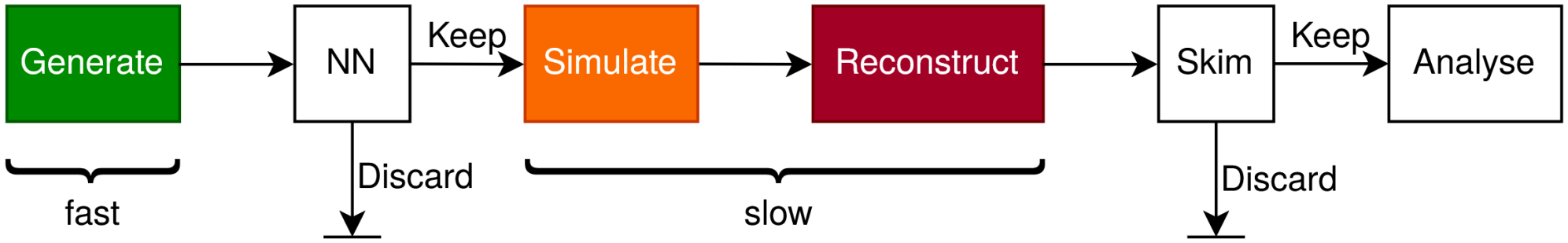
Transfer Learning for Smart Background Simulation at Belle II

David Giesegh, Nikolai Krug, Boyang Yu, Thomas Kuhr
LMU Munich

12.03.2025
KISS annual meeting

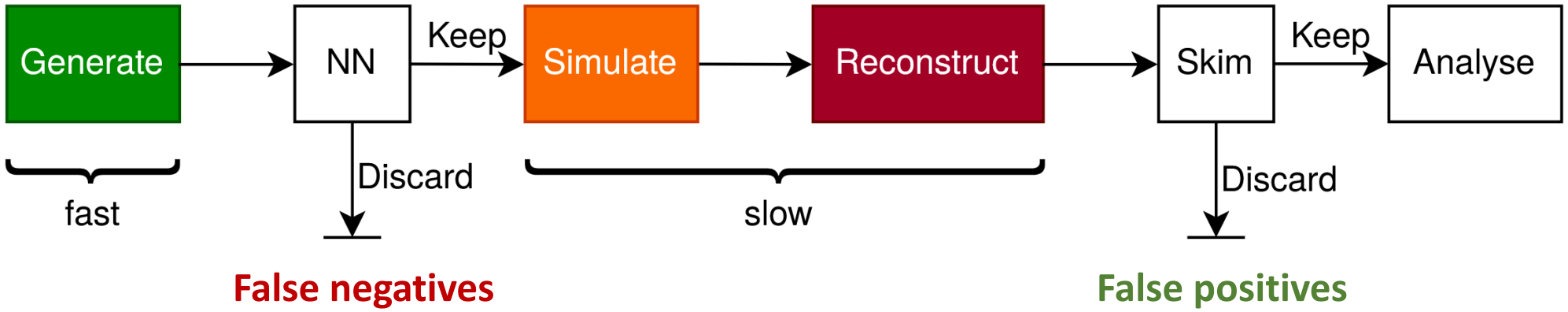
Smart Background Simulation

(Introduced in PhD Thesis by James Kahn (2019))



- **Skims** are centrally produced loose selections tailored to specific analysis types
  → Many skims throw away large fractions of costly simulated background events
- **Event generation** much faster than **detector simulation** and **event reconstruction** ($t_{slow}/t_{fast} \sim 100$)
- Typically, **no simple correlations** between features after generation and final selection criteria
- Perhaps a **neural network based classifier** can solve this issue?

Filtering procedure



- **False positives**: unproblematic (simply lower performance)
- **False negatives**: problematic!
  Discarded events are gone and may thus **bias distributions**
- Solution: **importance sampling**

Importance sampling

(Introduced in Master's Thesis by Boyang Yu (2021))

Idea:

- use NN output as **sampling probability p_i**
- **weight** each event by its **inverse probability ω_i**

→ **No bias by construction** (Disadvantage: weighted events with arbitrarily high weights)

	Probability
Event 1	0.75
Event 2	0.33
Event 3	0.01
Event 4	0.06
Event 5	0.50

	Weight
Event 1	1.33
Event 2	3.00
Event 3	X
Event 4	16.67
Event 5	X

Evaluation metric: speedup

- Goal: Train NN to achieve highest **speedup** (t_{noNN}/t_{NN}) for the same effective sample size
- **Effective sample size:**
  - := size of an unweighted sample that would yield the same **relative statistical uncertainty**
  - Estimate for number of expected events: $\mathrm{N_{exp}} = \sum_i \omega_i$
  - Estimate for variance: $\sigma^2 = \sum_i \omega_i^2$
  - Effective sample size: $\frac{\sqrt{N_{eff}}}{N_{eff}} = \frac{\sqrt{\sum_i \omega_i^2}}{\sum_i \omega_i} \Rightarrow N_{eff} = \frac{\left(\sum_i \omega_i\right)^2}{\sum_i \omega_i^2}$

Evaluation metric: speedup

- Goal: Train NN to achieve highest **speedup** (t_{noNN}/t_{NN}) for the same effective sample size
- Speedup also depends on:
  - **Assumed times** for generation t_{fast}, NN inference t_{NN} and simulation/reconstruction t_{slow} (roughly $t_{fast} : t_{NN} : t_{slow} \sim 1 : 1 : 100$)
  - **Skim efficiency** (retention rate)
    → lower retention increases potential speedup

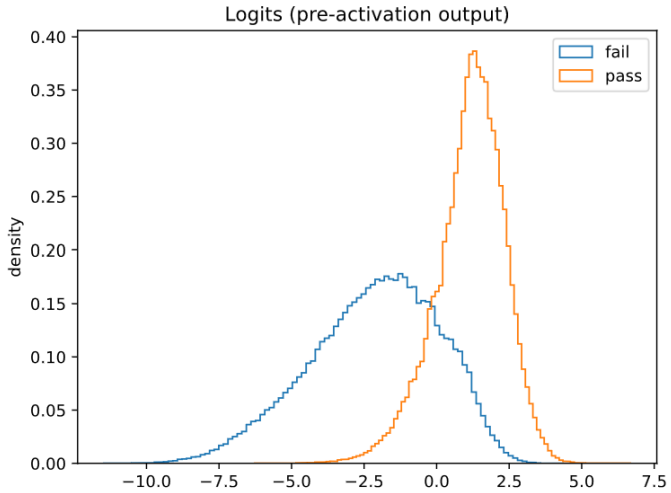
Evaluation metric: speedup

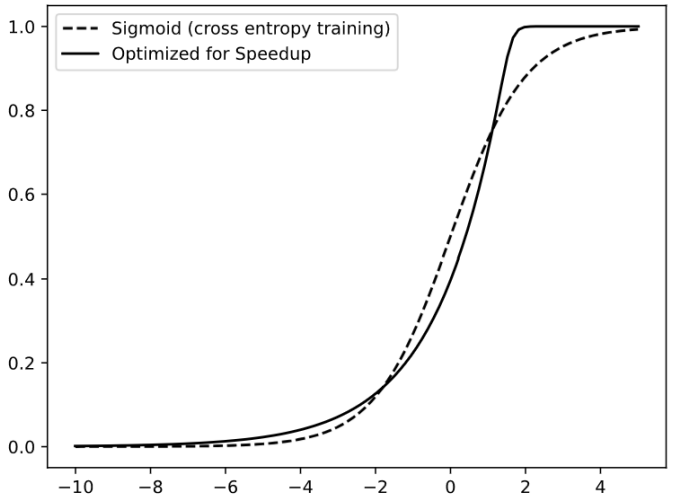
- Speedup $= t_{noNN}/t_{NN}$
- Define:
  - $t_1 \coloneqq t_{fast} + t_{NN} + t_{slow}$ (events accepted by NN)
  - $t_2 \coloneqq t_{fast} + t_{NN}$ (events rejected by NN)
  - $t_3 \coloneqq t_{fast} + t_{slow}$ (events with no NN)
- Use:
  - $t_{noNN} = t_3 \cdot N_{eff}$
  - $t_{NN} = t_1 \cdot (N_{TP} + N_{FP}) + t_2 \cdot (N_{FN} + N_{TN})$
- Some algebra later…

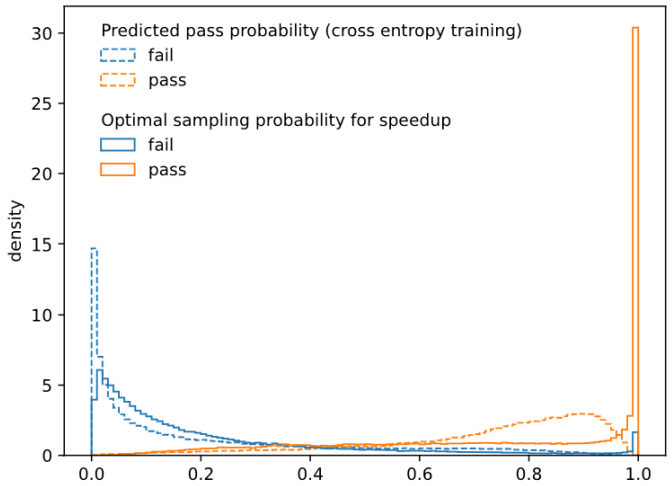
	Pass skim	Fail skim
NN accepts	TP	FP
NN rejects	FN	TN

$$\text{Speedup} = \frac{t_3/E[\omega_i]}{(\varepsilon\, f_{TP} + (1-\varepsilon)\, f_{FP})\, t_1 + (\varepsilon\, f_{FN} + (1-\varepsilon)\, f_{TN})\, t_2}$$

Optimizing for speedup







- Model trained with **cross entropy loss** and final sigmoid activation
- Idea: optimal **classification probability** for cross entropy should be related to optimal **sampling probability** for speedup by **monotonic function**

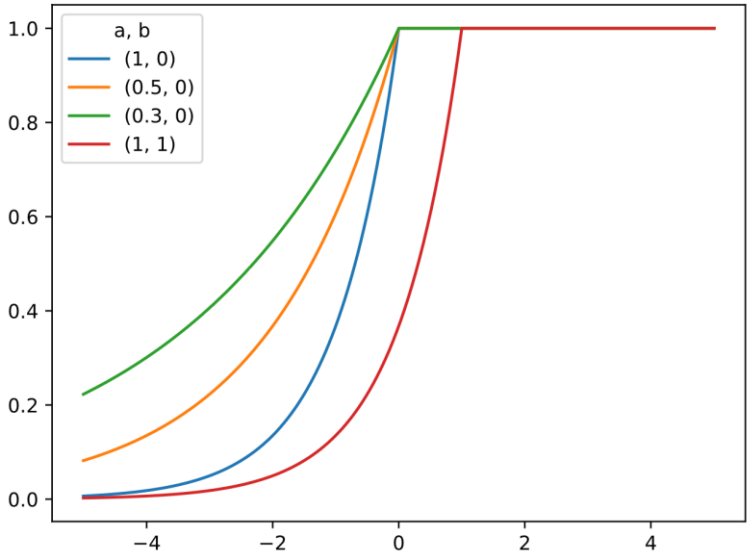
→ **Replace sigmoid activation by fitted “skewed” sigmoid optimized for speedup**

Optimizing for speedup

- Turns out generic skewed sigmoid $\left(1 + ae^{-c(x-b)}\right)^{-1/a}$ has too many parameters (no unique maximum)
- Tradeoff: shorter time ↔ narrower weight distribution
- Narrow weight distribution desirable → empirically leads to the following function:

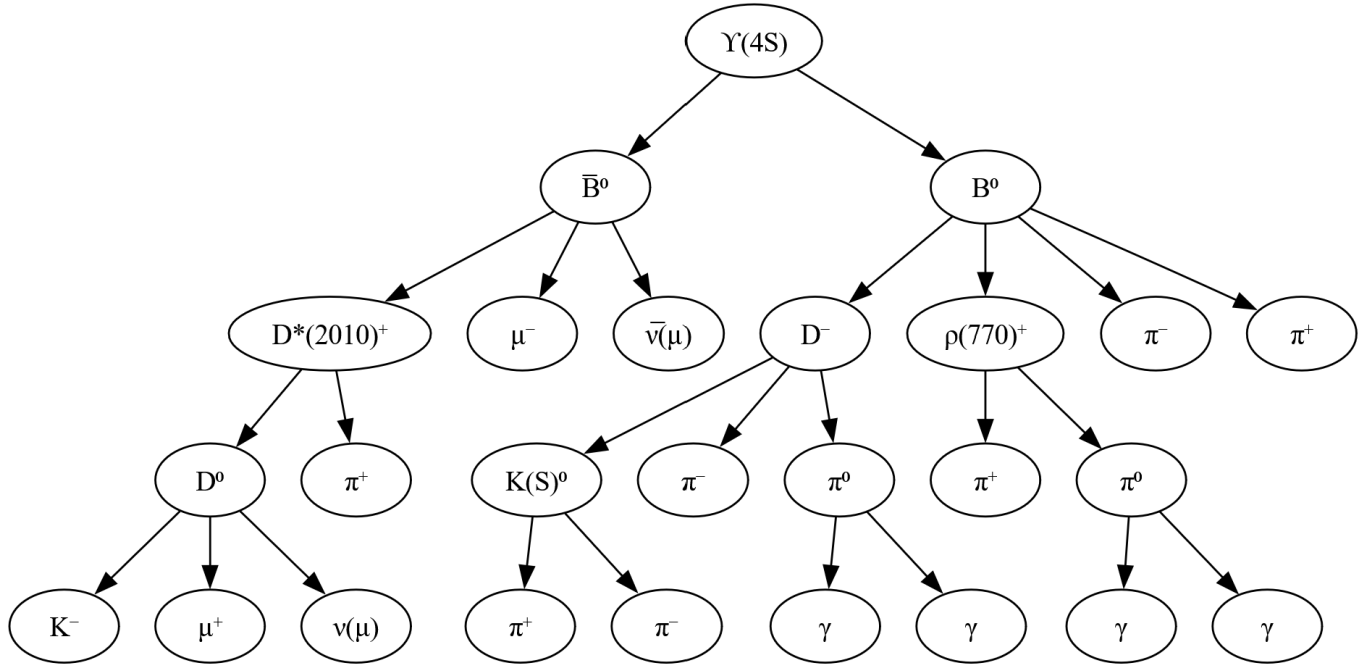
$$\min\left(e^{a(x-b)}, 1\right)$$

Two parameters:
- Scale a
- Shift b



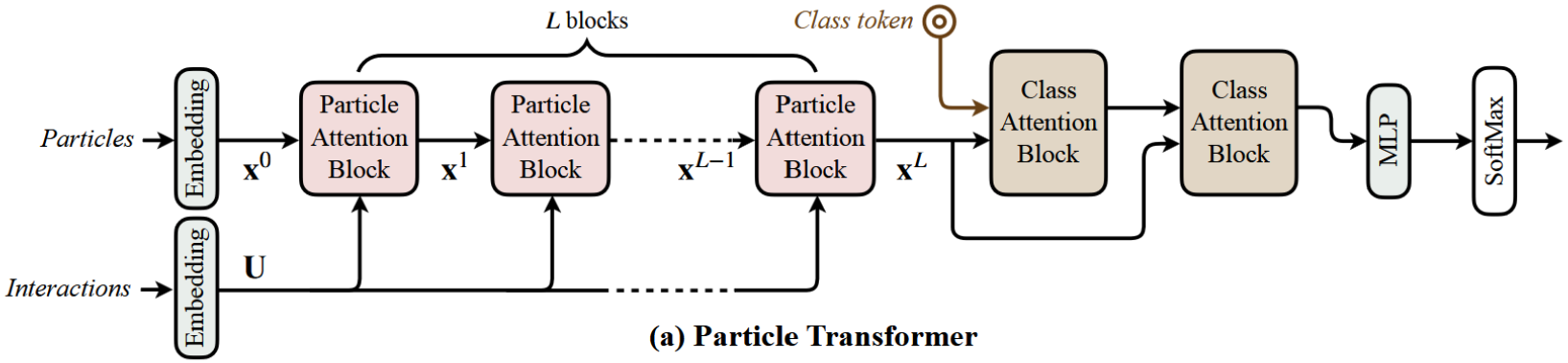
Dataset

- Generator level MC particles (hadrons, no gluons/quarks)
- List of particles with decay structure information (mother-daughter relations)
- Particle features:
  - PID
  - 4-momentum
  - Production 4-vertex

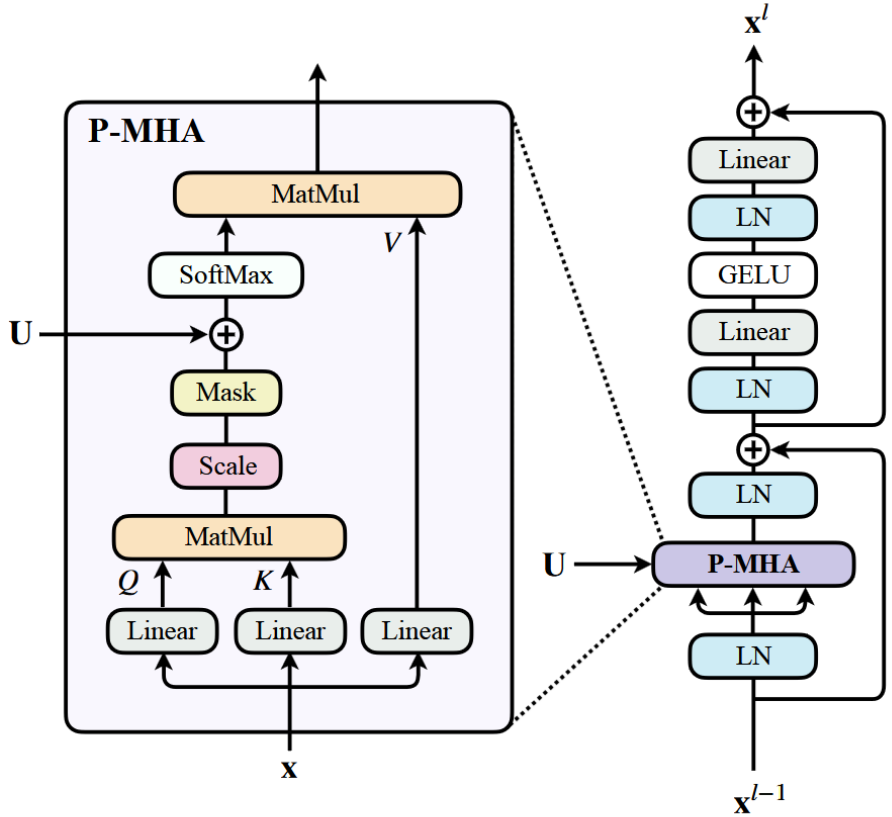


Model architecture

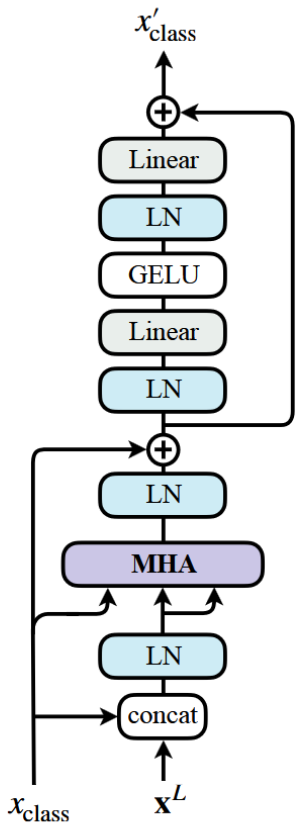
- Based on **ParT** (Particle Transformer for Jet Tagging, arxiv:2202.03772)
  → State-of-the-art performance in jet tagging by **pre-training** on large dataset and subsequent **fine-tuning** (e.g. for top tagging)
- Very generic architecture (essentially just a transformer)
- Supports edge features:
  - Adjacency matrix of decay graph
  - Two-particle invariant masses and angles
- 10 layers, $\sim$2 million parameters



(a) Particle Transformer



(b) Particle Attention Block



(c) Class Attention Block

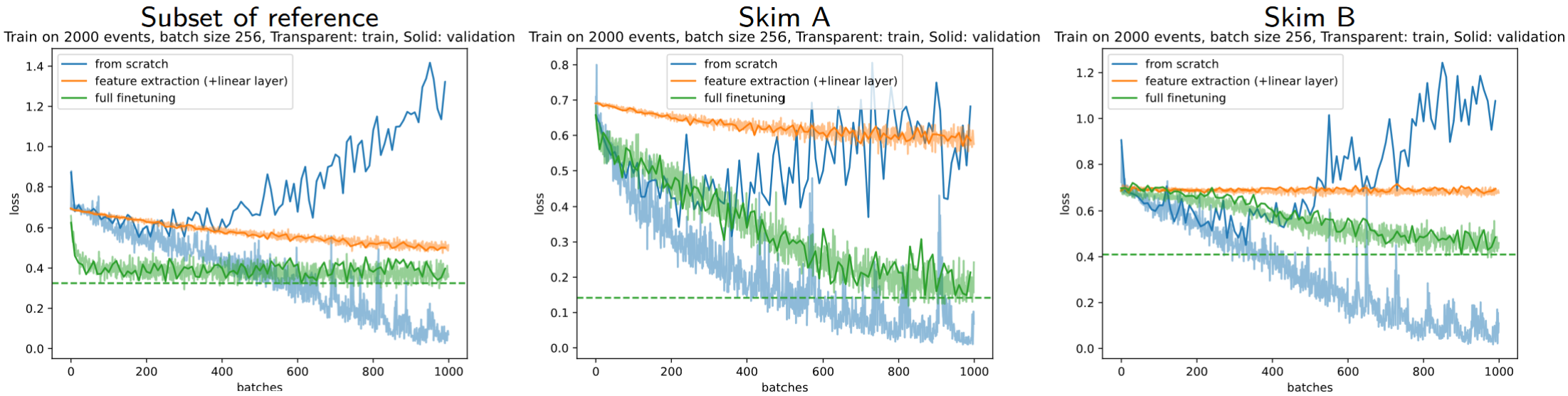
Transfer learning

- Fundamental problem:
  - Training transformer requires **large dataset**
  - But if a large dataset for a skim is already available: **what do I need the model for?**
- Solution: pre-trained models!
- Two approaches:

Feature extraction:
- Remove final layer and retrain only that
- Perhaps even just re-fit final sigmoid activation
- Only feasible for highly correlated skims

Whole model fine-tuning:
- Start with pre-trained model but adjust all parameters
- Perhaps reinitialize last layer (if output shape changes)
- Promising results in ParT paper

Transfer learning - Prestudy



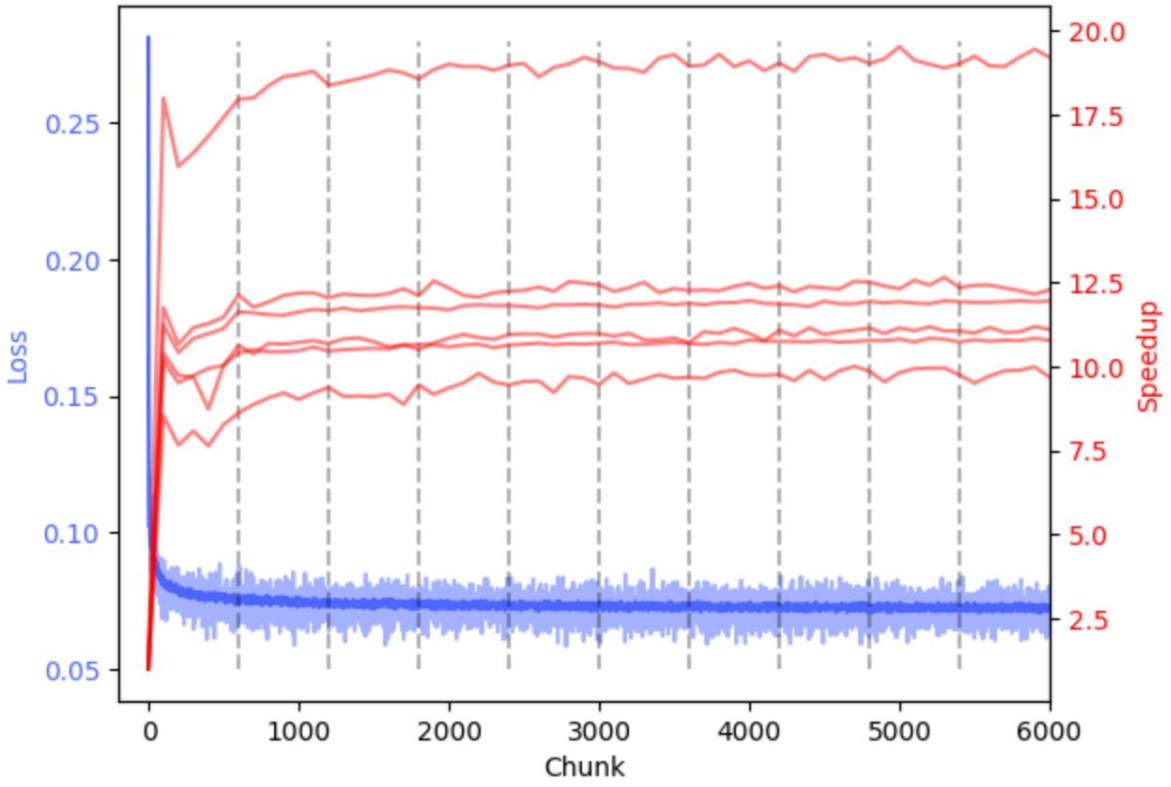
- Model trained on reference skim, then fine-tuned on 2000 events of new skim
- Feature extraction works poorly, especially with low correlations between skims (A and B)
- Training from scratch overtrains very quickly on small dataset
- Full fine-tuning looks promising, consistently yielding the best validation loss

Large scale training

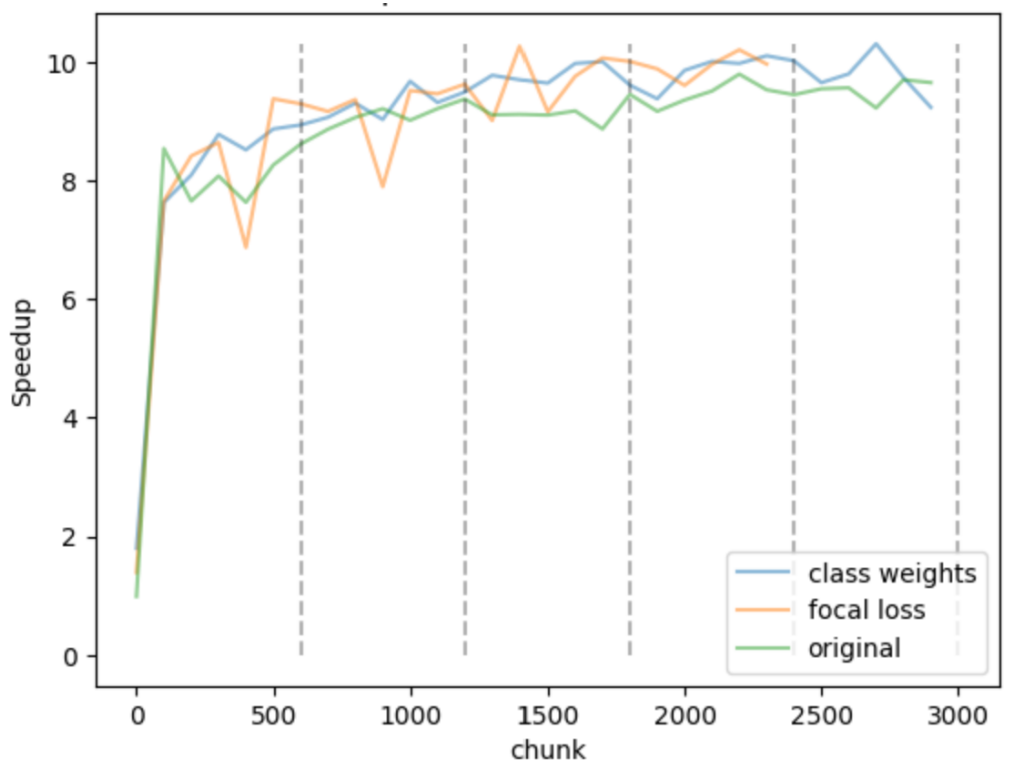
- Idea: to make flexible fine-tuning possible, pre-train model on **all available skims** at once
  - Currently samples for 51 centrally produced skims available
- 7 background types (useful for many analyses): $e^+e^- \to B^+B^-$, $B^0\bar{B}^0$, $q\bar{q}$ (q=u,d,s,c), $\tau^+\tau^-$
  - Condition model on background type
- Dataset balanced between different types, containing ~ 180 million events or $\sim 20 \mathrm{fb}^{-1}$
- Training
  - Without class weights (original)
  - With class weights
  - With class weights and empirical *focal loss* emphasizing difficult training examples

Training curves

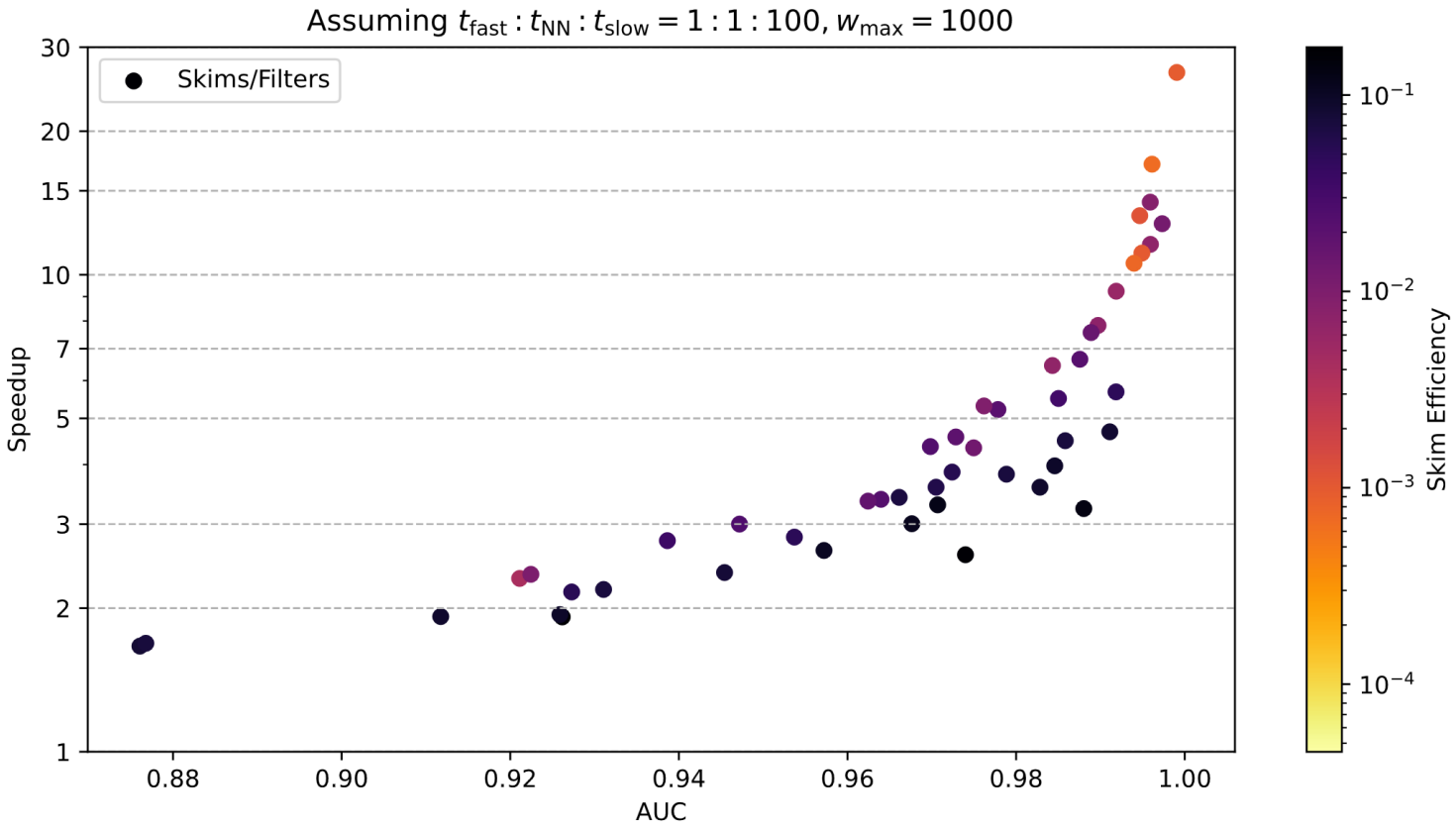
Original training



Training Comparison (example skim)



Speedups



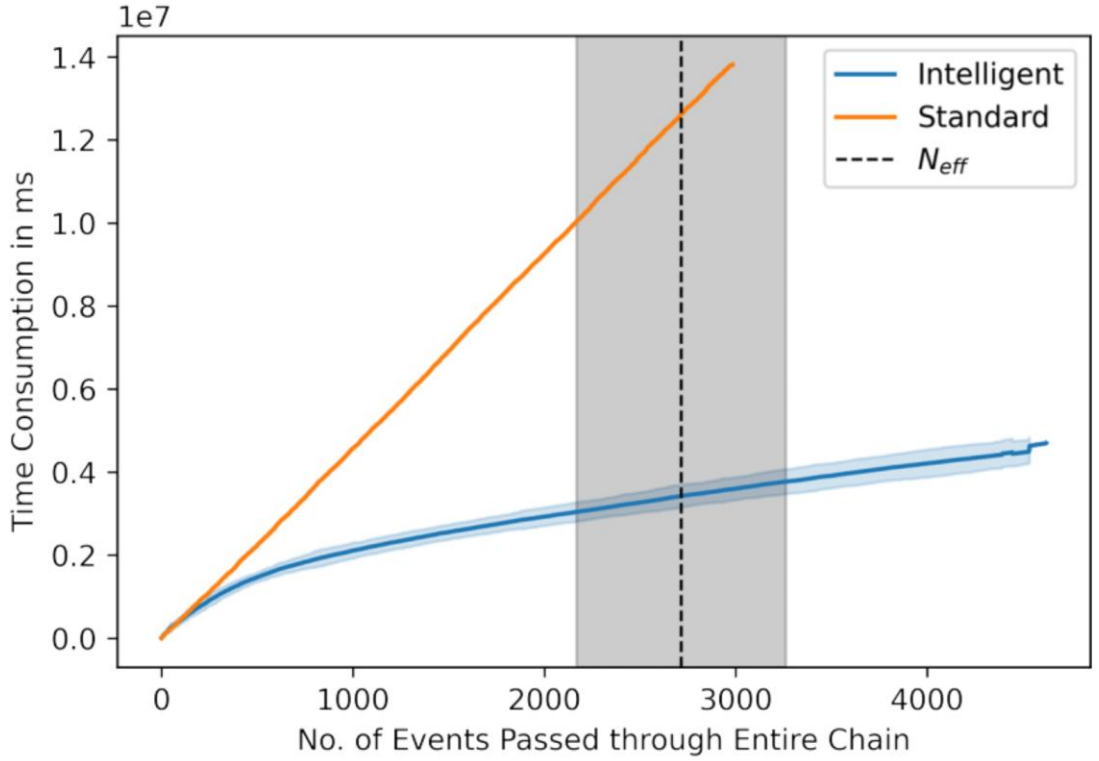
Achievable speedups **correlate with**

- **Separation power**
  (as measured by the area under the ROC curve)
- **Skim efficiency**
  (lower retention rates → higher potential for speedup)

Adaptive/Reinforcement Learning

(Introduced in Bachelor's Thesis by Daniel Pollmann (2024))

- Another possible idea:
  - Train model while producing data and running skim
  - Model becomes successively better, producing data more efficiently
- Pro: Overall time-saving, on-the-fly procedure in one step
- Contra: Requires implementing training loop in Belle II production software



Current work and outlook

Currently three main objectives:

- **Test** pre-trained model on already known skim under **different data taking conditions**
  - E.g. reprocessed data under a new software release or with more realistic beam background from so-called run-dependent MC
- **Fine-tune** and evaluate pre-trained model on **entirely new skim**
- Implement model inference in Belle II software to test model in an **actual skim production** and evaluate realistic speedup

Thank you for your attention!

Backup

Speedup derivation

- $t_{noNN} = t_3 \cdot N = t_3 \cdot \frac{\left(\sum_i \omega_i\right)^2}{\sum_i \omega_i^2} = t_3 \cdot \frac{\left(\varepsilon\, N_{gen}\right)^2}{\varepsilon^2\, N_{gen}\, E[\omega_i]} = \frac{t_3\, N_{gen}}{E[\omega_i]}$
- $t_{NN} = t_1 \cdot (N_{TP} + N_{FP}) + t_2 \cdot (N_{FN} + N_{TN}) = t_1 \cdot N_{gen}\,(\varepsilon\, f_{TP} + (1-\varepsilon)\, f_{FP}) + t_2 \cdot N_{gen}\,(\varepsilon\, f_{FN} + (1-\varepsilon)\, f_{TN})$

$$\text{Speedup} = \frac{t_3 / E[\omega_i]}{(\varepsilon\, f_{TP} + (1-\varepsilon)\, f_{FP})\, t_1 + (\varepsilon\, f_{FN} + (1-\varepsilon)\, f_{TN})\, t_2}$$

Hyperparameters

- Largely based on ParT:
  - 8 self attention blocks
  - 2 class attention blocks
  - 8 heads per multi-head attention block
  - Embedding size 128

  → Roughly 2 million parameters
- Differences:
  - Fewer norm layers
  - Embeddings for PDG ID and background type
  - Using 3 pair features (adjacency matrix, pair mass and angle)