15-779 Lecture 9: ML Compilers Part 2: Kernel Autotuning

Zhihao Jia

Computer Science Department

Carnegie Mellon University

Outline: Kernel Autotuning

- **triton.autotune: template + search**
- AutoTVM: template + learning-based cost model
- Ansor: sketch generation + annotation search

Autotuning in Triton

- `triton.autotune`: whenever the values of keys change, evaluate all configurations

```
@triton.autotune(configs=[
    triton.Config(kwargs={'BLOCK_SIZE': 128}, num_warps=4),
    triton.Config(kwargs={'BLOCK_SIZE': 1024}, num_warps=8),
  ],
  key=['x_size'] # the two above configs will be evaluated anytime
                 # the value of x_size changes
)
@triton.jit
def kernel(x_ptr, x_size, BLOCK_SIZE: tl.constexpr):
    ...
```

Autotuning Matrix Multiplication

```
def matmul_get_configs():
    return [triton.Config({'BLOCK_SIZE_M': BM, 'BLOCK_SIZE_N': BN, "BLOCK_SIZE_K": BK, "GROUP_SIZE_M": 8},
                          num_stages=s, num_warps=w)
        for BM in [128]
        for BN in [128, 256]
        for BK in [64, 128]
        for s in ([2, 3, 4])
        for w in [4, 8]
    ]
@triton.autotune(
    configs=matmul_get_configs(),
    key=["M", "N", "K"],
)
@triton.jit()
def matmul_kernel(a_ptr, b_ptr, c_ptr, M, N, K,
                  BLOCK_SIZE_M: tl.constexpr,
                  BLOCK_SIZE_N: tl.constexpr,
                  BLOCK_SIZE_K: tl.constexpr,
                  GROUP_SIZE_M: tl.constexpr,):
```

Python annotations: easy to use

Issue: time consuming to enumerate all configurations

Outline: Kernel Autotuning

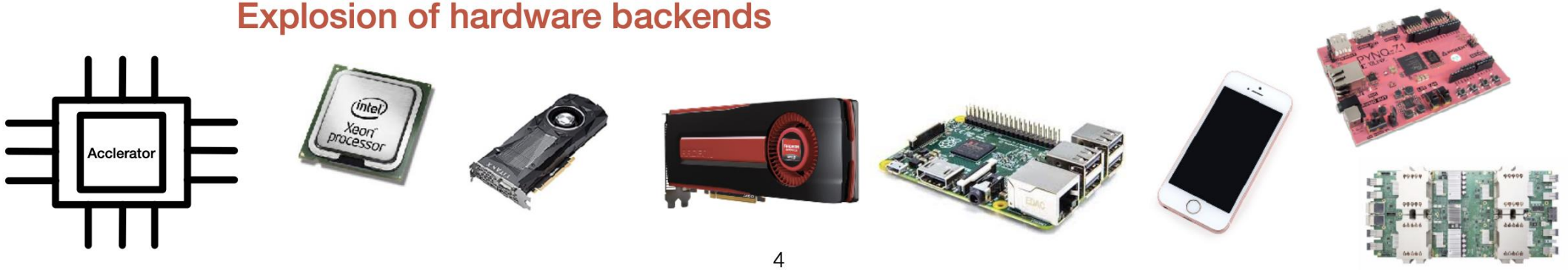
- triton.autotune: template + search
- **AutoTVM: template + learning-based cost model**
- Ansor: sketch generation + annotation search

TVM: A Learning-based Compiler for Deep Learning

Explosion of models and frameworks

Goal: efficiently deploy deep learning on modern hardware platforms

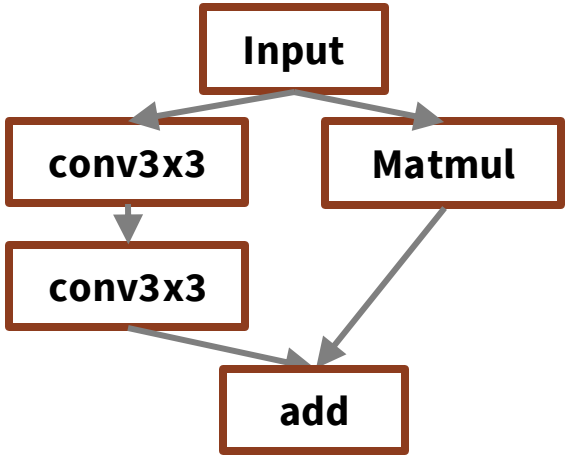
Explosion of hardware backends



4

* Slides from Tianqi Chen

Challenge: Billions of Possible Optimization Choices in the Search Space




```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Search Space of Possible Program Optimizations

Low-level Program Variants

```
inp_buffer AL[8][8], BL[8][8]
acc_buffer CL[8][8]
for yo in range(128):
  for xo in range(128):
    vdla.fill_zero(CL)
    for ko in range(128):
      vdla.dma_copy2d(AL, A[ko*8:ko*8+8][yo*8:yo*8+8])
      vdla.dma_copy2d(BL, B[ko*8:ko*8+8][xo*8:xo*8+8])
      vdla.fused_gemm8x8_add(CL, AL, BL)
    vdla.dma_copy2d(C[yo*8:yo*8+8,xo*8:xo*8+8], CL)
```

```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      for yi in range(8):
        for xi in range(8):
          for ki in range(8):
            C[yo*8+yi][xo*8+xi] +=
               A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```

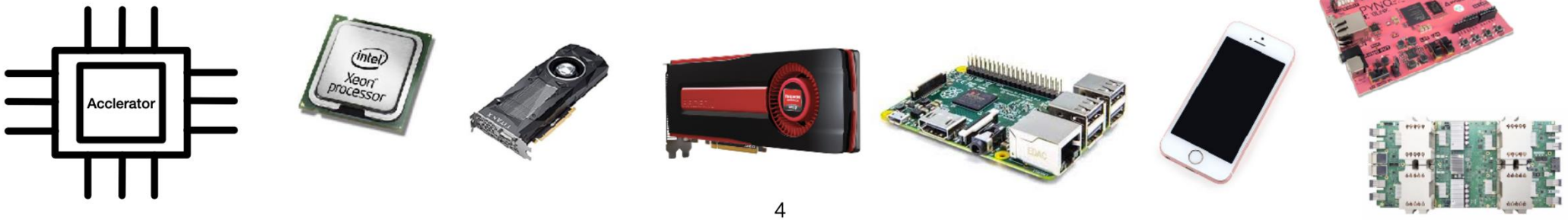
```
for y in range(1024):
  for x in range(1024):
    C[y][x] = 0
    for k in range(1024):
      C[y][x] += A[k][y] * B[k][x]
```

* Slides from Tianqi Chen

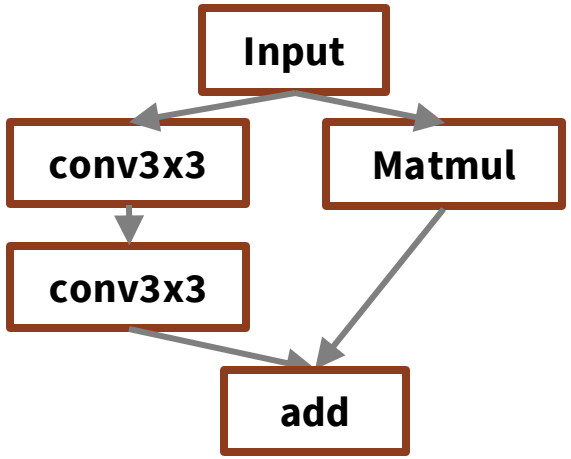
TVM: Learning-based Compiler for Deep Learning

Hardware-aware Search Space of Optimized Tensor Programs

Machine Learning based Program Optimizer



Hardware-aware Search Space



Tensor Expression Language (Specification)

```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Define search space of hardware aware mappings from expression to hardware program

Based on Halide's compute/schedule separation

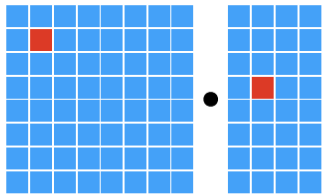
Hardware

* Slides from Tianqi Chen

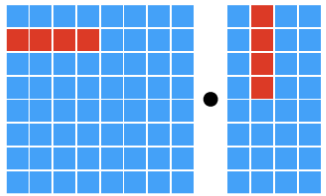
Hardware-aware Search Space

CPUs

Compute Primitives



scalar



vector

Memory Subsystem

implicitly managed

Loop Transformations

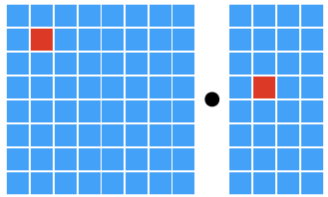
Cache Locality

Vectorization

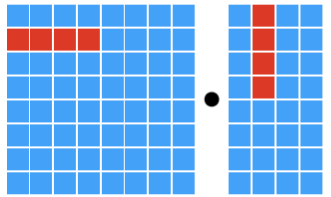
Hardware-aware Search Space

GPUs

Compute Primitives

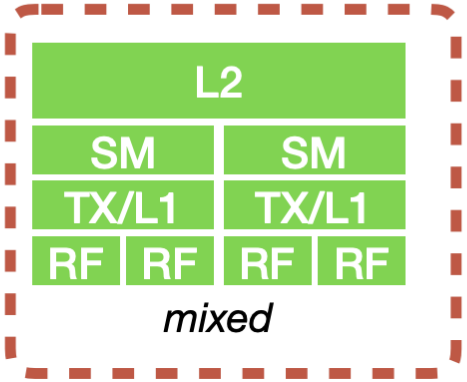


scalar



vector

Memory Subsystem



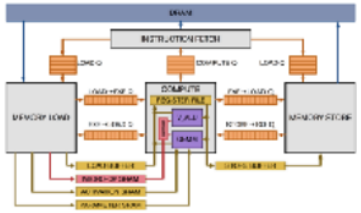
Shared memory among compute cores

Use of Shared Memory

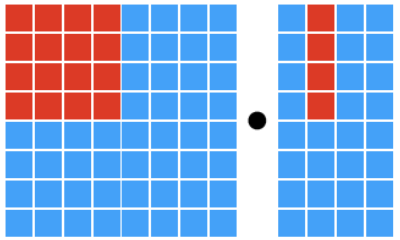
Thread Cooperation

Hardware-aware Search Space

TPU-like Specialized Accelerators

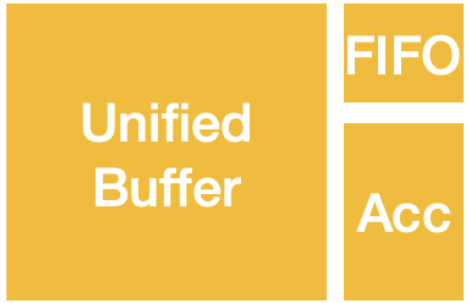


Compute Primitives



tensor

Memory Subsystem

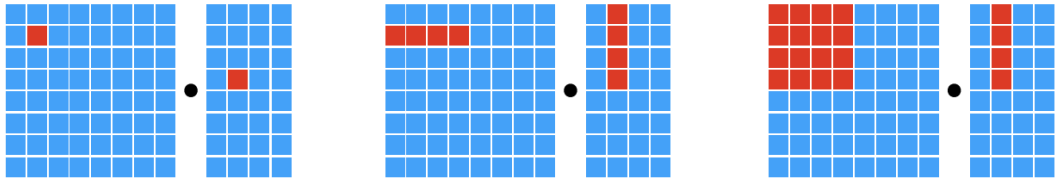


explicitly managed

Tensorization

Tensorization Challenge

Compute primitives



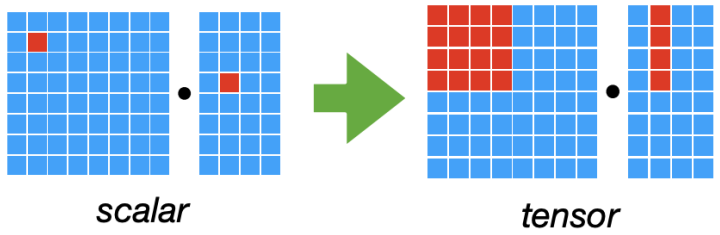
Hardware designer: declare tensor instruction interface with Tensor Expression

```
w, x = t.placeholder((8, 8)), t.placeholder((8, 8))
k = t.reduce_axis((0, 8))
y = t.compute((8, 8), lambda i, j:
              t.sum(w[i, k] * x[j, k], axis=k))

def gemm_intrin_lower(inputs, outputs):
   ww_ptr = inputs[0].access_ptr("r")
   xx_ptr = inputs[1].access_ptr("r")
   zz_ptr = outputs[0].access_ptr("w")
   compute = t.hardware_intrin("gemm8x8", ww_ptr, xx_ptr, zz_ptr)
   reset = t.hardware_intrin("fill_zero", zz_ptr)
   update = t.hardware_intrin("fuse_gemm8x8_add", ww_ptr, xx_ptr, zz_ptr)
   return compute, reset, update

gemm8x8 = t.decl_tensor_intrin(y.op, gemm_intrin_lower)
```

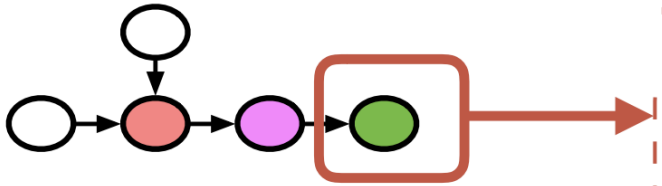
declare behavior

lowering rule to generate hardware intrinsics to carry out the computation

Tensorize: transform program to use tensor instructions



Hardware-aware Search Space



Tensor Expression Language

```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Primitives in prior work: Halide, Loopy

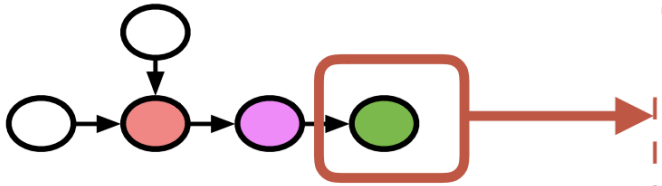
Loop Transformations | Thread Bindings | Cache Locality

New primitives for GPUs, and enable TPU-like Accelerators

Thread Cooperation | Tensorization | Latency Hiding | ...

Hardware

Hardware-aware Search Space



Tensor Expression Language

```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Billions of Possible Optimization Choices in the Search Space

- Loop Transformations
- Thread Bindings
- Cache Locality
- Thread Cooperation
- Tensorization
- Latency Hiding
- ...

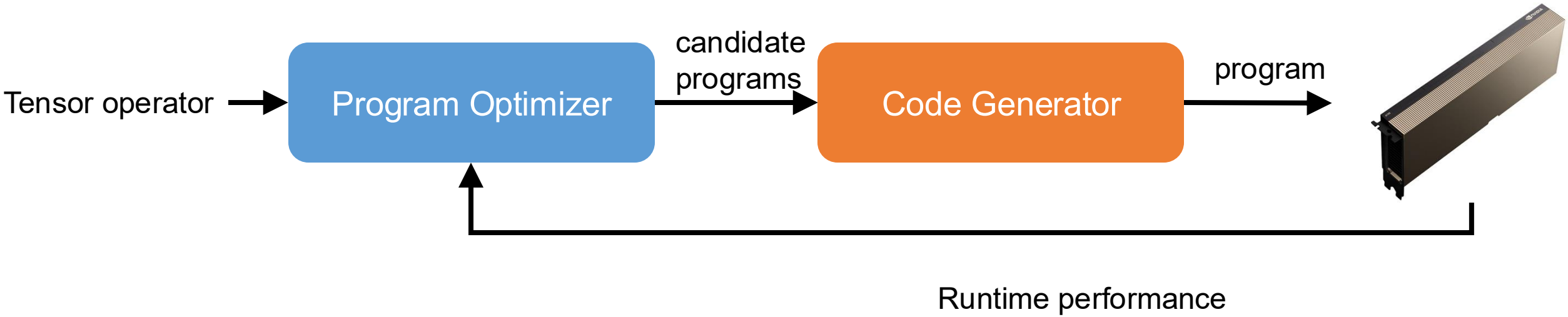
Hardware

TVM: Learning-based Compiler for Deep Learning

Hardware-aware Search Space of Optimized Tensor Programs

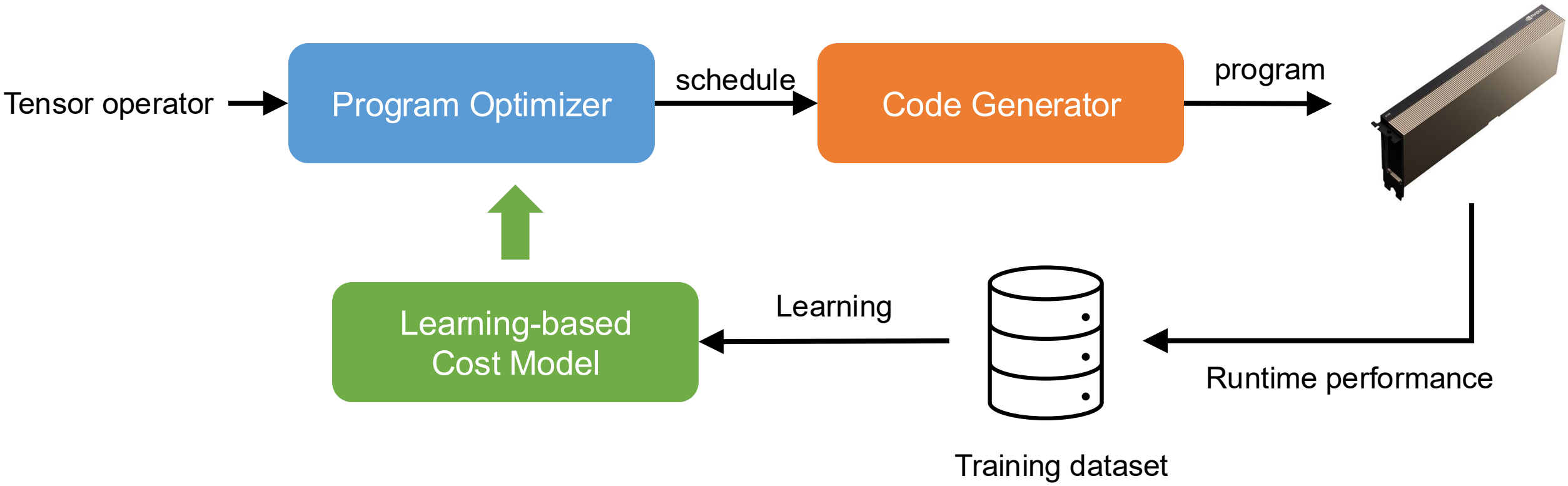
Learning based Program Optimizer

Learning-based Program Optimizer



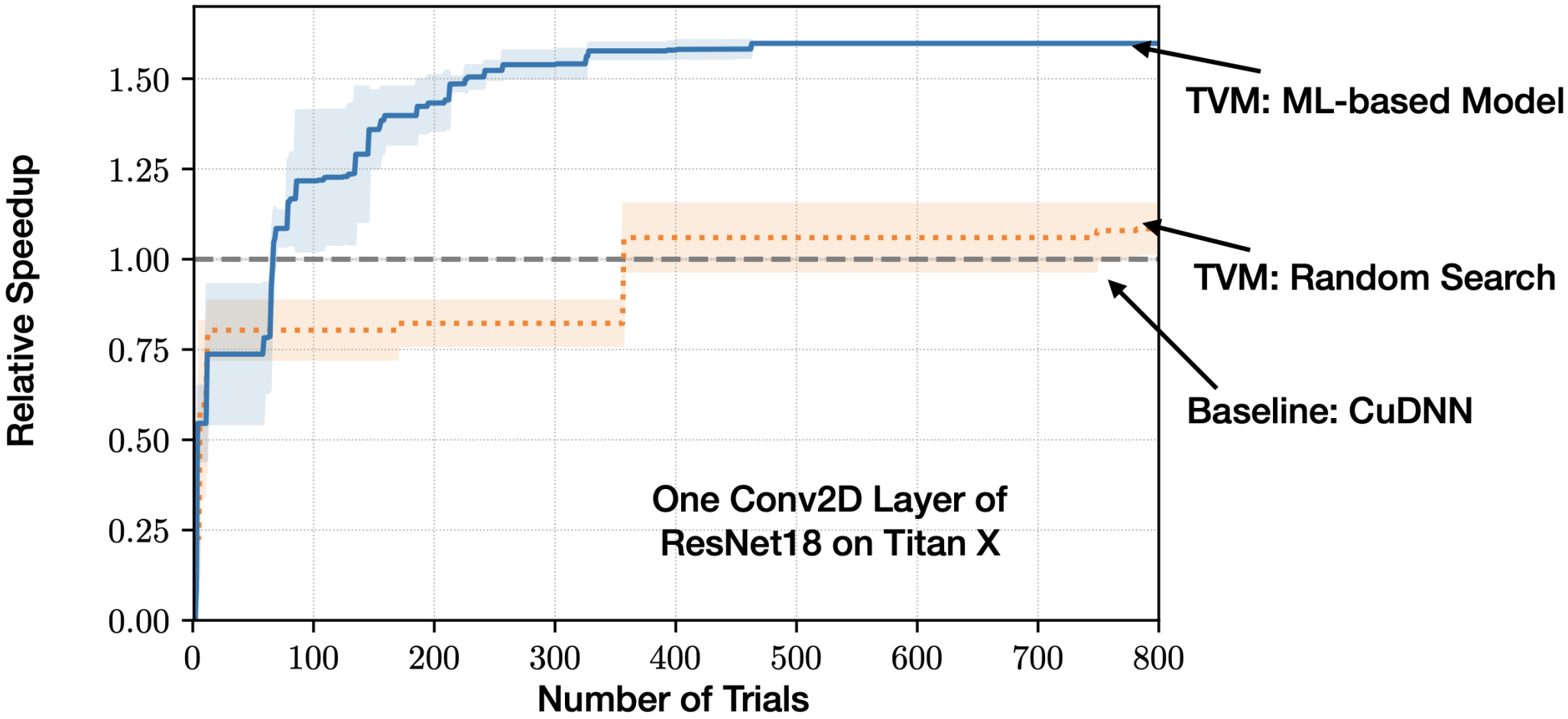
Issue: high experiment cost, each trial takes seconds

Learning-based Program Optimizer

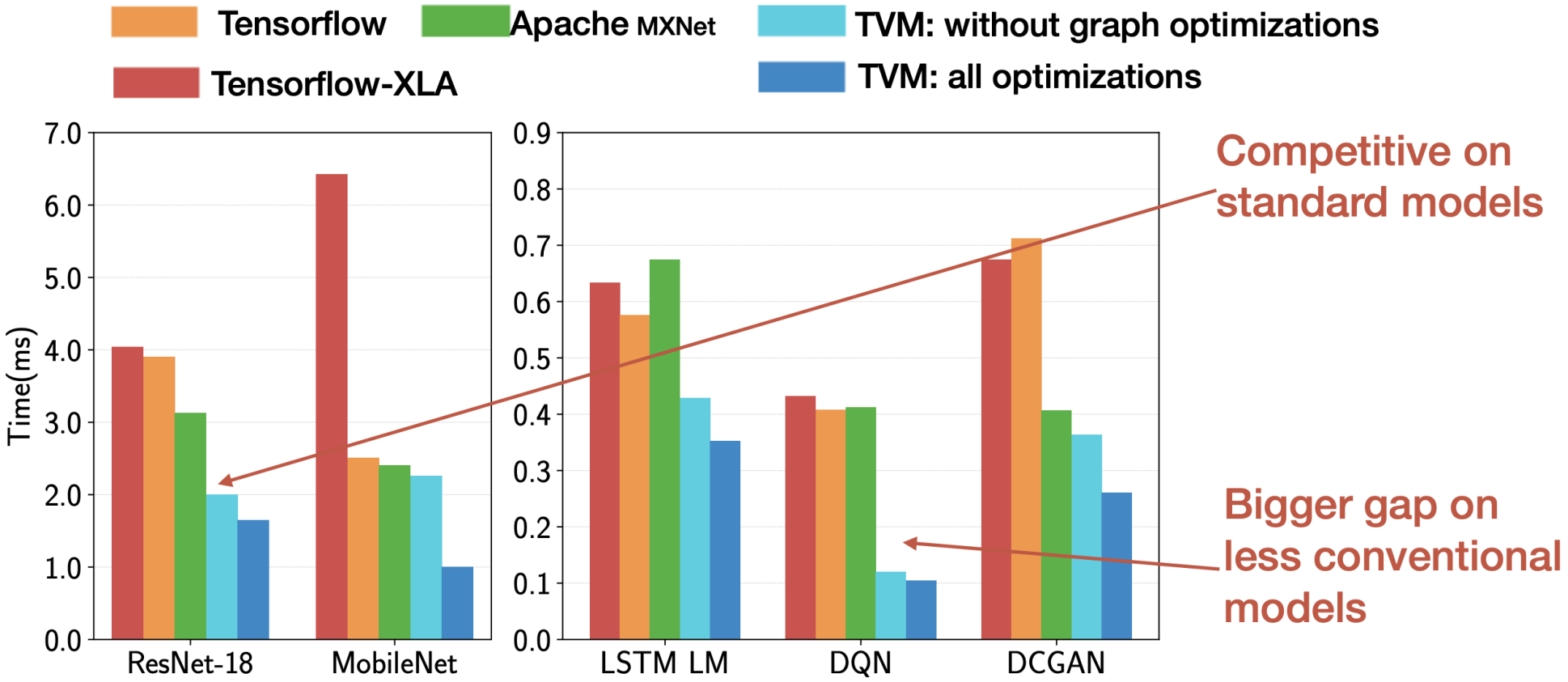


Adapt to hardware by learning, make prediction in milliseconds

Efficient ML-based Cost Model



End-to-end Inference Performance



Outline: Kernel Autotuning

- triton.autotune: manual template + search
- AutoTVM: manual template + learning-based cost model
- **Ansor: sketch generation + annotation search**

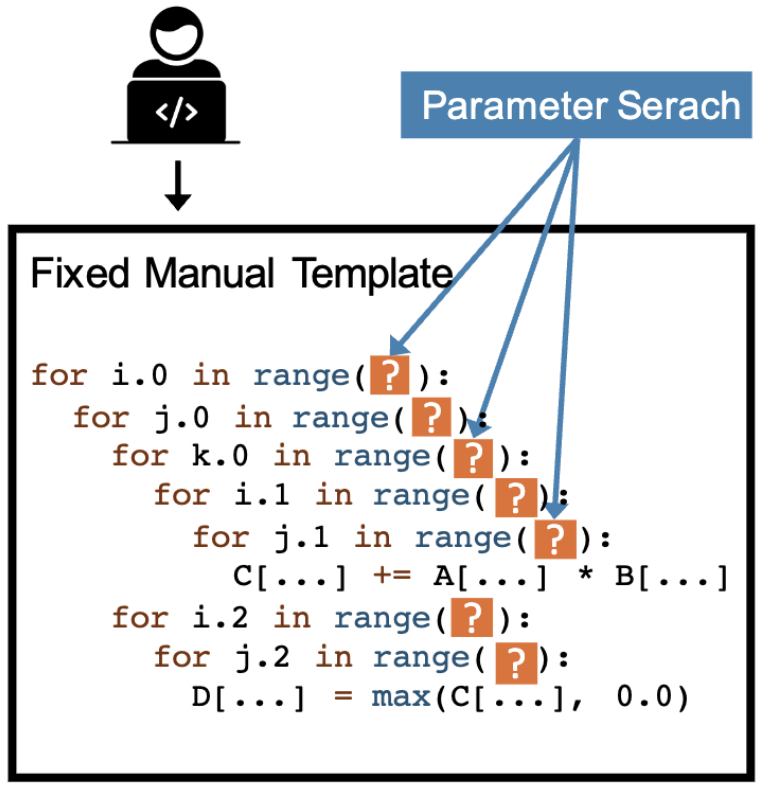
Main Issues with TVM and Triton

Templated-guided search

- Manually-written templates to define a search space

Drawbacks

- Not fully-automated -> requires huge manual effort
- Limited search space -> suboptimal performance



(a) Template-guided Search

Ansor: Generating High-Performance Tensor Programs for Deep Learning. OSDI 2020

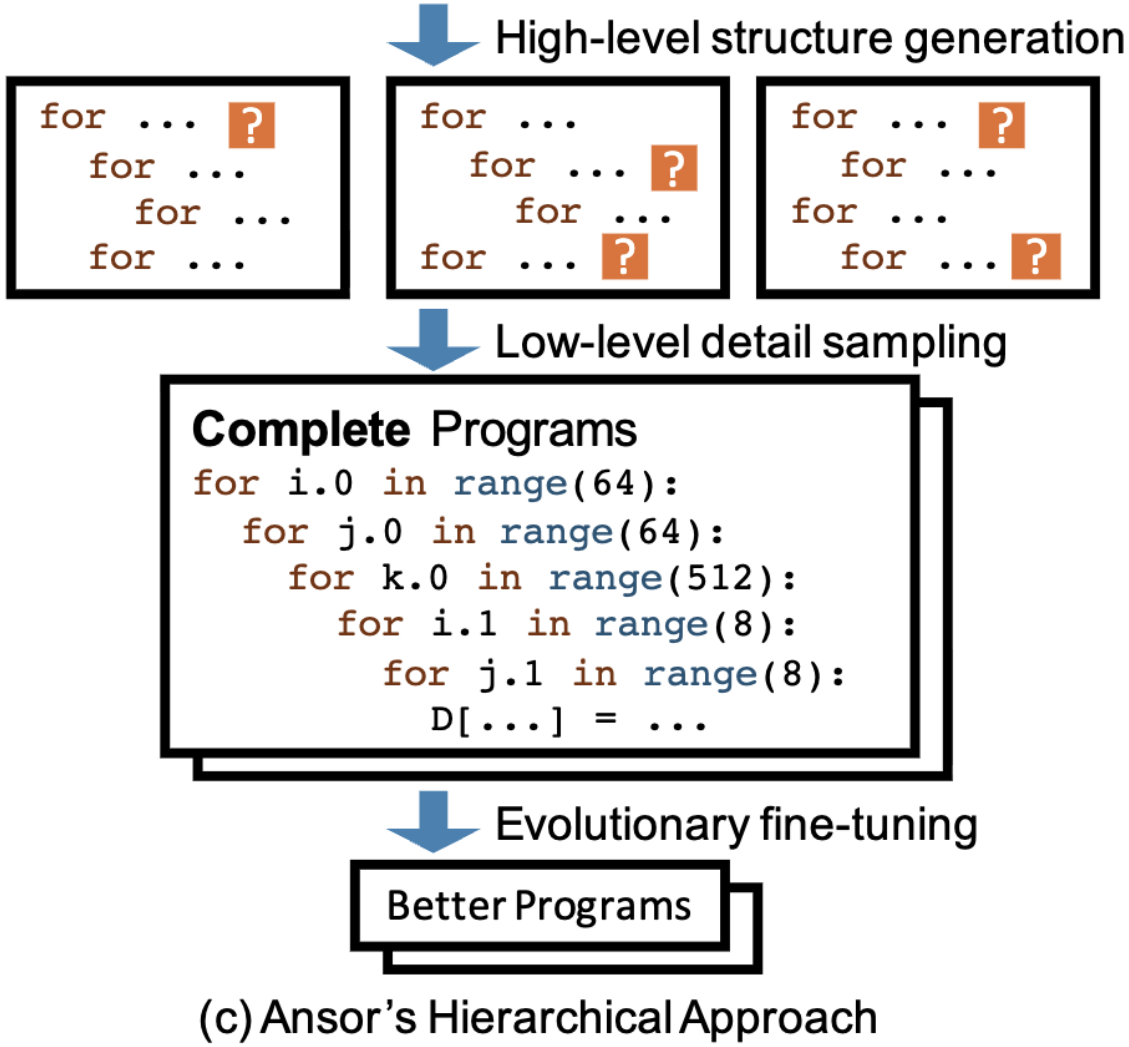
Ansor's Approach

Challenge 1. How to build a large search space automatically?

- Hierarchical search (sketch + annotation)

Challenge 2: How to search efficiently?

- Sample complete programs and fine-tune them

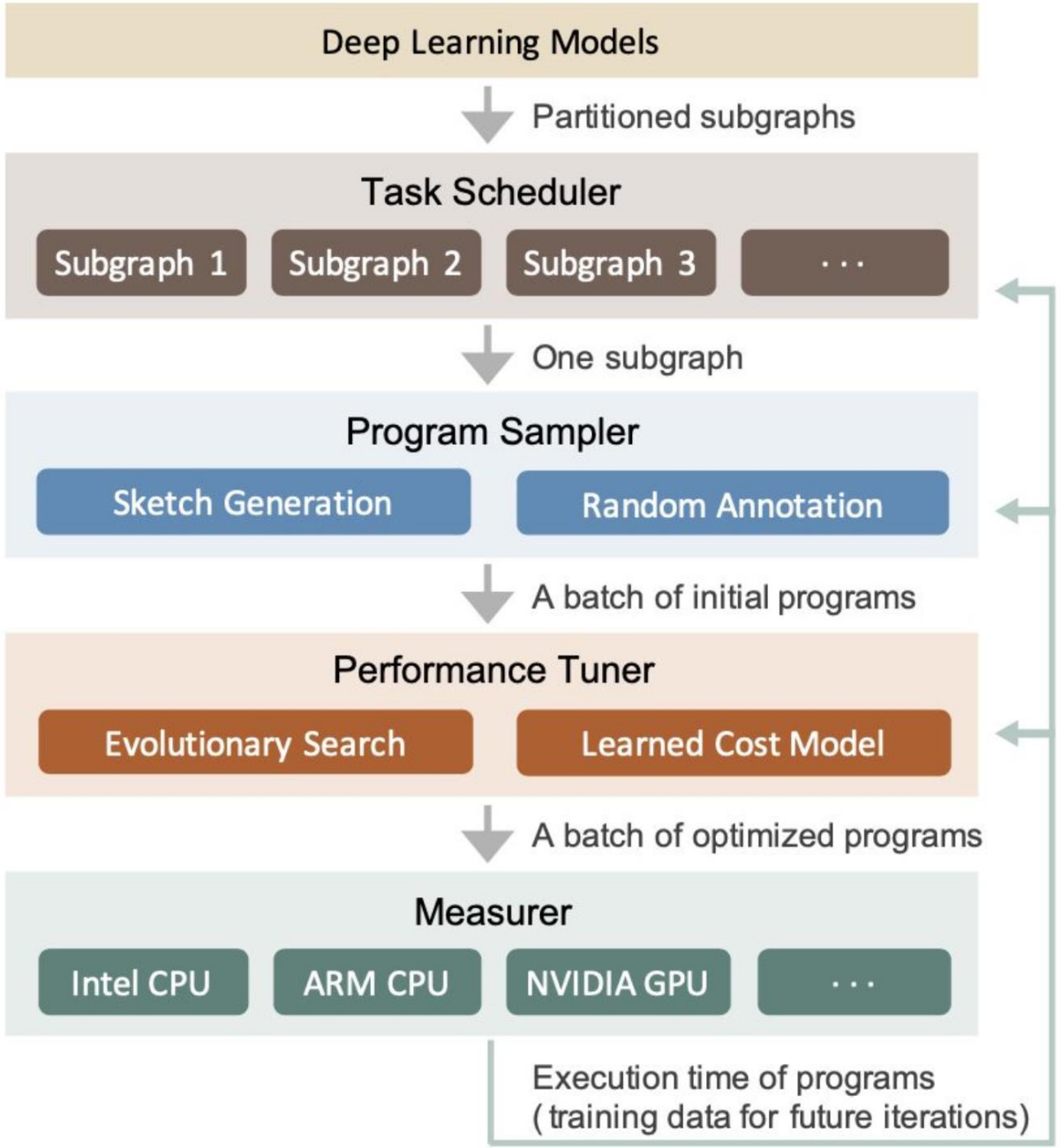


(c) Ansor's Hierarchical Approach

Ansor: Generating High-Performance Tensor Programs for Deep Learning. OSDI 2020

Ansor's Overview

- Inputs: set of DNNs
  - Partitioned into small subgraphs
- Three components
  - Task scheduler
  - Program sampler
  - Performance tuner



Ansor: Generating High-Performance Tensor Programs for Deep Learning. OSDI 2020

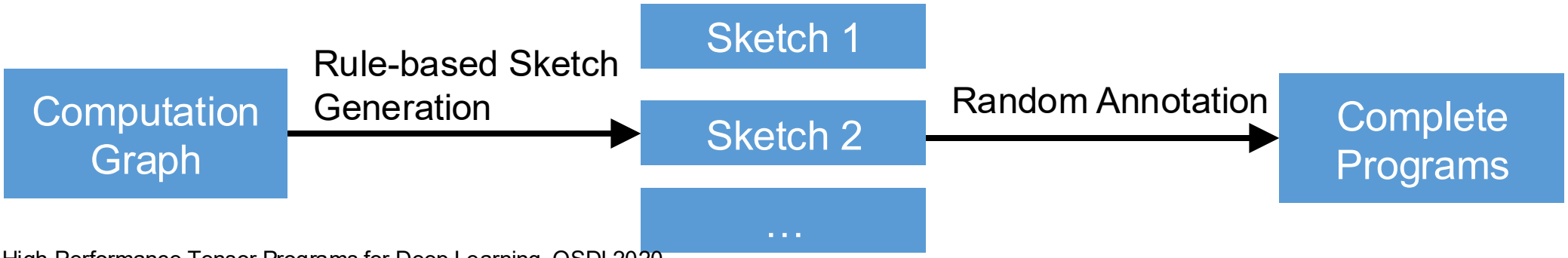
Program Sampling

- Goal: automatically construct a large search space and uniformly sample from the space

Approach: two-level hierarchical search space

- **Sketch:** a few good high-level structures
- **Annotation:** billions of low-level details

- Sampling process:



Sketch Generation Example 1/2

Example Input 1:

*** The mathmetical expression:**

$$C[i,j] = \sum_k A[i,k] \times B[k,j]$$

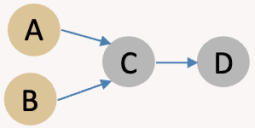
$$D[i,j] = \max(C[i,j], 0.0)$$

where $0 \le i,j,k < 512$

*** The corresponding naïve program:**

```
for i in range(512):
  for j in range(512):
    for k in range(512):
      C[i, j] += A[i, k] * B[k, j]
for i in range(512):
  for j in range(512):
    D[i, j] = max(C[i, j], 0.0)
```

*** The corresponding DAG:**



Derivation:

$$Input\ 1 \rightarrow \sigma(S_0, i=4) \xrightarrow{\text{Rule 1}} \sigma(S_1, i=3) \xrightarrow{\text{Rule 4}}$$
$$\sigma(S_2, i=2) \xrightarrow{\text{Rule 1}} \sigma(S_3, i=1) \xrightarrow{\text{Rule 1}} Sketch\ 1$$

Generated sketch 1

```
for i.0 in range(TILE_I0):
  for j.0 in range(TILE_J0):
    for i.1 in range(TILE_I1):
      for j.1 in range(TILE_J1):
        for k.0 in range(TILE_K0):
          for i.2 in range(TILE_I2):
            for j.2 in range(TILE_J2):
              for k.1 in range(TILE_I1):
                for i.3 in range(TILE_I3):
                  for j.3 in range(TILE_J3):
                    C[...] += A[...] * B[...]
        for i.4 in range(TILE_I2 * TILE_I3):
          for j.4 in range(TILE_J2 * TILE_J3):
            D[...] = max(C[...], 0.0)
```

"SSRSRSS" multi-level tiling + fusion

Sketch Generation Example 2/2

Example Input 2:

*** The mathmetical expression:**

$$B[i,l] = \max(A[i,l], 0.0)$$

$$C[i,k] = \begin{cases} B[i,k], & k < 400 \\ 0\ , & k \geq 400 \end{cases}$$

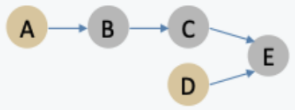
$$E[i,j] = \sum_k C[i,k] \times D[k,j]$$

where $0 \leq i < 8,\ 0 \leq j < 4,$
$0 \leq k < 512, 0 \leq l < 400$

*** The corresponding naïve program:**

```
for i in range(8):
  for l in range(400):
    B[i, l] = max(A[i, l], 0.0)
for i in range(8):
  for k in range(512):
    C[i, k] = B[i, k] if k < 400 else 0
for i in range(8):
  for j in range(4):
    for k in range(512):
      E[i, j] += C[i, k] * D[k, j]
```

*** The corresponding DAG:**



Generated sketch 2

```
for i in range(8):
  for k in range(512):
    C[i, j] = max(A[i,k], 0.0) if k<400 else 0
for i.0 in range(TILE_I0):
  for j.0 in range(TILE_J0):
    for i.1 in range(TILE_I1):
      for j.1 in range(TILE_J1):
        for k.0 in range(TILE_K0):
          for i.2 in range(TILE_I2):
            for j.2 in range(TILE_J2):
              for k.1 in range(TILE_I1):
                for i.3 in range(TILE_I3):
                  for j.3 in range(TILE_J3):
                    E.cache[...] += C[...] * D[...]
        for i.4 in range(TILE_I2 * TILE_I3):
          for j.4 in range(TILE_J2 * TILE_J3):
            E[...] = E.cache[...]
```

$$Input\ 2 \rightarrow \sigma(S_0, i=5) \xrightarrow{\text{Rule 5}} \sigma(S_1, i=5) \xrightarrow{\text{Rule 4}}$$
$$\sigma(S_2, i=4) \xrightarrow{\text{Rule 1}} \sigma(S_3, i=3) \xrightarrow{\text{Rule 1}}$$
$$\sigma(S_4, i=2) \xrightarrow{\text{Rule 2}} \sigma(S_5, i=1) \xrightarrow{\text{Rule 1}} Sketch\ 2$$

Generated sketch 3

```
for i in range(8):
  for k in range(512):
    C[i, k] = max(A[i, k], 0.0) if k < 400 else 0
for i in range(8):
  for j in range(4):
    for k_o in range(TILE_K0):
      for k_i in range(TILE_KI):
        E.rf[...] += C[...] * D[...]
for i in range(8):
  for j in range(4):
    for k_i in range(TILE_KI):
      E[...] += E.rf[...]
```

$$Input\ 2 \rightarrow \sigma(S_0, i=5) \xrightarrow{\text{Rule 6}} \sigma(S_1, i=4) \xrightarrow{\text{Rule 1}}$$
$$\sigma(S_2, i=3) \xrightarrow{\text{Rule 1}} \sigma(S_3, i=2) \xrightarrow{\text{Rule 2}}$$
$$\sigma(S_4, i=1) \xrightarrow{\text{Rule 1}} Sketch\ 3$$

Random Annotation Examples

Generated sketch 1

```
for i.0 in range(TILE_I0):
  for j.0 in range(TILE_J0):
    for i.1 in range(TILE_I1):
      for j.1 in range(TILE_J1):
        for k.0 in range(TILE_K0):
          for i.2 in range(TILE_I2):
            for j.2 in range(TILE_J2):
              for k.1 in range(TILE_I1):
                for i.3 in range(TILE_I3):
                  for j.3 in range(TILE_J3):
                    C[...] += A[...] * B[...]
        for i.4 in range(TILE_I2 * TILE_I3):
          for j.4 in range(TILE_J2 * TILE_J3):
            D[...] = max(C[...], 0.0)
```

Sampled program 1

```
parallel i.0@j.0@i.1@j.1 in range(256):
  for k.0 in range(32):
    for i.2 in range(16):
      unroll k.1 in range(16):
        unroll i.3 in range(4):
          vectorize j.3 in range(16):
            C[...] += A[...] * B[...]
  for i.4 in range(64):
    vectorize j.4 in range(16):
      D[...] = max(C[...], 0.0)
```

Sampled program 2

```
parallel i.2 in range(16):
  for j.2 in range(128):
    for k.1 in range(512):
      for i.3 in range(32):
        vectorize j.3 in range(4):
          C[...] += A[...] * B[...]
parallel i.4 in range(512):
  for j.4 in range(512):
    D[...] = max(C[...], 0.0)
```

Ansor: Generating High-Performance Tensor Programs for Deep Learning. OSDI 2020

Performance Fine-Tuning

Issue: Random samplings does not guarantee high performance

Generated sketch 1

```
for i.0 in range(TILE_I0):
  for j.0 in range(TILE_J0):
    for i.1 in range(TILE_I1):
      for j.1 in range(TILE_J1):
        for k.0 in range(TILE_K0):
          for i.2 in range(TILE_I2):
            for j.2 in range(TILE_J2):
              for k.1 in range(TILE_I1):
                for i.3 in range(TILE_I3):
                  for j.3 in range(TILE_J3):
                    C[...] += A[...] * B[...]
        for i.4 in range(TILE_I2 * TILE_I3):
          for j.4 in range(TILE_J2 * TILE_J3):
            D[...] = max(C[...], 0.0)
```

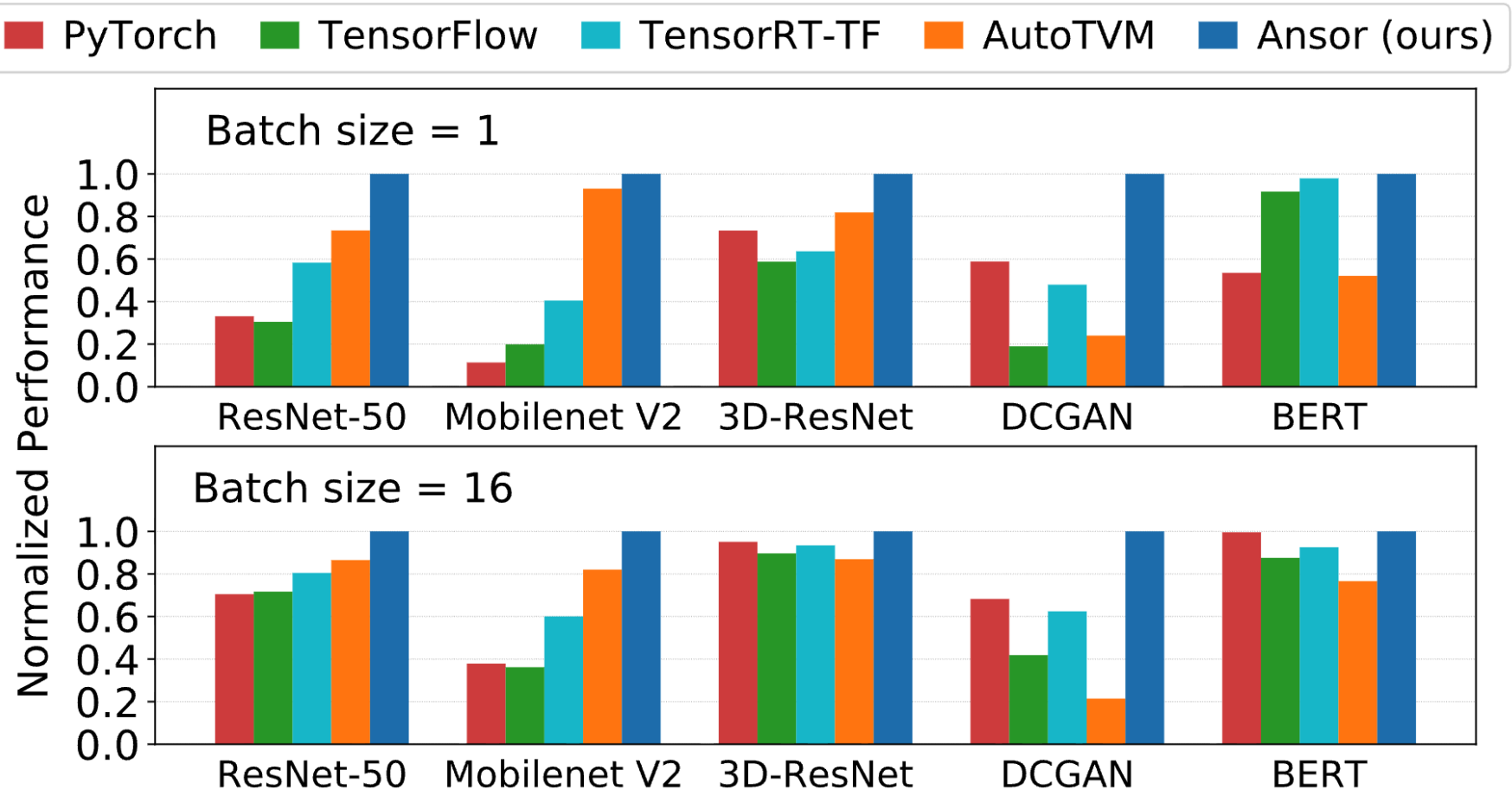

Sampled program 1

```
parallel i.0@j.0@i.1@j.1 in range(256):
  for k.0 in range(32):
    for i.2 in range(16):
      unroll k.1 in range(16):
        unroll i.3 in range(4):
          vectorize j.3 in range(16):
            C[...] += A[...] * B[...]
  for i.4 in range(64):
    vectorize j.4 in range(16):
      D[...] = max(C[...], 0.0)
```

Solution: Perform evolutionary search with learned cost model (AutoTVM)

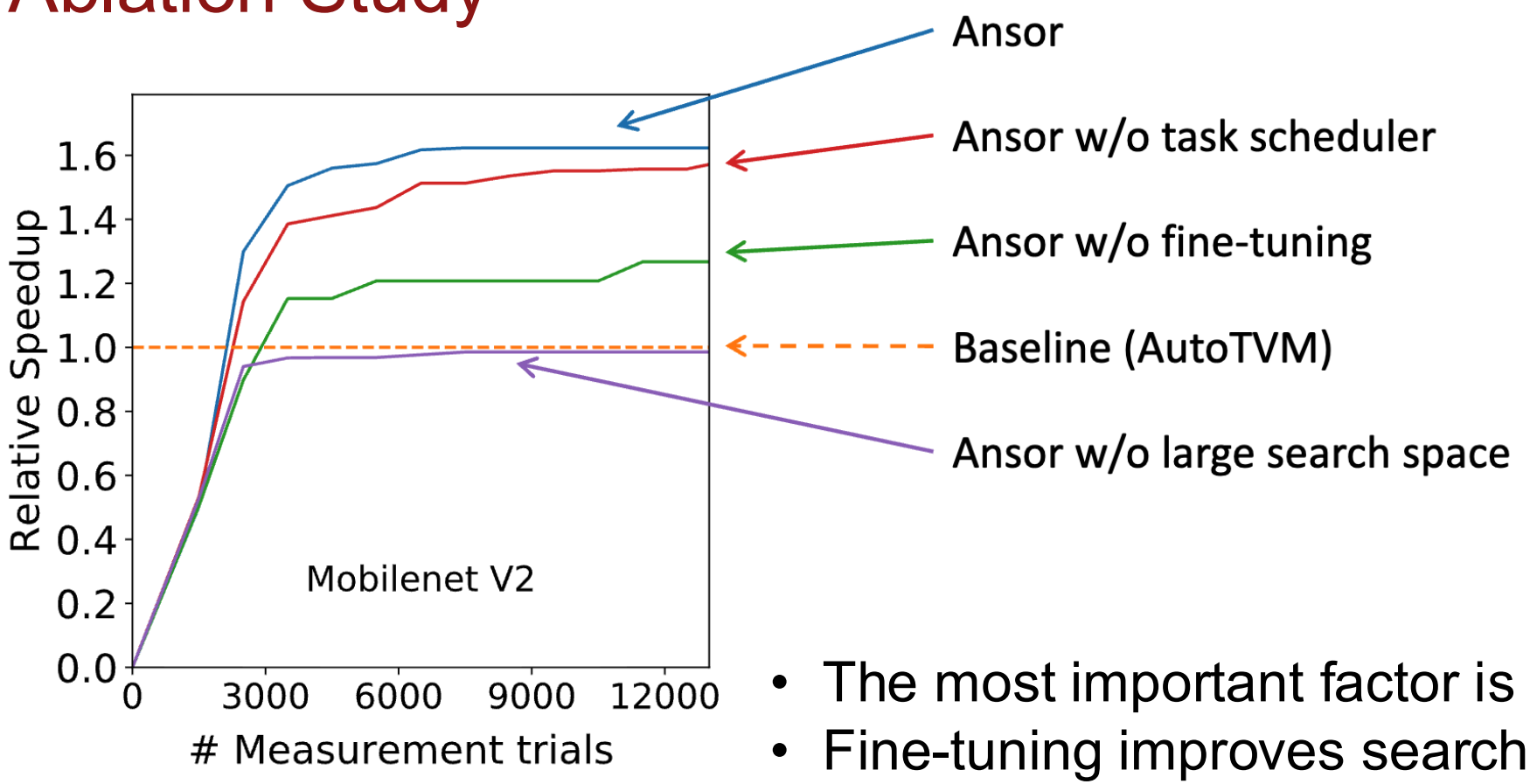
- Randomly mutate tile size
- Randomly mutate parallel/unroll/vectorize factor and granularity

Ansor Automatically Finds High-Performance Kernels



Ansor: Generating High-Performance Tensor Programs for Deep Learning. OSDI 2020

Ablation Study



- The most important factor is the search space
- Fine-tuning improves search results
- Match AutoTVM's performance with 10x less time

Comparing Triton, AutoTVM, and Ansor

	Search Space	Search Algorithm
triton.autotune	Manual template	Exhaustive
AutoTVM	Manual template	Learning-based cost model
Ansor	Sketch generation + random annotation	Learning-based cost model