TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions

Zhihao Jia et. al.

Presented by: Andreas Pletschko (ap2535)

26.11.2025

R244 – Large-Scale Data Processing and Optimization

Neural Networks as Computation Graphs

Modern frameworks transform a neural network implementation into a computation graph:

```
_temp_1 = W @ x
_temp_2 = _temp_1 + b
_temp_3 = relu(_temp_2)
...
C = _temp_n
```

Figure 1: Sample neural network code.

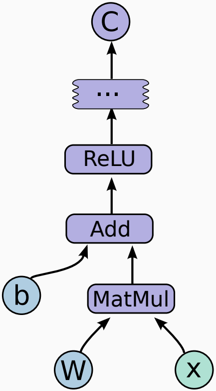
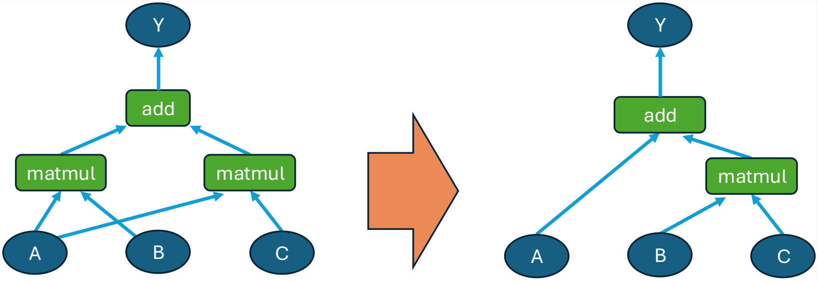


Figure 2: Corresponding computation graph, taken from [2].

Computation Graph Optimization

Graph structure can be optimized, without changing the semantics:

$$(A \times B) + (A \times C) = A \times (B + C)$$



Shortcomings of Existing Approaches

Existing frameworks, such as TensorFlow [2], TVM [3], MetaFlow [5] or TensorRT [1] have one or more of the following limitations:

1. Graph substitutions have to be **manually designed**
2. Graph substitutions are **not formally verified**
3. Graph substitutions are **applied greedily**, decreasing the possible graph search space
4. Graph and data layout are **not jointly optimized**

TASO's Contributions

Finding Graph Substitutions Automatically

TASO tries to automatically find a set of potential substitutions $\mathcal{S}$ by brute-force:

1. Enumerate all possible graphs of size $< n_{\text{threshold}}$, and for each graph:
   1.1 Calculate the `FingerPrint` (hashing computation output on a small set of inputs)
2. For all graph pairs with the same `FingerPrint`
   2.1 Compute graph outputs on a large set of inputs
   2.2 If equivalent on the larger test set: Add to substitution set $\mathcal{S}$

Example of an Automatically Generated Substitution

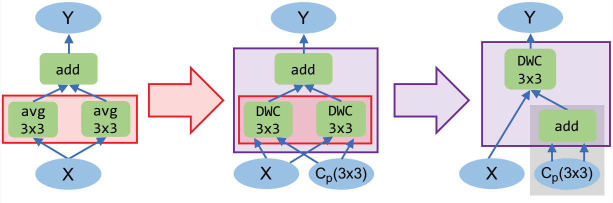


Figure 3: Example of a substitution generated by TASO, taken from [4].

Verification of Substitutions – 1/2

Just testing is not enough to guarantee equivalence

- Instead: Use manually defined **operator properties** to formally model the operators and prove the substitutions' correctness
- Example operator property of linearity of matrix multiplication:

$$\forall x, y, z . \mathtt{matmul}(x, \mathtt{ewadd}(y, z)) = \mathtt{ewadd}(\mathtt{matmul}(x, y), \mathtt{matmul}(x, z))$$

Verification of Substitutions – 1/2

Just testing is not enough to guarantee equivalence

- Instead: Use manually defined **operator properties** to formally model the operators and prove the substitutions' correctness
- Example operator property of linearity of matrix multiplication:

$$\forall x, y, z. \mathtt{matmul}(x, \mathtt{ewadd}(y, z)) = \mathtt{ewadd}(\mathtt{matmul}(x, y), \mathtt{matmul}(x, z))$$

Given the set of operator properties $\mathcal{P}$ and graph substitutions $\mathcal{S}$, use a first-order theorem prover to check for entailment

$$\mathcal{P} \models \mathcal{S}$$

Operator Property Verification

Substitution verification requires **correctness of operator properties** $\mathcal{P}$:

1. **Tensor-level:** Verify properties for all combinations of operator parameters and tensors of shape $< (4 \times 4 \times 4 \times 4)$
2. **Logic-level:** Check $\mathcal{P}$ for inconsistency and redundancy

Resulting graph substitutions might still contain lots of redundancies:

1. **Renamed input tensors:** If a substitution can be obtained from another one, simply by **renaming** one (or more) input tensors, remove all but the most general one.

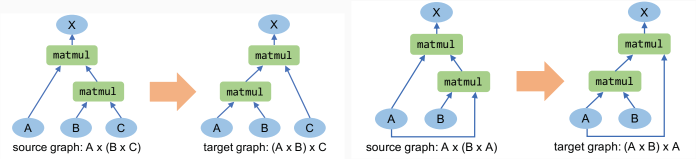


Figure 4: Two graph substitutions that are equivalent up to tensor names, taken from [4].

Graph Substitution Pruning

Resulting graph substitutions might still contain lots of redundancies:

2. **Common subgraphs:** If a substitution contains **common subgraphs on both sides**, try to remove the common subgraph and verify the result.

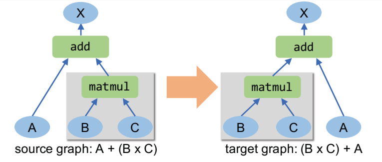


Figure 4: Graph substitution with a common subgraphs on both sides, taken from [4].

Graph and Layout Optimization

With the generated, verified and pruned substitution set $\mathcal{S}$, try to optimize the computation graph $\mathcal{G}_{in}$:

1. Initialize priority queue with $\mathcal{G}_{in}$
2. Take the currently best graph $\mathcal{G}$ from the priority queue
3. For **every substitution** $s \in \mathcal{S}$ and **possible data layout** l:
   - 3.1 Apply l and s on $\mathcal{G}$ to obtain $\mathcal{G}'$
   - 3.2 Check that $\mathcal{G}'$ contains no cycles
   - 3.3 Estimate graph runtime by summing over individual operator runtimes
   - 3.4 If graph runtime is at most α worse than the current optimum: Add $\mathcal{G}'$ to priority queue
4. Repeat from 2. as long as priority queue is not empty
5. Otherwise, return the graph with the lowest runtime

Evaluation

Evaluation Setup

TASO was evaluated on 5 different neural network architectures, against various frameworks:

- **Architectures:** ResNet, ResNeXt-50, NasNet-A, NasRNN, BERT
- **Baseline frameworks:** TensorFlow (XLA), TensorRT, MetaFlow, TVM

Inference Time Comparisons

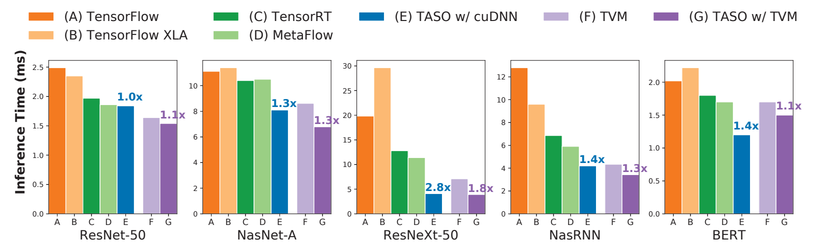


Figure 5: Inference time comparisons of different DNN frameworks and architectures, taken from [4].

Impact of Graph Substitution Size

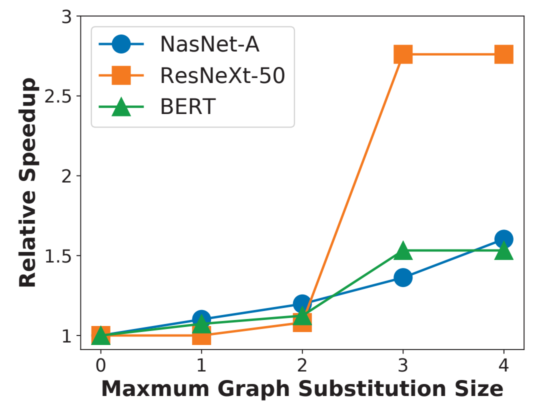


Figure 6: Relative speedup on different neural network architectures, with varying substitution generation thresholds.

Impact of Pruning Substitutions

Pruning Techniques	**Remaining Substitutions**	**Reduction v.s. Initial**
Initial	28744	1×
Input tensor renaming	17346	1.7×
Common subgraph	743	39×

Figure 7: Graph substitution set reductions after different pruning stages, taken from [4].

Impact of Jointly Optimizing Graph and Data Layout

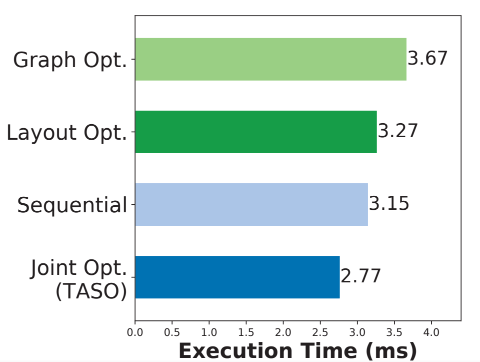


Figure 8: Inference time comparison of BERT using different optimization strategies, taken from [4].

Conclusion

Opinions

Strengths:

- **First-in-Class:** First automated generator for graph substitutions.
- **Versatility:** Functions as a framework-agnostic optimization backend.
- **Thorough evaluation:** Tested rigorously, including ablation.
- **Impact:** Tackles multiple critical bottlenecks inherent to manual heuristic definition.

Opinions

Strengths:

- **First-in-Class:** First automated generator for graph substitutions.
- **Versatility:** Functions as a framework-agnostic optimization backend.
- **Thorough evaluation:** Tested rigorously, including ablation.
- **Impact:** Tackles multiple critical bottlenecks inherent to manual heuristic definition.

Limitations:

- **Expert Knowledge Required:** Requires manual abstraction of operators into first-order logic.
- **Combinatorial Explosion:** Brute-force enumeration fails for subgraph sizes > 4.
- **Operator Scalability:** Only tested on a small operator set ($n = 12$); larger scaling is unclear.

TLDR

Key contributions:

- **Auto-generation:** Finds substitutions via brute-force backtracking.
- **Formal Verification:** Uses a theorem prover to ensure correctness.
- **Auto-pruning:** Automatically filters the substitution search space.
- **Joint Optimization:** Tunes graph structure and data layout together.
- **Proven Results:** Outperforms established frameworks.

Questions?

NVIDIA TensorRT.
https://developer.nvidia.com/tensorrt, 2017.

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng.
TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, Mar. 2016.

References ii

- T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, et al.
  {TVM}: An automated {End-to-End} optimizing compiler for deep learning.
  In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, 2018.
- Z. Jia, O. Padon, J. Thomas, T. Warszawski, M. Zaharia, and A. Aiken.
  TASO: Optimizing deep learning computation with automatic generation of graph substitutions.
  In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 47–62. ACM, 2019.
- Z. Jia, J. Thomas, T. Warszawski, M. Gao, M. Zaharia, and A. Aiken.
  Optimizing dnn computation with relaxed graph substitutions.
  Proceedings of Machine Learning and Systems, 1:27–39, 2019.

Total Optimization Overhead

The authors report an overhead of **10 minutes** for the overall procedure (generating and verifying substitutions, optimizing computation graph with the substitutions), with a maximum graph size threshold of 4