The Smart Background Project

Making simulation more efficient with ML

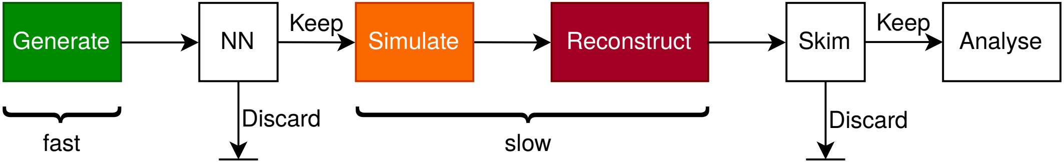
Nikolai Krug

LMU Munich

30 October 2024, LMU Joint Particle Physics Group Seminar

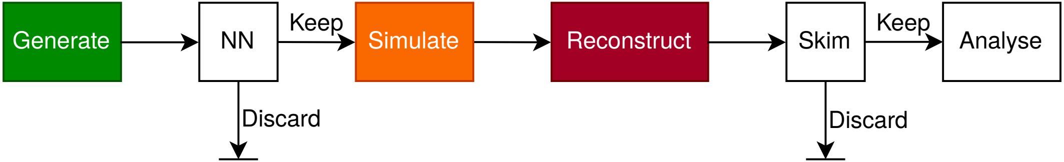
Selective/Smart background MC simulation

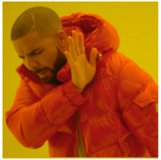
Introduced by James Kahn in his PhD thesis (2019):



- Event generation much faster than detector simulation/reconstruction (at Belle II)
  $\rightarrow$ O(10ms) vs O(1s) $\rightarrow t_{\text{fast}} : t_{\text{slow}} \approx 1 : 100$
- Many events discarded by filter (skim)
  $\rightarrow$ try to predict which events will be discarded, already after event generation
- Not always a clearly correlated variable on generator level
  $\rightarrow$ example: skim may use involved algorithms like FEI (Full Event Interpretation)
  $\rightarrow$ train an NN to be a good filter

The problem with naive filtering





False negatives

False positives

- **false positives** are not too problematic
  (we throw them away later by running the “true” skim)
- **false negatives** may produce bias (we can’t get them back)

The solution: Importance sampling

Boyang Yu's Master thesis (2021)

- Use NN output as **probability to keep event**
- Weight events by inverse probability
- No bias by construction, every event has a chance to be picked
- Train NN to provide **highest speedup** $\frac{t_{\mathrm{noNN}}}{t_{\mathrm{NN}}}$
  to produce same **effective sample size** $\frac{(\sum_i w_i)^2}{\sum_i w_i^2}$ after skimming
  $\rightarrow$ for large enough sample independent of sample size
- Speedup also depends on:
  - **assumed times** for generation (fast), NN inference, simulation/reconstruction (slow)
    $\rightarrow$ roughly expect $t_{\mathrm{fast}} : t_{\mathrm{NN}} : t_{\mathrm{slow}} = 1 : 1 : 100$
  - (target) **filter efficiency** (= retention rate)
    $\rightarrow$ can gain more if more events expected to be skipped
- Conceptionally similar to slicing strategy for MC filters at LHC
  $\rightarrow$ slicing is essentially importance sampling with discrete probabilities

Slicing

Who has seen samples like this?

```
mc15_13TeV.361324.Sherpa_CT10_Wmunu_Pt0_70_CVetoBVeto...
mc15_13TeV.361325.Sherpa_CT10_Wmunu_Pt0_70_CFilterBVeto...
mc15_13TeV.361326.Sherpa_CT10_Wmunu_Pt0_70_BFilter...
mc15_13TeV.361327.Sherpa_CT10_Wmunu_Pt70_140_CVetoBVeto...
mc15_13TeV.361328.Sherpa_CT10_Wmunu_Pt70_140_CFilterBVeto...
mc15_13TeV.361329.Sherpa_CT10_Wmunu_Pt70_140_BFilter...
mc15_13TeV.361330.Sherpa_CT10_Wmunu_Pt140_280_CVetoBVeto...
mc15_13TeV.361331.Sherpa_CT10_Wmunu_Pt140_280_CFilterBVeto...
mc15_13TeV.361332.Sherpa_CT10_Wmunu_Pt140_280_BFilter...
mc15_13TeV.361333.Sherpa_CT10_Wmunu_Pt280_500_CVetoBVeto...
mc15_13TeV.361334.Sherpa_CT10_Wmunu_Pt280_500_CFilterBVeto...
mc15_13TeV.361335.Sherpa_CT10_Wmunu_Pt280_500_BFilter...
mc15_13TeV.361336.Sherpa_CT10_Wmunu_Pt500_700_CVetoBVeto...
mc15_13TeV.361337.Sherpa_CT10_Wmunu_Pt500_700_CFilterBVeto...
mc15_13TeV.361338.Sherpa_CT10_Wmunu_Pt500_700_BFilter...
```

...

- Every slice is weighted by $w = \frac{N_{\text{gen,slice}}}{N_{\text{tot,exp}}}$, with $N_{\text{tot,exp}} = \epsilon_{\text{slice}} \sigma_{\text{process}} \int L \mathrm{d}t$
- Can also view this as sampling with probabilities: $p_{\text{slice}} = \frac{N_{\text{gen,slice}}}{N_{\text{tot,exp}}}$
- Note: $N_{\text{gen,slice}}$ may already be represented as a sum of weights

Effective sample size

- With weighted events our expected counts are estimated by $N_{\text{exp}} = \sum_i w_i$
- The variance of N_{exp} is then estimated by $\sigma^2_{N_{\text{exp}}} = \sum_i w_i^2$
  $\rightarrow$ that's what `.Sumw2()` is for in ROOT histograms
  $\rightarrow$ think: every event is a poisson count of 1, scaled by w_i $\rightarrow$ variance w_i^2
- What's the unweighted sample size that gives the same relative statistical uncertainty σ_{rel} as some weighted sample?

$$\underbrace{\frac{\sqrt{N}}{N} = \frac{1}{\sqrt{N}}}_{\sigma_{\text{rel,unweighted}}} \stackrel{!}{=} \underbrace{\frac{\sqrt{\sum_i w_i^2}}{\sum_i wi}}_{\sigma_{\text{rel,weighted}}} \rightarrow N = \frac{(\sum w_i)^2}{\sum w_i^2}$$

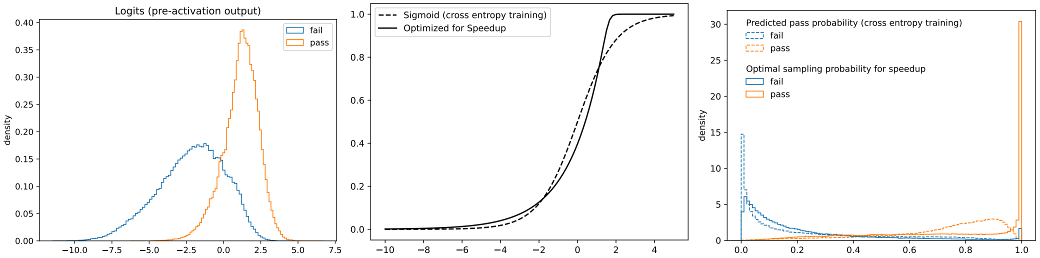
Speedup formula

	NN select	NN reject
Actual pass	TP	FN
Actual fail	FP	TN

$$\text{Speedup} = \frac{t_{\text{noNN}}}{t_{\text{NN}}} = \frac{t_3 \cdot a_{\text{eff}}}{(\epsilon \cdot f_{\text{TP}} + (1-\epsilon) \cdot f_{\text{FP}}) \cdot t_1 + ((1-\epsilon) \cdot f_{\text{TN}} + \epsilon \cdot f_{\text{FN}}) \cdot t_2}$$

- ϵ: Skim efficiency
- $f_{\text{TP}} = E[p_{\text{pass}}]$, $f_{\text{FP}} = E[p_{\text{fail}}]$, $f_{\text{TN}} = 1 - f_{\text{FP}}$, $f_{\text{FN}} = 1 - f_{\text{TP}}$
- $t_1 = t_{\text{fast}} + t_{\text{NN}} + t_{\text{slow}}$
- $t_2 = t_{\text{fast}} + t_{\text{NN}}$
- $t_3 = t_{\text{fast}} + t_{\text{slow}}$
- $a_{\text{eff}} = 1/E[w_{\text{pass}}]$ with $w_{\text{pass}} = 1/p_{\text{pass}}$

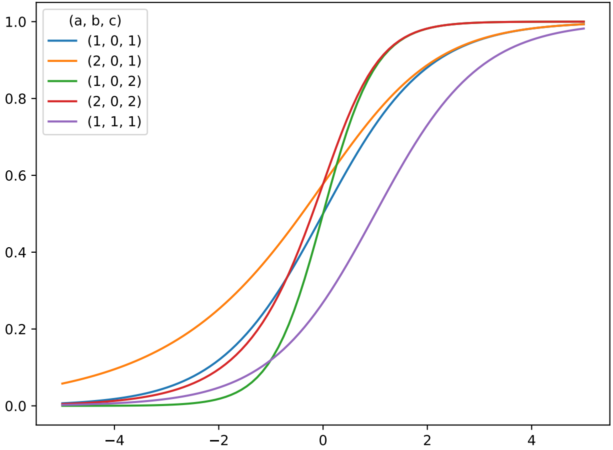
Sampling probability calibration - optimize speedup



- My claim: prediction from optimally trained (probabilistic) classifier should be related to optimal sampling probability by a monotonic transformation
  → higher probabilities to pass filter should come with higher sampling probabilities
- Could see this as using a "skewed" sigmoid activation on logits
  → can optimize this with 3 parameters for skewed sigmoid
- Seems no unique solution, some freedom of choice:
  → can reduce to 2 parameter function (see next slides)

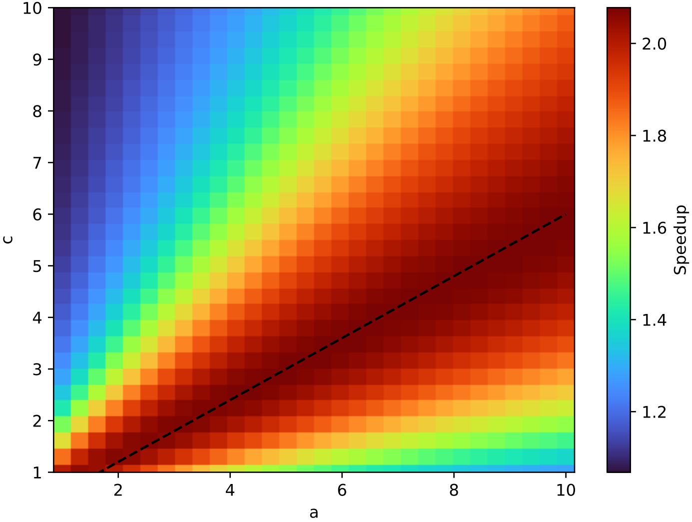
The 3 Parameters

$$f(x) = \frac{1}{\left(1 + ae^{-c(x-b)}\right)^{\frac{1}{a}}}$$



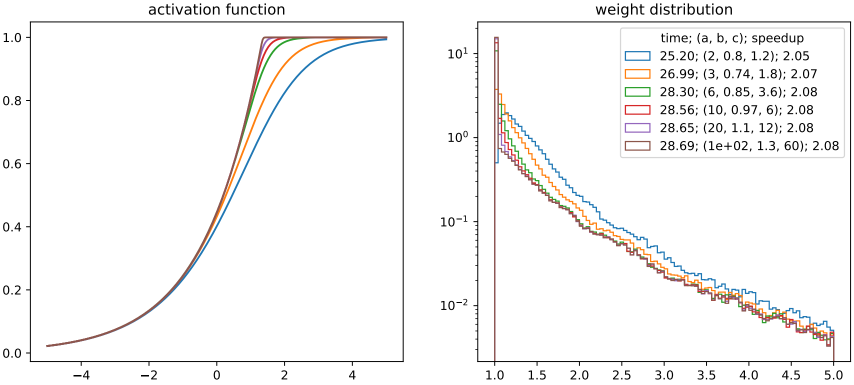
Not a unique maximum

Fix a and c, fit b:



Tradeoff

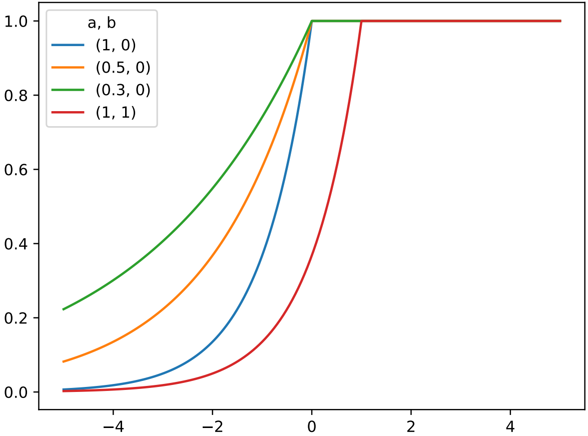
Can choose between shorter time (larger sample in same time), **or** narrower weight distribution:



$\rightarrow$ narrower weight distribution is desirable

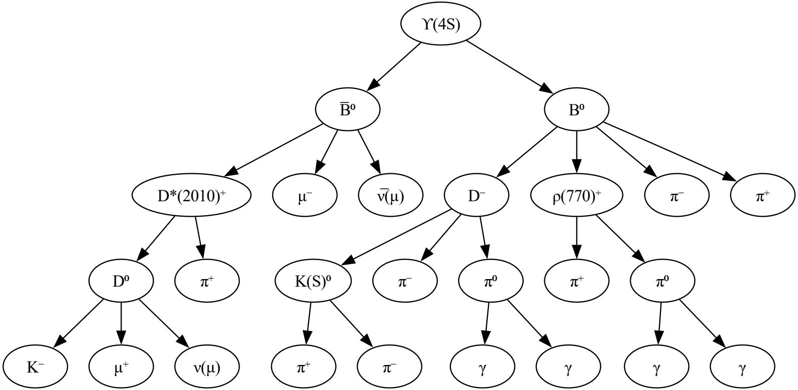
So this function is enough

$$f(x) = \min(e^{a(x-b)}, 1)$$



$\rightarrow$ parameters are a scalefactor a and a bias b

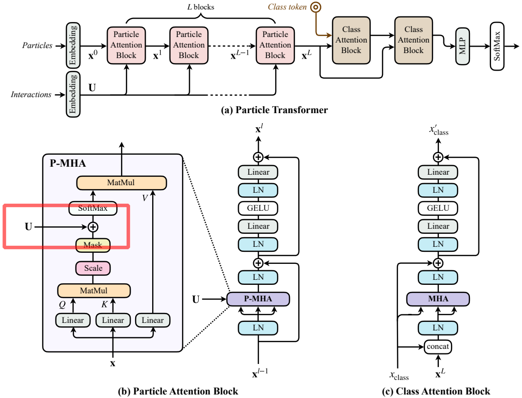
The dataset



- Using generator level MC record
  $\rightarrow$ hadron-level, no quarks/gluons
- List of particles with mother-daughter relations
- Particle features: PDG id, 4-momentum, production vertex position/time

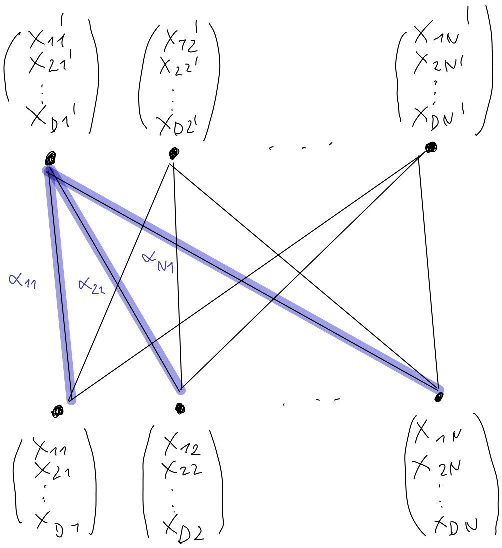
The model

based on *Particle Transformer for Jet Tagging* arXiv:2202.03772



- ParT achieved state-of-the-art performance in jet tagging by pre-training on their own **large dataset** (100M) + **fine tuning** (e.g top tagging)
- Architecture seems very generic
  → **essentially just a transformer**
  ("Attention is all you need (2017)")
- Supports edge features, in our case:
  - adjacency matrix of decay graph
    (had success using GNN architectures before)
  - angle between pairs
  - invariant masses between pairs
- For new skims/filters hope to be able to finetune pre-trained model
  → especially interesting for **low filter efficiencies**
- 10-layer, 2M parameter model

Self Attention



$$\underbrace{\begin{aligned} Q &= W_Q X \\ K &= W_K X \\ V &= W_V X \end{aligned}}_{\text{matrices}}$$

Learnable Parameters

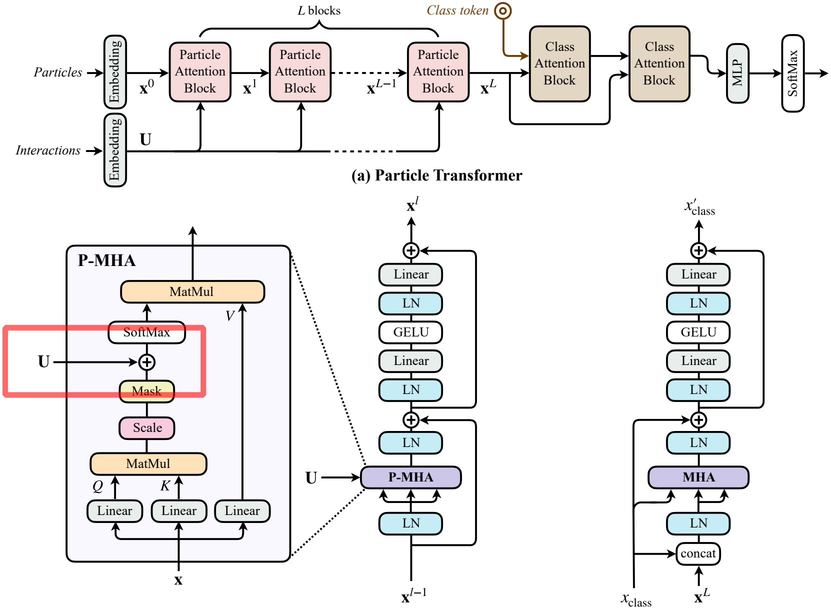
$$\boxed{X' = \text{softmax}\left(\frac{\overset{Q}{Q} \times \overset{K^T}{K^T}}{\sqrt{d_k}}\right)\overset{V}{V}}$$

$d_k = D$

e.g. $\vec{x}_1' = \vec{\alpha}\ \vec{V}_1$

$$e_{ij} = \vec{Q}_i \cdot \vec{K}_j$$

$$\alpha_{11} = \underset{k}{\text{softmax}}\left(e_{k1}\right)$$

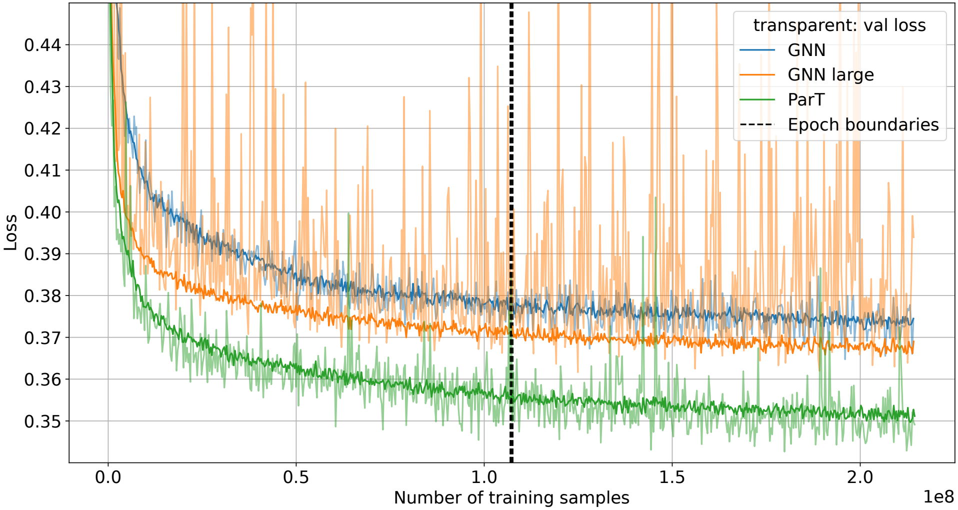


(a) Particle Transformer

(b) Particle Attention Block

(c) Class Attention Block

GNN vs ParT



$\rightarrow$ transformer model (green) better than GNN (blue, orange) for large dataset

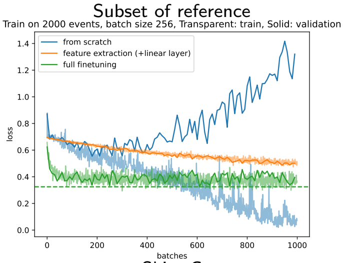
How to do transfer learning

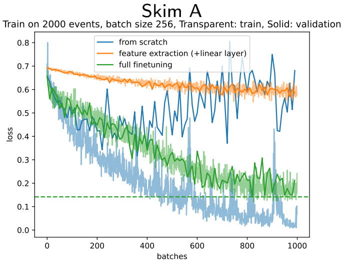
Looking at two methods:

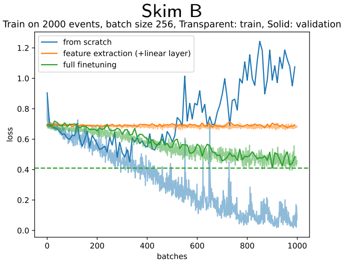
- **Feature extraction:** Remove final layer, only retrain that
  $\rightarrow$ simplest case: just retrain a single neuron
  $\rightarrow$ will likely only work if new skim highly correlated with something seen during training
- **Whole-model fine-tuning:** start with pre-trained model but adjust all parameters
  $\rightarrow$ also reinitialize last layer in case of different output
  $\rightarrow$ hyperparameters from ParT paper:
  - learning rate 0.0001/0.005 for pre-trained/last-layer parameters
  - weight decay 0.01

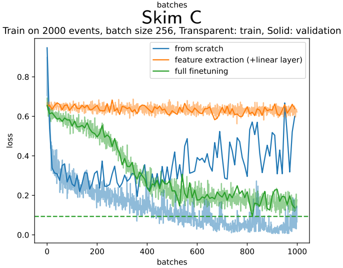
Fine tuning tests

fine tune model pre-trained on a reference skim





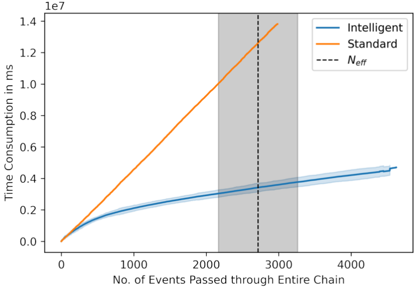




- As expected feature extraction bad if low linear correlation
- Train from scratch does typically not work well with only 2k (1k pass) events
- Full finetune shows promising results!

Adaptive/Reinforcement learning

Simulated reinforcement learning[1] for skim C



- For running on new skims could consider "reinforcement learning":
  → Train model while producing data and running skim
  → Model becomes successively better producing data more efficiently
- Advantage: Overall time saving, on-the-fly procedure in one step
- Disadvantage: need to implement training loop in production software

[1] from Daniel Pollmann's Bachelor thesis (2024)

Large scale training

- Many pre-defined skims with data available
  $\rightarrow$ many different definitions centrally run on large datasets
  $\rightarrow$ pre-train model on large datase that predicts probabilities for **all skims**
  $\rightarrow$ data with **51 different labels**

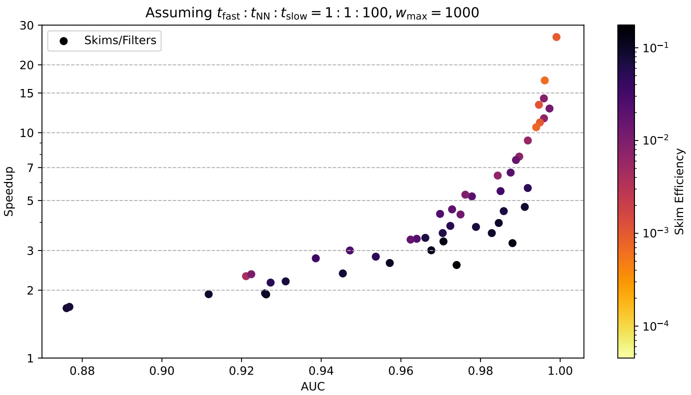
- Also **condition on background type**
  $\rightarrow$ 7 Generic samples: $B^{\pm,0}$ pairs, $q\bar{q}$ with 4 different q flavours, $\tau\bar{\tau}$
  (representative of $e^{+}e^{-}$ collisions at 10.58 GeV)

- Using dataset with $\approx$ 180M events (10% kept for testing),
  roughly balanced between all 7 generic samples
  $\rightarrow$ corresponds to roughly $20\,\mathrm{fb}^{-1}$ of simulated data

- No class weighting, just take labels as they come
  $\rightarrow$ partially overlapping $\rightarrow$ binary cross entropy term for each

- Hope: Diverse training dataset makes finetuning more flexible

Training results



- Training worked with similar setup as for fewer labels
- Achievable **speedups** for different skims **correlate with**
  - **Separation power** (AUC, area under ROC curve)
    $\rightarrow$ higher separation leads to higher speedup
  - **Skim efficiency**
    $\rightarrow$ lower skim efficiency tends to higher speedup

Summary and Conclusions

- We'd like to speedup our simulation with **NN assisted filters**
  $\rightarrow$ filter events that won't pass downstream selection before running expensive parts (detector simulation and reconstruction)
- Using **importance sampling** technique to avoid bias
  $\rightarrow$ continuous version of traditional "slicing" strategy
- Metric to optimize: **speedup** when producing same effective sample size
  $\rightarrow$ can be calibrated with parameterized logistic function
- Use **Transformer model** to generically capture MC generator information
- **Transfer learning** to capture skims with little training data seems promising
  $\rightarrow$ fine tuning seems to work also for skim selections not seen during training
  $\rightarrow$ may also help to **avoid retraining** when conditions/calibrations/selections change
  $\rightarrow$ also offers prospects for on-the-fly **reinforcement learning**
- Can run the pretraining on a **large dataset with many different skims**
  $\rightarrow$ maximize diversity of skim selections and inputs seen by the model

Backup

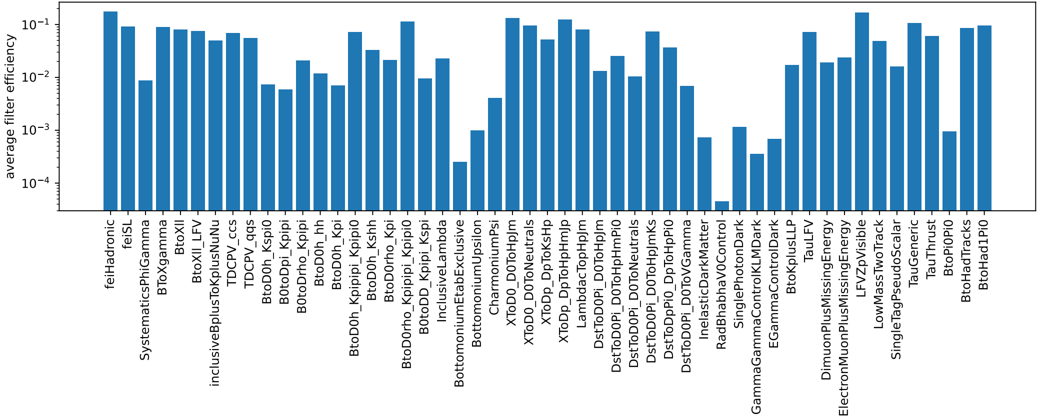
Hyperparameters

- Largely following architecture from ParT paper:
  - 8 Transformer blocks with self-attention
  - 2 Transformer blocks with class-attention
  - 8 Attention heads in each multi-head attention block
  - Embedding size 128
  - MLP hidden layers have 4 times the embedding size

  $\rightarrow$ around 2 Million parameters
- Modifications/Additions:
  - Fewer norm layers (Pre-LN transformer vs Normformer in ParT)
  - Embedding layer for PDG ID
  - Embedding layer for sample type
  - 3 pair features (Decay tree adjacency matrix, invariant masses, angle between pairs)

Filter efficiencies

Average over all samples where skim is available:



Correlations between pass events:
(only considering events where both skims are available)