GNN LST

Errata and early T5-NN results

April 11th, 2023

P. Chang, ***J. Guiang***

UC San Diego

UF UNIVERSITY of FLORIDA

Errata

Errata: Feature Normalization

- ML algorithms learn best when input features are normalized
  - When inputs span different numerical ranges, optimization tends to “bounce”
- So far, we have applied the following normalization:

$$x' = (x - x_{min})/(x_{max} - x_{min})$$

- **To-do:** determine how we would want to do this in practice
  - Strategy above requires seeing all of the data, but other strategies exist
- However, **there was a bug in implementation (next slide)**
  - Spoiler: it’s easy to switch row-wise with column-wise…
  - Lesson learned: check your matrix operations!

Errata: Feature Normalization

$$X = \begin{pmatrix} (x_1 \quad x_2 \quad x_3 \quad \dots \quad x_{N_f})_1 \\ (x_1 \quad x_2 \quad x_3 \quad \dots \quad x_{N_f})_2 \\ (x_1 \quad x_2 \quad x_3 \quad \dots \quad x_{N_f})_3 \\ \vdots \\ (x_1 \quad x_2 \quad x_3 \quad \dots \quad x_{N_f})_{N_e} \end{pmatrix}$$

N_e events

N_f features

Previous

$$x_i' = (x_i - x_{min})/(x_{max} - x_{min})$$

$$x_{min} = \min(x_1, x_2, \dots, x_{N_f})$$

$$x_{max} = \max(x_1, x_2, \dots, x_{N_f})$$

Current

$$x_i' = (x_i - x_{min})/(x_{max} - x_{min})$$

$$x_{min} = \min(x_{1,1}, x_{1,2}, \dots, x_{1,N_e})$$

$$x_{max} = \max(x_{1,1}, x_{1,2}, \dots, x_{1,N_e})$$

Errata: Feature Normalization

$$X = \begin{pmatrix} (x_1 & x_2 & x_3 & \dots & x_{N_f})_1 \\ (x_1 & x_2 & x_3 & \dots & x_{N_f})_2 \\ (x_1 & x_2 & x_3 & \dots & x_{N_f})_3 \\ & & \vdots & & \\ (x_1 & x_2 & x_3 & \dots & x_{N_f})_{N_e} \end{pmatrix}$$

N_e events

N_f features

Previous

$$x_i' = (x_i - x_{min})/(x_{max} - x_{min})$$

$$x_{min} = \min(x_1, x_2, \dots, x_{N_f})$$

$$x_{max} = \max(x_1, x_2, \dots, x_{N_f})$$

Current

$$x_i' = (x_i - x_{min})/(x_{max} - x_{min})$$

$$x_{min} = \min(x_{1,1}, x_{1,2}, \dots, x_{1,N_e})$$

$$x_{max} = \max(x_{1,1}, x_{1,2}, \dots, x_{1,N_e})$$

Errata: Feature Normalization

$$X = \begin{pmatrix} (x_1 & x_2 & x_3 & \dots & x_{N_f})_1 \\ (x_1 & x_2 & x_3 & \dots & x_{N_f})_2 \\ (x_1 & x_2 & x_3 & \dots & x_{N_f})_3 \\ & & \vdots & & \\ (x_1 & x_2 & x_3 & \dots & x_{N_f})_{N_e} \end{pmatrix}$$

N_e events

N_f features

Previous

$$x_i' = (x_i - x_{min})/(x_{max} - x_{min})$$

$$x_{min} = \min(x_1, x_2, \dots, x_{N_f})$$

$$x_{max} = \max(x_1, x_2, \dots, x_{N_f})$$

Current

$$x_i' = (x_i - x_{min})/(x_{max} - x_{min})$$

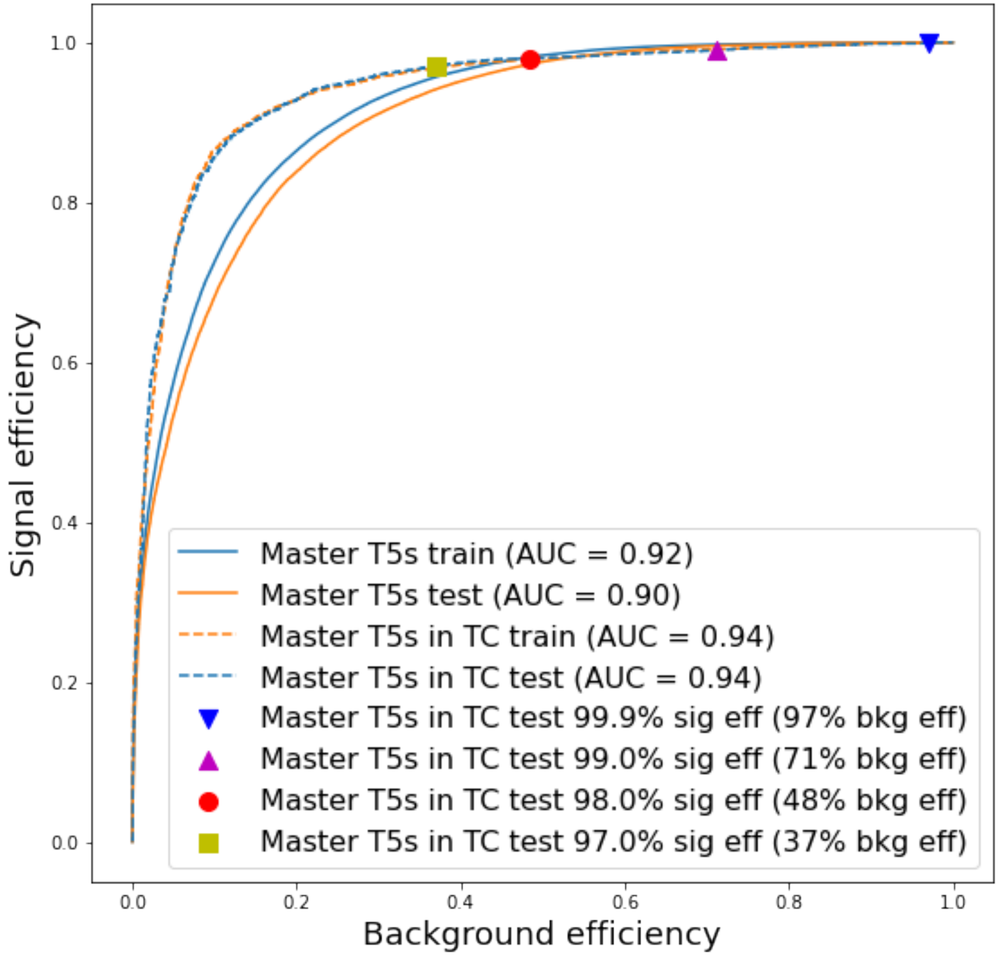
$$x_{min} = \min(x_{1,1}, x_{1,2}, \dots, x_{1,N_e})$$

$$x_{max} = \max(x_{1,1}, x_{1,2}, \dots, x_{1,N_e})$$

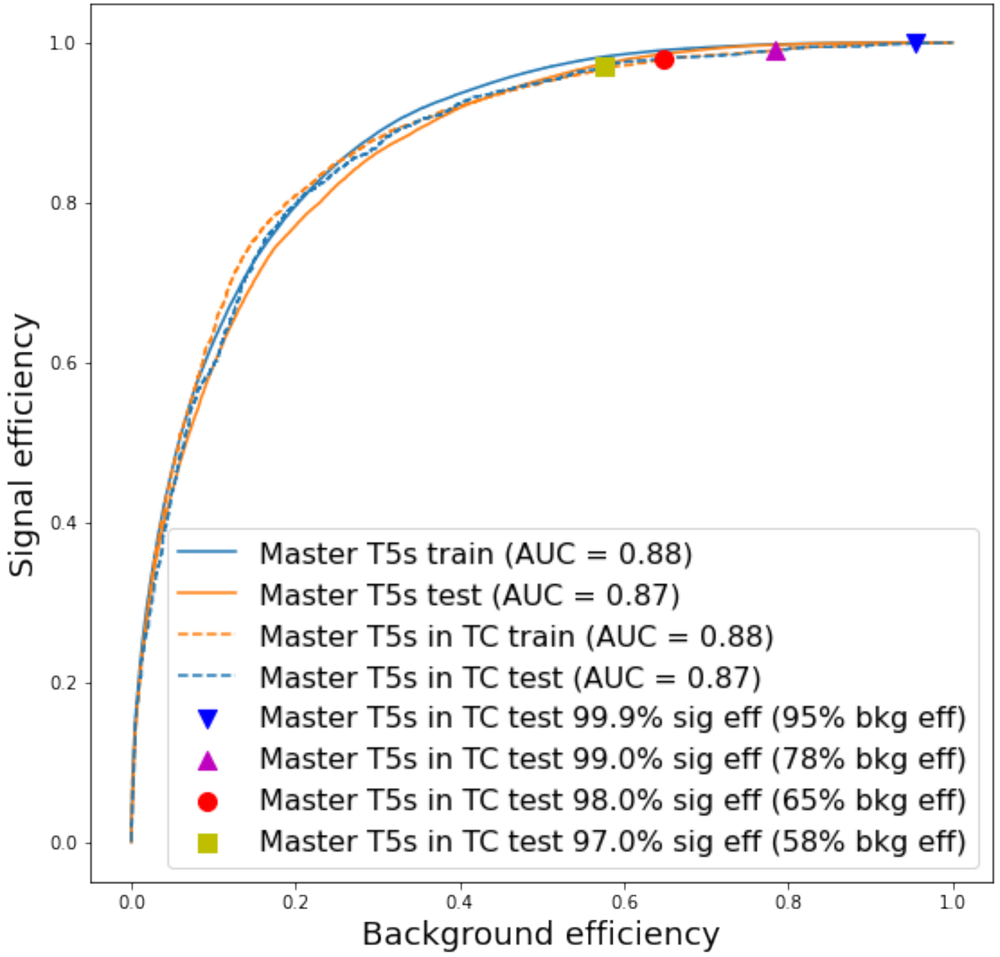
Errata: Trial 1 w/ Correct Normalization

Performance got… worse? (Ongoing investigation)

Previous normalization



Current normalization



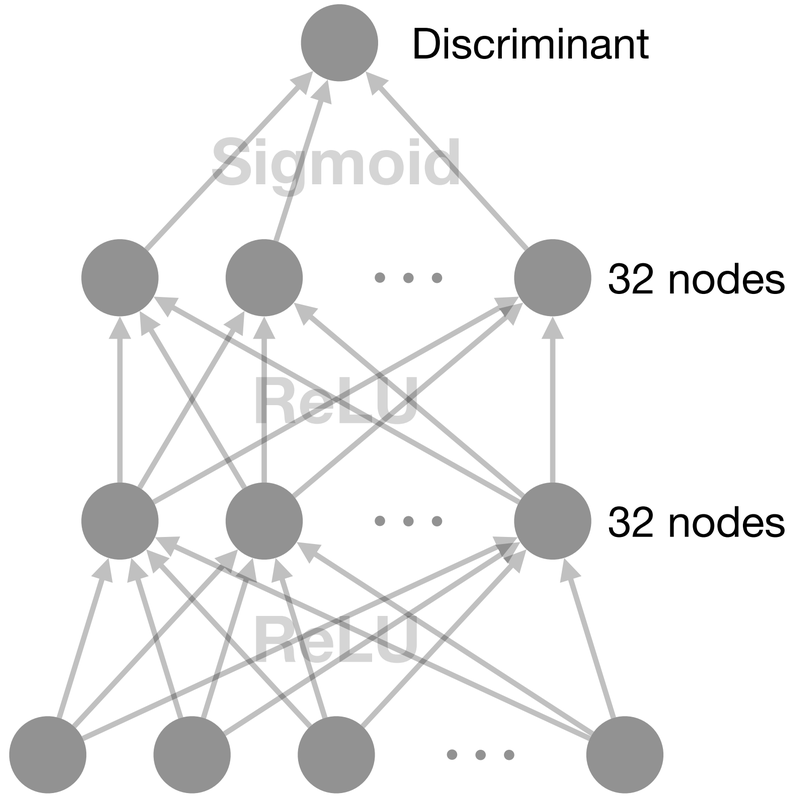
T5 Neural Network

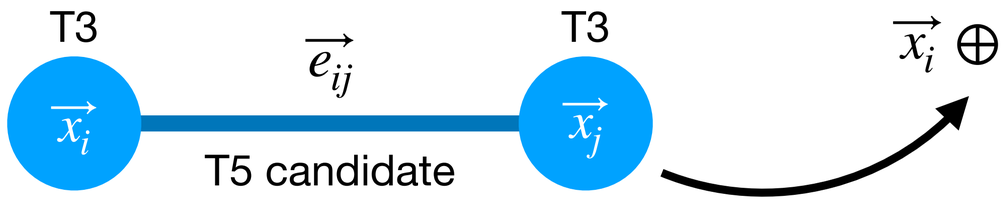
T5 Neural Network

Object	Feature
T3	p_T
	Inner anchor hit r, z, ϕ, η, δr, layer
	Middle anchor hit r, z, ϕ, η, δr, layer
	Outer anchor hit r, z, ϕ, η, δr, layer
T5 candidate	p_T, η, ϕ
	χ^2
	Inner T3 radius
	Bridge T3 radius
	Outer T3 radius

Scaled such that all features ∈ [0, 1]

Architecture



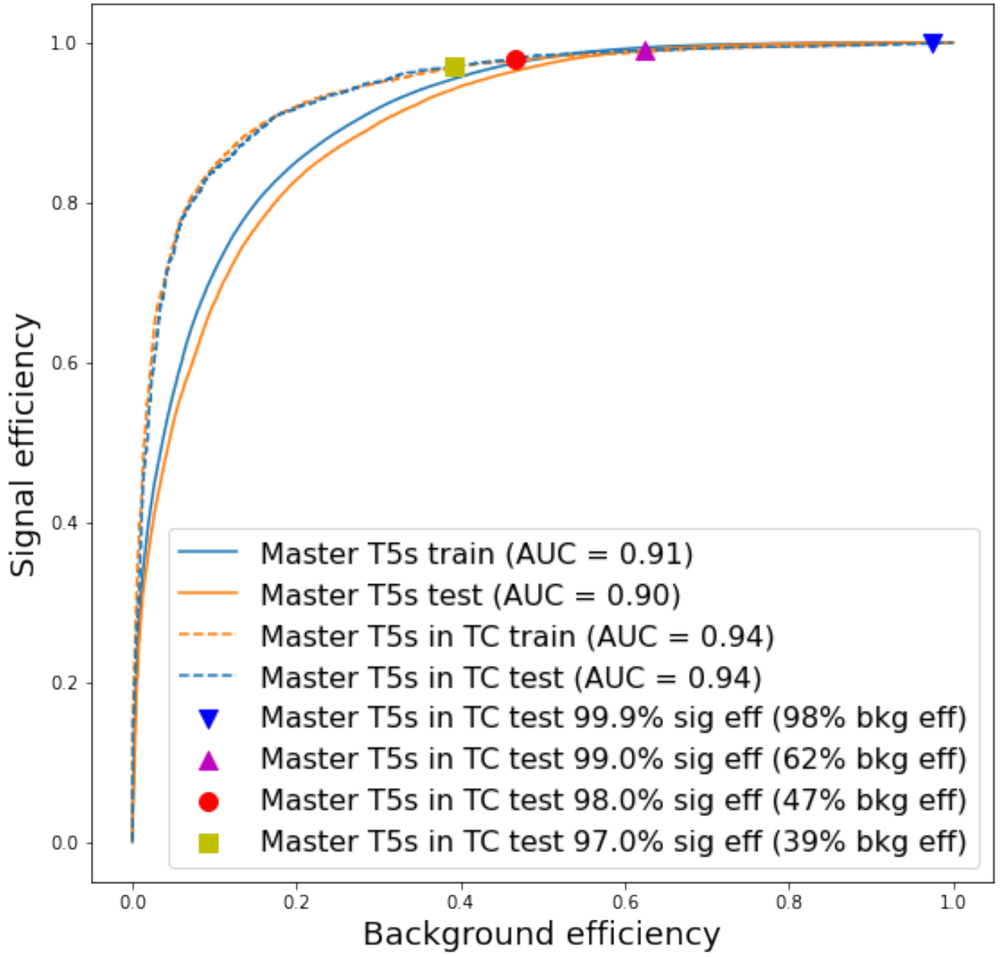


$\overrightarrow{x_i} \oplus \overrightarrow{e_{ij}} \oplus \overrightarrow{x_j}$

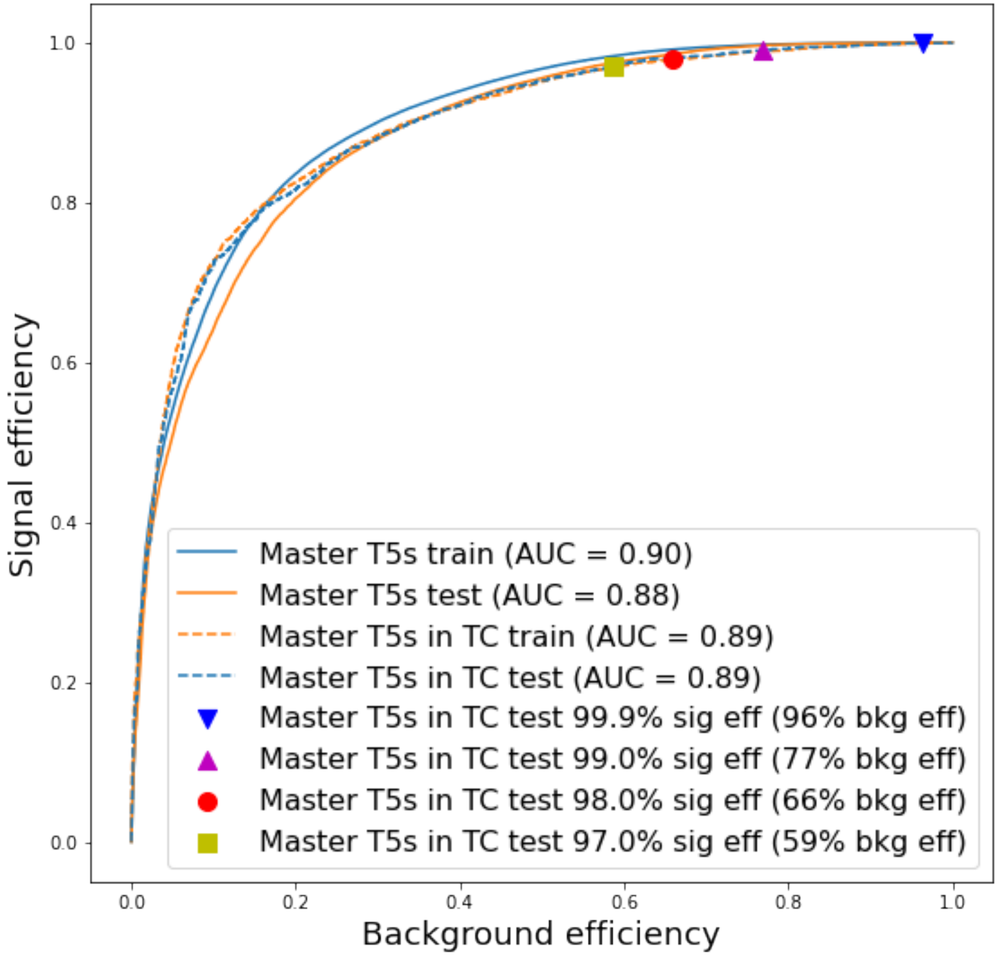
Trial 1: ~~GNN~~DNN Performance

Performance is comparable!

Previous normalization



Current normalization



Summary

- The previous GNN studies were performed with weirdly normalized inputs
  - Performance is somehow worse with “correctly” normalized inputs
  - Why? Investigation ongoing…
- Early NN studies show comparable performance to GNN for T5 classification
  - Fundamentally must be much faster than a GNN for inference
- Next steps:
  - Diver deeper into the normalization issue
  - Propagate T5 NN scores to output N-tuple and produce efficiency plots
  - Run T5 NN “Trial 2” (i.e. no χ^2 cuts)

Backup

Errata: Feature Normalization

```python
import torch

n_events = 5
n_features = 3
print(f"n events: {n_events}")
print(f"n features: {n_features}")
features1 = []
features2 = []
for i in range(n_features):
    feature = torch.rand(n_events)*100
    features1.append(feature.clone())
    feature -= feature.min()
    feature /= feature.max()
    features2.append(feature.clone())

features1 = torch.transpose(torch.stack(features1), 0, 1)
print(f"features1 before:\n{features1}")
print(f"features1 min:\n{features1.min(1, keepdim=True)[0]}")
features1 -= features1.min(1, keepdim=True)[0]
print(f"features1 - min:\n{features1}")
print(f"features1 max:\n{features1.max(1, keepdim=True)[0]}")
features1 /= features1.max(1, keepdim=True)[0]
print(f"features1 / max:\n{features1}")

features2 = torch.transpose(torch.stack(features2), 0, 1)
print(f"features2:\n{features2}")
```

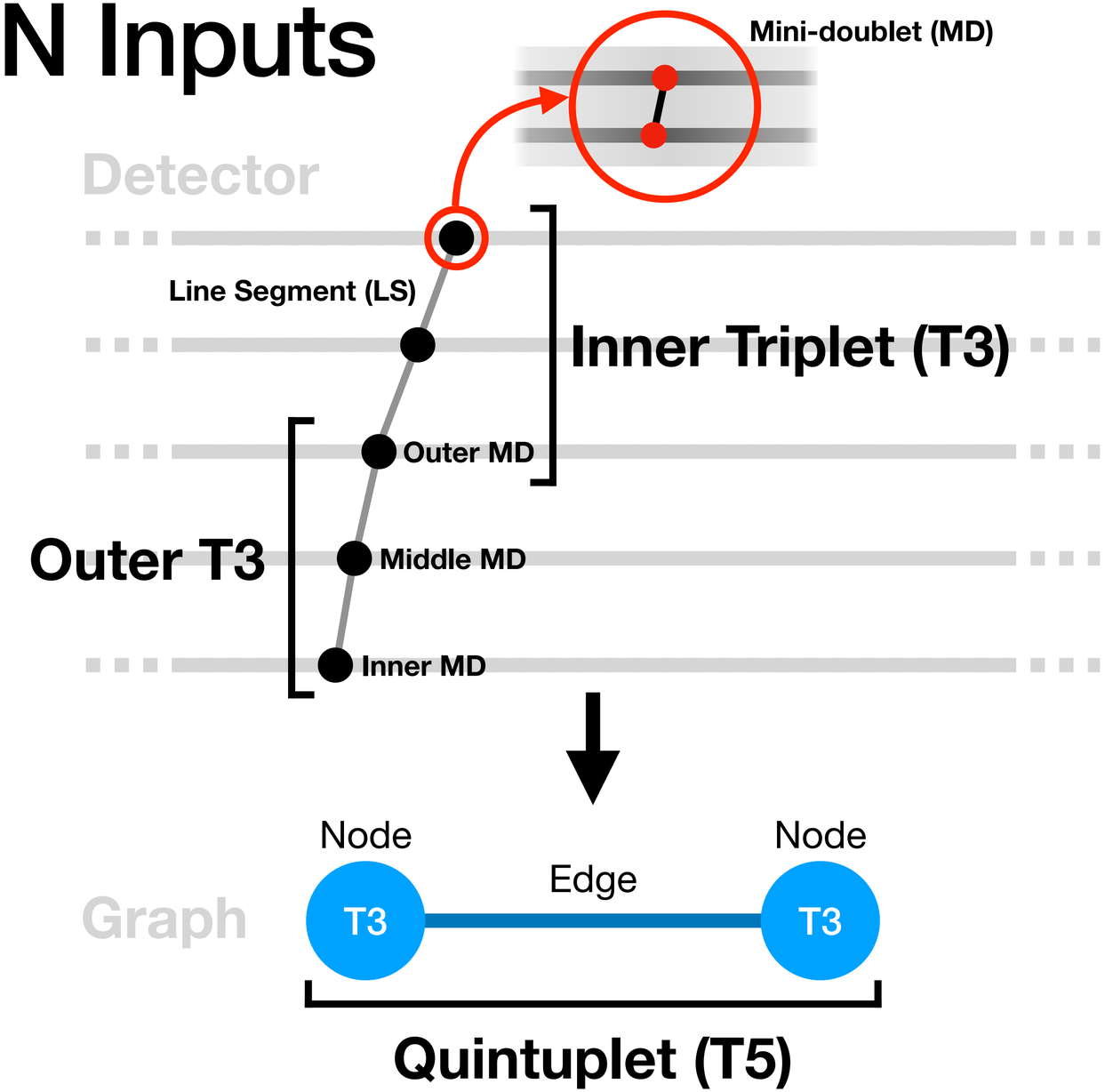
```
n events: 5
n features: 3
features1 before:
tensor([[54.8845, 25.2324, 41.6434],
        [44.0448, 40.9468, 53.0267],
        [ 7.2324, 89.2231, 73.3785],
        [33.2097, 81.7461, 35.6279],
        [97.5232, 90.1023, 29.6079]])
features1 min:
tensor([[25.2324],
        [40.9468],
        [ 7.2324],
        [33.2097],
        [29.6079]])
features1 - min:
tensor([[29.6522,  0.0000, 16.4110],
        [ 3.0979,  0.0000, 12.0799],
        [ 0.0000, 81.9907, 66.1460],
        [ 0.0000, 48.5364,  2.4182],
        [67.9153, 60.4944,  0.0000]])
features1 max:
tensor([[29.6522],
        [12.0799],
        [81.9907],
        [48.5364],
        [67.9153]])
features1 / max:
tensor([[1.0000, 0.0000, 0.5535],
        [0.2565, 0.0000, 1.0000],
        [0.0000, 1.0000, 0.8068],
        [0.0000, 1.0000, 0.0498],
        [1.0000, 0.8907, 0.0000]])
features2:
tensor([[0.5278, 0.0000, 0.2750],
        [0.4077, 0.2422, 0.5350],
        [0.0000, 0.9864, 1.0000],
        [0.2877, 0.8712, 0.1375],
        [1.0000, 1.0000, 0.0000]])
```

$$X = \begin{pmatrix} (x_1 & x_2 & x_3 & \dots & x_{N_f})_1 \\ (x_1 & x_2 & x_3 & \dots & x_{N_f})_2 \\ (x_1 & x_2 & x_3 & \dots & x_{N_f})_3 \\ & & \vdots & & \\ (x_1 & x_2 & x_3 & \dots & x_{N_f})_{N_e} \end{pmatrix}$$

GNN/DNN Inputs

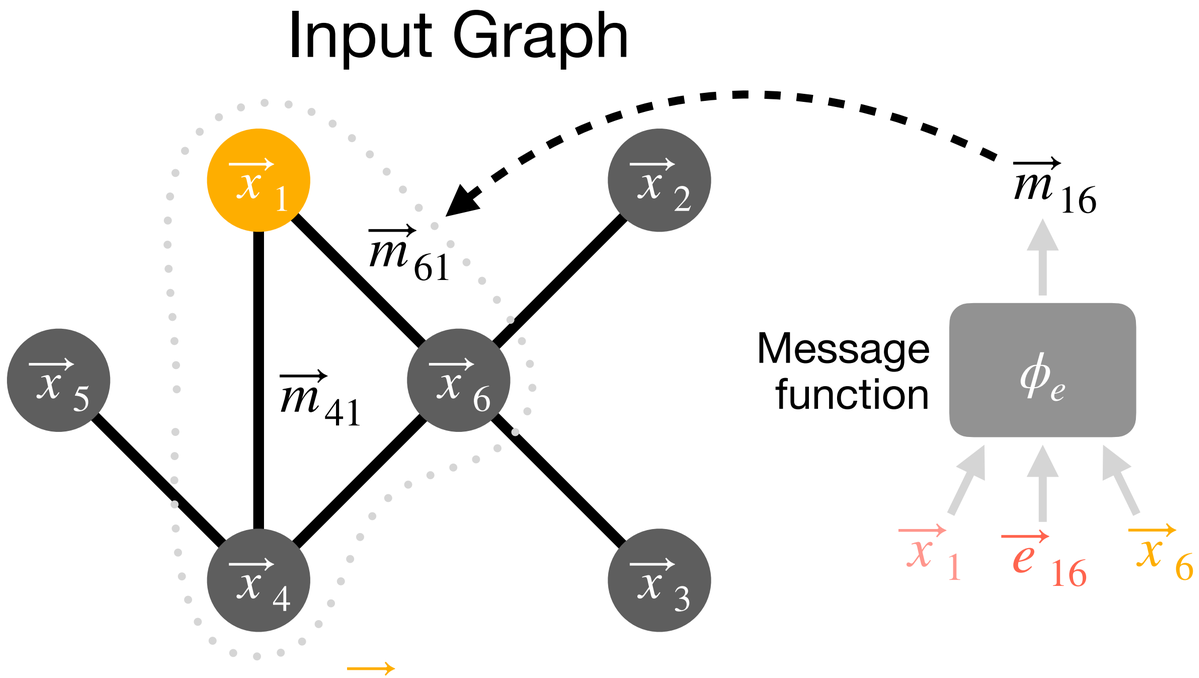
Object	Feature
Node (T3)	p_T
	Inner anchor hit r, z, ϕ, η, δr, layer
	Middle anchor hit r, z, ϕ, η, δr, layer
	Outer anchor hit r, z, ϕ, η, δr, layer
Edge (T5)	p_T, η, ϕ
	χ^2
	Inner T3 radius
	Bridge T3 radius
	Outer T3 radius

Scaled such that all features ∈ [0, 1]

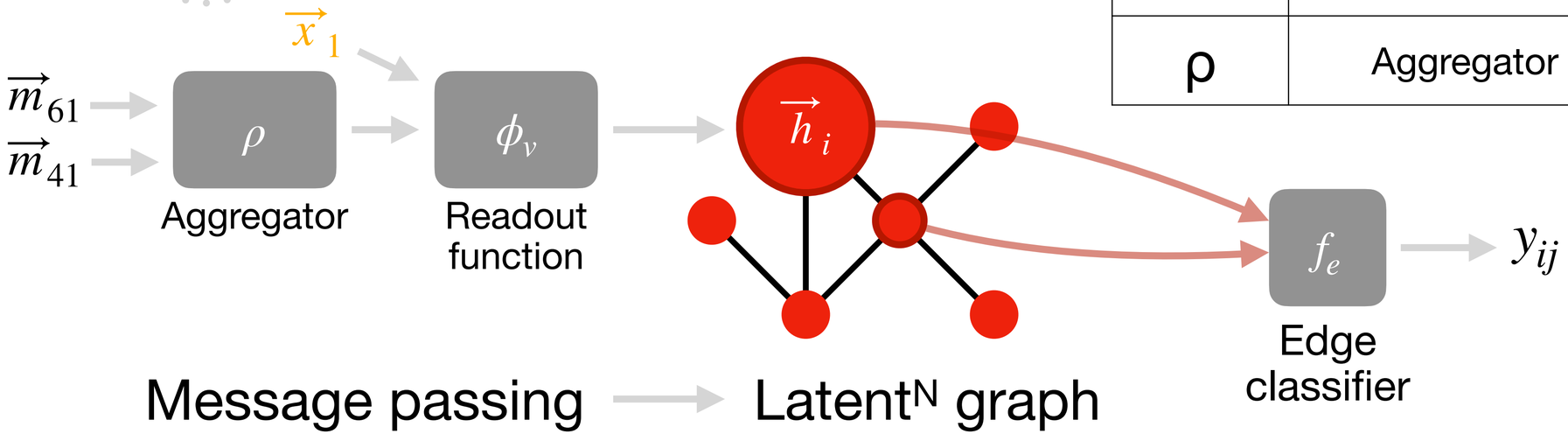


UC San Diego

GNN Configuration

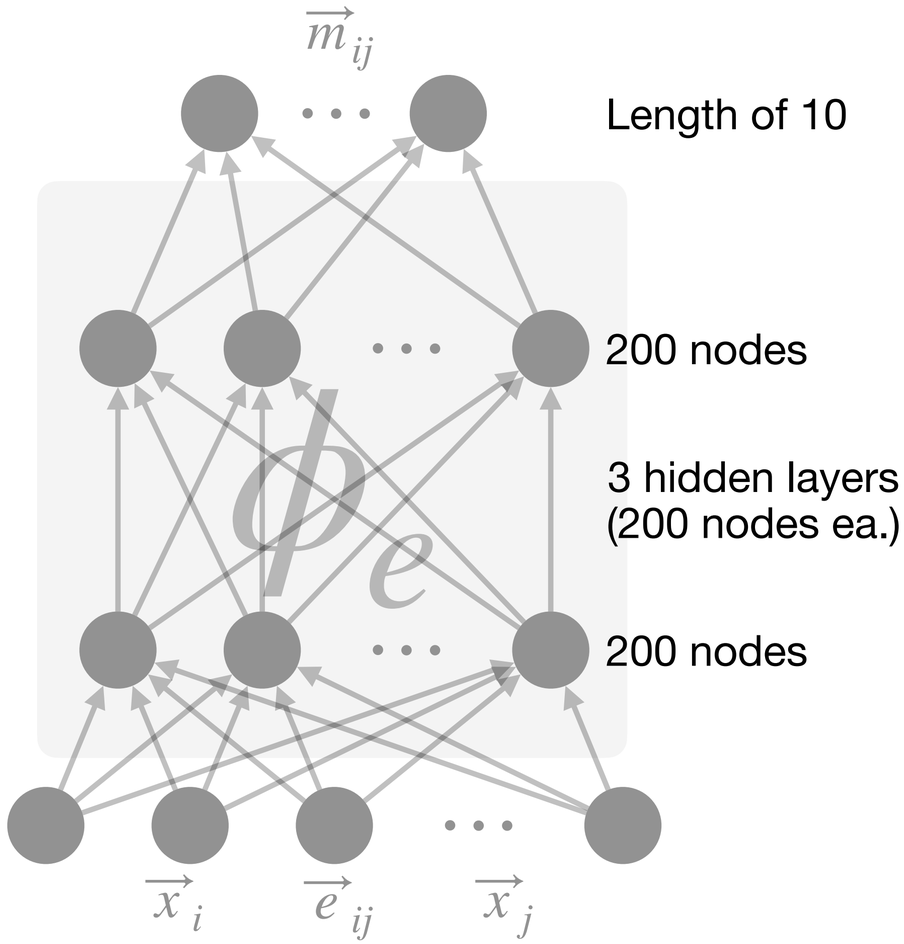


Symbol	Name	Description
ϕ_e	Message function	• Neural network • 4 hidden layers • 64 nodes per layer
ϕ_v	Readout function	• Neural network • 4 hidden layers • 64 nodes per layer
f_e	Edge classifier	• Neural network • 4 hidden layers • 64 nodes per layer
ρ	Aggregator	Sum

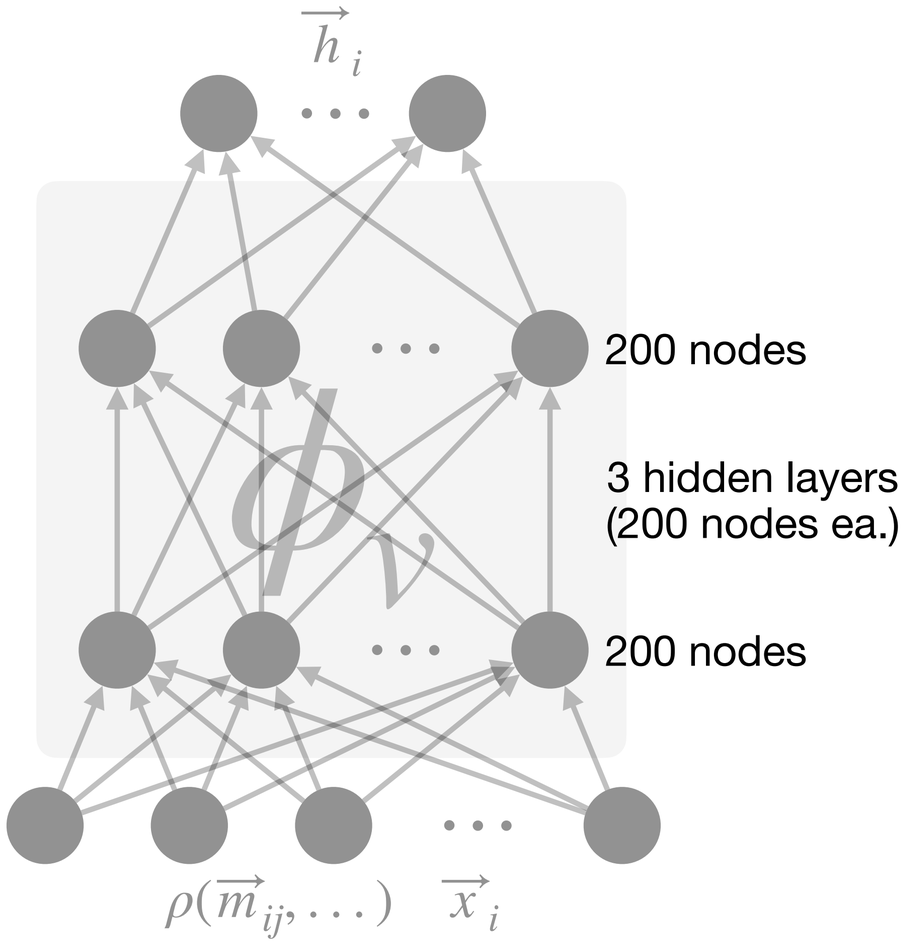


UC San Diego

GNN Internal NN Configurations



Message function



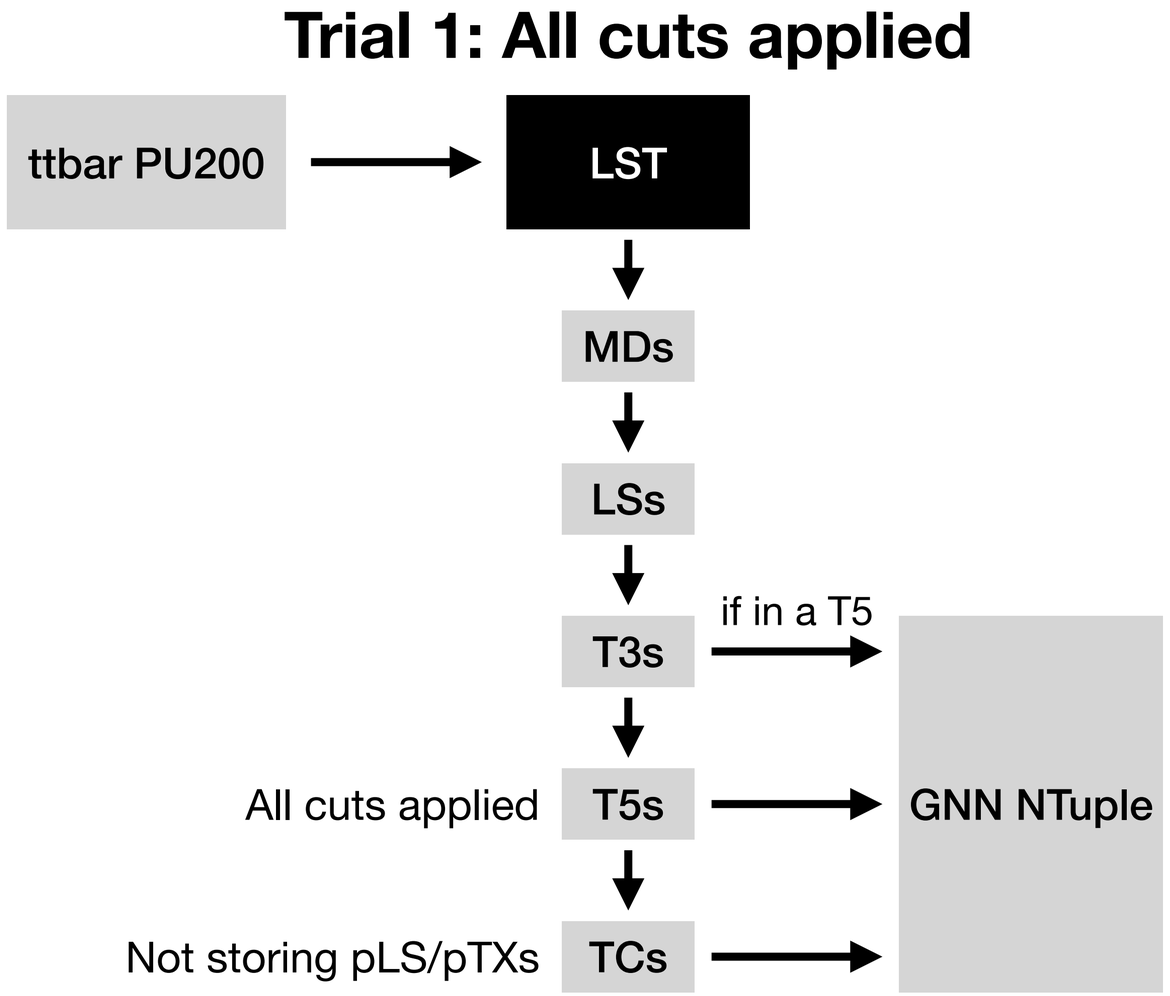
Readout function

UC San Diego

Trial 1: GNN NTuple

- Training on T5s/T3s without duplicate removal
- pLS/pTX objects are excluded from the TCs
  - i.e. only T5s in TCs
  - Duplicate removal is applied here
- **All cuts applied in T5 selection algo.**
- Target question:

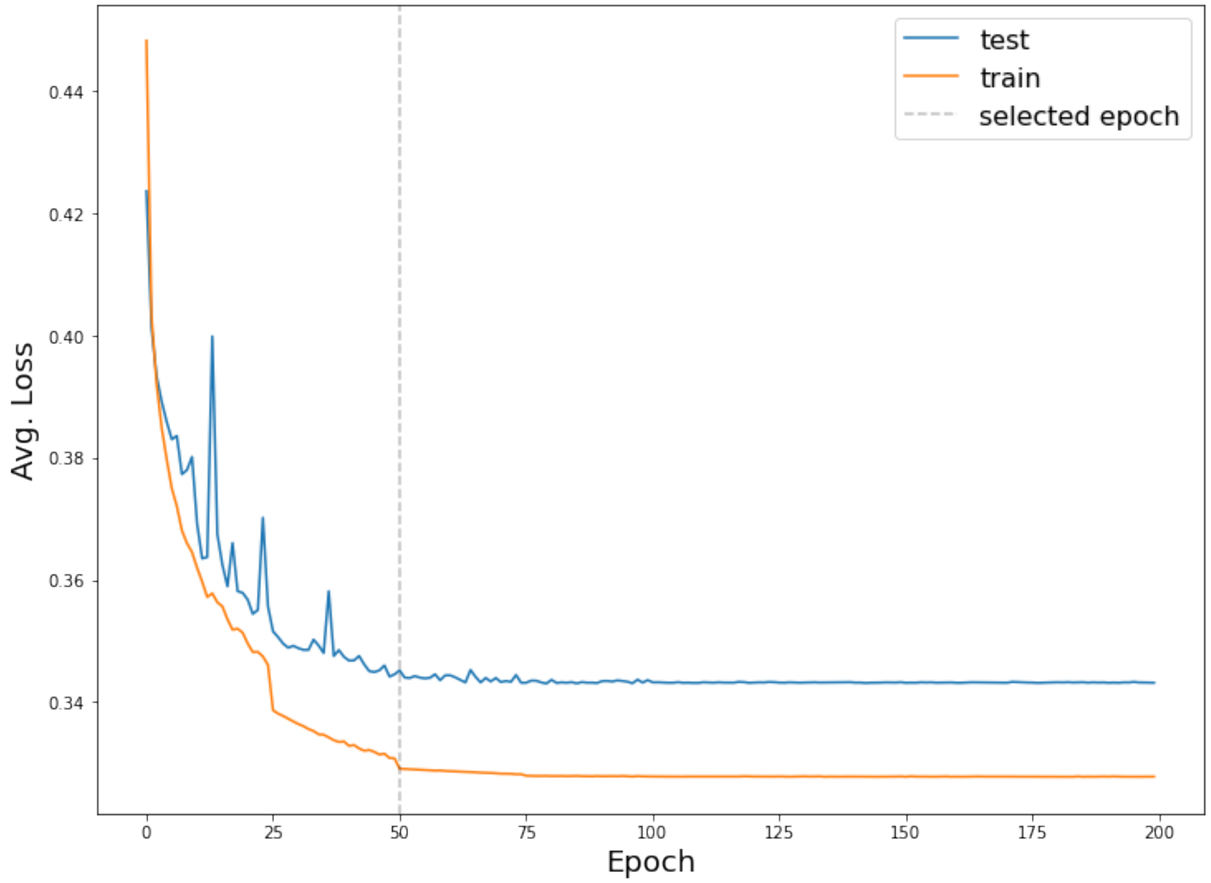
Can the GNN give us anything for free?

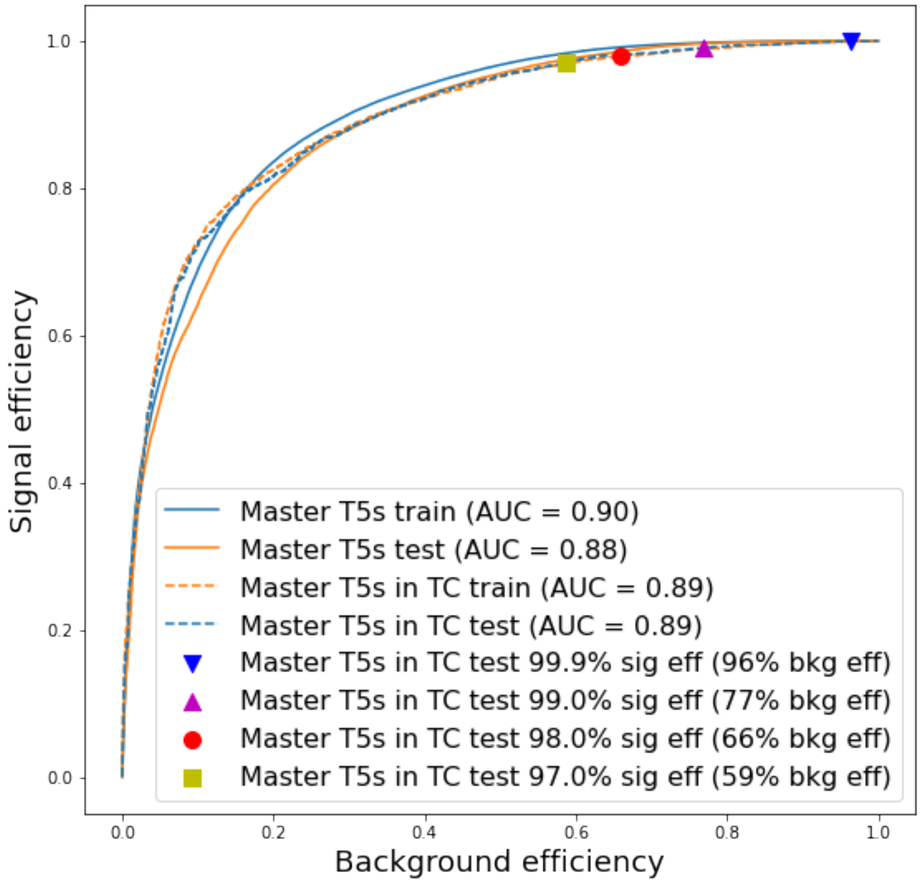


UC San Diego

Trial 1: ~~GNN~~DNN Performance

Showing inference ROC curves before/after duplicate removal (DR)





UC San Diego