Investigation of robustness of b-Tagging algorithms for the CMS Experiment

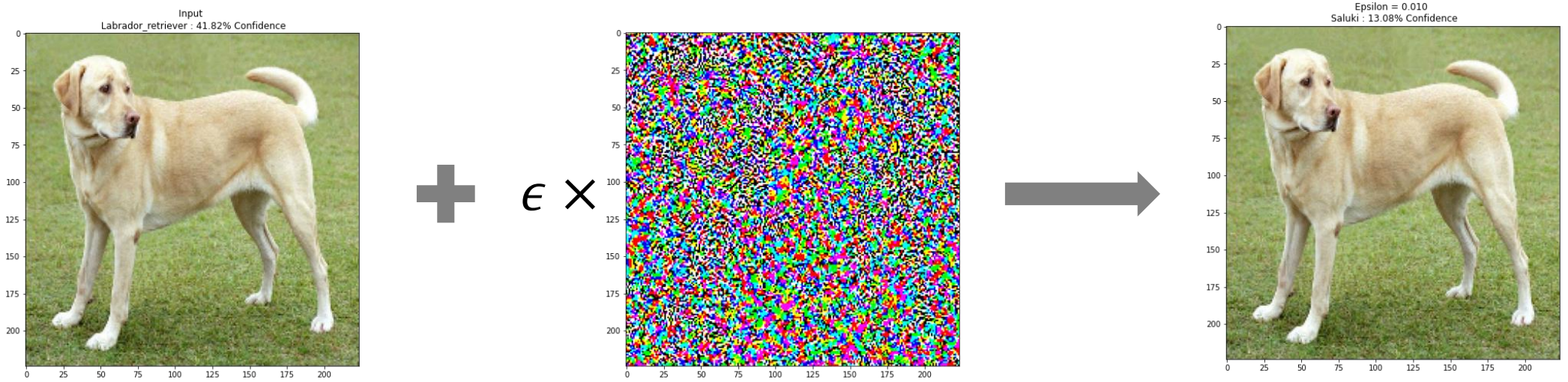
Annika Stein

Master's Defense Colloquium
22.10.2021

Outline

- Introduction AI safety: Application to Neural Networks & CMS b-Tagging
- Adversarial Attacks: Fast Gradient Sign Method
- Improving Robustness with Adversarial Training

AI safety: example for image classification



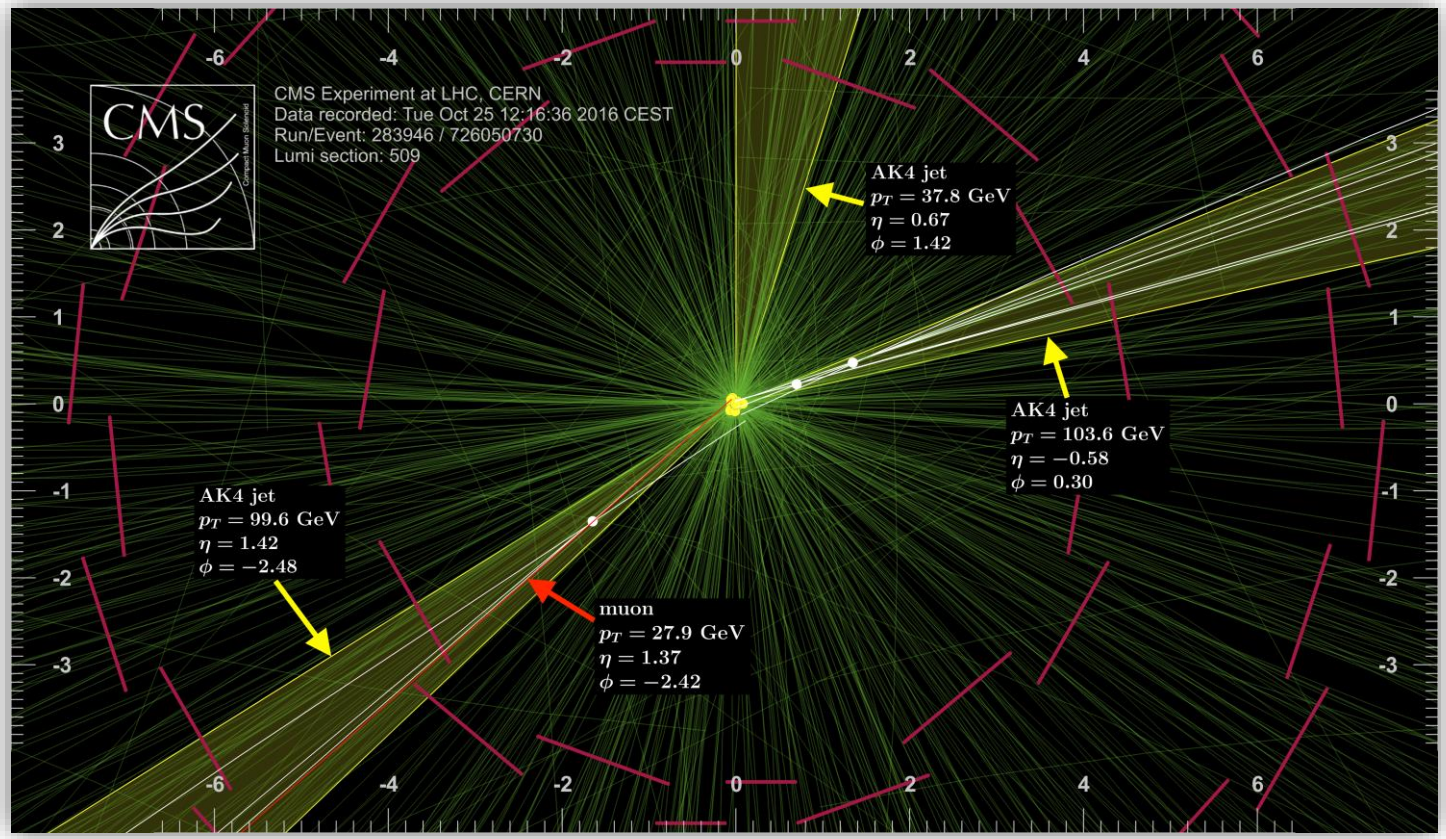
Classifier: labrador (breed of dog)

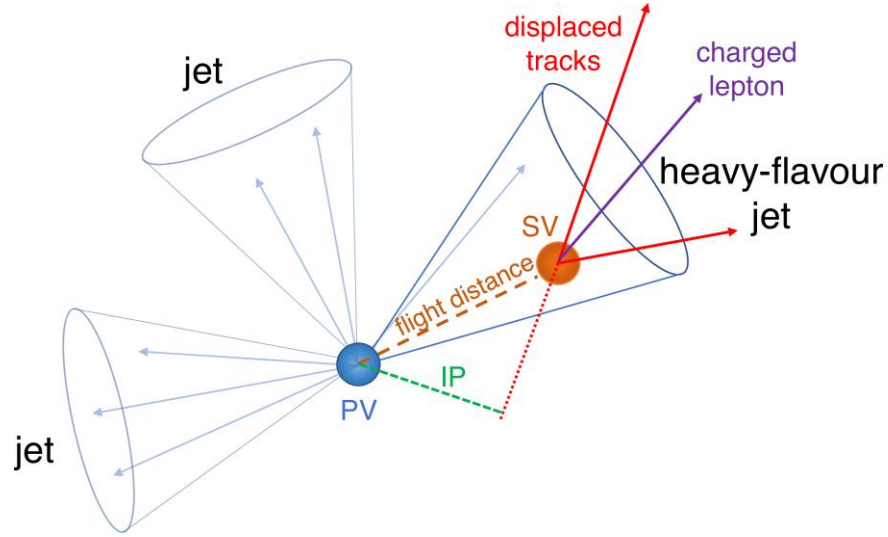
Classifier: saluki (breed of dog)
german: „Windhund“

→ Generate adversarial samples with perturbations that are not too easy to identify
→ Check their influence on the model performance

Application: jet heavy-flavour tagging at CMS

- Goal: identify the flavour of the parton (hadron) from which the jet originates





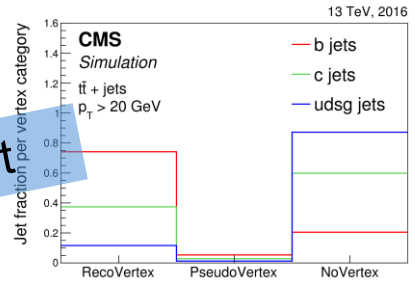
Heavy-flavour jets (b & c jets)

- Long lifetime of b/c hadrons → secondary vertex & displaced tracks
- Larger mass, harder fragmentation compared to light jets
- (Soft) charged lepton in 20% (10%) of the cases for b (c) jets

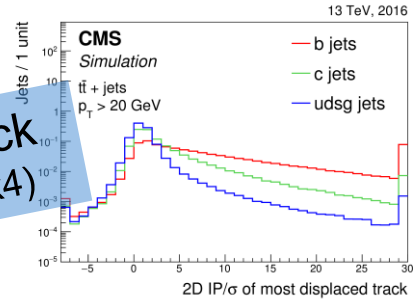
b-Tagging algorithms for CMS: DeepCSV

Inputs

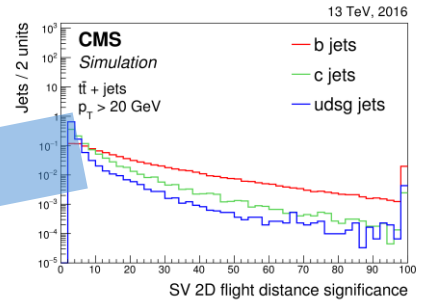
13x Jet



46x Track
(7x6 + 1x4)

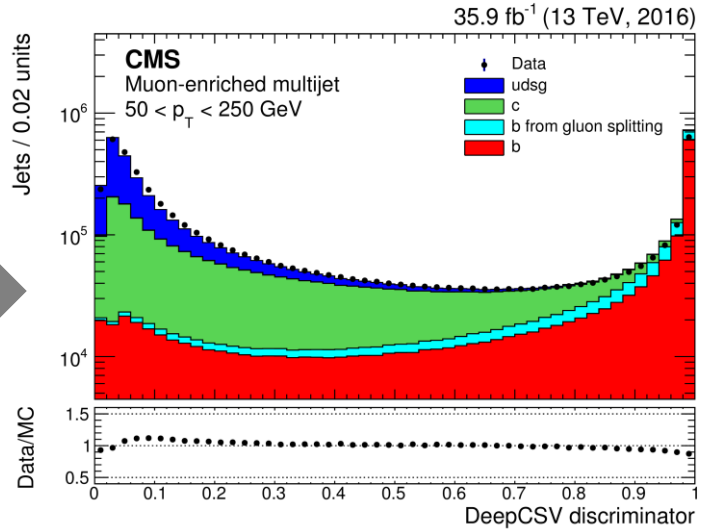


8x SV



Deep Neural Network
(DNN)

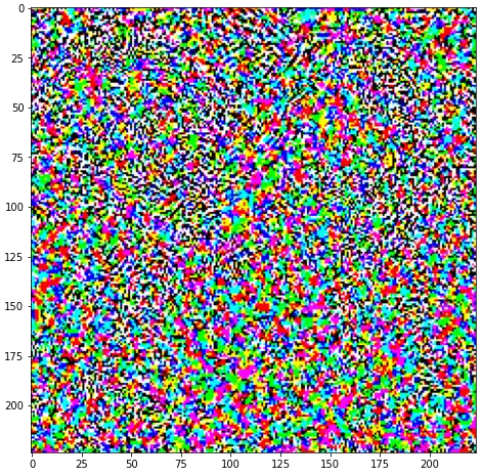
Outputs



AI safety: jet heavy-flavour tagging

Jet,
Track,
Secondary Vertex
properties of a **b jet**

$+ \quad \epsilon \times$



→

Slightly distorted
Jet,
Track,
Secondary Vertex
properties of a **b jet**

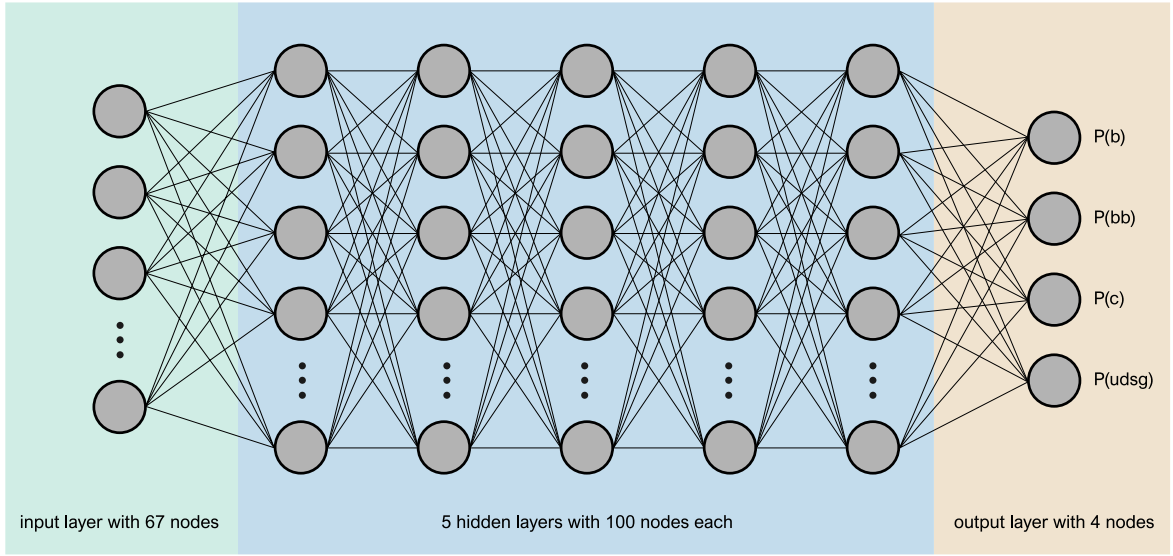
Classifier: b jet

Classifier: light jet

If one pixel alone can fool neural networks for image classification…

…could subtle mismodellings in our simulations cause wrong results in physics analysis?

Replicating DeepCSV model & nominal performance

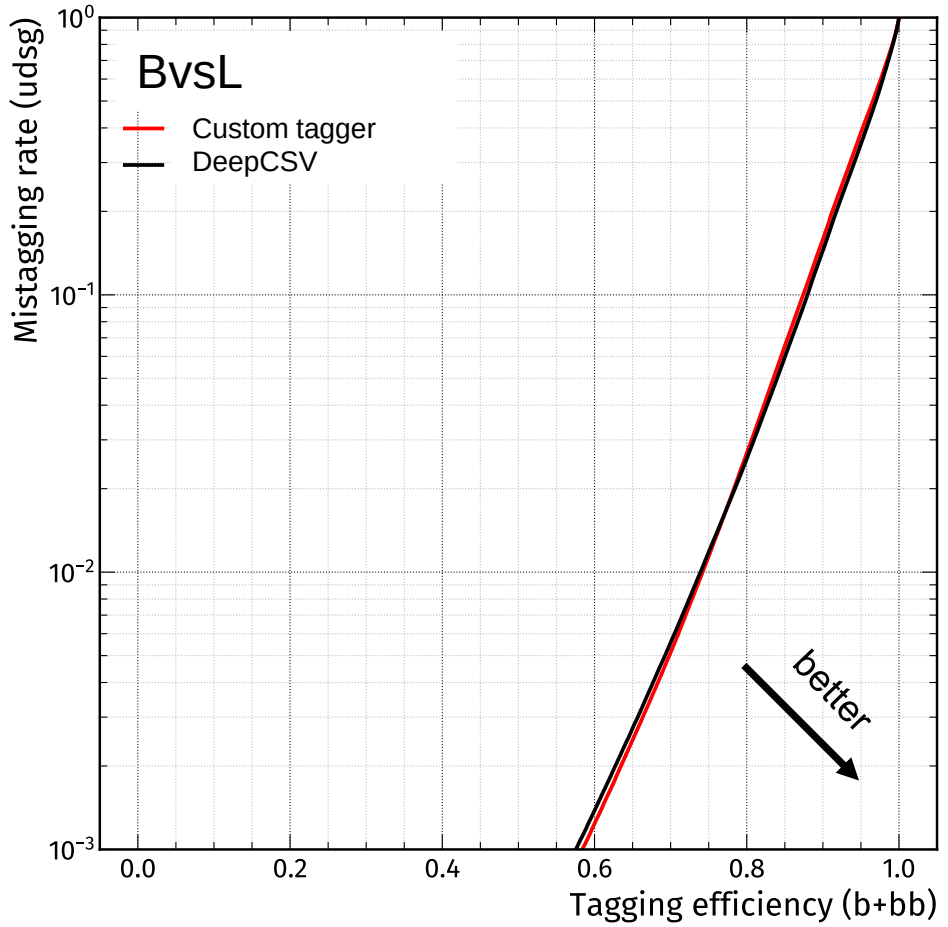


- DNN outputs are used to calculate discriminators, e.g.

$$\mathrm{BvsC} = \frac{\mathrm{P(b)} + \mathrm{P(bb)}}{\mathrm{P(b)} + \mathrm{P(bb)} + \mathrm{P(c)}}$$

$$\mathrm{BvsL} = \frac{\mathrm{P(b)}+\mathrm{P(bb)}}{\mathrm{P(b)}+\mathrm{P(bb)}+\mathrm{P(udsg)}}$$

- Receiver Operating Characteristic (ROC) curves
- Area under the ROC curve (AUC)



Part 2

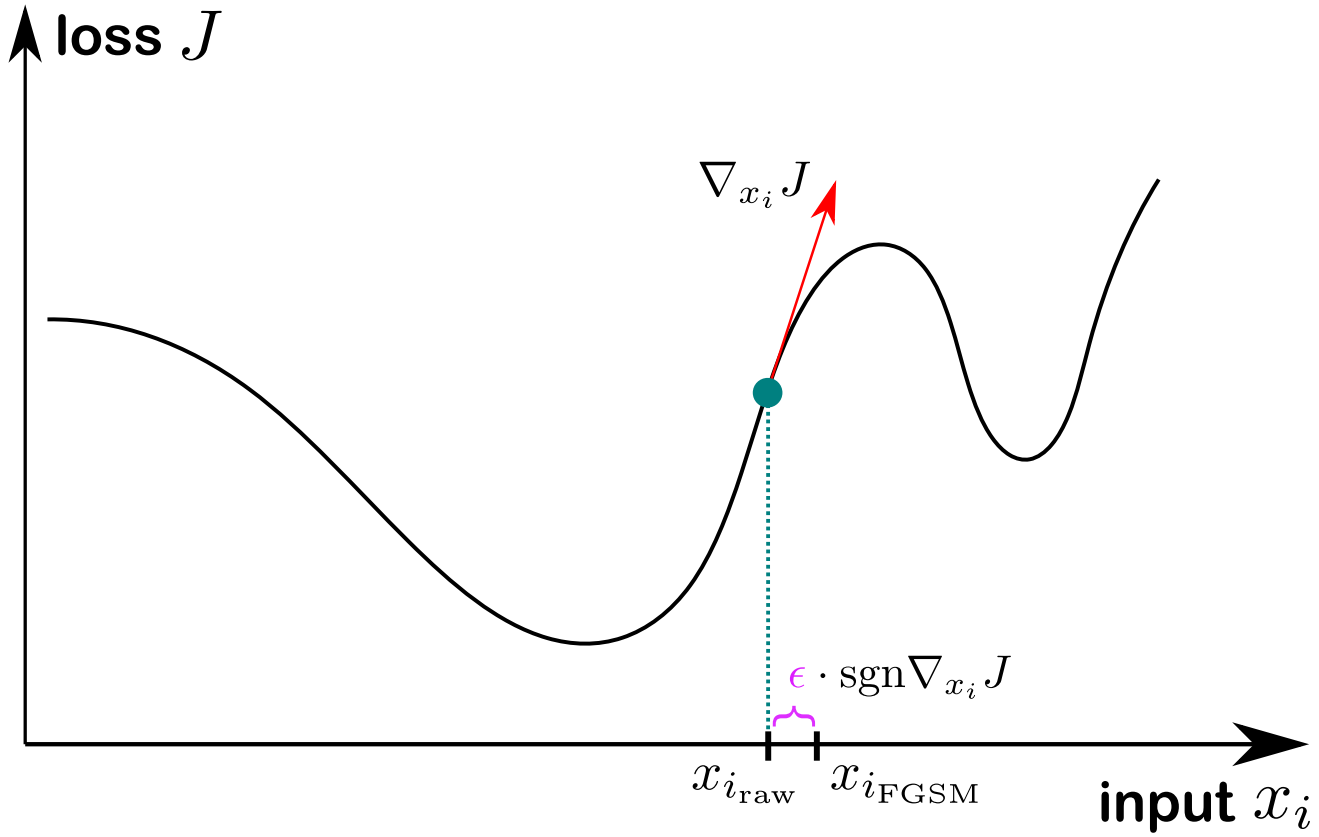
- Introduction AI safety: Application to Neural Networks & CMS b-Tagging
- Adversarial Attacks: Fast Gradient Sign Method
- Improving Robustness with Adversarial Training

Fast Gradient Sign Method (FGSM)

- **Maximize** the **loss function** with respect to the **inputs** to disturb the inputs systematically:

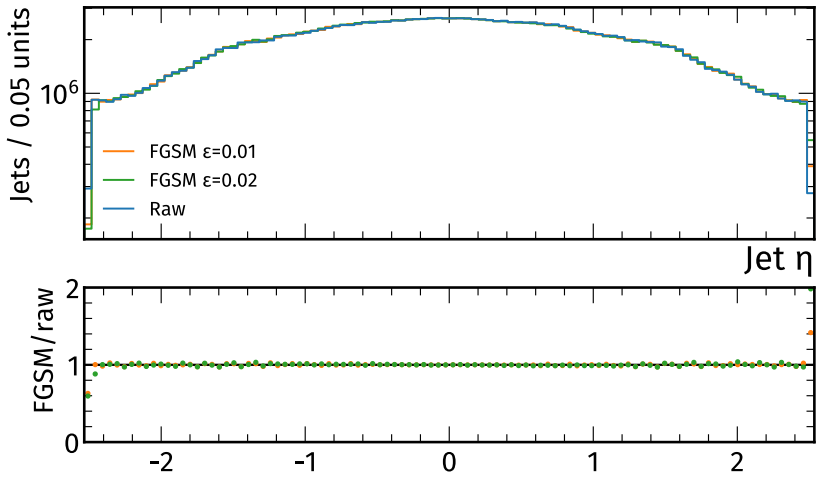
$$x_{FGSM} = x_{raw} + \epsilon \cdot \mathrm{sgn}(\nabla_{x_{raw}} J(y, x_{raw}))$$

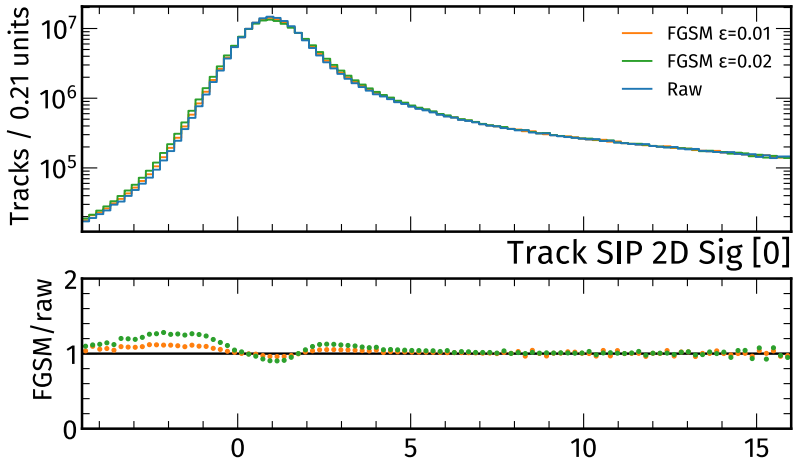
- Small ϵ are enough to affect the performance drastically

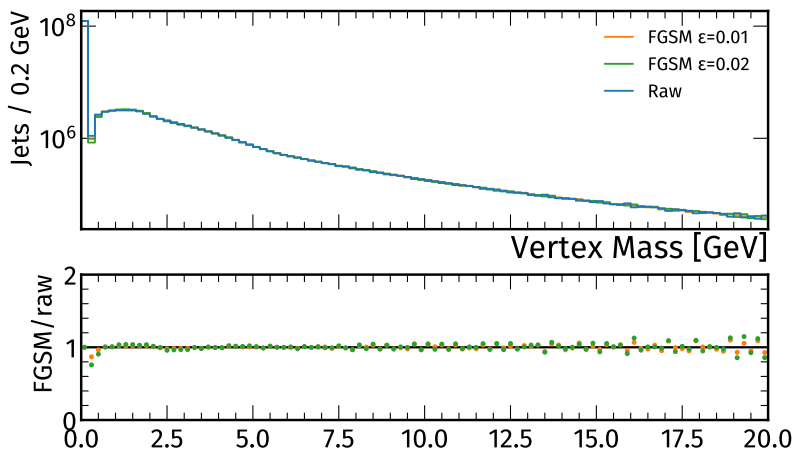


Distorted Inputs (FGSM)

- Apply FGSM attack to test sample







- Current setup uses the same ϵ for all features (excluding integer variables & excluding default values)

→ **Visibility** of the distortion **depends** on the chosen **variable**, displayed range, flatness / smoothness of the distribution, discriminating power

- Defining meaningful upper bounds per feature is necessary, otherwise distortions can become unphysical

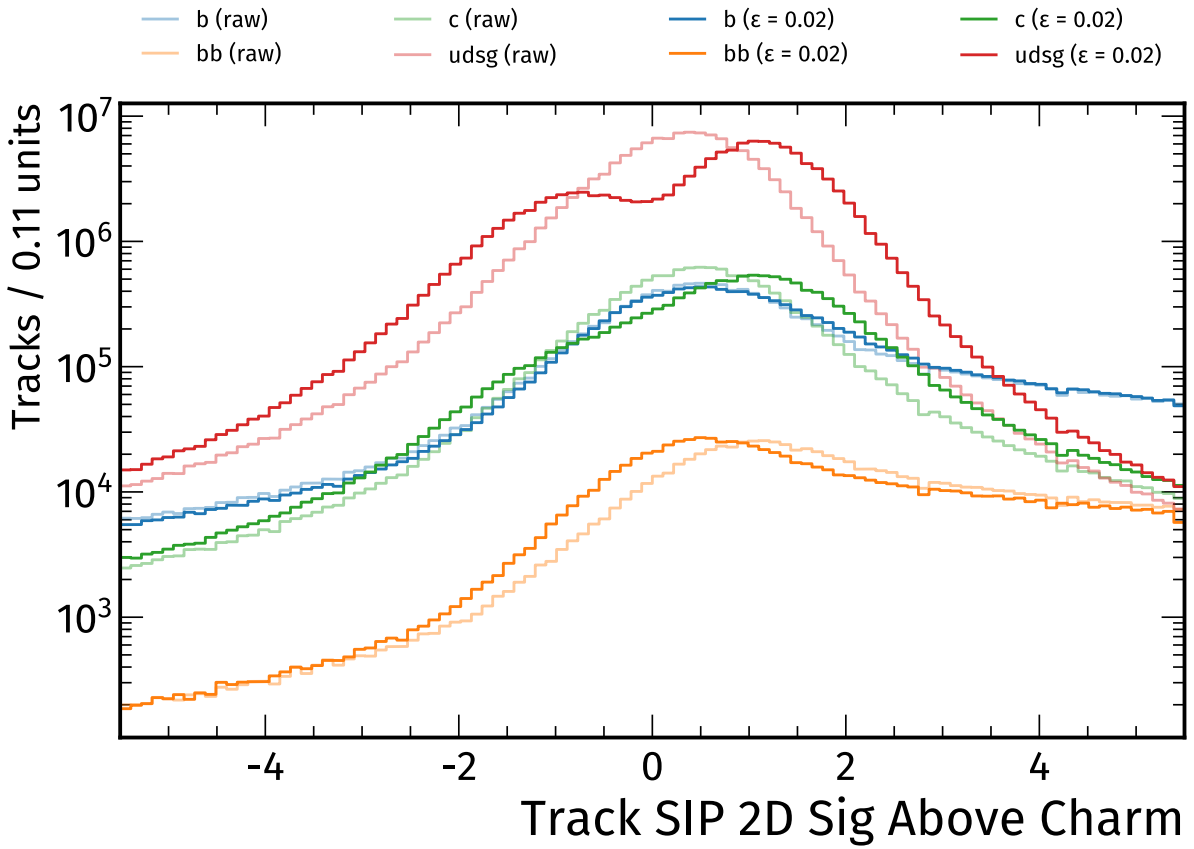
We leave this for future studies, concentrate on two parameters for now:

moderate $\epsilon = 0.01$ → for most comparisons, also used for adversarial training

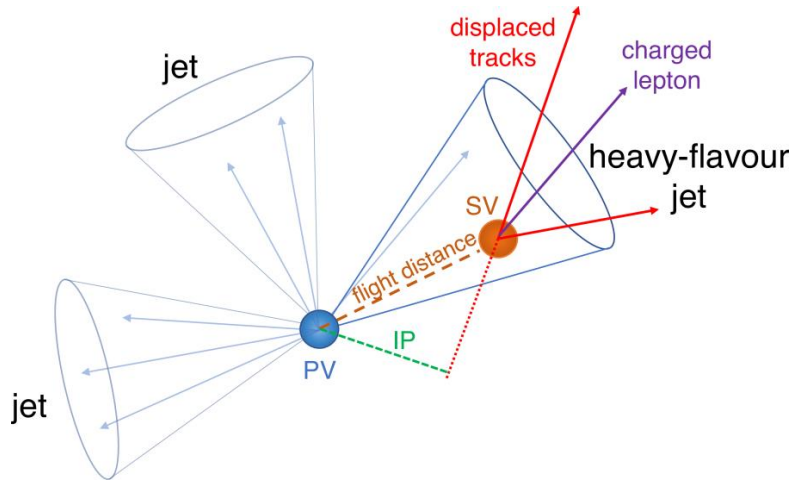
larger $\epsilon = 0.02$ → to visualize flavour dependency

Distorted Inputs (FGSM) – revisited

- So what does the attack actually do, regarding physical observables?



<u>Signed Impact Parameter (Significance)</u>
Originally, heavy-flavour jets are more abundant in the positive region, whereas the distribution for light jets should be approximately symmetric

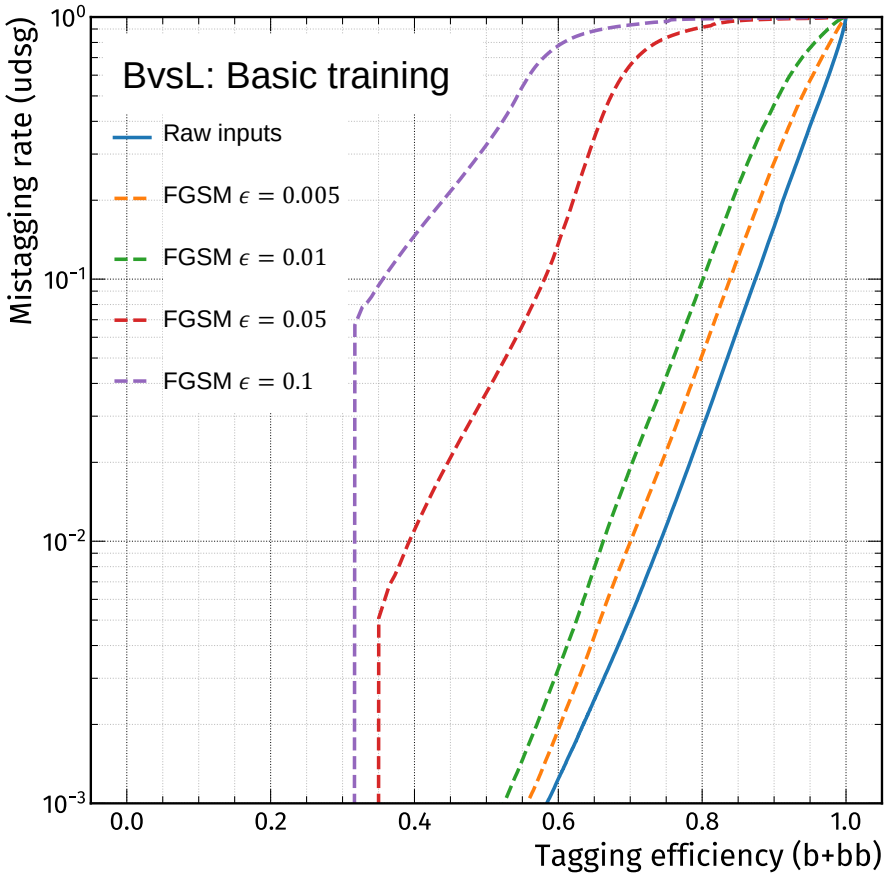


- b(b) jets move to the negative end, light jets more to the positive side (c jets ≈ in between)

→ Adversarial attack “inverts” physics, although noone *explicitly* told it to do so or specified *how!*

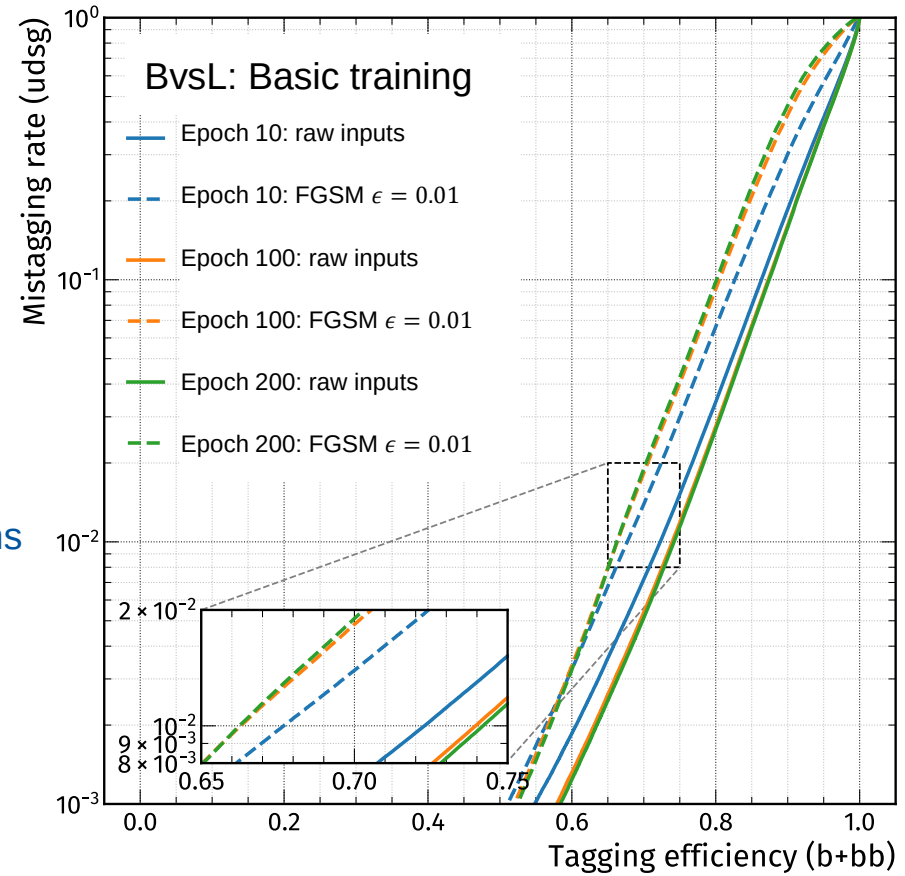
ROC curves (B versus L)

- Evaluate model on raw and distorted inputs, use ROC curves to investigate possible changes of performance



Same epoch, different ϵ

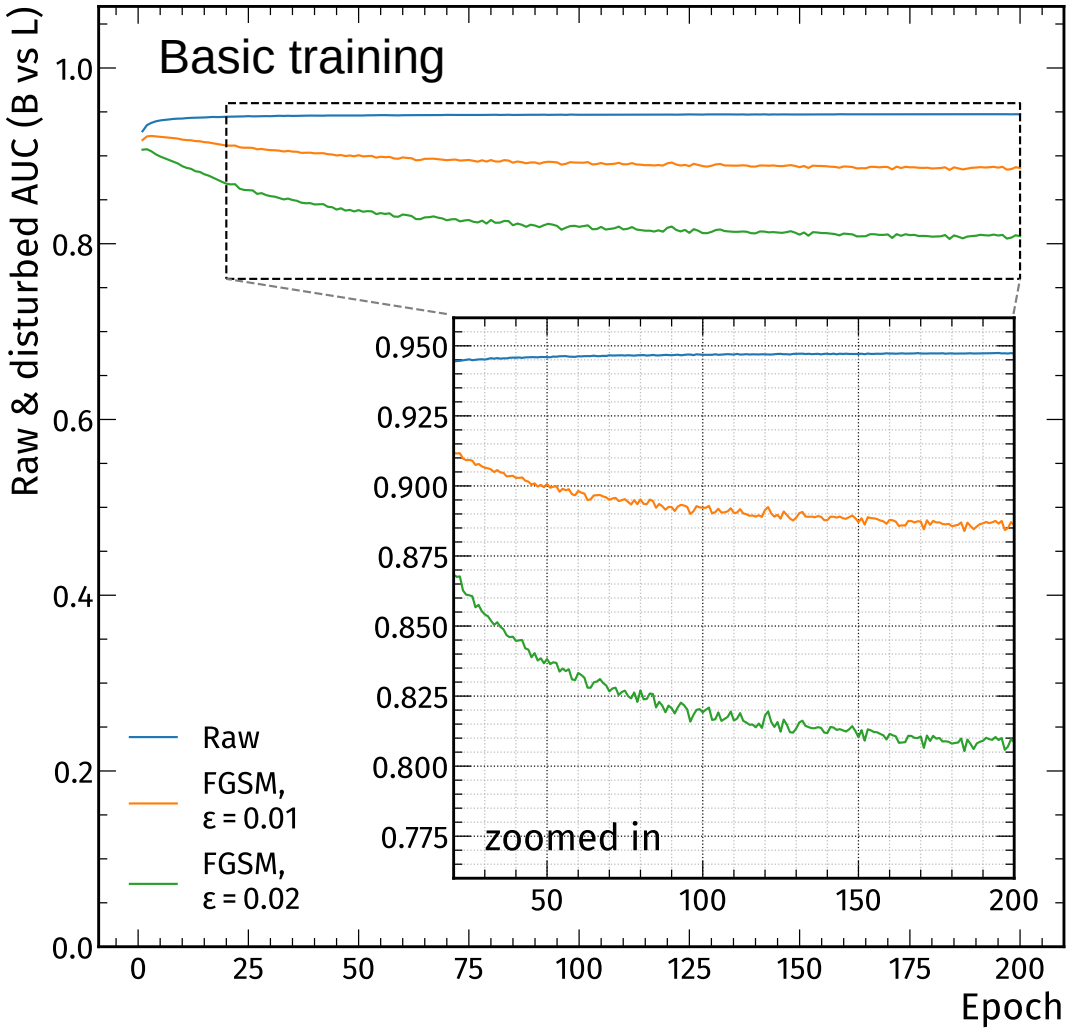
Same ϵ, different epochs



- Larger parameters for ϵ have higher impact on performance (as expected)
- Order of ROC curves for different epochs on distorted inputs is flipped w.r.t. raw inputs

Evolution of AUC with number of epochs (Basic Training + FGSM)

- Save model checkpoint after every epoch, run inference and compute AUC for the BvsL discriminator



- Performance on raw inputs increases slowly
- Impact of FGSM attack becomes more severe

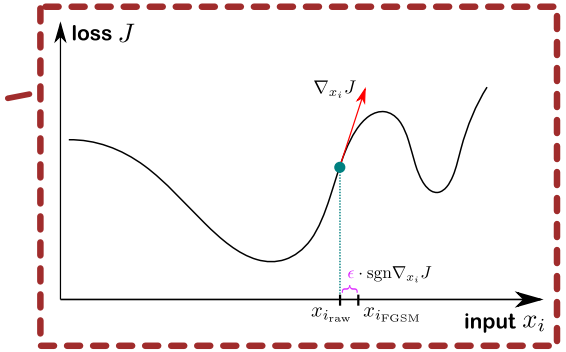
Tradeoff between performance and robustness!

Part 3

- Introduction AI safety: Application to Neural Networks & CMS b-Tagging
- Adversarial Attacks: Fast Gradient Sign Method
- Improving Robustness with Adversarial Training

Adversarial Training

FOR N EPOCHS:
    SPLIT WHOLE TRAINING SAMPLE INTO MINIBATCHES
    FOR EVERY MINIBATCH:
        DISTORT INPUTS (= APPLY FGSM)
        EVALUATE MODEL (FORWARD)
        COMPUTE LOSS (AND APPLY LOSS WEIGHTING)
        ACCUMULATE GRADIENTS OF LOSS (BACKWARD)
        UPDATE MODEL PARAMETERS

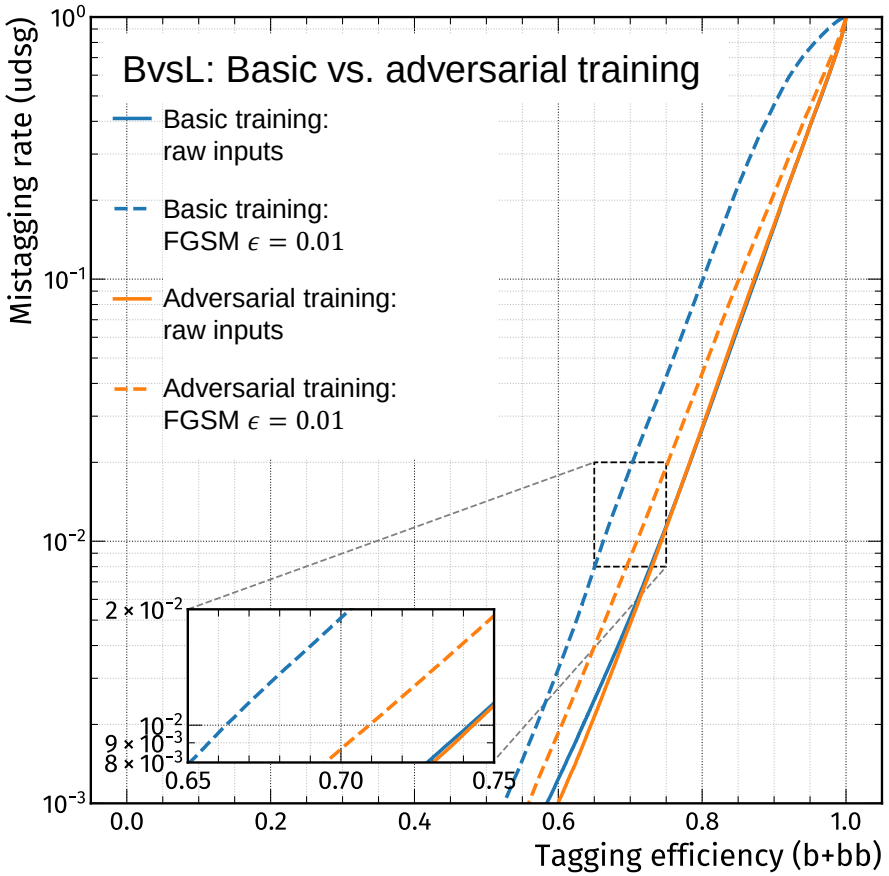


- Inject distorted inputs already during training phase
- Idea: model never sees raw inputs, so it should less likely learn simulation-specific artefacts

→ Improve robustness
Better generalization

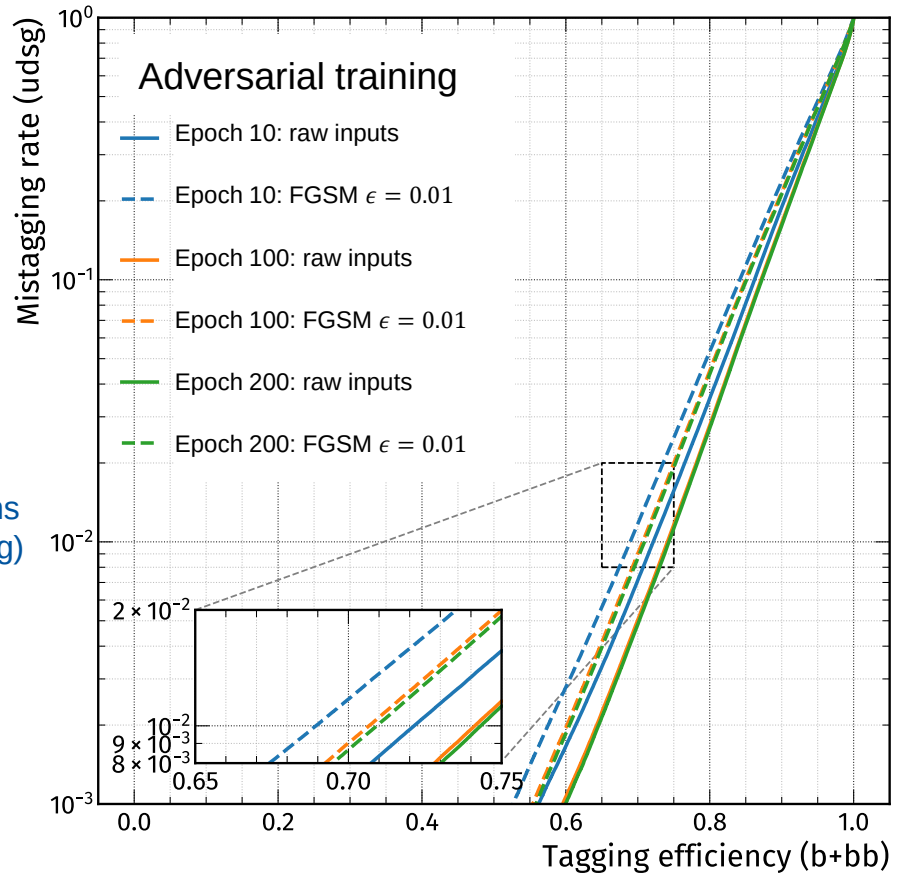
ROC curves (B versus L)

- Evaluate model on raw and distorted inputs, use ROC curves to investigate possible changes of performance



Same epoch and ϵ, comparing basic with adversarial training

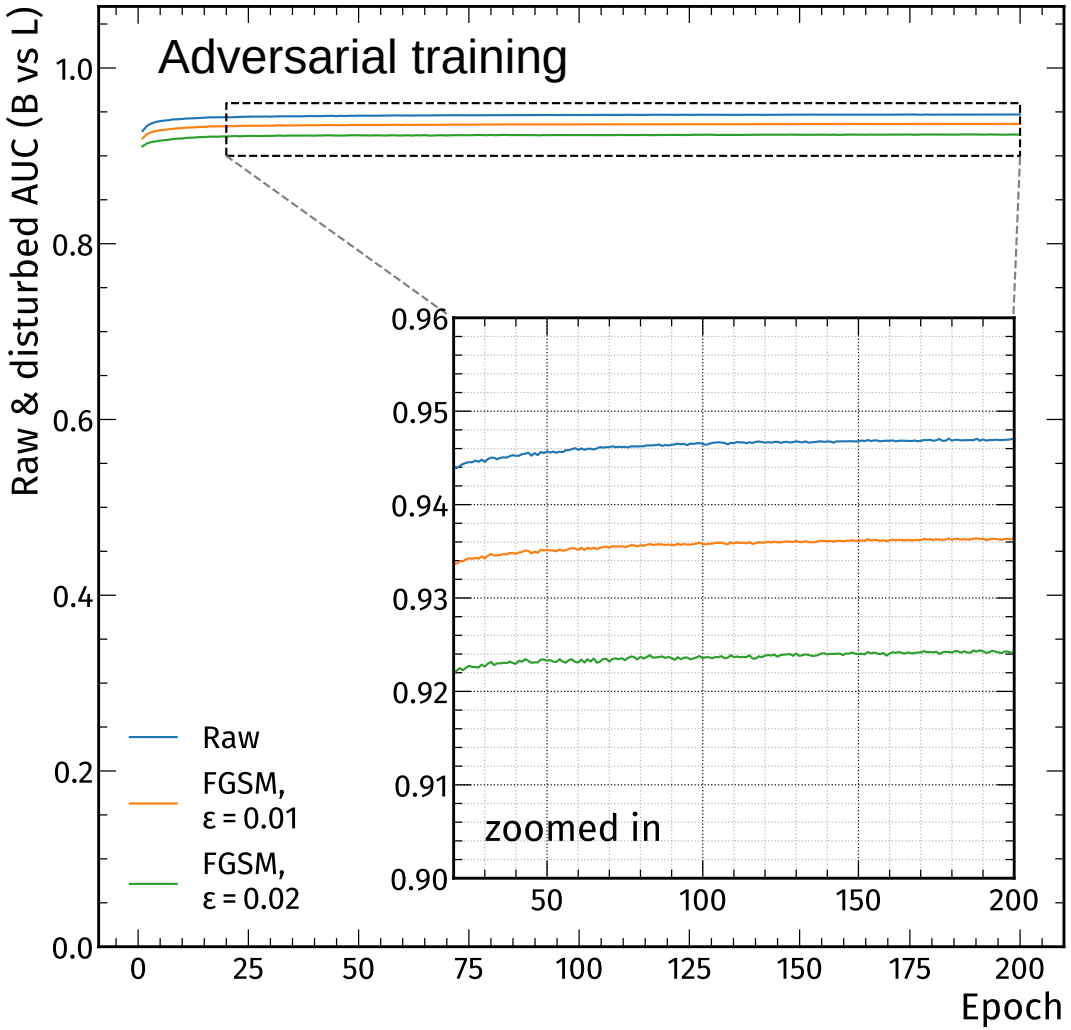
Same ϵ, different epochs (only adversarial training)



- FGSM affects basic training much more than adversarial training, with ≈ equal nominal performance!
- Order of ROC curves for different epochs of adversarial training is not affected by FGSM

Evolution of AUC with number of epochs (Basic Training + FGSM)

- Save model checkpoint after every epoch, run inference and compute AUC for the BvsL discriminator

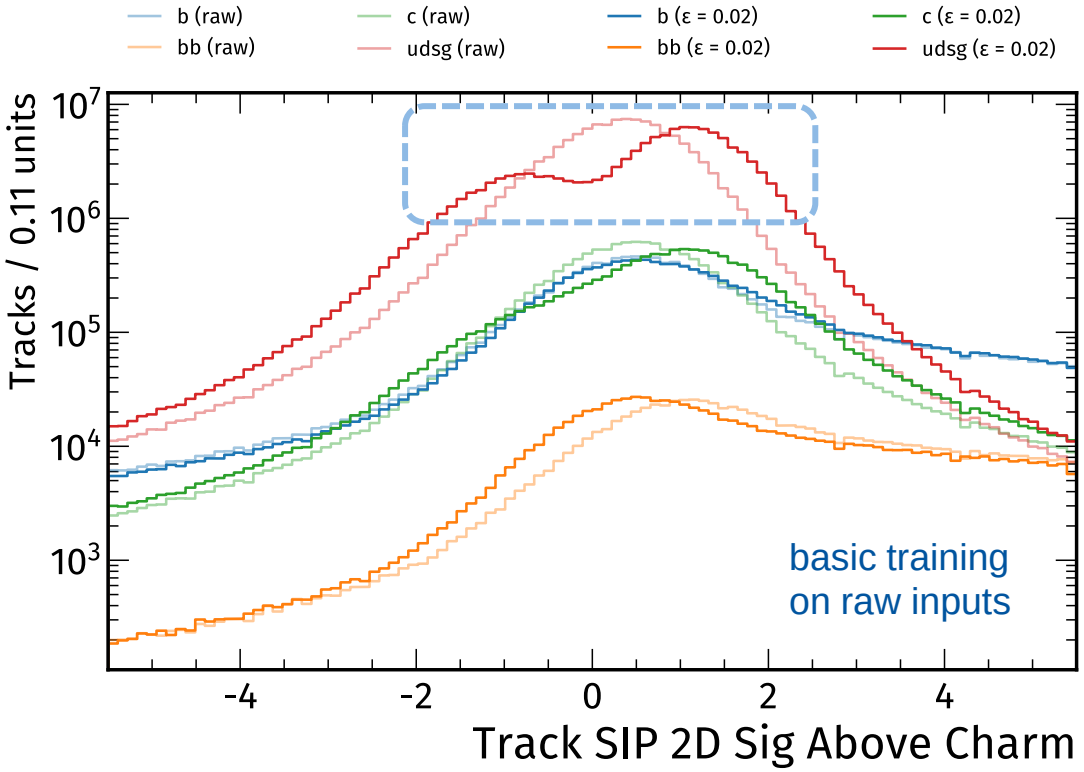


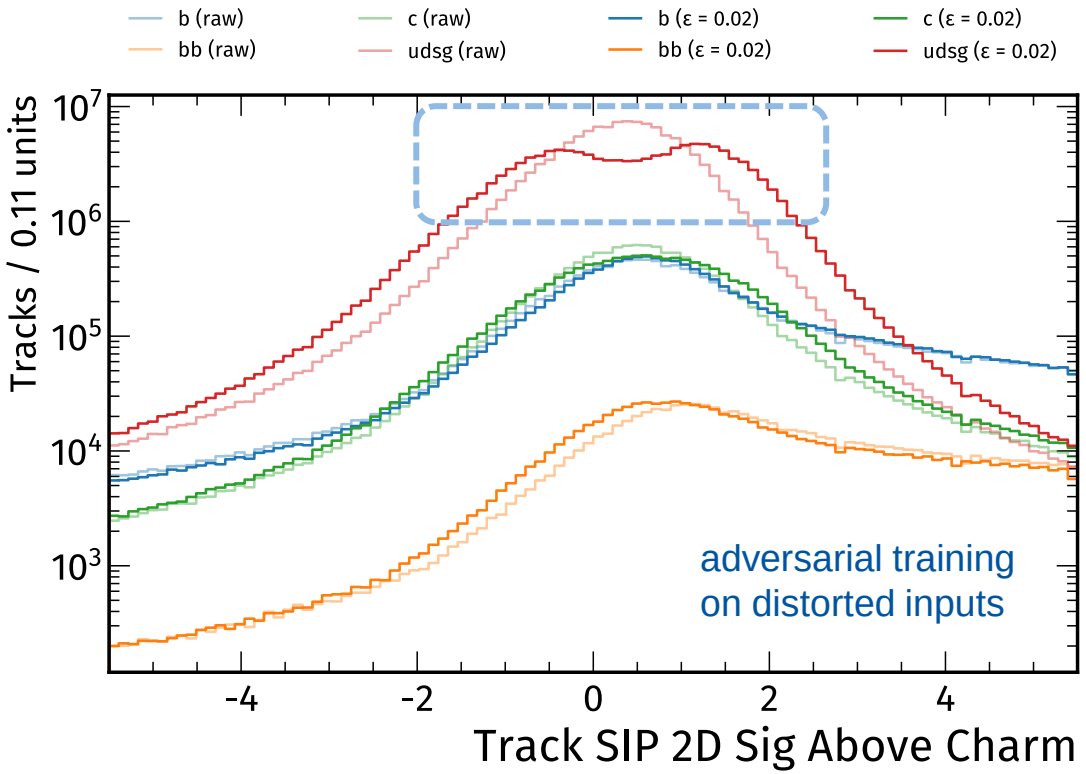
- Performance on raw inputs increases slowly
- Impact of FGSM attack stays the same

→ Only a **constant offset** between raw and FGSM!

Distorted Inputs (FGSM) – revisited 2.0

- Comparing the flavour-dependent impact of FGSM attacks for basic and adversarial training:



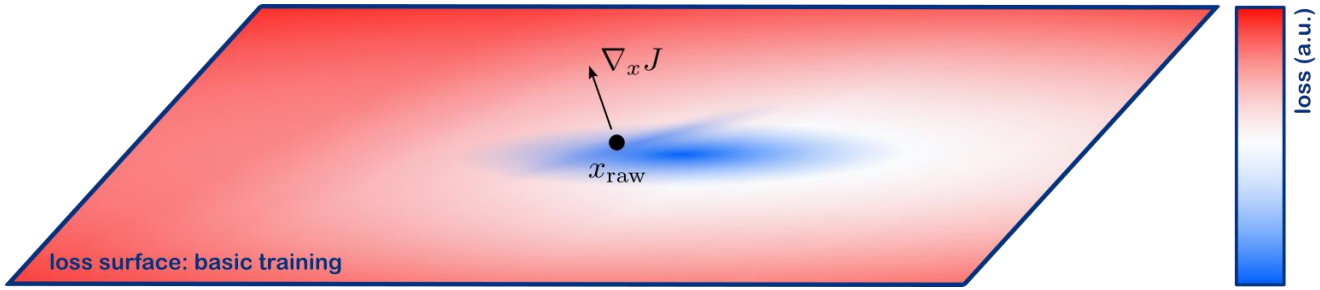


<u>A geometrical problem</u>
Basic training ⊗ FGSM → asymmetric shapes
Adversarial training ⊗ FGSM → symmetric shapes

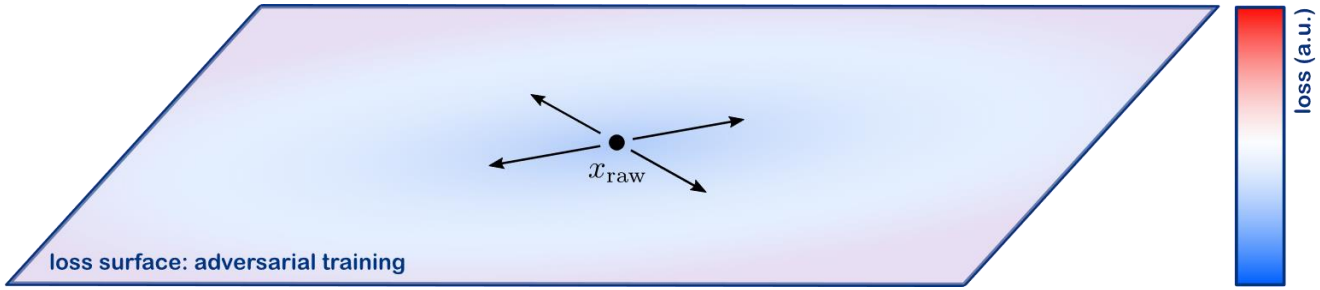
- Crafting adversarial inputs for adversarially trained model is almost like “coin-flipping”

Possible explanation of smaller susceptibility

- An attempt to understand why the adversarially trained model generalizes better / is more robust:



- Clear direction for first-order worst-case adversarial inputs for the basic training due to geometry of the loss surface



- With the assumption of a flat loss surface, there is no preferred direction for adversarial examples

Adversarially trained model is expected to be less vulnerable to mismodellings in simulation

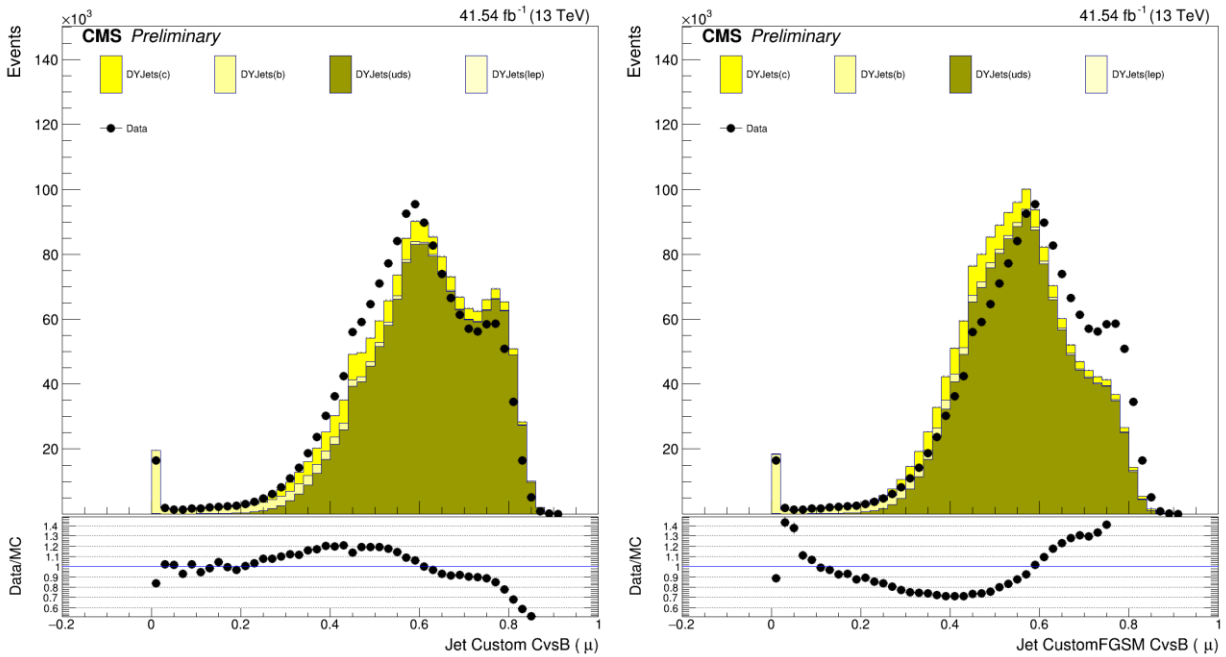
*visualizations are “handcrafted”, not the *actual* loss manifolds

Data/MC agreement: first studies with the custom tagger / adversarial training / adversarial attacks

- Example: muon channel of the Drell-Yan+jets selection (**light jet** enriched); CvsB discriminator

Slightly better raw / out-of-the-box agreement

Basic training

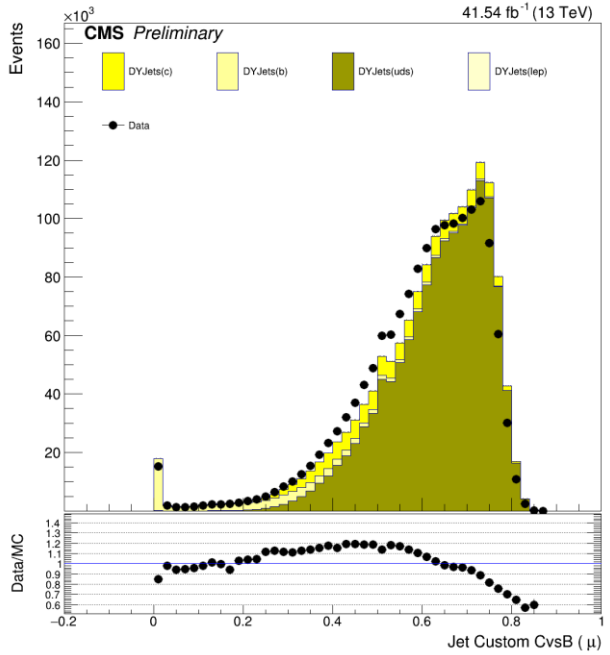


on raw inputs

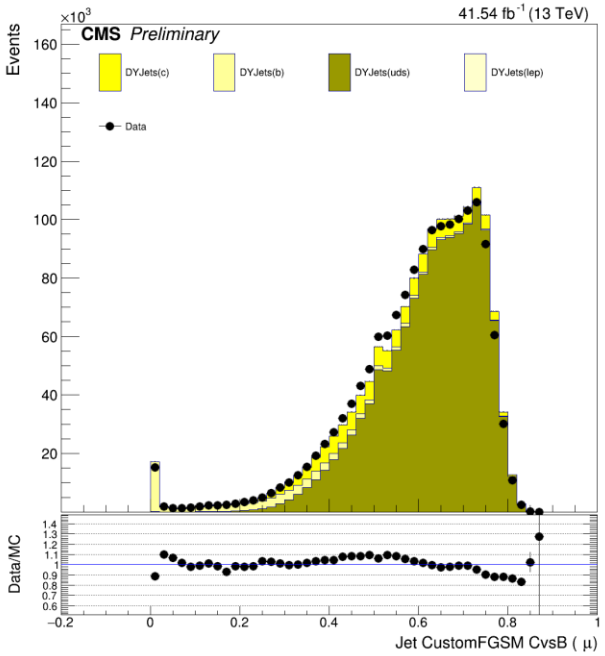
on systematically distorted inputs

Apply FGSM with $\epsilon = 0.01$ on simulation
→ already overcorrects

Adversarial training



on raw inputs



on systematically distorted inputs

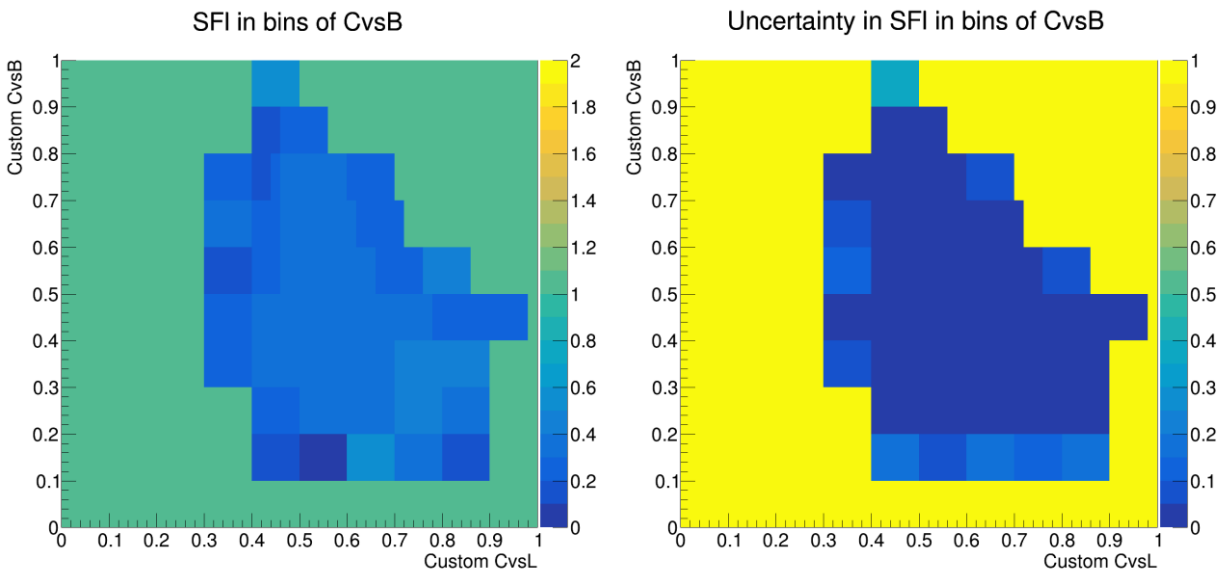
Apply FGSM with $\epsilon = 0.01$ on simulation
→ “just right”

*MC normalized to data

Scale factors: first studies with the custom tagger / adversarial training / adversarial attacks

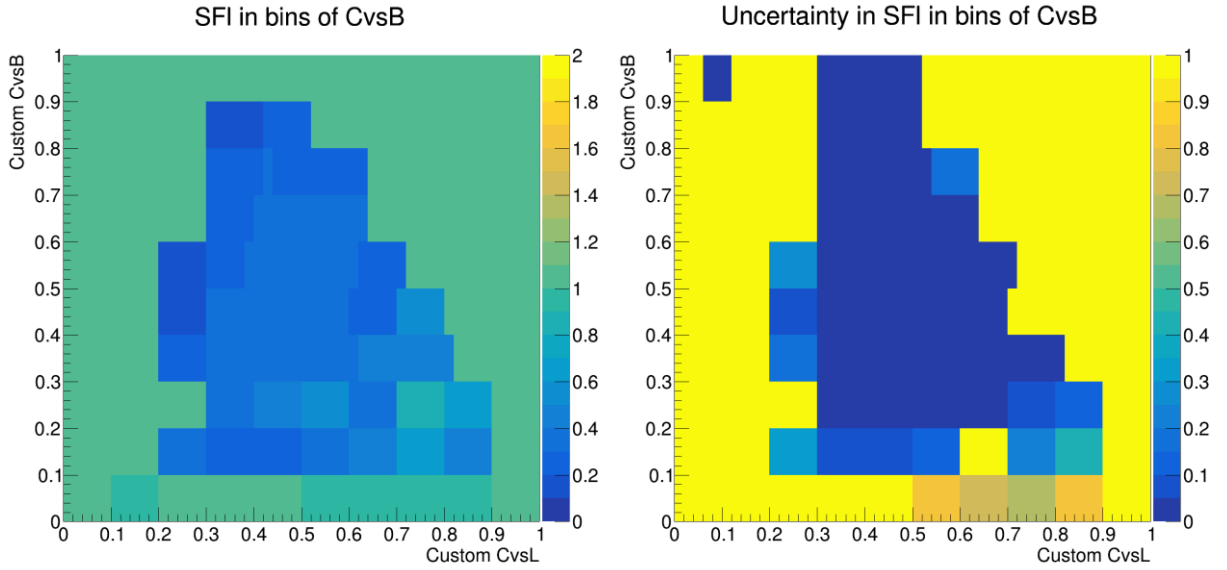
- Example: **light** selection, CvsB fixed binning, CvsL float

Basic training



Scale factors close to 1 only if uncertainty high (= from initialization)

Adversarial training



Scale factors close to 1 also for some bins with small uncertainty

So far, the effect of adversarial training is only marginal, but could be optimized.

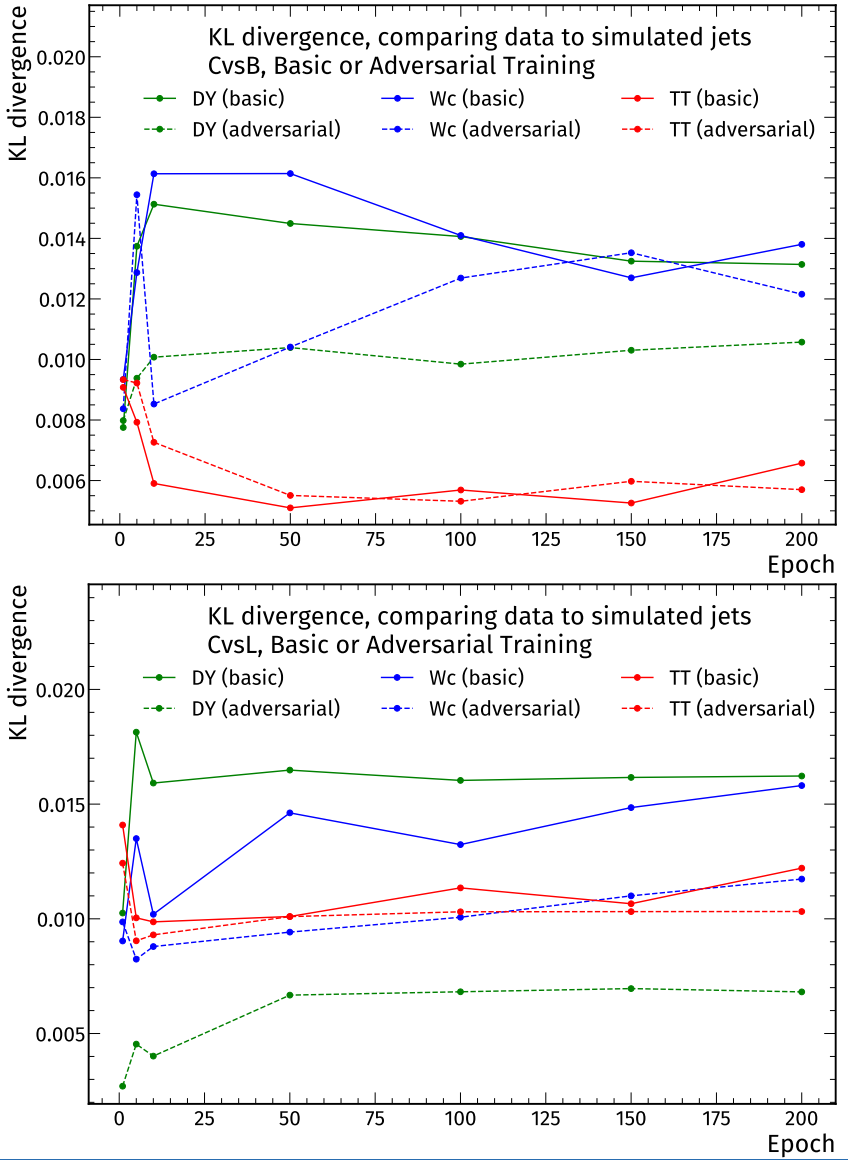
*statistical unc. only

Quantifying the Data/MC agreement: basic versus adversarial training

- Example: **CvsB and CvsL**, comparing **different epochs & selections**
- Kullback-Leibler divergence: $D_{KL}(P||Q) = \sum_k P_k \ln \frac{P_k}{Q_k}$

  Data (P) Simulation (Q)
  - Summary quantity
  - Evaluated at epochs 1, 5, 10, 50, 100, 150, 200
  - Depends on the binning (sum over bins, $k \in \{1, \ldots, 60\}$)
  - Best case (perfect agreement) would be $D_{KL}(P||Q) = 0$ if (and only if) the two distributions are identical

- **Adversarial training** (dashed lines) is **better** in the majority of the cases, with few exceptions



Conclusions

- AI safety studies for jet flavour tagging: **almost invisible disturbances** of the inputs result in **noticeable performance drops** → applicable & concerning for HEP
- Results are consistent with expectations: **model performance** improves with increasing number of epochs, but **susceptibility** towards adversarial attacks becomes larger as well
- **Robustness** improves with **adversarial training**: less vulnerable against adversarial attacks, (*slightly*) better agreement between data and simulation → robustness and **generalization** to data are connected, adversarial attacks replicate mismodellings (to some extent)
- Next steps / ideas for improvement:

Tagger	Strategy / paradigm	Evaluation technique
• DeepCSV → DeepJet (could show higher susceptibility due to large number of (low-level) inputs)	• Include physical constraints on the inputs (attacks needs to know how severe it can be, compared to syst. unc., results from commissioning); handling of defaults and integer features • Improve robustness against other, more complex adversarial attacks with suitable defenses (not only to first order) • Domain adaptation (couple evaluation on data / scale factors with the training)	• Discriminators, ROC, AUC • Inputs (histograms, correlations, AUC-score) • Loss landscape • Data/MC agreement / scale factors

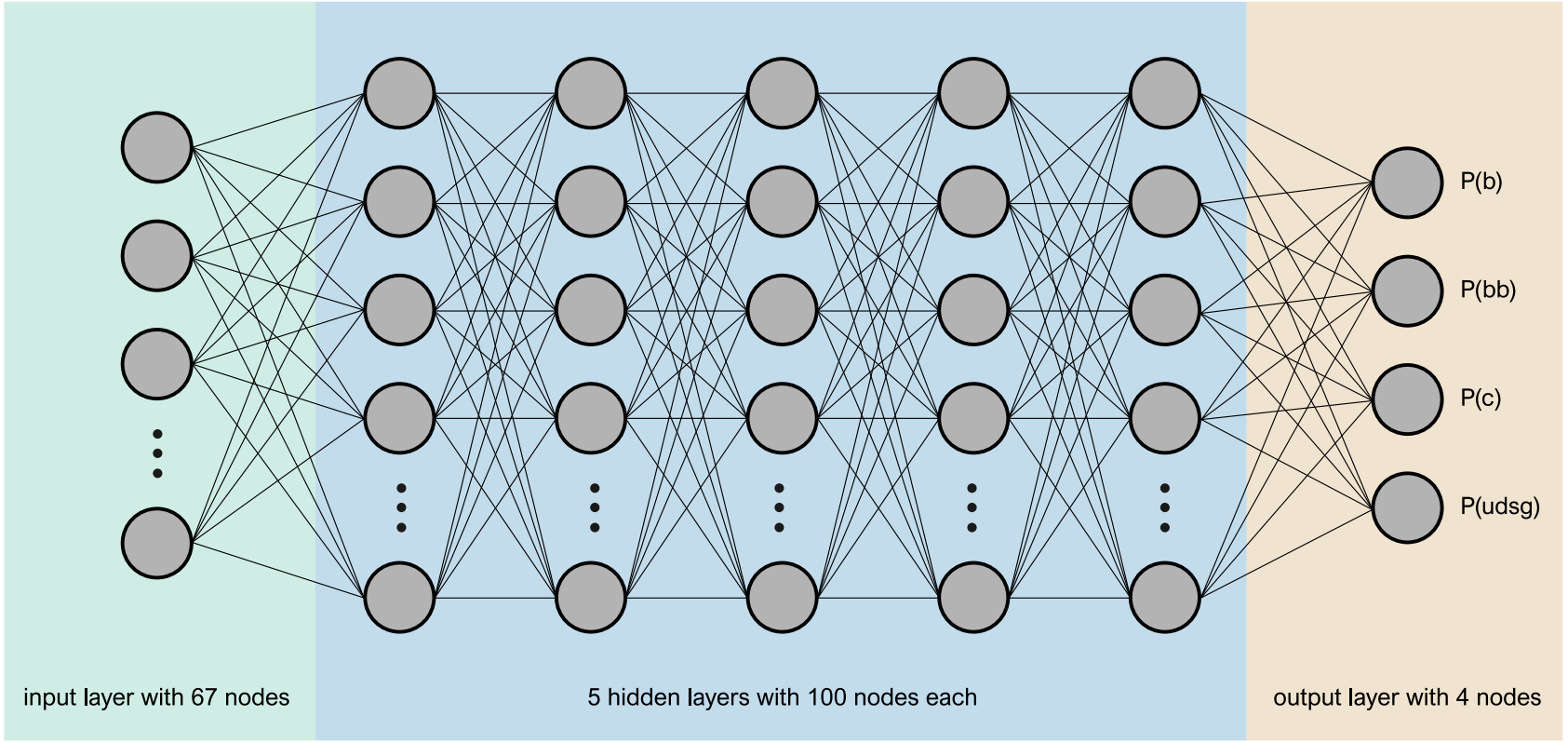
Thank you!

Backup

Model architecture (summary)

Use a fully connected deep neural network

- Training: 672M QCD & $t\bar{t}$ to semileptonic samples
  (75M for validation, 187M for testing)
- Reweighting in p_T, η, flavour
- Modified cross entropy focal loss function



Details

PyTorch, NVIDIA Tesla V100 GPU
47604 trainable parameters
Hadron-flavour based truth definition
Class imbalance:

b	bb	c	usdg
9.7%	0.7%	8.1%	81.5%

Hyperparameters

Batch size: 1M
Adam optimizer, learning rate decay initialized with 0.0001
200 epochs
$\gamma = 25$ (focusing parameter)
Defaults slighly below minima of input distributions

Samples

- Mixture of QCD multijet and semileptonic $t\bar{t}$ events

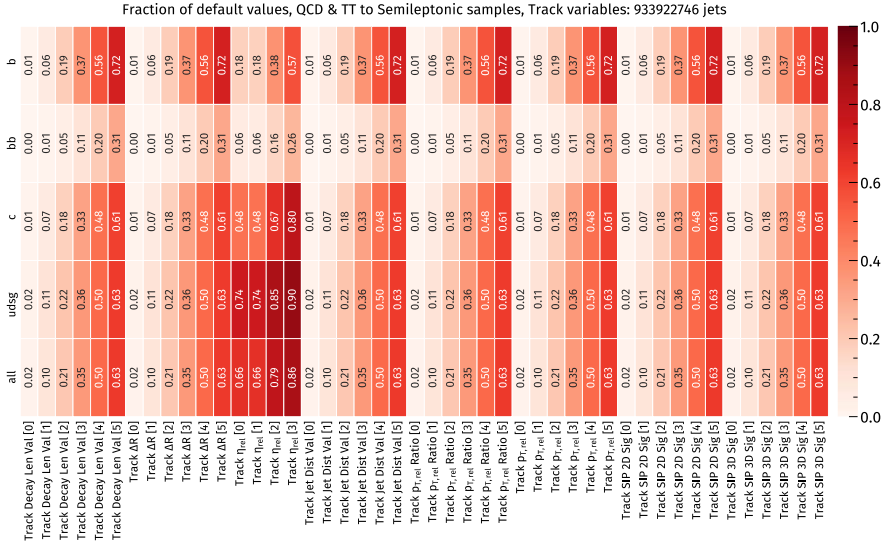
Table 2.1: Used processes with their respective filenames. The event count and the number of jets that could be retrieved from the chosen samples before any cleaning are shown in the additional columns.

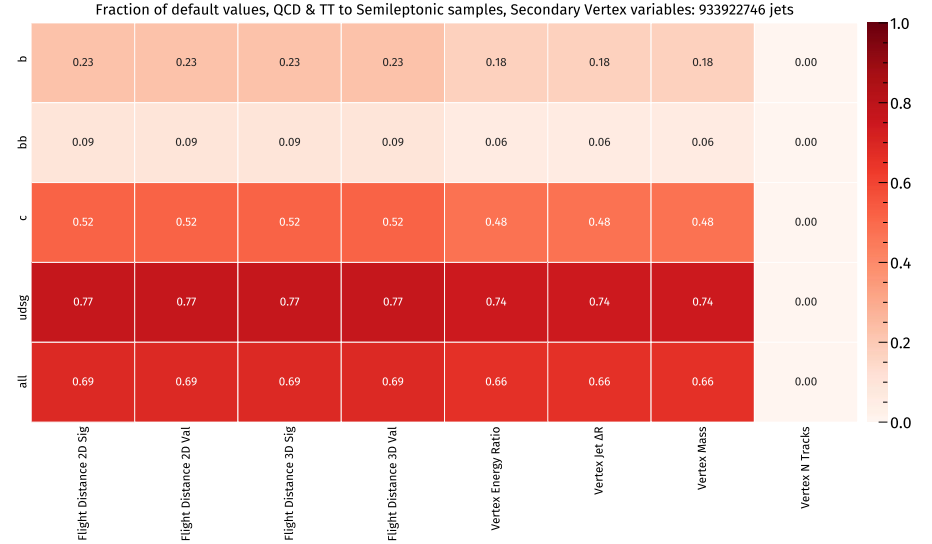
Sample name	Events	Jets
QCD_HT300to500_TuneCP5_13TeV-madgraph-pythia8	60316577	428503277
QCD_HT500to700_TuneCP5_13TeV-madgraph-pythia8	56207744	430806556
QCD_HT700to1000_TuneCP5_13TeV-madgraph-pythia8	19761895	156090013
QCD_HT1000to1500_TuneCP5_13TeV-madgraph-pythia8	16595628	135238130
QCD_HT1500to2000_TuneCP5_13TeV-madgraph-pythia8	11634434	97728944
QCD_HT2000toInf_TuneCP5_13TeV-madgraph-pythia8	5941306	50313754
TTToSemiLeptonic_TuneCP5_13TeV-powheg-pythia8	43732445	382423670

Inputs: cleaning

- Only use jets with $|\eta| < 2.5$ **and** $20\ \mathrm{GeV} < p_T < 1\ \mathrm{TeV}$
- Cut outliers of $\Delta R(\text{summed tracks}, \text{jet})$, i.e. require the angular distance between the summed four-momentum of the tracks and the jet axis to be < 0.3

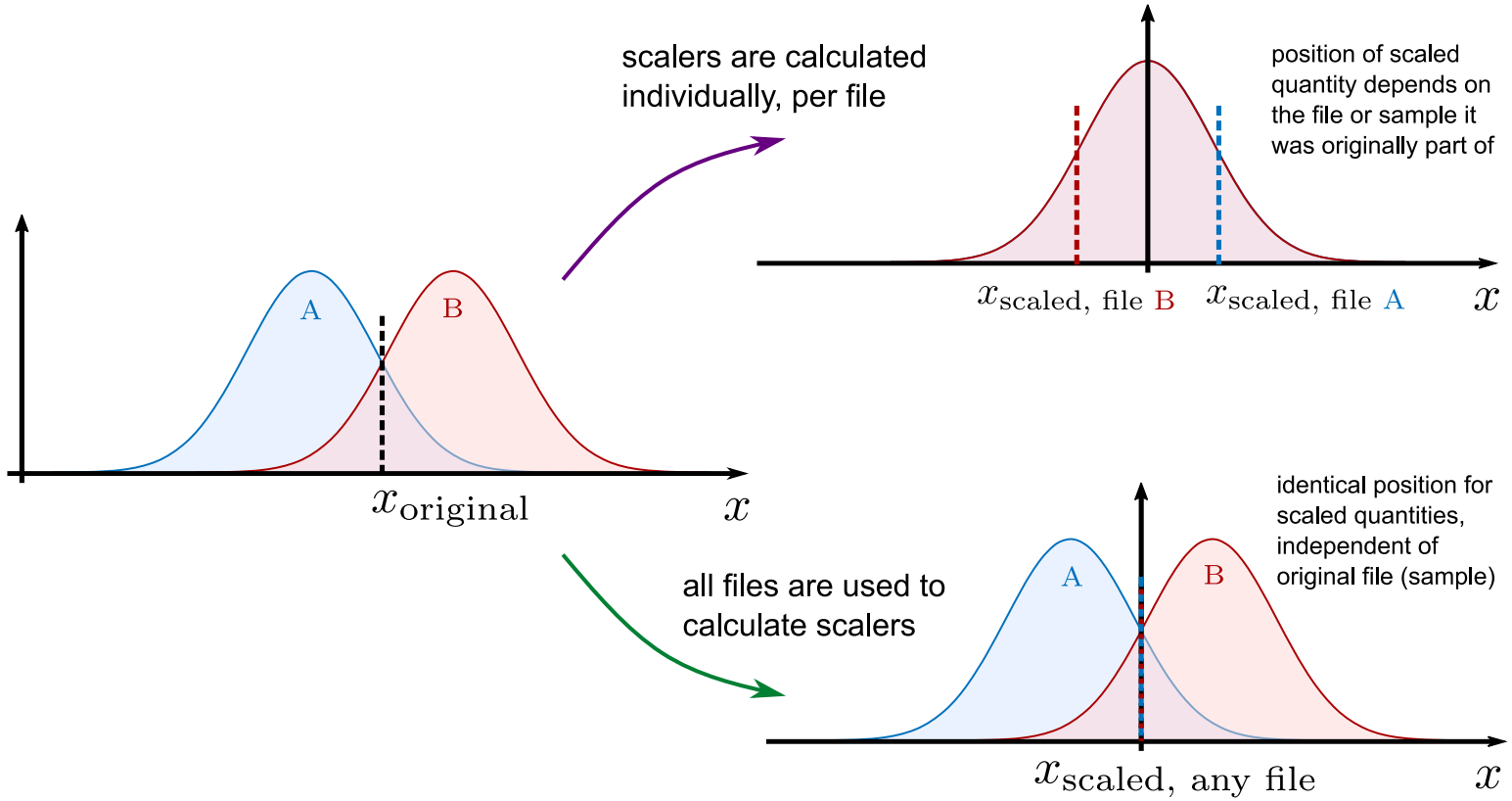
- Set **defaults**: just below the bulk distribution (average minima over ≈ 20 input files – *this was done by Nikolas Frediani*)
- Defaults are needed when the values are `-Inf, +Inf, NaN, -999, -1` or when less selected tracks are present than the number required to fill the variable with a meaningful value (otherwise one finds values like $+10^{34}$)





Inputs: preprocessing

- **Split** into training (72%), validation (8%) and test (20%) set to provide samples on which the model has not been trained on (check generalization on unseen data)
- **Normalize** to unit standard deviation, centered at 0 (based on all training samples)



Reweighting (theory)

- Reasons to perform reweighting:
  - Physics: **independence** of the distribution of the **kinematic variables** for the specific processes that are chosen for the training
    - Want to include p_T and η to take possible correlations with other variables into account
    - But: without reweighting, the tagger could learn some unwanted mappings between e.g. high p_T ↔ b jets, just because it is the case for e.g. semileptonic $t\bar{t}$ events → tagger would classify jets that emerge from different process based on the kinematic distributions of the training samples (would not generalize well) ⚠
  - On the practical side: massive **class imbalance** between b, bb, c and light jets

Table 2.5: Distribution of samples into training, validation or test set and the respective flavour content.

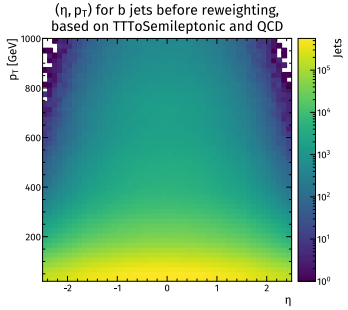
Sample → / Flavour ↓	Training	Validation	Test	All samples
b	65171355	7241434	18096125	90508914 (9.7 %)
bb	4849162	538355	1348007	6735524 (0.7 %)
c	54557491	6065387	15154480	75777358 (8.1 %)
l	547846145	60868752	152186053	760900950 (81.5 %)
All flavours	672424153 (72 %)	74713928 (8 %)	186784665 (20 %)	933922746 (100 %)

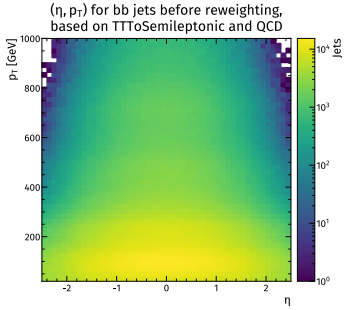
Reweighting (how it's done)

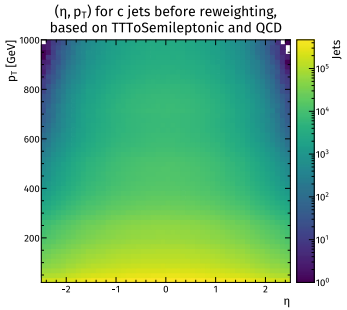
- two possible strategies currently implemented to get the weights, both use a $\mathbf{50 \times 50}$ **binning in** $\boldsymbol{p_T}$ **and** $\boldsymbol{\eta}$ and both also incorporate the class imbalance (i.e. with a factor 1 / flavour)
  - Target distribution: average per bin over all four flavours (used to study robustness)
  - Target distribution: *flat in* p_T *and* η (not used to study robustness)
- weights are calculated as the **quotient between target and original distribution** (per bin)
- every sample (**= jet**) is assigned a weight, based on the bin in which its p_T and η values reside
- **loss weighting** uses the element-wise product of the loss (per sample) with the sample weight → average this per batch and perform backpropagation with the scalar loss

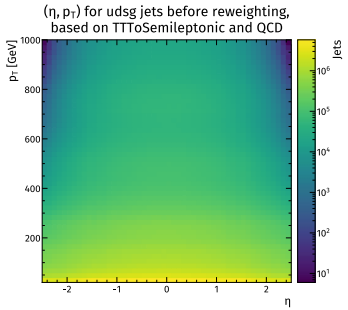
Reweighting (visualized) – average

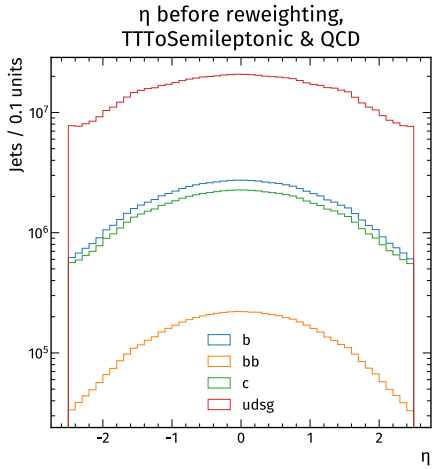
(p_T,η) distribution before reweighting

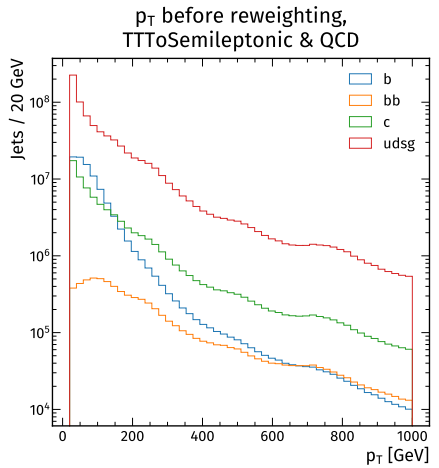




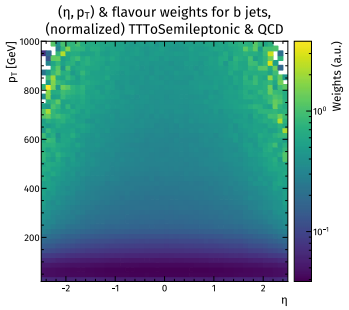


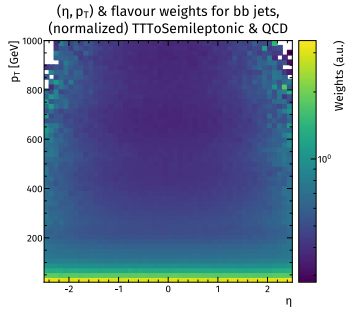


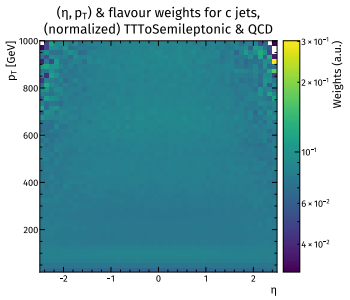


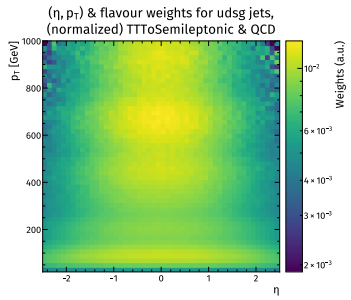


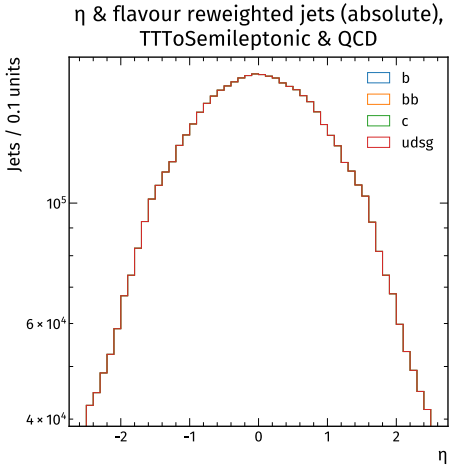
multiply with weights

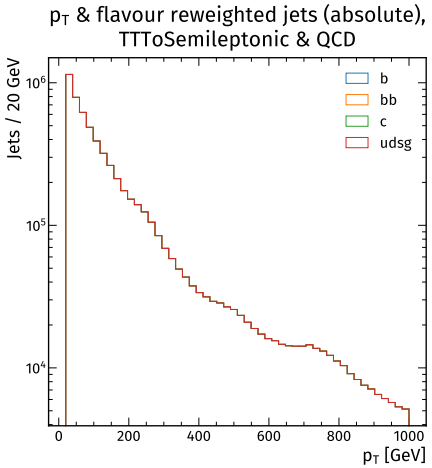






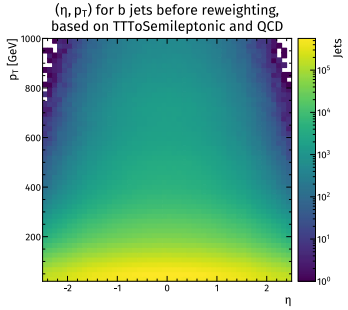


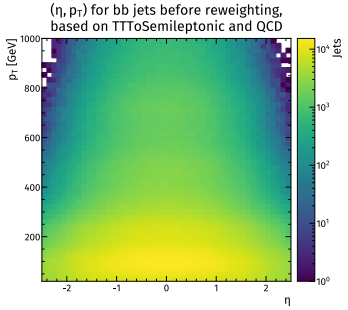


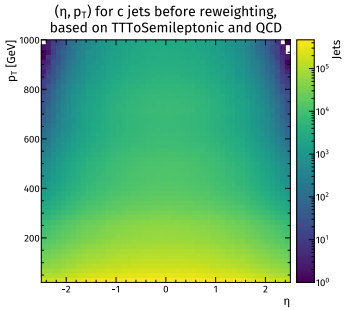


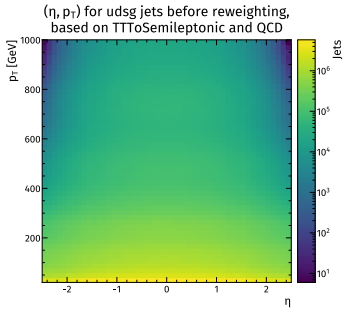
Reweighting (visualized) – flat

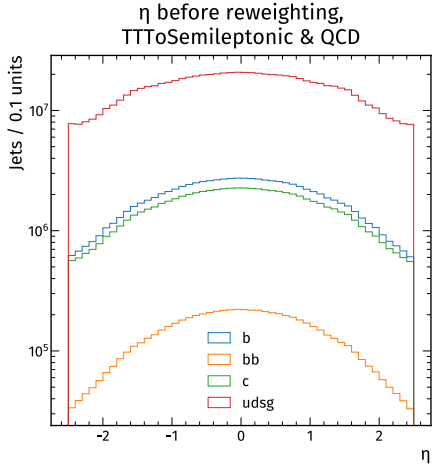
(p_T,η) distribution before reweighting

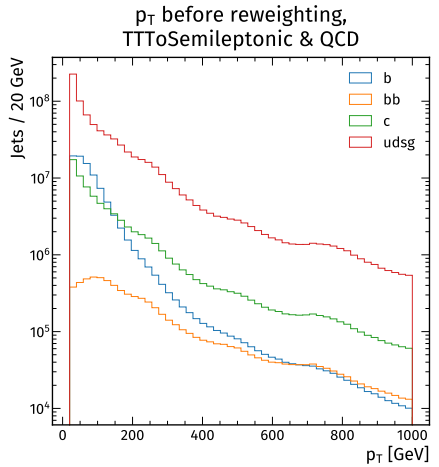




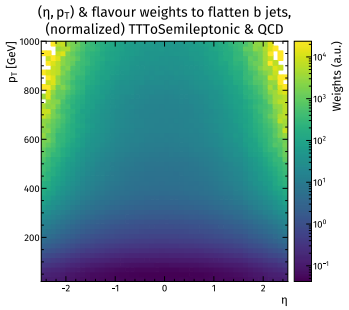


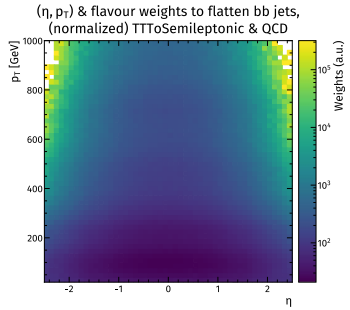


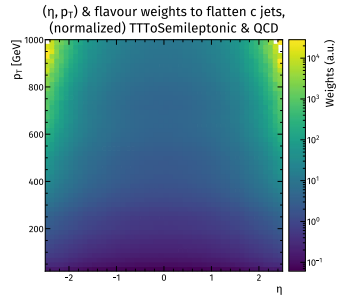


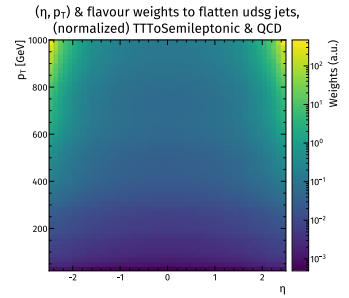


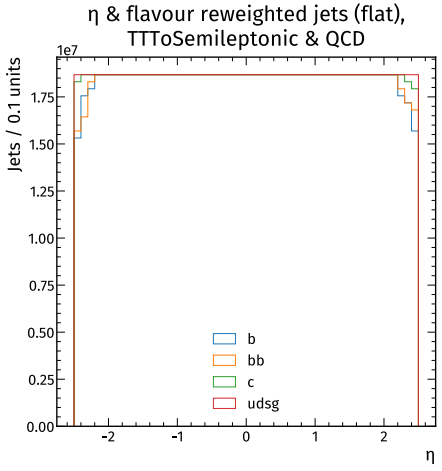
multiply with weights

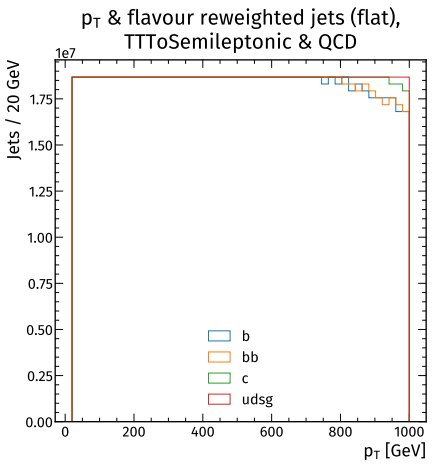






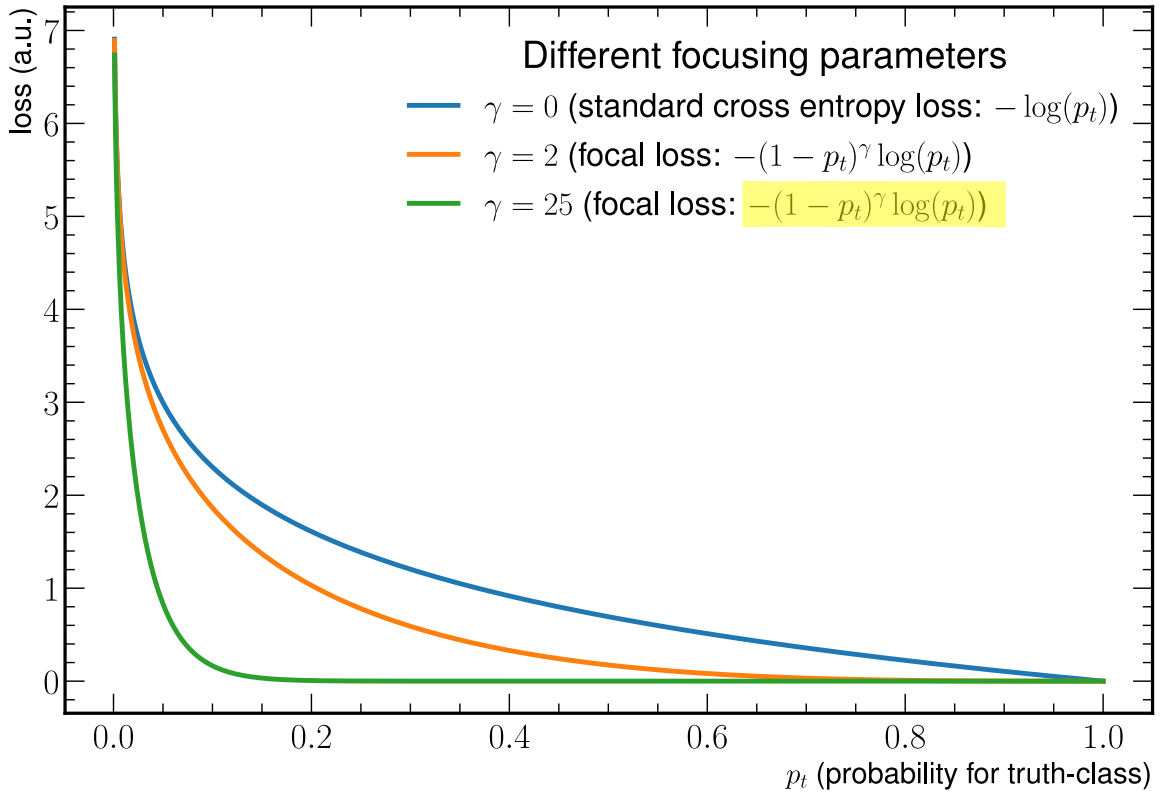




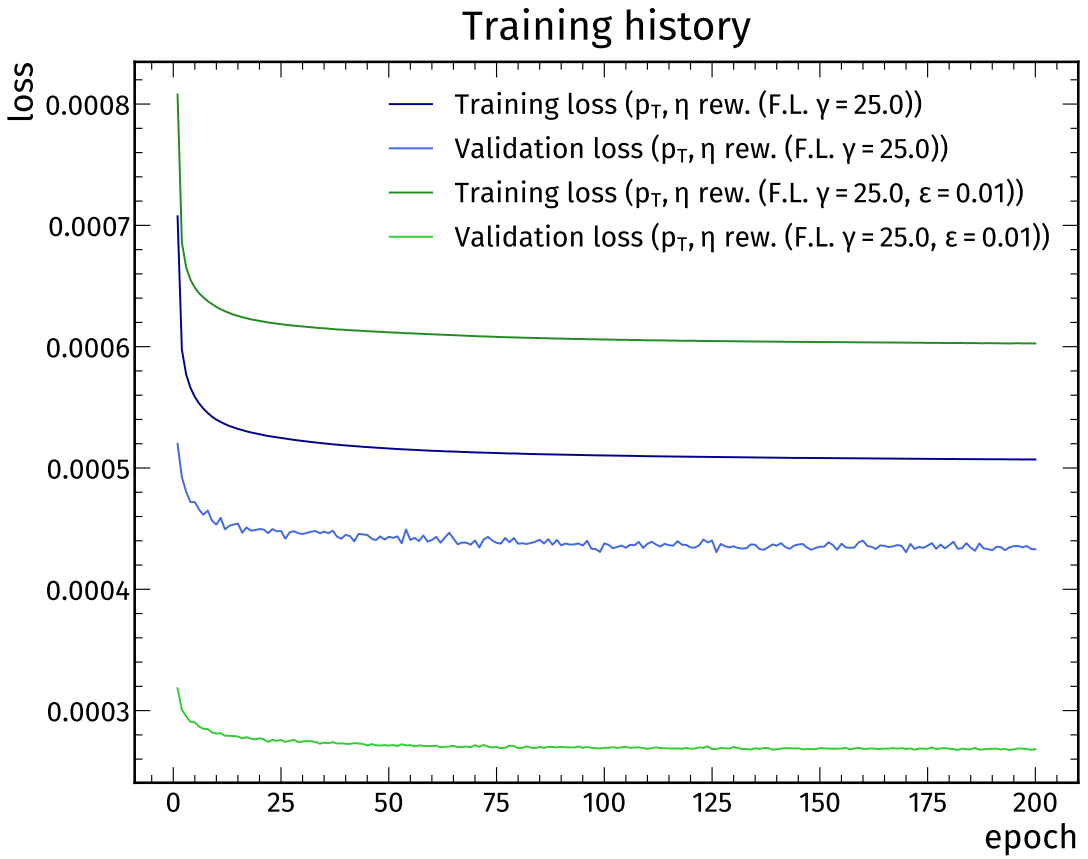


Focal loss

- Idea: additional reweighting of the loss with the goal to
  - **downweight** the **easy** to classify samples
  - give **more importance** to **hard** to classify samples
- Choose $\gamma = 25$ as the focusing parameter
- Effect: tagger distributions are more **spread out**
  - not only filled with many entries in the 0 or 1 bin
  - also enough statistics in between (good for scale factors)



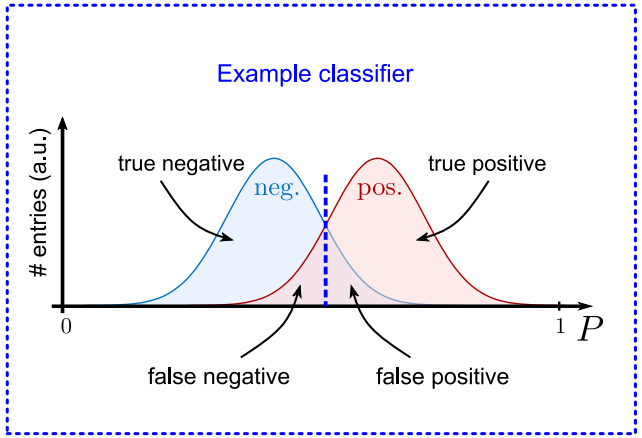
Convergence of loss over epoch

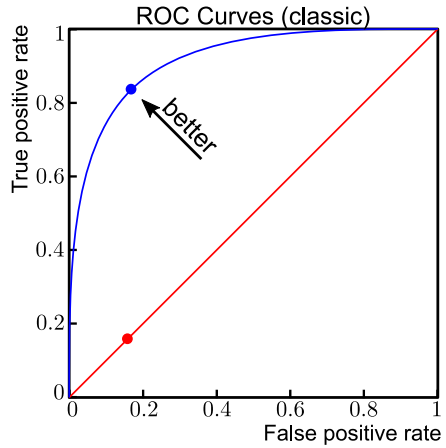


Just monitoring the training

- Training and validation loss
  - to check that model converges
  - no overfitting

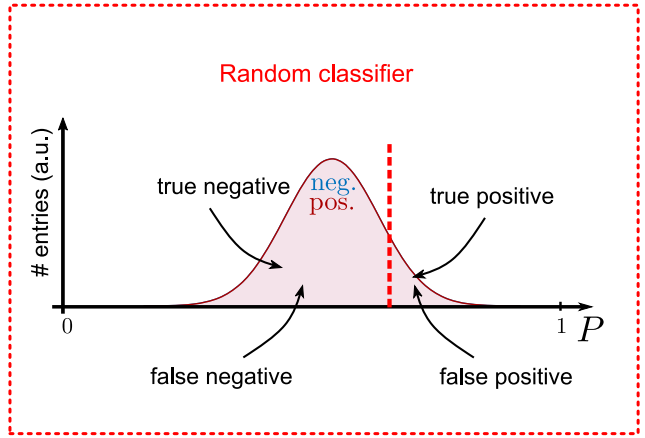
ROC curves (evaluation metric)

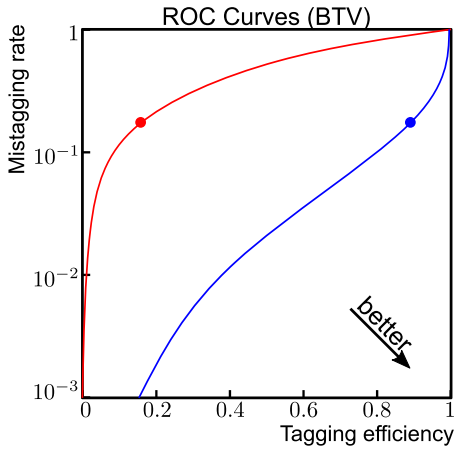




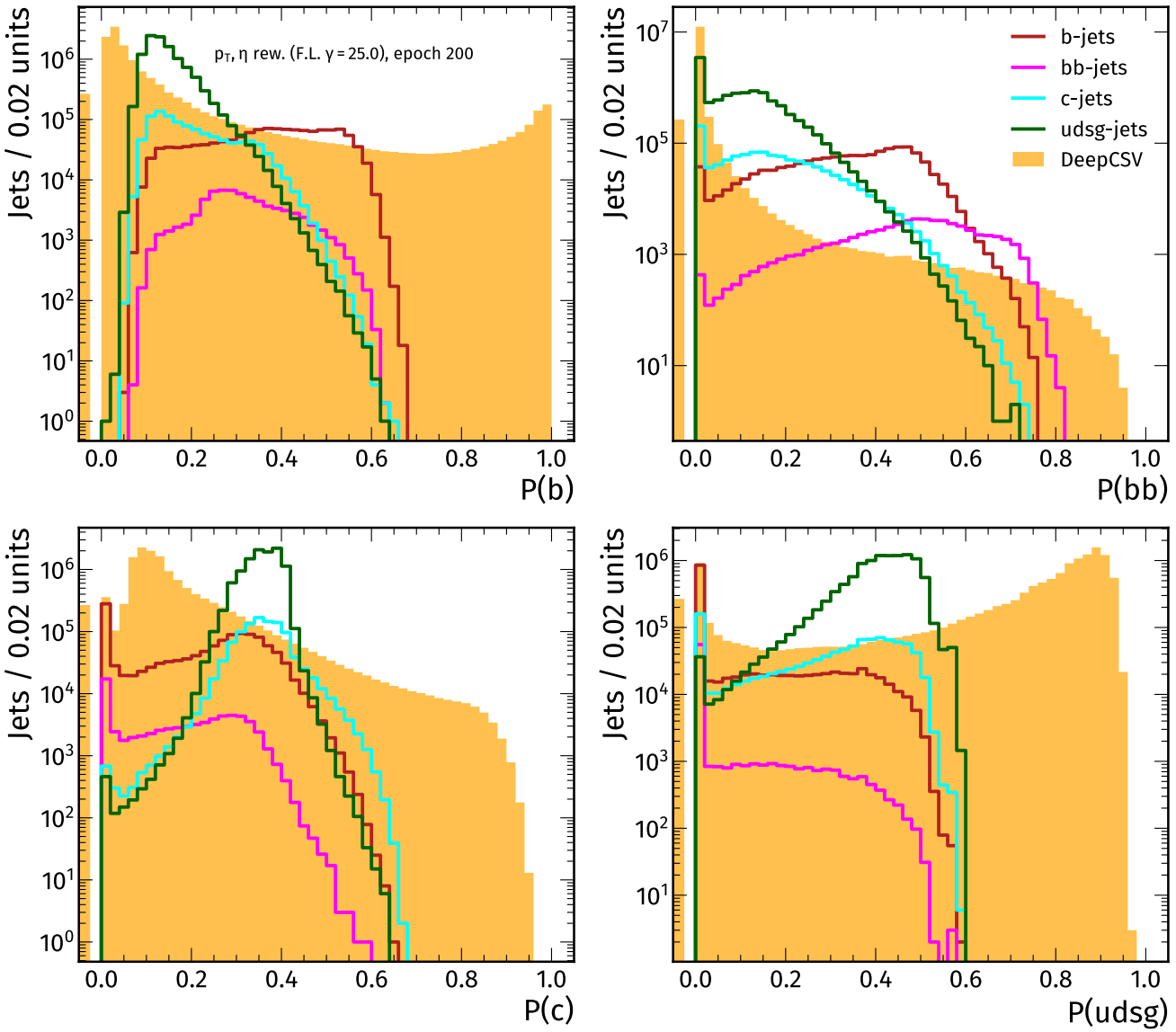
$0.5 < \mathrm{AUC} < 1$

$\mathrm{AUC} = 0.5$

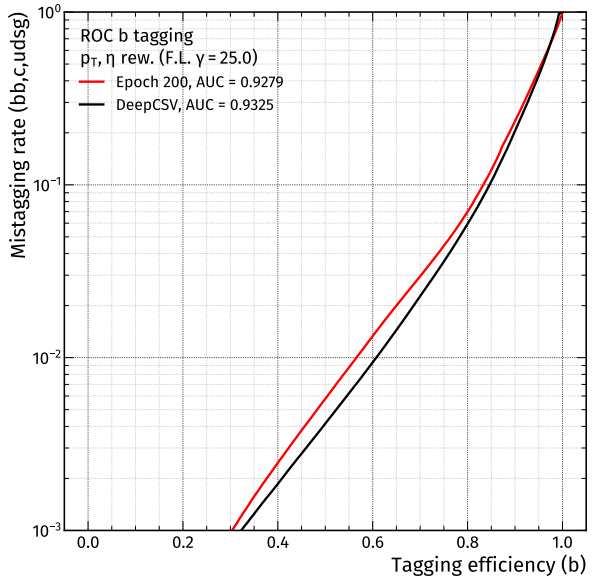


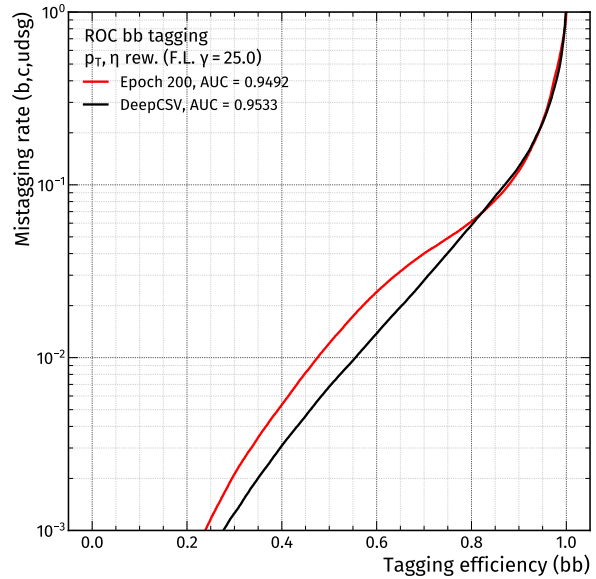


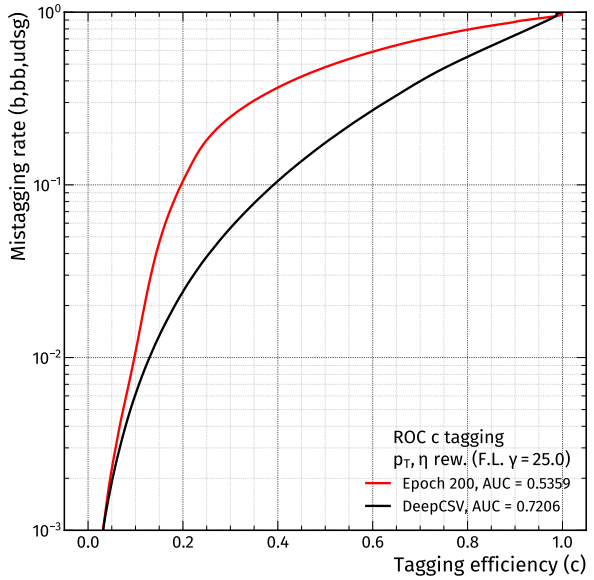
Nominal performance: tagger outputs

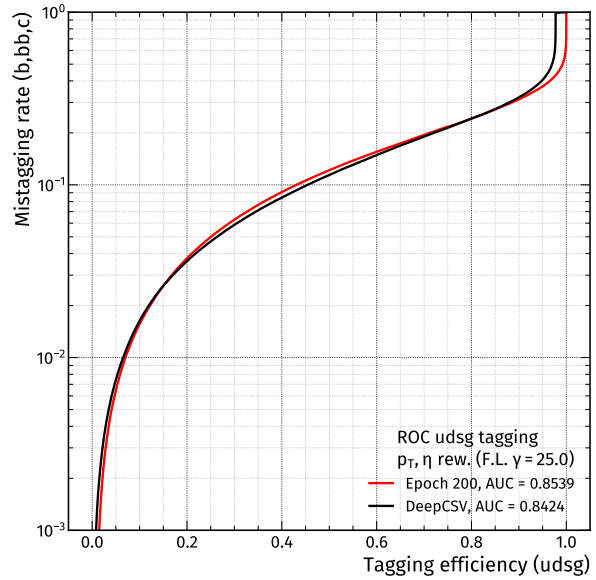


Nominal performance: ROC XvsAll





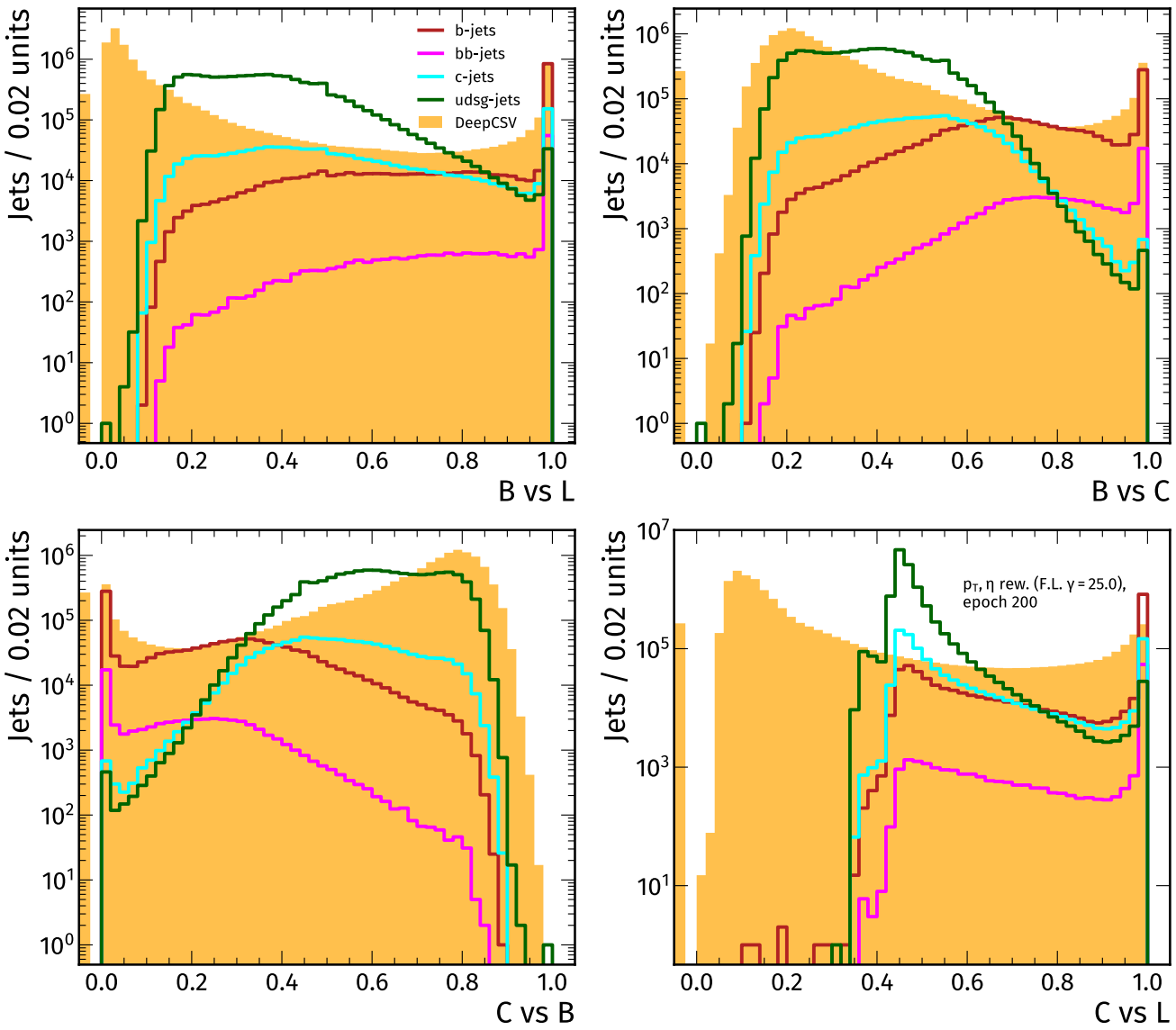




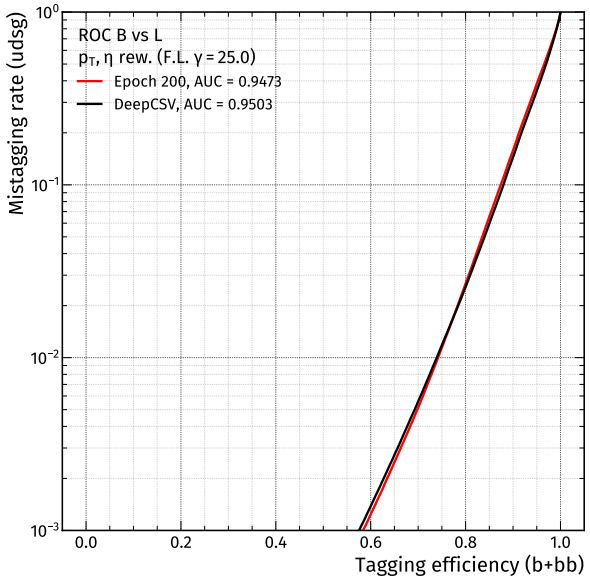
Definitions: discriminators

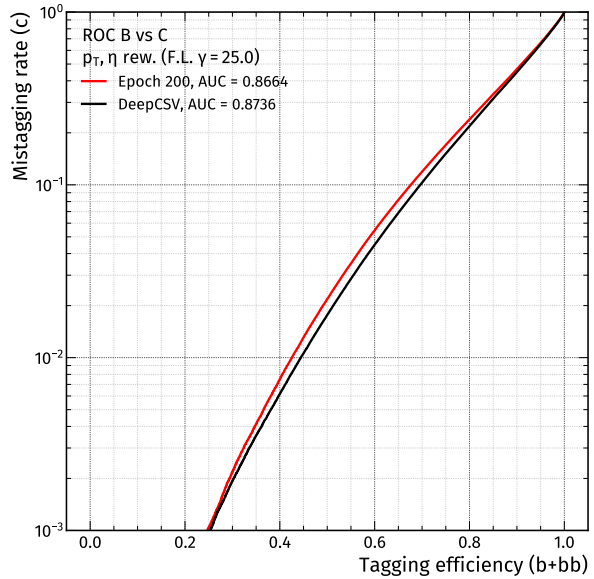
$$
\begin{aligned}
\text{BvsL} &= \frac{P(\text{b}) + P(\text{bb})}{1 - P(\text{c})} &&= \frac{P(\text{b}) + P(\text{bb})}{P(\text{b}) + P(\text{bb}) + P(\text{udsg})}\,, \\
\text{BvsC} &= \frac{P(\text{b}) + P(\text{bb})}{1 - P(\text{udsg})} &&= \frac{P(\text{b}) + P(\text{bb})}{P(\text{b}) + P(\text{bb}) + P(\text{c})}\,, \\
\text{CvsB} &= \frac{P(\text{c})}{1 - P(\text{udsg})} &&= \frac{P(\text{c})}{P(\text{b}) + P(\text{bb}) + P(\text{c})}\,, \\
\text{CvsL} &= \frac{P(\text{c})}{1 - (P(\text{b}) + P(\text{bb}))} &&= \frac{P(\text{c})}{P(\text{c}) + P(\text{udsg})}\,.
\end{aligned}
$$

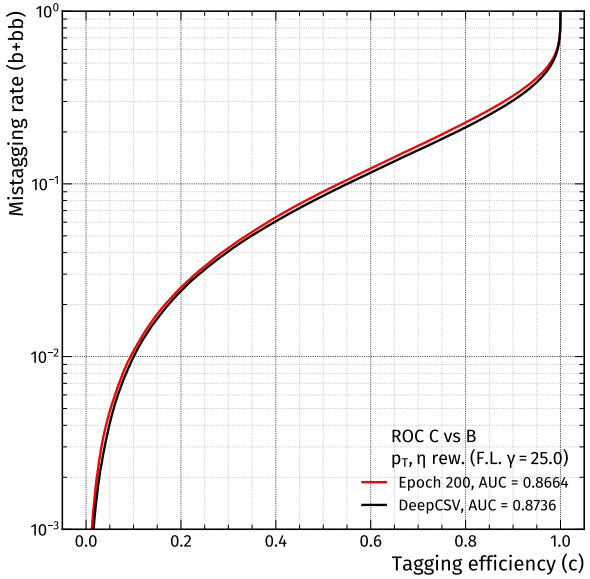
Nominal performance: discriminator shapes

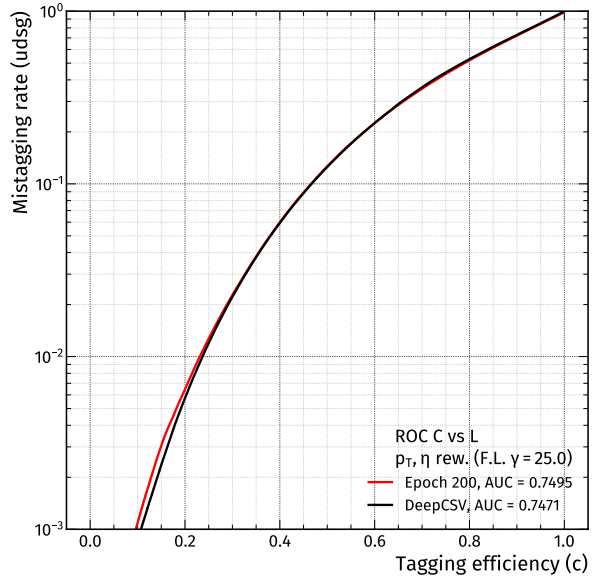


Nominal performance: ROC for all discriminators

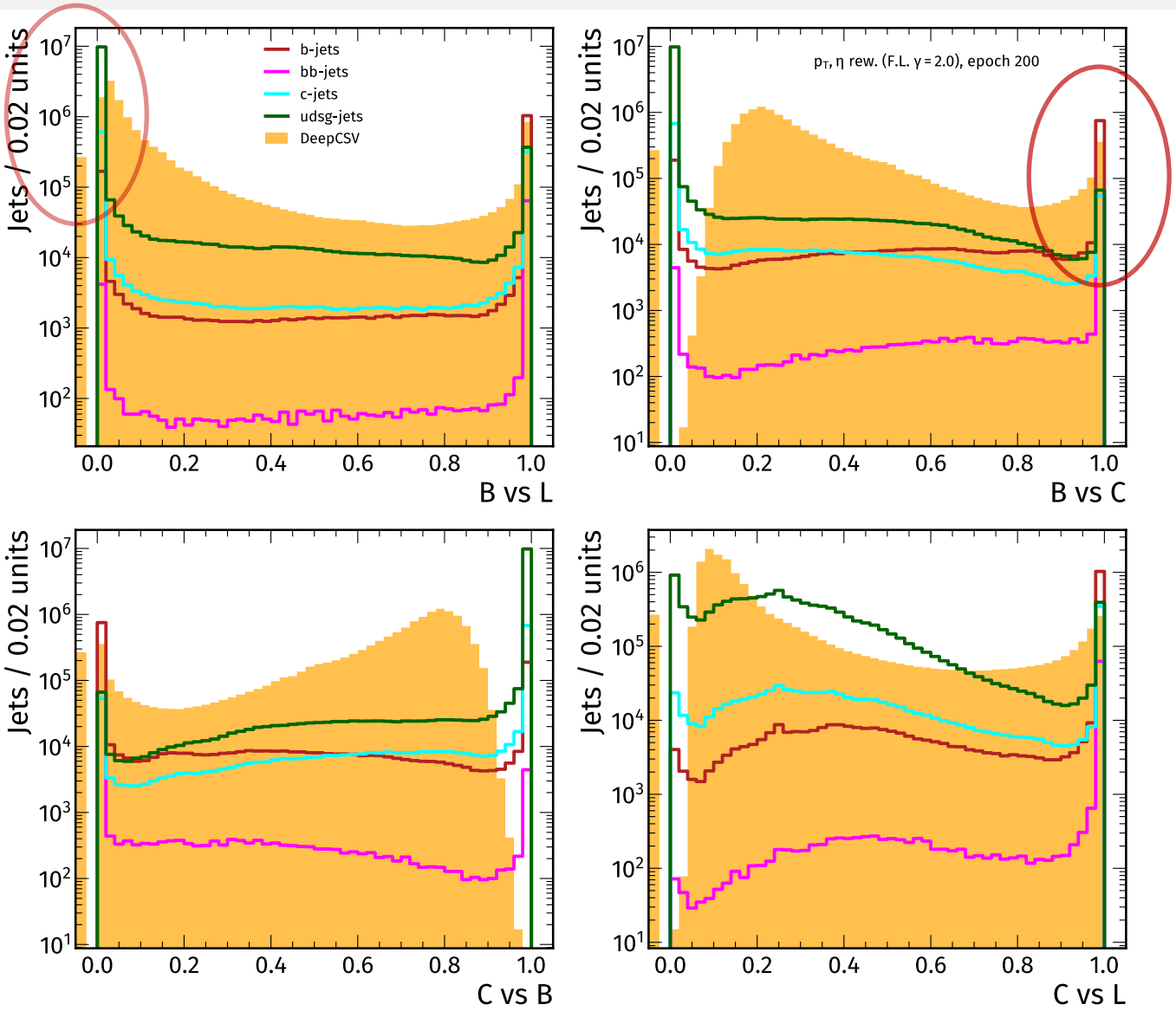








Discriminator shapes (with $\gamma = 2$)

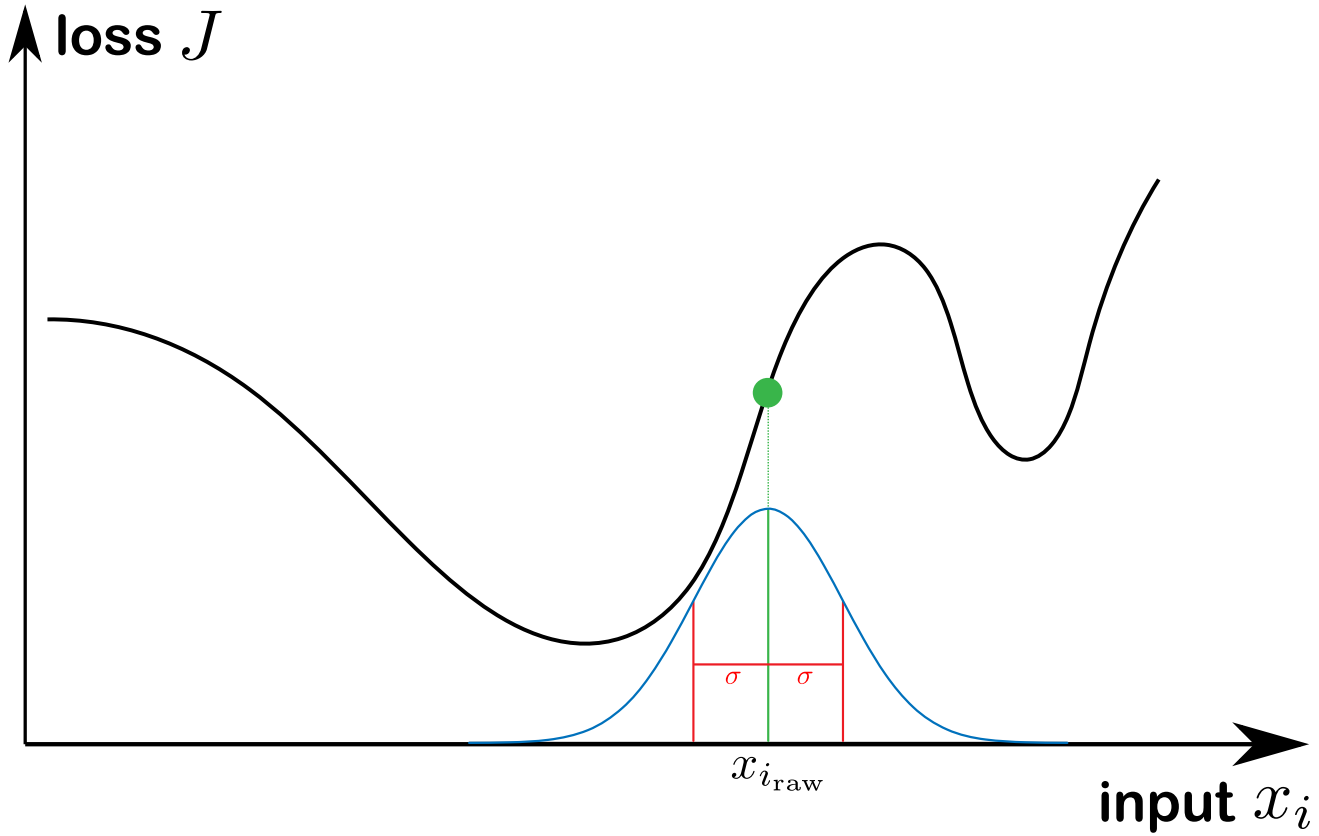


Gaussian Noise

- Add a noise term ξ to generate distorted inputs x_{noise}:

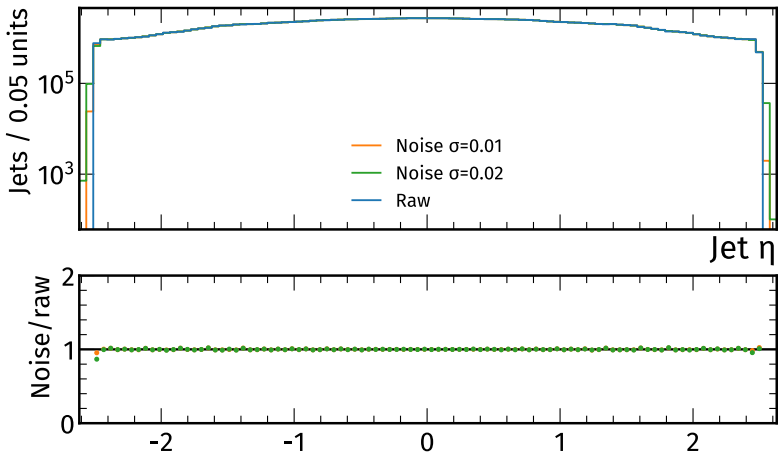
$$x_{noise} = x_{raw} + \xi$$

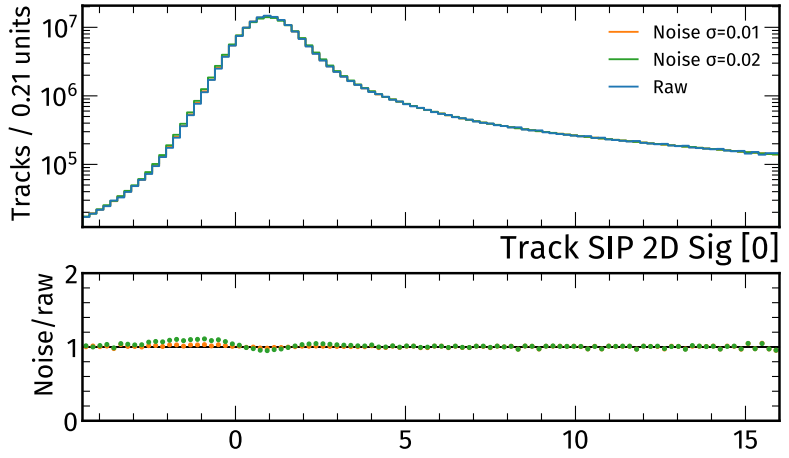
- ξ follows a Gaussian distribution centred at $\mu = 0$, the standard deviation σ can be varied

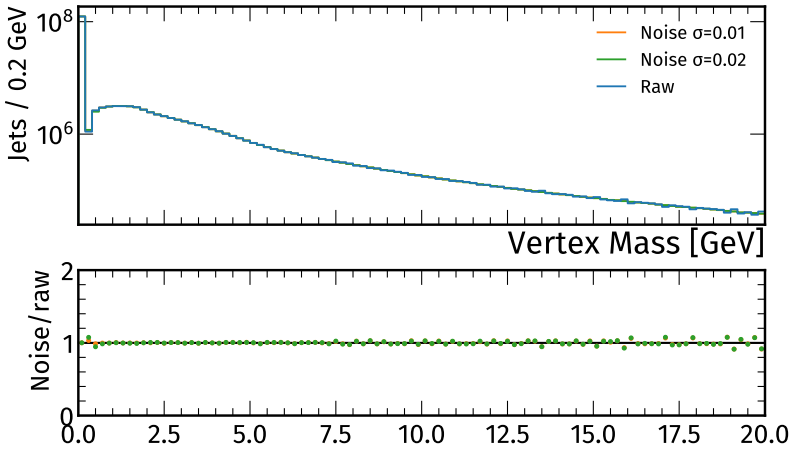


Distorted Inputs (Noise)

- Add Gaussian noise term to test sample





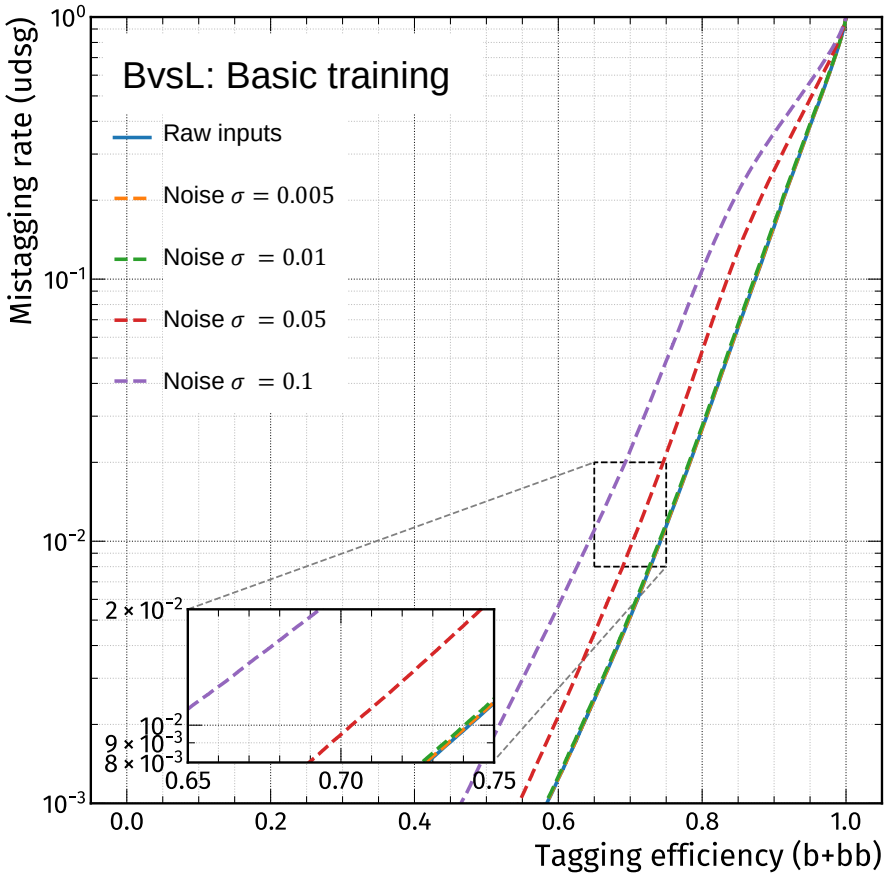


- Use small σ for the disturbance and check jet / track / SV properties

→ Almost invisible changes of the input shapes, but with a non-systematic, random approach only

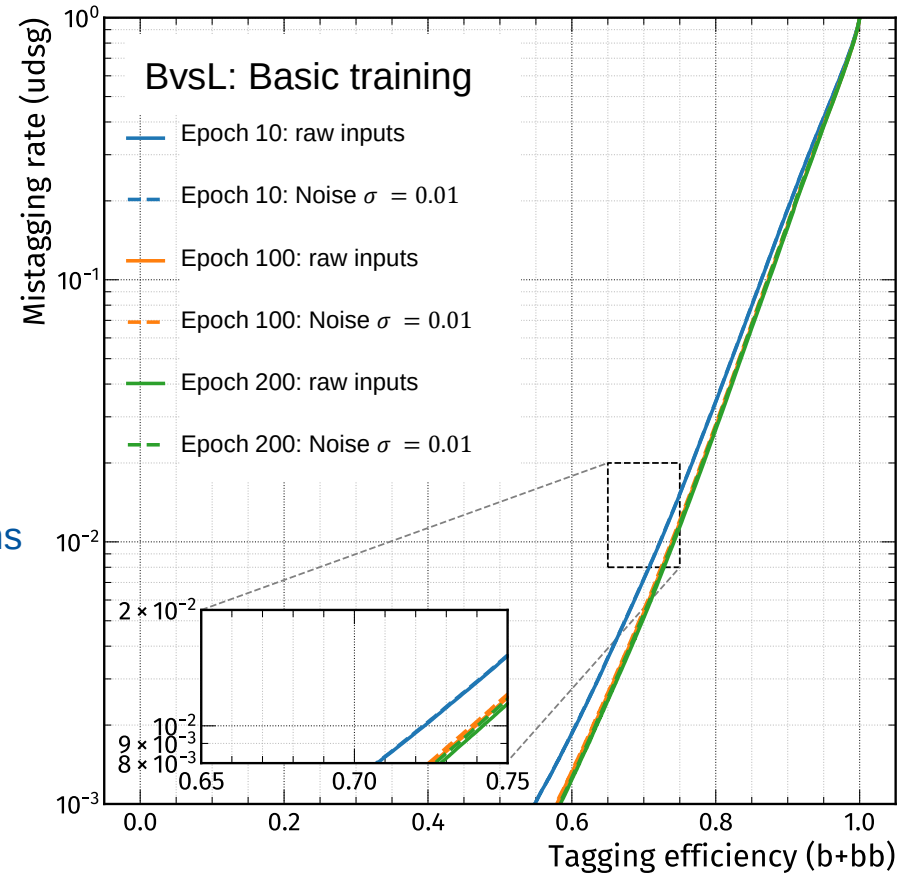
ROC curves (B versus L, Basic Training + Noise)

- Evaluate model on raw and distorted inputs, use ROC curves to investigate possible changes of performance



Same epoch, different σ

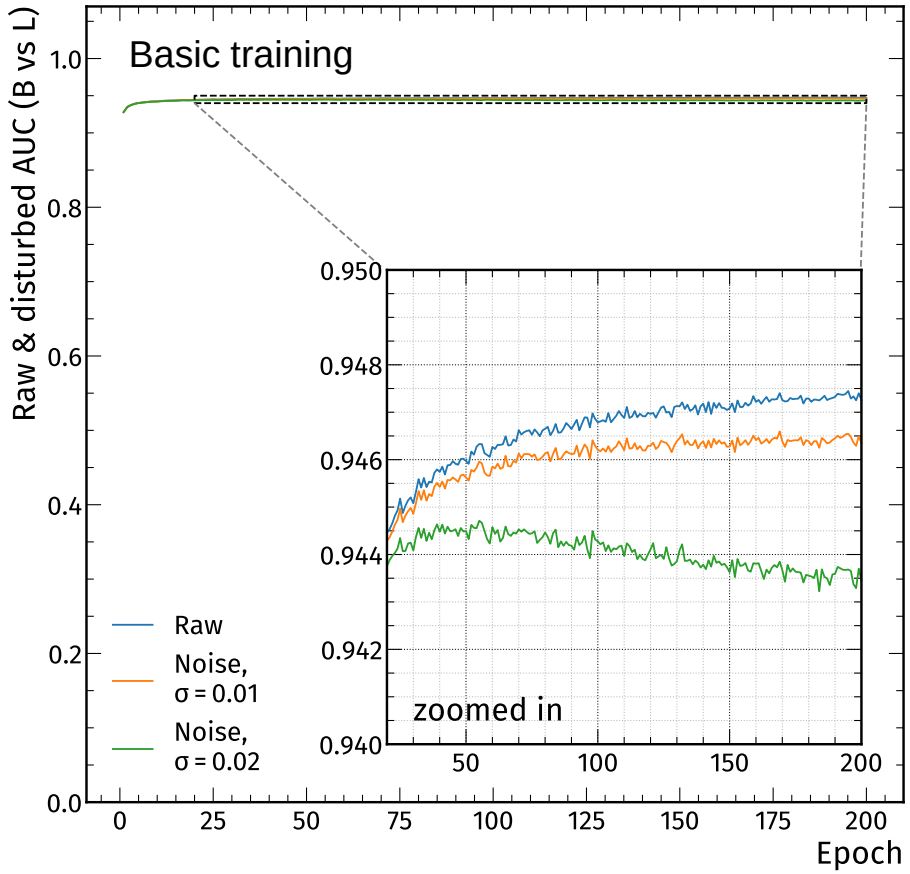
Same σ, different epochs



- Larger σ reduces performance more (as expected), overall: only marginal effects with reasonable σ
- Nominal performance improves with more epochs, practically no changes with small σ

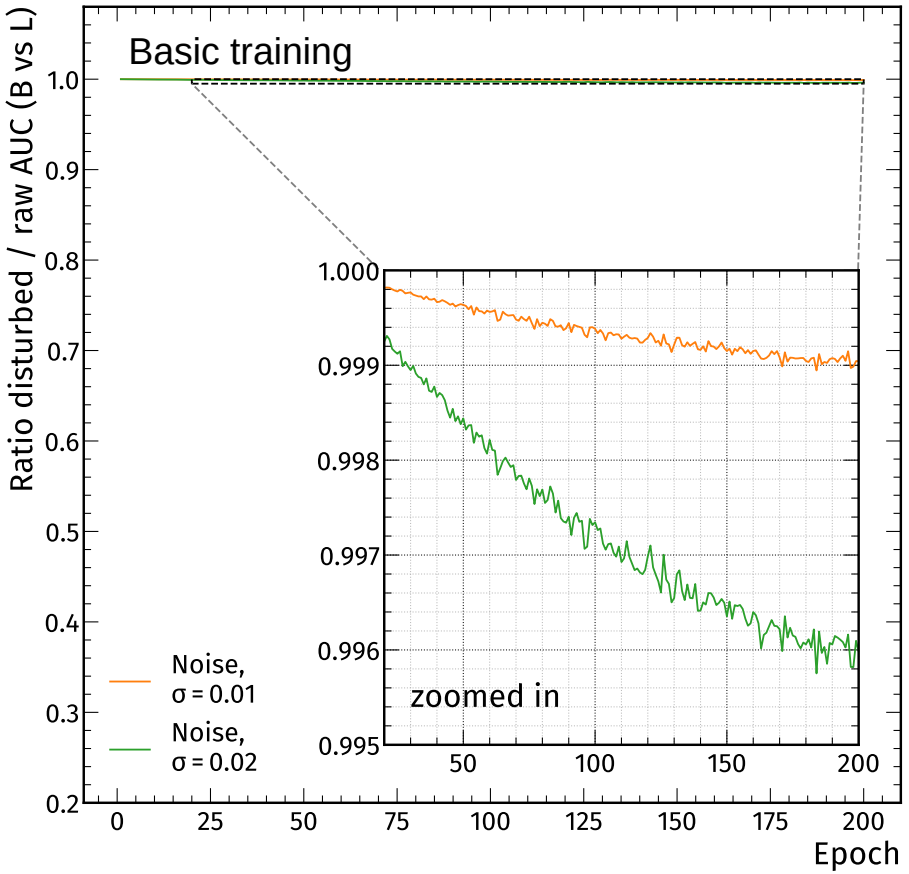
Evolution of AUC with number of epochs (Basic Training + Noise)

- Save model checkpoint after every epoch, run inference and compute AUC for the BvsL discriminator



Absolute values

Ratios Noise / raw

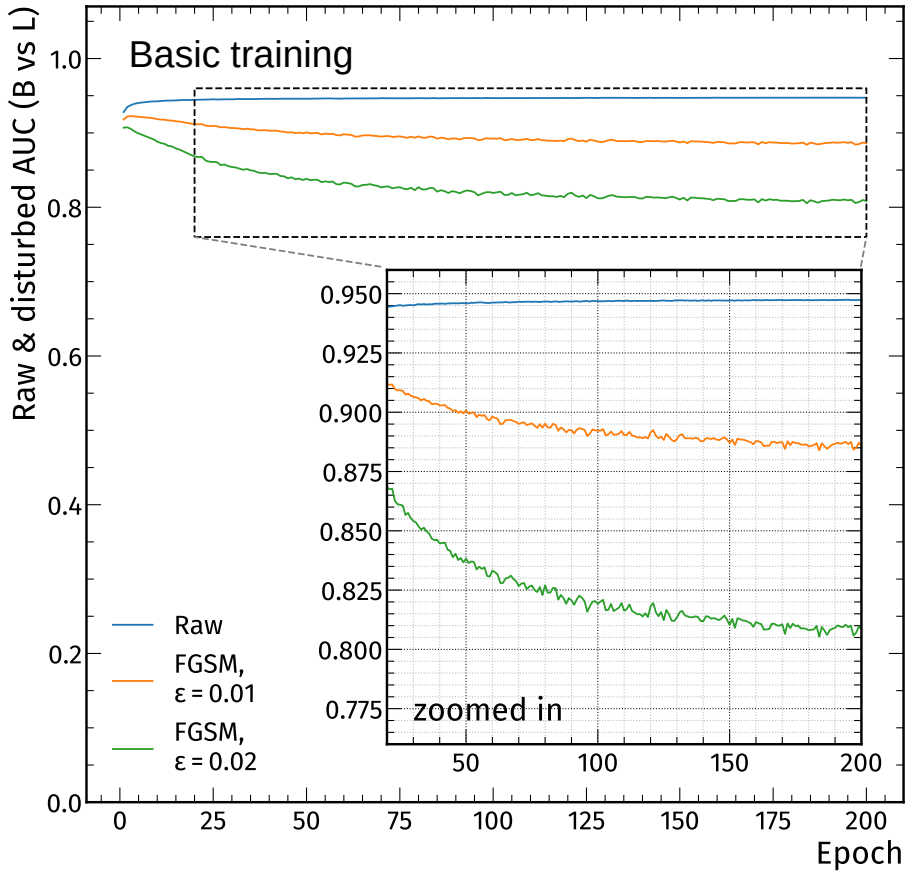


- Performance on raw inputs increases slowly
- Impact of noise term becomes more severe

→ Tradeoff between performance and robustness!
But nowhere near as bad as for the FGSM attack.

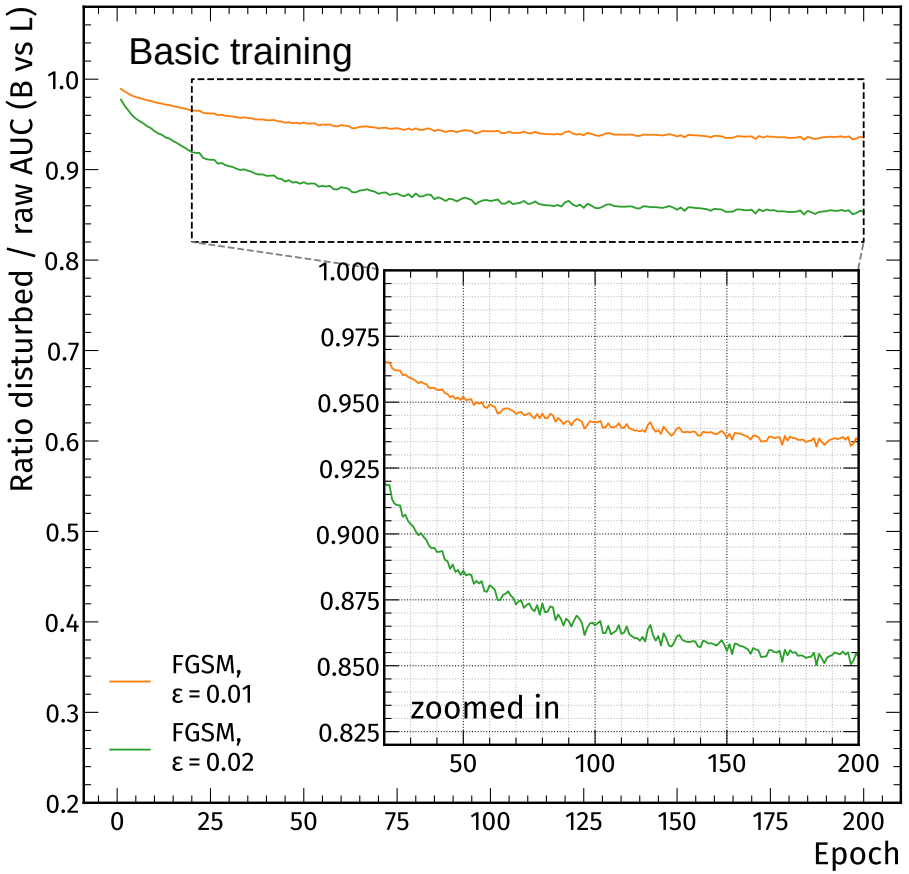
Evolution of AUC with number of epochs (Basic Training + FGSM)

- Save model checkpoint after every epoch, run inference and compute AUC for the BvsL discriminator



Absolute values

Ratios FGSM / raw

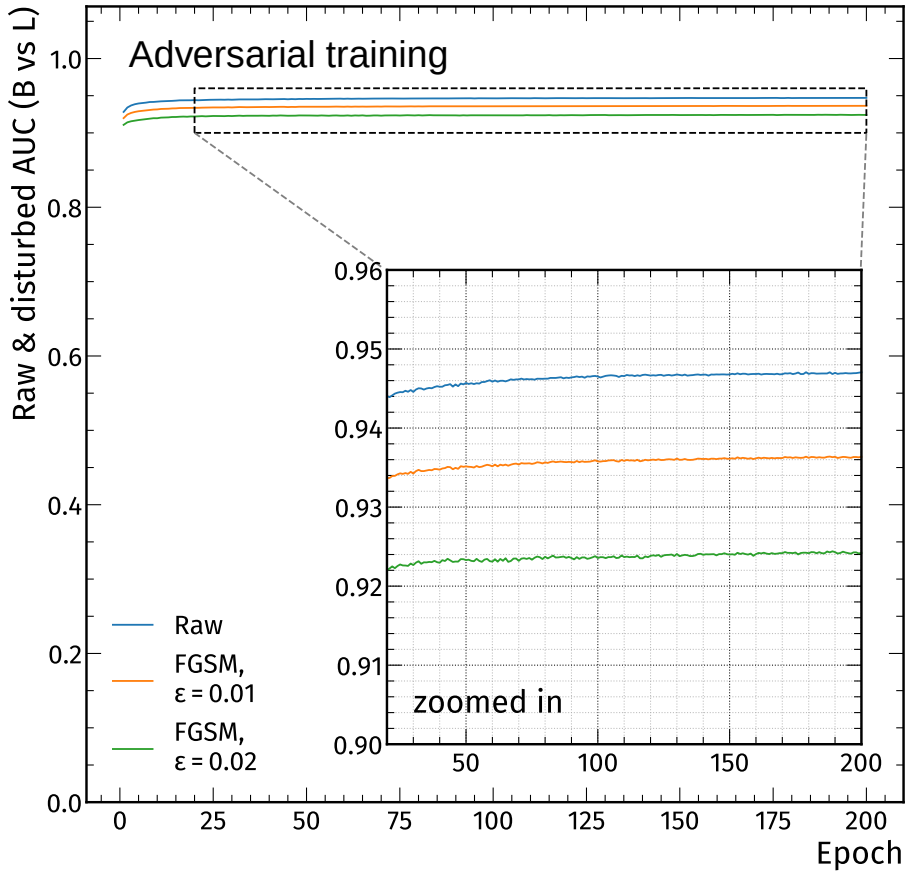


- Performance on raw inputs increases slowly
- Impact of FGSM attack becomes more severe

→ Tradeoff between performance and robustness!

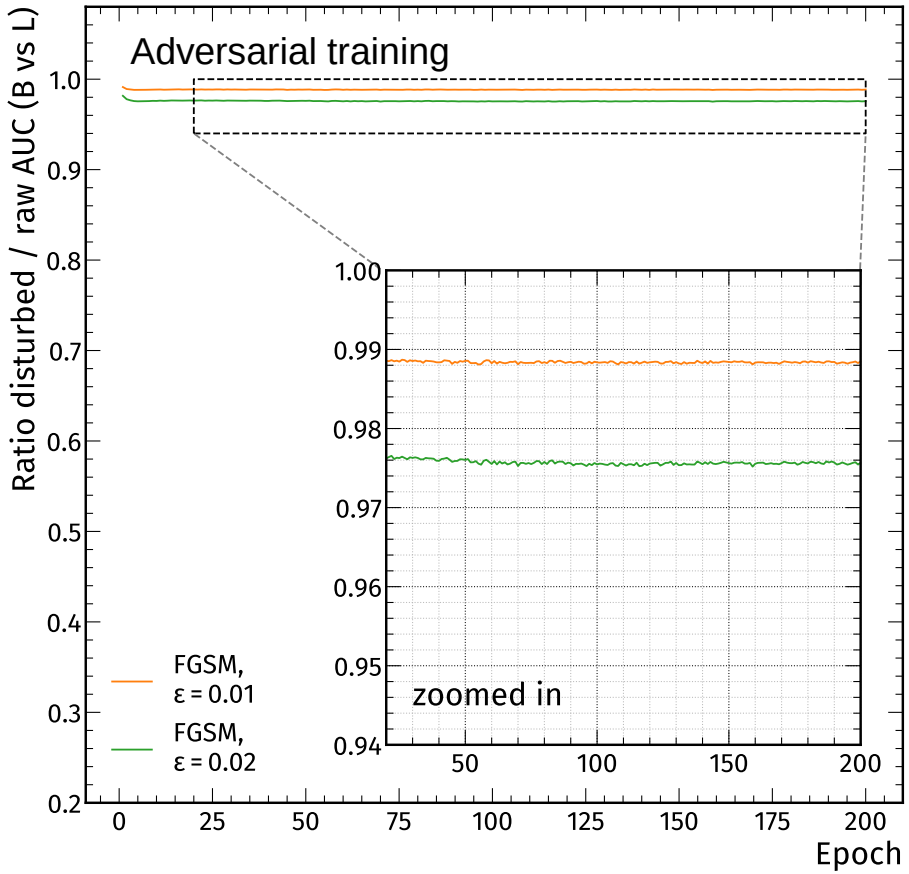
Evolution of AUC with number of epochs (Adversarial Training + FGSM)

- Save model checkpoint after every epoch, run inference and compute AUC for the BvsL discriminator



Absolute values

Ratios FGSM / raw

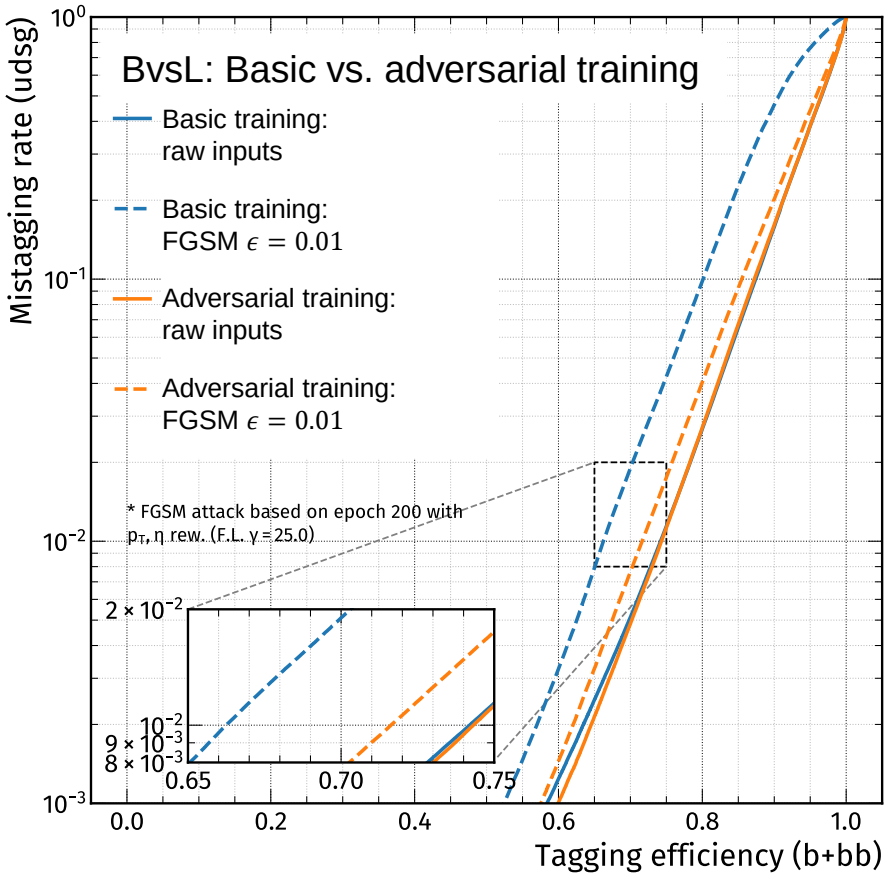


- Performance on raw inputs increases slowly
- Impact of FGSM attack stays the same

→ Only a constant offset between raw and FGSM!

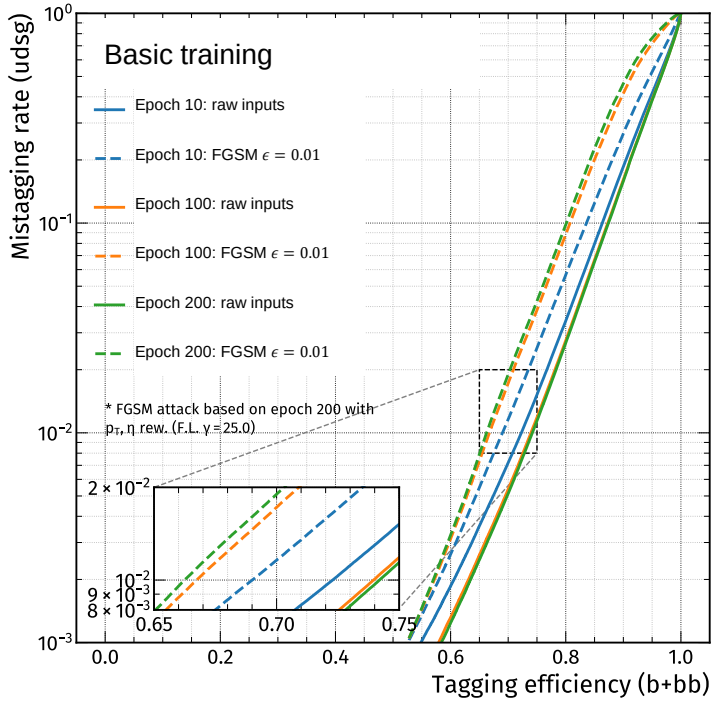
Cross-check: transferability of adversarial examples?

- Adversarial examples are created based on the basic model that has been trained on raw samples only



- The same adversarial examples also reduce the performance for the adversarially trained model
- But also in that case, the adversarial training is more robust

Also when injecting adversarial examples crafted from a single epoch (200) only, the other checkpoints stay partially susceptible



Variables (summary / utilized keys)

Inputs:

```
'Jet_eta','Jet_pt',
'Jet_DeepCSV_flightDistance2dSig','Jet_DeepCSV_flightDistance2dVal','Jet_DeepCSV_flightDistance3dSig','Jet_DeepCSV_flightDistance3dVal',
'Jet_DeepCSV_trackDecayLenVal_0',
'Jet_DeepCSV_trackDecayLenVal_1','Jet_DeepCSV_trackDecayLenVal_2','Jet_DeepCSV_trackDecayLenVal_3','Jet_DeepCSV_trackDecayLenVal_4','Jet_DeepCSV_trackDecayLenVal_5',
'Jet_DeepCSV_trackDeltaR_0','Jet_DeepCSV_trackDeltaR_1','Jet_DeepCSV_trackDeltaR_2','Jet_DeepCSV_trackDeltaR_3','Jet_DeepCSV_trackDeltaR_4','Jet_DeepCSV_trackDeltaR_5',
'Jet_DeepCSV_trackEtaRel_0','Jet_DeepCSV_trackEtaRel_1','Jet_DeepCSV_trackEtaRel_2','Jet_DeepCSV_trackEtaRel_3',
'Jet_DeepCSV_trackJetDistVal_0','Jet_DeepCSV_trackJetDistVal_1','Jet_DeepCSV_trackJetDistVal_2','Jet_DeepCSV_trackJetDistVal_3','Jet_DeepCSV_trackJetDistVal_4','Jet_DeepCSV_trackJetDistVal_5',
'Jet_DeepCSV_trackJetPt',
'Jet_DeepCSV_trackPtRatio_0','Jet_DeepCSV_trackPtRatio_1','Jet_DeepCSV_trackPtRatio_2','Jet_DeepCSV_trackPtRatio_3','Jet_DeepCSV_trackPtRatio_4','Jet_DeepCSV_trackPtRatio_5',
'Jet_DeepCSV_trackPtRel_0','Jet_DeepCSV_trackPtRel_1','Jet_DeepCSV_trackPtRel_2','Jet_DeepCSV_trackPtRel_3','Jet_DeepCSV_trackPtRel_4','Jet_DeepCSV_trackPtRel_5',
'Jet_DeepCSV_trackSip2dSigAboveCharm',
'Jet_DeepCSV_trackSip2dSig_0','Jet_DeepCSV_trackSip2dSig_1','Jet_DeepCSV_trackSip2dSig_2','Jet_DeepCSV_trackSip2dSig_3','Jet_DeepCSV_trackSip2dSig_4','Jet_DeepCSV_trackSip2dSig_5',
'Jet_DeepCSV_trackSip2dValAboveCharm',
'Jet_DeepCSV_trackSip3dSigAboveCharm',
'Jet_DeepCSV_trackSip3dSig_0','Jet_DeepCSV_trackSip3dSig_1','Jet_DeepCSV_trackSip3dSig_2','Jet_DeepCSV_trackSip3dSig_3','Jet_DeepCSV_trackSip3dSig_4','Jet_DeepCSV_trackSip3dSig_5',
'Jet_DeepCSV_trackSip3dValAboveCharm',
'Jet_DeepCSV_trackSumJetDeltaR','Jet_DeepCSV_trackSumJetEtRatio',
'Jet_DeepCSV_vertexCategory','Jet_DeepCSV_vertexEnergyRatio','Jet_DeepCSV_vertexJetDeltaR','Jet_DeepCSV_vertexMass',
'Jet_DeepCSV_jetNSecondaryVertices','Jet_DeepCSV_jetNSelectedTracks','Jet_DeepCSV_jetNTracksEtaRel','Jet_DeepCSV_vertexNTracks',
```

For comparison with DeepCSV:

```
'Jet_btagDeepB_b','Jet_btagDeepB_bb','Jet_btagDeepC','Jet_btagDeepL',
```

Creating the truth outputs was done with:

```
'Jet_nBHadrons','Jet_hadronFlavour'
```

Input variables (details) [1]

Table 2.2: Global jet variables. 13 properties that are stored on a per-jet basis have been selected. The definition of "jet" variable includes those features that are related to tracks, but are only available once, not for several tracks due to their global meaning. Adapted from Ref. [8].

Short name	Description
Jet η	The first, purely kinematic variable stores the pseudorapidity of the jet. (`Jet_eta`)
Jet p_{T}	Also, the transverse momentum of the jet is considered as an input variable. (`Jet_pt`)
Track Jet p_{T}	This is the transverse momentum of the jet, obtained from the sum of the transverse momenta of the tracks only. (`Jet_DeepCSV_trackJetPt`)
Track SIP 2D (3D) Val (Sig) Above Charm	In general, the impact parameter (IP) vector is the vector pointing from the primary vertex to the point of closest approach of the track. The signed impact parameter (SIP) can be obtained by taking the modulus of this quantity, but keeping the information about whether the angle between the IP vector and the jet axis is smaller than $\pi/2$ (SIP > 0) or not (SIP < 0). This can be done in three dimensions, in two dimensions (transverse to the beam line), or along the beam line in one dimension. Then, for a given track, the value as well as the significance (SIP$/\sigma_{\mathrm{SIP}}$) is stored. For the four variables considered here, the two-dimensional (2D) and three-dimensional (3D) SIP value and significance are taken for the first track that raises the combined invariant mass above the threshold of the charm quark mass (1.5 GeV). (`Jet_DeepCSV_trackSip2dValAboveCharm`, `Jet_DeepCSV_trackSip2dSigAboveCharm`, `Jet_DeepCSV_trackSip3dValAboveCharm`, `Jet_DeepCSV_trackSip3dSigAboveCharm`)
Track Sum Jet ΔR	First, the four-momentum vectors of the tracks are summed to get a total four-momentum. Then, the angular distance between this summed four-momentum and the jet axis is computed, using $\Delta R = \sqrt{(\Delta\phi)^2 + (\Delta\eta)^2}$. (`Jet_DeepCSV_trackSumJetDeltaR`)

Input variables (details) [2]

Track Sum Jet E_T Ratio	For this variable, the transverse energy of the total summed four-momentum of the tracks is divided by the transverse energy of the jet. (`Jet_DeepCSV_trackSumJetEtRatio`)
Vertex Category	With the vertex category, three cases can be defined that describe the result of the secondary vertex reconstruction algorithm. (`Jet_DeepCSV_vertexCategory`) • RecoVertex: at least one secondary vertex has been reconstructed in the jet. • PseudoVertex: the reconstruction has not found a secondary vertex and no fit is performed, but there are at least two tracks with a 2D impact parameter significance greater than two, whose combined invariant mass is 50 MeV away from the mass of the K_S^0. • NoVertex: this label is given to all jets that do not belong to one of the previous categories.
Jet N Secondary Vertices	This quantity lists the number of secondary vertices in the jet (of the RecoVertex category). (`Jet_DeepCSV_jetNSecondaryVertices`)
Jet N Selected Tracks	The number of selected tracks in the jet counts the number of tracks that remain after introducing special criteria related to the quality of the track reconstruction, criteria to reduce contamination from (for example) K_S^0 or Λ hadrons, or to minimize the contribution from pileup. The imposed conditions involve properties like a minimum p_T, a required number of pixel hits, a maximum threshold for the χ^2 of the track fit, or utilize the impact parameter or the angular distance between the track and the jet axis. (`Jet_DeepCSV_jetNSelectedTracks`)
Jet N Tracks η_{rel}	This is the number of tracks for which the property η_{rel} is available (has been computed out of the selected tracks). (`Jet_DeepCSV_jetNTracksEtaRel`)

Input variables (details) [3]

Table 2.3: Track variables. The tracks are ordered by the significance of their 2D signed impact parameter (as defined in Section 1.4.2 and Table 2.2). Then, the six highest ranked tracks (the first one having the highest 2D SIP significance) are selected and further described by seven properties. Another eighth property (Track η_{rel}) is only included for the first four tracks of that ranking. Adapted from Ref. [8].

Short name	Description
Track Decay Len Val	This is the distance between the primary vertex and the track, at the point of closest approach between the track and the jet axis. In the data set, this feature appears for the first six tracks. (`Jet_DeepCSV_trackDecayLenVal_0` to 5)
Track ΔR	For this input variable, the angular distance $\Delta R = \sqrt{(\Delta\phi)^2 + (\Delta\eta)^2}$ between the track and the jet axis is stored (six times). (`Jet_DeepCSV_trackDeltaR_0` to 5)
Track Jet Dist Val	To obtain the following quantity (for the first six tracks), the distance between the track and the jet axis at their point of closest approach has been calculated. (`Jet_DeepCSV_trackJetDistVal_0` to 5)
Track $p_{\text{T,rel}}$	The next feature constitutes the track momentum, perpendicular to the jet axis (in other words: the track p_{T} relative to the jet axis), available for the first six tracks. (`Jet_DeepCSV_trackPtRel_0` to 5)
Track $p_{\text{T,rel}}$ Ratio	Based on the previous quantity, this feature now uses the track's $p_{\text{T,rel}}$ and divides this value by the magnitude of the track momentum vector, again for the first six tracks. (`Jet_DeepCSV_trackPtRatio_0` to 5)
Track SIP 2D (3D) Sig	In the way it is used here, only the significance of the signed impact parameter (2D or 3D) is saved for the first six tracks (ordered by 2D SIP significance). (`Jet_DeepCSV_trackSip2dSig_0` to 5, `Jet_DeepCSV_trackSip3dSig_0` to 5)
Track η_{rel}	This variable is defined as the pseudorapidity of the track relative to the jet axis and is stored for maximally four tracks, ranked by 2D SIP significance as described above. (`Jet_DeepCSV_trackEtaRel_0` to 3)

Input variables (details) [4]

Table 2.4: Secondary vertex variables. Additionally, there are eight input variables related to the secondary vertex (SV) with the smallest uncertainty on its (2D) flight distance (if such a secondary vertex exists, i.e. if a secondary vertex could be reconstructed → vertex category RecoVertex). The more specific definition of the chosen secondary vertex is omitted from now on, but always meant implicitly. Adapted from Ref. [8].

Short name	Description
Flight Distance 2D (3D) Val (Sig)	With these quantities, the 2D (or 3D) distances between the primary and secondary vertex are taken into account. The 2D (or 3D) significances are used in addition to the values. (`Jet_DeepCSV_flightDistance2dVal`, `Jet_DeepCSV_flightDistance2dSig`, `Jet_DeepCSV_flightDistance3dVal`, `Jet_DeepCSV_flightDistance3dSig`)
Vertex Energy Ratio	This is the energy of the secondary vertex, divided by the energy of the total summed four-momentum vector of the selected tracks. (`Jet_DeepCSV_vertexEnergyRatio`)
Vertex Jet ΔR	Here, the ΔR between the flight direction of the secondary vertex (or the total summed four-momentum of the selected tracks) and the jet axis is stored for vertex category RecoVertex (PseudoVertex). (`Jet_DeepCSV_vertexJetDeltaR`)
Vertex Mass	This is the corrected mass of the secondary vertex for jets in the RecoVertex category, which takes care of the observed difference between the flight direction and the momentum of the SV when not all particle were correctly reconstructed or associated with the SV. The correction uses $\sqrt{M_{\mathrm{SV}}^2 + p^2 \sin^2\theta} + p \sin\theta$, with the invariant mass of the tracks associated with the SV, M_{SV}, the SV momentum p and the angle θ between the SV momentum and the vector pointing from the PV to the SV [8]. As the mass of the SV is directly related to the mass of the decaying hadron, this quantity offers high discriminating power. For jets in PseudoVertex category, this variable lists the invariant mass of the total summed four-momentum vector of the selected tracks. (`Jet_DeepCSV_vertexMass`)
Vertex N Tracks	For vertex category RecoVertex, this property counts the number of tracks associated with the secondary vertex (no. of selected tracks for cat. PseudoVertex). (`Jet_DeepCSV_vertexNTracks`)

Targets (details)

- `Jet_hadronFlavour == 5`
  At least one b hadron[2] is clustered inside the jet. This can be further specified by counting the number of b hadrons.
  - `Jet_nBHadrons <= 1` (b jets)
    Less than two b hadrons have been clustered in the jet (together with the already imposed condition on the hadron flavour, this is exactly one b hadron).
  - `Jet_nBHadrons > 1` (bb jets)
    More than one b hadron has been found after the jet clustering took place. These likely originate from gluon splitting, but their properties are mainly defined by the two (heavy-flavour) b hadrons, which explains the assigment to this main category.

  For the cases where differentiating between the number of b hadrons is not necessary, both subcategories can be combined to include b and bb jets (sometimes abbreviated with a capital B in various figures showing the performance of the classifier).
- `Jet_hadronFlavour == 4` (c jets)
  No b hadrons, but at least one c hadron results from the jet clustering.
- `Jet_hadronFlavour != 5 && Jet_hadronFlavour != 4` (udsg/light/l jets)
  Neither b, nor c hadrons are present in the jet. Thus, these jets are defined as udsg jets, including light-flavour, quark-initiated jets and those originating from gluons. Due to the small mass of uds quarks (see Fig. 1.1), and with gluons being massless, when discriminating the light jets from heavy-flavour jets, the properties of the aforementioned light jets are similar such that all four hypothetical subcategories (u, d, s, g jets) can be combined into the udsg category.

Supplementary material

Corresponding code can be found in the following repositories:

- https://git.rwth-aachen.de/annika-stein/ai_safety_2021/ (via RWTH SSO)
- https://github.com/AnnikaStein/VHcc-cTagSF (forked from https://github.com/mondalspandan/VHcc-cTagSF)

Previous talks:

- DPG, March 2021 (.pdf (at indico))
- D-CMS, September 2021 (.pdf (at indico))