GPU memory ressource management for Graph Neural Network (GNN) training on large graphs

14ème Journées Informatiques IN2P3/IRFU (14-17 novembre 2022)

Sylvain Caillou

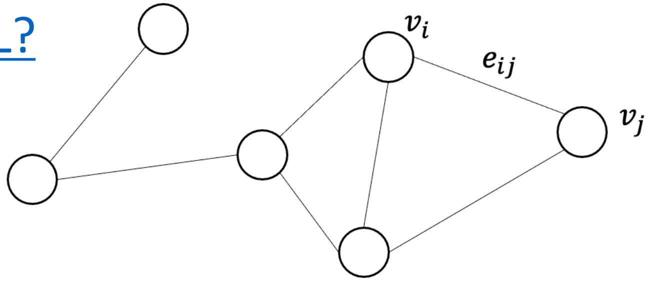
(On behalf of L2IT « Computing, Algorithms and Data » Team)

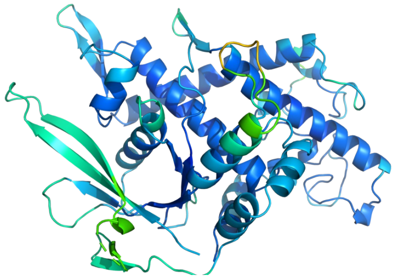
Laboratoire des 2 Infinis - Toulouse (L2IT-IN2P3), Université de Toulouse, CNRS, UPS

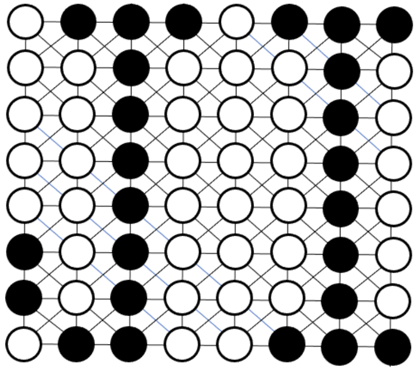
The rise of geometric ML and representation learning

⇒ Geometric and graph-based ML methods have become one of the hottest fields of AI research
⇒ Graph Neural Networks (GNNs) capture deep geometric and structural patterns in data represented as graph

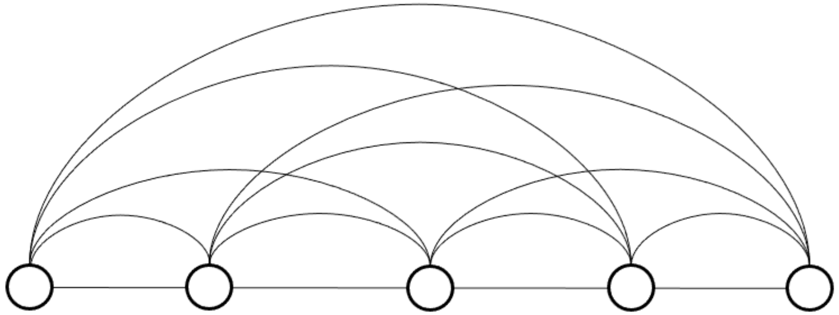
[What does 2022 hold for Geometric & Graph ML?](#)
Michael Bronstein







CNNs can be seen as a specific use case of GNN on regular grid graph



« Graphs » « represents » « relations » « between » « entities »

Transformers architectures in Natural Language Processing operate on fully connected graph

In 2021 triumph of Geometric ML and a paradigm shift in structural biology: Breakthrough in prediction of the 3D folding structure of a protein by AlphaFold 2 (deepmind)

Graph-based ML in HEP

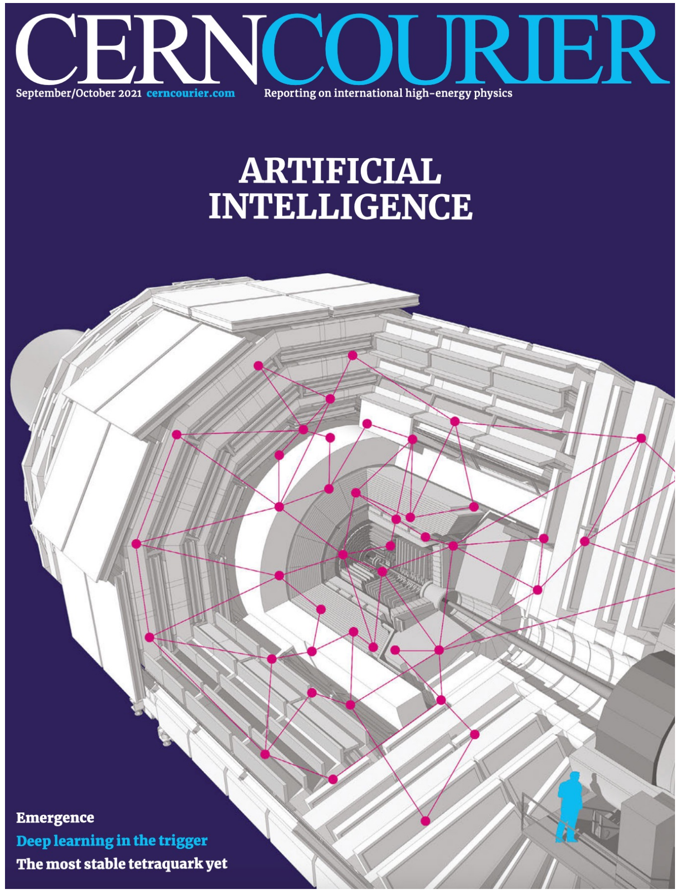
Since 2020 Graph-based ML algorithms are becoming increasingly popular for a large number of LHC physics tasks, including:

- ➢ Track Reconstruction
- ➢ Vertex Reconstruction
- ➢ Calorimeter Clustering
- ➢ Jet Clustering
- ➢ Event Reconstruction (Pileup Rejection, Particle Flow, Jet Assignment)

Now working to integrate Graph ML in production for in-line and off-line data processing systems

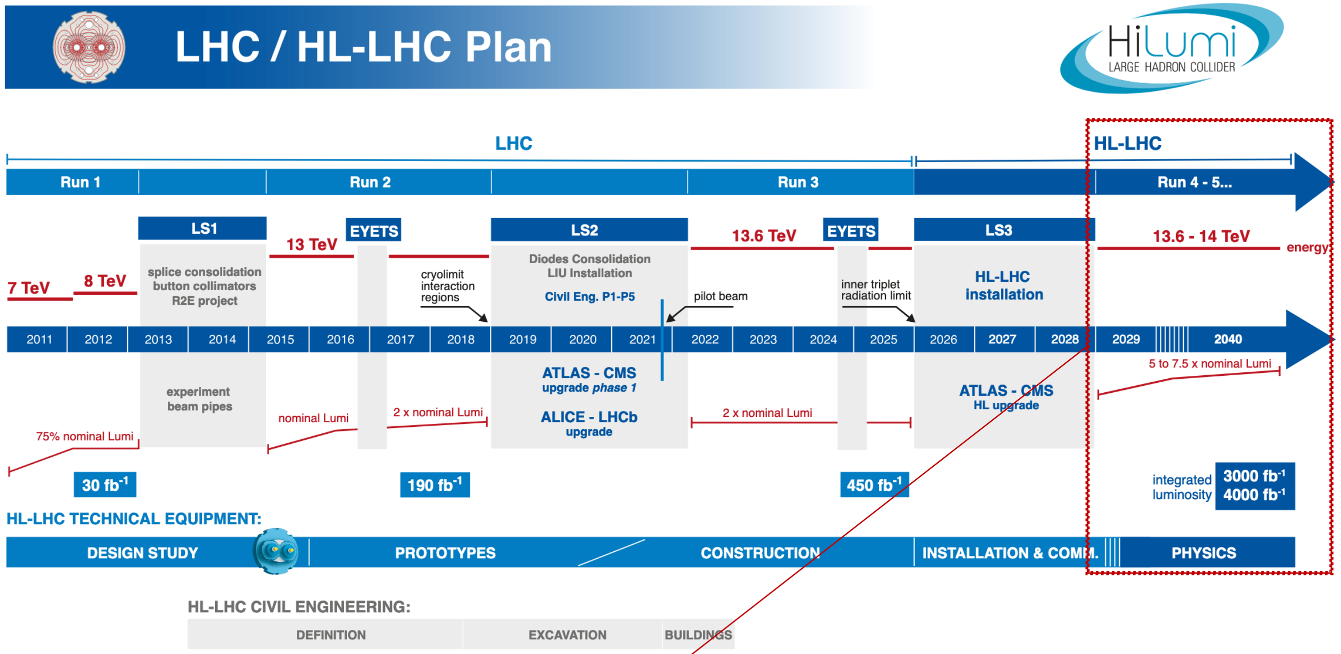
$\Rightarrow$ Starting to be use in Astrophysics, cosmology, nuclear physics, …

$\Rightarrow$ *Nature of physic experimental data brings to large graph representation and computing challenges for Graph-based ML algorithms*



September/October 2021

HL-LHC and unprecedented challenges for software and computing



⇒ In 2028, the rate of collisions provided by the LHC to the ATLAS detector will be increased by an order of magnitude.

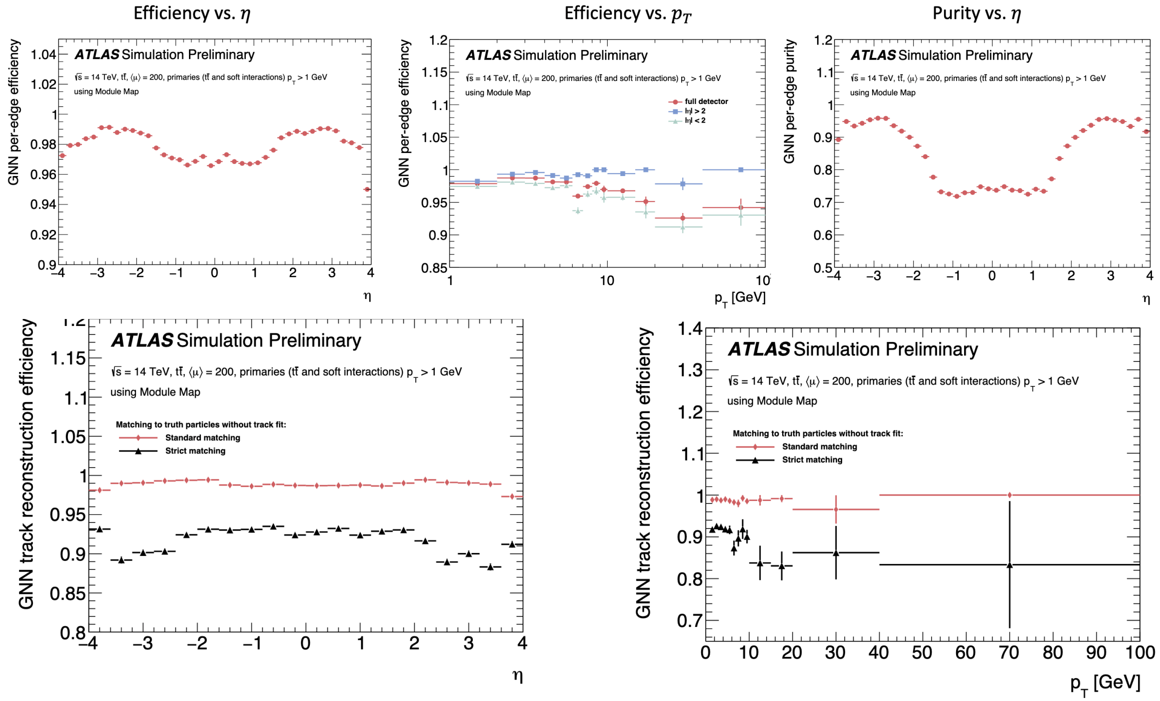
⇒ Existing tracks reconstruction algorithms based on *Kalman Filter* won't fit the combinatorics and will require unrealistic amount of CPU ressources

⇒ **The HL-LHC physics program could not be completed** (obviously not an option !)

GNN for charged particles tracking in ATLAS ITk

Graph Neural Networks (GNNs) perform pretty well to learn geometric pattern of the tracks

Close collaboration L2IT & ExatTrkX started in 2021 to construct a GNN-based track reconstruction algorithm for ATLAS ITk



First results on ITk published in 2022 more than encouraging

[ATLAS ITk Track Reconstruction with a GNN-based pipeline, C.Rougier et al., CTD 2022](#)
[Graph Neural Network track reconstruction for ATLAS ITk , D. Murnane et al., IML 2022](#)

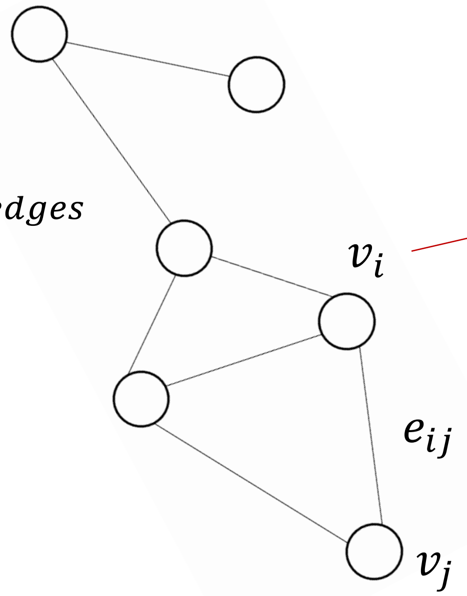
⇒ GNN-based algorithms now appear as competitive solutions for the future generation of charged particle track reconstruction algorithms which will have to be put into production for the HL-LHC

⇒ But to get this results we have to train our GNNs on very large graphs…

Data representation as graph

A graph represents the relations (*edges*) between a collection of entities (*nodes*).

$$G=(V,E), |V| = N_{nodes}, |E| = N_{edges}$$
$$V=\{v_i\}_{i\in\{0,\dots,N_{nodes}-1\}}$$
$$E=\left\{e_{ij}=(v_i,v_j)\right\}_{i,j\in\{0,\dots,N_{nodes}-1\}}$$



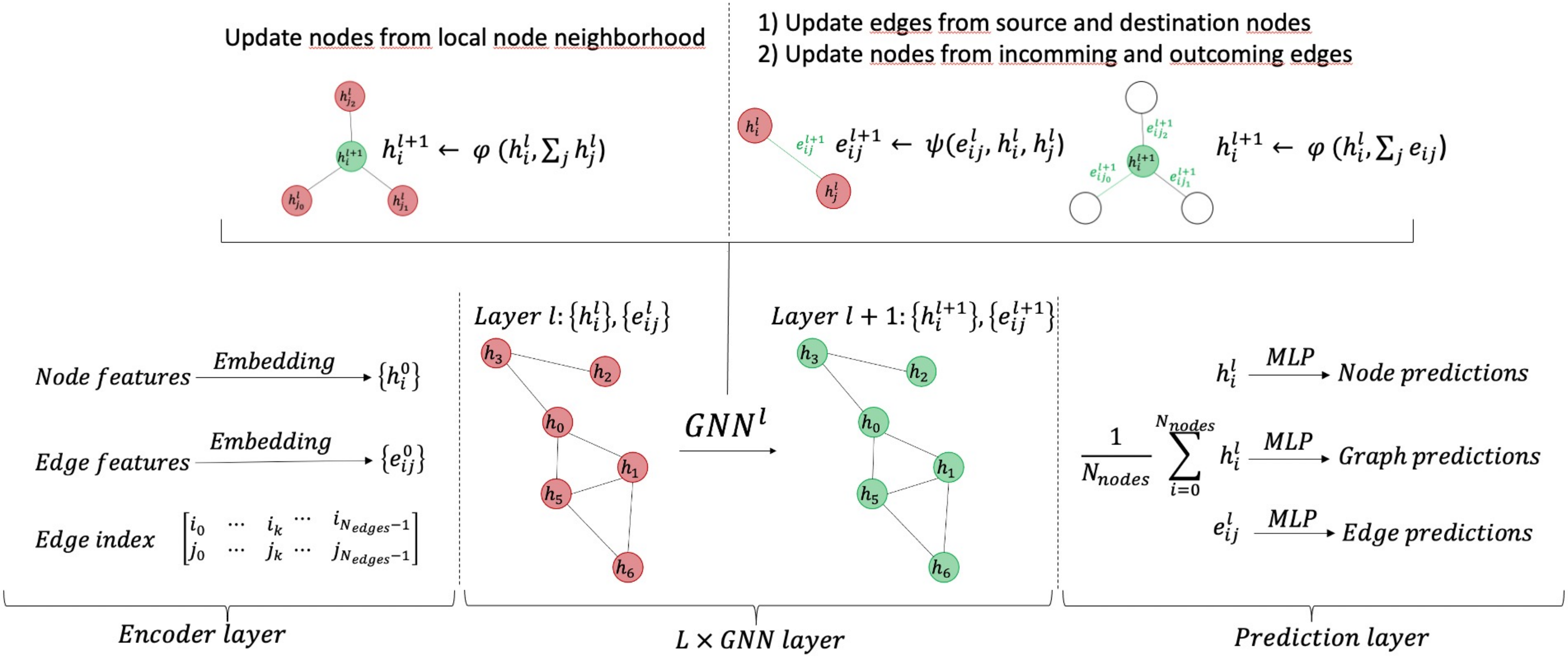
$$nodes\ features\ (N_{nodes}\times N_{nodes\_features})$$
$$\begin{bmatrix} \alpha_0^0 & \dots & \alpha_0^{N_{nodes\_features}-1} \\ \vdots & \dots & \vdots \\ \boxed{\alpha_i^0} & \boxed{\dots} & \boxed{\alpha_i^{N_{nodes\_features}-1}} \\ \vdots & \dots & \vdots \\ \alpha_{N_{nodes}-1}^0 & \dots & \alpha_{N_{nodes}-1}^{N_{nodes\_features}-1} \end{bmatrix}$$

$$edges\ features\ (N_{edges}\times N_{edges\_features})[optional]$$
$$\begin{bmatrix} \beta_0^0 & \dots & \beta_0^{N_{edges\_features}-1} \\ \vdots & & \vdots \\ \boxed{\beta_k^0} & \boxed{\dots} & \boxed{\beta_k^{N_{edges\_features}-1}} \\ \vdots & & \vdots \\ \beta_{N_{edges}-1}^0 & \dots & \beta_{N_{edges}-1}^{N_{edges\_features}-1} \end{bmatrix}$$

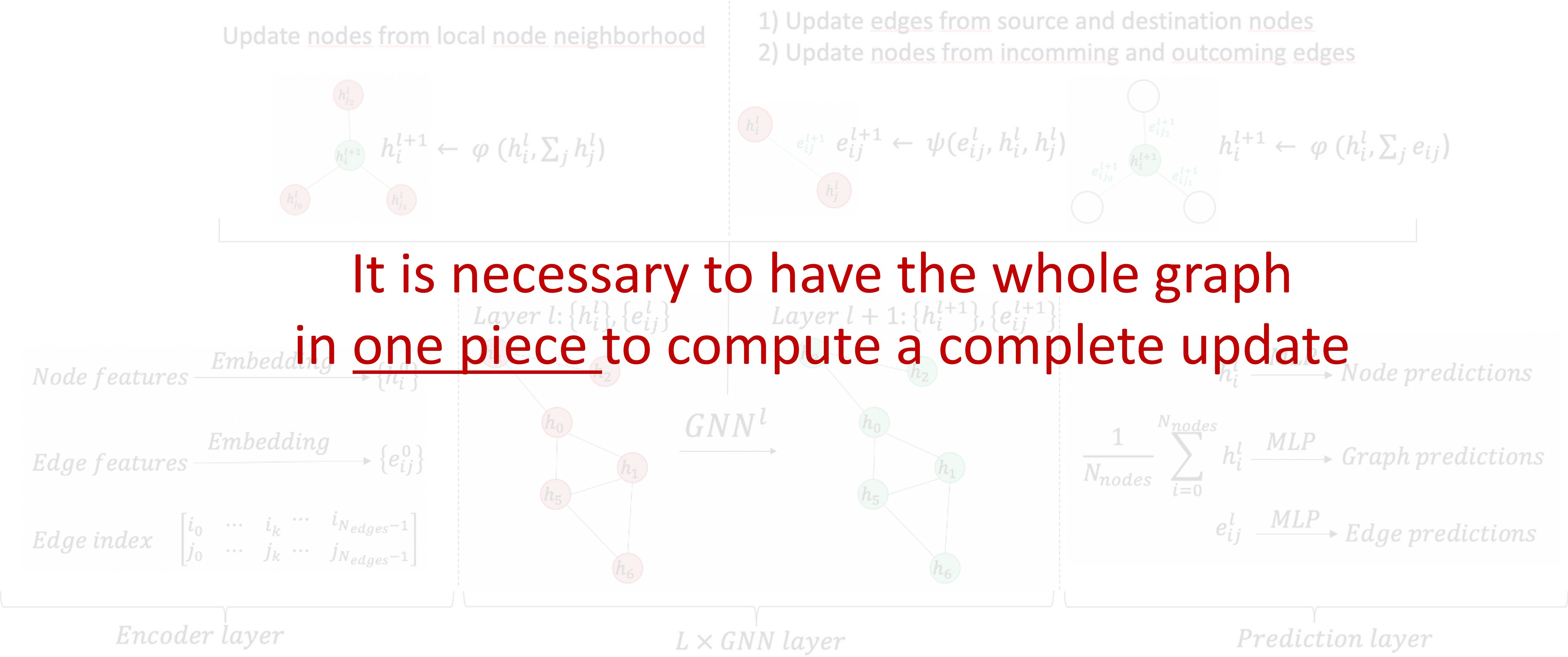
$$edge\ index\ (2\times N_{edges})$$
$$\begin{bmatrix} i_0 & \cdots & i_k & \cdots & i_{N_{edges}-1} \\ j_0 & \cdots & j_k & \cdots & j_{N_{edges}-1} \end{bmatrix} \forall e_{ij}=(v_i,v_j)\in E$$

Graph connectivity is represented as an Adjency matrix or in COO format

Graph Neural Networks models (Message Passing paradigm)

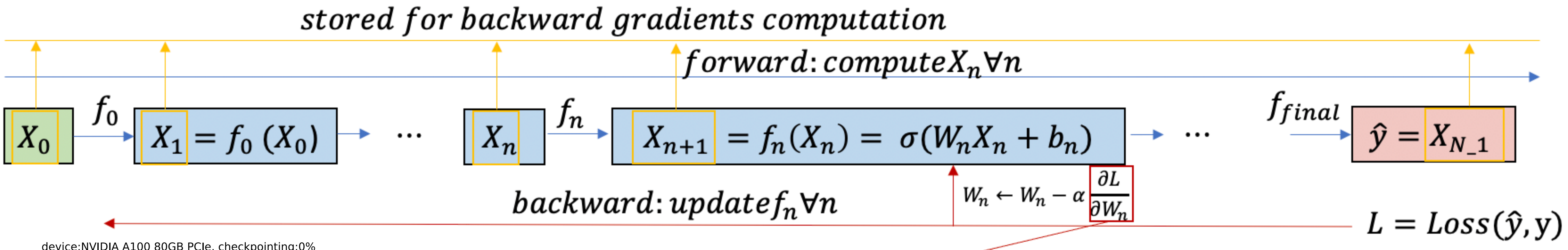


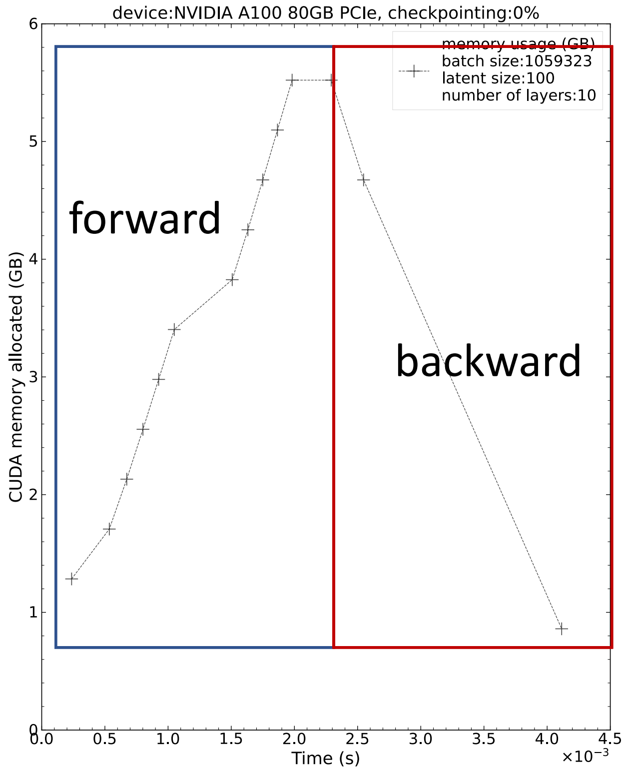
Graph Neural Networks models (Message Passing paradigm)



It is necessary to have the whole graph in <u>one piece</u> to compute a complete update

Neural Networks training and memory consumption





$$\frac{\partial L}{\partial W_n} = \underbrace{\frac{\partial L}{\partial X_{n+1}}}_{\text{upstream gradient}} \underbrace{\frac{\partial X_{n+1}}{\partial W_n}}_{\text{local gradient}} \rightarrow \frac{\partial \sigma(W_n X_n + b_n)}{\partial W_n} = \boldsymbol{X_n}$$

$\Rightarrow$ Store inputs for each layers as hidden states during forward pass

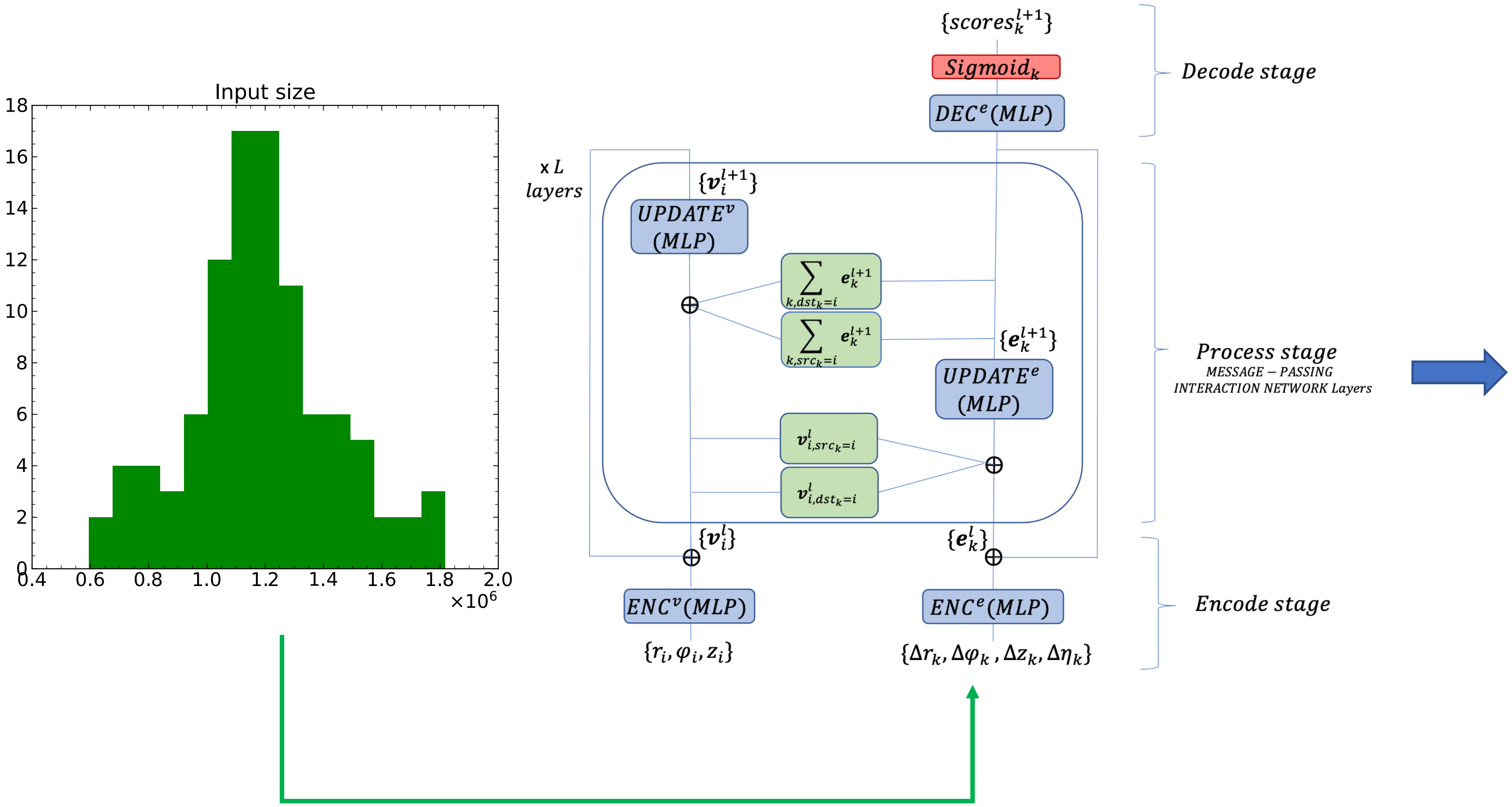
$\Rightarrow$ Consume these stored states during backward pass to compute the gradient

X_n: $input\ size \times latent\ size$ get huge in memory for large input size and/or latent size

$$total\ storage = input\ size \times latent\ size \times numbers\ of\ layers$$

Example : (2e6 x 100 x 100 x 4)/1e9 => O(80 GB)

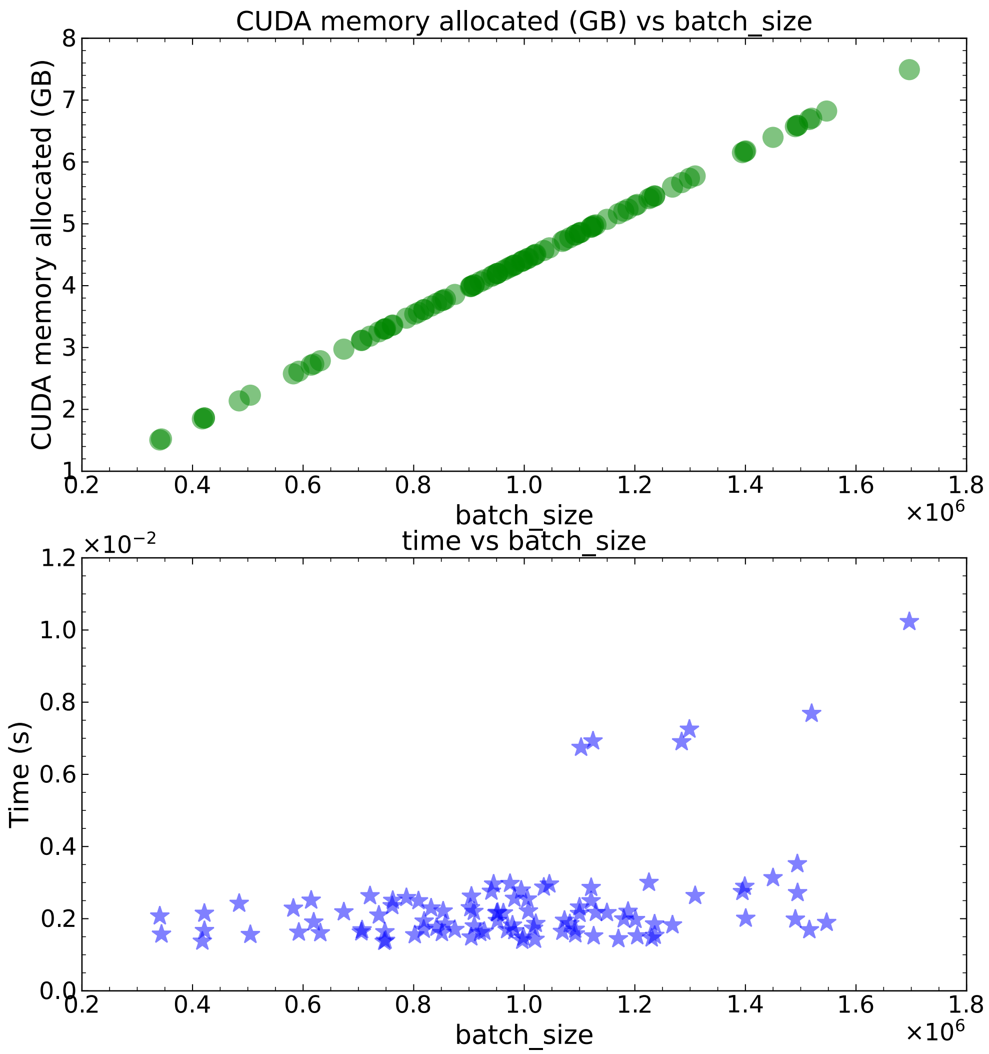
The never ending battle of the GPU memory consumption



Memory consumption

$O(10^2 \text{ GB})$

Neural Networks training and memory consumption

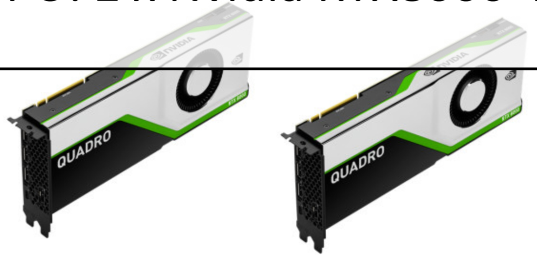


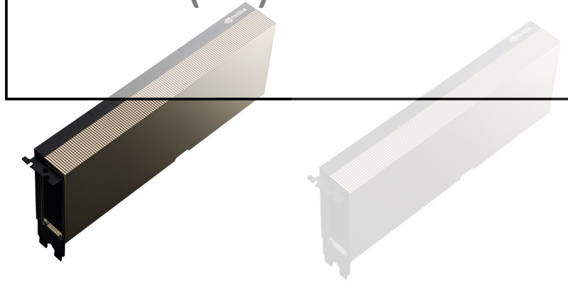
Specific configuration CC support

Host CPU: 1 TB RAM
GPU: 2 x Nvidia RTX8000 48 GB

Host CPU: 500 GB RAM
GPU: 1(+1) x Nvidia A100 80 GB

Host CPU: 500 GB RAM
GPU: 2 x Nvidia A100 80 GB

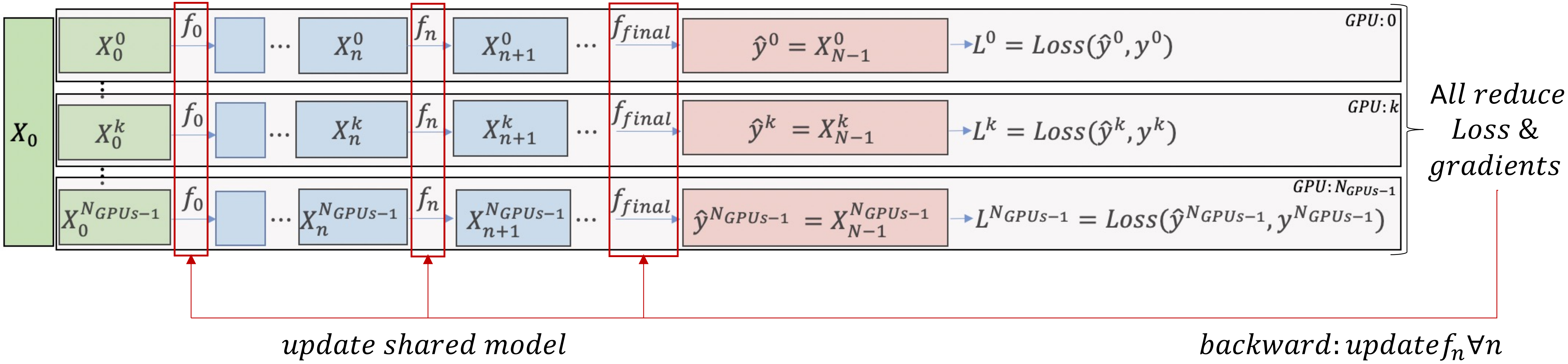




Special thanks to:
Benoît DELAUNAY
Vanessa HAMAR
Xiaomei NIU
Bertrand RIGAUX
…

Distributed Data Parralel

$\Rightarrow$ Distribute data
$\Rightarrow$ Share model
$\Rightarrow$ Computation operate on distributed data between GPUs with copy of the same model (same weights)

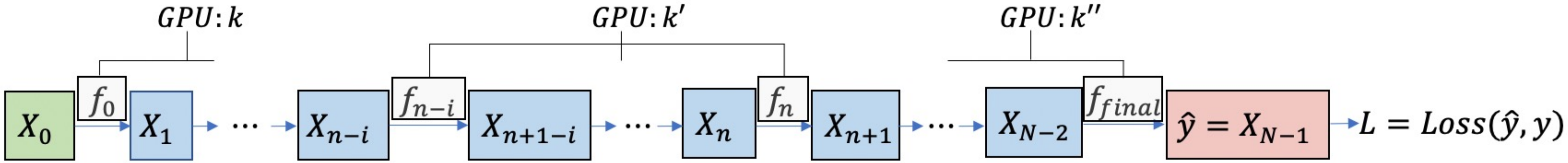


$\Rightarrow$ Usefull to distribute inputs between GPUs
$\Rightarrow$ Can be use for GNN models if the graphs are small (i.e. X_0 contains several small graphs)
If the graphs are large (i.e. X_0 contains only one graph) and it can not be cut

$\Rightarrow$ So Distributed Data Parralel approach is not a solution in case of large graph which won't fit in only one GPU

Distributed Model Parralel

$\Rightarrow$ Distribute model
$\Rightarrow$ Share data
$\Rightarrow$ Computation of share data is distributed between GPUs: Each GPU compute a part of the model

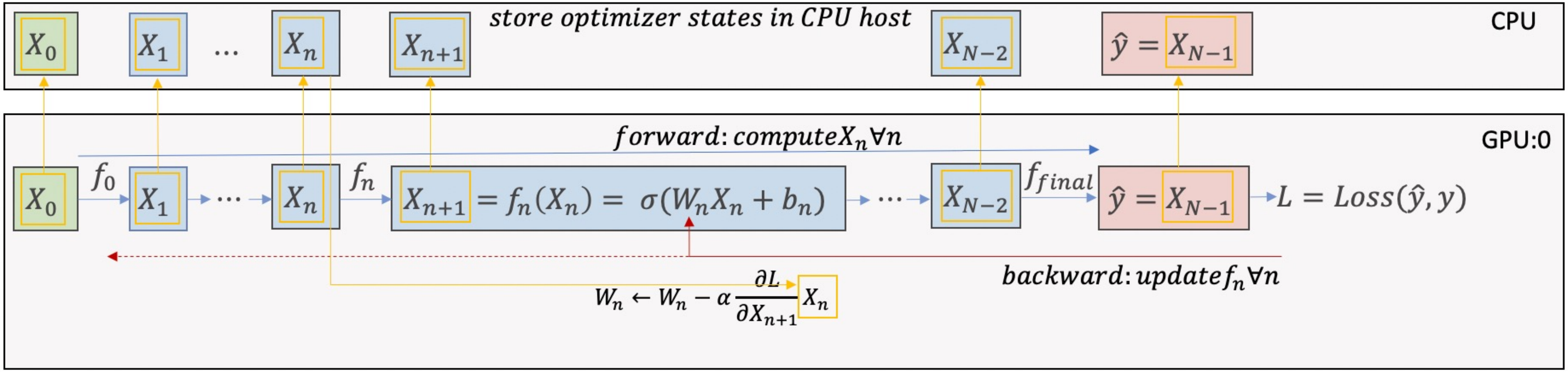


Usefull for models with a very large number of parameters (language model actually up to $O(10^{12})$ parameters)
$\Rightarrow$ Distribute between GPUs:
- $\Rightarrow$ Memory to store model parameters
- $\Rightarrow$ Computation time

$\Rightarrow$ Useless to solve GNN GPU memory consumption on large graph (whatever the number of model parameters)

Optimizer states and gradient offload

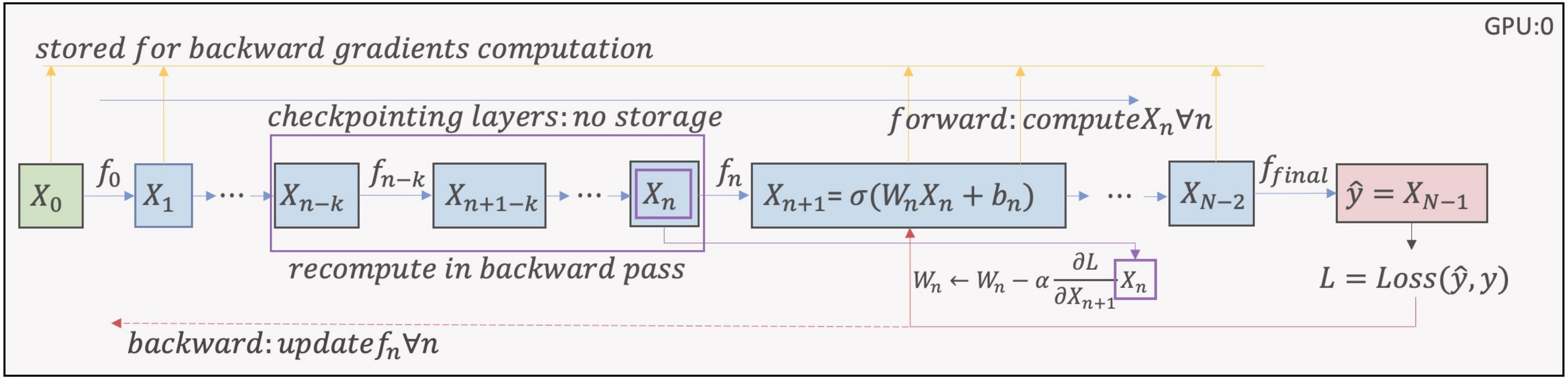
$\Rightarrow$ Swap on CPU all memory wich is not immediatly needed for computation
$\Rightarrow$ Swap back from CPU to GPU tensors wen they are needed



IBM: [Tensorflow Large Model Support (TFLMS)] and [Pytorch Large Model Support (PLMS)]
DeepSpeed: [ZeRO-Offload]

Pros: Allows to train very large model on large data
Cons: Slow down (a lot) the training beacause of I/O swapping between CPU host and GPU

Checkpointing



⇒ Checkpoint layers during forward pass

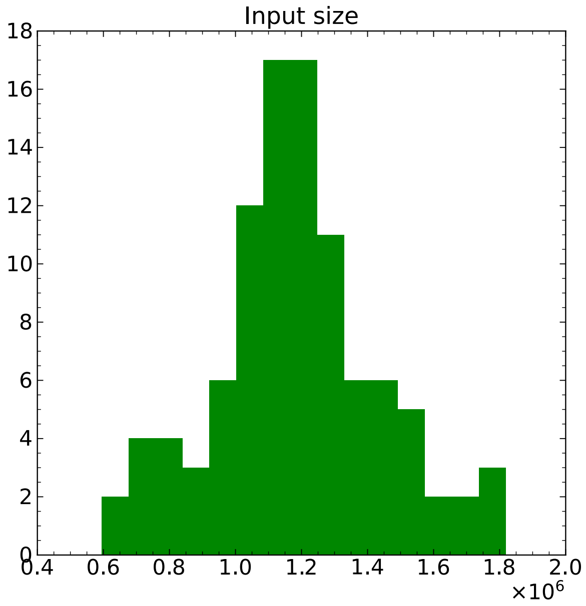
⇒ Intermediate states of checkpointed layers won't be store

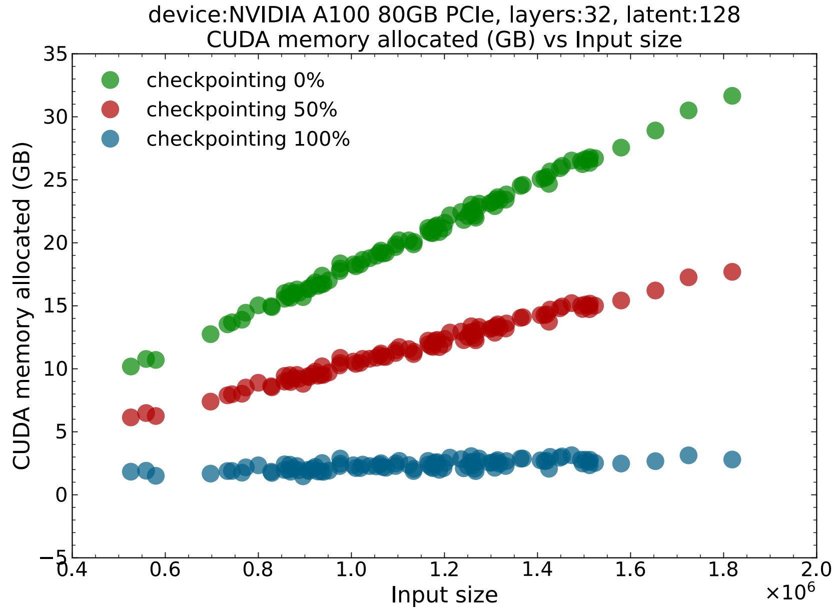
⇒ Intermediate states will have to be recompute during backward pass

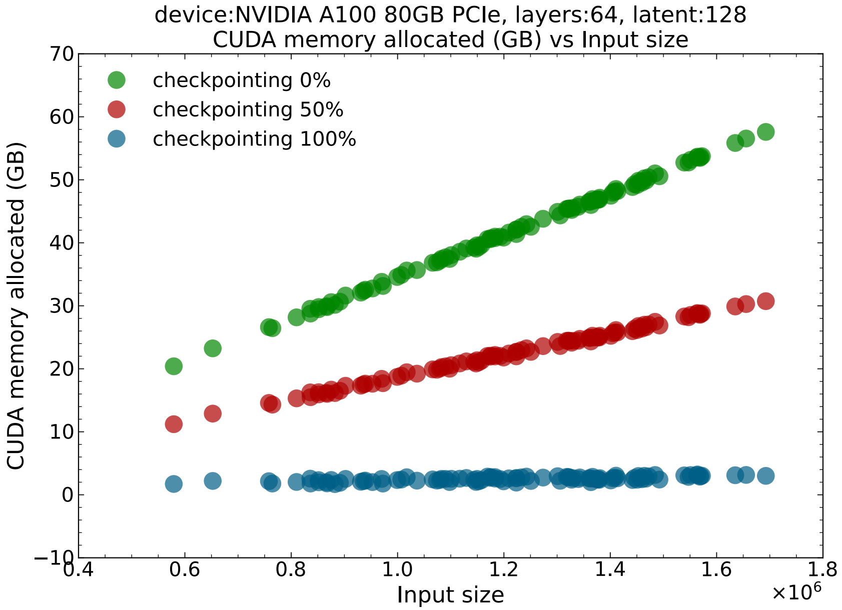
Pros: Save memory (depends of level of checkpointing)

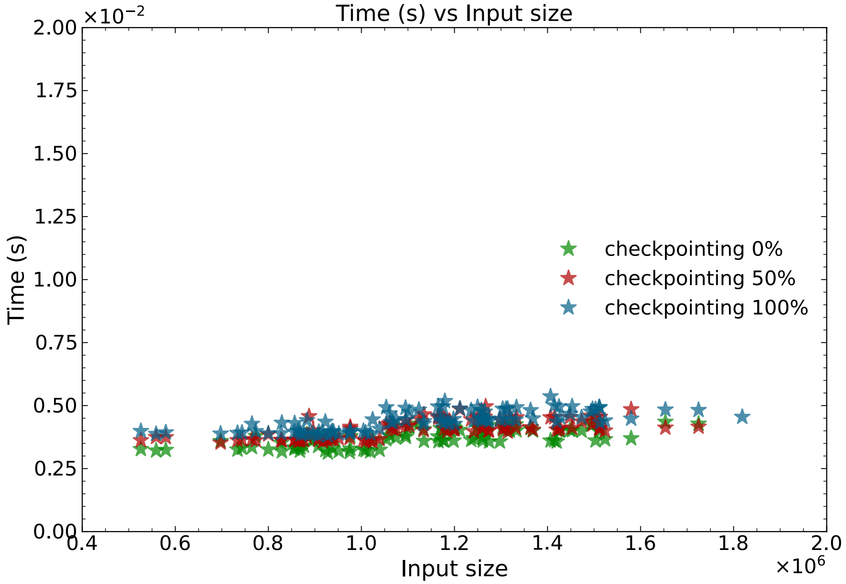
Cons: Come with a cost in time

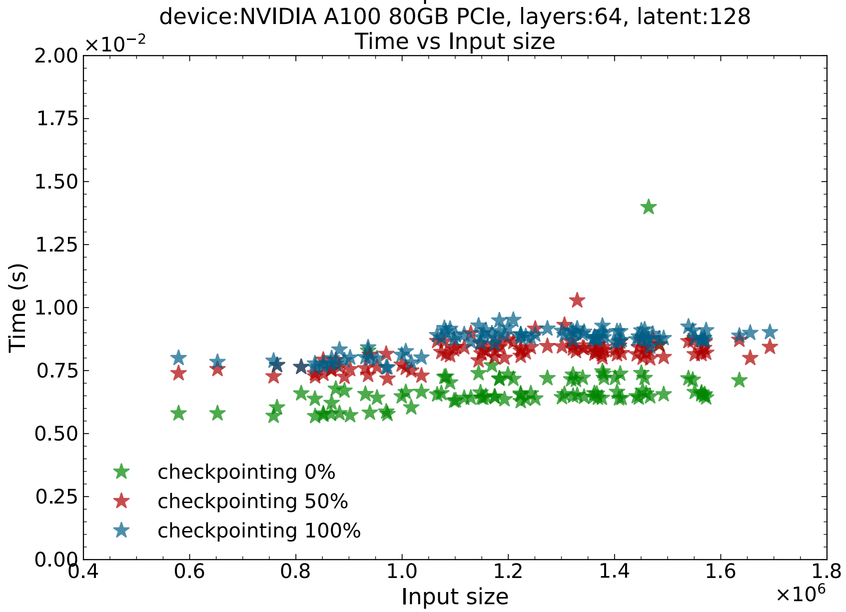
Checkpointing



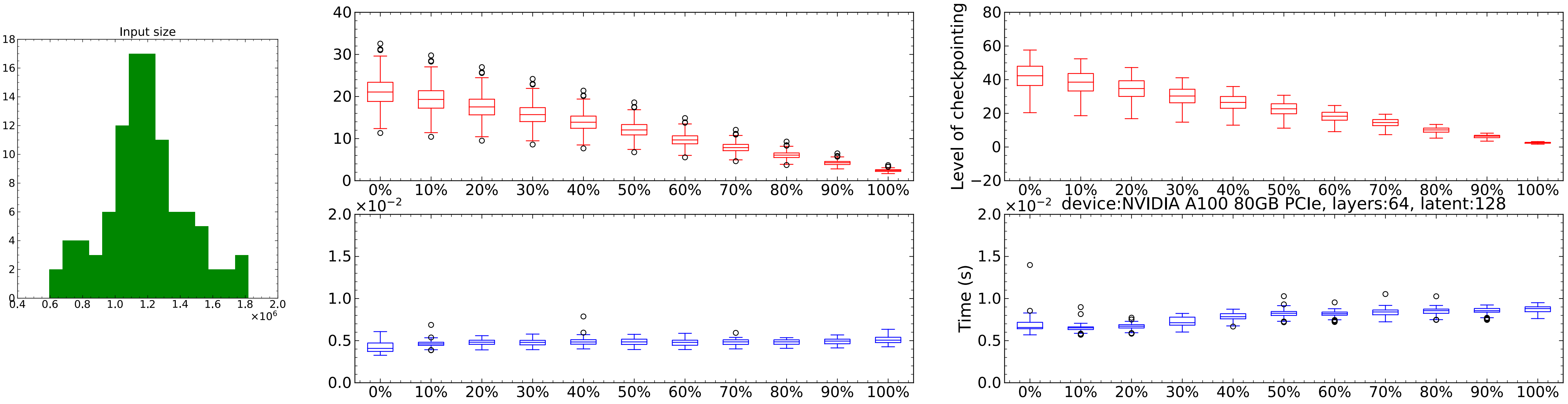








Checkpointing



What about sobriety ?

Applying checkpointing (in our use case)

$\Rightarrow$We divide by O(10) the time for training

Merci de votre attention

Backup

100 layers…