15-442/15-642: Machine Learning Systems

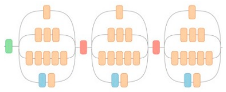
Graph-Level Optimizations

Tianqi Chen and Zhihao Jia

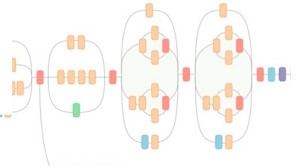
Carnegie Mellon University

2/18/24

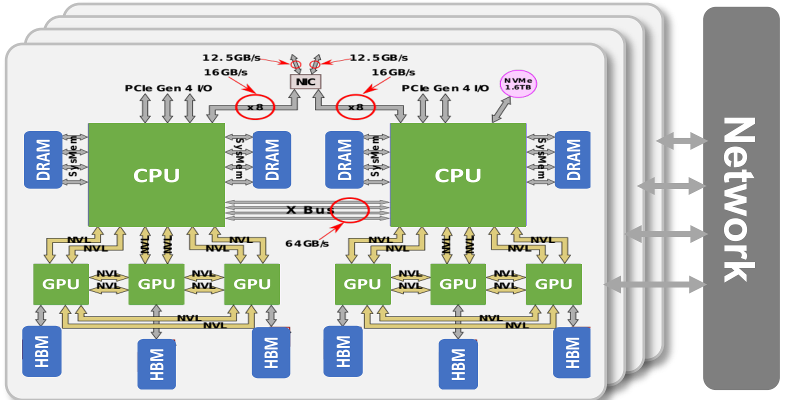
Recap: An Overview of Deep Learning Systems



ML Model

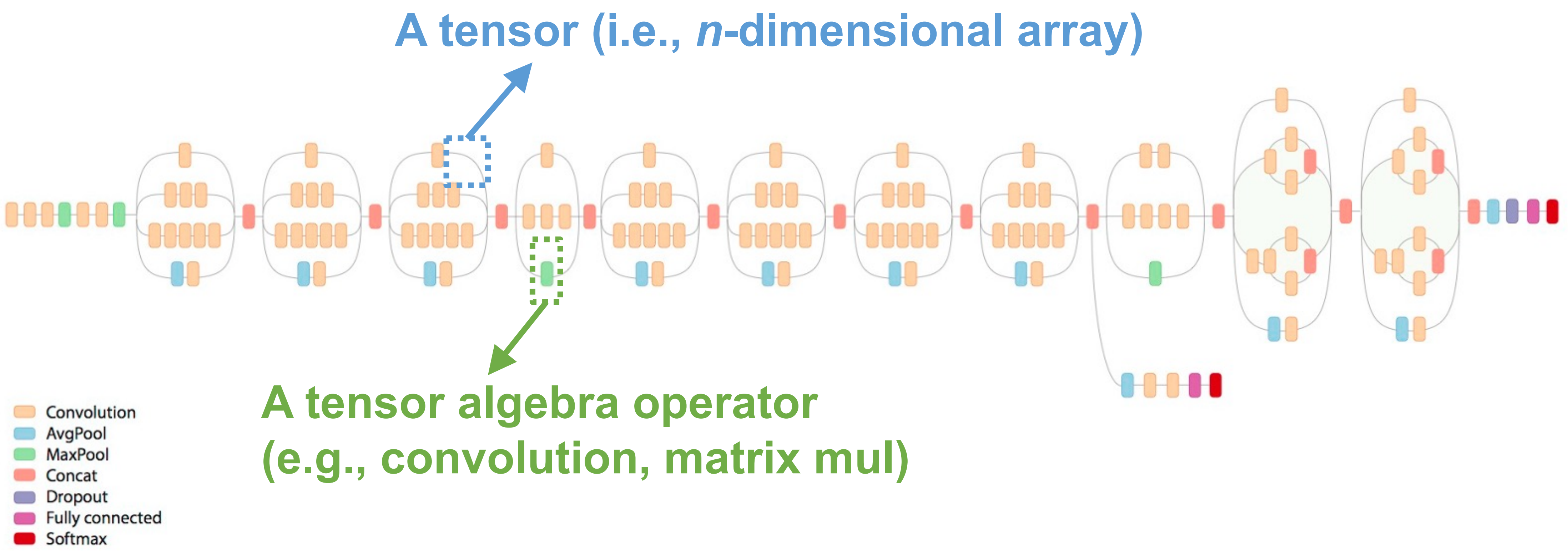


- Automatic Differentiation
- Graph-Level Optimization ← **Lecture 10**
- Parallelization / Distributed Training ← **Lecture 11, 12**
- Kernel Generation
- Memory Optimization ← **Lecture 13**

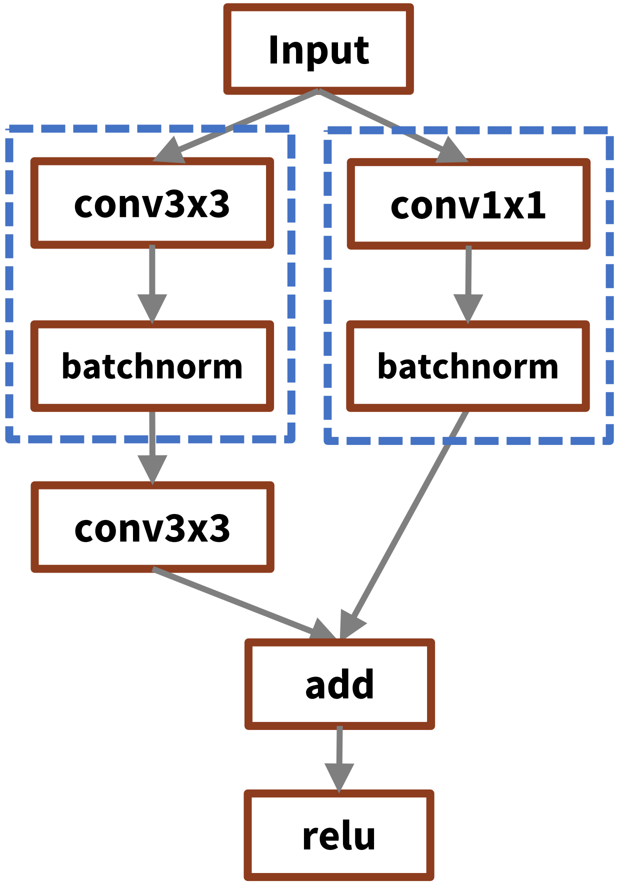


Recap: Deep Neural Network

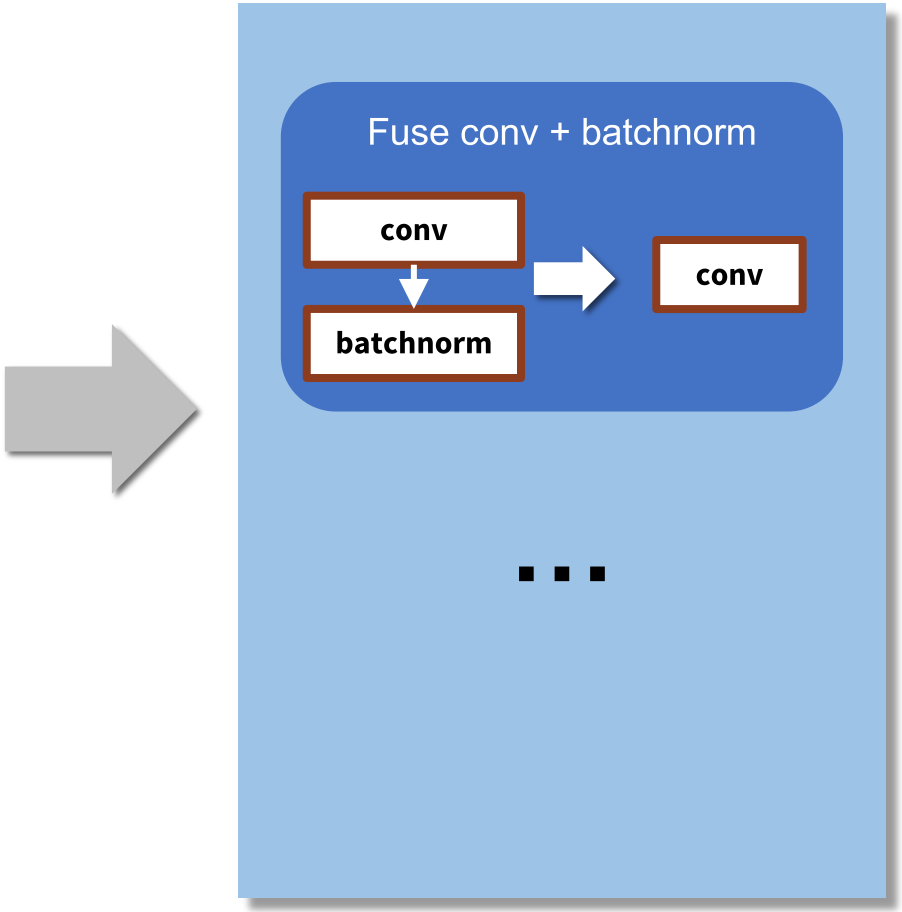
- Collection of simple trainable mathematical units that work together to solve complicated tasks



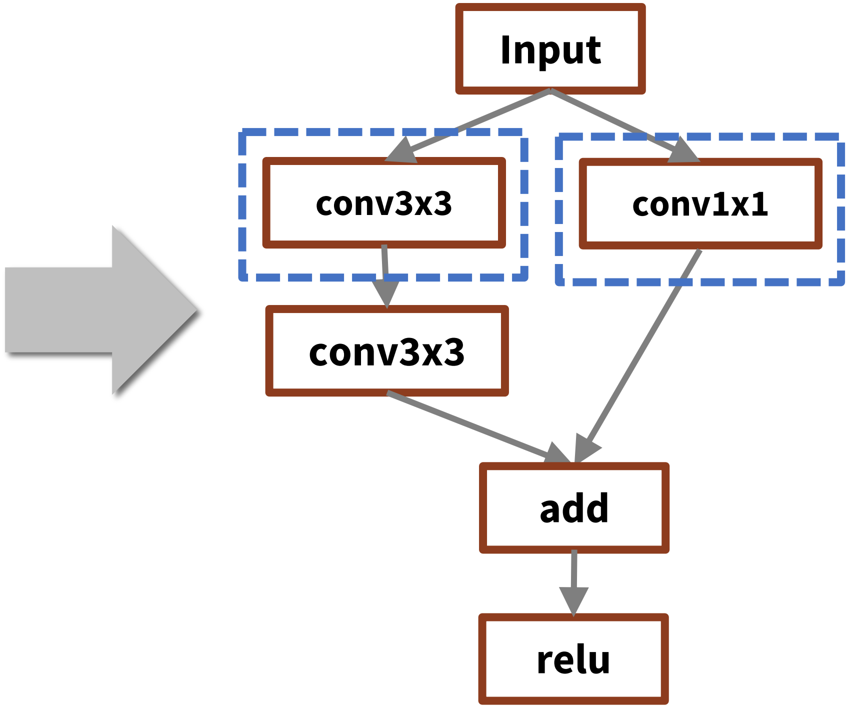
Graph-Level Optimizations



Input Computation Graph

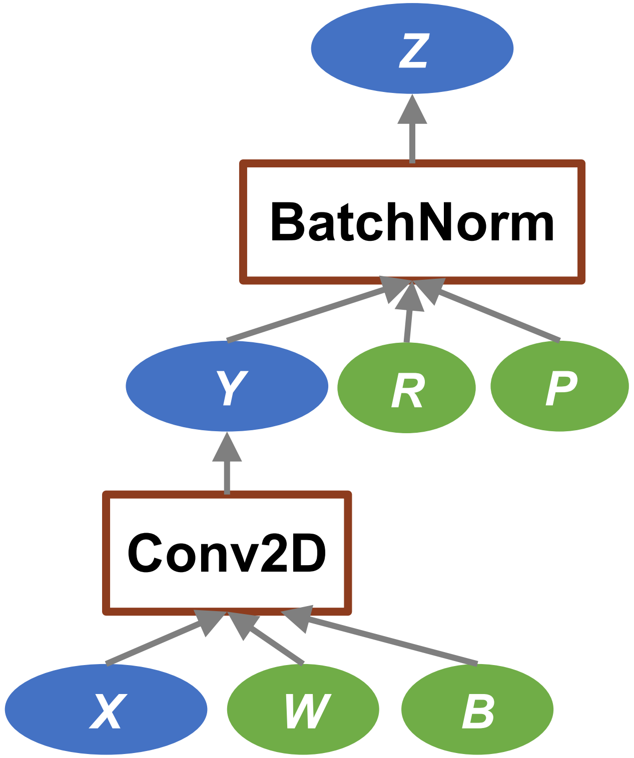


Potential graph transformations



Optimized Computation Graph

Example: Fusing Convolution and Batch Normalization

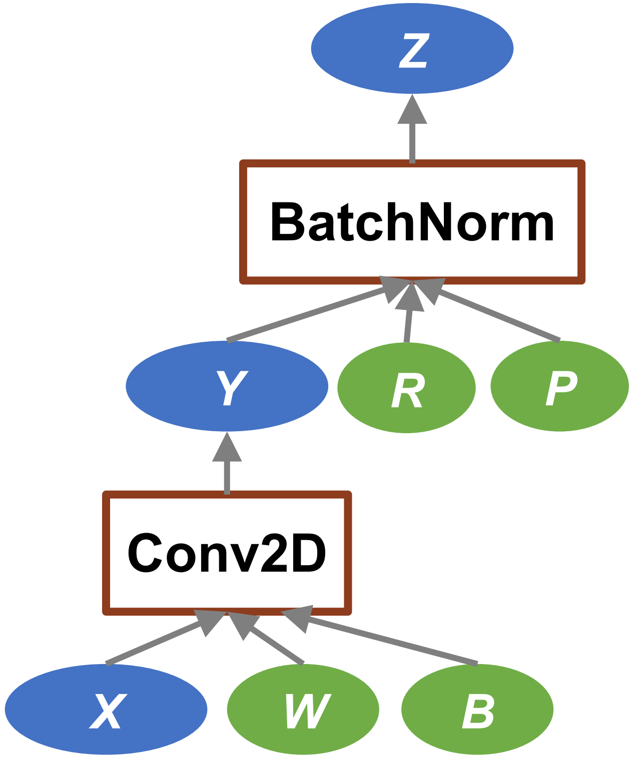


$$Z(n,c,h,w) = Y(n,c,h,w) * R(c) + P(c)$$

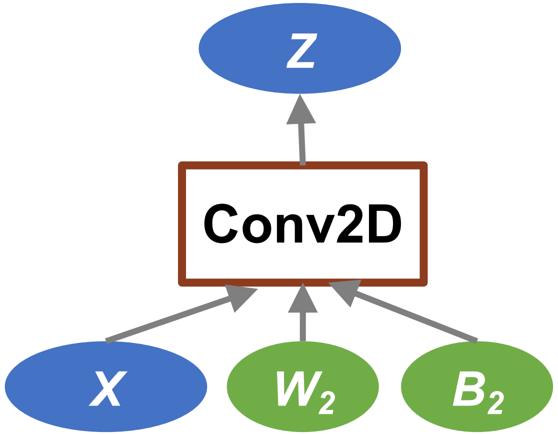
$$Y(n,c,h,w) = \left(\sum_{d,u,v} X(n,d,h+u,w+v) * W(c,d,u,v) \right) + B(n,c,h,w)$$

W, B, R, P are constant pre-trained weights

Fusing Conv and BatchNorm



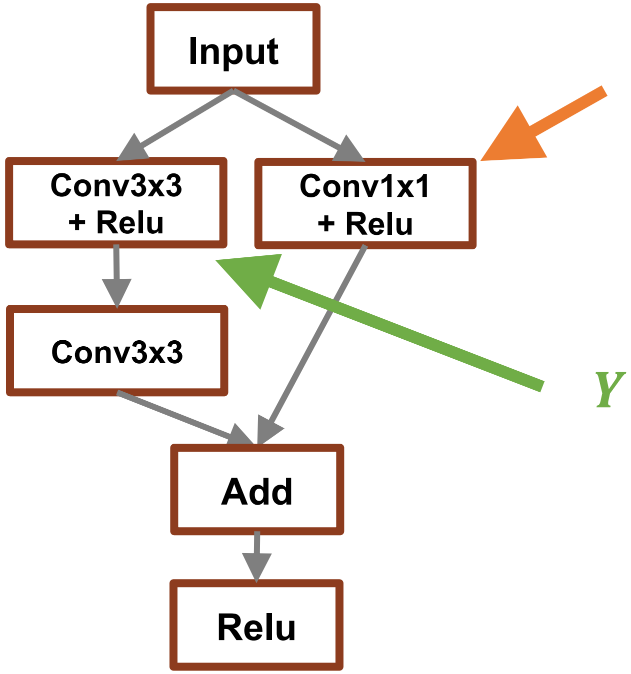
$$Z(n,c,h,w) = \left(\sum_{d,u,v} X(n,d,h+u,w+v) * W_2\,(c,d,u,v)\right) + B_2\,(n,c,h,w)$$



$$W_2(n,c,h,w) = W(n,c,h,w) * R(c)$$

$$B_2(n,c,h,w) = B(n,c,h,w) * R(c) + P(c)$$

Recap: Resnet Example

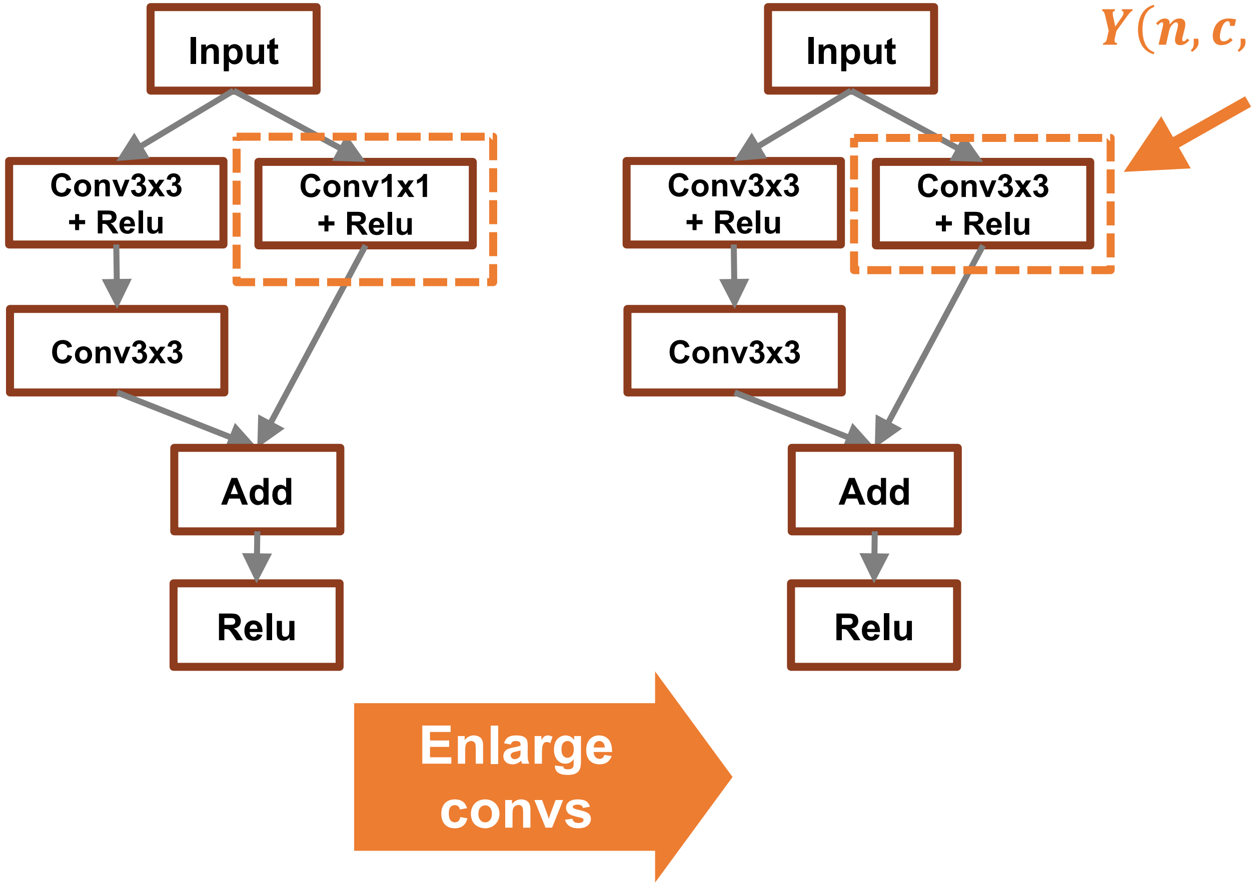


$$Y(n,c,h,w) = \sum_{d}^{D} \sum_{u=1}^{1} \sum_{v=1}^{1} X(n,d,h+u,w+v) * W(c,d,u,v)$$

$$Y(n,c,h,w) = \sum_{d}^{D} \sum_{u=1}^{3} \sum_{v=1}^{3} X(n,d,h+u,w+v) * W(c,d,u,v)$$

* Kaiming He. et al. Deep Residual Learning for Image Recognition, 2015

Recap: Resnet Example

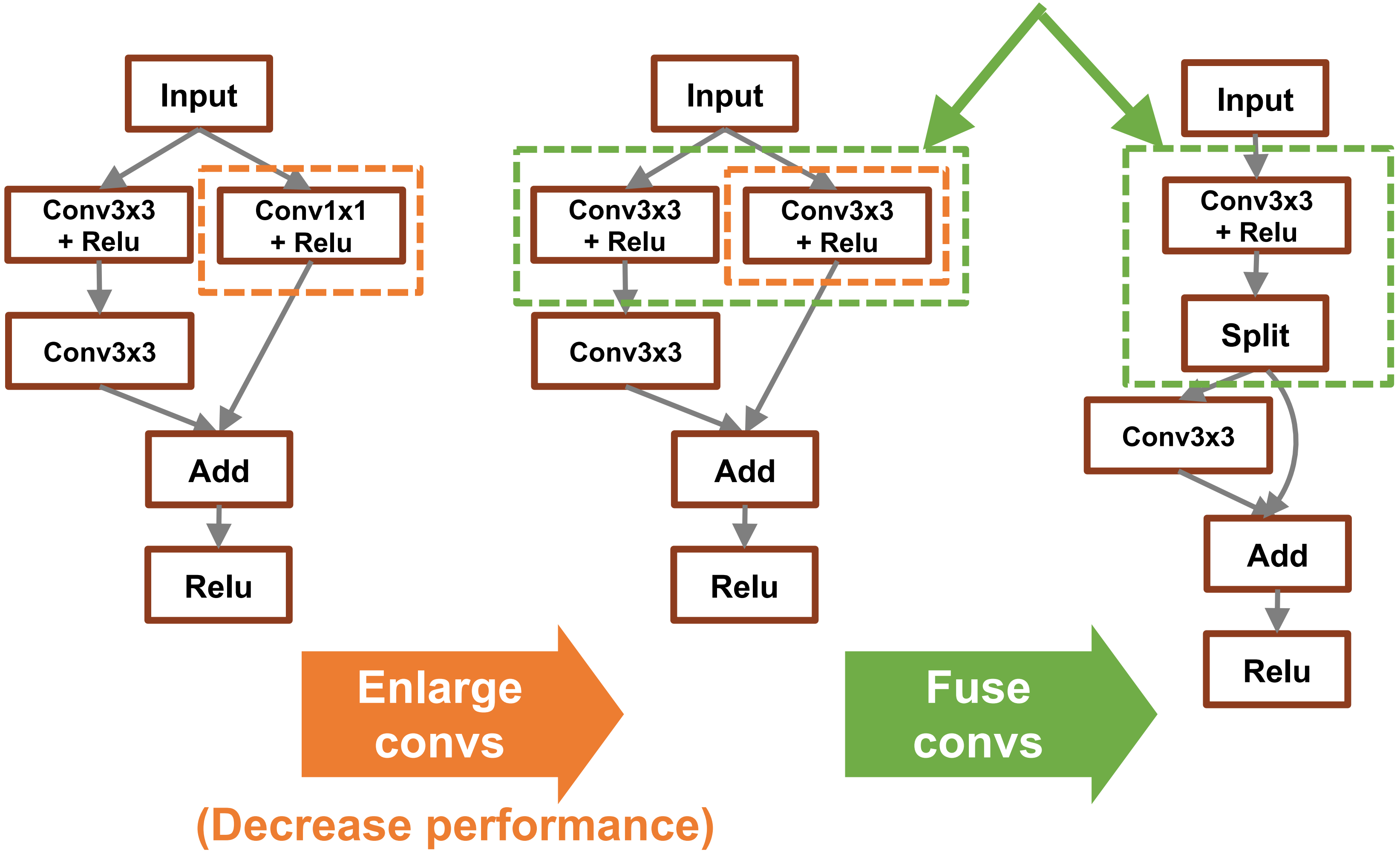


$$Y(n,c,h,w) = \sum_{d} \sum_{u=1}^{3} \sum_{v=1}^{3} X(n,d,h+u,w+v) * W(c,d,u,v)$$

(Decrease performance)

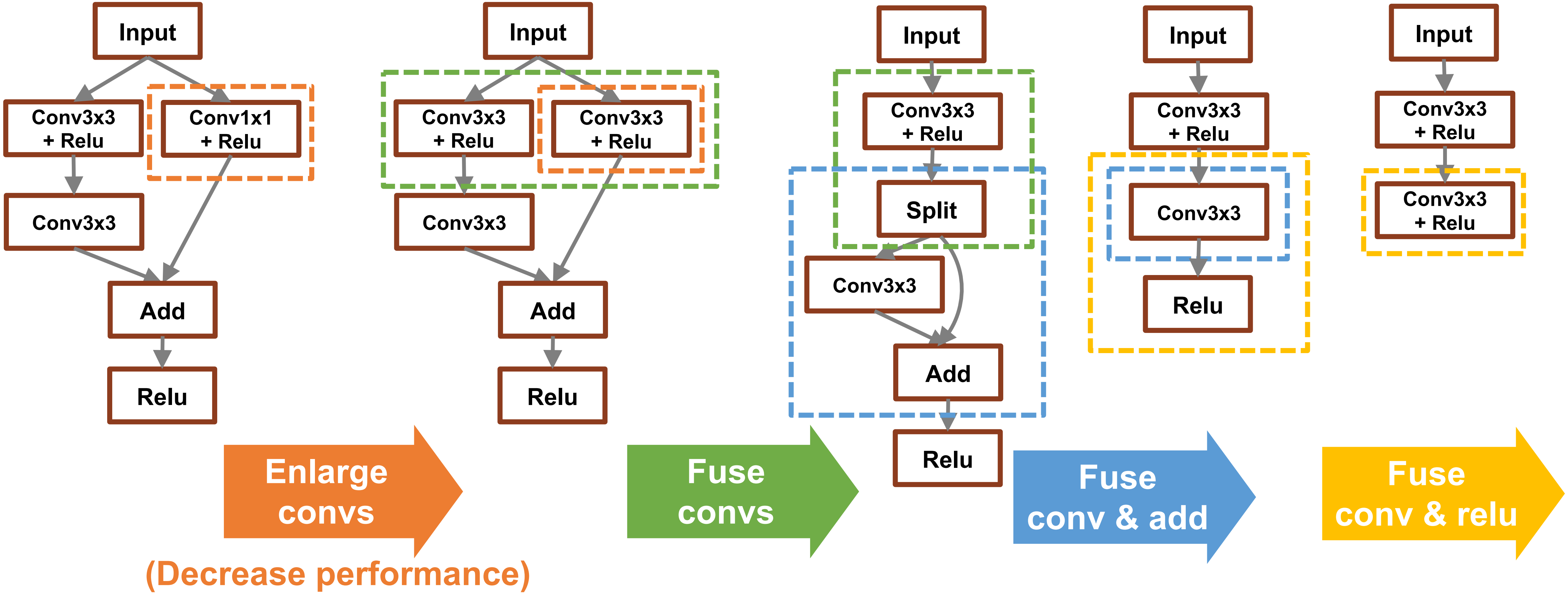
* Kaiming He. et al. Deep Residual Learning for Image Recognition, 2015

$$Y(n,c,h,w) = \sum_{d}^{D} \sum_{u=1}^{3} \sum_{v=1}^{3} X(n,d,h+u,w+v) * W'(c,d,u,v)$$



* Kaiming He. et al. Deep Residual Learning for Image Recognition, 2015

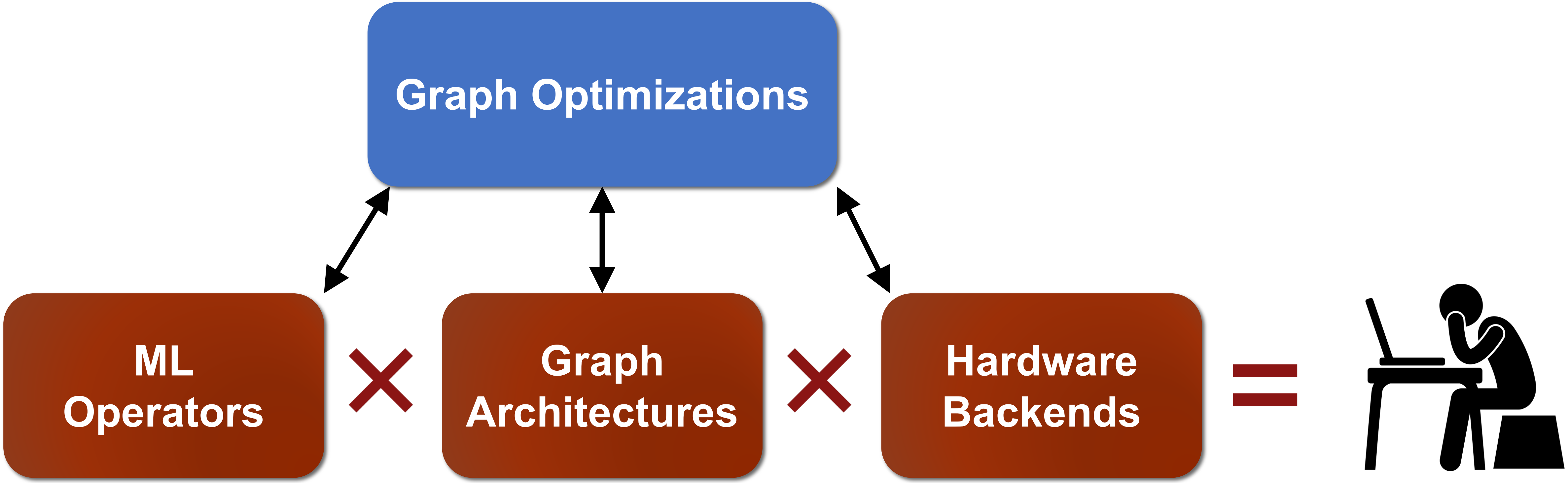
Recap: Resnet Example



The final graph is 30% faster on V100 GPU but 10% slower on K80 GPU.

* Kaiming He. et al. Deep Residual Learning for Image Recognition, 2016

Challenge of Graph Optimizations for ML



Infeasible to manually design graph optimizations for all cases

This Lecture

- TASO: Automatically Generate Graph Transformations
- PET: Discover Partially-Equivalent Graph Transformations

TASO: Optimizing Deep Learning with Automatic Generation of Graph Substitutions

TASO: Optimizing Deep Learning Computation with Automated Generation of Graph Substitutions. SOSP'19.

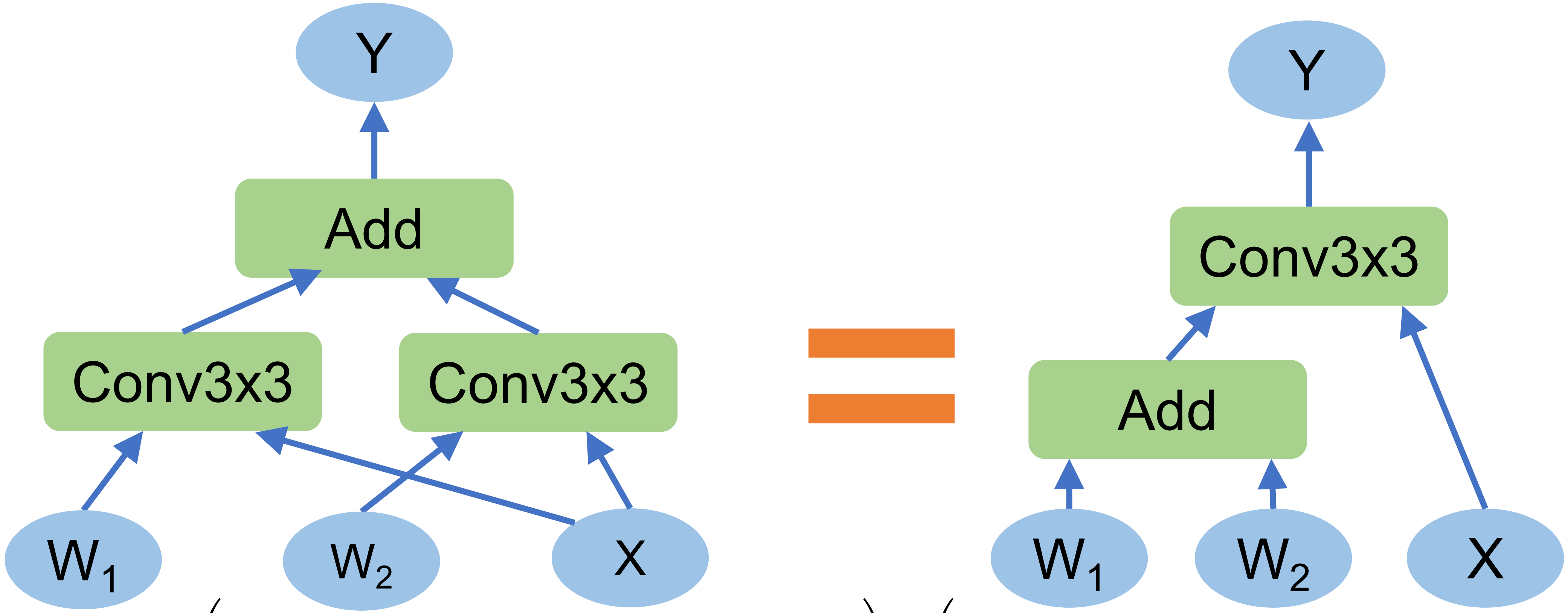
TASO: Tensor Algebra SuperOptimizer

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

- **Less engineering effort:** 53,000 LOC for manual graph optimizations in TensorFlow → 1,400 LOC in TASO
- **Better performance:** outperform existing optimizers by up to 3x
- **Stronger correctness:** formally verify all generated substitutions

TASO: Optimizing Deep Learning Computation with Automated Generation of Graph Substitutions. SOSP'19.

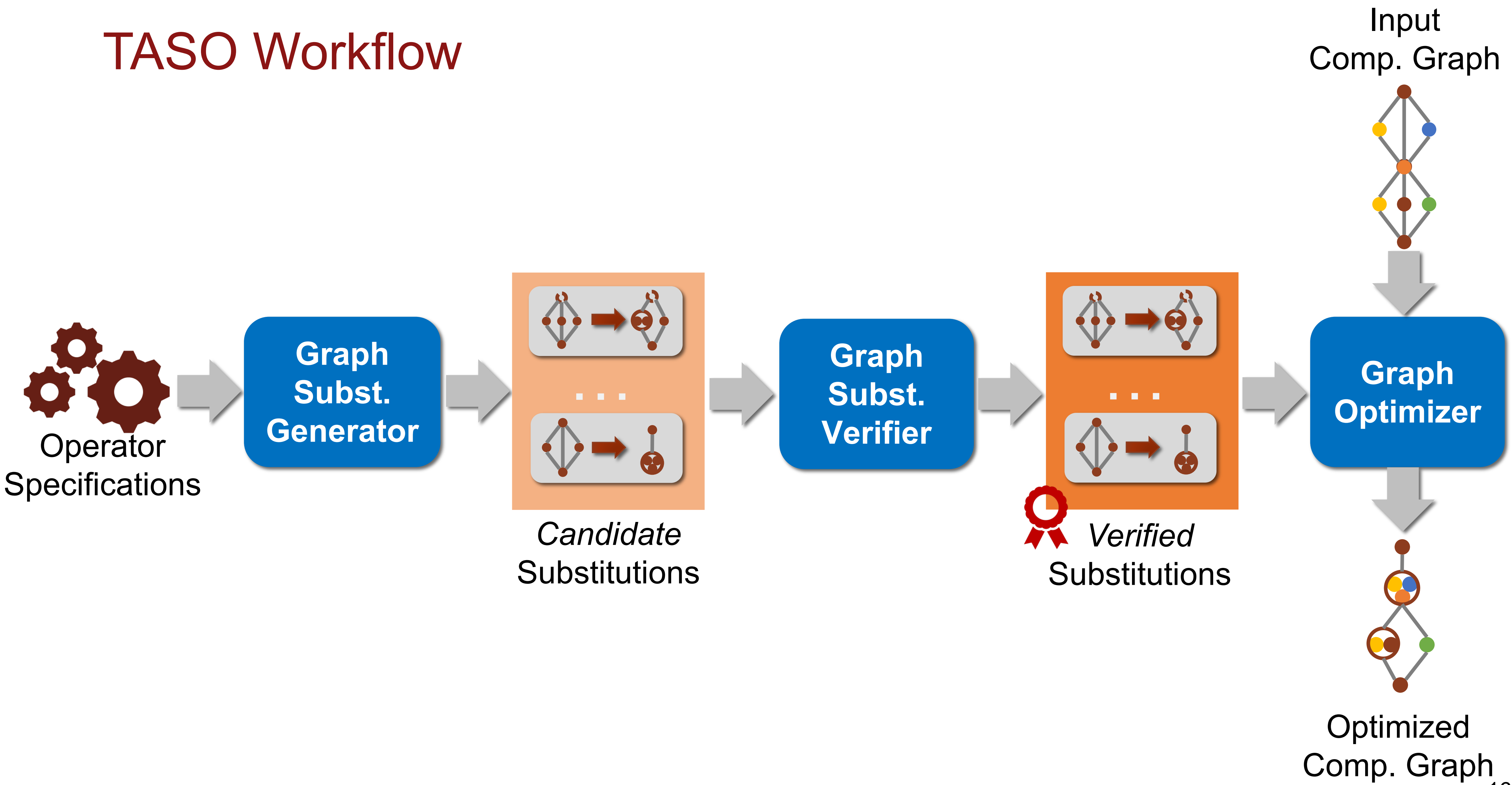
Graph Substitution

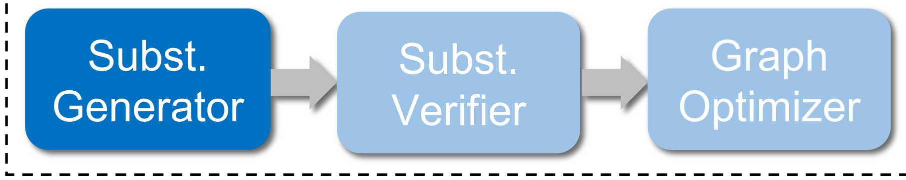


$$Y(n,c,h,w) = \left(\sum_{d,u,v} X(n,d,h+u,w+v) * W1(c,d,u,v)\right) + \left(\sum_{d,u,v} X(n,d,h+u,w+v) * W2(c,d,u,v)\right)$$

$$\Leftrightarrow Y(n,c,h,w) = \sum_{d,u,v} X(n,d,h+u,w+v) * \left((W_1\ (c,d,u,v) + W_2\ (c,d,u,v)\right)$$

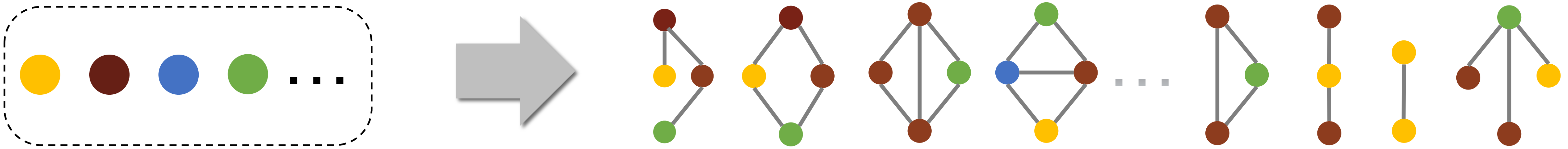
TASO Workflow



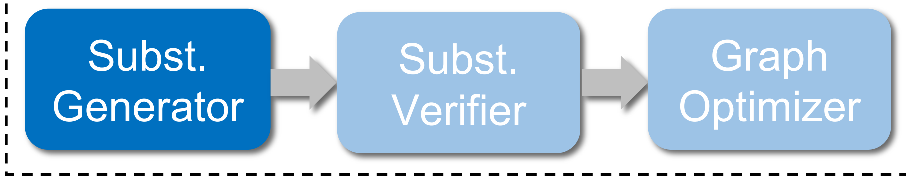


Graph Substitution Generator

Enumerate **<u>all possible</u>** graphs up to a fixed size using available operators

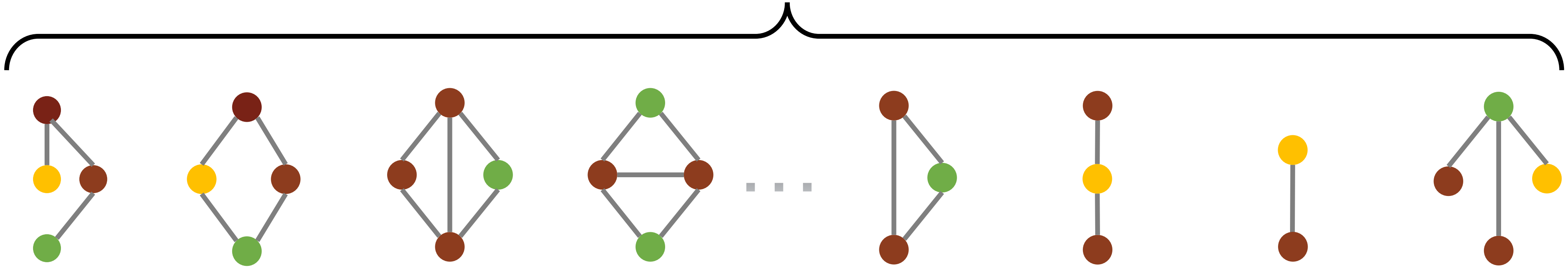


Operators supported by hardware backend



Graph Substitution Generator

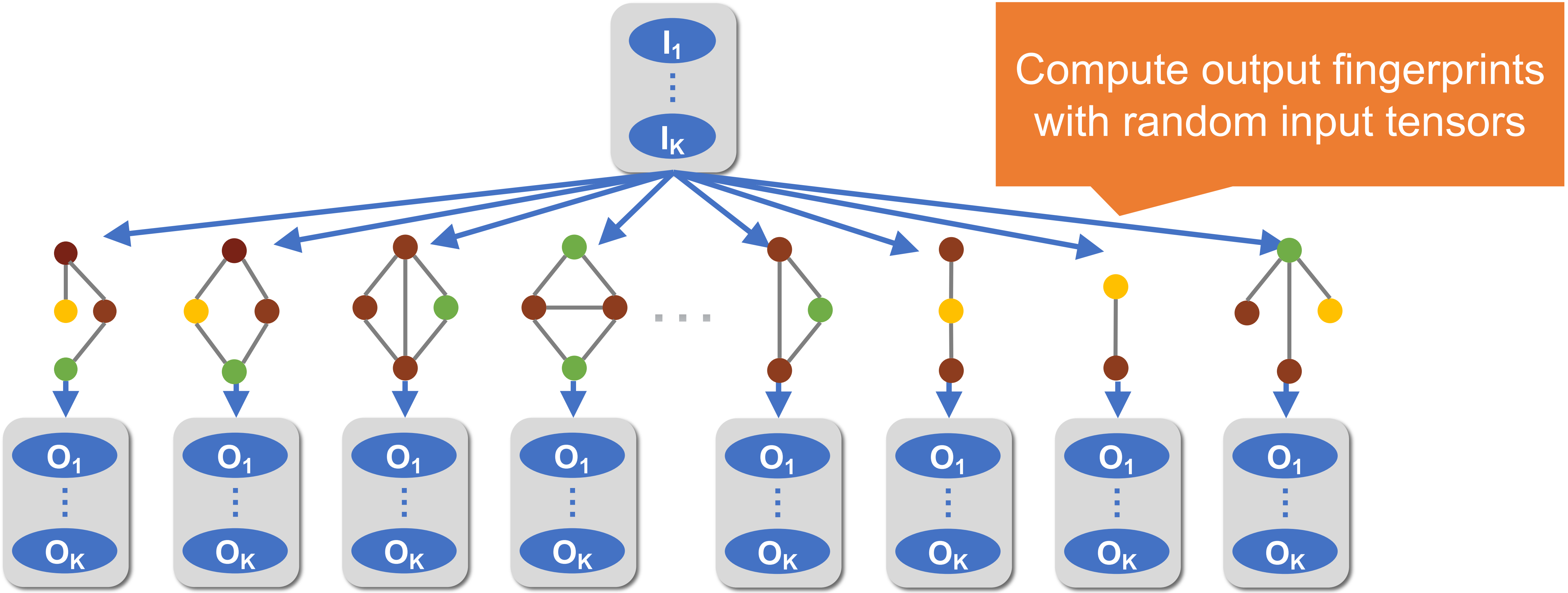
66M graphs with up to **4** operators

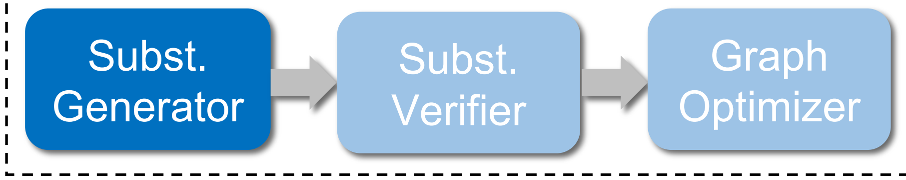


A substitution = a pair of equivalent graphs

Explicitly considering all pairs does not scale

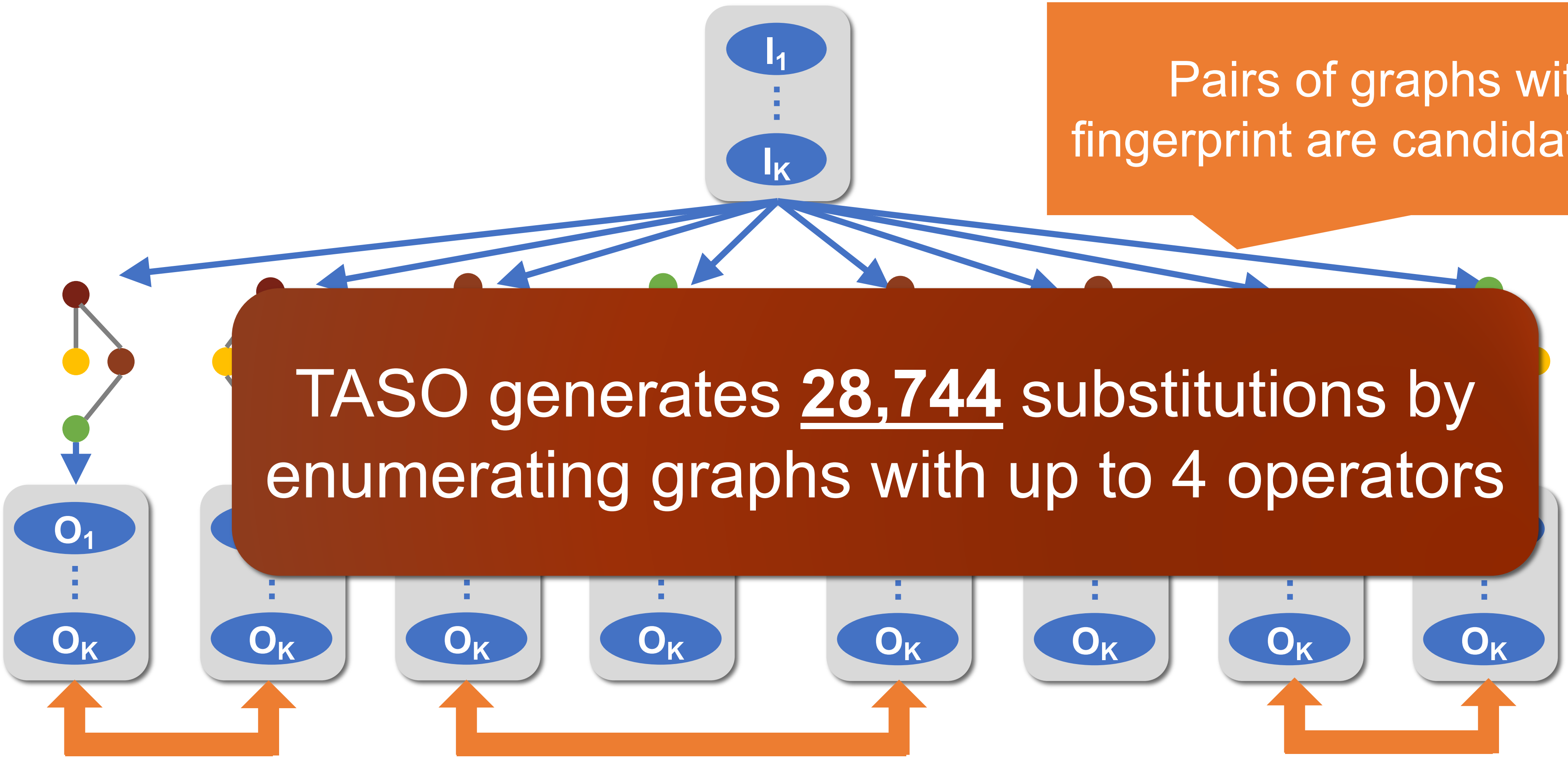
Graph Substitution Generator



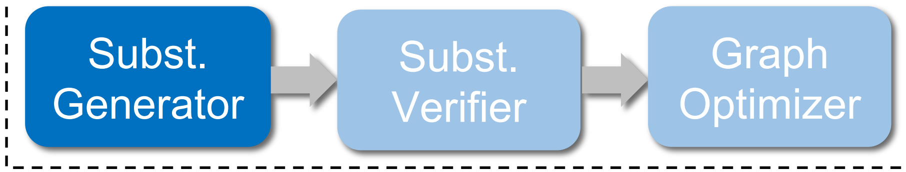


Graph Substitution Generator

Pairs of graphs with identical fingerprint are candidate substitutions



TASO generates **28,744** substitutions by enumerating graphs with up to 4 operators

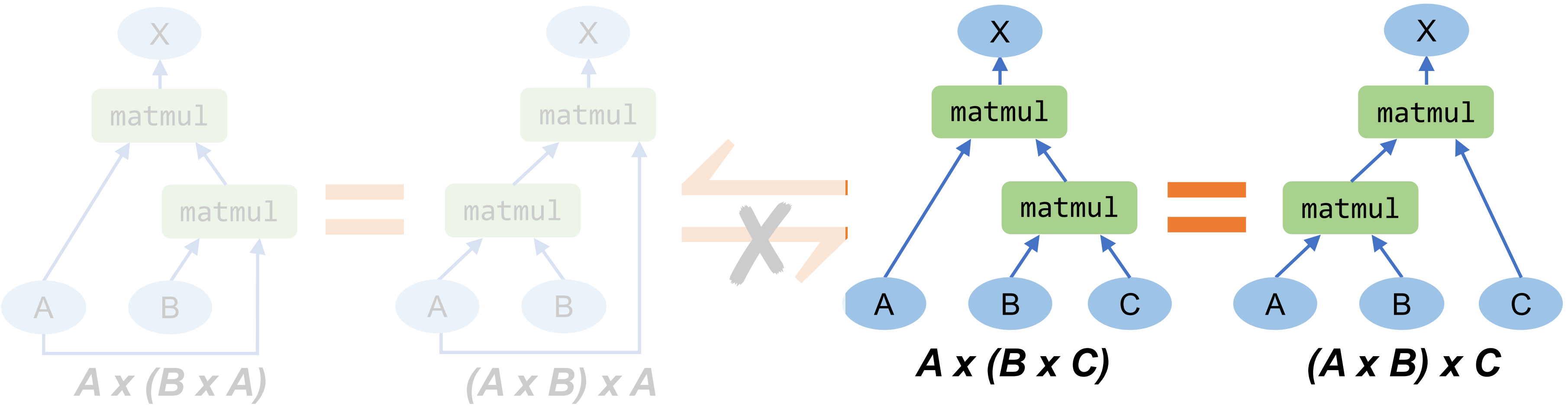


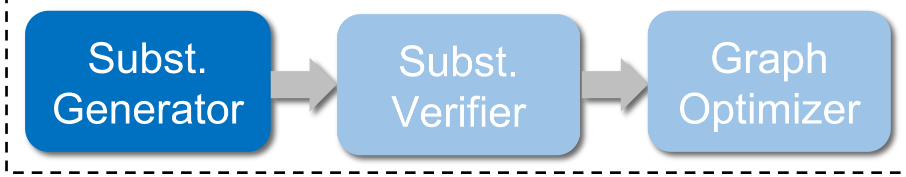
Pruning Redundant Substitutions

28,744 substitutions

Input Tensor Renaming

17,346 substitutions





Pruning Redundant Substitutions

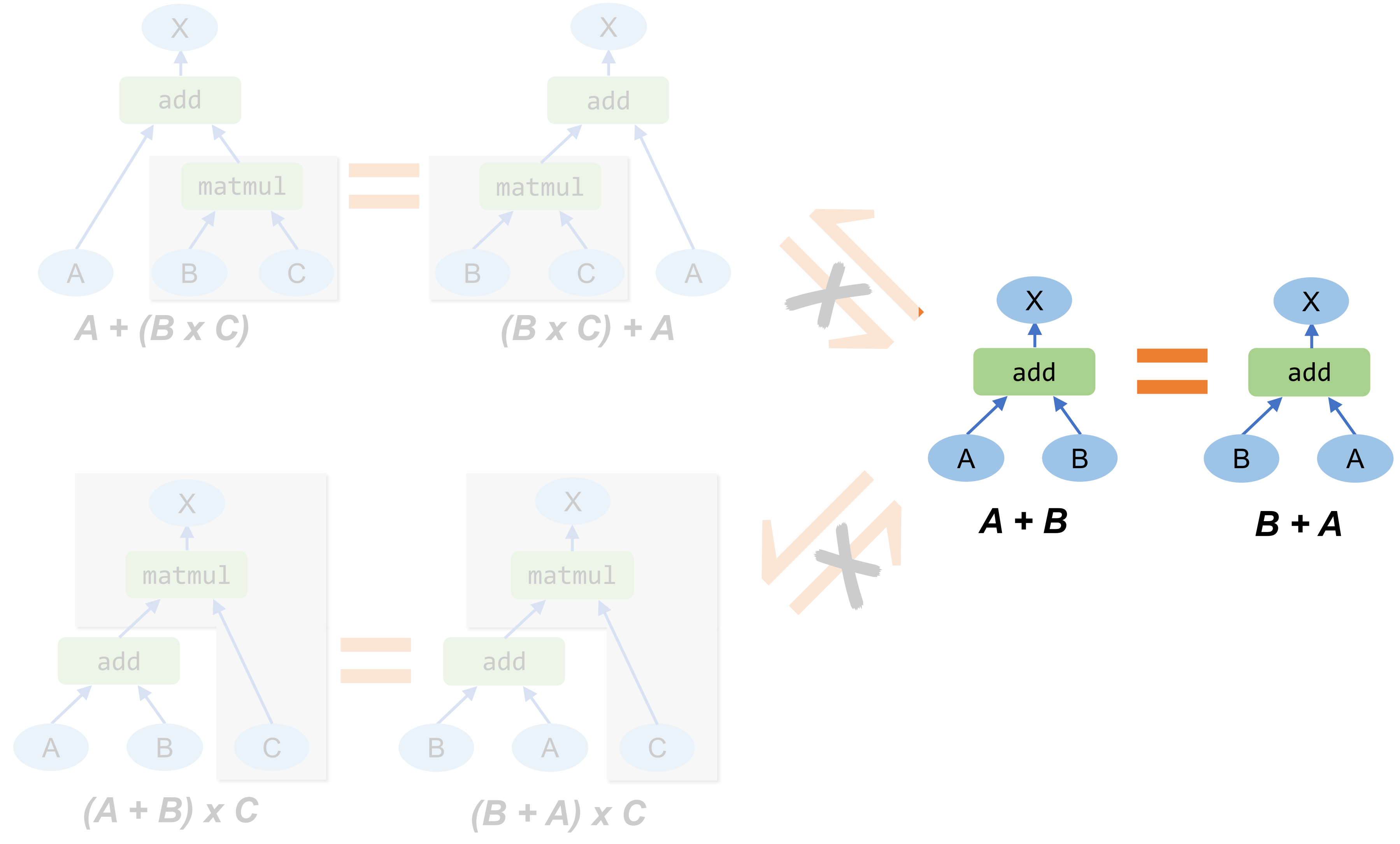
28,744 substitutions

↓ Input Tensor Renaming ↓

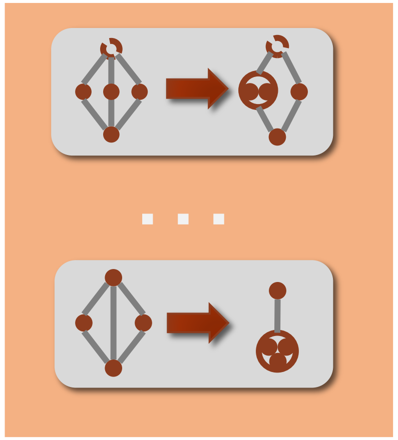
17,346 substitutions

↓ Common Subgraphs ↓

743 substitutions



Graph Substitution Verifier

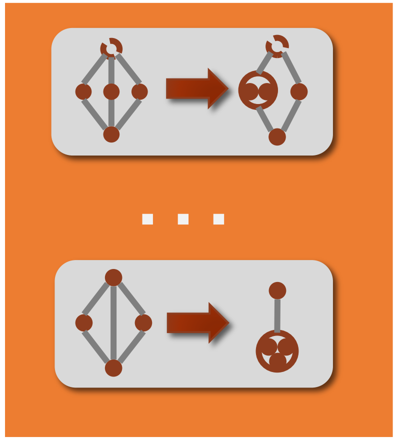


Candidate Substitutions

Graph Subst. Verifier

P1. conv is distributive over concatenation
P2. conv is bilinear
…
Pn.

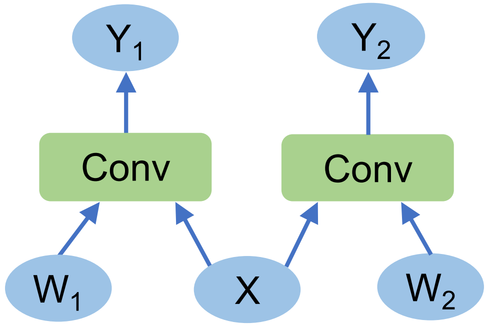
Operator Specifications



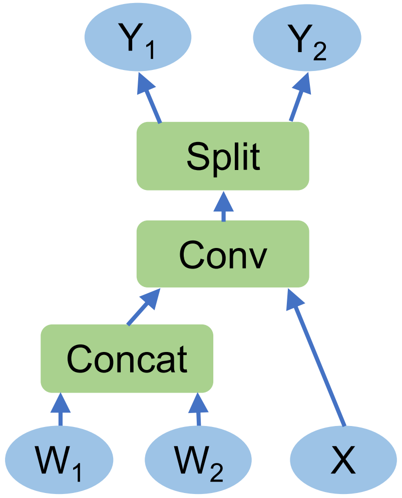
Verified Substitutions

$$\forall x, w_1, w_2 .$$
$$Conv(x, Concat(w_1, w_2)) = Concat(Conv(x, w_1), Conv(x, w_2))$$

Verification Workflow



(Conv(x, w_1), Conv (x, w_2))



Split(Conv(x, Concat(w_1, w_2)))

$$\forall x, w_1, w_2 .\ \big(Conv(x, w_1), Conv(x, w_2)\big) = Split\Big(Conv\big(x, Concat(w_1, w_2)\big)\Big)$$

Automated Theorem Prover

P1. $\forall x, w_1, w_2 .$ $Conv\big(x, Concat(w_1, w_2)\big) = Concat\big(Conv(x, w_1), Conv(x, w_2)\big)$

P2. …

Operator Specifications

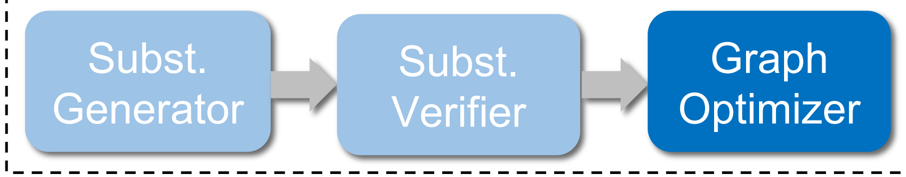
Verification Effort

TASO generates all **743** substitutions in 5 minutes, and verifies them against **43** operator properties in 10 minutes

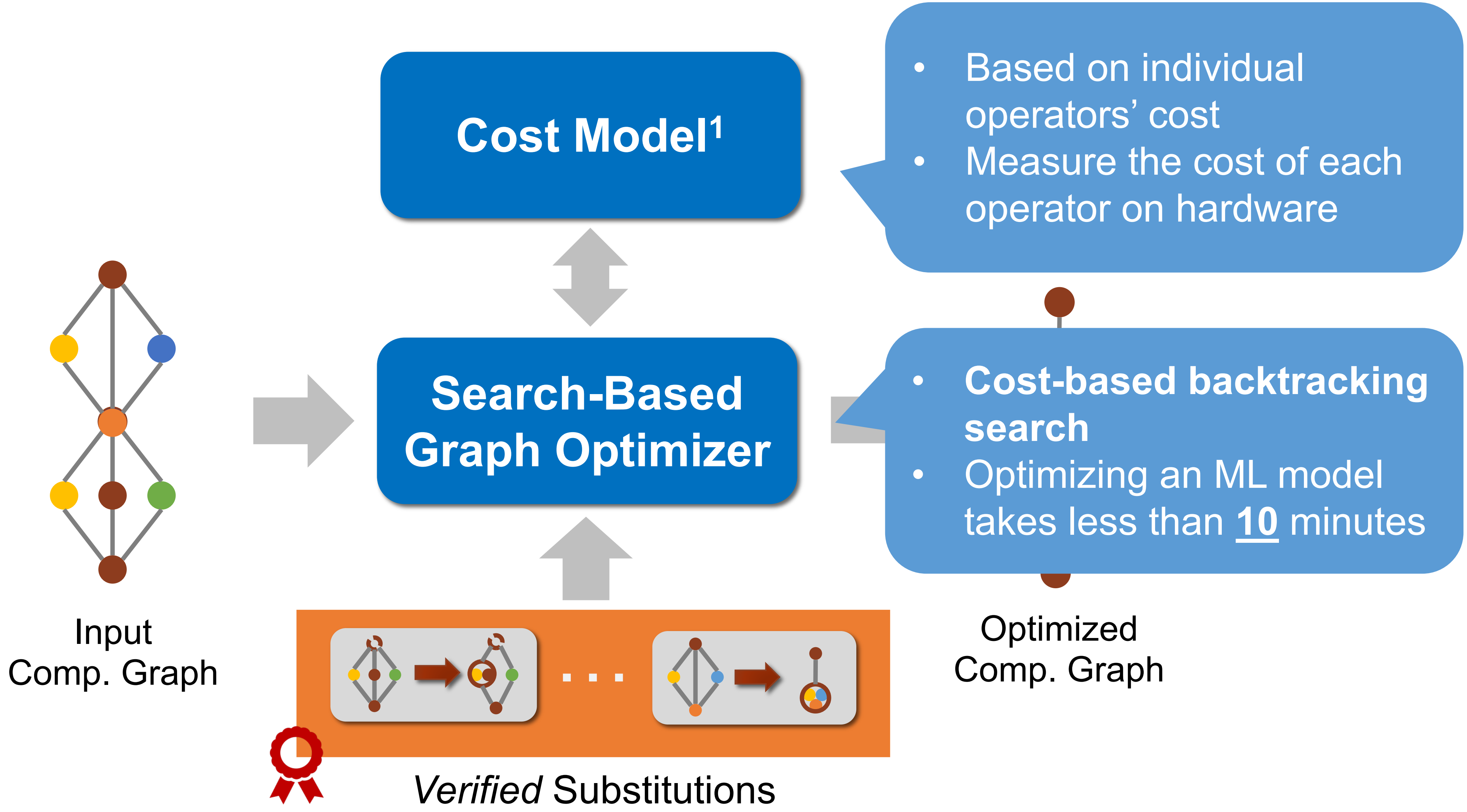
Supporting a new operator requires **a few hours** of human effort to specify its properties

Operator specifications in TASO ≈ **1,400** LOC
Manual graph optimizations in TensorFlow ≈ **53,000** LOC

Operator Property	Comment
$\forall x, y, z.\ \mathtt{ewadd}(x, \mathtt{ewadd}(y, z)) = \mathtt{ewadd}(\mathtt{ewadd}(x, y), z)$	ewadd is associative
$\forall x, y.\ \mathtt{ewadd}(x, y) = \mathtt{ewadd}(y, x)$	ewadd is commutative
$\forall x, y, z.\ \mathtt{ewmul}(x, \mathtt{ewmul}(y, z)) = \mathtt{ewmul}(\mathtt{ewmul}(x, y), z)$	ewmul is associative
$\forall x, y.\ \mathtt{ewmul}(x, y) = \mathtt{ewmul}(y, x)$	ewmul is commutative
$\forall x, y, z.\ \mathtt{ewmul}(\mathtt{ewadd}(x, y), z) = \mathtt{ewadd}(\mathtt{ewmul}(x, z), \mathtt{ewmul}(y, z))$	distributivity
$\forall x, y, w.\ \mathtt{smul}(\mathtt{smul}(x, y), w) = \mathtt{smul}(x, \mathtt{smul}(y, w))$	smul is associative
$\forall x, y, w.\ \mathtt{smul}(\mathtt{ewadd}(x, y), w) = \mathtt{ewadd}(\mathtt{smul}(x, w), \mathtt{smul}(y, w))$	distributivity
	ommutativity
	is its own inverse
	ommutativity
	ommutativity
	ommutativity
	ssociative
	inear
	inear
	d transpose
	linear
$\forall s, p, x, y, w.\ \mathtt{smul}(\mathtt{conv}(s, p, \mathsf{A}_{\mathsf{none}}, x, y), w) = \mathtt{conv}(s, p, \mathsf{A}_{\mathsf{none}}, \mathtt{smul}(x, w), y)$	conv is bilinear
$\forall s, p, x, y, z.\ \mathtt{conv}(s, p, \mathsf{A}_{\mathsf{none}}, x, \mathtt{ewadd}(y, z)) = \mathtt{ewadd}(\mathtt{conv}(s, p, \mathsf{A}_{\mathsf{none}}, x, y), \mathtt{conv}(s, p, \mathsf{A}_{\mathsf{none}}, x, z))$	conv is bilinear
	linear
	nvolution kernel
	$\mathsf{A}_{\mathsf{relu}}$ applies relu
	ommutativity
	conv. with $\mathsf{C}_{\mathsf{pool}}$
	rnel
	atrix
	entity
$\forall a, x, y.\ \mathtt{split}_0(a, \mathtt{concat}(a, x, y)) = x$	split definition
	ition
	f concatenation
	ommutativity
	ommutativity
	ommutativity
	ommutativity
	ion and transpose
	ion and matrix mul.
	ion and matrix mul.
	tion and conv.
	concatenation and conv.
$\forall s, p, x, y, z, w.\ \mathtt{conv}(s, p, \mathsf{A}_{\mathsf{none}}, \mathtt{concat}(1, x, z), \mathtt{concat}(1, y, w)) = \mathtt{ewadd}(\mathtt{conv}(s, p, \mathsf{A}_{\mathsf{none}}, x, y), \mathtt{conv}(s, p, \mathsf{A}_{\mathsf{none}}, z, w))$	concatenation and conv.
$\forall k, s, p, x, y.\ \mathtt{concat}(1, \mathtt{pool}_{\mathsf{avg}}(k, s, p, x), \mathtt{pool}_{\mathsf{avg}}(k, s, p, y)) = \mathtt{pool}_{\mathsf{avg}}(k, s, p, \mathtt{concat}(1, x, y))$	concatenation and pooling
$\forall k, s, p, x, y.\ \mathtt{concat}(0, \mathtt{pool}_{\mathsf{max}}(k, s, p, x), \mathtt{pool}_{\mathsf{max}}(k, s, p, y)) = \mathtt{pool}_{\mathsf{max}}(k, s, p, \mathtt{concat}(0, x, y))$	concatenation and pooling
$\forall k, s, p, x, y.\ \mathtt{concat}(1, \mathtt{pool}_{\mathsf{max}}(k, s, p, x), \mathtt{pool}_{\mathsf{max}}(k, s, p, y)) = \mathtt{pool}_{\mathsf{max}}(k, s, p, \mathtt{concat}(1, x, y))$	concatenation and pooling

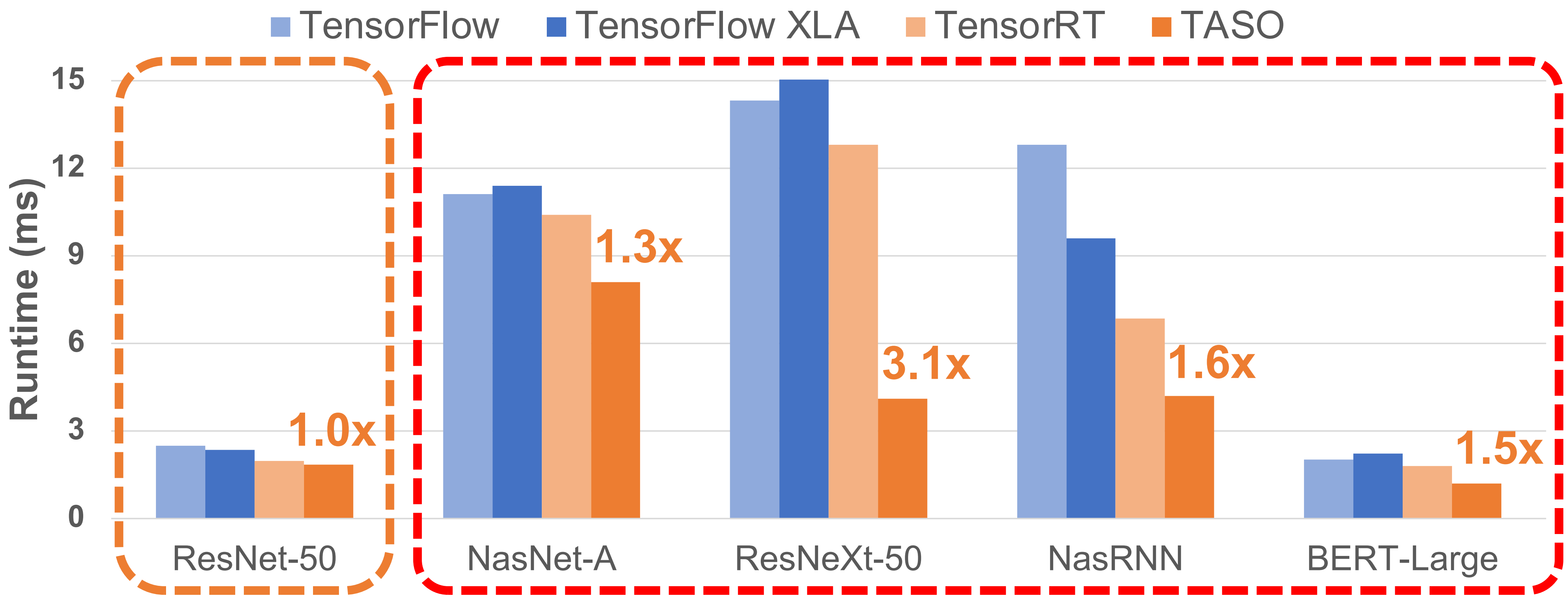


Search-Based Graph Optimizer



1. Exploring Hidden Dimensions in Parallelizing Convolutional Neural Networks. ICML'18.

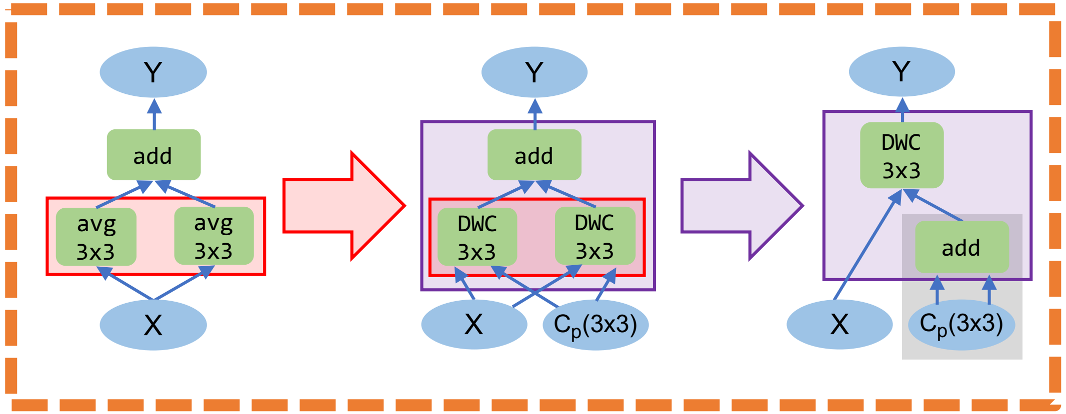
End-to-end Inference Performance (Nvidia V100 GPU)

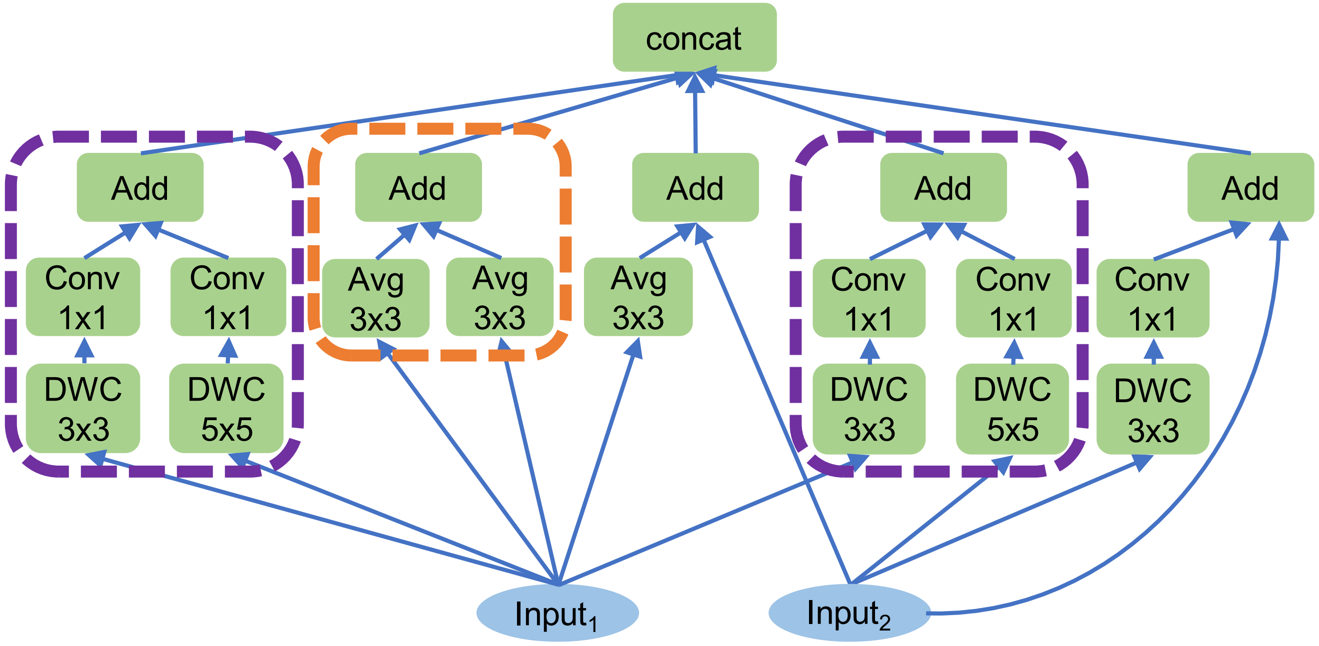


Competitive on standard models

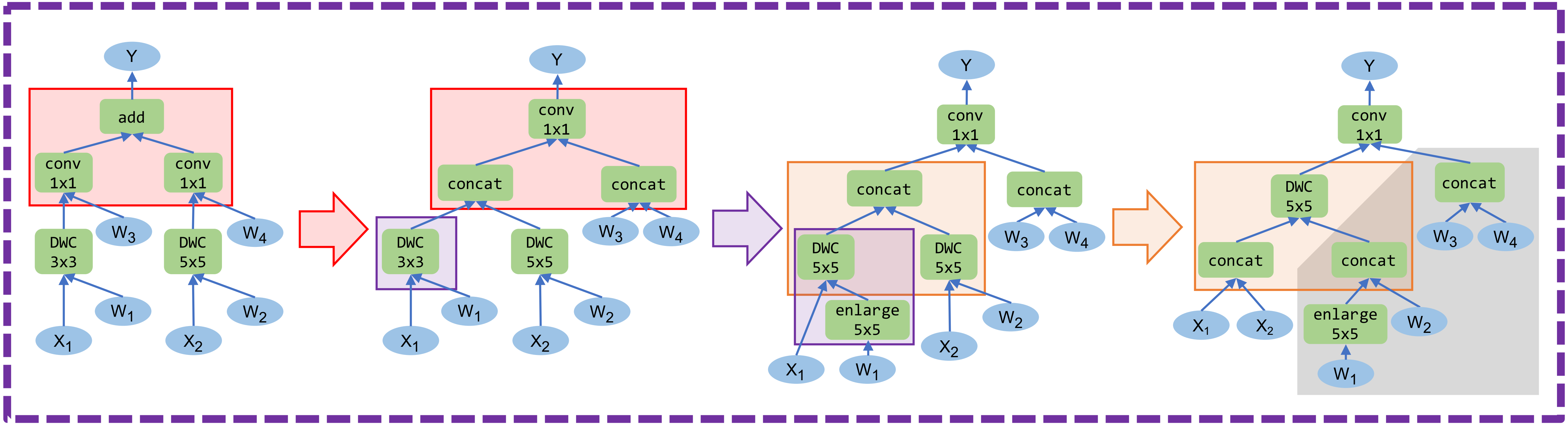
Larger speedups on emerging models

Case Study: NASNet





*DWC: depth-wise convolution



Why TASO is a SuperOptimizer?

What is the difference between optimizer and super-optimizer?

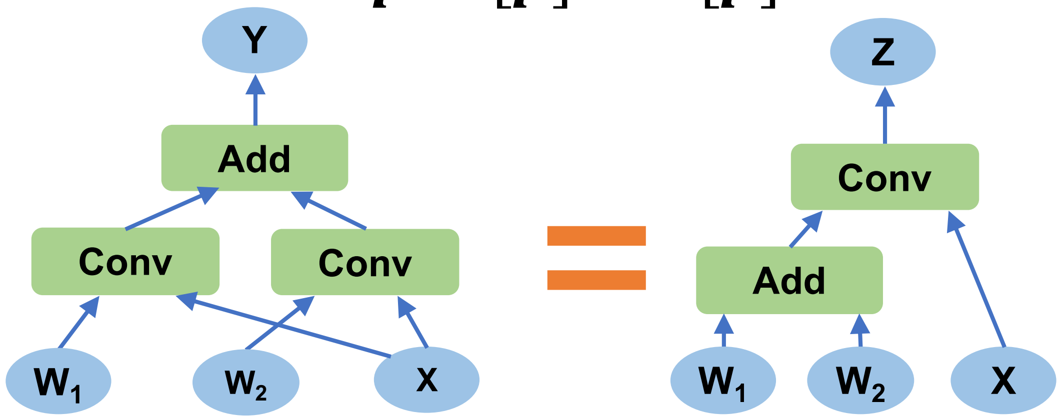
Goal: gradually *improve* an input program by greedily applying optimizations

Goal: automatically find an *optimal* program for an input program

PET:
Optimizing Tensor Programs with Partially Equivalent Transformations and Automated Corrections

Motivation: Fully v.s. Partially Equivalent Transformations

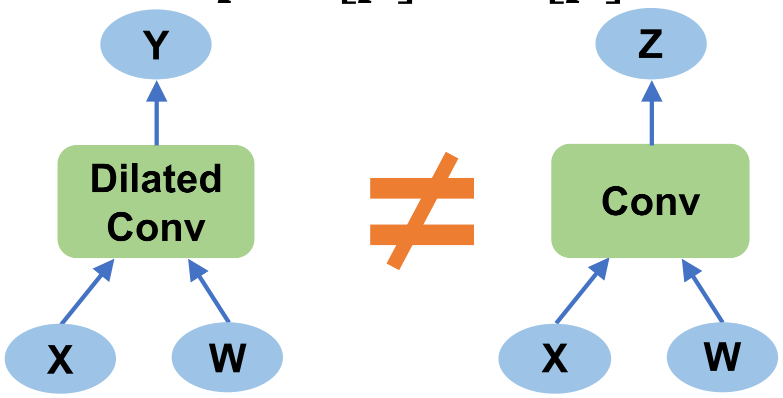
$$\forall \boldsymbol{p}.\ \boldsymbol{Y}[\boldsymbol{p}] = \boldsymbol{Z}[\boldsymbol{p}]$$



Fully Equivalent Transformations

- Pro: preserve functionality
- Con: miss optimization opportunities

$$\exists \boldsymbol{p}.\ \boldsymbol{Y}[\boldsymbol{p}] \neq \boldsymbol{Z}[\boldsymbol{p}]$$



Partially Equivalent Transformations

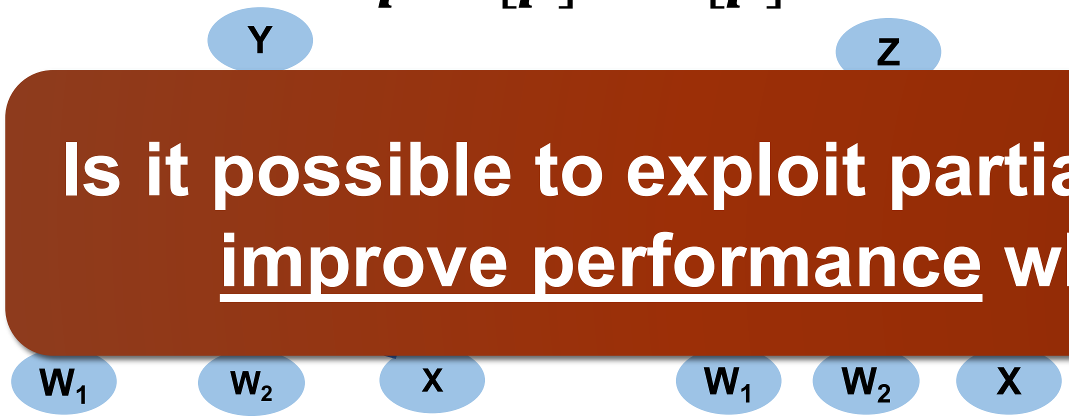
- Pro: better performance
  - Faster ML operators
  - More efficient tensor layouts
  - Hardware-specific optimizations
- Con: potential accuracy loss

Motivation: Fully v.s. Partially Equivalent Transformations

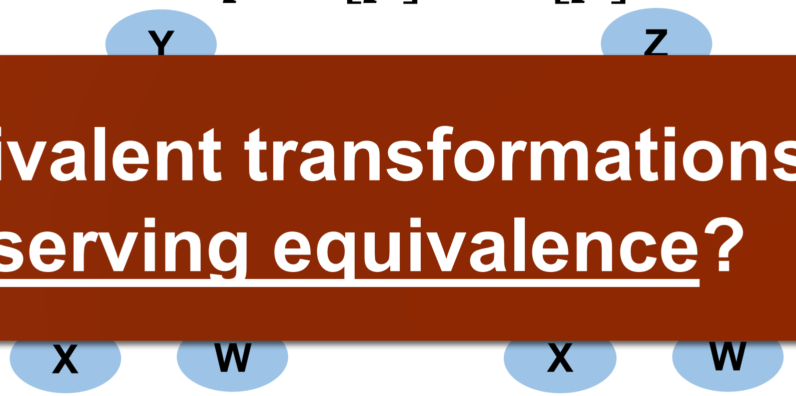
$\forall p.\ Y[p] = Z[p]$

$\exists p.\ Y[p] \neq Z[p]$

Is it possible to exploit partially equivalent transformations to <u>improve performance</u> while <u>preserving equivalence</u>?



Fully Equivalent Transformations



Partially Equivalent Transformations

Pro: preserve functionality

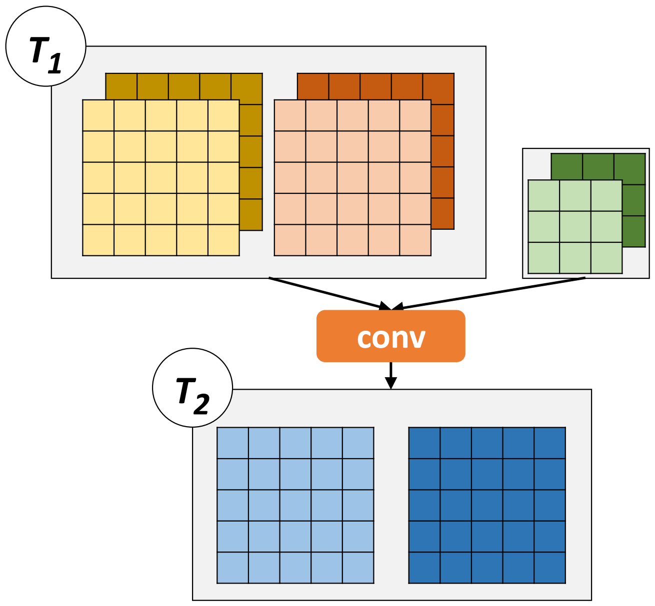
Con: miss optimization opportunities

Pro: better performance

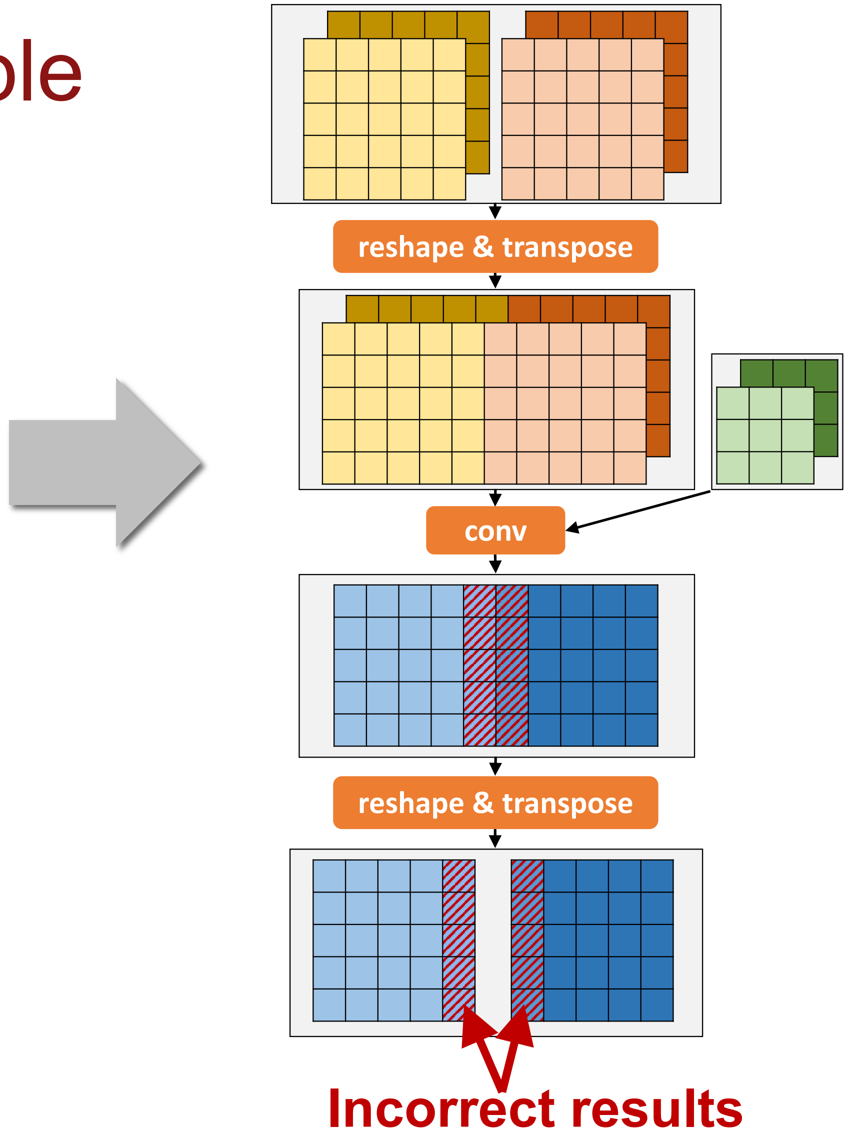
- Faster ML operators
- More efficient tensor layouts
- Hardware-specific optimizations

Con: potential accuracy loss

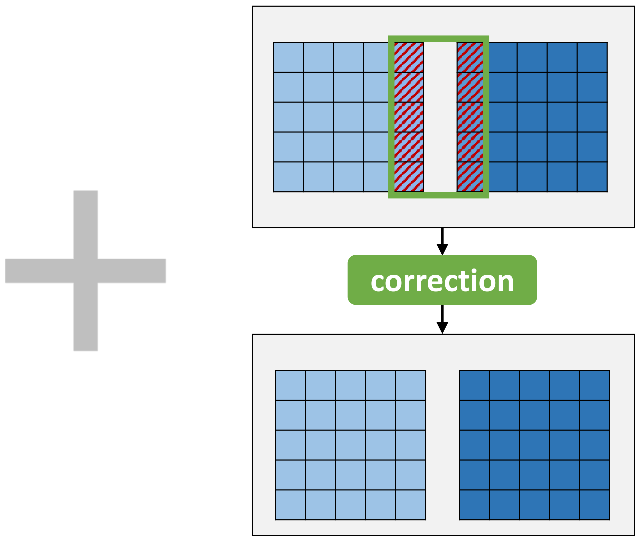
Motivating Example



Input Program

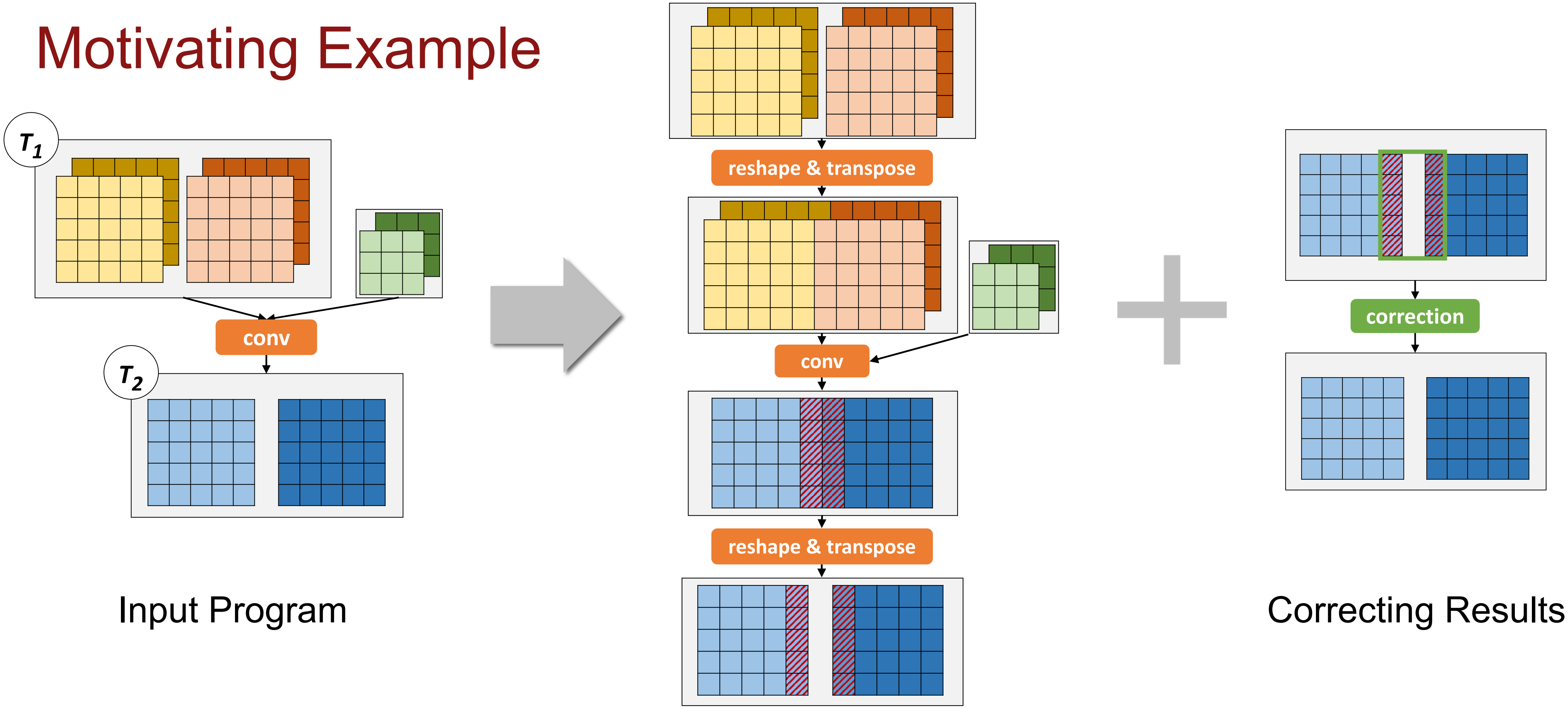


Partially Equivalent Transformation



Correcting Results

Motivating Example



Input Program

Correcting Results

- Transformation and correction lead to **1.2x** speedup for ResNet-18
- Correction preserves end-to-end equivalence

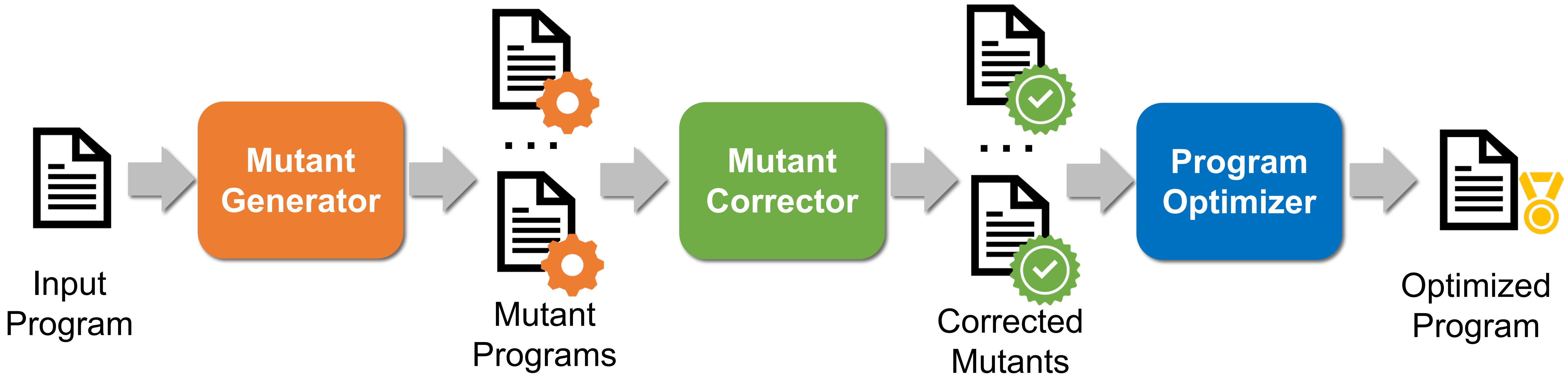
PET

- **First tensor program optimizer** with partially equivalent transformations

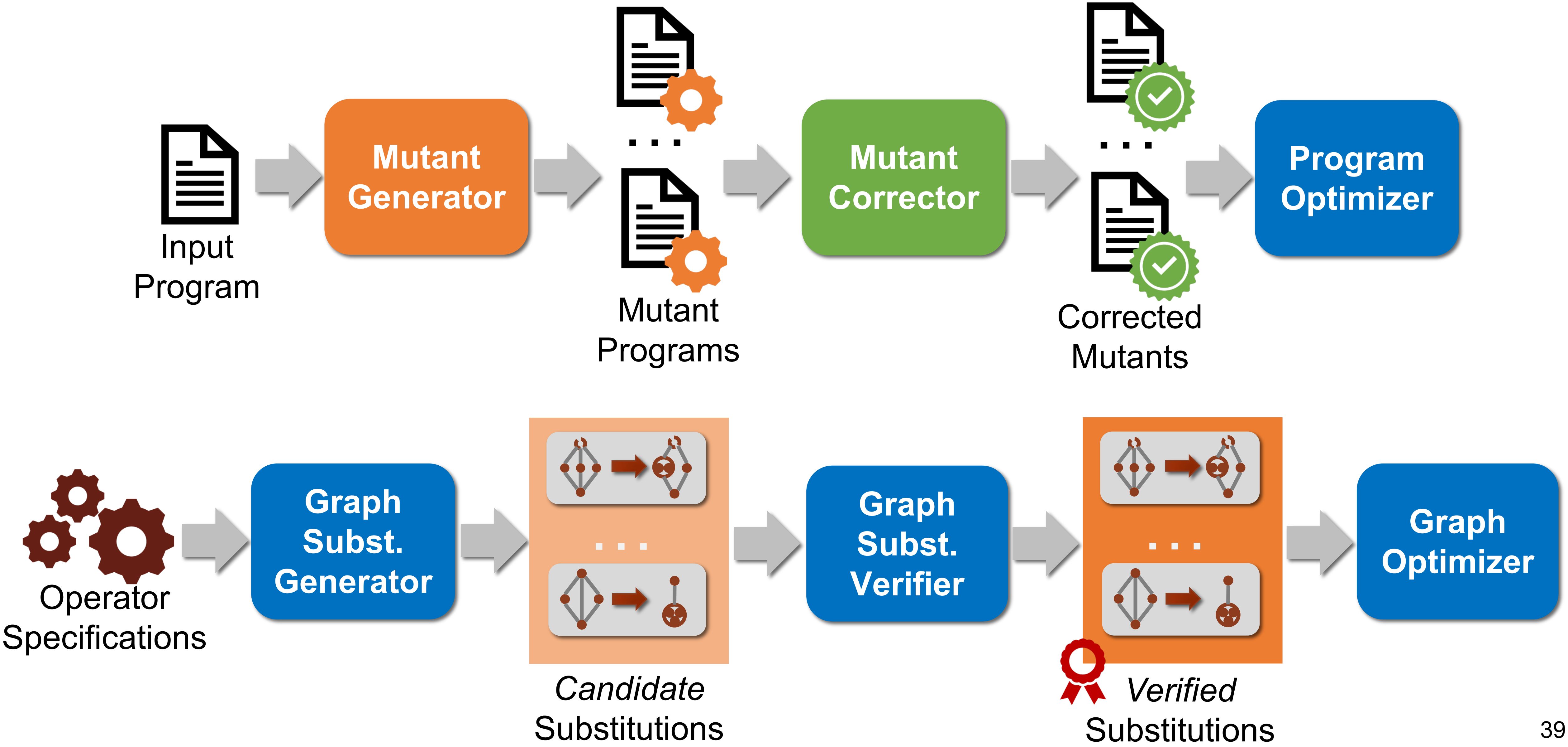
- **Larger optimization space** by combining fully and partially equivalent transformations
- **Better performance**: outperform existing optimizers by up to **2.5x**
- **Correctness**: automated corrections to preserve end-to-end equivalence

PET: Optimizing Tensor Programs with Partially Equivalent Transformations and Automated Corrections. OSDI'21

PET Overview



PET vs TASO


Input Program
Mutant Generator
Mutant Programs
Mutant Corrector
Corrected Mutants
Program Optimizer
Operator Specifications
Graph Subst. Generator
Candidate Substitutions
Graph Subst. Verifier
Verified Substitutions
Graph Optimizer

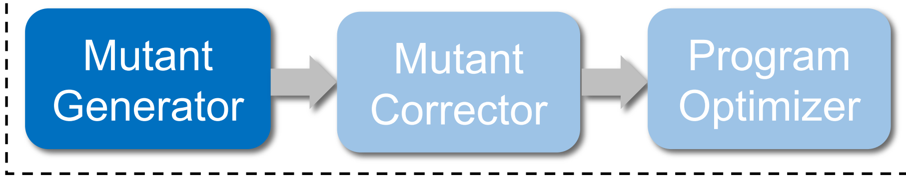
Key Challenges

1. How to generate partially equivalent transformations?

Superoptimization

2. How to correct them?

Multi-linearity of DNN computations

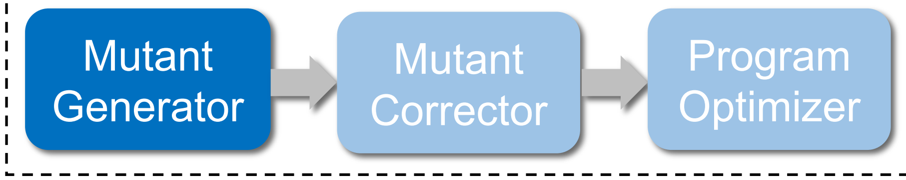


Mutant Generator

Superoptimization adopted from TASO[1]

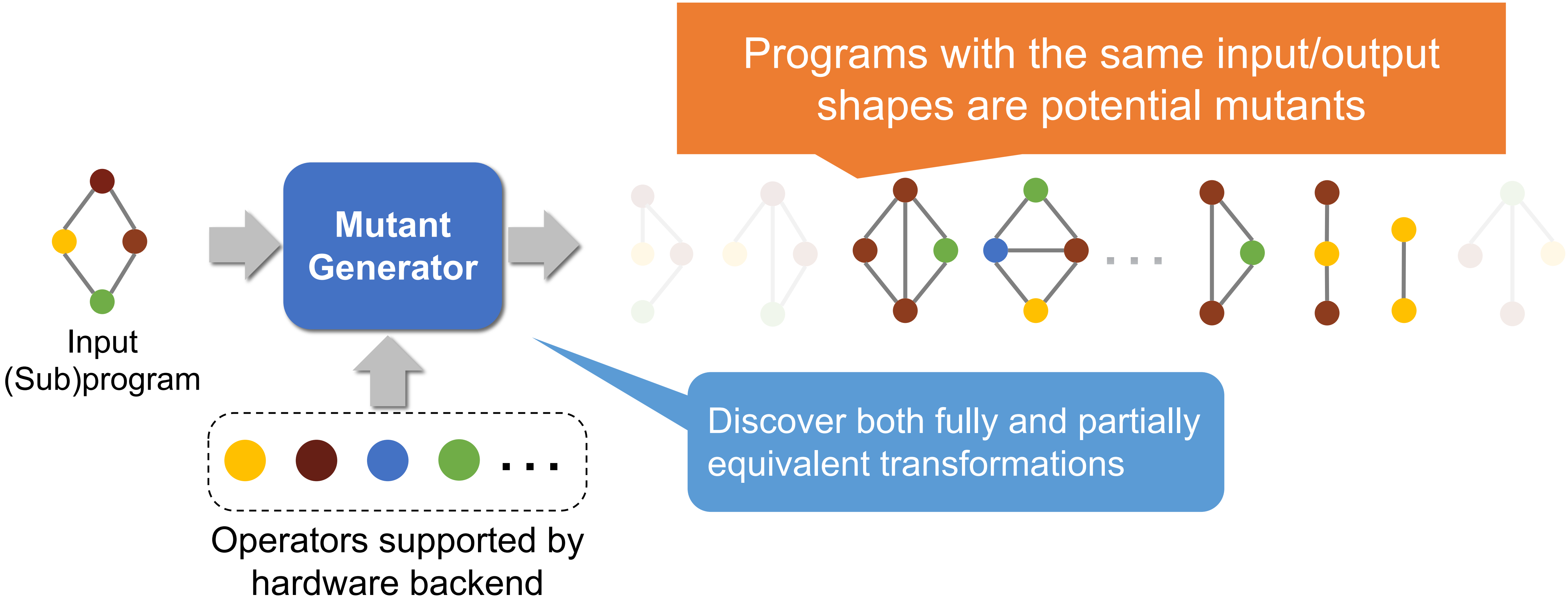
Enumerate **all possible** programs up to a fixed size using available operators

1. TASO: Optimizing Deep Learning Computation with Automated Generation of Graph Substitutions. SOSP'19.



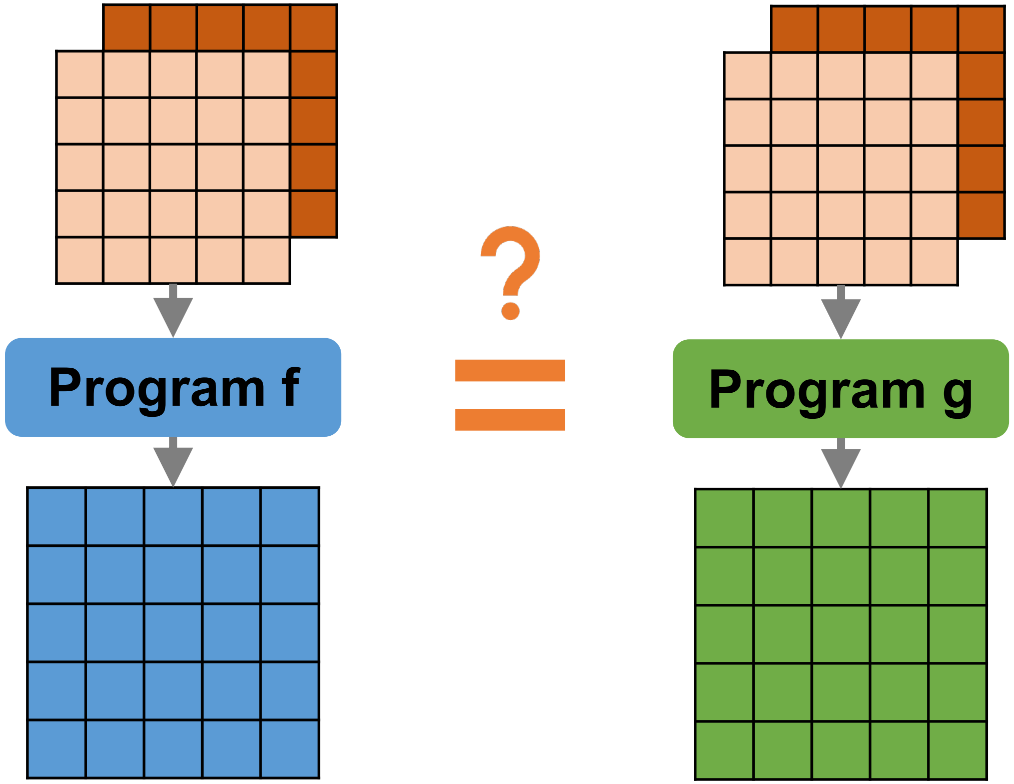
Mutant Generator

Superoptimization adopted from TASO[1]



1. TASO: Optimizing Deep Learning Computation with Automated Generation of Graph Substitutions. SOSP'19.

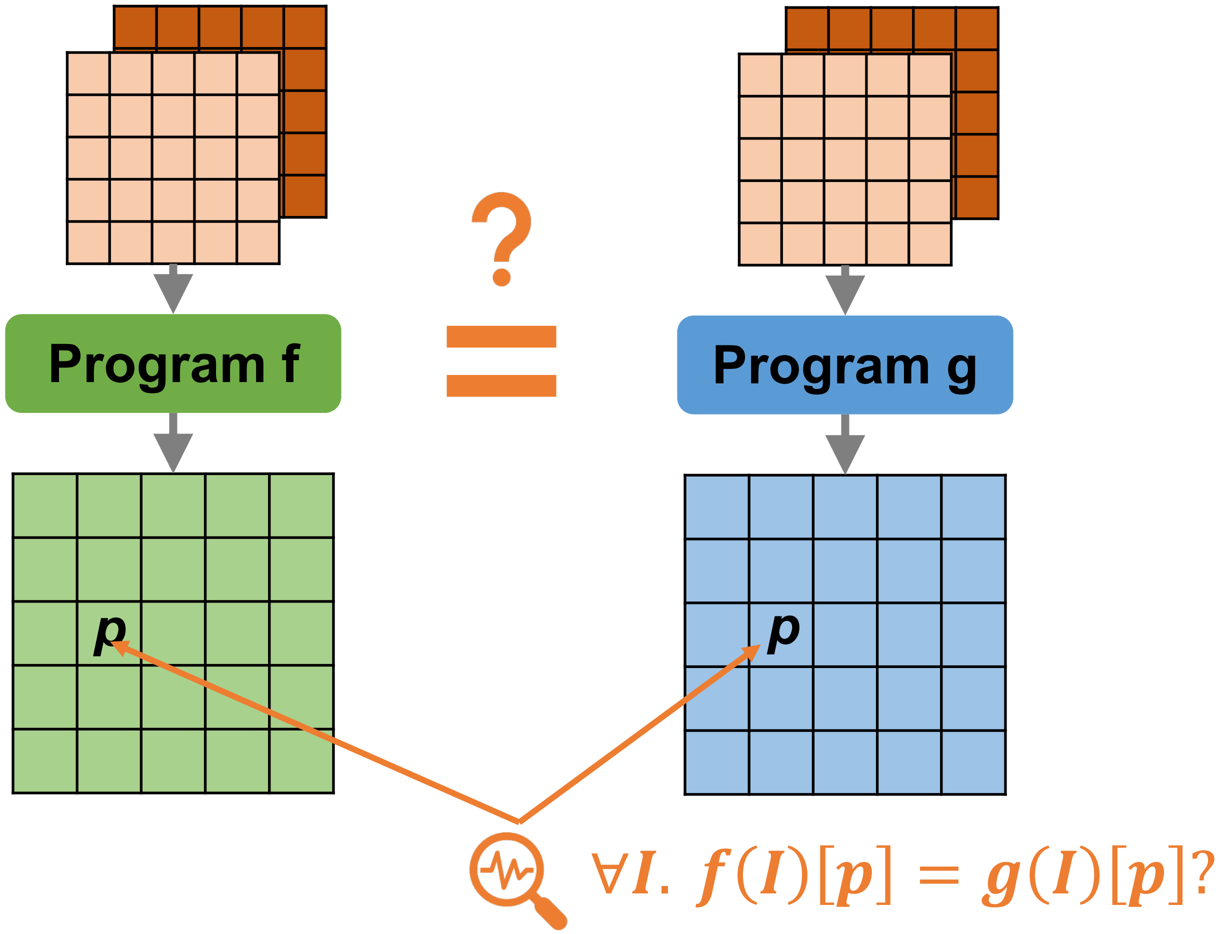
Challenges: Examine Transformations



1. Which part of the computation is not equivalent?
2. How to correct the results?

A Strawman Approach

- **Step 1**: Explicitly consider all output positions (m positions)
- **Step 2**: For each position ***p***, examine all possible inputs (n inputs)



Require O(m * n) examinations, but both m and n are too large to explicitly enumerate

Multi-Linear Tensor Program (MLTP)

- A program $\boldsymbol{f}$ is multi-linear if the output is linear to all inputs
  - $\boldsymbol{f}(I_1, \dots, X, \dots, I_n) + \boldsymbol{f}(I_1, \dots, Y, \dots, I_n) = \boldsymbol{f}(I_1, \dots, X + Y, \dots, I_n)$
  - $\alpha \cdot \boldsymbol{f}(I_1, \dots, X, \dots, I_n) = \boldsymbol{f}(I_1, \dots, \alpha \cdot X, \dots, I_n)$

- DNN computation = MLTP + non-linear activations

Majority of the computation

O(m * n) examinations in strawman approach → MLTP → **O(1) examinations in PET's approach**

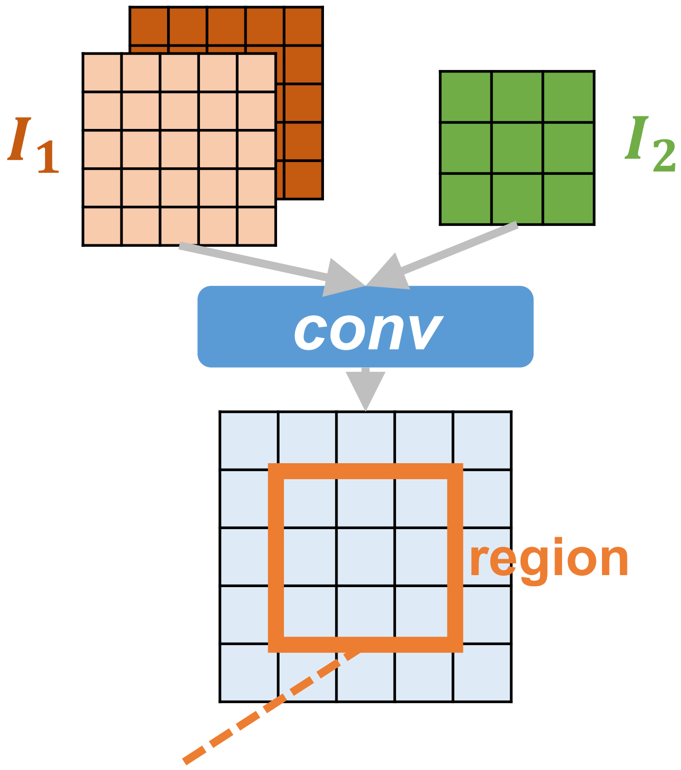
Insight #1: No Need to Enumerate All Output Positions

Group all output positions with an identical summation interval into a region

***Theorem 1**: For two MLTPs **f** and **g**, if **f=g** for O(1) positions in a region, then **f=g** for all positions in the region

Only need to examine O(1) positions for each region.

Complexity: **O(m * n) → O(n)**



$$conv(c,h,w) = \sum_{d=0}^{D-1} \sum_{x=-1}^{1} \sum_{y=-1}^{1} I_1(d, h+x, w+y) \times I_2(d, c, x, y)$$

Summation interval

*Proof details available in the paper

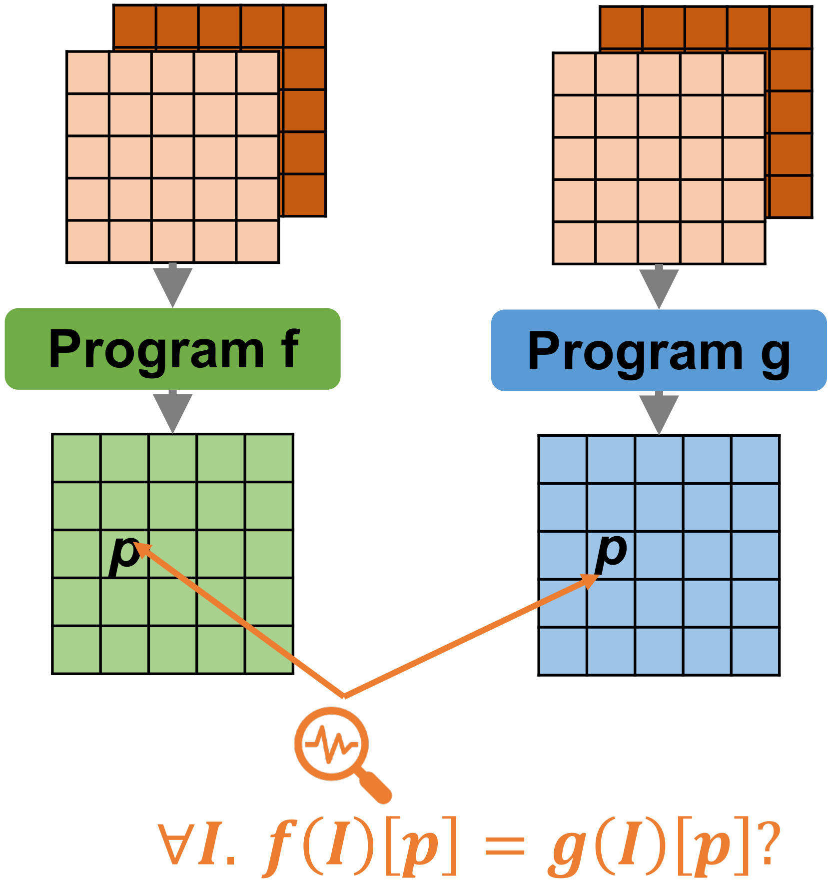
Insight #2: No Need to Consider All Possible Inputs

Examining equivalence for a single position is still challenging

Theorem 2**: If $\exists I.\ f(I)[p] \neq g(I)[p]$, then the probability that **f** and **g** give identical results on ***t random integer inputs is $(\frac{1}{2^{31}})^t$

Run ***t*** random tests for each position $\boldsymbol{p}$

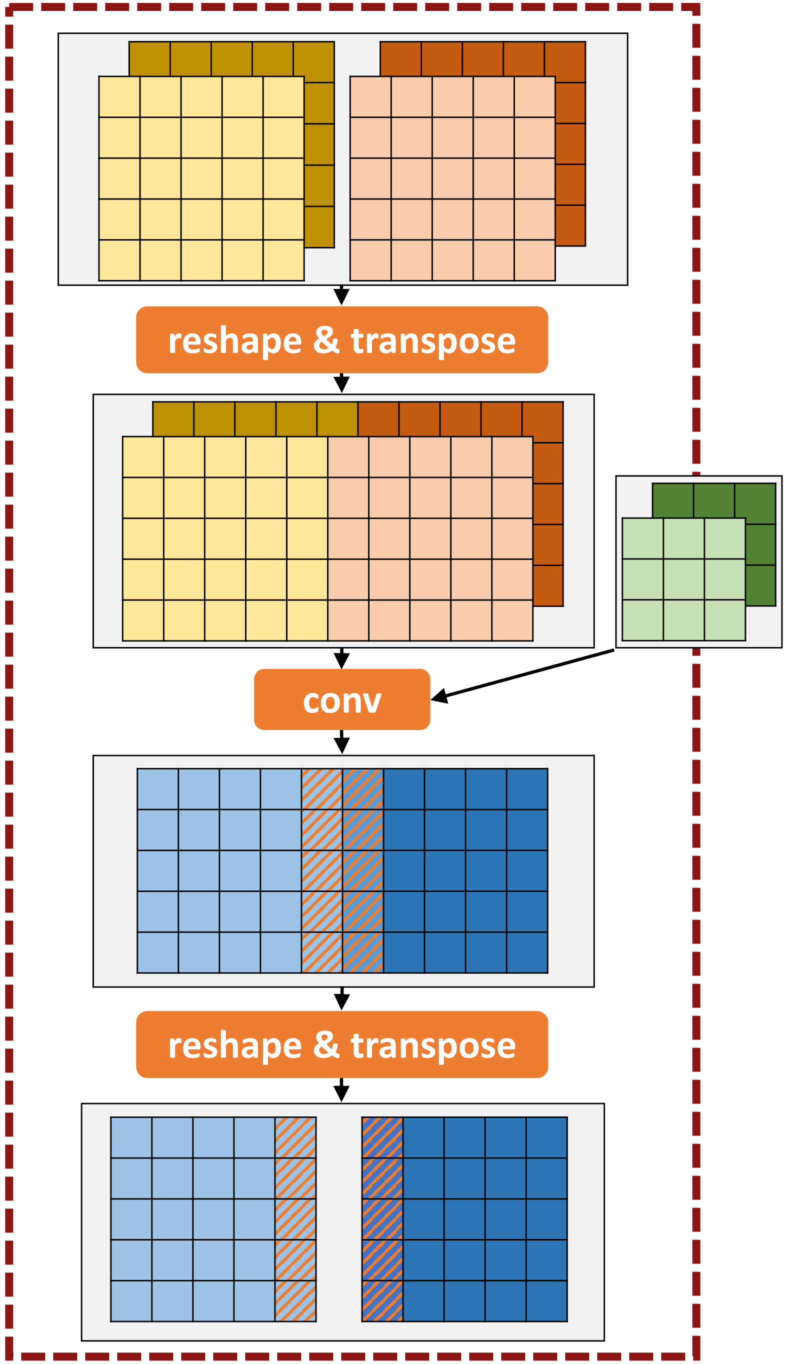
Complexity: **O(n) → O(*t*) = O(1)**



*Proof details available in the paper

Mutant Corrector

Goal: **quickly** and **efficiently** correcting the outputs of a mutant program

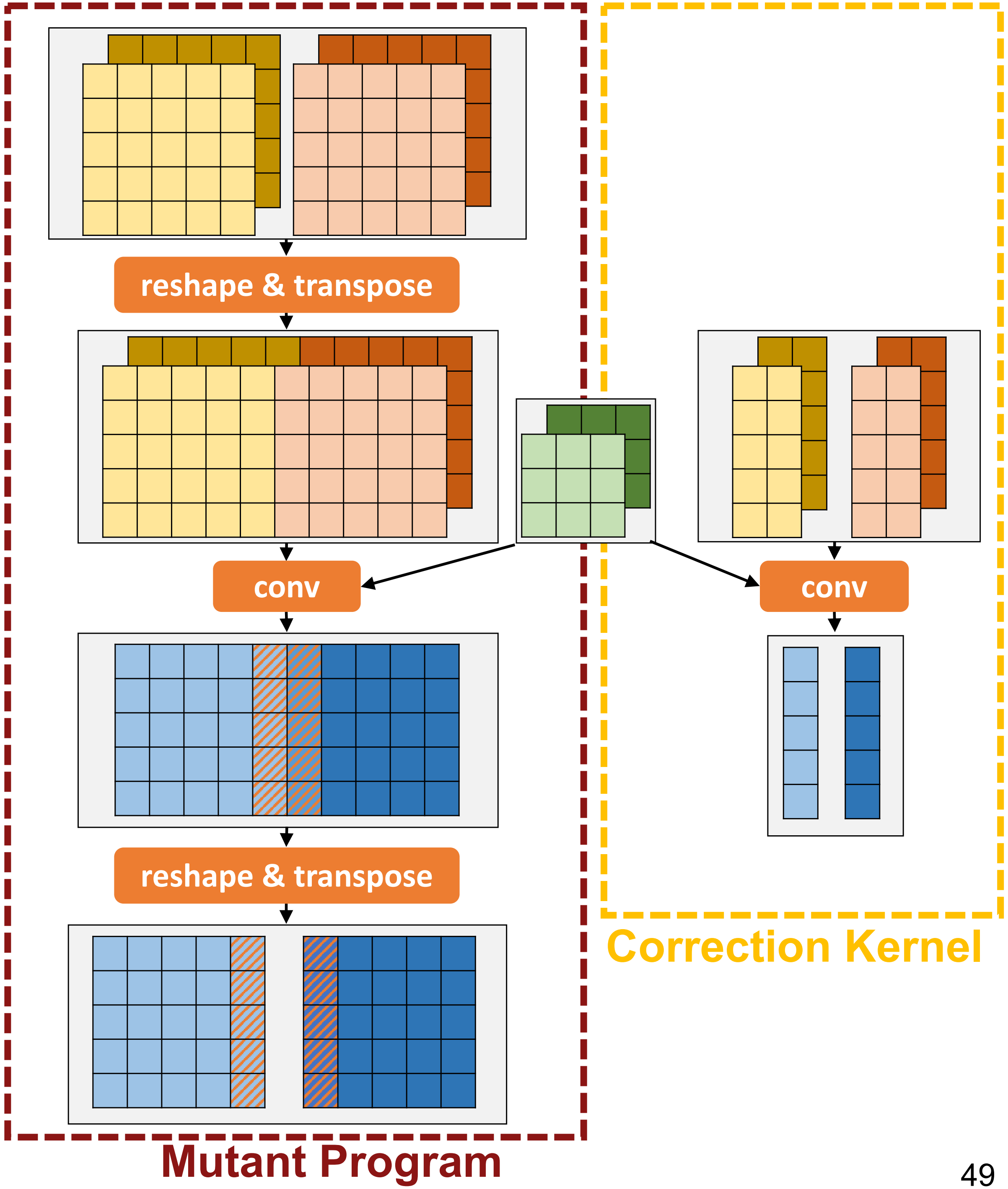


Mutant Program

Mutant Corrector

Goal: **quickly** and **efficiently** correcting the outputs of a mutant program

Step 1: recompute the incorrect outputs using the original program



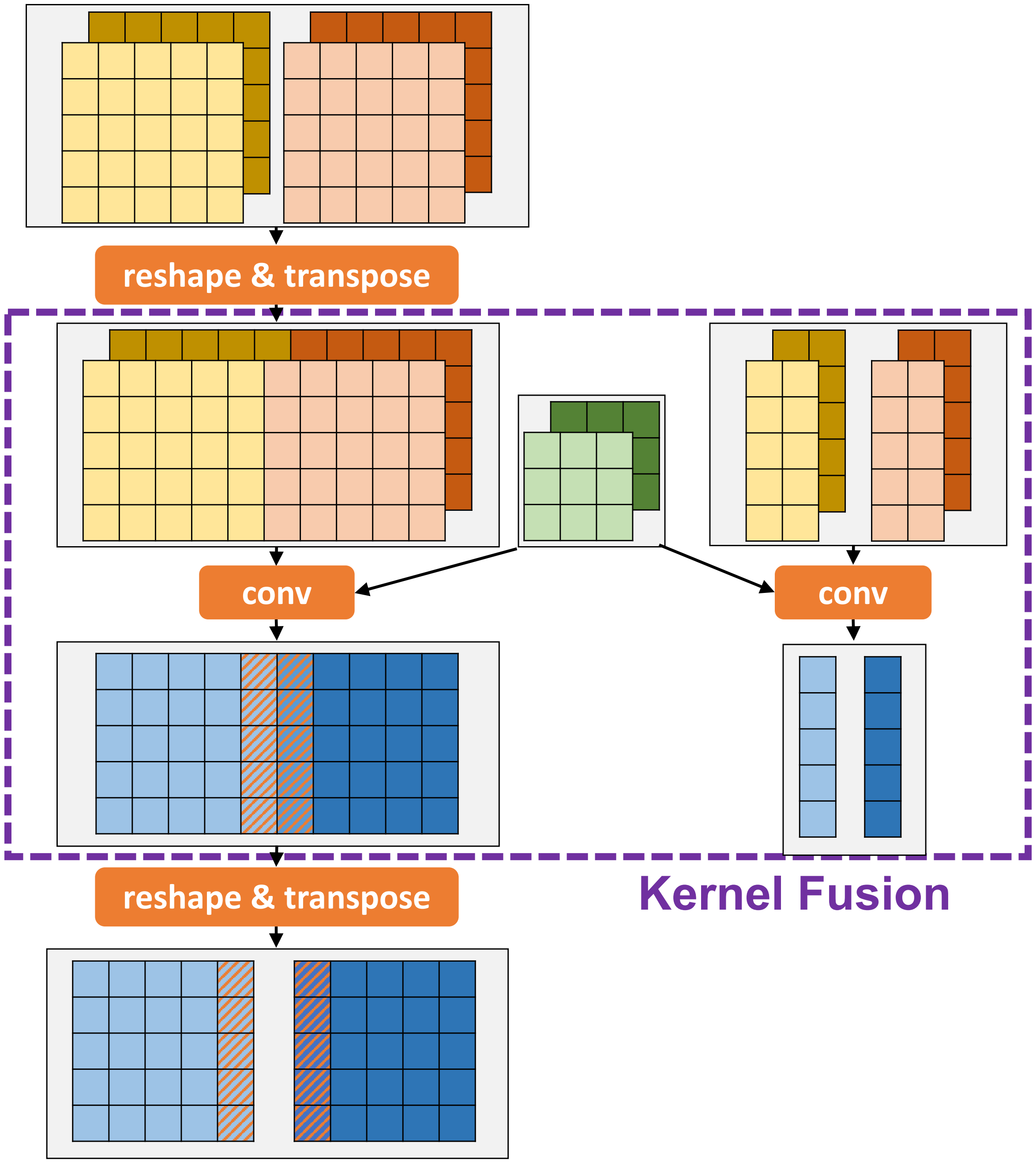
Mutant Corrector

Goal: **quickly** and **efficiently** correcting the outputs of a mutant program

Step 1: recompute the incorrect outputs using the original program

Step 2: opportunistically fuse correction kernels with other operators

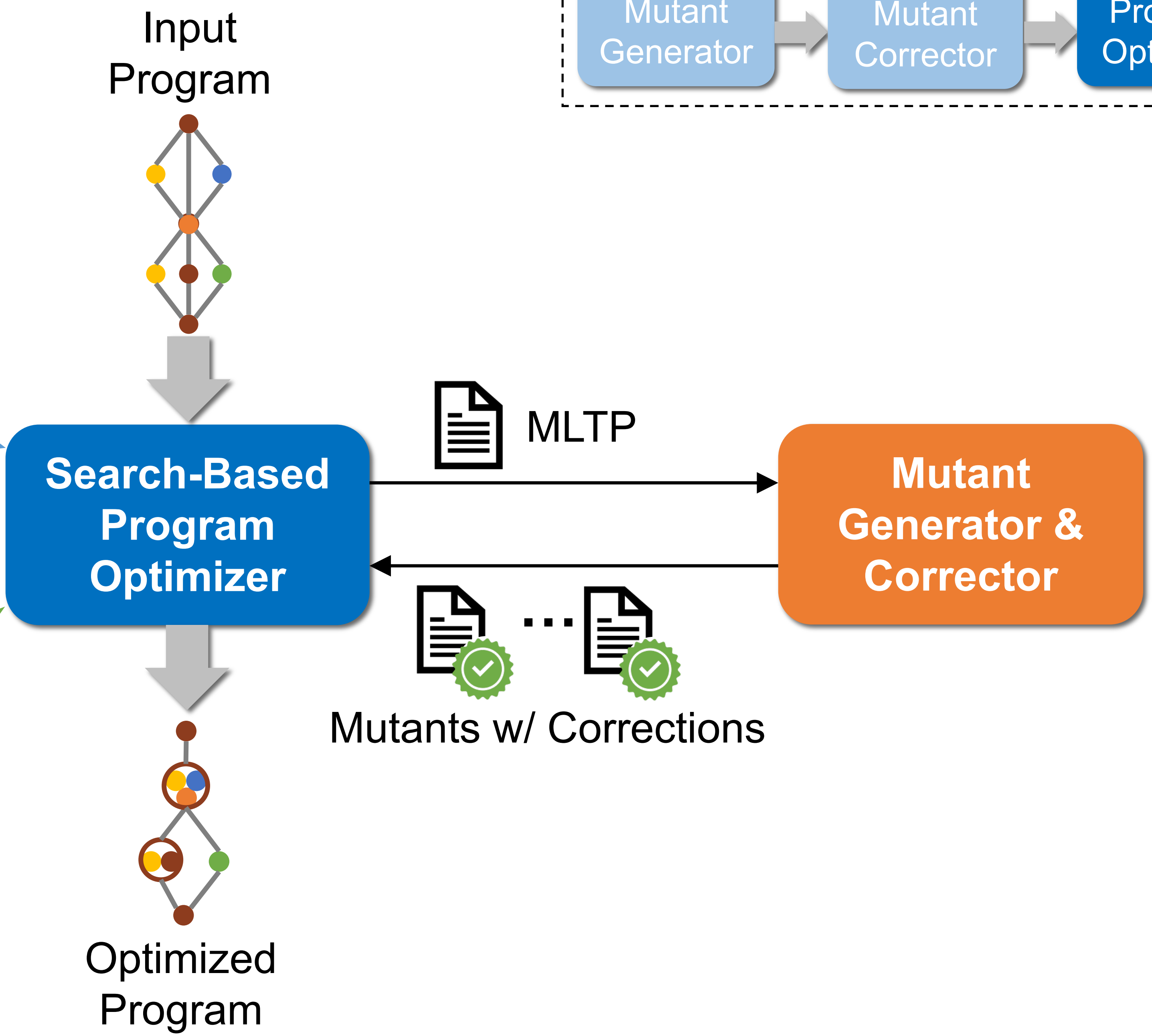
Correction introduces less than 1% overhead



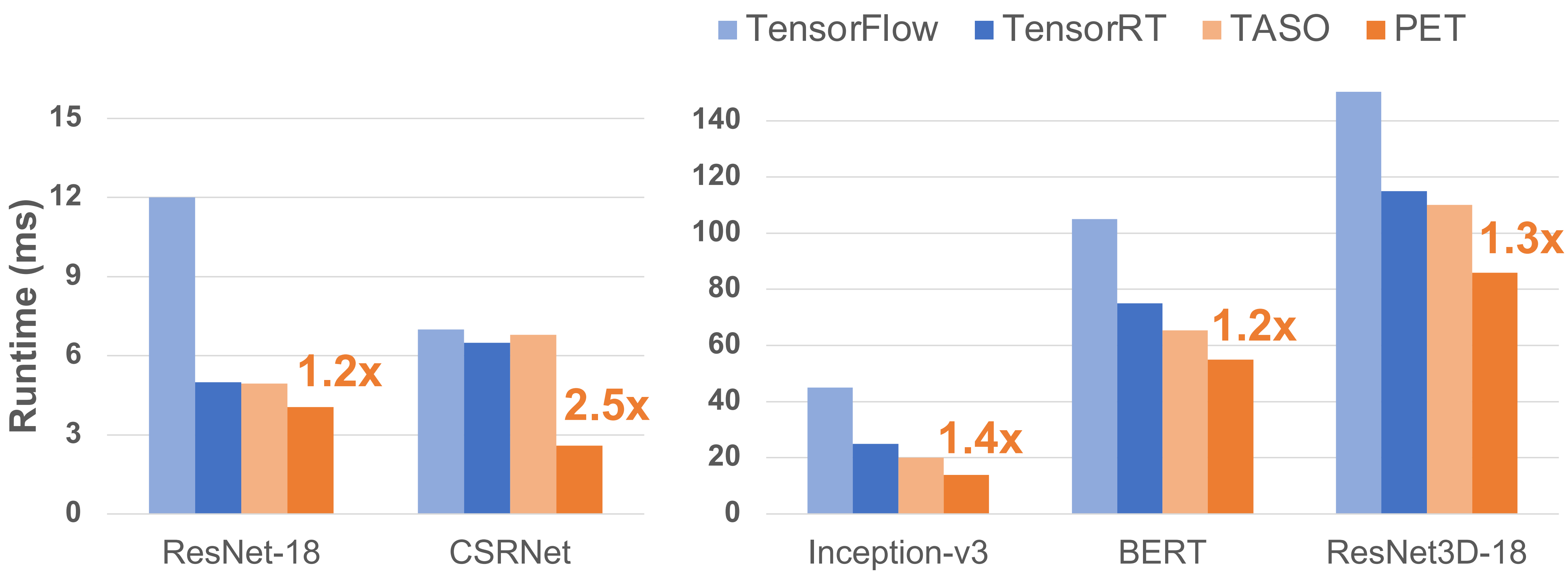
Program Optimizer

- **Beam search**
- Optimizing a DNN architecture takes less than <u>**30**</u> minutes

Other optimizations:
- Operator fusion
- Constant folding
- Redundancy elimination



End-to-end Inference Performance (Nvidia V100 GPU)



PET outperforms existing optimizers by 1.2-2.5x
by combining fully and partially equivalent transformations

Recap: PET

- A tensor program optimizer with partially equivalent transformations and automated corrections

- **Larger optimization space** by combining fully and partially equivalent transformations
- **Better performance**: outperform existing optimizers by up to **2.5x**
- **Correctness**: automated corrections to preserve end-to-end equivalence

From Equivalent to Non-Equivalent Optimizations for ML

TASO

Equivalent optimizations

→

PET

Partially-equivalent optimizations w/ automated corrections

→

Non-equivalent optimizations w/ accuracy guarantees

Model Pruning, Quantization, Distillation, etc.

Questions to Discuss

1. How does PET differ from TASO in generating graph transformations?

2. How does PET differ from TASO in verifying/correcting transformations?

3. How can we combine graph optimizations with kernel optimizations?