Placeto: Learning Generalizable Device Placement Algorithms for Distributed Machine Learning

By Ravichandra Addanki, Shaileshh Bojja Venkatakrishnan, Shreyan Gupta, Hongzi Mao, Mohammad Alizadeh

Asbjorn Lorenzen

November 20, 2024

Introduction to Placeto

- **Goal:** Automate device placement for distributed neural network training.
- **Problem:** Previous methods lack generalizability and require retraining for each new computation graph.
- **Placeto Solution:** Develop a generalizable placement policy to predict placements for unseen computation graphs without retraining.

Key Innovations of Placeto

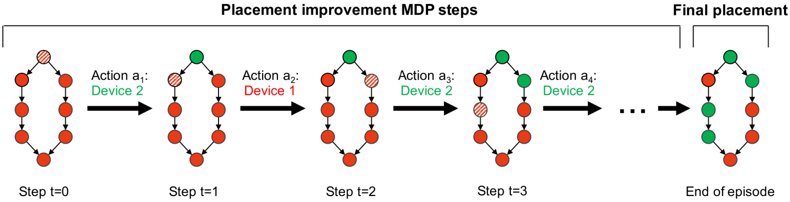
- **Graph Embeddings:** Encodes structure of computation graphs. Improvement from RNNs, which depend on node labels and sequence
- **Iterative Placement Policy:** Sequentially improves placement per node, unlike one-shot placements.

Why Generalizability?

- **Challenges in Model Development:** Frequent retraining is too slow.
- ...especially when using temporary environments, or when iteratively developing a model.
- **Goal of Placeto:** Create a policy, not just a placement. Transferable to different computation graphs within the same family.

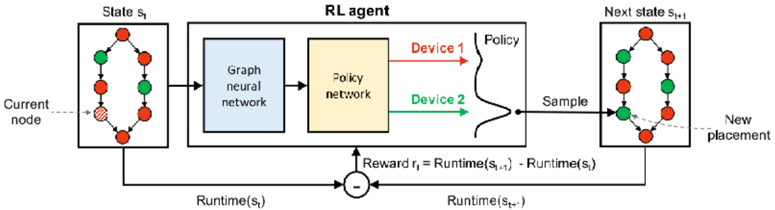
Placeto's Placement Improvement Steps

1. **Iterative Node Placement:** Processes the computation graph node by node.
2. **Placement Improvement Policy:** Predicts optimal device for each node iteratively.



Reinforcement Learning (RL) in Placeto

- **Mapping Goal:** Vertex (ops) $\rightarrow$ Device.
- **Training via RL:** Optimizes placements iteratively across similar computation graphs.



Graph Embedding Techniques

- **Forming the graph:** Group adjacent ops together to create vertices. Data passing between op groups becomes edges.
- **Op group attributes:** `total_runtime`, `output_tensor_size`, `current_placement`, `is_node_current`, `is_node_done`). Collected from on-device measurements.
- **Local neighborhood summarization:** Aggregate neighborhood data for each node. Let f, g be MLPs and $\mathbf{x}_v$ be data from vertex v. Perform message passing:

$$\mathbf{x}_v \leftarrow g\left(\sum_{u \in N(v)} f(\mathbf{x}_u)\right)$$

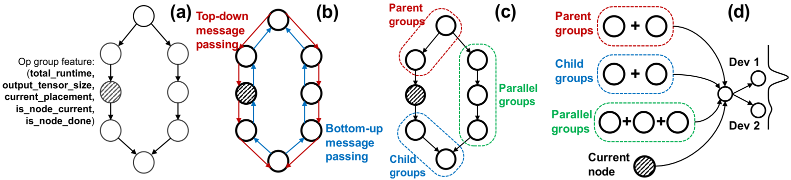
  Perform message passing for two groups: in-neighbors and out-neighbors (incoming and outgoing data flow). Repeat k times to propagate data through graph.

Graph Embedding Techniques

- **Pooling summaries:** Aggregate embeddings at each node to create a summary of entire graph. Use similar function as neighborhood summaries, but with 3 sets: $S_{parents}(v), S_{children}(v), S_{parallel}(v)$.

$$h\left(\sum_{u \in S_i(v)} l(\mathbf{x}_u)\right)$$

Finally, concatenate the three aggregations. This is input to policy network.



Markov Decision Process (MDP) Setup

- **State s:** Computation graph (embedding).
- **Actions:** Update node placements, transitioning to new states.
- **Rewards:** Negative run time at final step or incremental improvements between steps. Punish exceeding memory limit.
- **Policy:** Device placement based on graph embedding.

Training Process Overview

- **RL Algorithm:** Standard policy-gradient with graph sampling.
- **Sampling graphs:** Each episode, sample a graph $G \in \mathcal{G}_T$ from the training graphs, and compute placements on it.
- **Generalization:** Training parameters shared across episodes. Because of the embeddings, they are sharable across graphs, generalizing well to unseen graphs.

Experimental Setup

- **Computation graphs:** Use Tensorflow to generate computation graph given any NN model.
- **Real Models:** Inception-V3, NMT, NASNet.
- **Synthetic Datasets:** Generates similar graphs, e.g. by varying hyperparameters of other models or using automatic model design (ENAS).
- **Baseline Comparisons:** Single GPU, Scotch (static mapper), Human Expert, RNN-based approach.
- **Simulating executions:** Use a simulator instead of measuring elapsed time on real hardware. Only use simulator for training purposes.

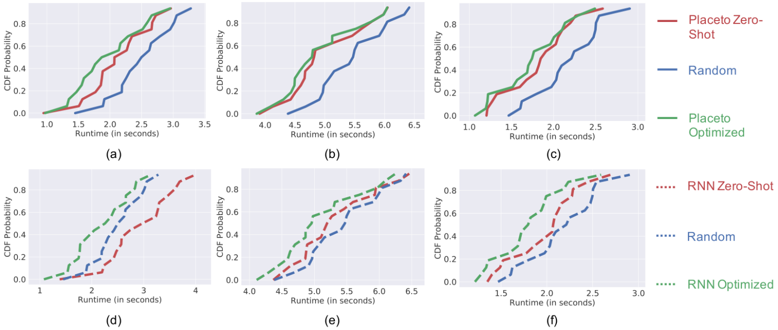
Results and Evaluation

- **Goal:** Optimize performance (runtime of placement) and training time. Evaluate on real models.
- **Performance:** Placeto performs slightly better than RNN based approach, but requires up to 6.1× fewer placements sampled.

	Placement runtime (sec)							Training time (# placements sampled)		Improvement	
Model	CPU only	Single GPU	#GPUs	Expert	Scotch	Placeto	RNN-based	Placeto	RNN-based	Runtime Reduction	Speedup factor
Inception-V3	12.54	1.56	2	1.28	1.54	1.18	1.17	1.6 K	7.8 K	- 0.85%	**4.8** ×
			4	1.15	1.74	1.13	1.19	5.8 K	35.8 K	5%	**6.1** ×
NMT	33.5	OOM	2	OOM	OOM	2.32	2.35	20.4 K	73 K	1.3 %	**3.5** ×
			4	OOM	OOM	2.63	3.15	94 K	51.7 K	**16.5 %**	0.55 ×
NASNet	37.5	1.28	2	0.86	1.28	0.86	0.89	3.5 K	16.3 K	3.4%	**4.7** ×
			4	0.84	1.22	0.74	0.76	29 K	37 K	2.6%	**1.3** ×

Generalizability

- **Synthetic data:** Evaluated on synthetic graph families.
- **Zero-Shot Testing:** Placeto Zero-Shot closely matches optimized performance.
- **RNN Limitations:** RNN Zero-Shot performs poorly due to dependency on node indices.



Strengths

- Novel use of GNNs for device placement
- Finds improved placement significantly faster than earlier approaches

Weaknesses

- Generalization is limited, and its limits are unclear from the paper.
- Generalization is only tested on very similar graphs (synthetically generated for solving the same problem)
- Generalization is trained on various graphs in the same 'family'. Should show generalization from one graph to another similar graph.
- No proper definition of 'graph families'