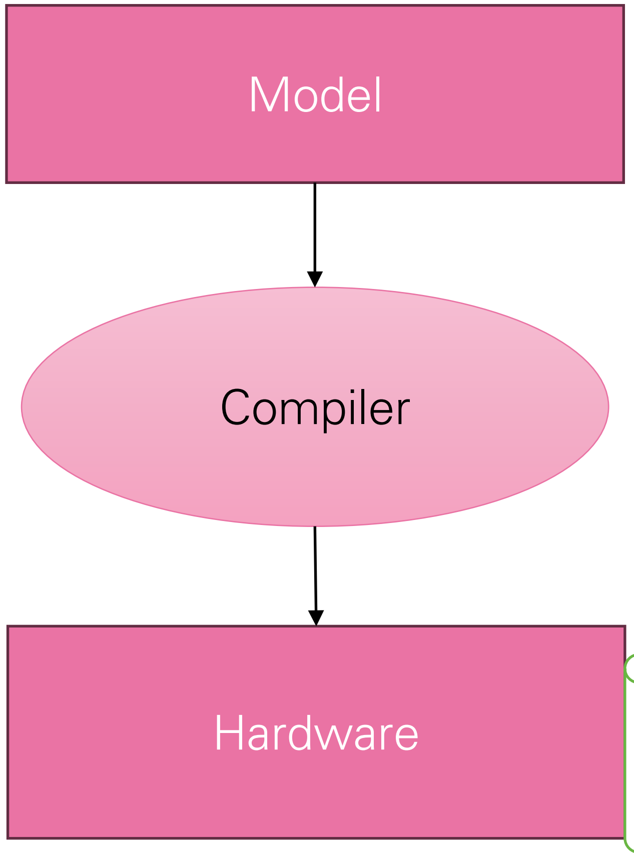
Transferable Graph Optimisers for ML Compilers

Zhou et al.

Google

What’s the problem?

1. Device placement
2. Operation Scheduling
3. Operation Fusion



Lots of data, lots of compute power, big graphs of computation

Cloud computation, Accelarators ie. GPUs, TPUs (Distributed, and different architectures have different strengths)

What was previous work, why was it insufficient?

- Device-specific optimisation by compilers (TensorFlow, XLA, Glow, MLIR) – need to have seen device before
- Heuristic methods on each individual problem ie. Auto-tuning etc.
- Other Reinforcement Learning methods:
  - expensive to train,
  - focused on one problem with no knowledge sharing

Learning solutions need resource efficiency, speed,
AND
To tackle optimisations that affect each other!

A Reinforcement Learning Approach

- GraphSAGE to capture topological information in the compuatational graph
- Scalable attention network to capture long-ranged dependencies
- Feature modulation to allow specialisation on graph type without increasing parameter numbers

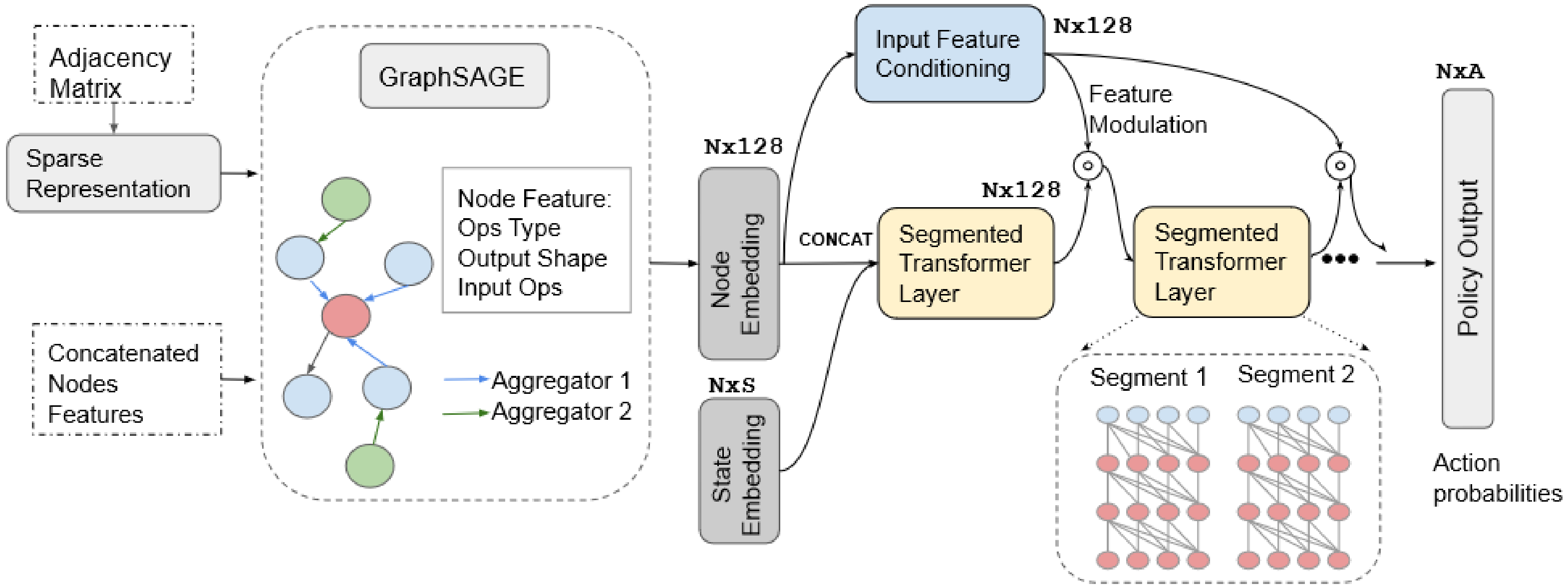
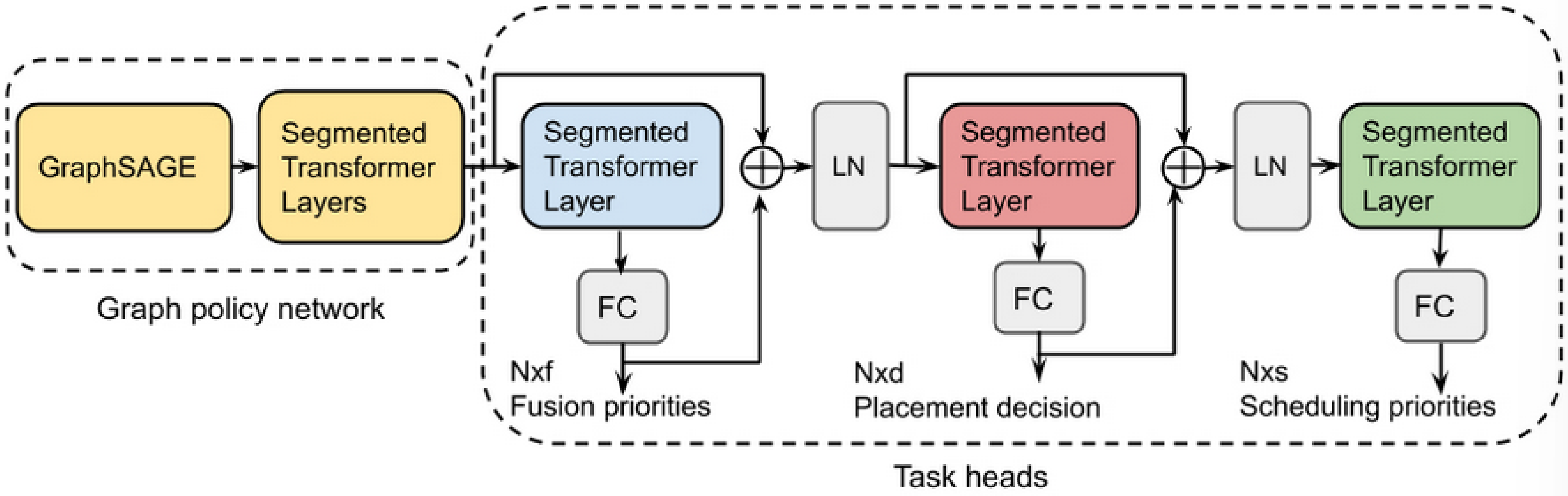


Figure 3: Overview of GO: An end-to-end graph policy network that combines graph embedding and sequential attention. N: Number of Nodes, a: Size of the action space (number of devices, number of priority levels, etc.). Node features are sorted in topological order.

Multiple Dependent Optimisation Tasks

- Recurrent attention layers for each task
- Parameters shared across tasks.



Multi-task policy network that extends GO's policy network with additional recurrent attention layers for each task and residual connections. GE: Graph Embedding, FC: Fully-Connected Layer, Nxf: fusion action dimension, Fxd: placement action dimension, Nxs: scheduling action dimension.

Evaluation

- Methods

Proximal Policy Optimisation

Large negative reward for bad optimisation

6 different architectures

4 baseline comparisons

Up to 80000 nodes (8-layers)

- Findings

Model (#devices)	GO-one (s)	HP (s)	METIS (s)	HDP (s)	Run time speed up over HP / HDP	Search speed up over HDP
2-layer RNNLM (2)	0.173	0.192	0.355	0.191	9.9% / 9.4%	2.95x
4-layer RNNLM (4)	0.210	0.239	0.503	0.251	13.8% / 16.3%	1.76x
8-layer RNNLM (8)	0.320	0.332	OOM	0.764	3.8% / 58.1%	27.8x
2-layer GNMT (2)	0.301	0.384	0.344	0.327	27.6% / 14.3%	30x
4-layer GNMT (4)	0.350	0.469	0.466	0.432	34% / 23.4%	58.8x
8-layer GNMT (8)	0.440	0.562	OOM	0.693	21.7% / 36.5%	7.35x
2-layer Transformer-XL (2)	0.223	0.268	0.37	0.262	20.1% / 17.4%	40x
4-layer Transformer-XL (4)	0.230	0.27	OOM	0.259	17.4% / 12.6%	26.7x
8-layer Transformer-XL (8)	0.350	0.46	OOM	0.425	23.9% / 16.7%	16.7x
Inception (2) b32	0.229	0.312	OOM	0.301	26.6% / 23.9%	13.5x
Inception (2) b64	0.423	0.731	OOM	0.498	42.1% / 29.3%	21.0x
AmoebaNet (4)	0.394	0.44	0.426	0.418	26.1% / 6.1%	58.8x
2-stack 18-layer WaveNet (2)	0.317	0.376	OOM	0.354	18.6% / 11.7%	6.67x
4-stack 36-layer WaveNet (4)	0.659	0.988	OOM	0.721	50% / 9.4%	20x
GEOMEAN	-	-	-	-	**20.5% / 18.2%**	**15x**

Table 2: Run time comparison between GO-one, human expert, TensorFlow METIS, and hierarchical device placement (HDP) on six graphs (RNNLM, GNMT, Transformer-XL, Inception, AmoebaNet, and WaveNet). Search speed up is the policy network training time speed up compared to HDP (reported values are averages of six runs).

Speedup	TF default	SA	GO-one	Speedup	TF default	SA	GO-one
NMT (2GPU)	2.82	3	**3.19 (+0.37)**	RNNLM (8GPU)	-2.39	-2.38	**-2.27 (+0.11)**
NMT (4GPU)	-0.89	5.34	**12.03 (+12.92)**	TRF-XL (2GPU)	24.27	25.1	**28.51 (+4.24)**
NMT (8GPU)	10.47	10.47	**12.65 (+2.18)**	TRF-XL (4GPU)	17.05	19.32	**19.99 (+2.94)**
RNNLM (4GPU)	1.04	1.06	**1.23 (+0.19)**	TRF-XL (8GPU)	21.66	26.25	**31.48 (+9.82)**

Table 3: Speedup of each fusion policy normalized to the no-fusion case (reported in %). The number in the parentheses is the improvement of our work over the default fusion.

Strengths

- Generalises across different graphs and tasks – move varied set
- Work on entire graph at once instead of just one node at a time – capture long distance dependencies
- Speed-ups
- Scalable – works on >10000 nodes
- Adaptable to different architectures
- 21% improvement over human experts and 18% improvement over the prior state of the art with 15x faster convergence than simulated annealing

Critique

- Their figures make no sense
- Reproducibility
- Loss of explainability
- Mainly putting together existing components (GraphSage, Transformer, etc.) and so limited novelty from an ML perspective
- Poor explanation of why each technique is useful and why decisions were made

Who used it? Where might it be used?

- Author Suggestions:
  - Benchmark evaluation on new hardware
  - Less effort for maintenance when new hardware is released
  - Decrease carbon footprint of machine learning
- Wider usage:
  - Quite new so not much usage yet
  - People are doing similar work:
  - Could use it for finding new strategies for compiler optimisation in general, or to look at relationships between coupled optimisation problems

Discussion

- Would you use a compiler that you can’t explain?
- Is ML compilation in this way substantially different enough from things like autotuning?

References

- Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter Ma, Qiumin Xu, Hanxiao Liu, Mangpo Phitchaya Phothilimtha, Shen Wang, Anna Goldie, Azalia Mirhoseini, and James Laudon. 2020. Transferable graph optimizers for ML compilers. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS'20). Curran Associates Inc., Red Hook, NY, USA, Article 1161, 13844–13855.
- Kaufman, Samuel J. et al. “A Learned Performance Model for Tensor Processing Units.” Conference on Machine Learning and Systems (2021).
- Xie, Xinfeng et al. “A Transferable Approach for Partitioning Machine Learning Models on Multi-Chip-Modules.” ArXiv abs/2112.04041 (2021): n. pag.
- Addanki, Ravichandra, et al. "Placeto: Learning generalizable device placement algorithms for distributed machine learning." arXiv preprint arXiv:1906.08879 (2019).
- Valentin Dalibard, Michael Schaarschmidt, and Eiko Yoneki. 2017. BOAT: Building Auto-Tuners with Structured Bayesian Optimization. In Proceedings of the 26th International Conference on World Wide Web (WWW '17). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 479–488. https://doi.org/10.1145/3038912.3052662
- William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 1025–1035.