Primary Vertex identification using deep learning in the ATLAS Experiment

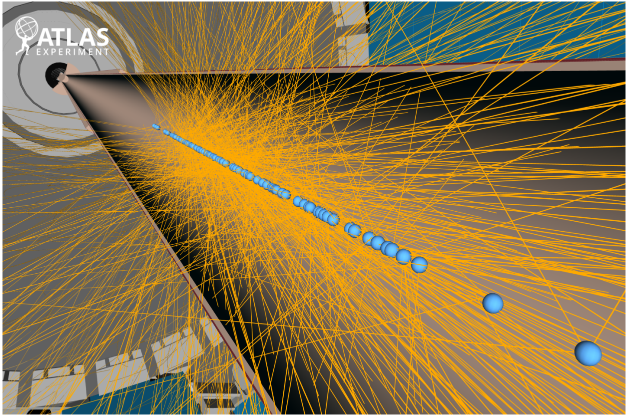
Rocky Garg, Qi Bin Lei, Lauren Tompkins

Stanford University

December 18th, 2024

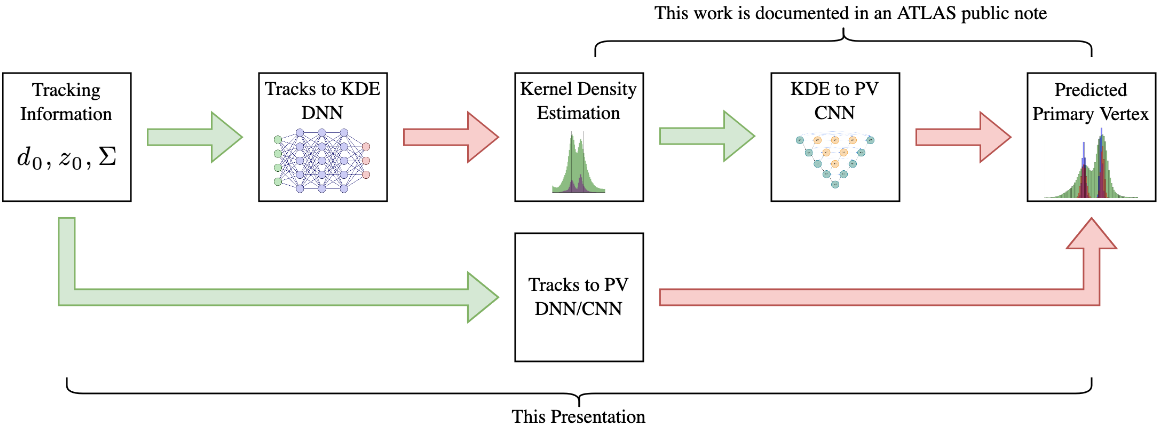
ATLAS EXPERIMENT

- Current vertexing algorithms are iterative by nature
- Motivations for ML
  - Fast and accurate predictions
  - Parallel training of events
  - Integration of new variables
  - Fine tuning for future upgrades/projects

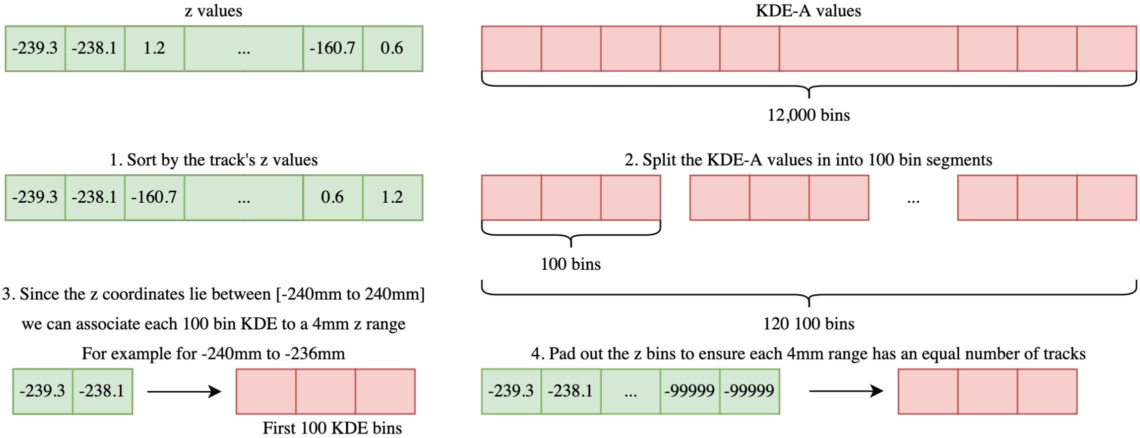


Simulation of Top-Antitop Pair Production at HL-LHC [Link]

PV-Finder Overview



Data Pre-Processing



5. Use sorted/binned index and length padding on d and covariance matrix values

Kernel Density Estimation

- Track's Gaussian Probability Density

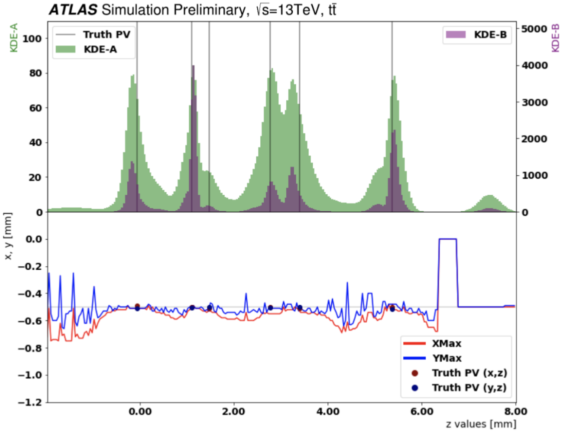
$$\mathbb{P}(r) = \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\{\alpha\}$$

$$\alpha = -\frac{1}{2}\left((d-d_0),(z-z_0)\right)^T \Sigma^{-1}\left((d-d_0),(z-z_0)\right)$$

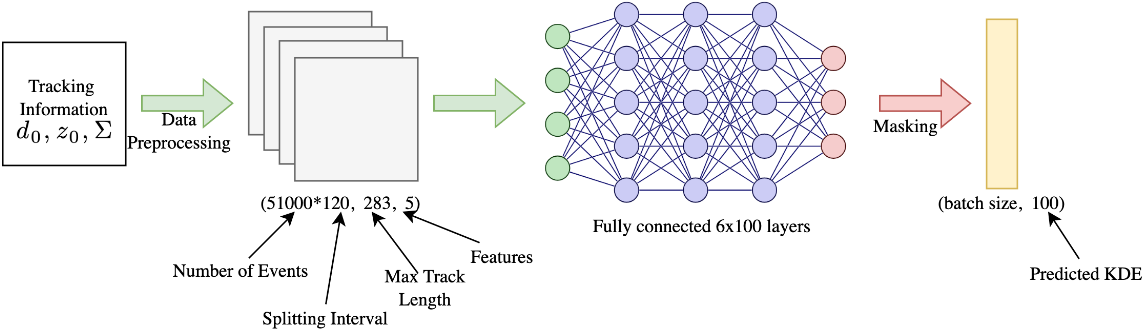
KDE-A = $\sum_{\text{tracks}} \mathbb{P}(r)$

KDE-B = $\sum_{\text{tracks}} \mathbb{P}(r)^2$

- KDEs are composed of 12,000 bins
- Current analytical KDE production:
  - Evaluates the Gaussian PDF for all particle track
  - ~50 million evaluations per track per event

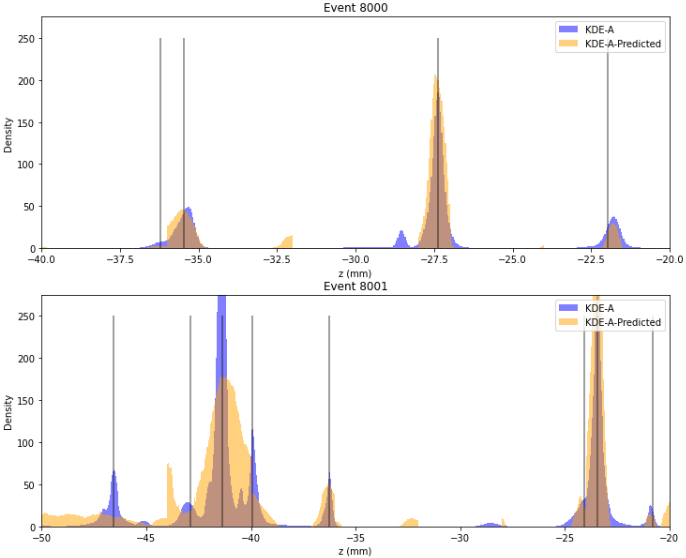


From ATLAS Public Note [1]

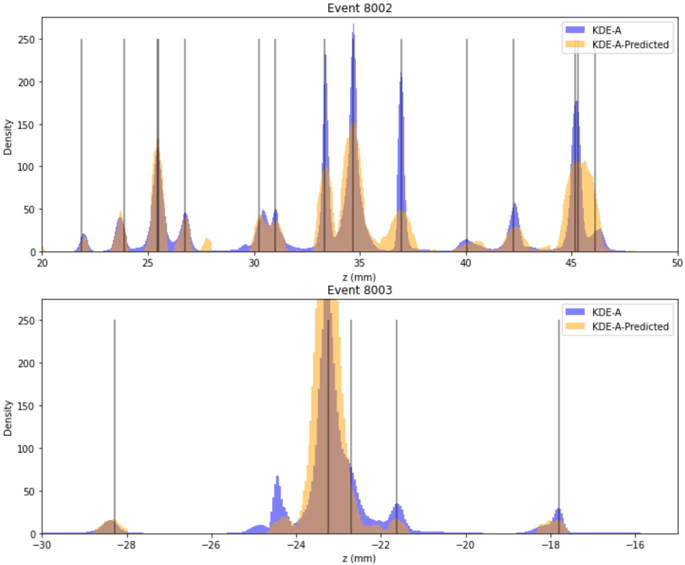


Tracking Information
d_0, z_0, Σ
Data Preprocessing
(51000*120, 283, 5)
Number of Events
Splitting Interval
Max Track Length
Features
Fully connected 6x100 layers
Masking
(batch size, 100)
Predicted KDE

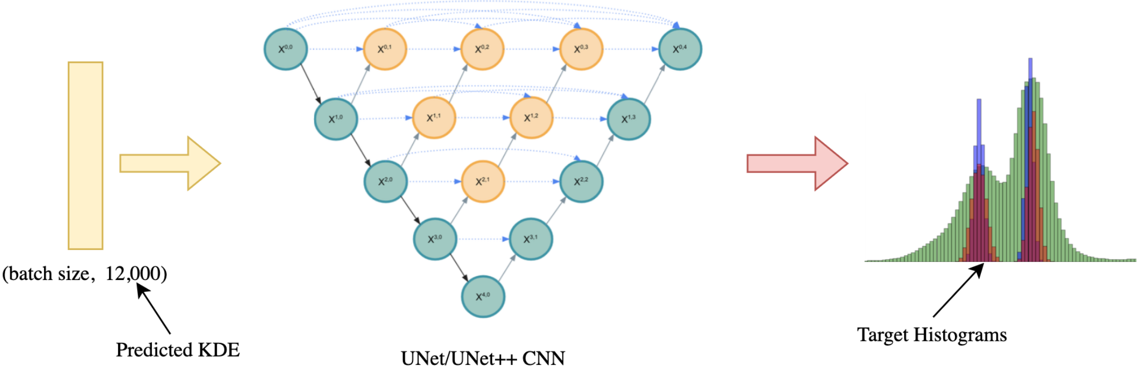
Example of Tracks to KDE Output (1)



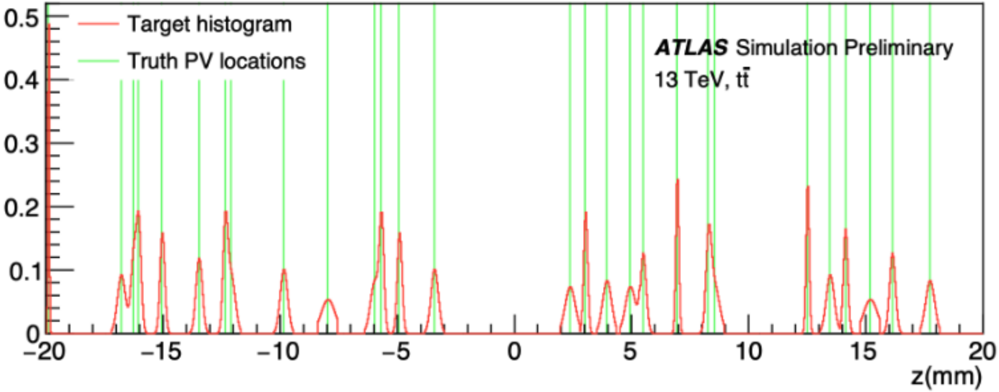
Example of Tracks to KDE Output (2)



KDE to Hists Architecture



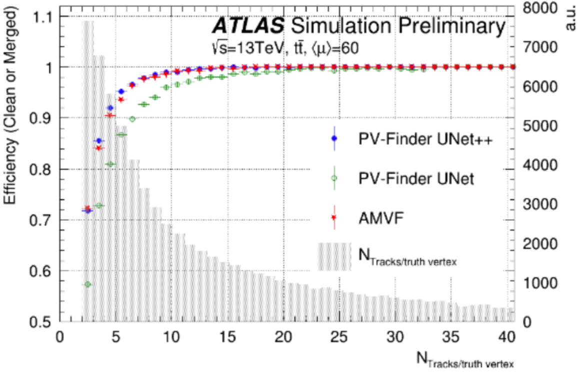
Primary Vertex Information



Target Truth Information [1]

Previous Work

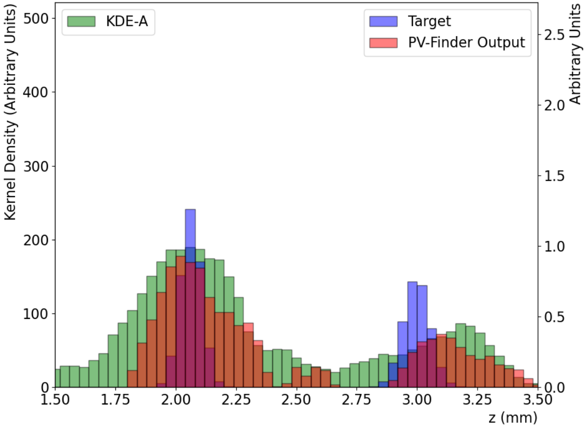
- Trained using all KDEs produced from the combinatorial method
- Efficiency $= \frac{\text{\# reconstructed vertices}}{\text{\# truth vertices}}$
- Prior work noted in pubnote [1]



Model	Efficiency	False Positive Rate (# / event)
PV-Finder UNet++	94.2%	1.5
PV-Finder UNet	88.7%	2.6
AMVF	93.9%	0.8

Example End-to-end KDE to Hists Output

- Training on predicted KDE-A
  - Using prior outputs from tracks to KDE
  - ~ 3 days of training
- Efficiency: $\sim 62\%$
- False Positive Rate: 5.5
  - Only improvements from here with more information



Summary

- Machine learning provides a viable avenue for primary vertex identificaiton
- NN-KDE network trained on one feature (KDE-A) is performing well
- Upcoming Work:
  - Train NN-KDE on other KDEs (KDE-B-z, KDE-A-x, KDE-A-y)
  - Training UNet and UNet++ architectures on NN-KDEs
  - Compare preliminary efficiency with AMVF [1]

Thanks for listening!

Relevant Links and Directories

- Ananya’s prior work on tracks to KDE
  - Link to Gitlab
  - Link to Google Doc Documentation
- LHCb PVFinder
  - Link to Gitlab
- ATLAS PV Finder
  - Link to Gitlab

Bibliography (1)

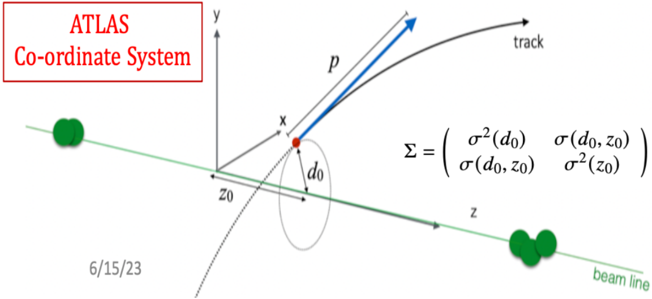
[1] ATLAS. “Primary Vertex identification using deep learning in ATLAS”. In: (2023). ATL-PHYS-PUB-2023-011.

[2] ATLAS Collaboration. “Development of ATLAS Primary Vertex Reconstruction for LHC Run 3”. In: (2019). ATL-PHYS-PUB-2019-015.

Backup

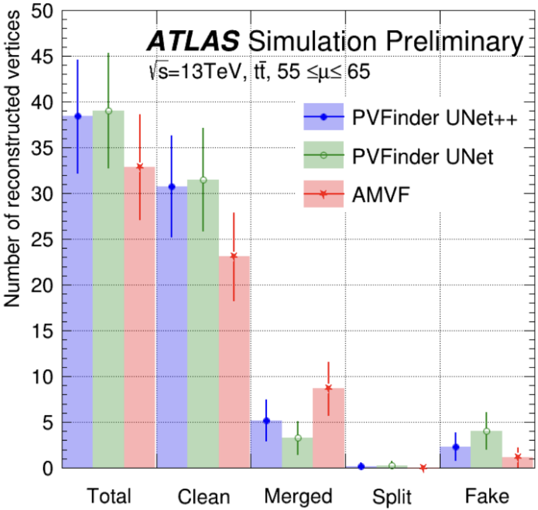
Tracks

- Gathering charged particle measurements readout from the detector
- Calculate trajectory estimation
- Used as inputs for reconstruction efforts and analysis



Reconstructed Track Data

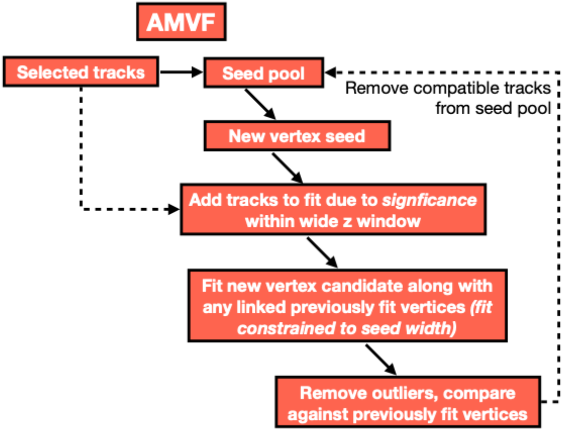
Clean, Merged, Split, Fake Prior Results



[1]

Current Primary Vertex Algorithm: AVMF

- Global approach to vertex finding and fitting
- Finds and fits tracks based on compatibility to different primary vertices
- Limitations
  1. Parallelization
  2. Time scale increases non linearly with number of PVs



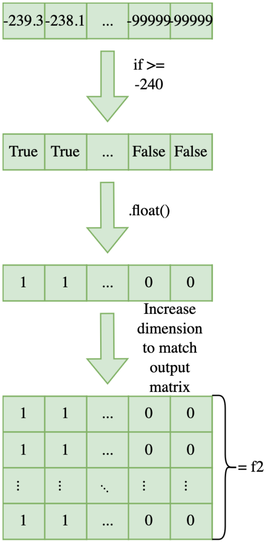
[2]

Data Statistics

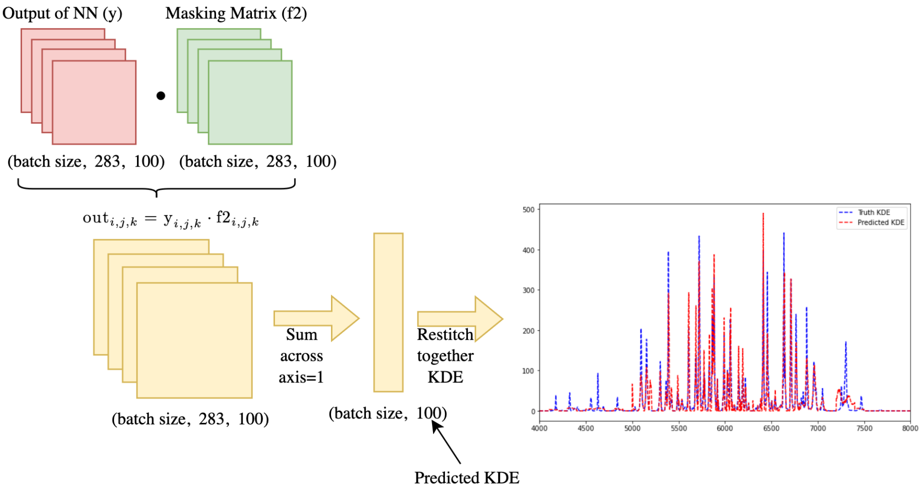
- Total Events: 51000
- Input Features: $d_0, z_0, \sigma(d_0), \sigma(z_0), \sigma(d_0, z_0)$ (~ 1000 tracks per event)
  - $z_0 \in [-240\text{mm}, 240\text{mm}]$
- Output Label: KDE-A (12,000 Bins)
- How should we make it easier for the neural network to learn?
  - Padding input features make it all the same length
  - Splitting KDE-A bins

Masking

- A mask is used to identify valid tracks that should contribute to the final result (not padded value)
- The masking matrix, f2, is then multiplied with the output of the neural network to either
  1. Zero out outputs of the neural network that do not have any track data associated with it
  2. Contribute values that will be summed to produce the predicted KDE bins

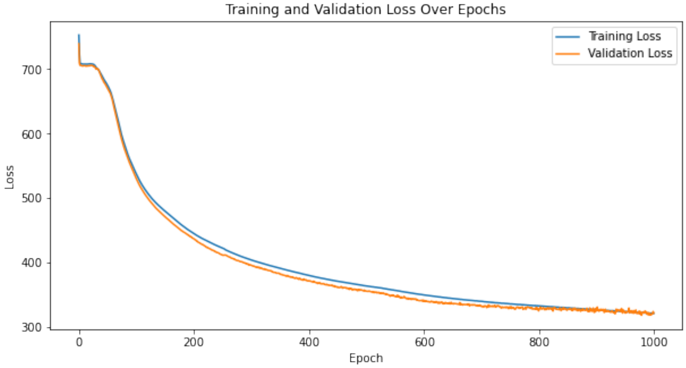


Masking Output

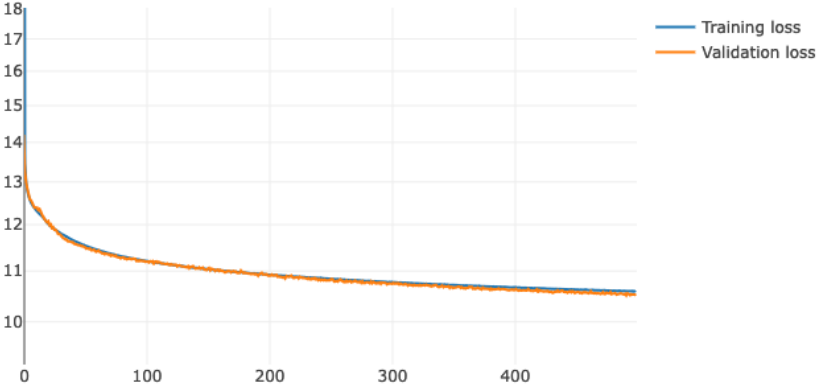


Loss Function

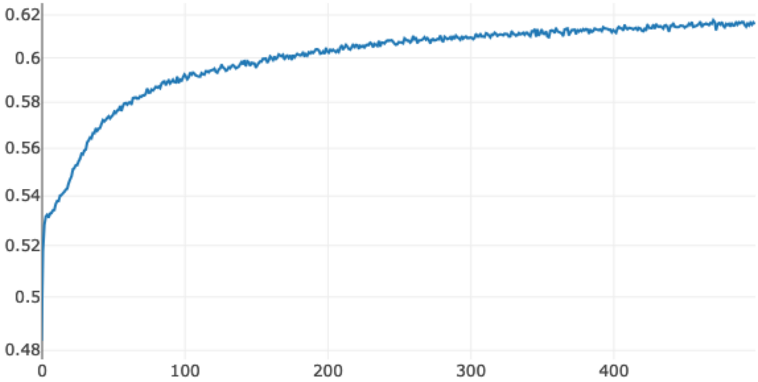
- Currently working with mean square error loss
  - Will try out weighted mean square error loss to capture the peaks of the KDE better
- Plateus for the first $\sim 50 - 100$ epochs then drops exponentially
- $\mathrm{MSE} = \frac{1}{n}\sum(\mathrm{KDE}_{\mathrm{pred}} - \mathrm{KDE}_{\mathrm{truth}})^2$



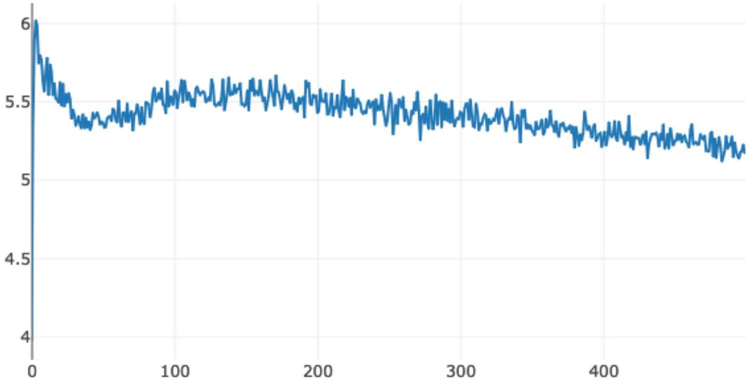
KDE to Hists Loss



KDE to Hists Efficiency



KDE to Hists FPR



KDE to Hists Results