# Ansor : Generating High-Performance Tensor Programs for Deep Learning

**Lianmin Zheng**, Chenfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph Gonzalez, Ion Stoica

Berkeley UNIVERSITY OF CALIFORNIA

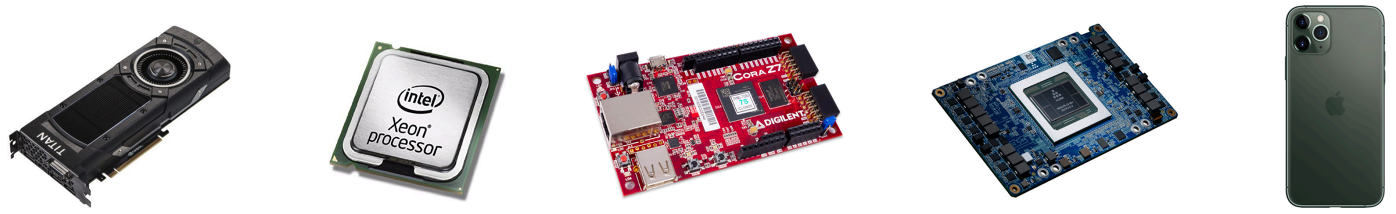
Alibaba Group

aws

# Deep Learning System Stack

PyTorch TensorFlow mxnet

**Operator Library** (CuDNN, MKL-DNN, ARM Compute Library, ...)



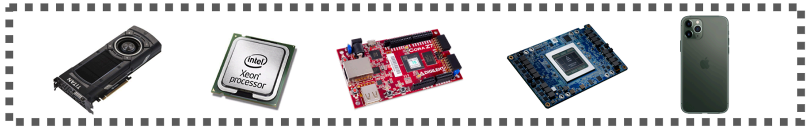
# Introducing Compiler

PyTorch TensorFlow mxnet

Compute declaration

**compiler**

Optimized code



A dense layer with ReLU activation

- **Math expression:**

$$dense_{b,o} = \sum_i data_{b,i} \times weight_{o,i}$$

$$relu(b,o) = \max(dense_{b,o}, 0)$$

- **Declaration**:

Halide

```
dense(o, b) += data(i, b) * weight(i, o);
relu(o, b) = max(dense(o, b), 0.0)
```

TVM

```
dense = compute(shape, lambda b, o: sum(data[b,i] * weight[o,i], i))
relu  = compute(shape, lambda b, o: max(dense[b,o], 0.0))
```

**Billions of possible implementations for it!**

# Related Work on Generating High-Performance Tensor Programs

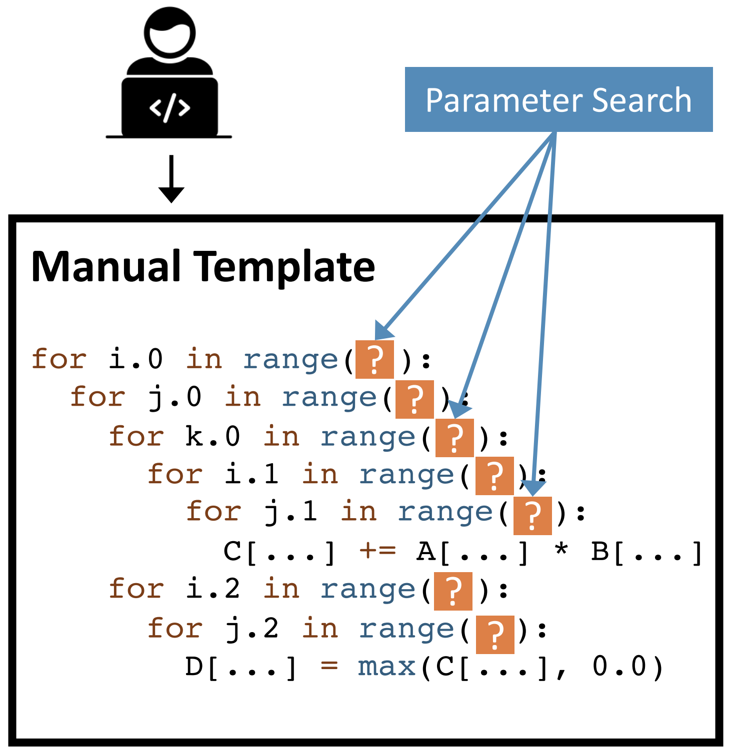
# TVM's Approach

## AutoTVM: Template-guided search

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

## Drawbacks

- Not fully-automated -> Requires huge manual effort
- Limited search space -> Does not achieve optimal performance



# Halide’s Auto-scheduler

## Sequential Construction Based Search

Use beam search to generate the programs sequentially

## Drawbacks

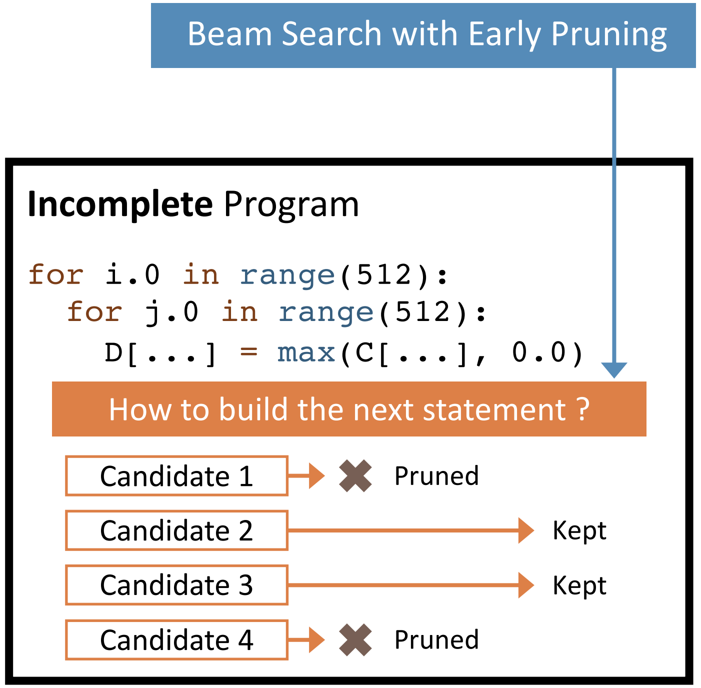
- Intermediate candidates are incomplete programs

  -> The cost model cannot do accurate prediction

- Sequential order

  -> The error accumulates

  -> Limits the design of the search space



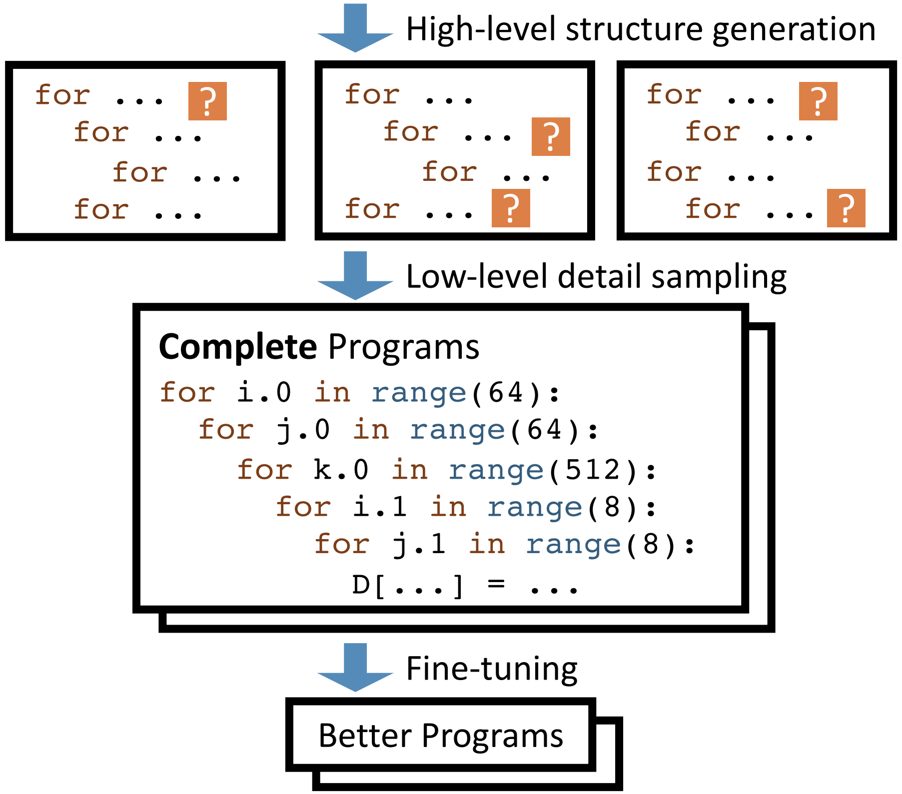
# Challenges and our approach

**C1: How to build a large search space automatically?**

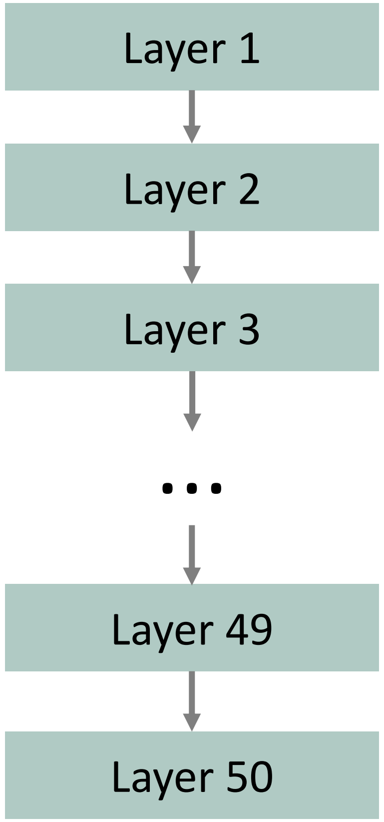
- Use a hierarchical search space

**C2: How to search efficiently?**

- Sample complete programs and fine-tune them



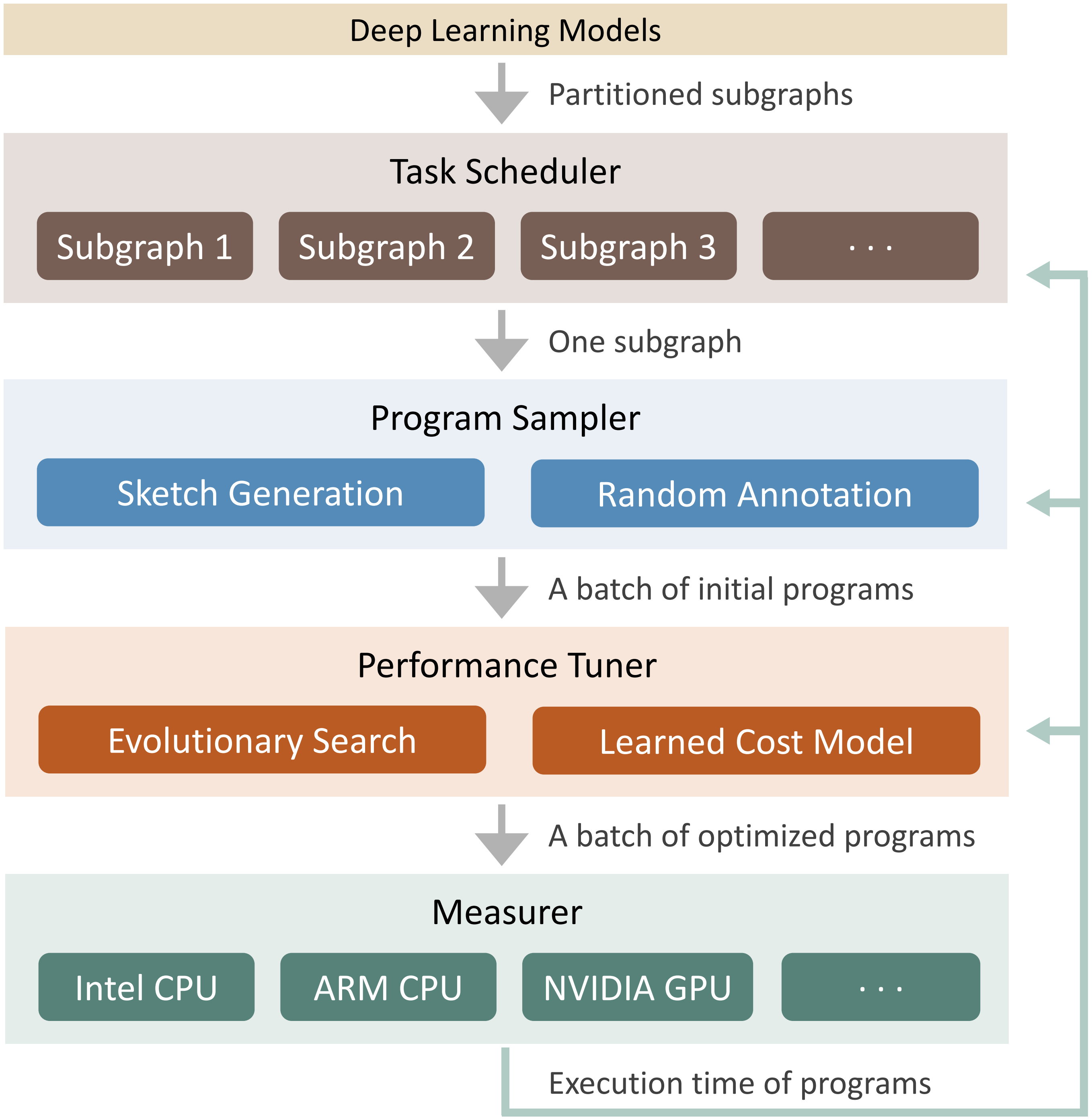
# Challenges and our approach



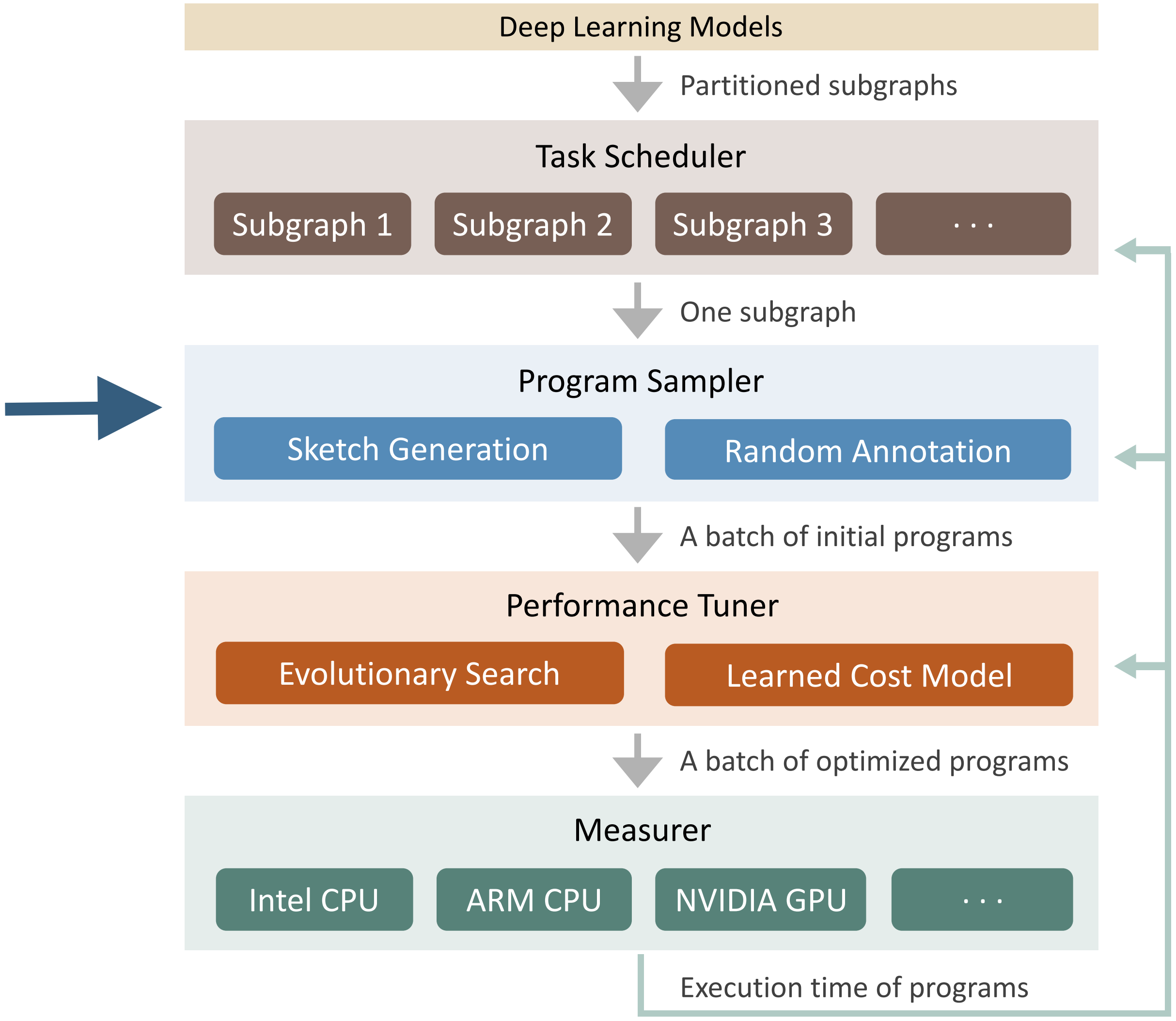

Need to generate programs for all layers -> A lot of search tasks

- **C3: How to allocate resource for many search tasks?**
  - Utilize a task scheduler to prioritize important tasks

# System Overview


Deep Learning Models
Partitioned subgraphs
Task Scheduler
Subgraph 1
Subgraph 2
Subgraph 3
· · ·
One subgraph
Program Sampler
Sketch Generation
Random Annotation
A batch of initial programs
Performance Tuner
Evolutionary Search
Learned Cost Model
A batch of optimized programs
Measurer
Intel CPU
ARM CPU
NVIDIA GPU
· · ·
Execution time of programs

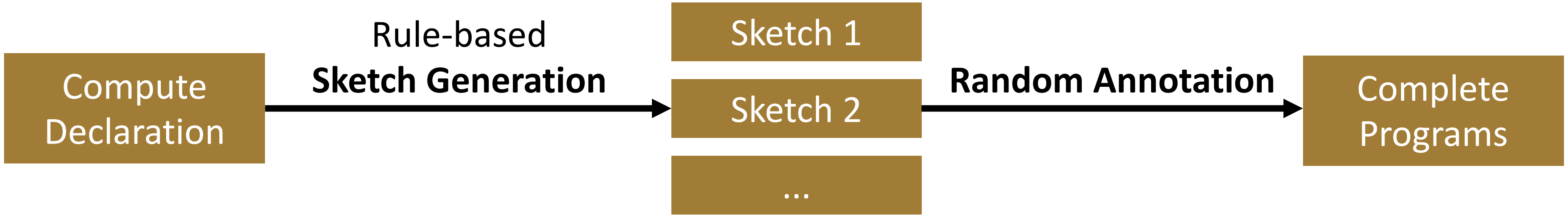
# Program Sampling



# 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 1/2

Example Input 1:

*** The mathmetical expression:**

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

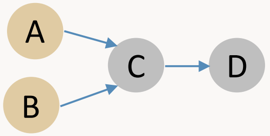
$$D[i,j] = \max(C[i,j], 0.0)$$

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

*** 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 Examples 2/2

**Example Input 2:**

*** The mathmetical expression:**

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

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

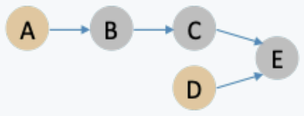
$$E[i,j] = \sum_k C[i,k] \times D[k,j]$$

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[...]
```

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

**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[...]
```

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

# 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)
```

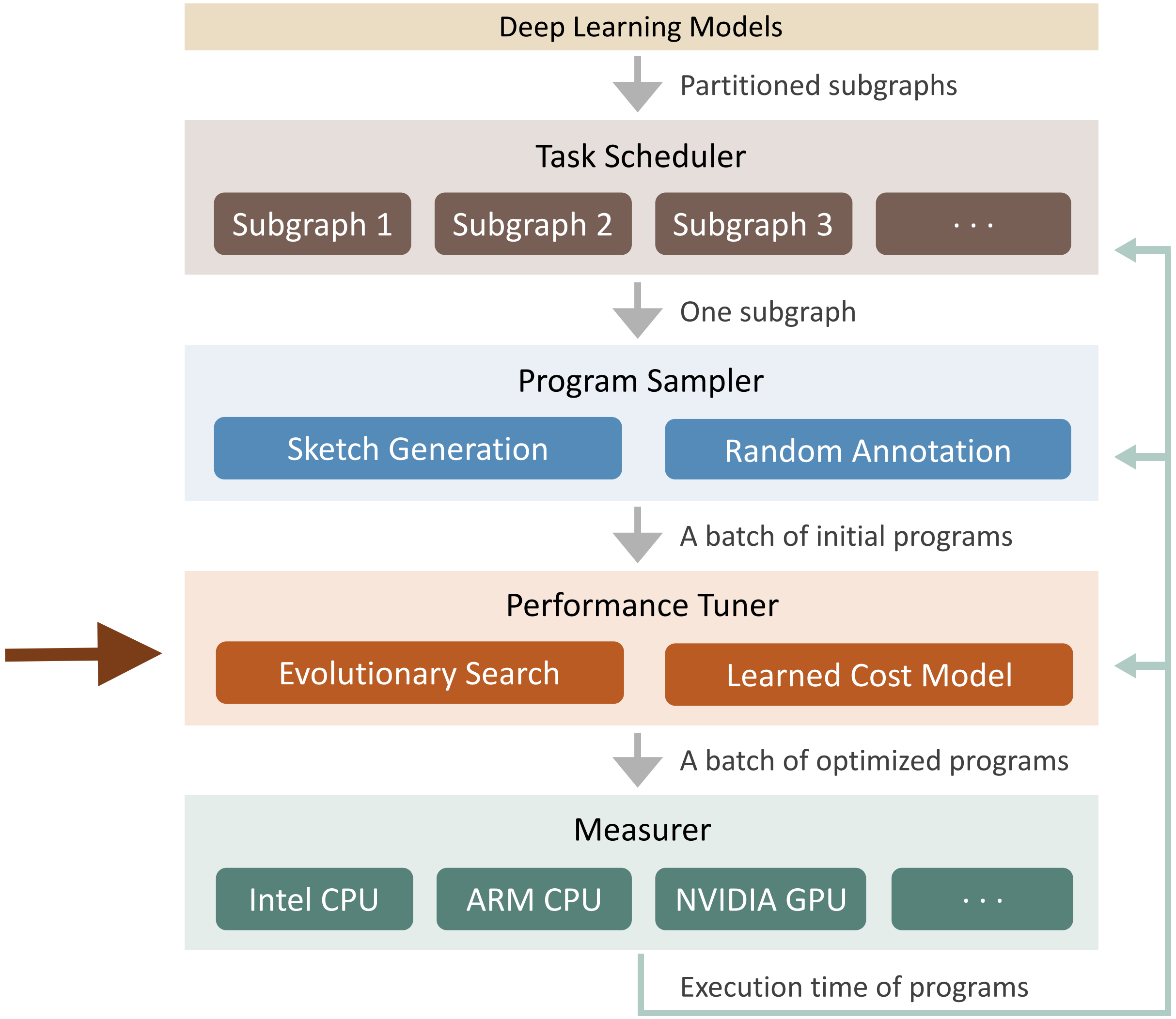
Sampled program 1

```
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

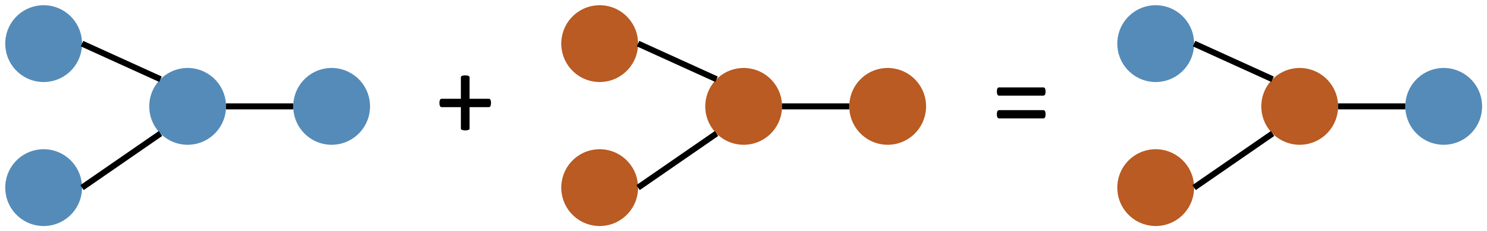
```
parallel i.2 in range(16):
  for j.2 in range(128):
    for k.1 in range(512):
      for i.3 in range(32):
        vectorize j.3 in range(4):
          C[...] += A[...] * B[...]
parallel i.4 in range(512):
  for j.4 in range(512):
    D[...] = max(C[...], 0.0)
```

# Performance Fine-tuning



# Evolutionary Search

- Random sampling does not guarantee the performance
- Perform evolutionary search with learned cost model on sampled programs
  - **mutation**
    - Randomly mutate tile size
    - Randomly mutate parallel/unroll/vectorize factor and granularity
    - Randomly mutate computation location
  - **crossover**



# Learned Cost Model

- Predict the score of each non-loop innermost statement

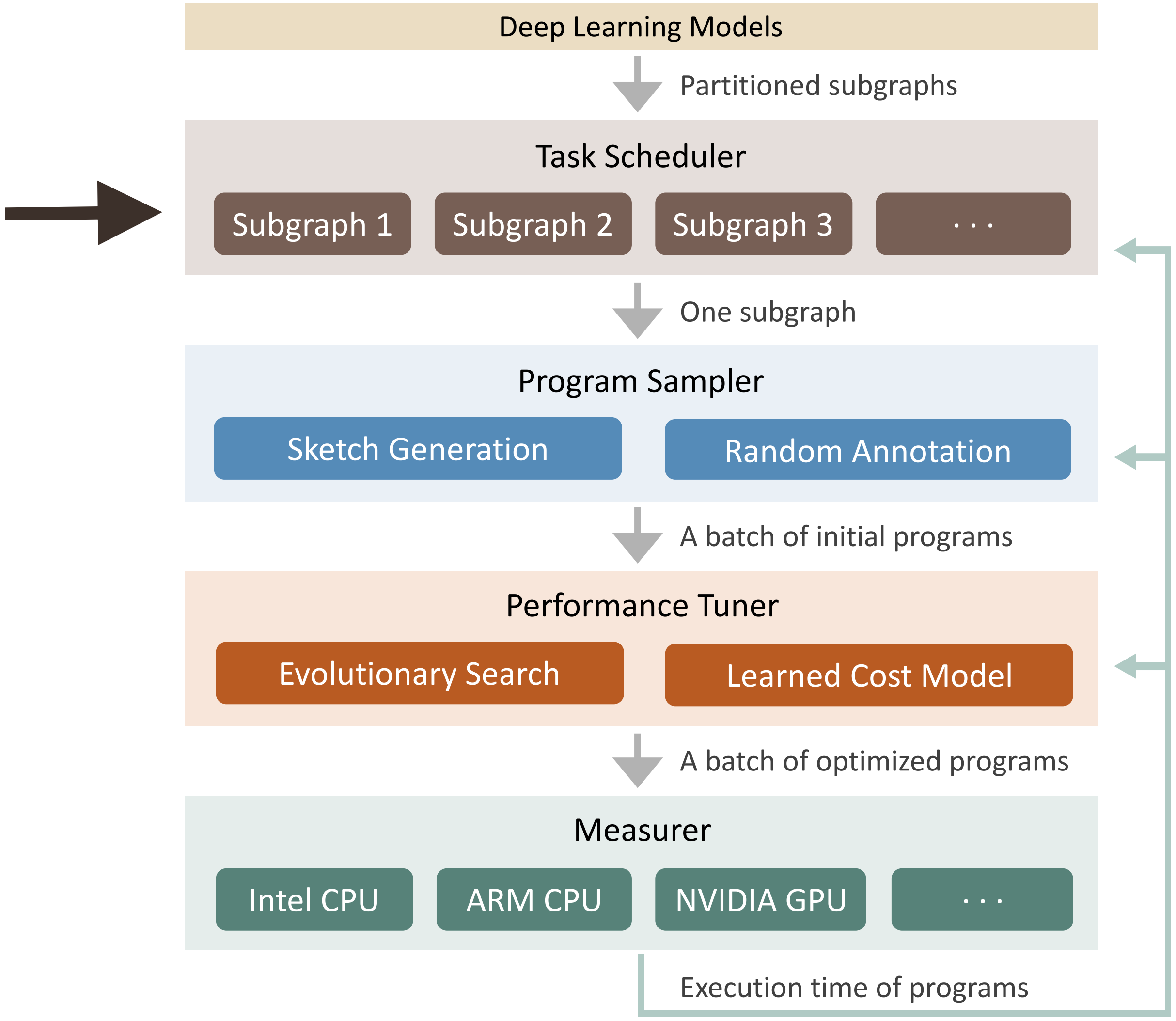
**Example:**

```
                     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

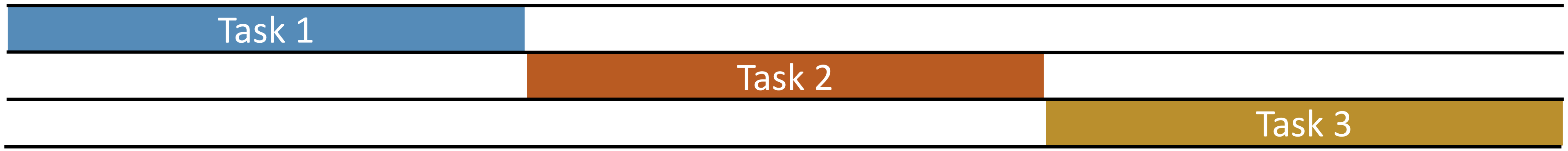
- Extract features for every non-loop innermost statement:
  - used cache lines, used memory, reuse distance, arithmetic intensity, ...
- Train on-the-fly with measured programs (typically less than 30,000)

# Task Scheduler

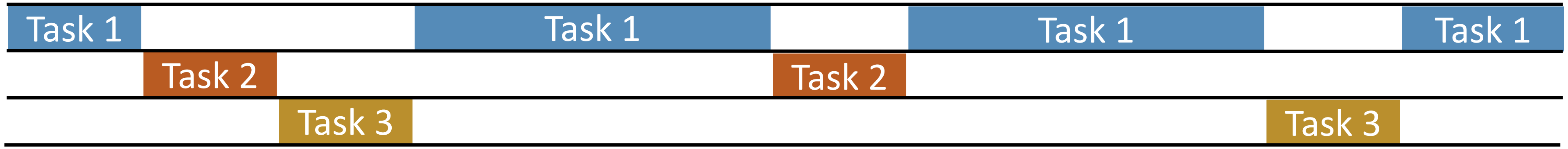


# Task Scheduler

- 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

# Evaluation Results

**Three levels** : single operator, subgraph and network

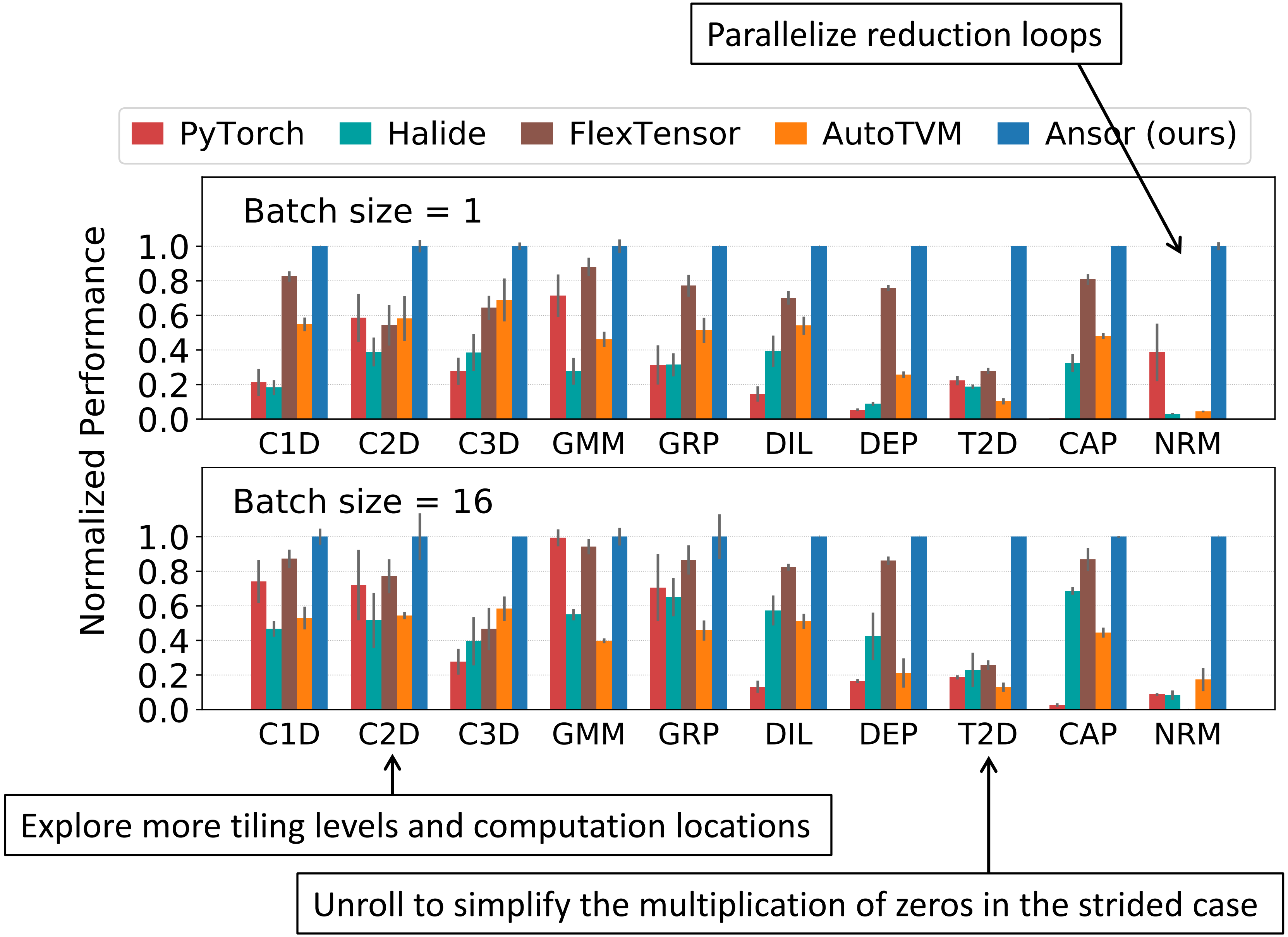
# Single Operator

**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:**

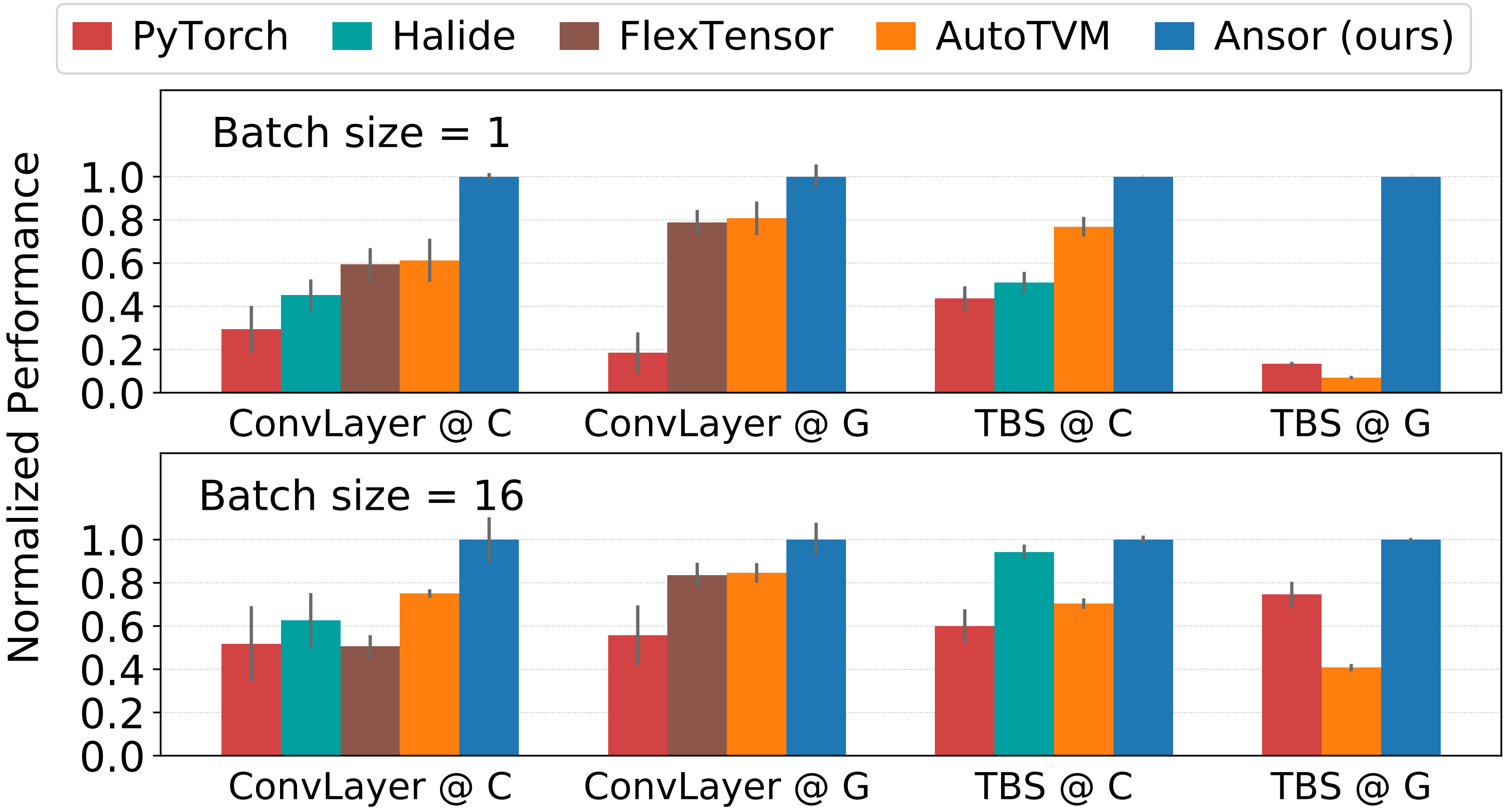
"@C" for Intel CPU (8124M)

"@G" for NVIDIA (V100)

**Subgraphs:**

ConvLayer = conv2d + bn + relu

TBS = transpose + batch_matmul + softmax




**Analysis:**

Comprehensive coverage of the search space gives 1.1 – 14.2× speedup against the best alternative.

# Network

**Platforms:**

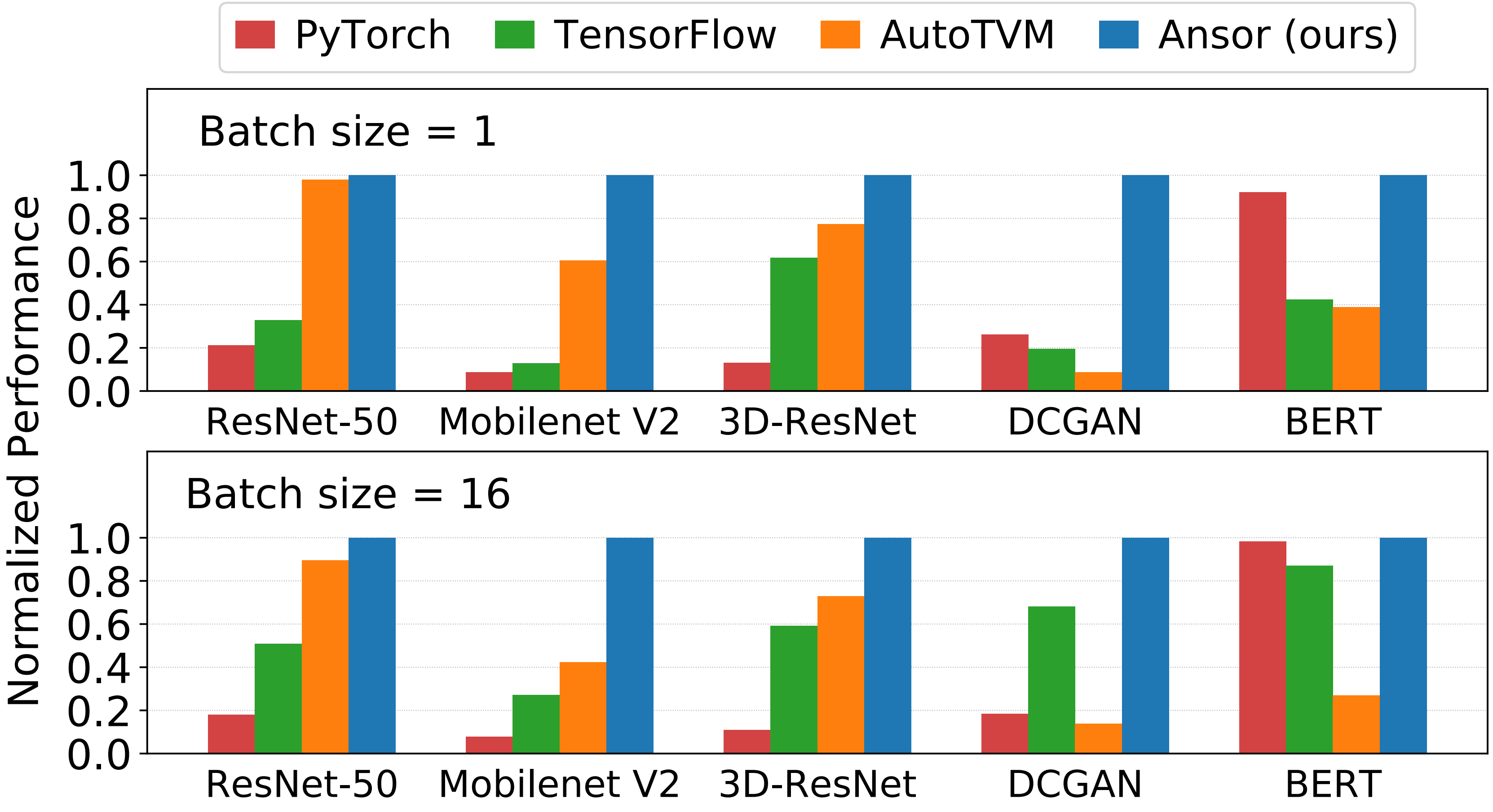
Intel CPU (8124M)

NVIDIA GPU (V100)

ARM CPU (A53)

**Networks:**

ResNet-50, Mobilenet V2, 3D-ResNet, DCGAN, BERT




**Intel CPU**

**Analysis**

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

# Network

**Platforms:**

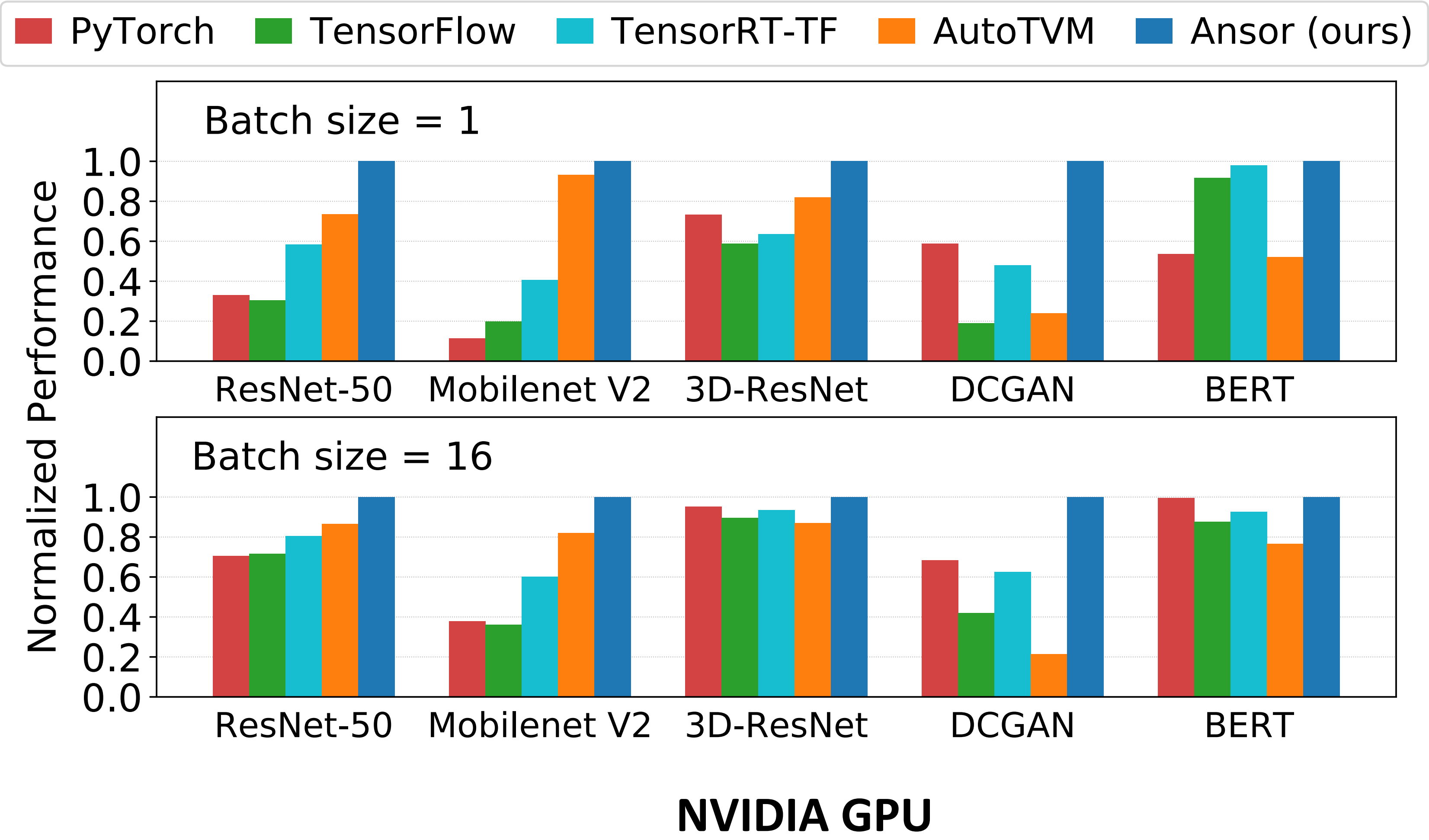
Intel CPU (8124M)

NVIDIA GPU (V100)

ARM CPU (A53)

**Networks:**

ResNet-50, Mobilenet V2, 3D-ResNet, DCGAN, BERT




**Analysis**

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

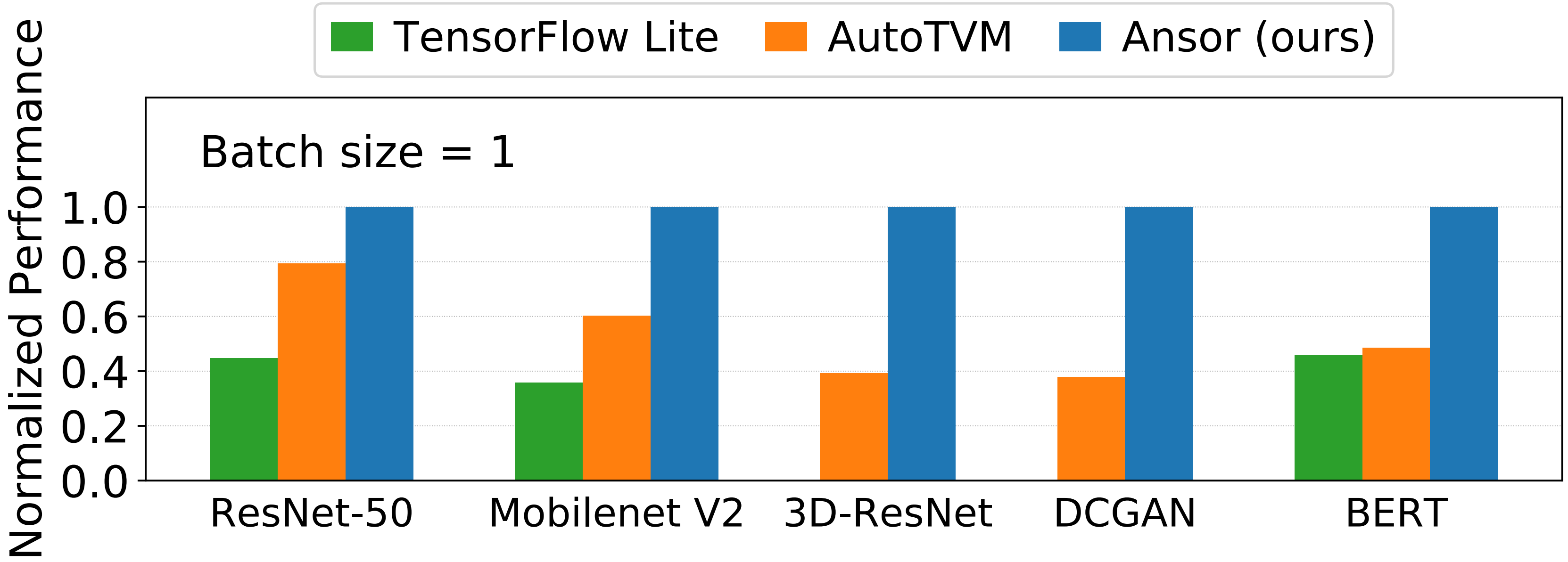
# Network

**Platforms:**

Intel CPU (8124M)
NVIDIA GPU (V100)
ARM CPU (A53)

**Networks:**

ResNet-50, Mobilenet V2,
3D-ResNet, DCGAN, BERT

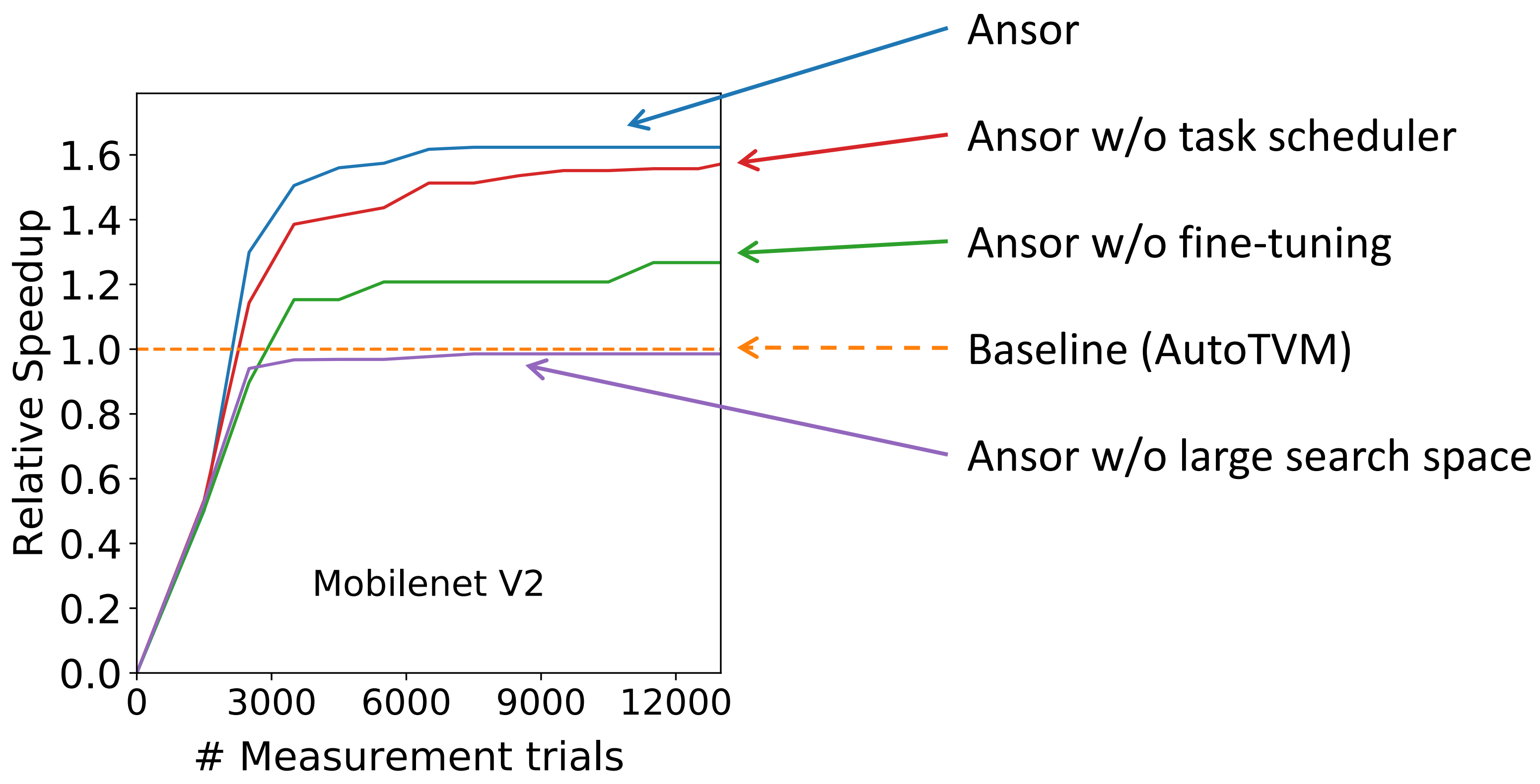



**ARM CPU**

**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

# Summary

- Search-based compilation enables to generate high-performance tensor programs for deep learning

- Ansor introduces techniques to improve the search in three aspects:
  - Large search space
  - Efficient search algorithm
  - Smart search scheduling

- Thank you for listening
- Email me to ask follow-up questions: lianminzheng@gmail.com