

Tensor Program Optimization with Probabilistic Programs

Junru Shao, Xiyu Zhou, Siyuan Feng, Bohan Hou,
Ruihang Lai, Hongyi Jin, Wuwei Lin, Masahiro
Masuda, Cody Hao Yu, Tianqi Chen

Presenter: Qianyi Liu





Context

- Current deep learning frameworks relies on **vendor-specific operator libraries** (e.g. CuDNN) to optimise deployment of neural networks on hardware
 - Choose from logically equivalent programs with significantly different performance
 - Huge engineering effort + domain knowledge

- **Automatic program optimisation** — machine learning
 - Two crucial components
 - A search space (loop transformation, vectorisation, threading patterns, and hardware acceleration)
 - Learning-based search algorithms

A typical workflow for automatic tensor program optimization

Key elements in automatic tensor program optimization



Examples

```
for i in range(1024):
  for j in range(1024):
    for k in range(1024):
      C[i, j] += A[i, k] * B[j, k]
```

```
for i0, j0 in grid(16, 8):
  for i1, j1 in grid(8, 16):
    for k0 in range(1024):
      for i2, j2 in grid(8, 8):
        C[...] += ...
```

```
for i0, j0 in grid(64, 8):
  for i1, j1 in grid(4, 32):
    for k0 in range(64):
      for i2, j2 in grid(4, 4):
        for k1 in range(16):
          C[...] += ...
```

...



MetaSchedule

- The search space itself fundamentally limits the best possible performance search algorithms can get.
- Defining the search space for a wide range of tensor programs is challenging
 - $S(e_0)$ is highly dependent on e_0
 - Differs in different hardware domains
 - Hardware and model settings evolve -> update $S(e_0)$
- This paper aims to provide a programmable abstraction to construct $S(e_0)$ in a composable and modular way
- **MetaSchedule**: a domain-specific probabilistic programming language abstraction to construct a search space of tensor programs

Stochastic Search Space Construction

- Parameterize an optimisation search space by the initial program followed by a sequence of transformations on the program
- Allow further parameterization of each transformation step with random variables, drawn from sampling distributions

Parameterization

Initial tensor program: e_0

```
① for i in range(1024):  
    B[i] = ReLU(A[i])
```

Transformation	Parameterization
① Split	$i, 32, 8, 4$
② Parallelize	i_0
③ Vectorize	i_2

Equivalent Programs Induced by Parameterized Transformation

Equivalent intermediate program: e_1

```
for i0 in range(32): ②  
    for i1 in range(8):  
        for i2 in range(4): ③  
            i = i0 * 32 + i1 * 4 + i2  
            B[i] = ReLU(A[i])
```

parameterized by: $e_0 + ①$

Equivalent optimized program: e^*

```
parallel for i0 in range(32):  
    for i1 in range(8):  
        i = i0 * 32 + i1 * 4  
        B[i : i + 4] =  
            ReLU(A[i : i + 4])
```

parameterized by: $e_0 + ①②③$

Defining stochastic transformation in MetaSchedule

Probabilistic Program

```
def Probabilistic-Program():  
    # ① Loop tiling for Dense  
     $\theta_0, \theta_1 \sim \text{Sample-Tile}(i, \text{parts}=2)$   
     $\theta_2, \theta_3 \sim \text{Sample-Tile}(j, \text{parts}=2)$   
     $i_0, i_1 = \text{Split}(i, [\theta_0, \theta_1])$   
     $j_0, j_1 = \text{Split}(j, [\theta_2, \theta_3])$   
    Reorder( $i_0, j_0, i_1, j_1$ )  
  
    # ② ReLU fusion  
     $\theta_{\text{ReLU}} \sim \text{Sample-Compute-Location}(\text{ReLU})$   
    Compute-At(ReLU,  $\theta_{\text{ReLU}}$ )
```

sampling

transformation

Stochastic Transformation

```
# Dense: Apply ①  
for i in range(512):  
    for j in range(256):  
        for k in range(16):  
            C[...] += ...  
# ReLU:  
for i' in range(512):  
    for j' in range(256):  
        D[...] = ...
```

```
# Dense:  
for  $i_0, j_0$  in grid( $\theta_0, \theta_2$ ):  
    for  $i_1, j_1$  in grid( $\theta_1, \theta_3$ ):  
        for k in range(16):  
            C[...] += ...  
# ReLU: Apply ②  
for i' in range(512):  
    for j' in range(256):  
        D[...] = ...
```

$\theta_{\text{ReLU}} \text{ is } j_1$

$\theta_{\text{ReLU}} \text{ is } j_0$

```
# Fused Dense + ReLU  
#  $\theta_{\text{ReLU}}$ : Deep fusion under  $j_1$   
for  $i_0, j_0$  in grid( $\theta_0, \theta_2$ ):  
    for  $i_1, j_1$  in grid( $\theta_1, \theta_3$ ):  
        for k in range(16):  
            C[...] += ...  
     $\theta_{\text{ReLU}} =$   
        for i', j' in grid( $\theta_4, \theta_5$ ):  
            D[...] = ...
```

```
# Fused Dense + ReLU  
#  $\theta_{\text{ReLU}}$ : Shallow fusion under  $j_0$   
for  $i_0, j_0$  in grid( $\theta_0, \theta_2$ ):  
    for  $i_1, j_1$  in grid( $\theta_1, \theta_3$ ):  
        for k in range(16):  
            C[...] += ...  
     $\theta_{\text{ReLU}} =$   
        for i', j' in grid( $\theta_4, \theta_5$ ):  
            D[...] = ...
```

Modular Search Space Composition

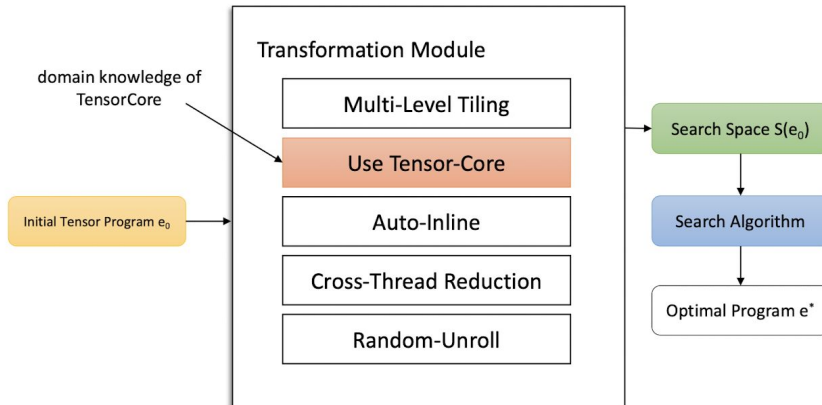
- Aim: make transformation reusable, make MetaSchedule more easy to use
- Introduce **transformation module**
 - Atomic stochastic transformation
 - Composition of program analysis, sampling as well as smaller transformations

Transformation Module

```
def Multi-Level-Tiling(loop_nest: List[Loop]):  
    tiles: List[List[Loop]] = [list() for _ in range(5)]  
    def tile_loop(loop: Loop, tile_ids: List[int]):  
        {0} = Sample-Tile(loop, parts=len(tile_ids))  
        tiled_loops = Split(loop, {0})  
        for i, tile in zip(tile_ids, tiled_loops):  
            tiles[i].append(tile)  
    for i in loop_nest:  
        if is_spatial_loop(i): tile_loop(i, [0, 1, 3])  
        elif is_reduction_loop(i): tile_loop(i, [2, 4])  
    Reorder(list_concat(tiles));
```

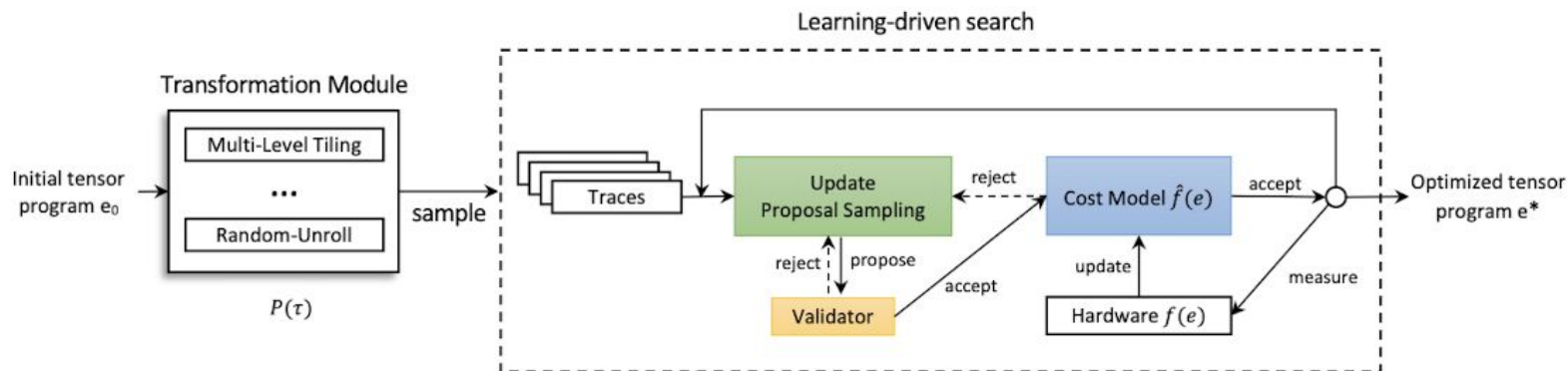
stochastic transformations

analysis



A generic learning-driven framework to find an optimized program

1. Search algorithm samples the MetaSchedule program to obtain a collection of traces
2. An evolutionary search algorithm that proposes a new variant of the trace by mutating the RV -> validator + cost model -> accept
3. Proxy cost model: a tree-boosting-based cost model – updated throughout the process

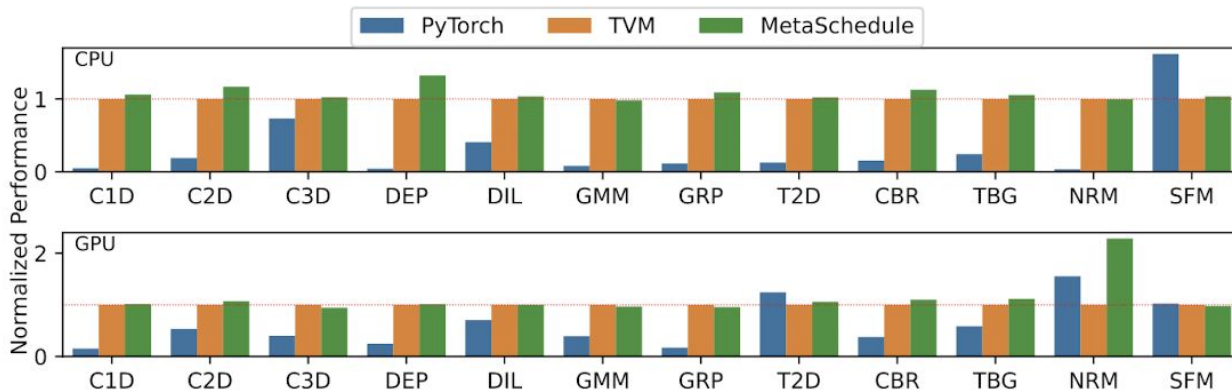




Experiment 1: Expressiveness to cover common optimisation techniques

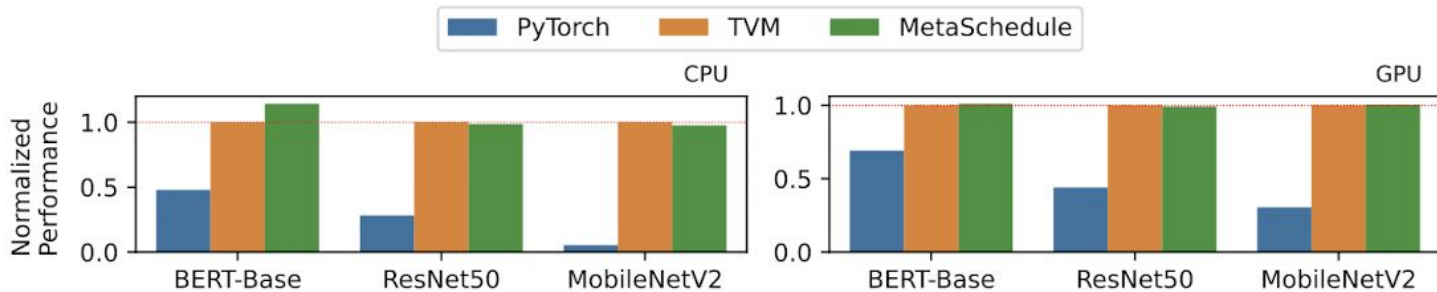
Target: a diverse set of operators and subgraphs

- MetaSchedule: our approach
- TVM (AutoTVM and Ansor) — SOTA tensor program optimisation system
- PyTorch — optimised with vendor libraries





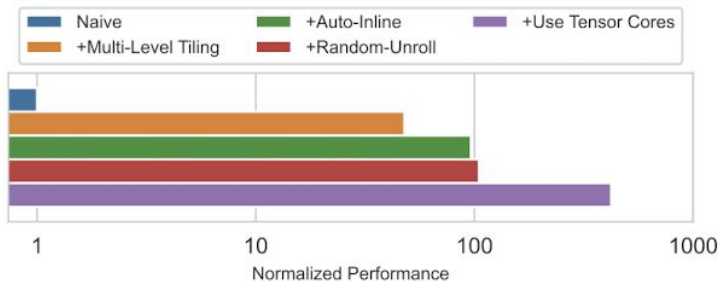
Experiment 2: optimising End-to-End deep learning models



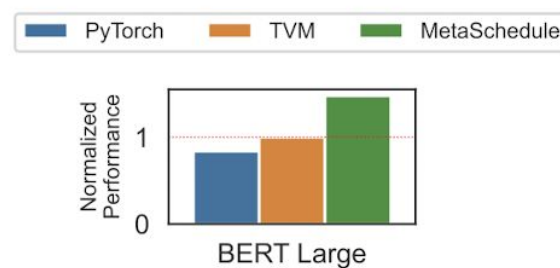
Conclusion: MetaSchedule performance is on parity with TVM, while surpassing PyTorch in all cases -> the MetaSchedule framework delivers end-to-end performance

Experiment 3: Search space composition and hardware-specific modules

- By progressively enriching the search space, the performance of optimized tensor programs consistently increases -> translate to end-to-end model performance
- Convenience of customization and composition



(a) Performance with different search spaces.



(b) BERT-Large Performance.



Takeaways

Pros:

- A novel piece of work — MetaSchedule, probabilistic programmable abstraction
- Decouples the search space construction from the search — enabling further customisation without surgical changes to the system
- A simple yet powerful generalisation of existing tensor program optimisation methods

Cons:

- Lack of further evaluation on the search space construction process and program optimisation process
- Didn't explain in detail the advantage over previous deterministic approaches using other DSLs



Discussion