Pretrained Event Classification Model for Collider Experiments

Shuo Han, Joshua Ho, Ryan Roberts, Haichen Wang
University of California, Berkeley
Lawrence Berkeley National Laboratory

November 6, 2024
ML4Jets 2024

Berkeley
UNIVERSITY OF CALIFORNIA

BERKELEY LAB

Introduction

- Machine Learning is one of the most powerful analysis tools we have access to
- However, ML models are expensive to train:
  - GPU Hours
  - Large sample of simulated data
- In collider physics, each experiment carries out hundreds of measurements, most of which require many iterations of training neural network models.
- Our goal: develop a single model that can be used for a wide range of tasks
  - Increases the overall performance
  - Decreases the training time
  - Works even when limited in training statistics
- This development could contribute to a future foundational model for particle physics

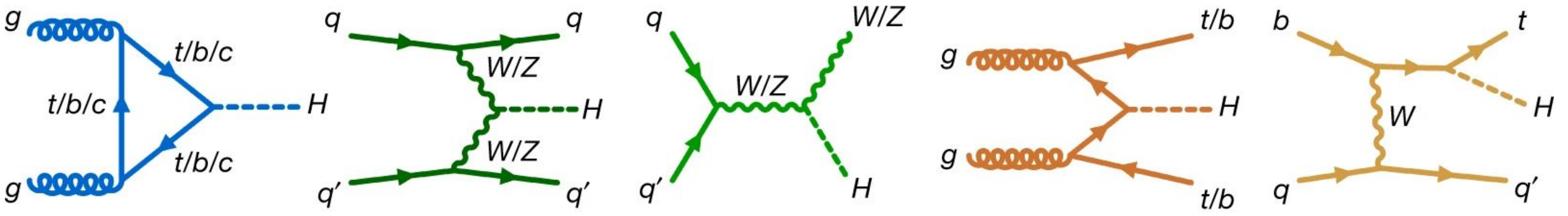
Statement of Problem

- As a proof of principle, we start with a classification task
- Our goal is to improve binary classification by using a pretrained model for:
  - Higgs production processes
  - Top quark related processes
- I will be discussing techniques for training a pretrained model, as well as how to finetune it for analysis
- Comparing the performance between baseline model with models that utilize the pretrained model
- Techniques to introduce model interpretability → model similarity
- GPU resources cost comparison between model frameworks

Training Setup: Pretraining Model Data

Pretrained Model Training Data:

- Higgs production processes: ttH, tHjb, ggF, VBF, WH, ZH
- Top quark processes: ttyy, tttt, single top, ttbar, ttW, ttt
- Statistics: ~120M total (~10M per class)

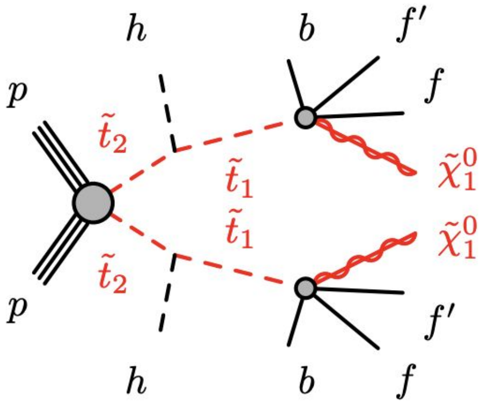


Training Setup: Analysis Model Data

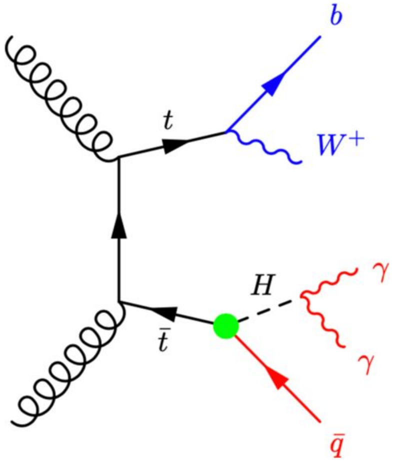
Example Analysis Tasks:

- ttH CP Even vs CP Odd (H → γγ)
- FCNC vs tHjb (H → inclusive)
- stop vs ttH (H → inclusive)
- WH vs ZH (H → inclusive)
- ttW vs ttt

Statistics: ~20M (10M per class)



stop



FCNC

“Fine-tuning”: The act of taking the pretrained model and specializing it for a specific analysis task.

Training Setup: Inputs

Graph Neural Networks (GNNs) are a natural choice because of the point-cloud-like structure of our data

We will be using GNNs as a proof of concept for exploring techniques to be used in developing a Foundational Model

Input Features:

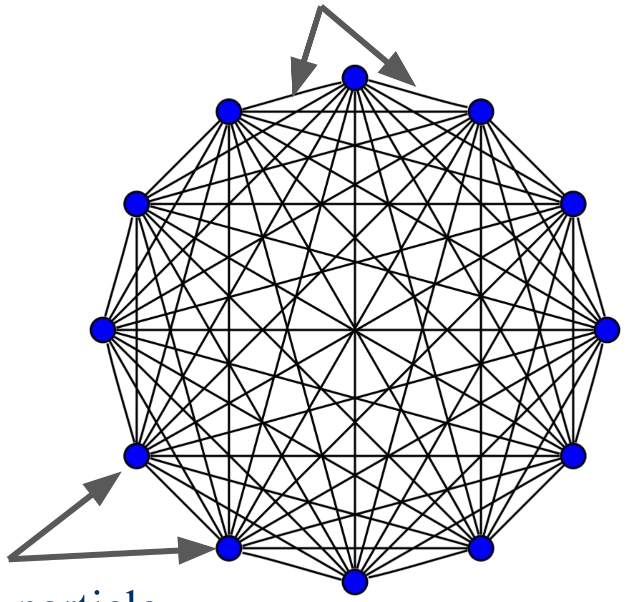
- Reconstructed objects: particle 4-vectors
- Particle Labels: type, b-tagging, lepton charge

Edge Features:

- Angular and Translational Separation

Global Features:

- Number of particles in each graph

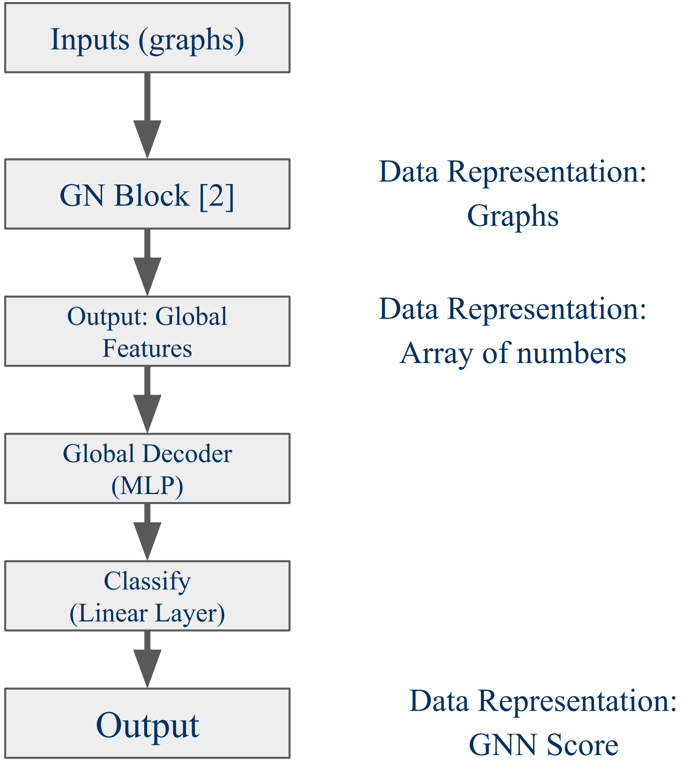


Training Setup: Baseline Model Architecture

Baseline Model:
A standard GNN trained for binary classification for one of our example analysis tasks.

Implementation of the model used in the ATLAS 4 tops observation [1]

Pytorch and Deep Graph Library (DGL)



[1] The ATLAS Collaboration: Eur. Phys. J. C 83, 496 (2023)
[2] arXiv:1806.01261

Training Setup: Pretrained Model Architecture

Pretraining Model:
Same architecture as before, but trained on **large and diverse** dataset, with a different training goal.

Multi-Class Classification:
The goal of the pretraining is to separate the data by process

Predictions:

- P(ttH)
- P(ggF)
- P(WH)
- … etc

Multi-Label Classification:
The pretraining will predict various variables corresponding to the kinematics of the system

Prediction:

- Exists: higgs_exists, top1_exists, …
- Pt: higgs_pt, top1_pt, …
- η: higgs_eta, top1_eta, …
- ***ϕ***: higgs_phi, top1_phi, …

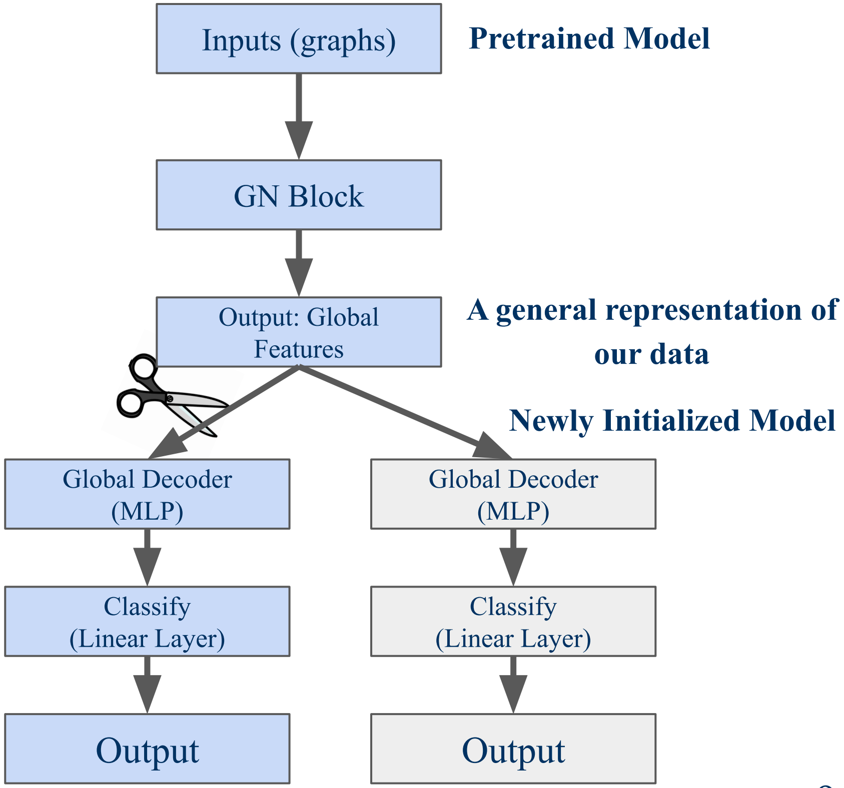
Training Setup: Fine-tuning Model Architecture

Fine-tuned Model:

Specializing the pretrained model for various analysis tasks.

- Starting point: uses the weights of the pretrained model, with a re-initialized MLP at the end
- Adjust the learning rate
    - Continue training the pretraining at a lower learning rate (~10% of the regular LR)
    - Train the newly initialized model at a regular learning rate

NOT transferred learning because the pretraining is still trainable – we saw a decrease in performance when compared to baseline



Results (Overall Performance)

Utilizing full statistics:

- 120M Pretraining
- 20M Analysis

Immediate Performance:

- Utilizing a pretrained model gives the model an initial boost in performance
- Seen in all analysis tasks

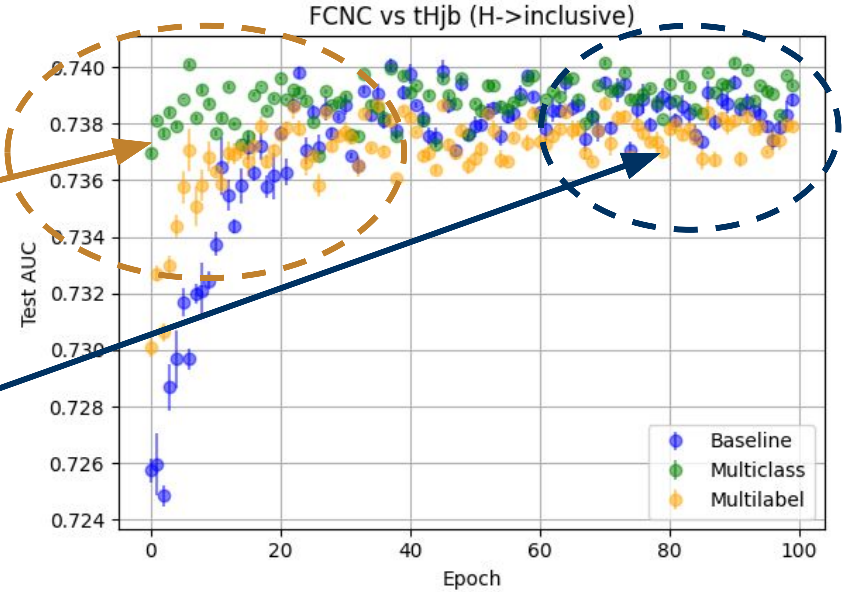
Ultimate Performance:

- The ultimate performance is increased when using the Multiclass pretraining
- Seen only in some of the analysis tasks

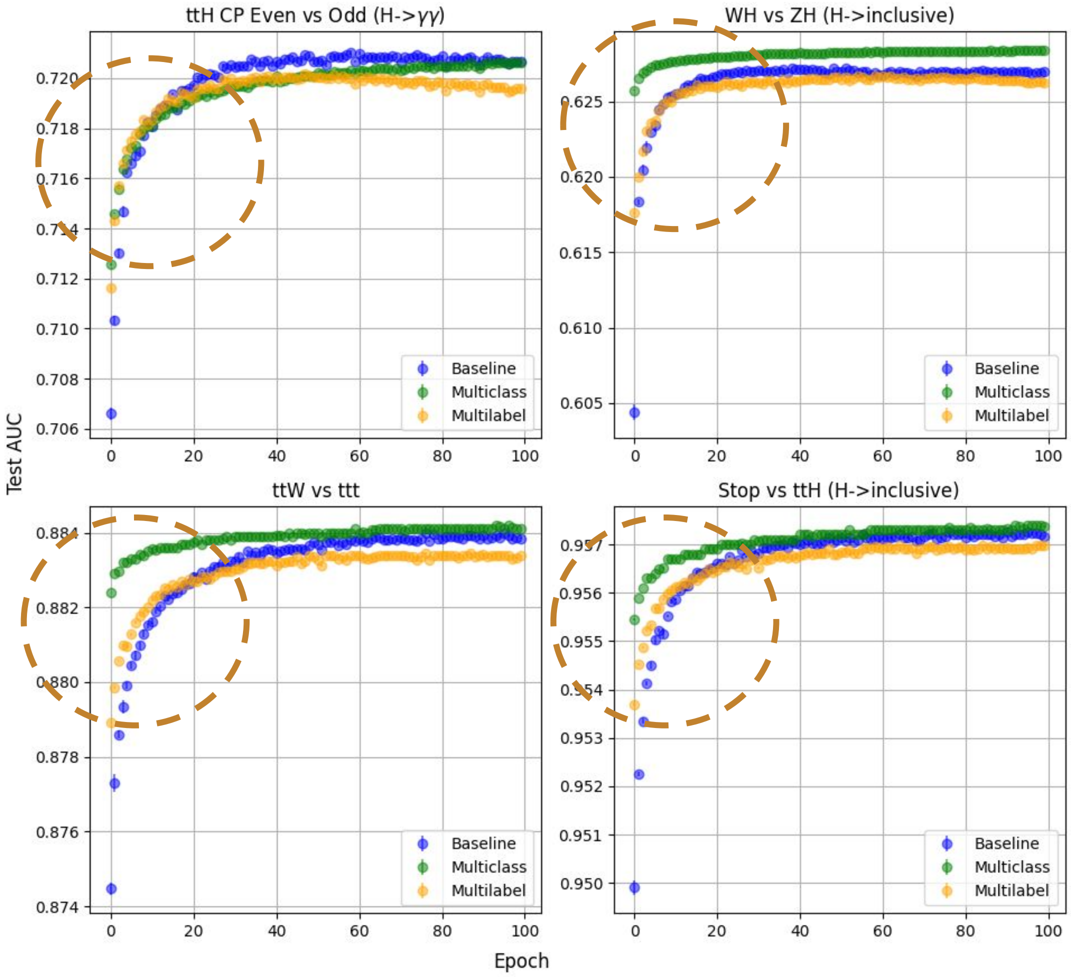
Use Cases:

- Training is expensive and we can only afford a few epochs

Performance for Example Analysis Tasks



Performance For Example Analysis Tasks



Results (Training Data Scaling)

Lacking Statistics:

- There is a significant increase in performance when the training data is lacking (up to 15% improvement in AUC)

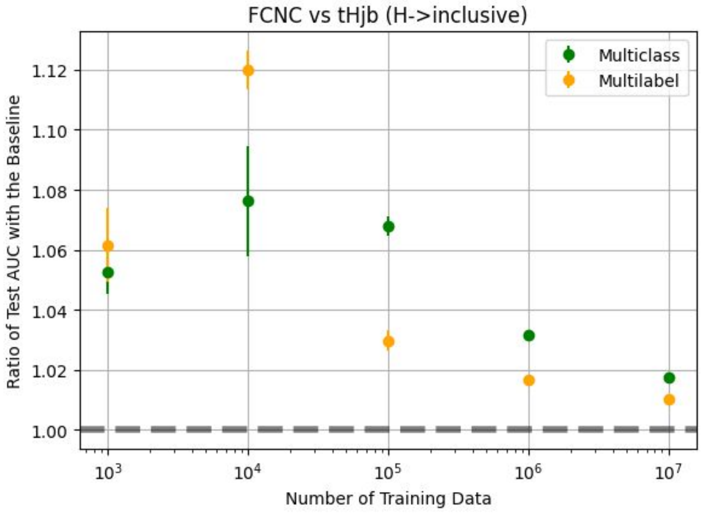
Large Number of Training Data:

- Small increase in performance (0 to 2% increase in AUC)

Use Cases:

- Training on data (unsupervised clustering)
- Division of data into smaller phase space
- Signal region statistics lacking after stringent selection
- Simulation data too expensive

Performance Gains from Pretraining vs Number of Training Data: Full Training



Performance Gains from Pretraining vs Number of Training Data: Full Training
ttH CP Even vs Odd (H->γγ)
WH vs ZH (H->inclusive)
ttW vs ttt
Stop vs ttH (H->inclusive)
Multiclass
Multilabel
Ratio of Test AUC with the Baseline
Number of Training Data

Similarity Tools

Why is the pretraining useful?

- Calculate the similarity between different models, use this information to gain insight on the information contained inside each model

Similarity Metric: Centered Kernel Alignment (CKA) [3]

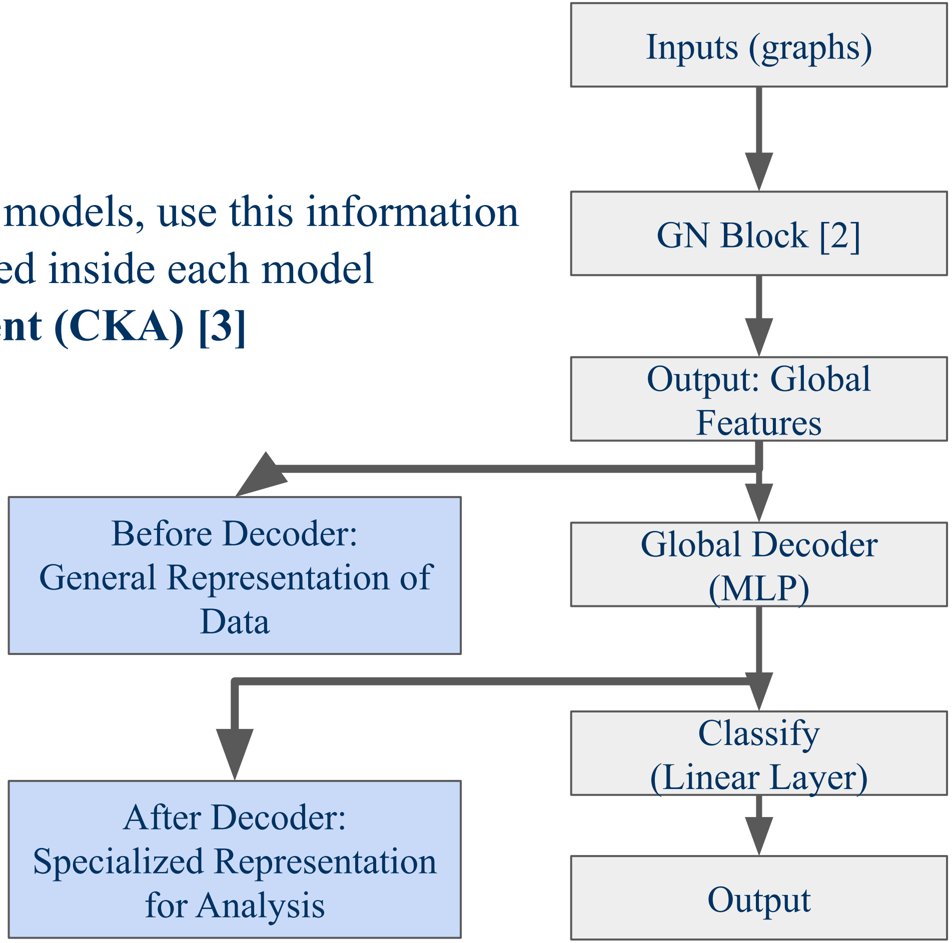
let $A, B =$ samples for CKA with shape (n, d)

$n =$ number of events

$d =$ size of representation

let $I = (n \times n)$ matrix of $\frac{1}{n}$

$$A' = AA^T$$
$$B' = BB^T$$
$$A'' = A' - IA' - A'I + IA'I$$
$$B'' = B' - IB' - B'I + IB'I$$
$$CKA(A, B) = \frac{A'' \cdot B''}{||A''|| \cdot ||B''||}$$



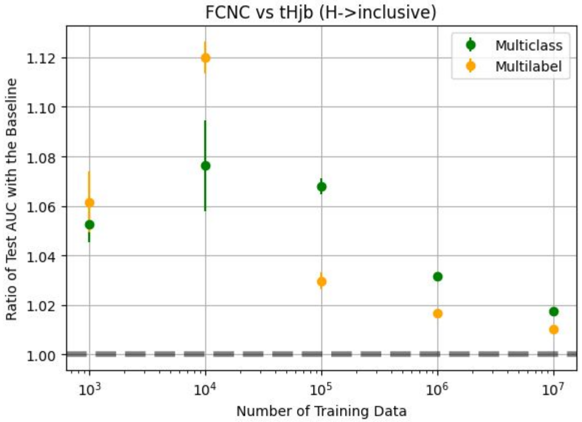
[2] arXiv:1806.01261
[3] S. Kornblith, et. al. In International Conference on Machine Learning, p. 3519–3529, 2019.

Similarity Results

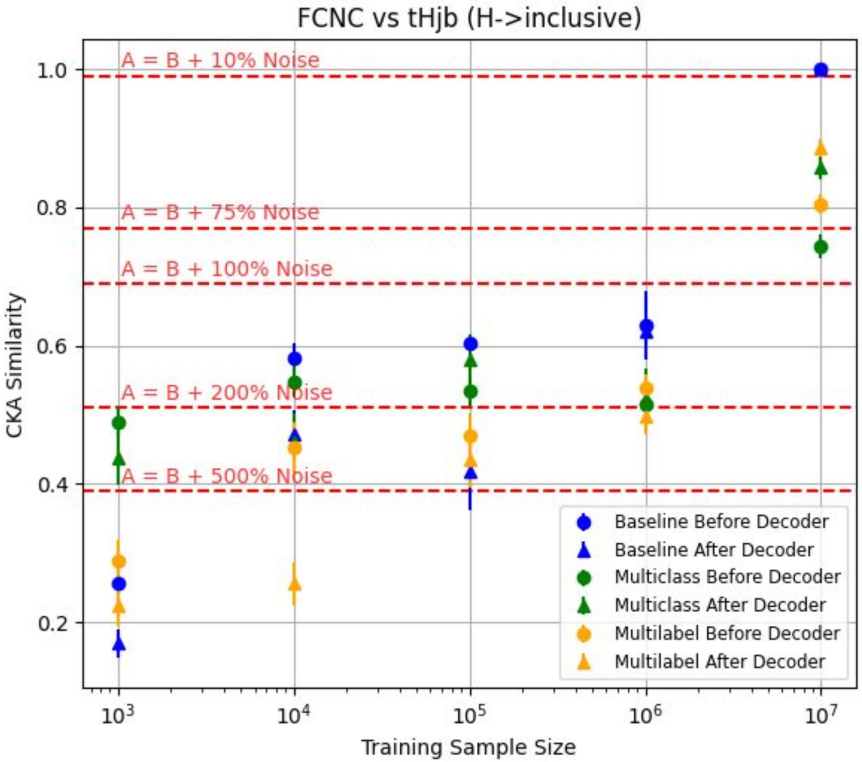
Why is the pretraining useful?

The CKA results tell us that the Multiclass and Multilabel Models utilize a different representations of collision events with respect to the Baseline.

Performance Gains from Pretraining vs Number of Training Data: Full Training



Similarity with Baseline Ultimate Performance

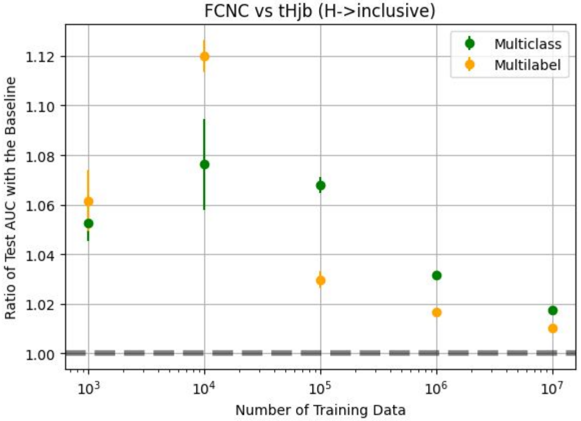


Similarity Results

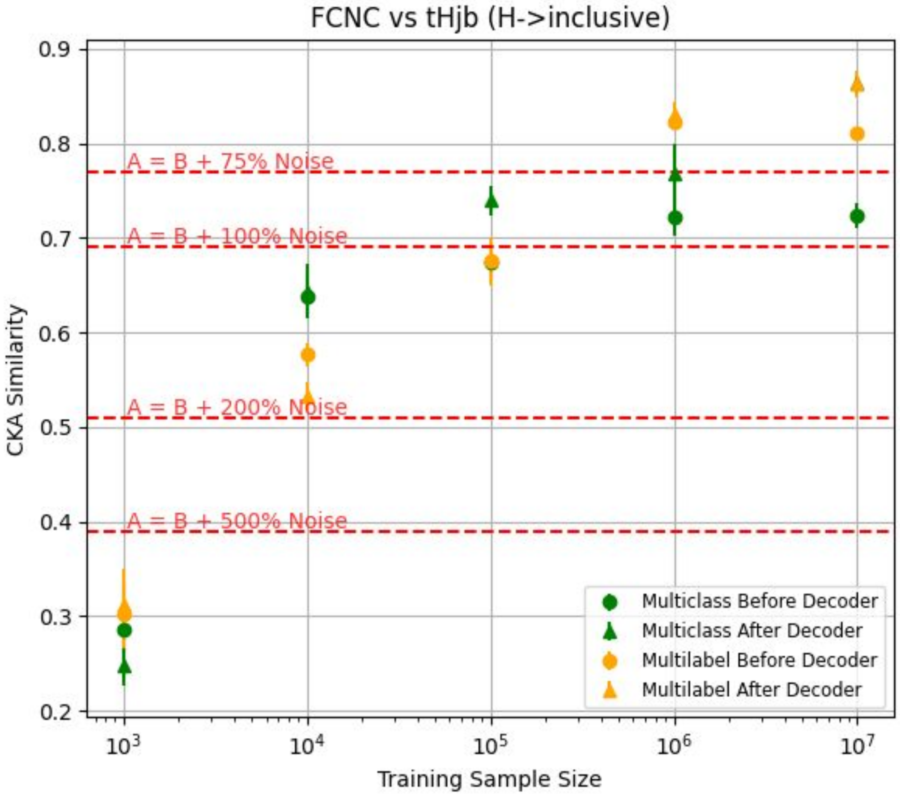
Why is the pretraining useful?

The CKA results tell us that the Multiclass and Multilabel Models utilize a different representations of collision events with respect to the Baseline.

Performance Gains from Pretraining vs Number of Training Data: Full Training



Similarity with Baseline Model



Resources For Training (Full Statistics)

GPU hours to get achieve performance of 99.9% of the baseline ultimate performance

- Multiclass Pretrianing: 45.5 GPU hours
- Multilabel Pretraining: 60.0 GPU hours

Faster than baseline	Slower than baseline

Ratio with Baseline GPU hours

Training Task	Baseline (GPU Hours)	Multiclass / Baseline	Multilabel / Baseline
ttH CP Even vs Odd	2.71	2.20	3.57
FCNC vs tHjb	2.35	0.30	4.17
ttW vs ttt	3.33	0.15	1.28
Stop vs ttH	1.78	0.27	1.01
WH vs ZH	1.98	0.15	2.52

Resources For Training (Full Statistics)

GPU Hours Required (preliminary results for these different models)

- Multiclass Pretraining: 45.5 GPU hours
- Multilabel Pretraining: 60.0 GPU hours
- Baseline: 2.43 GPU hours
- Fine tuning based on a Multiclass pretraining: 1.59 GPU hours
  - 65% faster than the baseline

If we account for the initial cost of the pretrainings, the Multiclass setup will use less GPU hours after a total of 60 separate trainings

- For every analysis, it is common to train the model many times for development, debugging, ensemble training, etc.

Conclusions

- Evidence shows utilizing pretrained models **increase performance** when **limited in statistics** or **limited in epochs trained**
- Utilizing the pretrained models achieves the same performance as the baseline faster time – these models **converge faster**
- We can use model similarity metrics to probe these complex models and gain insight on the information that they have learned
- We see that utilizing these pretrained models also leads to a **decrease in GPU resources required**

Thank you.

BACKUP

Results (Immediate Performance)

Lacking Statistics:

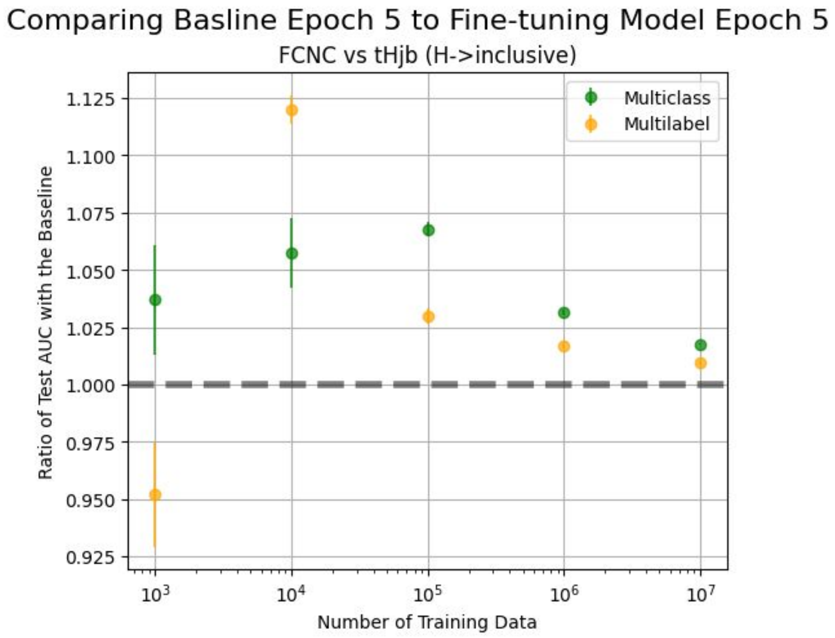
- There is a significant increase in performance when the training data is lacking (up to 13% improvement in AUC)

Large Number of Training Data:

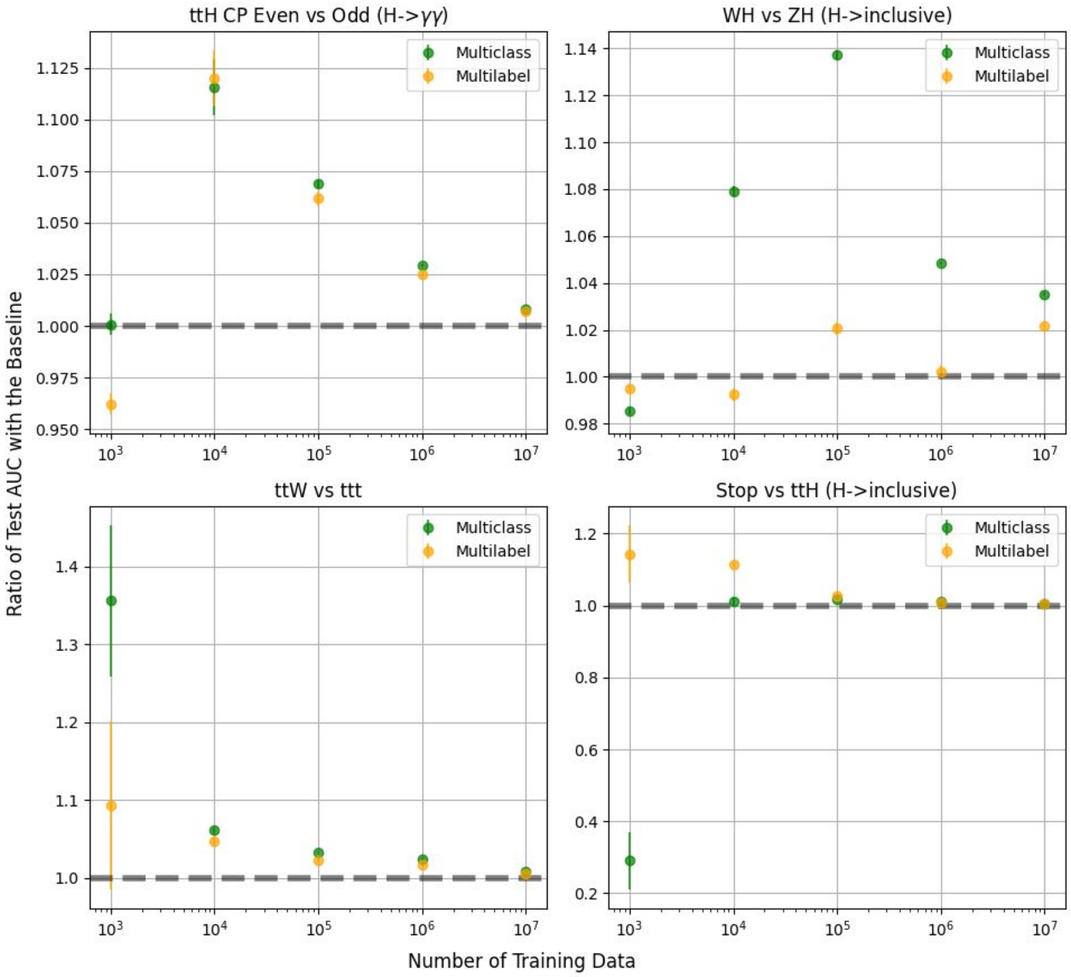
- The performance of these fine-tuned models are in general, slightly better than the baseline
- 0 ~ 2 % improvement

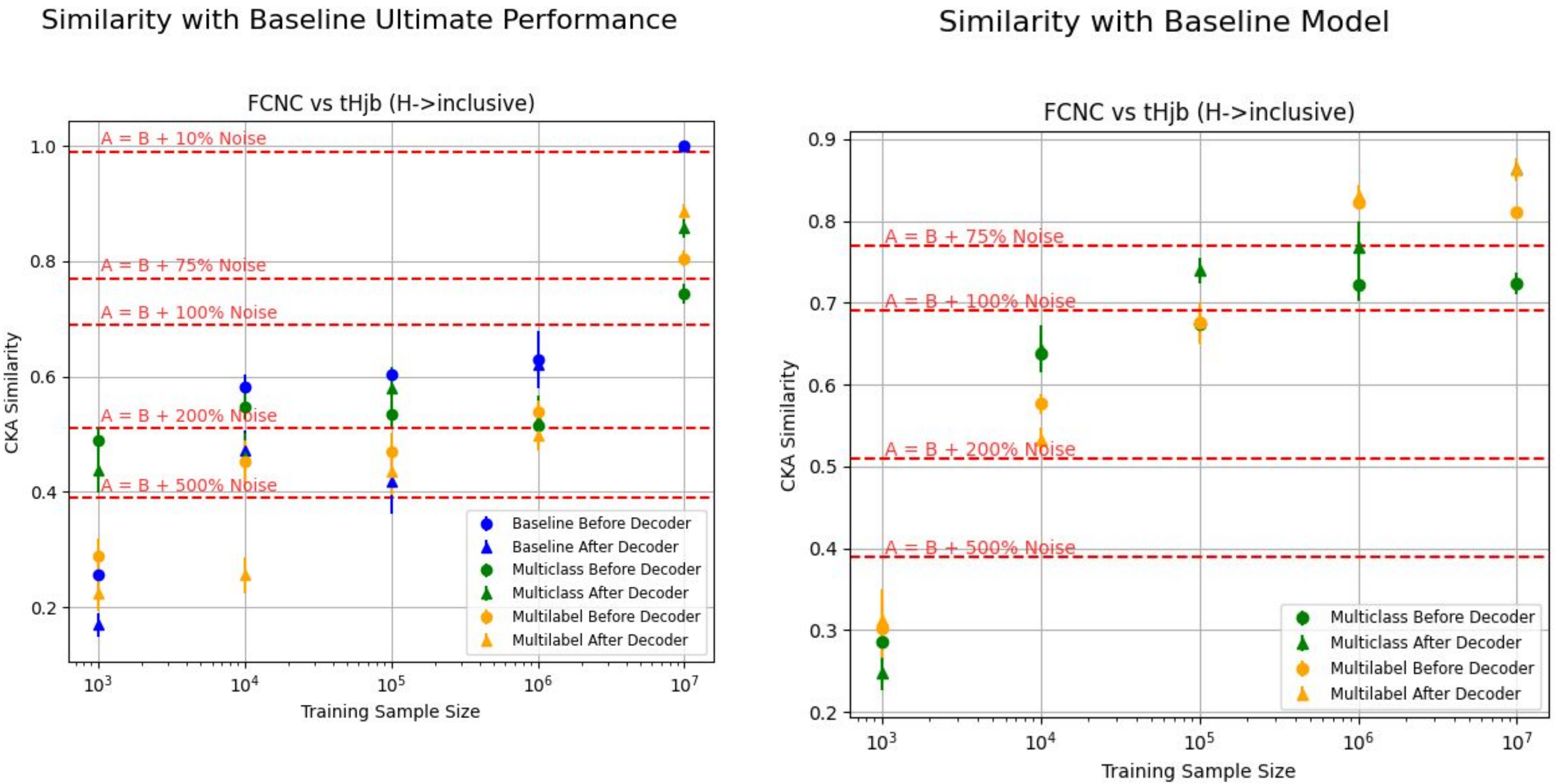
Uses Cases:

- Training is expensive, and only a few epochs can be trained

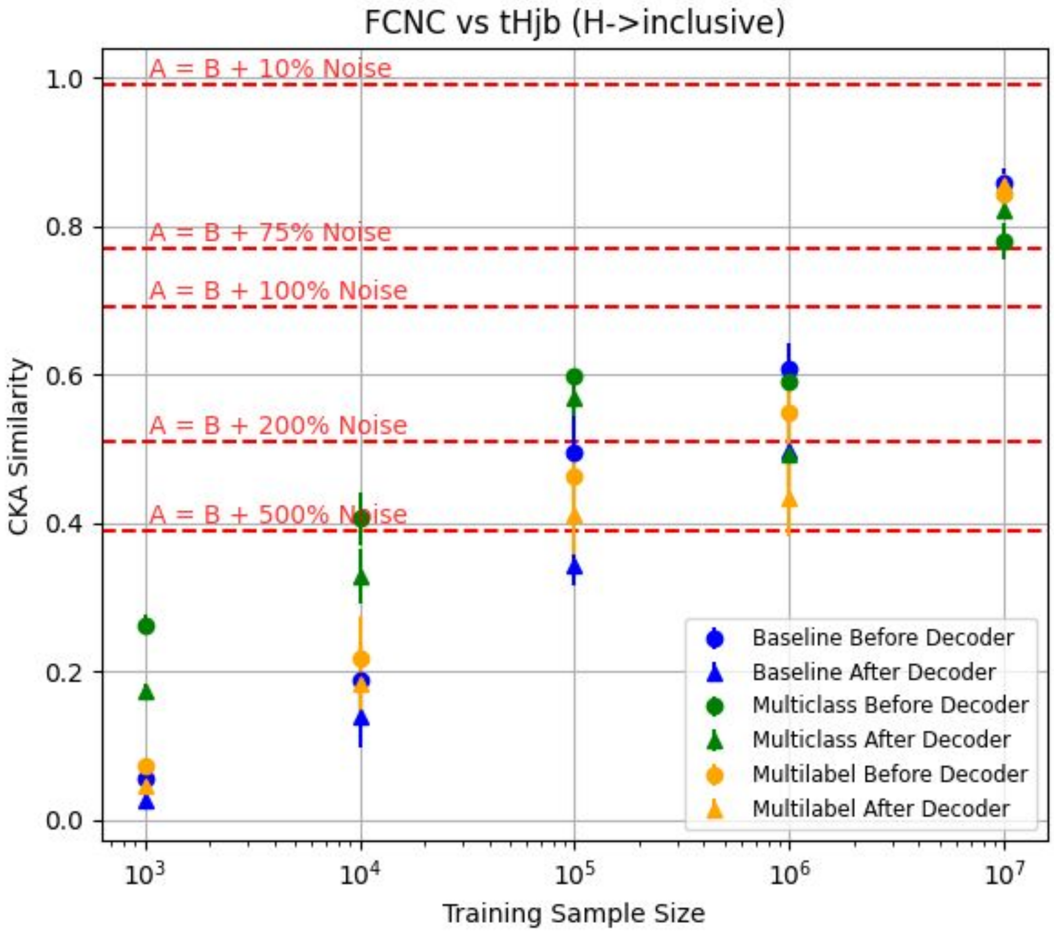


Comparing Basline Epoch 5 to Fine-tuning Model Epoch 5




Similarity with Baseline Ultimate Performance
FCNC vs tHjb (H->inclusive)
A = B + 10% Noise
A = B + 75% Noise
A = B + 100% Noise
A = B + 200% Noise
A = B + 500% Noise
CKA Similarity
Training Sample Size
Baseline Before Decoder
Baseline After Decoder
Multiclass Before Decoder
Multiclass After Decoder
Multilabel Before Decoder
Multilabel After Decoder
Similarity with Baseline Model
FCNC vs tHjb (H->inclusive)
A = B + 75% Noise
A = B + 100% Noise
A = B + 200% Noise
A = B + 500% Noise
CKA Similarity
Training Sample Size
Multiclass Before Decoder
Multiclass After Decoder
Multilabel Before Decoder
Multilabel After Decoder

Similarity of Model Immediate Performance with Baseline Ultimate Performance



Inputs

Nodes: (num_nodes, 7)
7 features (defined in config):

- pt
- eta
- phi
- Calc_E
- jet_btag
- charge
- NODE_TYPE

num_nodes = num_graphs * nodes_per_graph

- nodes_per_graph varies

Edges: (num_edges, 3)
3 features (defined in root_gnn_base/dataset/EdgeDataset:

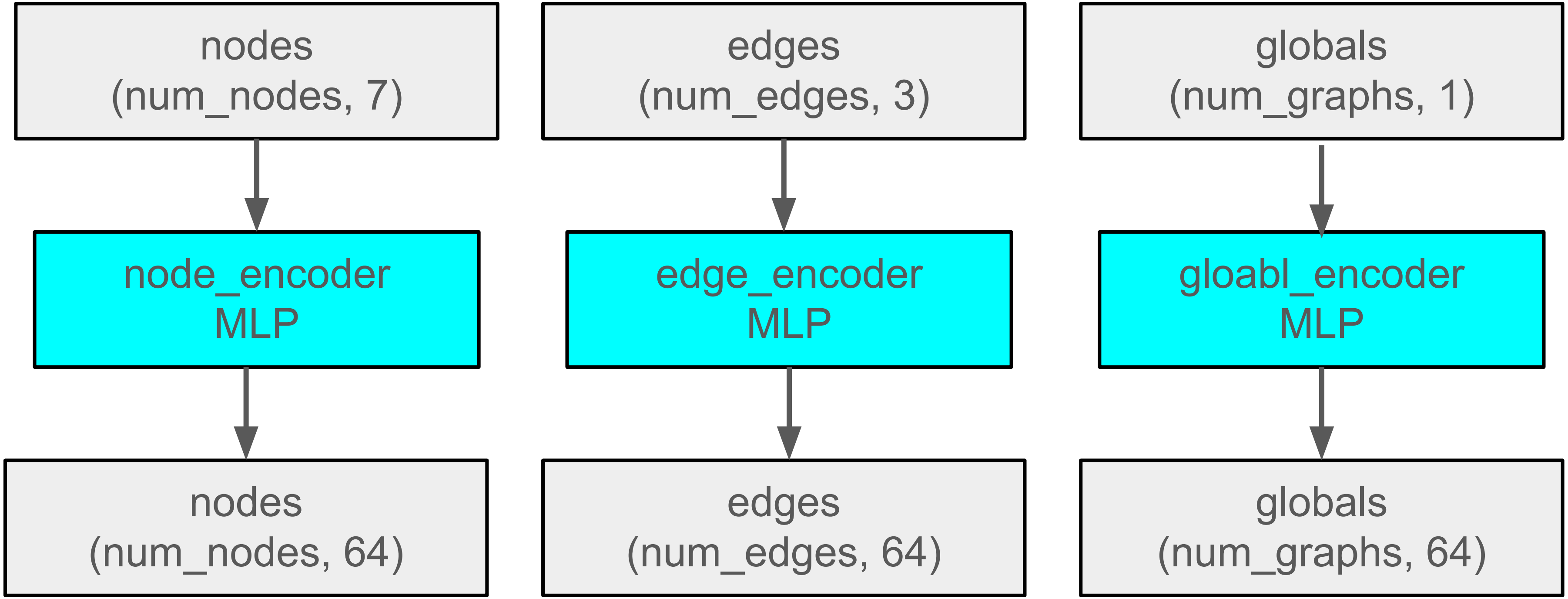
- deta
- dphi
- dR

num_edges = fully connected each node in each graph

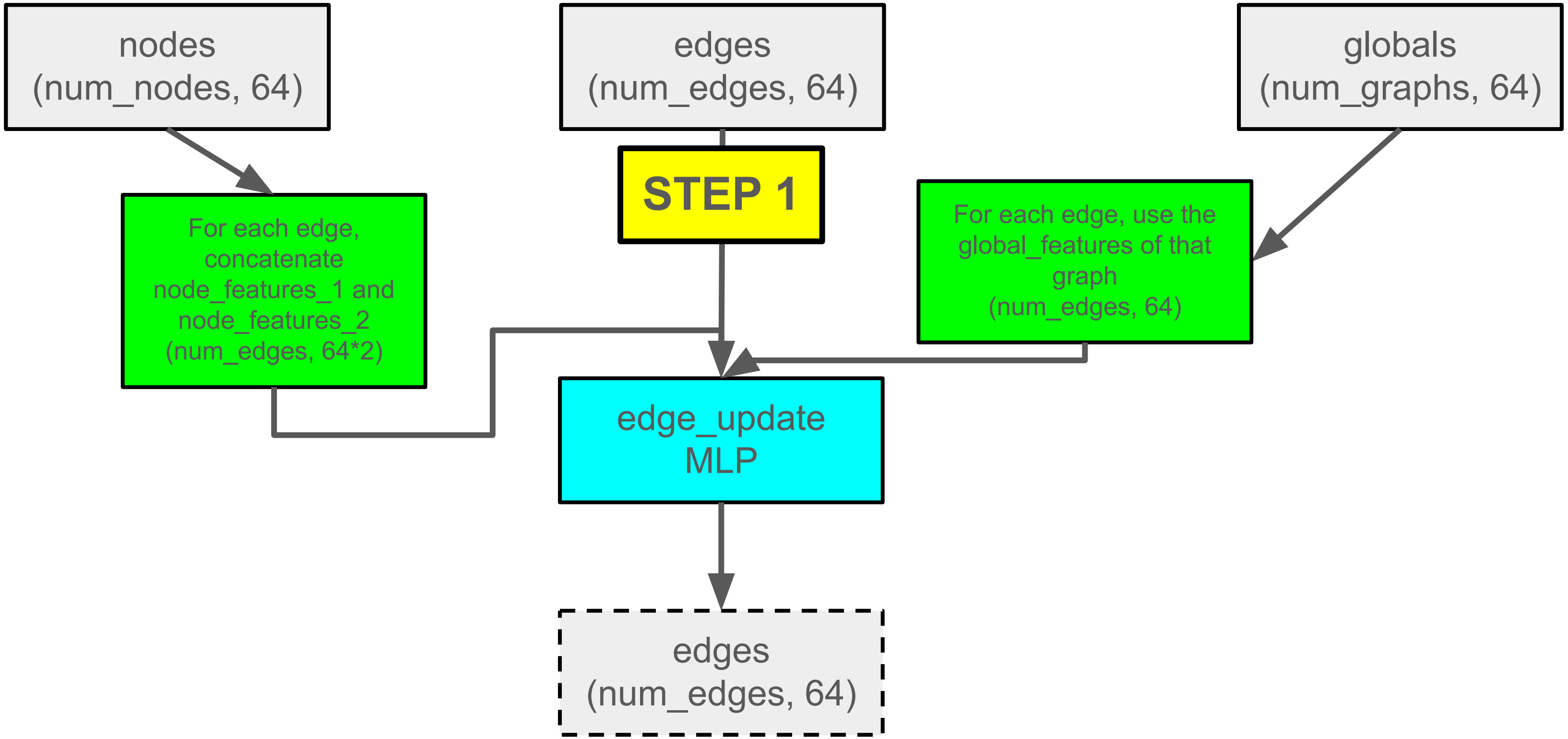
Globals: (num_graphs, 1)
1 feature (can change in config)
Default:

- number of nodes in graph

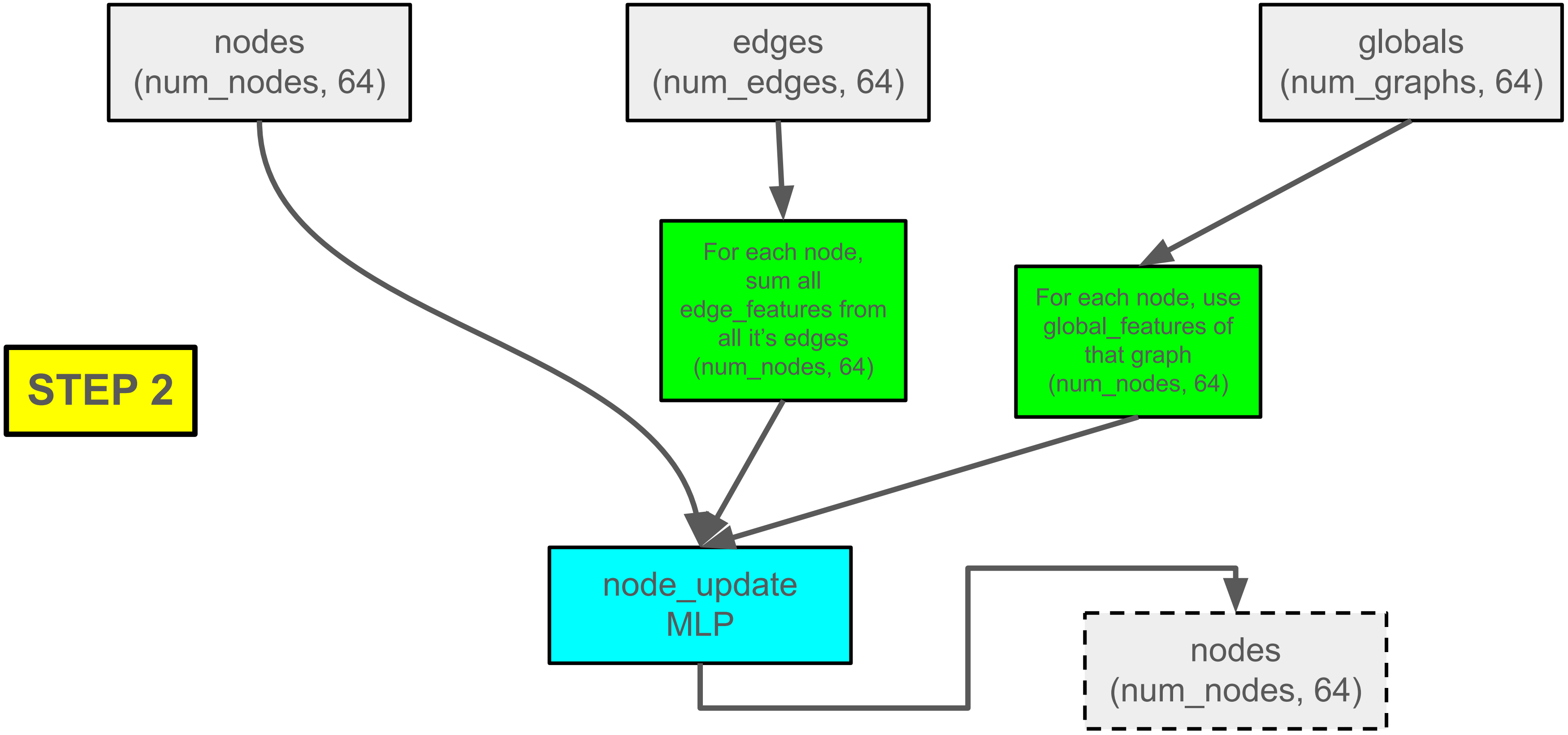
Step 1: Encoder Step



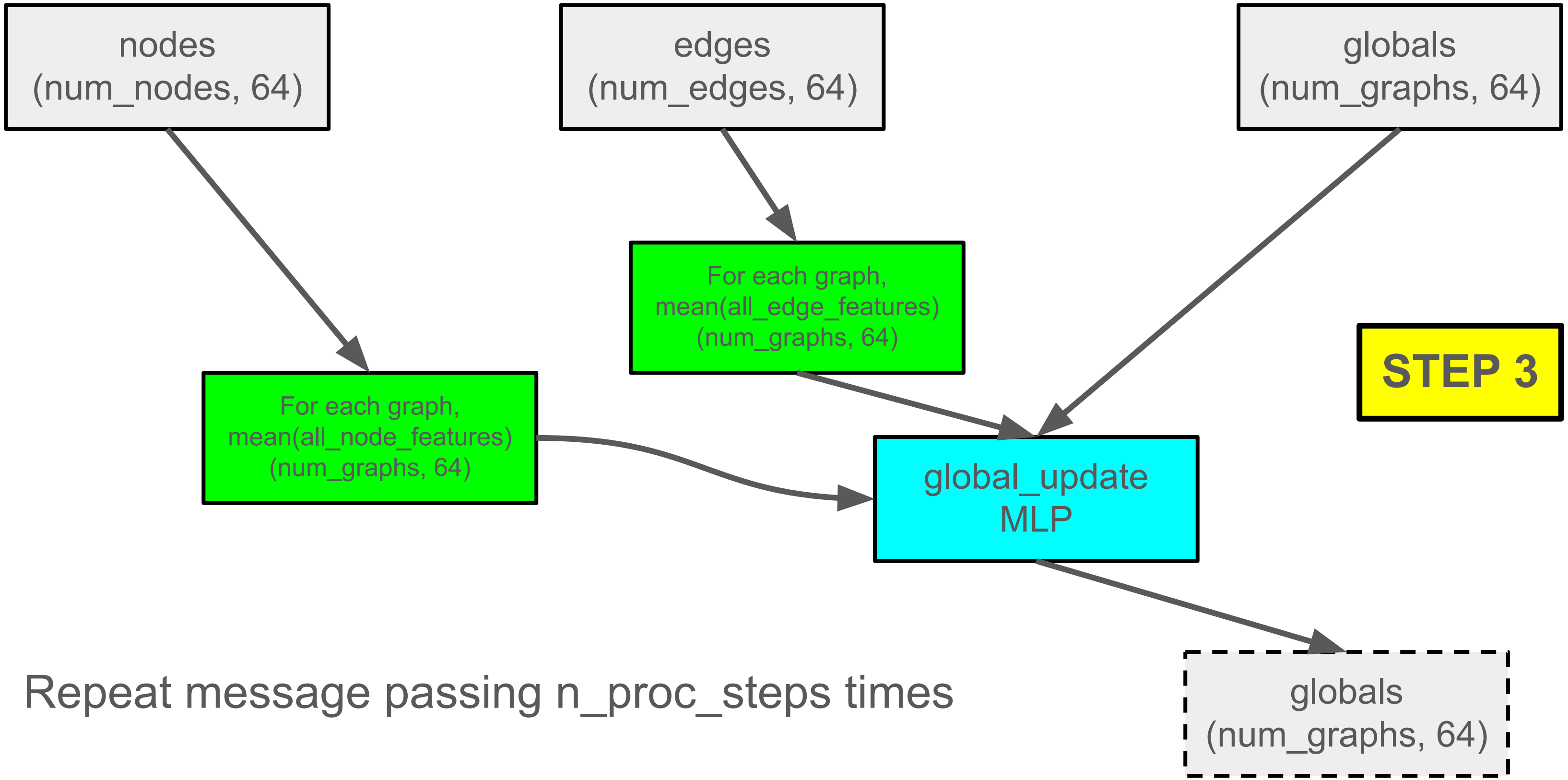
Message Passing Breakdown: Step 1 (Edge_Update)



Message Passing Breakdown: Step 2 (Node_Update)

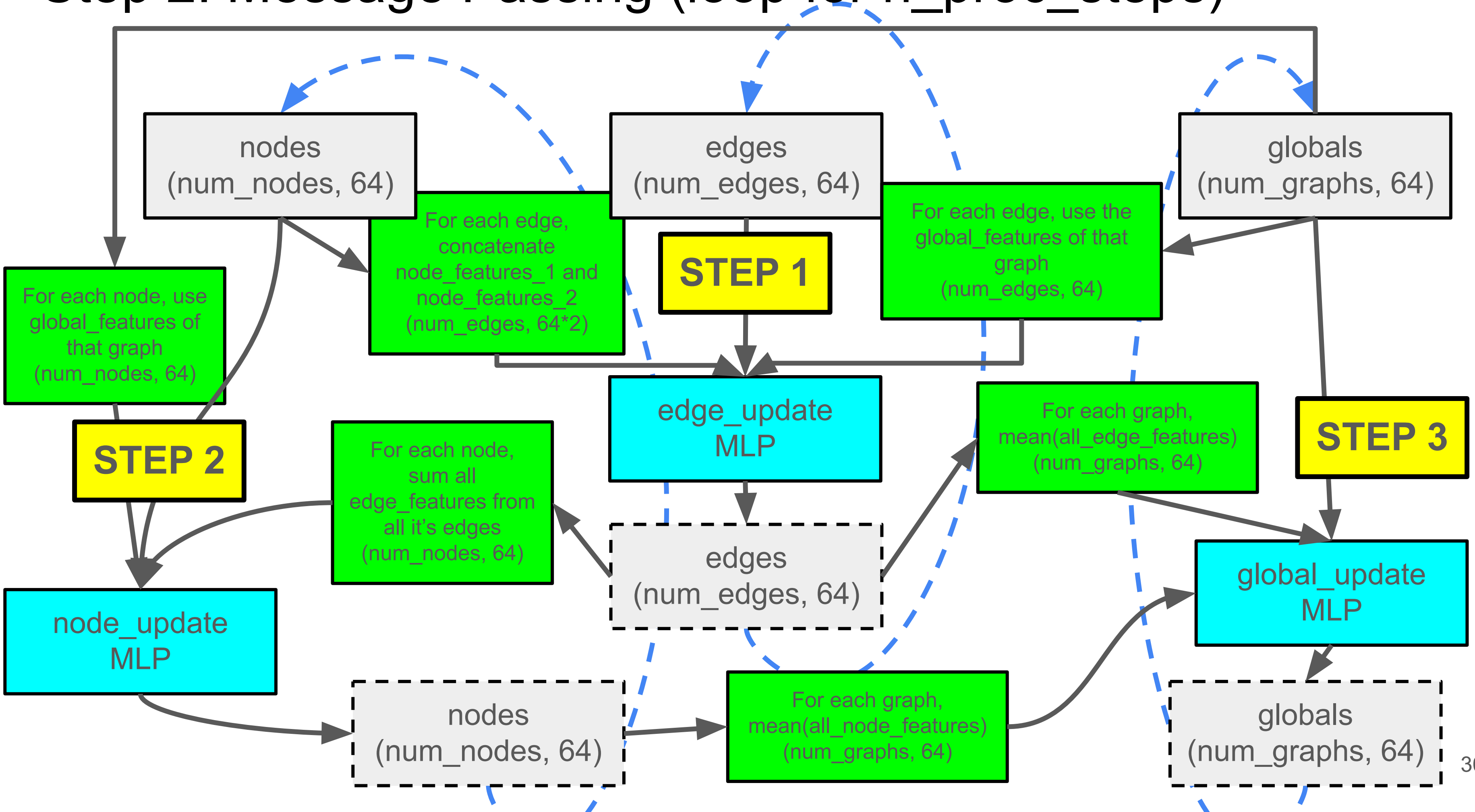


Message Passing Breakdown: Step 3 (Global_Update)



Repeat message passing n_proc_steps times

Step 2: Message Passing (loop for n_proc_steps)



Step 3: Decoder