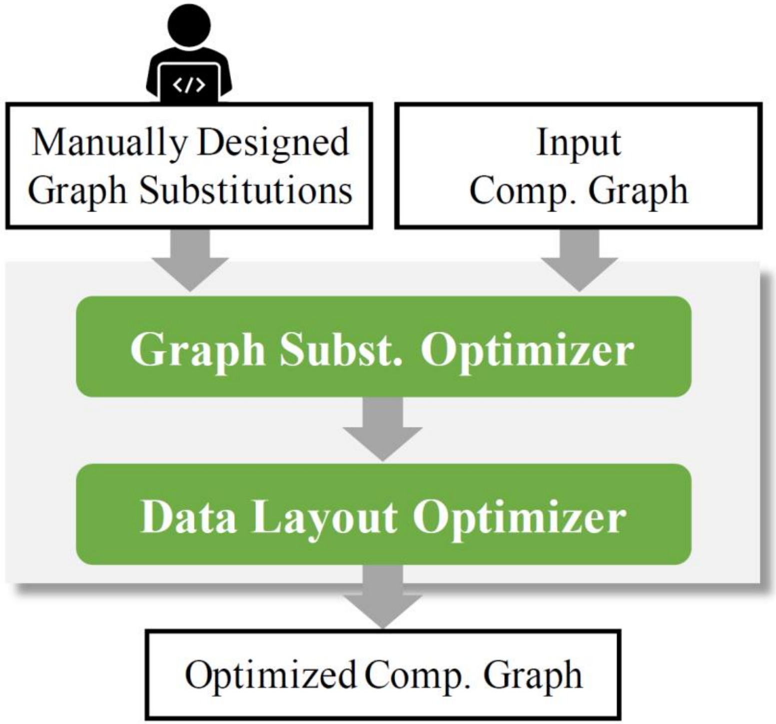
UNIVERSITY OF CAMBRIDGE

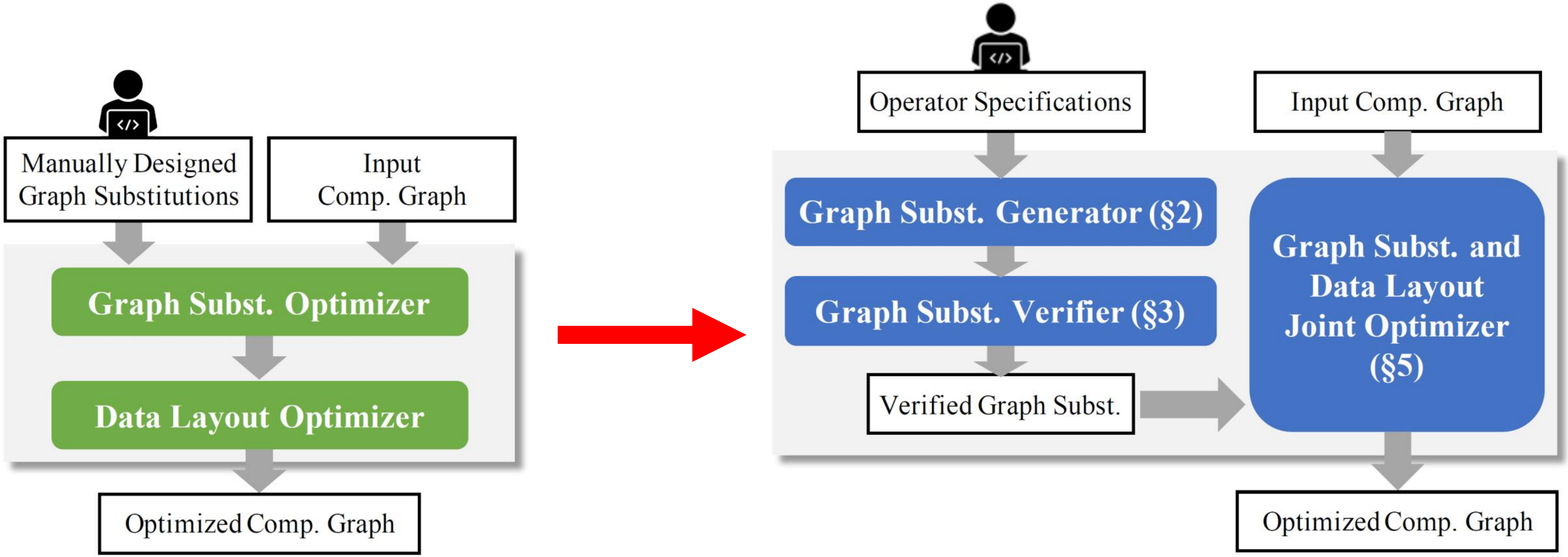
TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions

Jia, Z., Padon, O., Thomas, J., Warszawski, T., Zaharia, M., & Aiken, A. (2019, October). In Proceedings of the 27th ACM Symposium on Operating Systems Principles (pp. 47-62).

R244 Large-scale data processing and optimisation
Presentation by Martin Graf on 16/11/2022

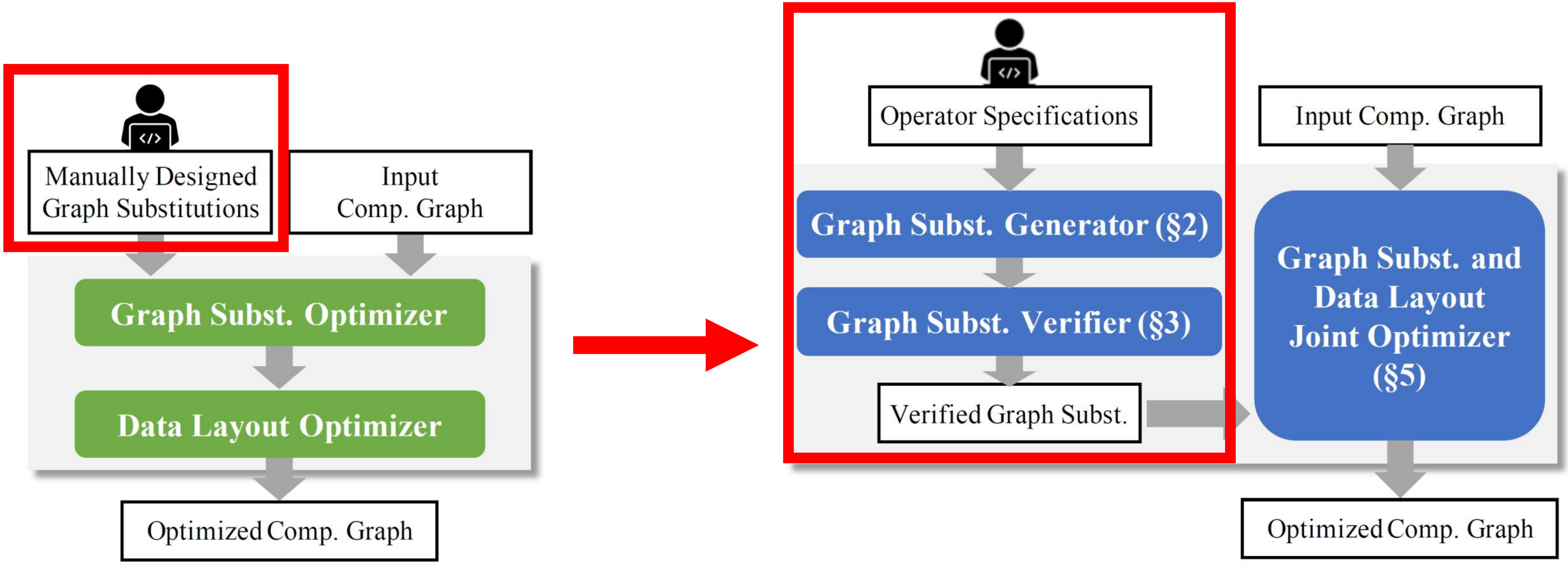


(a) Existing DNN frameworks.



(a) Existing DNN frameworks.

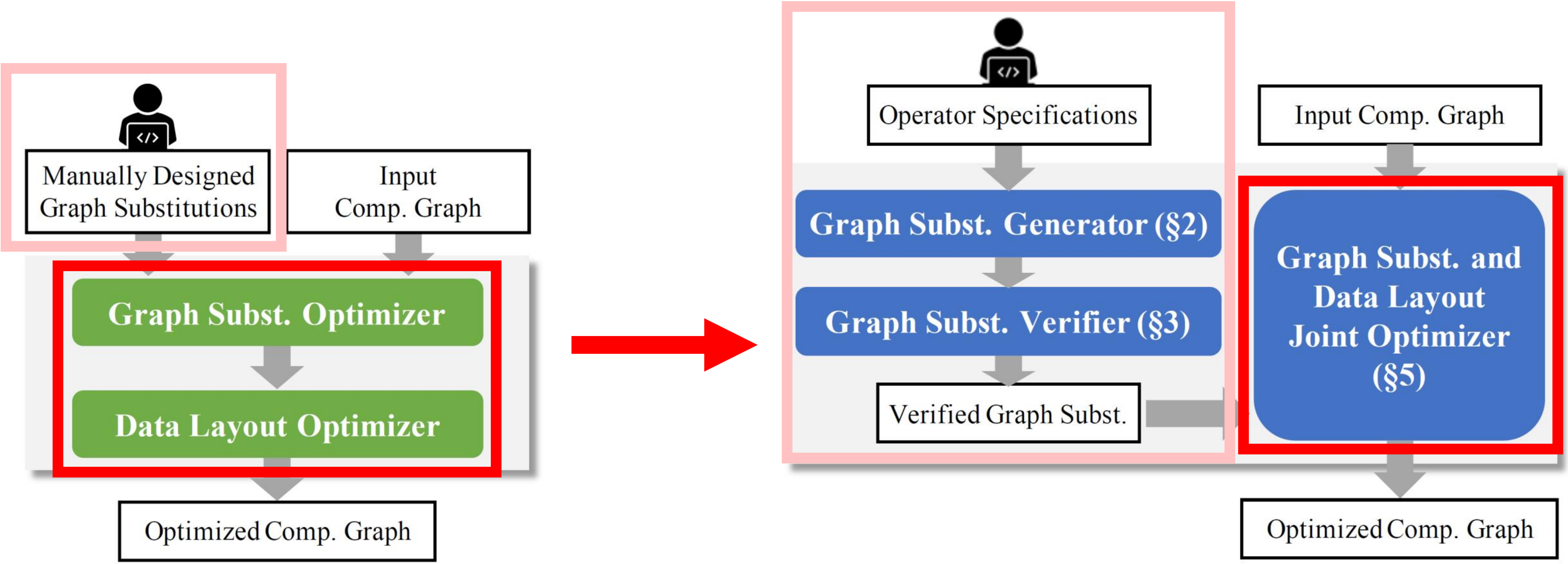
(b) TASO.



(a) Existing DNN frameworks.

(b) TASO.

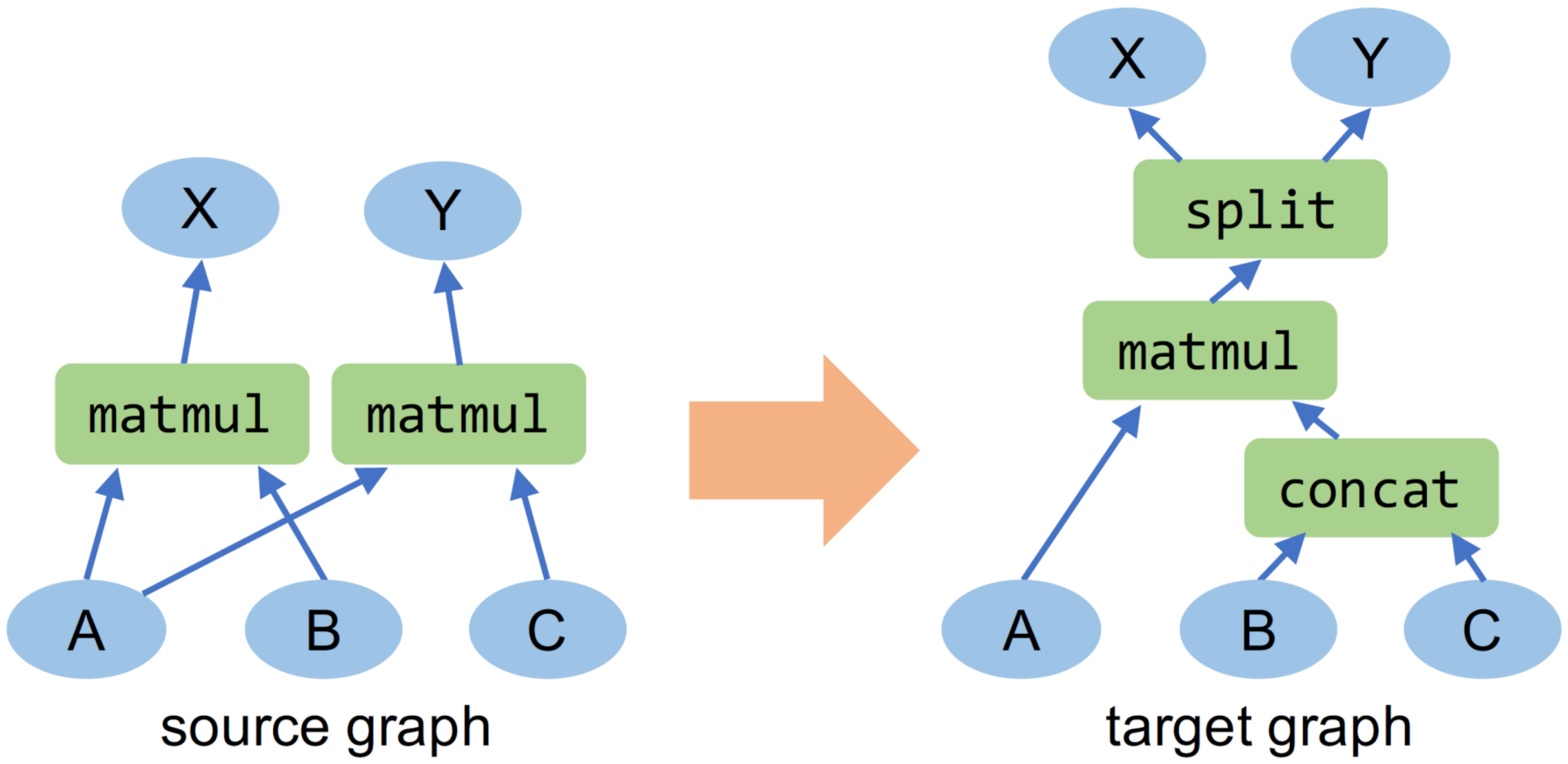
UNIVERSITY OF CAMBRIDGE



(a) Existing DNN frameworks.

(b) TASO.

UNIVERSITY OF CAMBRIDGE



(b) Fusing two matrix multiplications using concatenation and split.

Algorithm 1 Graph substitution generation algorithm.

Input: A set of operators $\mathcal{P}$, and a set of input tensors $\mathcal{I}$.
Output: Candidate graph substitutions $\mathcal{S}$.

// Step 1: enumerating potential graphs.
$\mathcal{D} = \{\}$ // $\mathcal{D}$ is a graph hash table indexed by their fingerprints.
$\textsc{Build}(1, \emptyset, \mathcal{I})$
function $\textsc{Build}(n, \mathcal{G}, \mathcal{I})$
    if $\mathcal{G}$ contains duplicated computation **then**
        return
    $\mathcal{D} = \mathcal{D} + (\textsc{FingerPrint}(\mathcal{G}), \mathcal{G})$
    if $n < threshold$ **then**
        for $op \in \mathcal{P}$ **do**
            for $i \in \mathcal{I}$ and i is a valid input to op **do**
                Add operator op into graph $\mathcal{G}$.
                Add the output tensors of op into $\mathcal{I}$.
                $\textsc{Build}(n + 1, \mathcal{G}, \mathcal{I})$
                Remove operator op from $\mathcal{G}$.
                Remove the output tensors of op from $\mathcal{I}$.

// Step 2: testing graphs with identical fingerprint.
$\mathcal{S} = \{\}$
for $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{D}$ with the same $\textsc{FingerPrint}(\cdot)$ **do**
    if $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent for all test cases **then**
        $\mathcal{S} = \mathcal{S} + (\mathcal{G}_1, \mathcal{G}_2)$
return $\mathcal{S}$

Enumerate potential graphs

UNIVERSITY OF CAMBRIDGE

Algorithm 1 Graph substitution generation algorithm.

Input: A set of operators $\mathcal{P}$, and a set of input tensors $\mathcal{I}$.
Output: Candidate graph substitutions $\mathcal{S}$.

// Step 1: enumerating potential graphs.
$\mathcal{D} = \{\}$ // $\mathcal{D}$ is a graph hash table indexed by their fingerprints.
$\textsc{Build}(1, \emptyset, \mathcal{I})$
function $\textsc{Build}(n, \mathcal{G}, \mathcal{I})$
    if $\mathcal{G}$ contains duplicated computation **then**
        return
    $\mathcal{D} = \mathcal{D} + (\textsc{FingerPrint}(\mathcal{G}), \mathcal{G})$
    if $n < threshold$ **then**
        for $op \in \mathcal{P}$ **do**
            for $i \in \mathcal{I}$ and i is a valid input to op **do**
                Add operator op into graph $\mathcal{G}$.
                Add the output tensors of op into $\mathcal{I}$.
                $\textsc{Build}(n + 1, \mathcal{G}, \mathcal{I})$
                Remove operator op from $\mathcal{G}$.
                Remove the output tensors of op from $\mathcal{I}$.

// Step 2: testing graphs with identical fingerprint.
$\mathcal{S} = \{\}$
for $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{D}$ with the same $\textsc{FingerPrint}(\cdot)$ **do**
    if $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent for all test cases **then**
        $\mathcal{S} = \mathcal{S} + (\mathcal{G}_1, \mathcal{G}_2)$
return $\mathcal{S}$

Enumerate potential graphs

Collect and test candidate substitutions

UNIVERSITY OF CAMBRIDGE

Algorithm 1 Graph substitution generation algorithm.

Input: A set of operators $\mathcal{P}$, and a set of input tensors $\mathcal{I}$.
Output: Candidate graph substitutions $\mathcal{S}$.

// Step 1: enumerating potential graphs.
$\mathcal{D} = \{\}$ // $\mathcal{D}$ is a graph hash table indexed by their fingerprints.
$\textsc{Build}(1, \emptyset, \mathcal{I})$
function $\textsc{Build}(n, \mathcal{G}, \mathcal{I})$
  if $\mathcal{G}$ contains duplicated computation **then**
    return
  $\mathcal{D} = \mathcal{D} + (\textsc{FingerPrint}(\mathcal{G}), \mathcal{G})$
  if $n < threshold$ **then**
    for $op \in \mathcal{P}$ **do**
      for $i \in \mathcal{I}$ and i is a valid input to op **do**
        Add operator op into graph $\mathcal{G}$.
        Add the output tensors of op into $\mathcal{I}$.
        $\textsc{Build}(n + 1, \mathcal{G}, \mathcal{I})$
        Remove operator op from $\mathcal{G}$.
        Remove the output tensors of op from $\mathcal{I}$.

// Step 2: testing graphs with identical fingerprint.
$\mathcal{S} = \{\}$
for $\mathcal{G}_1, \mathcal{G}_2 \in \mathcal{D}$ with the same $\textsc{FingerPrint}(\cdot)$ **do**
  if $\mathcal{G}_1$ and $\mathcal{G}_2$ are equivalent for all test cases **then**
    $\mathcal{S} = \mathcal{S} + (\mathcal{G}_1, \mathcal{G}_2)$
return $\mathcal{S}$

Enumerate potential graphs
→ Fingerprinting + Hash Collisions
(Bansal, 2006)

Collect and test candidate substitutions

UNIVERSITY OF CAMBRIDGE

Table 2. Operator properties used for verification. The operators are defined in Table 1, and the properties are grouped by the operators they involve. Logical variables w, x, y, and z are of type tensor, and variables a, c, k, p, and s are of type parameter. The variable a is used for the axis of concatenation and split, c for the activation mode of convolution, k for the kernel shape of pooling, p for the padding mode of convolution and pooling, and s for the strides of convolution and pooling.

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
$\forall x, y.\ \texttt{transpose}(\texttt{matmul}(x, y)) = \texttt{matmul}(\texttt{transpose}(y), \texttt{transpose}(x))$	matmul and transpose
$\forall s, p, c, x, y, w.\ \texttt{conv}(s, p, c, \texttt{smul}(x, w), y) = \texttt{conv}(s, p, c, x, \texttt{smul}(y, w))$	conv is bilinear
$\forall s, p, x, y, w.\ \texttt{smul}(\texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, x, y), w) = \texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, \texttt{smul}(x, w), y)$	conv is bilinear
$\forall s, p, x, y, z.\ \texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, x, \texttt{ewadd}(y, z)) = \texttt{ewadd}(\texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, x, y), \texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, x, z))$	conv is bilinear
$\forall s, p, x, y, z.\ \texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, \texttt{ewadd}(x, y), z) = \texttt{ewadd}(\texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, x, z), \texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, y, z))$	conv is bilinear
$\forall s, c, k, x, y.\ \texttt{conv}(s, \mathsf{P}_{\texttt{same}}, c, x, y) = \texttt{conv}(s, \mathsf{P}_{\texttt{same}}, c, x, \texttt{enlarge}(k, y)),$	enlarge convolution kernel
$\forall s, p, x, y.\ \texttt{conv}(s, p, \mathsf{A}_{\texttt{relu}}, x, y) = \texttt{relu}(\texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, x, y))$	conv with $\mathsf{A}_{\texttt{relu}}$ applies relu
$\forall x.\ \texttt{relu}(\texttt{transpose}(x)) = \texttt{transpose}(\texttt{relu}(x))$	operator commutativity
$\forall s, p, x, k.\ \texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, x, \mathsf{C}_{\texttt{pool}}(k)) = \texttt{pool}_{\texttt{avg}}(k, s, p, x)$	pooling by conv. with $\mathsf{C}_{\texttt{pool}}$
$\forall k, x.\ \texttt{conv}(1, \mathsf{P}_{\texttt{same}}, \mathsf{A}_{\texttt{none}}, x, \mathsf{I}_{\texttt{conv}}(k)) = x$	identity kernel
$\forall x.\ \texttt{matmul}(x, \mathsf{I}_{\texttt{matmul}}) = x$	identity matrix
$\forall x.\ \texttt{ewmul}(x, \mathsf{I}_{\texttt{ewmul}}) = x$	ewmul identity
$\forall a, x, y.\ \texttt{split}_0(a, \texttt{concat}(a, x, y)) = x$	split definition
$\forall a, x, y.\ \texttt{split}_1(a, \texttt{concat}(a, x, y)) = y$	split definition
$\forall x, y, z, w.\ \texttt{concat}(0, \texttt{concat}(1, x, y), \texttt{concat}(1, z, w)) = \texttt{concat}(1, \texttt{concat}(0, x, z), \texttt{concat}(0, y, w))$	geometry of concatenation
$\forall a, x, y, w.\ \texttt{concat}(a, \texttt{smul}(x, w), \texttt{smul}(y, w)) = \texttt{smul}(\texttt{concat}(a, x, y), w)$	operator commutativity
$\forall a, x, y, z, w.\ \texttt{concat}(a, \texttt{ewadd}(x, y), \texttt{ewadd}(z, w)) = \texttt{ewadd}(\texttt{concat}(a, x, z), \texttt{concat}(a, y, w))$	operator commutativity
$\forall a, x, y, z, w.\ \texttt{concat}(a, \texttt{ewmul}(x, y), \texttt{ewmul}(z, w)) = \texttt{ewmul}(\texttt{concat}(a, x, z), \texttt{concat}(a, y, w))$	operator commutativity
$\forall a, x, y.\ \texttt{concat}(a, \texttt{relu}(x), \texttt{relu}(y)) = \texttt{relu}(\texttt{concat}(a, x, y))$	operator commutativity
$\forall x, y.\ \texttt{concat}(1, \texttt{transpose}(x), \texttt{transpose}(y)) = \texttt{transpose}(\texttt{concat}(0, x, y))$	concatenation and transpose
$\forall x, y, z.\ \texttt{concat}(1, \texttt{matmul}(x, y), \texttt{matmul}(x, z)) = \texttt{matmul}(x, \texttt{concat}(1, y, z))$	concatenation and matrix mul.
$\forall x, y, z, w.\ \texttt{matmul}(\texttt{concat}(1, x, z), \texttt{concat}(0, y, w)) = \texttt{ewadd}(\texttt{matmul}(x, y), \texttt{matmul}(z, w))$	concatenation and matrix mul.
$\forall s, p, c, x, y, z.\ \texttt{concat}(0, \texttt{conv}(s, p, c, x, z), \texttt{conv}(s, p, c, y, z)) = \texttt{conv}(s, p, c, \texttt{concat}(0, x, y), z)$	concatenation and conv.
$\forall s, p, c, x, y, z.\ \texttt{concat}(1, \texttt{conv}(s, p, c, x, y), \texttt{conv}(s, p, c, x, z)) = \texttt{conv}(s, p, c, x, \texttt{concat}(0, y, z))$	concatenation and conv.
$\forall s, p, x, y, z, w.\ \texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, \texttt{concat}(1, x, z), \texttt{concat}(1, y, w)) =$ $\texttt{ewadd}(\texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, x, y), \texttt{conv}(s, p, \mathsf{A}_{\texttt{none}}, z, w))$	concatenation and conv.
$\forall k, s, p, x, y.\ \texttt{concat}(1, \texttt{pool}_{\texttt{avg}}(k, s, p, x), \texttt{pool}_{\texttt{avg}}(k, s, p, y)) = \texttt{pool}_{\texttt{avg}}(k, s, p, \texttt{concat}(1, x, y))$	concatenation and pooling
$\forall k, s, p, x, y.\ \texttt{concat}(0, \texttt{pool}_{\texttt{max}}(k, s, p, x), \texttt{pool}_{\texttt{max}}(k, s, p, y)) = \texttt{pool}_{\texttt{max}}(k, s, p, \texttt{concat}(0, x, y))$	concatenation and pooling
$\forall k, s, p, x, y.\ \texttt{concat}(1, \texttt{pool}_{\texttt{max}}(k, s, p, x), \texttt{pool}_{\texttt{max}}(k, s, p, y)) = \texttt{pool}_{\texttt{max}}(k, s, p, \texttt{concat}(1, x, y))$	concatenation and pooling

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×

UNIVERSITY OF CAMBRIDGE

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**
    // $\textsc{Layout}(\mathcal{G}, s)$ returns possible layouts applying s on $\mathcal{G}$.
    for layout $l \in \textsc{Layout}(\mathcal{G}, s)$ **do**
      // $\textsc{Apply}(\mathcal{G}, s, l)$ applies s on $\mathcal{G}$ with layout l.
      $\mathcal{G}' = \textsc{Apply}(\mathcal{G}, s, l)$
      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 $\mathcal{G}_{opt}$

Cost-based backtracking algorithm from MetaFlow
(Jia, 2019)
=
Priority Queue
+
Nested for loop over all substitutions/data layouts

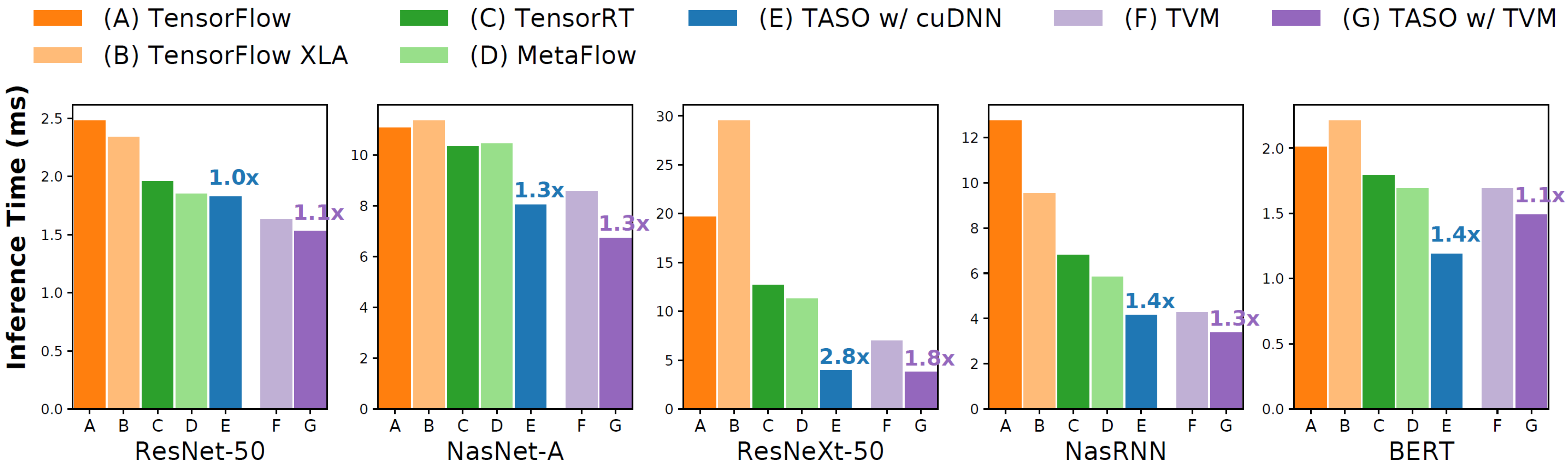


Figure 7. End-to-end inference performance comparison among existing DNN frameworks and TASO. The experiments were performed using a single inference sample, and all numbers were measured by averaging 1,000 runs on a NVIDIA V100 GPU. We evaluated the TASO's performance with both the cuDNN and TVM backends.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

UNIVERSITY OF CAMBRIDGE

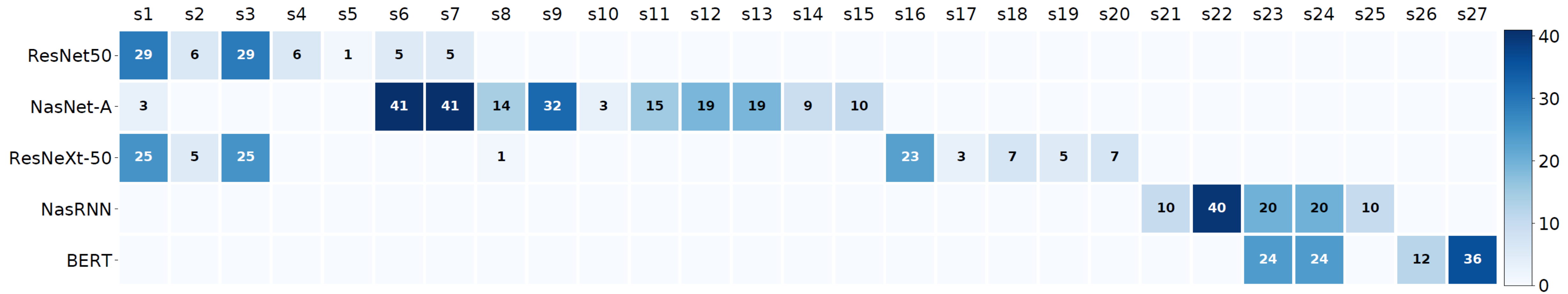


Figure 10. A heat map of how often the verified substitutions are used to optimize the five DNN architectures. Only substitutions used in at least one DNN are listed. For each architecture, the number indicates how many times a substitution is used by TASO to obtain the optimized graph.

UNIVERSITY OF CAMBRIDGE

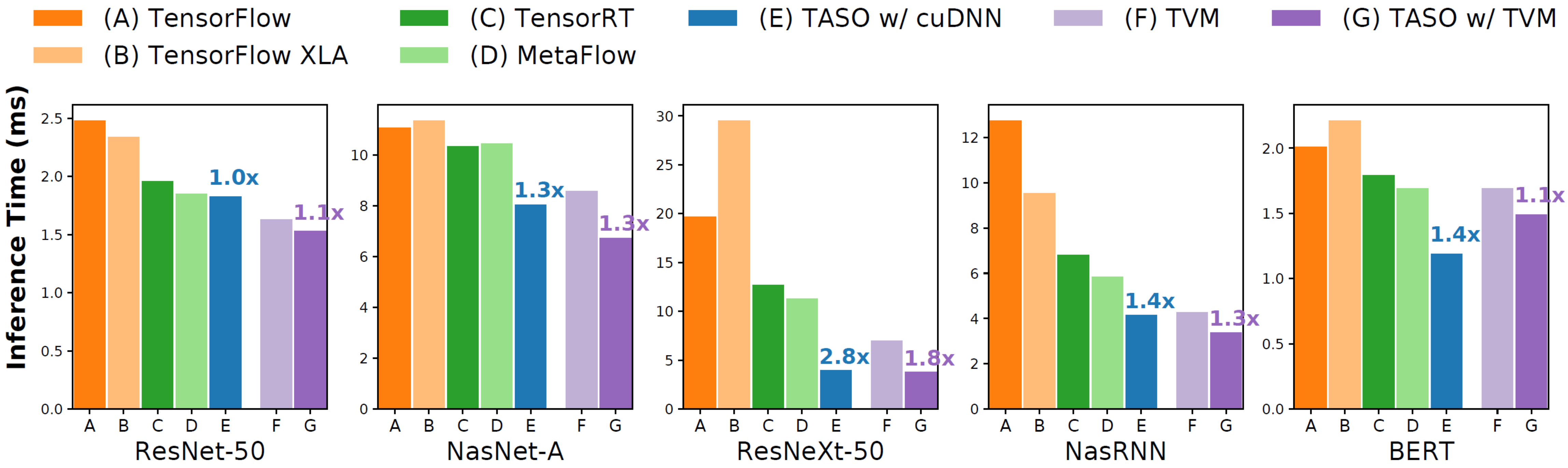


Figure 7. End-to-end inference performance comparison among existing DNN frameworks and TASO. The experiments were performed using a single inference sample, and all numbers were measured by averaging 1,000 runs on a NVIDIA V100 GPU. We evaluated the TASO's performance with both the cuDNN and TVM backends.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

UNIVERSITY OF CAMBRIDGE

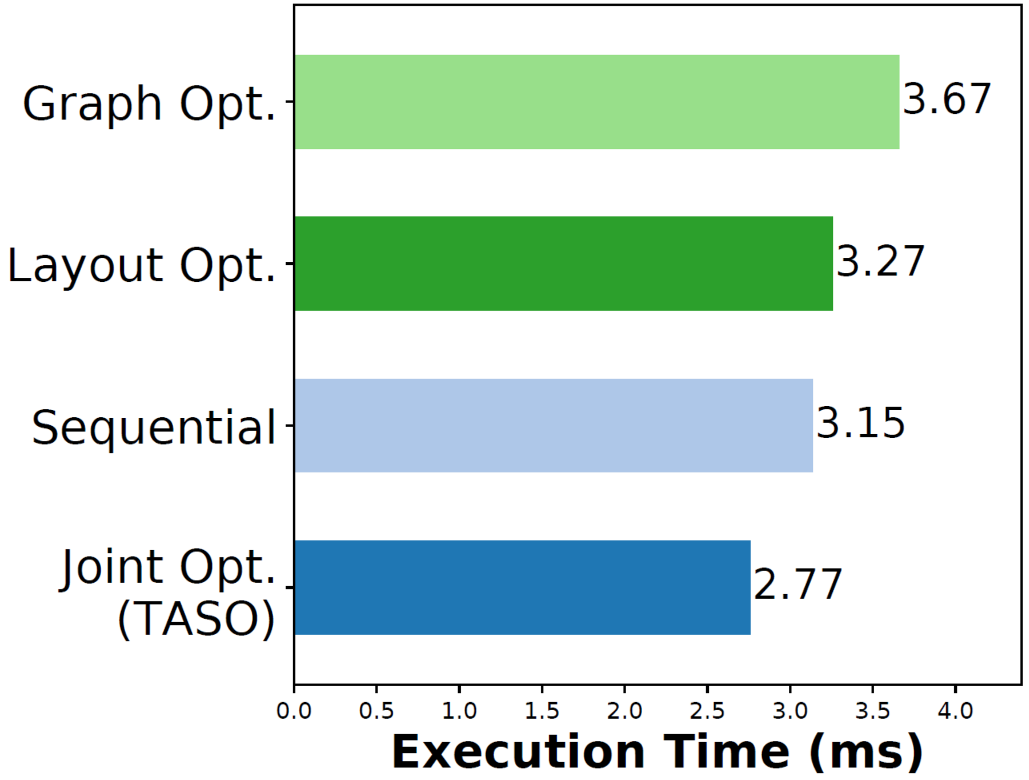


Figure 12. End-to-end inference performance comparison on BERT using different strategies to optimize graph substitution and data layout.

Evaluation

Con:

- 743 graph substitutions > 43 operator properties

Evaluation: 743 substitutions > 43 operator properties

All substitutions found by TASO are valid given the user provided properties.

Evaluation: 743 substitutions > 43 operator properties

All substitutions found by TASO are valid given the user provided properties.

→ There is a sequence of applications of user provided properties proving the correctness of the substitution.

Evaluation: 743 substitutions > 43 operator properties

All substitutions found by TASO are valid given the user provided properties.

→ There is a sequence of applications of user provided properties proving the correctness of the substitution.

Each user provided property is equivalent to a substitution.

Evaluation: 743 substitutions > 43 operator properties

All substitutions found by TASO are valid given the user provided properties.

→ There is a sequence of applications of user provided properties proving the correctness of the substitution.

Each user provided property is equivalent to a substitution.

→ **All substitutions are sequences of substitutions provided by the user.**

UNIVERSITY OF CAMBRIDGE

Evaluation

Con:

- 743 graph substitutions > 43 operator properties
- Why prune all redundant substitutions?

Evaluation

Con:

- 743 graph substitutions > 43 operator properties
- Why prune all redundant substitutions?
- Cost measure of joint optimizer seems oversimplified

Evaluation

Con:

- 743 graph substitutions > 43 operator properties
- Why prune all redundant substitutions?
- Cost measure of joint optimizer seems oversimplified

Pro:

- Lots of real-world considerations

Evaluation

Con:

- 743 graph substitutions > 43 operator properties
- Why prune all redundant substitutions?
- Cost measure of joint optimizer seems oversimplified

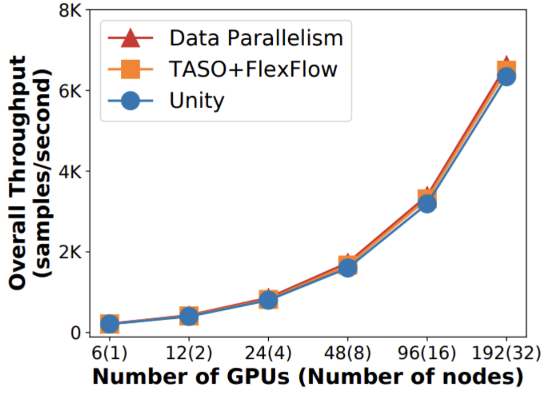
Pro:

- Lots of real-world considerations
- Many individual influential contributions

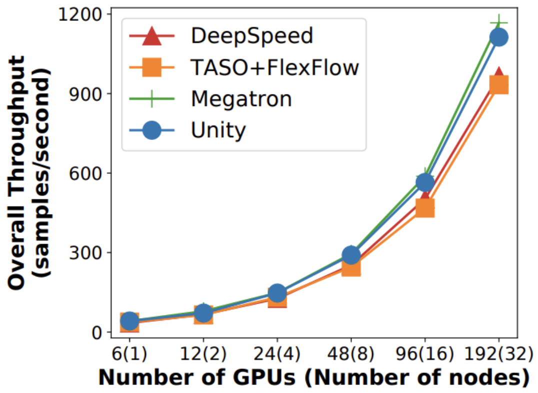
Impact and Follow Up Work

- PET: non-fully equivalent graph substitutions
  (Wang, 2021)
- Equality Saturation: improves joint optimization algorithm
  (Yang, 2021)
- Unity: jointly optimizes graph substitutions, data layout, and parallelism
  (Unger, 2022)

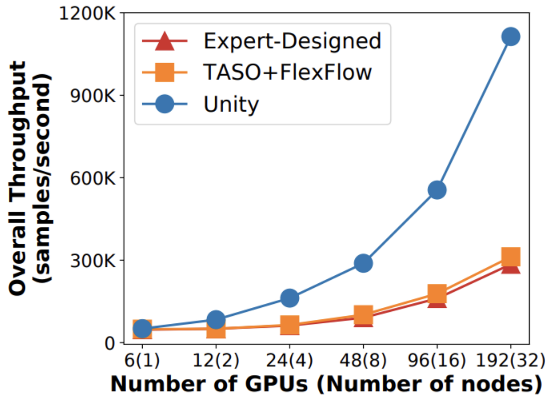
UNIVERSITY OF CAMBRIDGE



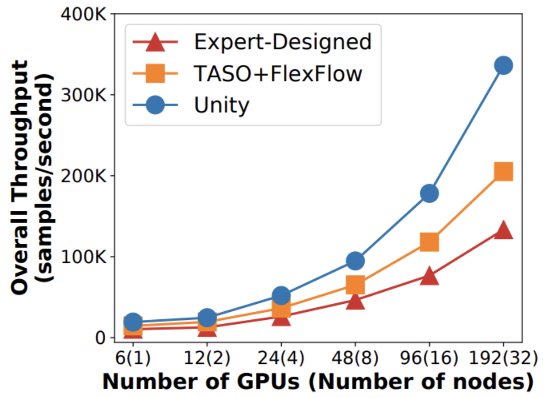
(a) ResNeXt-50.



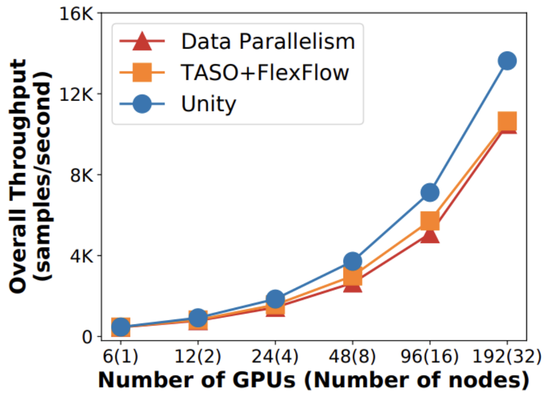
(b) BERT-Large.



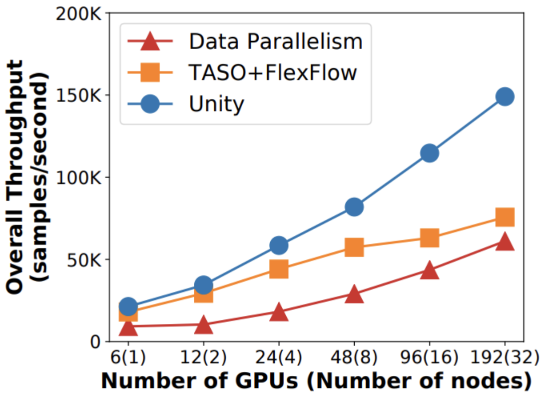
(c) DLRM.



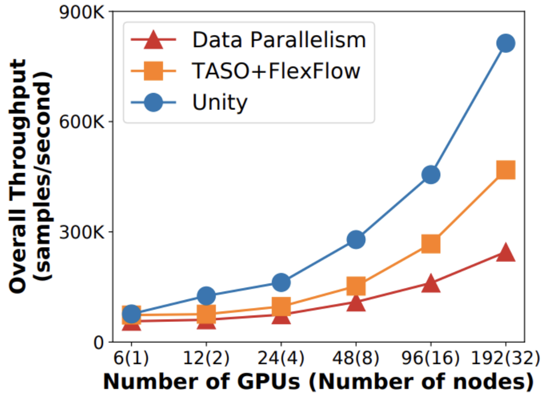
(d) CANDLE-Uno.



(e) Inception-v3.



(f) MLP.



(g) XDL.

(Unger, 2022)

UNIVERSITY OF CAMBRIDGE

References

- Bansal, Sorav, and Alex Aiken. "Automatic generation of peephole superoptimizers." ACM SIGARCH Computer Architecture News 34.5 (2006): 394-403.
- Jia, Zhihao, et al. "Optimizing DNN computation with relaxed graph substitutions." Proceedings of Machine Learning and Systems 1 (2019): 27-39.

UNIVERSITY OF CAMBRIDGE

References

- Jia, Zhihao, et al. "TASO: optimizing deep learning computation with automatic generation of graph substitutions." Proceedings of the 27th ACM Symposium on Operating Systems Principles. 2019.
- Unger, Colin, et al. "Unity: Accelerating {DNN} Training Through Joint Optimization of Algebraic Transformations and Parallelization." 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). 2022.

UNIVERSITY OF CAMBRIDGE

References

- Wang, Haojie, et al. "{PET}: Optimizing Tensor Programs with Partially Equivalent Transformations and Automated Corrections." 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21). 2021.
- Yang, Yichen, et al. "Equality saturation for tensor graph superoptimization." Proceedings of Machine Learning and Systems 3 (2021): 255-268.

UNIVERSITY OF CAMBRIDGE

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

Jia, Z., Padon, O., Thomas, J., Warszawski, T., Zaharia, M., & Aiken, A. (2019, October). In Proceedings of the 27th ACM Symposium on Operating Systems Principles (pp. 47-62).

R244 Large-scale data processing and optimisation
Presentation by Martin Graf on 16/11/2022