X-RLflow: Graph Reinforcement Learning for Neural Network Subgraphs Transformation

Guoliang He, Sean Parker and Eiko Yoneki, MLSys, 2023

Presented by Yavuz Ferhatosmanoglu

Motivation

- Existing tensor graph superoptimisation systems (e.g., TASO) use greedy or backtracking search approaches
  - Optimisation can miss globally optimal graphs
- Cost model estimates can have large discrepancy with actual costs

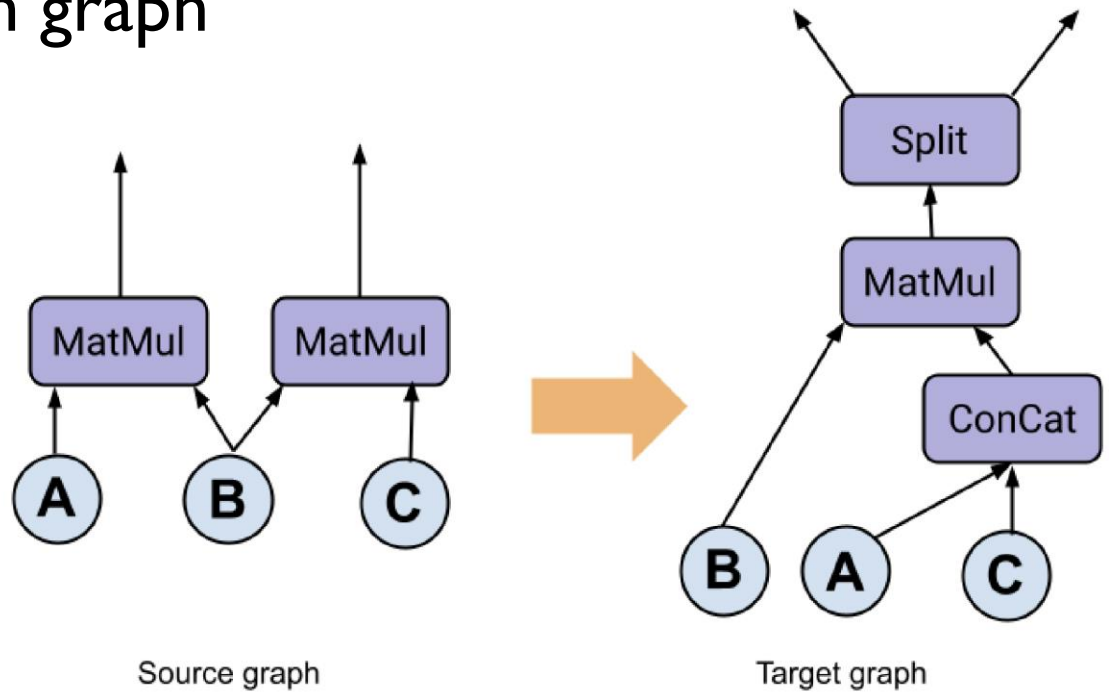
DNNs	Cost model	E2E	Diff (%)
DALL-E	1.8269	1.7324	5.2%
InceptionV3	8.3650	9.2098	10.1%
BERT	1.0453	1.1264	7.8%
Squeezenet	1.3082	1.4006	7.1%
ResNext-50	6.1545	7.6498	24%
T-T	2.4828	2.7281	9.9%

X-RLflow: RL Contribution

- RL can tolerate short-term performance decrease to maximise long-term rewards
  - Sequential graph transformations can lead to globally optimal tensor graph
- RL works well with **sparse** or **delayed** reward scenarios
  - End-to-end inference is performed every N iterations to inform the optimisation
- RL offers generalisation by reusing trained agents
  - RL agent can be applied to graphs with different tensor shapes

X-RLflow: Graph Rewrite Rules

- Before optimisation phase: graph rewrite rules are generated by TASO's generator
- During optimisation: candidates are generated by **pattern matching** rules onto the computation graph

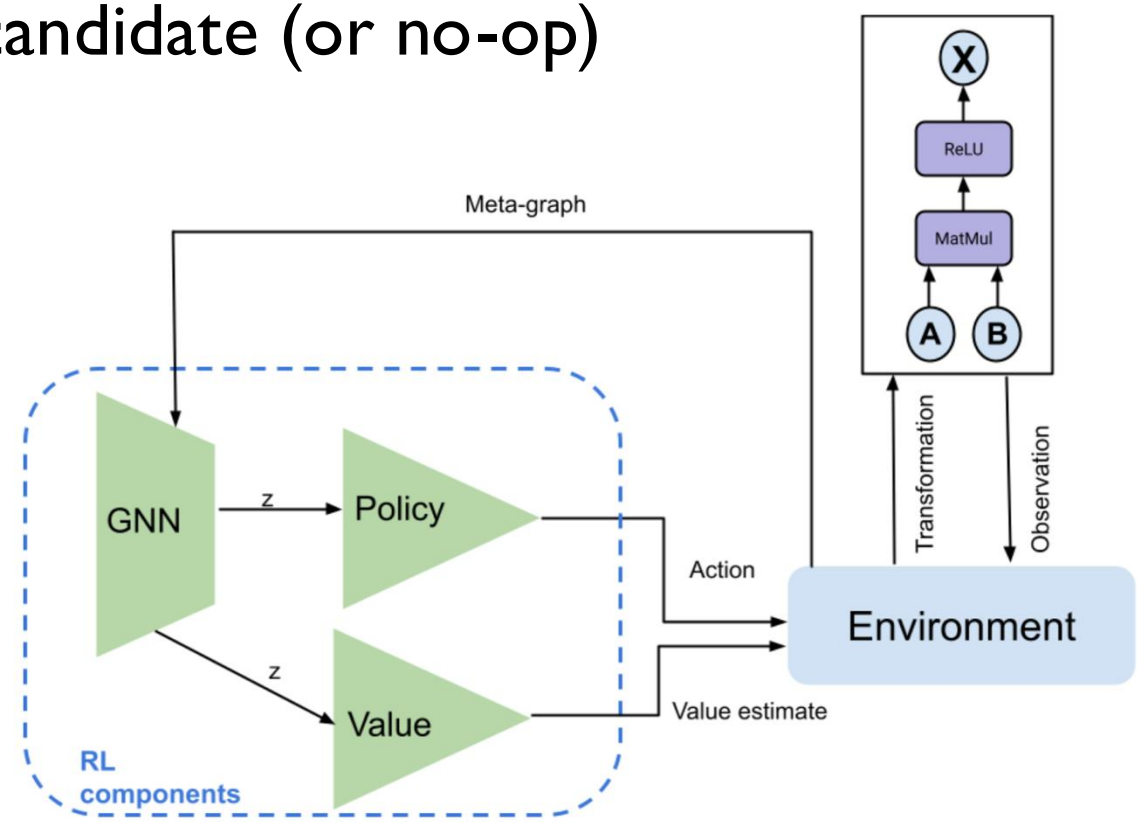


Source graph

Target graph

X-RLflow: RL Architecture

1. Environment generates candidate graphs
2. Current graph and candidates are encoded into a state vector via a GNN
3. Action: agent selects a candidate (or no-op)



X-RLflow: RL Formulation

- Default reward function:

$$r_t = \frac{RT_{t-1} - RT_t}{RT_0} * 100$$

- Learning algorithm: Proximal Policy Optimisation (PPO) with clipping

$$J = \mathcal{L}_{clip} + c_1\mathcal{L}_{vf} + c_2\mathcal{L}_{entropy}$$

  - End-to-end training is performed by the use of a single objective function
  - PPO enables distributed training through mini-batch updates

X-RLflow: The GNN

- Update nodes through edge and node features

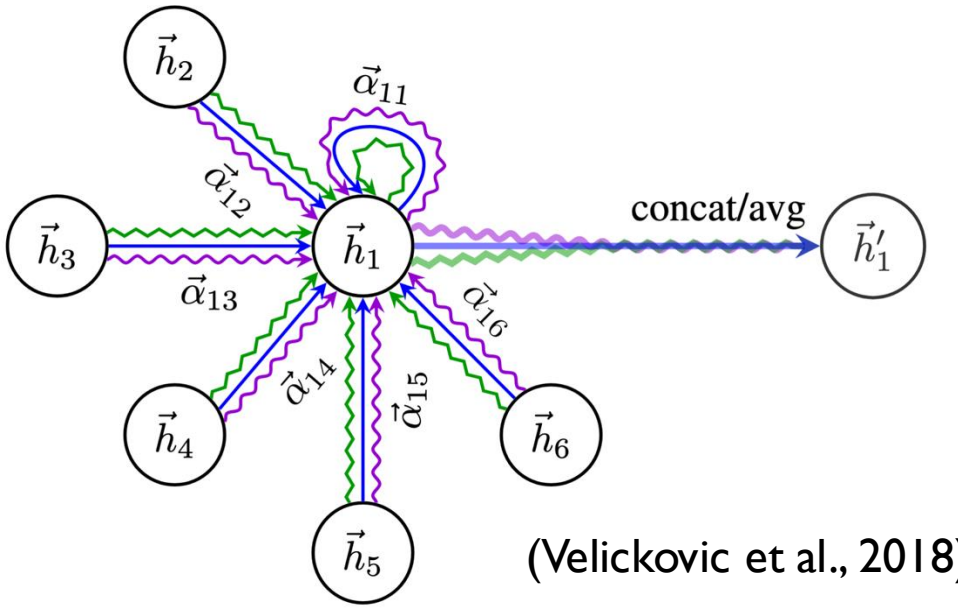
$$\vec{h'}_i = \sigma\{W(\sum_{j \in \mathcal{E}_i} \vec{e}_j \| \vec{h}_i)\}$$

- Several graph attention layers to update node representations

$$\vec{h'}_i = \sigma(\sum_{j \in \mathcal{N}_i} \alpha_{i,j} W \vec{h}_j)$$

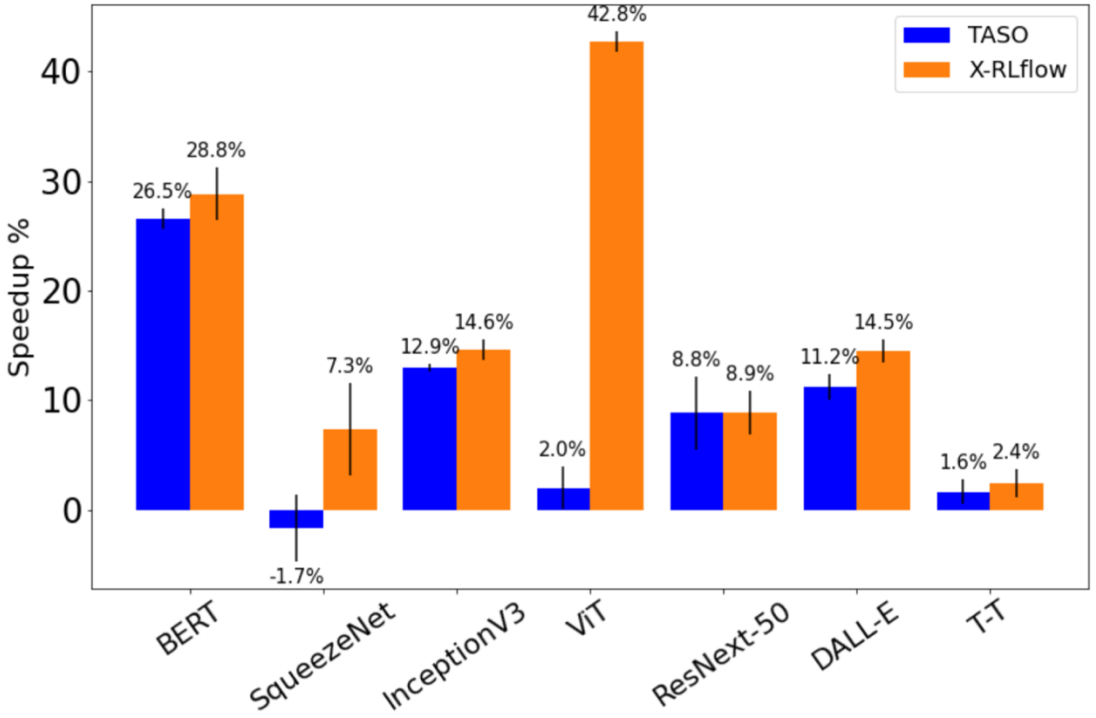
- Global update layer

$$\vec{g'} = \sigma(\sum_{\mathcal{N}} \vec{h} \| \vec{g})$$

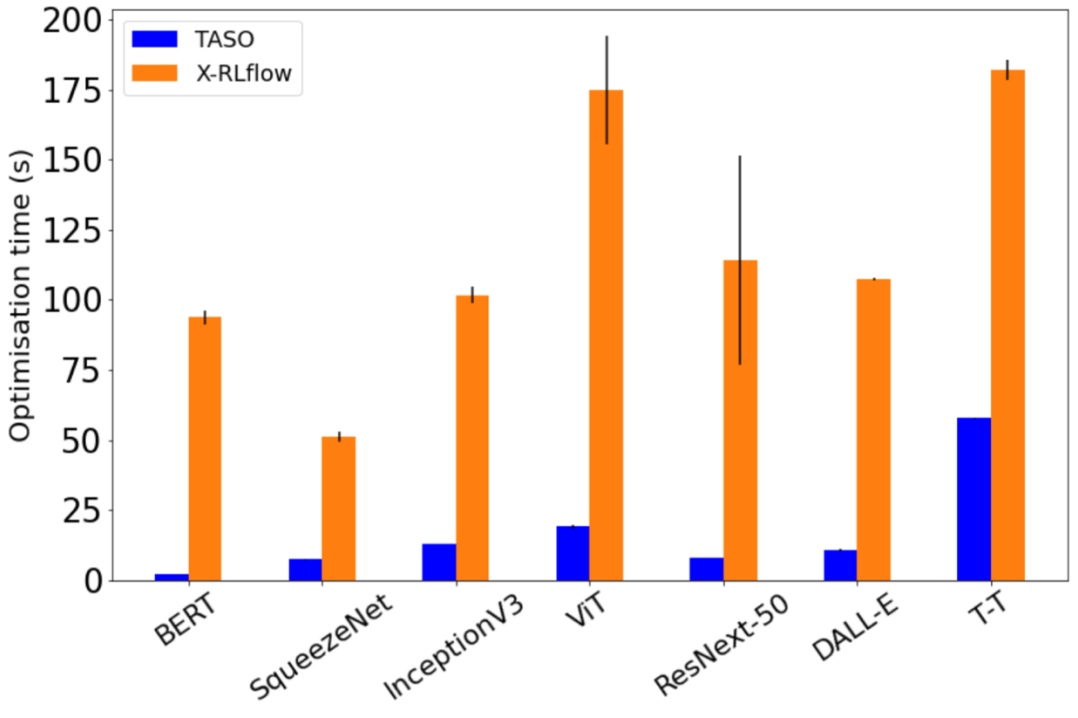


(Velickovic et al., 2018)

Evaluation: End-to-End Benchmarks



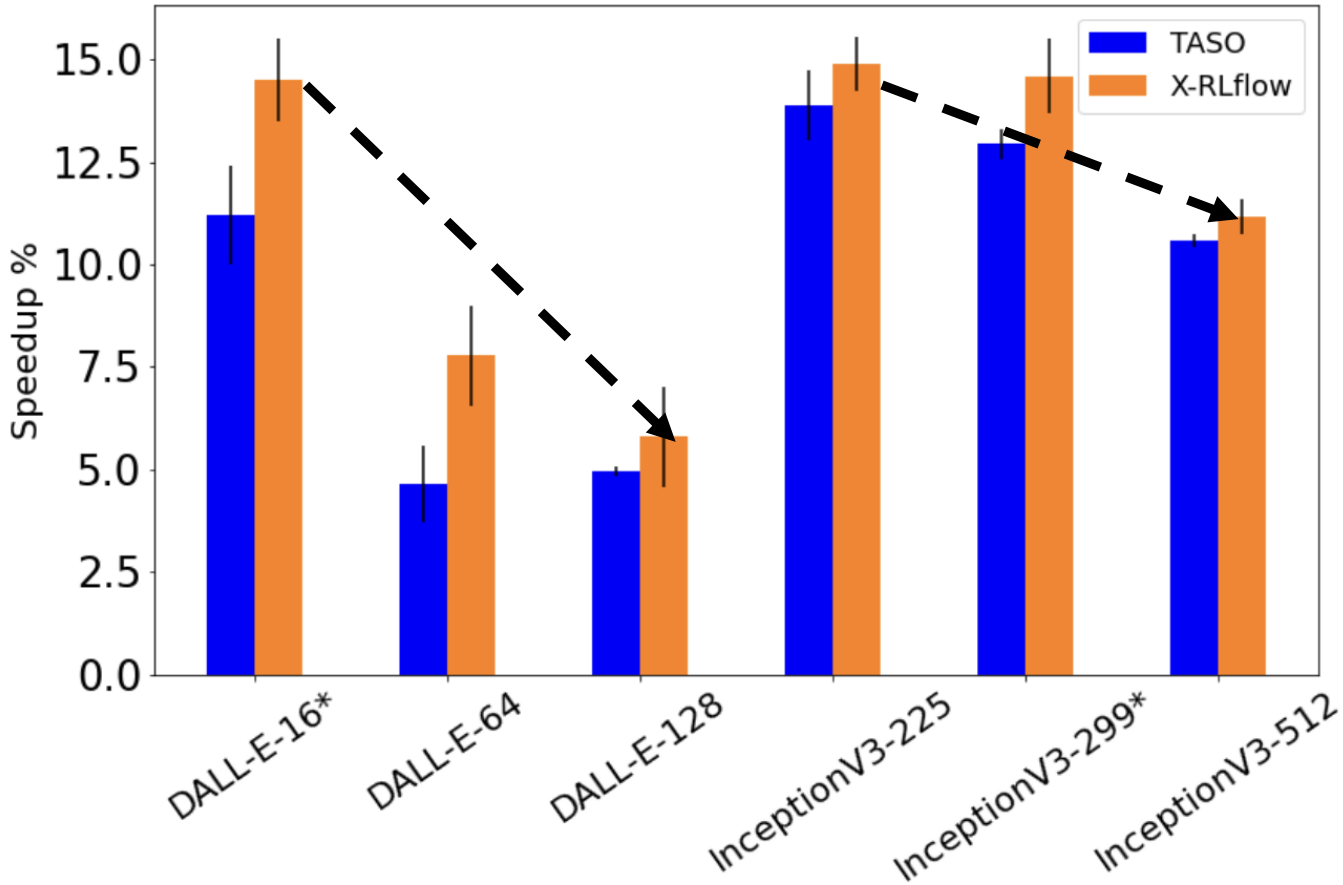
End-to-end inference speedup



Optimisation time

Evaluation: Generalisation

- Train X-RLflow in a static environment once, run on different input tensor shapes



Final Thoughts: Strengths

- Novel approach applying RL and GNNs to graph superoptimisation
- Use of real inference latency as reward instead of proxy signals
- Strong evaluation in terms of the range of end-to-end models
- Substantial improvements over TASO and Tensat
  - Particularly for transformer architectures
- Ability to zero-shot generalise to varying tensor shapes

Final Thoughts: Criticisms and Discussion

- High optimisation time, with training time not quantified
  - Unsuitable for just-in-time compiled scenarios
  - Training data could be generated in parallel, or RL agent could be run on GPU
- Generalisation to different tensor graphs is not supported
  - Model-based RL with multi-graph training could enable this
- State space can become infeasibly large
  - Incorporating **equality saturation** can allow RL to treat semantically identical graphs as single entities

Thank you!
Any Questions?

X-RLflow: Learning Algorithm

- Clip objective:

$$\mathcal{L}_{clip} = -\mathbb{E}_G\{\min(\frac{\pi_\theta}{\pi_{\theta_k}} \cdot A^{\pi_{\theta_k}}, clip(\frac{\pi_\theta}{\pi_{\theta_k}}, 1-\epsilon, 1+\epsilon) A^{\pi_{\theta_k}})\}$$

- Value loss:

$$\mathcal{L}_{vf} = \mathbb{E}_G\{(V_\theta(s_t) - V_{target})^2\}$$

- Final objective:

$$J = \mathcal{L}_{clip} + c_1\mathcal{L}_{vf} + c_2\mathcal{L}_{entropy}$$

Evaluation: Comparison with Tensat

- Tensat: builds equality graph and employs equality saturation
- X-RLflow performs better on more complex graphs