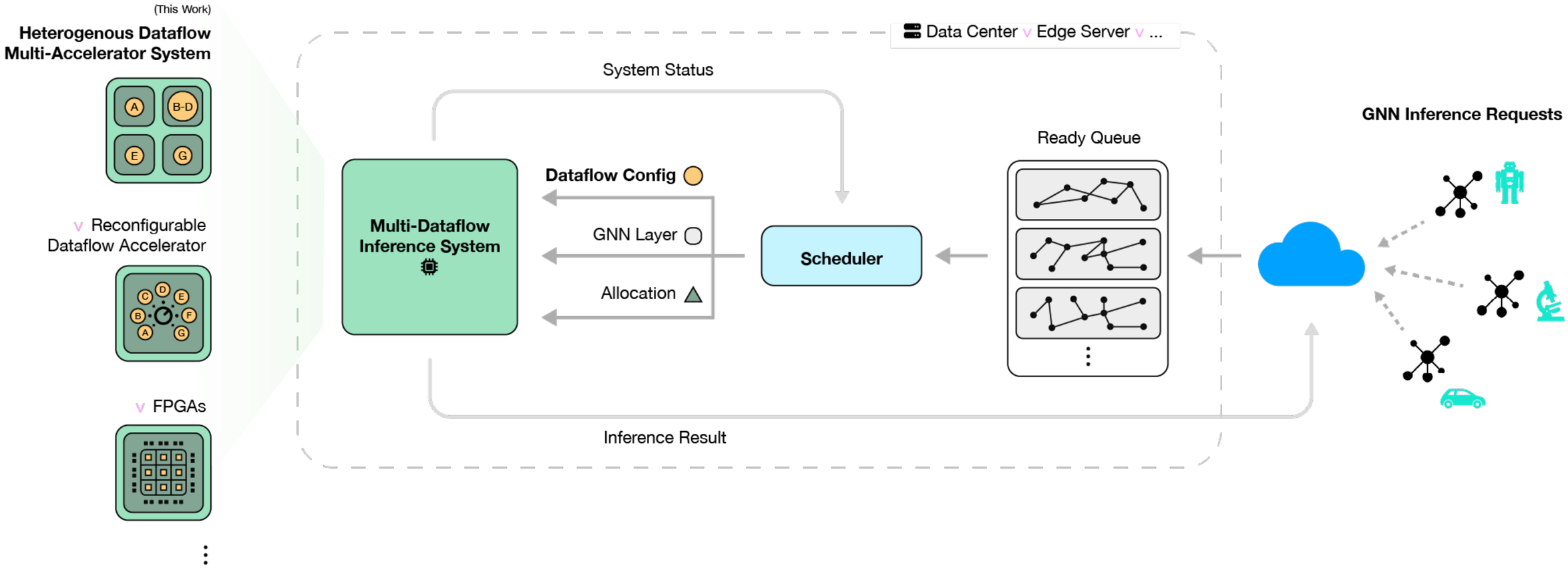
A Data-Driven Approach to Dataflow-Aware Online Scheduling for Graph Neural Network Inferences

Pol Puidgemont[1], Enrico Russo[2], Axel Wassington[1], Abhijit Das[1], Sergi Abadal[1], Maurizio Palesi[2]

[1] Universitat Politècnica de Catalunya, Spain
[2] University of Catania, Italy

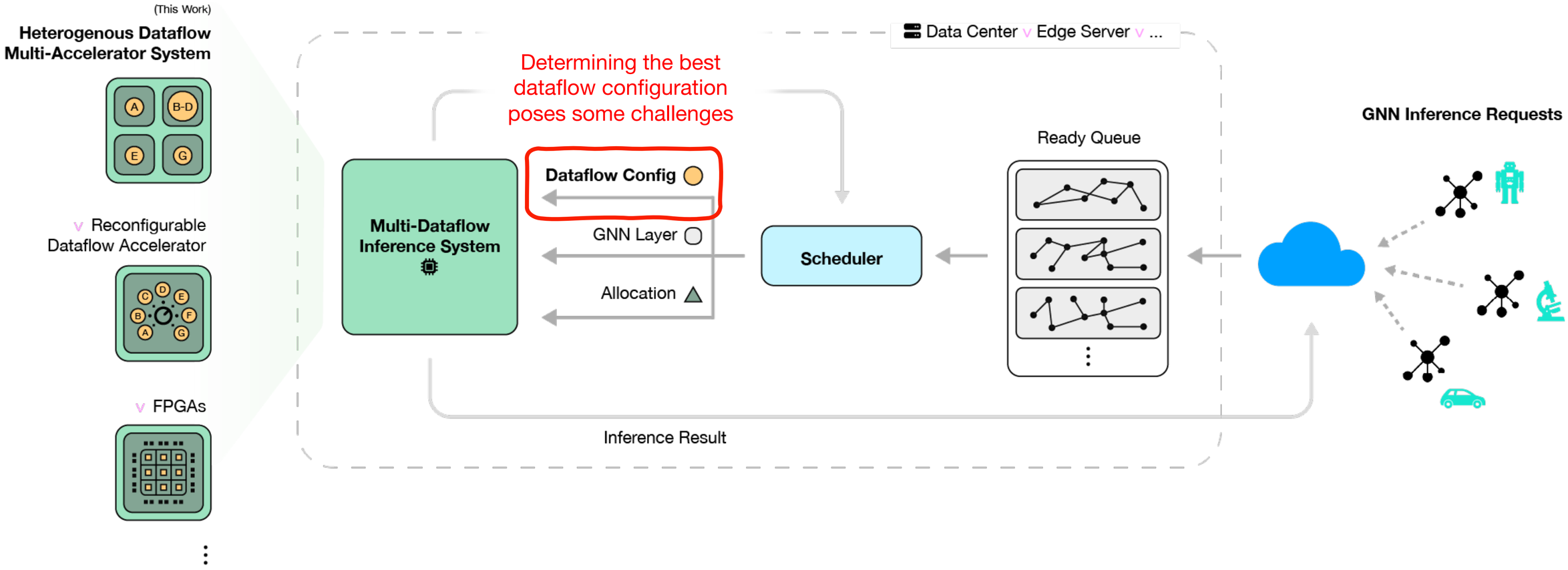
Overview

Efficiently running **Graph Neural Networks inferences** in online, real-time, or streaming contexts requires optimal mapping and dataflow configurations, which must adapt dynamically to the changing input graph.



Overview

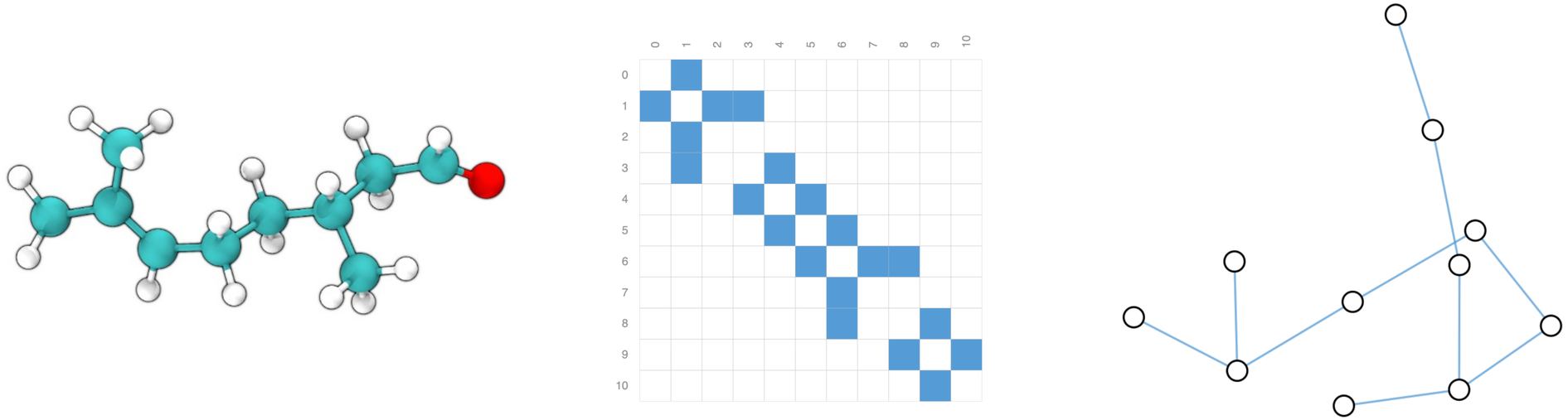
Efficiently running **Graph Neural Networks inferences** in online, real-time, or streaming contexts requires optimal mapping and dataflow configurations, which must adapt dynamically to the changing input graph.



Background

Graph Neural Networks

Graph Neural Networks (GNNs) are a specialized type of neural network designed to process and analyze **graph-structured data**, which consists of **nodes** (data points) and **edges** (relationships between points).



From left to right. Citronella molecule. Adiacency matrix representation. Graph representation.
From https://distill.pub/2021/gnn-intro/

Background

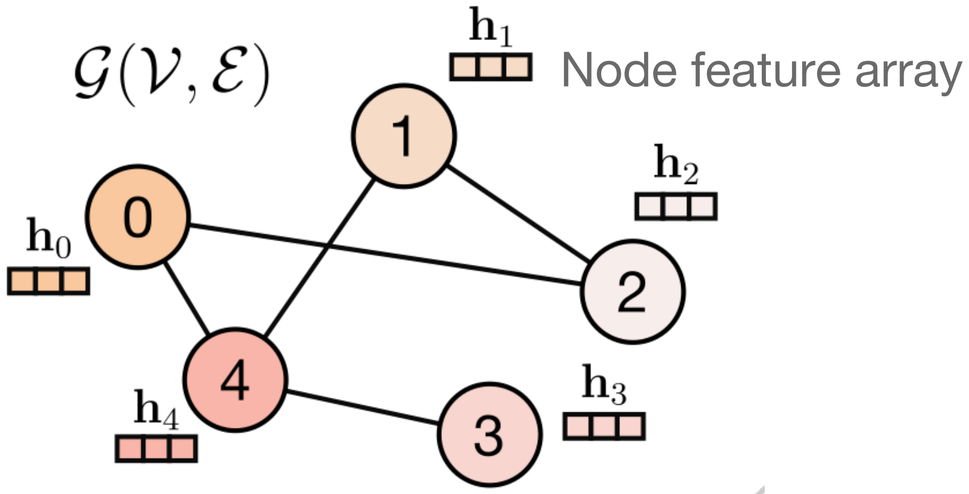
Graph Neural Networks

GNNs are increasingly employed in a wide range of applications: **social network analysis**, **recommender systems**, **fraud detection**, **transport networks** and **logistics**, **smart energy grids**, **cybersecurity**, **drug discovery**, etc.

Background

Graph Neural Networks

A Graph Neural Network (GNN) is a parametrized transformation of graph attributes designed to preserve permutation invariance and graph symmetries. In our case, the input graph is represented by an adjacency matrix and real-valued feature arrays for the nodes, as illustrated below:



1	0	1	0	1
0	1	1	0	1
1	1	1	0	0
0	0	0	1	1
1	1	0	1	1

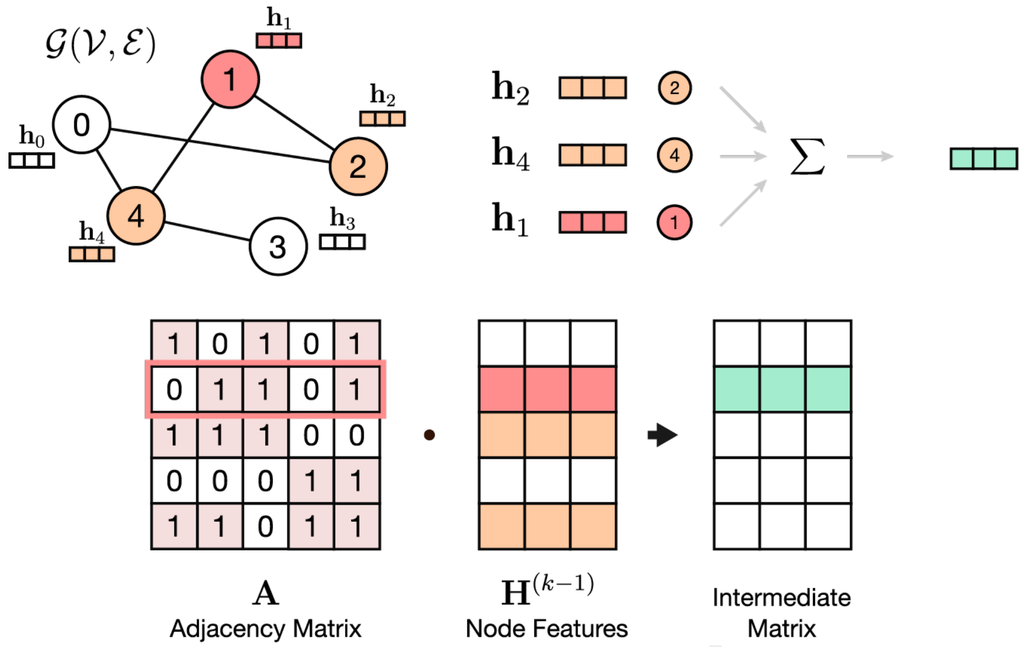
$\mathbf{A}$

Adjacency Matrix

Background

Graph Neural Networks

A commonly adopted transformation in each layer (e.g. in Graph Convolutional Networks) consists in a two-phase procedure applied node-wise.

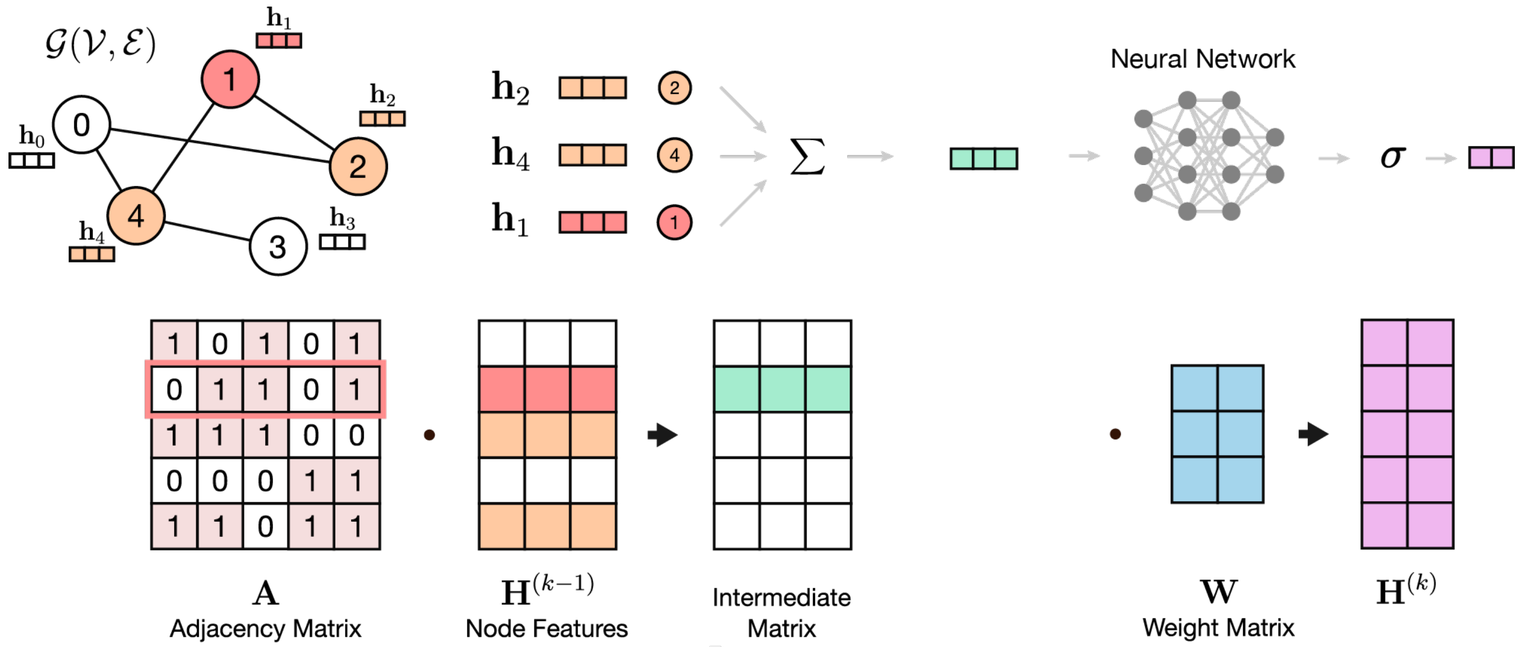


The first phase is **aggregation** consist in summing the features of the adjacent nodes.

Background

Graph Neural Networks

A commonly adopted transformation in each layer (e.g. in Graph Convolutional Networks) consists in a two-phase procedure applied node-wise.



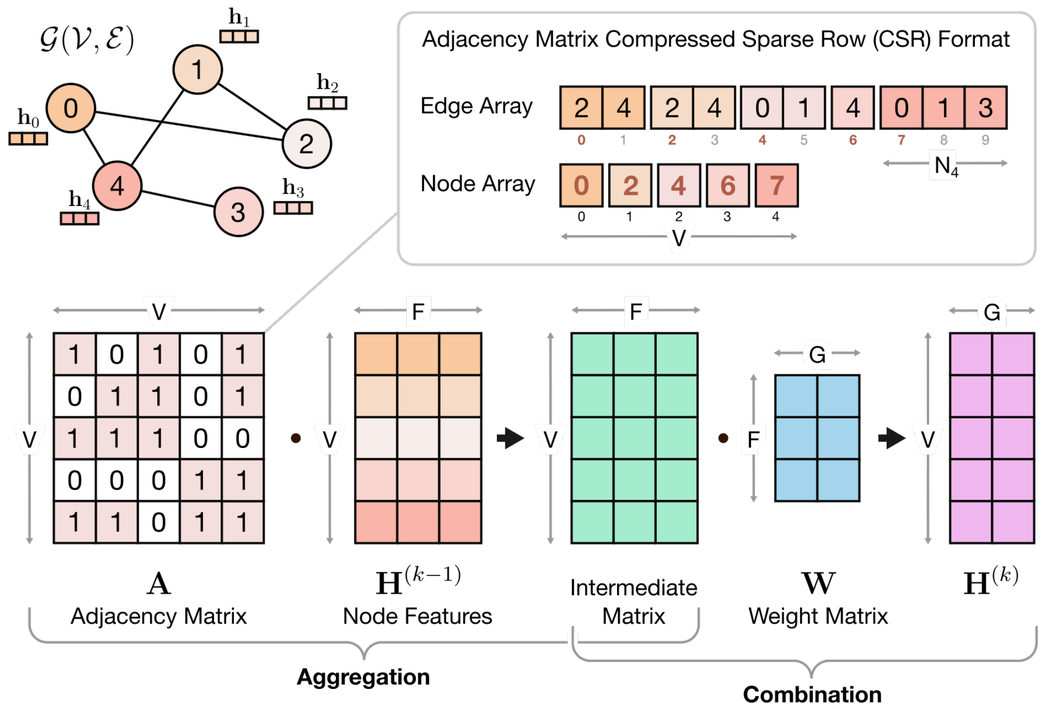
The first phase is **aggregation** consist in summing the features of the adjacent nodes.

The second phase is **combination** and consist in a fully connected layer.

Background

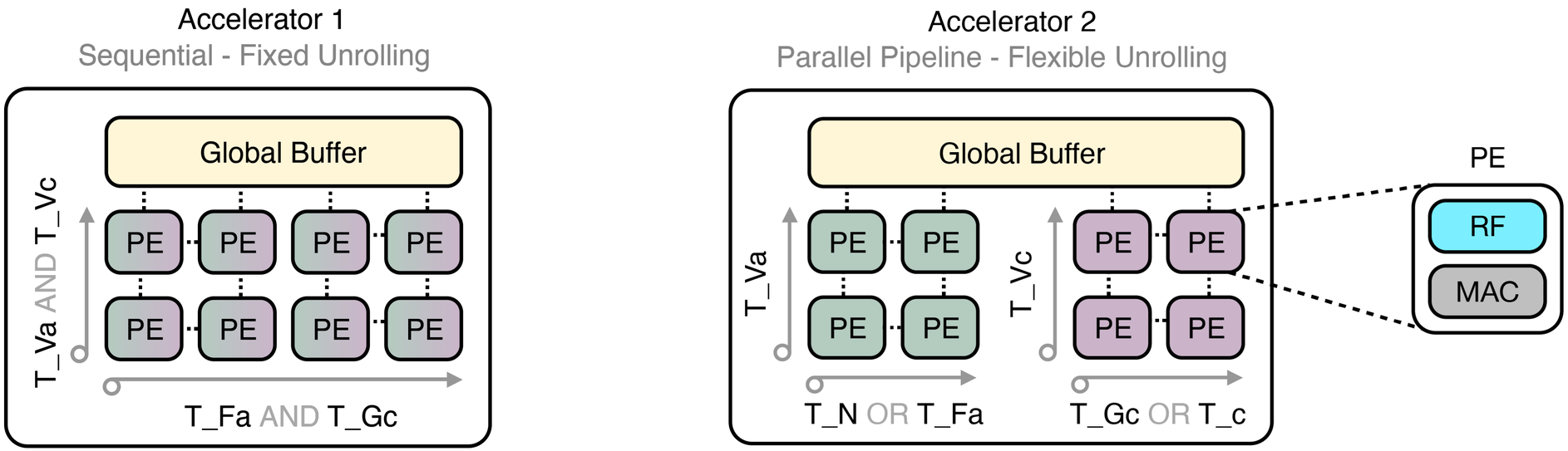
Graph Neural Networks

The computation for many GNN models can be expressed as a dense-sparse matrix multiplication:



Background

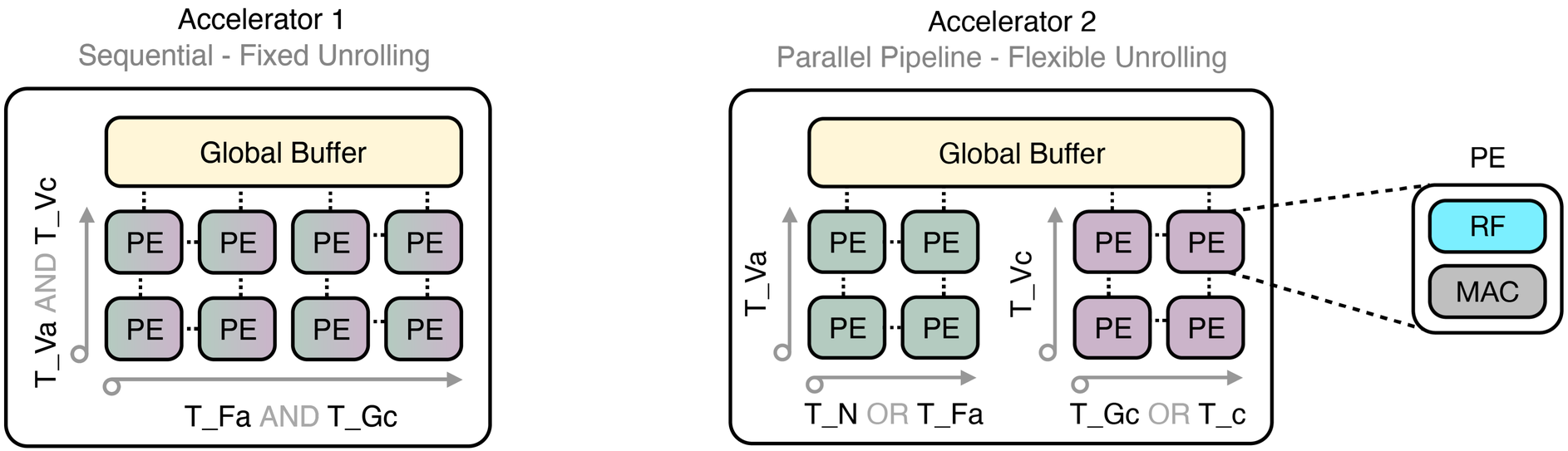
Spatial Accelerators for GNNs



We focus on large on programmable spatial accelerators with high parallelism Opportunities. These spatial accelerators can efficiently execute both the SpMM and Dense GEMM kernels of the two GNN layers phases.

Background

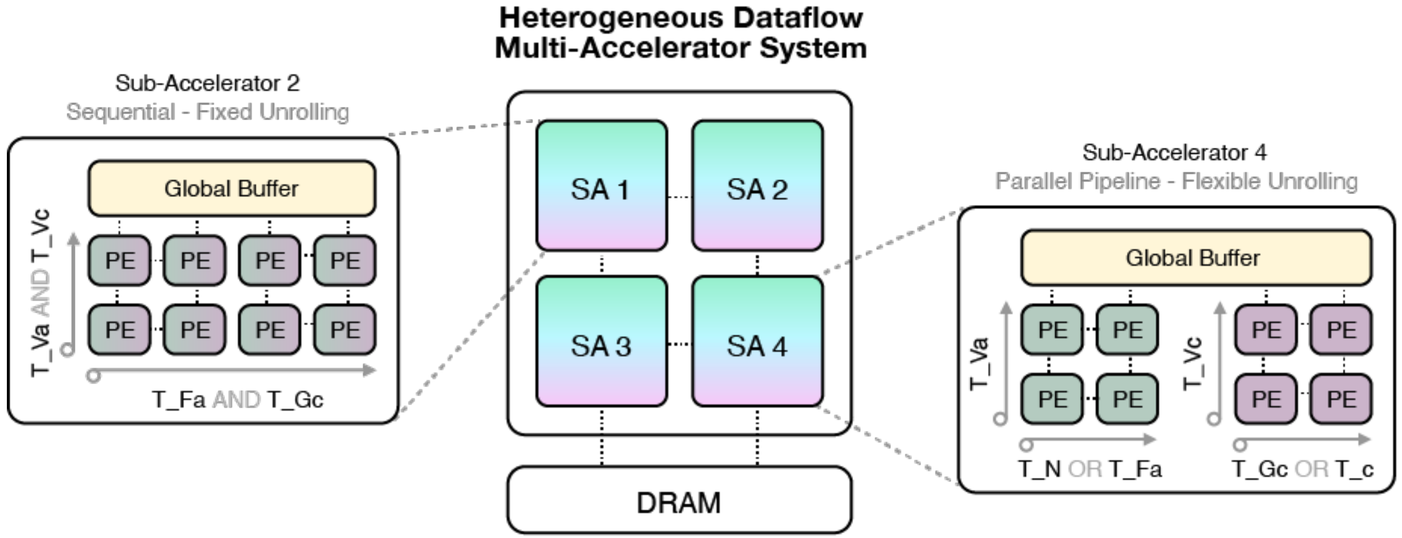
Spatial Accelerators for GNNs



Each accelerator feature a global buffer and a set of PEs interconnected with a Network-on-Chip. **Different unrolling dimensions and dataflow capabilities can be supported.** The PEs could be splitted between the two phases or be capable of runing computaitons for both phases.

Background

Heterogenous Dataflow Multi-Accelerator Systems



Flexibility in supported dataflows and interconnects can come with additional overhead.
A multi-dataflow accelerator can also be obtianed combining multiple accelerators with limited flexibility in a single system, i.e. a heterogenous dataflow multi-accelerator system, similarly, to what has been proposed for traditional DNNs by Kwon et al.

Kwon et al. "Heterogeneous dataflow accelerators for multi-DNN workloads." HPCA 2021

Background

GNN Mapping

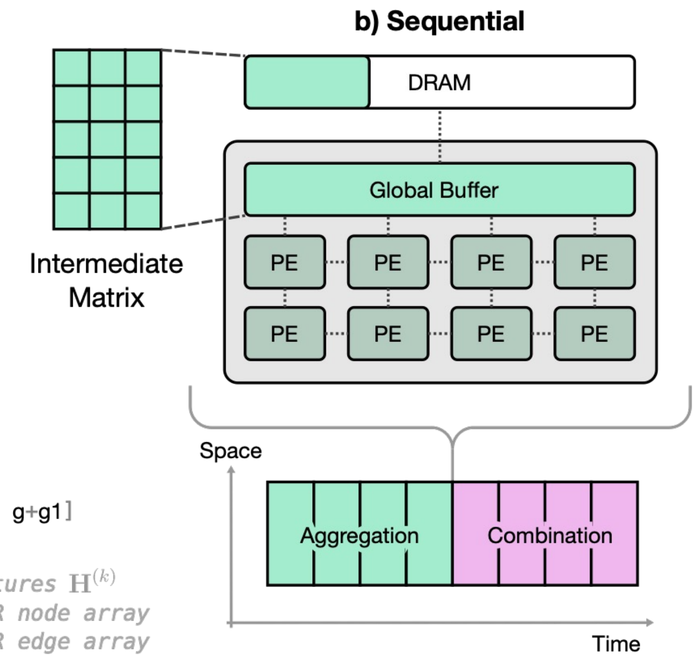
a) Sequential Loop Nest

```
# Aggregation
for v in range(0, V, T_Va):
  for f in range(0, F, T_Fa):
    parallel-for v1 in range(0, T_Va, 1):
      for n in range(A_node[v+v1], A_node[v+v1+1]):
        parallel-for f1 in range(0, T_Fa, 1):
          parallel-for n1 in range(0, T_N, 1):
            M[ v+v1, f+f1 ] += H[ A_edge[n+n1], f+f1 ]

# Combination
for v in range(0, V, T_Fc):
  for g in range(0, G, T_Gc):
    for f in range(0, F, T_Fc):
      parallel-for v1 in range(0, T_Vc):
        parallel-for g1 in range(0, T_G):
          parallel-for f1 in range(0, T_Fc):
            H_new[v+v1, g+g1] += M[v+v1, f+f1] * W[f+f1, g+g1]

# Legend
H: input node features H^(k-1)        H_new: output node features H^(k)
M: intermediate matrix                A_node: adj. matrix CSR node array
W: weights W                          A_edge: adj. matrix CSR edge array
```

b) Sequential



Similarly to traditional DNNs, the algorithm for the two phases can be represented as **two loop nests** executed sequentially.

Background

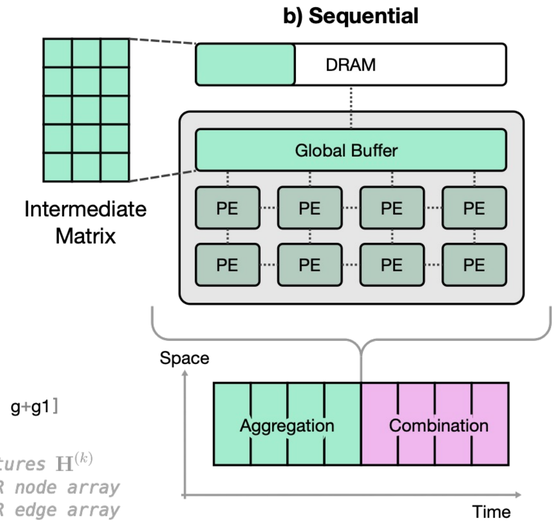
GNN Mapping

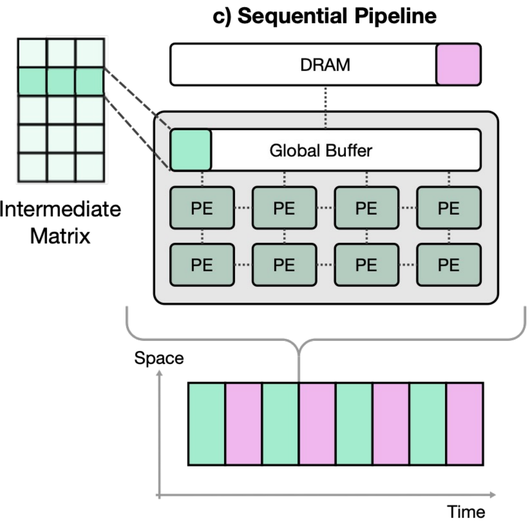
a) Sequential Loop Nest

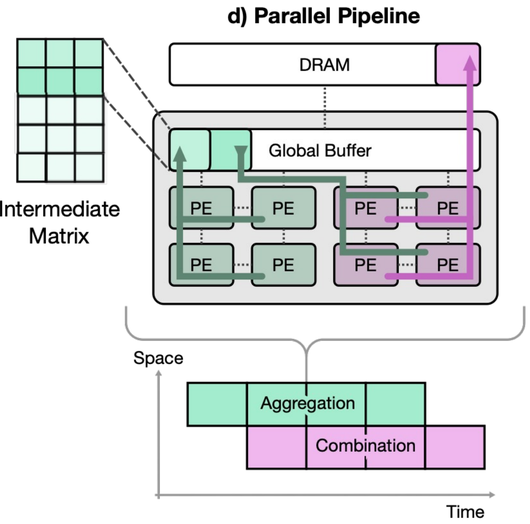
```
# Aggregation
for v in range(0, V, T_Va):
  for f in range(0, F, T_Fa):
    parallel-for v1 in range(0, T_Va, 1):
      for n in range(A_node[v+v1], A_node[v+v1+1]):
        parallel-for f1 in range(0, T_Fa, 1):
          parallel-for n1 in range(0, T_N, 1):
            M[ v+v1, f+f1 ] += H[ A_edge[n+n1], f+f1 ]

# Combination
for v in range(0, V, T_Fc):
  for g in range(0, G, T_Gc):
    for f in range(0, F, T_Fc):
      parallel-for v1 in range(0, T_Vc):
        parallel-for g1 in range(0, T_G):
          parallel-for f1 in range(0, T_Fc):
            H_new[v+v1, g+g1] += M[v+v1, f+f1] * W[f+f1, g+g1]

# Legend
H: input node features H^(k-1)      H_new: output node features H^(k)
M: intermediate matrix              A_node: adj. matrix CSR node array
W: weights W                        A_edge: adj. matrix CSR edge array
```







However, to optimise data movement, the computation of the two phases can be interleaved or overlapped. Hence, three possible **inter-phase dataflows** can be selected: sequential, sequential pipeline and parallel pipeline. Each requiring specific tiling strategies.

Background

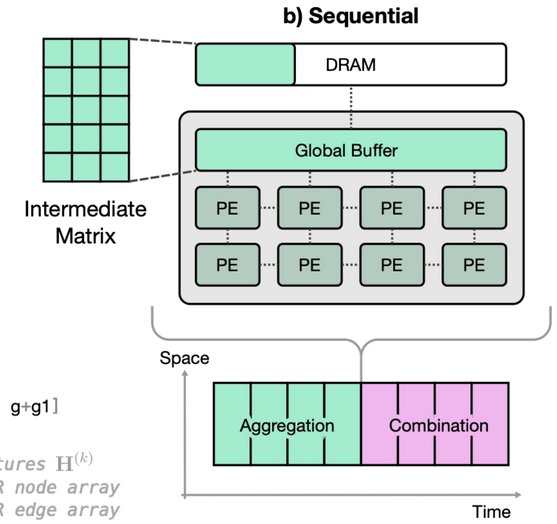
GNN Mapping

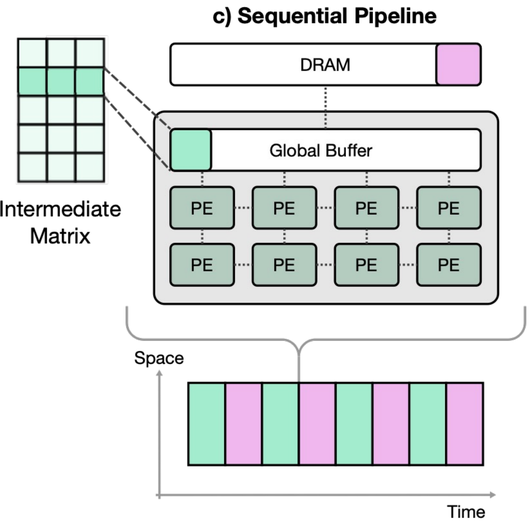
a) Sequential Loop Nest

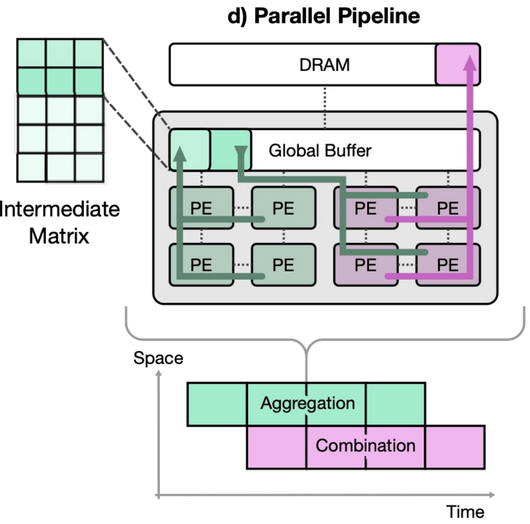
```
# Aggregation
for v in range(0, V, T_Va):
  for f in range(0, F, T_Fa):
    parallel-for v1 in range(0, T_Va, 1):
      for n in range(A_node[v+v1], A_node[v+v1+1]):
        parallel-for f1 in range(0, T_Fa, 1):
          parallel-for n1 in range(0, T_N, 1):
            M[ v+v1, f+f1 ] += H[ A_edge[n+n1], f+f1 ]

# Combination
for v in range(0, V, T_Fc):
  for g in range(0, G, T_Gc):
    for f in range(0, F, T_Fc):
      parallel-for v1 in range(0, T_Vc):
        parallel-for g1 in range(0, T_G):
          parallel-for f1 in range(0, T_Fc):
            H_new[v+v1, g+g1] += M[v+v1, f+f1] * W[f+f1, g+g1]

# Legend
H: input node features H^(k-1)        H_new: output node features H^(k)
M: intermediate matrix                A_node: adj. matrix CSR node array
W: weights W                          A_edge: adj. matrix CSR edge array
```







Furthermore, for both phases the **tiling sizes** and **unrolling dimensions** have to be chosen as in traditional DNNs mappings, but the N dimension, i.e. the number of neighbours, varies for each node and for each graph.

Problem

DNN vs GNN Mapping

DNN

A DNN layer can be **mapped offline** considering its shape and architecture. The optimal mapping does not depend on the specific layer input, e.g. image.

DNN mapping usually involves only **dense** tensors.

Inter-layer optimization can be applied.

GNN

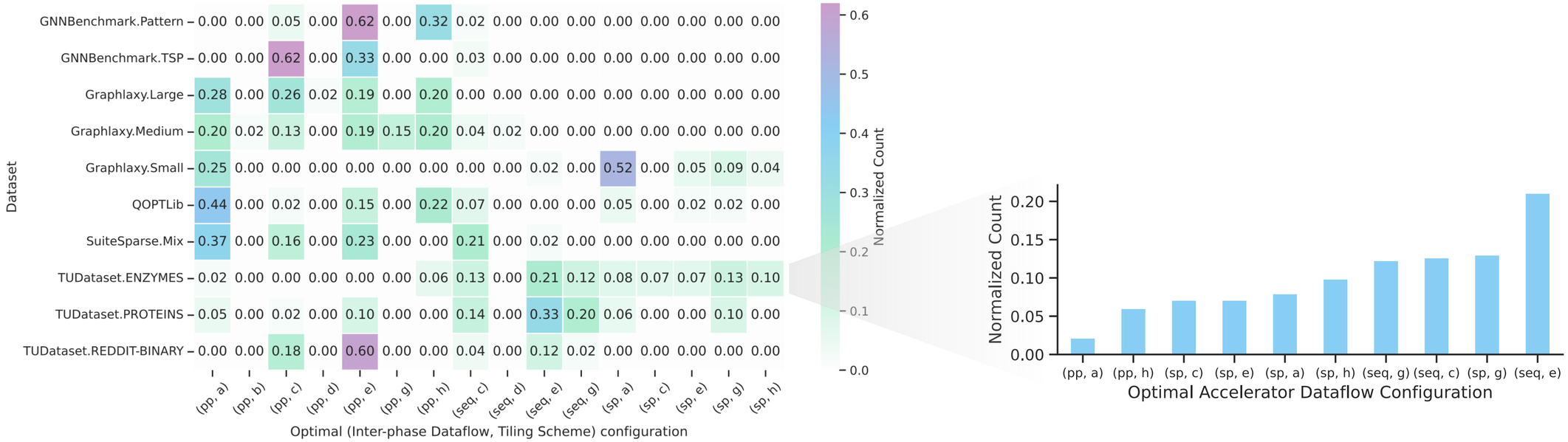
A GNN layer computation also depends on contingent input graph topology, hence, the optimal mapping must be found **for each input instance**.

GNN mapping has to consider **sparsity** of the adjacency matrix.

Inter-phase dataflow can be optimized.

Problem

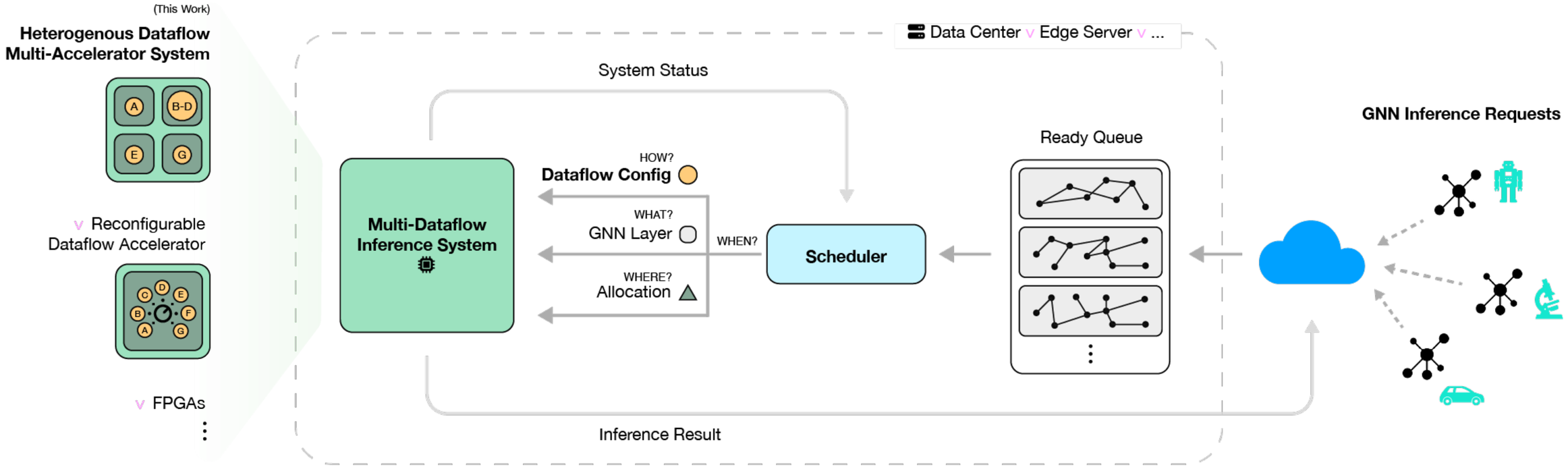
The best mapping of a GNN has to be found for each input graph instance and there is no *one-fits-all* solution



Problem

Many GNNs application scenarios involve real-time or streaming GNN inferences such as fraud detection, malicious user detection in social networks, load balancing in energy grids, smart cities, etc.

Mapping and scheduling of GNN inferences requires online optimization.



Problem

How to find the best GNN mapping/dataflow on-the-fly?

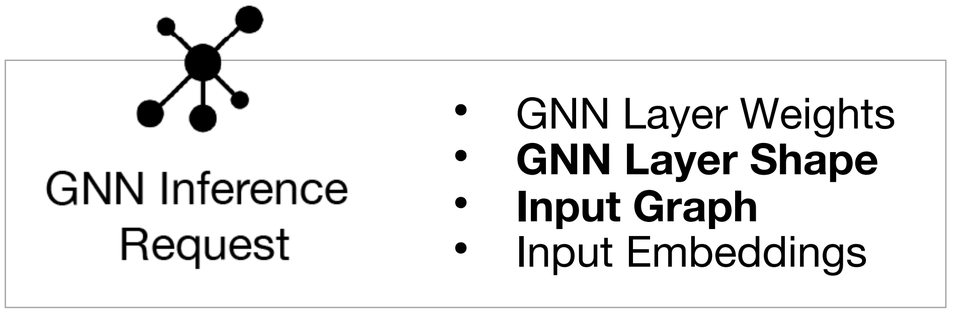
Proposed solution

A data-driven approach

Each GNN request is associated to **a GNN model**, **an input graph and embeddings**.

Thus, each GNN request can translate to multiple **GNN layer inference requests** that are stored in a **ready queue** of jobs ready to be run on the serving system.

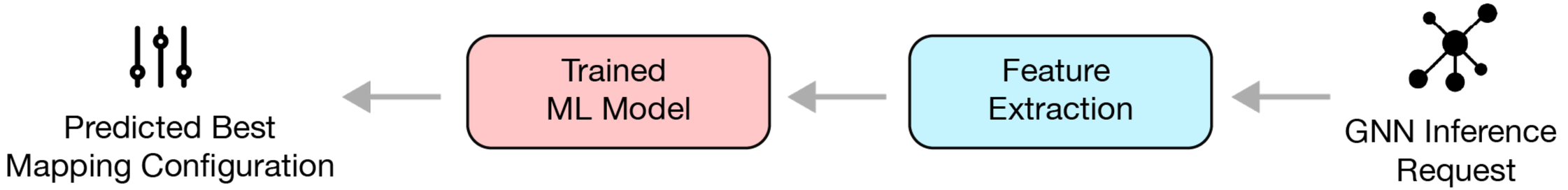
The relevant information to choose the optimal dataflow is the graph structure (num. of nodes, num. of edges, density, degree distribution, etc.) and the layer shape (number of output and input features).



Proposed solution

A data-drive approach

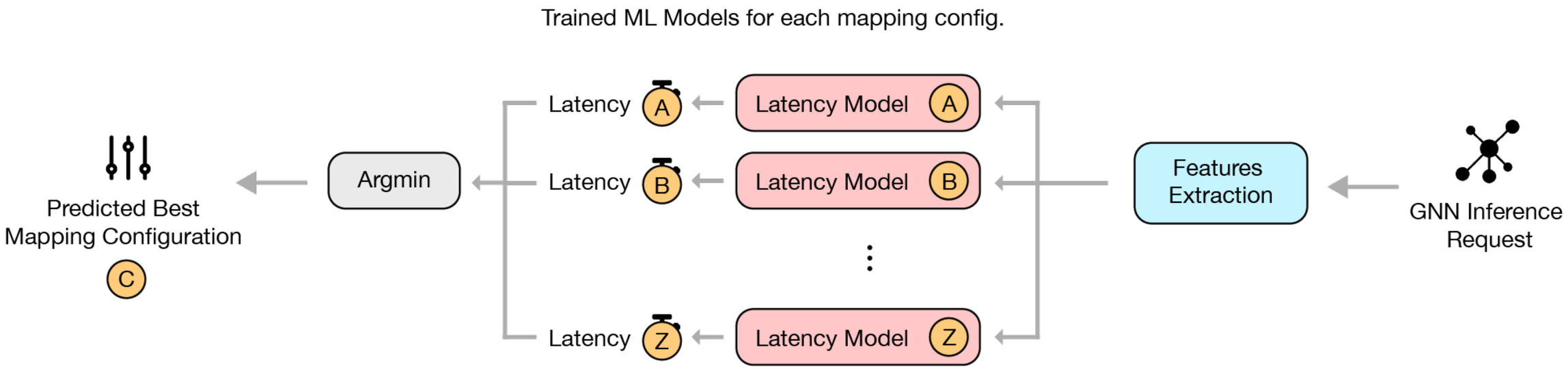
We investigated using a trained ML model to predict the best mapping in terms of latency for an incoming inference request.



Proposed solution

A data-drive approach

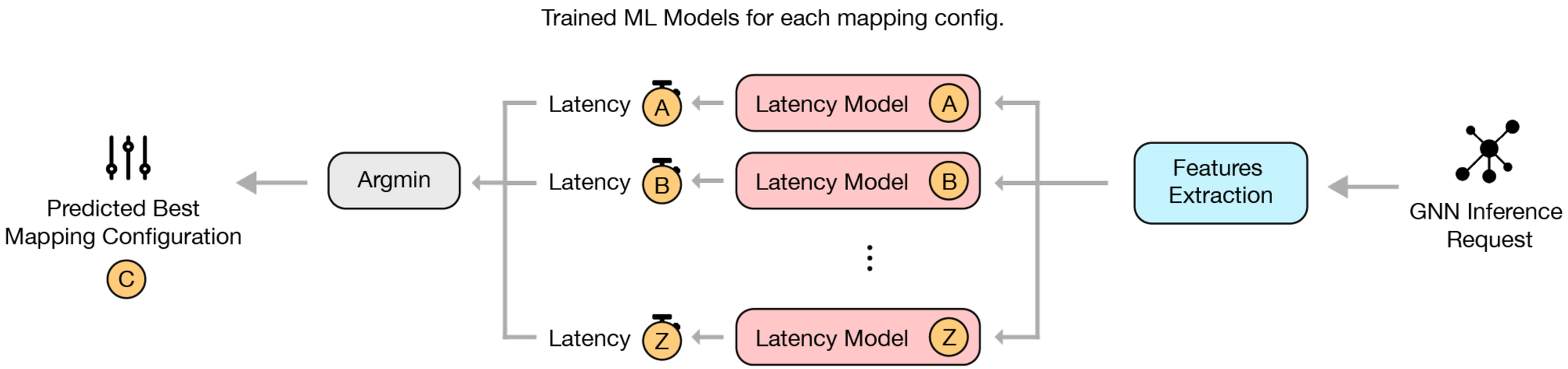
We considered a latency prediction model each mapping configuration (inter-phase dataflow and tiling strategy).



Proposed solution

A data-drive approach

The ensemble model allows us to find both ranking of suboptimal mappings and the latency for better online scheduling algorithms.



Proposed solution

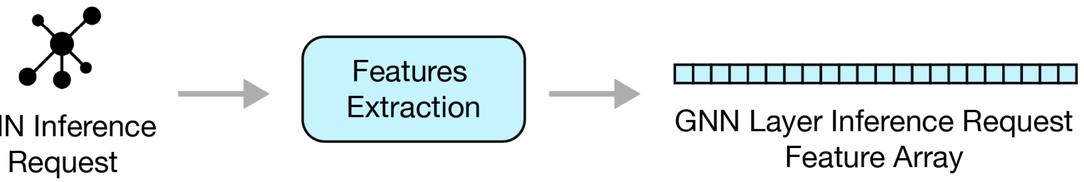
A data-drive approach

In particular, we considered all the **3 inter-phase dataflows** and the following **8 tiling configurations** totaling **24 possible mapping configurations**.

Tiling	Aggregation			Combination		
ω	T_Va	T_Fa	T_N	T_Vc	T_Gc	T_Fc
a	$*$	$\min(\text{Num. PEs},F)$	1	$*$	1	$\min(\text{Num. PEs},F)$
b	$*$	$\min(2,F)$	$\lfloor F/2 \rfloor$	$*$	1	$\min(2,F)$
c	$*$	$\min(8,F)$	$\lfloor F/2 \rfloor$	$*$	1	$\min(8,F)$
d	$*$	1	1	$*$	1	1
e	$*$	$\min(18,F)$	$\lfloor F/2 \rfloor$	$*$	1	$\min(18,F)$
f	$*$	1	$\min(18,V)$	$*$	1	1
g	$*$	$\min(18,F)$	1	$*$	1	$\min(85,F)$
h	$*$	1	$\min(18,V)$	$*$	1	$\min(85,F)$

Proposed solution

Feature Extraction



We considered input features of GNN inference requests for latency prediction models.

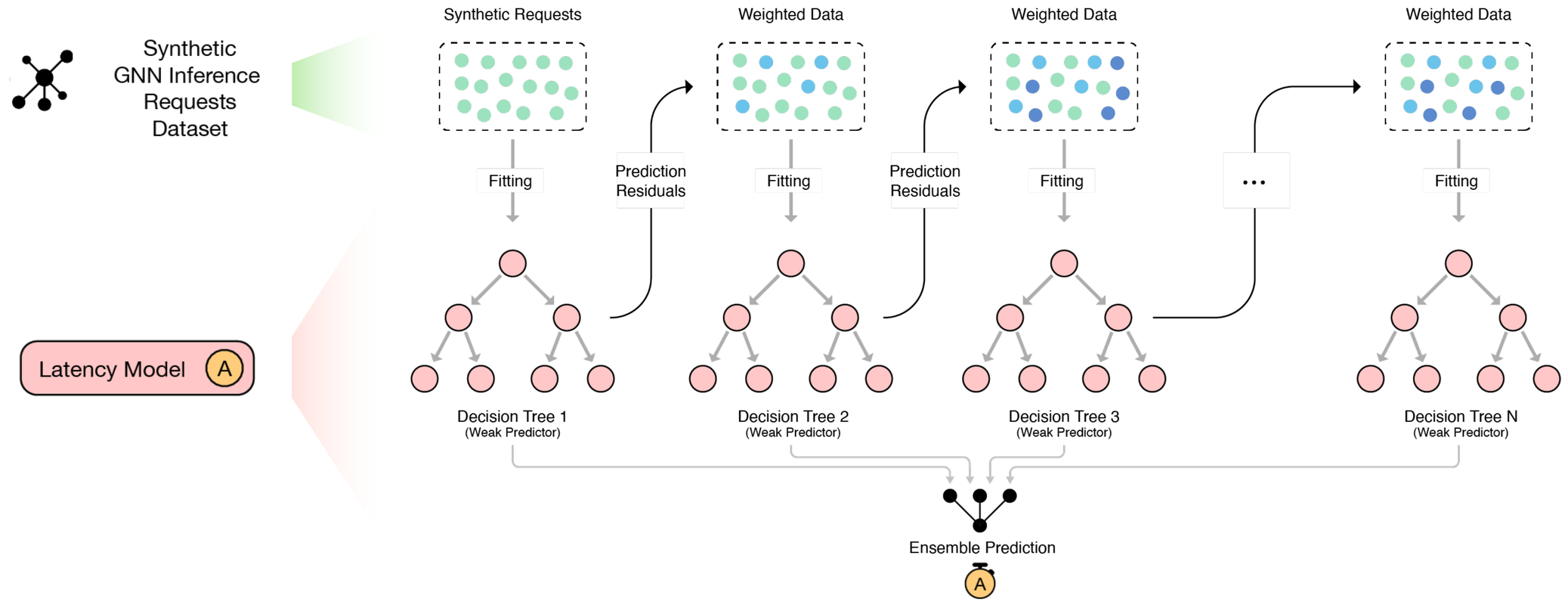
Simple features represent basic GNN layer characteristics, such as input/output features and nodes, while composite features are latency estimation formulas based on these attributes.

	Description		Symbol and/or Equation
base	Number of nodes and edges		V and E
	Spatial tiling factors		T_Va,T_Fa,T_N,T_Vc,T_Gc,T_Fc
	Graph density		$E \times V^{-2}$
	Clustering coefficient, measuring the probability that the adjacent nodes of a node are connected		[24]
	Seven node degrees quantiles normalized with respect to maximum degree including minimum degree		$Q_i, \quad \forall i \in \{1,...,7\}$
base+features	Number of operations, assuming a dense matrix mul. for the aggregation phase	S_1	$V \times F \times (G + \text{Mean Degree})$
	Estimation of the number of cycles for the combination phase	S_2	$\frac{V \times F \times G}{\text{T\_Vc} \times \text{T\_Fc} \times \text{T\_G}}$
	Estimation of the number of cycles for the aggregation phase (dense mat. mul.)	S_3	$\frac{V \times F \times \text{Mean Degree}}{\text{T\_N} \times \text{T\_Fa}}$
	Estimation of the latency for sequential inter-phase dataflow	S_4	$S_1 + S_3$
	Cycles estimation for aggregation phase assuming CSR encoding	S_5	$\sum_v^{\mathcal{V}} \frac{N_v \times F}{\min(N_v, \text{T\_N}) \times \text{T\_F}}$
	Cycles estimation for sequential inter-phase dataflow assuming CSR encoding	S_6	$S_3 + \frac{S_5}{\text{T\_Va}}$

Proposed solution

ML Models

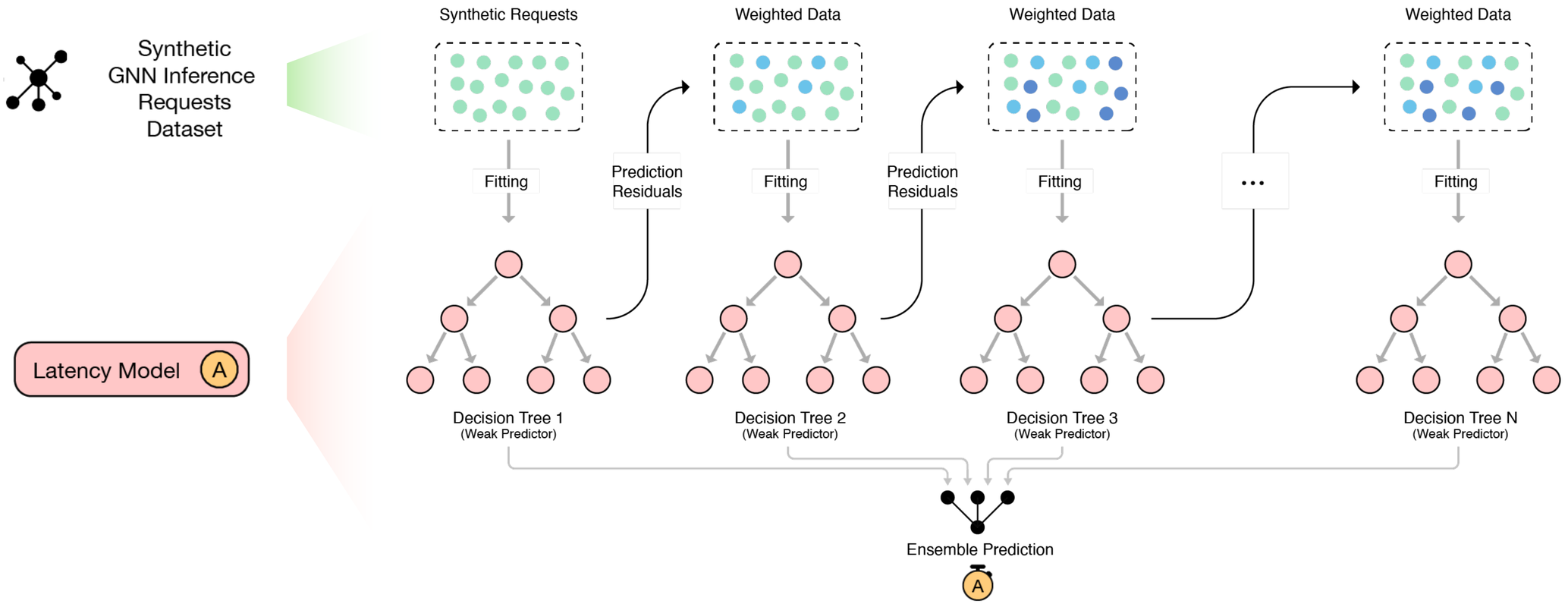
The latency regressors were implemented as **gradient boosting trees** using LightGBM framework



Proposed solution

ML Models

For better generalization capability we train the models to predict the **logarithm of the layency**.



Proposed solution

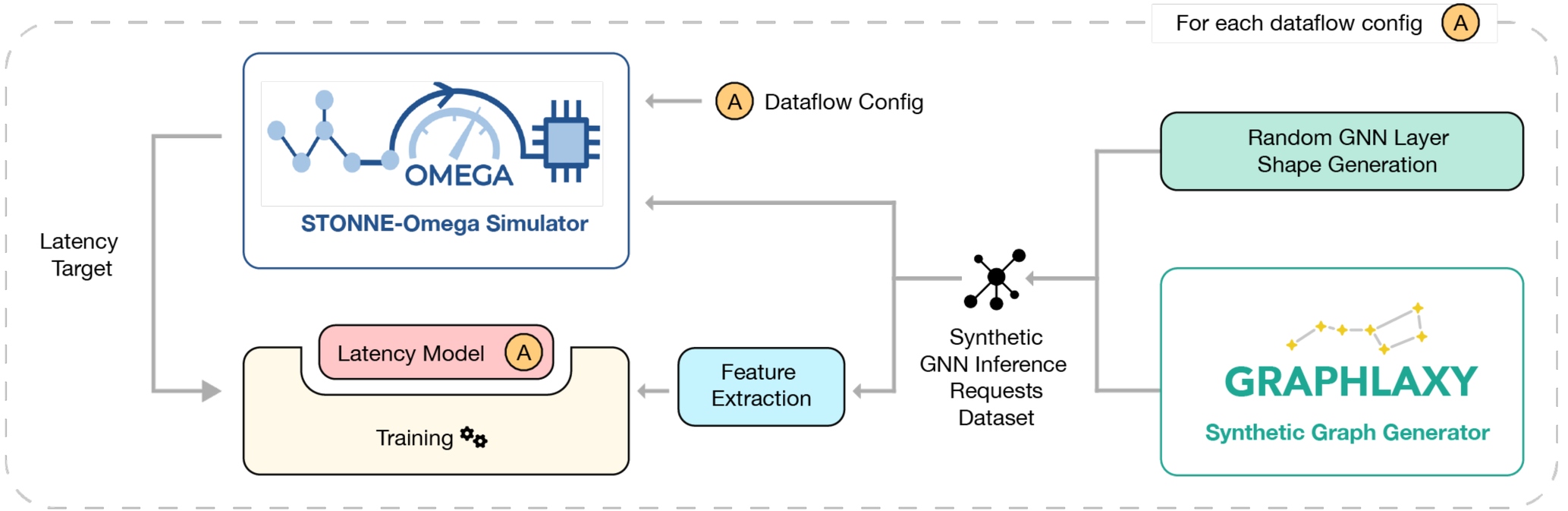
Training

The latency targets were obtained using **STONNE-Omega Simulator**

Garg, Raveesh, et al. "Understanding the design-space of sparse/dense multiphase GNN dataflows on spatial accelerators." 2022 IEEE International Parallel and Distributed Processing Symposium

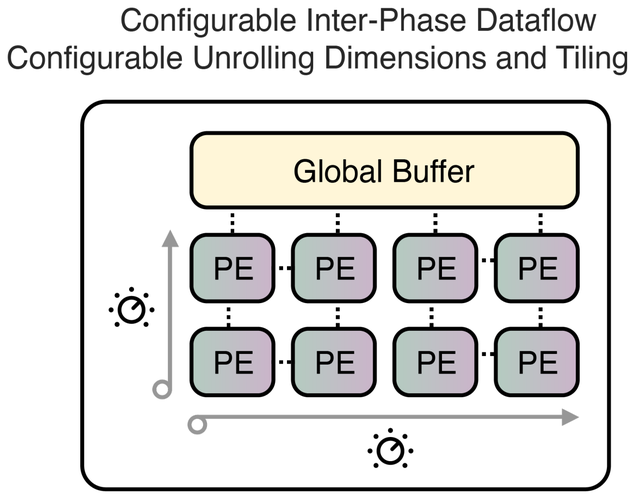
The training was performed on datasets of synthetic graphs generated using **Graphlaxy**

A. Wassington, S. Abadal, "Bias reduction via cooperative bargaining in synthetic graph dataset generation.", 2022



Experiments

Prediction Accuracy in Offline Single Accelerator Setup



We evaluated the total execution time for different graph datasets on highly configurable single accelerator system. The optimal total execution time is achieved when each request is executed with the best dataflow configuration.

Experiments

Prediction Accuracy

Dataset Name	MAPE ↓	Top-1 acc. (%) ↑	Top-3 acc. (%) ↑	Improvement over random (%) ↑	Improvement over best fixed (%) ↑	Degradation over optimal (%) ↓
Graphlaxy.Medium	3.78	91.28	99.62	93.63	58.42	0.56
Graphlaxy.Large	4.83	83.63	98.20	96.27	8.99	0.86
Graphlaxy.Small	17.36	46.94	75.51	97.93	0.49	5.12
SparseSuite.Mix	13.32	62.79	81.40	89.79	8.04	5.24
QOPTLib	24.99	56.41	74.36	82.45	20.98	14.94
PATTERN	40.17	33.13	68.56	84.04	-6.30	9.40
TSP	29.13	62.18	92.85	87.38	2.18	4.22
ENZYMES	20.23	19.52	40.41	91.95	-3.29	12.10
PROTEINS	15.82	30.45	56.76	91.49	1.24	11.56
REDDIT-BINARY	14.16	35.08	76.45	85.96	0.74	8.68

Less than 15% degradation in execution time over optimal with the predicted dataflows.

Experiments

Ablation study on composite features and log prediction

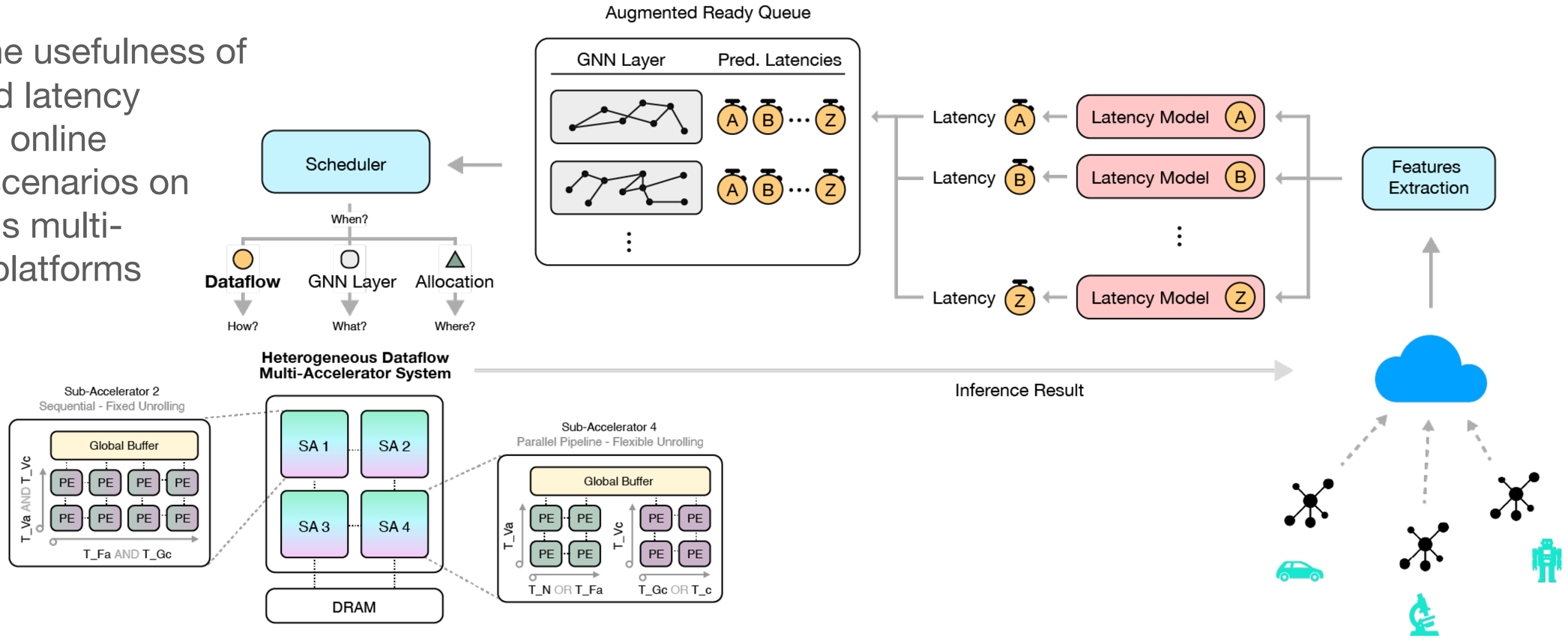
Dataset Name	Model Name	MAPE ↓	Degradation over optimal (%) ↓
Graphlaxy.Medium	base	35.77	48.12
	base+features	10.76	4.06
	base+log	4.95	0.84
	base+features+log	3.78	0.56
SparseSuite.Mix	base	97.04	92.68
	base+features	31.68	12.52
	base+log	19.09	5.87
	base+features+log	13.32	5.24
QOPTLib	base	515.94	92.94
	base+features	152.67	86.60
	base+log	38.22	21.18
	base+features+log	24.99	14.94

Considering **composite features** and predicting **logarithm of the latency** achieves the highest accuracy.

Experiments

Online Scheduling Setup: Multi-Accelerator System

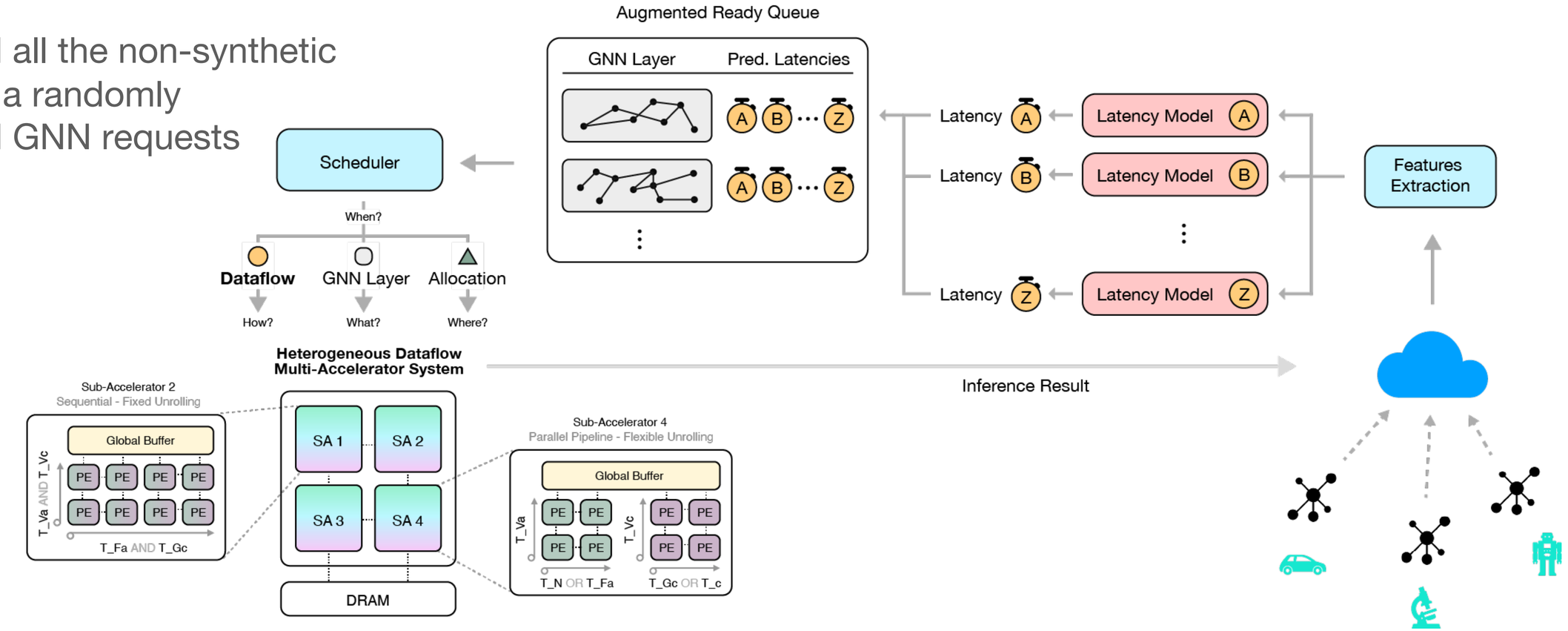
We tested the usefulness of the proposed latency predictors in online scheduling scenarios on heterogenous multi-accelerator platforms



Experiments

Online Scheduling Setup: Multi-Accelerator System

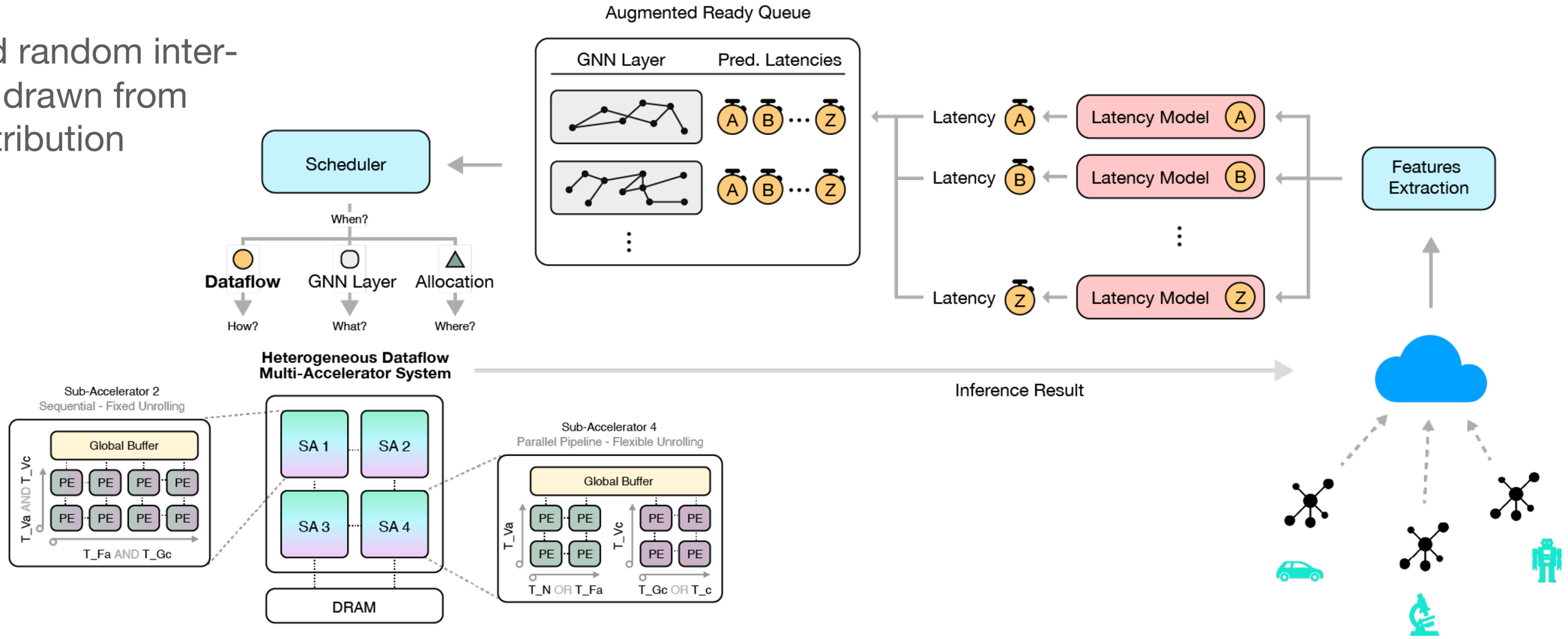
We merged all the non-synthetic datasets in a randomly permutated GNN requests stream.



Experiments

Online Scheduling Setup: Multi-Accelerator System

We assumed random inter-arrival times drawn from a Pareto distribution



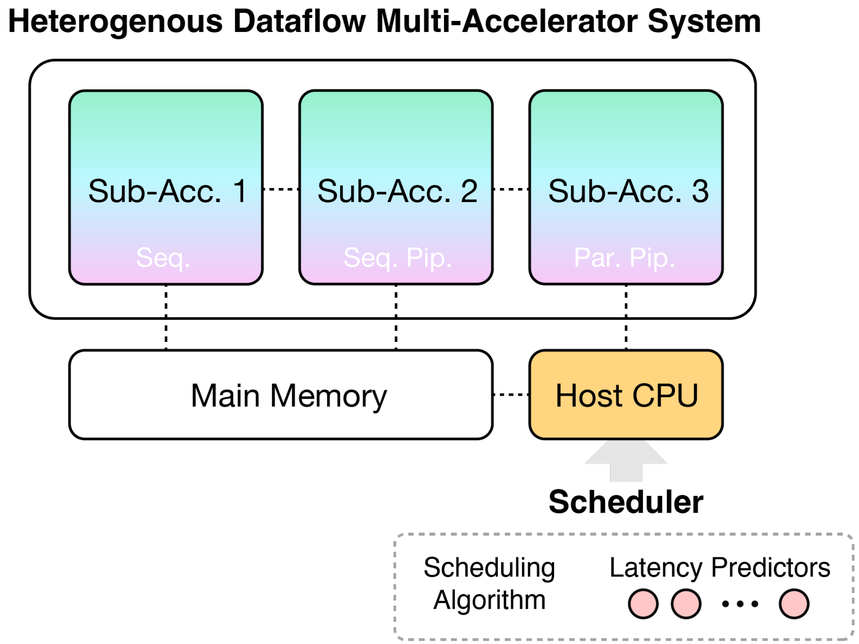
Experiments

Heterogenous Dataflow Multi-Accelerator System

For the online scheduling experiments, we considered a system consisting in **3 sub-accelerators**.

Each sub-accelerator (SA) support a different inter-phase dataflow (Sequential, Seq. Pipeline, Parallel Pipeline), but it is **flexible in terms of loop unrolling dimensions**.

Furthermore each SA features **512 processing elements** with 64B local buffer each.



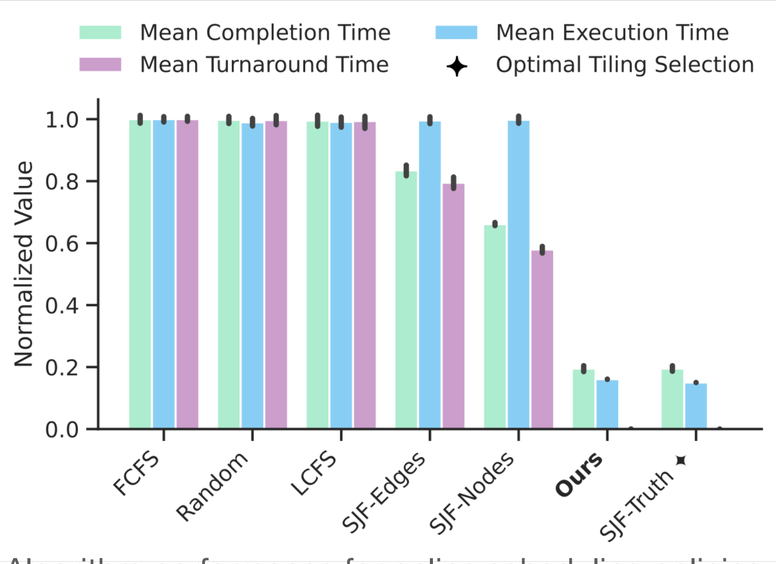
Experiments

Considered Online Scheduling Algorithm

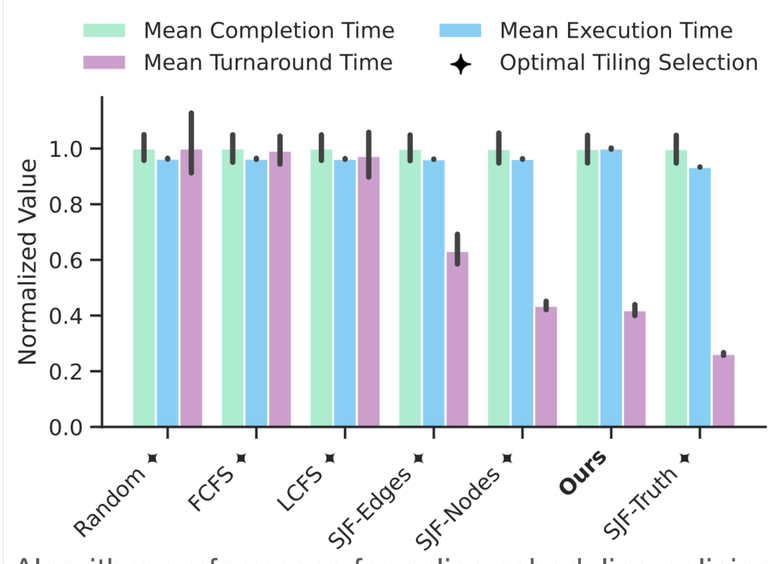
We considered a **shortest-job-next algorithm** which minimizes job waiting time. We assumed a work-conserving scenario in which once a SA is free, the GNN layer in the ready queue with shortest predicted execution time is chosen to be executed next.

Experiments

Online scheduling performance



Algorithm performance for online scheduling policies with baselines using random tiling selection.



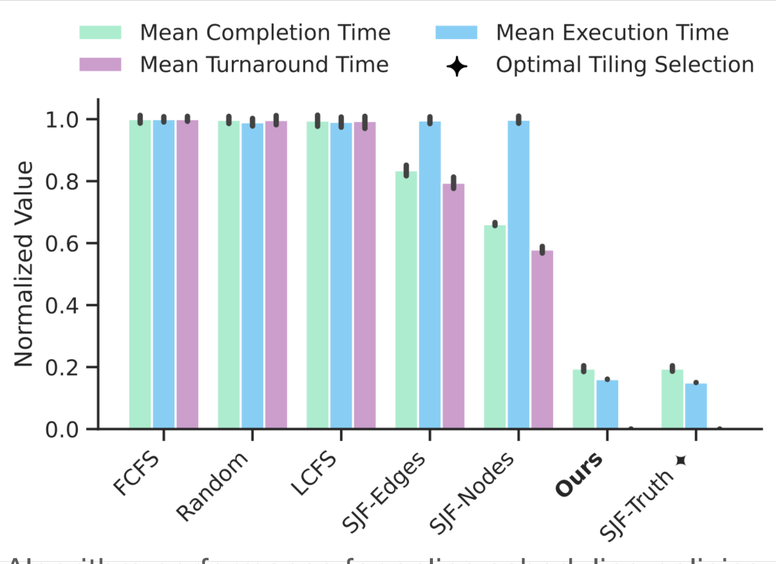
Algorithm performance for online scheduling policies with baselines using theoretical best tiling selection.

✦ := Not feasible in practice

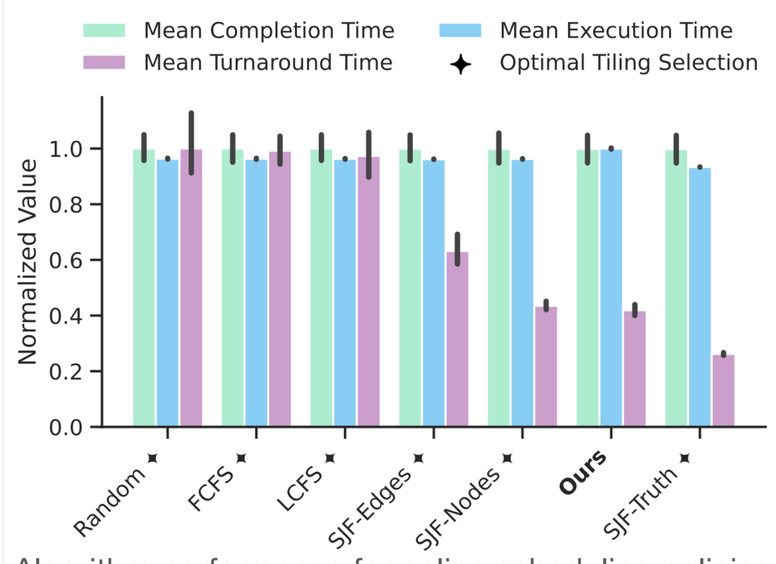
The proposed approach, i.e. a **shortest-job-first based on the latency predictions,** allows **83.88%** and **99.95% reduction** in **execution time** and **turnaround time** respectively, with respect to the best performing **feasible** scheduling algorithm, namely **shortest-job-first based on number of nodes**

Experiments

Online scheduling performance



Algorithm performance for online scheduling policies with baselines using random tiling selection.



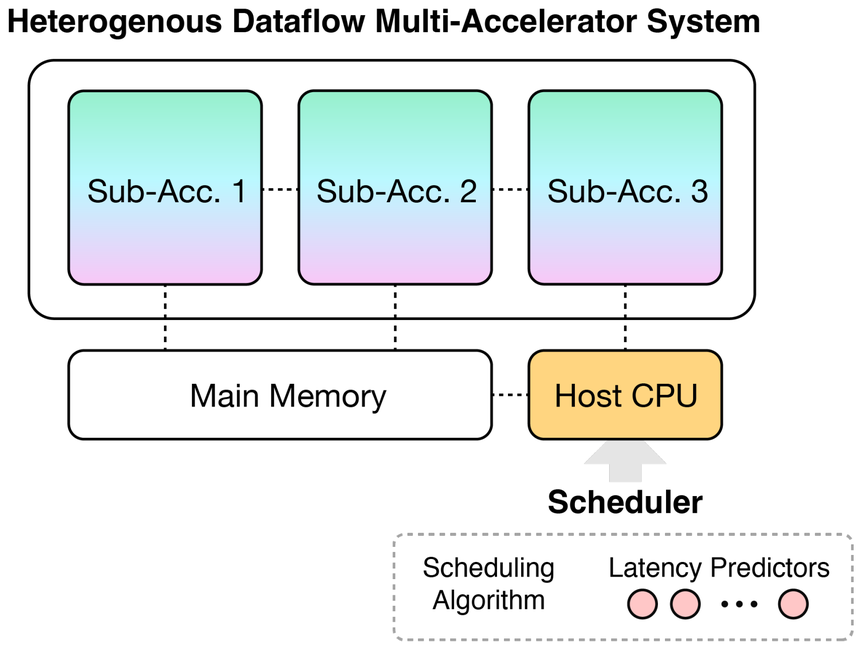
Algorithm performance for online scheduling policies with baselines using theoretical best tiling selection.

✦ := Not feasible in practice

While, with respect to the best considered **non-feasible** scheduling algorithm, time, a **shortest-job-first based on actual execution times**, the proposed solution achieves **1.07x** and **1.60x higher mean execution time** and **mean turnaround time,** respectively.

Experiments

Online scheduling overhead



In our setup, compiled gradient boosting tree predictors run on the CPU parallel to inference executions. The mean latency prediction phase duration for each graph was only **12.3% of the mean job waiting time**, avoiding any turnaround time increase.

Notably, the latency prediction phase is independent of GNN request sizes, suggesting that overhead could be further reduced for larger graphs.

Conclusion

- We presented a data-driven approach to dataflow latency evaluation of GNN workloads, based on gradient boosting trees.
- We showed the usefulness of such predictors in an online scheduling scenario featuring multi-dataflow GNN accelerators.
- Several limitations remain to be addressed in future works. For instance, only a subset of possible mappings has been considered, and the proposed methodology requires to train a model for each tiling/dataflow configuration.
- Future work could also focus on more complex online scheduling algorithms informed by the predictions, customized for the specific hardware system.

Thanks

enrico.russo@phd.unict.it