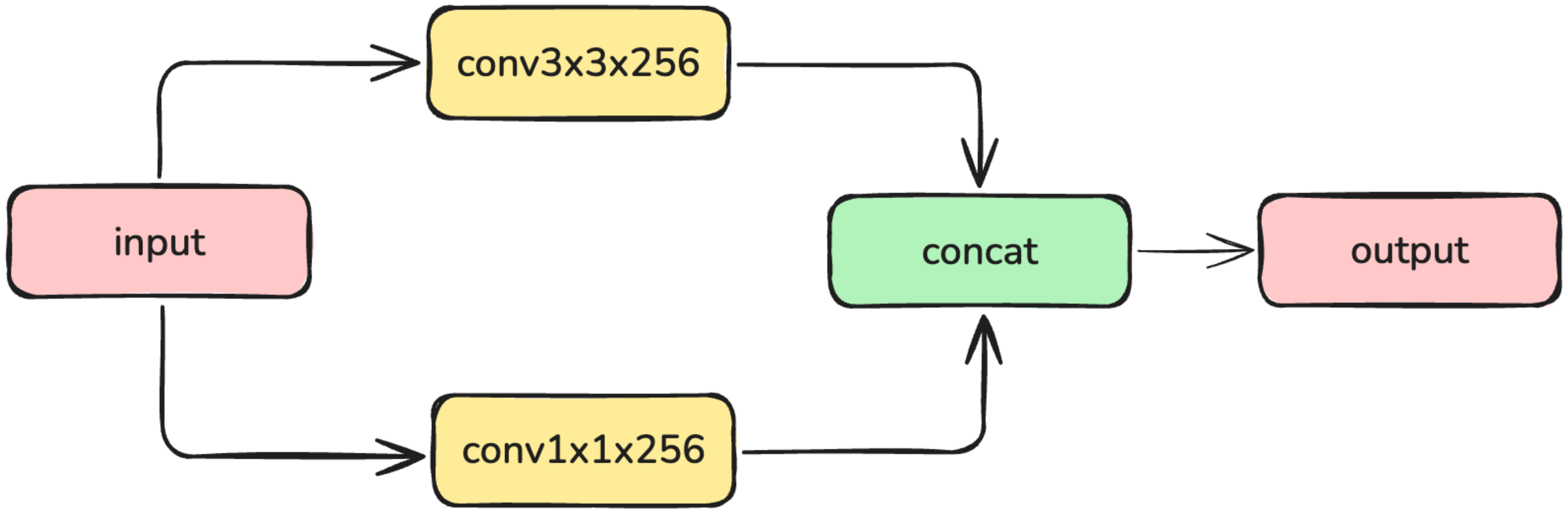
EinNet: Optimizing Tensor Programs with Derivation-Based Transformations

Zheng et al.

Sidharrth Nagappan
University of Cambridge
sn666@cam.ac.uk

Why?

- DNNs are Directed Acyclic Graphs (DAGs)
- DNNs made up of tensor operators, matrix multiplications, etc.
- How to optimise these graphs to run as efficiently as possible on different hardware?



Existing Approaches

Operator-Level Optimisers

Optimize tensor operators via schedule search

Separate computation definition from execution plan

tvm

Ansor

Graph-Level Optimizers

Use superoptimization to find graph transformations

Enumerate possible subgraphs over predefined operators

jiazhihao/**TASO**

The Tensor Algebra SuperOptimizer for Deep Learning

thu-pacman/**PET**

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

Limitations

Operator-Level Optimisers

- Optimize tensor operators via schedule search
- Separate computation definition from execution plan

Graph-Level Optimizers

- Use superoptimization to find graph transformations
- Enumerate possible subgraphs over predefined operators

Restricted to **Predefined Operator Representable (POR) Transformations**

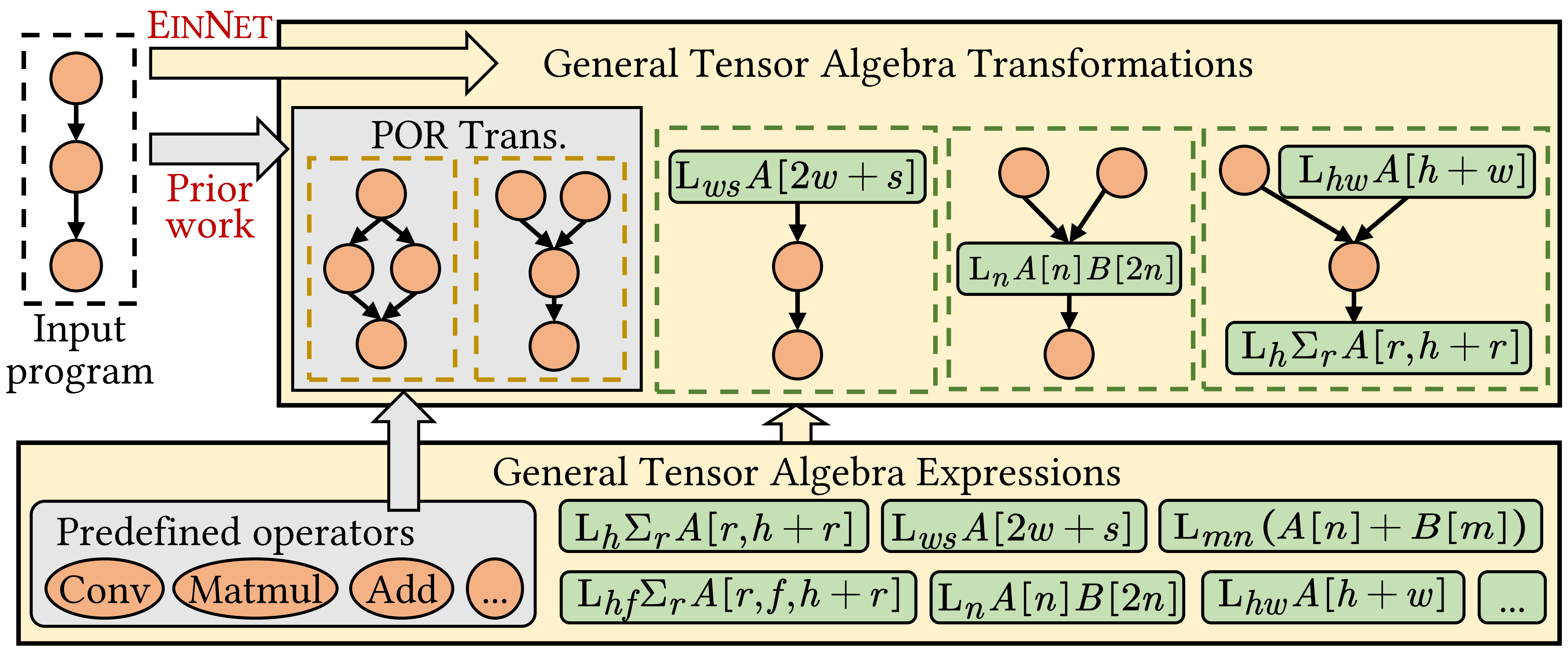
- *Can only rearrange or combine existing operators like convolution or matrix multiplication*

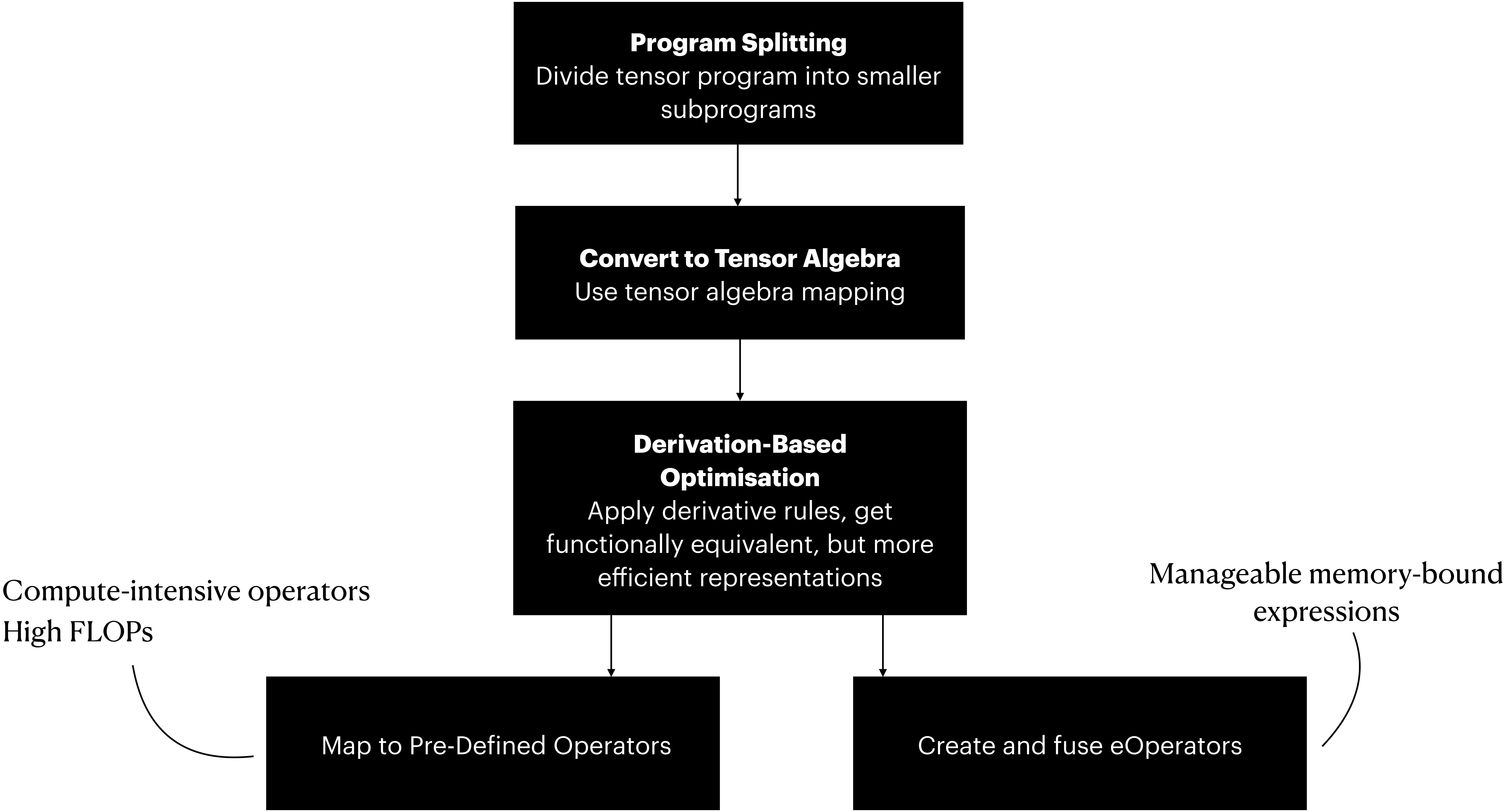
Can't optimise transformations requiring new or custom operators

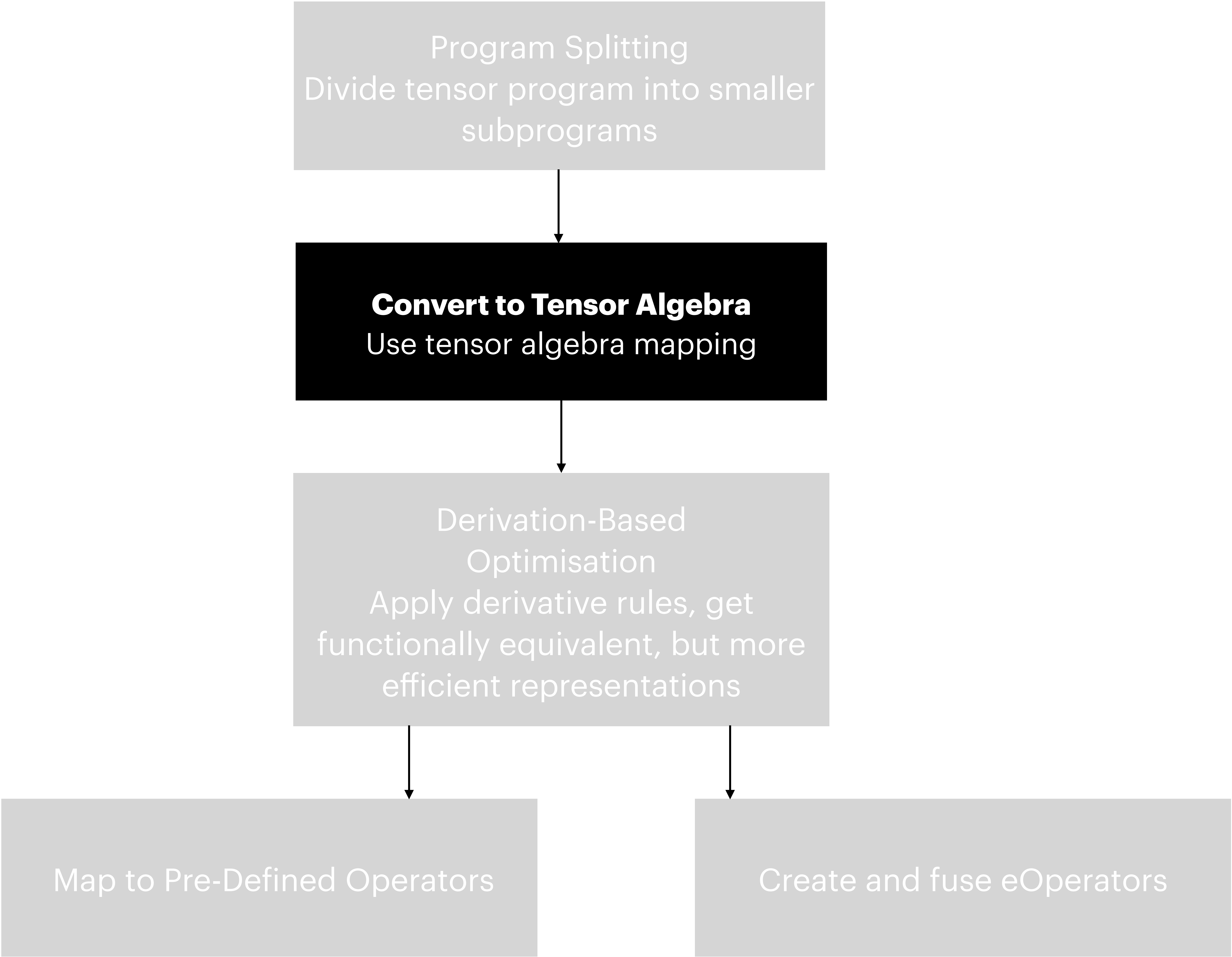
Can’t modify computation semantics

Solution

- Break down tensor computations into general tensor algebra instead of pre-defined operators
- **Derivation-Based Transformations** Apply mathematical derivation rules to tensor algebra expressions
- **Automatic Operator Creation** Generate new operators (eOperators) as needed.




Program Splitting
Divide tensor program into smaller subprograms
Convert to Tensor Algebra
Use tensor algebra mapping
Derivation-Based Optimisation
Apply derivative rules, get functionally equivalent, but more efficient representations
Compute-intensive operators
High FLOPs
Manageable memory-bound expressions
Map to Pre-Defined Operators
Create and fuse eOperators


Program Splitting
Divide tensor program into smaller subprograms
Convert to Tensor Algebra
Use tensor algebra mapping
Derivation-Based Optimisation
Apply derivative rules, get functionally equivalent, but more efficient representations
Map to Pre-Defined Operators
Create and fuse eOperators

What is Tensor Algebra

Linear Algebra Rules That We Learn

Scalars, Vectors, Matrices

$\downarrow$ Apply Rules

Higher Dimensional Data

Tensor Algebra

Traversal Notation *L*

Iteration over output tensor dimensions

$$L_{i=0}^{N-1}$$

Summation Notation

Reduction (e.g. sum) over dimensions

$$\sum_{k=0}^{K-1}$$

Matrix Multiplication

$$C[i,j] = \sum_{k=0}^{K-1} A[i,k] \times B[k,j]$$

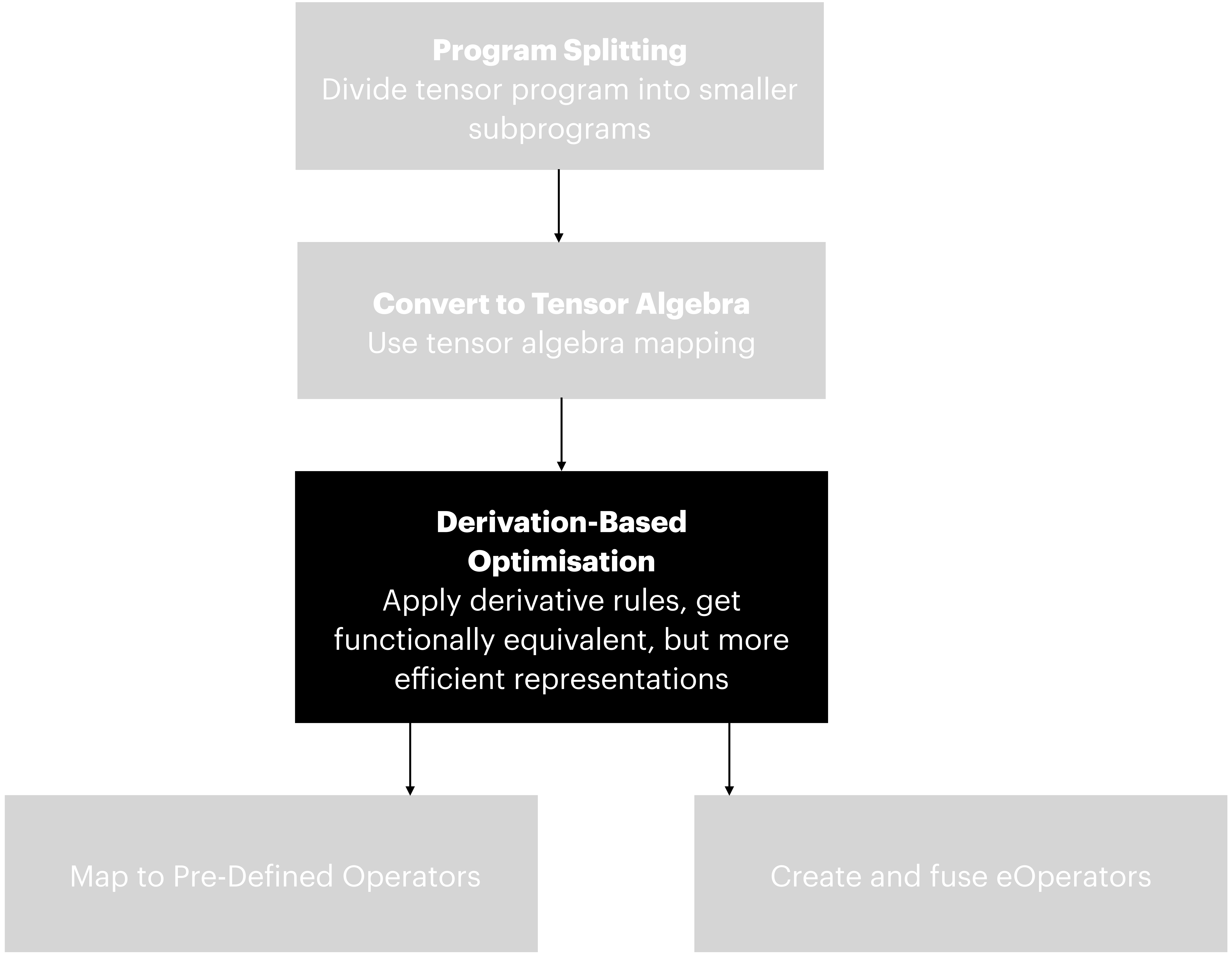
Standard Form

$$\downarrow$$

$$L_{i=0}^{M-1}\left(L_{j=0}^{N-1}\left(\sum_{k=0}^{K-1} A[i,k] \times B[k,j]\right)\right)$$

Tensor Algebra Form

Outer loops followed by inner summation


Program Splitting
Divide tensor program into smaller subprograms
Convert to Tensor Algebra
Use tensor algebra mapping
Derivation-Based Optimisation
Apply derivative rules, get functionally equivalent, but more efficient representations
Map to Pre-Defined Operators
Create and fuse eOperators

Derivation Rules

Transform tensor algebra expressions into equivalent, potentially more efficient forms.

Intra-Expression Derivation

Summation Splitting

$$\sum_{x=0}^{X-1}\sum_{y=0}^{Y-1} f(x,y) = \sum_{x=0}^{X-1}\left(\sum_{y=0}^{Y-1} f(x,y)\right)$$

Splits a summation into nested summations for partial computation and reuse.

Traversal Merging

$$L_x L_y f(x,y) \rightarrow L_{(x,y)} f(x,y)$$

Combines two separate traversals into a single traversal

Variable Substitution

$$f(h+r, w+s) \rightarrow f(t,s)$$

Transforms indices to simplify expressions

Boundary Tightening

$$L_{i=a-k}^{b+k} f(i) \rightarrow L_{i=a}^{b} f(i)$$

Restricts iteration ranges to exclude unnecessary computations

Inter-Expression Derivation

Expression Splitting

Divide an expression into independent parts

Expression Merging

Combine independent expressions into one.

Expression Fusion

Fuse dependent expressions to reduce overhead.

Program Splitting
Divide tensor program into smaller subprograms

Convert to Tensor Algebra
Use tensor algebra mapping

Derivation-Based Optimisation
Apply derivative rules, get functionally equivalent, but more efficient representations

Might create an eOperator that efficiently adds offsets to a matrix — not usually found in standard libraries

Use TVM Kernel Generator

Map to Pre-Defined Operators

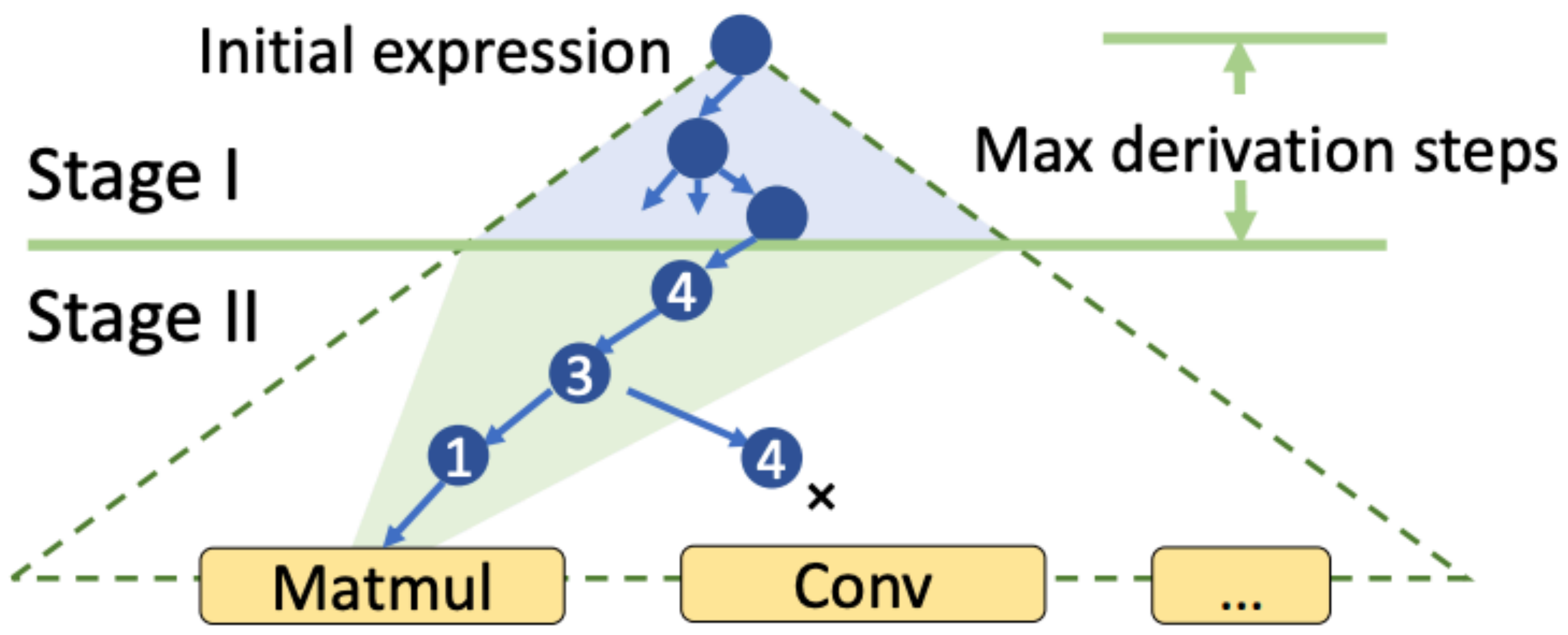
Create and fuse eOperators

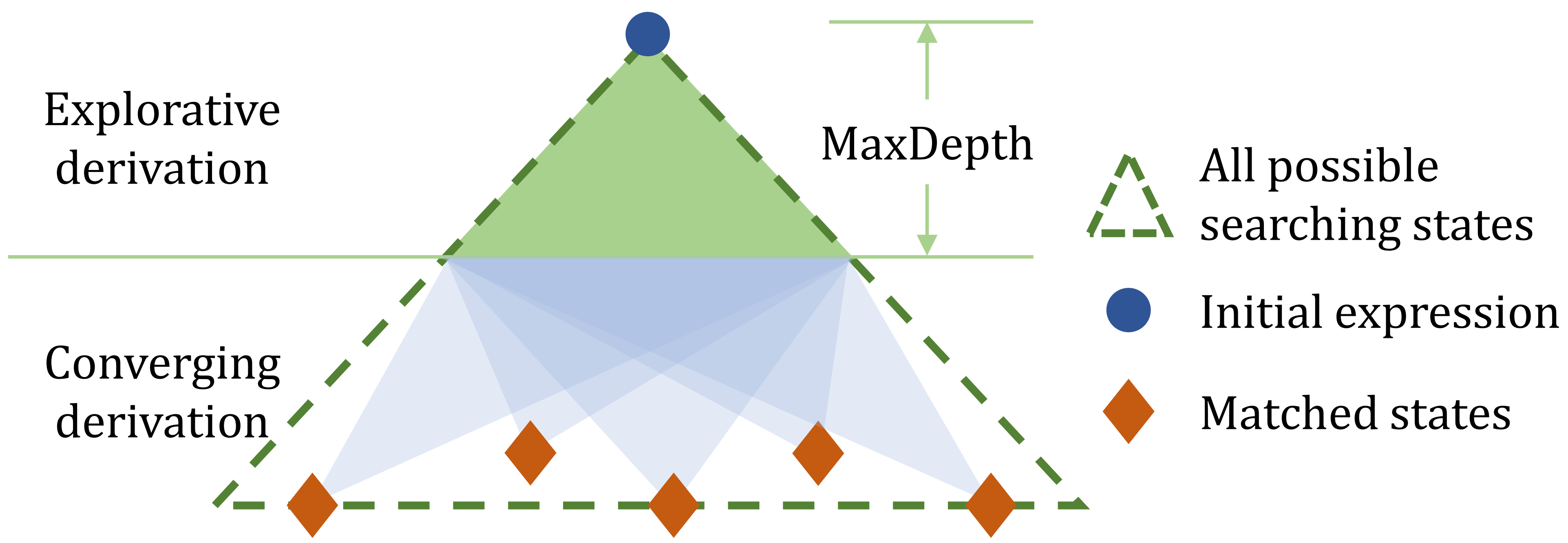
HUGE Search Space

Distance-Guided Search and Redundancy Pruning

Guide the search process towards expressions that are likely to be mappable to existing highly-optimized operators in libraries like cuDNN or cuBLAS

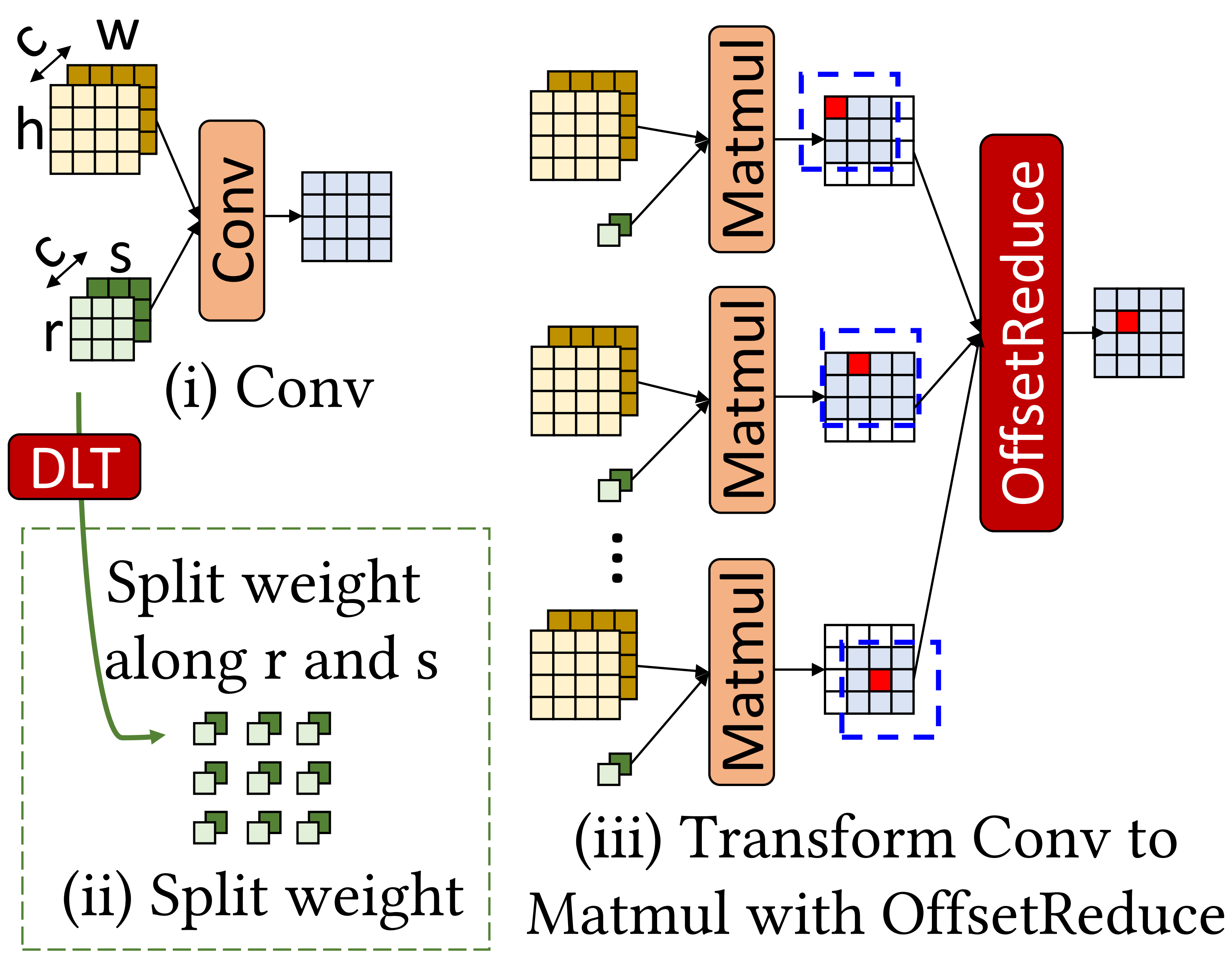
Expression Distance - difference between a given expression and the canonical expression of a target operator





Expression Distance - difference between a given expression and the canonical expression of a target operator

Optimising a 3 x 3 Convolution



$$O[n,h,w,f] = \sum_{c=0}^{C-1} \sum_{r=0}^{2} \sum_{s=0}^{2} I[n,h+r,w+s,c] \times K[r,s,f,c]$$

3 x 3 Convolution

$$\sum_{r=0}^{2} \sum_{s=0}^{2} \left(\sum_{c=0}^{C-1} I[n,h+r,w+s,c] \times K[r,s,f,c] \right)$$

Split summations *for intermediate computation use*

Optimising a **3 x 3** Convolution

$$t = h + r, \quad s' = w + s \qquad O[n, t, s', f] = \sum_{c=0}^{C-1} I[n, t, s', c] \times K[t - h, s' - w, f, c]$$

New variables *to simplify expression*

$$m = t \times W + s' \quad O'[m, f] = \sum_{k=0}^{C-1} A[m, k] \times B[k, f]$$

Merge traversals *over t and s′ into a single dimension m*

$$O'[m, n] = A[m, k] \times B[k, n] \qquad m = t \times W + s, \quad k = c, \quad n = r \times 3 + s \times 1 + f$$

Reshape tensors *to fit the matrix multiplication paradigm*

$$O[n, h, w, f] = \text{OffsetReduce}(O'[m, f], \text{offsets})$$

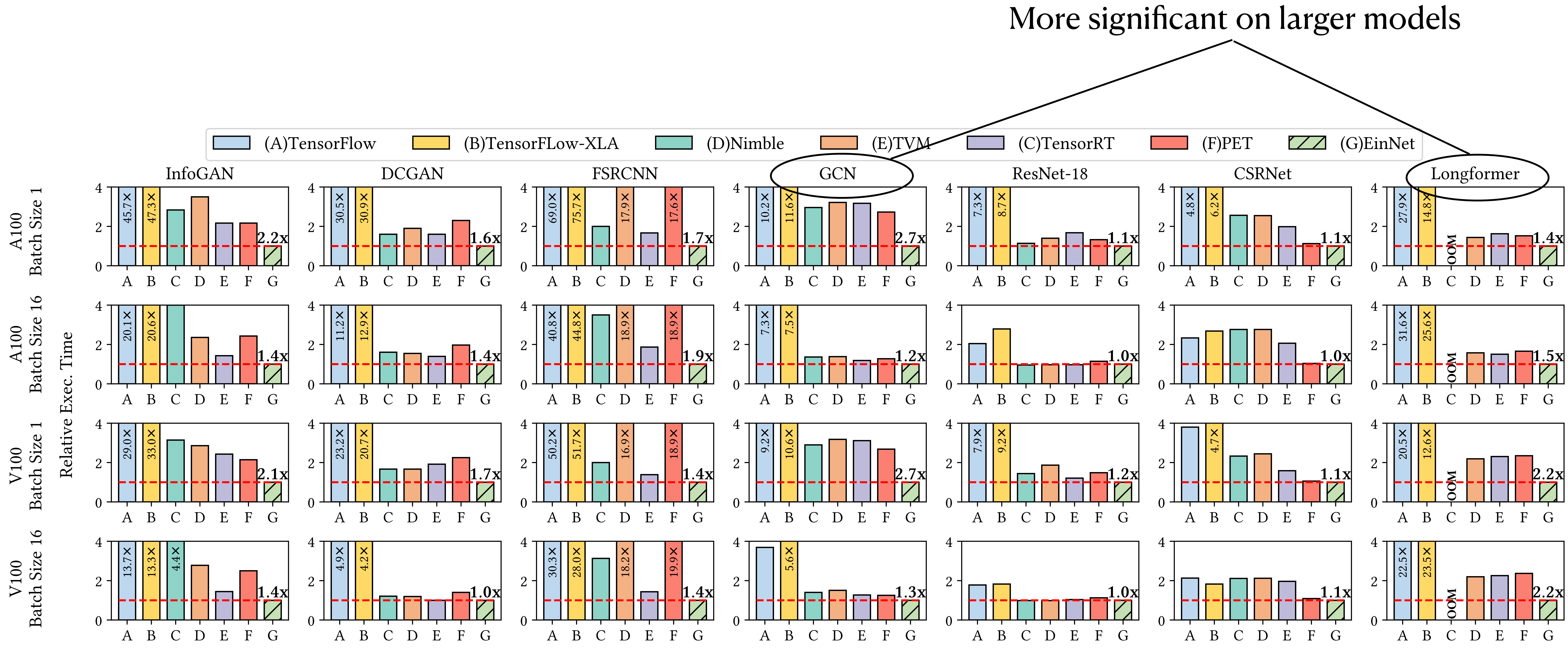
Handle offsets and intermediate computation, create an ***OffsetReduce eOperator***

2x Speed-Up compared to cuDNN

Evaluation / Results

- Run on computation graphs of InfoGAN, DCGAN, FSRCNN, GCN, ResNet-18, CSRNet, Longformer (*why Longformer?*)
- 2.72x speedup on A100 GPUs, 2.68x speed on V100 GPUs
- Shown to work with existing kernels - cuBLAS/cuDNN, AutoTVM, and Ansor
  - Certain transformations beneficial only on some backends
  - Customizing transformations for each backend is beneficial

- **For GCN** *(remember my talk 3 weeks ago)* - transformed spatially separable convolutions into faster matrix operations


More significant on larger models
(A)TensorFlow
(B)TensorFLow-XLA
(D)Nimble
(E)TVM
(C)TensorRT
(F)PET
(G)EinNet
InfoGAN
DCGAN
FSRCNN
GCN
ResNet-18
CSRNet
Longformer
A100 Batch Size 1
A100 Batch Size 16
V100 Batch Size 1
V100 Batch Size 16
Relative Exec. Time
45.7× 47.3× 2.2x
30.5× 30.9× 1.6x
69.0× 75.7× 17.9× 17.6× 1.7x
10.2× 11.6× 2.7x
7.3× 8.7× 1.1x
4.8× 6.2× 1.1x
27.9× 14.8× OOM 1.4x
20.1× 20.6× 1.4x
11.2× 12.9× 1.4x
40.8× 44.8× 18.9× 18.9× 1.9x
7.3× 7.5× 1.2x
1.0x
1.0x
31.6× 25.6× OOM 1.5x
29.0× 33.0× 2.1x
23.2× 20.7× 1.7x
50.2× 51.7× 16.9× 18.9× 1.4x
9.2× 10.6× 2.7x
7.9× 9.2× 1.2x
4.7× 1.1x
20.5× 12.6× OOM 2.2x
13.7× 13.3× 4.4× 1.4x
4.9× 4.2× 1.0x
30.3× 28.0× 18.2× 19.9× 1.4x
5.6× 1.3x
1.0x
1.1x
22.5× 23.5× OOM 2.2x

EINNET outperforms PET.

Since PET > TASO, reasonable to assume that EINNET also outperforms TASO.

Strengths

- Converting to barebones tensor algebra expands search space for optimisation
- Creates new "eOperators" that cater to very specific operations
  - First-of-its-kind
- Derivation-based optimisation is mathematically sound
- Can alter optimisation strategies based on backend (cuDNN/Ansor/etc.)
- **Distance-guided search** is an innovative way of restricting search space while ensuring the final equation is still computable with the available backend engine
- Effort was taken to test on a variety of model flavours - CNN / GNN / Transformer

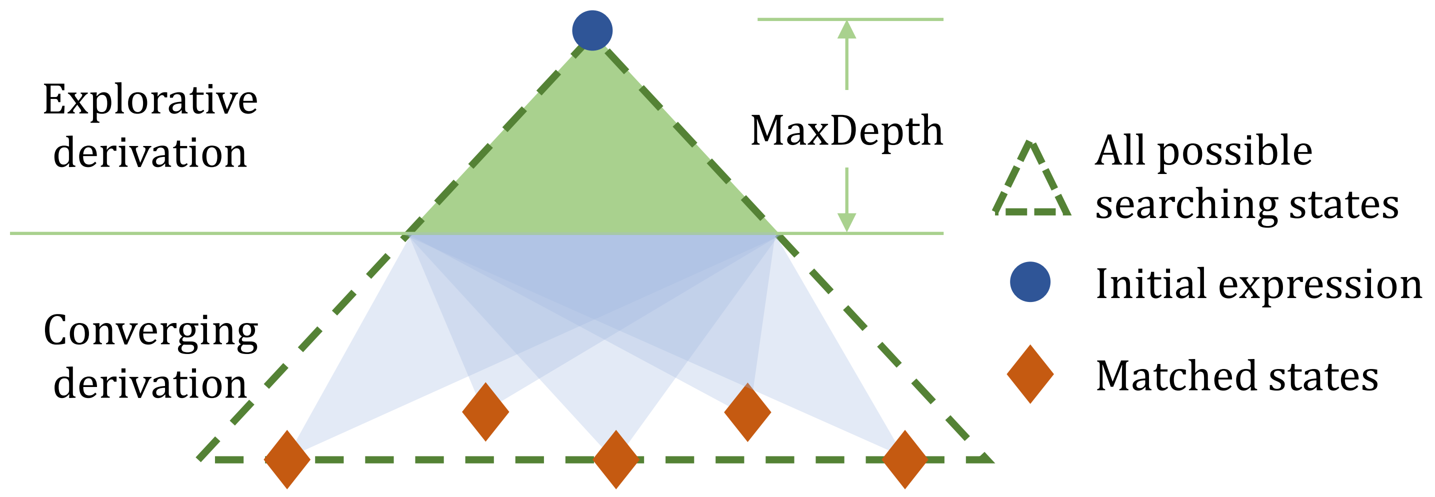
Weaknesses

- No discussion about compilation time
  - How long does EinnNet take to optimise a computation graph, compared to its competitors?

"search spends no more than two hours for most models, which depends on the number of operators contained in models"

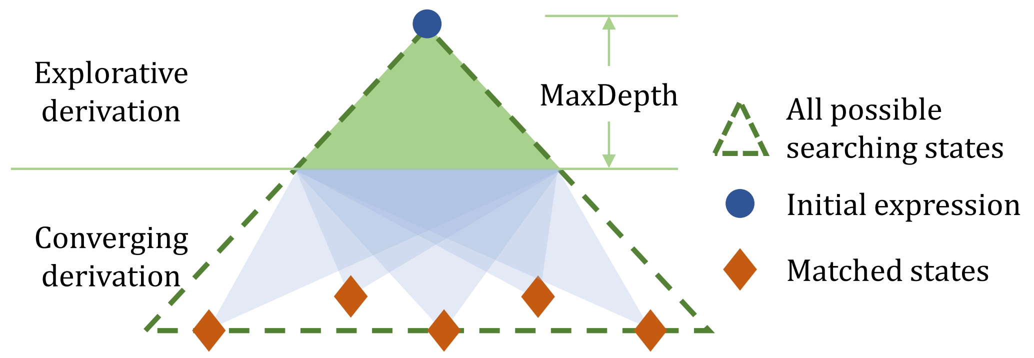
Weaknesses

- No discussion about compilation time
  - *search spends no more than two hours for most models, which depends on the number of operators contained in models*
  - How long does EinnNet take to optimise a computation graph, compared to its competitors?
- Search space is defined by heuristic of maximum search depth
  - While speedups achieved by different search depths compared, unclear how changing size of the search space actually impacts performance



Weaknesses

- No discussion about compilation time
  - *search spends no more than two hours for most models, which depends on the number of operators contained in models*
  - How long does EinnNet take to optimise a computation graph, compared to its competitors?
- Search space is defined by heuristic of maximum search depth



  - While speedups achieved by different search depths compared, unclear how changing size of the search space actually impacts performance
- How big are each of the models?
  - ResNet, Longformer, GANs come in many sizes, which were used - num model parameters not mentioned

README   Apache-2.0 license

To quickly evaluate our system, we offer a server equipped with the necessary hardware, software dependencies, and baseline frameworks. Each directory contains a `run.sh` script that generates the evaluation results on the provided server. If you wish to run these scripts on a different environment, you should update the environment variables within them.

And this is the README! 🤦

https://github.com/zhengly123/OSDI23-EinNet-AE

Future Work

- Adaptive heuristics of search space
  - See how changing depth affects performance
- Analyse time taken for optimisation as model size and complexity changes
- See whether these methods scale to TPUs as well
- Theoretically prove that optimisation is efficient on other common operations like pooling / batch normalisation
- Distributed tensor algebra?
  - How would it work when model layers are split and you have distributed tensors?

Thank you, questions?