# Robustness of b-Tagging algorithms for the CMS Experiment

Annika Stein

Xavier Coubez, Nikolas Frediani, Spandan Mondal,
Andrzej Novak, Alexander Schmidt

III. Physikalisches Institut A

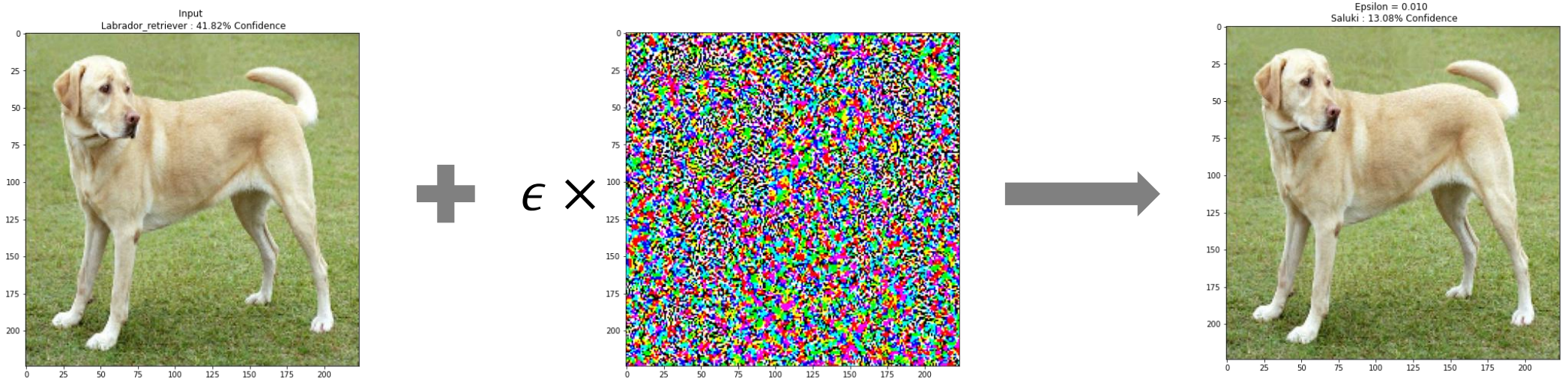
RWTH AACHEN UNIVERSITY

**FSP CMS Workshop 2021**
**Online, 23.09.2021**

# 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



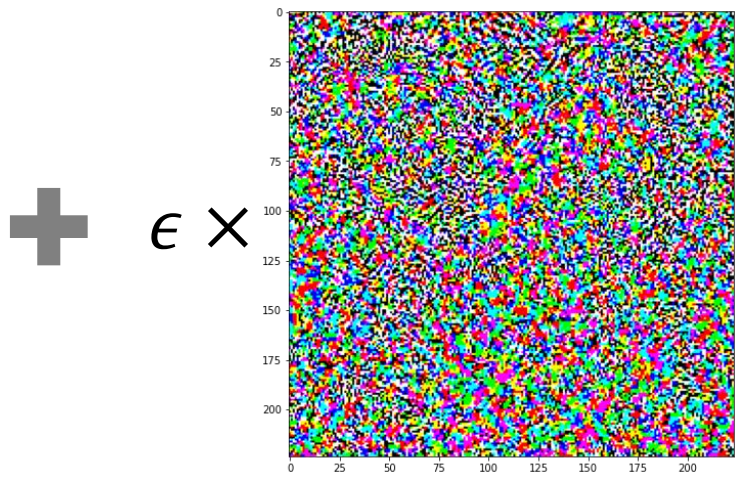

Classifier: labrador (breed of dog)

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

# AI-safety: jet heavy-flavour tagging

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




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

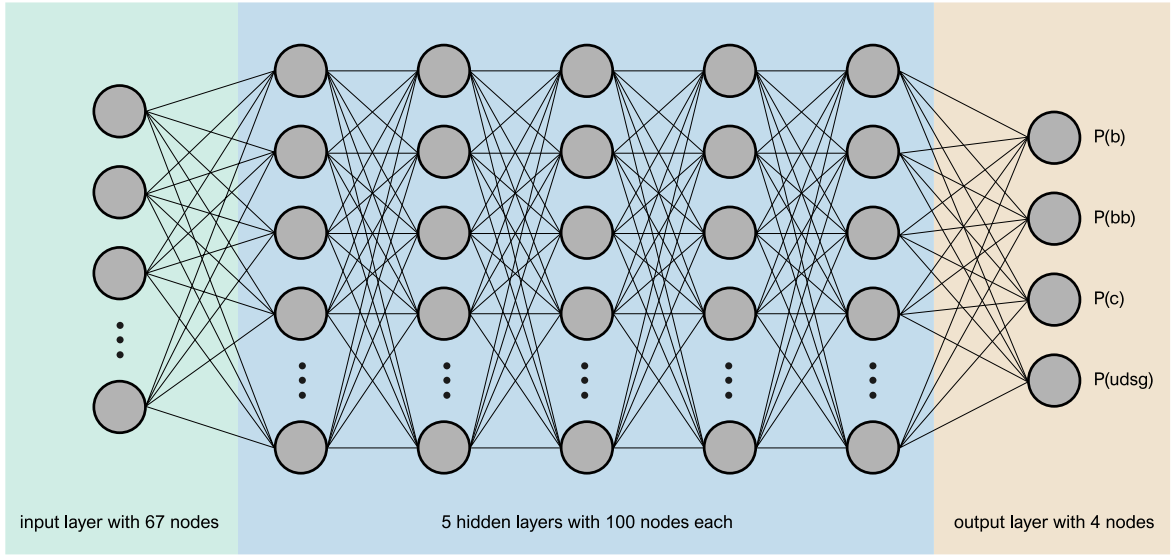
Classifier: b-jet

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

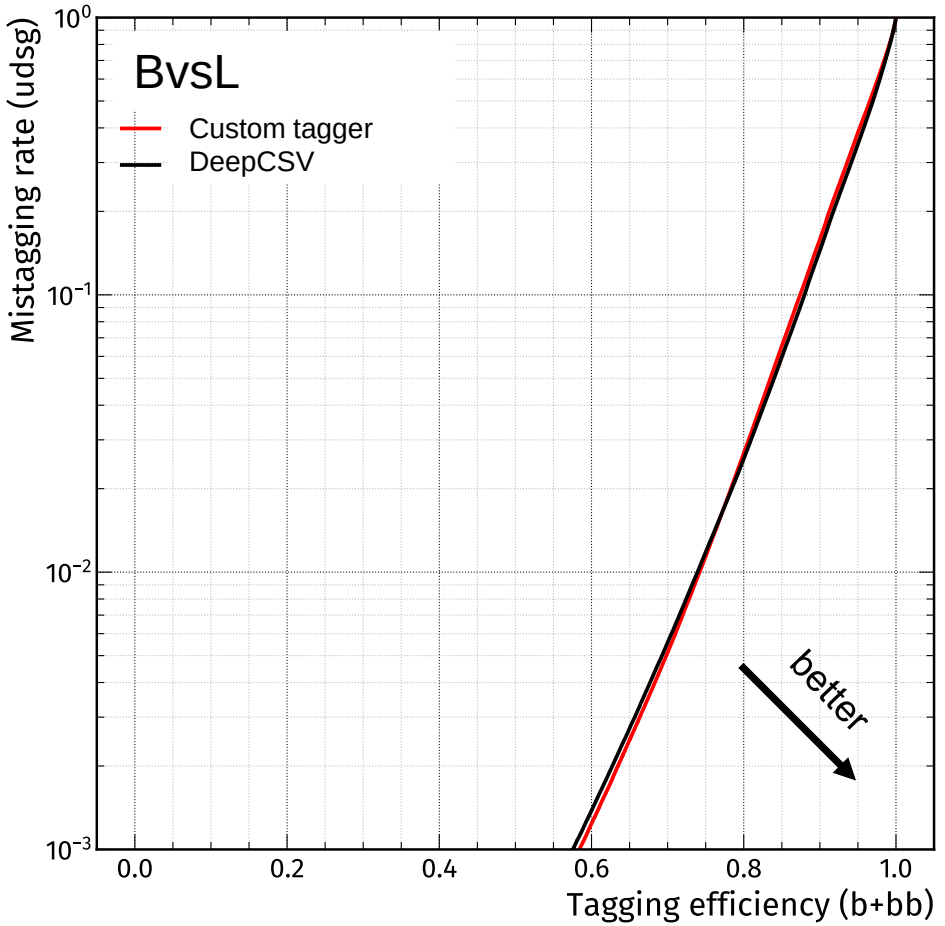



- DNN outputs are used to calculate discriminators, e.g.

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

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

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



# Part 2

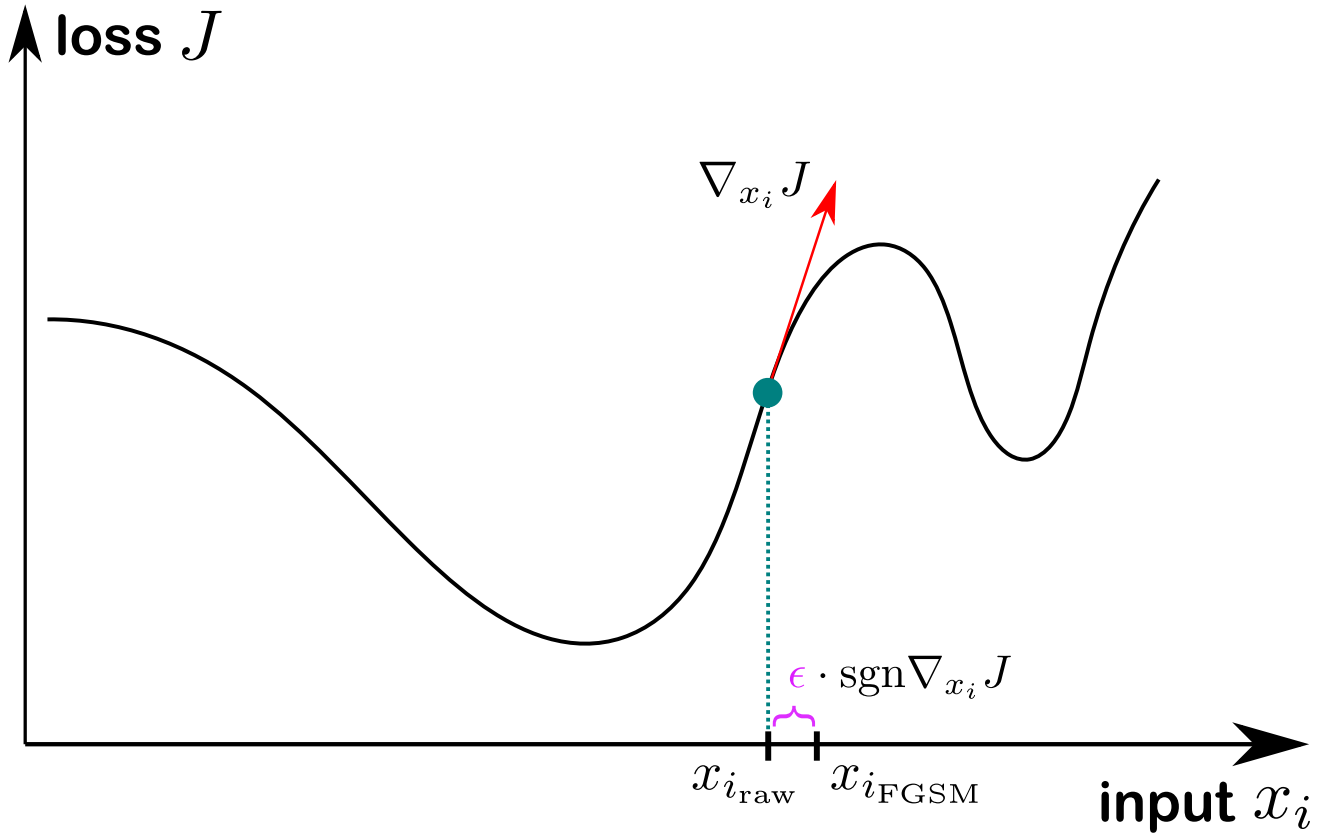
- Introduction AI-safety: Application to Neural Networks & CMS b-Tagging
- Adversarial Attacks: Fast Gradient Sign Method
- Improving Robustness with Adversarial Training

# Fast Gradient Sign Method (FGSM)

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

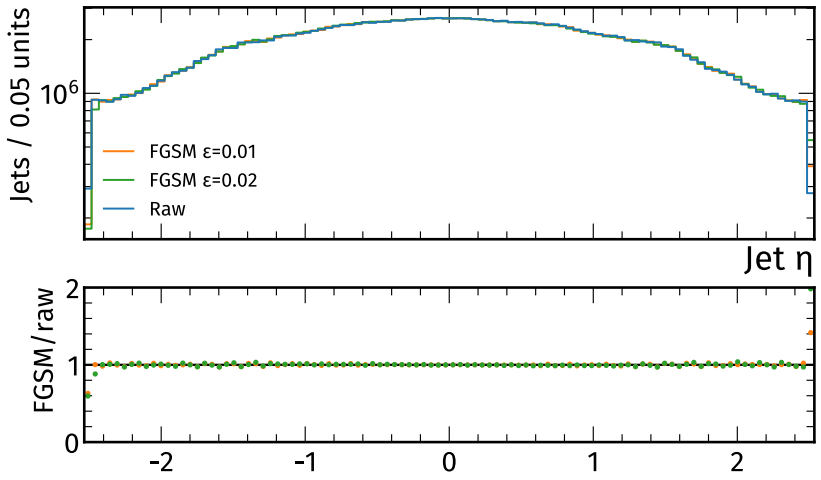
$$x_{FGSM} = x_{raw} + \epsilon \cdot \mathrm{sgn}(\nabla_{x_{raw}} J(y, x_{raw}))$$

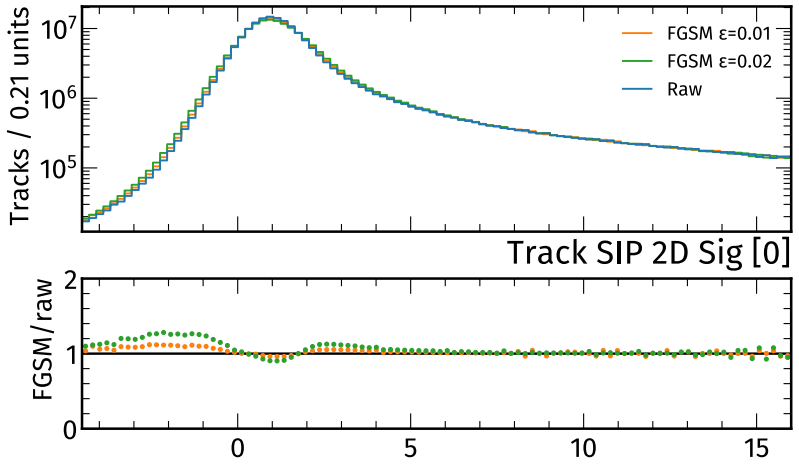
- Small $\epsilon$ are enough to affect the performance drastically

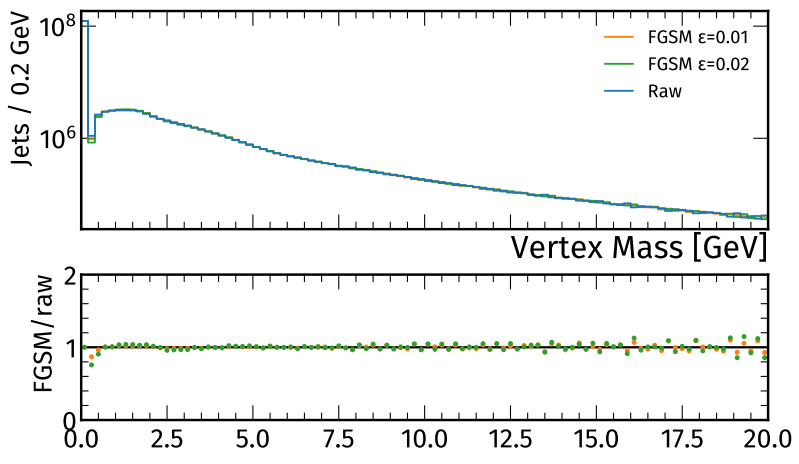


# Distorted Inputs (FGSM)

- Apply FGSM attack to test sample




- Current setup uses the same $\epsilon$ 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

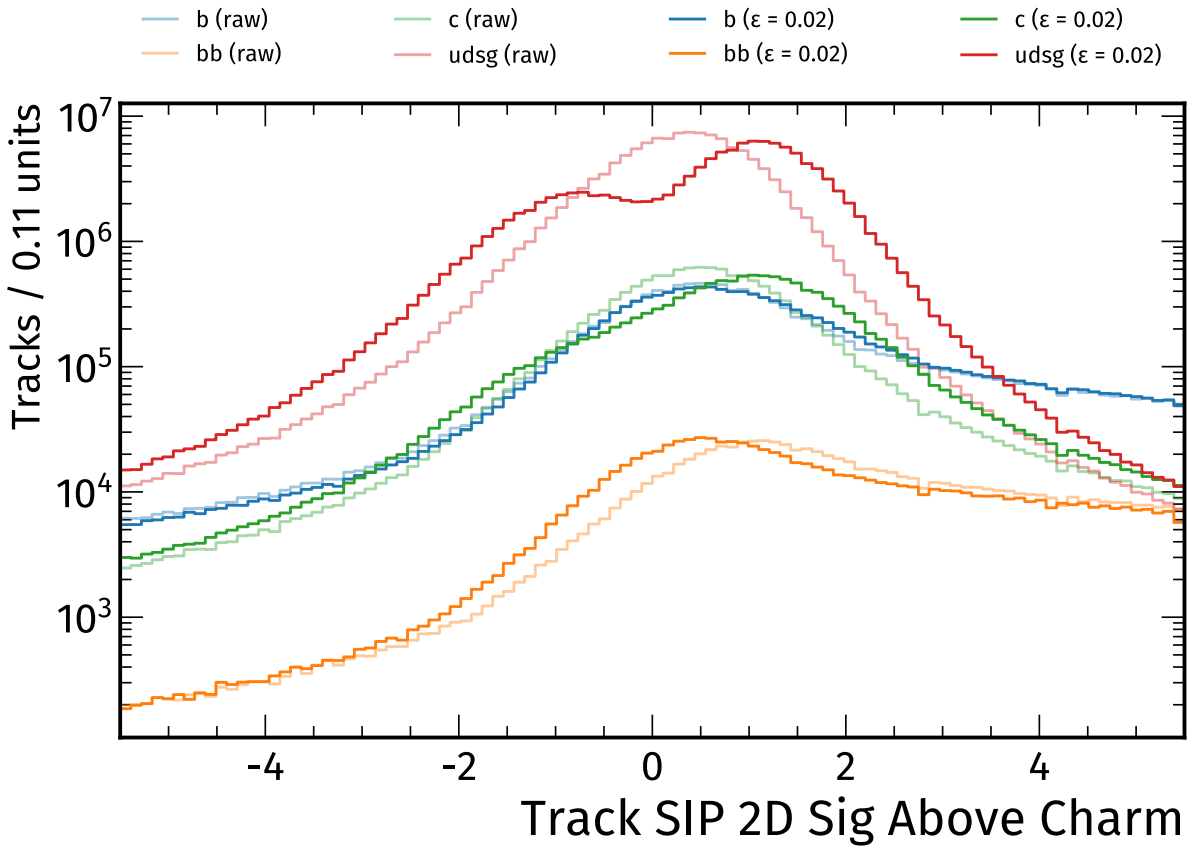
We leave this for future studies, concentrate on two parameters for now:

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

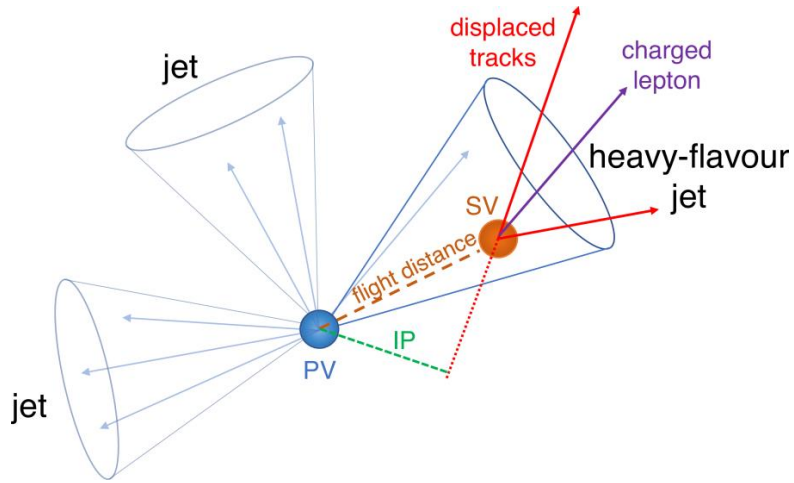
larger $\epsilon = 0.02$ → to visualize flavour dependency

# 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

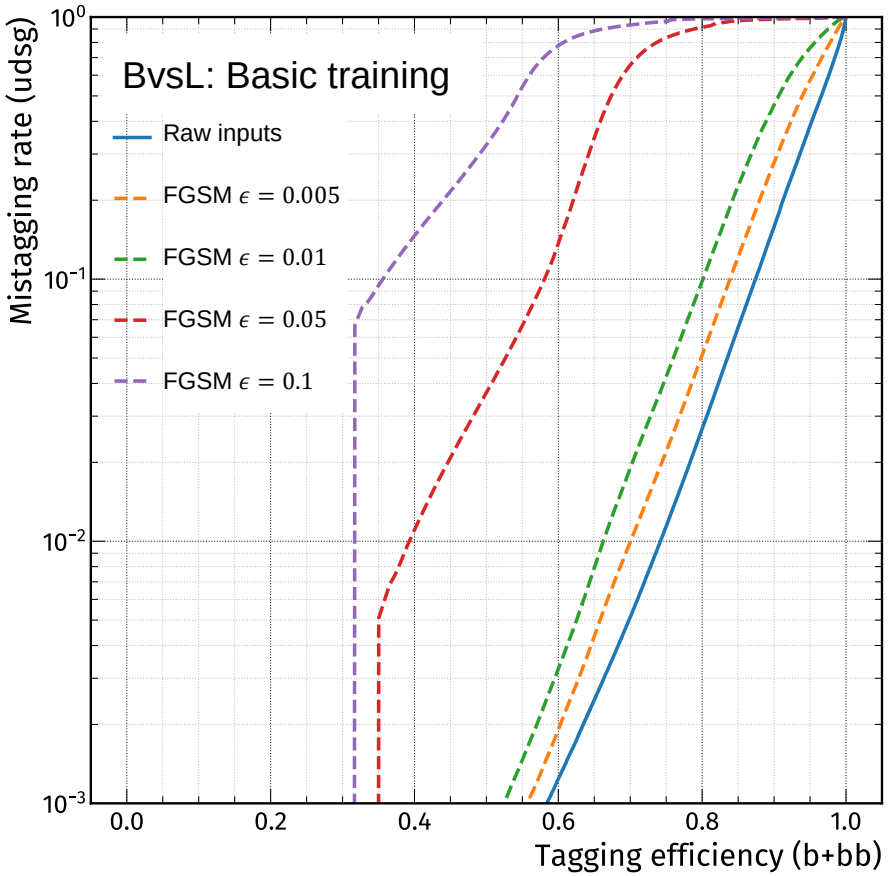



- b(b)-jets move to the negative end, light-jets more to the positive side (c-jets ≈ in between)

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

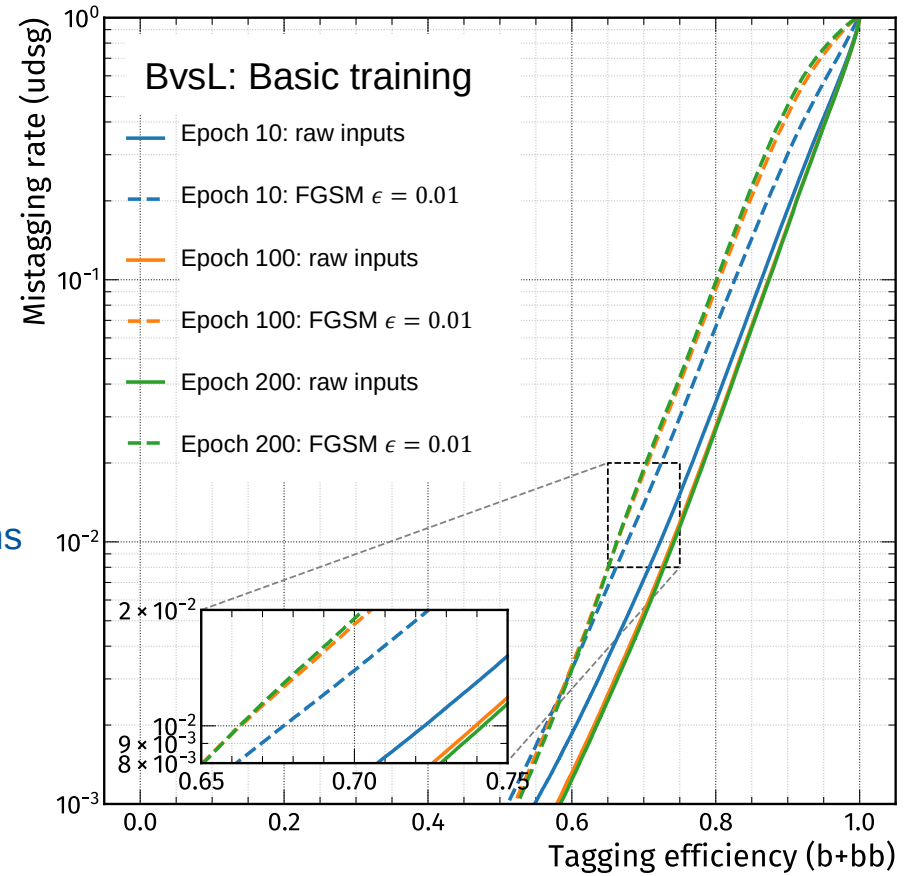
# ROC curves (B versus L)

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




Same epoch, different $\epsilon$

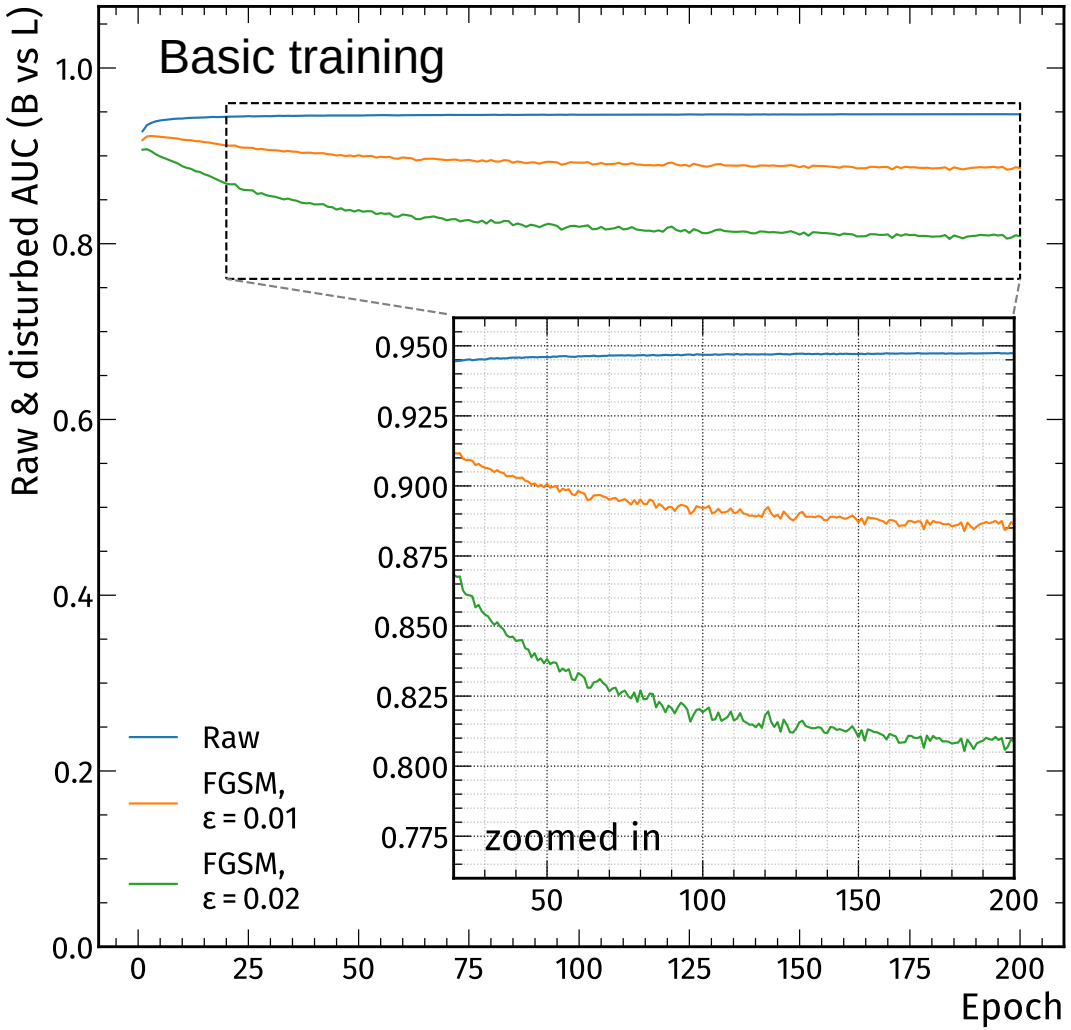
Same $\epsilon$, different epochs




- Larger parameters for $\epsilon$ 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

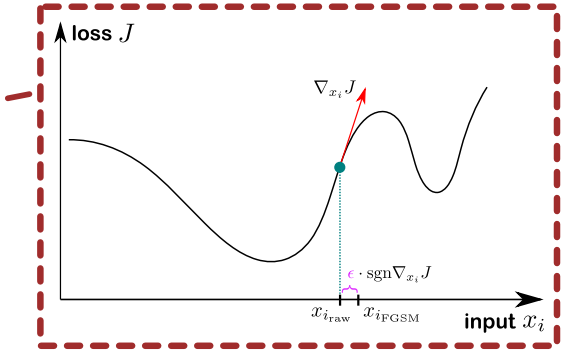
→ Tradeoff between performance and robustness!

# 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

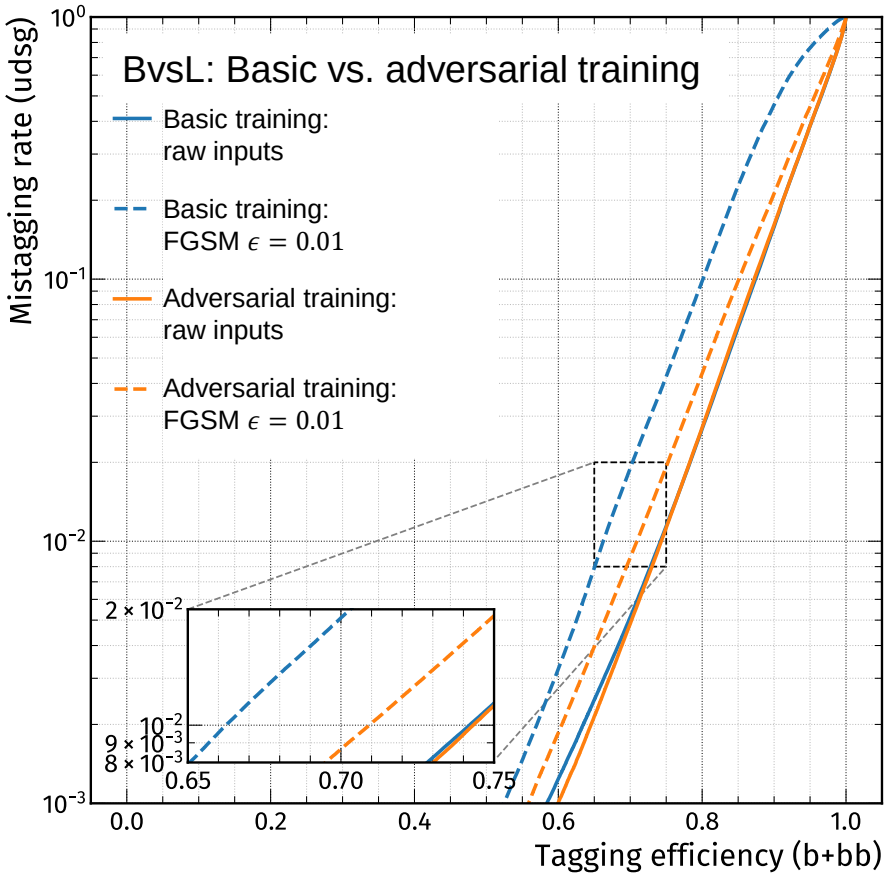



- Inject distorted inputs already during training phase
- Idea: model never sees raw inputs, so it should less likely learn simulation-specific artefacts

→ Improve robustness
Better generalization

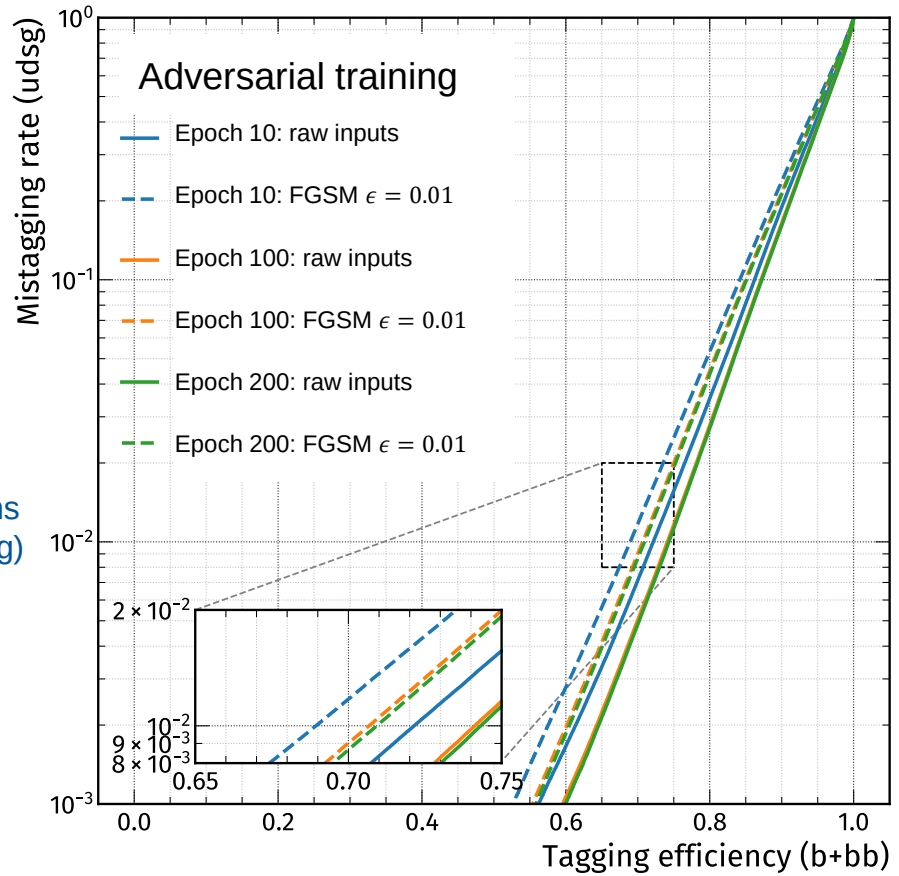
# ROC curves (B versus L)

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




Same epoch and $\epsilon$, comparing basic with adversarial training

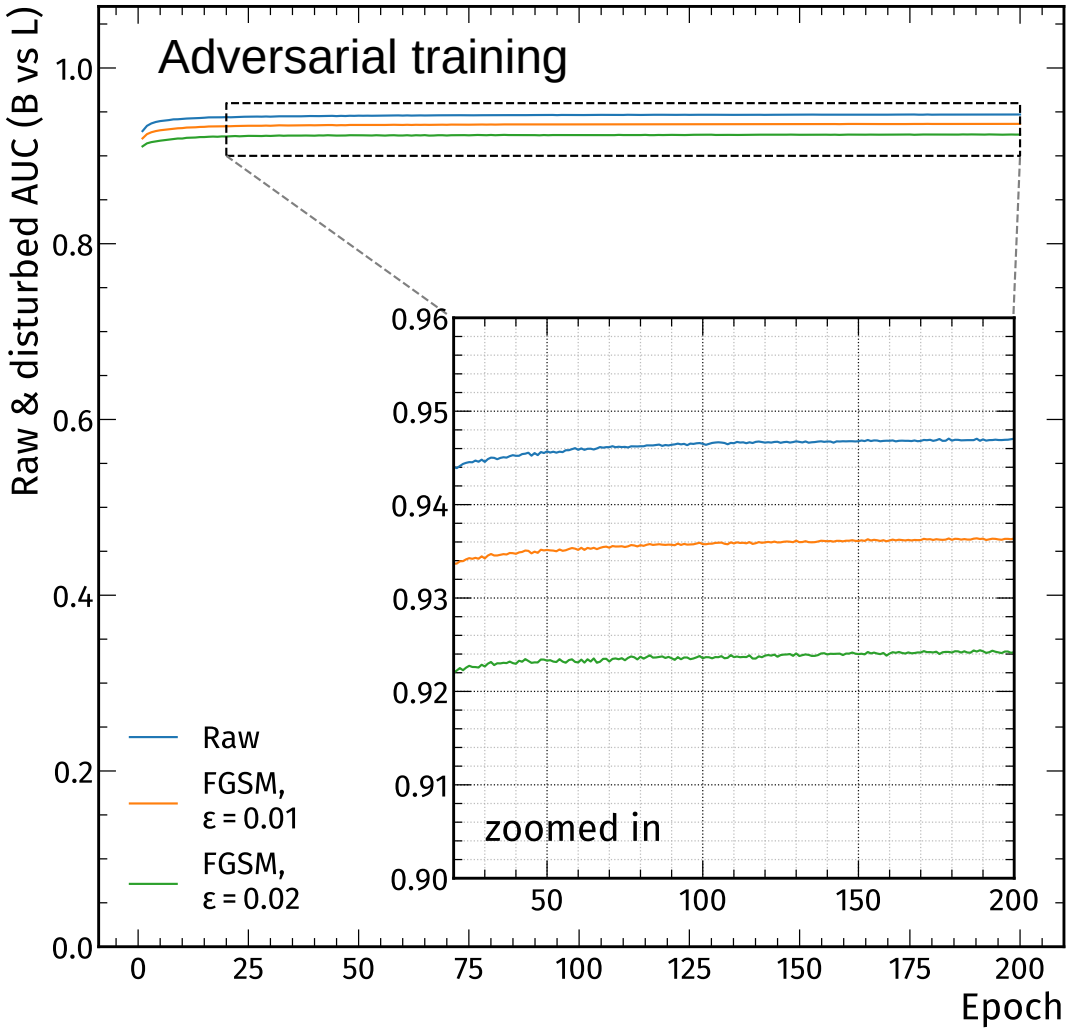
Same $\epsilon$, different epochs (only adversarial training)




- 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

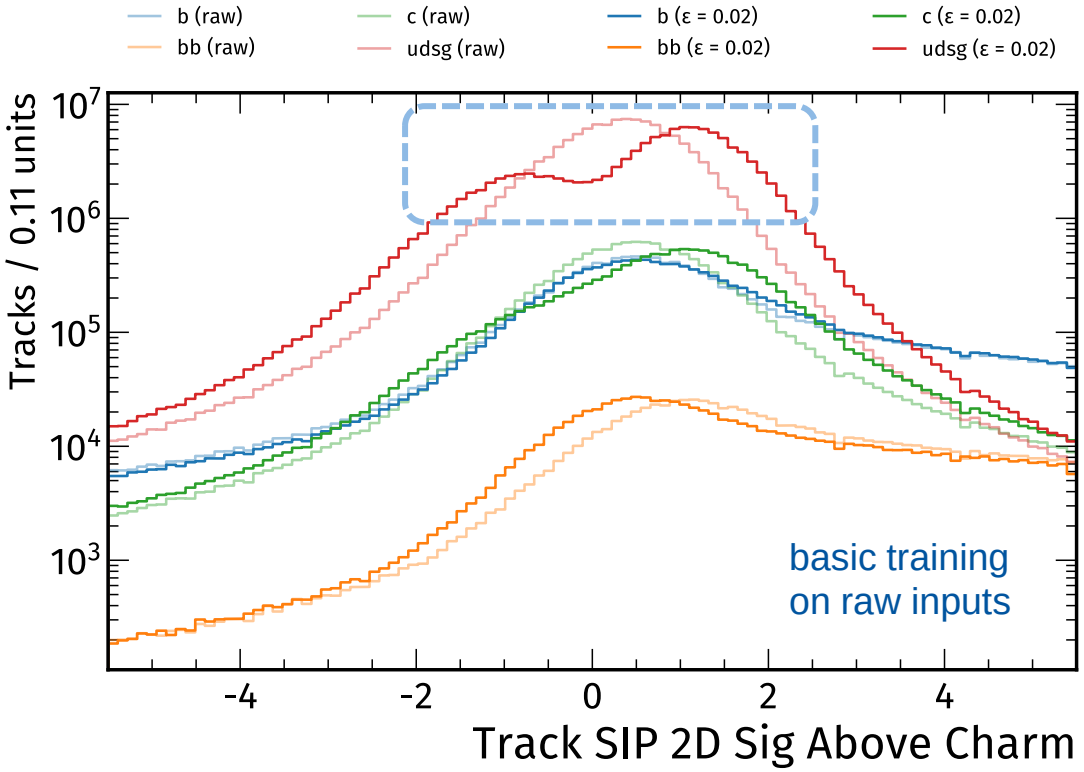


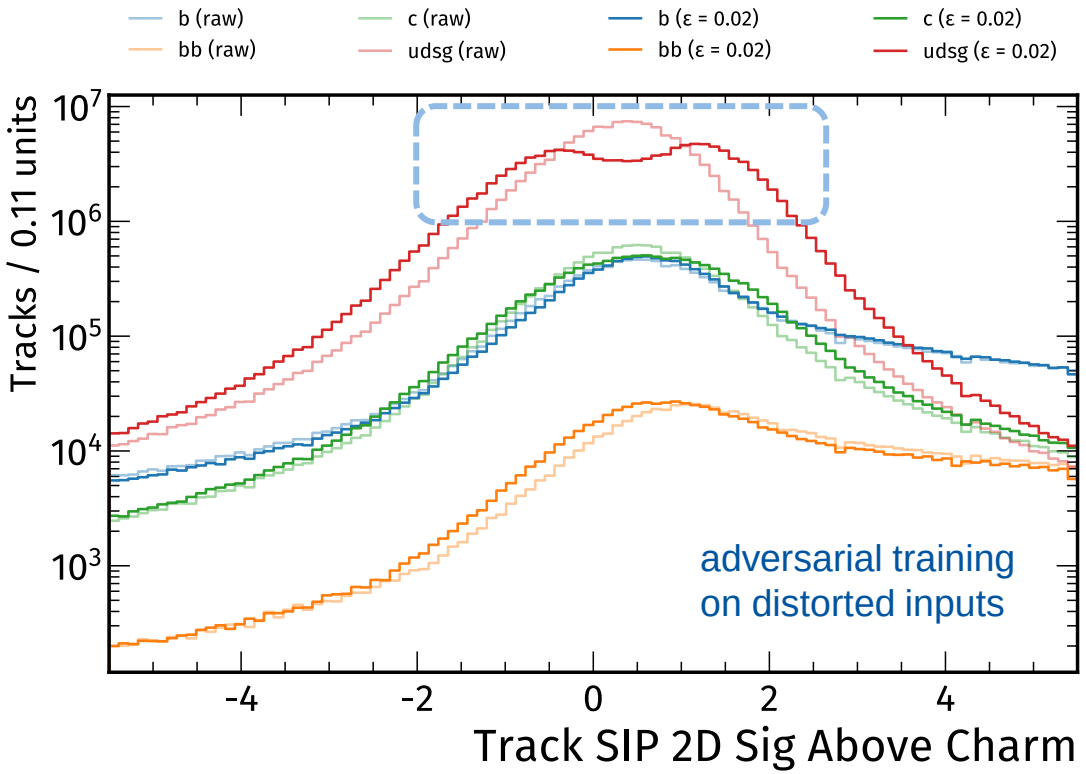

- Performance on raw inputs increases slowly
- Impact of FGSM attack stays the same

→ 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:




A geometrical problem

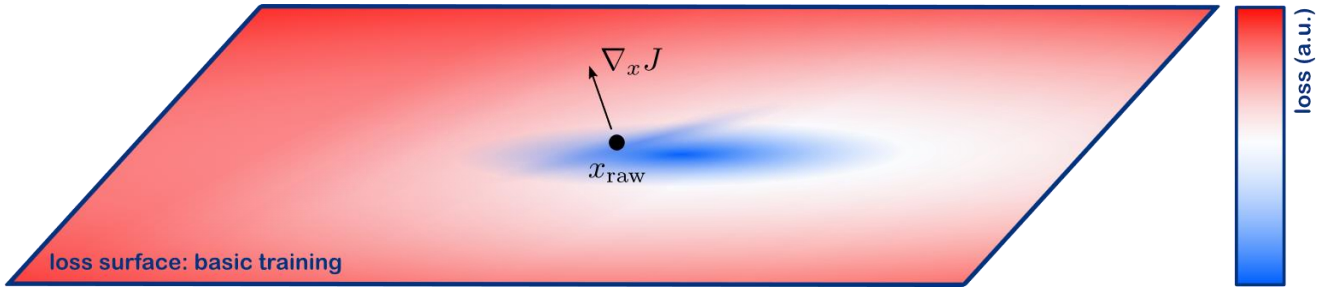
Basic training ⊗ FGSM → asymmetric shapes

Adversarial training ⊗ FGSM → symmetric shapes

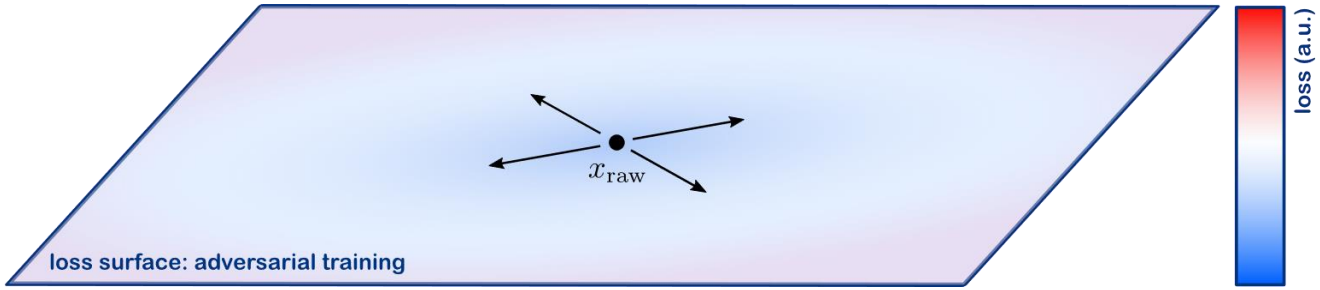
- Crafting adversarial inputs for adversarially trained model is almost like ‚coin-flipping‘

# 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

# Conclusions

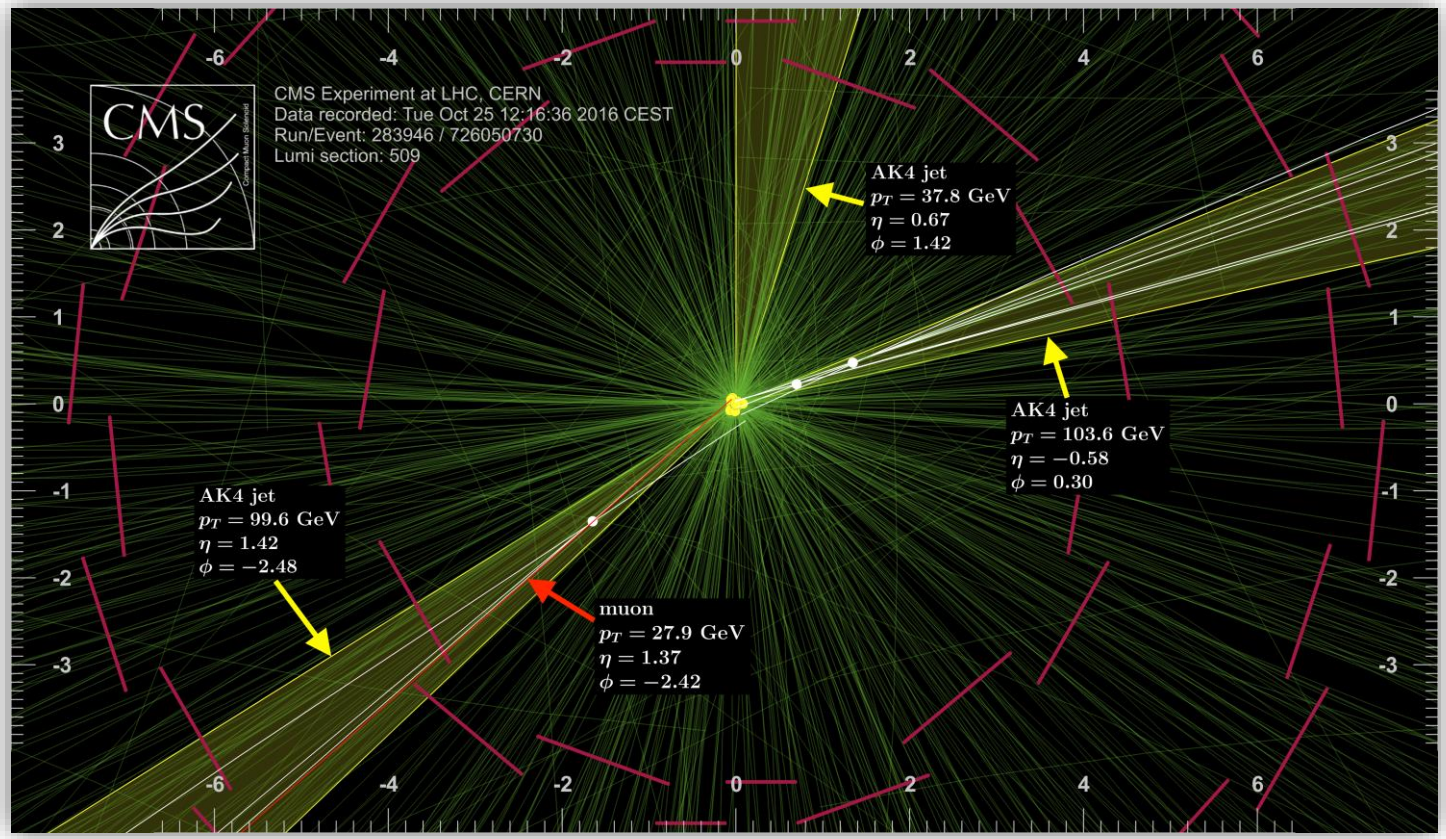
- AI-safety studies for jet flavour tagging have been done for the first time: almost invisible disturbances of the inputs result in noticeable performance drops → applicable & concerning for HEP
- Results are consistent with expectations: model performance improves with increasing number of epochs, but susceptibility towards adversarial attacks becomes larger as well
- Robustness improves with adversarial training: less vulnerable against adversarial attacks, mismodellings in simulation would less likely hinder generalization / application to data

- Next steps could be:
  - Improving the current FGSM attack: include constraints for each feature
  - Investigating the influence on the scale factors
  - Applying other attacks of higher complexity (& corresponding defense strategies)
  - Moving to other taggers like DeepJet with high-dimensional, low-level feature space

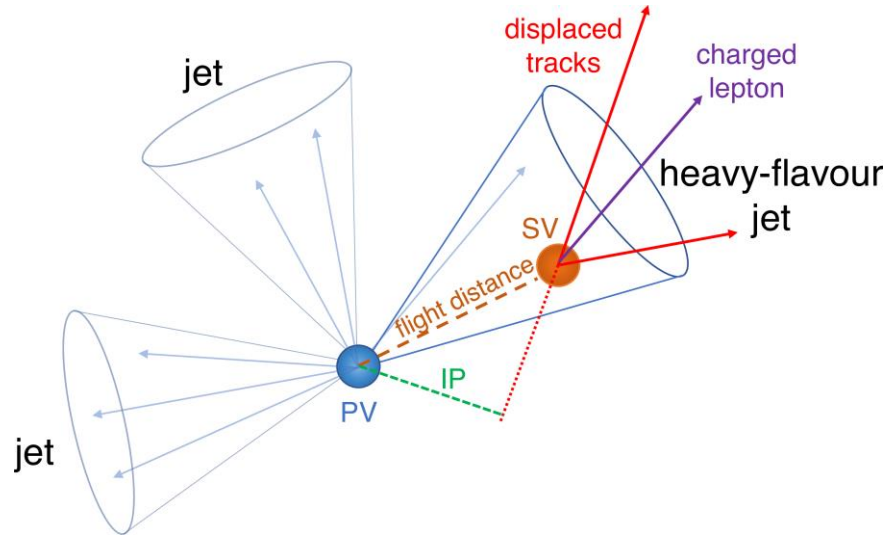
# Backup

# Application: jet heavy-flavour tagging at CMS

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





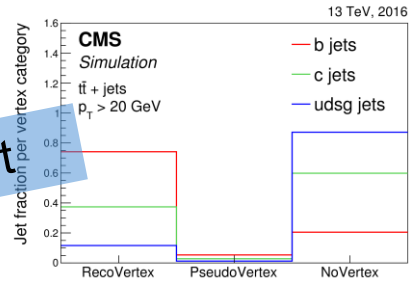

<u>Heavy-flavour jets (b & c-jets)</u>

- 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)

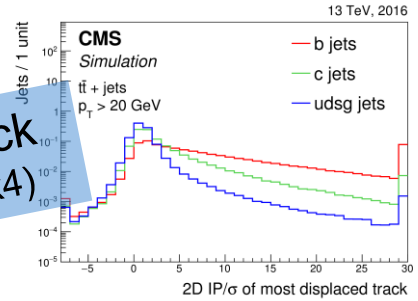
# b-Tagging algorithms for CMS: DeepCSV

Inputs

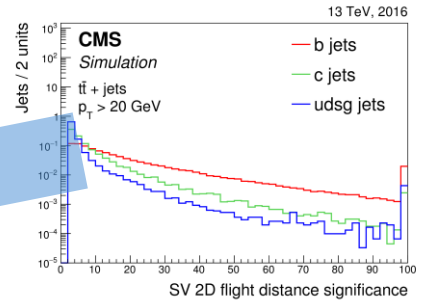
13x Jet




46x Track
(7x6 + 1x4)

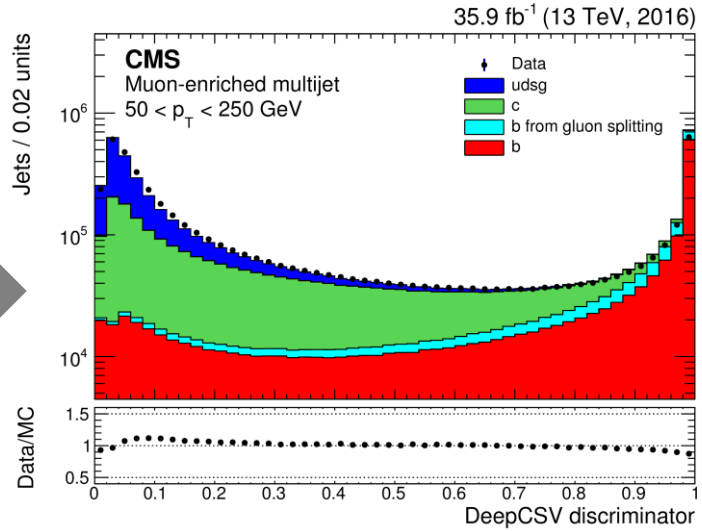



8x SV




Deep Neural Network
(DNN)

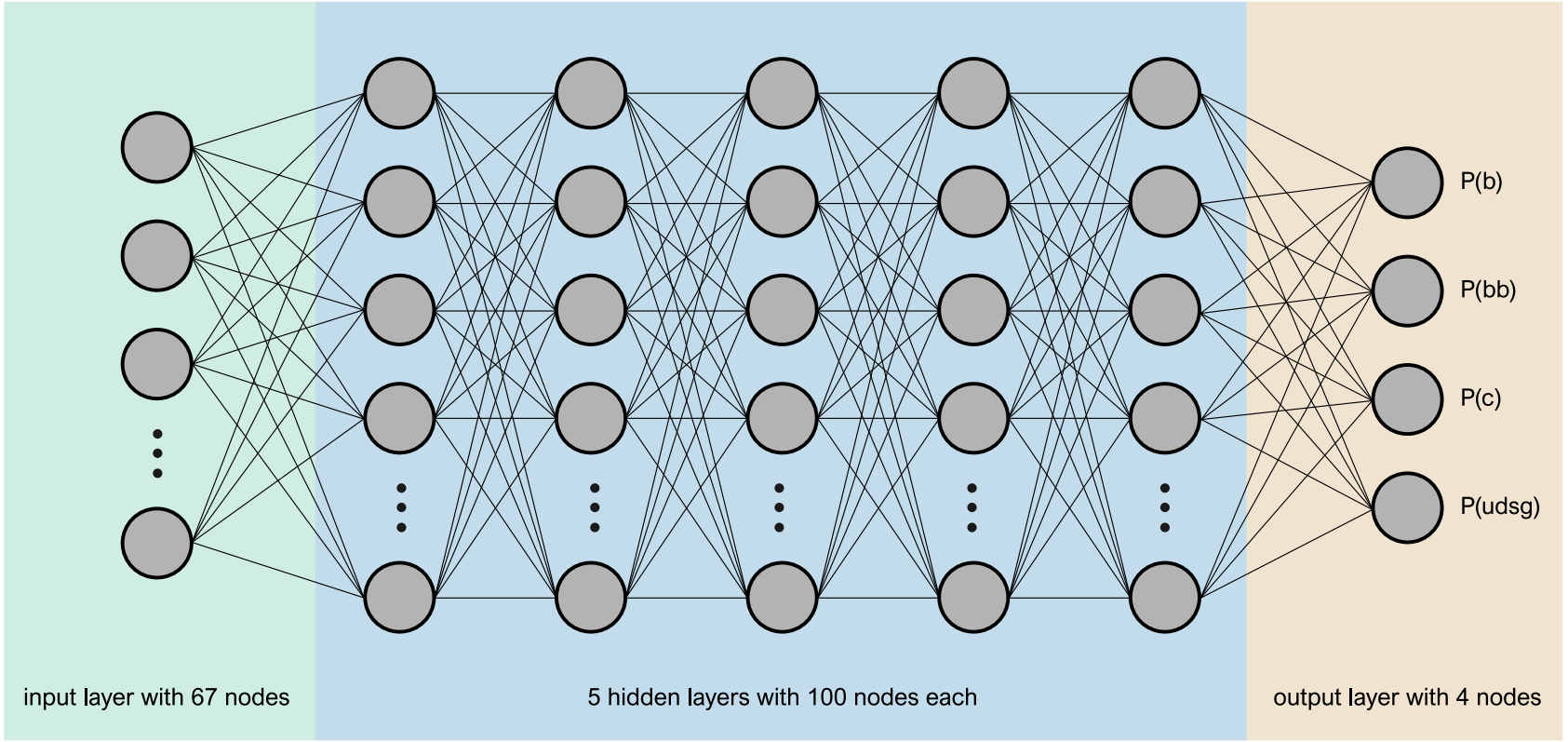
Outputs



# Model architecture

Use a fully connected deep neural network

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




Details

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

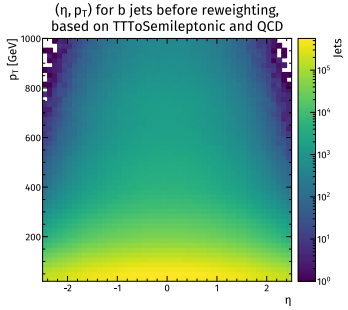
| b | bb | c | usdg |
|---|---|---|---|
| 9.7% | 0.7% | 8.1% | 81.5% |

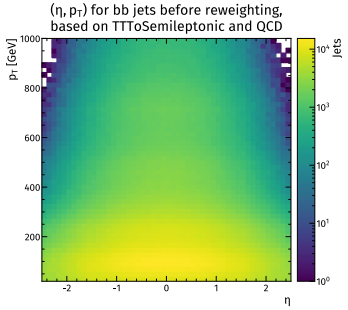
Hyperparameters

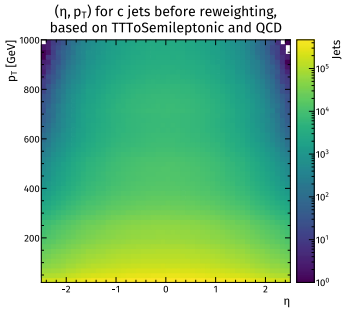
Batch size: 1M
Adam optimizer, learning rate decay initialized with 0.0001
200 epochs
$\gamma = 25$ (focusing parameter)
Defaults slighly below minima of input distributions

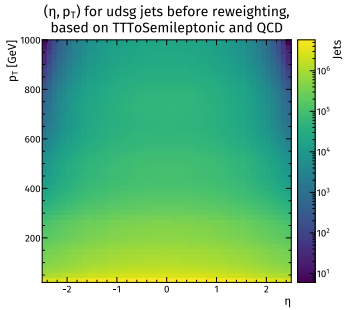
# Reweighting

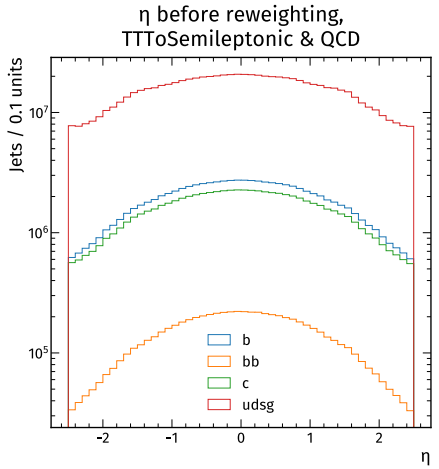
## ($p_T$,$\eta$) distribution before reweighting

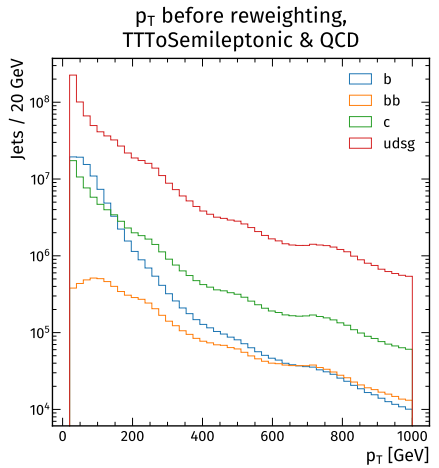


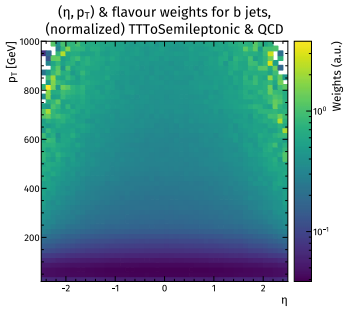



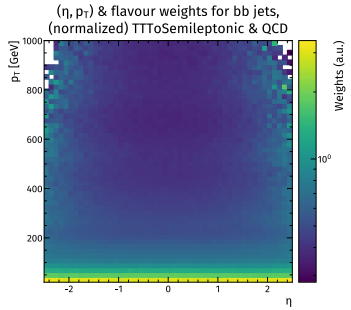



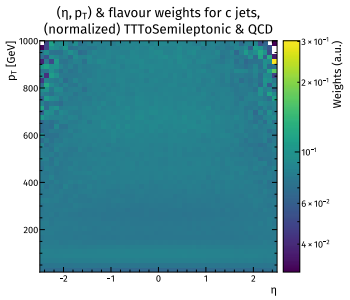



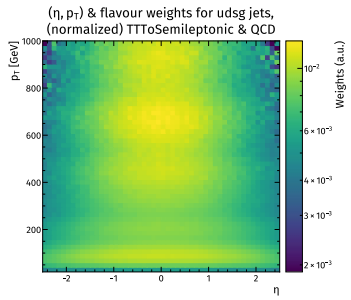



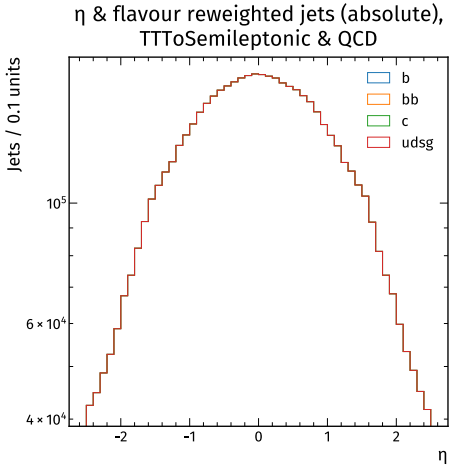

## multiply with weights

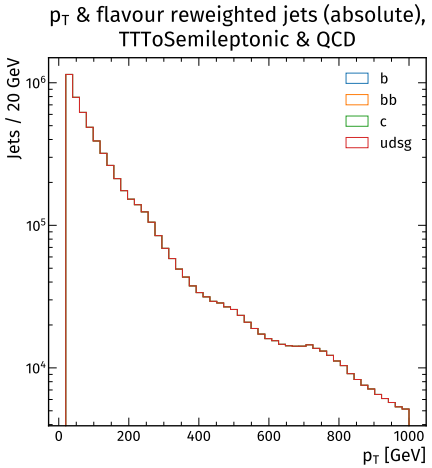


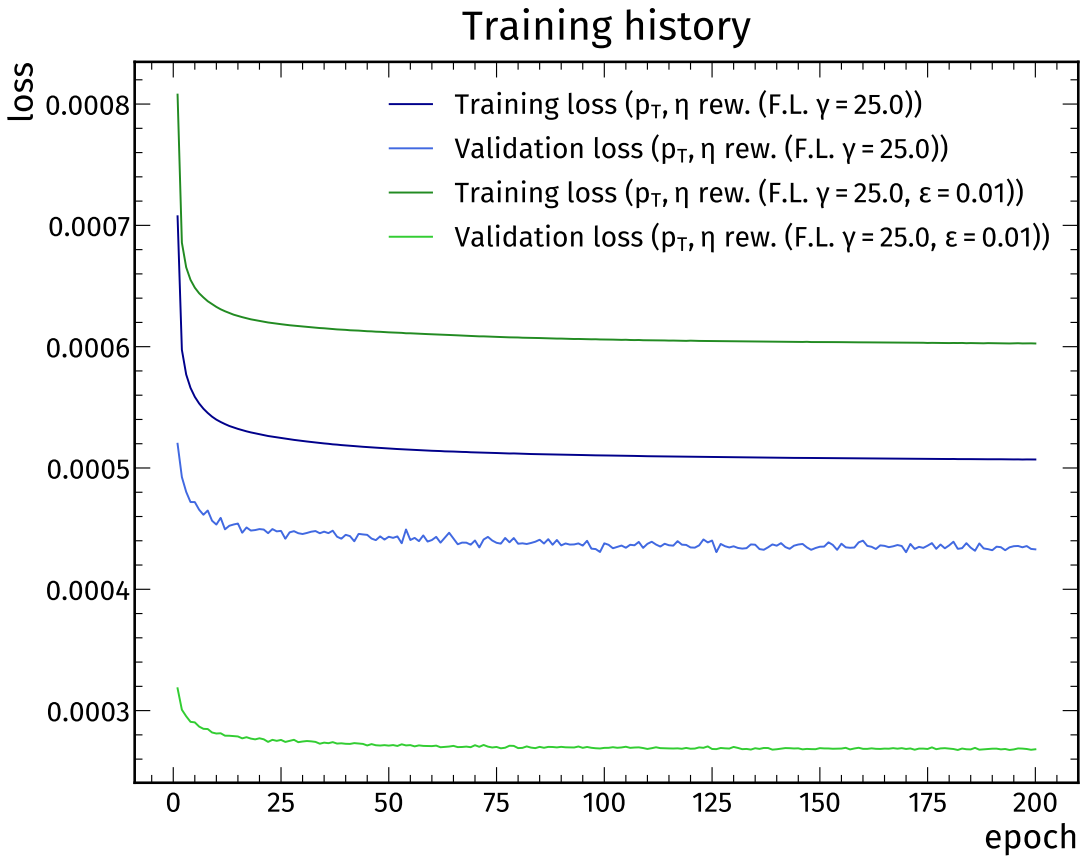



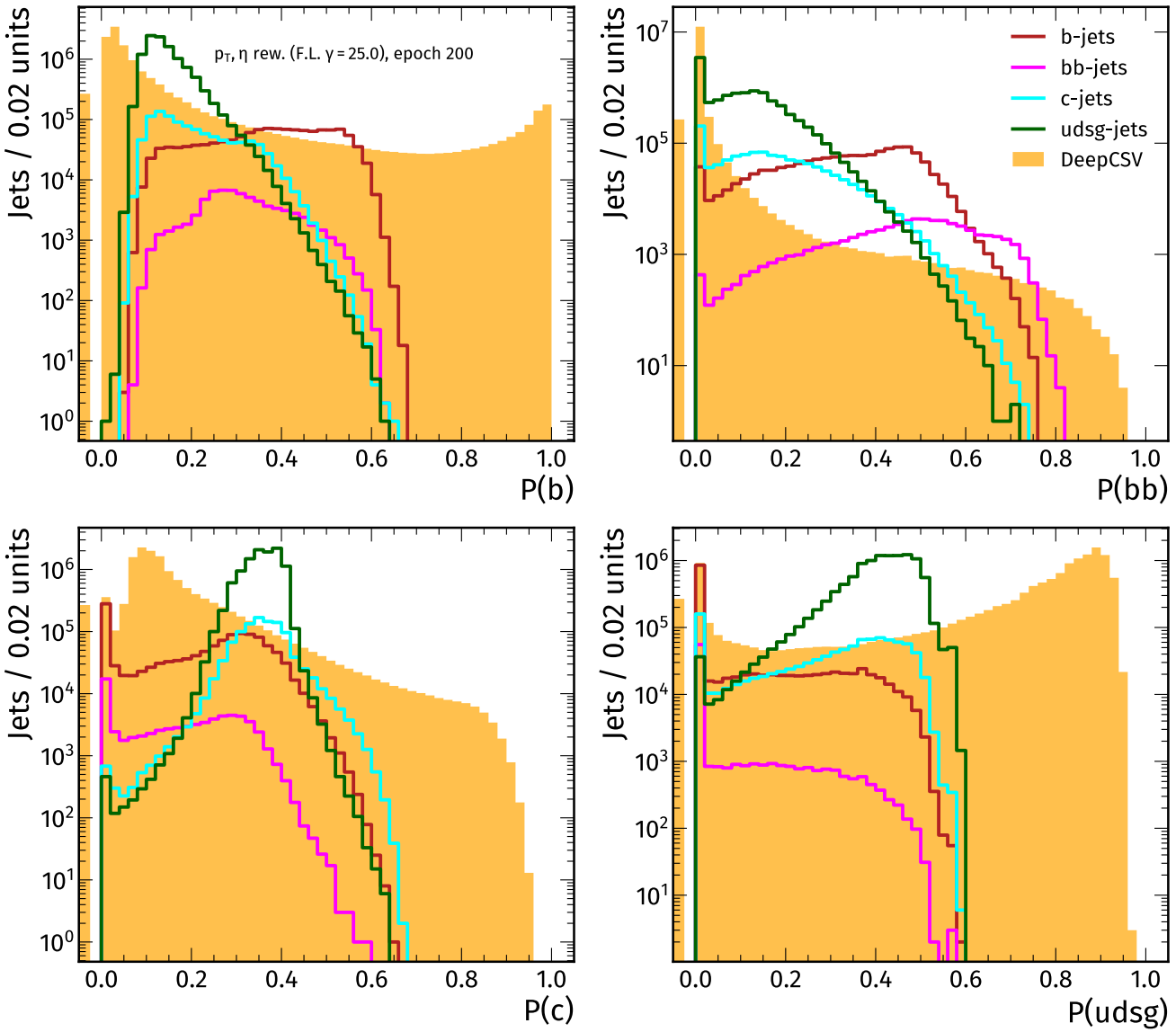
# Convergence of loss over epoch



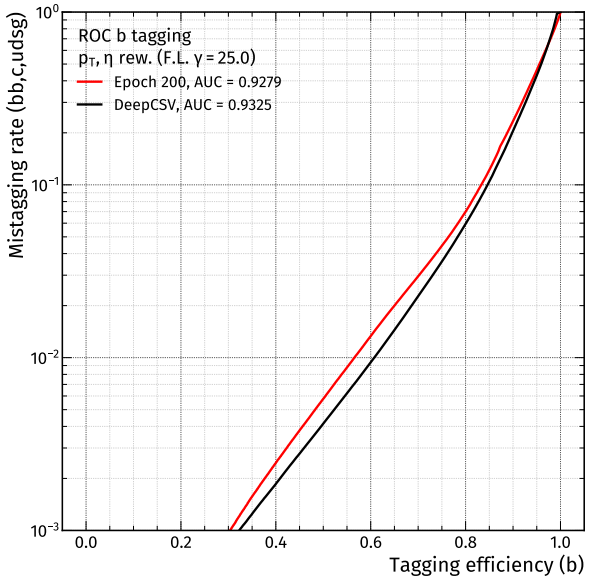

Just monitoring the training

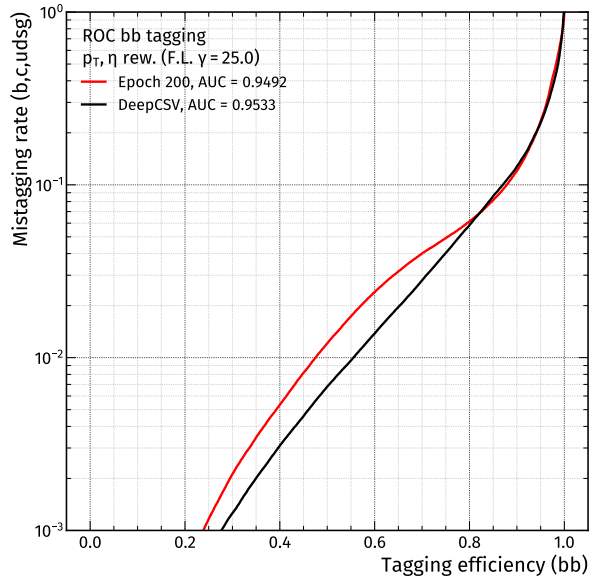
- Training and validation loss
  - to check that model converges
  - no overfitting

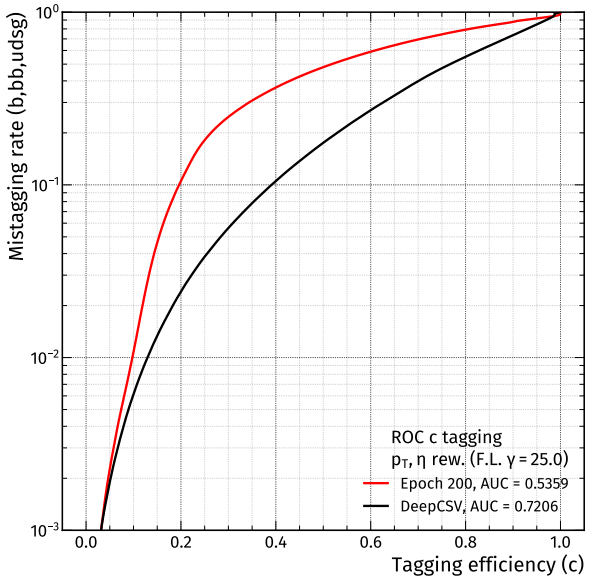
# Nominal performance: tagger outputs

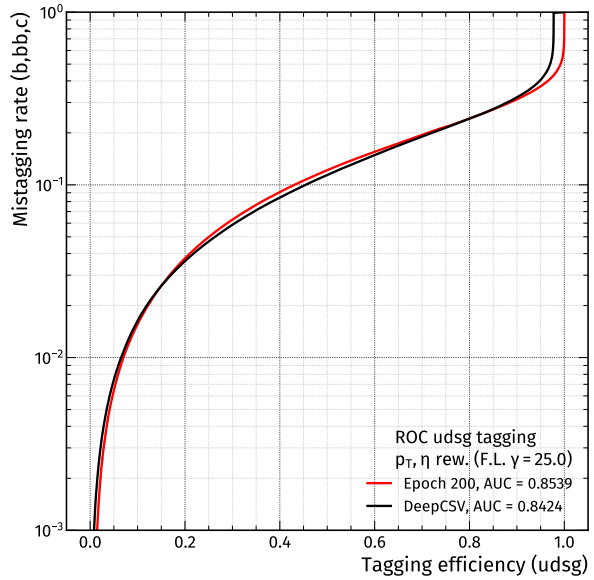


# Nominal performance: ROC XvsAll

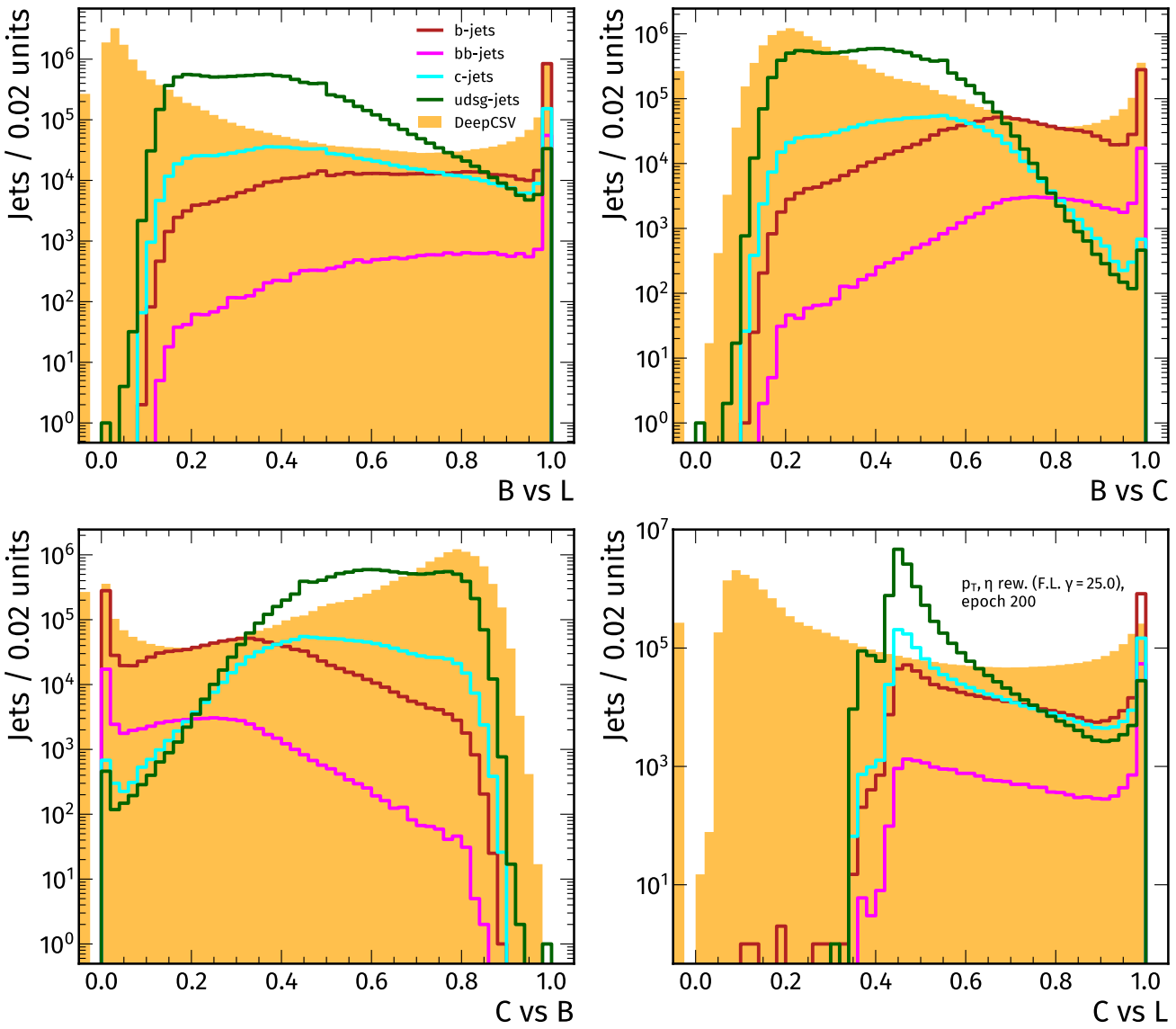


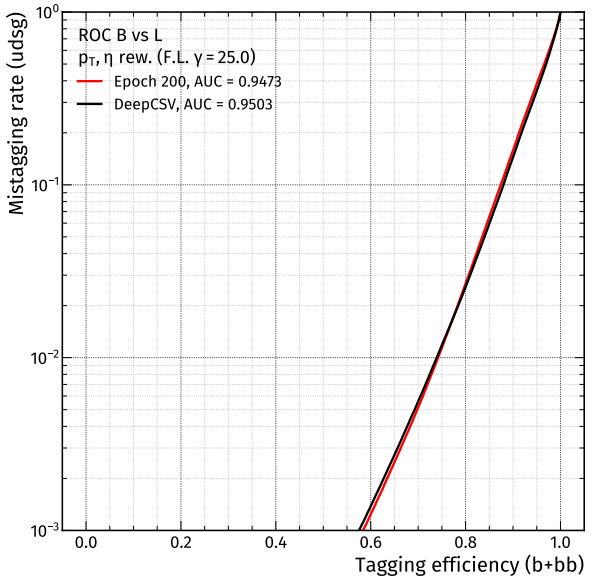



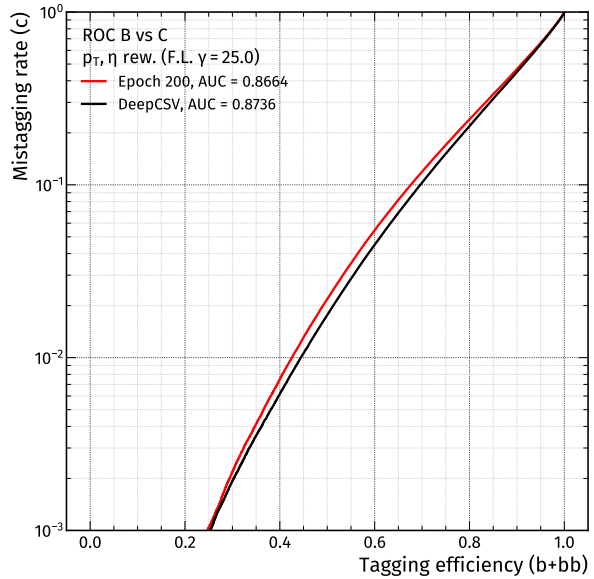



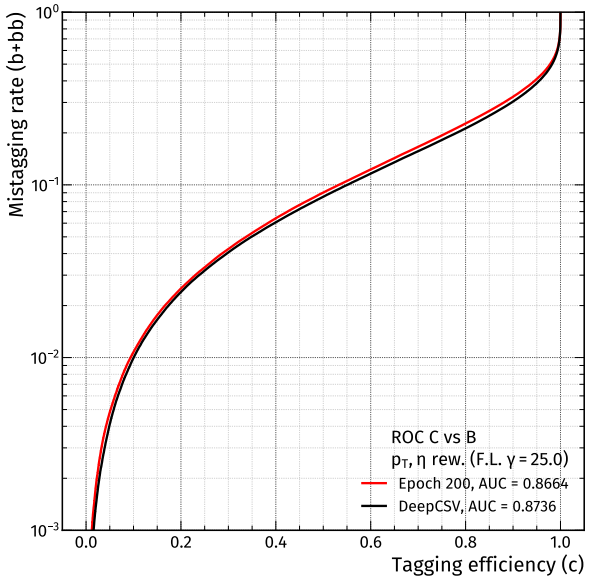
# Nominal performance: discriminator shapes

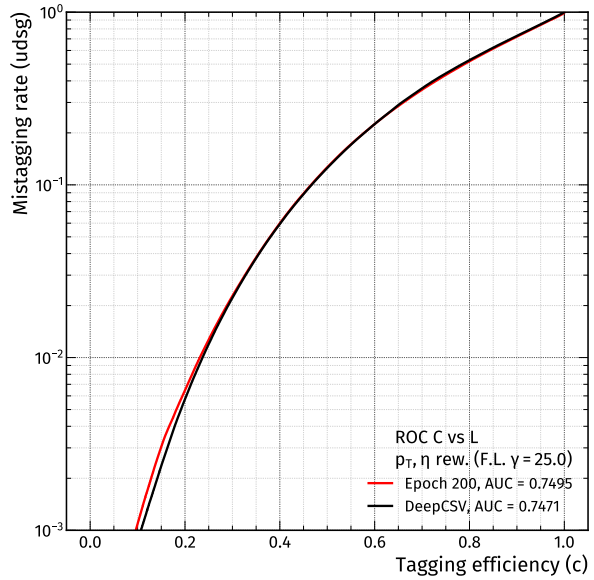


# Nominal performance: ROC for all discriminators



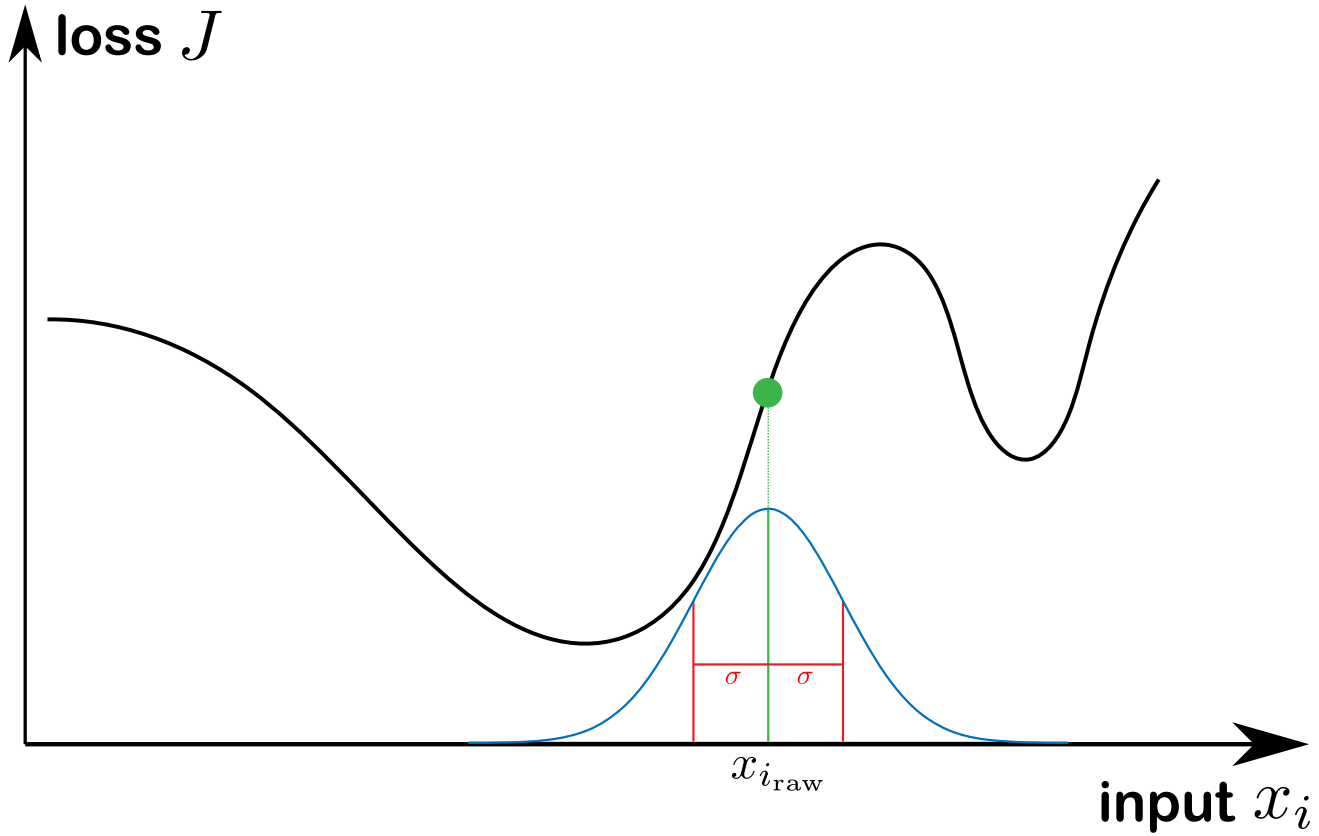



# Gaussian Noise

- Add a noise term $\xi$ to generate distorted inputs $x_{noise}$:

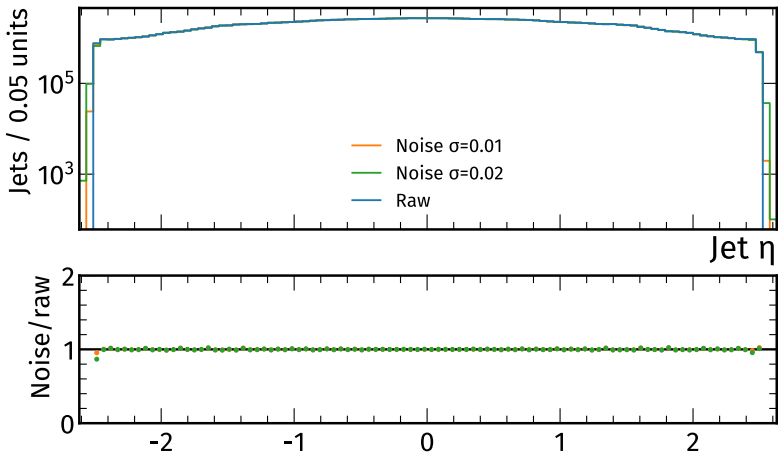
$$x_{noise} = x_{raw} + \xi$$

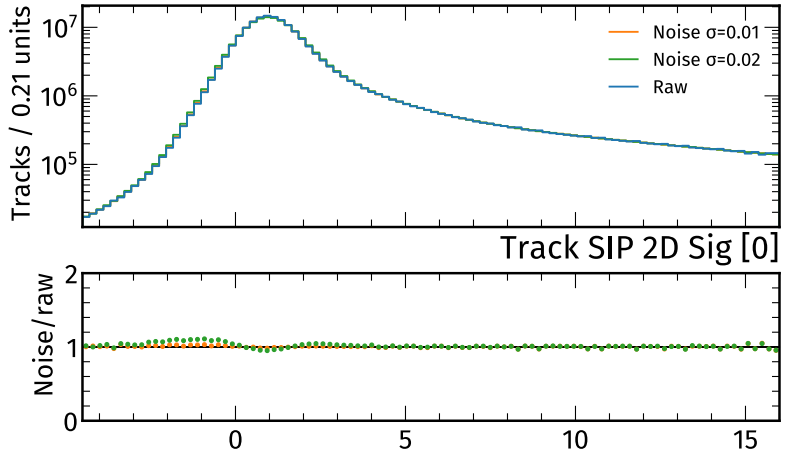
- $\xi$ follows a Gaussian distribution centred at $\mu = 0$, the standard deviation $\sigma$ can be varied

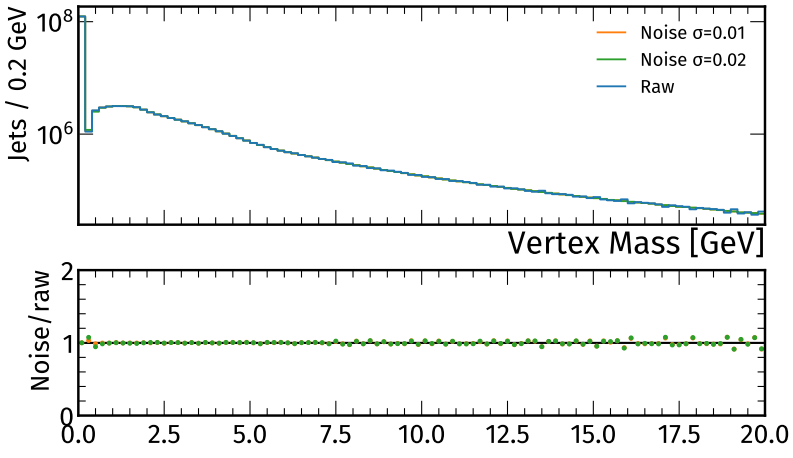


# Distorted Inputs (Noise)

- Add Gaussian noise term to test sample



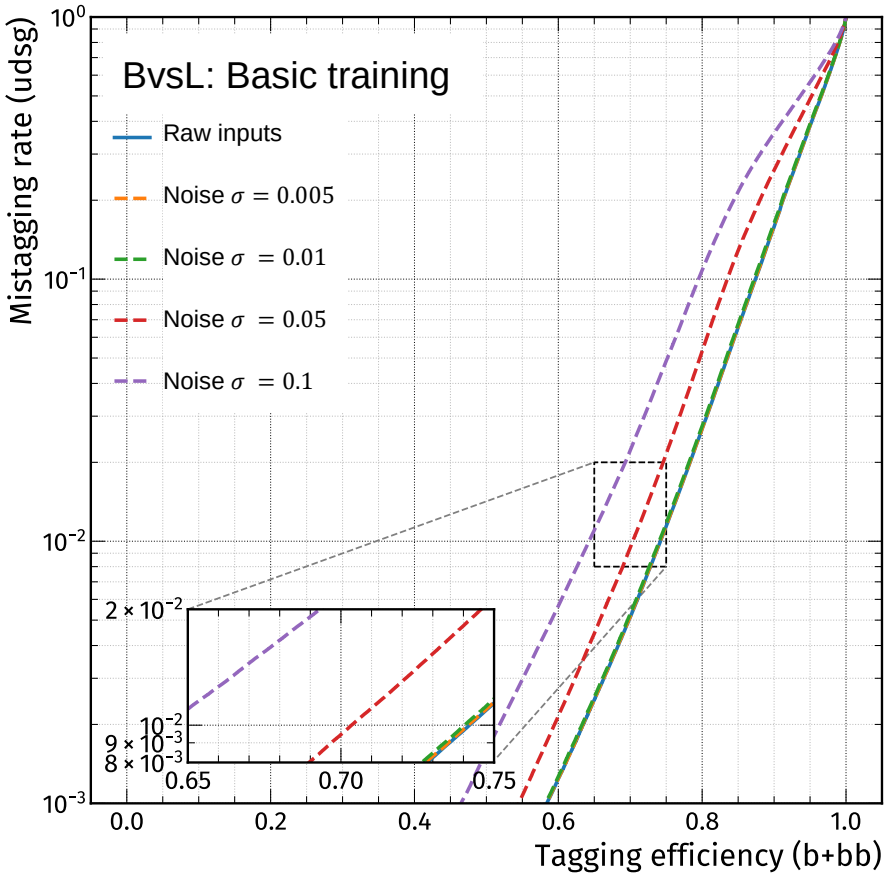




- Use small $\sigma$ for the disturbance and check jet / track / SV properties

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

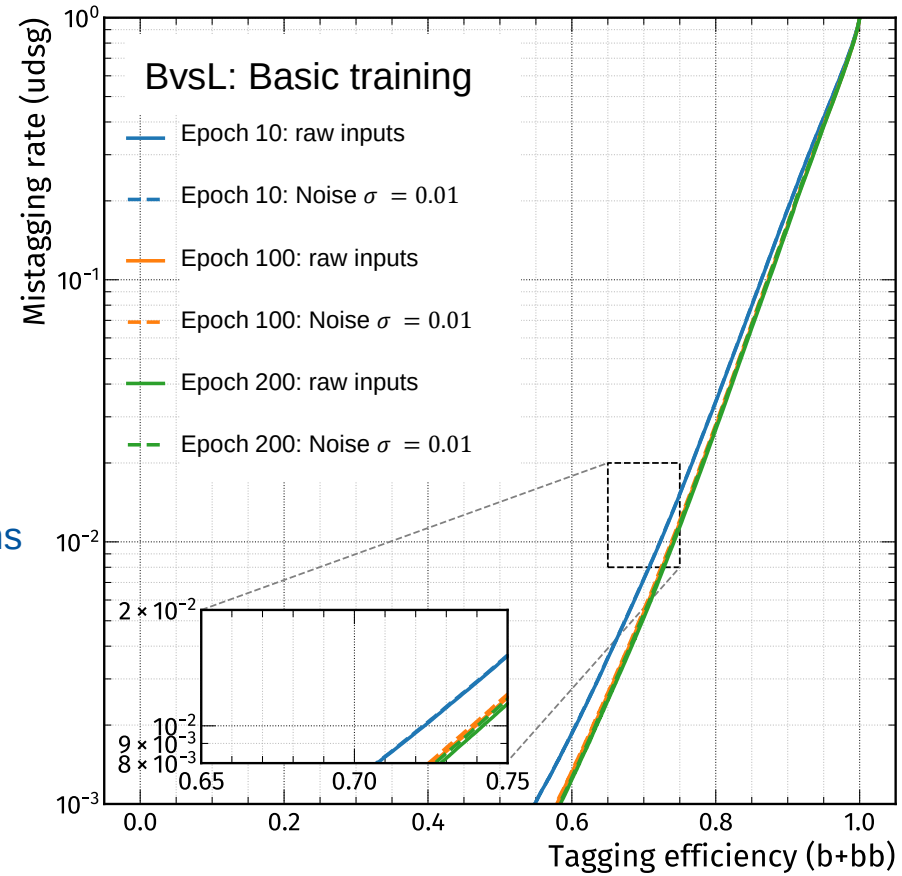
# ROC curves (B versus L, Basic Training + Noise)

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




Same epoch, different $\sigma$

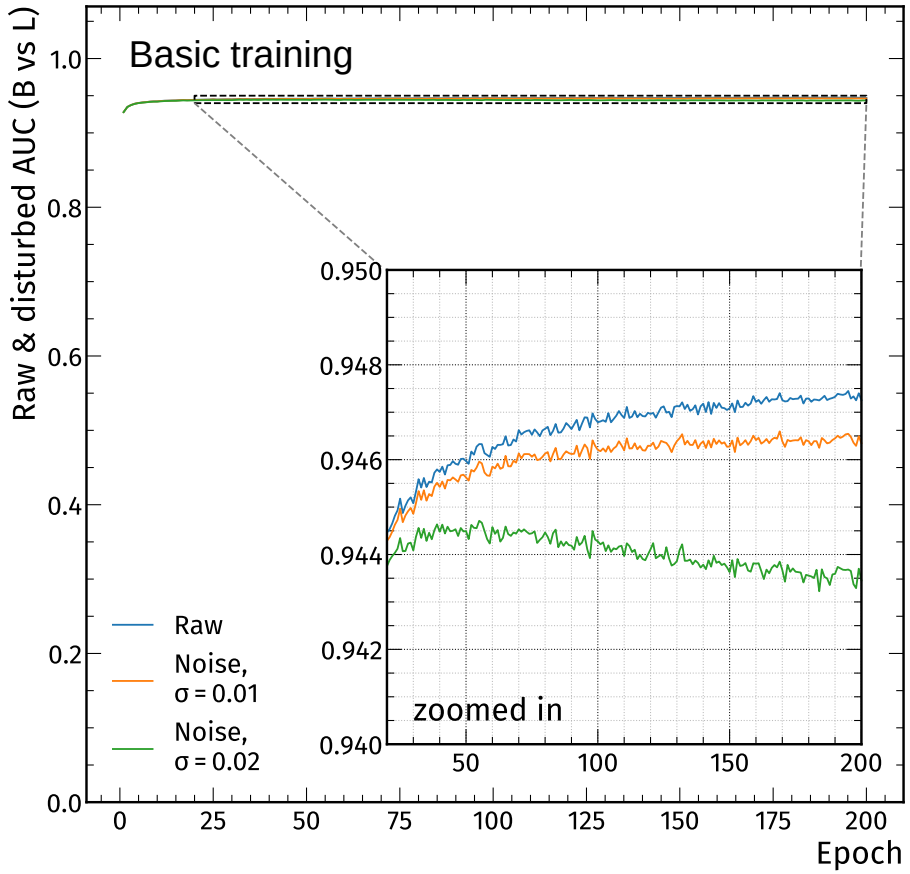
Same $\sigma$, different epochs




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

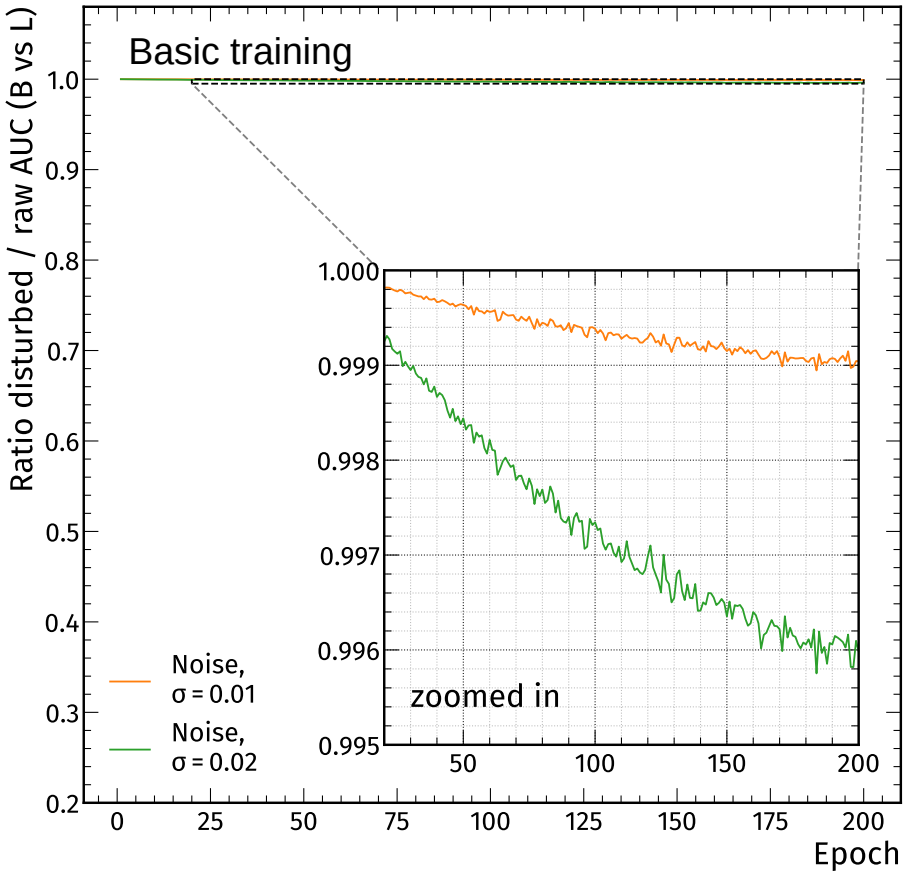
# Evolution of AUC with number of epochs (Basic Training + Noise)

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




Absolute values

Ratios Noise / raw

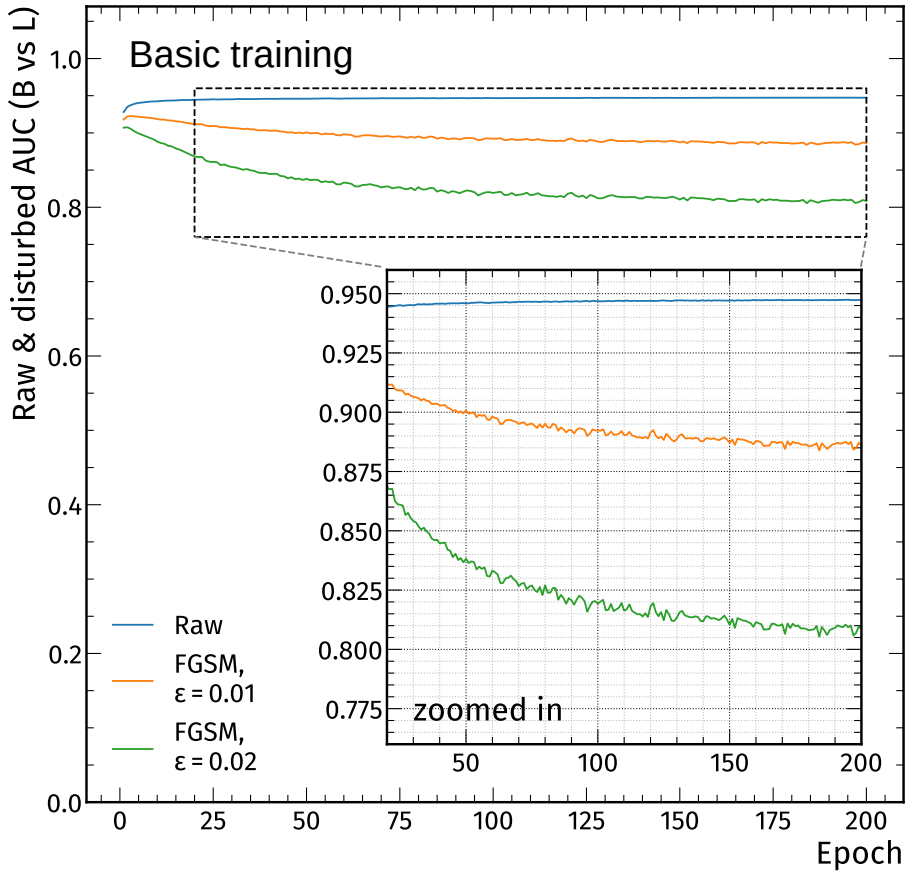



- Performance on raw inputs increases slowly
- Impact of noise term becomes more severe

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

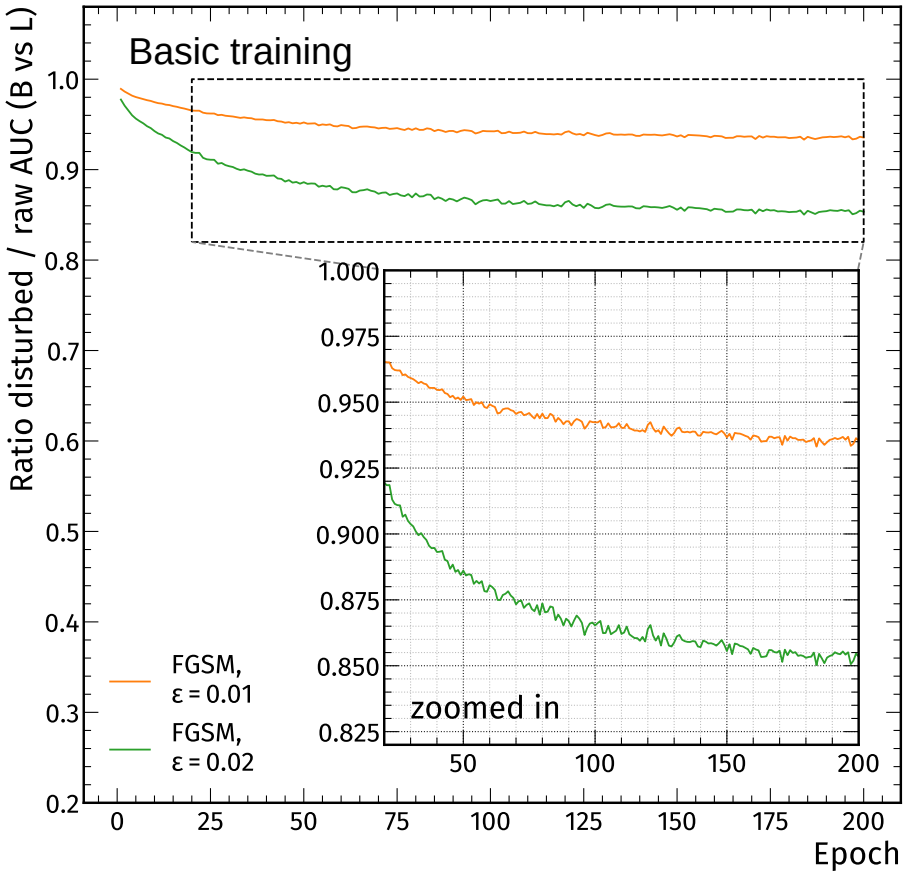
# Evolution of AUC with number of epochs (Basic Training + FGSM)

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




Absolute values

Ratios FGSM / raw

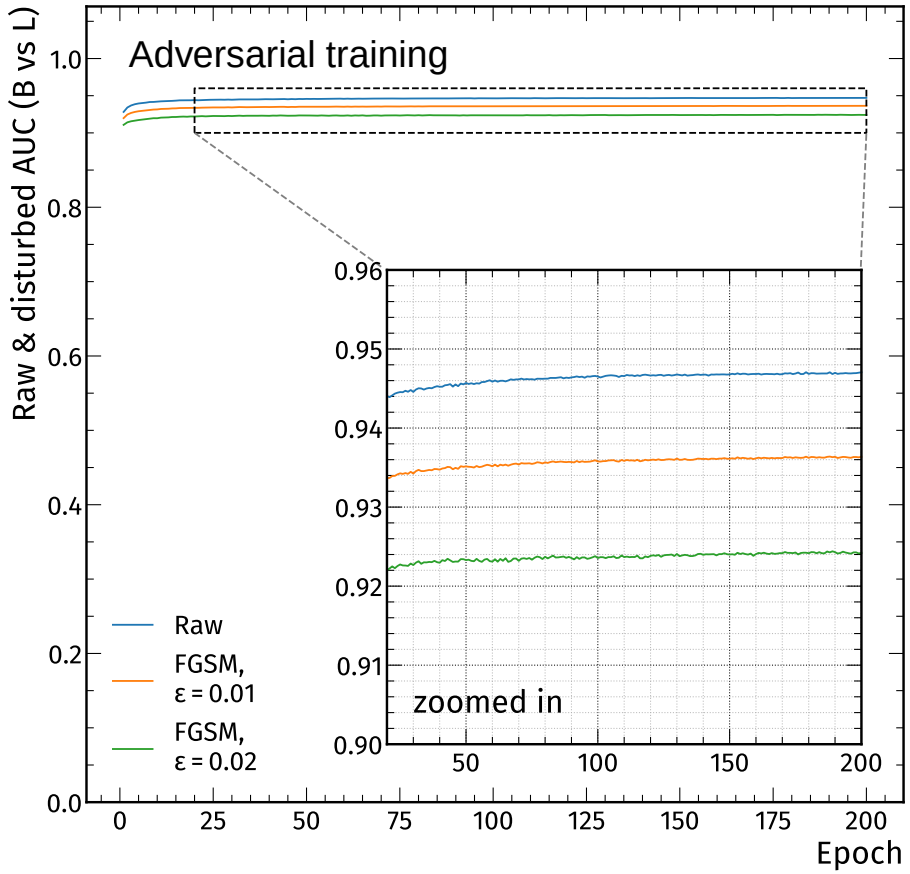



- Performance on raw inputs increases slowly
- Impact of FGSM attack becomes more severe

→ Tradeoff between performance and robustness!

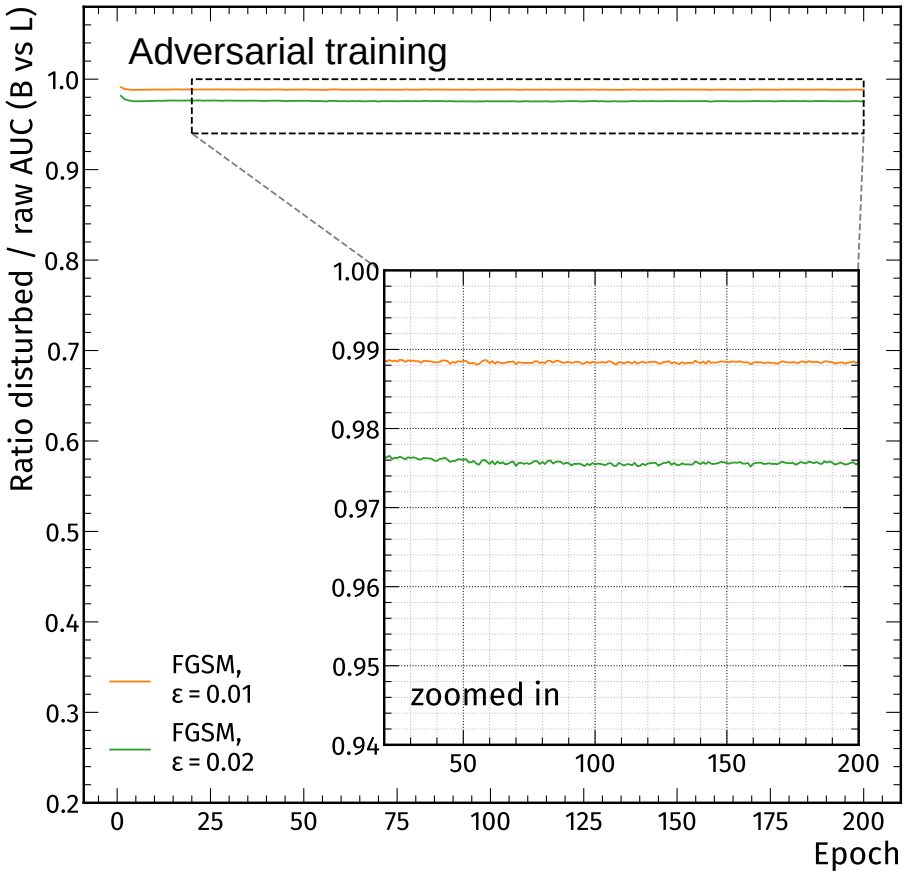
# Evolution of AUC with number of epochs (Adversarial Training + FGSM)

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




Absolute values

Ratios FGSM / raw

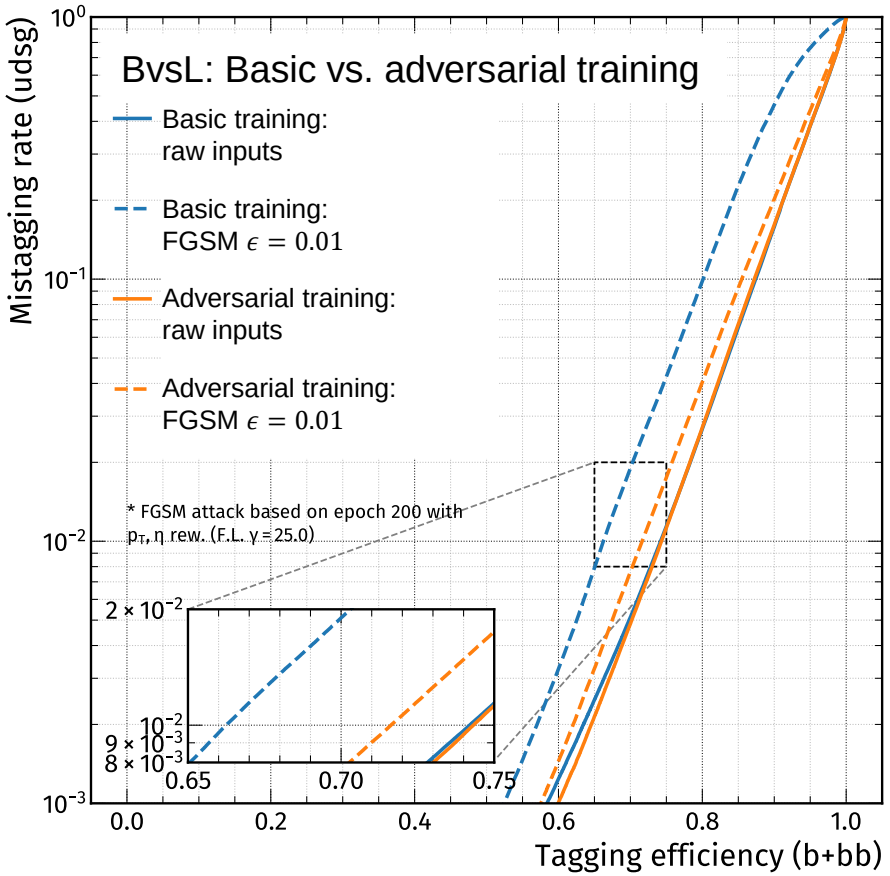



- Performance on raw inputs increases slowly
- Impact of FGSM attack stays the same

→ Only a constant offset between raw and FGSM!

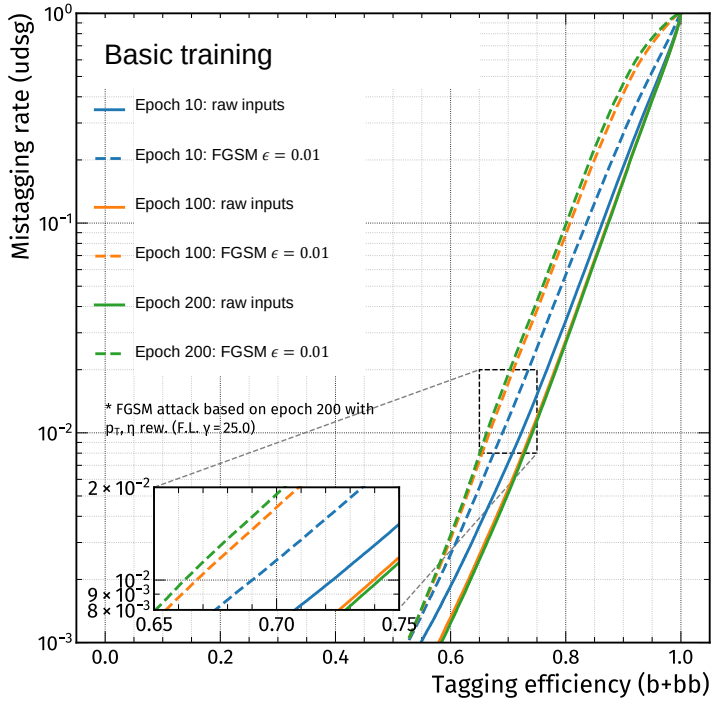
# Cross-check: transferability of adversarial examples?

- 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

**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'
```

# References


1) I. J. Goodfellow, J. Shlens and C. Szegedy, *Explaining and Harnessing Adversarial Examples*, ICLR, (2015), arXiv:1412.6572.

2) The CMS Collaboration, *Identification of heavy-flavour jets with the CMS detector in pp collisions at 13 TeV*, JINST **13** P05011, (2018), arXiv:1712.07158.

3) J. Su, D. V. Vargas and S. Kouichi, *One pixel attack for fooling deep neural networks*, (2017), arXiv:1710.08864

4) B. Nachman and C. Shimmin, *AI Safety for High Energy Physics*, (2019), arXiv:1910.08606.

5) C. Shimmin, *ipython notebooks for my MLHEP 2020 tutorial on adversarial attacks on jets*, MLHEP, (2020), https://github.com/cshimmin/advjets-mlhep2020 (last accessed: 13.09.2021)

6) N. Frediani, *First studies in AI-safety for jet flavour tagging at the CMS experiment,* Bachelor thesis, (2020).

7) A. Chakraborty et. al., *Adversarial Attacks and Defences: A Survey*, (2018), arXiv:1810.00069.


**Figures:**



*Other figures*: own work