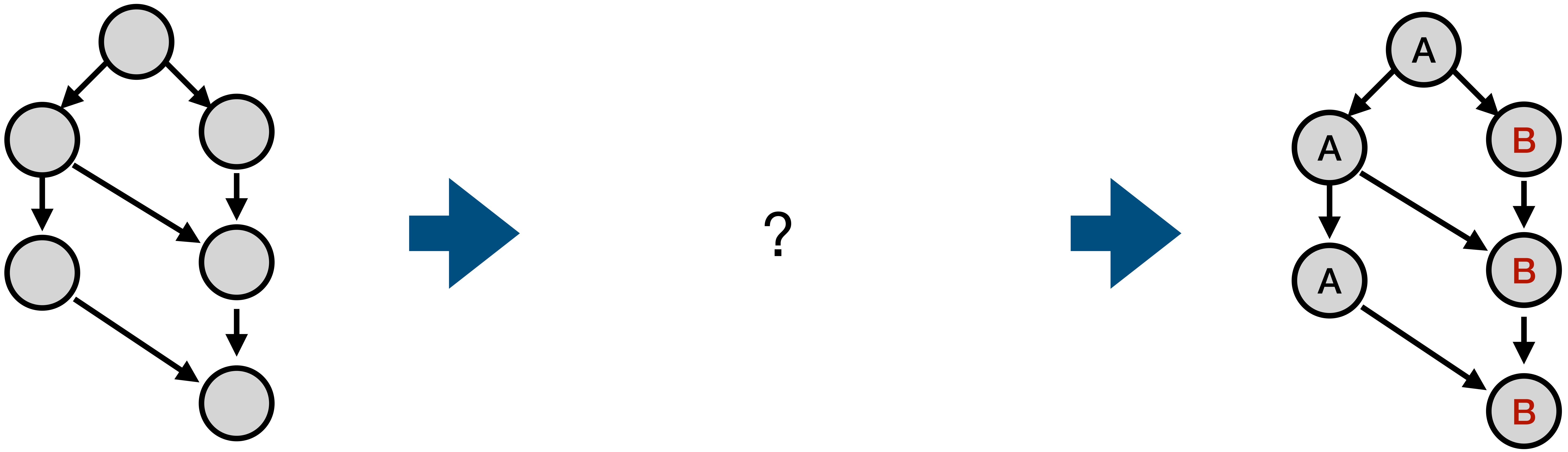
Device Placement Optimization with Reinforcement Learning

Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, Jeff Dean

Zak Singh 22/11/21

Device Placement

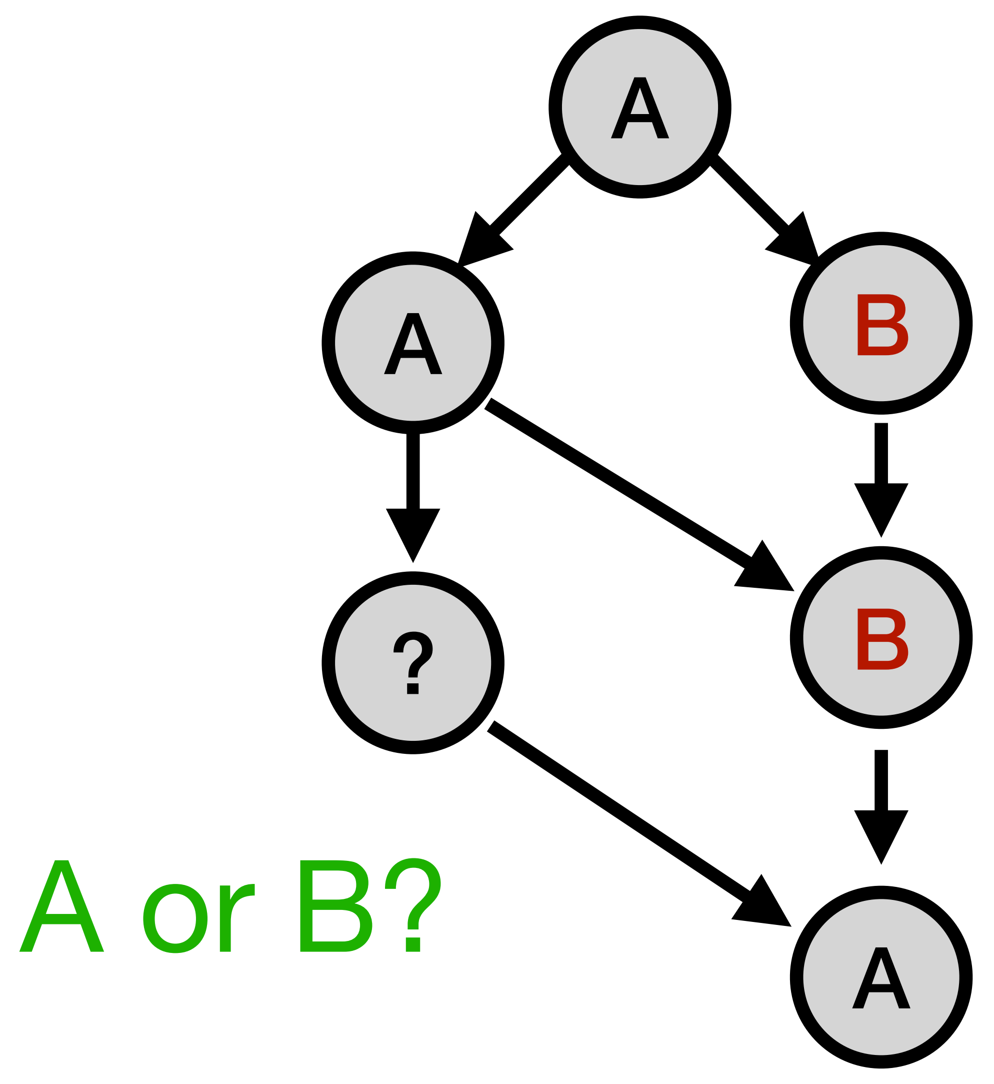


Computation graph

Placement

Placement Challenges

- Heterogeneous clusters (may have mix of CPU/GPUs)
- Traditionally done by a human expert or algorithmic methods (graph partitioning)
    - Unfeasible for complex computation graphs
- Can’t use regular deep learning as our reward (runtime) is non-differentiable -> need to use Reinforcement Learning (RL)
- The placement of a node should take into account the placement of its neighborhood
    - Requires some type of state or *‘memory*' as we place each operation



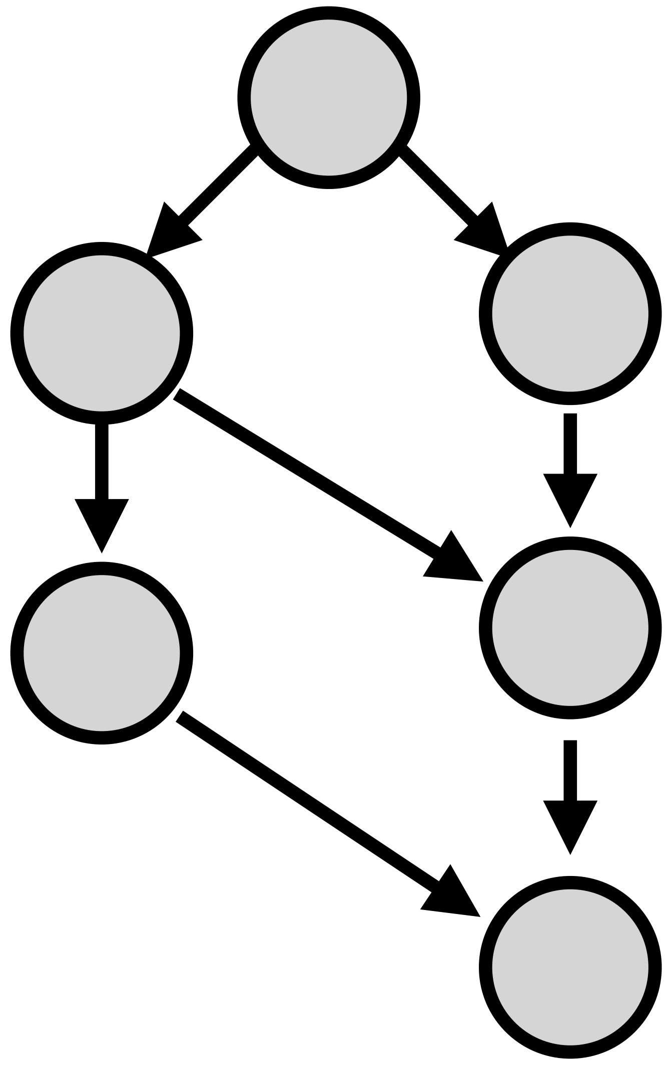
The proposed solution:

Use a **sequence-to-sequence** model as a **RL policy network** to place operations to devices

Sequence-to-sequence

- Ex: translate a sentence in Spanish to a sentence in English
- May not be a one-to-one mapping (English sentence may be shorter or longer than the Spanish one)
- Typically structured with two RNNs:
    - “Encoder” network takes our Spanish sentence and converts it to a latent representation
    - “Decoder” network takes the latent representation and converts it to English

Sequence-to-sequence in our domain



Encoder RNN

- Maps operators to latent space:
  - Type (MatMul, conv2d) +
  - Size of operation’s output tensors +
  - Adjacency information

Decoder LSTM

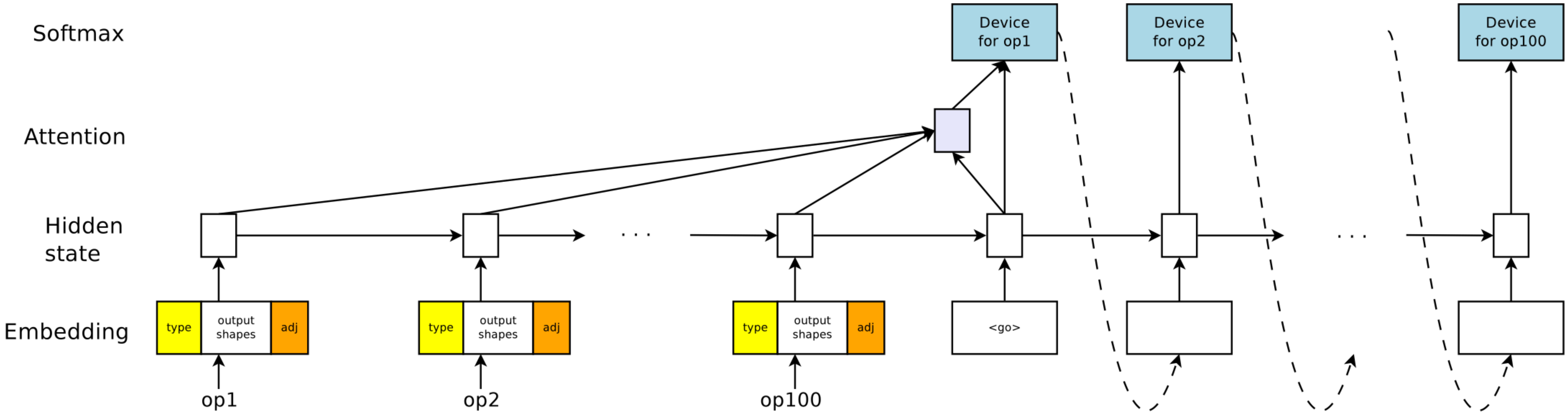
- Fixed number of timesteps equal to the number of nodes
- At each step, output the device for the operator corresponding to that timestep
- This assignment is then fed as input to the next decoder timestep

Recurrent Neural Networks & LSTMs

- RNNs maintain internal state, allowing information from past inputs to stay present over time
  - Does so by having cycles which feed activations from prior time step as inputs to the network
  - Often used for sequence data: NLP, speech recognition, financial trading, etc.
- **Problem**: RNNs fail to learn when there are large gaps between the relevant input event and target signal (e.g. more than 10)
  - Vanishing/exploding gradient as inputs cycle through the network's recurrent connections

LSTMs handle this!

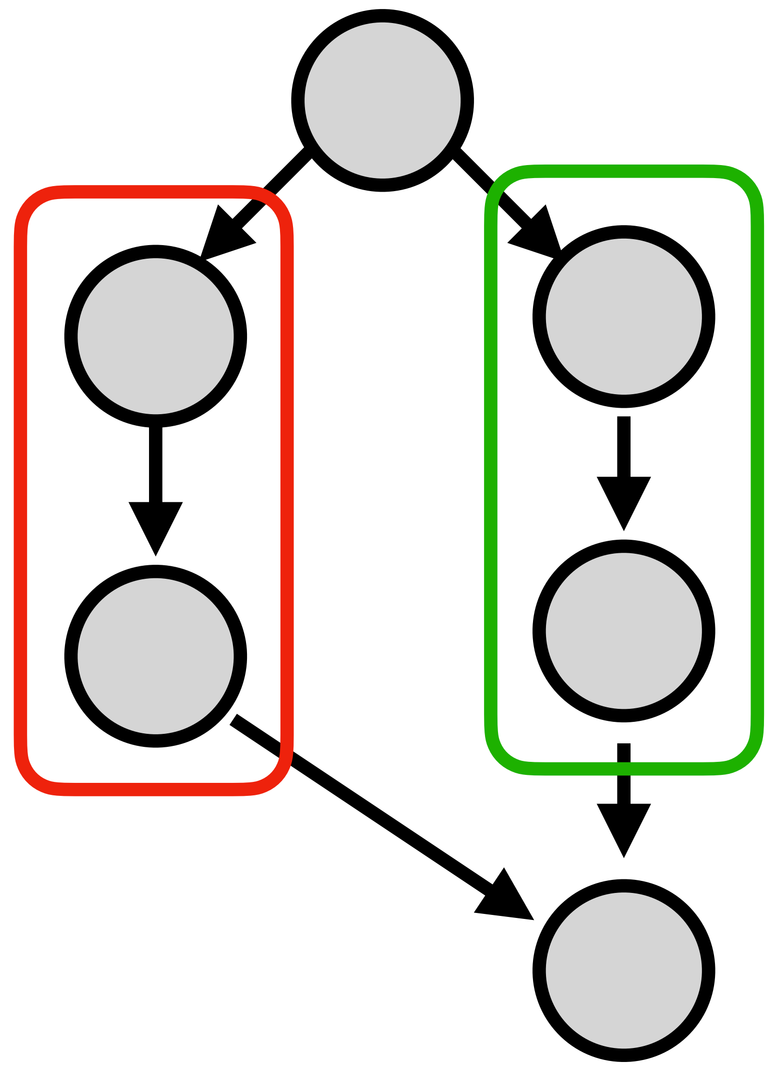
Policy Network



- Uses REINFORCE policy gradient algorithm to minimize running time (our reward signal)
    - Running time = one forward pass + one backward pass + one parameter update

Co-location heuristics

- Problem: Tensorflow graphs can have tens of thousands of nodes
    - Would take too long to run all of them through LSTM
- Solution: group operators via heuristics
    - If operation A is only used by operation B, they are co-located
    - All operations in an LSTM “step” are co-located
- This shrinks problem space: no longer finding placement for ALL nodes; we only have to solve placement for each group
- <u>Required</u> to make training time reasonable



6 -> 4 placements

Model	#operations	#groups
RNNLM	8943	188
NMT	22097	280
Inception-V3	31180	83

Table 1. Model statistics.

Benchmarks

RNNLM
Recurrent Neural Network Language Model

Many LSTM cells in a ‘grid’ structure, where each is only dependent on two of its neighbors. Therefore highly parallelizable

NMT
Neural Machine Translation with attention mechanism

Similar to RNNLM, but more hidden states, so much more computationally expensive

Inception-V3
Image recognition and visual feature extraction

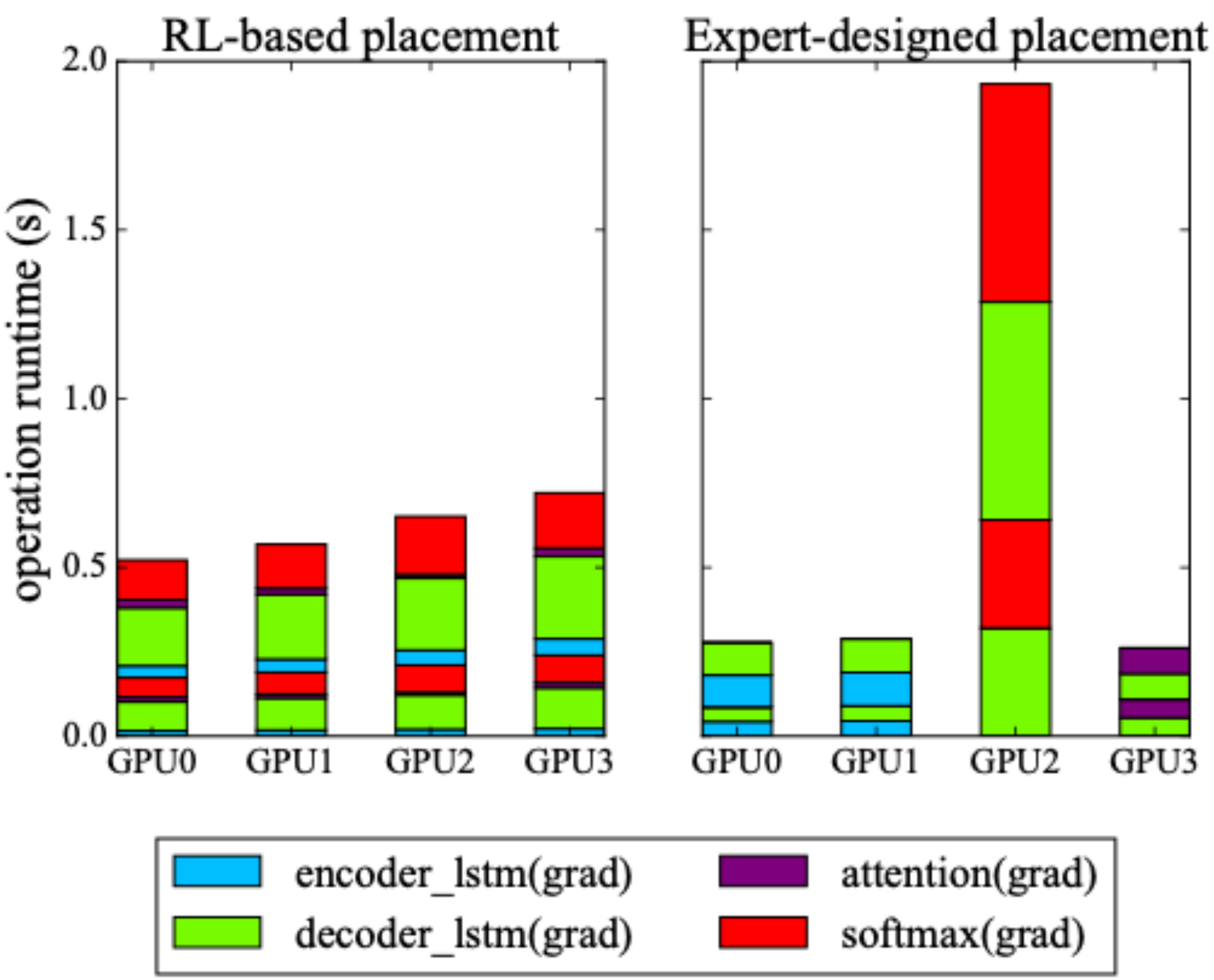
Convolutional network. Lots of parallelization within each “block” of conv + pooling etc., but blocks must be executed sequentially

Results

Tasks	Single-CPU	Single-GPU	#GPUs	Scotch	MinCut	Expert	RL-based	Speedup
RNNLM (batch 64)	6.89	**1.57**	2	13.43	11.94	3.81	**1.57**	0.0%
			4	11.52	10.44	4.46	**1.57**	0.0%
NMT (batch 64)	10.72	OOM	2	14.19	11.54	4.99	**4.04**	23.5%
			4	11.23	11.78	4.73	**3.92**	20.6%
Inception-V3 (batch 32)	26.21	**4.60**	2	25.24	22.88	11.22	**4.60**	0.0%
			4	23.41	24.52	10.65	**3.85**	19.0%

Results (cont.)

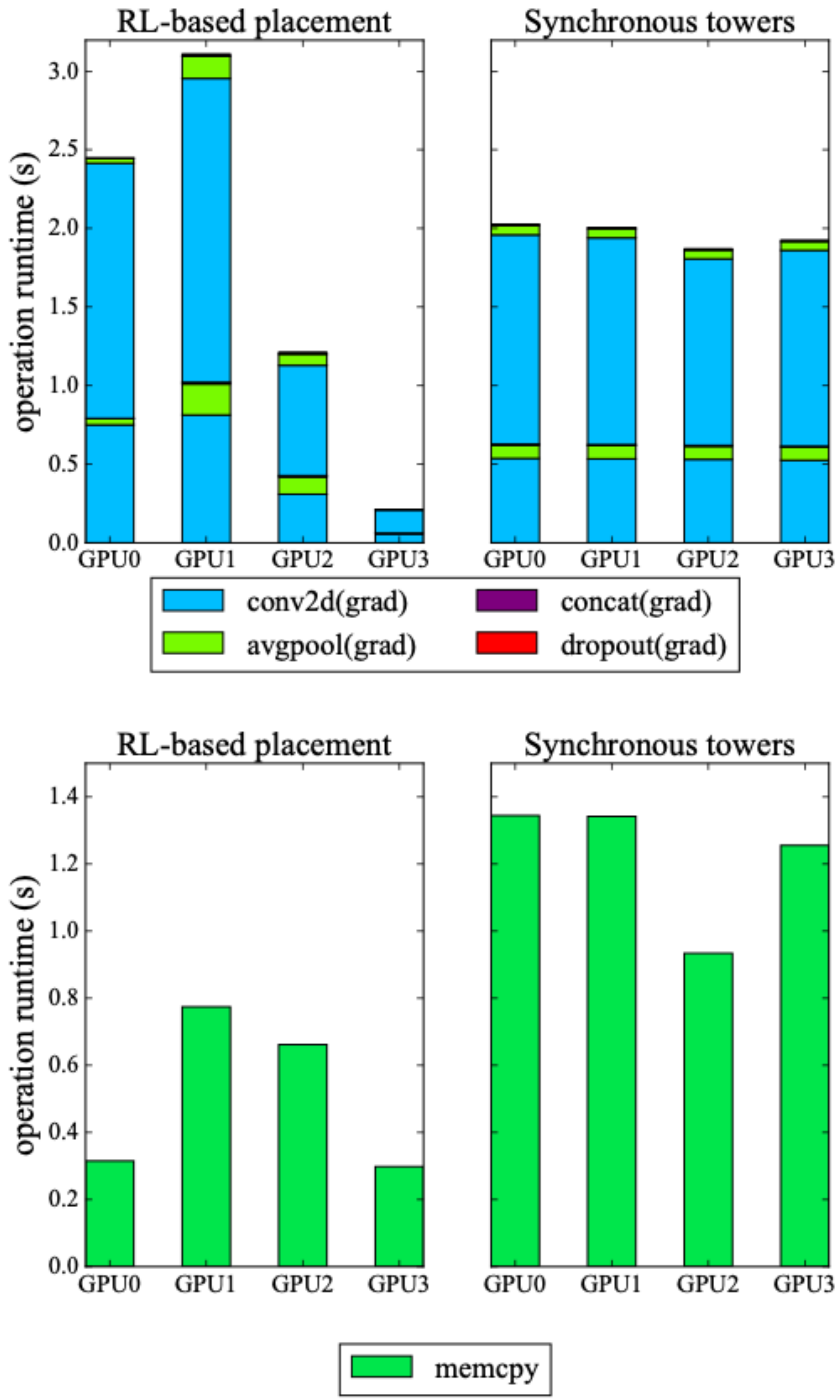
- RL agent achieves better balance…



NMT model

Results (cont.)

- But only when it makes sense to!



Inception-V3

Problems…

- Network must be re-trained for each computation graph; therefore training time is an important metric. 12 to 27 hours on their benchmarks!
- Co-location heuristics are a “necessary evil” to improve training time. Some graphs would be uncomputable otherwise.
- **Downsides**:
    - Some good placements are made impossible (i.e. an LSTM step cannot be parallelized using their heuristics)
    - The user must configure which heuristics should be used on their computation graph.
        - Back to using human experts!

Follow-up:

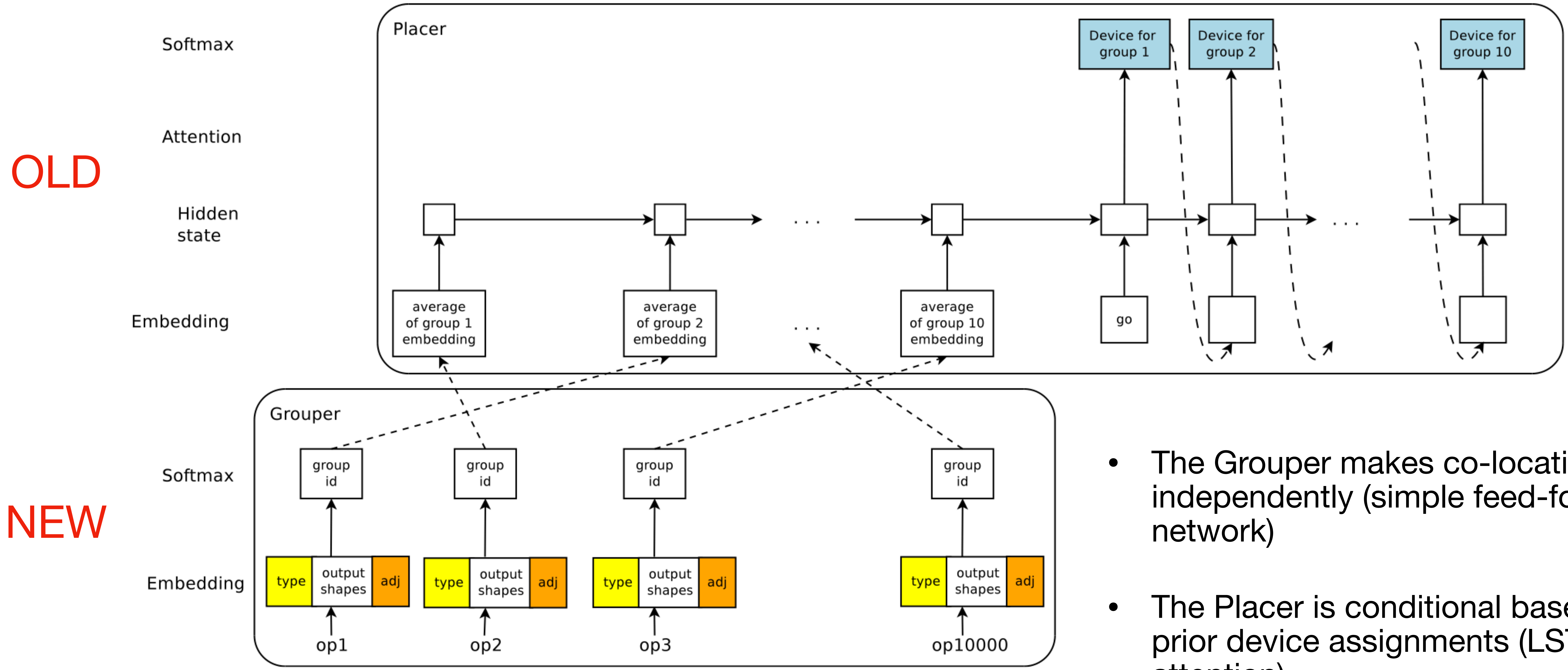
A Hierarchical Model for Device Placement

Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le, Jeff Dean

How it works

- Replace the co-location heuristics with a network which learns to assign operations to groups. (The “Grouper”)
- Use the previous LSTM approach as before to find placements for each group. (The “Placer”)
- **Why?**
    - No more human involvement (co-location is automatically learned)
    - Can handle large graphs (by grouping down until its feasible to solve)
    - Can find placements that co-location heuristics would omit

Architecture



- The Grouper makes co-location decisions independently (simple feed-forward network)
- The Placer is conditional based upon prior device assignments (LSTM + attention)

Results

Tasks	CPU Only	GPU Only	#GPUs	Human Expert	Scotch	MinCut	Hierarchical Planner	Runtime Reduction	
Inception-V3	0.61	0.15	2	0.15	0.93	0.82	**0.13**	16.3%	
ResNet	-	1.18	2	1.18	6.27	2.92	**1.18**	0%	
RNNLM	6.89	1.57	2	1.57	5.62	5.21	**1.57**	0%	
NMT (2-layer)	6.46	OOM	2	2.13	3.21	5.34	**0.84**	60.6%	v. 19.0%
NMT (4-layer)	10.68	OOM	4	3.64	11.18	11.63	**1.69**	53.7%	v. OOM
NMT (8-layer)	11.52	OOM	8	**3.88**	17.85	19.01	4.07	-4.9%	v. OOM

In Conclusion

- Training time limitation still present. Hierarchical approach is 3hrs instead of 27hrs, but still not insignificant.
- These are the first two papers to use RL for device placement
- A small set of others works have tried this since, with varying success:
    - REGAL: use RL to tune a genetic algorithm to solve placement
    - Placeto: use a GNN to learn representations, then use RL for placement (no RNNs)
        - Big bonus: generalizable to other graphs! (No more retraining)
            - Still not fully successful…

Questions?