Neural Networks for Full Phase-space Reweighting and Parameter Tuning

Anders Andreassen

in collaboration with Ben Nachman
arXiv:1907.08209

Introduction

- Parton shower Monte Carlo programs are great!

Introduction

- Parton shower Monte Carlo programs are great!
- But: Three key simulation challenges:

1. Insufficient number of events
2. Unaccounted for biases from simulation parameters
3. Not utilizing the full phase space for tuning

Introduction

- Parton shower Monte Carlo programs are great!
- But: Three key simulation challenges:

1. Insufficient number of events
   - More and faster simulations
   - ML for for augmenting and replacing simulations

2. Unaccounted for biases from simulation parameters
   - Automated event weights for selected parameters

3. Not utilizing the full phase space for tuning
   - Limited to 1D inputs assumed to be independent

See paper for references

Deep neural networks using Classification for Tuning and Reweighting (DCTR)

Introduction to Reweighting

- Consider two MC simulations, on a phase space Ω,
  described by probability densities $p_0(x), p_1(x)$ for $x \in \Omega$

Introduction to Reweighting

- Consider two MC simulations, on a phase space Ω,
  described by probability densities $p_0(x), p_1(x)$ for $x \in \Omega$

- The ideal event-level weight, $w(x) = p_0(x)/p_1(x)$,
  would morph one simulation into the other

Introduction to Reweighting

- Consider two MC simulations, on a phase space Ω,
  described by probability densities $p_0(x), p_1(x)$ for $x \in \Omega$

- The ideal event-level weight, $w(x) = p_0(x)/p_1(x)$,
  would morph one simulation into the other

- Lots of proposals for how to estimate $w(x)$ using machine learning
  - E.g. from a classifier

Introduction to Reweighting

- Consider two MC simulations, on a phase space Ω,
  described by probability densities $p_0(x), p_1(x)$ for $x \in \Omega$

- The ideal event-level weight, $w(x) = p_0(x)/p_1(x)$,
  would morph one simulation into the other

- Lots of proposals for how to estimate $w(x)$ using machine learning
  - E.g. from a classifier

- Let $f(x)$ be a neural network optimized with binary crossentropy

$$\text{loss}(f(x)) = -\sum_{i \in \mathbf{0}} \log f(x_i) - \sum_{i \in \mathbf{1}} \log(1 - f(x_i))$$

then

$$f(x)/(1 - f(x)) \approx p_0(x)/p_1(x)$$

Introduction to Reweighting

- Consider two MC simulations, on a phase space Ω,
  described by probability densities $p_0(x), p_1(x)$ for $x \in \Omega$

- The ideal event-level weight, $w(x) = p_0(x)/p_1(x)$,
  would morph one simulation into the other

- Lots of proposals for how to estimate $w(x)$ using machine learning
  - E.g. from a classifier

- Let $f(x)$ be a neural network optimized with binary crossentropy

$$\text{loss}(f(x)) = -\sum_{i \in \mathbf{0}} \log f(x_i) - \sum_{i \in \mathbf{1}} \log(1 - f(x_i))$$

then

$$f(x)/(1 - f(x)) \approx p_0(x)/p_1(x)$$

DCTR uses the full phase space $x \in \Omega$

Continuous Reweighting

- Consider two simulations from the same MC program with different parameters θ
  - E.g. evaluation of uncertainties, $p_{\theta_0}(x)$ and $p_{\theta_0+\delta_\theta}(x)$

Continuous Reweighting

- Consider two simulations from the same MC program with different parameters θ
  - E.g. evaluation of uncertainties, $p_{\theta_0}(x)$ and $p_{\theta_0+\delta_\theta}(x)$

- Extend reweighting procedure: $f(x) \rightarrow f(x, \theta)$

Continuous Reweighting

- Consider two simulations from the same MC program with different parameters θ
  - E.g. evaluation of uncertainties, $p_{\theta_0}(x)$ and $p_{\theta_0+\delta_\theta}(x)$

- Extend reweighting procedure: $f(x) \rightarrow f(x, \theta)$

$$f(x,\theta) = \underset{f'}{\operatorname{argmax}} \sum_{i \in \boldsymbol{\theta_0}} \log f'(x_i,\theta) + \sum_{i \in \boldsymbol{\theta}} \log(1 - f'(x_i,\theta))$$

Tuning with DCTR

- A well trained model $f(x, \theta)$ minimizes the loss for any θ

$$f(x, \theta) = \underset{f'}{\operatorname{argmax}} \sum_{i \in \boldsymbol{\theta_0}} \log f'(x_i, \theta) + \sum_{i \in \boldsymbol{\theta}} \log(1 - f'(x_i, \theta))$$

Tuning with DCTR

- A well trained model $f(x, \theta)$ minimizes the loss for any θ

$$f(x,\theta) = \underset{f'}{\operatorname{argmax}} \sum_{i \in \boldsymbol{\theta_0}} \log f'(x_i, \theta) + \sum_{i \in \boldsymbol{\theta}} \log(1 - f'(x_i, \theta))$$

- Given a sample with θ_1 unknown (e.g. data), then

$$\theta^* = \underset{\theta'}{\operatorname{argmax}} \sum_{i \in \boldsymbol{\theta_0}} \log f(x_i, \theta') + \sum_{i \in \boldsymbol{\theta_1}} \log(1 - f(x_i, \theta'))$$

implies $\theta^* = \theta_1$

Tuning with DCTR

- A well trained model $f(x, \theta)$ minimizes the loss for any θ

$$f(x,\theta) = \underset{f'}{\operatorname{argmax}} \sum_{i \in \boldsymbol{\theta_0}} \log f'(x_i, \theta) + \sum_{i \in \boldsymbol{\theta}} \log(1 - f'(x_i, \theta))$$

- Given a sample with θ_1 unknown (e.g. data), then

$$\theta^* = \underset{\theta'}{\operatorname{argmax}} \sum_{i \in \boldsymbol{\theta_0}} \log f(x_i, \theta') + \sum_{i \in \boldsymbol{\theta_1}} \log(1 - f(x_i, \theta'))$$

  implies $\theta^* = \theta_1$

- Can find solution by gradient descending on θ'

DCTR Summary:

A deep neural network classifier to estimate the likelihood ratio

$$w(x, \theta) = p_1(x, \theta)/p_0(x, \theta)$$

as a continuous function of any MC parameter θ
using the full phase space $x \in \Omega$

Use DCTR to :
- reweight one simulation to another
- tune MC to match an unknown sample, e.g. real data

Network Architecture

Want to use full phase space information

- JUNIPR:

  Estimates likelihood $p_0(x,\theta)$ and $p_1(x,\theta)$ separately,
  then take the ratio $p_0(x,\theta)/p_1(x,\theta)$

AA, Feige, Frye, Schwartz
JUNIPR: 1804.09720
Binary JUNIPR: 1906.10137

- Particle Flow Network (PFN) or ParticleNet:

  Classifiers estimating $p_0(x,\theta)/p_1(x,\theta)$ directly

Komiske, Metodiev, Thaler: 1810.05165
Qu, Gouskos: 1902.08570

Network Architecture

- Particle Flow Network (PFN)
  - default hyper-parameters from arXiv:1810.05165

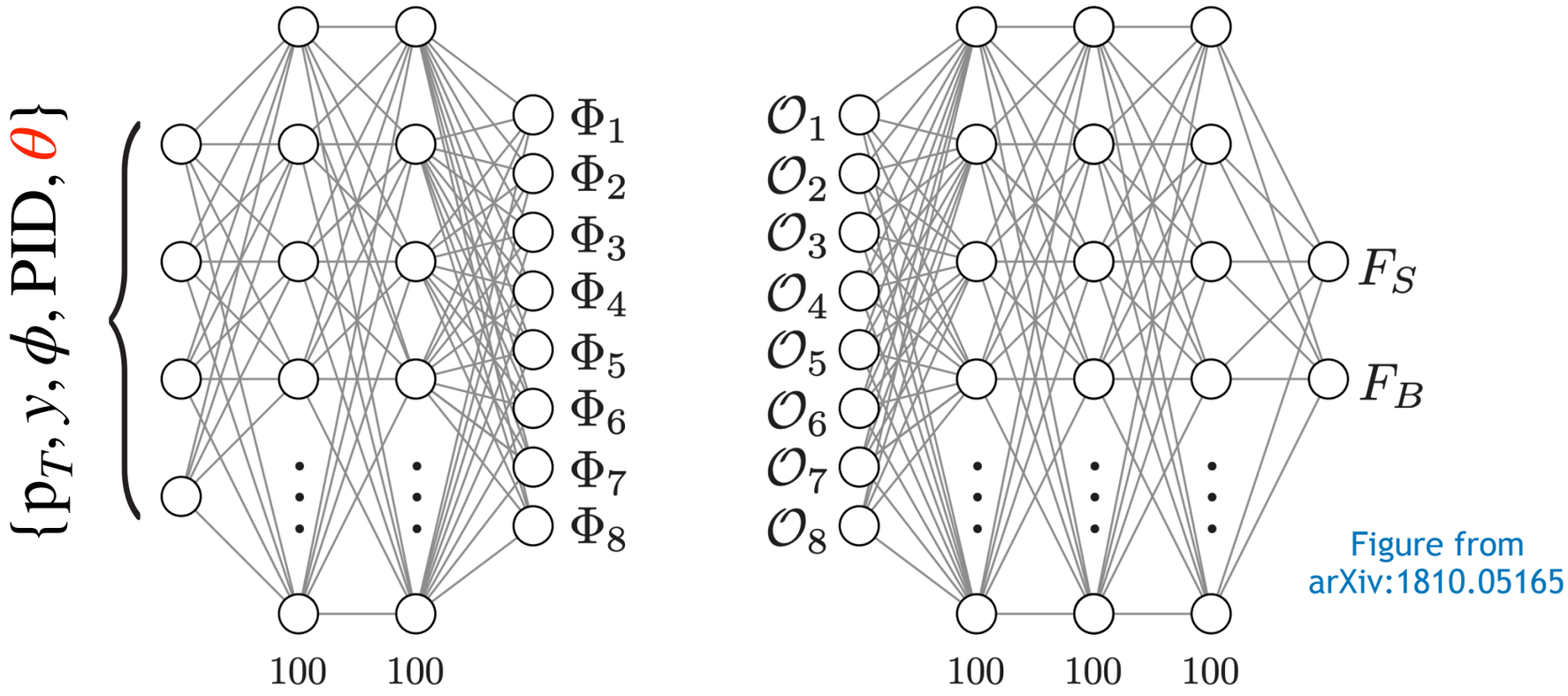


Figure from arXiv:1810.05165

- Latent space dimension $\ell = 128$

- 10-15 min to train on a NVIDIA GeForce GTX 1080 GPU

MC Simulation Data

- PYTHIA 8.230
  $e^+e^- \to Z \to \text{dijets}$
  Clustered with anti-k_t, $R = 0.8$

- Training data: 1M events at
  - Default Monash tune
  - `Timeshower` : `alphaSvalue` $\in [0.10, 0.18]$
  - `StringZ` : `aLund` $\in [0.50, 0.90]$
  - `StringFlav` : `probStoUD` $\in [0.10, 0.30]$

Default value
0.1365
0.68
0.217

MC Simulation Data

- PYTHIA 8.230
  $e^+e^- \to Z \to \text{dijets}$
  Clustered with anti-k_t, $R = 0.8$

- Training data: 1M events at
  - Default Monash tune
  - `Timeshower` : `alphaSvalue` $\in [0.10, 0.18]$
  - `StringZ` : `aLund` $\in [0.50, 0.90]$
  - `StringFlav` : `probStoUD` $\in [0.10, 0.30]$

  Default value
  0.1365
  0.68
  0.217

- 1M events sampling in all three parameters simultaneously

MC Simulation Data

- PYTHIA 8.230
  $e^+e^- \to Z \to \text{dijets}$
  Clustered with anti-k_t, $R = 0.8$

- Training data: 1M events at
  - Default Monash tune
  - `Timeshower` : `alphaSvalue` $\in [0.10, 0.18]$
  - `StringZ` : `aLund` $\in [0.50, 0.90]$
  - `StringFlav` : `probStoUD` $\in [0.10, 0.30]$

  Default value
  0.1365
  0.68
  0.217

- 1M events sampling in all three parameters simultaneously

- +1M validation events at fixed parameter points

Reweighting Results

Timeshower : AlphaSvalue

- Classifier trained to distinguish
  - default Monash
  - $\alpha_s \in [0.10, 0.18]$

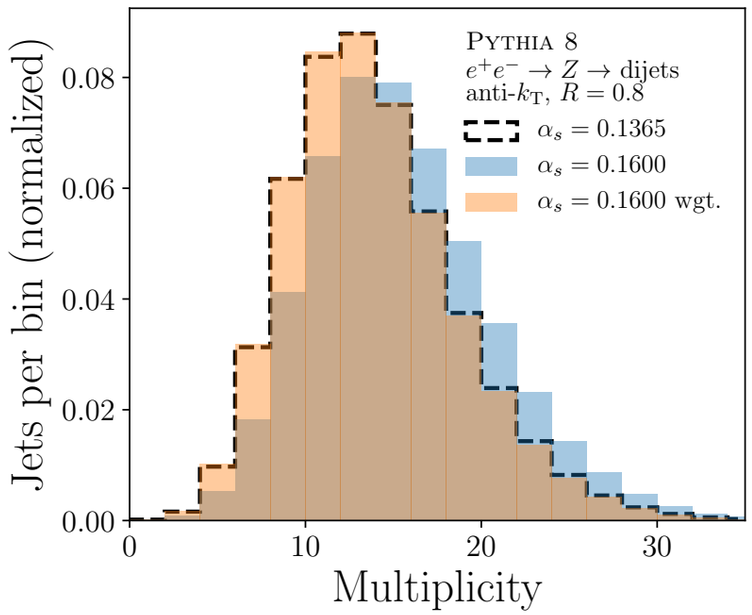
- Validation:
  Reweight $\alpha_s = 0.16$
  to default

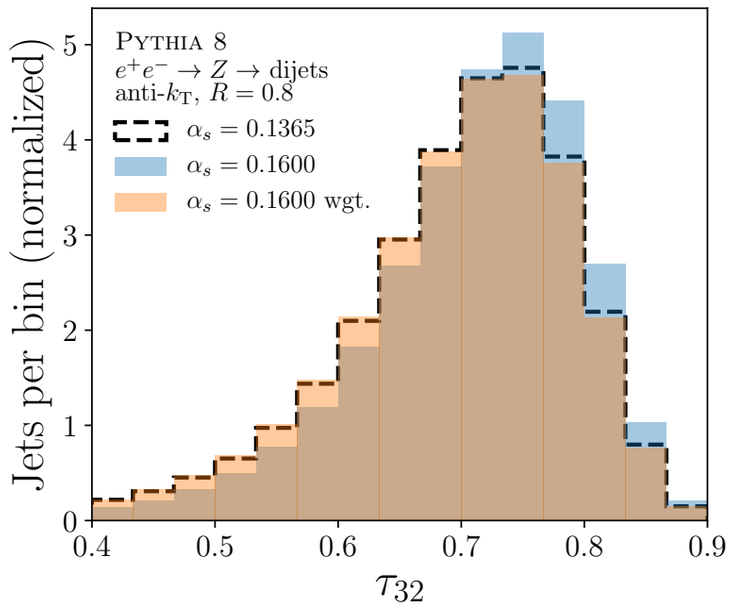
Timeshower : AlphaSvalue

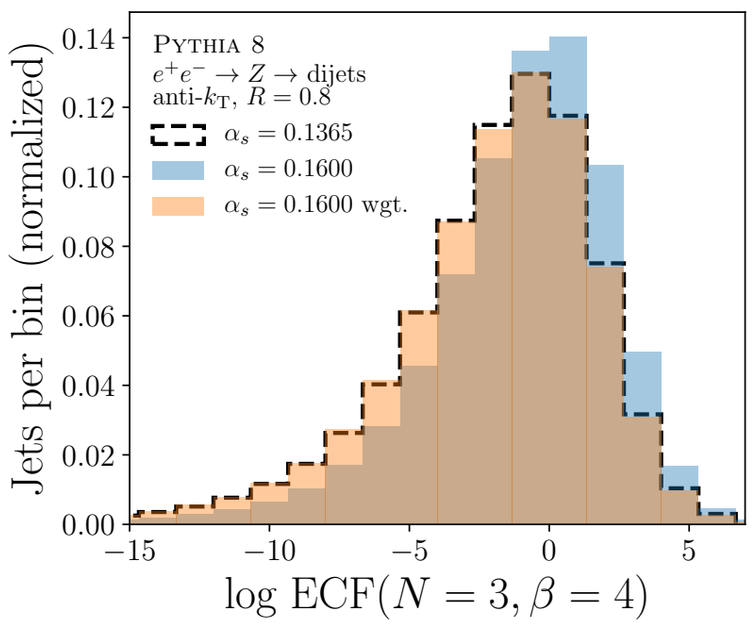
- Classifier trained to distinguish
  - default Monash
  - $\alpha_s \in [0.10, 0.18]$

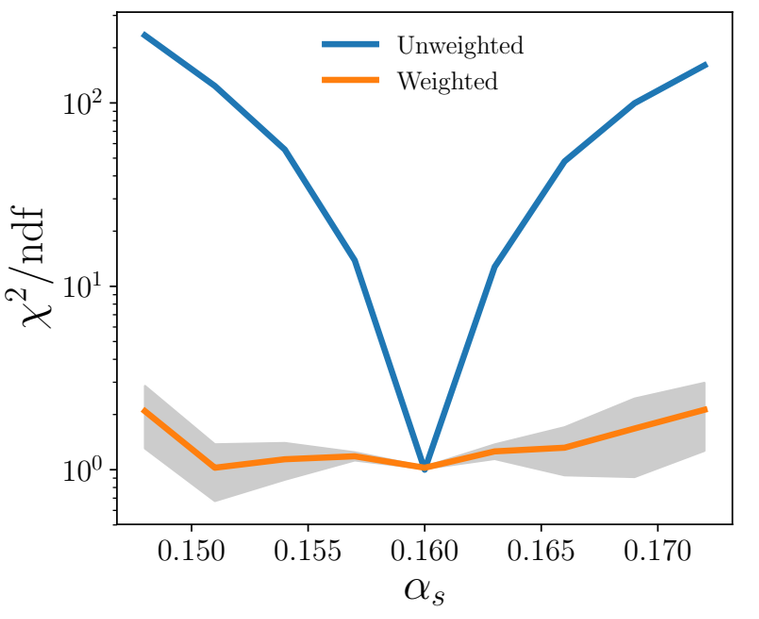
- Validation:
  Reweight $\alpha_s = 0.16$
  to default

- $\chi^2/\mathrm{ndf} \approx 1$



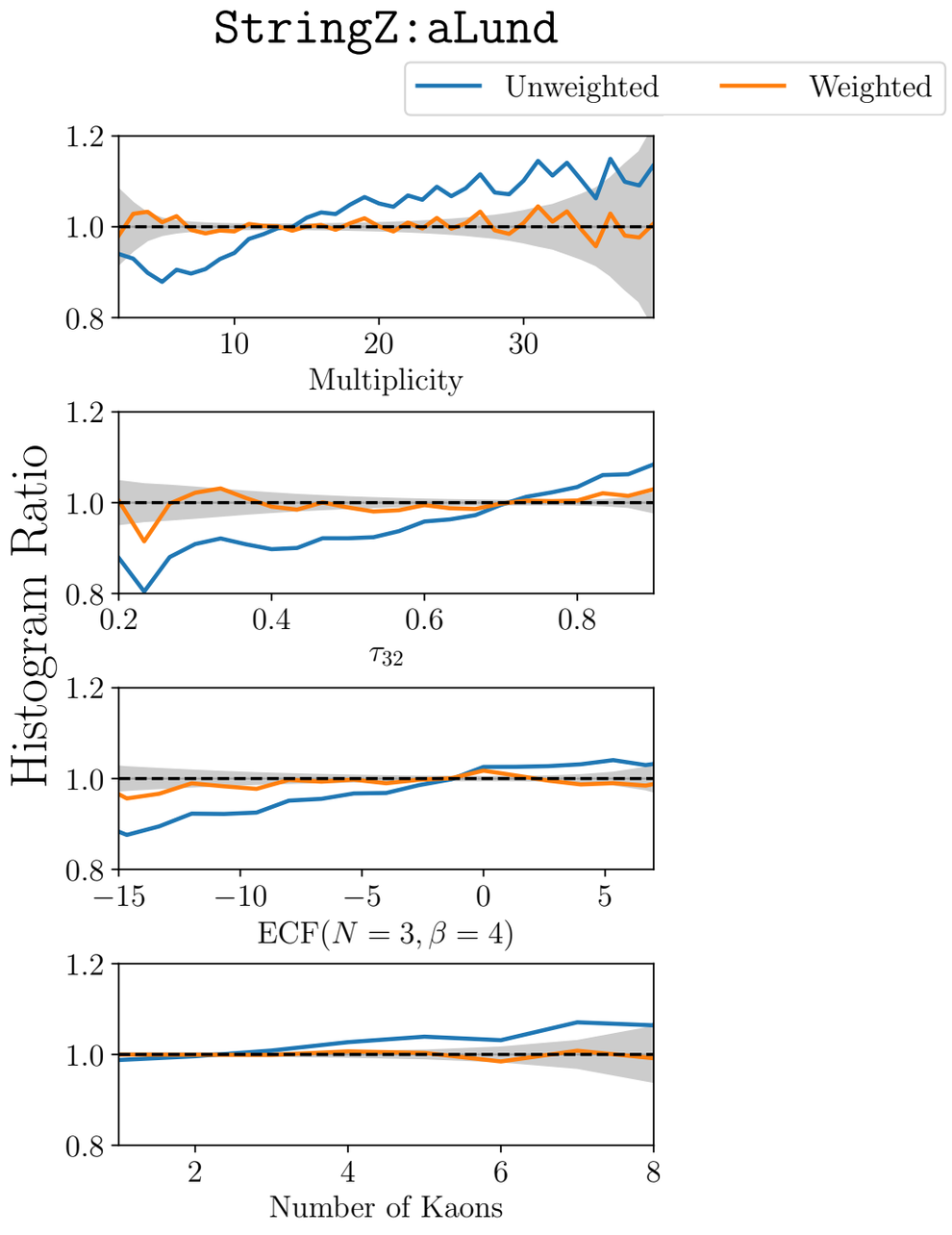






Reweighting less sensitive parameters

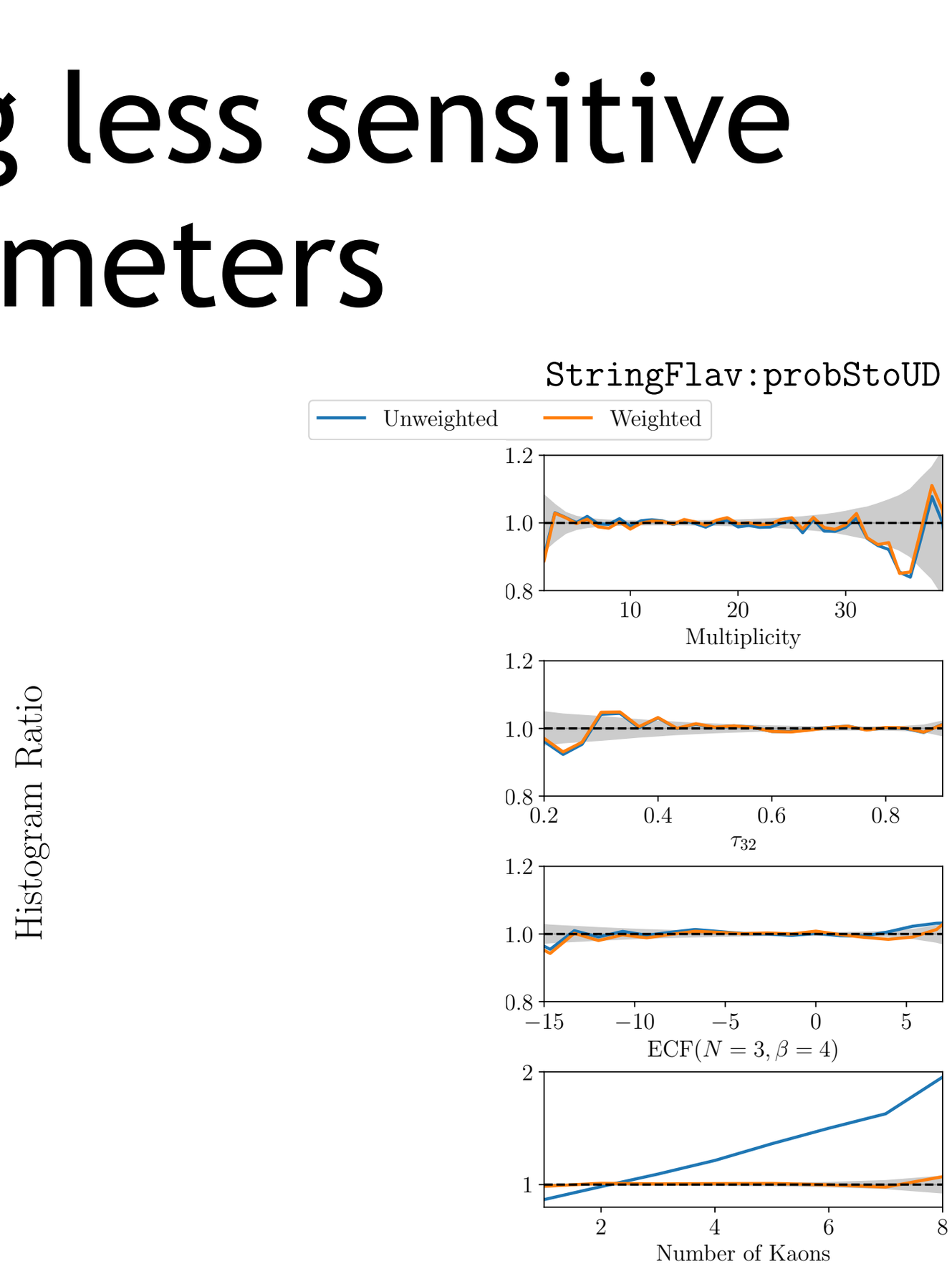
- `StringZ : aLund`
  percent-level differences on multiple observables



Reweighting less sensitive parameters

- `StringZ` : `aLund`
  percent-level differences on multiple observables

- `StringFlav` : `probStoUD`
  only affects strange particles, like kaons

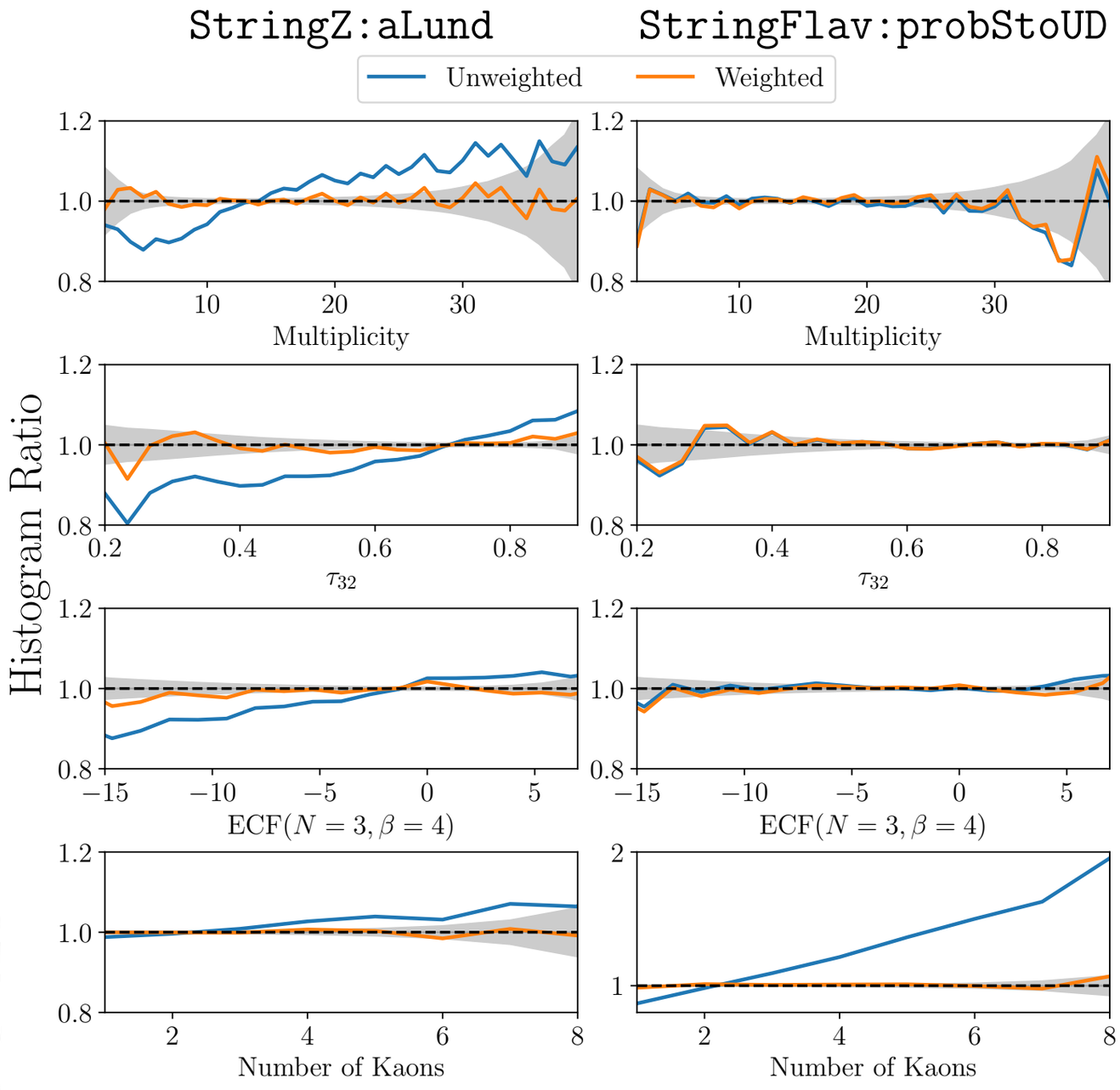


Reweighting less sensitive parameters

- `StringZ:aLund` percent-level differences on multiple observables

- `StringFlav:probStoUD` only affects strange particles, like kaons

Reweighting with DCTR effectively captures subtle and isolated modifications to the phase space

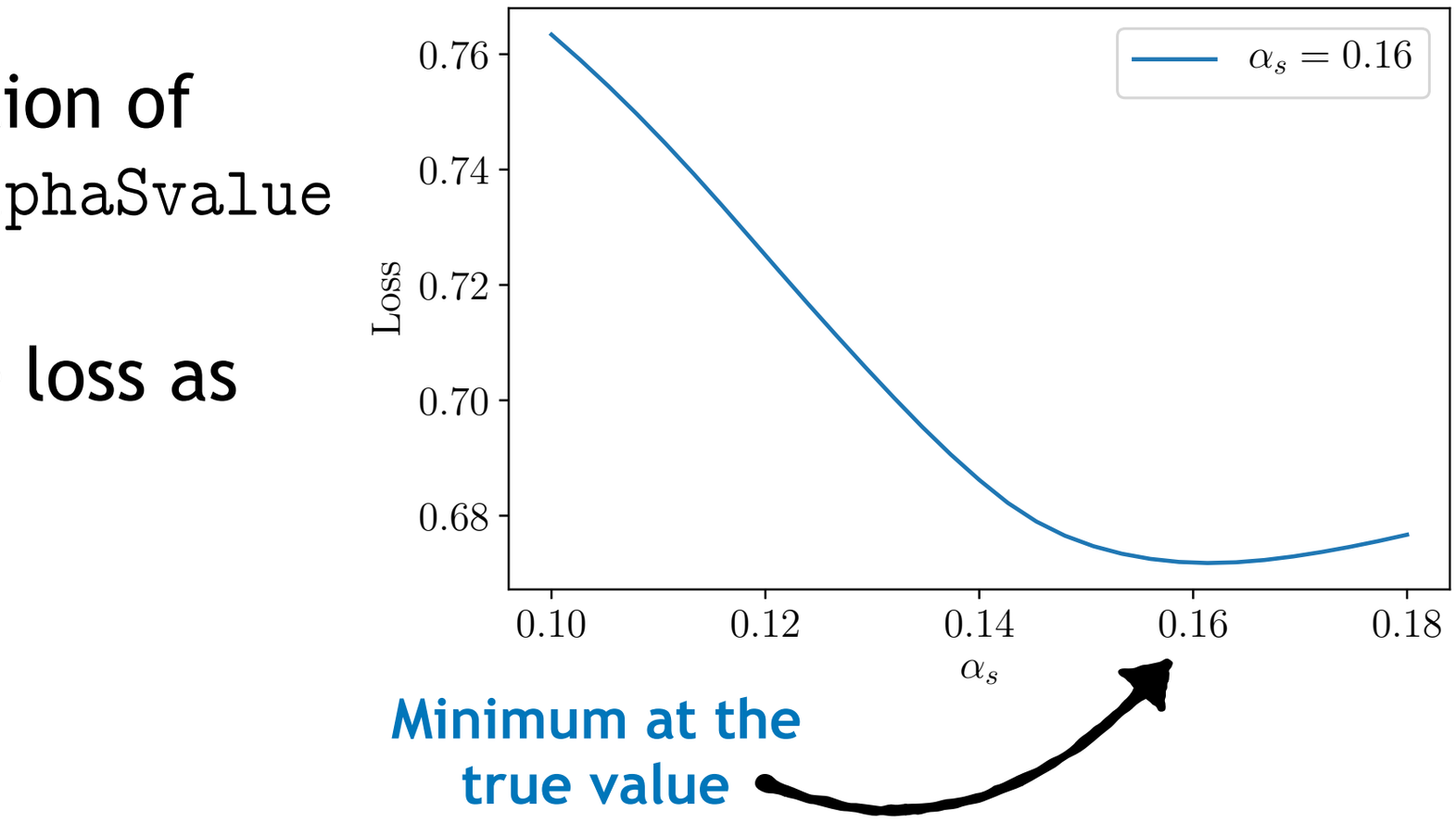


Tuning Results

1D Fit with DCTR

- Consider the variation of
  α_s = `TimeShower:alphaSvalue`

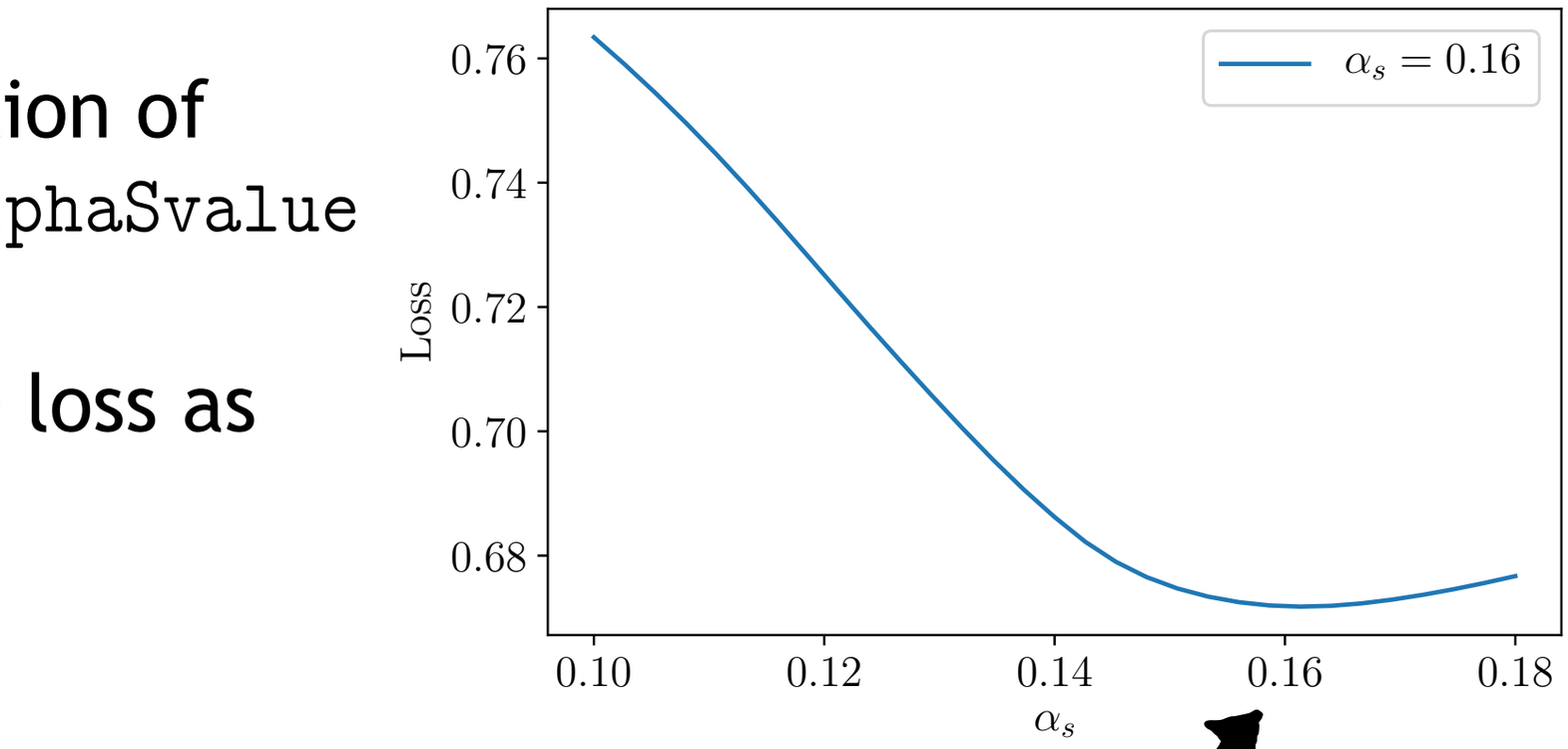
- Explicitly calculate loss as a function of α_s



1D Fit with DCTR

- Consider the variation of
  α_s = `TimeShower:alphaSvalue`

- Explicitly calculate loss as a function of α_s



Minimum at the true value

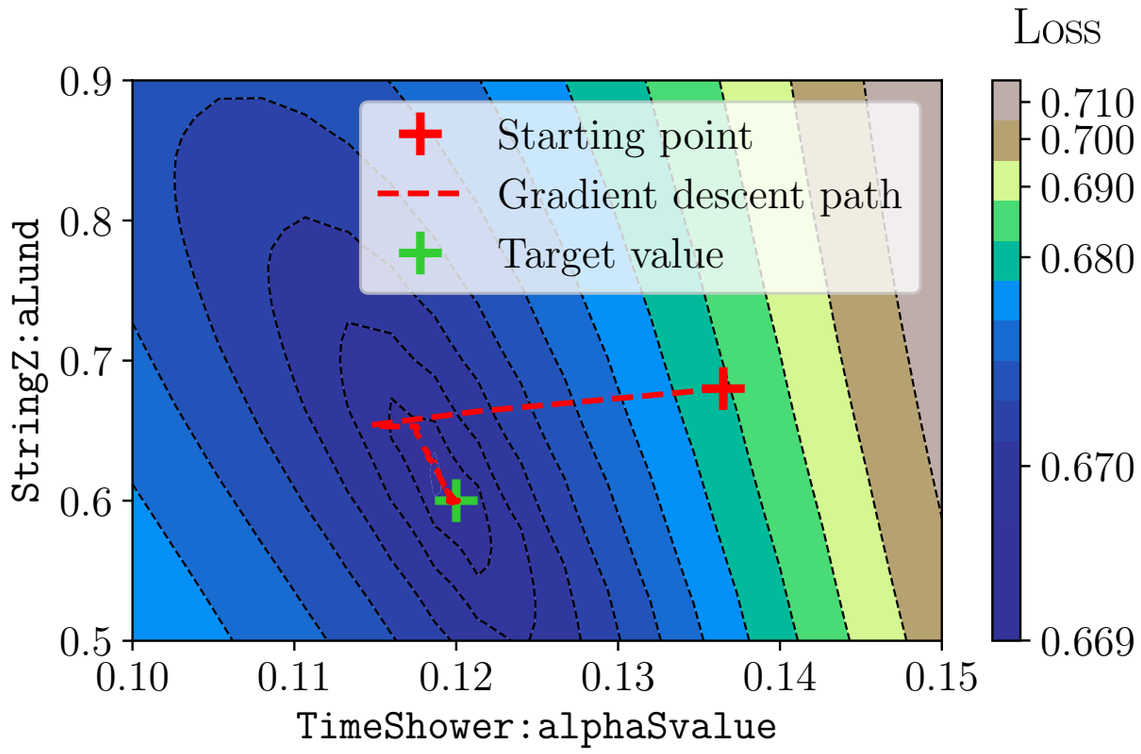
- Results for three separate 1D fits:

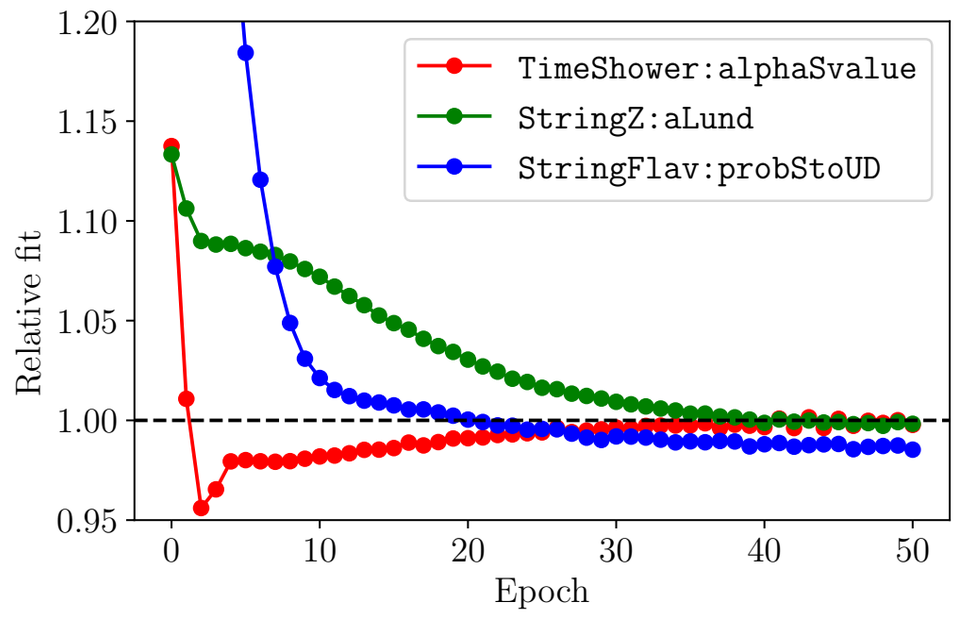
Parameter	Target value	Fit value
`TimeShower:alphaSvalue`	0.1600	0.1601 ± 0.0018
`StringZ:aLund`	0.8000	0.7980 ± 0.0257
`StringFlav:probStoUD`	0.2750	0.2754 ± 0.0065

Mean and standard deviation over 10 runs

Fitting in 3D with DCTR

- MC sample with variations in all three parameter simultaneously
- Gradient descent using Adam





Fitting in 3D with DCTR

Mean and standard deviation over 20 runs:

	Parameter	Target value	Fit value
Val.	TimeShower:alphaSvalue	0.1200	0.1195 ± 0.0022
	StringZ:aLund	0.6000	0.6276 ± 0.0373
	StringFlav:probStoUD	0.1200	0.1203 ± 0.0071

Fitting in 3D with DCTR

Mean and standard deviation over 20 runs:

	Parameter	Target value	Fit value
Val.	TimeShower:alphaSvalue	0.1200	0.1195 ± 0.0022
	StringZ:aLund	0.6000	0.6276 ± 0.0373
	StringFlav:probStoUD	0.1200	0.1203 ± 0.0071
Blinded	TimeShower:alphaSvalue	??	0.1707 ± 0.0022
	StringZ:aLund		0.7425 ± 0.0453
	StringFlav:probStoUD		0.1422 ± 0.0065

Fitting in 3D with DCTR

Mean and standard deviation over 20 runs:

	Parameter	Target value	Fit value
Val.	TimeShower:alphaSvalue	0.1200	0.1195 ± 0.0022
	StringZ:aLund	0.6000	0.6276 ± 0.0373
	StringFlav:probStoUD	0.1200	0.1203 ± 0.0071
Blinded	TimeShower:alphaSvalue	0.1700	0.1707 ± 0.0022
	StringZ:aLund	0.7500	0.7425 ± 0.0453
	StringFlav:probStoUD	0.1400	0.1422 ± 0.0065

Fitting in 3D with DCTR

Mean and standard deviation over 20 runs:

	Parameter	Target value	Fit value
Val.	TimeShower:alphaSvalue	0.1200	0.1195 ± 0.0022
	StringZ:aLund	0.6000	0.6276 ± 0.0373
	StringFlav:probStoUD	0.1200	0.1203 ± 0.0071
Blinded	TimeShower:alphaSvalue	0.1700	0.1707 ± 0.0022
	StringZ:aLund	0.7500	0.7425 ± 0.0453
	StringFlav:probStoUD	0.1400	0.1422 ± 0.0065

Similar uncertainty

1D:

Parameter	Target value	Fit value
TimeShower:alphaSvalue	0.1600	0.1601 ± 0.0018
StringZ:aLund	0.8000	0.7980 ± 0.0257
StringFlav:probStoUD	0.2750	0.2754 ± 0.0065

Conclusions and Outlook

DCTR:
Deep neural network using Classification
for Tuning and Reweighting

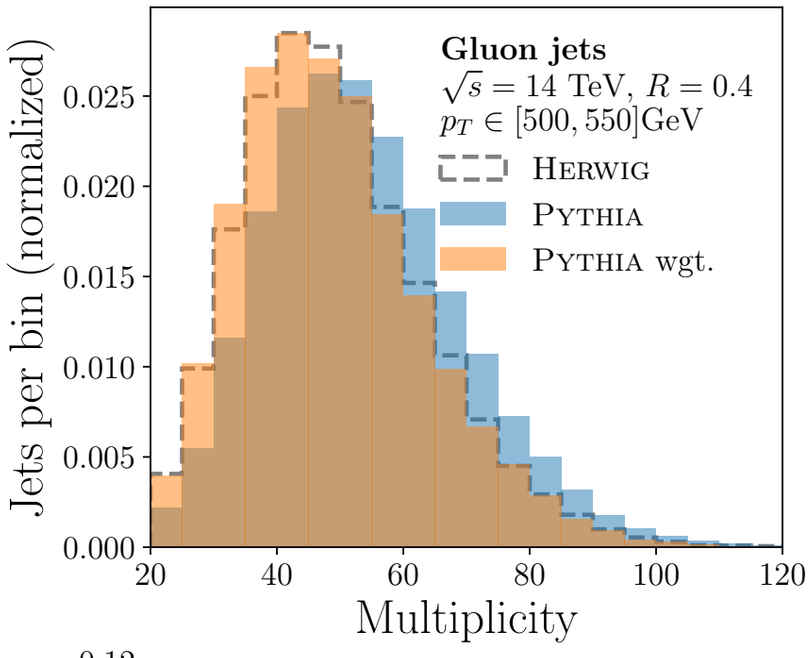
- Discrete reweighting:
  - New full-detector simulated samples if one other full simulation exists
    E.g. comparing Pythia vs Herwig, or legacy data analysis where original detector simulation is not available

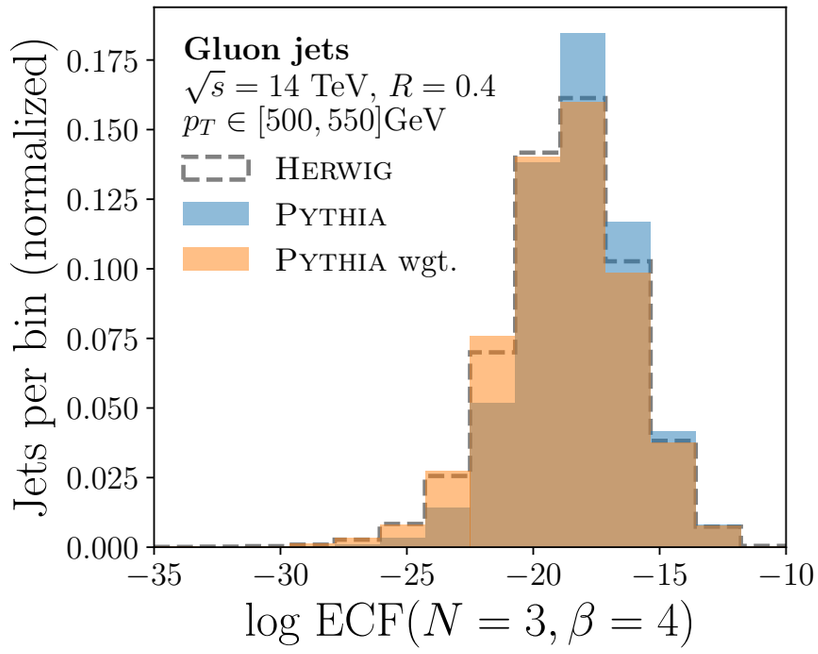
- Continuous reweighting:
  - Systematic parameter variations for uncertainty estimation
    where automated MC variations are not available/possible.

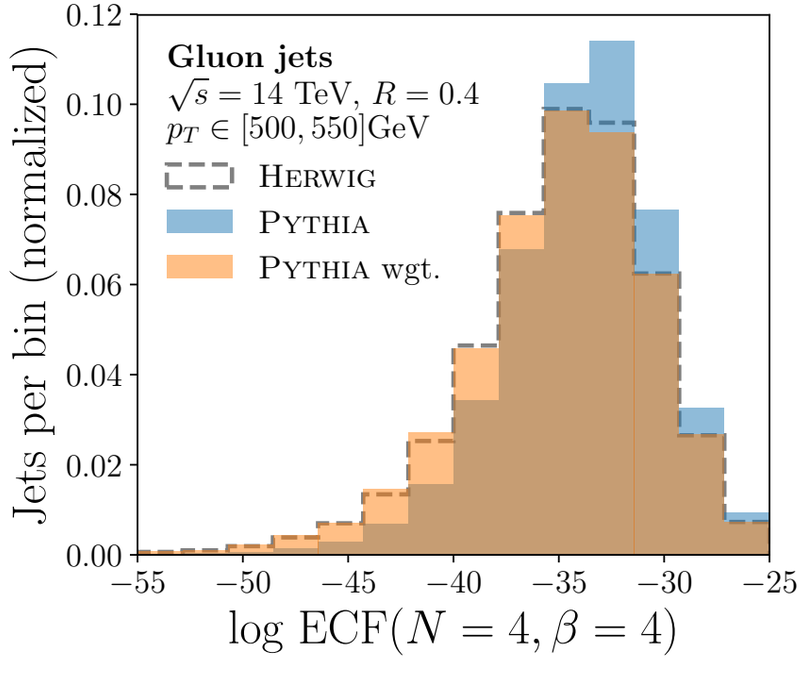
- Parameter tuning:
  - Tune MC with full phase-space information

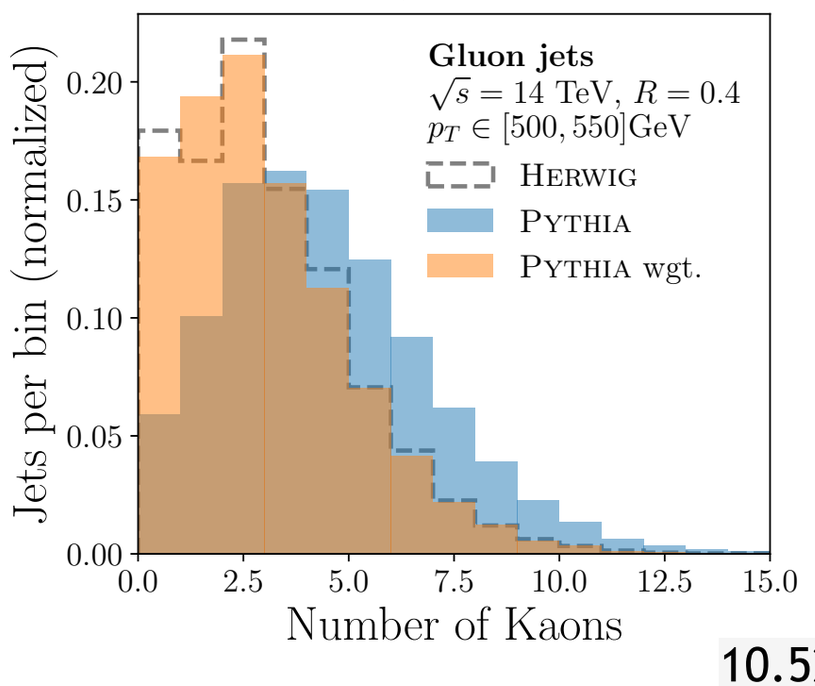
Backup Slides

pp Reweighting Pythia & Herwig



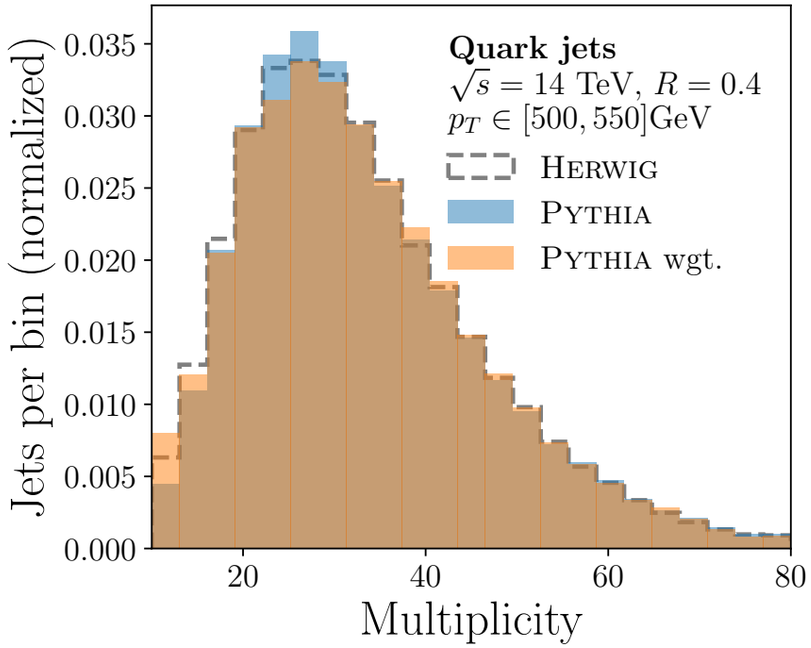


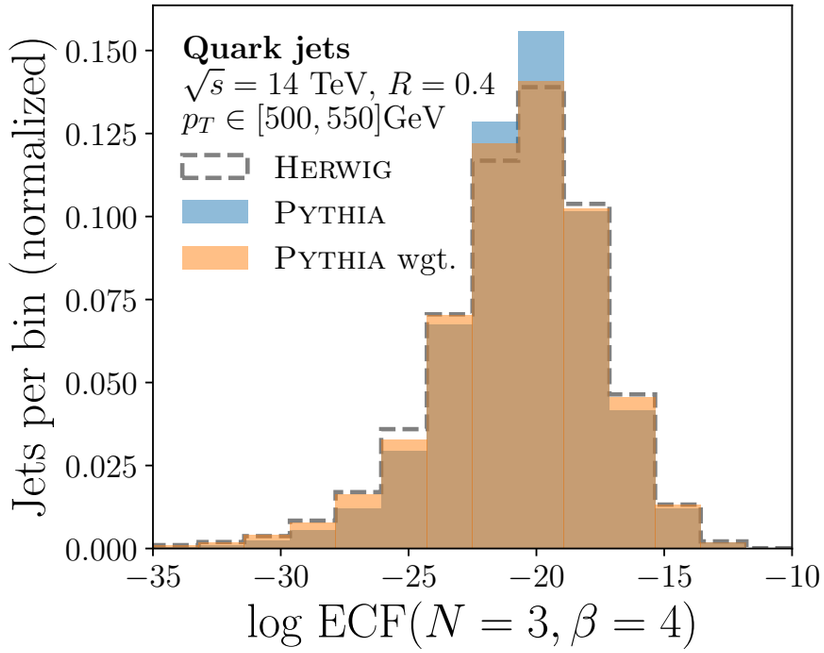


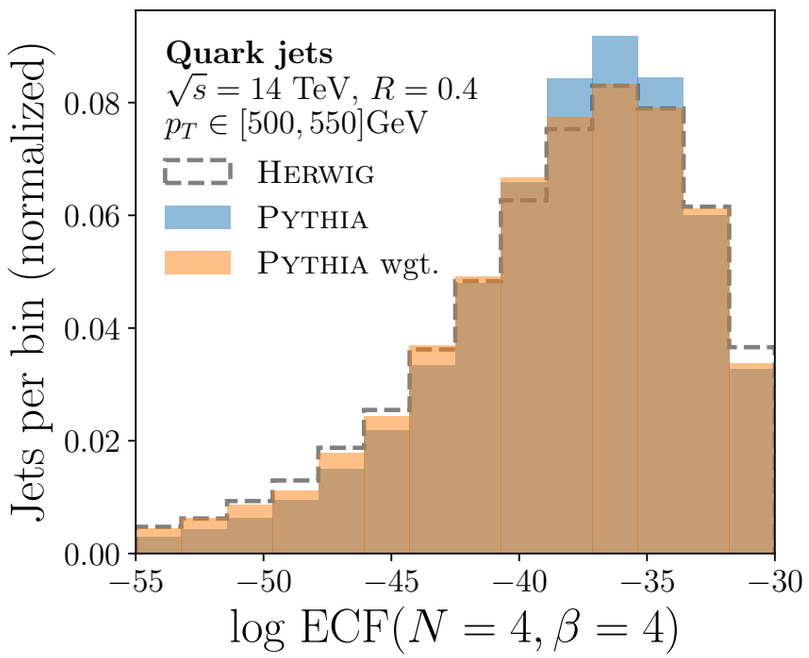


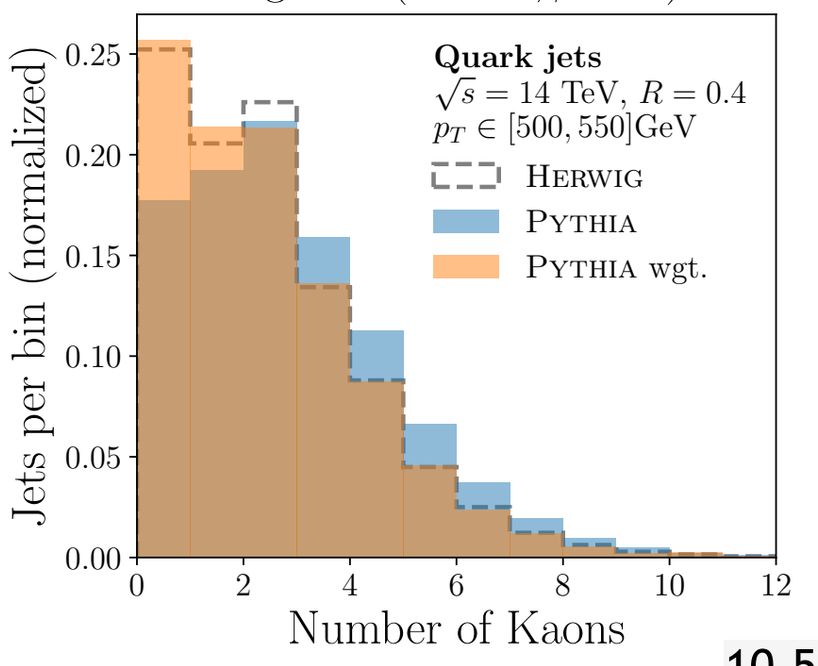
Samples from
10.5281/zenodo.2658764
10.5281/zenodo.3164691

pp Reweighting Pythia & Herwig









Samples from
10.5281/zenodo.2658764
10.5281/zenodo.3164691