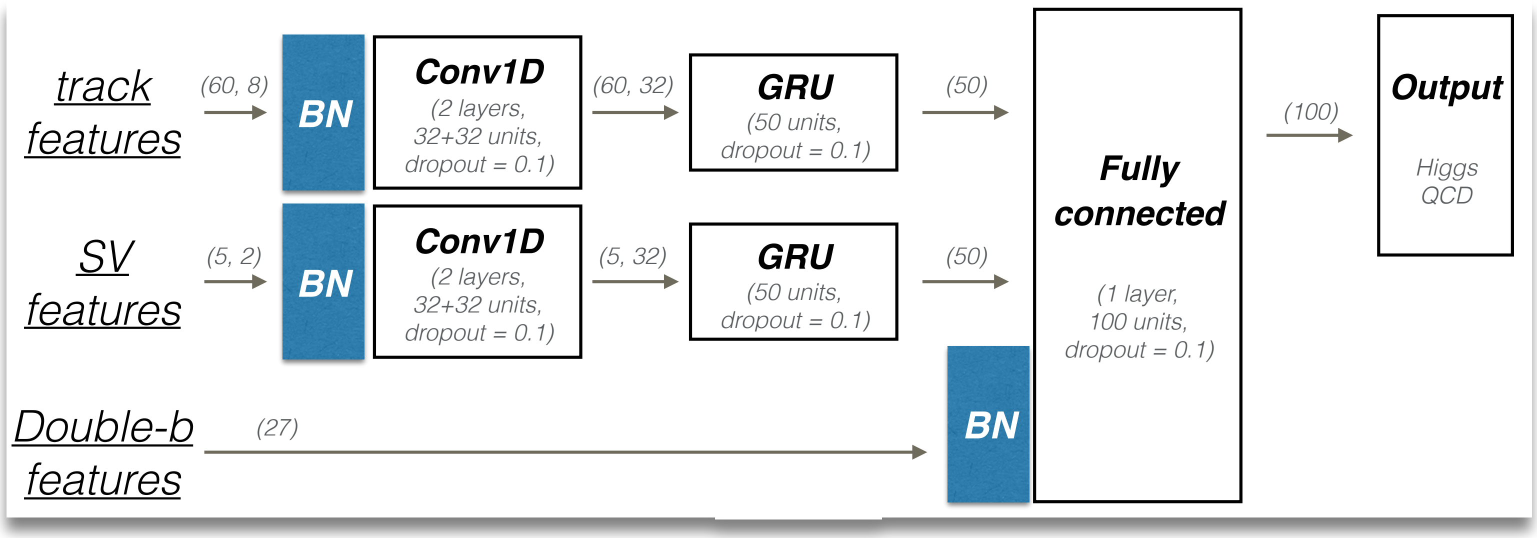


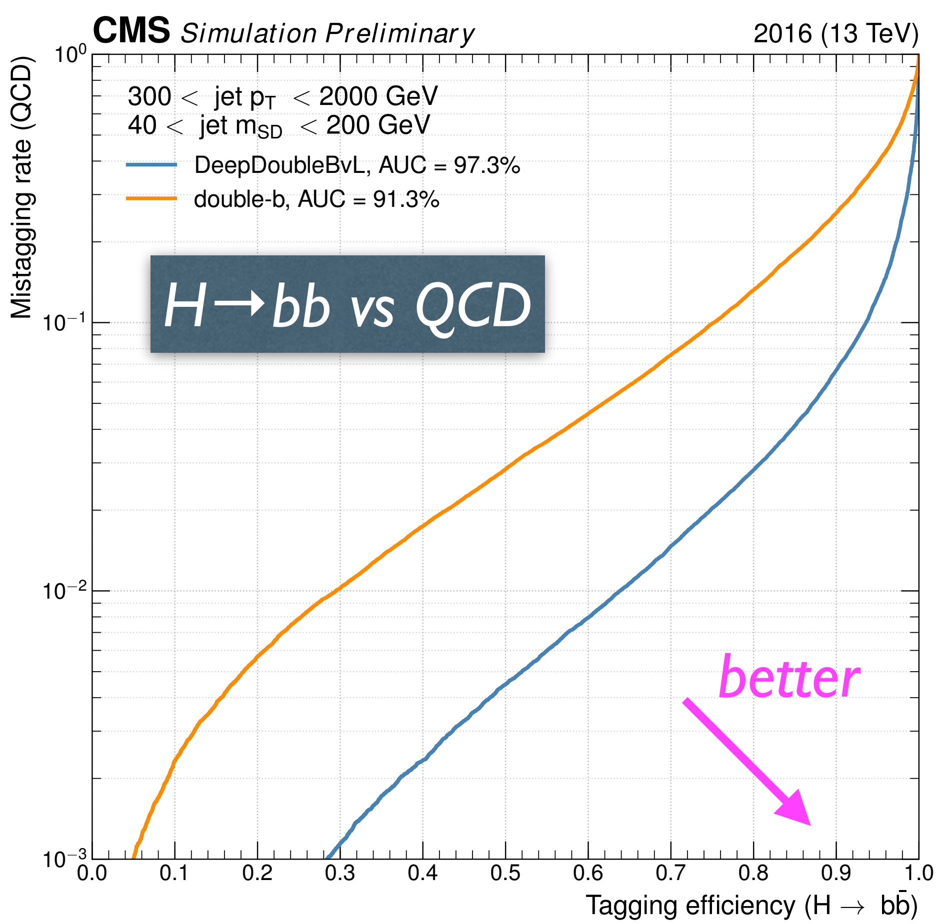


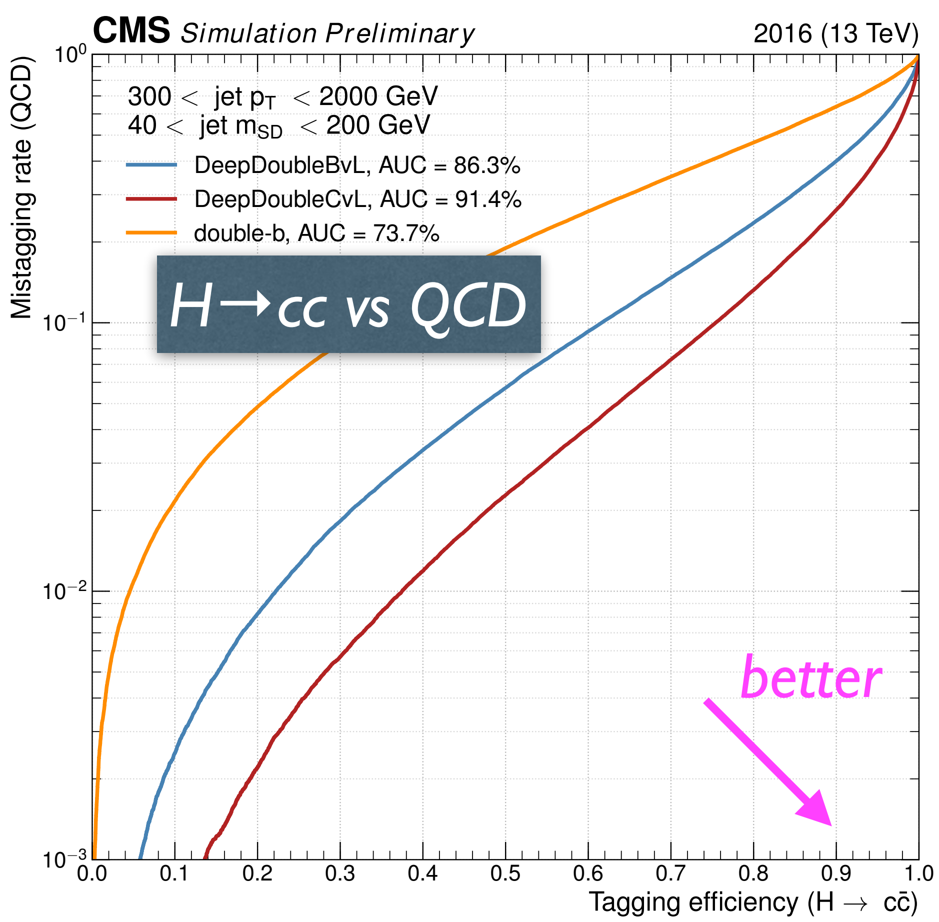
CMS-DP-2018-033

DeepDoubleX

- Boosted jet flavour tagger for bb/cc







CMS-DP-2018-046

DeepAK8

- Multi-class boosted jet tagger for top / W / Z / H

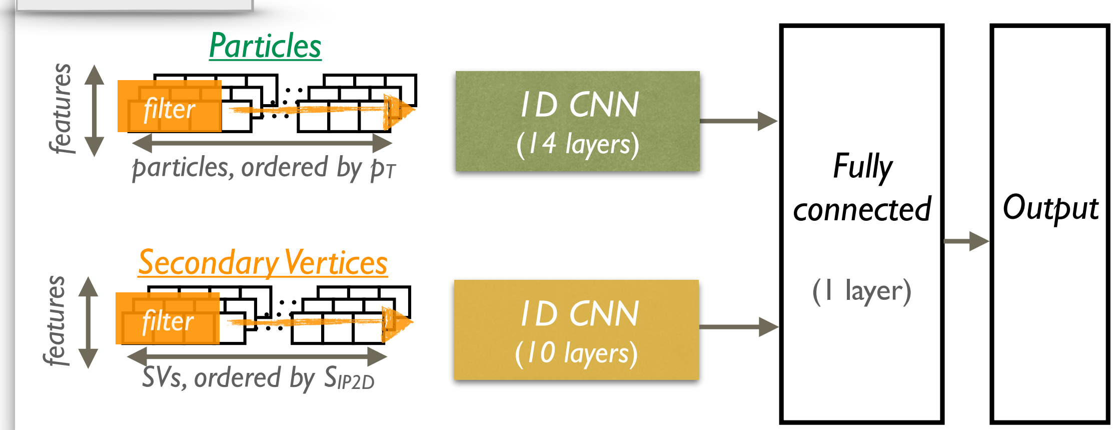
Inputs

Substructure — Particles
- Up to 100 PF candidates(*)
- Sorted in descending p_T order
- Uses basic kinematic variables, Puppi weights, and track properties (quality, covariance, displacement, etc.)

Flavour — Secondary vertices
- Up to 7 SVs(*) (inside jet cone)
- Sorted in descending S_{IP2D} order
- Uses SV kinematics and properties (quality, displacement, etc.)

(*) Number chosen to include all candidates for ≥ 90% of the events

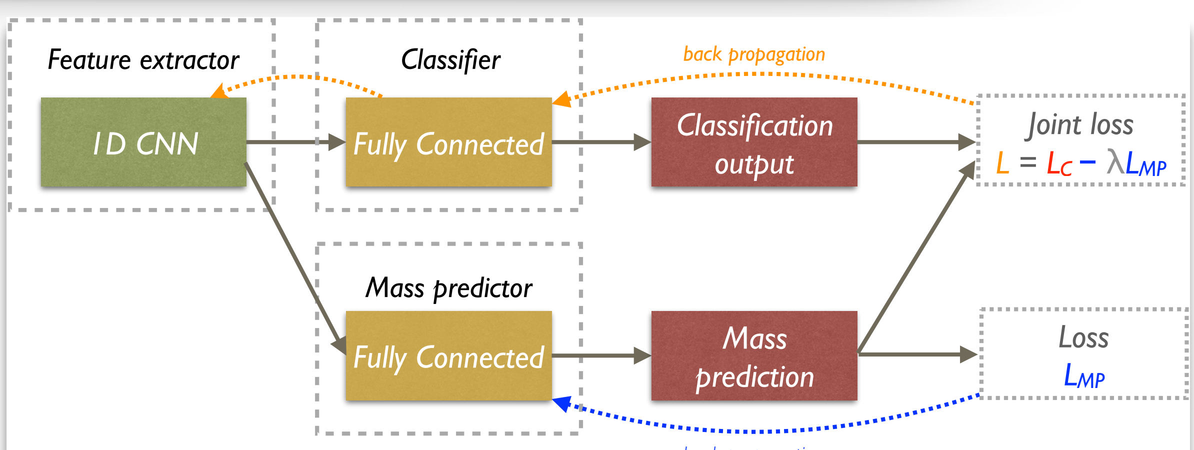
Architecture

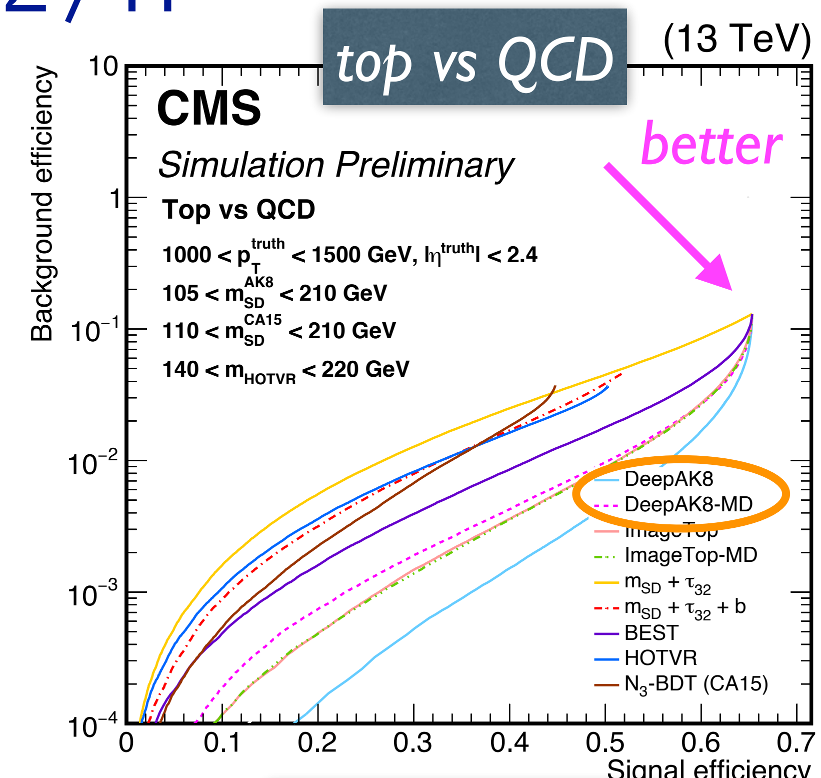


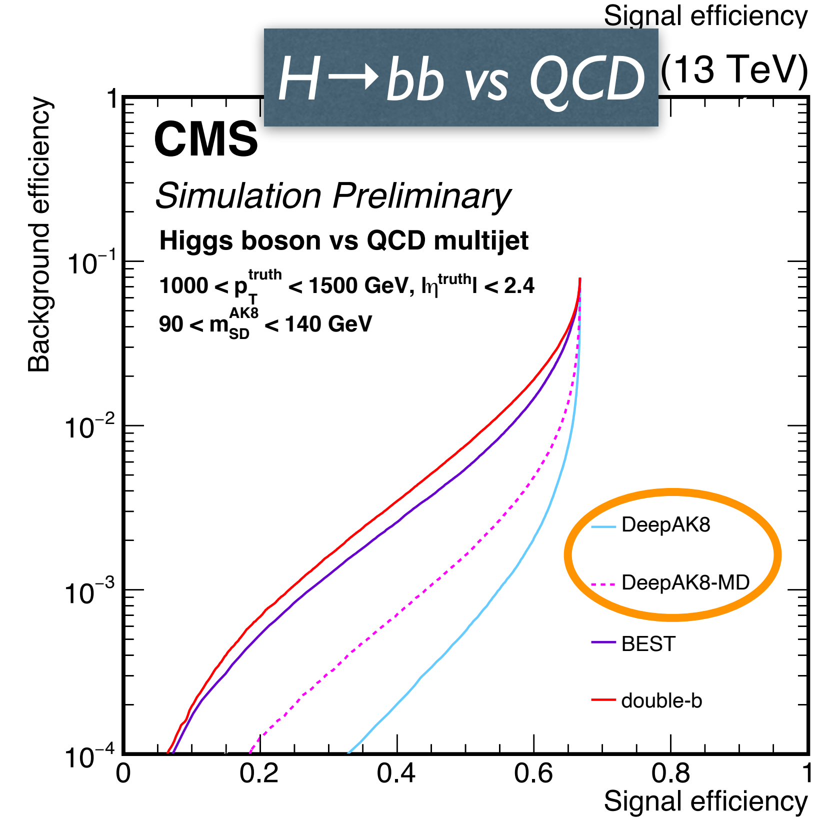
Output

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)

DeepAK8-MD: mass decorrelation w/ adversarial training



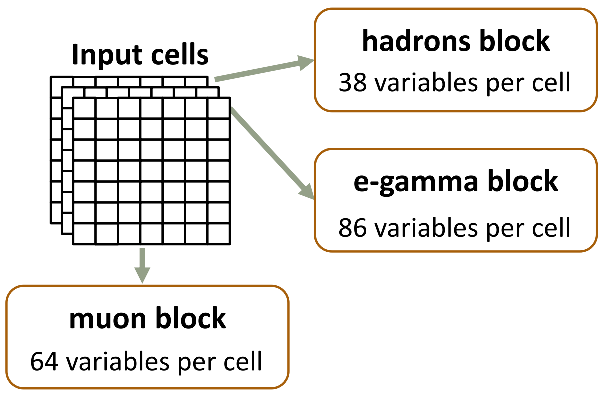




CMS-DP-2017-049 CMS-PAS-JME-18-002

DeepTau

- CNN-based hadronic tau identification algorithm

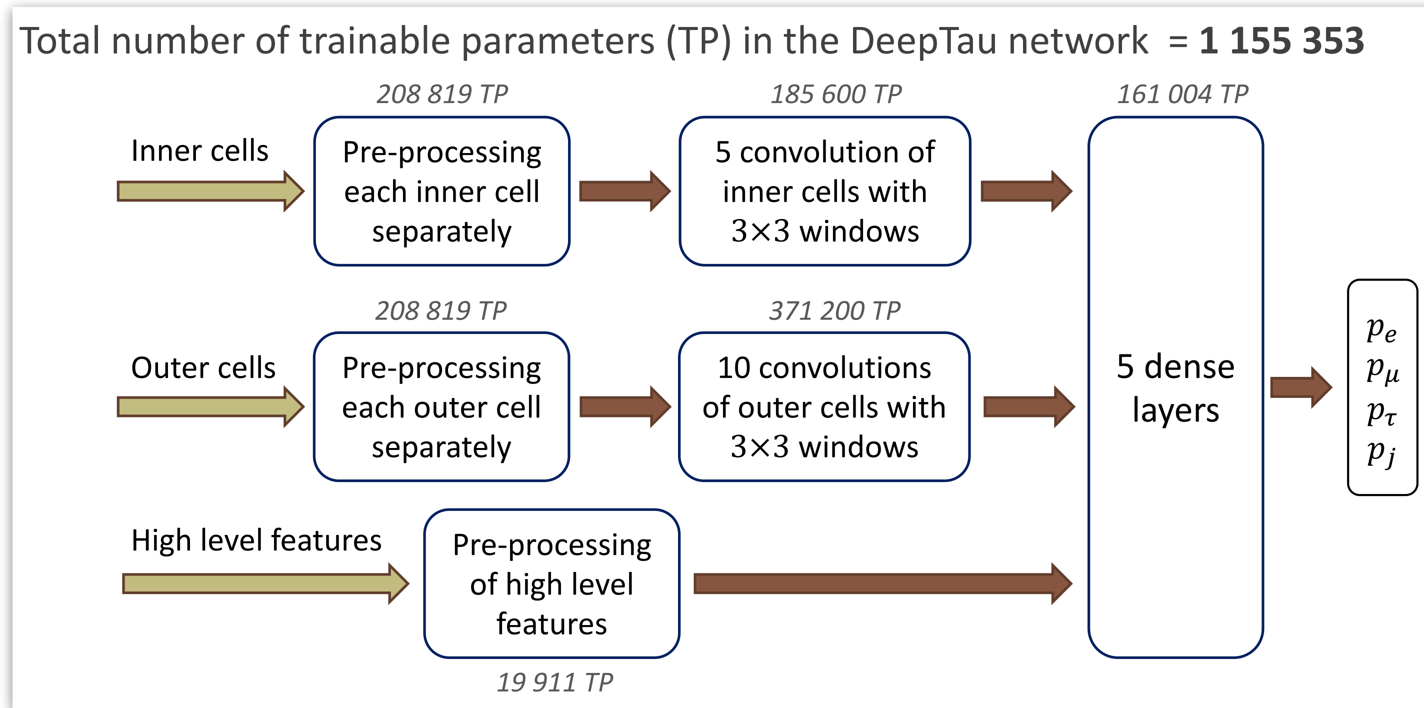


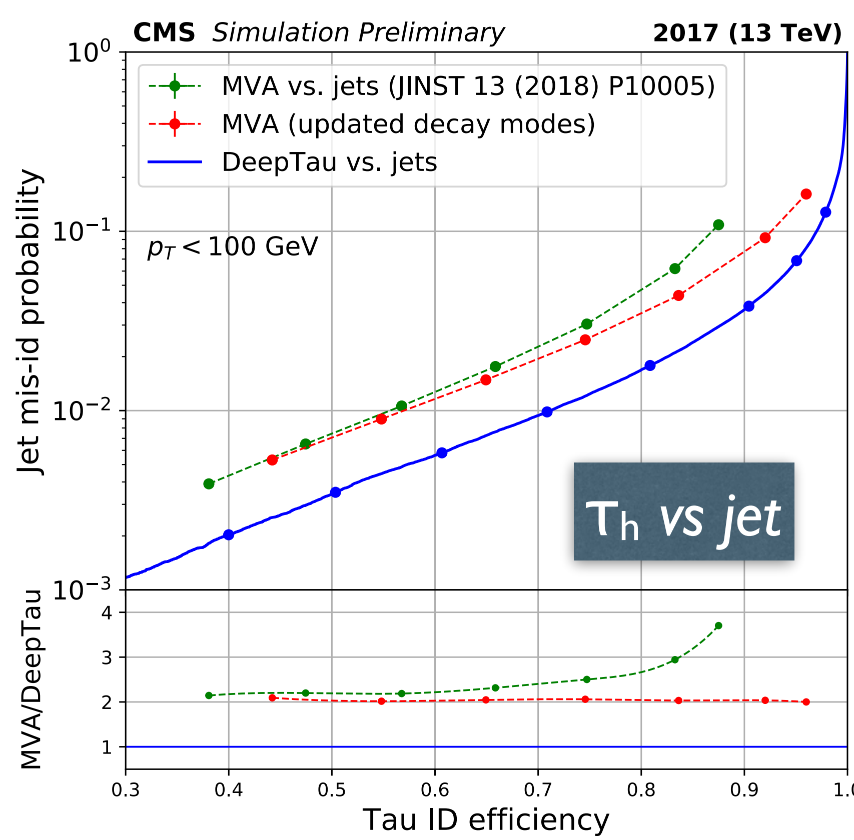
Inner cone = signal cone

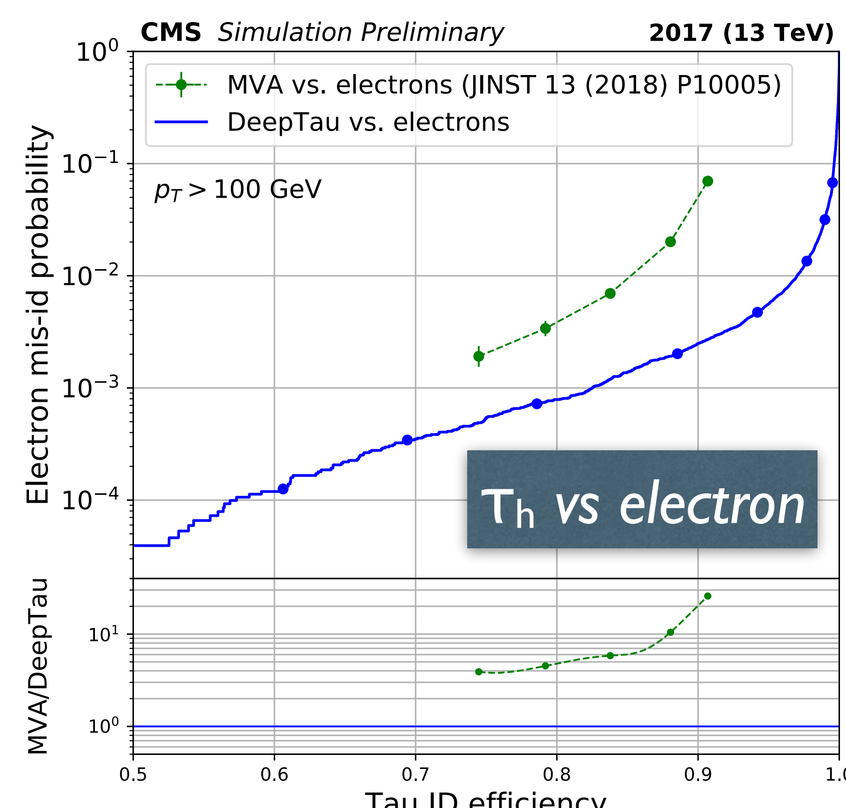
$$\Delta R < \max\left[\min\left(0.1, \frac{3}{p_T^\tau}\right), 0.05\right]$$

Outer cone = signal & isolation cones

$$\Delta R < 0.5$$



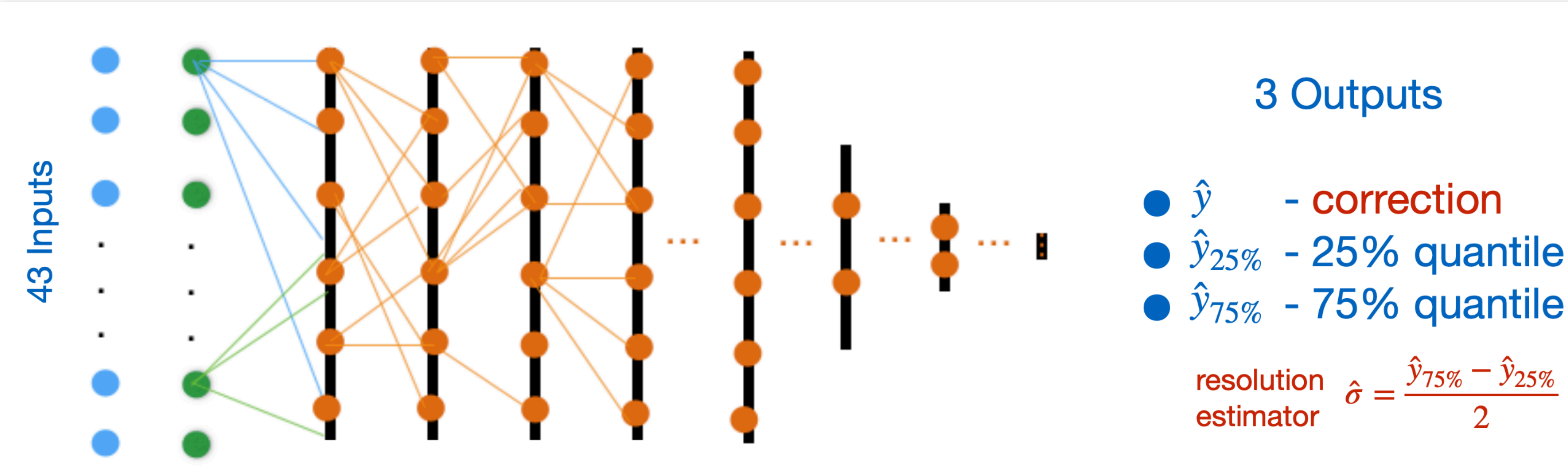




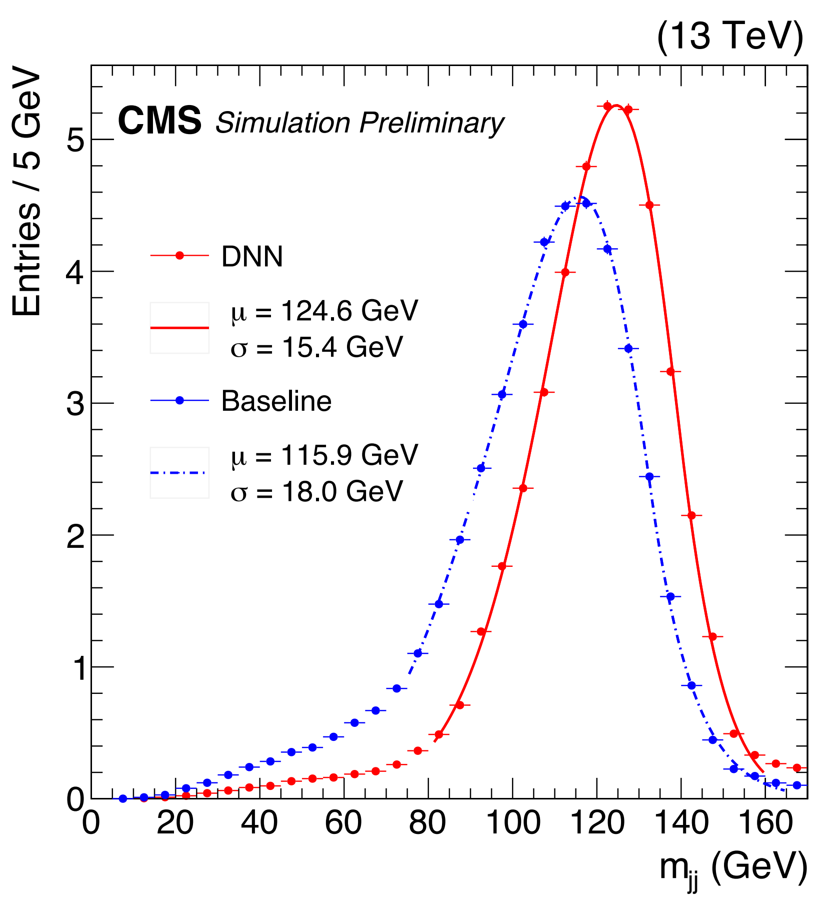
CMS-DP-2019-033

B-Jet Energy Regression

- Simultaneous estimation of the b-jet energy and its resolution



More details in [N. Chernyavskaya's talk]



Joint loss function for correction (Huber) and resolution (quantiles) :

$$\boxed{Loss = Huber(y, F(x)) + \rho_{0.75}(y - F(x)) + \rho_{0.25}(y - F(x))}$$

13% improvement in per-jet relative resolution
20% improvement in dijet mass resolution

Successfully applied to the CMS H→bb observation analysis

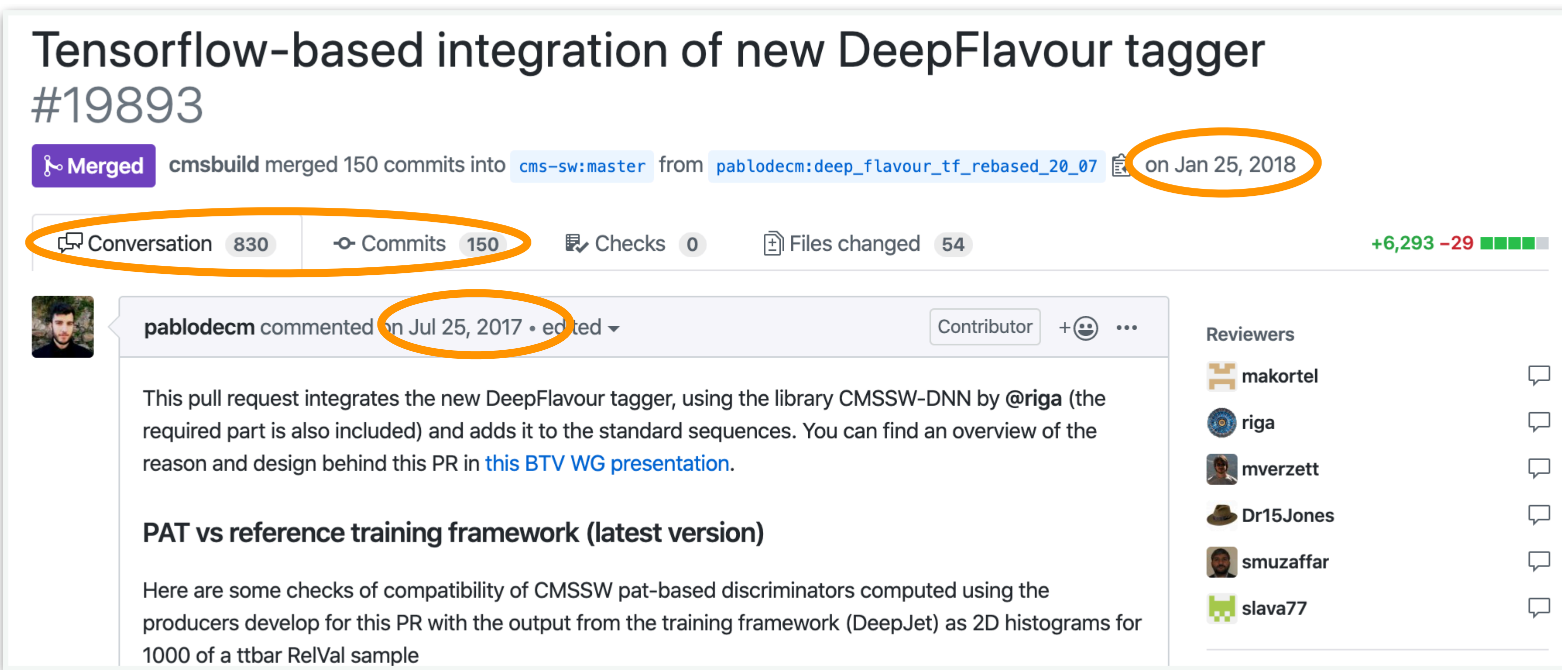
[CMS-PAS-HIG-18-027]

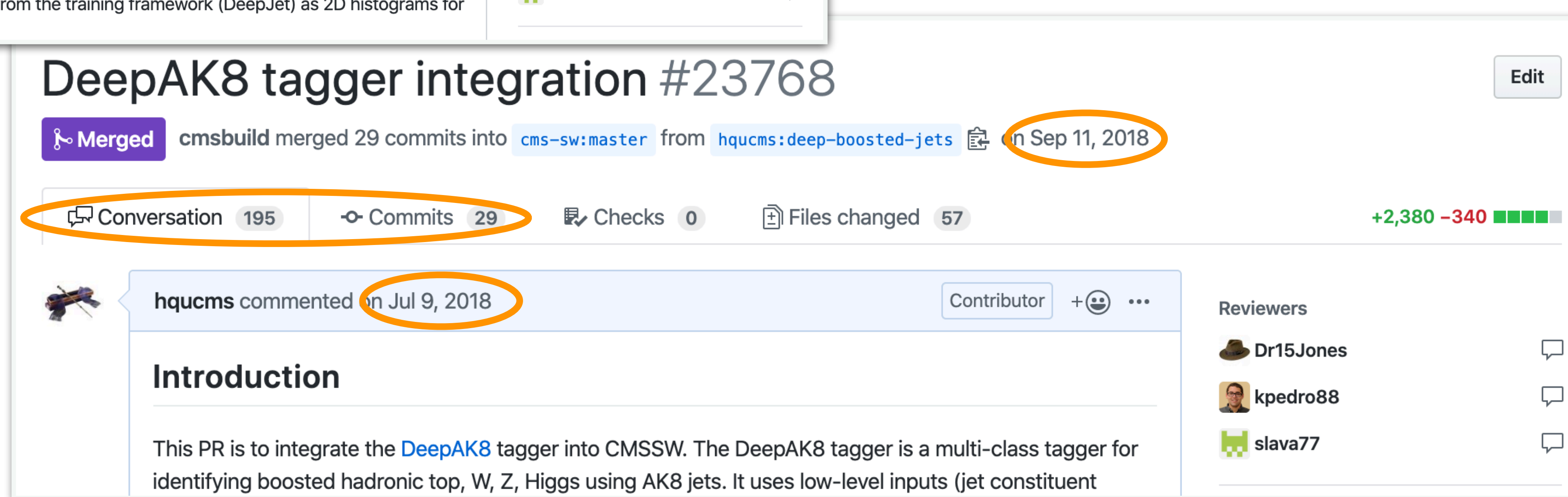
From Development to Deployment

- The development of a DL model takes lots of effort
  - *a good DL model = input feature selection + training dataset preparation + network architecture design + hyperparameter optimization + …*
- Next step: deploying to production!
  - but…

From Development to Deployment

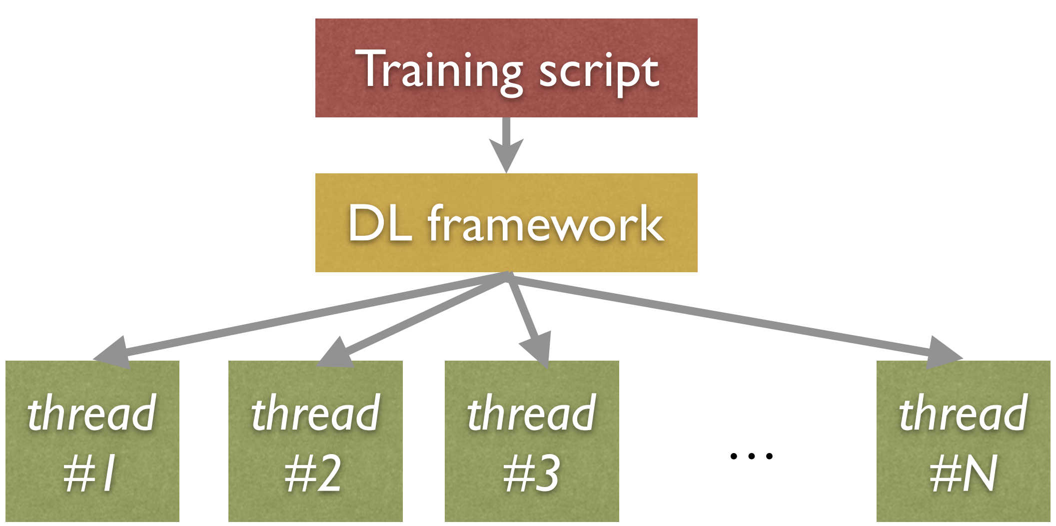
- The development of a DL model takes lots of effort
  - *a good DL model = input feature selection + training dataset preparation + network architecture design + hyperparameter optimization + …*
- Next step: deploying to production!
  - but…



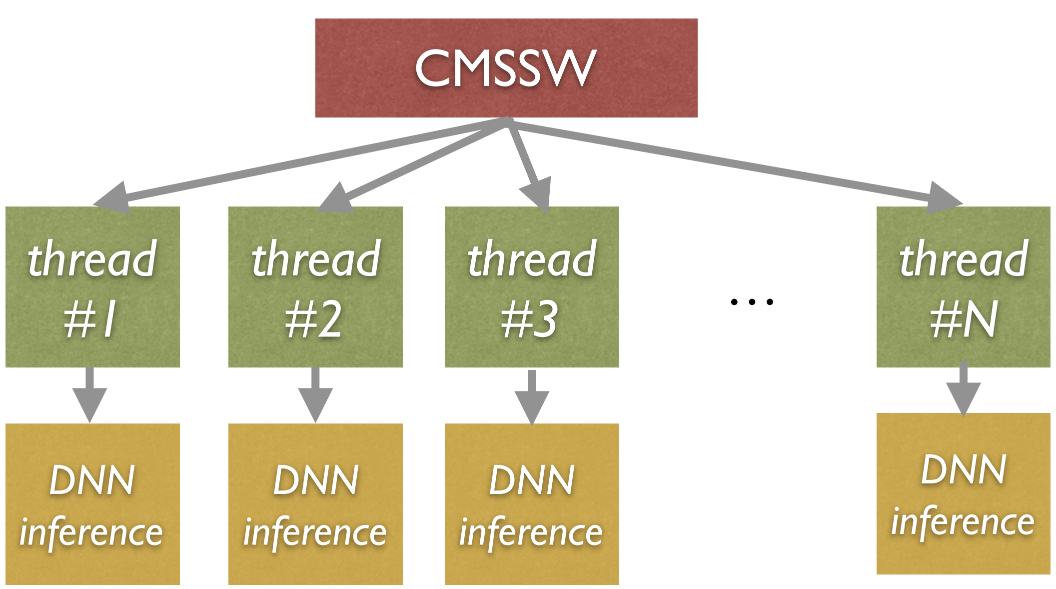


TRAINING VS INFERENCE

- Training
  - python based
  - typically on GPU
  - large batch size
    - O(100) to O(1000)
  - standalone environment
    - weak constraints on timing/ memory
    - multi-threading managed by the DL framework (e.g., TensorFlow)



- Inference
  - C/C++ based
  - on CPU
  - small batch size
    - O(1) to O(10), often just 1
  - integrated in the experimental software (e.g., CMSSW)
    - tight constraints on timing/memory
    - multi-threading managed by the experimental software



ML Inference Engines in CMSSW

- A number of DL frameworks have been integrated into CMSSW to support the new DNN-based algorithms
  - TensorFlow
    - DeepJet
    - DeepDoubleX
    - DeepTau
    - b-jet energy regression
  - MXNet
    - DeepAK8
  - ONNX Runtime
    - new development
    - can support all these models by converting to ONNX format
- Modifications are needed for all of them to work nicely with CMSSW

TensorFlow

- TensorFlow
  - most widely used framework (together with Keras)
  - complicated to build and integrate (requires Google's build system bazel, etc.)
- Issue 1: Multi-Threading
  - upon startup, TF creates *lots* of threads in its thread pool for parallel data loading and parallelism within/between operators
  - good for end-users who runs only 1 thread to call TF, but not good for HEP frameworks that typically manage their own threading schemes (CMSSW uses TBB)
  - solved with the implementation of two custom sessions
    - NTSession (default in CMSSW): disable multi-threading
    - TBBSession: threads scheduled by Intel's TBB
- Issue 2: Memory Consumption
  - TF Graphs obtained after training can be quite large (e.g., 150 MB for DeepJet)
  - memory footprint can be reduced by a factor of O(10-100) by:
    - converting *variables* to *constant* tensors (`freeze_graph`)
    - removing ancillary information needed only for training
    - a number of tools available online, e.g., keras_to_tensorflow
  - further reduction: load TF graph only once and share it among all threads (sessions)

MXNet

- MXNet
  - a DL framework focused on efficiency and scalability
  - relatively straightforward to build and integrate (cmake build system, standard BLAS library)
  - exported models are ready-to-use for inference (model json + binary parameter file)
- Issue 1: multi-threading
  - similar problem as TF: MXNet creates and manages its own thread pool
  - solution: use MXNet's "NaiveEngine" (no threading) and make it "thread_local" (so each thread can call it independently)
    - need to re-assign the resources (workspace) in each run to ensure thread-safety (more details in M. Verzetti's talk last year)
- Issue 2: BLAS library
  - DeepAK8 inference runs 4-5x slower in CMSSW than w/ standalone MXNet
  - the problem was tracked to the use of the BLAS library
    - the standalone MXNet links to OpenBLAS statically
    - MXNet in CMSSW is built to link with OpenBLAS dynamically, but a slower BLAS library (glsblas) is used by other softwares (e.g., ROOT) and loaded first, thus providing the BLAS symbols to MXNet
    - solved by linking to OpenBLAS consistently in all CMS softwares

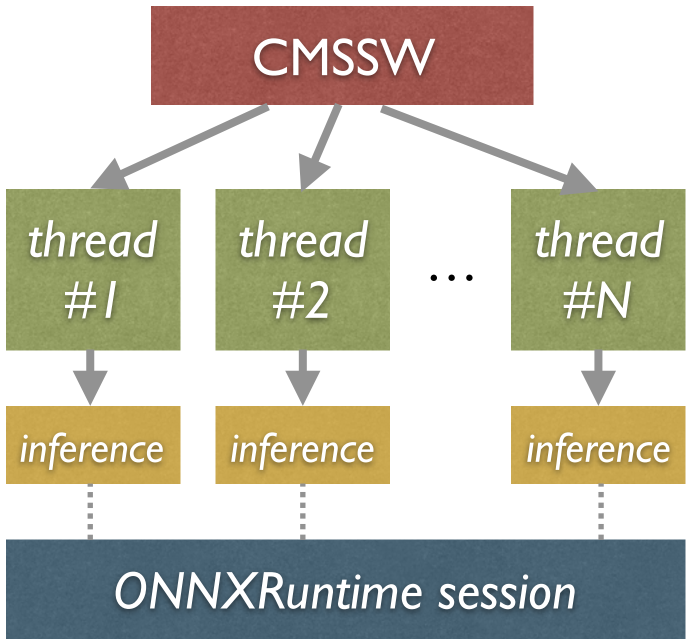
ONNX Runtime

ONNX RUNTIME

- Open Neural Network Exchange (ONNX)
  - an open source format for ML models w/ increasing adoption
  - supports most of the main-stream DL operators
  - conversion tools available for most of the DL frameworks: Keras/TF, PyTorch, MXNet, etc.
- ONNX Runtime
  - "a performance-focused complete scoring engine for ONNX models"
  - advantages:
    - flexibility: can support a wide range of models via ONNX
    - speed: optimized for inference (including on CPUs), rather than training (TF/MXNet/PyTorch etc.)
    - thread-safety: "Multiple threads can invoke the Run() method on the same inference session object."
  - caveats:
    - ONNX may not support all, especially novel ML models

ONNX Runtime Integration

- A few modifications to make it work better w/ CMSSW
  - configured it to run in a "no-threading" mode
    - i.e., each CMSSW thread uses the global inference session object to run inference concurrently with no extra threads
    - setting `intra_op_num_threads` and `inter_op_num_threads` to 1 gives the desired behavior (i.e., it does not create any new threads)
    - however, need to <u>remove a hard-coded thread pool</u> for the LSTM operator
      - likely will be fixed officially in the future



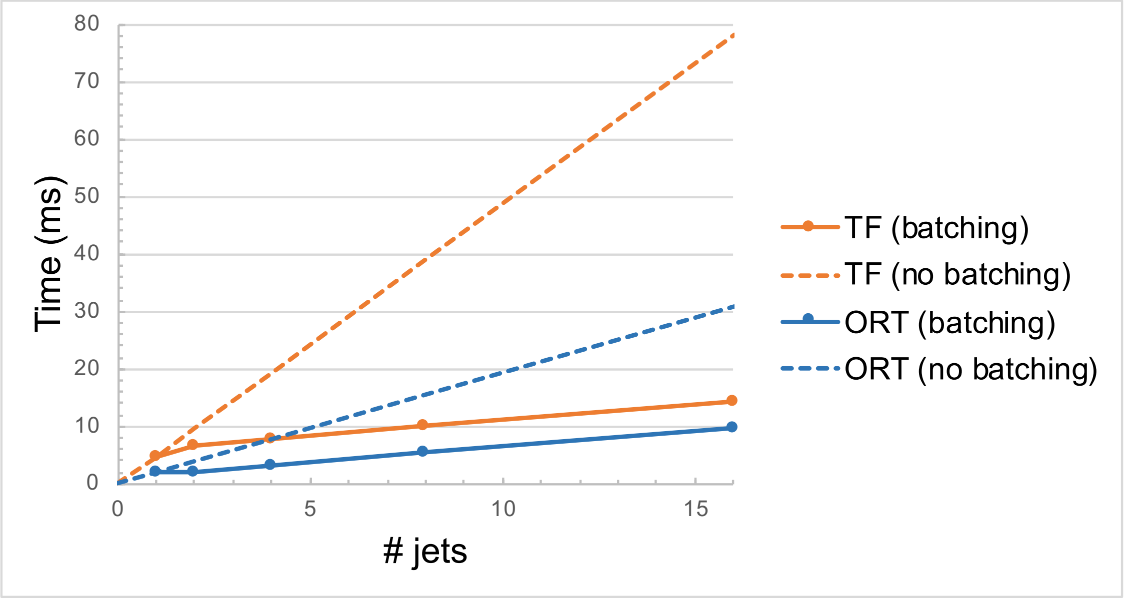
  - introduced an environment variable to control the runtime kernel selection
    - ONNX Runtime's math library (MLAS) selects the fastest compute kernel *dynamically* based on the available CPU instruction sets
      - outputs are not bitwise equal on different CPU architectures as different instructions (SSE/AVX/AVX2/etc.) will be used – causes trouble for PR validation
    - <u>added an environment variable</u> to control the allowed instruction sets
      - default to using only SSE: not attempting to use more advanced instructions (like AVX)
      - ensures bitwise reproducibility across different CPU architectures
      - dynamic kernel selection can be switched on for production to save run time

ONNX Runtime: Timing

- Significant speed-up w/ ONNX Runtime compared to the TF/MXNet based implementation (cms-sw/cmssw#28112)
  - depending on the network architecture, the speed-up varies from 10-15% to ~10x
  - the use of newer vector instructions (e.g., AVX) can bring further improvements

Time (s) / event	Baseline	ONNX Runtime (SSE)	Speed-up w.r.t baseline	ONNX Runtime (AVX)	Speed-up w.r.t baseline
DeepTau	0.039245	0.053057	*0.74*	0.024901	*1.58*
DeepJet	0.058576	0.009333	*6.28*	0.006735	*8.70*
DeepAK8	0.003538	0.003222	*1.10*	0.002107	*1.68*
DeepAK8-MD	0.003598	0.003153	*1.14*	0.002078	*1.73*
DeepDoubleBvL	0.004457	0.000451	*9.88*	0.000363	*12.28*
DeepDoubleCvB	0.004514	0.000445	*10.14*	0.000355	*12.72*
DeepDoubleCvL	0.004997	0.000478	*10.45*	0.000398	*12.56*

- Another observation: batch evaluation can bring substantial speed-up in some cases
  - right: DeepJet inference w/ and w/o batching

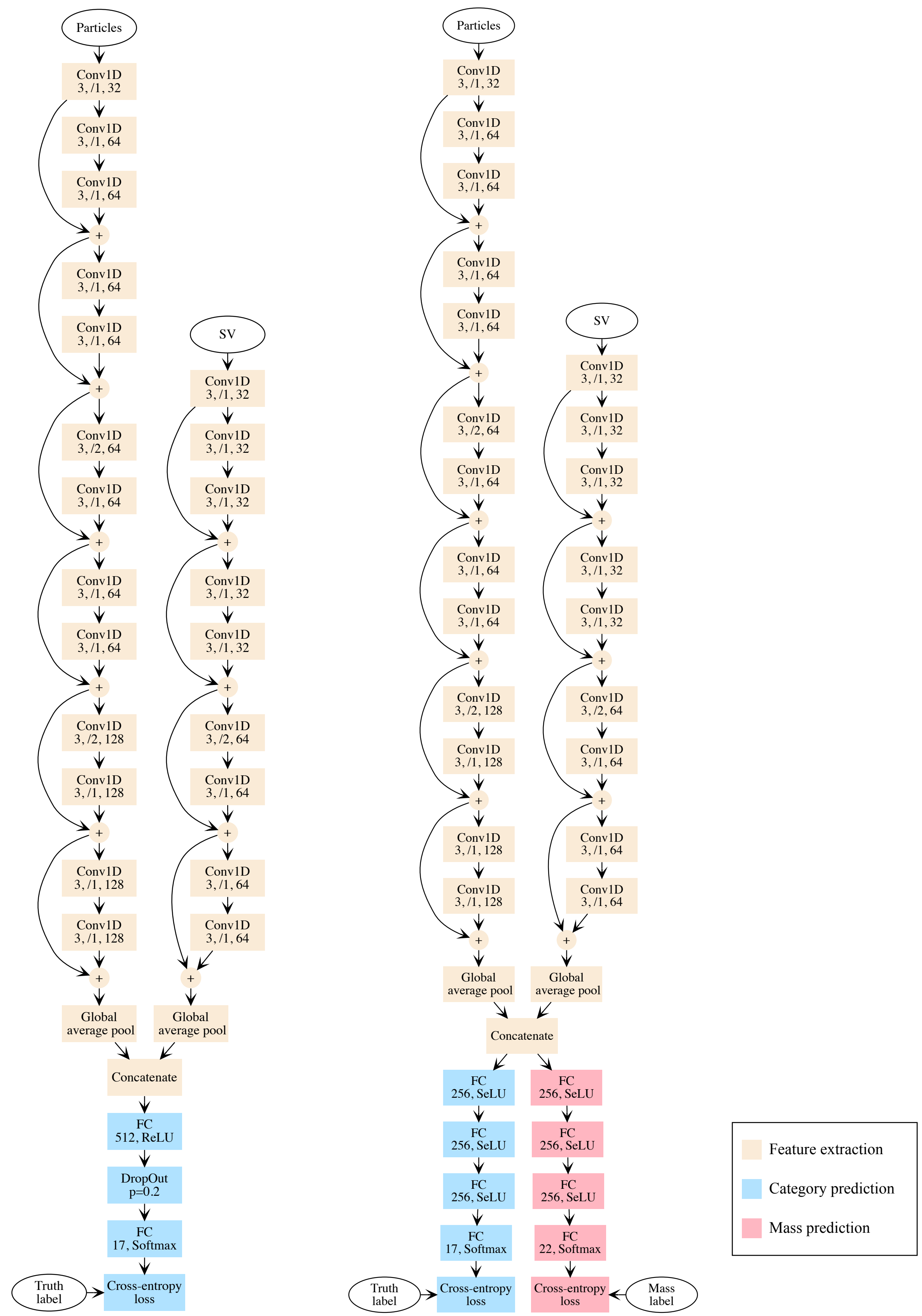


SUMMARY

- A number of DL-based object reconstruction and identification algorithms have been developed in CMS over the past few years
  - significant improvement in performance
  - successfully applied to several challenging analyses and led to very competitive results
- The integration of DL frameworks into CMSSW is often a challenging task
  - multi-threading schemes
  - resource constraints (CPU time/memory)
- ML inference starts to become a sizable fraction of the event processing time
  - crucial to investigate how to accelerate ML inference
    - e.g., new frameworks, new DNN architectures, new hardware, etc…

Backups

DEEPAK8: ARCHITECTURE

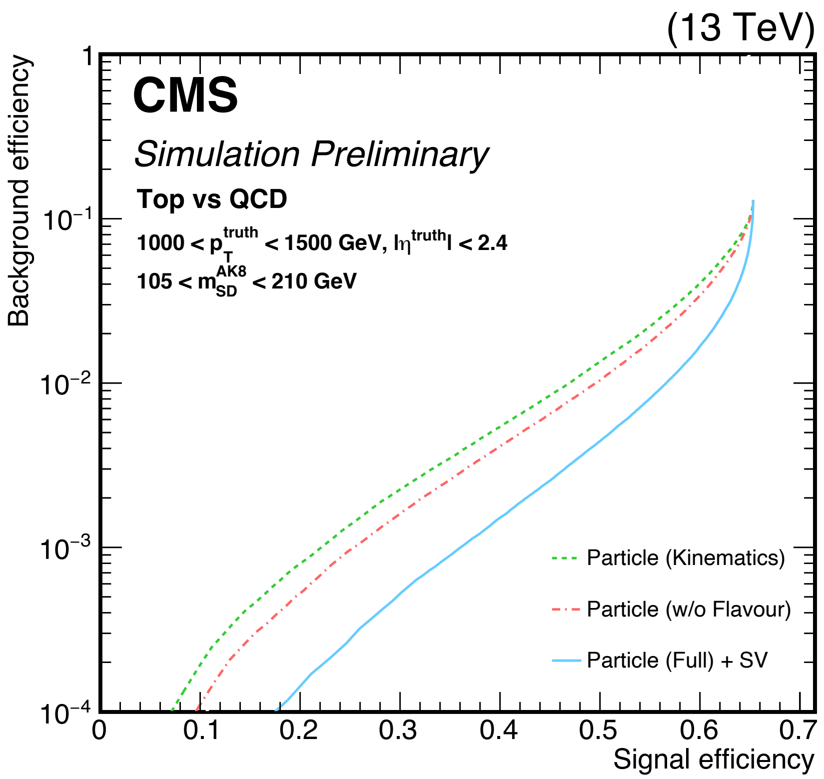


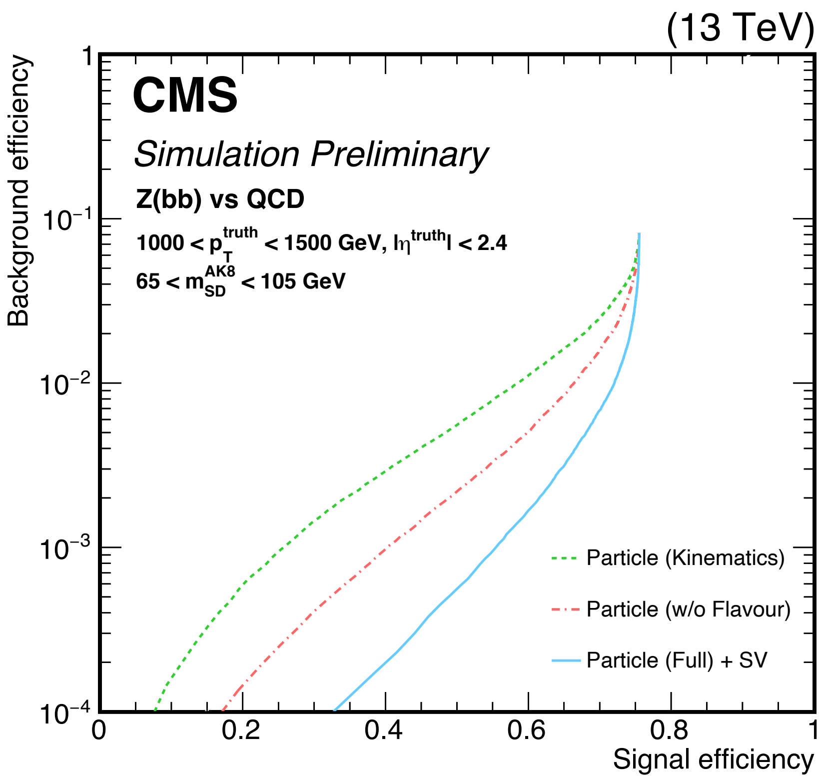
(a) (b)

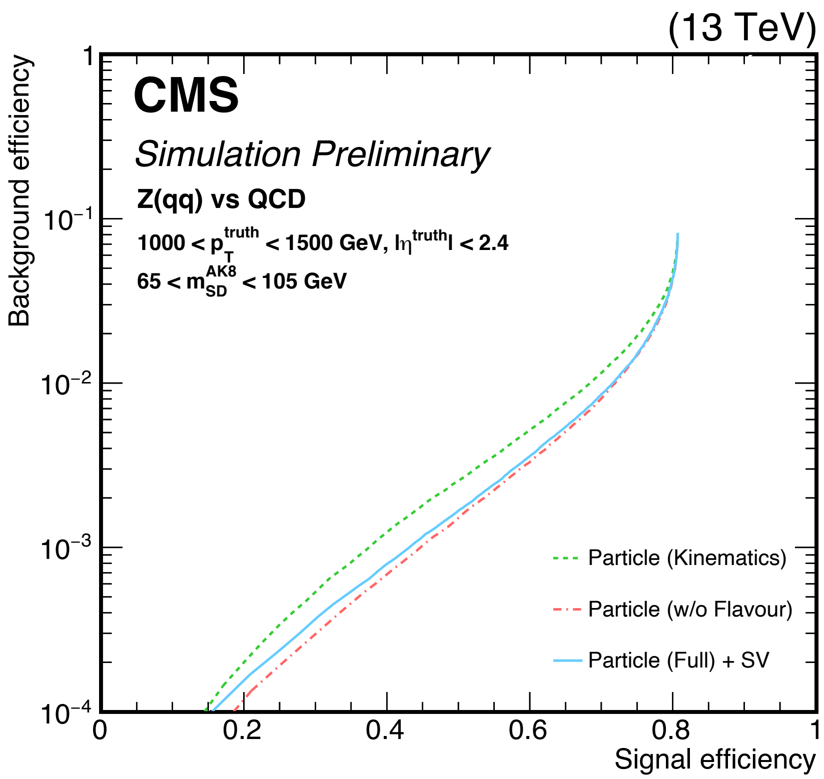
CMS-PAS-JME-18-002

Ablation Study of DeepAK8

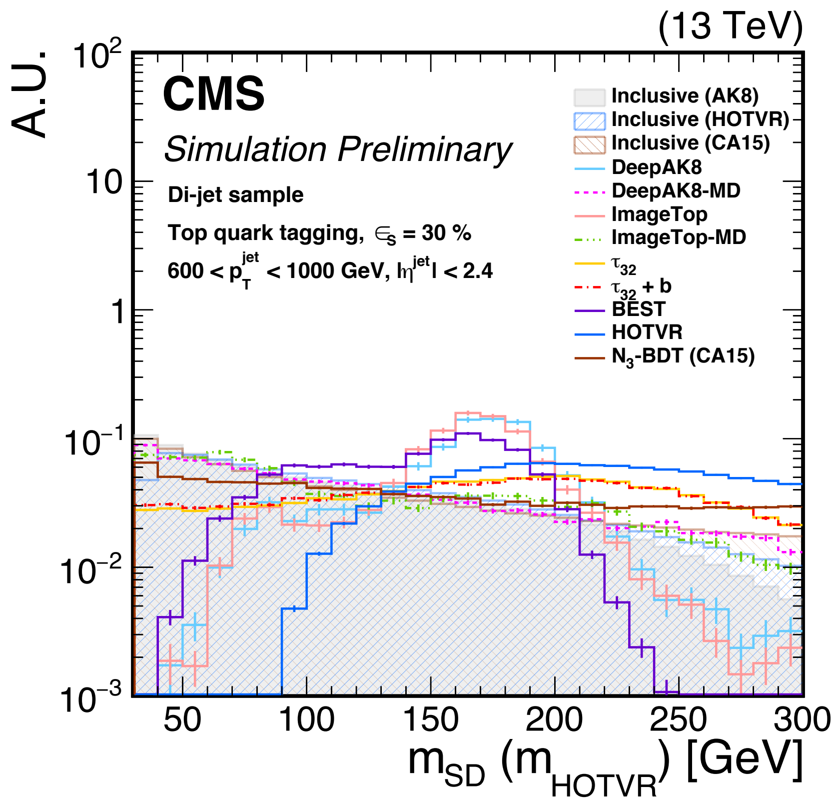
- DeepAK8 shows substantial gain compared to traditional approaches [CMS-PAS-JME-18-002]
- To understand the main sources of the improvement, alternative versions of DeepAK8 were trained using a subset of the input features
  - Particle (kinematics): only kinematic info of PF candidates
    - four momenta, distances to the jet and subjet axes, etc.
  - Particle (w/o Flavour): adding experimental info
    - charge, particle identification, track quality, etc.
  - Particle Full + SV (the full DeepAK8): adding features related to heavy-flavour tagging
    - track displacement, track-vertex association, SV features, etc.

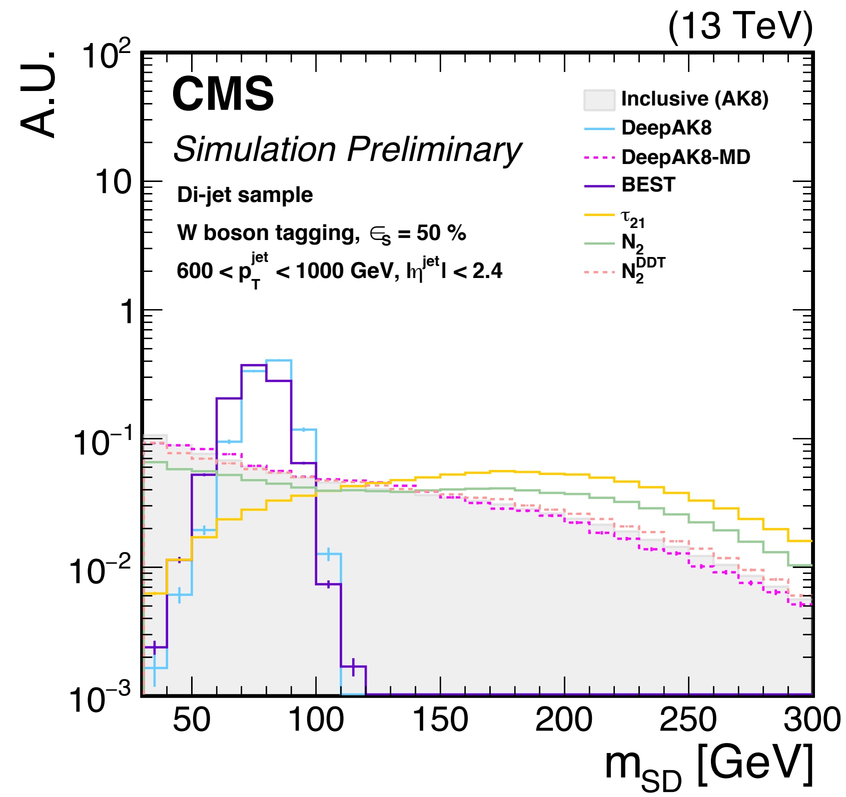


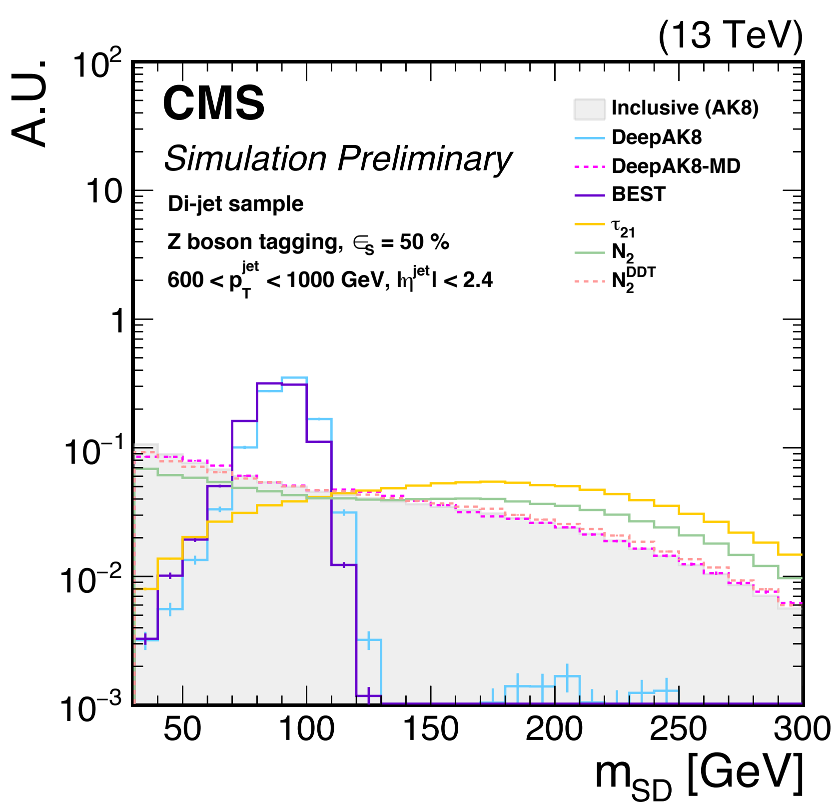


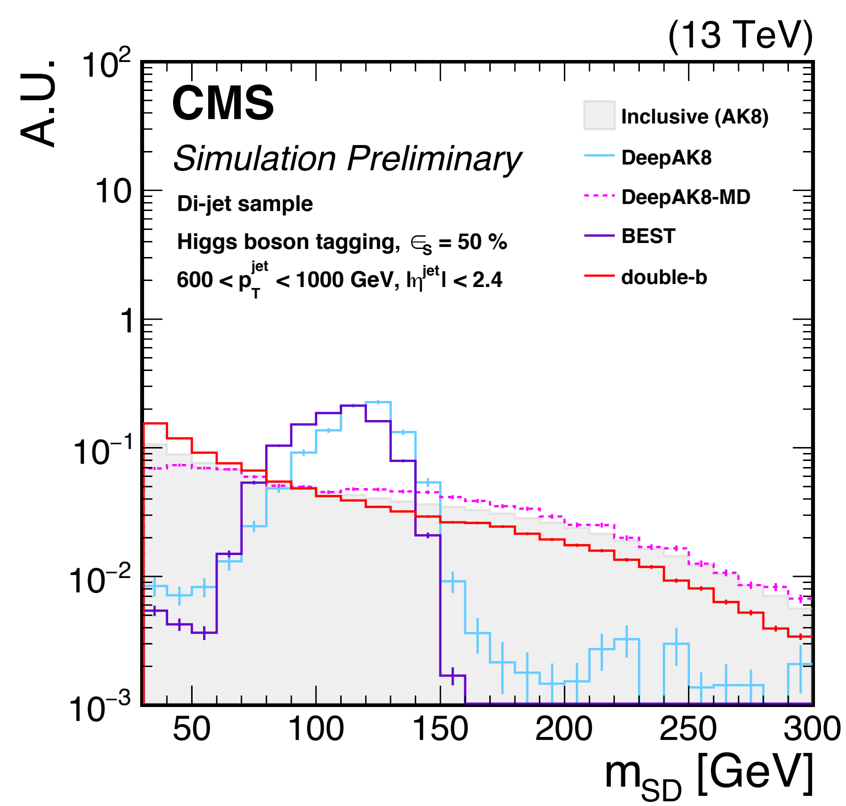


Mass Decorrelation









CMS-PAS-JME-18-002

DeepTau: Algorithm

- Input variables
  - **1 global event variable**: the average energy deposition density (ρ)
  - **42 high-level variables** that are used during tau reconstruction or proven to provide discriminating power by previous tau POG studies
  - For each candidate reconstructed within the tau signal or isolation cones, information about 4-momentum, track quality, relation with the primary vertex, calorimeter clusters, and muon stations is used, if available:
    - From **7 to 27 variables** (depending on the candidate type) for each **particle flow candidate**
    - **37 variables** for each fully reconstructed **electron candidate**
    - **37 variables** for each fully reconstructed **muon candidate**
- Candidates belonging to the inner and outer cones are separated and split into two grids with $\eta \times \varphi$ cell size of 0.02×0.02 (0.05×0.05) for the inner (outer) cone
- Network architecture:
  - High level variables and each input cell are pre-processed by a few fully connected dense layers
  - For the inner (outer) grid, the pre-processed cell data are fed into 5 (10) 2D convolutional layers with 3×3 window size, which result in 64 features that are passed to the next step
  - All features from previous steps are combined and passed through 5 dense layers
  - Probabilities of the reconstructed tau candidate being electron, muon, quark or gluon jet, or hadronic tau are estimated by the 4 NN outputs

CMS-DP-2019-033

DeepTau: Performance in Data

Distribution of the visible $\mu\tau$ mass for 2018 data

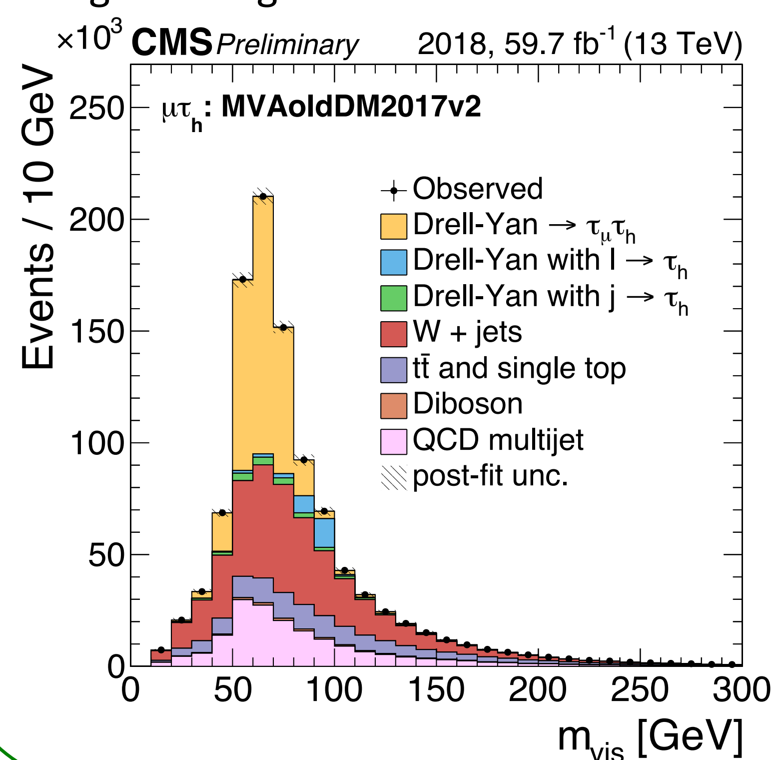
Event selection:

- well identified and isolated muon with $p_T > 25$ GeV, $|\eta| < 2.4$, $|dz| < 0.2$ cm
- tau candidates with $p_T > 20$ GeV, $|\eta| < 2.3$, $|dz| < 0.2$ cm
- $\mu\tau$ pair with an opposite charge and $\Delta R(\mu, \tau) > 0.5$

- Contribution from all SM processes (except QCD) are modelled by MC simulation
- QCD estimated from a sideband region in data

Selection using discriminators from JINST 13 (2018) P10005:

- Tight WP against jets
- VLoose WP against electrons
- Tight WP against muons

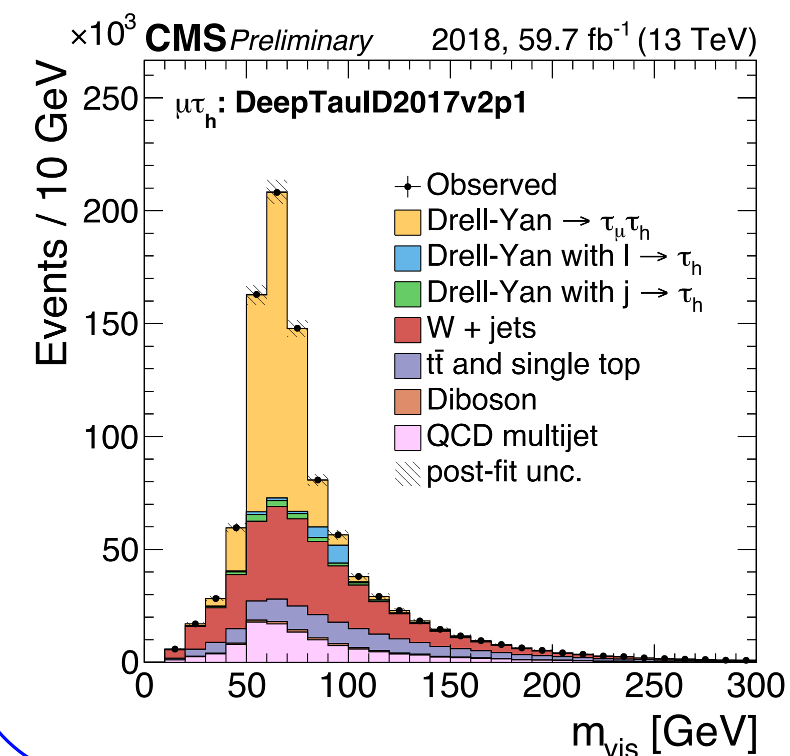


With DeepTau selection, the yield from **genuine τ_h** <u>increases by ≈ 20%</u>, while yield from **fakes** <u>decreases by ≈ 23%</u>

In both plots modelled contributions are fit to the data

Selection using DeepTau IDs:

- Tight WP against jets
- VVLoose WP against electrons
- VLoose against muons



CMS-DP-2019-033