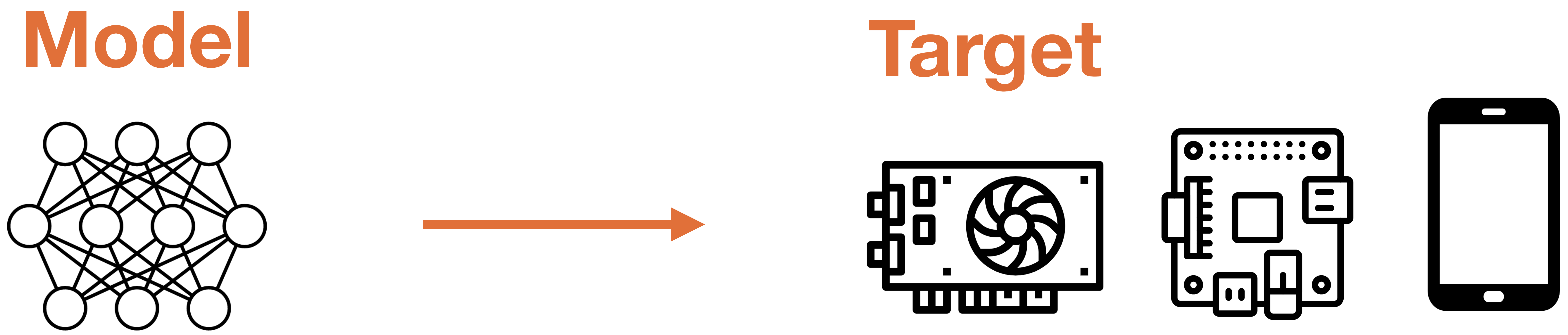
Learning to Optimize Tensor Programs

Tianqi Chen, Lianmin Zheng, Eddie Yan, Ziheng Jiang, Thierry Moreau, Luis Ceze, Carlos Guestrin, Arvind Krishnamurthy

Elias Jaeaesaari 2/9/2022

Overview

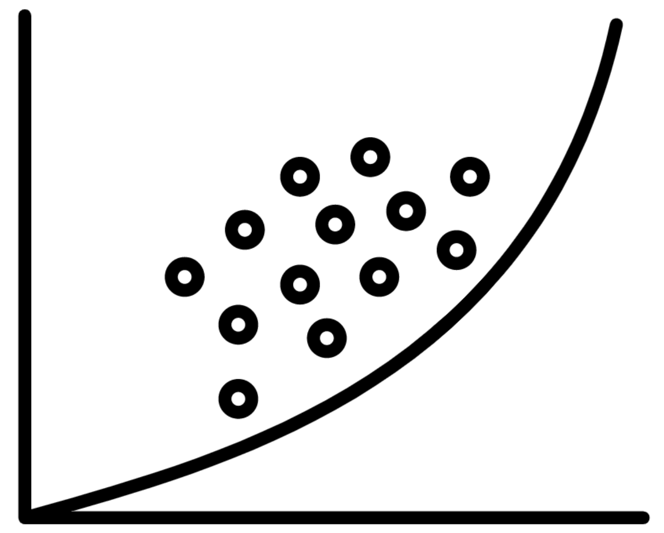
We want to efficiently deploy models to a variety of platforms



Overview

Recall: TVM allows us to separate the algorithm (what is computed) of a program from its schedule (how it is computed)

AutoTVM finds the best schedule automatically by using



Cost-model based search

Transfer learning

The problem

$$C_{ij} = \sum_k A_{ki} B_{kj}$$

Possibly millions of programs!

```
for y in range(128):
  for x in range(128):
    C[y][x] = 0
    for k in range(128):
      C[y][x] += A[k][y] * B[k][x]
```

```
for yo in range(1024 / ty):
  for xo in range(1024 / tx):
    C[yo*ty:yo*ty+ty][xo*tx:xo*tx+x] = 0
    for k in range(1024):
      for yi in range(ty):
        for xi in range(tx):
          C[yo*ty+yi][xo*tx+xi] +=
            A[k][yo*ty+yi] * B[k][xo*tx+xi]
```

```
for yo in range(128):
  for xo in range(128):
    intrin.fill_zero(C[yo*8:yo*8+8][xo*8:xo*8+8])
    for ko in range(128):
      intrin.fused_gemm8x8_add(
        C[yo*8:yo*8+8][xo*8:xo*8+8],
        A[yo*8:yo*8+8][xo*8:xo*8+8],
        B[yo*8:yo*8+8][xo*8:xo*8+8])
```

...

How do we efficiently explore the space of possible programs?

Hyperparameter optimization

Find the best set of hyperparameters by optimizing a costly to evaluate fitness function f

Sequential Model-based Global Optimization

1. Choose $x^\star$ according to a surrogate model
2. Evaluate $f(x^\star)$
3. Add $(x, f(x^\star))$ to training data
4. Fit a new surrogate model
5. Repeat

E.g. Gaussian process or tree-structured Parzen estimator

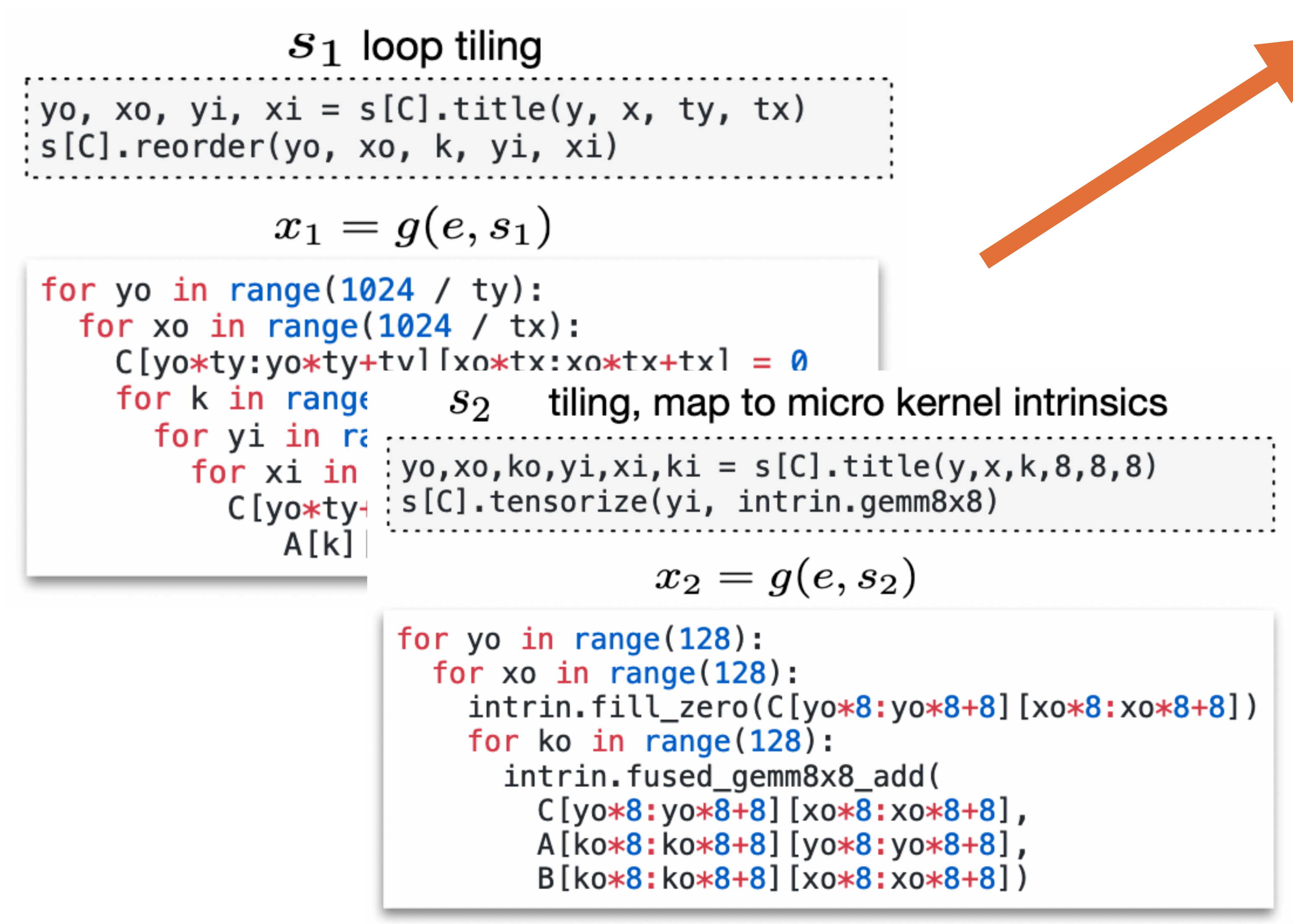
Assume f is a black-box and expensive to evaluate

AutoTVM

Expression

$$C_{ij} = \sum_k A_{ki} B_{kj}$$

Schedule space



s_1 loop tiling

```
yo, xo, yi, xi = s[C].title(y, x, ty, tx)
s[C].reorder(yo, xo, k, yi, xi)
```

$x_1 = g(e, s_1)$

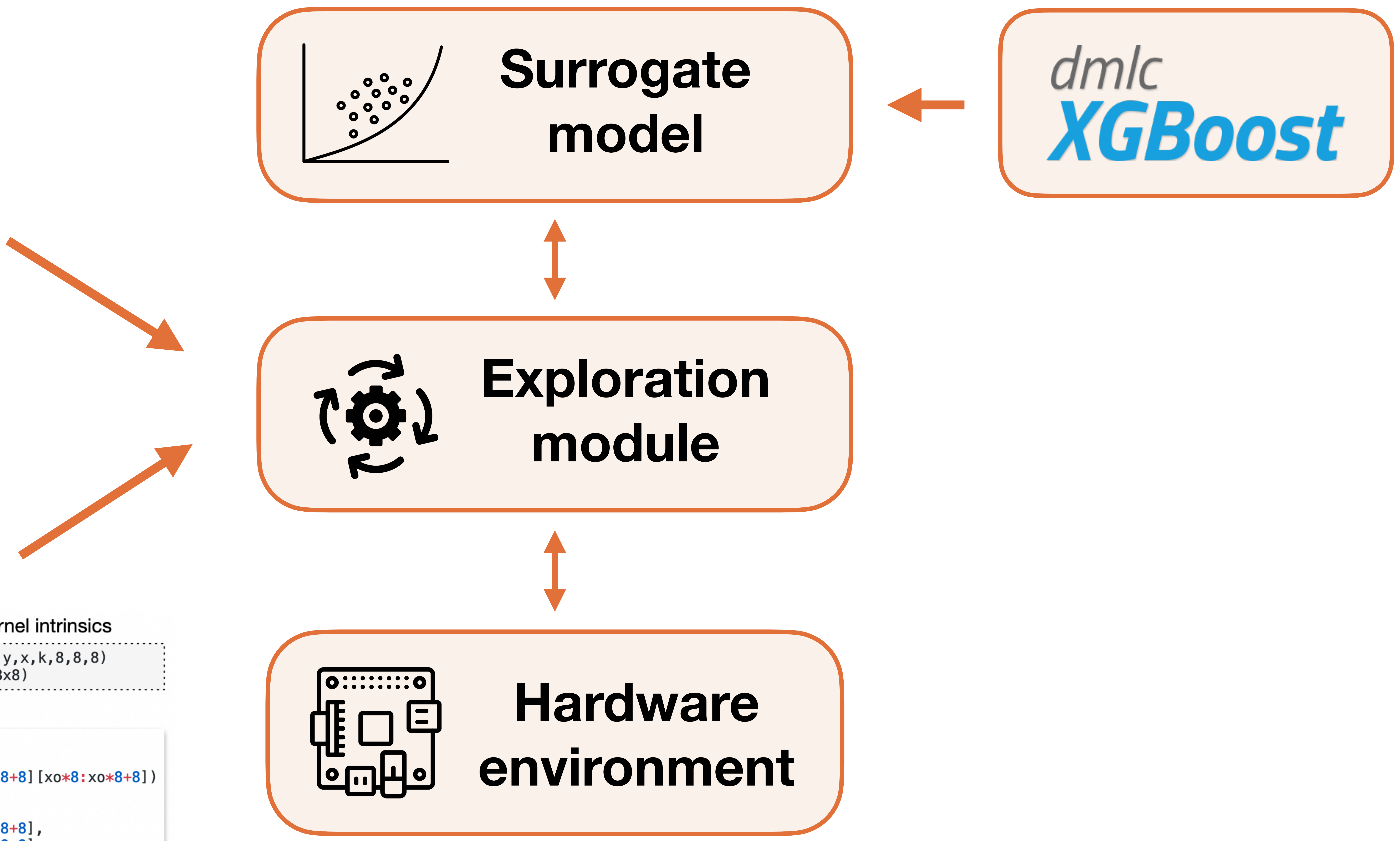
```
for yo in range(1024 / ty):
  for xo in range(1024 / tx):
    C[yo*ty:yo*ty+ty][xo*tx:xo*tx+tx] = 0
    for k in range
      for yi in ra
        for xi in
          C[yo*ty
            A[k]
```

s_2 tiling, map to micro kernel intrinsics

```
yo,xo,ko,yi,xi,ki = s[C].title(y,x,k,8,8,8)
s[C].tensorize(yi, intrin.gemm8x8)
```

$x_2 = g(e, s_2)$

```
for yo in range(128):
  for xo in range(128):
    intrin.fill_zero(C[yo*8:yo*8+8][xo*8:xo*8+8])
    for ko in range(128):
      intrin.fused_gemm8x8_add(
        C[yo*8:yo*8+8][xo*8:xo*8+8],
        A[ko*8:ko*8+8][yo*8:yo*8+8],
        B[ko*8:ko*8+8][xo*8:xo*8+8])
```

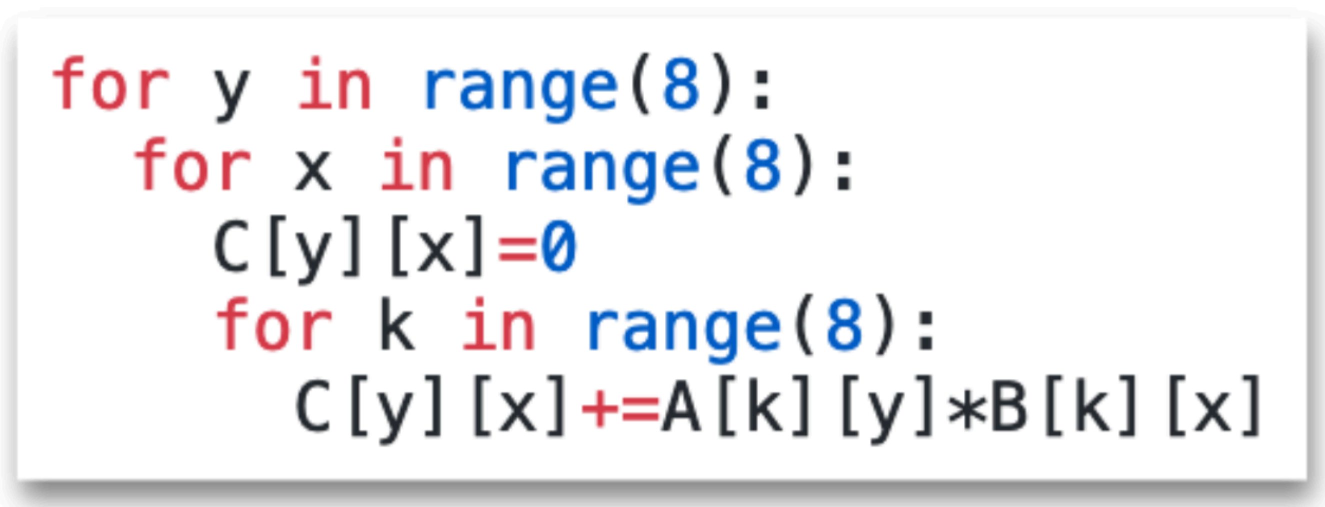


Feature representation

Extract features from AST for every loop variable:

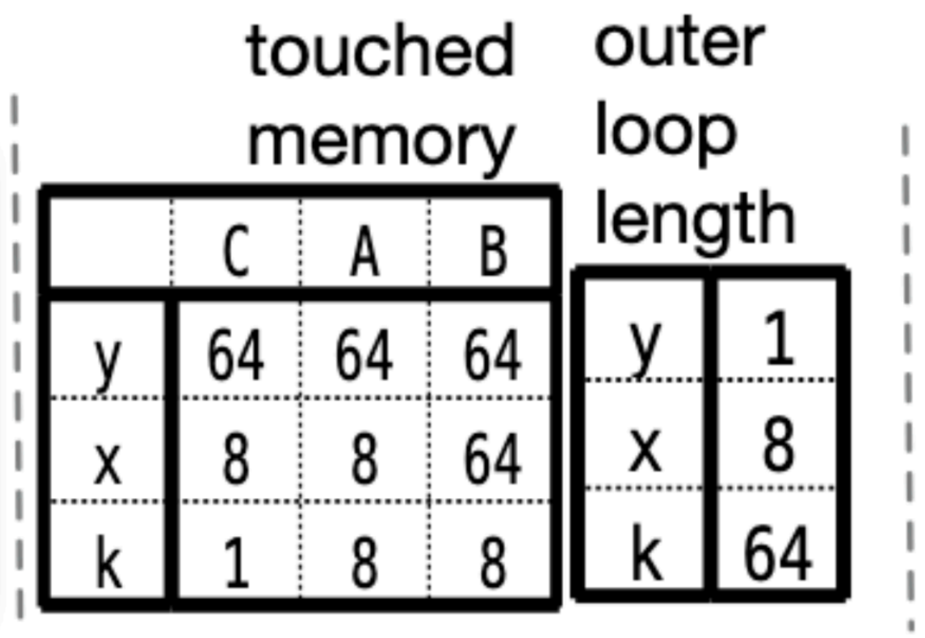
Loop length, vectorizable, parallelizable, …

For every buffer: touch count, reuse ratio, …



```
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	A	B
y	64	64	64
x	8	8	64
k	1	8	8

outer loop length

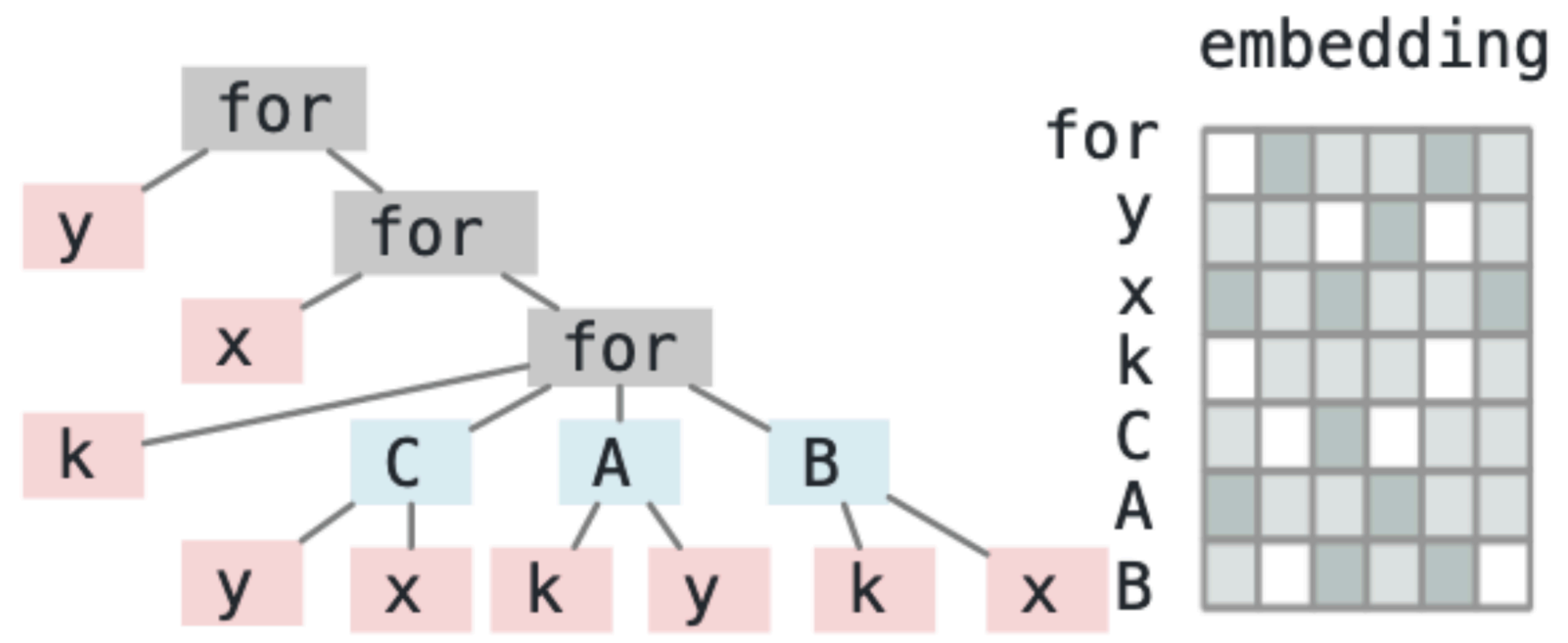
y	1
x	8
k	64

(b) Loop context vectors

Feature representation

Alternative representation using TreeGRU

Recursively encode AST into an embedding vector



(c) Vanilla TreeGRU

Choosing candidates

Encourage diversity by maximizing

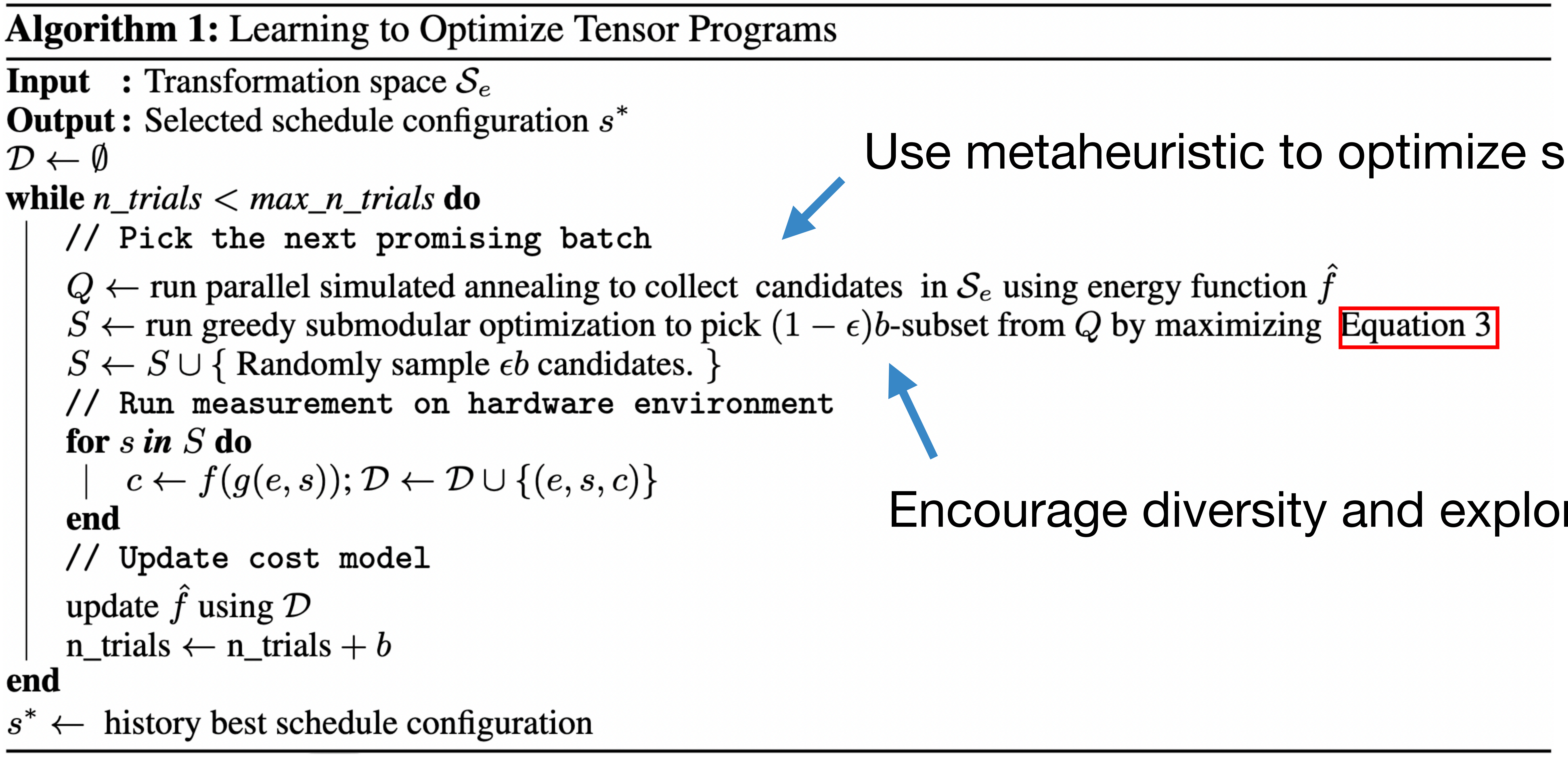
$$L(S) = -\sum_{s \in S} \text{surrogate\_model}(s) + \alpha \times (\text{\# configuration components covered by } S)$$

Balance quality

Balance diversity

Encourage exploration by choosing ϵb candidates randomly

Putting it all together



Algorithm 1: Learning to Optimize Tensor Programs

```
Input  : Transformation space $\mathcal{S}_e$
Output : Selected schedule configuration $s^*$
$\mathcal{D} \leftarrow \emptyset$
while $n\_trials < max\_n\_trials$ do
    // Pick the next promising batch
    $Q \leftarrow$ run parallel simulated annealing to collect candidates in $\mathcal{S}_e$ using energy function $\hat{f}$
    $S \leftarrow$ run greedy submodular optimization to pick $(1-\epsilon)b$-subset from $Q$ by maximizing Equation 3
    $S \leftarrow S \cup \{$ Randomly sample $\epsilon b$ candidates. $\}$
    // Run measurement on hardware environment
    for $s$ in $S$ do
        $c \leftarrow f(g(e, s)); \mathcal{D} \leftarrow \mathcal{D} \cup \{(e, s, c)\}$
    end
    // Update cost model
    update $\hat{f}$ using $\mathcal{D}$
    n_trials $\leftarrow$ n_trials $+ b$
end
$s^* \leftarrow$ history best schedule configuration
```

Use metaheuristic to optimize surrogate

Encourage diversity and exploration

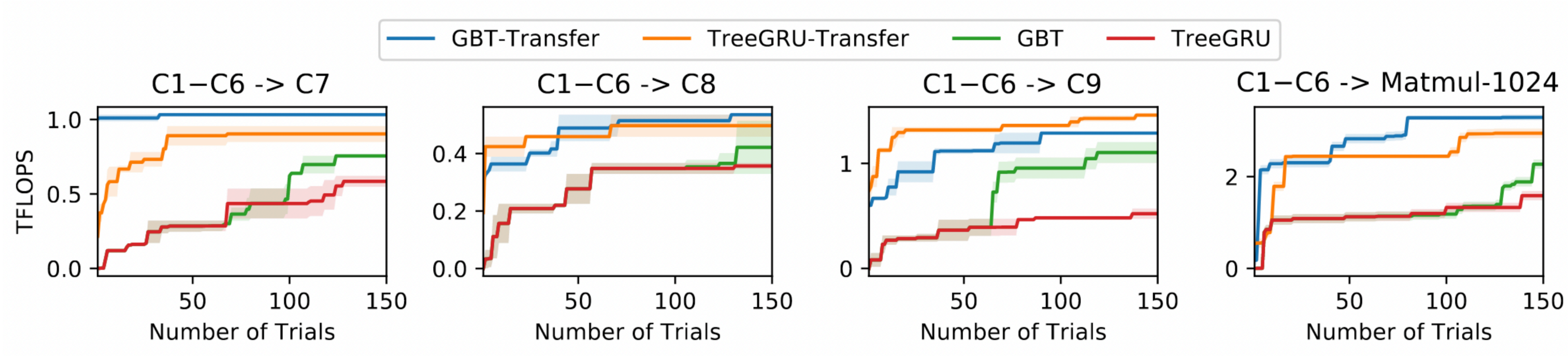
Transfer learning

In practice, we need to optimize many workloads

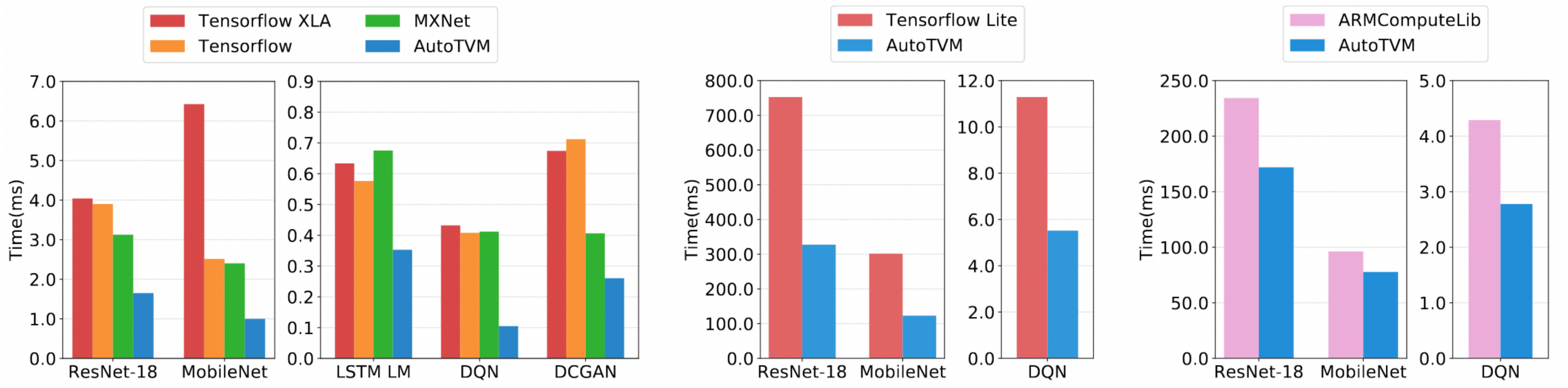
Transfer what has been learned from previous workloads:

1. **Use a transferable representation**
2. **Combine** $\hat{f}(x) = \hat{f}^{(\text{global})}(x) + \hat{f}^{(\text{local})}(x)$

Results


GBT-Transfer
TreeGRU-Transfer
GBT
TreeGRU
C1−C6 -> C7
C1−C6 -> C8
C1−C6 -> C9
C1−C6 -> Matmul-1024
TFLOPS
Number of Trials

Results



(a) NVIDIA TITAN X End2End (b) ARM Cortex-A53 End2End (c) ARM Mali-T860 End2End

Summary

By using an efficient cost model and transfer learning, AutoTVM quickly and automatically finds efficient schedules

…with some caveats

1. Need hand-crafted rules to express schedule templates
2. Cost model relies on hand-crafted features
3. Learning does not generalize across different domains

Discussion questions

- AutoTVM uses transfer learning to speed up the optimization across workloads. Could we use transfer learning to speed up the optimization e.g. across different CPUs? What about from CPUs to GPUs?
- The GBT cost model in AutoTVM relies on a set of hand-crafted features. What are the drawbacks of using such a feature set?
- Hyperparameter optimization algorithms often use uncertainty estimates when choosing candidates. Why is this less important for AutoTVM?

Ansor: Generating High-Performance Tensor Programs for Deep Learning

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

Background

- Low-latency execution of Deep Neural Networks (DNN)

- Existing deep learning frameworks map the operators in DNNs to vendor-provided kernel libraries to achieve high performance
  - Growing diversity of hardware platforms
  - Hard to manually tune these libraries

- High-performance tensor programs are needed

Background

- Template-guided search
  - TVM
  - FlexTensor

❑ Template design is a complicated process
❑ **Manual** templates only cover **limited program structures**

❑ Fails to include optimizations involving multiple operators

- Sequential construction based search
  - Halide

❑ The cost model trained on complete programs **cannot accurately predict** the final performance of **incomplete** programs
❑ The fixed order of sequential decisions **limits the search space**
❑ Sequential construction based search is not scalable

Challenges - Ansor

Main deficiencies of existing solutions:

- Predefined manually-written templates
- Aggressive pruning and evaluating incomplete programs
- Limited rules are used to construct the search space

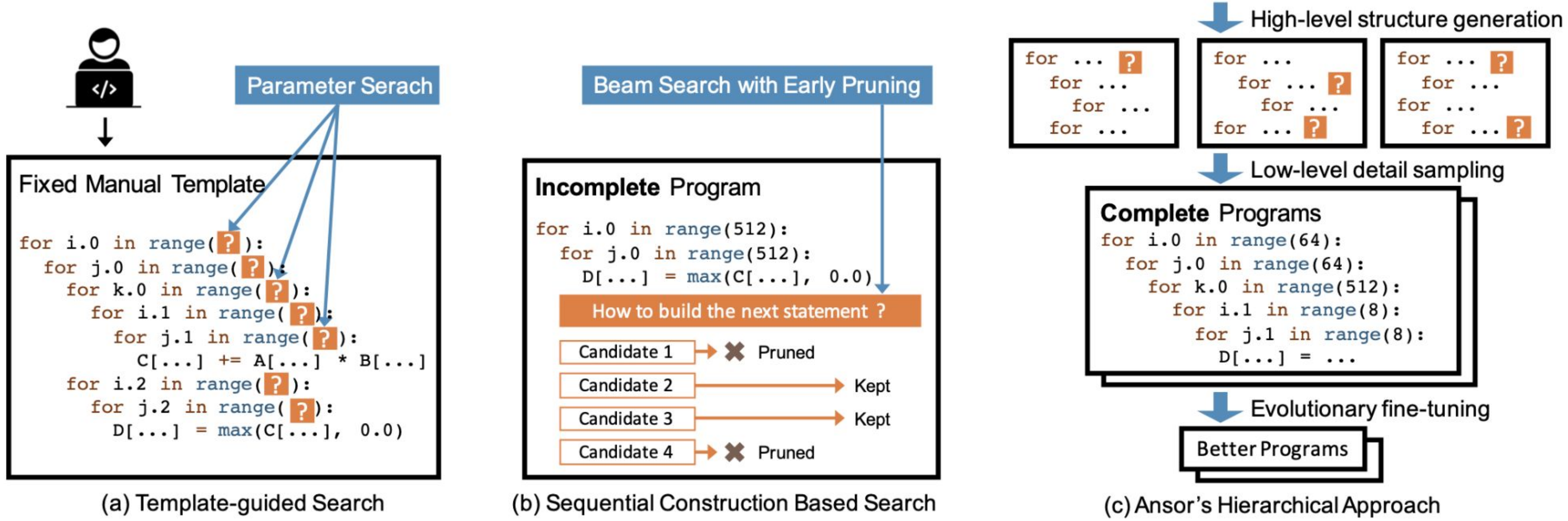
Search space not large enough!

Challenges:

★ Automatically construct a search space that is large enough

★ Search efficiently without comparing incomplete programs

★ Prioritize subgraphs critical to the end-to-end performance when optimizing an entire DNN with many subgraphs


Parameter Serach
Fixed Manual Template
(a) Template-guided Search
Beam Search with Early Pruning
Incomplete Program
How to build the next statement ?
Candidate 1
Pruned
Candidate 2
Kept
Candidate 3
Kept
Candidate 4
Pruned
(b) Sequential Construction Based Search
High-level structure generation
Low-level detail sampling
Complete Programs
Evolutionary fine-tuning
Better Programs
(c) Ansor's Hierarchical Approach

```
for i.0 in range(?):
  for j.0 in range(?):
    for k.0 in range(?):
      for i.1 in range(?):
        for j.1 in range(?):
          C[...] += A[...] * B[...]
    for i.2 in range(?):
      for j.2 in range(?):
        D[...] = max(C[...], 0.0)
```

```
for i.0 in range(512):
  for j.0 in range(512):
    D[...] = max(C[...], 0.0)
```

```
for ... ?
  for ...
    for ...
  for ...
```

```
for ...
  for ... ?
    for ...
for ... ?
```

```
for ... ?
  for ...
for ...
  for ... ?
```

```
for i.0 in range(64):
  for j.0 in range(64):
    for k.0 in range(512):
      for i.1 in range(8):
        for j.1 in range(8):
          D[...] = ...
```

Figure 2: Search strategy comparison

Ansor

- Input: set of DNNs
  - Partitioned into small subgraphs
- Three components:
  - Program Sampler
  - Performance Tuner
  - Task scheduler

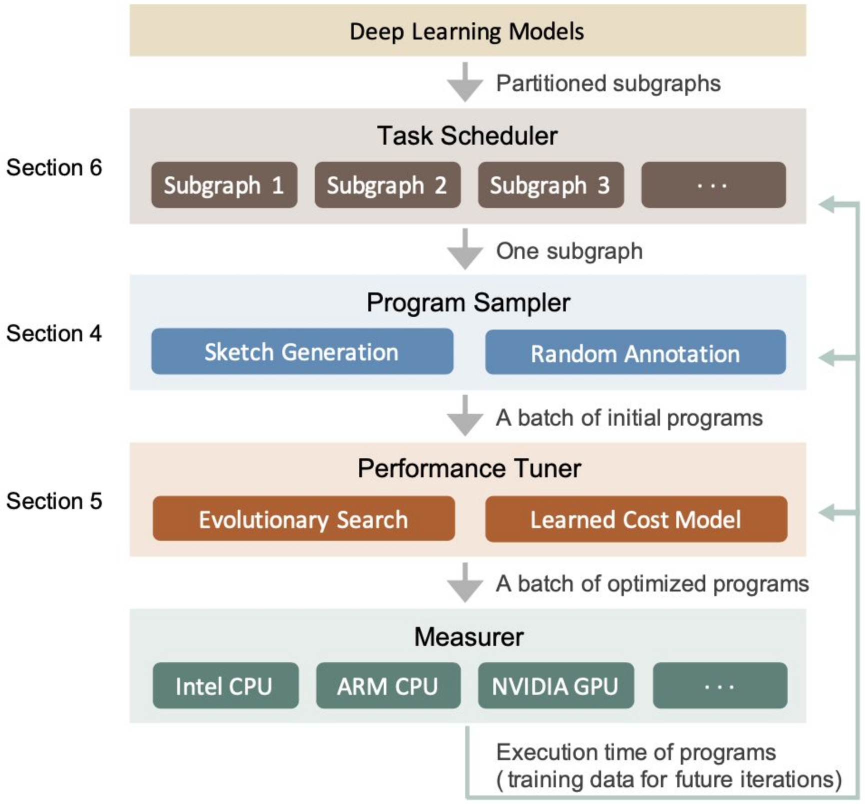


Figure 4: System Overview

Program Sampling

Program Sampler
Sketch Generation
Random Annotation

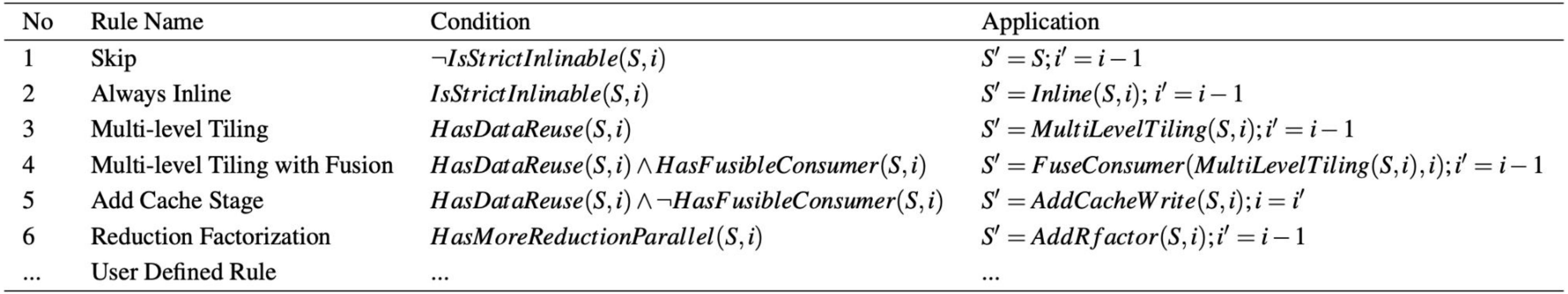
- Limited search space
  - Automatically expand the search space by recursively applying a set of flexible derivation rules
- Evaluating incomplete programs
  - Randomly sample complete programs in the search space

Hierarchical representation of the search space

- Top level: sketch
  - Generate sketches
- Lower level: annotation
  - Randomly annotate the sketches

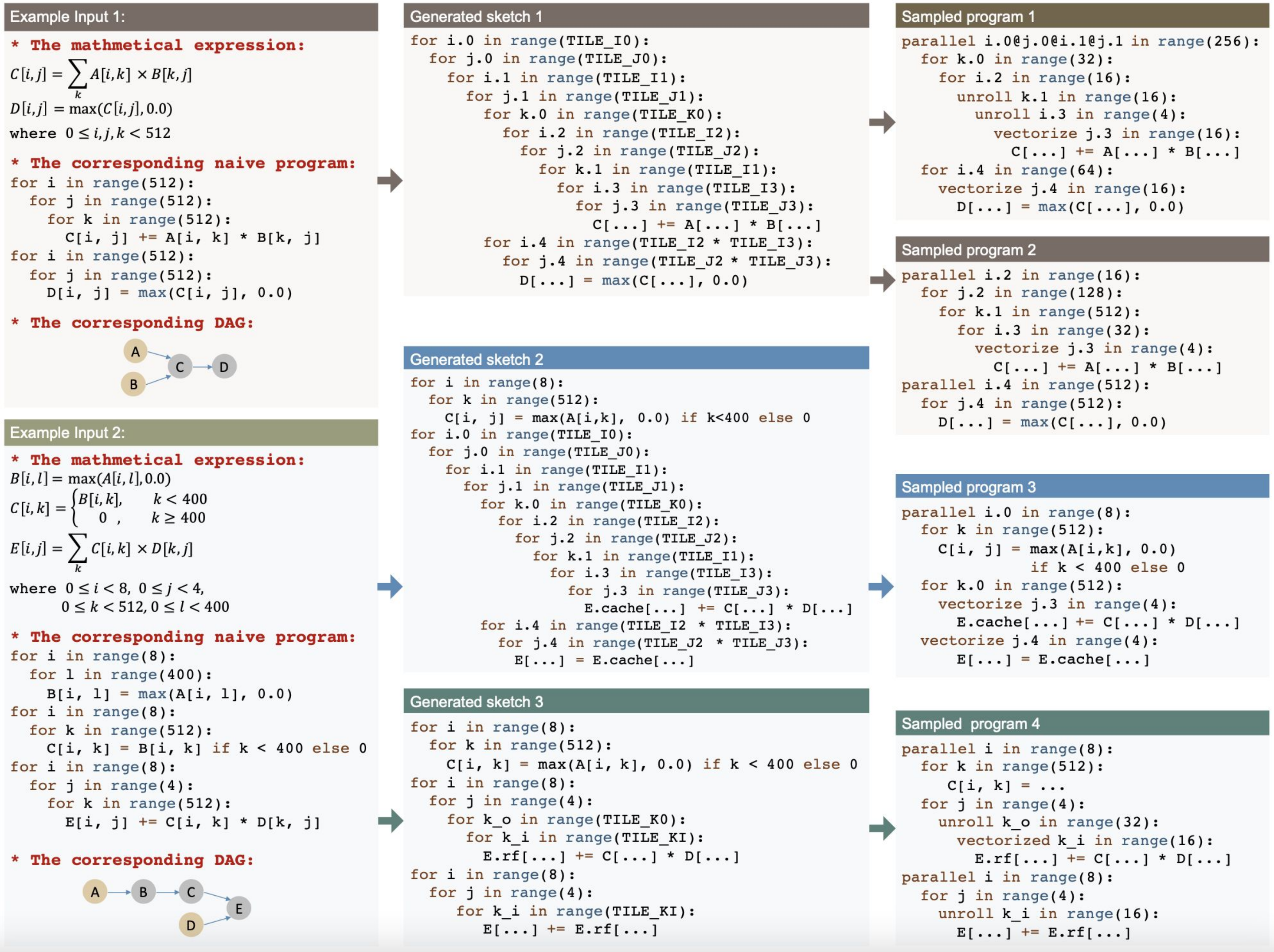
Program Sampling

Program Sampler
Sketch Generation
Random Annotation



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

Table 1: Derivation rules used to generate sketches



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 naive 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:**

$$\begin{tikzcd} A \arrow[dr] & & \\ & C \arrow[r] & D \\ B \arrow[ur] & & \end{tikzcd}$$

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)
```

Example Input 2:

*** The mathmetical expression:**

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

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

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

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

*** The corresponding naive 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:**

$$\begin{tikzcd} A \arrow[r] & B \arrow[r] & C \arrow[dr] & \\ & & & E \\ & & D \arrow[ur] & \end{tikzcd}$$

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[...]
```

Sampled program 3

```
parallel i.0 in range(8):
  for k in range(512):
    C[i, j] = max(A[i,k], 0.0)
                if k < 400 else 0
  for k.0 in range(512):
    vectorize j.3 in range(4):
      E.cache[...] += C[...] * D[...]
  vectorize j.4 in range(4):
      E[...] = E.cache[...]
```

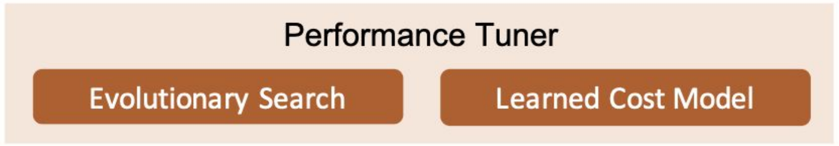
Generated sketch 3

```
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[...]
```

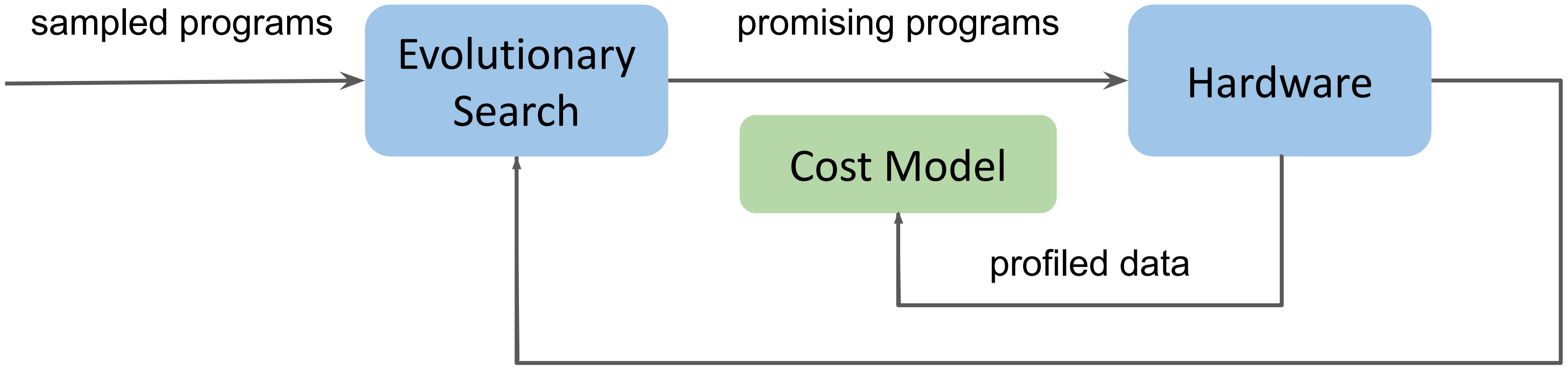
Sampled program 4

```
parallel i in range(8):
  for k in range(512):
    C[i, k] = ...
  for j in range(4):
    unroll k_o in range(32):
      vectorized k_i in range(16):
        E.rf[...] += C[...] * D[...]
parallel i in range(8):
  for j in range(4):
    unroll k_i in range(16):
      E[...] += E.rf[...]
```

Performance Fine-Tuning



- Evolutionary search
- Learned cost model



Performance Fine-Tuning - Evolutionary Search

- Select some programs from the current generation according to certain probabilities
- Apply one of the evolution operations to generate a new program
    - Tile size mutation
    - Parallel mutation
    - Pragma mutation
    - Computation location mutation
    - Node-based crossover

Apply to general tensor programs and handle a search space with complicated dependency

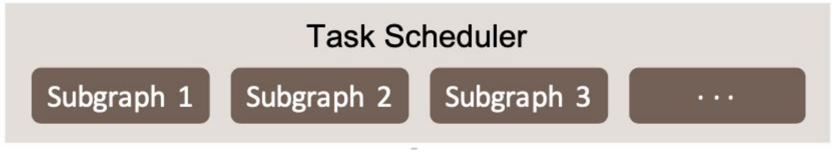
Perform out-of-order modifications to programs, addressing the sequential limitations

Performance Fine-Tuning - Learned Cost Model

$$loss(f, P, y) = w_p(\sum_{s \in S(P)} f(s) - y)^2 = y(\sum_{s \in S(P)} f(s) - y)^2$$

- Has great portability since a single model design can be reused for different hardware backends
- Gives relatively accurate estimations of the fitness of programs
- Querying the model is actually orders of magnitudes faster than the actual measurement

Task Scheduler



- Select the subgraphs that are more important to the overall performance
- A task: the process performed to generate high-performance programs for a subgraph
- One unit of time resources: one iteration of selecting a task, generating a batch of promising programs for the subgraph, and measuring the program on hardware

Task Scheduler

t = (t1, t2, …, tn), initialized to t = (1, 1, …, 1)

Objective: minimize the end-to-end cost, i.e. $\text{minimize } f(g_1(t), g_2(t), \ldots, g_3(t))$

$$f_1 = \textstyle\sum_{j=1}^{m} \sum_{i \in S(j)} w_i \times g_i(t)$$

$$f_2 = \textstyle\sum_{j=1}^{m} \max\left(\sum_{i \in S(j)} w_i \times g_i(t), L_j\right)$$

$$f_3 = -\left(\textstyle\prod_{j=1}^{m} \frac{B_j}{\sum_{i \in S(j)} w_i \times g_i(t)}\right)^{\frac{1}{m}}$$

$$f_4 = \textstyle\sum_{j=1}^{m} \sum_{i \in S(j)} w_i \times \max(g_i(t), ES(g_i, t))$$

Table 2: Examples of objective functions for multiple neural networks

Task Scheduler

Approximate the gradient of f to choose the task $i = \mathrm{argmax}_i \left|\frac{\partial f}{\partial t_i}\right|$

$$\frac{\partial f}{\partial t_i} \approx \frac{\partial f}{\partial g_i}(\alpha \frac{g_i(t_i) - g_i(t_i - \Delta t)}{\Delta t} + (1-\alpha)(\min(-\frac{g_i(t_i)}{t_i}, \beta \frac{C_i}{\max_{k\in N(i)} V_k} - g_i(t_i))))$$

Evaluation - Single Operator Benchmark

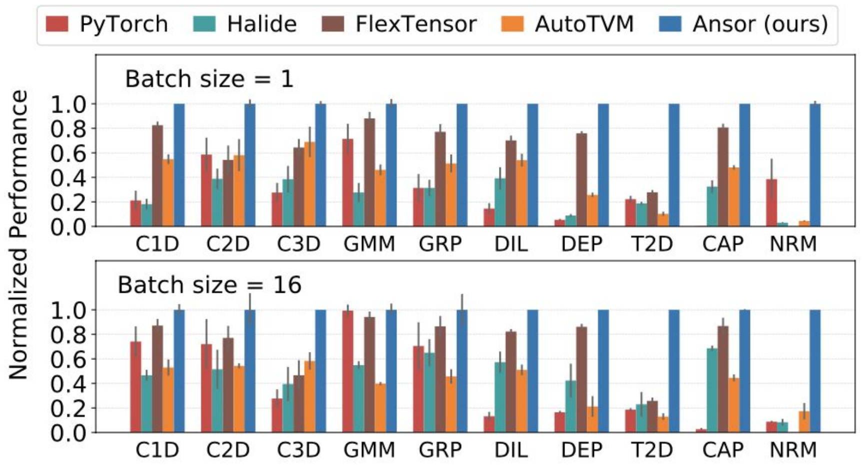


Figure 6: Single operator performance benchmark on a 20- core Intel-Platinum-8269CY

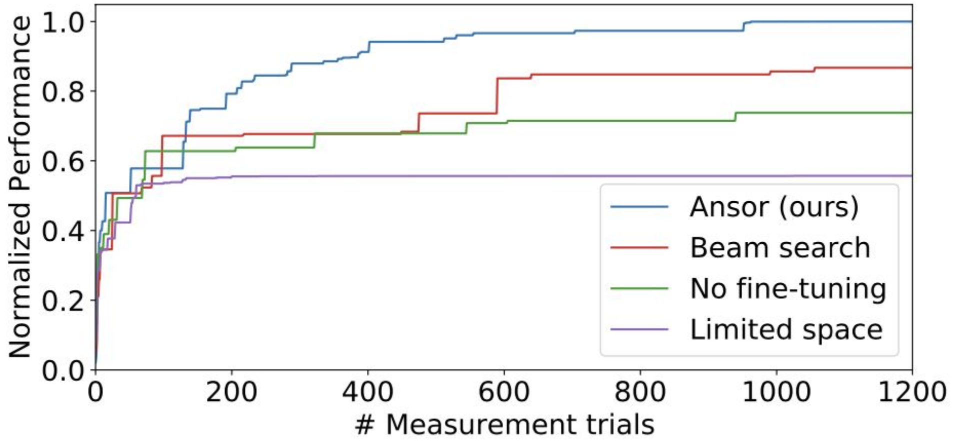


Figure 7: Ablation study of four variants of Ansor on a convolution operator

Evaluation - Subgraph Benchmark

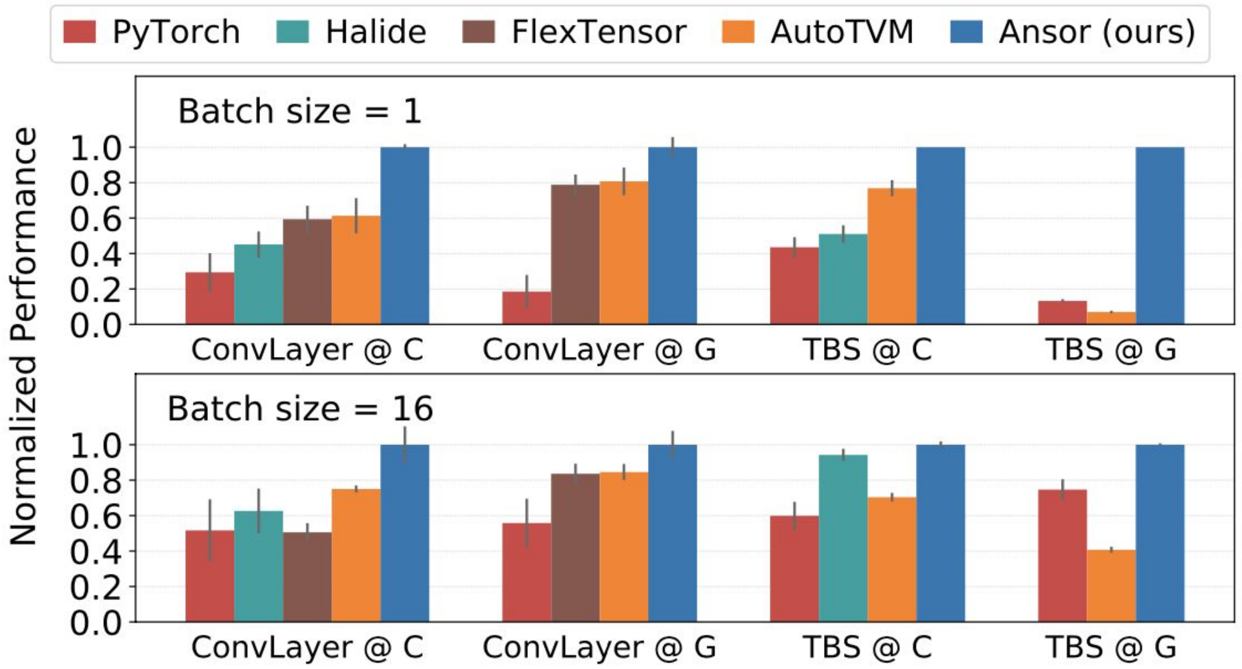


Figure 8: Subgraph performance benchmark on a 20-core Intel-Platinum-8269CY and an NVIDIA V100

Evaluation - End-to-End Network Benchmark

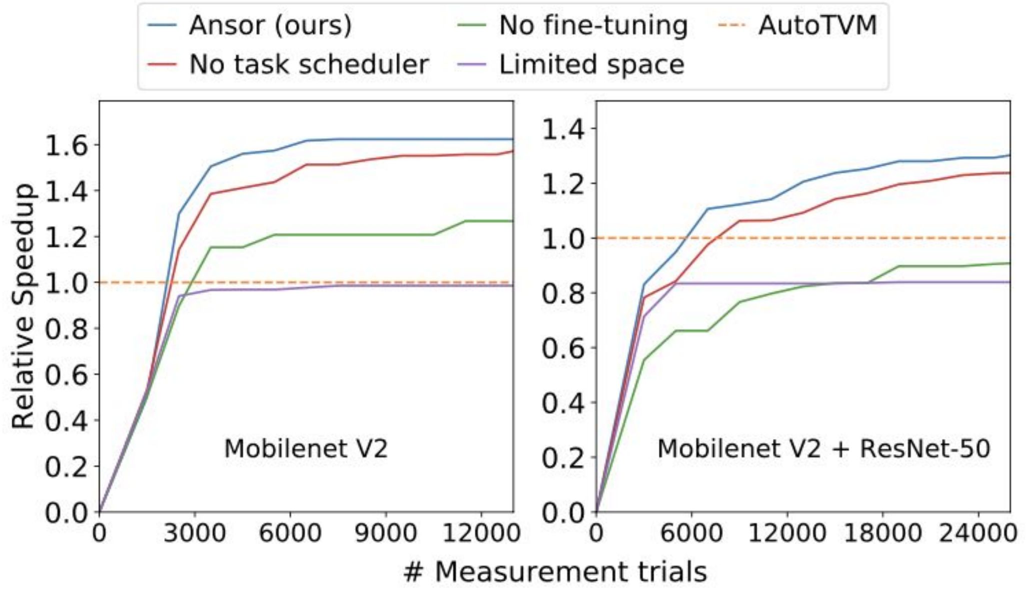


Figure 10: Network performance auto-tuning curve

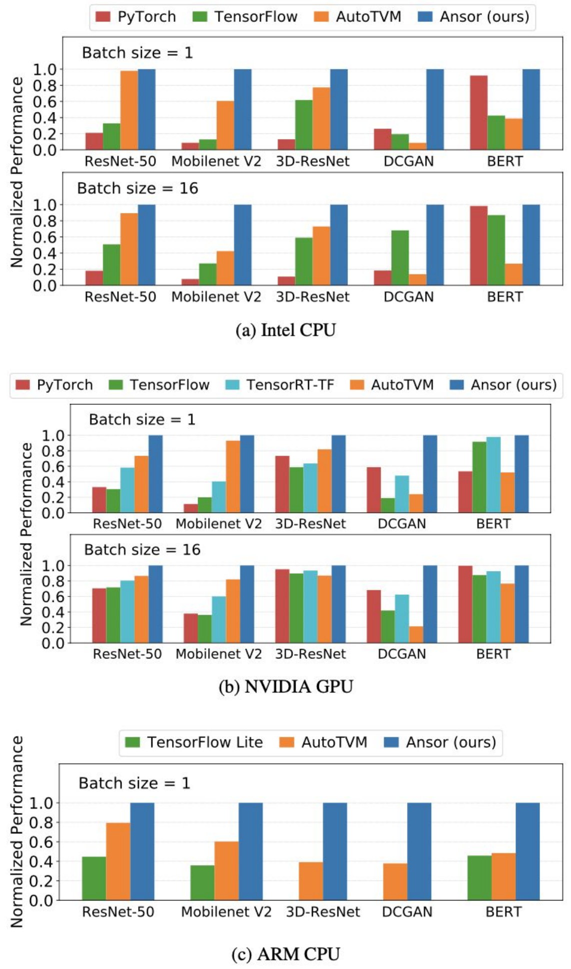
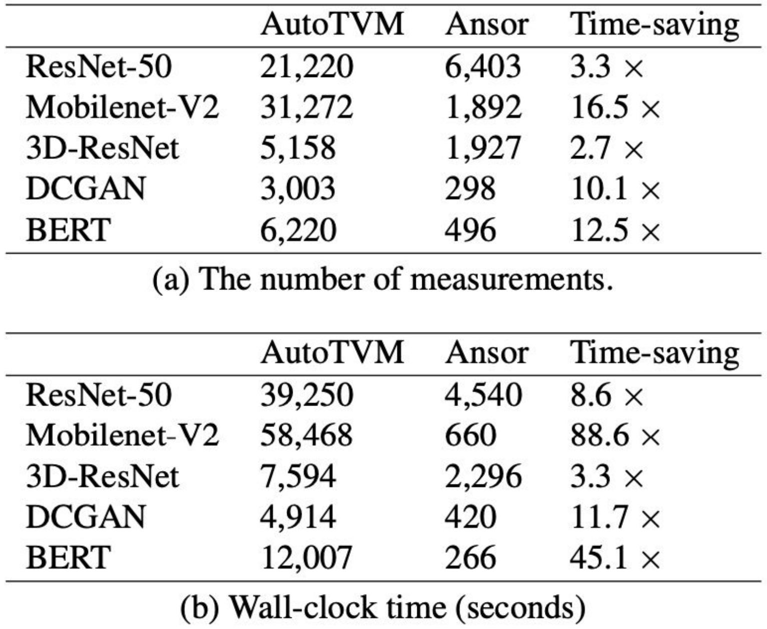


Figure 9: Network inference performance benchmark on three hardware platforms

Evaluation - Search Time



	AutoTVM	Ansor	Time-saving
ResNet-50	21,220	6,403	3.3 ×
Mobilenet-V2	31,272	1,892	16.5 ×
3D-ResNet	5,158	1,927	2.7 ×
DCGAN	3,003	298	10.1 ×
BERT	6,220	496	12.5 ×

(a) The number of measurements.

	AutoTVM	Ansor	Time-saving
ResNet-50	39,250	4,540	8.6 ×
Mobilenet-V2	58,468	660	88.6 ×
3D-ResNet	7,594	2,296	3.3 ×
DCGAN	4,914	420	11.7 ×
BERT	12,007	266	45.1 ×

(b) Wall-clock time (seconds)

Table 3: The number of measurements and wall-clock time used for Ansor to match the performance of AutoTVM on the Intel CPU (batch size=1)

Evaluation - Cost Model Evaluation

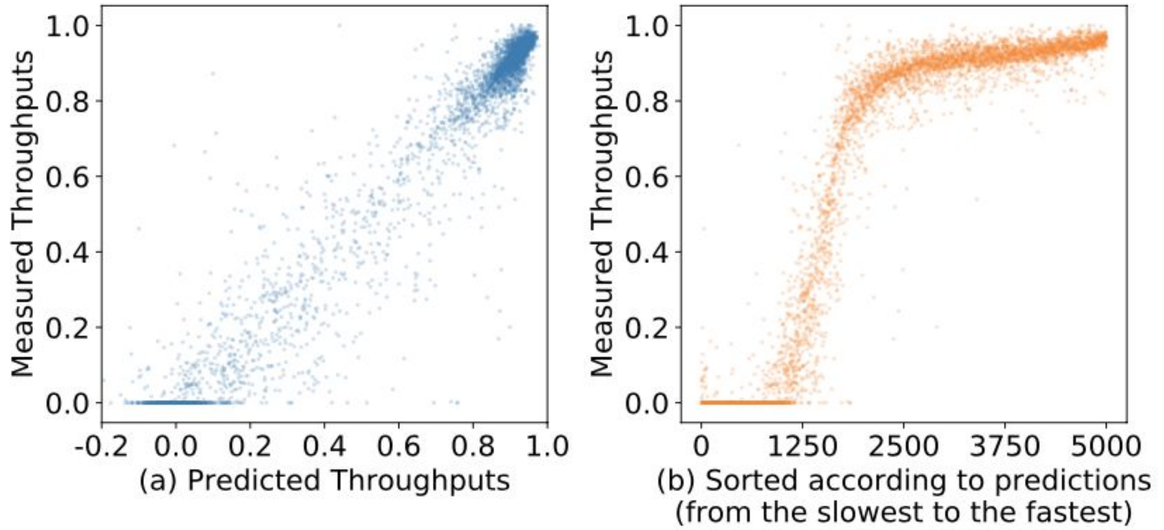


Figure 11: Measured throughputs vs. predicted throughputs.

Summary

Ansor: an automated search framework that generates high-performance tensor programs for deep neural networks

Compared with previous existing search strategies:

- Explores more optimization combinations by sampling programs from a hierarchical representation of the search space.
- Fine-tunes the sampled programs with evolutionary search and uses a learned cost model to identify the best programs.
- Utilizes a task scheduler to simultaneously optimize multiple subgraphs in deep neural networks.

Discussion

1. Besides XGBoost and the TreeGRU used in AutoTVM, is there any other model that might further improve the performance?
2. As stated in the paper, Ansor currently only supports dense operators. To support sparse operators that are commonly used in sparse neural networks and graph neural networks, how can we redesign the search space?