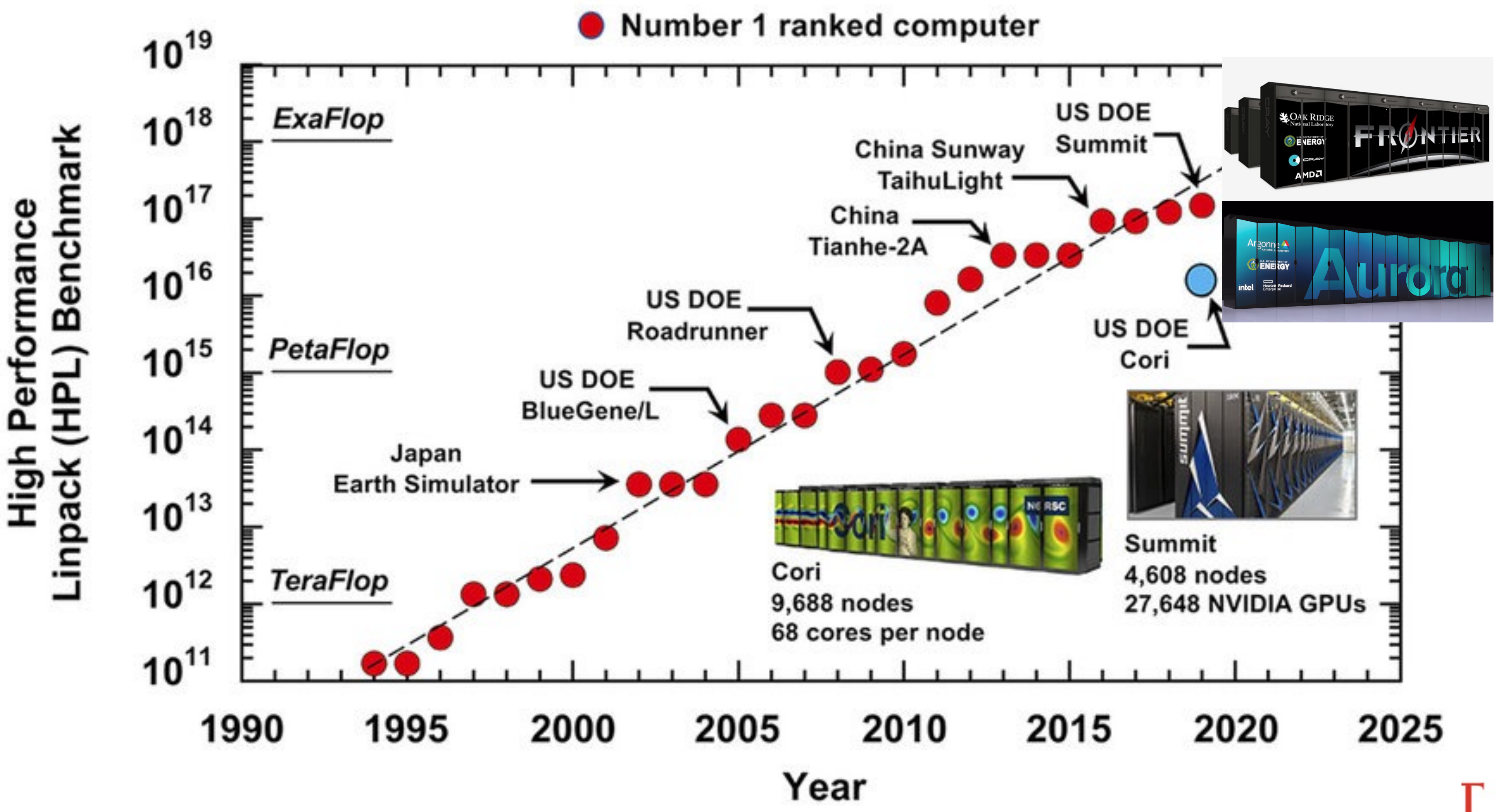
Advancing Intelligent Scheduling for Complex Large-Scale Systems

Jing Li
Department of Computer Science
Ying Wu College of Computing
jingli@njit.edu

NJIT
New Jersey Institute of Technology

Increasingly Large-Scale Supercomputers

e.g., Frontier has 8,699,904 combined CPU and GPU cores.



Increasingly Large-Scale Supercomputers

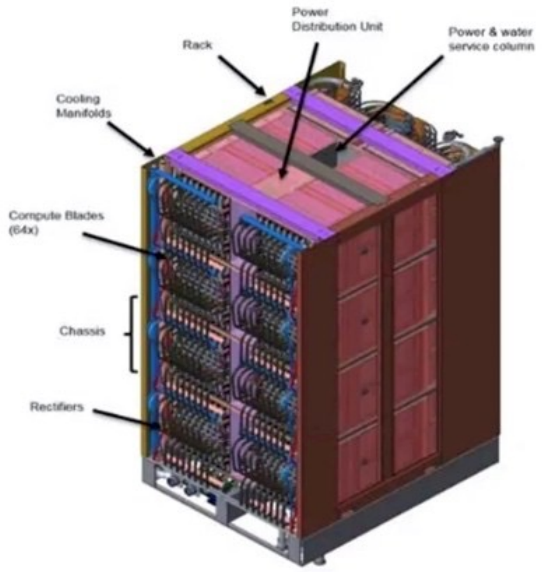
e.g., Frontier has 8,699,904 combined CPU and GPU cores.

System

- 2 EF Peak DP FLOPS
- 74 compute racks
- 29 MW Power Consumption
- 9,408 nodes
- 9.2 PB memory (4.6 PB HBM, 4.6 PB DDR4)
- Cray Slingshot network with dragonfly topology
- 37 PB Node Local Storage
- 716 PB Center-wide storage
- 4000 ft^2 foot print

Olympus rack

- 128 AMD nodes
- 8,000 lbs
- Supports 400 KW



AMD node

- 1 AMD "Trento" CPU
- 4 AMD MI250X GPUs
- 512 GiB DDR4 memory on CPU
- 512 GiB HBM2e total per node (128 GiB HBM per GPU)
- Coherent memory across the node
- 4 TB NVM
- GPUs & CPU fully connected with AMD Infinity Fabric
- 4 Cassini NICs, 100 GB/s network BW

Compute blade

- 2 AMD nodes

All water cooled, even DIMMS and NICs

Challenges in Scheduling for HPC and SC

- HPC systems become more complex to achieve increasing computing capability, making it challenging for applications to fully leverage large-scale and heterogeneous resources.
- Scientific computing workflows are more complex and dynamic, with varying execution times and resource needs, which cannot be sufficiently captured by traditional bulk-synchronous models.
- Manual resource allocation decisions are time-consuming and challenging for application developers.
- Heuristic-based decisions often result in under-utilization of resources and inadequate performance.
- The underlying scheduling problems are more intricate than those in classical scheduling theory.

My Previous Journey in Scheduling Theory

- Support parallel applications with different performance needs
  - Minimize max latency:
    - Prove that FIFO is $O(1+\varepsilon)$-speed $O(1)$-competitive and develop a more practical randomized policy with similar theoretical performance
  - Minimize average latency:
    - Prove that Latest-Arrival-Processor-Sharing is $O(1+\varepsilon)$-speed $O(1)$-competitive and the more practical shortest job first is $O(2+\varepsilon)$-speed $O(1)$-competitive
  - Maximize profit of meeting individual deadlines
  - Meet hard deadlines
  - Meet hard deadlines + soft deadlines
  - Etc.

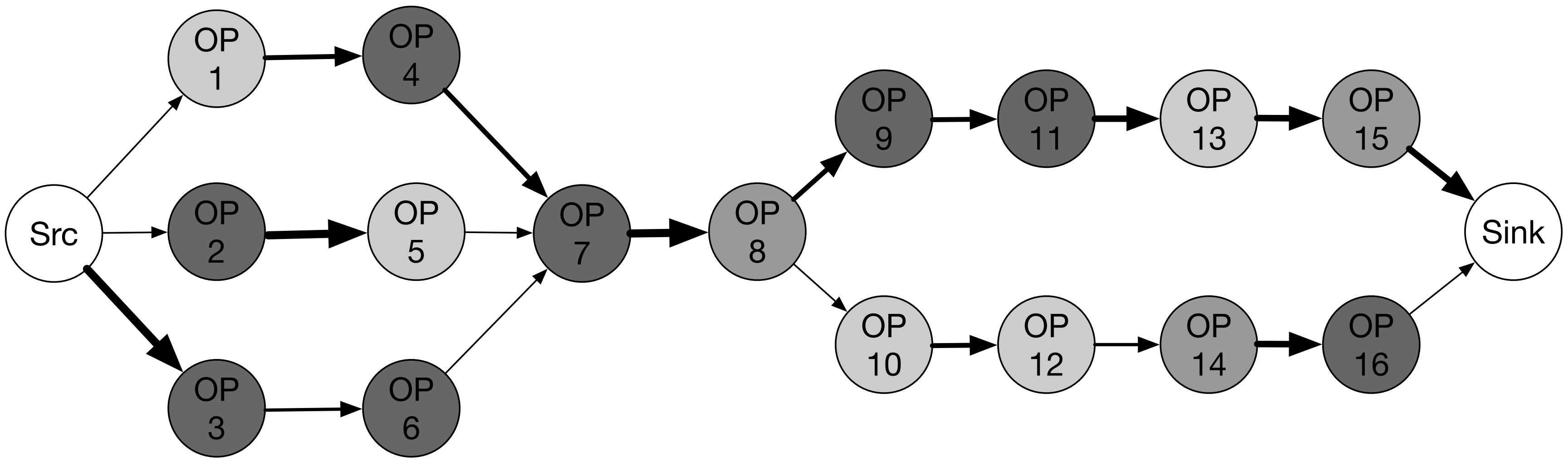
Apply Reinforcement Learning-Based Approach

- Resource allocation for workflow graphs is a combinatorial optimization problem with a huge search space.
- The capability of generalizing to unseen graphs requires learning global topological information important for scheduling.
- Scaling to large graphs and complex systems is challenging.

- We applied deep reinforcement learning to stream processing:
  - Develop a generalizable RL model for device placement [AAAI'20]
  - Develop a new RL formulation for larger graphs [IPDPS'23]

Stream Processing

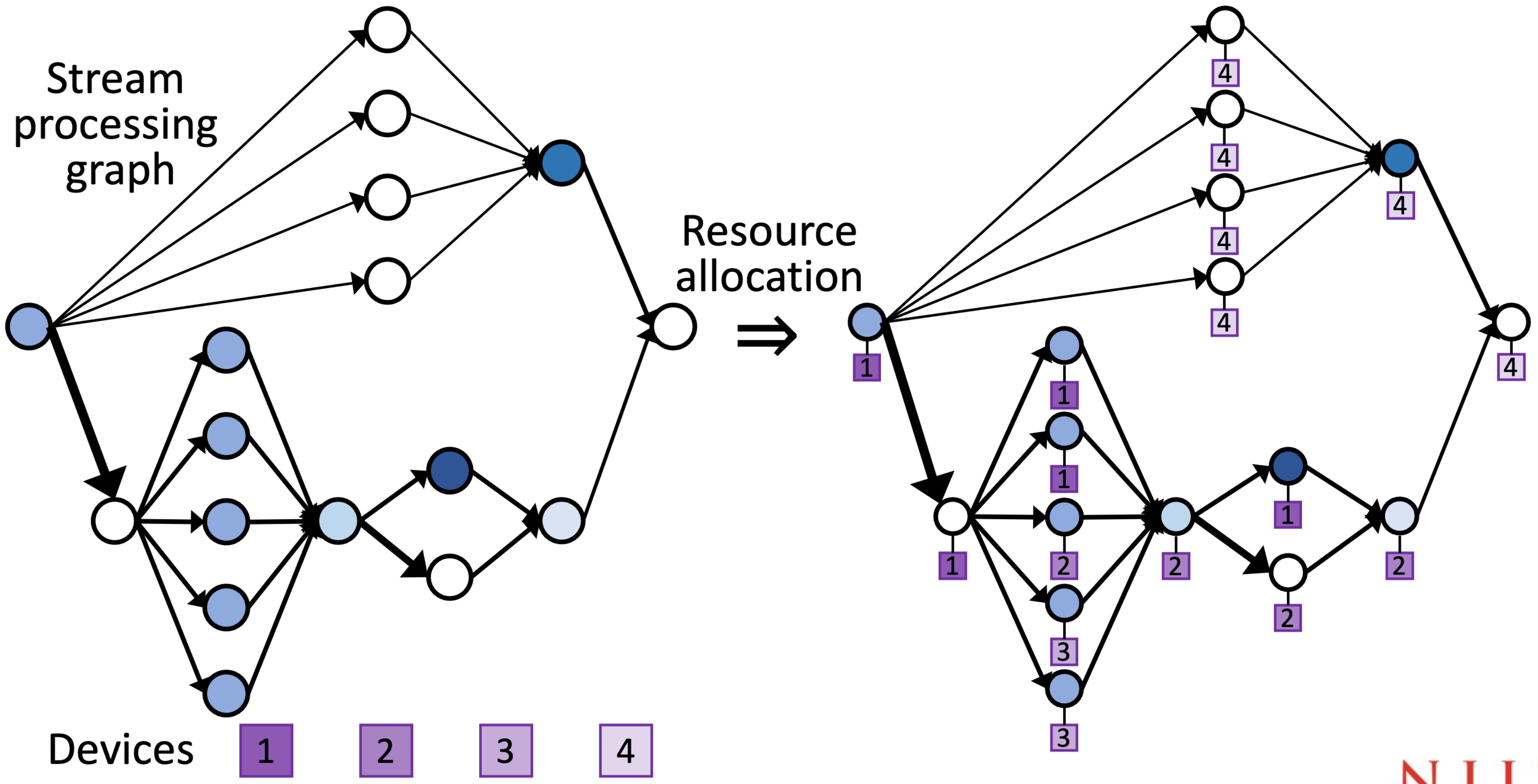
- Stream processing is widely used to analyze online arrival data with high throughput and generate live results in a timely manner
- The computation and communications in stream processing can be represented as a Directed Acyclic Graph (DAG)



The nodes of the graph represent operators labeled with the amount of computation work. The darker color of an operator illustrates higher CPU utilization. The directed edges depict data flowing from the source to destination operators. The edge width represents the amount of data flowing through that connection.

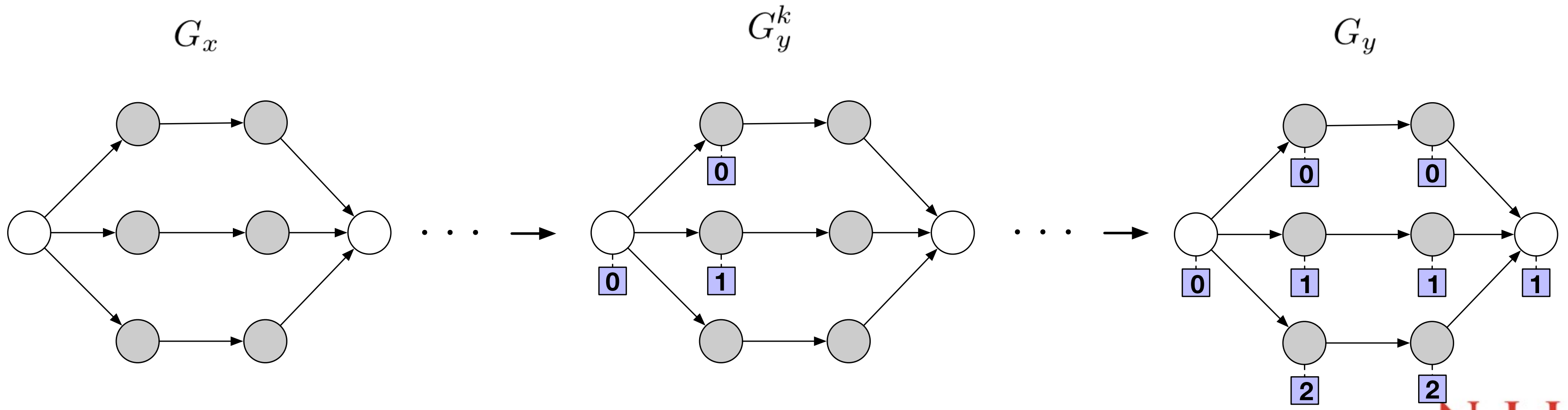
Device Placement for Stream Processing Graphs

- The main performance objective of stream processing is to achieve high processing throughput on the available computing devices via good resource allocation for the operators.



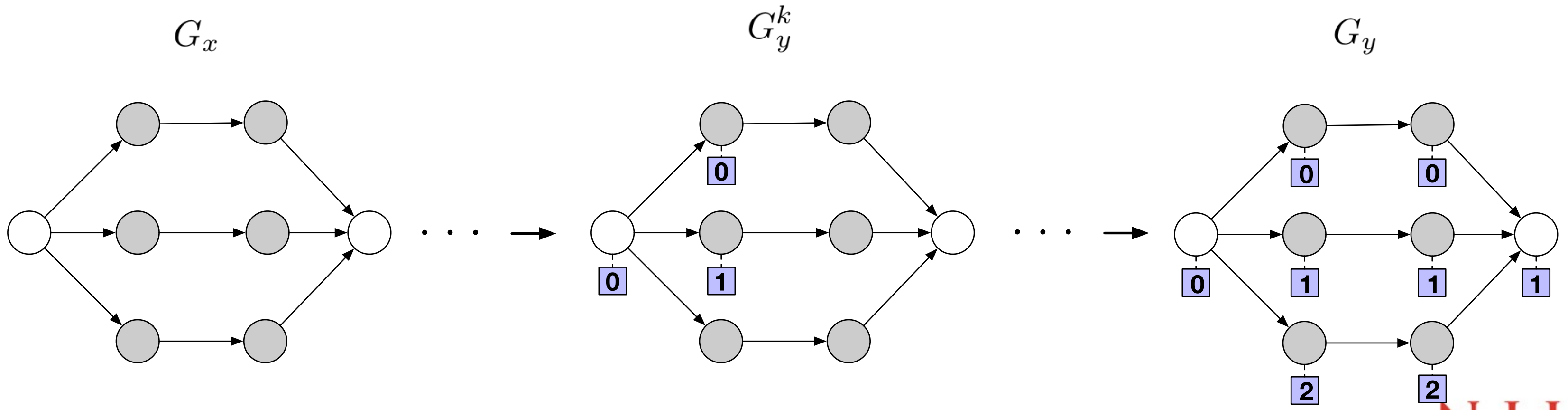
Applying RL to Device Placement

- Intuitively, the device prediction of one node is usually highly influenced by the device assignments of its upstream nodes.
- The prediction of the resource allocation is to assign each operator node in the graph to a device, conditioning on the graph property and the assignments of other nodes.



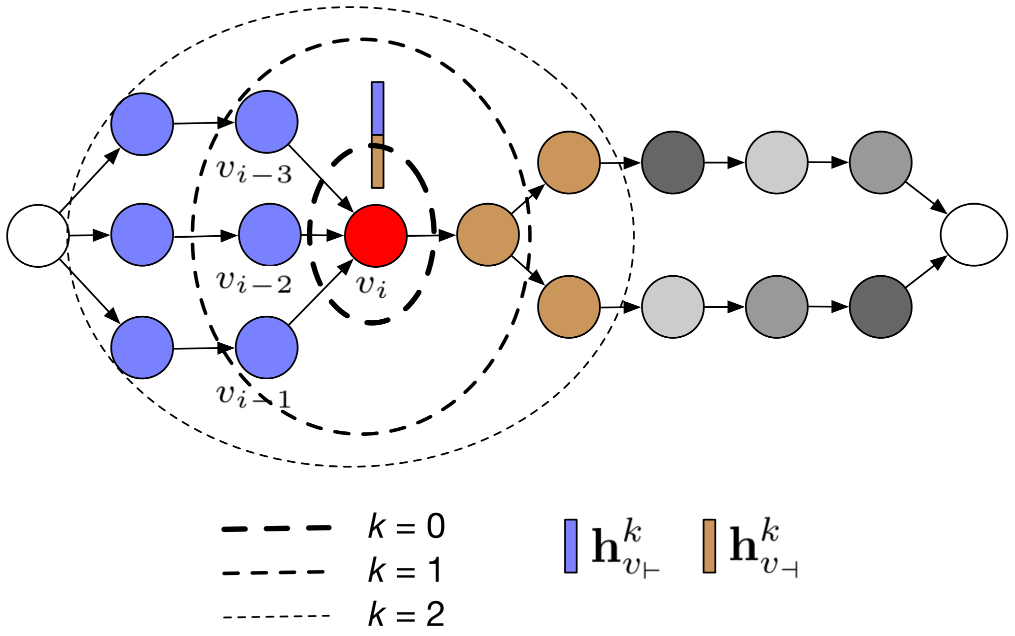
Challenges for Applying RL

- Good allocations require globally balancing computation and communication load → need to learn the global information of the entire graph and the relation between devices' assignments.
- Generalization to unseen graphs → need to capture the desired properties of graph topology into the graph representations and the representation space must be transferable among different graphs
  - Prior work: train and test on the same graph



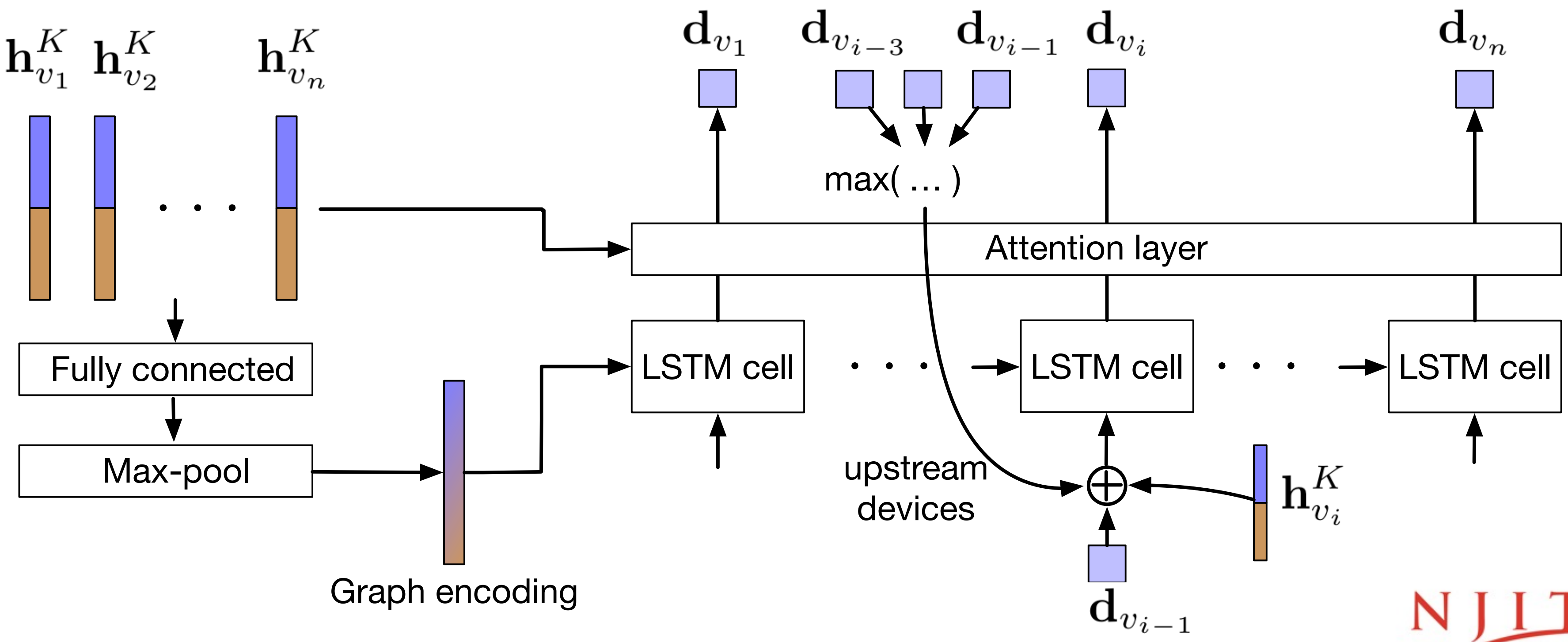
Stream Graph Encoding

- Our model embeds the input graph to an embedding space using graph convolution:
  - Embed both node and edge feature into nodes
  - Distinguish upstream and downstream neighbors
  - Aggregate neighbor embeddings
  - Perform *k* convolutions to reach *k*-hop neighbors



Graph-Aware Encoder-Decoder Model

- Graph embedding and whole graph encoding provides global topological information
- Embedding device assignments and aggregating upstream devices further encourages the model be aware of the assignments of the upstream nodes to help its assignment prediction



RL Training using CEPSim Simulator

- It is infeasible to get the ground truth (optimal) allocation.
- We follow the RL setting, where the model makes a sequence of decisions (i.e., our decoder) and gets delayed reward (i,e., the throughput of the predicted graph allocation).
  - We apply the apply the REINFORCE algorithm to compute the policy gradients and learn the network parameters.
  - Due to the sparsity of good resource allocations over the large search space, for each training graph, we maintain a memory buffer to store the good samples with high reward.
- DRL relies on the evaluation of numerous resource allocation trials to learn better allocations.
- We use CEPSim, a simulator for cloud-based complex event processing and stream processing system, to evaluate the resource allocations.

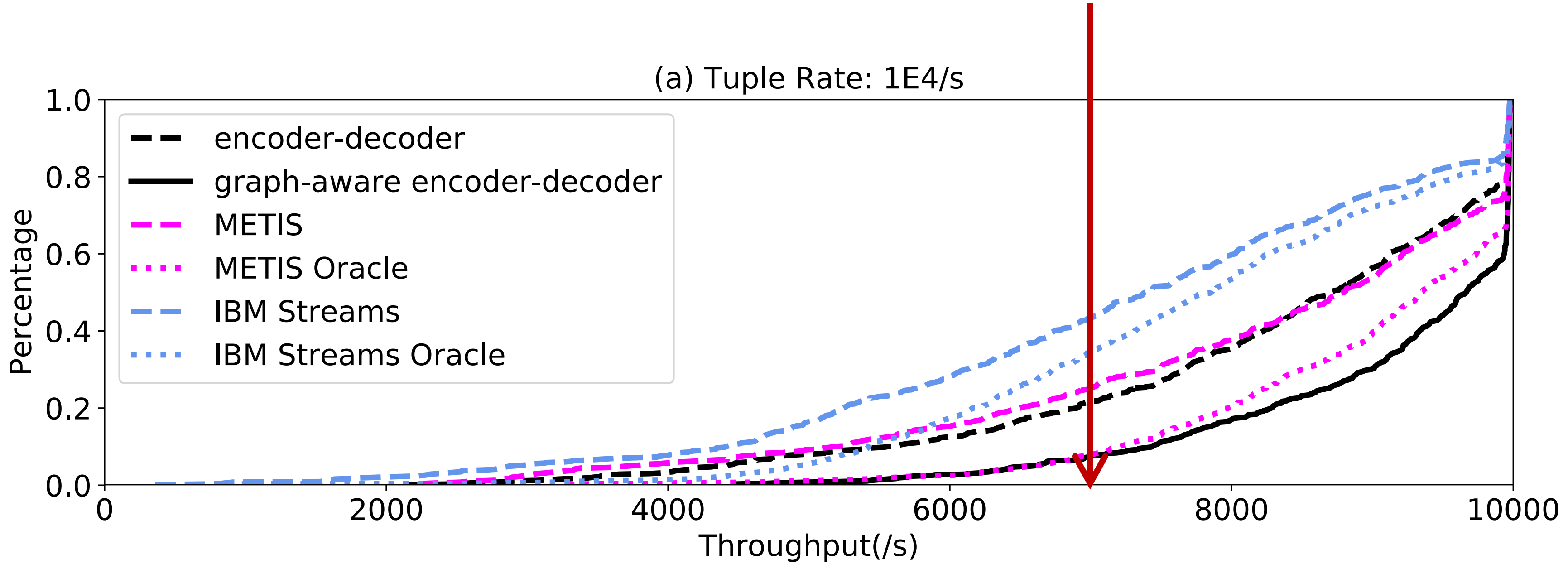
Experiment Setup

- We created create a synthetic benchmark containing 3,150 graphs with various graph topology and number of operators.
- We compare with several baselines:
  - Encoder-decoder is a prior model designed for device placements for a single (Tensorflow) graph.
  - METIS is a heuristic-based graph partitioning method.
  - IBM Streams is a streaming platform used in production.
- Our model can automatically determine the best number of devices to use, while METIS and IBM Streams cannot:
  - METIS Oracle and IBM Streams Oracle exhaustively try out different numbers of devices to use and take the number with the highest throughput.

Evaluation Results

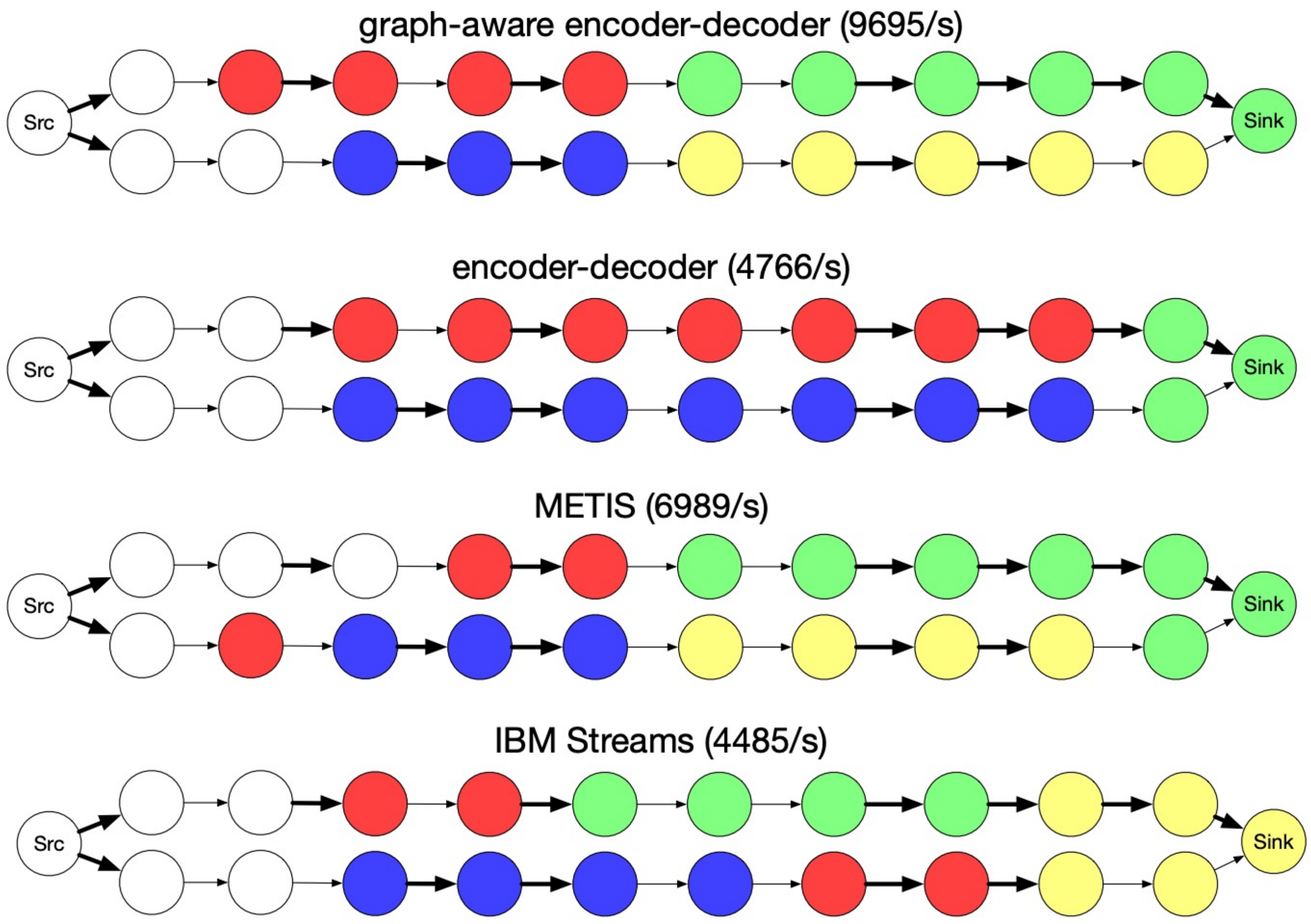
- Cumulative Distribution Function (CDF) of throughputs

IBM Streams: ~40% graphs have throughput < 7000
METIS: ~20% graphs have throughput < 7000
Model: ~5% graphs have throughput < 7000



Qualitative Comparison

- Our model outperforms other baselines by avoiding cutting the heavy edge while balancing the workload in each partition.

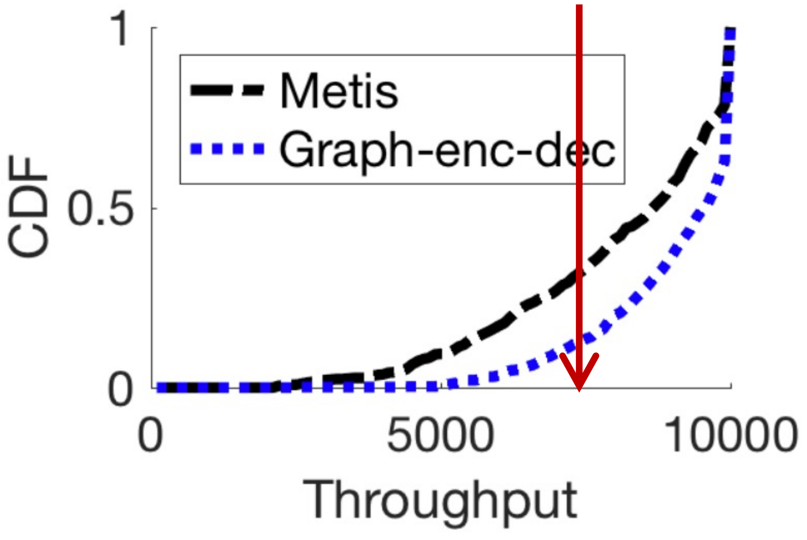


Deficiency for Larger Graphs

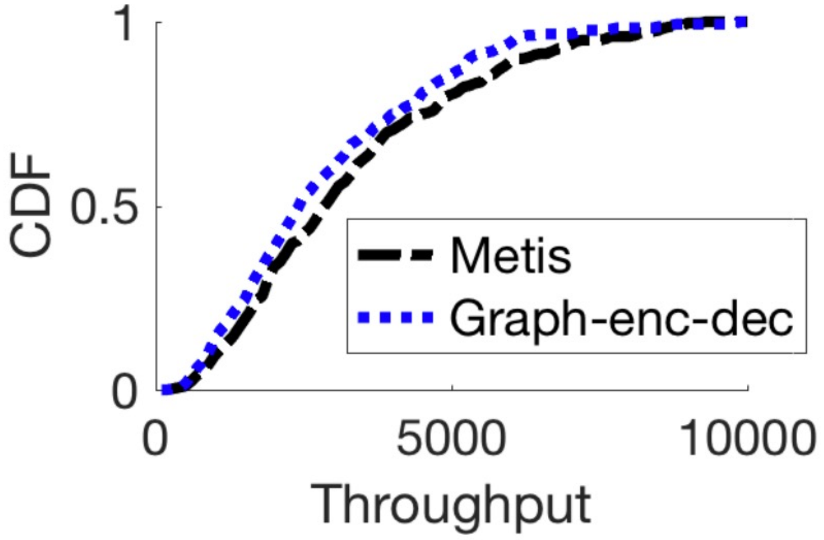
- For large graphs, directly applying graph encoders cannot sufficiently handle long-distance dependency and global information needed for the joint optimization in scheduling.

Metis: ~40% graphs have throughput < 7500

Model: ~15% graphs have throughput < 7500



(a) Small graphs (4∼26 nodes)

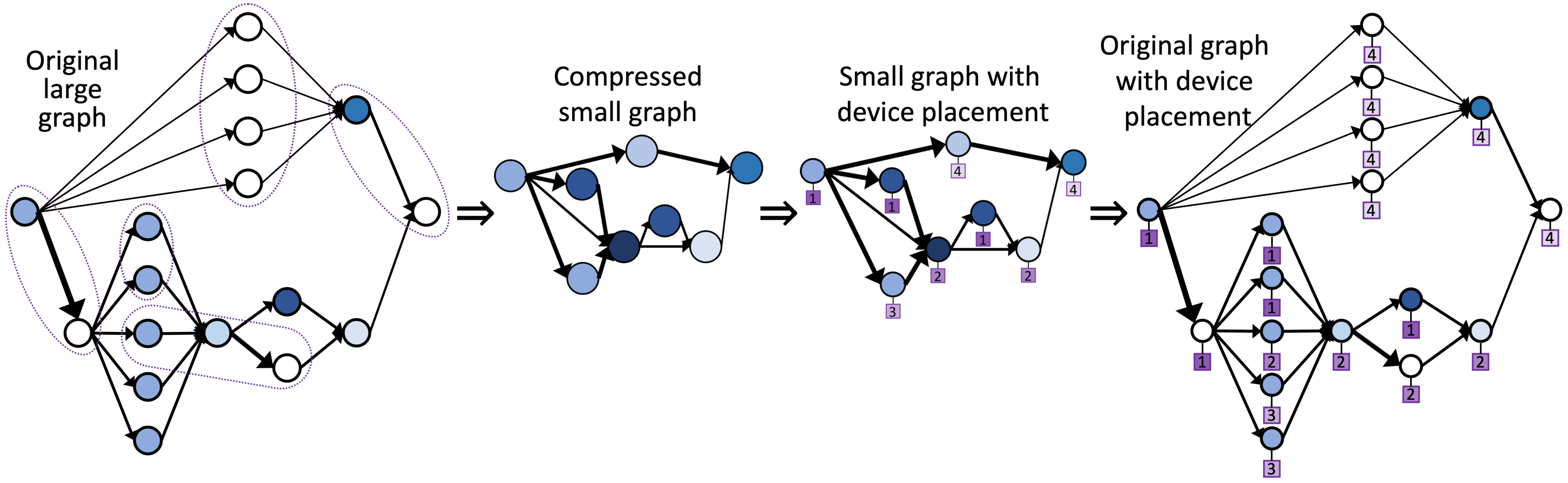


(b) Big graphs (100∼200 nodes)

Cumulative Distribution Function (CDF) of throughputs of 300 stream processing graphs with different topologies

Leverage Theoretical Intuition

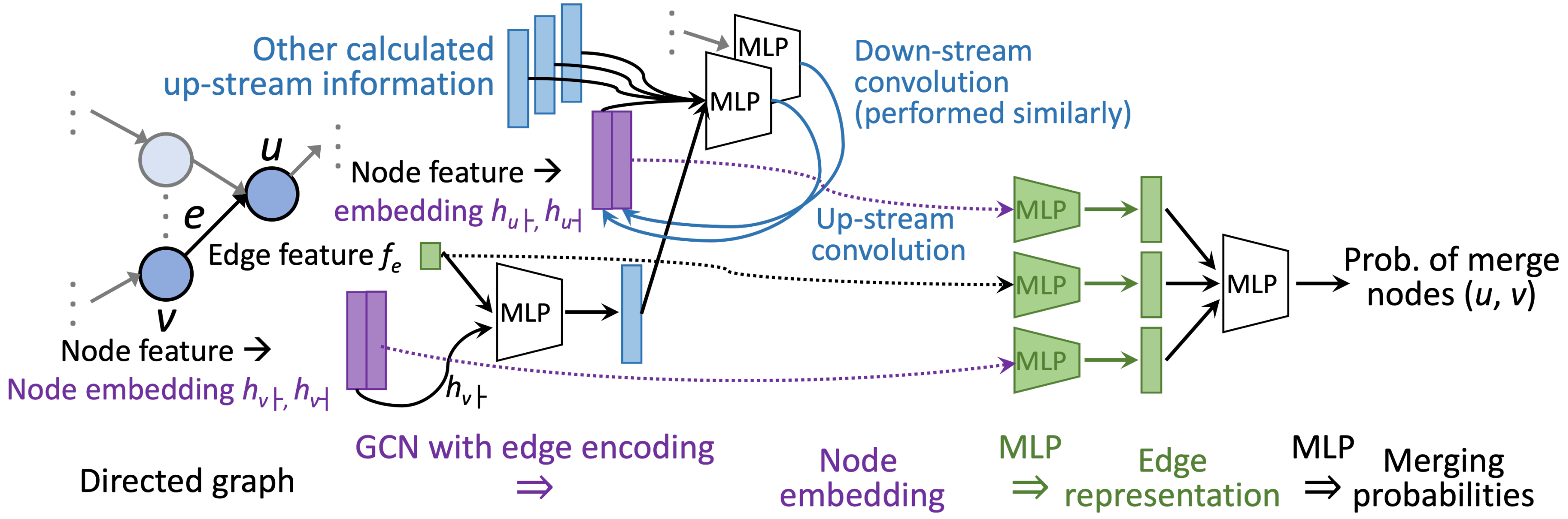
- Communication costs of some edges are more likely to be the bottleneck for maximizing throughputs of large stream graphs.
- Such information can be captured into edge representations without global topological structures and thus easier to learn.


Original large graph
⇒
Compressed small graph
⇒
Small graph with device placement
⇒
Original graph with device placement
1
2
3
4

- The coarsening-partitioning paradigm coarsens a large graph to a smaller one that is easier to handle for a partitioning model.

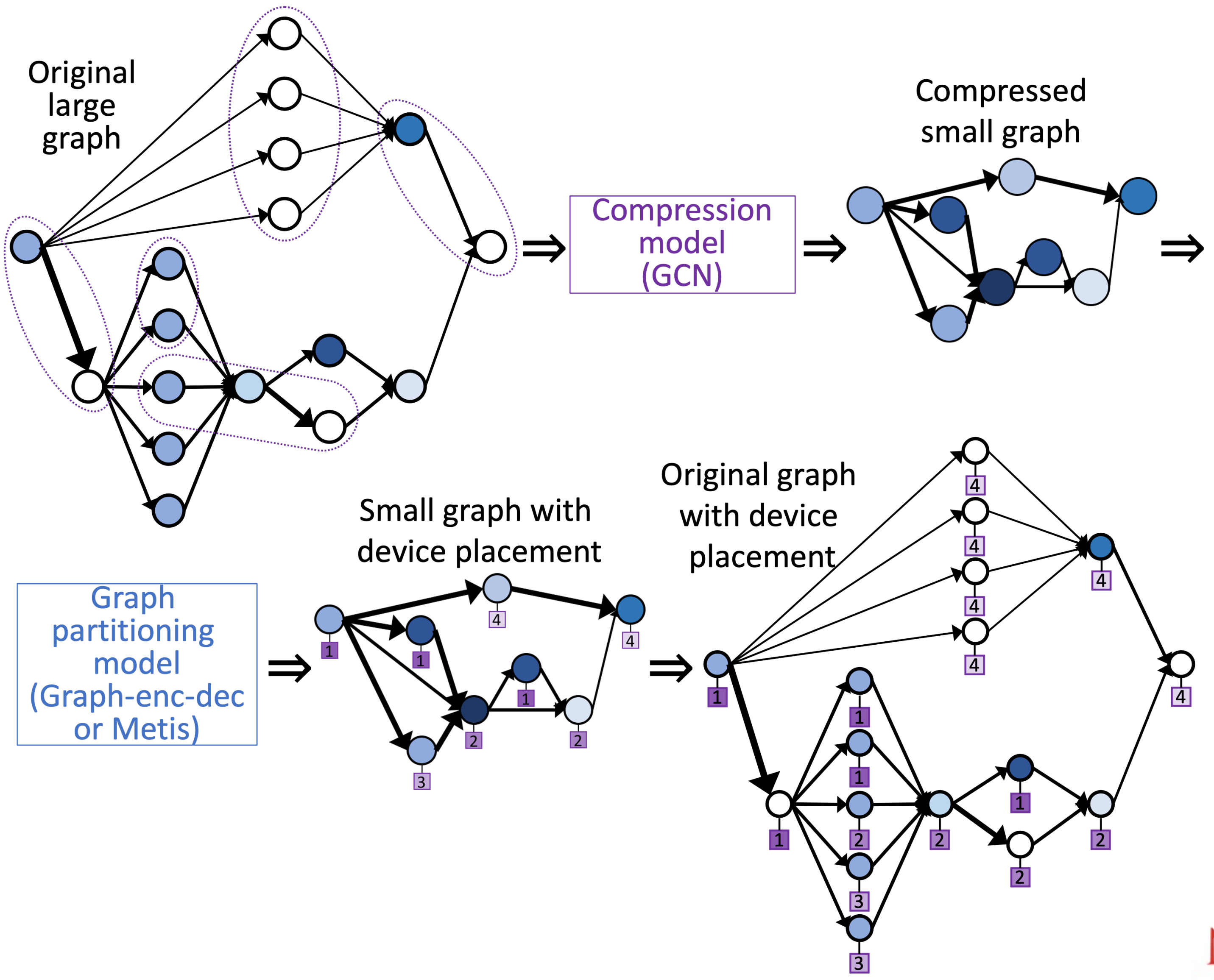
Edge-Collapsing Prediction

- We formulate the graph coarsening problem as predicting whether to collapse an edge connecting two nodes.
- GCN with edge encoding makes better use of the edge information for graph encoding.
- Edge representation combines both node and edge information for the edge-collapsing prediction.



NJIT

Coarsening-Partitioning Framework

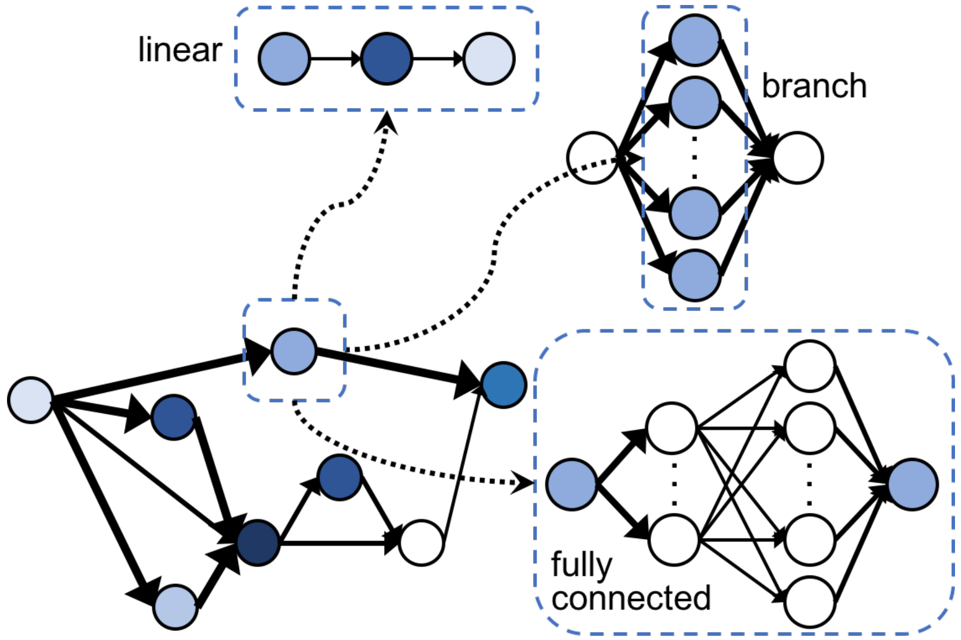

Original large graph
Compression model (GCN)
Compressed small graph
Graph partitioning model (Graph-enc-dec or Metis)
Small graph with device placement
Original graph with device placement

Curriculum Learning for Large Stream Graphs

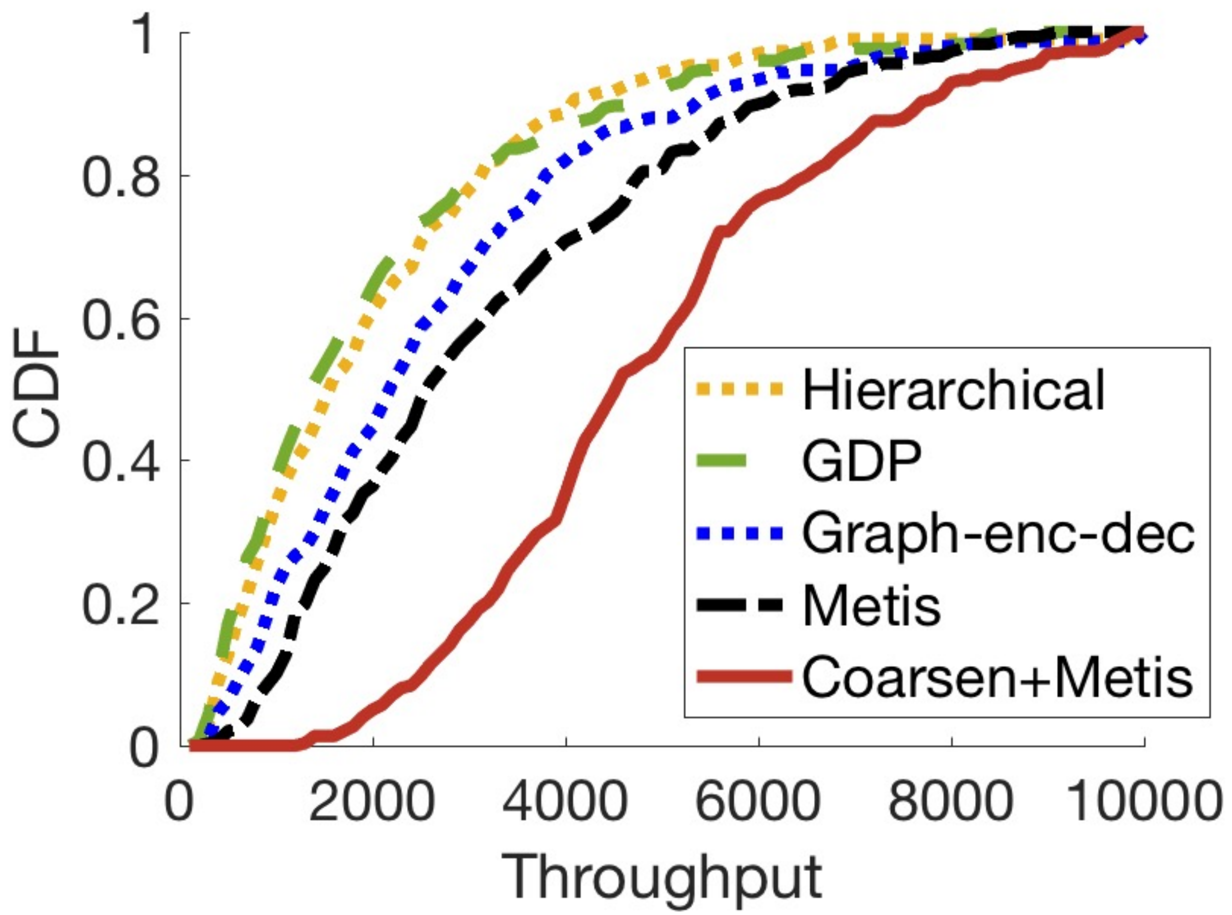
- RL training directly on large graphs suffers from cold start.
- Two curriculum learning techniques to guide our model training and help the model reach convergence faster:
  - Curriculum based on the levels of graph sizes and device numbers
    - Use the model obtained for the previous level to continue training (i.e., fine-tuning) this model for the next level
    - e.g., 100~200 on 10 devices → 400~500 on 10 devices → 1,000~2,000 on 20 devices
  - Heuristic-guided training signals

Data Set Generation

- The stream graphs with various sizes are generated to resemble the topological structures of real-world applications in stream processing systems.
- Three basic types of stream subgraphs (linear, branch, and fully connected structures) are used to recursively generate more complicated processing logic, such as linear chains, loops, trees, and multi-stages.

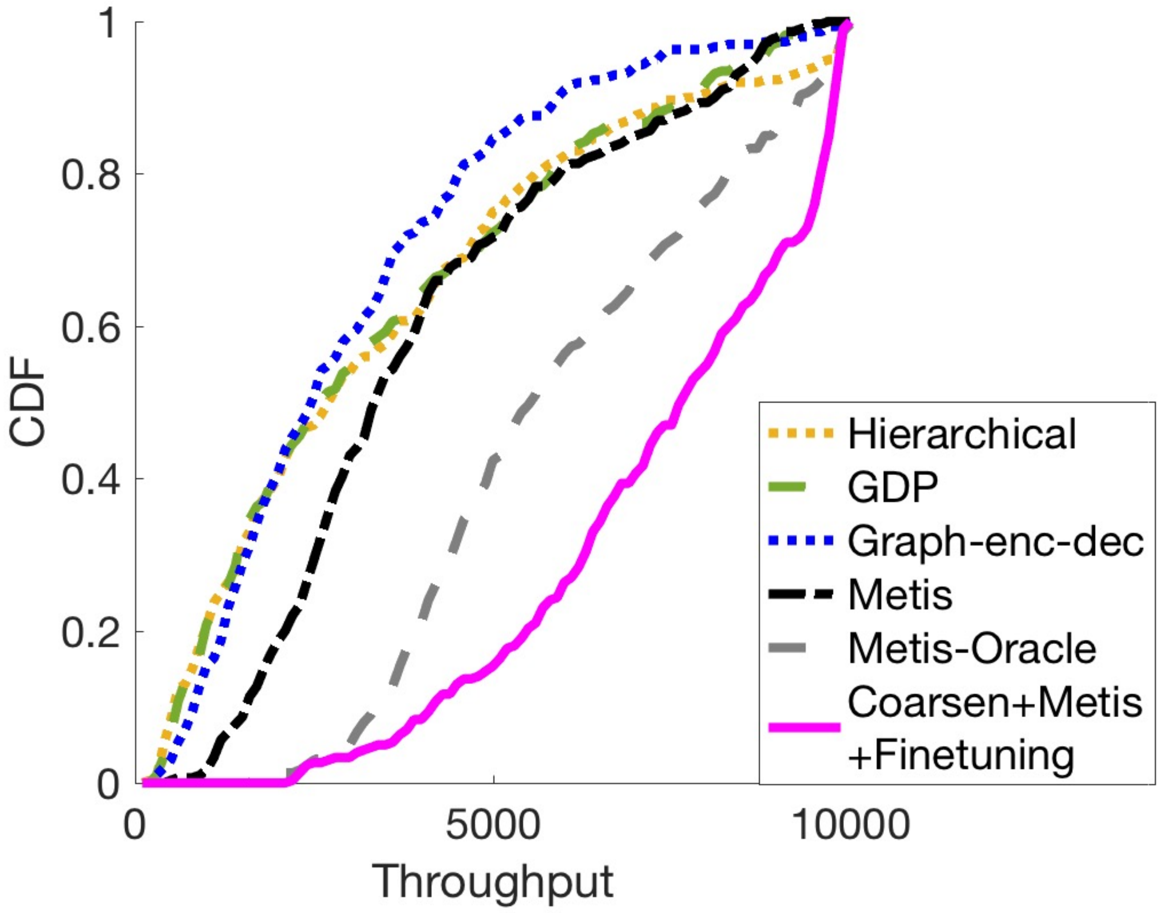


Evaluation Results



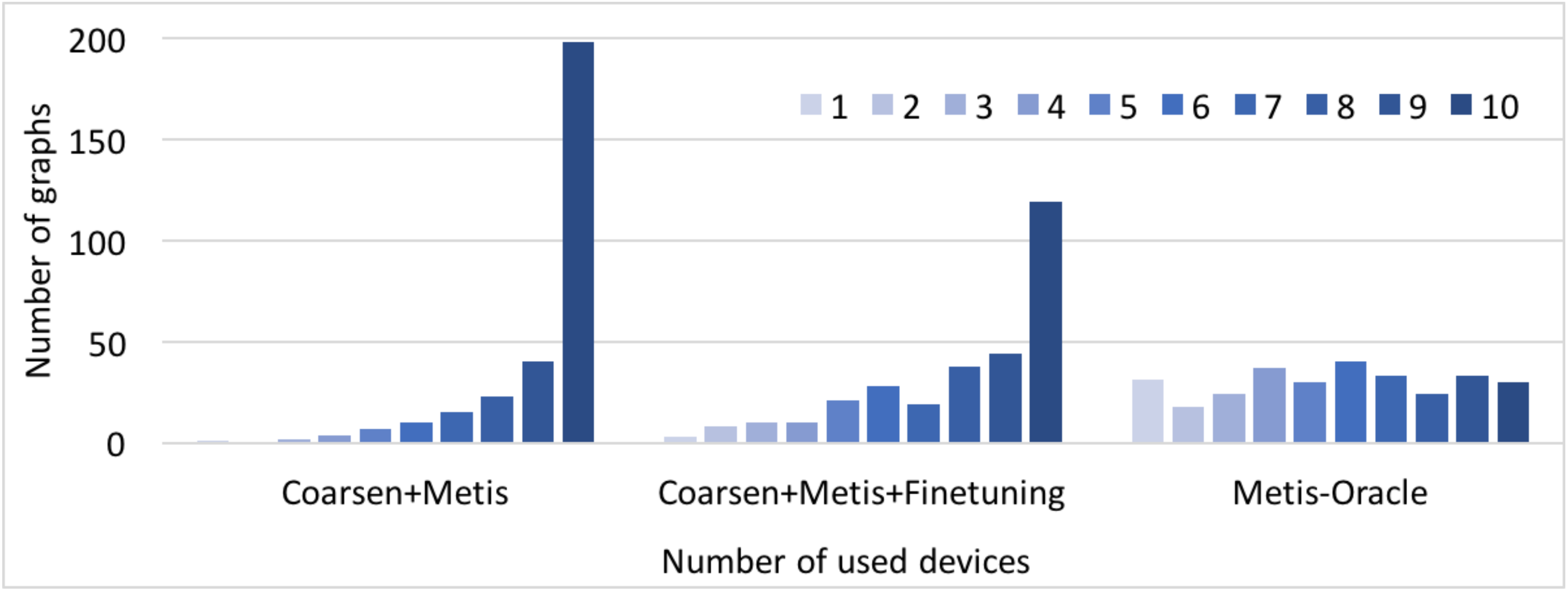
Throughput Cumulative Distribution Function (CDF) under our approach and various baseline methods in settings with 100~200 nodes per graph

Evaluation Results



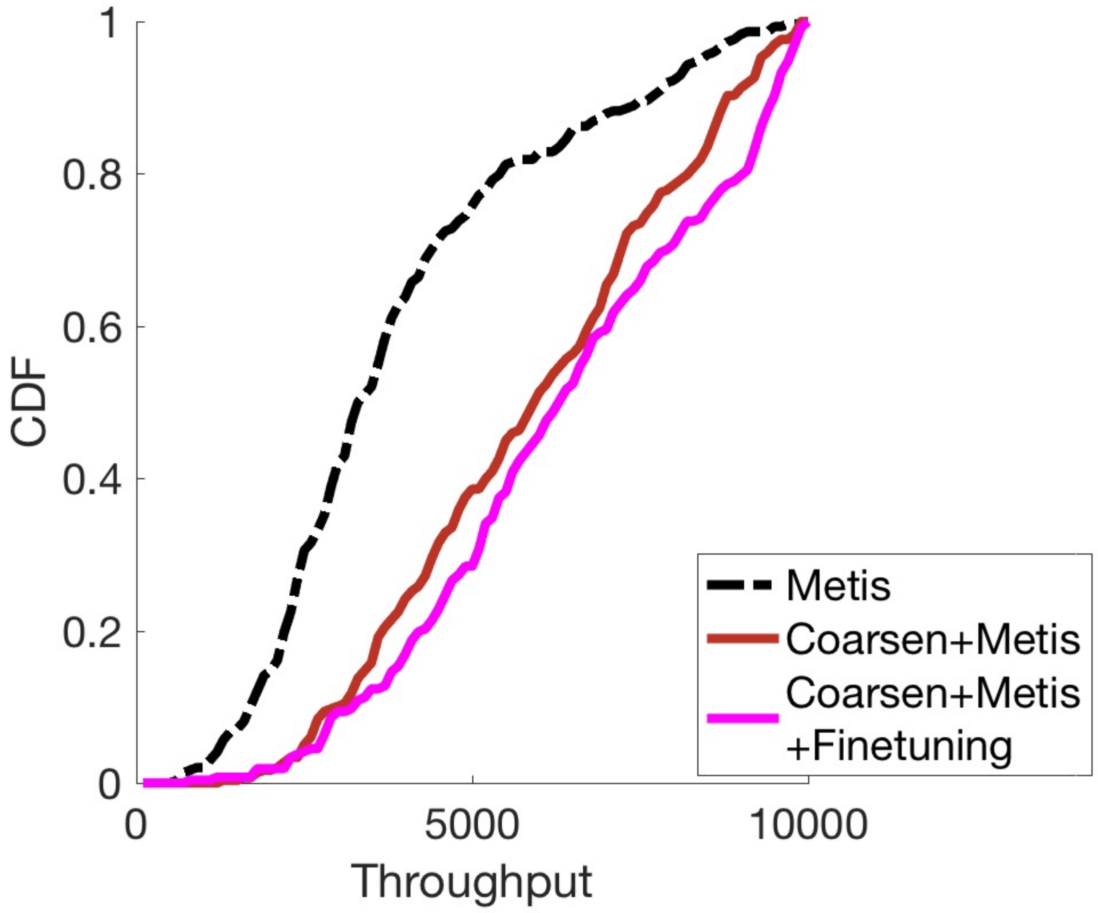
Throughput Cumulative Distribution Function (CDF) under our approach and various baseline methods in settings with 400~500 nodes per graph

Evaluation Results



Device usage histogram for data with 400~500 operators per graph and 10 devices.

Evaluation Results



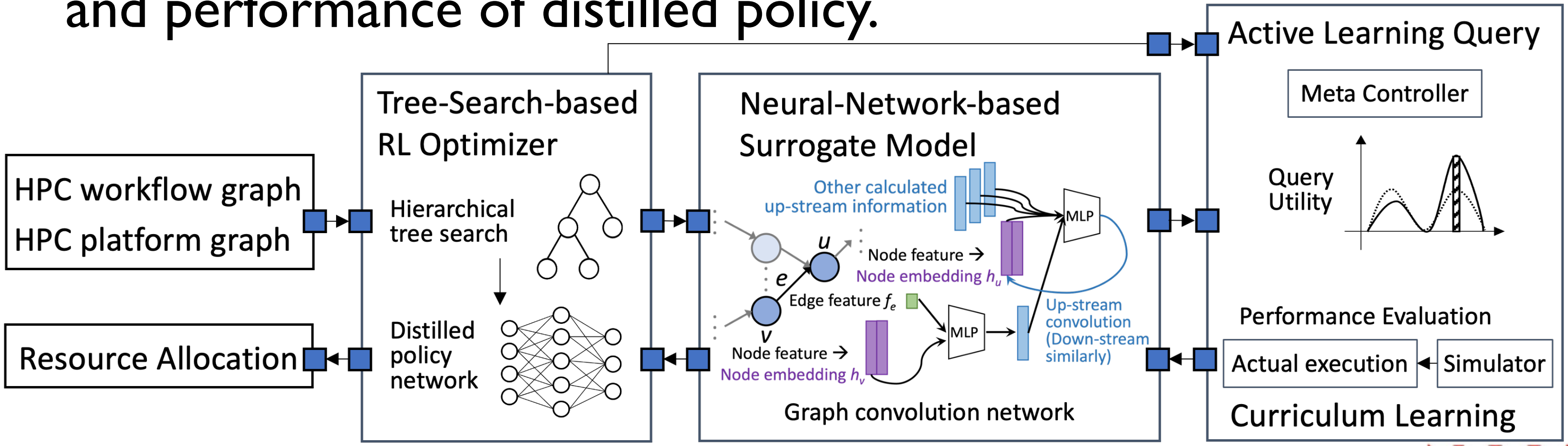
Model trained by data with 400∼500 operators per graph and 10 devices, while evaluated by data with 1000∼2000 operators per graph and 20 devices.

Lessons Learned

- Resource allocation for unseen workload is a combinatorial optimization problem, which is challenging for training the model and searching for optimal allocations.
- Off-the-shelf machine learning models do not work well, so we must incorporate the theoretical scheduling intuitions into problem formulation and model design.
- Model training requires huge training data, but obtaining the actual performance of a workload given a specific allocation is time-consuming and costly.

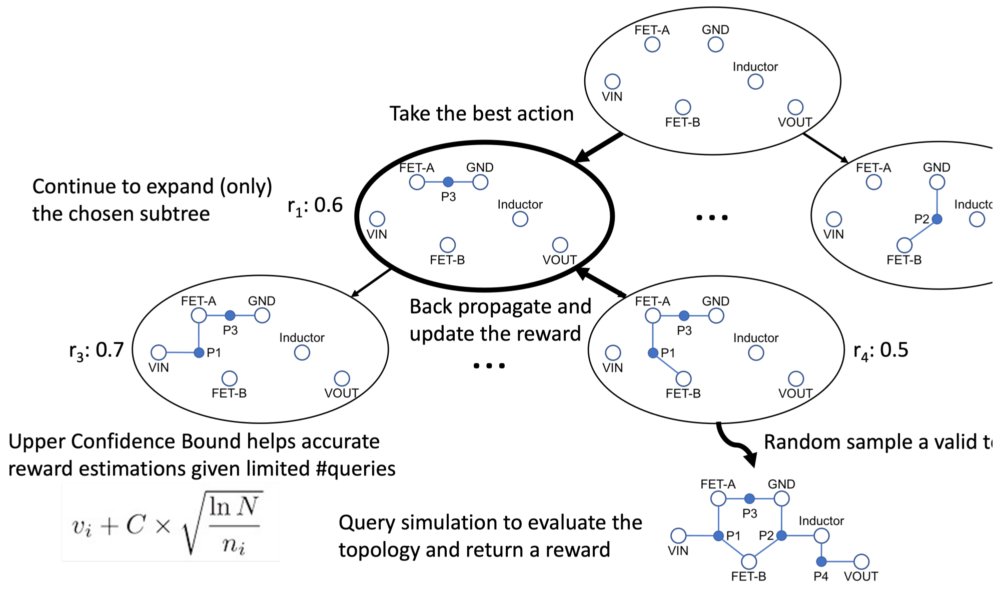
Develop an Intelligent Scheduling Framework

- Design a tree-search-based approach to find high-quality resource allocations and distill the knowledge into policy.
- Devise deep learning-based surrogate models for fast evaluation of the quality of allocations without executing workflows.
- Develop active learning and curriculum learning strategies to guide the acquisition and usage of additional real execution data to enhance accuracy of surrogate model and performance of distilled policy.

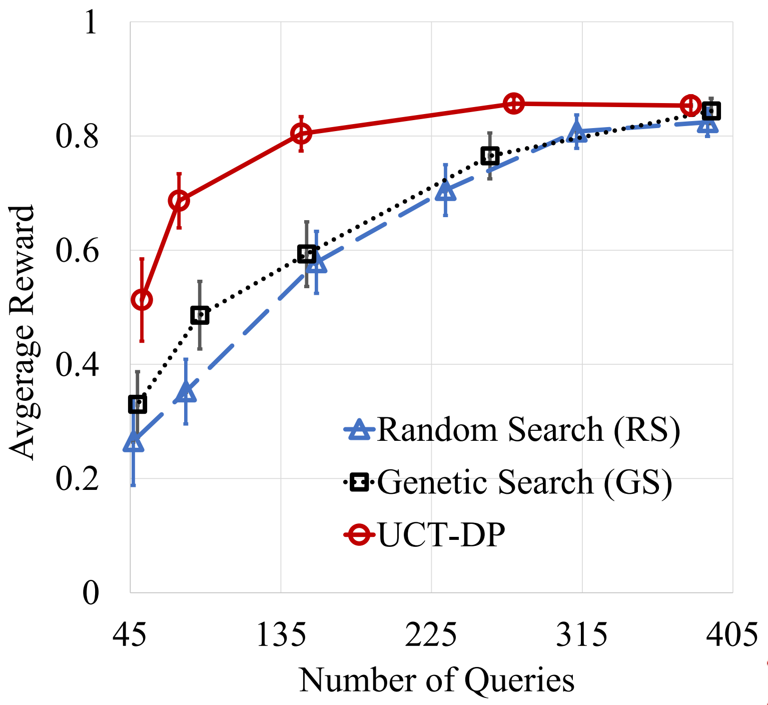


Tree-Search-Based Approach

- We have applied the tree-search-based approach on a circuit design task and achieved success [ICCAD'21].
- For scheduling, how to design the tree structure to sufficiently capture the semantic of topologies for efficient exploration?

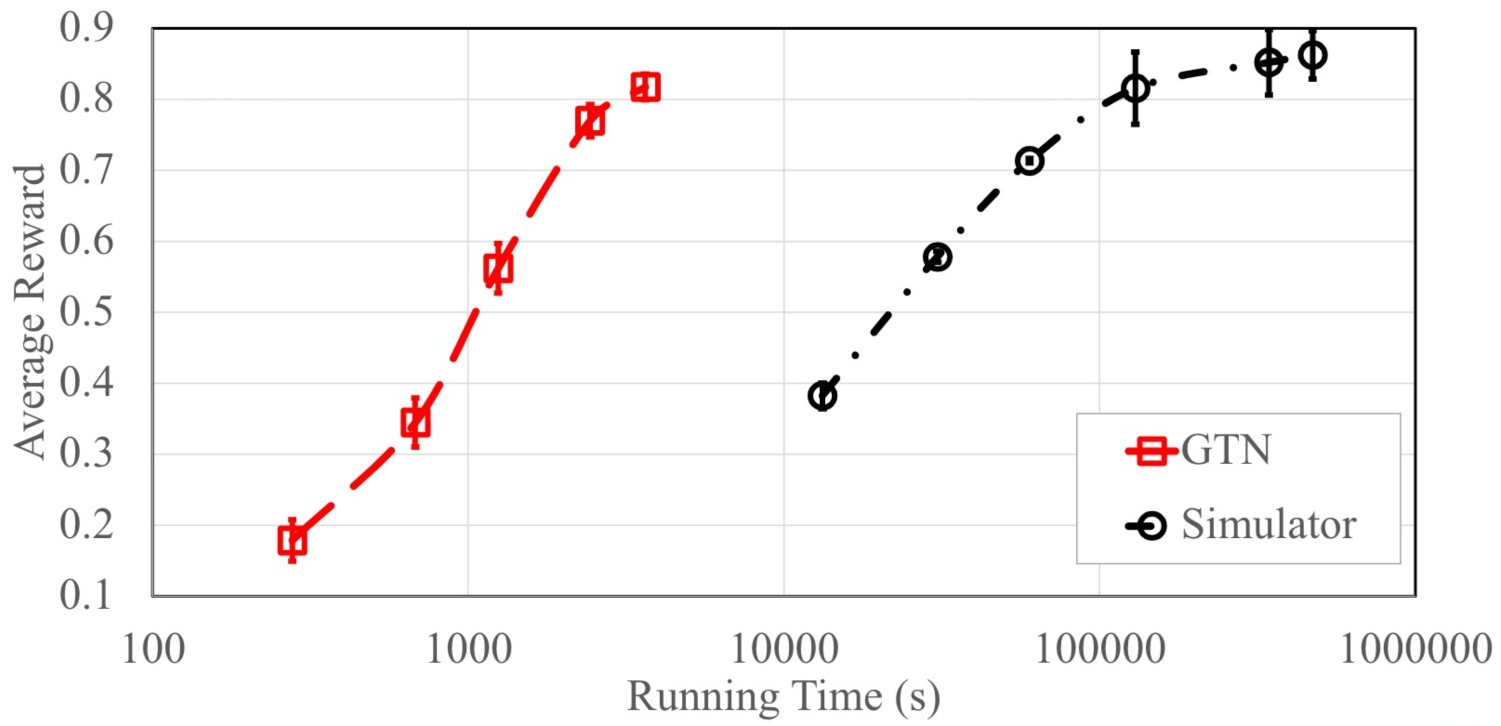

Take the best action
Continue to expand (only) the chosen subtree
r_1: 0.6
Back propagate and update the reward
r_3: 0.7
r_4: 0.5
Random sample a valid t
Upper Confidence Bound helps accurate reward estimations given limited #queries
Query simulation to evaluate the topology and return a reward
FET-A
GND
Inductor
VIN
FET-B
VOUT
P1
P2
P3
P4

$$v_i + C \times \sqrt{\frac{\ln N}{n_i}}$$



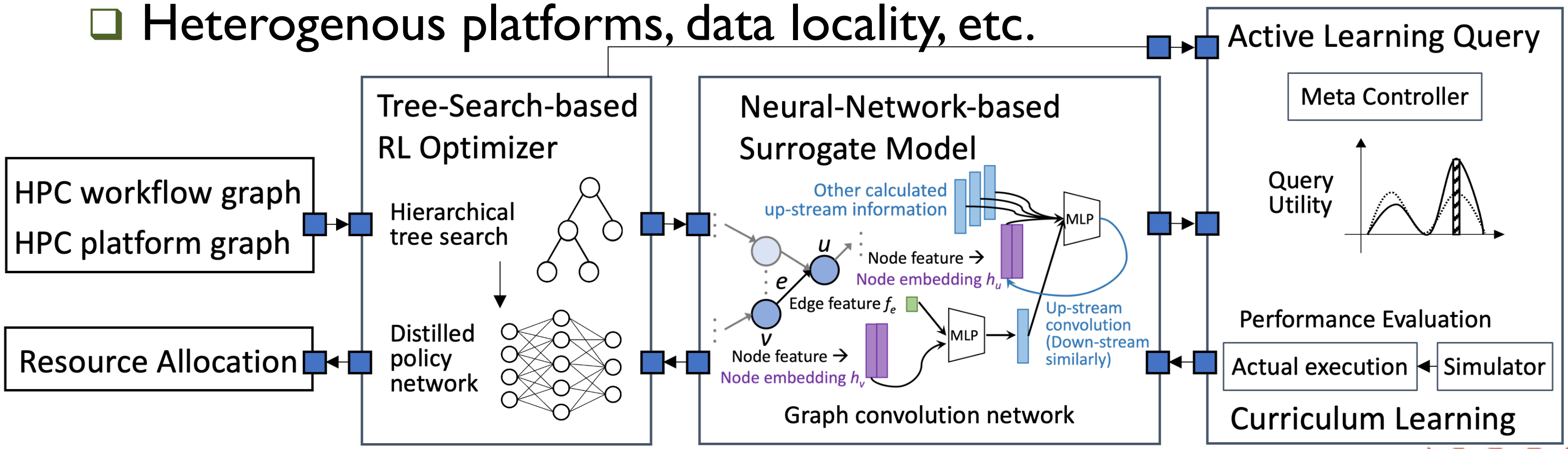
Neural-Network-Based Surrogate Model

- We have developed a graph-transformer-based surrogate model for a circuit design task and achieved success [DAC'24].
- For scheduling, how to structure the model to capture the important scheduling information underlying diverse topologies?



Scheduling in HPC and Supercomputers

- Different levels of scheduling and resource allocations
  - Allocate resources to multiple workflows
  - Distribute workload of each workflow on its allocated resources
- Different performance objectives and workload/system characteristics
  - Minimize latency, meet deadline, minimize energy consumption, resilience considerations, data storage constraints, etc.
  - Heterogenous platforms, data locality, etc.



Thank you

NJIT New Jersey Institute of Technology