ML ALGORITHMS ON A GPU CLUSTER @ NAPOLI

A D’AVANZO, G. CARLINO, F. CIROTTO, A. D’ONOFRIO, A. DORIA, G. SABELLA, B. SPISSO

Meeting annuale ATLAS Italia Computing, Bologna, 16/12/2025

Introduction: presentation roadmap

- The High Performance Computing (HPC) INFN Cluster: where the GPUs are
- The ML algorithms: what actually runs on them
- ATLAS use-case: how it’s done
- Performances

Introduction: presentation roadmap

- **The High Performance Computing (HPC) INFN Cluster: where the GPUs are**
- The ML algorithms: what actually runs on them
- ATLAS use-case: how it’s done
- Performances

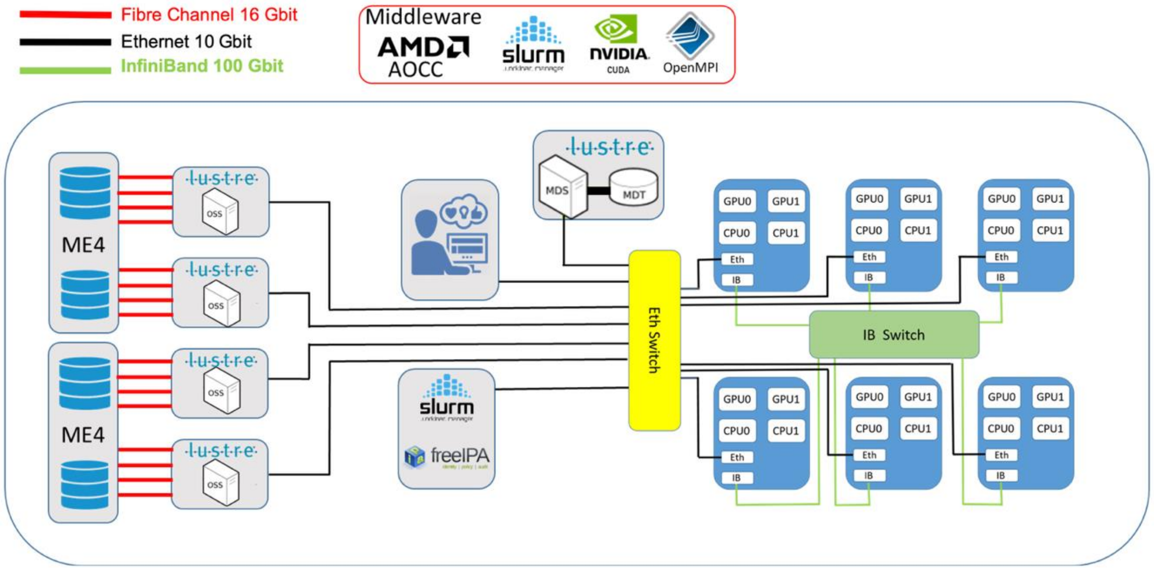
The HPC INFN CLUSTER

- HPC Cluster: Network of computing hardware (CPUs, GPUs, storages ecc.) and software (managment tools) interconnected by an infrastructure capable of fast data transfer and communication among its nodes
  - Based on the foundational “seeds” of the new IBiSCo computational resources at the INFN Naples
  - Provided with workflow automation, made flourished and functional through knowledge and expertise

3 years

General structure

- Base architecture with Alma9 OS: Slurm + Nis + Infinibad + Lustre + CUDA (**hyperthreading disabled**)
  1. 6x PowerEdge R7525 many-core compute nodes with GPUs (2 nodes recently added!)
  2. 2x Dell Powervault ME4 storage system with a gross capacity of approximately 1000 TB
  3. 100 Gbit/s InfiniBand interconnect
  4. 2x AMD EPYC 7742 CPUs, 128 (64x2) cores @ 2.250 GHz
  5. 2x NVIDIA V100 16 GB PCIe3.0 GPUs
  6. DDR4 main memory of 1200 GB
  7. 2x 446.63 GB SSD SATA and 2x 3576.38 GB SSD SATA

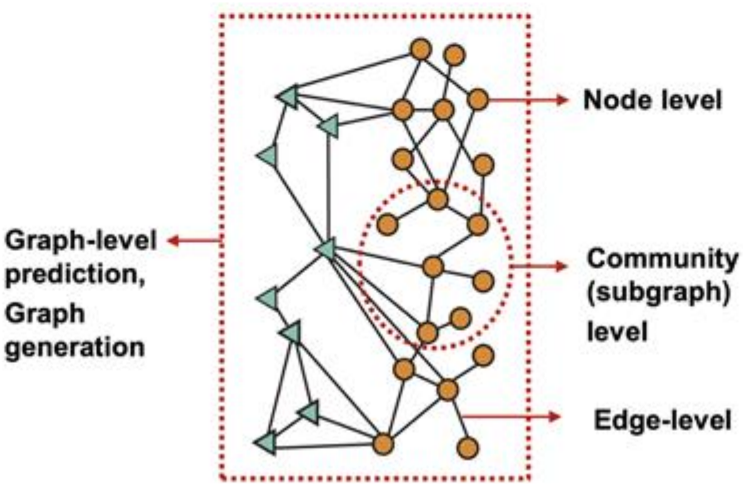


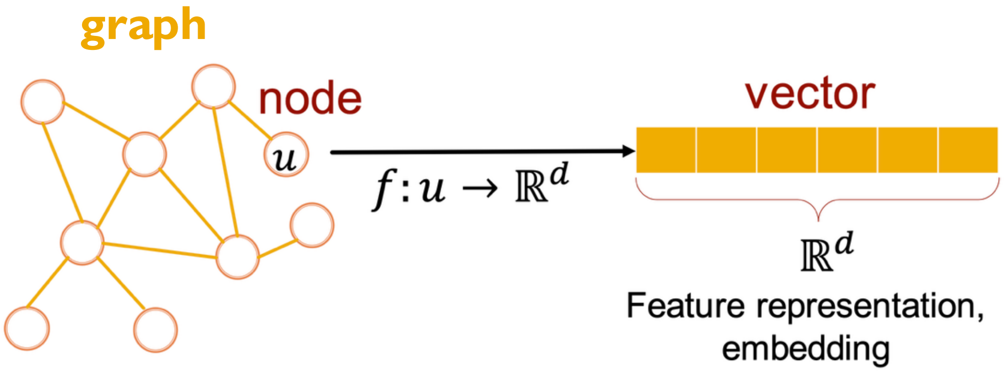
Introduction: presentation roadmap

- The High Performance Computing (HPC) INFN Cluster: where the GPUs are
- **The ML algorithms: what actually runs on them**
- ATLAS use-case: how it’s done
- Performances

Graphs in data

- **<u>Graph Neural Networks</u>** (GNNs) are ML architectures built specifically to make predictions on graphs, mathematical structures consisting of nodes connected by some kind of relation (edge).
  - Nodes and edges typically contain **features** specific to each element and each pair
  - Ideal in case of data that naturally fit this structure (protein chains, social networks ecc.)
  - Training used to learn the vector representation (embedding h_v) of each node of the input graphs by a **message passing** mechanism.

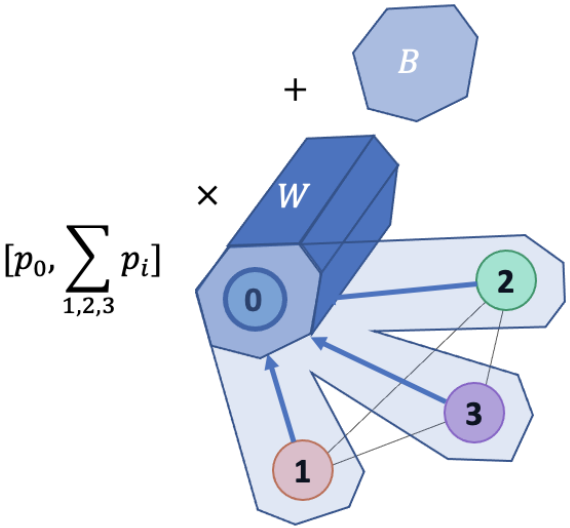




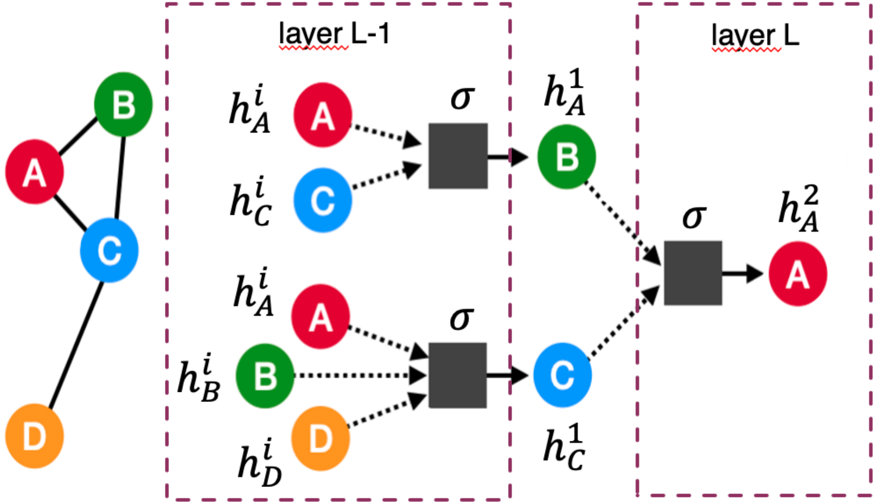
<u>Several task levels</u>, carried out by processing the final node embeddings in certain ways.

Graph Neural Networks (GNNs)

- The embeddings are updated at each layer by aggregating the information passed between the **target node** and the nodes from its closest neighbourhood → message passing



GNN schematic

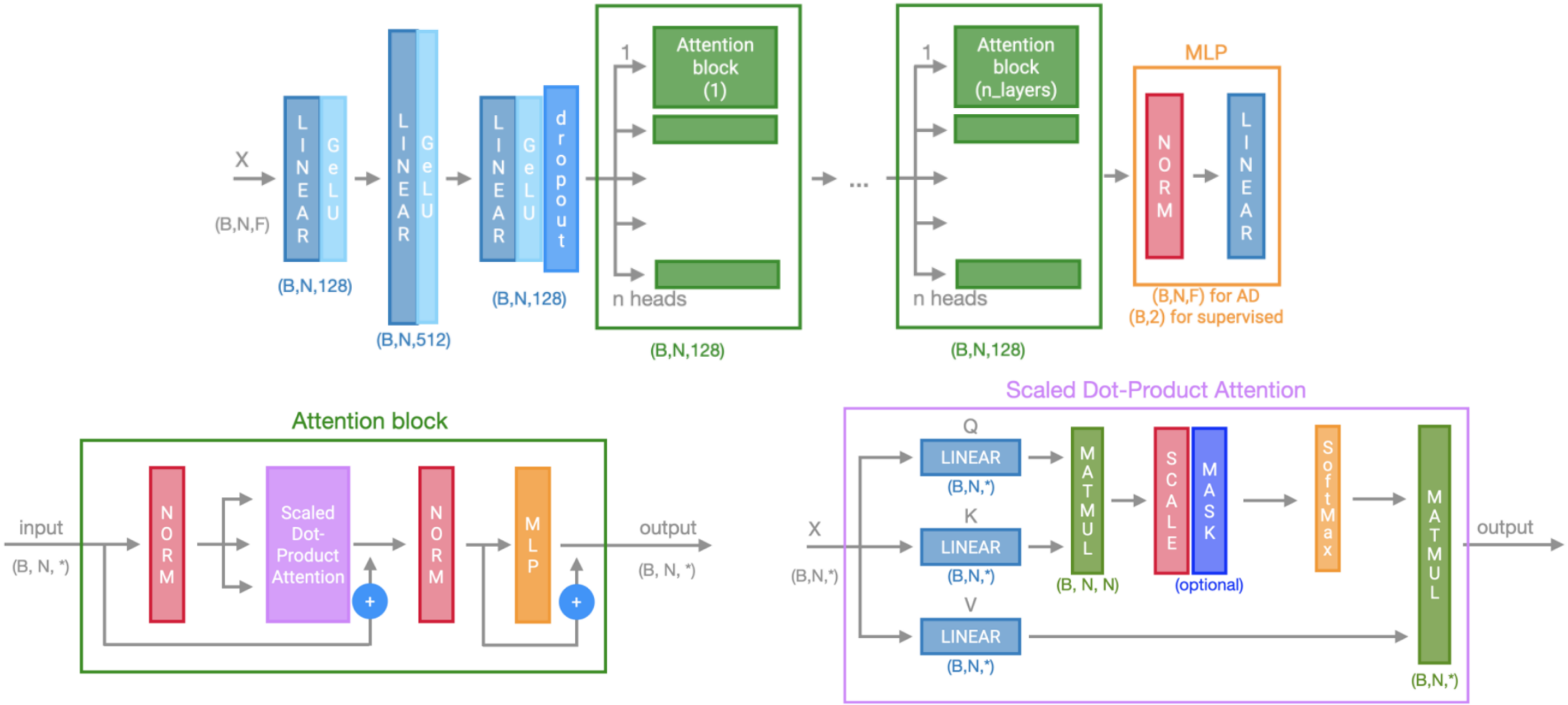


- Message passing can be extremely versatile, supporting also edge features updating at each epoch (Edge GNN) or Attention mechanism applied to messages (Graph Attention Network)

Transformer

- Deep Learning architectures that require classical vectorial input of size (B,N,F)
  - **Equivalent to fully connected graph input to GNN!**
- Based on Attention Mechanism, robust and handles graph structure internally

B = batch size
N = number of objects
F = number of features



Introduction: presentation roadmap

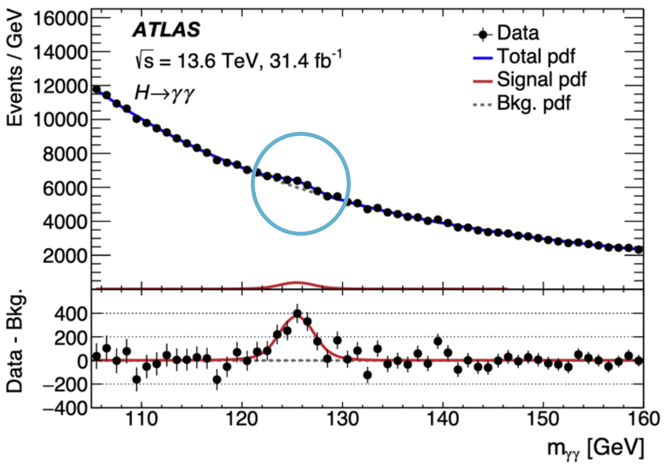
- The High Performance Computing (HPC) INFN Cluster: where the GPUs are
- The ML algorithms: what actually runs on them
- **ATLAS use-case: how it's done**
- Performances

ATLAS use case: Machine Learning in ATLAS

- Becoming more and more prominent in every context of our collaboration
  - Flavour tagging (GN2), searches (semi-visible jets), precision measuments (xsections), background estimation (YXH search), simulations (normalizing flow)

- Essential in model agnostic searches for new physics
  - Anomaly detection (Two-body invariant mass), weakly unsupervised (CWoLa)

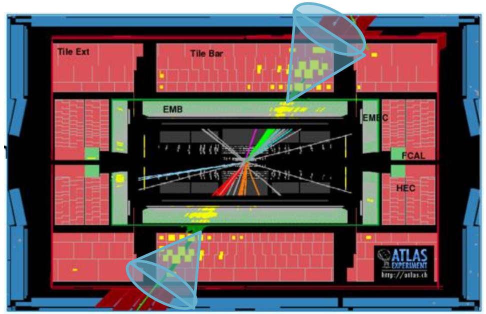
Eur.Phys.J.C 84 (2024) 1, 78



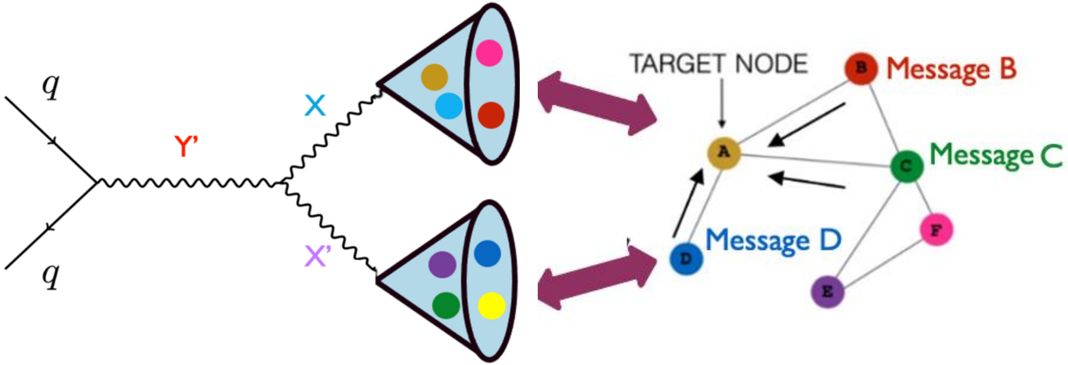
ATLAS use case: BSM resonance search

- Search for new heavy resonances decay into new particles in **fully hadronic** final states from pp collisions with **Anomaly Detection**
  - Final state quark hadronize in the hadronic calorimeter and jets are reconstructed from their energy deposits (constituents)

Hadronic calorimeter

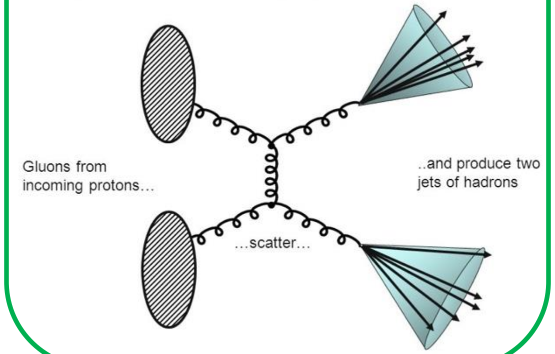


Targeted signal



Jets have sparse structure, suitable for graph representation

Expected background: QCD dijet



Interest: Y' mass (~ TeV) >> X – X' masses (~ O(10^2) GeV)

ATLAS use case: workflow

- 6x2 GPUs at disposal, access them by SLURM workload manager:
  - srun -w ibisco-gpu02 --partition=gpus --gpus-per-node=1 --cpus-per-task=10 --comment "JUPYTER" --pty bash –l
  - Device can then be selected at code level via NVIDIA CUDA API

- Jobs can be launched from terminal or using Jupyter notebook server instanced (--comment "JUPYTER")
  - Alternatively, jobs can be sent directly on a gpu by the command *sbatch* followed by a configured script

- Python environments where packages reside are built inside Miniconda Manager tool and activated

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
admin	up	infinite	2	down*	fair-gpu07,xc-gpu08
admin	up	infinite	1	mix	ibisco-gpu01
admin	up	infinite	5	idle	ibisco-gpu[02-06]
sequential*	up	7-00:00:00	2	down*	fair-gpu07,xc-gpu08
sequential*	up	7-00:00:00	1	mix	ibisco-gpu01
sequential*	up	7-00:00:00	5	idle	ibisco-gpu[02-06]
gpus	up	7-00:00:00	2	down*	fair-gpu07,xc-gpu08
gpus	up	7-00:00:00	1	mix	ibisco-gpu01
gpus	up	7-00:00:00	5	idle	ibisco-gpu[02-06]
parallel	up	7-00:00:00	2	down*	fair-gpu07,xc-gpu08
parallel	up	7-00:00:00	1	mix	ibisco-gpu01
parallel	up	7-00:00:00	5	idle	ibisco-gpu[02-06]
hparallel	up	1-00:00:00	1	mix	ibisco-gpu01
hparallel	up	1-00:00:00	5	idle	ibisco-gpu[02-06]

job.sh

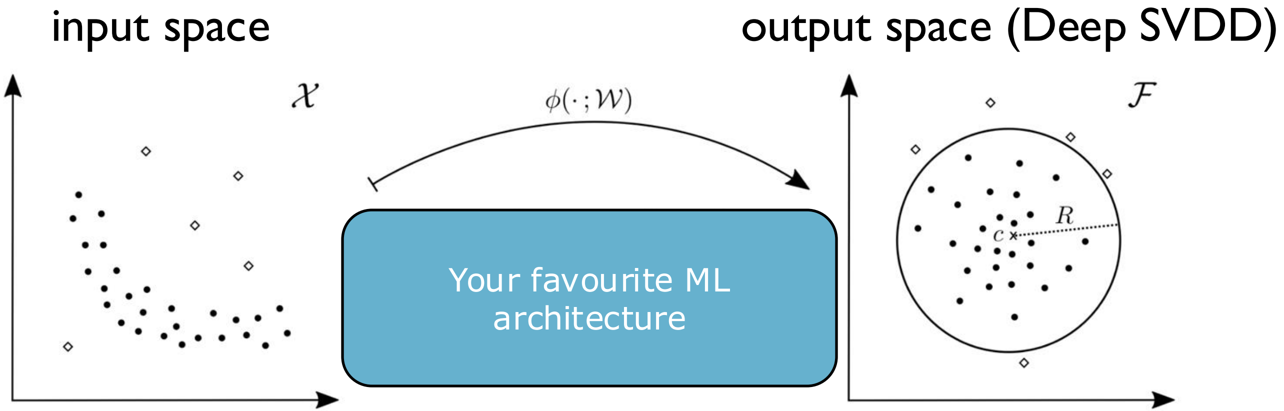
```
#!/bin/bash

#SBATCH --job-name=test_EGAT_mega
#SBATCH --output=test_EGAT_mega.txt
#SBATCH --partition=gpus
#SBATCH --account=ibisco
#SBATCH --nodelist=ibisco-gpu03
#SBATCH --gpus-per-node=2
#SBATCH --cpus-per-task=10
```

sbatch job.sh

ATLAS use case: Unsupervised Anomaly Detection

● = background
◇ = signal

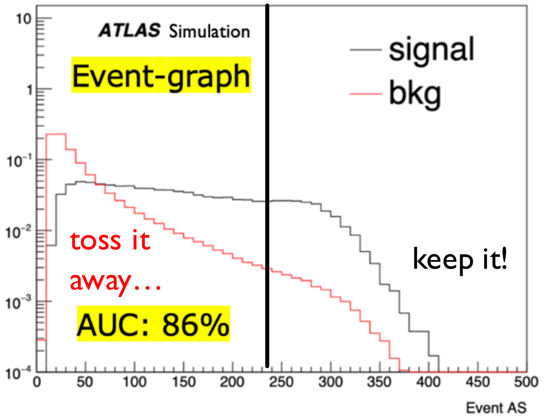


➢ **Key concept**: Unsupervised training on background (we can use a portion of data!)

➢ Network maps graph features from parameters space X → F by Deep Support Vector Data Description (SVDD) objective

$$\min_{W} \frac{1}{N}\sum_{i=1}^{N} \|\Phi(X,W) - c\|^2$$

➢ From prediction an Anomaly Score (AS) per event is derived



$$s(\boldsymbol{x}) = \|\phi(\boldsymbol{x};\mathcal{W}^*) - \mathbf{c}\|^2$$

Introduction: presentation roadmap

- The High Performance Computing (HPC) INFN Cluster: where the GPUs are
- The ML algorithms: what actually runs on them
- ATLAS use-case: how it’s done
- **Performances**

Performance: Graph dataset

- Training the ML algorithm requires first of all the datasets
  - **Transformer**: MonteCarlo ROOT ntuples, doesn’t require further processing
  - **GNN**: DGL Graph datasets, requires their creation from ROOT ntuples

- Graph dataset creation is very expensive computationally, and was optimized by parallelizing the job on multiple CPU cores through **Parallel** with "loky" backend (joblib python package)
  - Chunks of the initial ntuple are processed on the selected number of cores available
  - Can be set at will, but the ideal case is 1 chuck for each core

- We also parallelized on more nodes if available, using Parallel on multiple nodes

Signal sample (17k events)

# nodes	# chunks	Execution time
60	10	5 minutes
60	60	2 minutes
120	120	50 seconds

Background sample (434k events)

# nodes	# chunks	Execution time
60	60	30 minutes
120	120	15 minutes

Performance: GNN training

- Training the GNN architecture takes time and memory to store Pytorch tensors
  - Based on pre-message passing and post-message passing MLPs (Multi-Layer Perceptron) and EGAT (Edge Graph Attention Network)
  - Overall number of parameters: 3,894,942

Architecture	1 MLP (3 layers) → 5 layers GNN → 1 MLP (3 layers)
Loss	DeepSVDD
Input dimension	5
Output dimension	256
Dataset size	1.1 M (1M background : 100k signal)
Dataset split	Training: 20% (background only), Validation: 1%, test: 79%
Batch size	1024

Processor	RAM (GB)	RAM (%)	Time (s/epoch)
CPU	95	9	6200
GPU	22.880	70	200

Time factor (CPU/GPU)	CPU Occupation (%)
31	97

Performance: Transformer training

- Transformer architecture consists of multiple layers of MultiHead Attention blocks, introduced by MLP layer
  - Overall number of parameters: 16,190,400

Architecture	1 MLP (1 layer) → 6 layers MHA → 1 MLP (1 layer)
Loss	DeepSVDD
Input dimension	5
Output dimension	255
Dataset size	1.1 M (1M background : 100k signal)
Dataset split	Training: 30% (background only), Validation: 1%, test: 69%
Batch size	128

Processor	RAM (GB)	RAM (%)	Time (s/epoch)
CPU	15	1	51000
GPU	18.196	56	1350

Timefactor (CPU/GPU)	CPU Occupation (%)
37.7	110

Conclusions

- The HPC INFN cluster is a solid infrastructure which provides to those who need an environment to launch jobs

- ATLAS use-case: **Anomaly Detection with Graph Neural Networks in Run 3**
  - Workflow is optimized to expoit resources at best thanks to SLURM and Miniconda environments
  - ML algorithms making use of Transformers and Graph Neural Networks

- GPUs are a game changer, any job can be easily managed by ibisco nodes

- Improvements:
  - Easy: use both GPUs per node in training phase
  - Hard: use multiple GPUs from other nodes in training phase

Thank you for your attention!

BACKUP