Device Placement Optimization with Reinforcement Learning

Presentation by: Arav Agarwal

Overview

How do we optimally map operations onto hardware in DL applications?

Overview

CONV 5x5

MEAN POOL

FULLY-CONNECTED 1

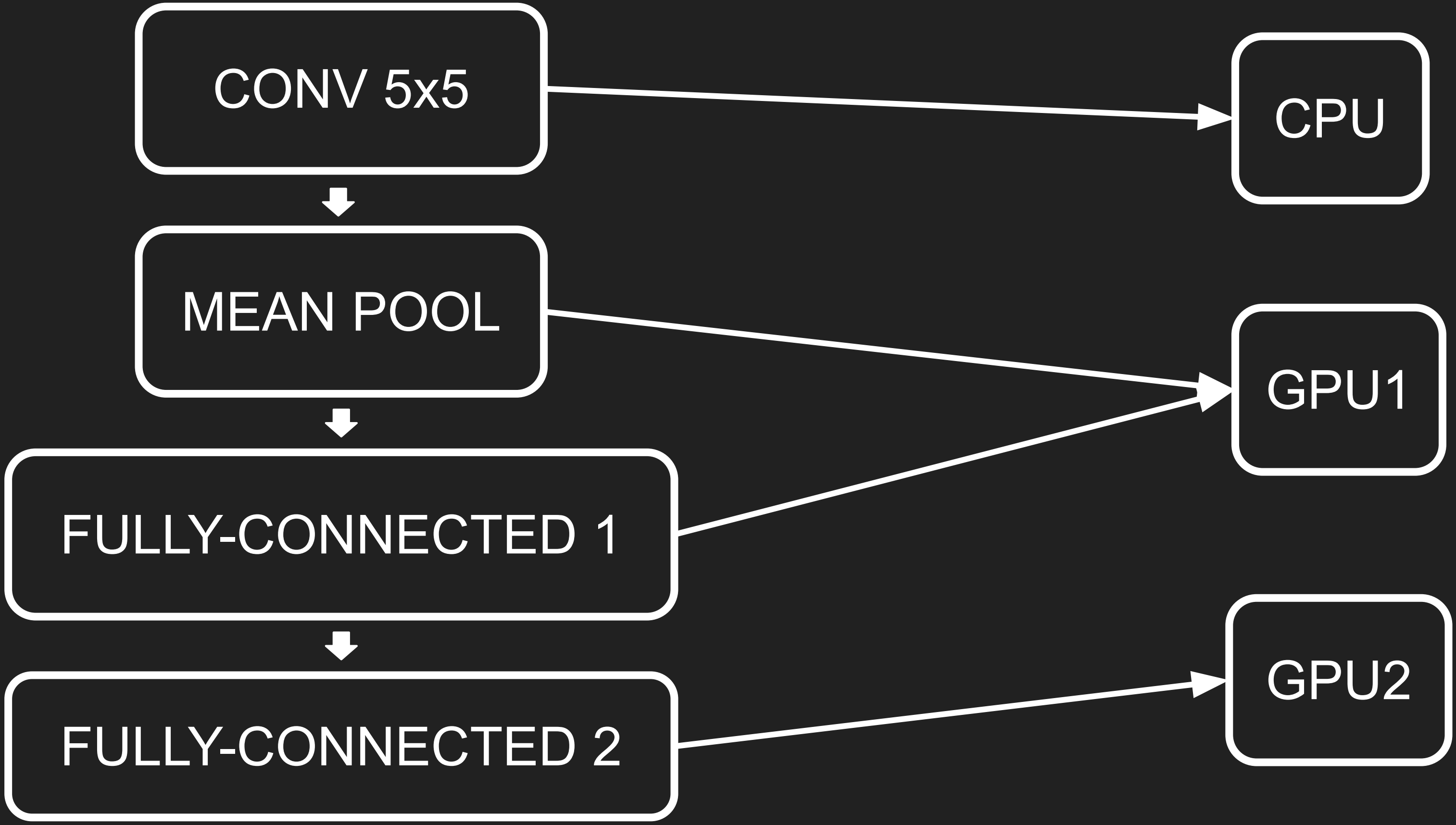
FULLY-CONNECTED 2

CPU

GPU1

GPU2

Overview


CONV 5x5
MEAN POOL
FULLY-CONNECTED 1
FULLY-CONNECTED 2
CPU
GPU1
GPU2

Problem & Challenges

- How can we optimize this problem, given that it’s a combinatorial optimization problem, in some guided way?
- How can we find novel solutions to this problem, in a way guided only by the runtime (i.e. what we care about?)

Methods and Contributions

IDEA: Use Reinforcement Learning!

ENVIRONMENT

1. Treat the process of assigning operators to devices as a sequence of interactions with a computational environment
2. Reward models which reduce the total execution time, forcing them to optimize our objective

MODEL

1. Given a placement of one operator on one device to another, place another.
2. Improve placement given this reward signal iteratively.

Methods and Contributions

ENVIRONMENT DETAILS

- Using generic runtime measurements to try to measure progress is problematic as it’s inherently a noisy process.
- Instead, authors choose to use sqrt(runtime), as that can better differentiate low runtimes.
- Treat this optimization process as a learning task with a sparse signal, only telling the model the reward at the END.
- When a placement is infeasible, set the runtime to a large constant.

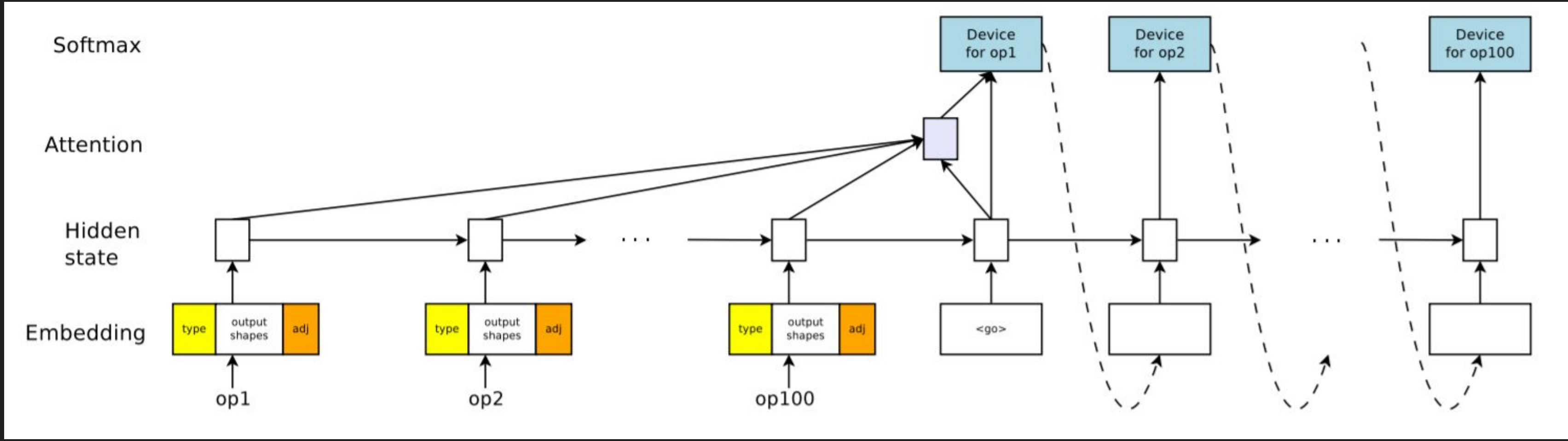
Methods and Contributions

MODEL DETAILS

- Use the REINFORCE gradient estimator with a Moving Average Baseline of RunTime to optimize an LSTM encoder-decoder.
- Feed in specifically tuples of (Operation Type, Output Shape(s), and OHE-encoded Adjacency Information)

Methods and Contributions

MODEL DETAILS



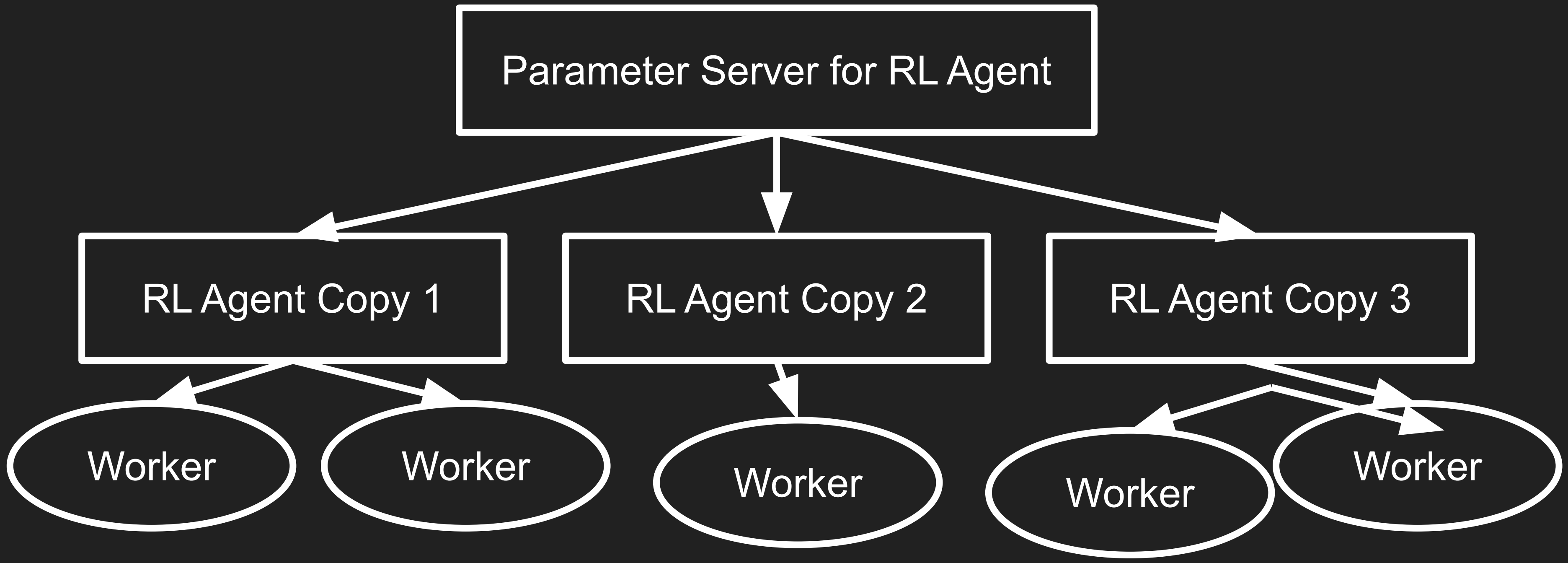
Methods and Contributions

REDUCING COMPLEXITY

- With thousands of operations and noisy training, optimizing everything at once can be a challenge.
- Authors decide to reduce complexity from placing all devices to placing **colocation groups**, i.e. groups of operators which they pre-define as being on the same device.
    - Follow TensorFlow’s Co-Location Groups
    - Follow the heuristic that, is A->B is a portion of the graph, then (A,B) is a colocation group.

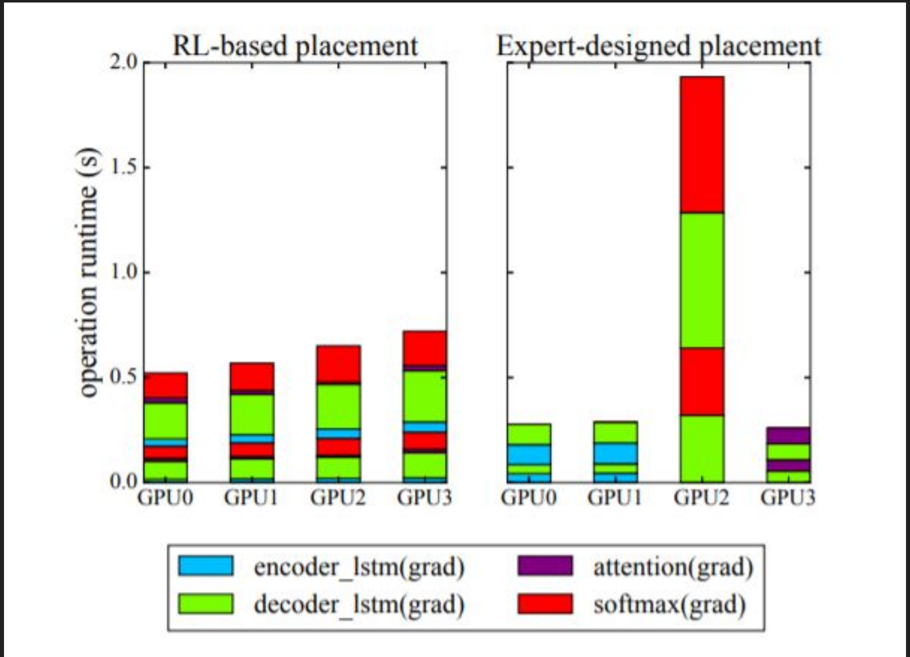
Methods and Contributions

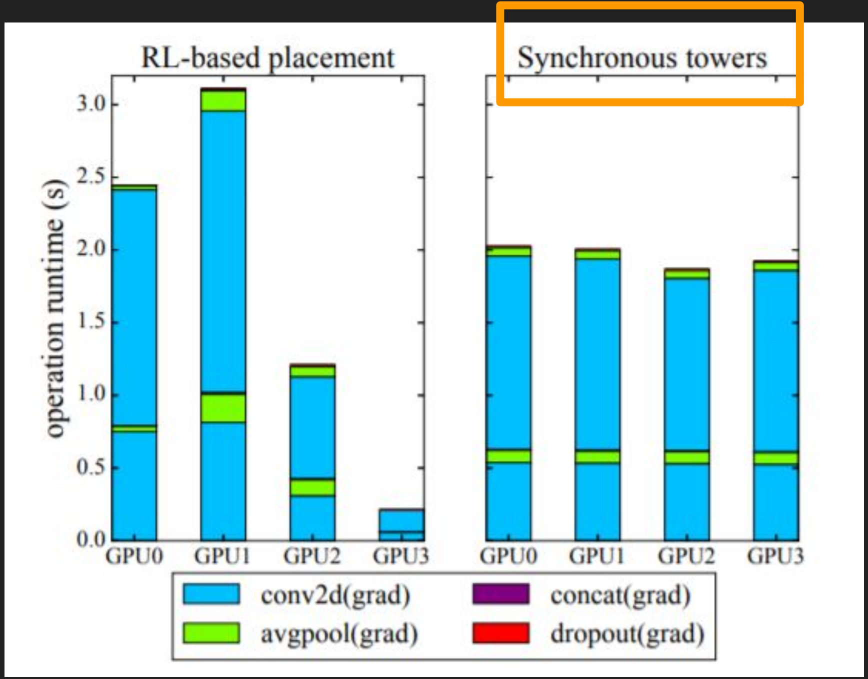
DISTRIBUTED TRAINING



Results and Discussion

Improve Upon Baselines





Results and Discussion

- Colocation groups seem to work, but there was no ablation analysis comparing the results of using other colocation groups instead.
- Modelling concerns dealing with RNNs are not addressed, but could be an easy way to extend this research
- No way for the network to attain easy “complete” information of the graph, nor is there information about the memory size required to host the particular parameters inputs.

Questions

- How can we more intelligently model this problem of assigning operators to devices? Currently, the way they encode the inherent graph topology of the network is naive; can we turn this into some graph-coloring problem?
- Part of how this system is assessed is through the measure of how well it “balances load across GPUs”, where load is measured as runtime on a GPU. Is this an accurate assessment of load, or is there a better way to incorporate this assumption into the RL framework?
- How can we add communication costs into this framework? Given that splitting some loads across GPUs might be more or less intense compared to others, can we find a good way to incorporate network information into this process?

Carnegie Mellon University

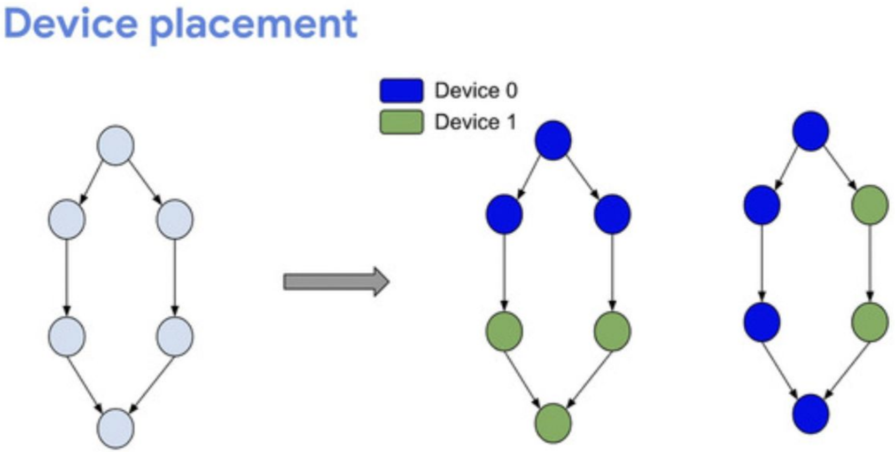
Transferable Graph Optimizers for ML Compilers

Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter Ma, Qiumin Xu, Hanxiao Liu, Phitchaya Mangpo Phothilimthana, Shen Wang, Anna Goldie, Azalia Mirhoseini, James Laudon

Presented by Yuanxin Wang

Feb 16, 2022

Motivation - Graph Optimization Tasks

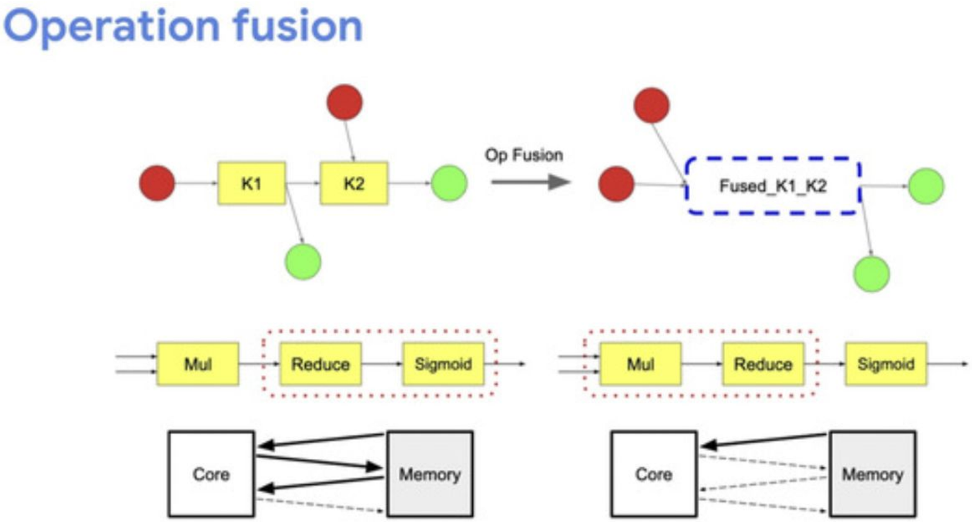


inter-device network bandwidth
peak device memory
co-location constraints
....

Input: Graph
Output: Nxd placement decision

Carnegie Mellon University

Motivation - Graph Optimization Tasks

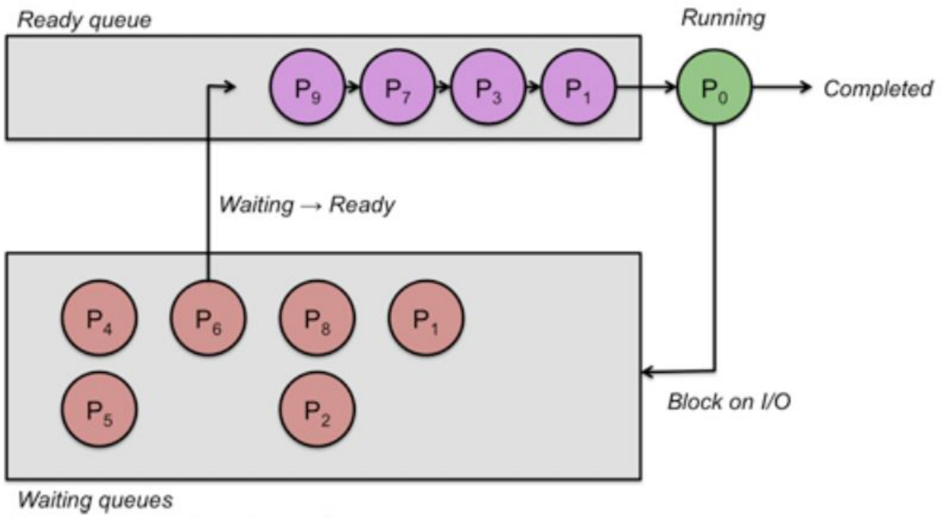


The order and choice of fusion make a difference!

Input: Graph
Output: Nxf fusion actions

Motivation - Graph Optimization Tasks

Operation Scheduling



First Come – First Served

FIFO Ready Queue
Dependency of Ops: different device and communication overhead

Input: Graph
Output: Nxs scheduling actions

Carnegie Mellon University

Motivation - Limitations of Existing Work

- Why not manually optimize using heuristics or search?
  - Suboptimal
  - Hard to develop and maintain

- What's wrong with existing learning based methods?
  - Cannot generalize to unseen graphs and newer architectures
  - Poor sample efficiency
  - Only optimize single task without capturing task dependencies (e.g., placement and schedule)

Goal: Develop **generalizable** end-to-end method for **joint** task optimization

Carnegie Mellon University

Proposed Methods - GO for Single Task

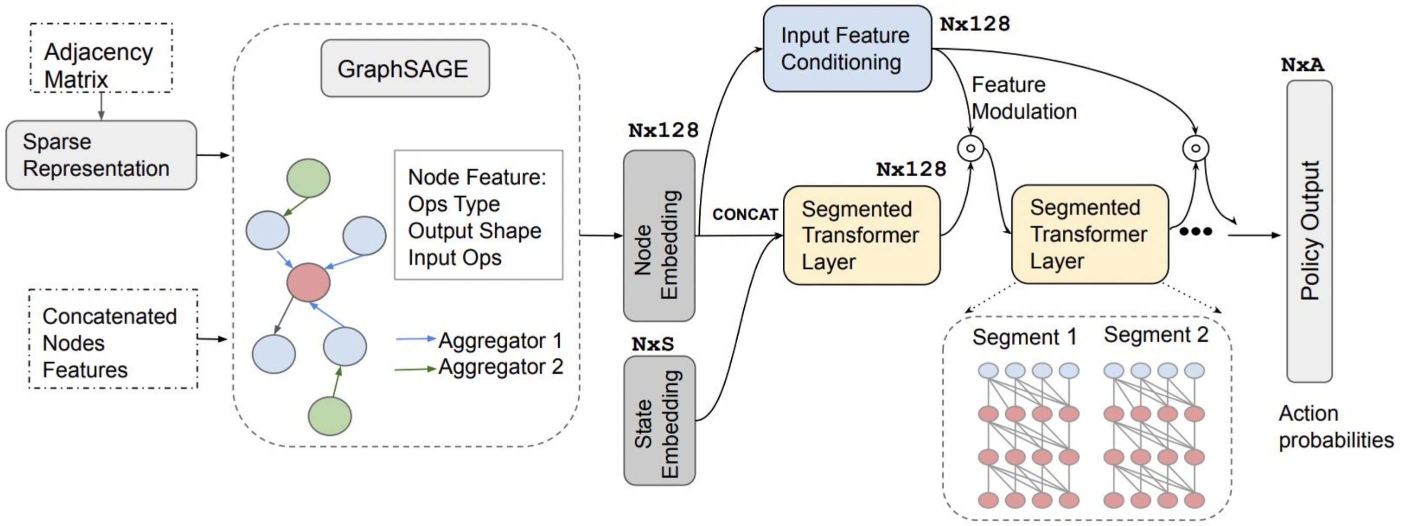
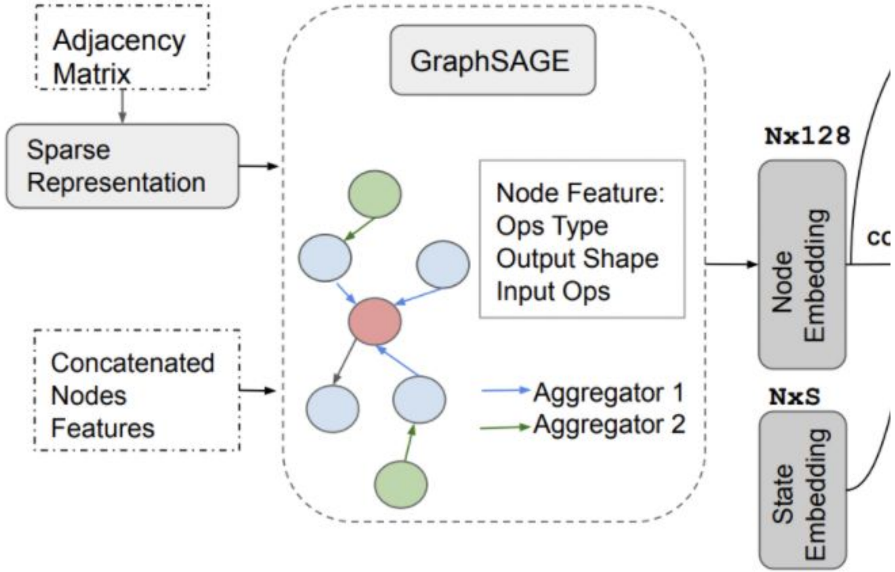


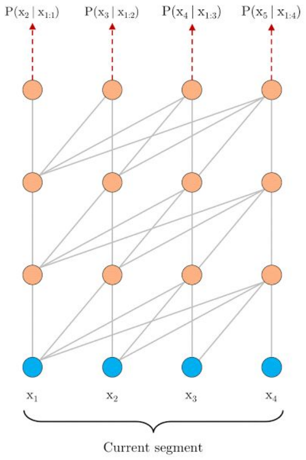
Figure 3: Overview of GO: An end-to-end graph policy network that combines graph embedding and sequential attention. N: Number of Nodes, a: Size of the action space (number of devices, number of priority levels, etc.). Node features are sorted in topological order.

Carnegie Mellon University

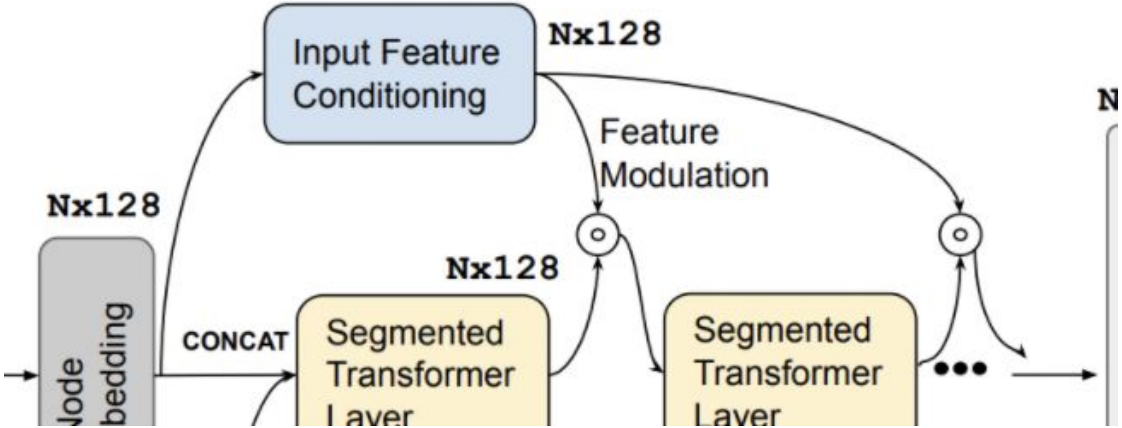
Proposed Methods - Graph Embedding Network



Proposed Methods - Scalable Attention Network



Proposed Methods - Feature Modulation Mechanism

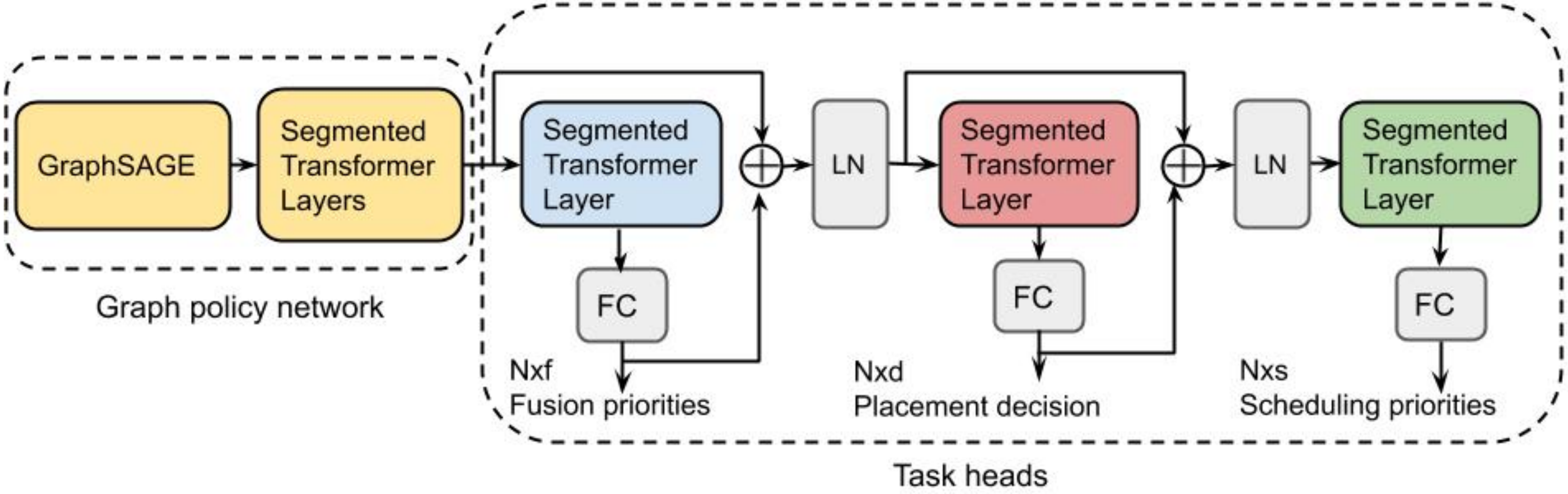


Different domains: CV, NLP, Speech...

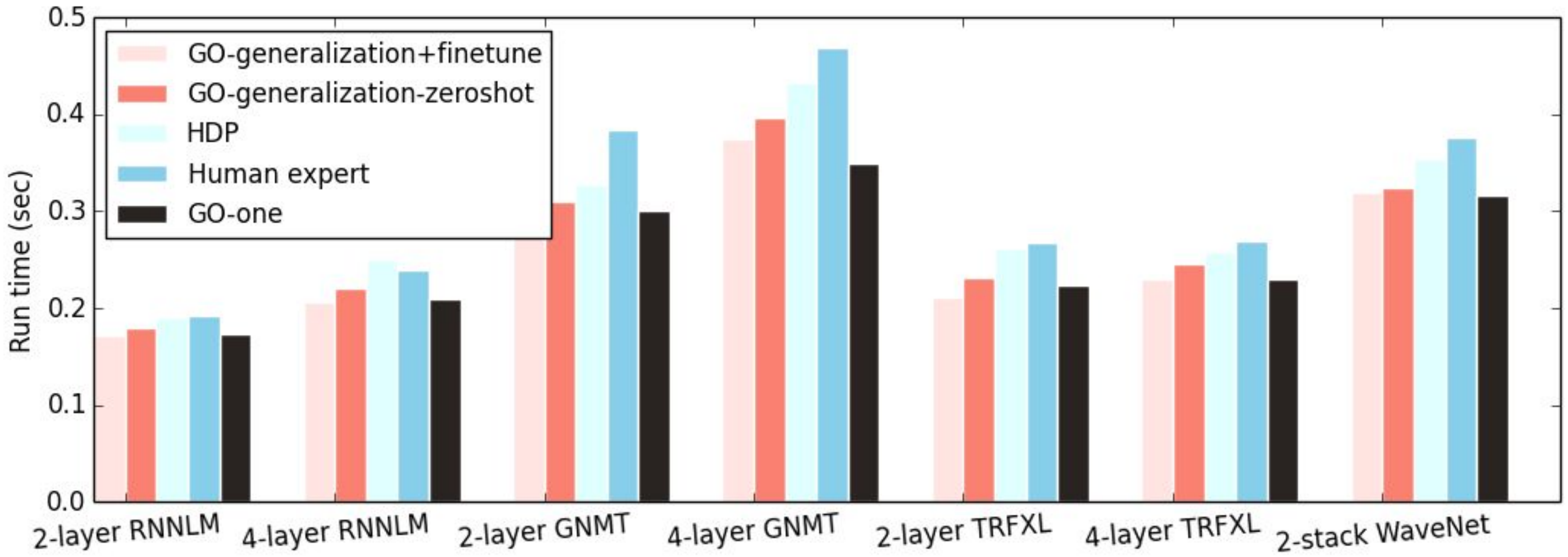
Different architecture and Ops...

Proposed Methods - GO Architecture (Animated)

Proposed Method - GO for Multi Task

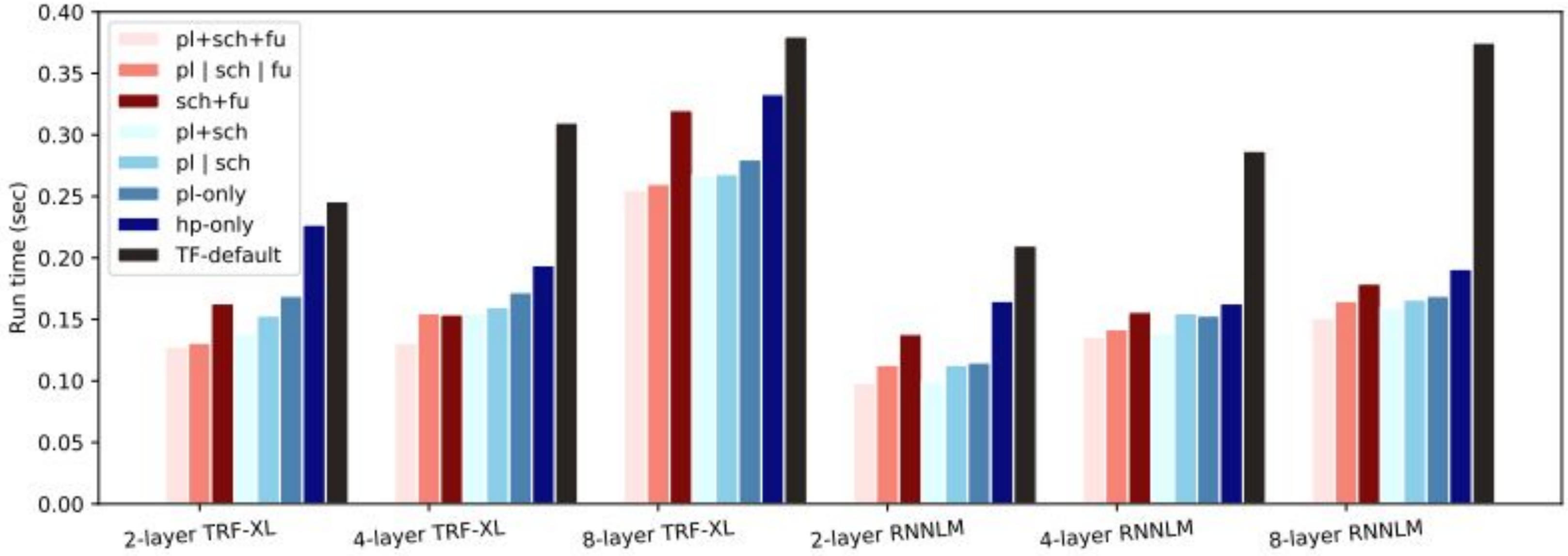


Results - Single Task (Placement)



Carnegie Mellon University

Results - Multi Task (Placement + Scheduling + Fusion)



Carnegie Mellon University

Takeaways and Reflections

Takeaways:

- An end-to-end, RL-based network to jointly improve three graph optimization tasks with significant speed-up and generalization compared to default optimizers.
- Save engineering efforts as well as being more optimal
- Reduce overall carbon footprint
- Evaluate the behavior of benchmark workloads on unseen architectures

Reflection:

- Loss of Explainability

Related New Research

A graph placement methodology for fast chip design from Google Brain and CI2 Team

A Learned Performance Model for Tensor Processing Units from MLSys 21’

HASCO: Towards Agile HArdware and Software CO-design for Tensor Computation from ISCA 21’

Discussion Questions

- Why some learning-based method cannot generalize to unseen graphs?
- Compare and contrast ML vs heuristics based methods for graph optimization (not just accuracy, but speed, maintainability, explainability, etc...). Can we combine the advantages of both?
- Why feature modulation is necessary since we already have the graph embedding?

Carnegie Mellon University

References

Presented paper blog and talk :
https://ai.googleblog.com/2020/12/end-to-end-transferable-deep-rl-for.html

https://crossminds.ai/video/transferable-graph-optimizers-for-ml-compilers-606ff050f43a7f2f827c17ab/

Figure for scheduling:

https://people.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html

Segment-level recurrence in Transformer-XL:
https://ai.googleblog.com/2019/01/transformer-xl-unleashing-potential-of.html#:~:text=positional%20encoding%20scheme.-,Segment%2Dlevel%20Recurrence,processes%20the%20next%20new%20segment.

Carnegie Mellon University

Carnegie Mellon University

Thank you!