AutoTVM & AutoScheduler

Presenter : Jaehun Ryu

jaehunryu@postech.ac.kr

Introduction

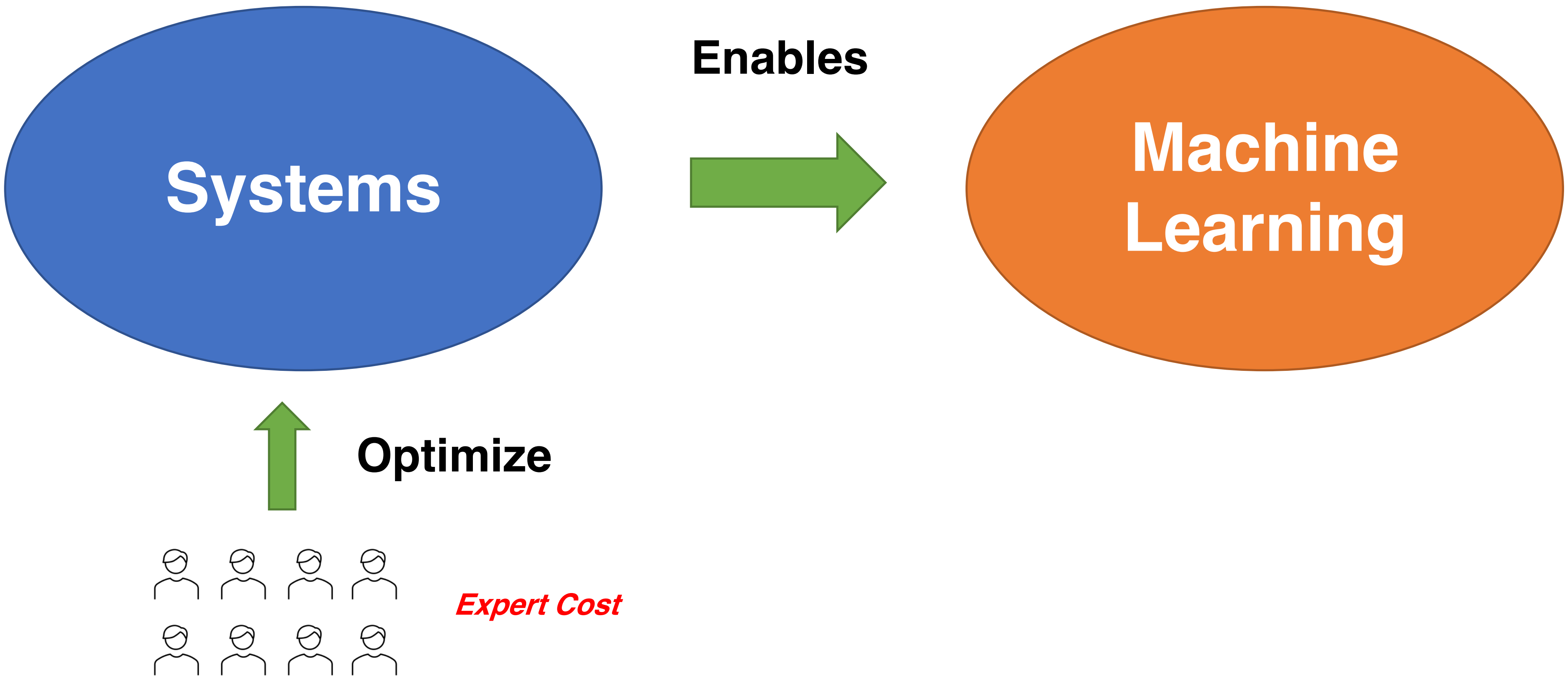
- AutoTVM : Template-based Auto Tuning
  - Learning to optimize tensor programs(NIPS18,Chen et al)

- AutoScheduler : Template-free Auto Scheduling
  - Ansor: Generating High-Performance Tensor Programs for Deep Learning(OSDI 20,Zheng et al)

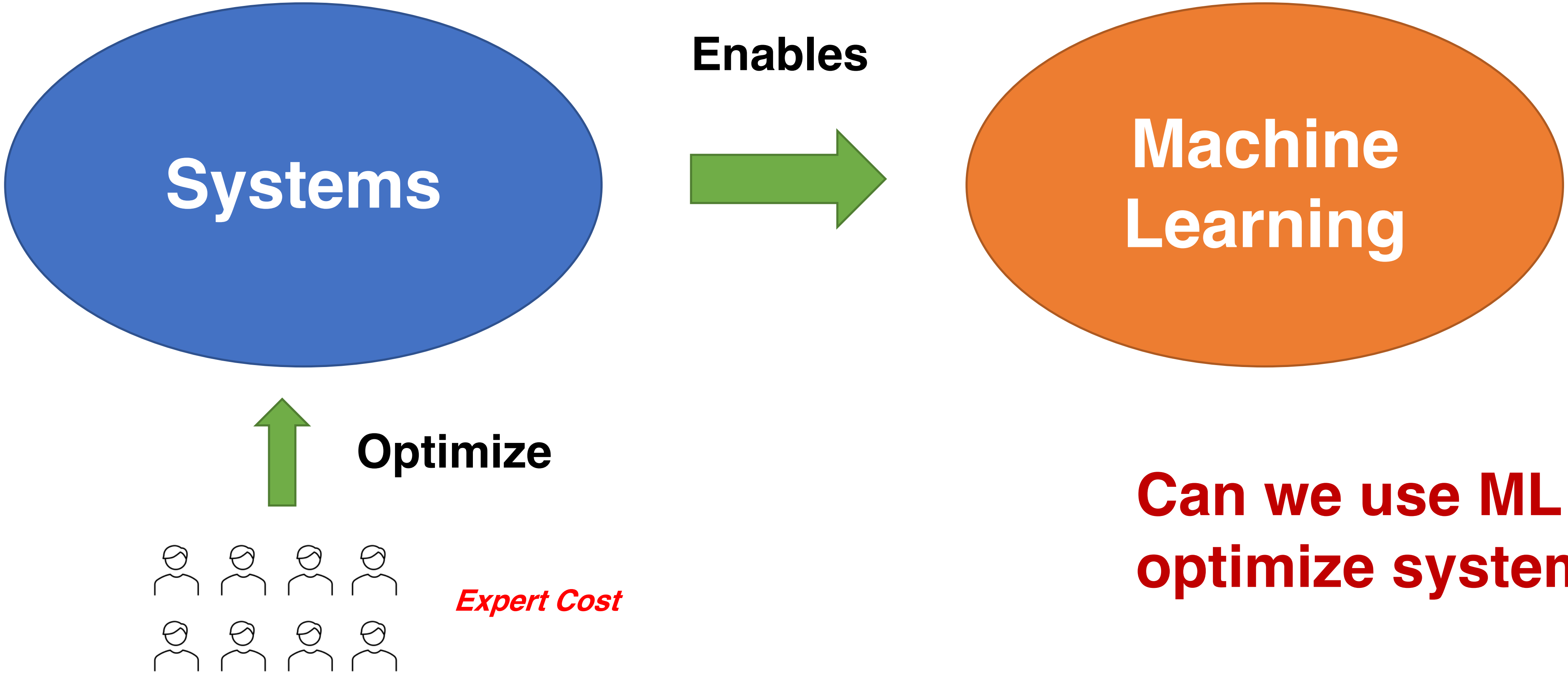
Both are TVM built-in autotuning methods.

Autotuning

Current Learning Systems

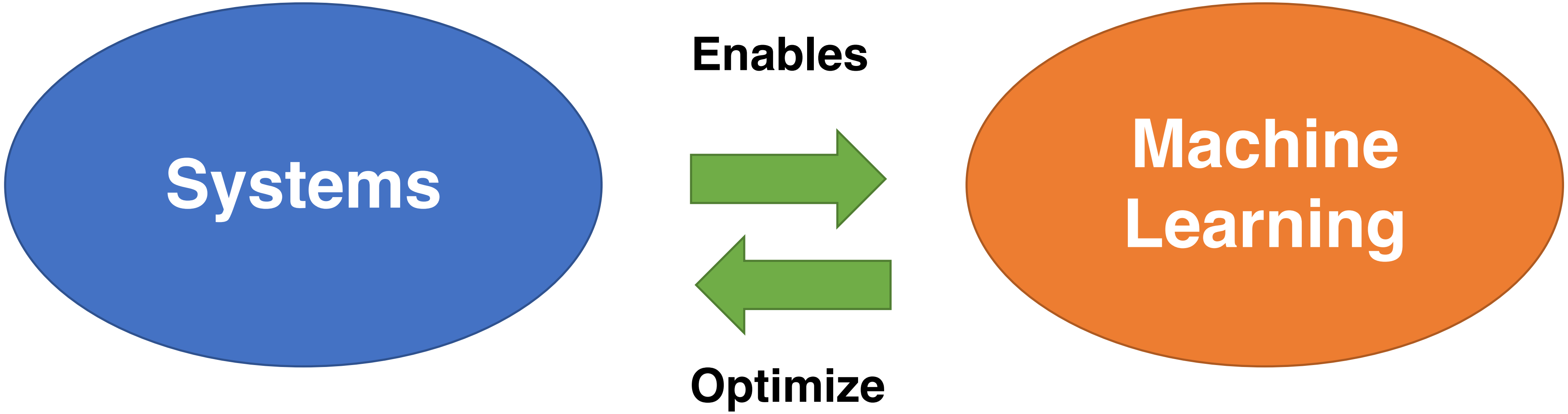


Current Learning Systems

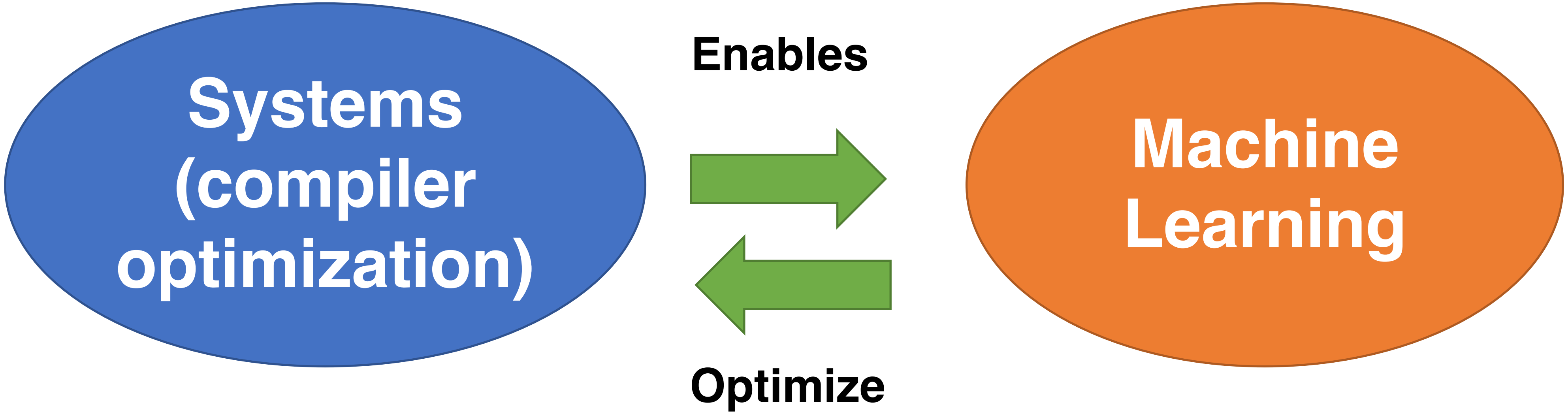


Can we use ML to optimize systems

Learning-based Learning Systems



Learning-based Learning Systems



Autotuning

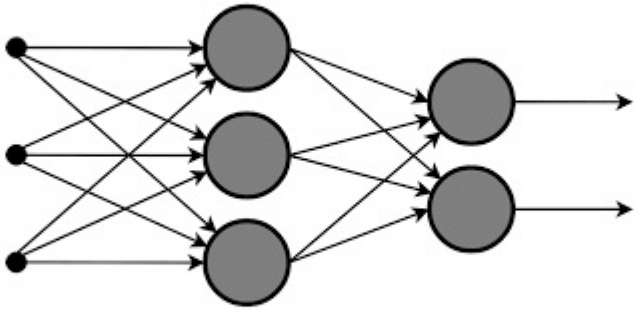
Learning to optimize tensor programs

Learning to optimize tensor programs

- **Why do we need machine learning for systems**
- How to build intelligent systems with learning

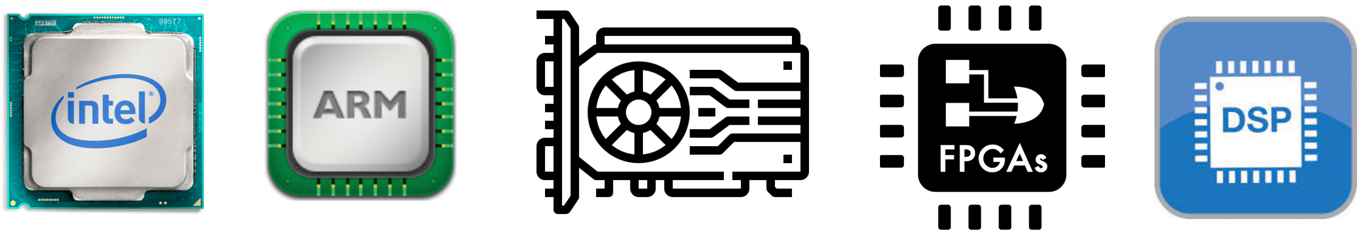
Problem: Deep Learning Deployment

Model

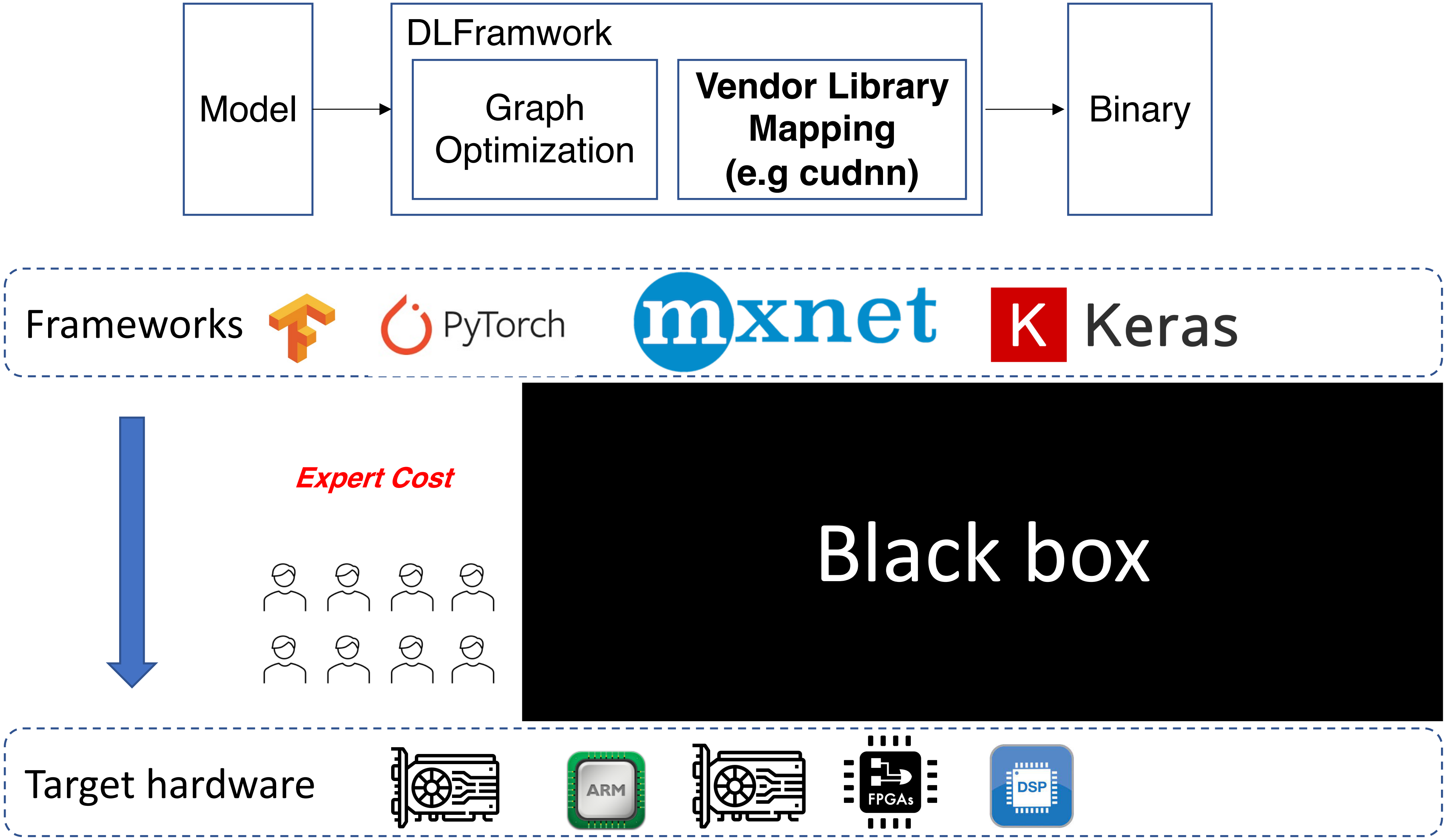


Deploy

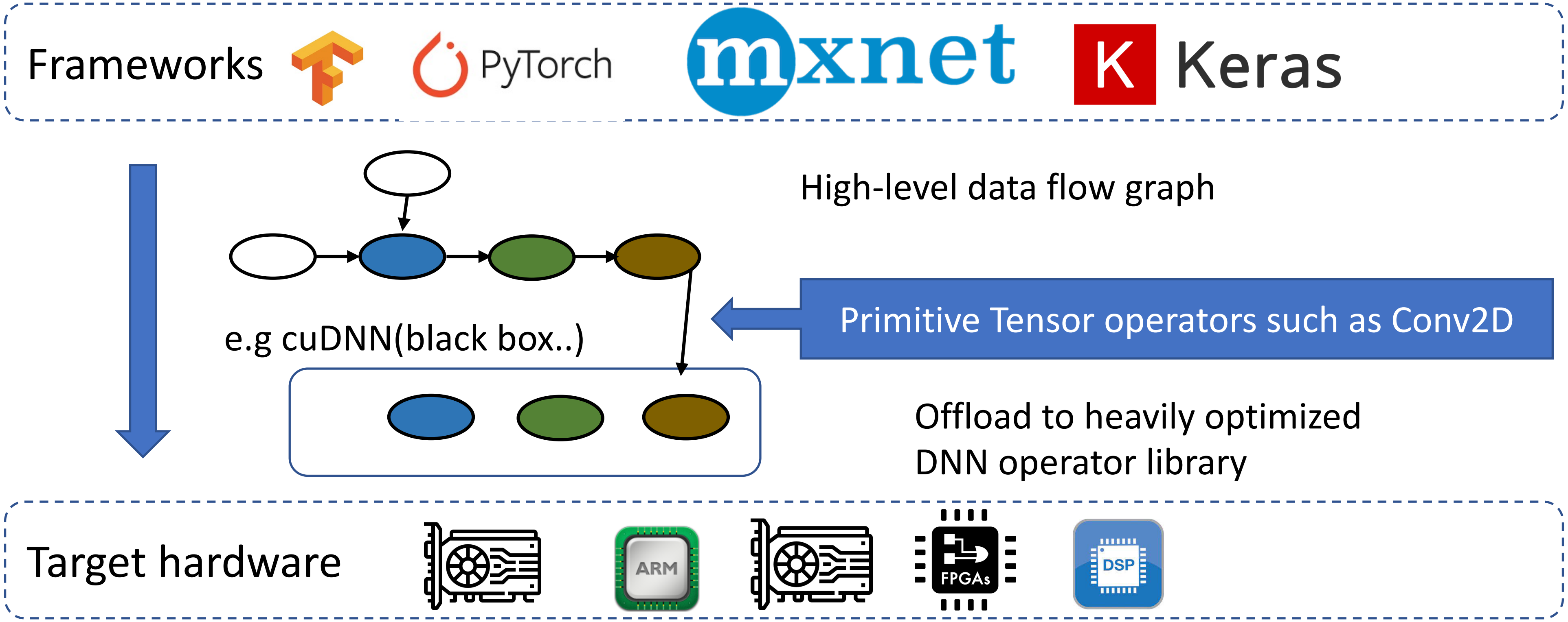
Hardware Backends



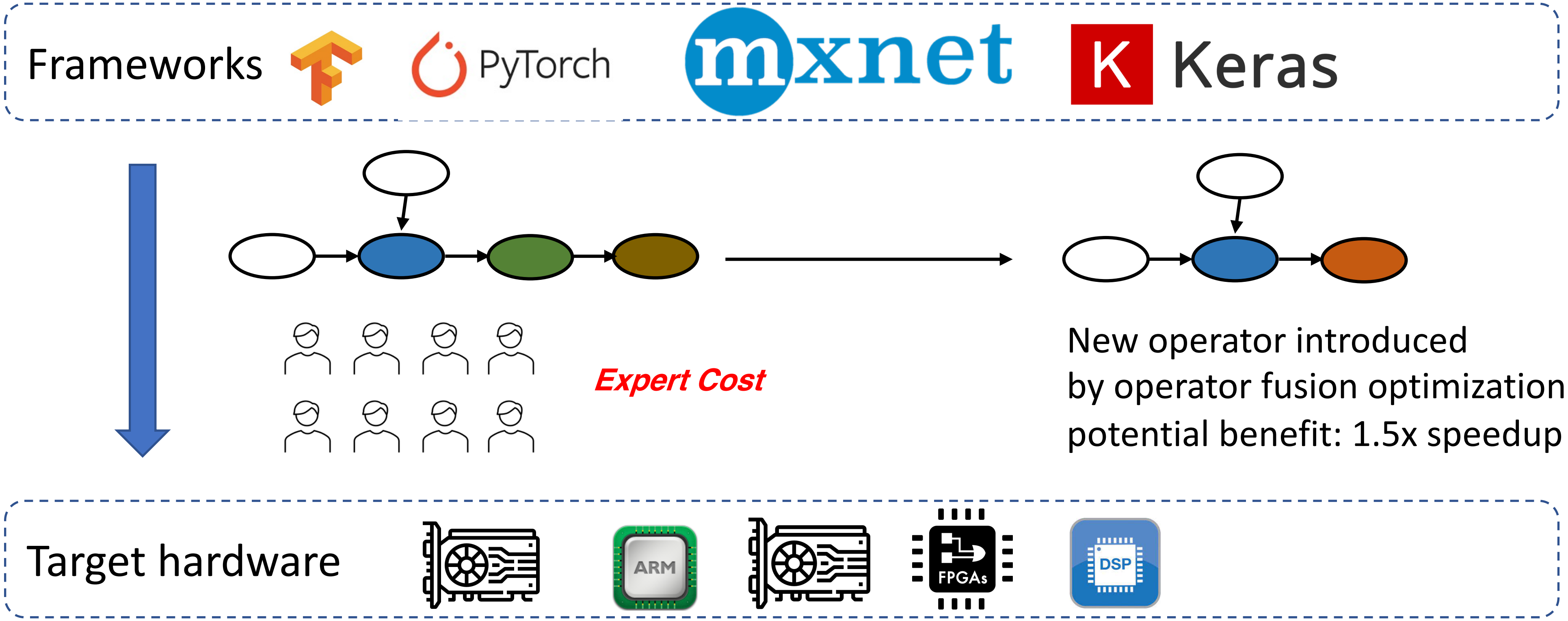
Existing Deep Learning Frameworks



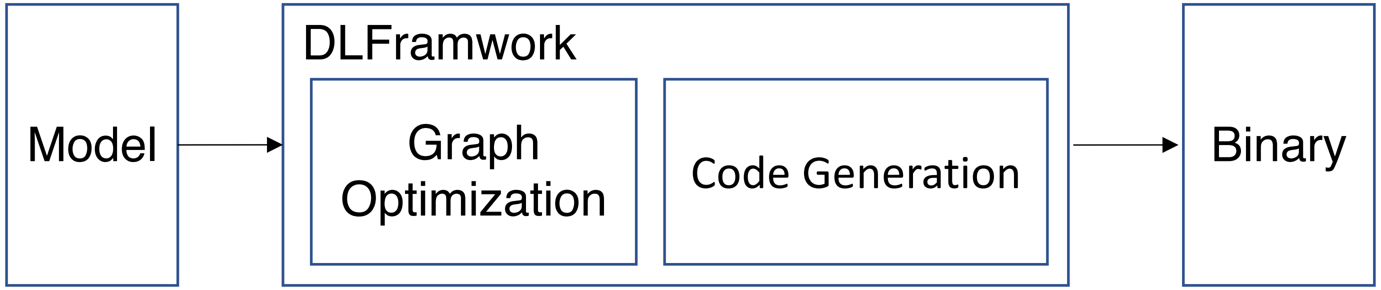
Existing Deep Learning Frameworks



Existing Deep Learning Frameworks



Learning-based Learning System



Frameworks PyTorch mxnet Keras

Machine Learning based Program Optimizer

Directly generate optimized program
for new operator workloads and hardware

Target hardware ARM FPGAs DSP

Learning to optimize tensor programs

- Why do we need machine learning for systems
- **How to build intelligent systems with learning**

Problem Setting

Tensor Expression(high level expression)

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

Lowering

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

```
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]
```

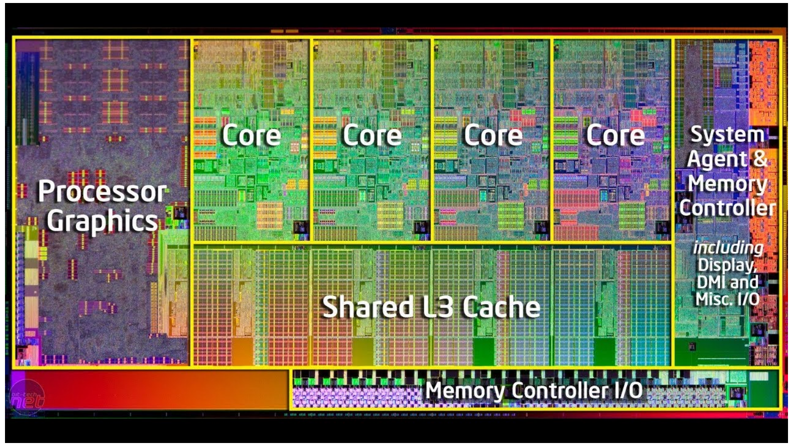
```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      for yi in range(8):
        for xi in range(8):
          for ki in range(8):
            C[yo*8+yi][xo*8+xi] +=
               A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```

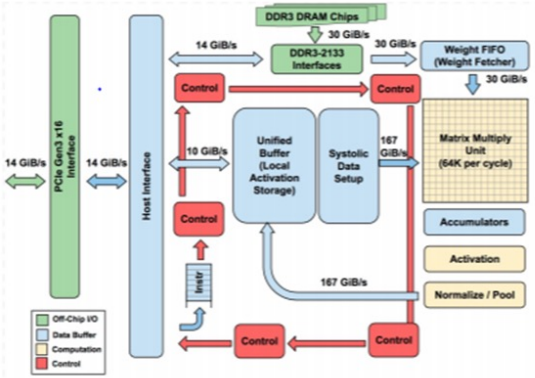
```
inp_buffer AL[8][8], BL[8][8]
acc_buffer CL[8][8]
for yo in range(128):
  for xo in range(128):
    vdla.fill_zero(CL)
    for ko in range(128):
      vdla.dma_copy2d(AL, A[ko*8:ko*8+8][yo*8:yo*8+8])
      vdla.dma_copy2d(BL, B[ko*8:ko*8+8][xo*8:xo*8+8])
      vdla.fused_gemm8x8_add(CL, AL, BL)
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

Optimization Choices in a Search Space

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

It is hard to consider *all hardware characteristics.*
=> Template-based autotune





Systolic Array on the Google TPU

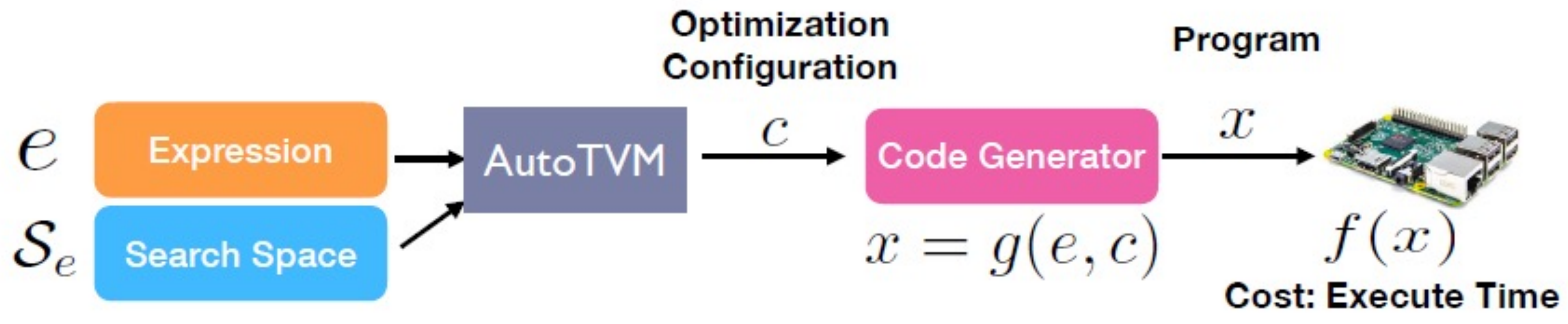
Optimization Choices in a Search Space

```python
@autotvm.template("tutorial/conv2d_no_batching")
def conv2d_no_batching(N, H, W, CO, CI, KH, KW, stride, padding):
    assert N == 1, "Only consider batch_size = 1 in this template"

    data = te.placeholder((N, CI, H, W), name="data")
    kernel = te.placeholder((CO, CI, KH, KW), name="kernel")
    conv = topi.nn.conv2d_nchw(data, kernel, stride, padding, dilation=1, out_dtype="float32")
    s = te.create_schedule([conv.op])

    ##### space definition begin #####
    n, f, y, x = s[conv].op.axis
    rc, ry, rx = s[conv].op.reduce_axis

    cfg = autotvm.get_config()
    cfg.define_split("tile_f", f, num_outputs=4)
    cfg.define_split("tile_y", y, num_outputs=4)
    cfg.define_split("tile_x", x, num_outputs=4)
    cfg.define_split("tile_rc", rc, num_outputs=3)
    cfg.define_split("tile_ry", ry, num_outputs=3)
    cfg.define_split("tile_rx", rx, num_outputs=3)
    cfg.define_knob("auto_unroll_max_step", [0, 512, 1500])
    cfg.define_knob("unroll_explicit", [0, 1])
    ##### space definition end #####

    # inline padding
    pad_data = s[conv].op.input_tensors[0]
    s[pad_data].compute_inline()
    data, raw_data = pad_data, data
```

Operation Template

```
// attr [iter_var(nn.outer, )] pragma_auto_unroll_max_step = 0
// attr [iter_var(nn.outer, )] pragma_unroll_explicit = 0
for (nn.outer, 0, 1) {
  // attr [iter_var(blockIdx.z, , blockIdx.z)] thread_extent = 4
  // attr [iter_var(blockIdx.y, , blockIdx.y)] thread_extent = 3
  // attr [iter_var(blockIdx.x, , blockIdx.x)] thread_extent = 3
  // attr [iter_var(vthread, , vthread)] virtual_thread = 2
  // attr [iter_var(vthread, , vthread)] virtual_thread = 1
  // attr [iter_var(vthread, , vthread)] virtual_thread = 1
  // attr [iter_var(threadIdx.z, , threadIdx.z)] thread_extent = 1
  // attr [iter_var(threadIdx.y, , threadIdx.y)] thread_extent = 1
  // attr [iter_var(threadIdx.x, , threadIdx.x)] thread_extent = 1
  // attr [compute.local] storage_scope = "local"
  allocate compute.local[float32 * 1 * 1 * 1 * 1]
  for (rc.outer, 0, 4) {
    for (ry.outer, 0, 3) {
      for (rx.outer, 0, 3) {
        // attr [pad_temp.shared] storage_scope = "shared"
        allocate pad_temp.shared[float32 * 1 * 1 * 1 * 1]
        // attr [iter_var(threadIdx.z, , threadIdx.z)] thread_extent = 1
        // attr [iter_var(threadIdx.y, , threadIdx.y)] thread_extent = 1
        // attr [iter_var(threadIdx.x, , threadIdx.x)] thread_extent = 1
        for (ax0.ax1.fused.ax2.fused.ax3.fused.inner.inner.inner, 0, 1) {
          pad_temp.shared[0] = placeholder[(((((rc.outer*64) + (blockIdx.y*16)) + (ry.outer
        }
        // attr [placeholder.shared] storage_scope = "shared"
        allocate placeholder.shared[float32 * 2 * 1 * 1 * 1]
        // attr [iter_var(threadIdx.z, , threadIdx.z)] thread_extent = 1
        // attr [iter_var(threadIdx.y, , threadIdx.y)] thread_extent = 1
        // attr [iter_var(threadIdx.x, , threadIdx.x)] thread_extent = 1
        for (ax0.ax1.fused.ax2.fused.ax3.fused.inner.inner.inner, 0, 2) {
          placeholder.shared[ax0.ax1.fused.ax2.fused.ax3.fused.inner.inner.inner] = placeh
        }
        for (rc.inner, 0, 1) {
          for (ry.inner, 0, 1) {
            for (rx.inner, 0, 1) {
              for (nn.c, 0, 1) {
                for (ff.c, 0, 1) {
                  for (yy.c, 0, 1) {
                    for (xx.c, 0, 1) {
                      compute.local[0] = (compute.local[0] + (pad_temp.shared[0]*placeholde
                    }
```

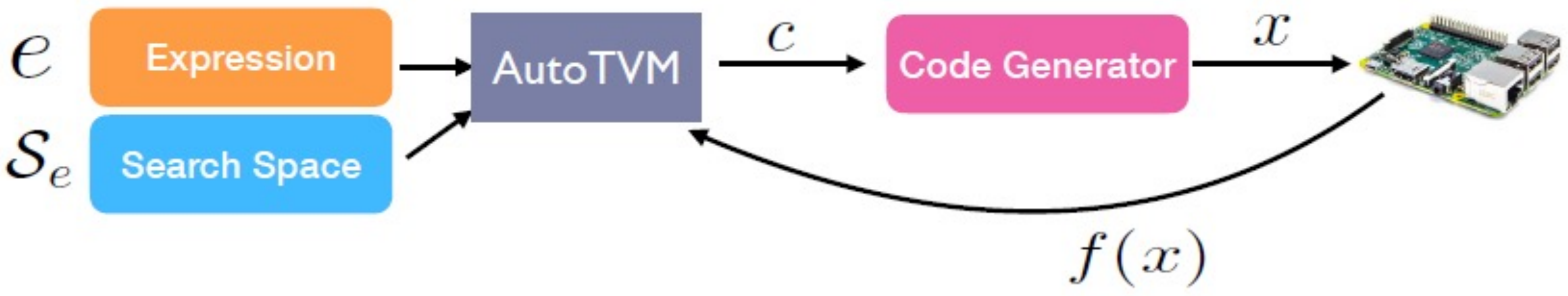
Problem Formalization



Objective $argmin_{c \in \mathcal{S}_e} f(g(e, c))$

Black-box Optimization

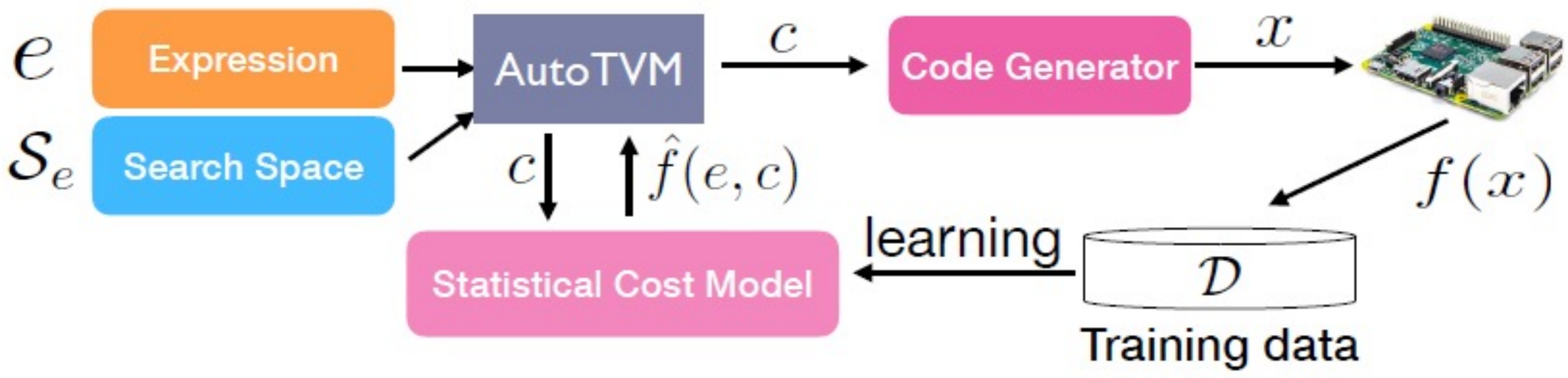
Try each configuration until we find a good one



Challenge: lots of experimental trials, each trial costs ~1 second

Statistical Cost Model

Use machine learning to learn a statistical cost model



Benefit: Automatically adapt to hardware type

Challenge: How to design the cost model

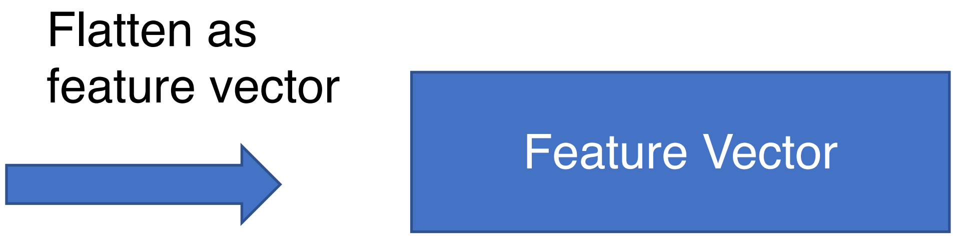
Loop Context Feature

```python
for y in range(8):
  for x in range(8):
    C[y][x]=0
    for k in range(8):
      C[y][x]+=A[k][y]*B[k][x]
```

(a) Low level AST

	touched memory C	touched memory A	touched memory B	outer loop length
y	64	64	64	1
x	8	8	64	8
k	1	8	8	64

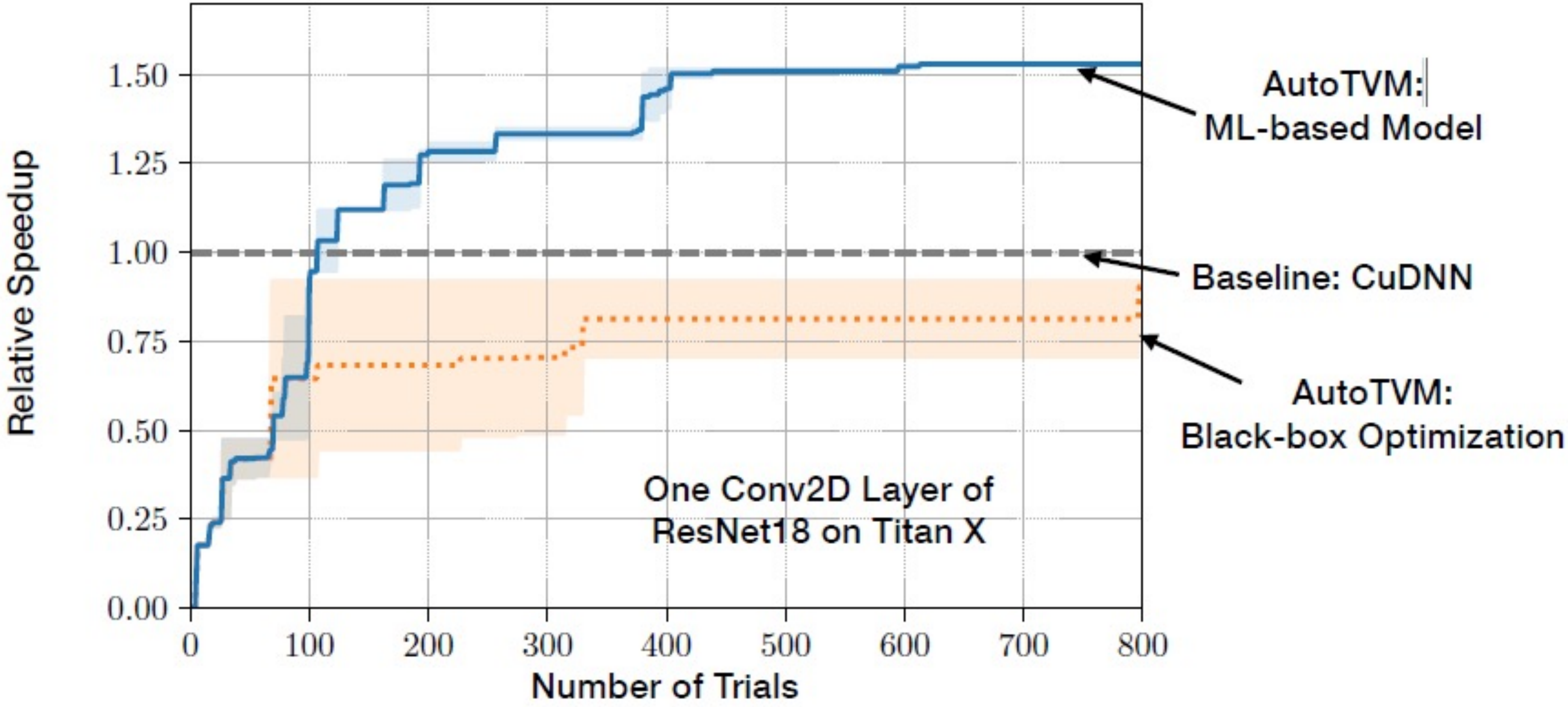
(b) Loop context vectors



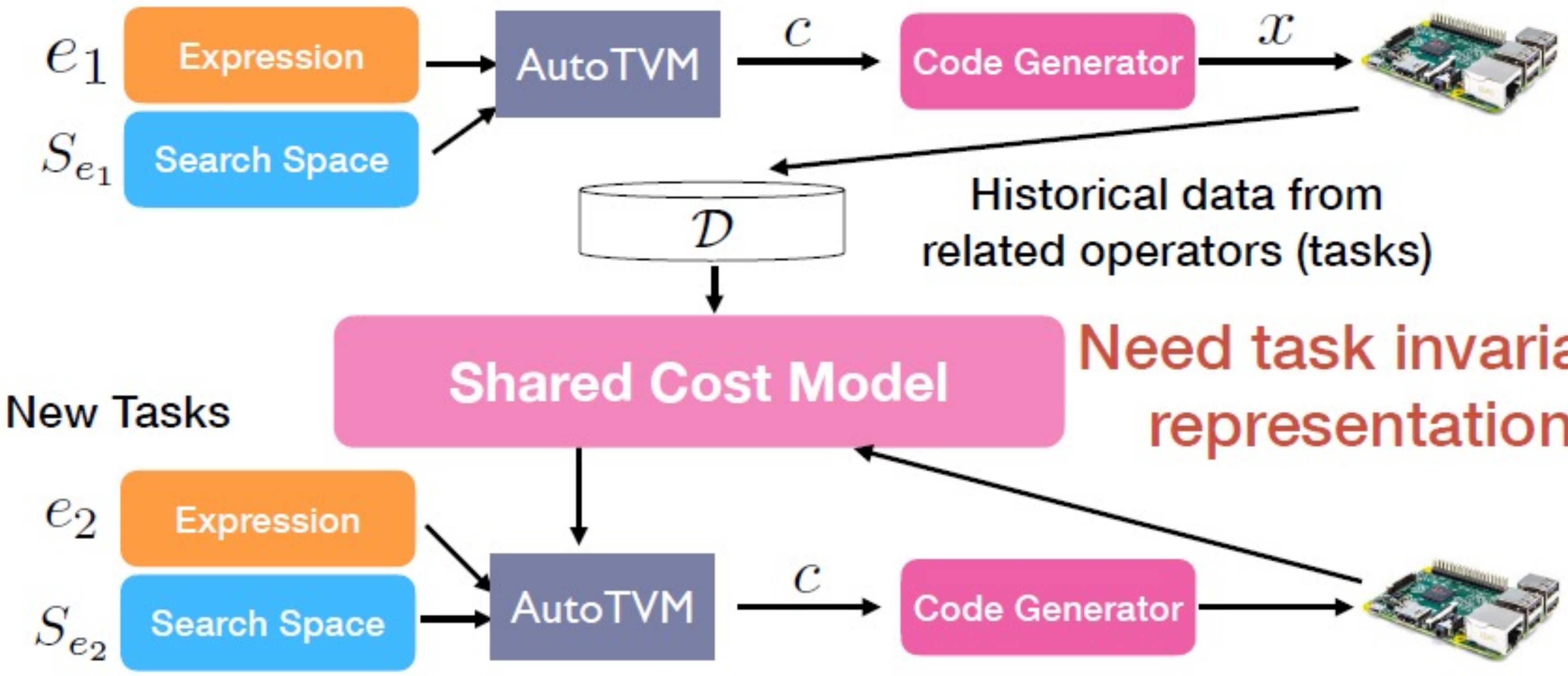
Feature Name		Description
length		The length of this loop
annotation		One-hot annotation of this loop (can be vectorize, unrolled, paralleled, ...)
top-down		The product of the lengths of outer loops
bottom-up		The product of the lengths of inner loops
access pattern (for every buffer)	touch count	The number of touched elements
	reuse ratio	Reuse ratio of this buffer (= bottom-up / touch count)
	stride	Coefficent of this loop varialbe in the index expression

Table 2: Listing of loop context feature

Effectiveness of ML based Model



Transferable Cost Model



Need task invariant representation

Impact of Transfer Learning

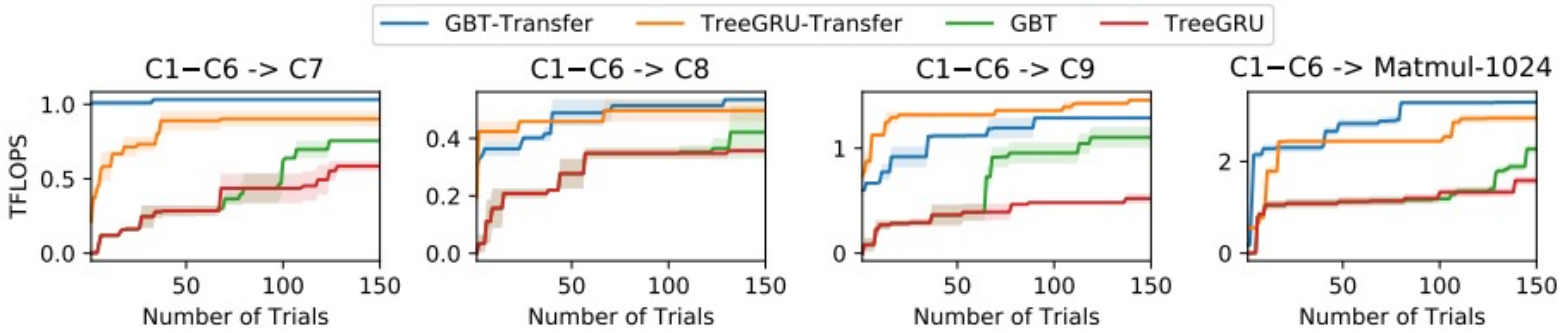


Figure 8: Impact of transfer learning. Transfer-based models quickly found better solutions.

Mxnet : v1.1
TF-GPU:v1.7
TFLite:7558b085
ARM Compute Library :v18.03

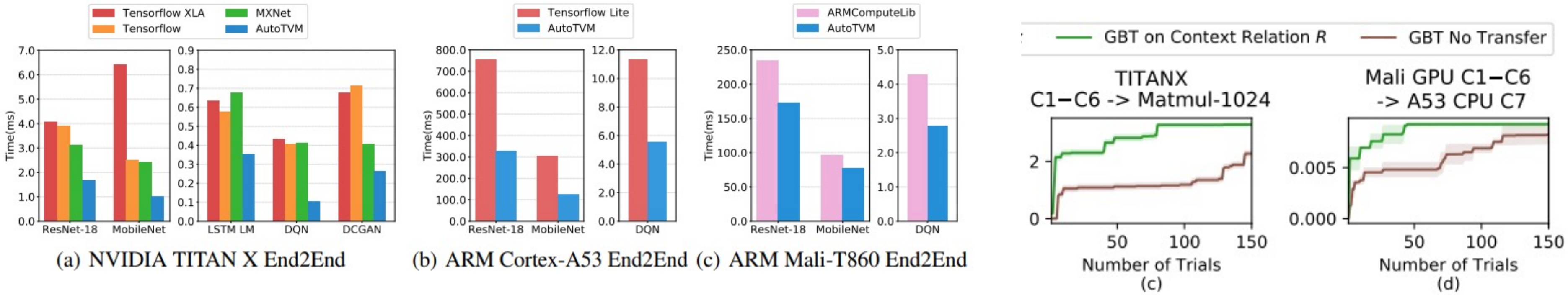


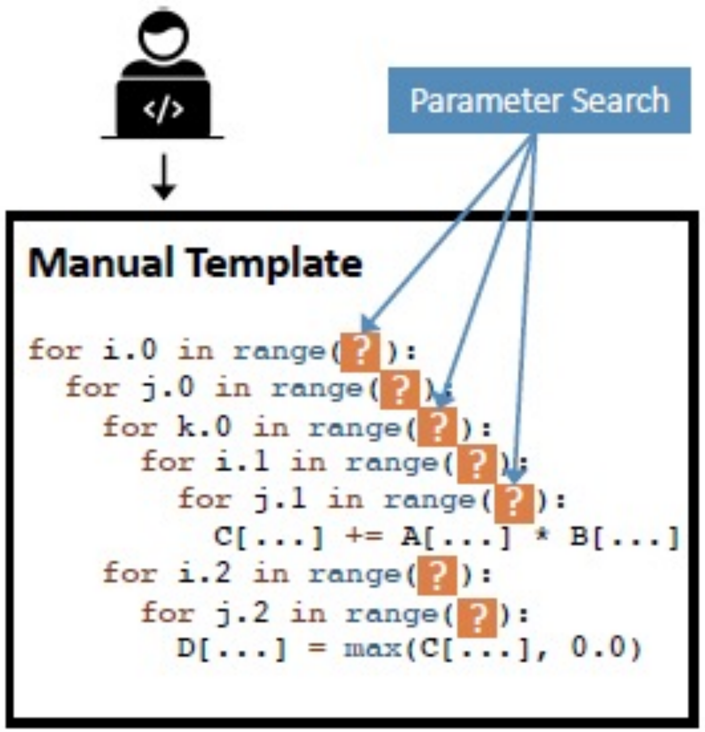
Figure 11: End-to-end performance across back-ends. [2]AutoTVM outperforms the baseline methods.

Ansor: Generating High-Performance Tensor Programs for Deep Learning

TVM's Approach

AutoTVM: Template-guided search

Use **templates** to define the search space for every operator



Drawbacks

- Not fully-automated -> Requires huge manual effort(15K lines of code)
- Limited search space -> Does not achieve optimal performance

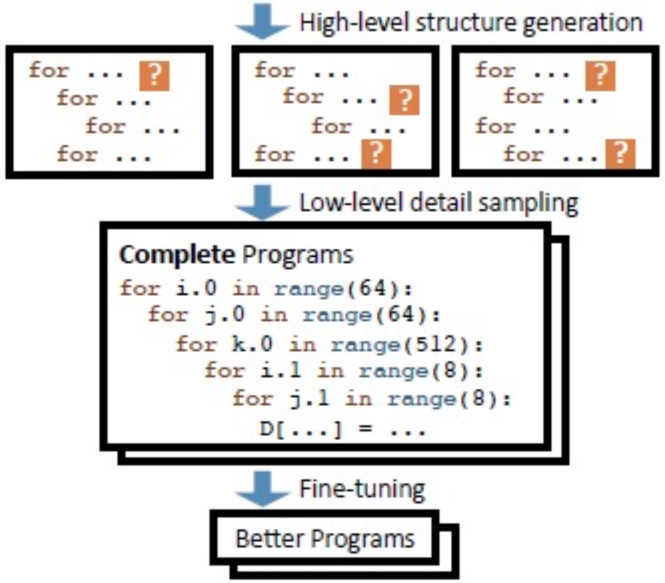
Challenges and ansor's approach

C1: How to build a large search space automatically?

- Use a hierarchical search space

C2: How to search efficiently?

- Sample complete programs and fine-tune them

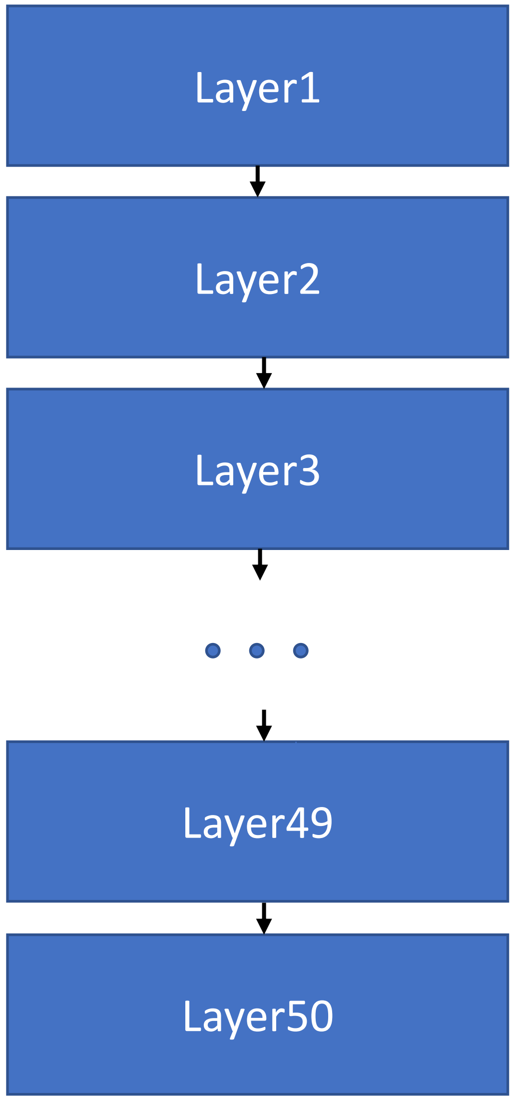


Challenges and ansor's approach

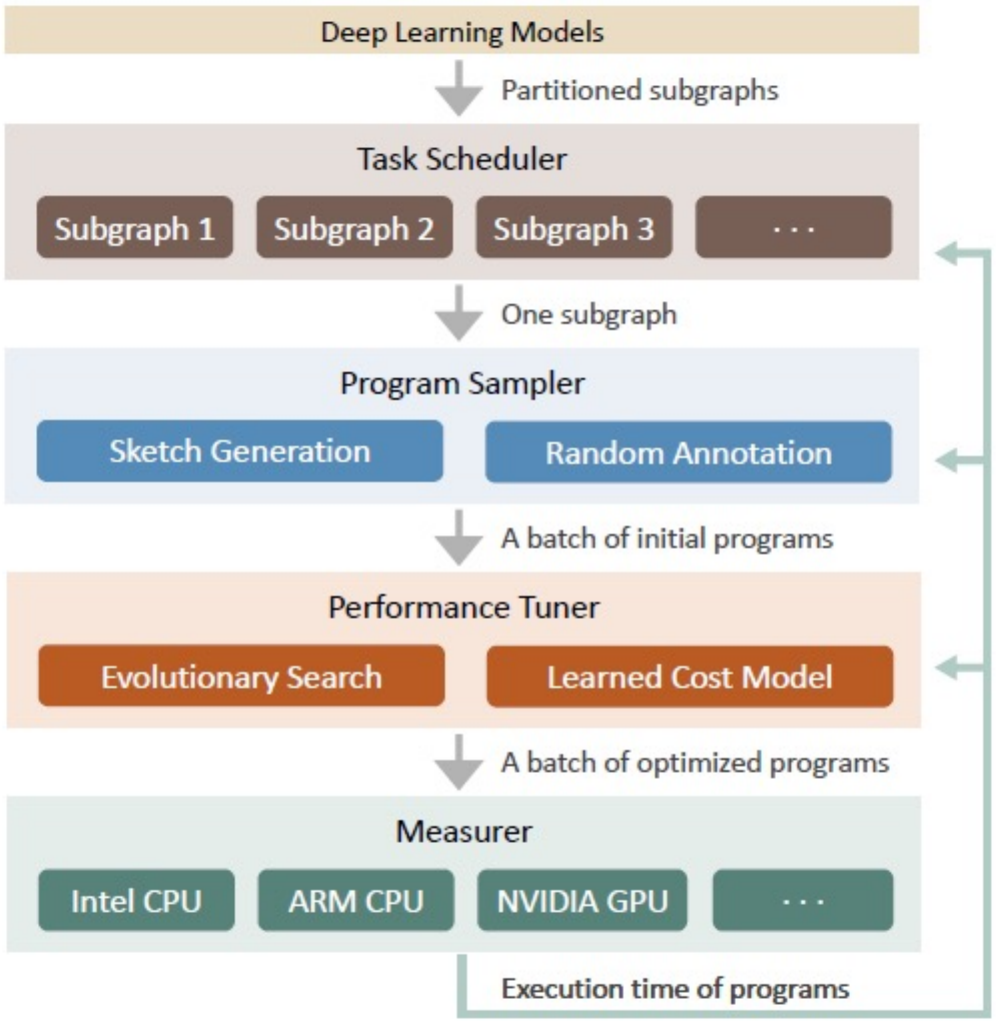
Need to generate programs for all layers -> A lot of search tasks

C3: How to allocate resource for many search tasks?

- Utilize a task scheduler to prioritize important tasks



Ansor Overview

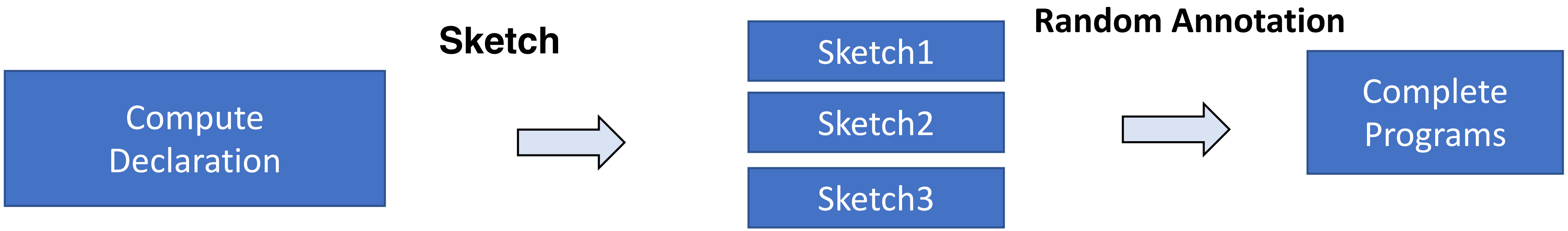


Program Sampling

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

- **Approach**
  - Two-level hierarchical search space: **Sketch** + **Annotation**
  - **Sketch:** a few good high-level structures
  - **Annotation:** billions of low-level details

Sampling Process



Sketch Generation Examples

Example Input 1:

* **The mathmetical expression:**

$$C[i,j] = \sum_{k} A[i,k] \times B[k,j]$$

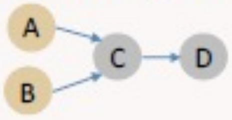
$$D[i,j] = \max(C[i,j], 0.0)$$

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

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 Examples

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):
```

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[...]
or 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

No	Rule Name	Condition
1	Skip	$\neg IsStrictInlinable(S,i)$
2	Always Inline	$IsStrictInlinable(S,i)$
3	Multi-level Tiling	$HasDataReuse(S,i)$
4	Multi-level Tiling with Fusion	$HasDataReuse(S,i) \wedge HasFusibleConsumer(S,i)$
5	Add Cache Stage	$HasDataReuse(S,i) \wedge \neg HasFusibleConsumer(S,i)$
6	Reduction Factorization	$HasMoreReductionParallel(S,i)$
...	User Defined Rule	...

Random Annotation Examples

- Parallelize some outer loop
- Vectorize some inner loop
- unroll few inner loop
- randomly fill tile size

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)
```

Learned Cost Model

Predict the score of each non-loop innermost statement

Example:

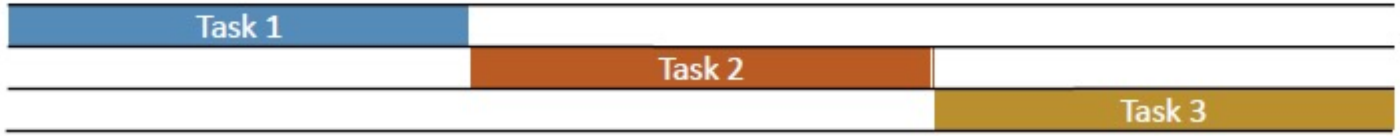
```
              for i in range(10):
                for j in range(10):
Statement B:        B[i][j] = A[i] * 2
              for i in range(10):
Statement C:    C[i] = B[i][i] - 3
```

Cost = Cost of Statement B + Cost of Statement C

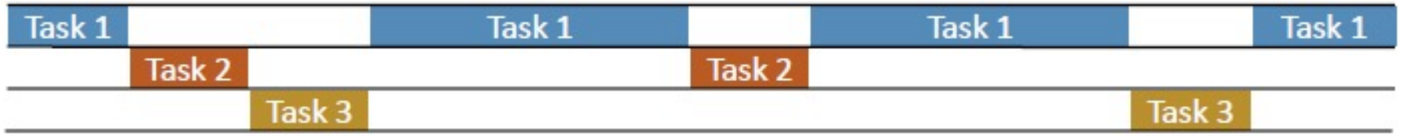
- Extract features for every non-loop innermost statement:
  - used cache lines, used memory, reuse distance, arithmetic intensity, ...
- Train on-the-fly with measured programs (typically less than 30,000)

Task Scheduler

- There are many **subgraphs** (search tasks) in a network
  - Example: ResNet-50 has 29 unique subgraphs after partition
- **Existing systems**: sequential optimization with a fixed allocation



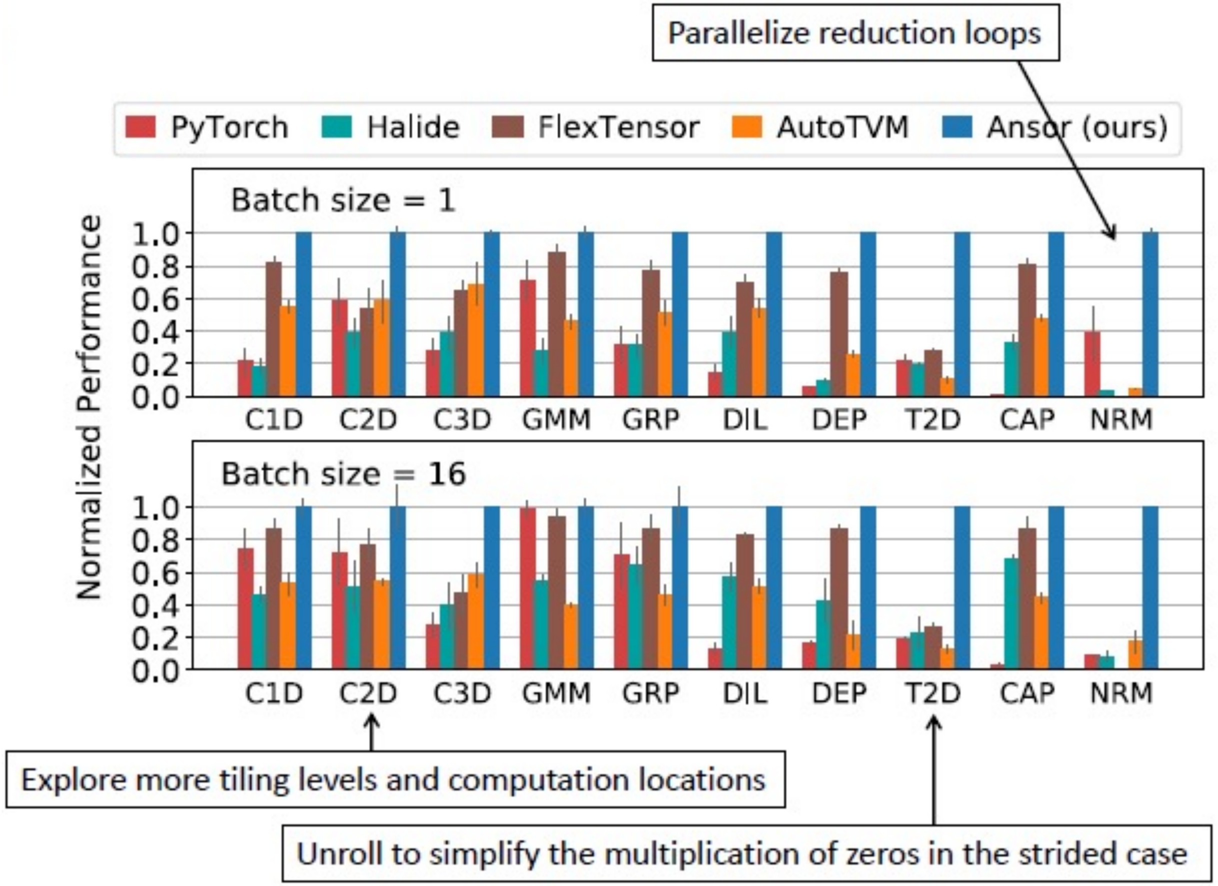
- **Our task scheduler**: slice the time and prioritize important subgraphs



- Predict each task's impact on the end-to-end objective function
  - Using optimistic guess and similarity between tasks

Single Operator

Platform:
Intel-Platinum 8124M (18 cores)
Operators:
conv1d (C1D), conv2d (C2D),
conv3d (C3D), matmul (GMM)
group conv2d (GRP),
dilated conv2d (DIL)
depthwise conv2d (DEP),
conv2d transpose (T2D),
capsule conv2d (CAP),
matrix 2-norm (NRM)

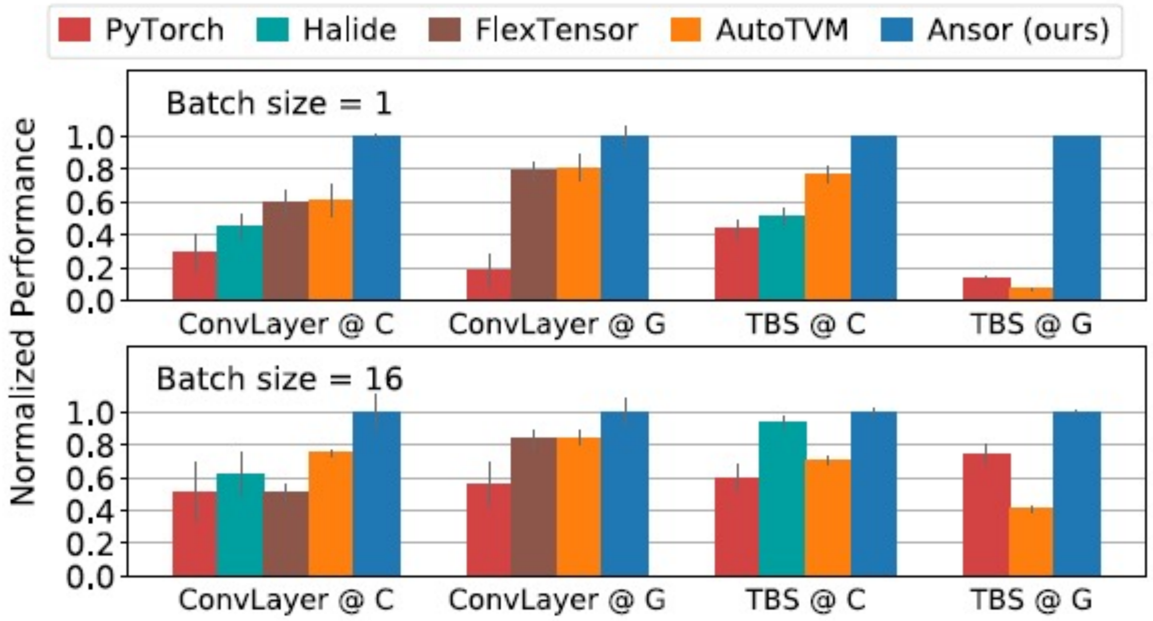


Analysis:
For most test cases, the best programs found by Ansor are outside the search space of existing search-based frameworks.

Subgraph

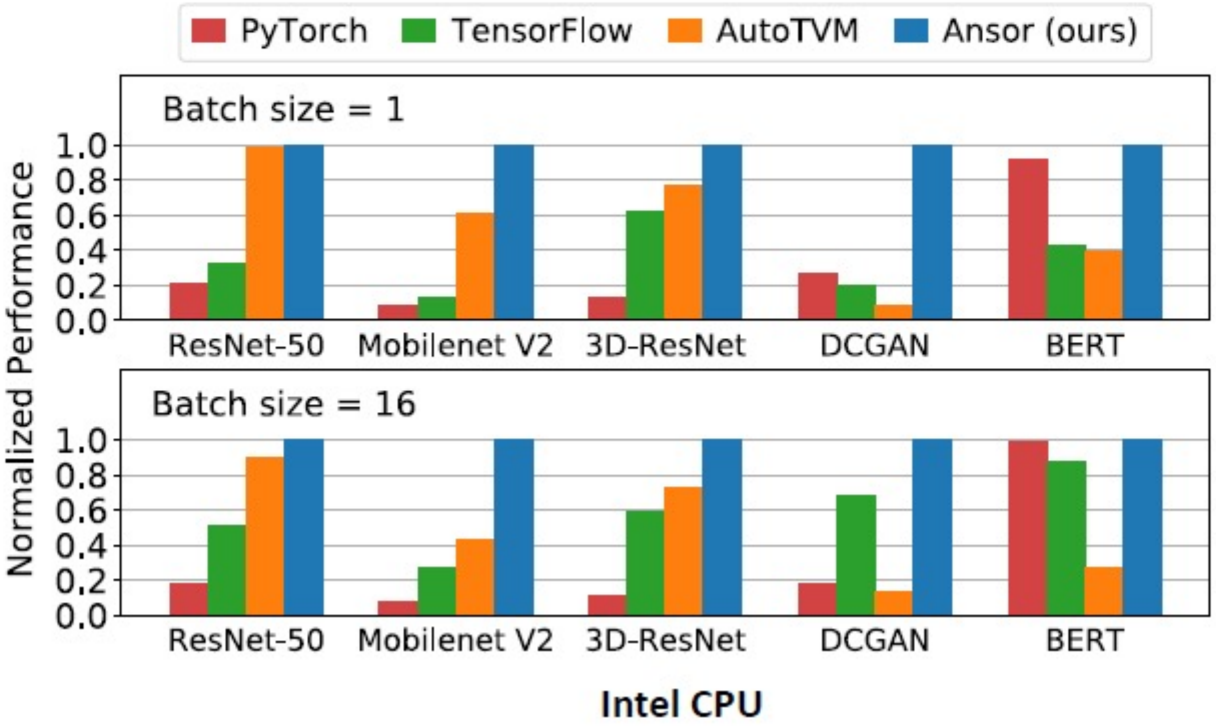
Platforms:
"@C" for Intel CPU (8124M)
"@G" for NVIDIA (V100)
Subgraphs:
ConvLayer = conv2d + bn + relu
TBS = transpose + batch_matmul
+ softmax

Library Version
PyTorch (v1.5 with torch script)
TensorFlow (v2.0 with graph mode)
TensorRT (v6.0 with TensorFlow integration)
TensorFlow Lite (V2.0)



Network

Platforms:
Intel CPU (8124M)
NVIDIA GPU (V100)
ARM CPU (A53)
Networks:
ResNet-50, Mobilenet V2,
3D-ResNet, DCGAN, BERT

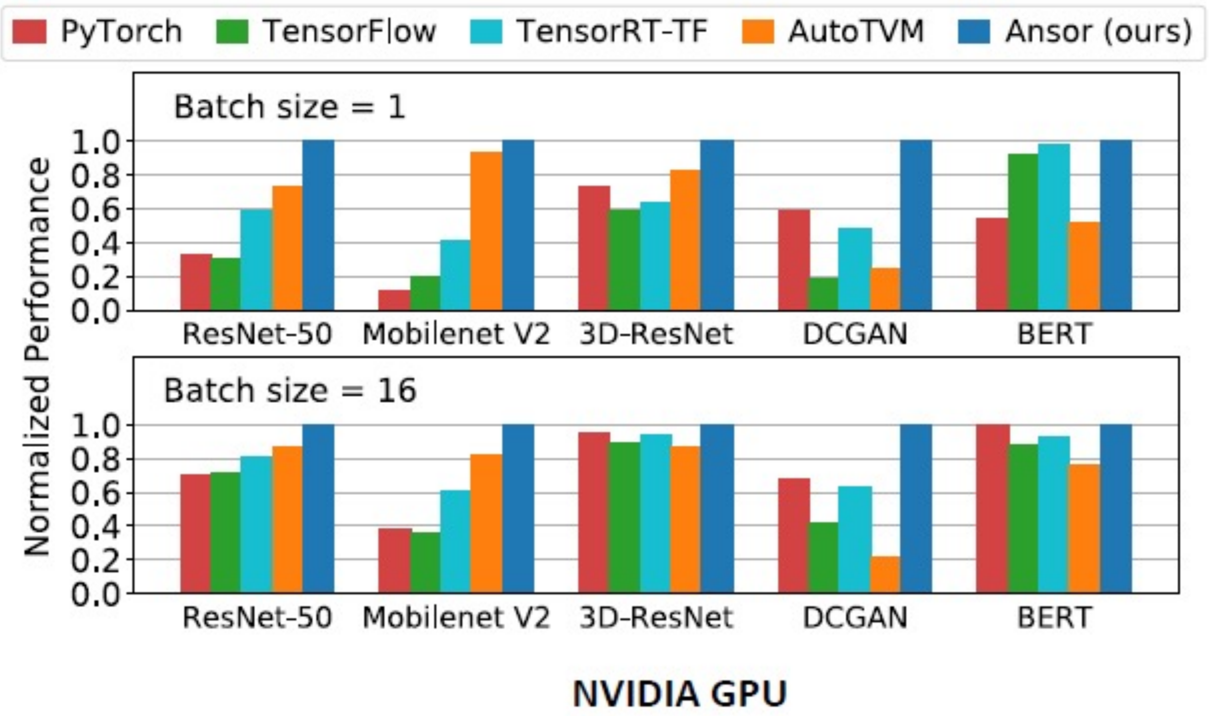


Library Version
PyTorch (v1.5 with torch script)
TensorFlow (v2.0 with graph mode)
TensorRT (v6.0 with TensorFlow integration)
TensorFlow Lite (V2.0)

Analysis
- Ansor performs best or equally the best in all test cases with up to **3.8x** speedup

Network

Platforms:
Intel CPU (8124M)
NVIDIA GPU (V100)
ARM CPU (A53)
Networks:
ResNet-50, Mobilenet V2,
3D-ResNet, DCGAN, BERT

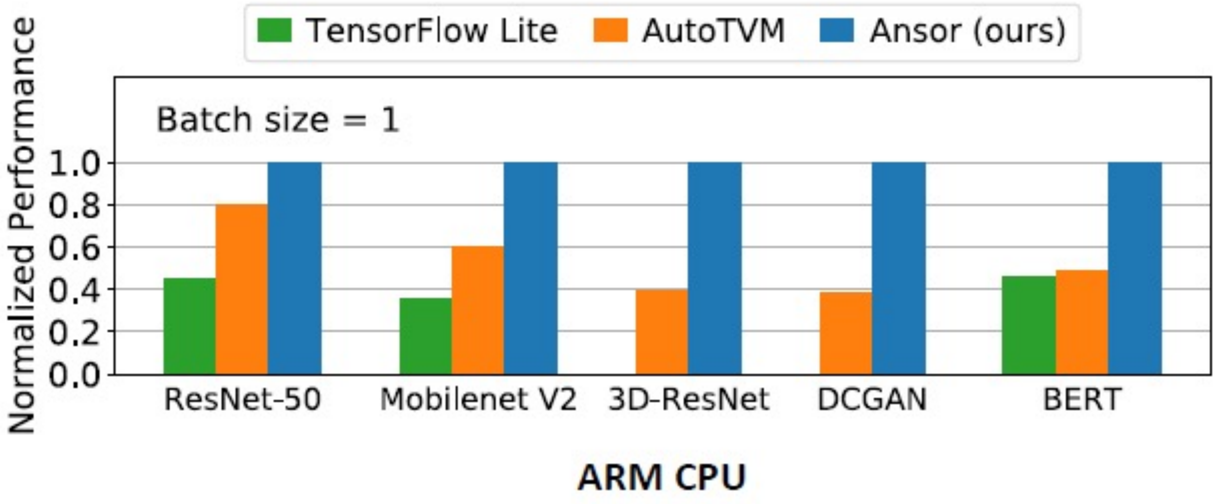


Library Version
PyTorch (v1.5 with torch script)
TensorFlow (v2.0 with graph mode)
TensorRT (v6.0 with TensorFlow integration)
TensorFlow Lite (V2.0)

Analysis
- Ansor performs best or equally the best in all test cases with up to **3.8x** speedup

Network

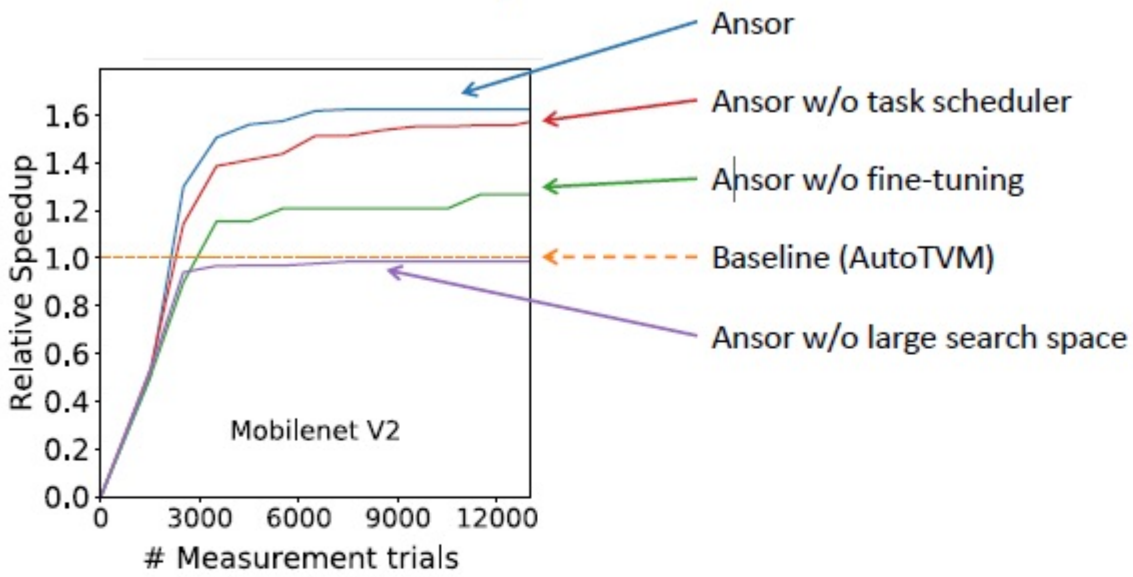
Platforms:
Intel CPU (8124M)
NVIDIA GPU (V100)
ARM CPU (A53)
Networks:
ResNet-50, Mobilenet V2,
3D-ResNet, DCGAN, BERT



Library Version
PyTorch (v1.5 with torch script)
TensorFlow (v2.0 with graph mode)
TensorRT (v6.0 with TensorFlow integration)
TensorFlow Lite (V2.0)

Analysis
- Ansor performs best or equally the best in all test cases with up to **3.8x** speedup
- Ansor delivers portable performance

Ablation Study



Analysis

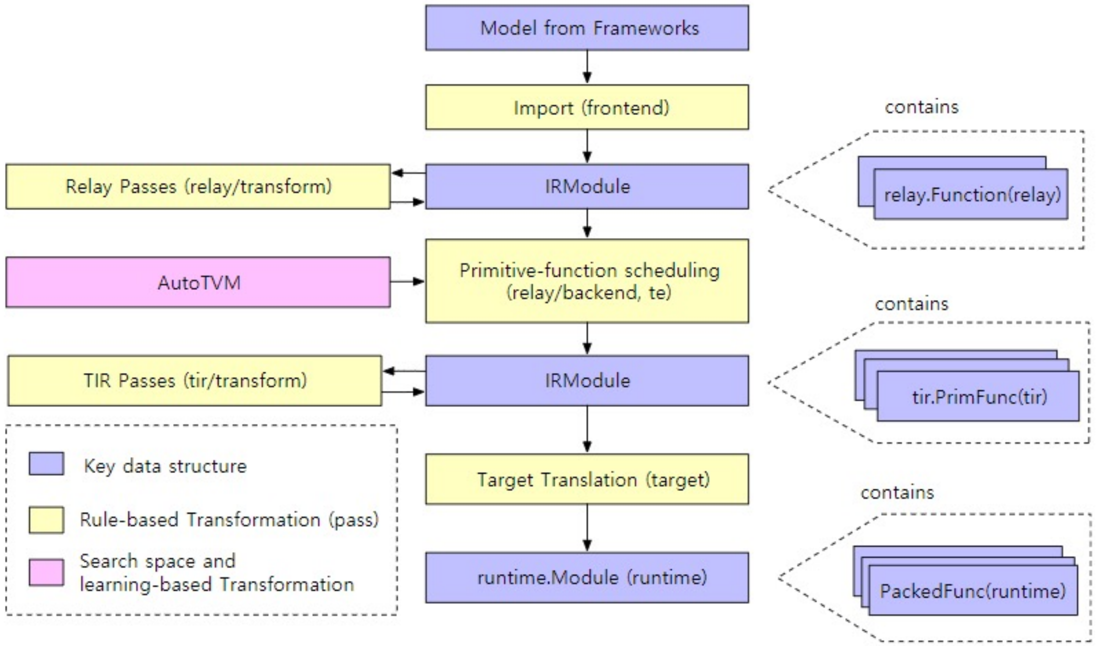
- The most important factor is the search space
- Fine-tuning improves the search results significantly
- Task scheduler accelerates the search
- Match the performance of AutoTVM with 10x less search time

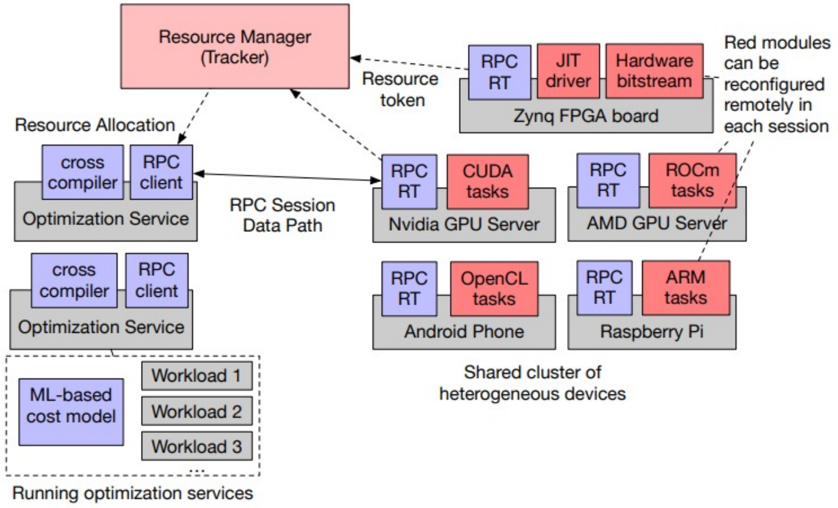
Use case

- AutoTVM
  - Well known parameter space
  - Special operation(quantization,Winograd) and hardware(tensor-core)

- AutoScheduler
  - The effect of the parameter is unclear
  - Need more potential performance gain
  - when have a problem creating a sufficient template

Use case





- AutoTVM template
- Rule based transformation
- Runtime

Use case

```
# extract workloads from relay program
print("Extract tasks...")
mod, params, input_shape, _ = get_network(network, batch_size=1)
tasks = autotvm.task.extract_from_program(
    mod["main"],
    target=target,
    params=params,
    ops=(relay.op.get("nn.conv2d"),),
)
```

Use case

```python
#### DEVICE CONFIG ####
target = tvm.target.cuda()

#### TUNING OPTION ####
network = "resnet-18"
log_file = "%s.log" % network
dtype = "float32"

tuning_option = {
    "log_filename": log_file,
    "tuner": "xgb",
    "n_trial": 2000,
    "early_stopping": 600,
    "measure_option": autotvm.measure_option(
        builder=autotvm.LocalBuilder(timeout=10),
        runner=autotvm.LocalRunner(number=20, repeat=3, timeout=4, min_repeat_ms=150),
    ),
}
```

AutoTVM

```python
print("Begin tuning...")
measure_ctx = auto_scheduler.LocalRPCMeasureContext(repeat=1, min_repeat_ms=300, timeout=10)

tuner = auto_scheduler.TaskScheduler(tasks, task_weights)
tune_option = auto_scheduler.TuningOptions(
    num_measure_trials=200,  # change this to 20000 to achieve the best performance
    runner=measure_ctx.runner,
    measure_callbacks=[auto_scheduler.RecordToFile(log_file)],
)

tuner.tune(tune_option)
```

Ansor

Tips

- Search space design
- # of trial
- Kinds of exploration algorithm
- Cost model
- Hyper-parameter

Thanks

References

- Learning to optimize tensor programs paper
- Learning to optimize tensor programs slide
- Dive into Deep Learning Compiler
- Ansor slide
- Ansor paper
- Tvm docs