# Adversarial Attacks and Defenses in High Energy Physics

TIMO SAALA ON BEHALF OF THE AI SAFETY PROJECT

## Motivation

- The laws of physics induce correlations among experimental observables
  - e.g. the rest mass of a particle and its energy in a collision are positively correlated

- Often, simulation is used for training DNNs
  - These simulated datasets are validated against real data
  - This is mostly done using the underlying 1D variable distributions

- Hypothesis: If a NN focuses more on the correlation, the classification performance will become more robust

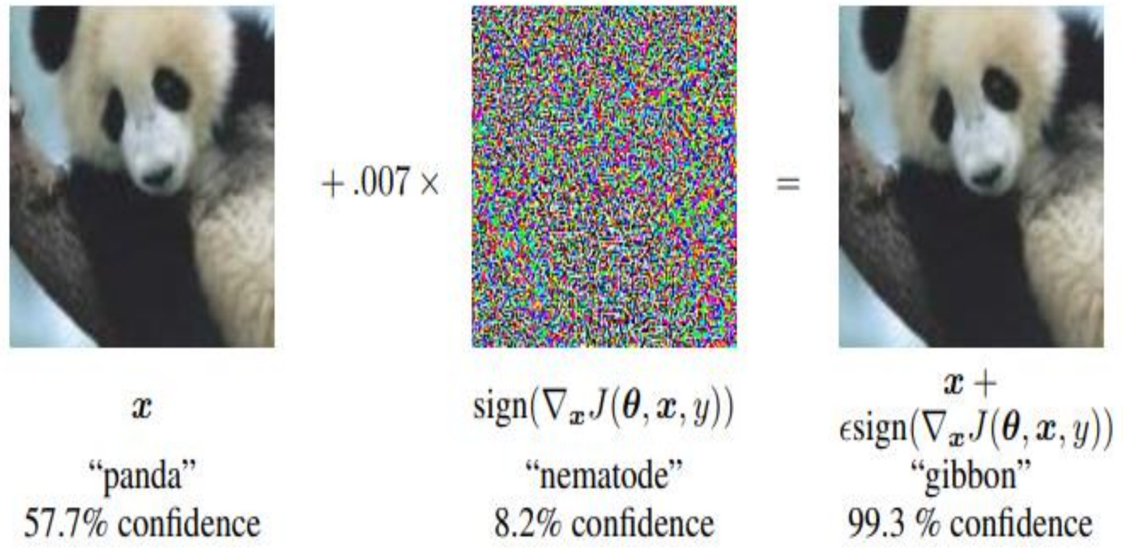
- Idea: Construct adversarial examples that force the network to concentrate more on correlations, rather than the 1D variable distributions

# I. Introduction
# - What are Adversaries?

- Input to a ML model
  - Purposefully constructed to produce maximally incorrect results
  - While keeping the (objective) perturbation minimal

- State-of-the-art algorithms include:
  - Fast Gradient Sign Method (FGSM)
  - Projected Gradient Descent (PGD)

- Both approaches:
  - Leverage the gradient of the loss w.r.t the input
  - To then alter the input in the direction of this gradient


$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Example of a misclassification caused by an adversarial attack / example. Taken from „Explaining and Harnessing Adversarial Examples“ by Ian J. Goodfellow et. al.

# I. Introduction

## - Model

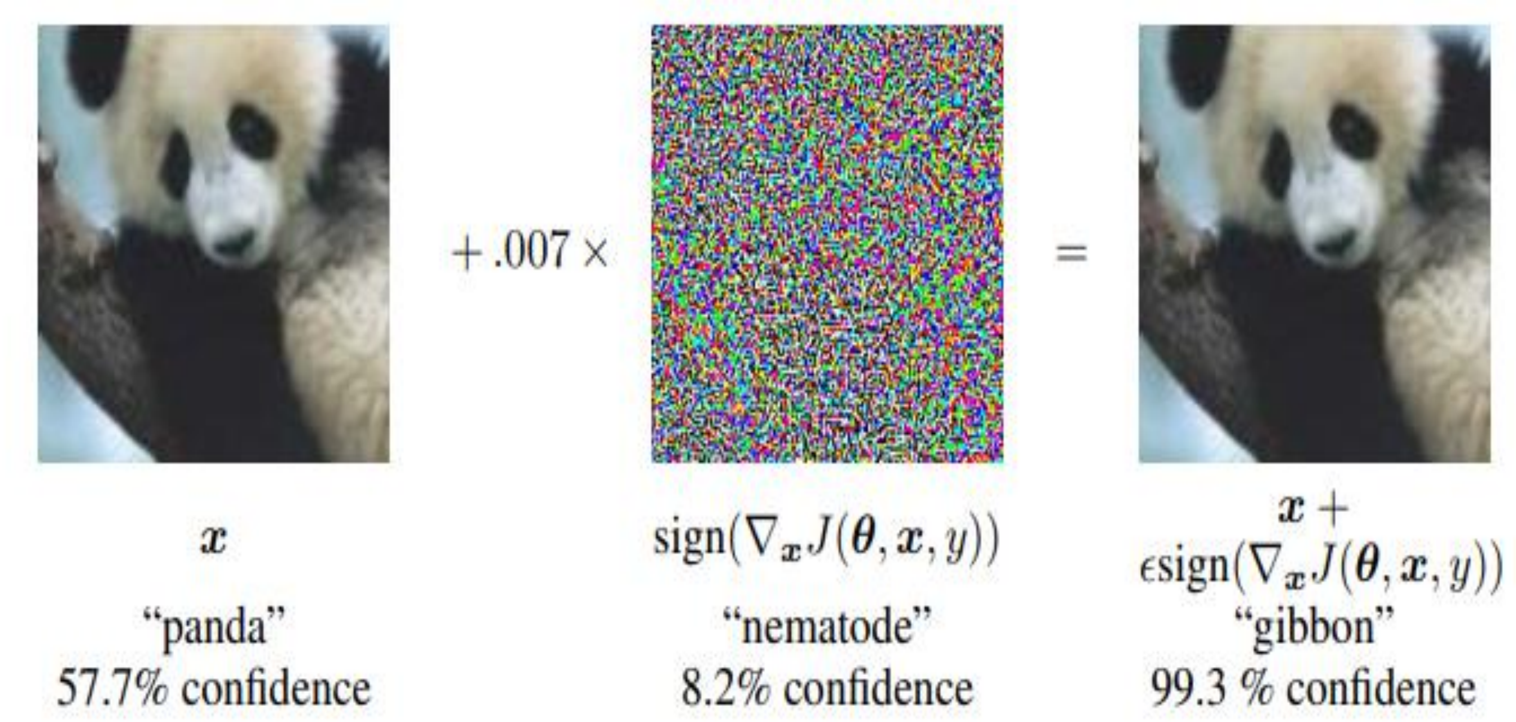
- In total, we use four models, two models on HEP data, one on medical data, and one on weather data.
  - Each model uses tabular data as inputs

- Here we focus on a single model on HEP data:
- Based on TopoDNN from G. Kasieczka et. al „The Machine Learning Landscape of Top Taggers“
  - 87 input features: $p_T, \eta, and\ \phi$ of leading 30 jet constituents (ordered by highest $p_T$)
    - - $\eta_0, \phi_0, \eta_1$ as these take always identical values due to pre-processing
  - Rebuilt using CMS Open Data from 2012 run
  - Classify between TT and WW Jets

# II. Random Distribution Shuffle Attack

## - Motivation

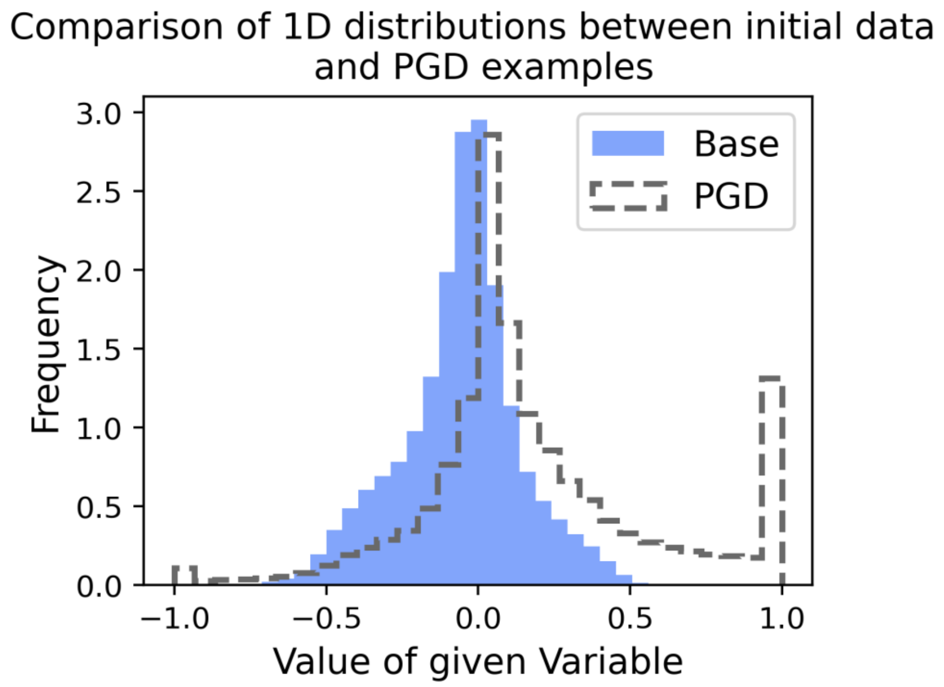
- „Normal“ adversarial generators try to minimize the perceived change of a single input


$+.007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

# II. Random Distribution Shuffle Attack

## - Motivation

- „Normal“ adversarial generators significantly change 1D variable distributions

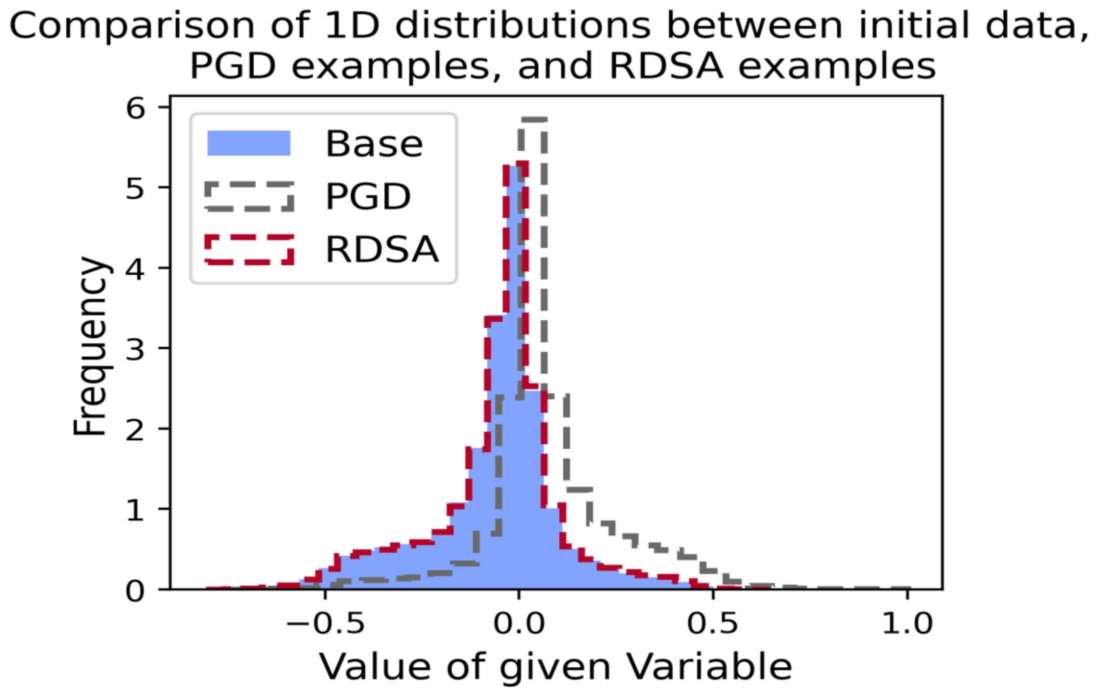


# II. Random Distribution Shuffle Attack

## - Motivation

- 1D variable distributions are an important concept in many HEP analyses

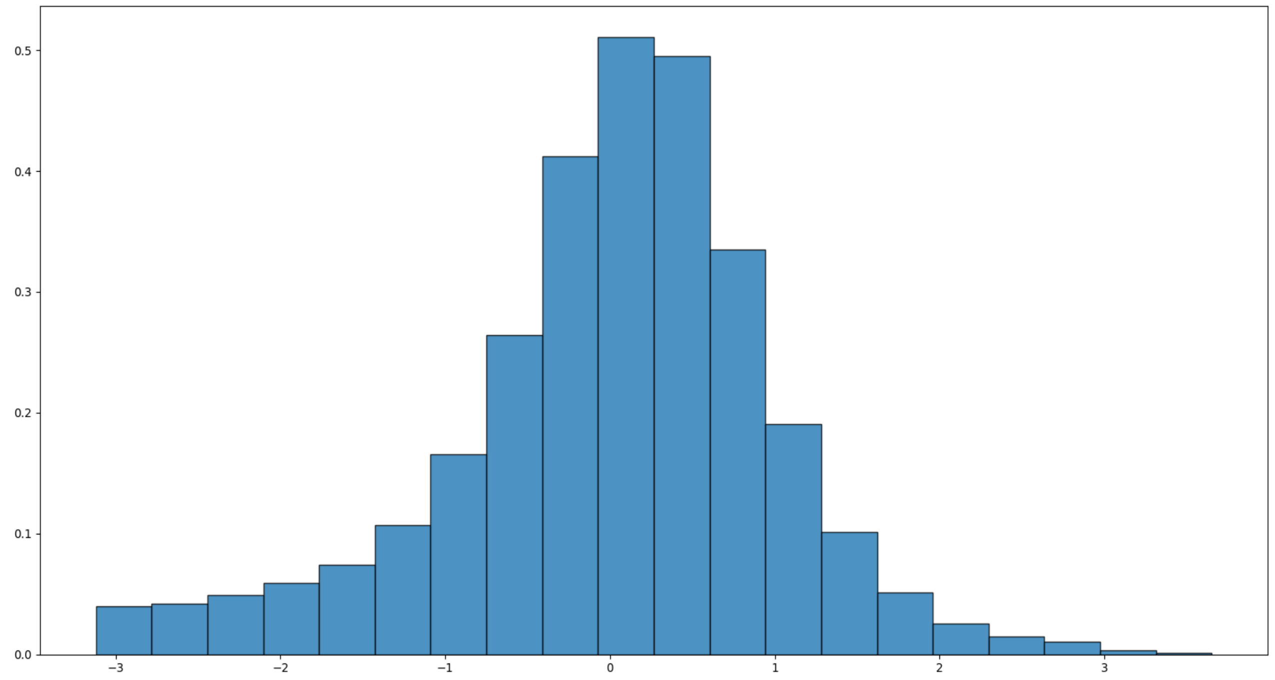
=> **Develop adversarial generator that seeks to minimize the change to 1D variable distributions**



# II. Random Distribution Shuffle Attack

## - Algorithm (Setup)

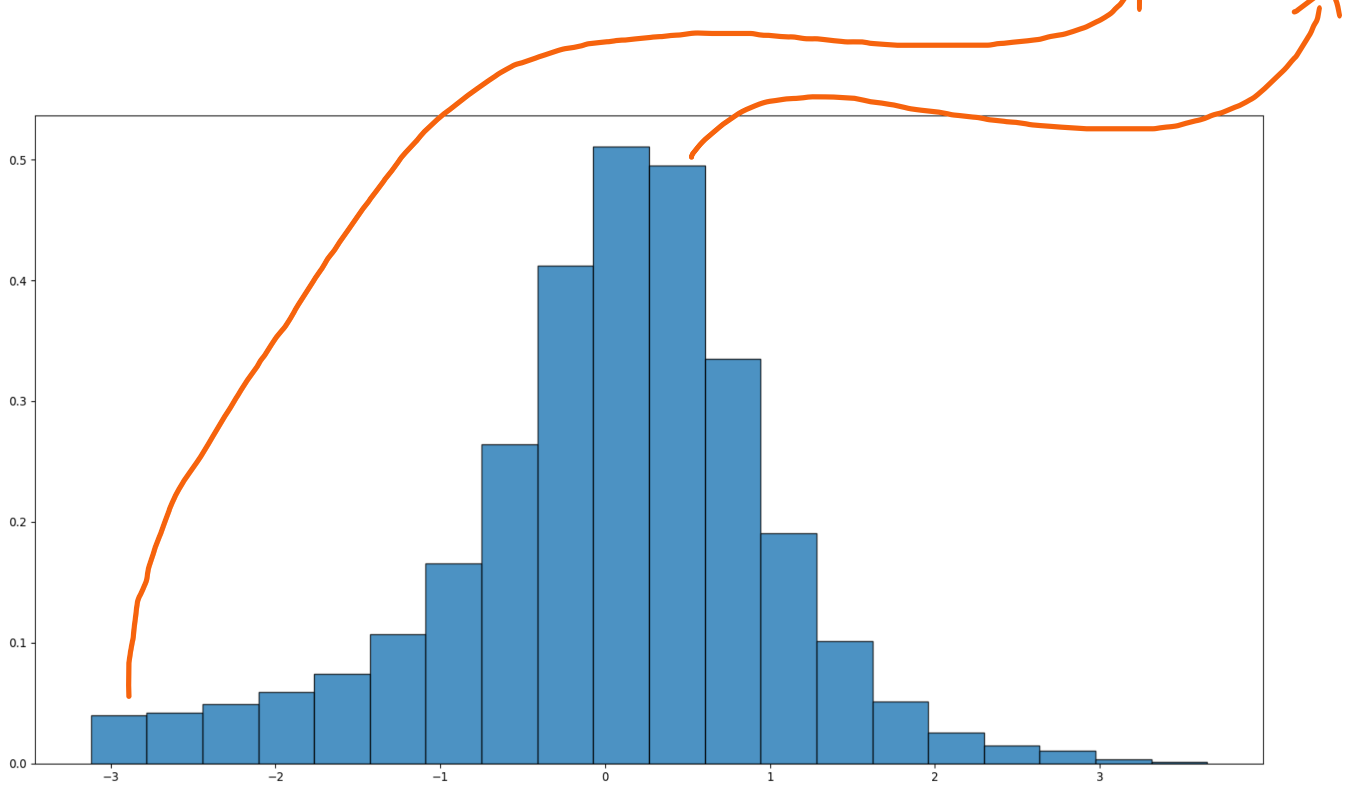
- For every variable, generate finely binned histograms over the entire dataset (e.g. test data)



# II. Random Distribution Shuffle Attack

## - Algorithm (Setup)

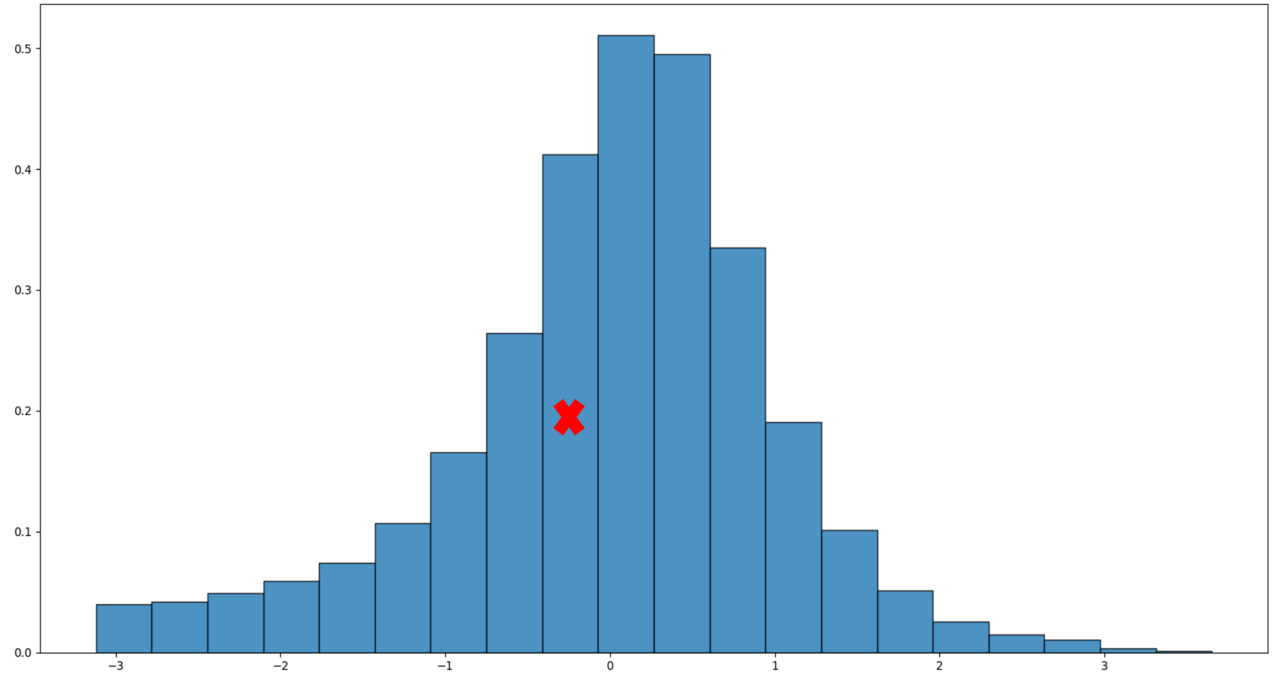
- For every variable, take y-values (frequencies) as probabilty weights (0.03, …, 0.49, …)



# II. Random Distribution Shuffle Attack

## - Algorithm (Iteration)

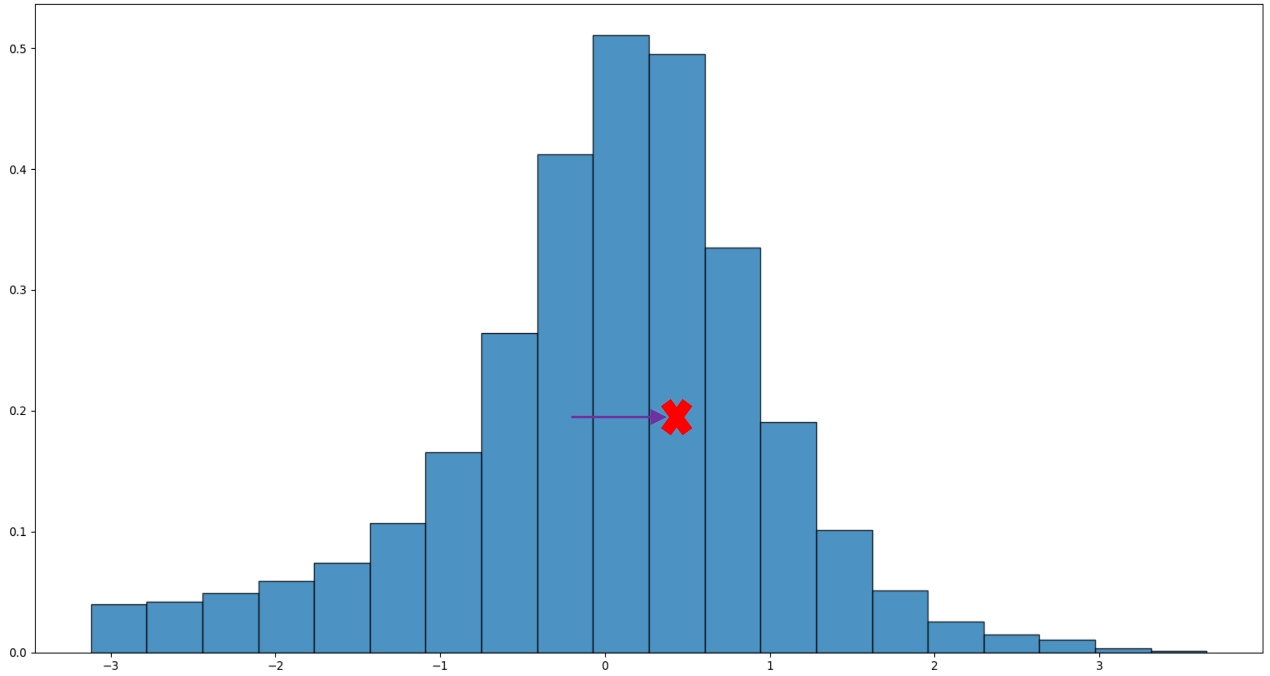
- Take a single input I, iterate over all input features
- For each input feature (X): shuffle probabilitiscally within the previously calculated frequencies



# II. Random Distribution Shuffle Attack

## - Algorithm (Iteration)

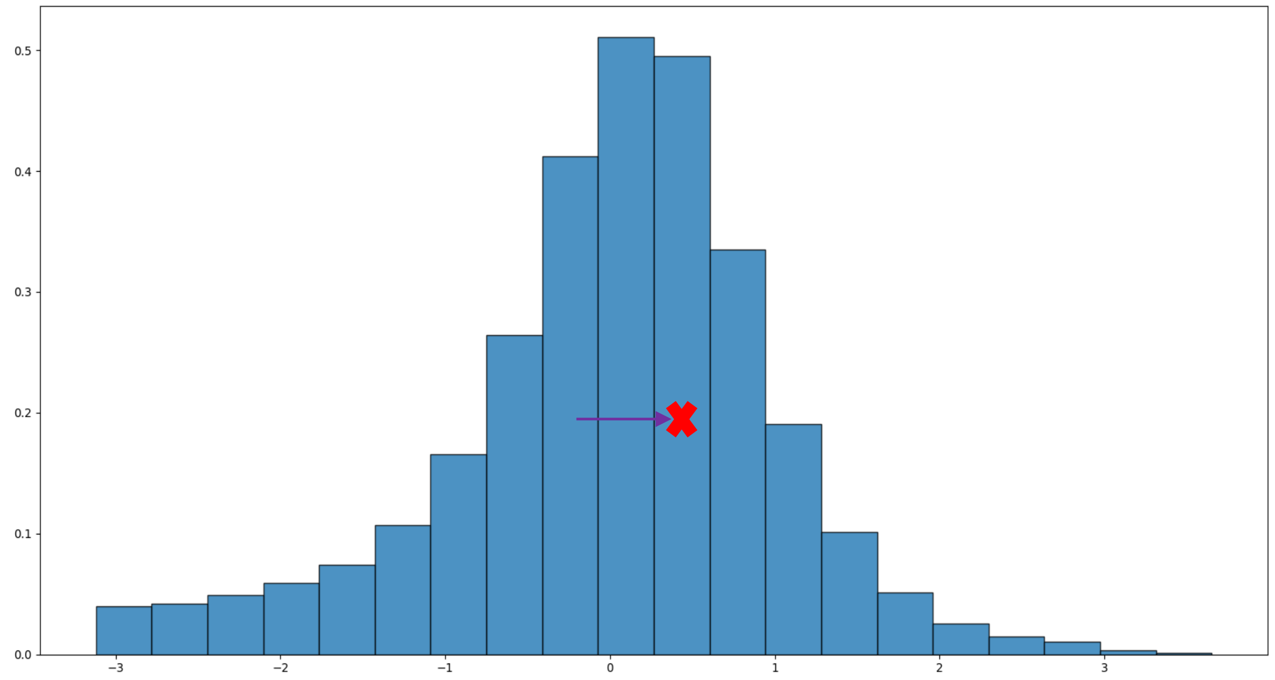
- Take a single input I, iterate over all input features
- For each input feature (X): shuffle probabilitiscally within the previously calculated frequencies



# II. Random Distribution Shuffle Attack

## - Algorithm (Iteration)

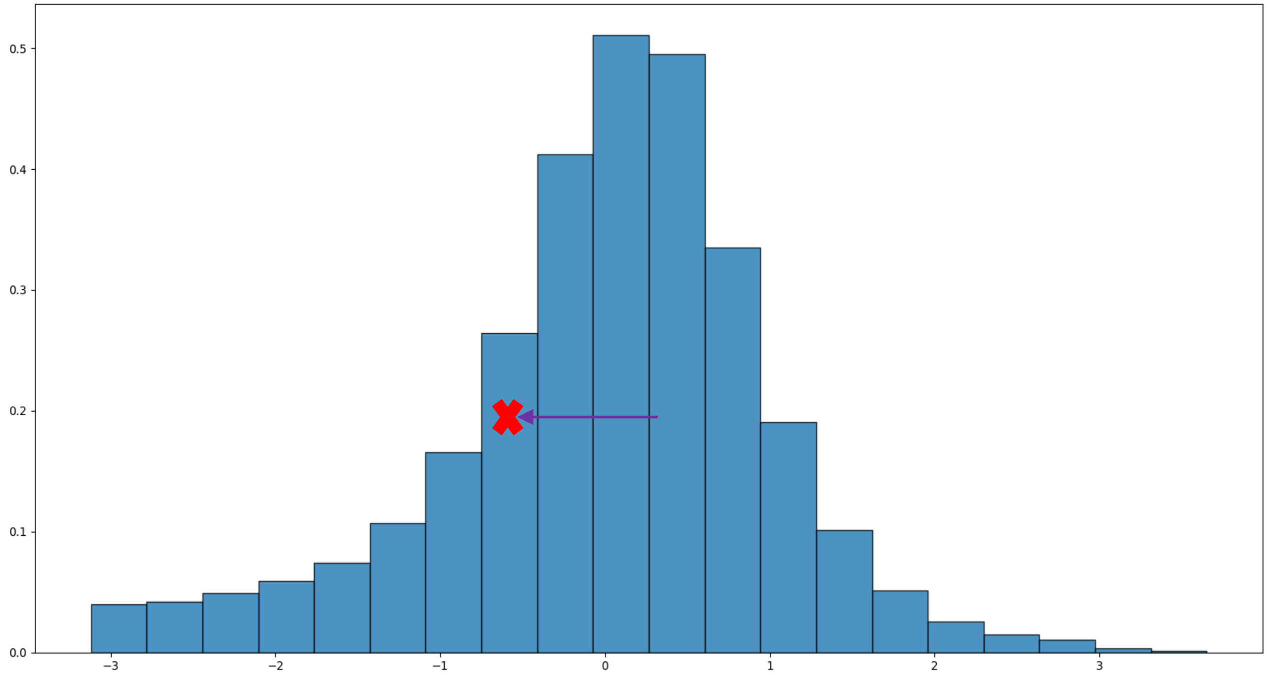
- Take a single input I, iterate over all input features
- After all input features have been shuffled, test if this combination is adversarial



# II. Random Distribution Shuffle Attack

## - Algorithm (Iteration)

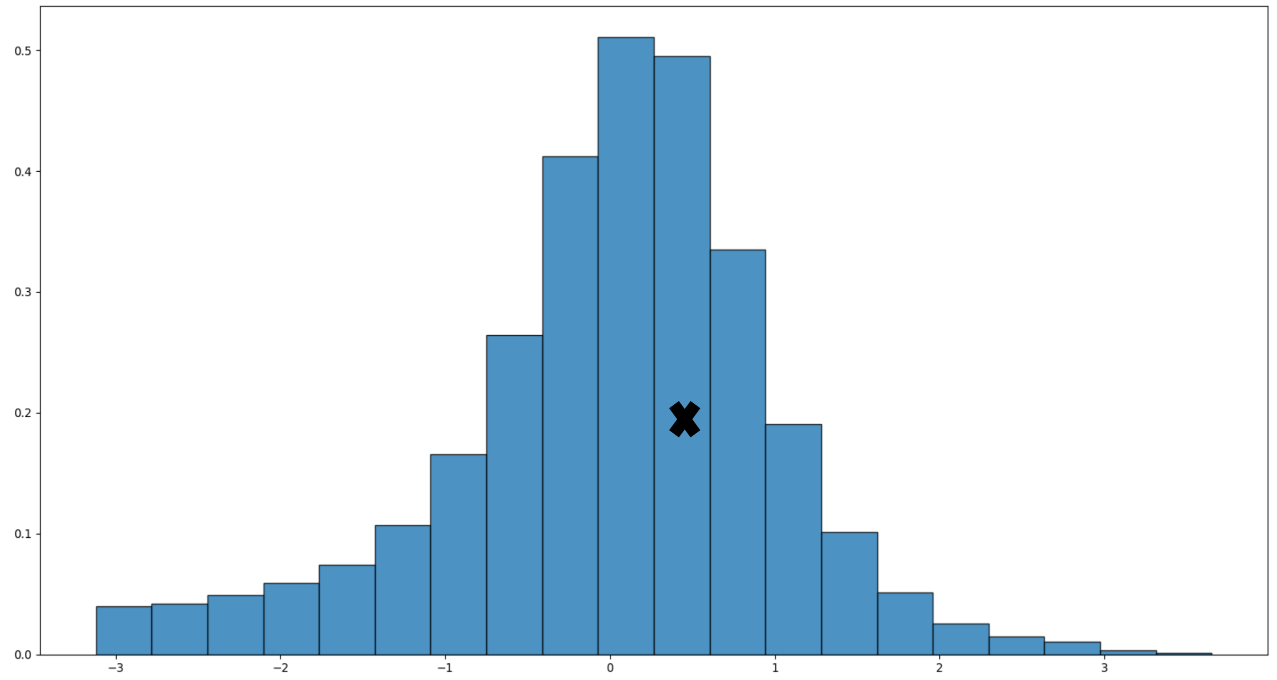
- After all input features have been shuffled, test if this combination is adversarial
- If no, continue shuffling (up to a defined amount of tries)



# II. Random Distribution Shuffle Attack

## - Algorithm (Iteration)

- After all input features have been shuffled, test if this combination is adversarial
- If yes, terminate and return adversary



## II. Random Distribution Shuffle Attack

### - Algorithm (Caveats)

- Instead of always shuffling every variable, the amount of variables to be shuffled nVars for the given data set can be set
- Then, for each input: sample nVars randomly from the set of input features to be shuffled
- Additionally, can define the maximum amount of shuffle attempts to be done

# III. Performance of RDSA

## - Attack

- Apply varying „strengths“ of RDSA and PGD (as a Reference)

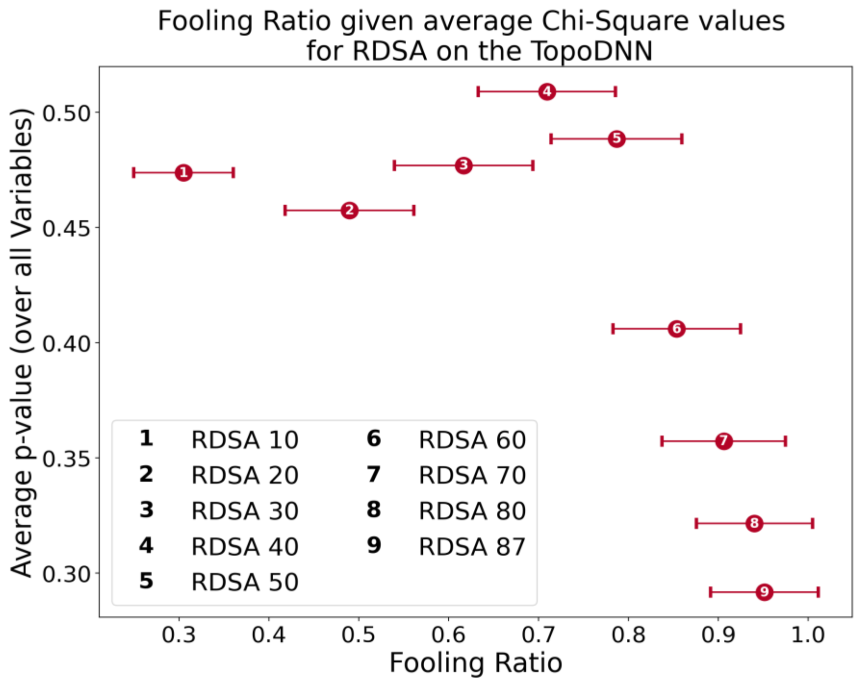
- RDSA :
  - Shuffle between 10 and 87 (all) variables each time (in increments of 10 / 7)
    - => 10, 20, …, 80, 87
  - Do the shuffling for a maximum of 100 attempts for each input

- PGD:
  - Choose maximum allowed perturbation $\epsilon$ between 0.5 and 4.0 in increments of 0.5
  - Perturbation per step = 0.0035 * epsilon, total of 40 steps

# III. Performance of RDSA

## - Attack

- As metrics here, consider the Fooling Ratio (x-Axis),
- And the p-values (y-Axis) resulting from a $\chi^2$ test



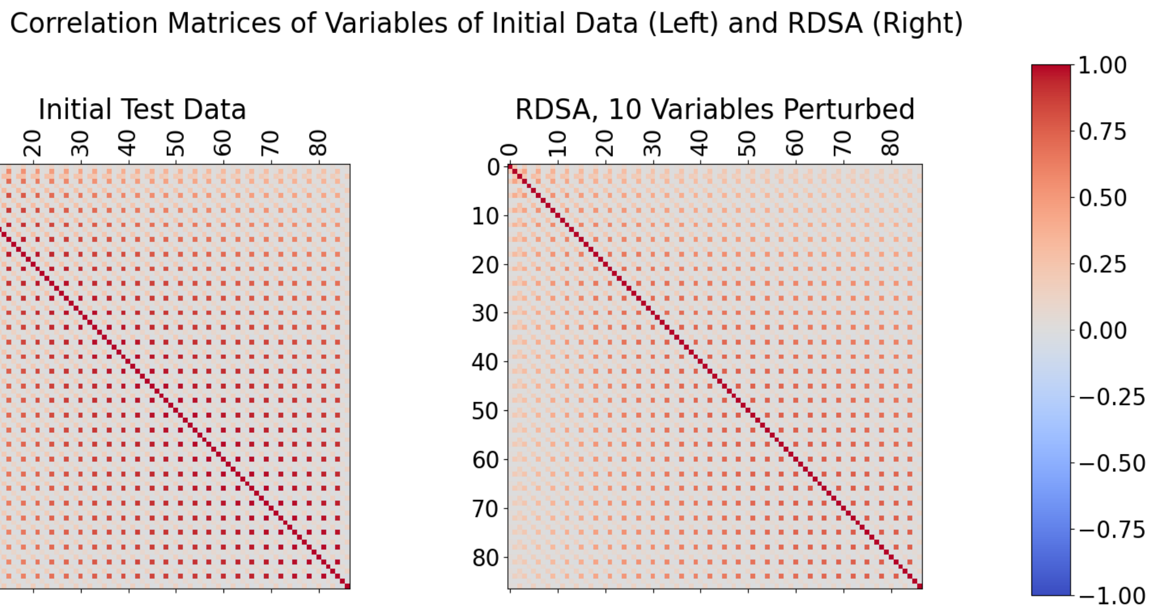
Remarks:

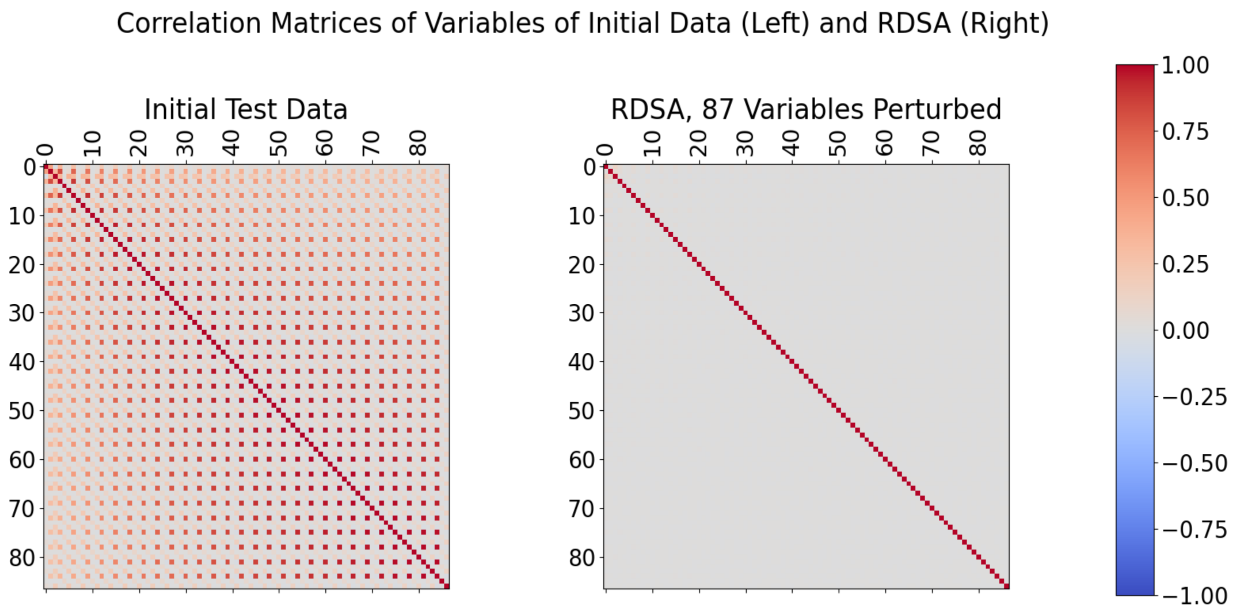
- PGD Always 0
- We split the data set for the $\chi^2$ test in half
  => p-values of 1 (or close to 1) are impossible

# III. Performance of RDSA

## - Attack

- An interesting and perhaps useful property of the attack:
- Correlations of input features „vanish“ when shuffling all of them





## III. Improving the Network Robustness

### - Retraining

- Use novel RDSA for data augmentation
  - Compare also with CTGAN (from Lei Xu et. al. „Modeling Tabular data using Conditional GAN“ and PGD as augmentation

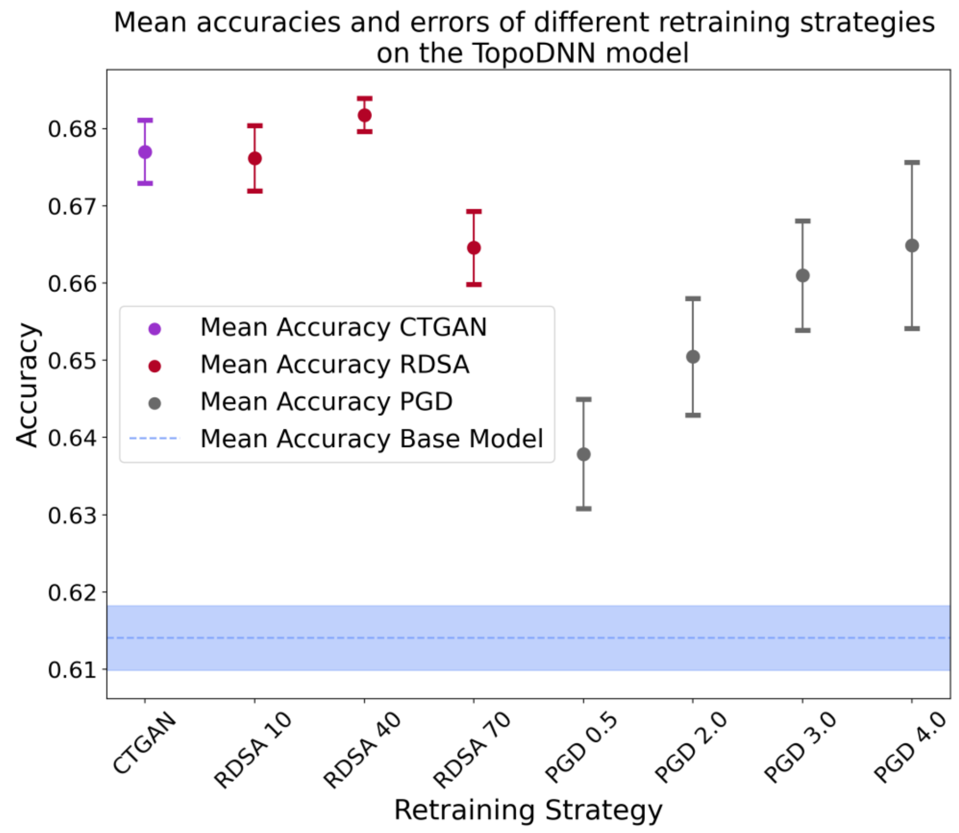
- Assumption: By „fixing“ the 1D distributions of the training set, the model focuses more on learning higher dimensional statistical moments
  - e.g. Correlation

- To this end, purposefully reduce training data size to bring model to data-starved regions
  - From ~280k inputs for training to ~70k

# III. Improving the Network Robustness

## - Retraining

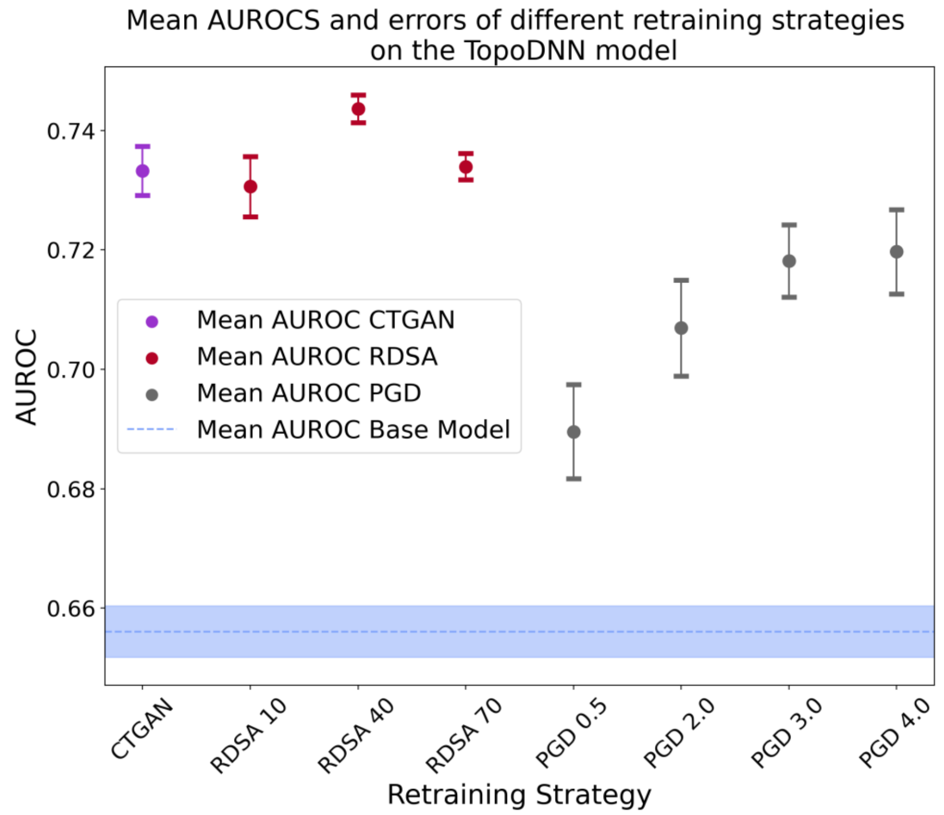
- As this is a binary classification task, consider mainly two metrics
- The accuracy of the augmented models,
- Errors here are given by the standard deviation



# III. Improving the Network Robustness

## - Retraining

- As this is a binary classification task, consider mainly two metrics
- The accuracy of the augmented models, and the AUROC
- Errors here are given by the standard deviation



## IV. Conclusion

- Introduced novel attack, achieving high fooling ratios while keeping the changes to 1D distributions minimal
  - Additionally tested on three further models, one more HEP, one medicine, and one weather
  - This attack also vanishes the correlations between the input features
  - This attack would go „unnoticed“ when comparing 1D variable distributions

- Can be used as an (at least) competitive data augmentation method
  - According to our tests conducted
  - Keeping the 1D distributions of the training set very similar
  - => possibly forcing the network to focus on higher dimensional statistical moments