Optimizing DNN Computation with Relaxed Graph Substitutions

Zhihao Jia, James Thomas, Todd Warszawski, Mingyu Gao, Matei Zaharia, Alex Aiken

MLSys19

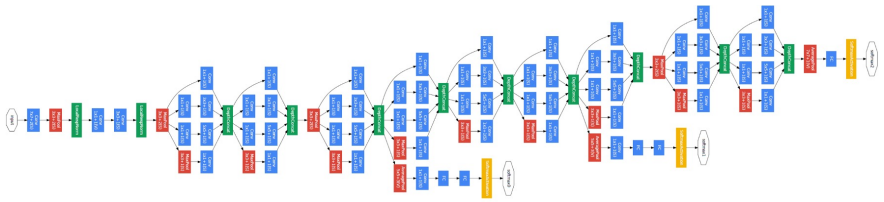
Stanford University

Summary of works

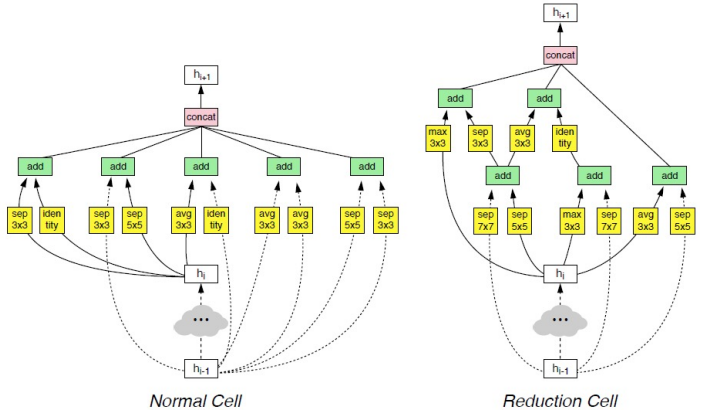
- Optimizing DNN Computation with Relaxed Graph Substitutions
  - It proposed automatic graph-optimization(device specific) which is not applied rule-based(general,greedy) graph optimization(XLA,TensorRT).

- TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions
  - Previous work has a limitation which has to define substitution rule, So this work proposes the automatic generation of graph substitutions.

Complex and Diverse DNN Computation Graphs



Convolution Neural Networks



Neural Architecture Search

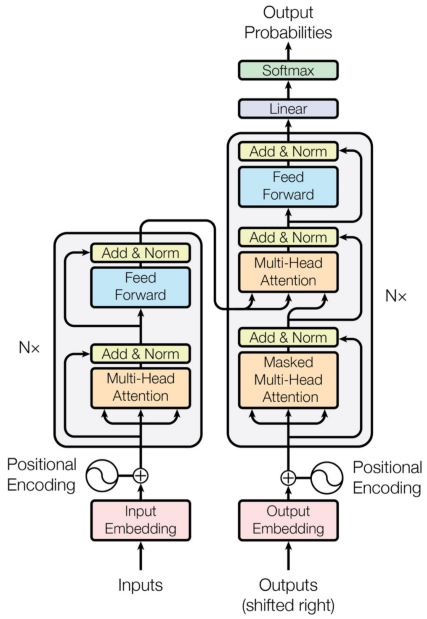
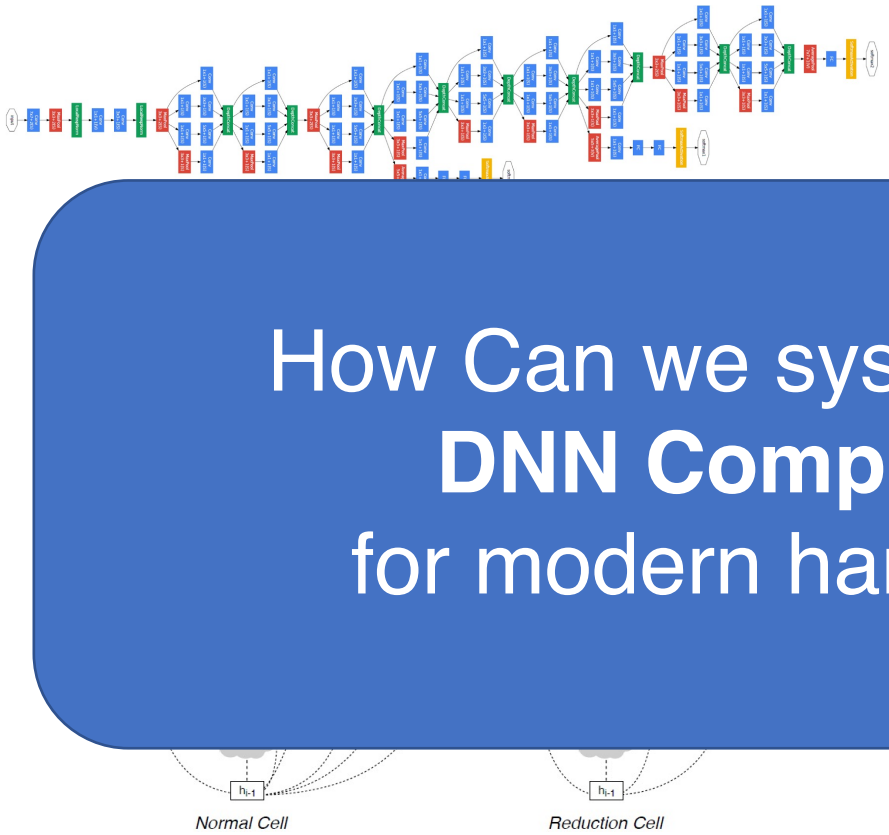
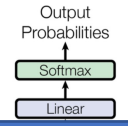


Figure 1: The Transformer - model architecture.

Attention models

Complex and Diverse DNN Computation Graphs





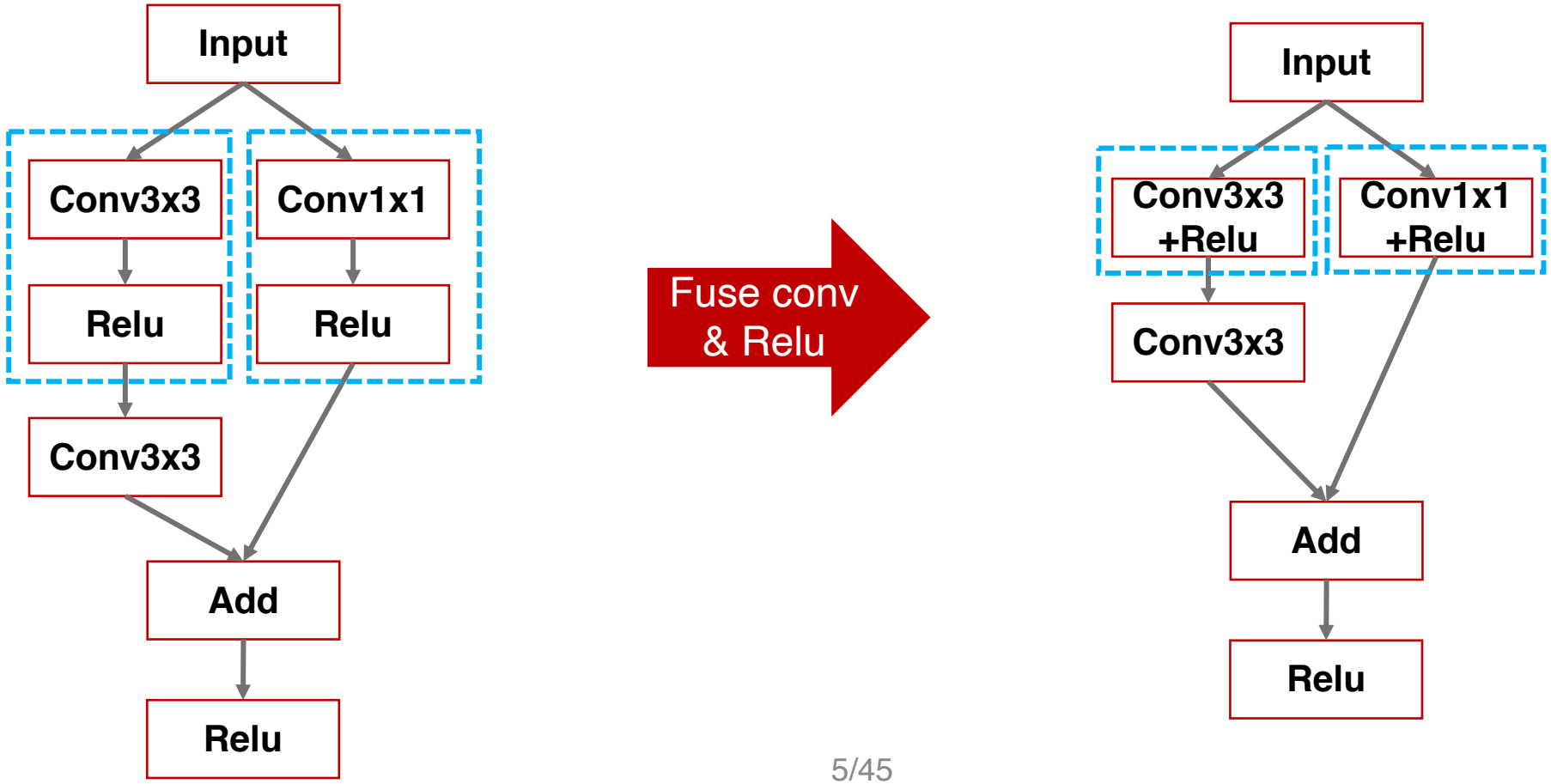
How Can we systematically optimize
DNN Computation graphs
for modern hardware platforms?

Neural Architecture Search

Attention models

Current Practice : Heuristic Approaches

- Perform graph transformations designed by domain experts.
  - E.g., fuse a convolution and a Relu into a “Conv+Relu”

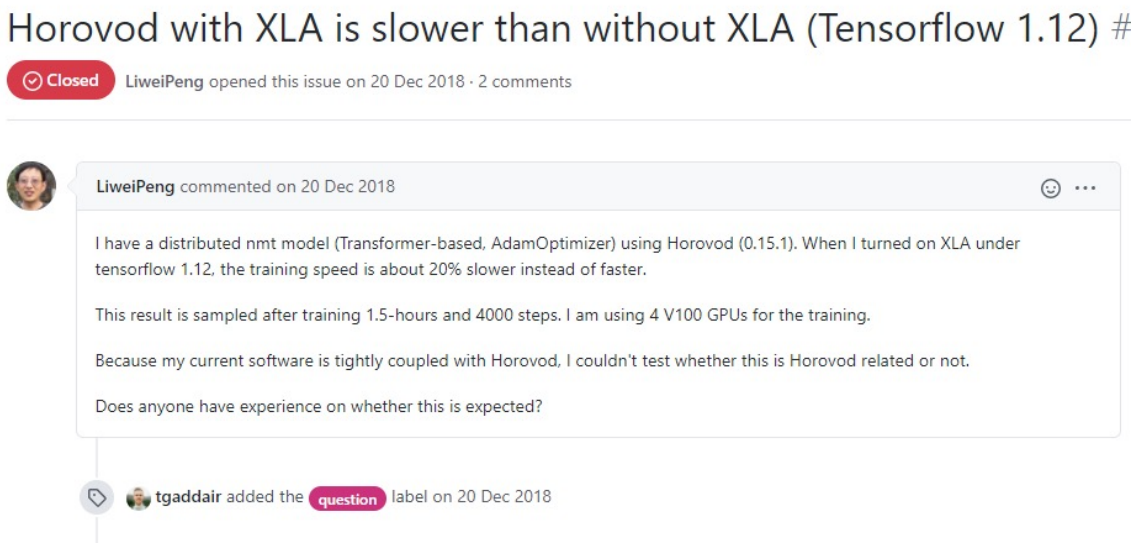


Limitation of heuristic Approaches

Robustness
Expert's heuristics do not apply to all models/hardware

Limitation of heuristic Approaches

Robustness
Expert's heuristics do not apply to all models/hardware



Horovod with XLA is slower than without XLA (Tensorflow 1.12) #

Closed LiweiPeng opened this issue on 20 Dec 2018 · 2 comments

LiweiPeng commented on 20 Dec 2018

I have a distributed nmt model (Transformer-based, AdamOptimizer) using Horovod (0.15.1). When I turned on XLA under tensorflow 1.12, the training speed is about 20% slower instead of faster.

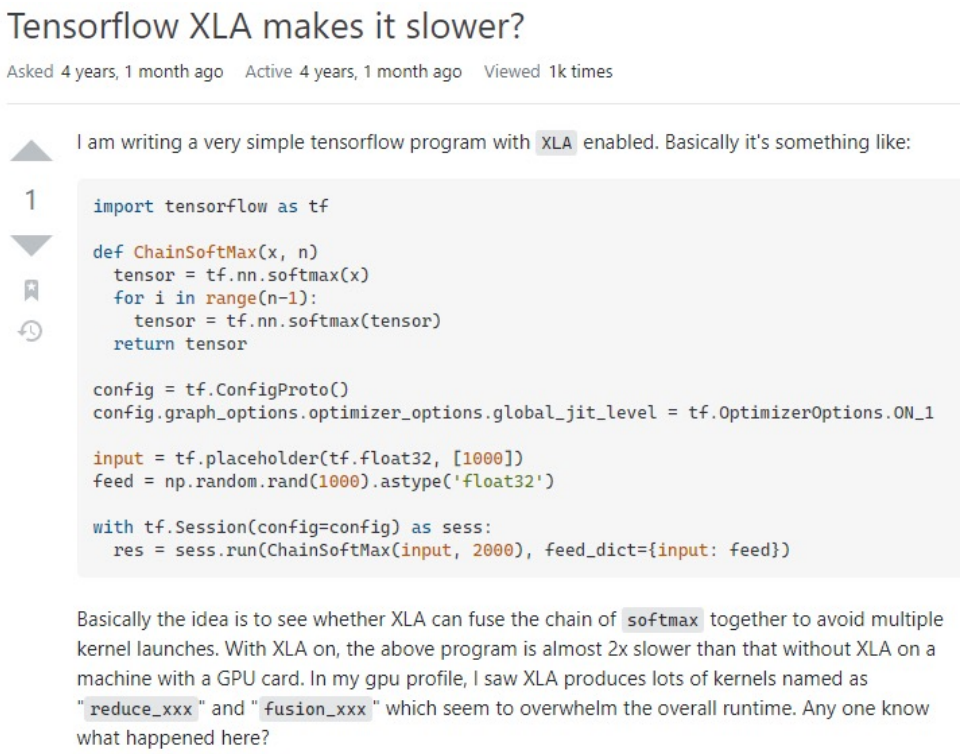
This result is sampled after training 1.5-hours and 4000 steps. I am using 4 V100 GPUs for the training.

Because my current software is tightly coupled with Horovod, I couldn't test whether this is Horovod related or not.

Does anyone have experience on whether this is expected?

tgaddair added the question label on 20 Dec 2018

When I turned on XLA training speed is **about 20 % slower.**



Tensorflow XLA makes it slower?

Asked 4 years, 1 month ago Active 4 years, 1 month ago Viewed 1k times

1

I am writing a very simple tensorflow program with `XLA` enabled. Basically it's something like:

```
import tensorflow as tf

def ChainSoftMax(x, n)
  tensor = tf.nn.softmax(x)
  for i in range(n-1):
    tensor = tf.nn.softmax(tensor)
  return tensor

config = tf.ConfigProto()
config.graph_options.optimizer_options.global_jit_level = tf.OptimizerOptions.ON_1

input = tf.placeholder(tf.float32, [1000])
feed = np.random.rand(1000).astype('float32')

with tf.Session(config=config) as sess:
  res = sess.run(ChainSoftMax(input, 2000), feed_dict={input: feed})
```

Basically the idea is to see whether XLA can fuse the chain of `softmax` together to avoid multiple kernel launches. With XLA on, the above program is almost 2x slower than that without XLA on a machine with a GPU card. In my gpu profile, I saw XLA produces lots of kernels named as "`reduce_xxx`" and "`fusion_xxx`" which seem to overwhelm the overall runtime. Any one know what happened here?

With XLA, my program is **almost 2x slower** than that without XLA

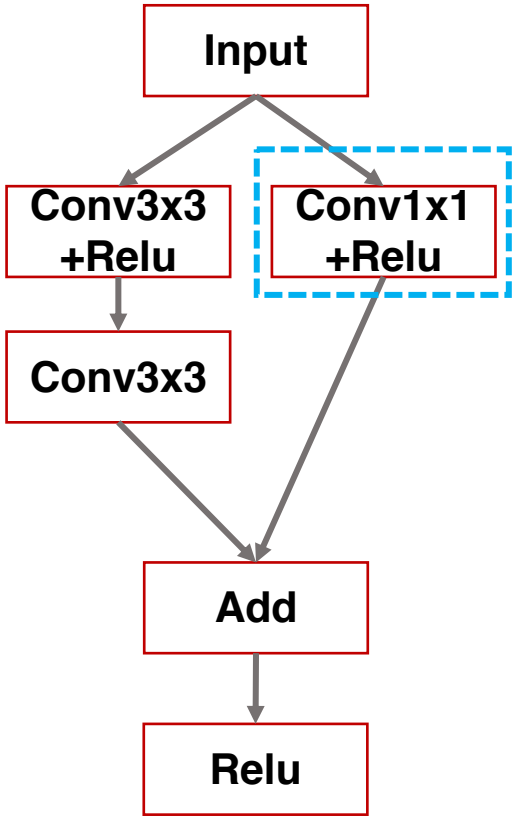
Limitation of heuristic Approaches

Robustness
Expert's heuristics do not apply to all models/hardware

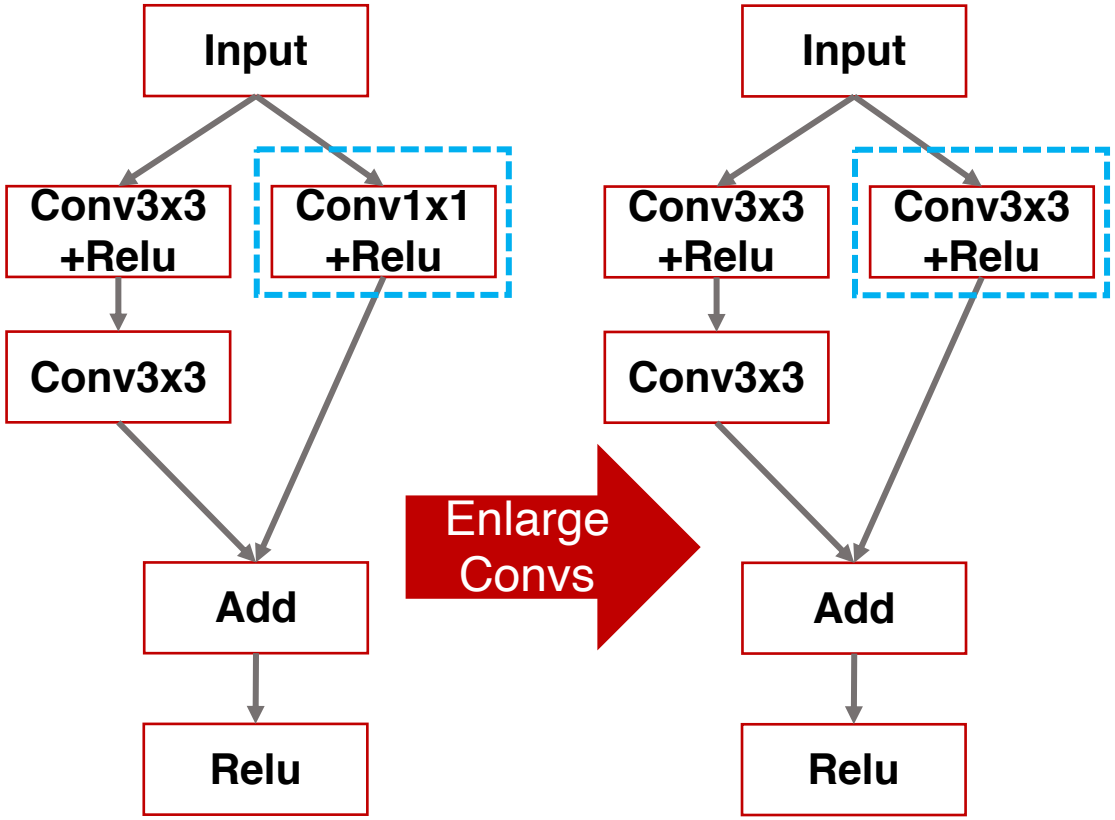
Scalability
New operators and graphs structures require more rules

Performance
Miss subtle optimizations for specific graphs

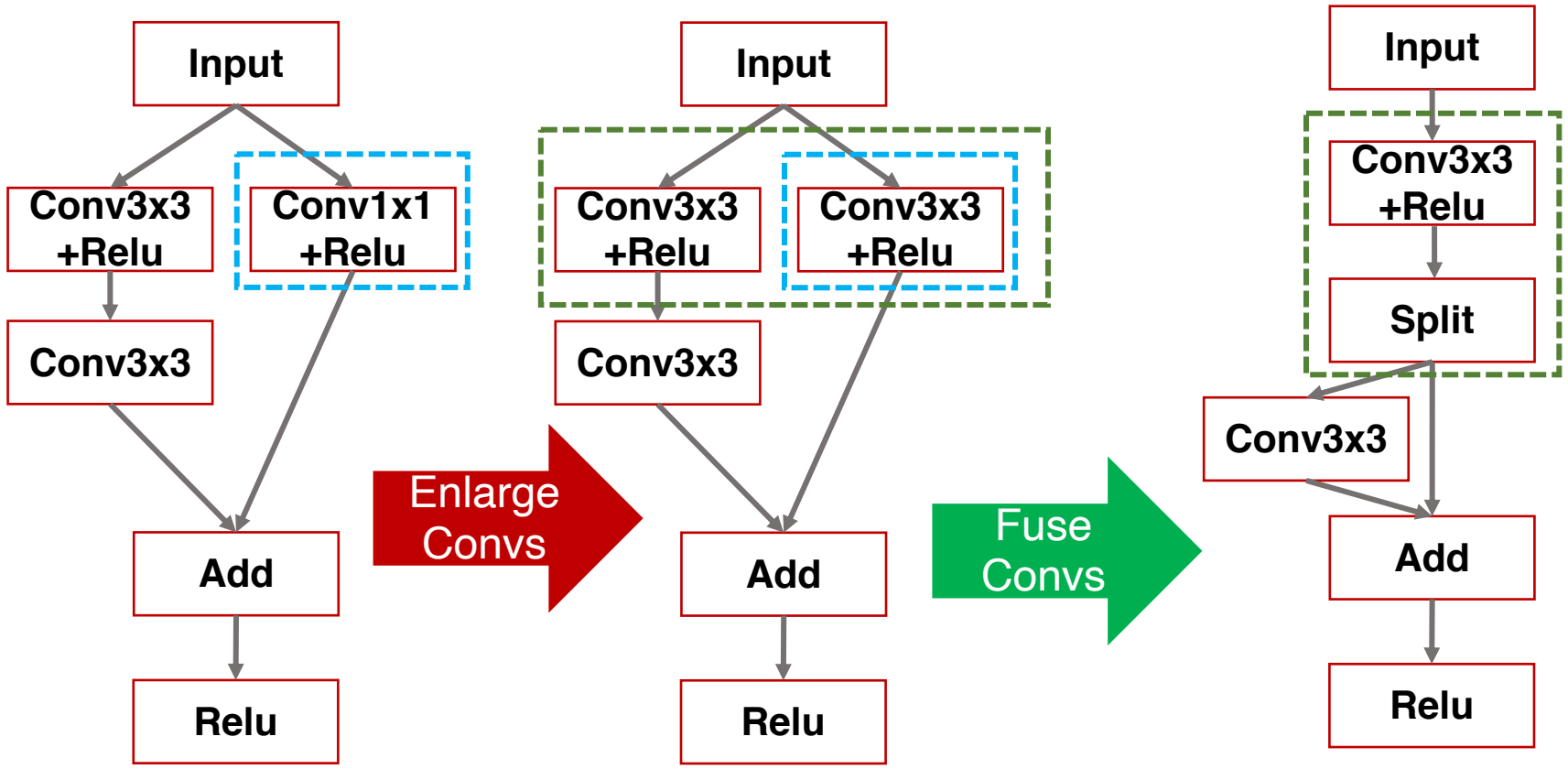
A Missing Graph Optimization



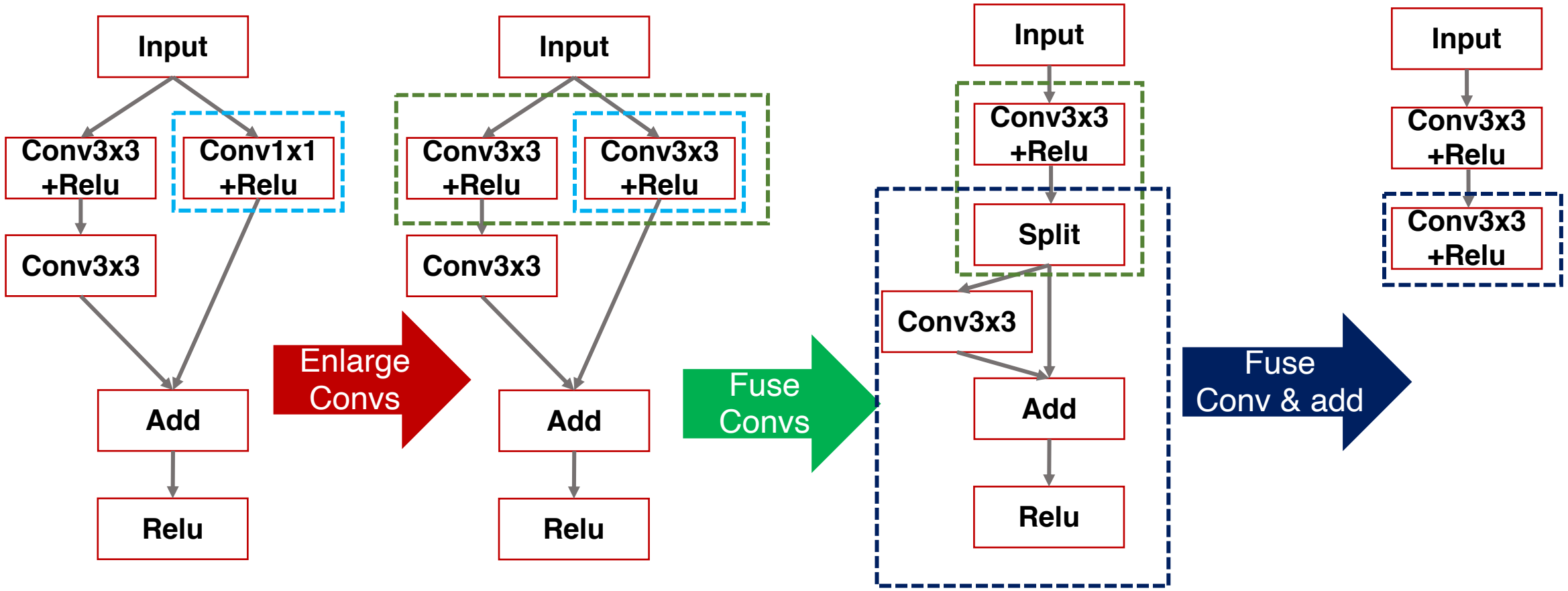
A Missing Graph Optimization



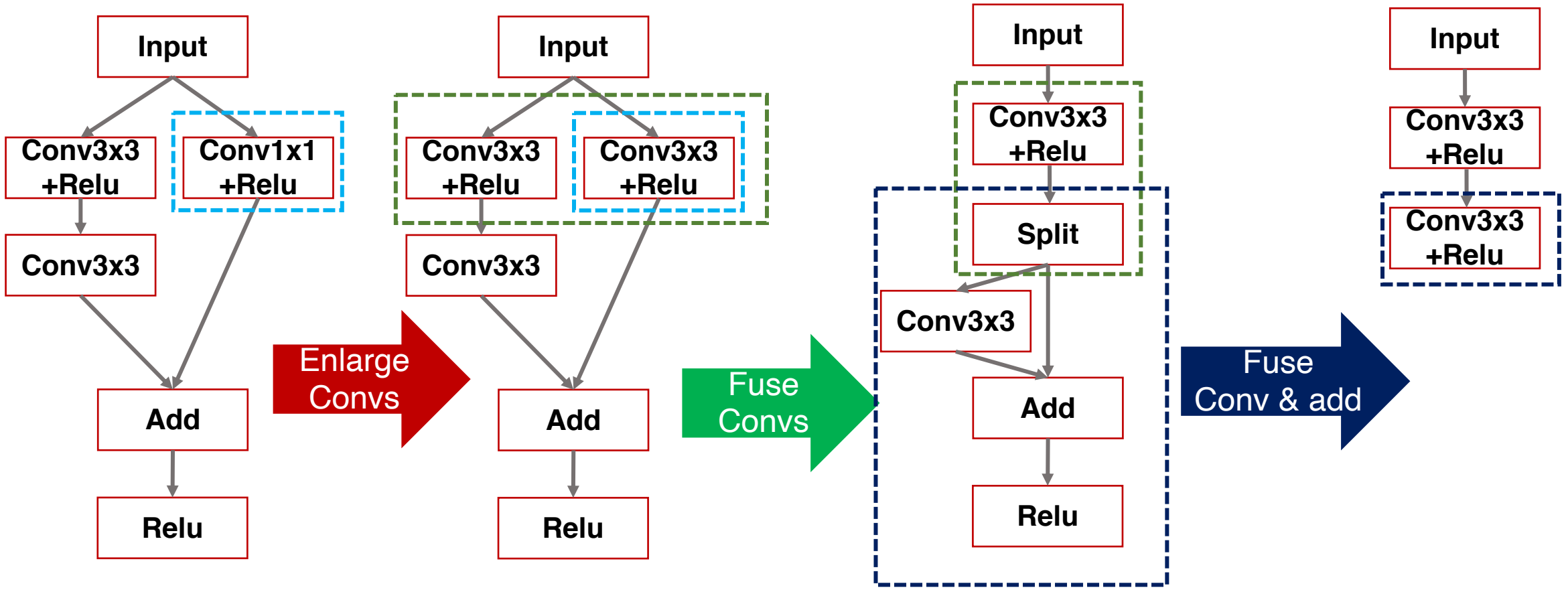
A Missing Graph Optimization



A Missing Graph Optimization



A Missing Graph Optimization



The Final Graph is 1.3 x faster on V100 but 10% slower on K80

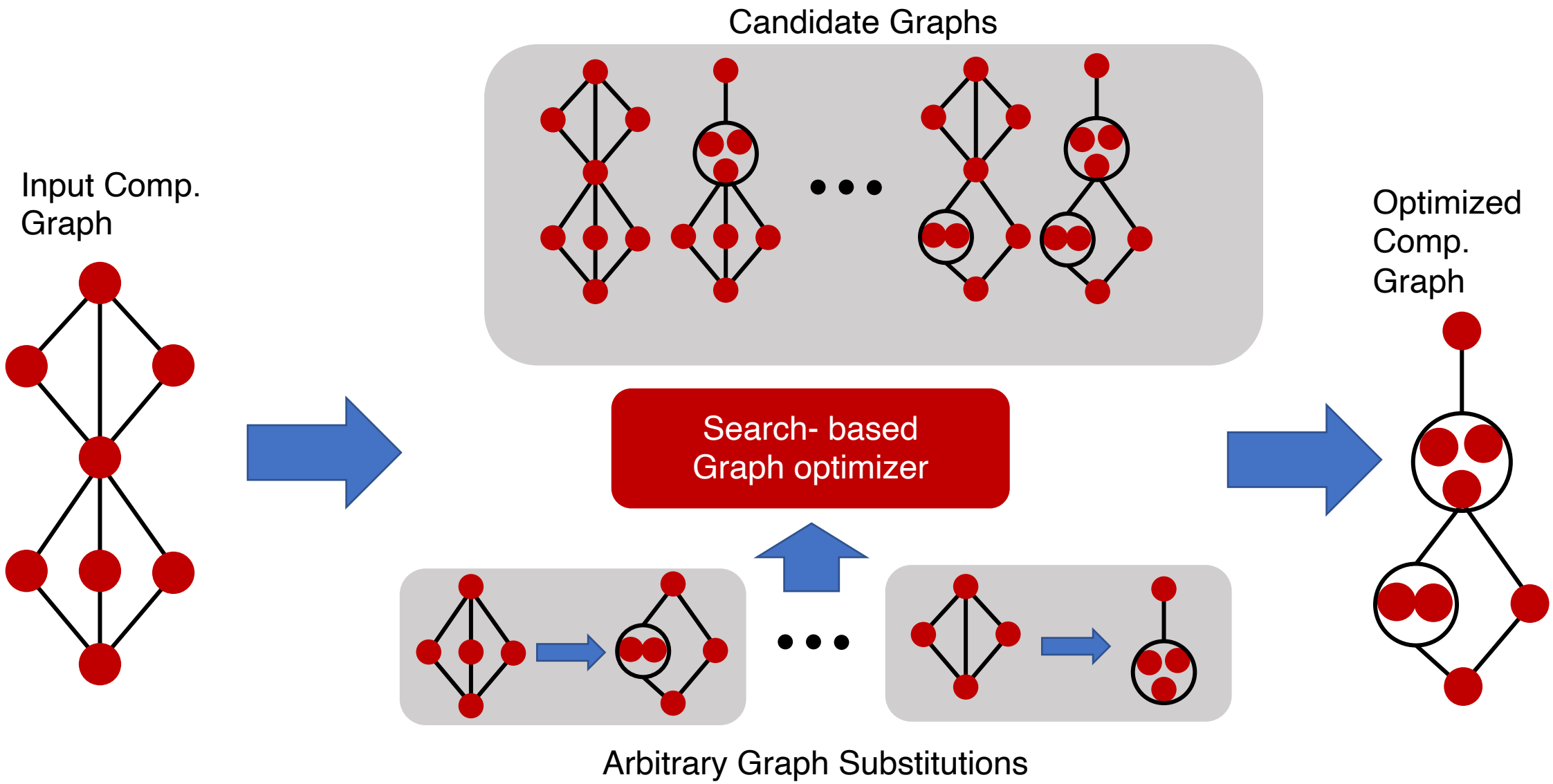
A Search based Approach

Arbitrary graph Substitution

A Search algorithm to explore the space of potential graphs

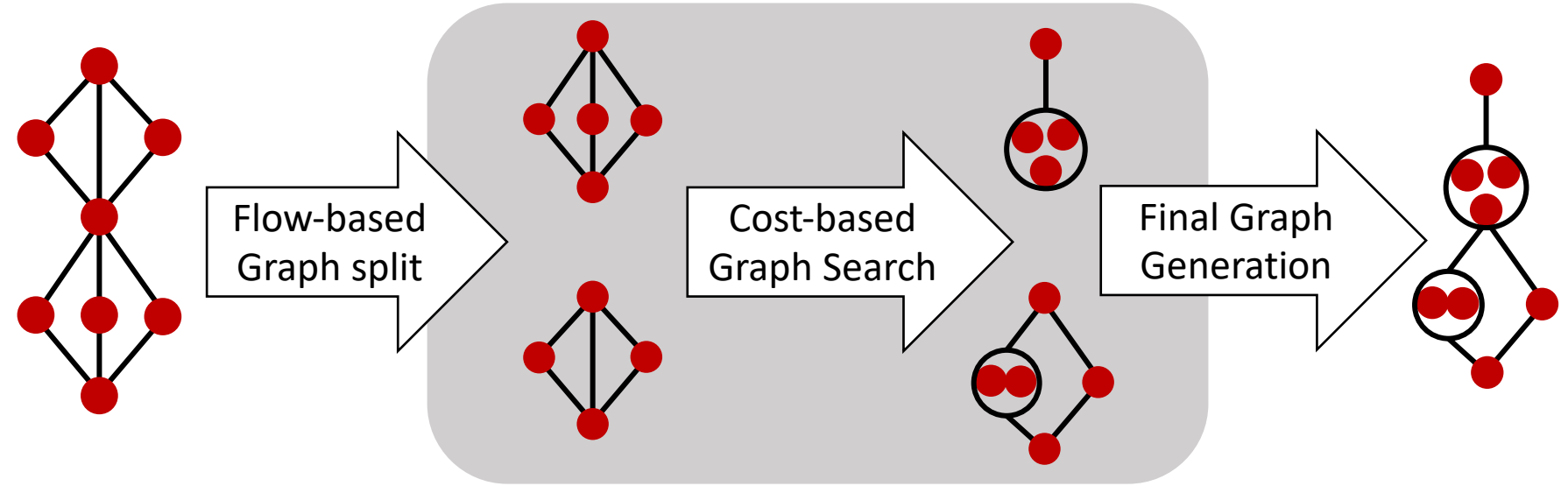
Optimized DNN Computation graph

MetaFlow overview



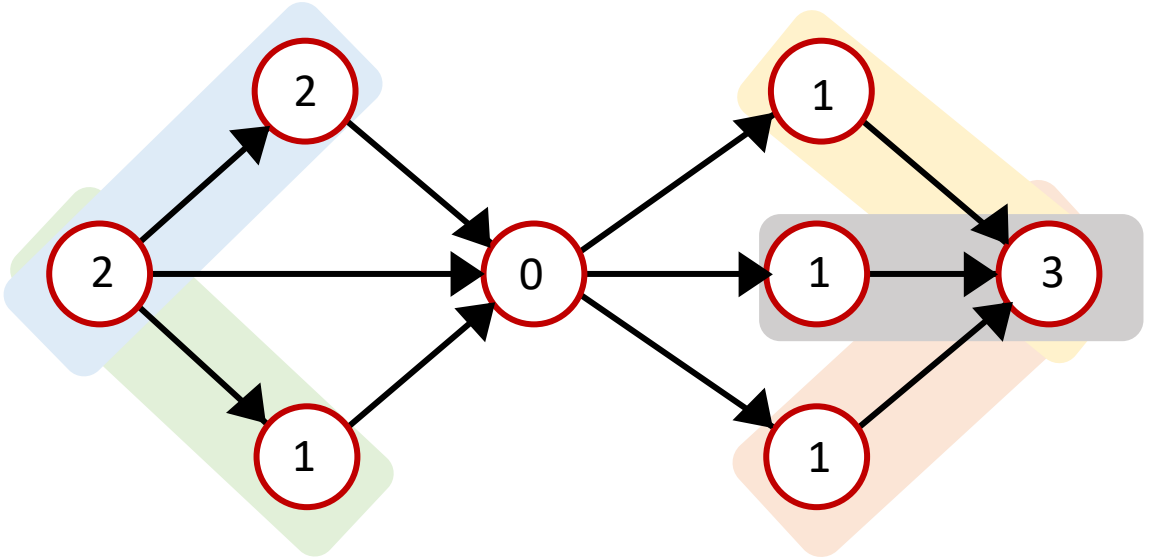
Challenges

- Irreversible graph substitutions (e.g 1x1 => 3x3)
  - Solution : Cost-based backtracking search
- Large search space of today's DNNs
  - Solution : Flow-based splitting to partition large graphs



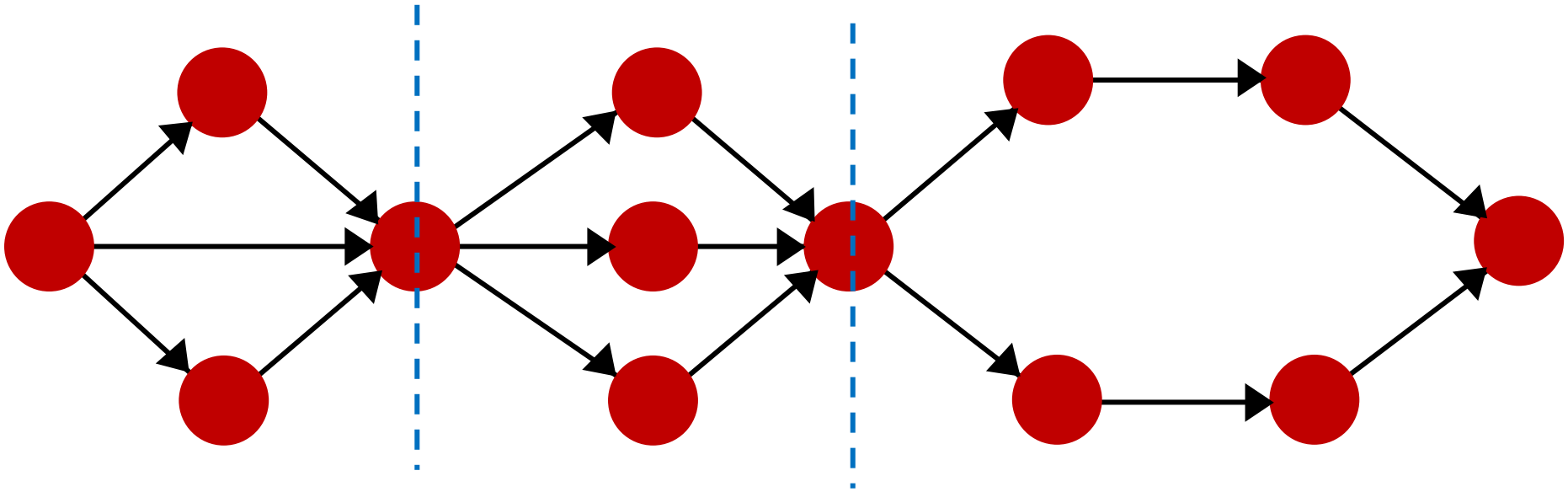
Flow-based recursive splitting

- **Problem** : Even good algorithms are too expensive for large graphs
- **Goal** : Break into small graphs, but minimize lost graph substitutions
- **Idea** : Formulate to a max-flow min-cut problem
  - Capacity(o) = # graph substitutions spanning operator o



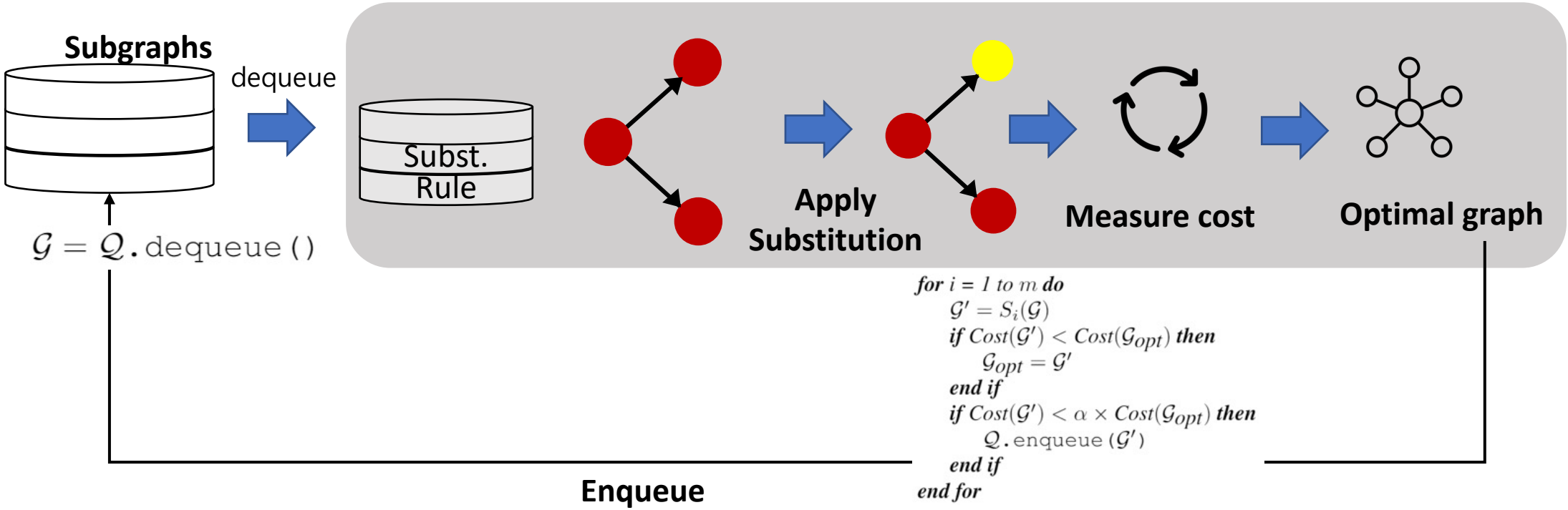
Cost based backtracking Search (CBS)

- Cost model
  - Based on the sum of individual operator costs
  - Measure the cost of each distinct operator on hardware
- Backtracking search for exploring irreversible substitutions



Cost based backtracking Search (CBS)

- Cost model
  - Based on the sum of individual operator costs
  - Measure the cost of each distinct operator on hardware
- Backtracking search for exploring irreversible substitutions

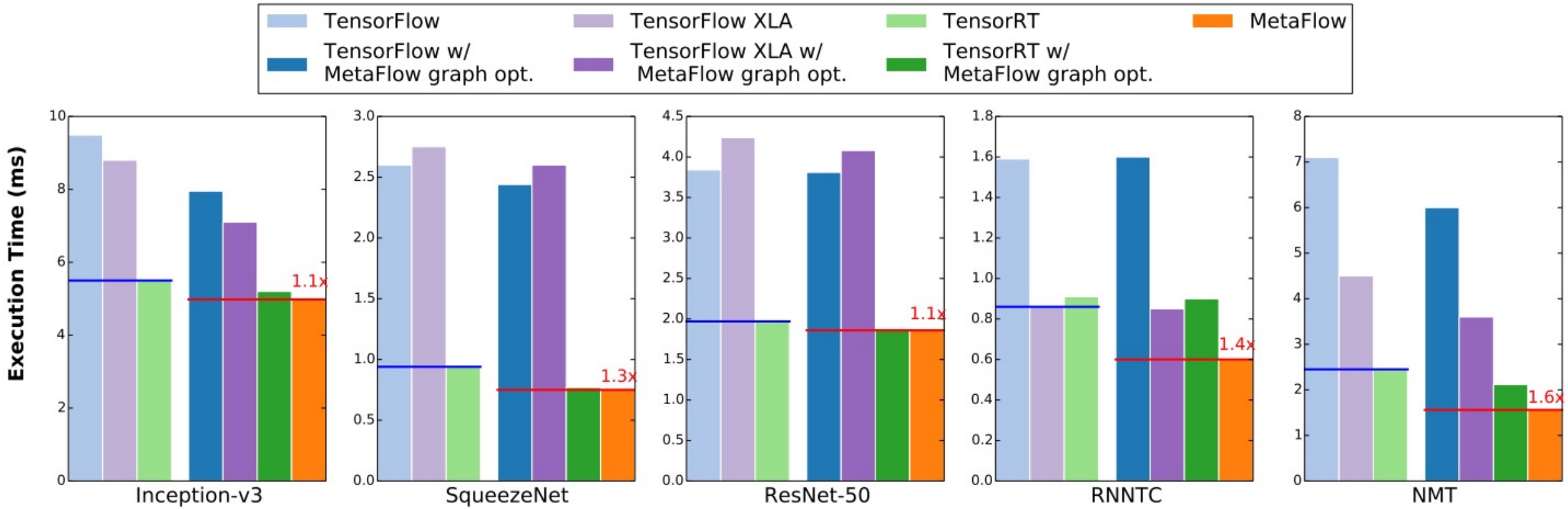


Evaluation

- Baselines
  - Tensorflow
  - Tensorflow XLA
  - TensorRT
- Benchmark DNNs
  - Inception-v3, SqueezeNet and ResNet-50
  - RNN Text Classification, Neural Machine Translation

Graph substitutions	
Performance decreasing	8
Operator fusions	10
Algebraic identities	4
Total	**22**

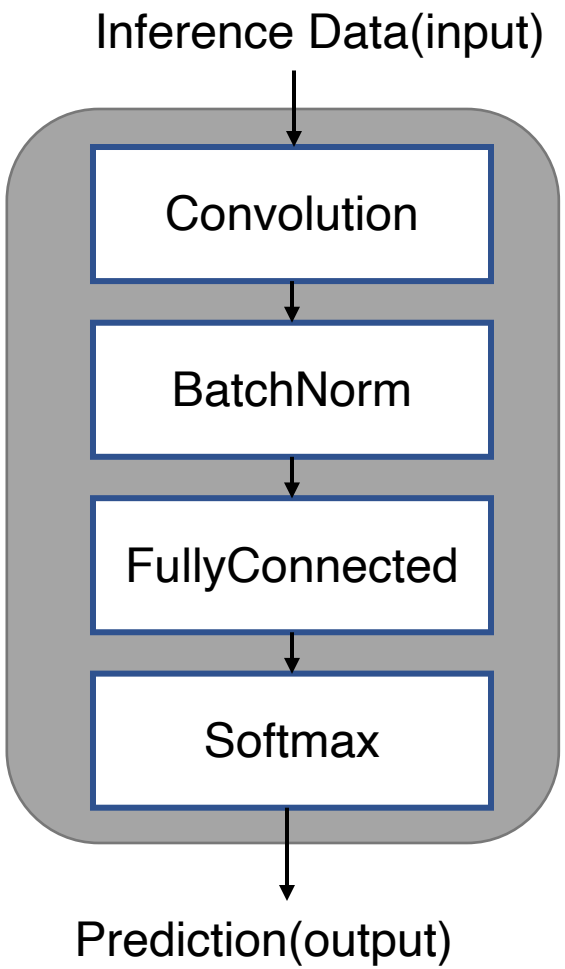
Can MetaFlow Improve End-to-End inference?



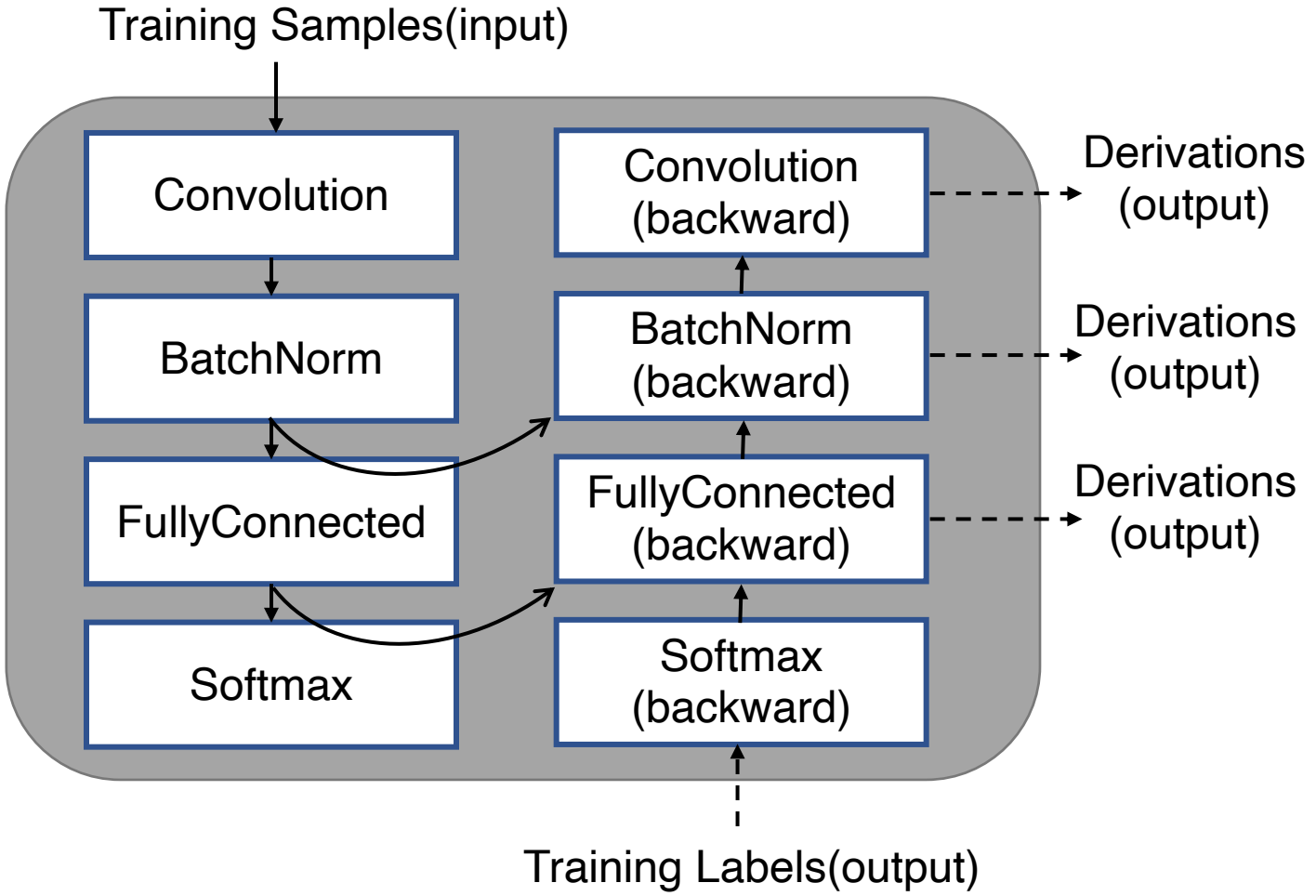
- Relative speedup over best of (TF,XLA,TensorRT) is from 1.1x up to 1.6x
- The **orange bars** indicate the inference time of MetaFlow's optimized graphs on the MetaFlow engine
- The improvement is achieved **without loss of model accuracy**

Can MetaFlow Improve Training Performance?

- Inference and training are defined as different computation graphs

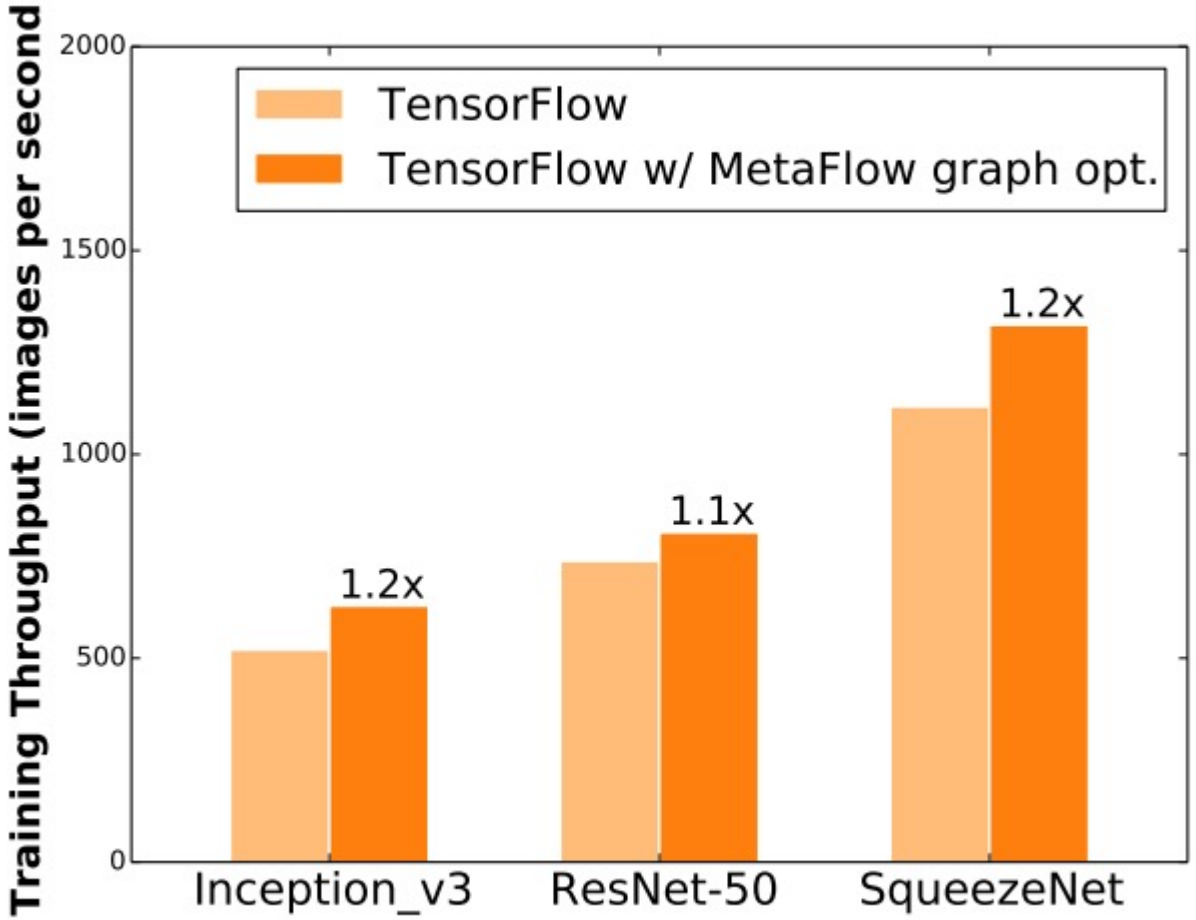


Inference Graph



Training Graph

Can MetaFlow Improve Training Performance?

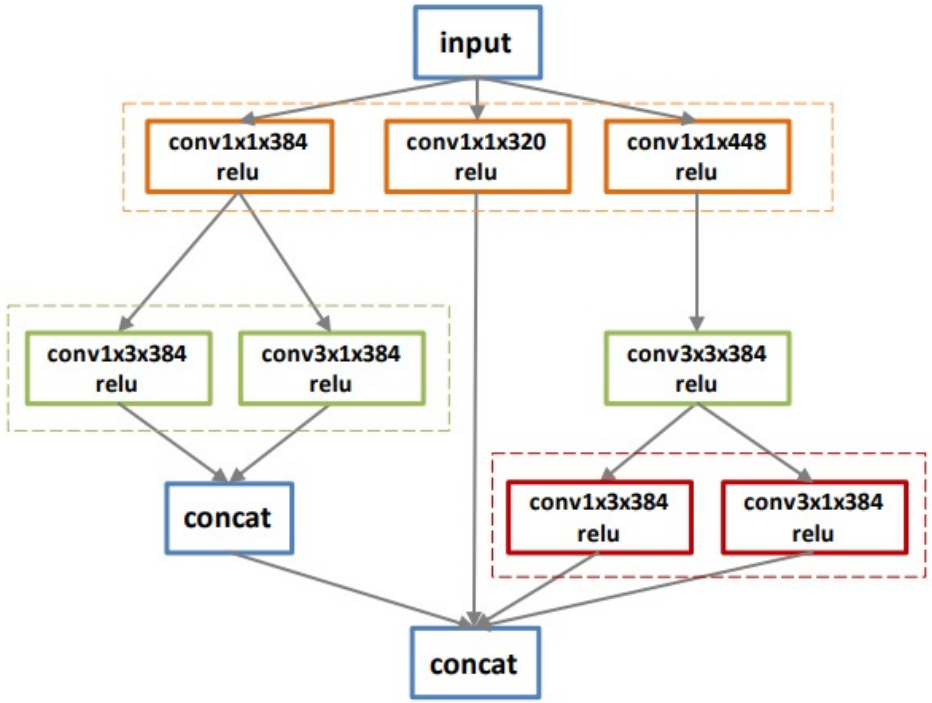


The experiments were performed on 4 NVIDIA V100 GPUs with data parallelism and a global batch size of 64

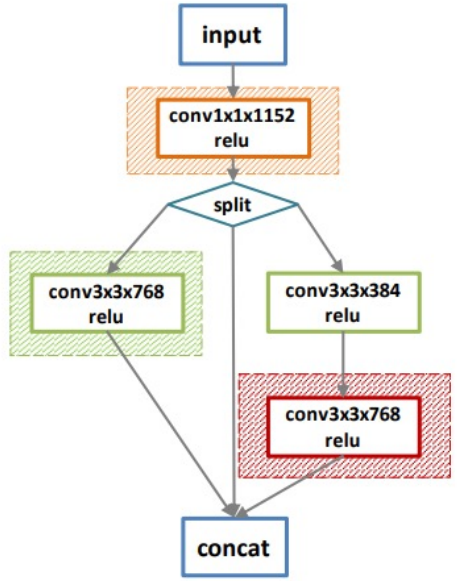
Smaller speedup for training due to additional data dependencies

Increase training throughput by up to 1.2×

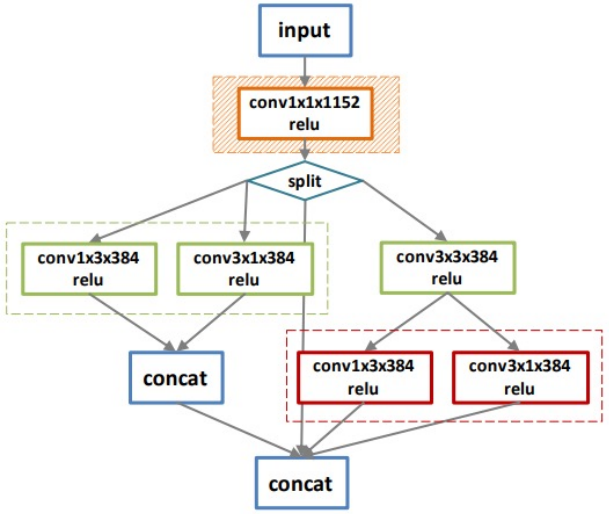
Hardware-specific Optimized Graphs



(a) Original graph.



(b) Optimized graph for V100.



(c) Optimized graph for K80.

Effectiveness of Cost model

Table 2. Performance comparison between MetaFlow and TensorRT on multiple cost dimensions. The experiments were performed on a NVIDIA V100 GPU. For TensorRT, the cost metrics are collected through its `Profiler` interface. The device utilization is computed by normalizing the FLOPs by the execution time (TFLOPs per second). For each cost dimension, a number in bold shows the one with better performance.

DNN	**Execution Time** (ms)		**Memory Accesses** (GB)		**Launched Kernels**		**FLOPs** (GFLOPs)		**Device Utilization**	
	TensorRT	MetaFlow	TensorRT	MetaFlow	TensorRT	MetaFlow	TensorRT	MetaFlow	TensorRT	MetaFlow
Inception-v3	5.51	**5.00**	95.4	**62.2**	138	**115**	**5.68**	5.69	1.03	**1.14**
SqueezeNet	0.94	**0.75**	62.1	**46.1**	50	**40**	**0.64**	1.00	0.68	**1.35**
ResNet50	1.97	**1.86**	37.2	**35.8**	70	**67**	**0.52**	0.54	0.26	**0.29**
RNNTC	0.91	**0.60**	1.33	**1.17**	220	**83**	0.22	**0.20**	0.24	**0.33**
NMT	2.45	**1.56**	5.32	**4.68**	440	**135**	0.84	**0.78**	0.34	**0.50**

Search Algorithm Performance

Table 3. Performance comparison between MetaFlow's backtracking search (with $\alpha = 1.05$) and a baseline exhaustive search on AlexNet, VGG16, ResNet18, and an Inception module shown in Figure 7a. A check mark indicates the backtracking search found the same optimal graph as the exhaustive search under the cost model.

Graph	Exhaustive Search	Backtracking Search	Same Result?
AlexNet	5.0 seconds	0.1 seconds	✓
VGG16	2.3 minutes	0.2 seconds	✓
InceptionE	12.8 minutes	0.29 seconds	✓
ResNet18	3.1 hours	0.99 seconds	✓

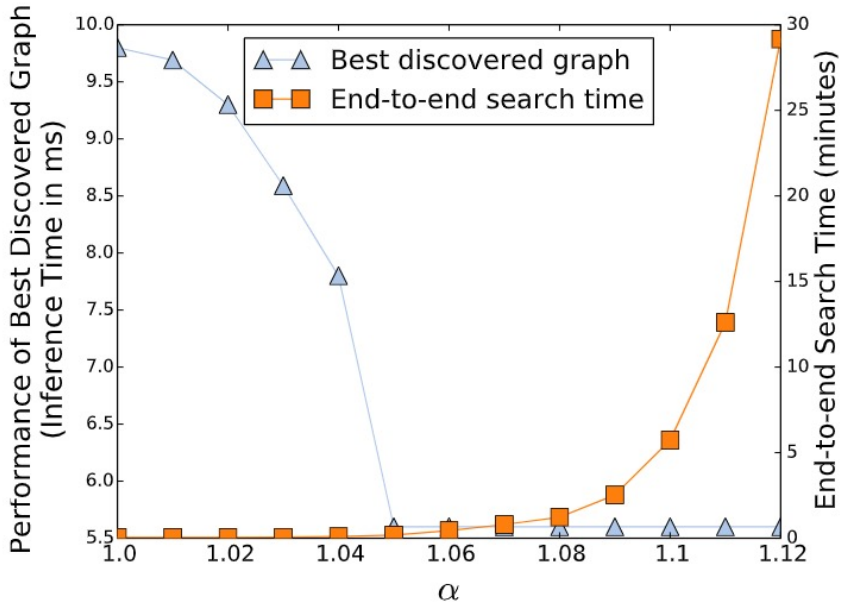


Figure 10. The performance of the best discovered graphs (shown as the red line) and the end-to-end search time for running Inception-v3 on a V100 GPU with different α.

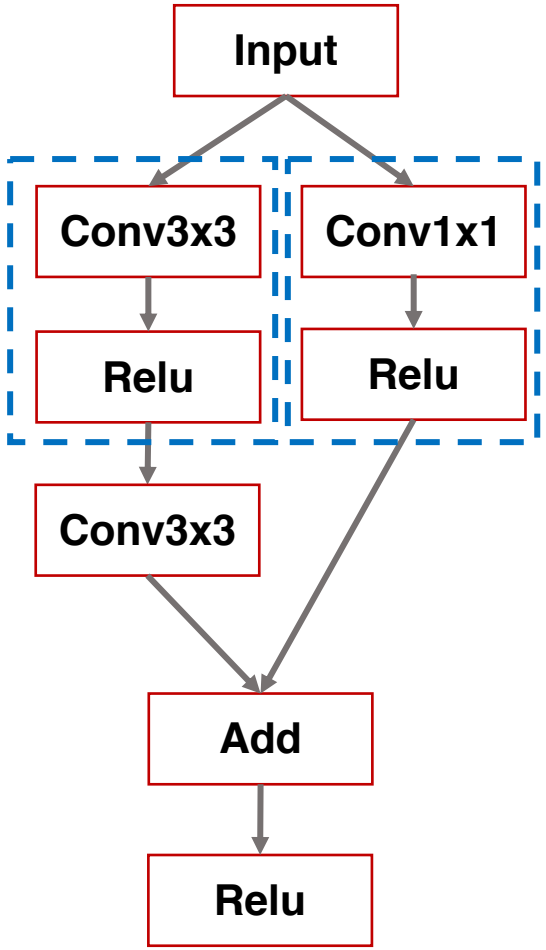
TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions

Zhihao Jia, Oded Padon, James Thomas,Todd Warszawski, Matei Zaharia, Alex Aiken

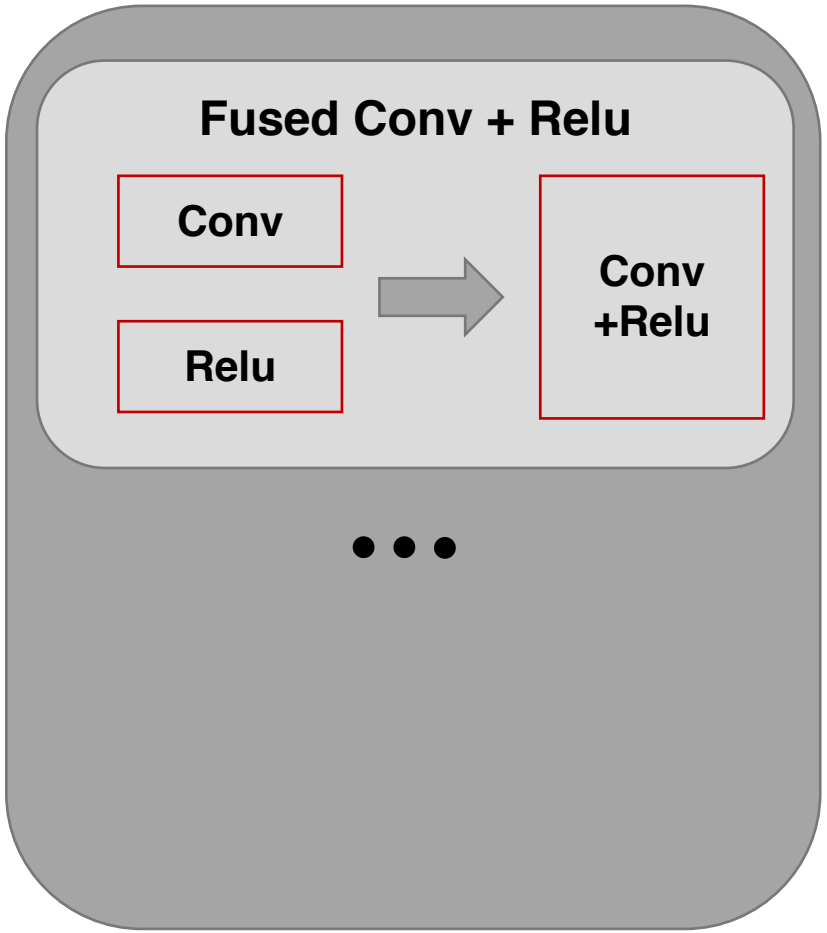
SOSP 19

Stanford University

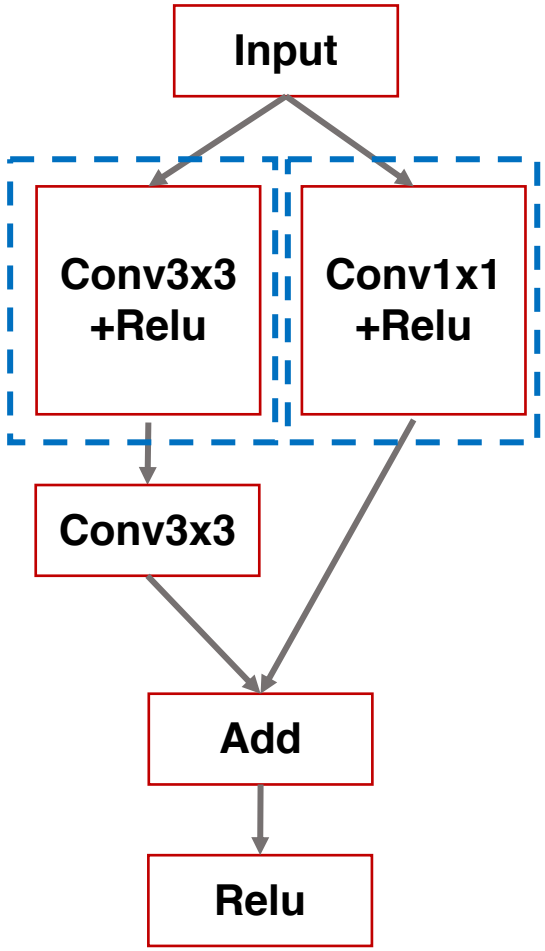
Current Rule-based DNN Optimizations



Computation Graph

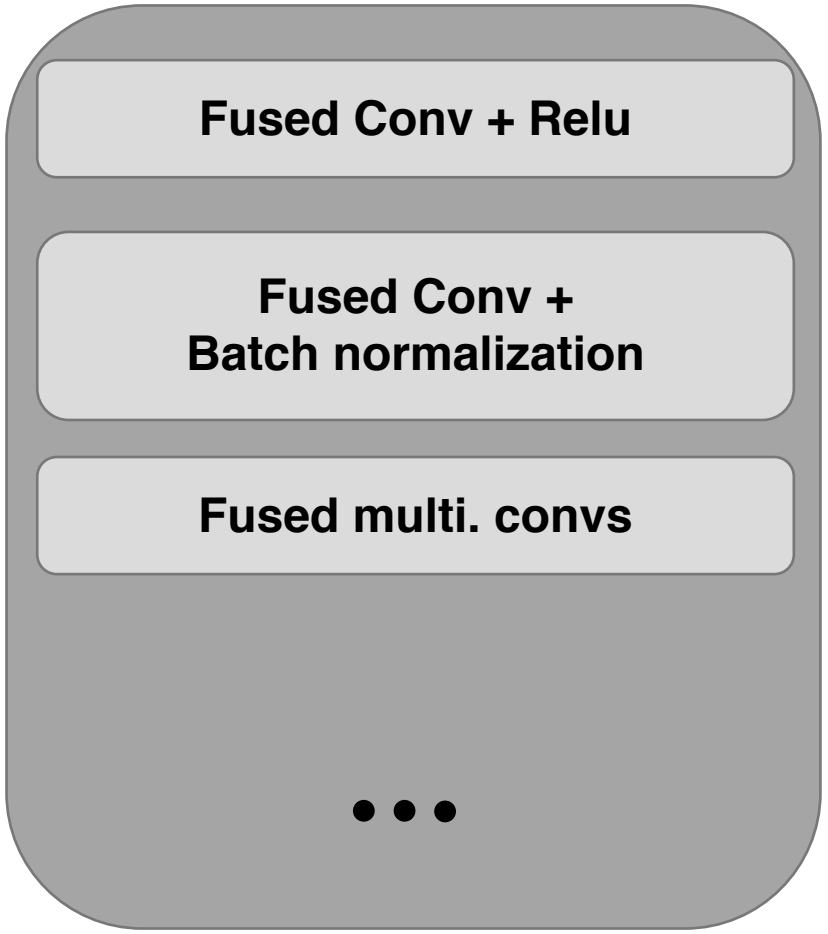


Rule-based Optimizer



Optimized Graph

Current Rule-based DNN Optimizations



Rule-based Optimizer

```
#include "tensorflow/tools/graph_transforms/file_utils.h"
#include "tensorflow/tools/graph_transforms/transform_utils.h"
#if !defined(PLATFORM_WINDOWS)
#include <pwd.h>
#include <unistd.h>
#endif

namespace tensorflow {
namespace graph_transforms {

using tensorflow::strings::Scanner;

Status ParseTransformParameters(const string& transforms_string,
                                TransformParameters* params_list) {
  params_list->clear();
  enum {
    TRANSFORM_NAME,
    TRANSFORM_PARAM_NAME,
    TRANSFORM_PARAM_VALUE,
  } state = TRANSFORM_NAME;
  StringPiece remaining(transforms_string);
  StringPiece match;
  StringPiece transform_name;
  StringPiece parameter_name;
  StringPiece parameter_value;
  TransformFuncParameters func_parameters;
  while (!remaining.empty()) {
    if (state == TRANSFORM_NAME) {
      // Reset the list of parameters.
      func_parameters.clear();
      // Eat up any leading spaces.
      Scanner(remaining).AnySpace().GetResult(&remaining, &match);
      if (remaining.empty()) {
      } else {
        // Add a transform with no parameters.
        params_list->push_back({string(transform_name), func_parameters});
        transform_name = "";
        state = TRANSFORM_NAME;
      }
```

Tensorflow currently
Includes ~ 200 rules
(~53,000 LOC)

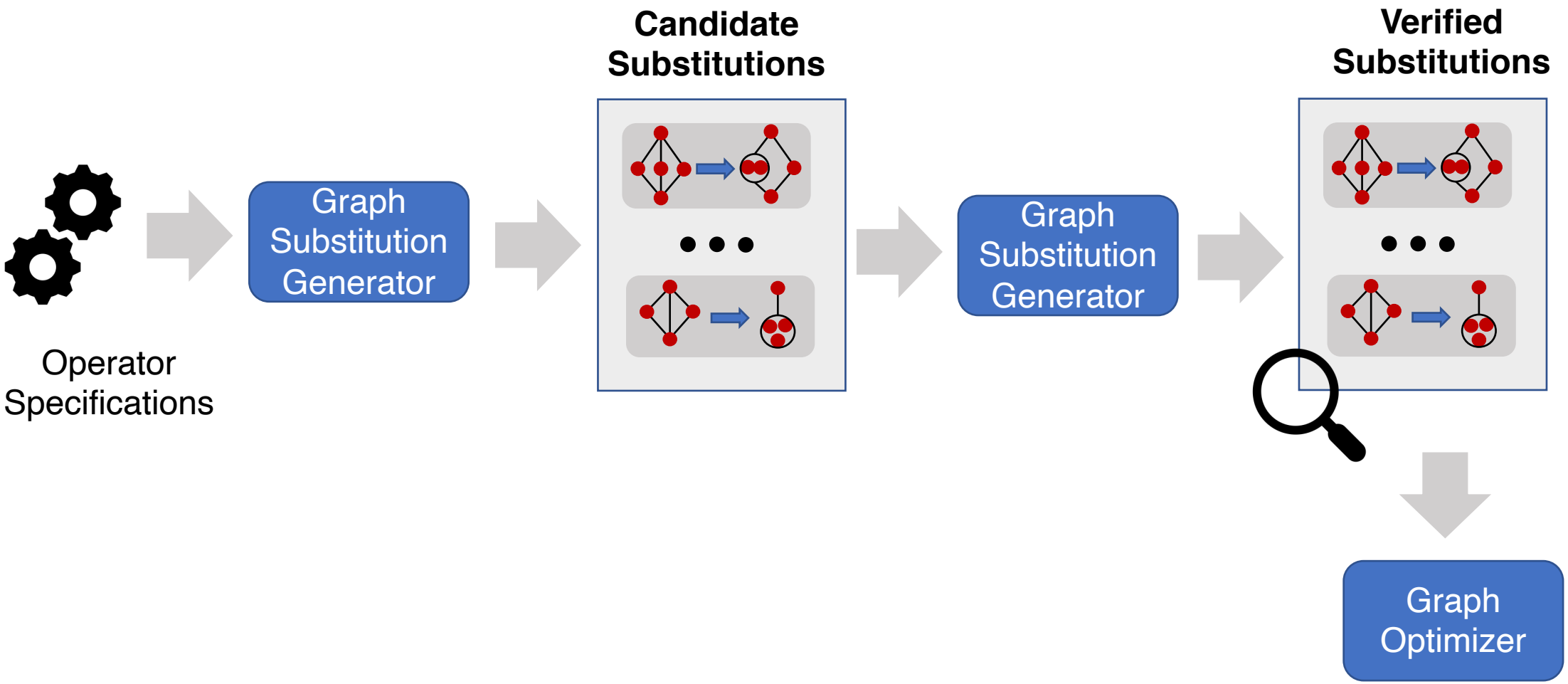
TASO : Tensor Algebra SuperOptimizer

- Key idea : replace manually-designed graph optimizations with **automated generation and verification** of graph substitutions for deep learning

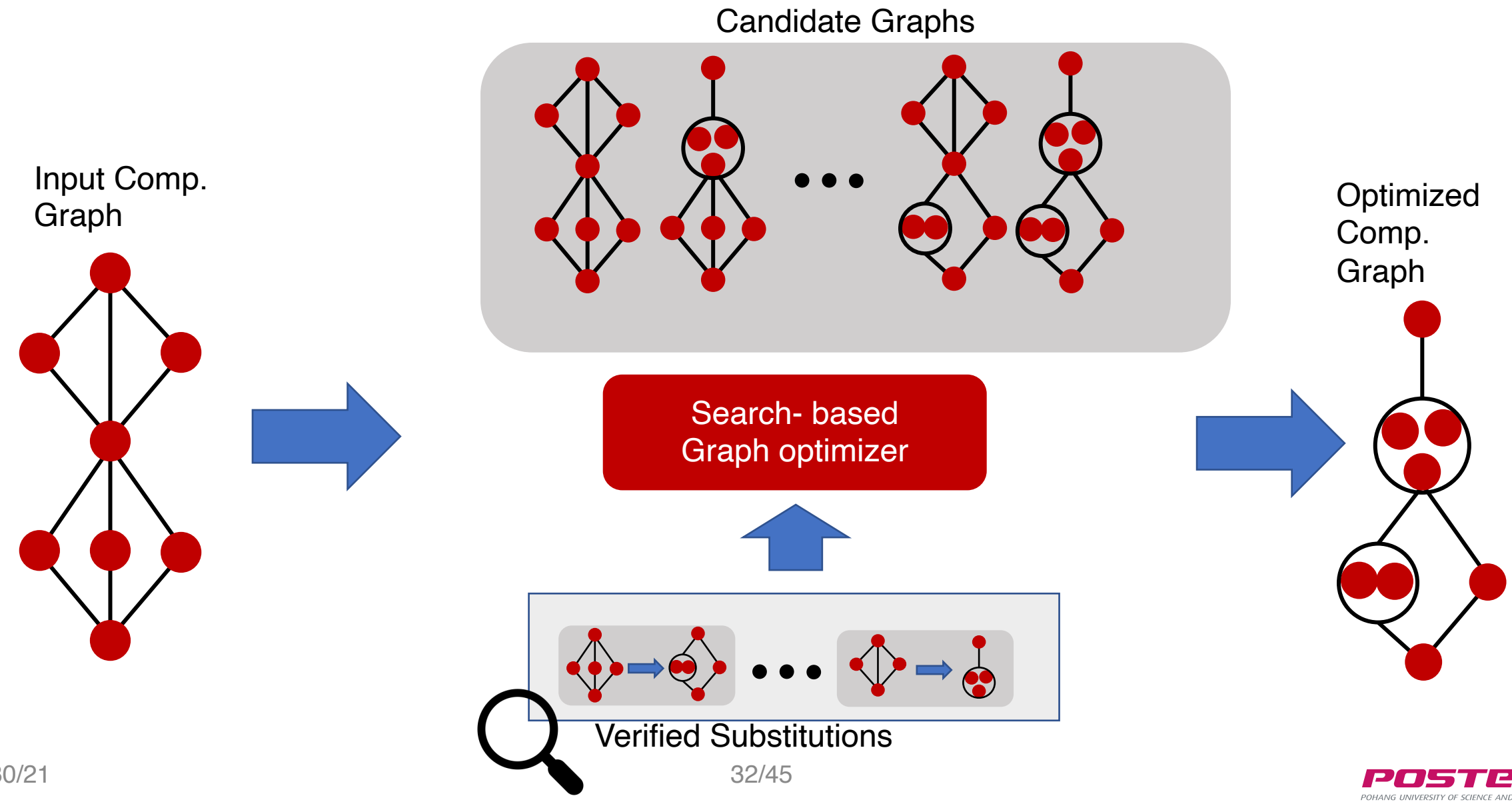
- **Less engineering effort** : **53,000** LOC for manual graph optimization in TensorFlow -> **1,400** LOC in TASO

- **Better performance**: outperformance existing optimizers by up to **2.8x**

TASO Workflow



TASO overview

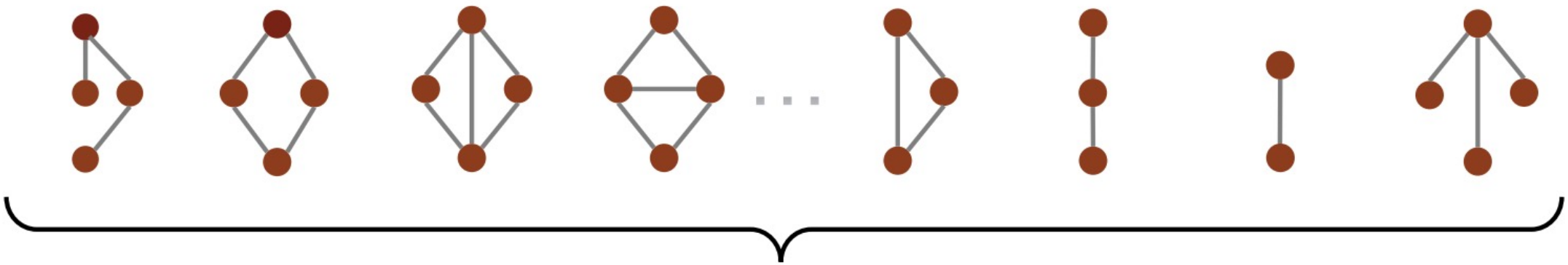


Key Challenges

1. How to generate potential substitutions?
   - Graph fingerprints
2. How to verify their correctness
   - Operator specifications + theorem prover

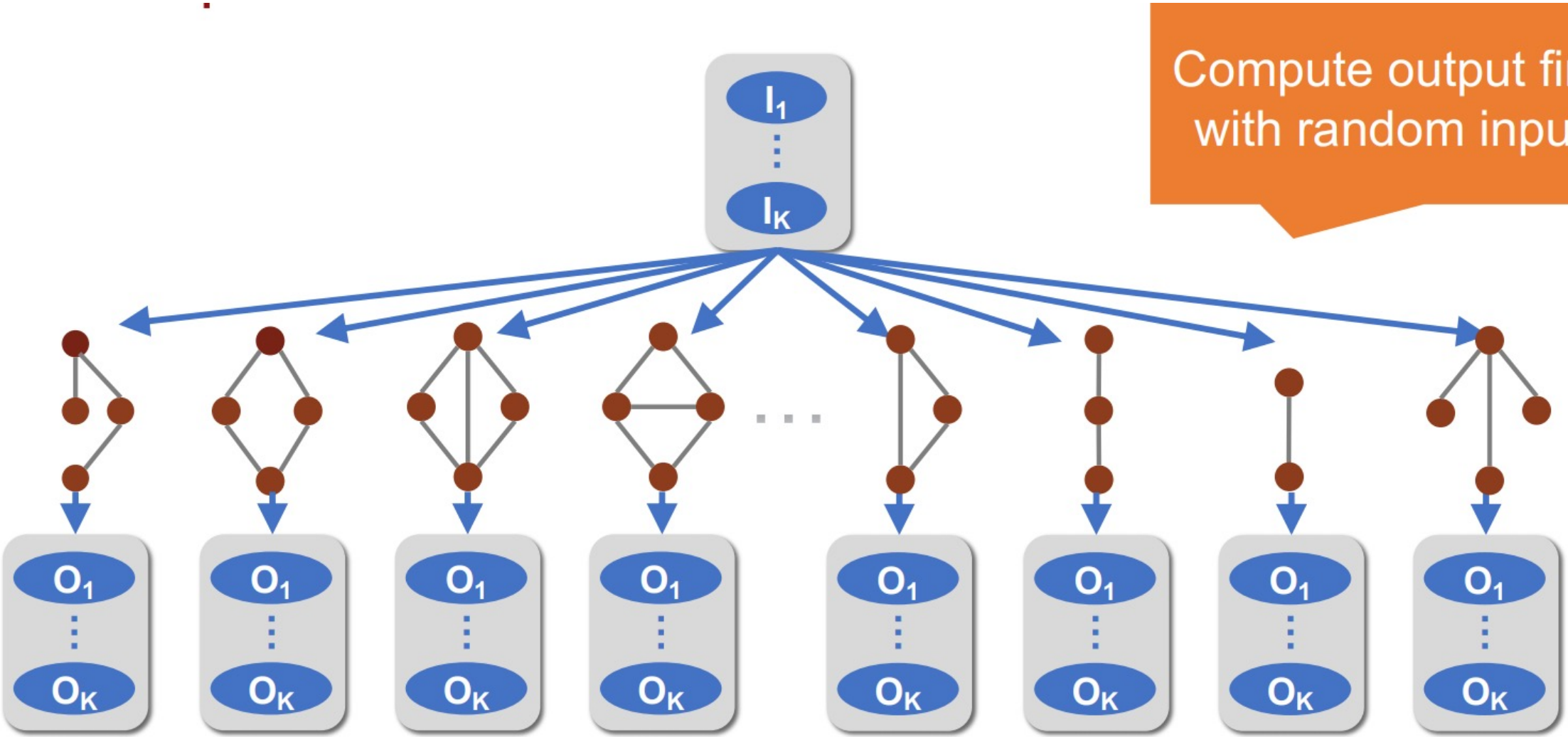
Graph Substitution Generator

Enumerate all possible subgraphs up to a fixed size using available operators

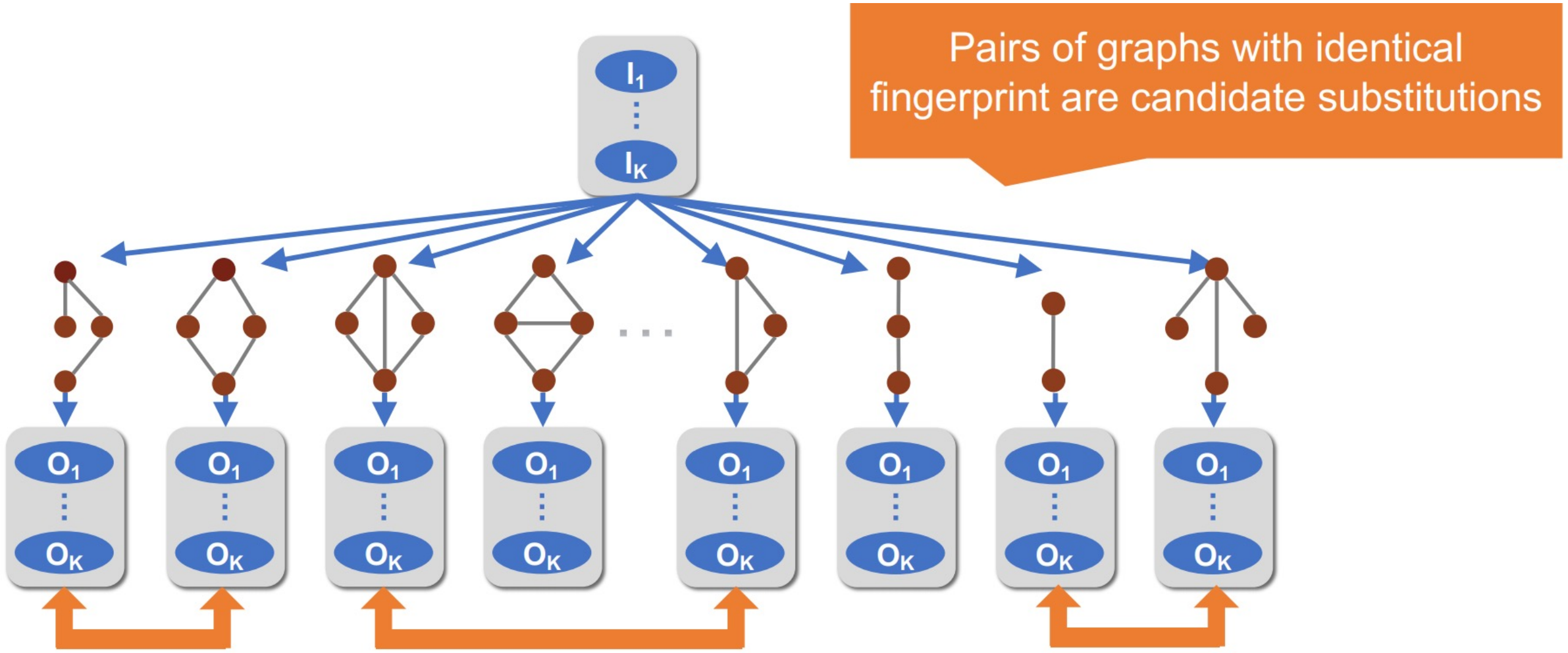


~66M graphs with up to 4 operators

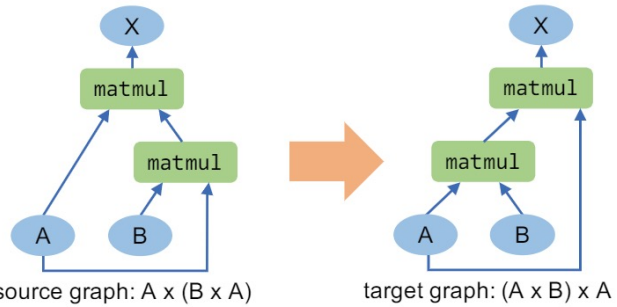
Graph Substitution Generator



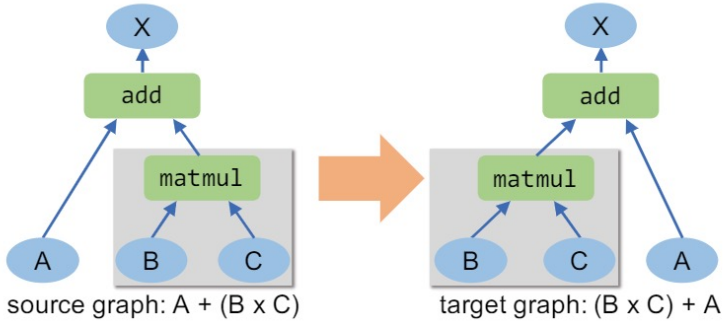
Graph Substitution Generator



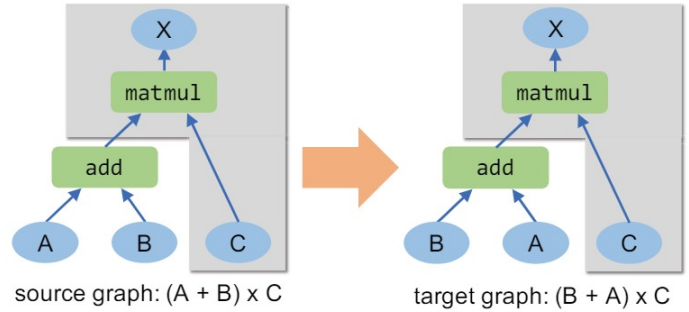
Pruning Redundant Substitutions



(a) A redundant substitution that is equivalent to Figure 2a by renaming input tensor C with A.



(b) A redundant substitution with a common subgraph.



(c) A redundant substitution with a common subgraph.

Table 3. The number of remaining graph substitutions after applying the pruning techniques in order.

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

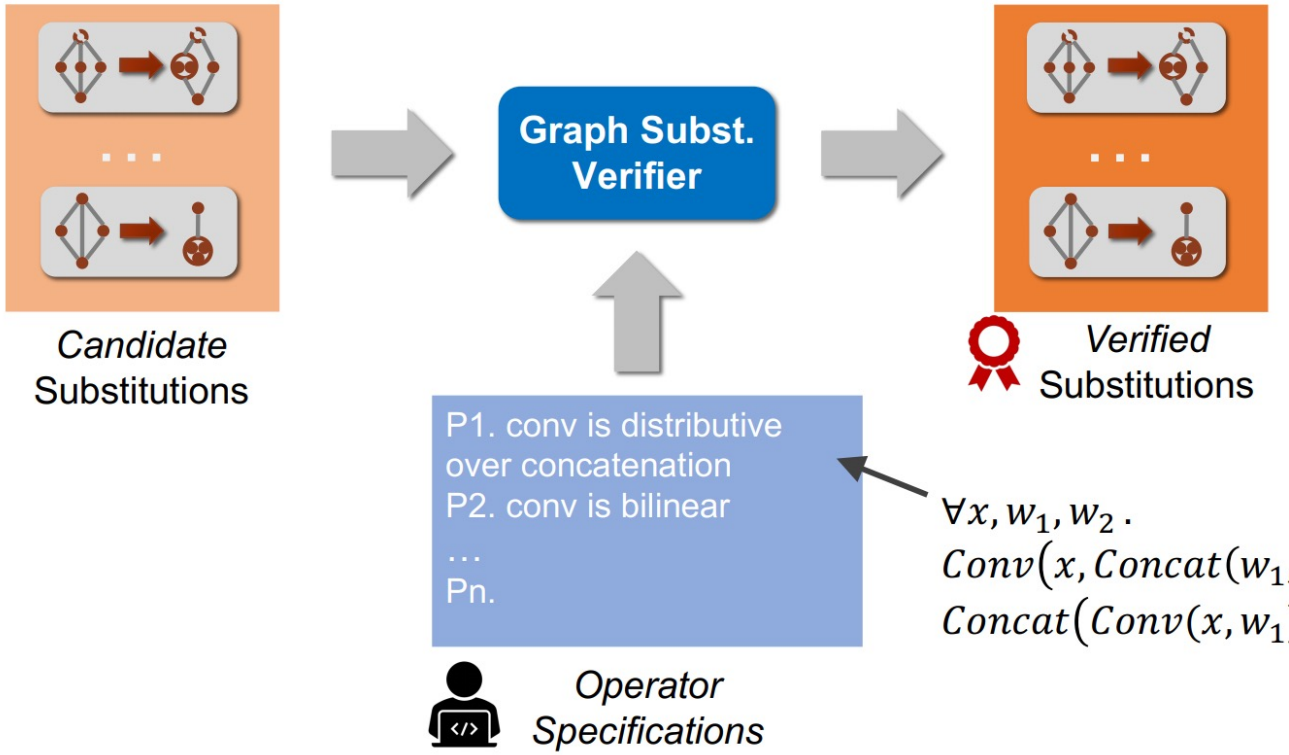
- **Common subgraph**
- **Input tensor renaming**
  - The both graphs contain a common operator with the same input tensors
  - the common subgraph includes all the outputs in both the source and target graphs.

Graph Substitution Generator

TASO generates ~**29,000** substitutions by enumerating graphs w/ up to 4 operators

743 substitutions remain after applying pruning techniques to eliminate redundancy

Graph Substitution Verifier

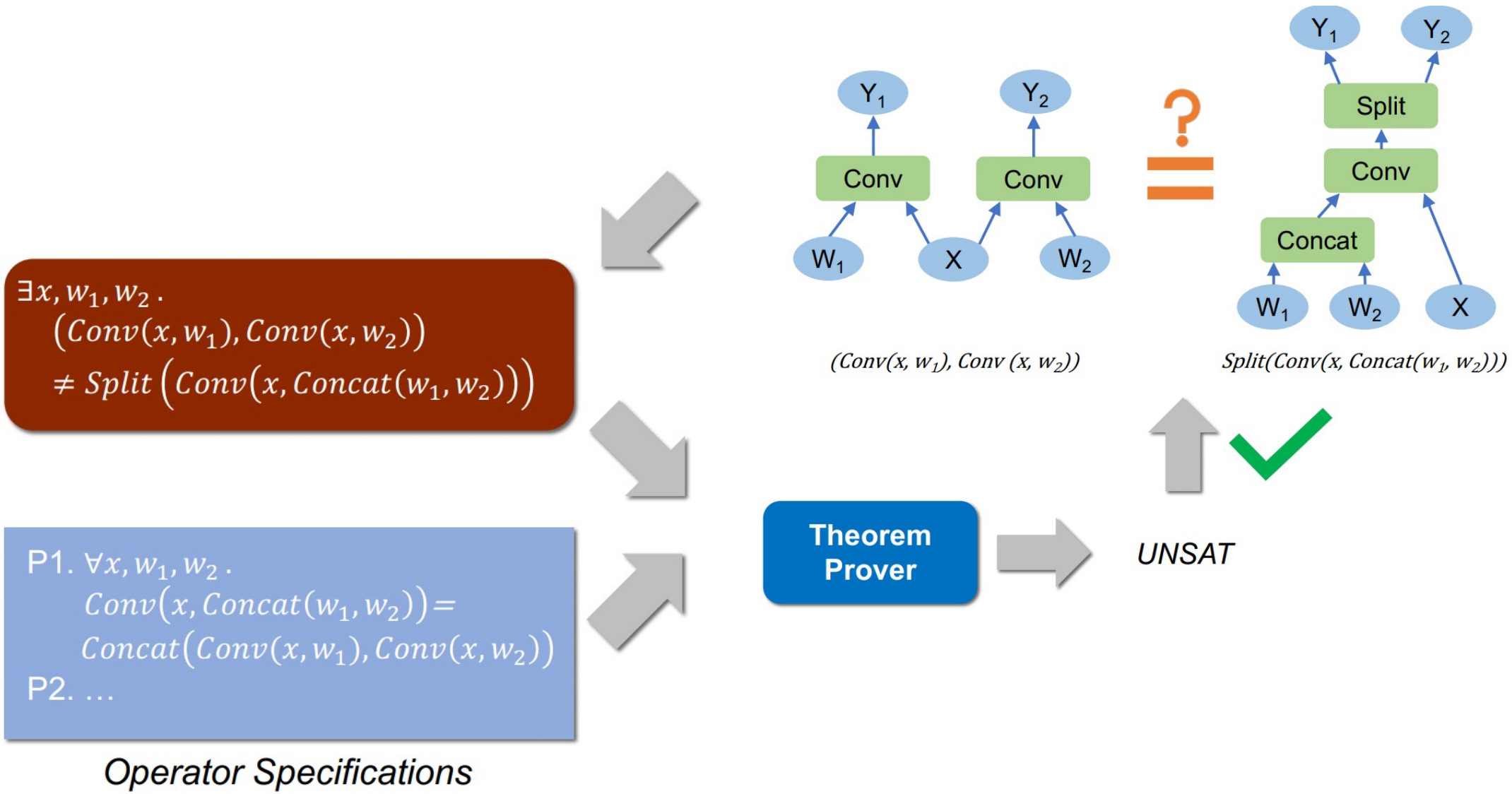


Operator properties used for verification

Operator Property	Comment
$\forall x, y, z.\ \texttt{ewadd}(x, \texttt{ewadd}(y, z)) = \texttt{ewadd}(\texttt{ewadd}(x, y), z)$	ewadd is associative
$\forall x, y.\ \texttt{ewadd}(x, y) = \texttt{ewadd}(y, x)$	ewadd is commutative
$\forall x, y, z.\ \texttt{ewmul}(x, \texttt{ewmul}(y, z)) = \texttt{ewmul}(\texttt{ewmul}(x, y), z)$	ewmul is associative
$\forall x, y.\ \texttt{ewmul}(x, y) = \texttt{ewmul}(y, x)$	ewmul is commutative
$\forall x, y, z.\ \texttt{ewmul}(\texttt{ewadd}(x, y), z) = \texttt{ewadd}(\texttt{ewmul}(x, z), \texttt{ewmul}(y, z))$	distributivity
$\forall x, y, w.\ \texttt{smul}(\texttt{smul}(x, y), w) = \texttt{smul}(x, \texttt{smul}(y, w))$	smul is associative
$\forall x, y, w.\ \texttt{smul}(\texttt{ewadd}(x, y), w) = \texttt{ewadd}(\texttt{smul}(x, w), \texttt{smul}(y, w))$	distributivity
$\forall x, y, w.\ \texttt{smul}(\texttt{ewmul}(x, y), w) = \texttt{ewmul}(x, \texttt{smul}(y, w))$	operator commutativity
$\forall x.\ \texttt{transpose}(\texttt{transpose}(x)) = x$	transpose is its own inverse
$\forall x, y.\ \texttt{transpose}(\texttt{ewadd}(x, y)) = \texttt{ewadd}(\texttt{transpose}(x), \texttt{transpose}(y))$	operator commutativity
$\forall x, y.\ \texttt{transpose}(\texttt{ewmul}(x, y)) = \texttt{ewmul}(\texttt{transpose}(x), \texttt{transpose}(y))$	operator commutativity
$\forall x, w.\ \texttt{smul}(\texttt{transpose}(x), w) = \texttt{transpose}(\texttt{smul}(x, w))$	operator commutativity
$\forall x, y, z.\ \texttt{matmul}(x, \texttt{matmul}(y, z)) = \texttt{matmul}(\texttt{matmul}(x, y), z)$	matmul is associative
$\forall x, y, w.\ \texttt{smul}(\texttt{matmul}(x, y), w) = \texttt{matmul}(x, \texttt{smul}(y, w))$	matmul is linear
$\forall x, y, z.\ \texttt{matmul}(x, \texttt{ewadd}(y, z)) = \texttt{ewadd}(\texttt{matmul}(x, y), \texttt{matmul}(x, z))$	matmul is linear

Given the operator properties, TASO use a first-order theorem prover(Z3)

Verification Workflow



Joint Optimizer

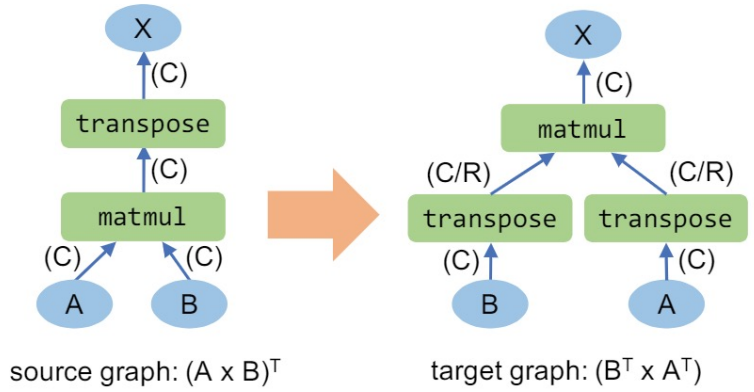


Figure 5. A graph substitution using the transpose of matrix multiplication. `matmul` and `transpose` indicate matrix multiplication and transpose, respectively. The parentheses show the potential layouts for each tensor in the source and target graphs, where `C` and `R` indicate the column-major and row-major layouts of a tensor.

Example of Layout Optimization

Algorithm 2 Cost-Based Backtracking Search

Input: an input graph $\mathcal{G}_{in}$, verified substitutions $\mathcal{S}$, a cost model $Cost(\cdot)$, and a hyper parameter α.
Output: an optimized graph.

$\mathcal{P} = \{\mathcal{G}_{in}\}$ // $\mathcal{P}$ *is a priority queue sorted by Cost.*
while $\mathcal{P} \neq \{\}$ **do**
  $\mathcal{G} = \mathcal{P}$.dequeue()
  for substitution $s \in \mathcal{S}$ **do**
    // LAYOUT$(\mathcal{G}, s)$ *returns possible layouts applying* s *on* $\mathcal{G}$.
    for layout $l \in$ LAYOUT$(\mathcal{G}, s)$ **do**
      // APPLY$(\mathcal{G}, s, l)$ *applies* s *on* $\mathcal{G}$ *with layout* l.
      $\mathcal{G}' =$ APPLY$(\mathcal{G}, s, l)$
      if $\mathcal{G}'$ is valid **then**
        if $Cost(\mathcal{G}') < Cost(\mathcal{G}_{opt})$ **then**
          $\mathcal{G}_{opt} = \mathcal{G}'$
        if $Cost(\mathcal{G}') < \alpha \times Cost(\mathcal{G}_{opt})$ **then**
          $\mathcal{P}$.enqueue($\mathcal{G}'$)
return $\mathcal{G}_{opt}$

MetaFlow Approach + Layout Optimization

End-to-end Inference Performance

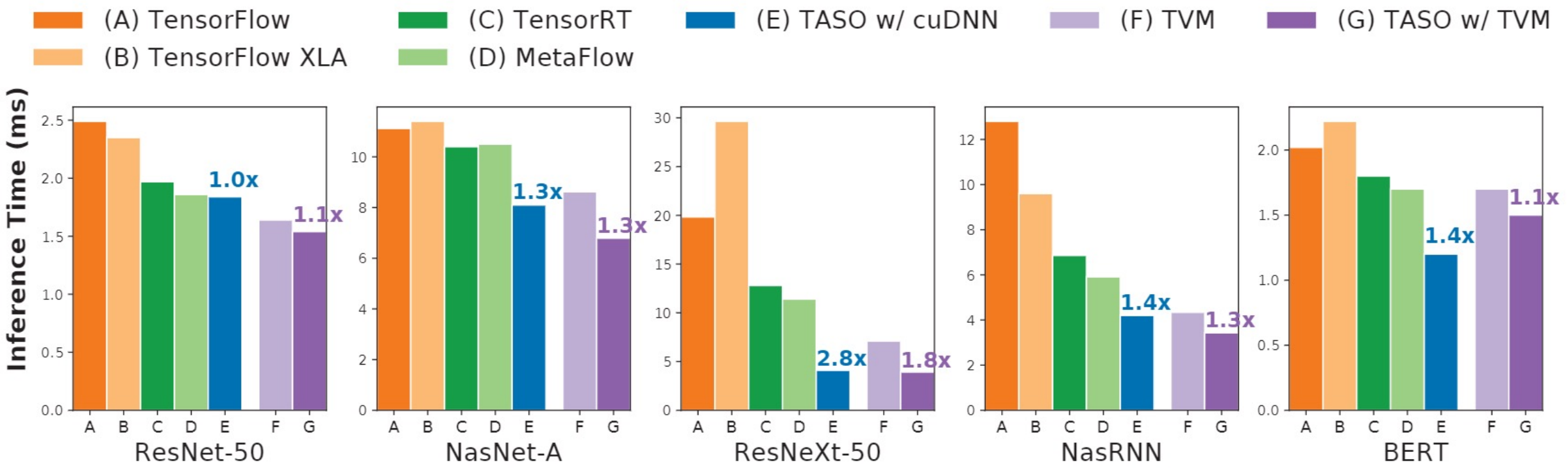
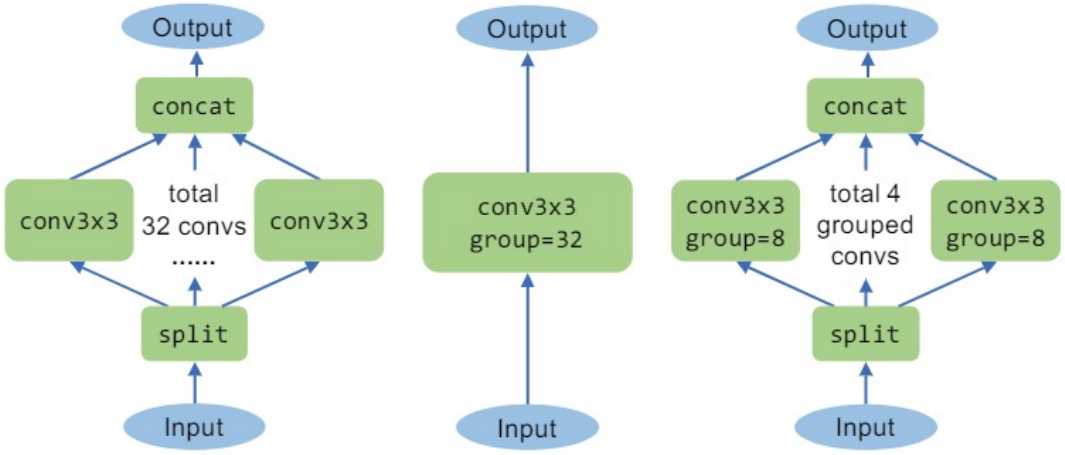
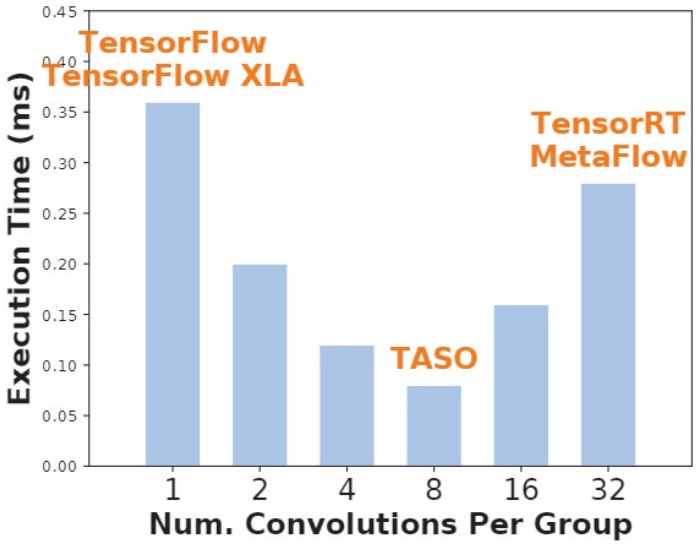


Figure 7. End-to-end inference performance comparison among existing DNN frameworks and TASO. The experiments were performed using a single inference sample, and all numbers were measured by averaging 1,000 runs on a NVIDIA V100 GPU. We evaluated the TASO's performance with both the cuDNN and TVM backends. For each DNN architecture, the numbers above the TASO bars show the speedup over the best existing approach with the same backend.

Heatmap of Used Substitutions(ResNext)



(a) Multi-branch convolution. **(b)** Grouped convolution. **(c)** Multi-branch grouped convolution.



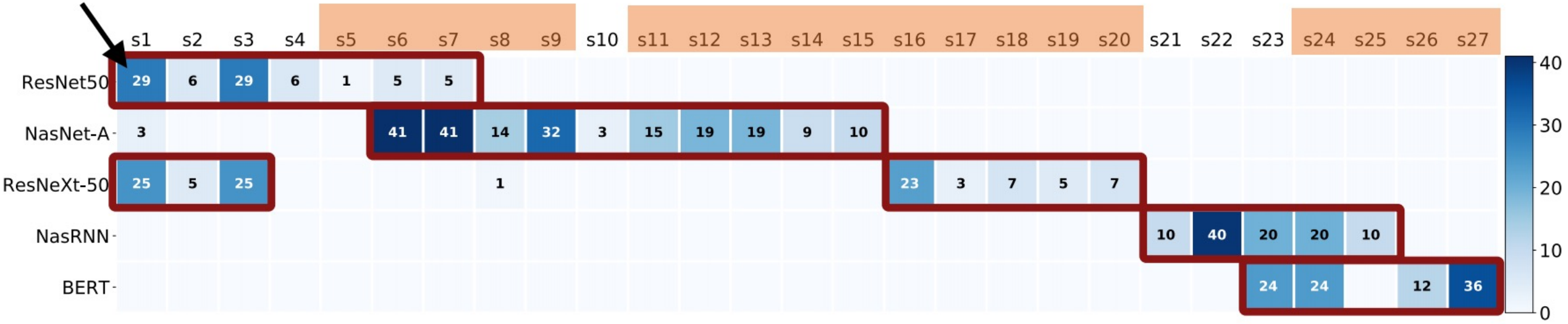
(d) Performance comparison.

It also requires a larger cache to save intermediate states for all convolution in group size 32)

Heatmap of Used Substitutions

How many times a subst.
is used to optimize a DNN

Not covered in TensorFlow



Different DNN models require **different** substitutions.

Ablation study

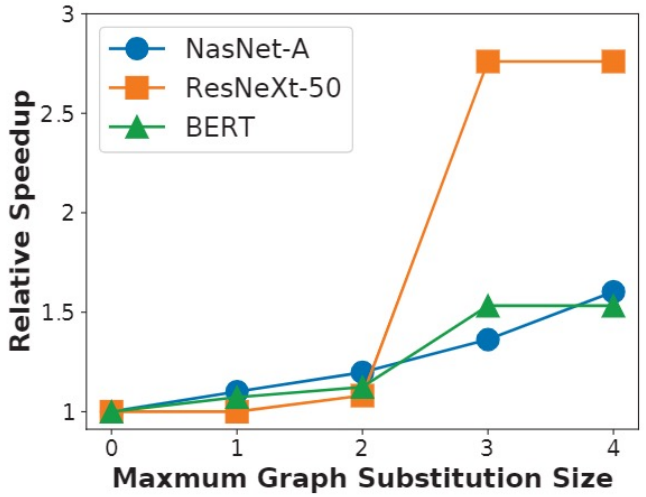


Figure 11. Performance comparison by using graph substitutions with different size limitations. The y-axis shows the relative speedups over the input computation graphs.

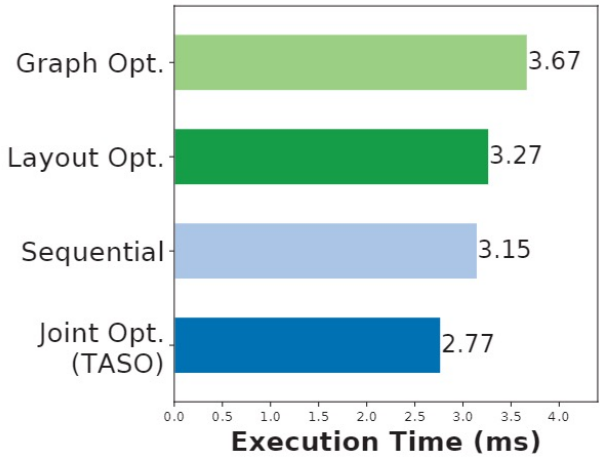


Figure 12. End-to-end inference performance comparison on BERT using different strategies to optimize graph substitution and data layout.