Machine Learning Compiler: An Overview

Bojian Zheng

2022/4/1 @Amazon Reading Group

Agenda for Today

- Why Machine Learning Compilers?
  - State-of-the-Art Machine Learning Frameworks Design, and Flaws:
  1. Vendor libraries not delivering the optimal performance.

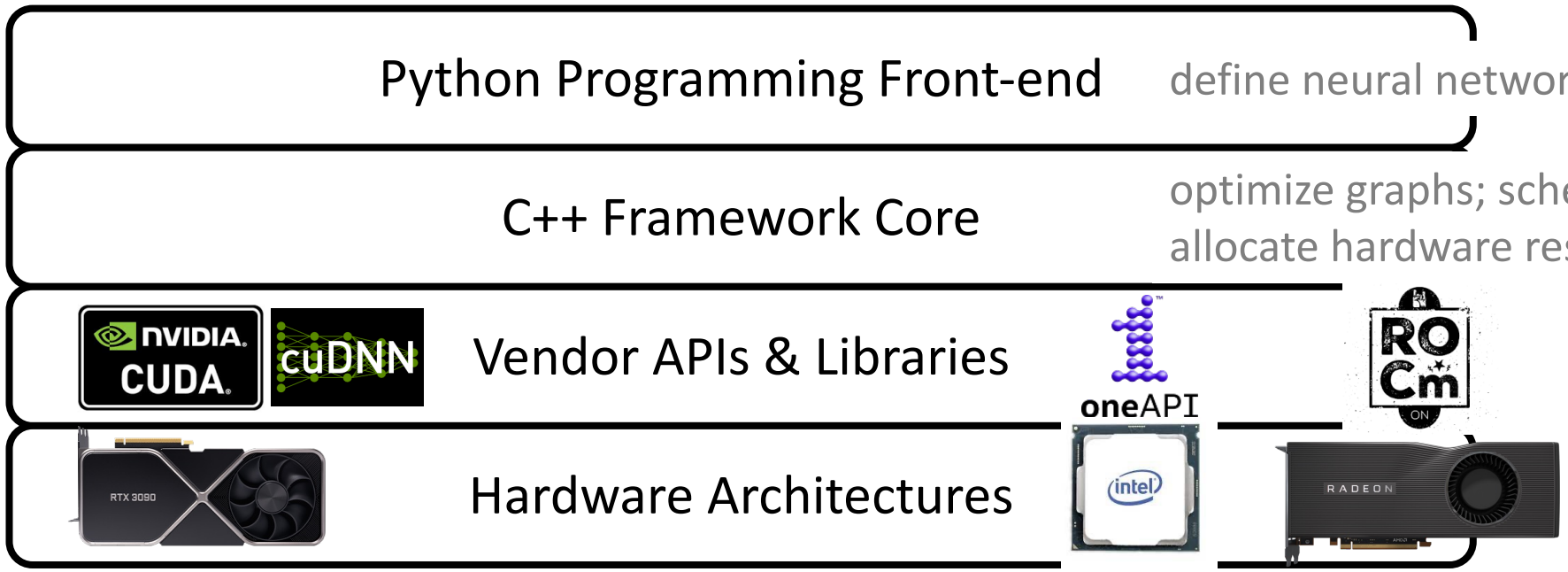
- 2 Classes of Machine Learning Compilers:
  - **Halide[1]/TVM[2]**: Easier to write high-performance programs.

[1] J. Ragan-Kelley et al. *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines*. PLDI 2013
[2] T. Chen et al. *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*. OSDI 2018

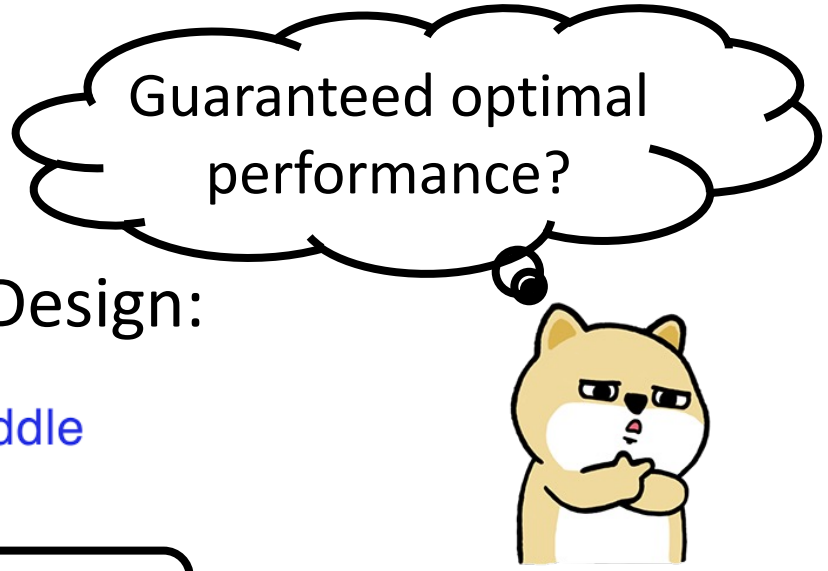
Why Machine Learning Compilers?

- State-of-the-Art Machine Learning Frameworks Design:

TensorFlow PYTORCH mxnet PaddlePaddle

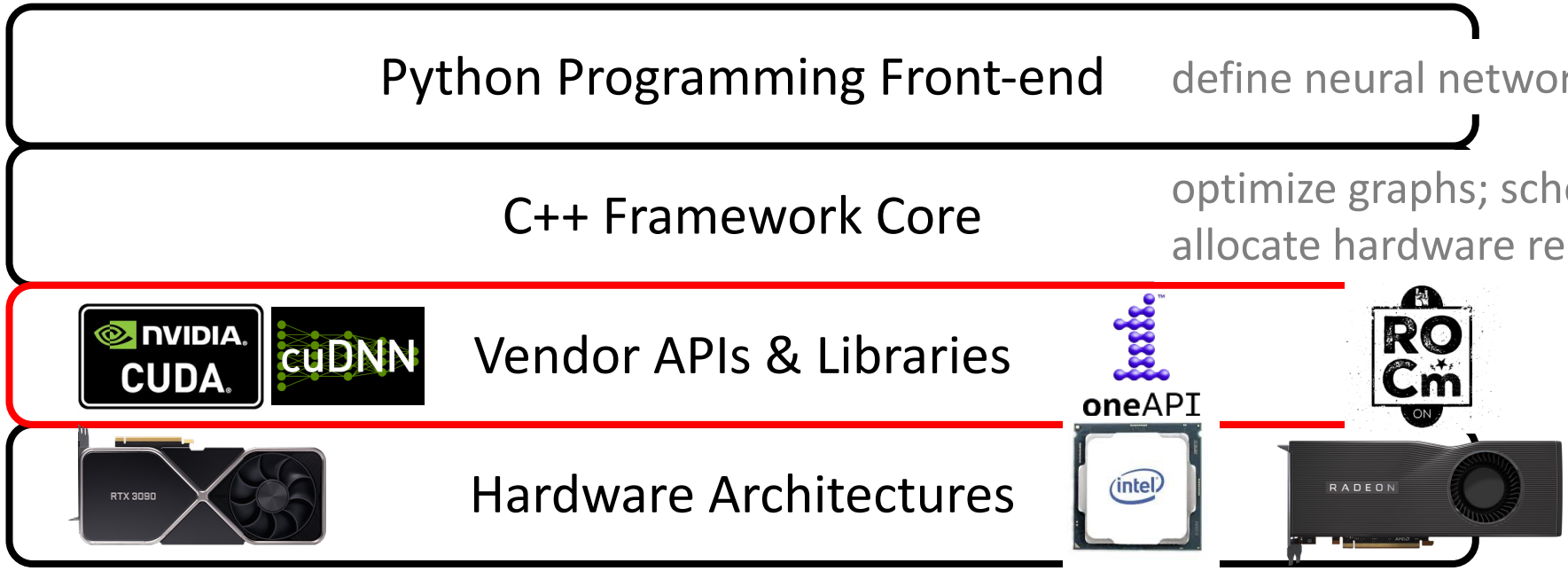


Machine Learning Frameworks



- State-of-the-Art Machine Learning Frameworks Design:

TensorFlow PYTORCH mxnet PaddlePaddle



Vendor Libraries

- Example Workload:

$$Y = XW^\top \quad X : [M, K], W : [N, K]$$

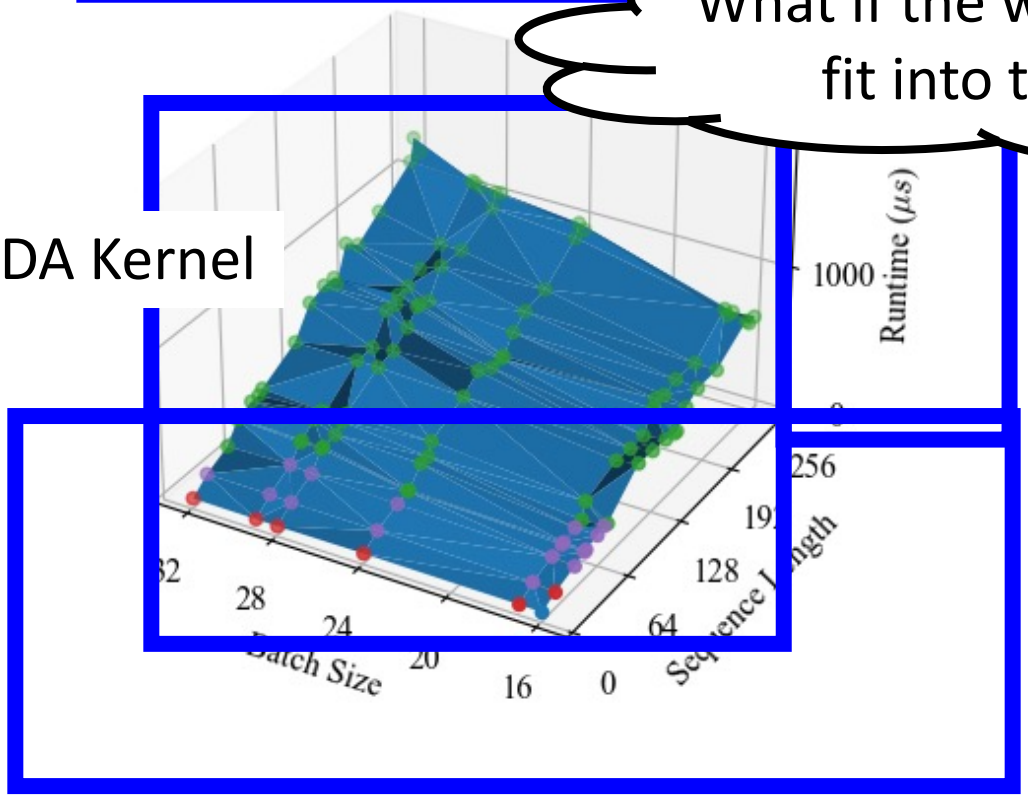
- Q: How to efficiently handle for all cases of (M, K, N)?
- Solutions (e.g., cuBLAS):
  - Provide efficient kernels that cover to all the use cases. E.g.,

$$y = xw^\top \quad \begin{cases} x : [32, *], w : [32, *] \\ x : [64, *], w : [64, *] \\ x : [128, *], w : [128, *] \end{cases}$$

  - Dispatch at runtime to the most suitable kernel.

Vendor Libraries

$Y = XW^T, X : [B \times T, 768], W : [768, 768]$



What if the workloads **DO NOT** fit into those kernels?

1 Color = 1 Unique CUDA Kernel

Even as the shapes vary, cuBLAS only invokes a **handful** of kernels.

Vendor Libraries

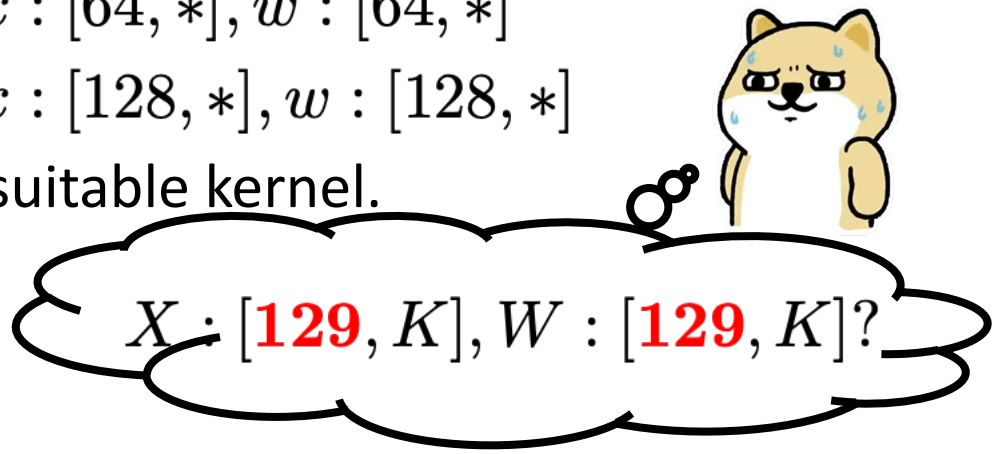
- Example Workload:

$$Y = XW^\top \quad X : [M, K], W : [N, K]$$

- Q: How to efficiently handle for all cases of (M, K, N)?
- Solutions (e.g., cuBLAS):
  - Provide efficient kernels that cover to all the use cases. E.g.,

$$y = xw^\top \quad \begin{cases} x : [32, *], w : [32, *] \\ x : [64, *], w : [64, *] \\ x : [128, *], w : [128, *] \end{cases}$$

  - Dispatch at runtime to the most suitable kernel.



Vendor Libraries

- Leads to **sub-optimal** performance (up to $13\times$) due to
  - Low Hardware Utilization[1] and/or
  - Redundant Computations (by padding)[2]
- Develop high-performance customized kernels?
  - Requires huge expertise + engineering efforts: thousands of lines per operator per architecture[3].

[1] F. Yu et al. *Towards Latency-aware DNN Optimization with GPU Runtime Analysis and Tail Effect Elimination*. arXiv 2020

[2] Alibaba. *Bringing TVM into TensorFlow for Optimizing Neural Machine Translation on GPU*. 2018

[3] NVIDIA. *CUTLASS: CUDA Templates for Linear Algebra Subroutines*. https://github.com/NVIDIA/cutlass

Tensor Program Compilers

- State-of-the-Arts: Halide[1] & TVM[2]
  - Halide: image processing
  - TVM: machine learning & better GPU support
- Objective: Easier to write high-performance programs.
- Key Idea: Abstracts low-level implementations using **schedules**.

45 lines of Python Scheduling Code

```
s = create_schedule(Y.op)
Y_local = s.cache_write([Y], "local")
i, j, k = tuple(Y_local.op.axis)
i_o_i, i_i = s[Y_local].split(i, factor=8)
i_o_o_i, i_o_i = s[Y_local].split(i_o_i, factor=2)
i_o_o_o, i_o_o_i = s[Y_local].split(i_o_o_i, factor=1)
...
```

[1] J. Ragan-Kelley et al. *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines*. PLDI 2013
[2] T. Chen et al. *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*. OSDI 2018

Tensor Program Compilers

- State-of-the-Arts: Halide[1] & TVM[2]
  - Halide: image processing
  - TVM: machine learning & better GPU support
- Objective: Easier to write high-performance programs.
- Key Idea: Abstracts low-level implementations using **schedules**.

45 lines of Python Scheduling Code

```
s = create_schedule(Y.op)
Y_local = s.cache_write([Y], "local")
i, j, k = tuple(Y_local.op.axis)
i_o_i, i_i = s[Y_local].split(i, factor=8)
i_o_o_i, i_o_i = s[Y_local].split(i_o_i, factor=2)
i_o_o_o, i_o_o_i = s[Y_local].split(i_o_o_i, factor=1)
...
```

scheduling primitives

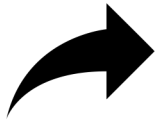
[1] J. Ragan-Kelley et al. *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines*. PLDI 2013
[2] T. Chen et al. *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*. OSDI 2018

Tensor Program Compilers

- State-of-the-Arts: Halide[1] & TVM[2]
  - Halide: image processing
  - TVM: machine learning & better GPU support
- Objective: Easier to write high-performance programs.
- Key Idea: Abstracts low-level implementations using **schedules**.

45 lines of Python Scheduling Code

```
s = create_schedule(Y.op)
Y_local = s.cache_write([Y], "local")
i, j, k = tuple(Y_local.op.axis)
i_o_i, i_i = s[Y_local].split(i, factor=8)
i_o_o_i, i_o_i = s[Y_local].split(i_o_i, factor=2)
i_o_o_o, i_o_o_i = s[Y_local].split(i_o_o_i, factor=1)
...
```



946 lines of CUDA Code

```
__global__ default_function(...) {
  ...
  float Y_local[128];
  __shared__ float X_shared[768];
  __shared__ float W_shared[384];
  ...
}
```

[1] J. Ragan-Kelley et al. *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines*. PLDI 2013
[2] T. Chen et al. *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*. OSDI 2018

Tensor Program Compilers

- State-of-the-Arts: Halide[1] & TVM[2]
  - Halide: image processing
  - TVM: machine learning & better GPU support
- Objective: Easier to write high-performance programs.
- Key Idea: Abstracts low-level implementations using **schedules**.

45 lines of Python Scheduling Code

```
s = create_schedule(Y.op)
Y_local = s.cache_write([Y], "local")
i, j, k = tuple(Y_local.op.axis)
i_o_i, i_i = s[Y_local].split(i, factor=8)
i_o_o_i, i_o_i = s[Y_local].split(i_o_i, factor=2)
i_o_o_o, i_o_o_i = s[Y_local].split(i_o_o_i, factor=1)
...
```

946 lines of CUDA Code

```
__global__ default_function(...) {
  ...
  float Y_local[128];
  __shared__ float X_shared[768];
  __shared__ float W_shared[384];
  ...
}
```

automatic inference

[1] J. Ragan-Kelley et al. *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines*. PLDI 2013
[2] T. Chen et al. *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*. OSDI 2018

Tensor Program Compilers

Pros

- Easily programed, modified and parameterized.

Cons

- Less flexible
  - What if my transformations are NOT supported by primitives?
- Not quite “easily” programmed.

45 lines of Python Scheduling Code

```
s = create_schedule(Y.op)
Y_local = s.cache_write([Y], "local")
i, j, k = tuple(Y_local.op.axis)
i_o_i, i_i = s[Y_local].split(i, factor=8)
i_o_o_i, i_o_i = s[Y_local].split(i_o_i, factor=2)
i_o_o_o, i_o_o_i = s[Y_local].split(i_o_o_i, factor=1)
...
```

Tensor Program Compilers

Pros

- Easily programed, modified and parameterized.

Cons

- Less flexible
  - What if my transformations are NOT supported by primitives?
- Not quite “easily” programmed.
- Optimal schedules are hardware-specific.

45 lines of Python Scheduling Code

```
s = create_schedule(Y.op)
Y_local = s.cache_write([Y], "local")
i, j, k = tuple(Y_local.op.axis)
i_o_i, i_i = s[Y_local].split(i, factor=8)
i_o_o_i, i_o_i = s[Y_local].split(i_o_i, factor=2)
i_o_o_o, i_o_o_i = s[Y_local].split(i_o_o_i, factor=1)
...
```

Why $(8, 2, 1)$?

Want to generate high-performance schedules **automatically**.

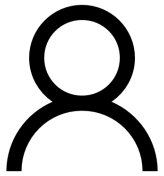
Auto-Scheduler

- State-of-the-Arts: Halide[1] and TVM[2] Auto-Schedulers
- Objective: Given a compute definition, automatically find a high-performance schedule.

[1] A. Adams et al. *Learning to Optimize Halide with Tree Search and Random Programs*. ACM Transactions on Graphics (TOG) 2019
[2] L. Zheng et al. *Ansor: Generating High-Performance Tensor Programs for Deep Learning*. OSDI 2020

Auto-Scheduler System Overview

$$Y = XW^\top \quad X : [M, K], W : [N, K]$$

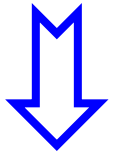


Operator Specification

Shape Description

```
for i in range(2048):
  for j in range(2304):
    for k in range(768):
      Y[i][j] += X[i][k] * W[j][k]
```

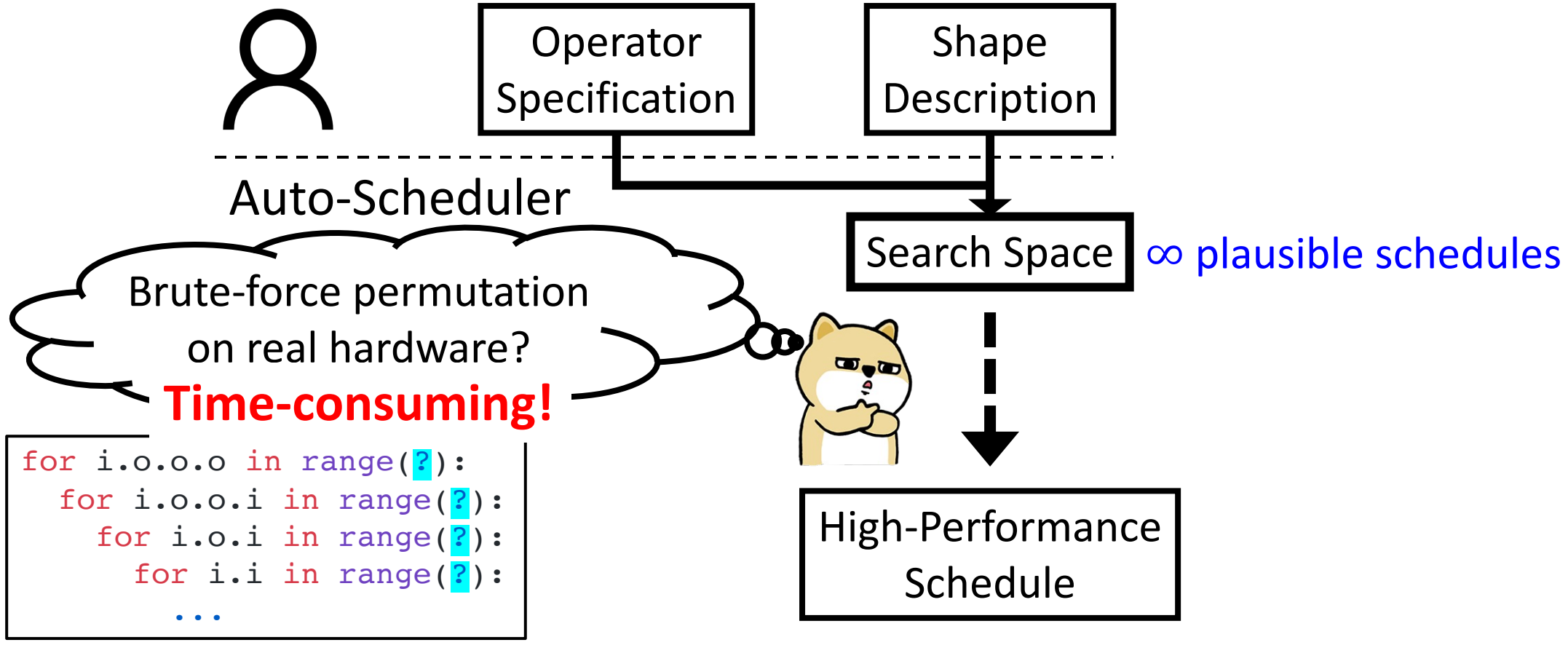
$$(M, K, N) = (2048, 768, 2304)$$



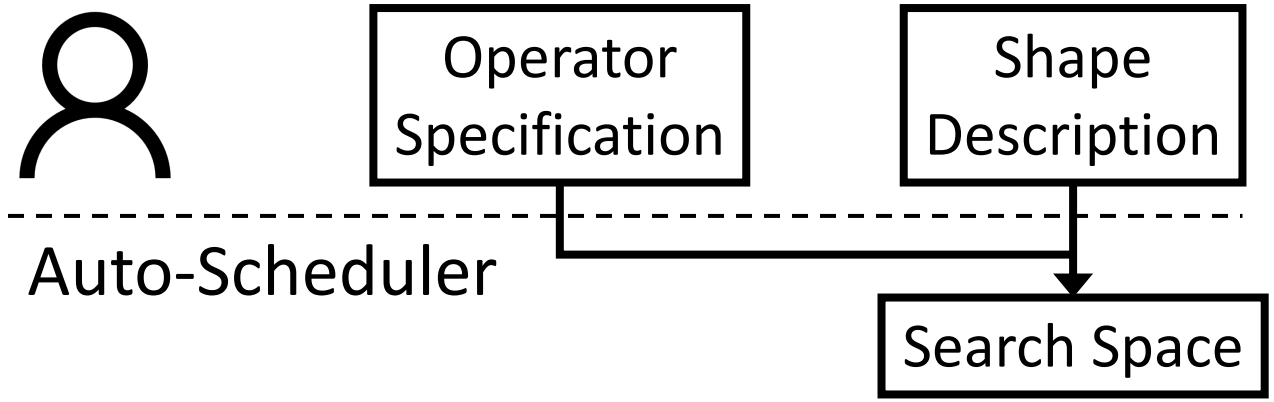
Sketch Generation Rules

```
for i.o.o.o in range(?):
  for i.o.o.i in range(?):
    for i.o.i in range(?):
      for i.i in range(?):
        ...
```

Auto-Scheduler System Overview



Auto-Scheduler System Overview

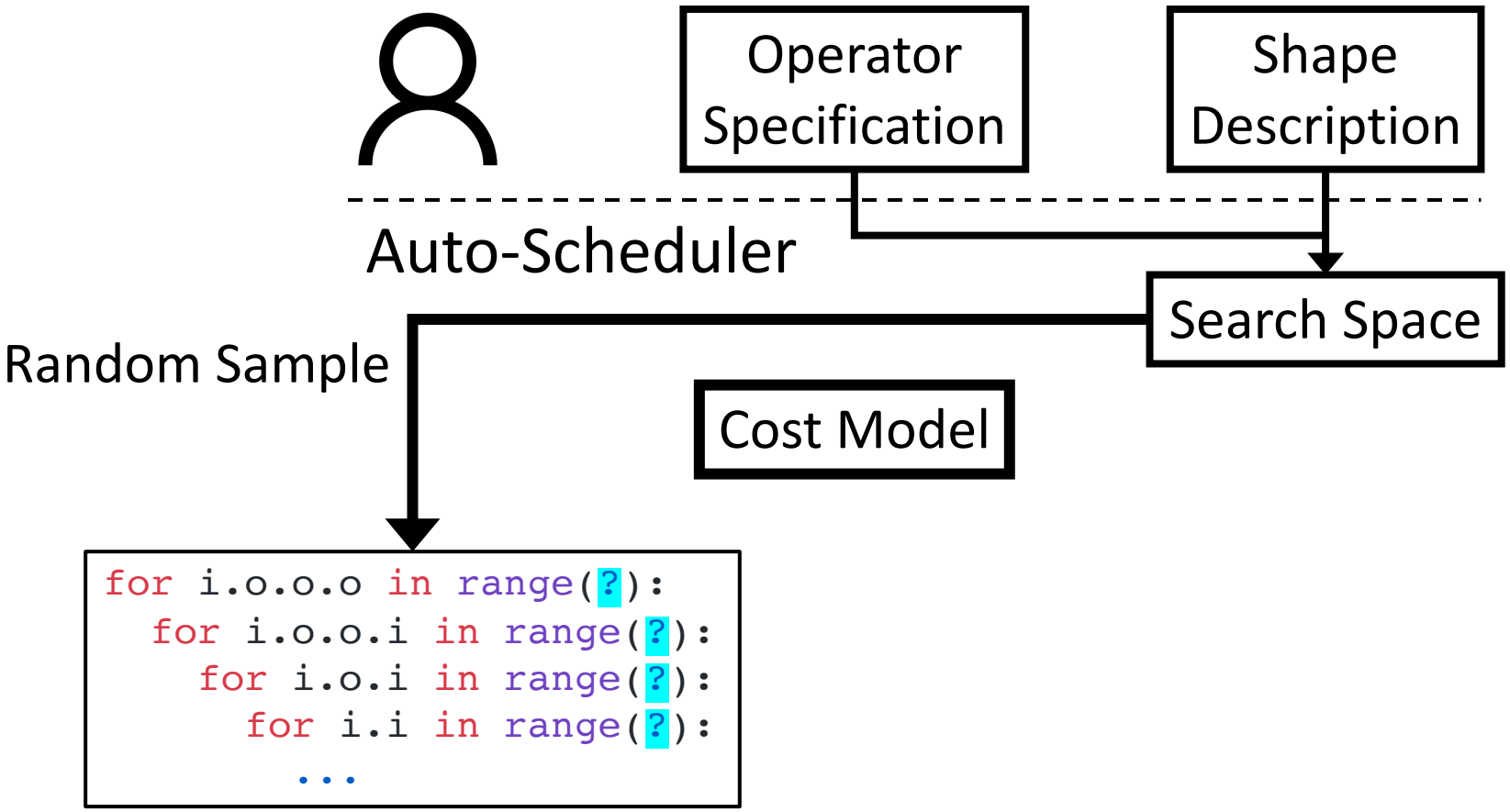


Predict performance of many schedules simultaneously

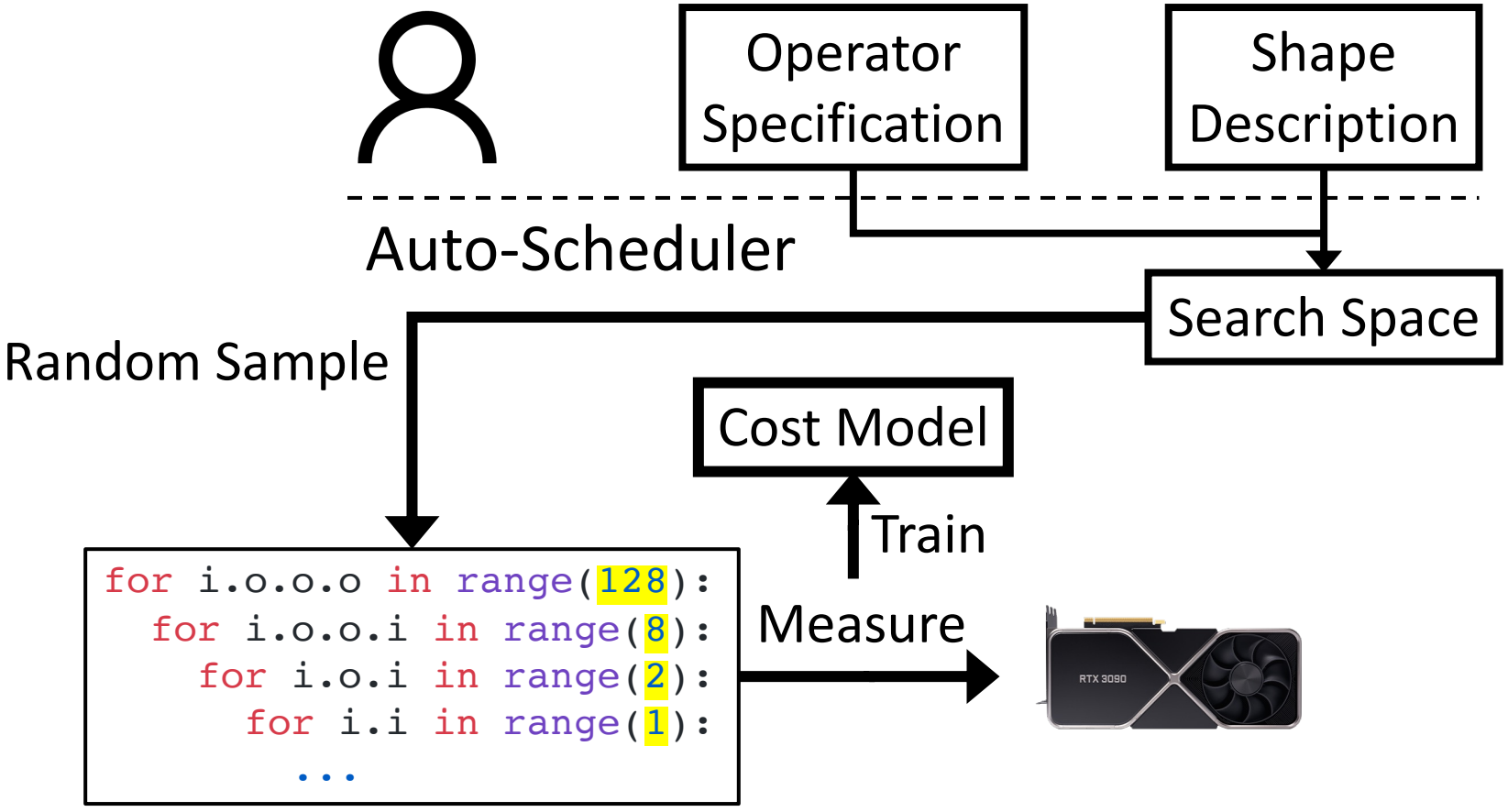
Cost Model

```
for i.o.o.o in range(?):
  for i.o.o.i in range(?):
    for i.o.i in range(?):
      for i.i in range(?):
        ...
```

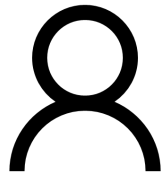
Auto-Scheduler System Overview



Auto-Scheduler System Overview



Auto-Scheduler System Overview



Operator Specification

Shape Description

Auto-Scheduler

Search Space

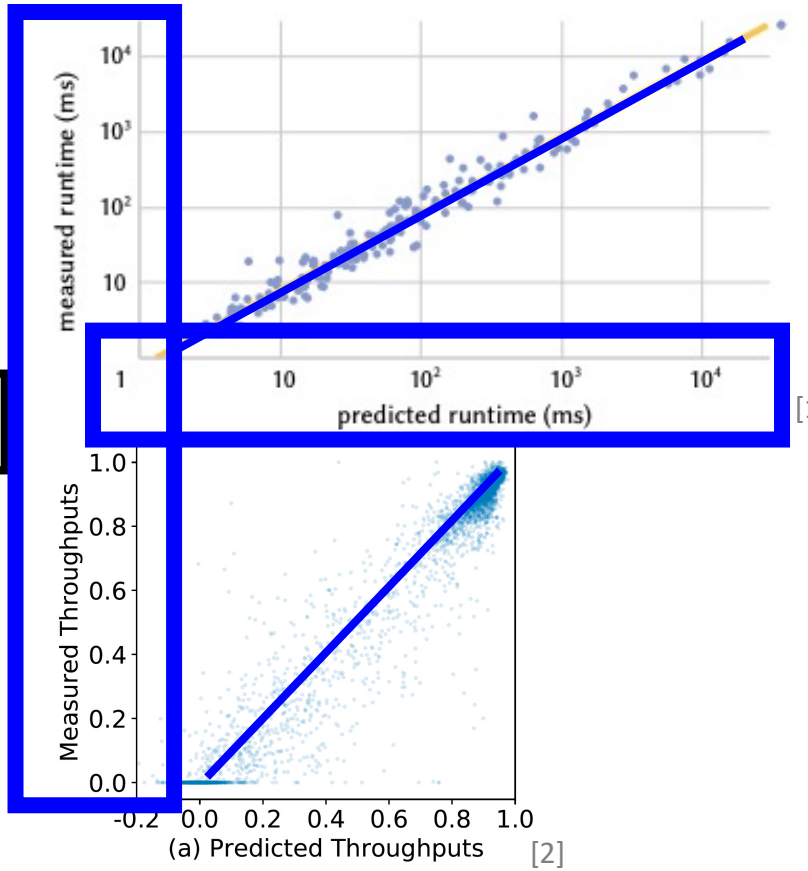
Random Sample

Learned Cost Model

Train

```
for i.o.o.o in range(128):
  for i.o.o.i in range(8):
    for i.o.i in range(2):
      for i.i in range(1):
        ...
```

Measure



[1]

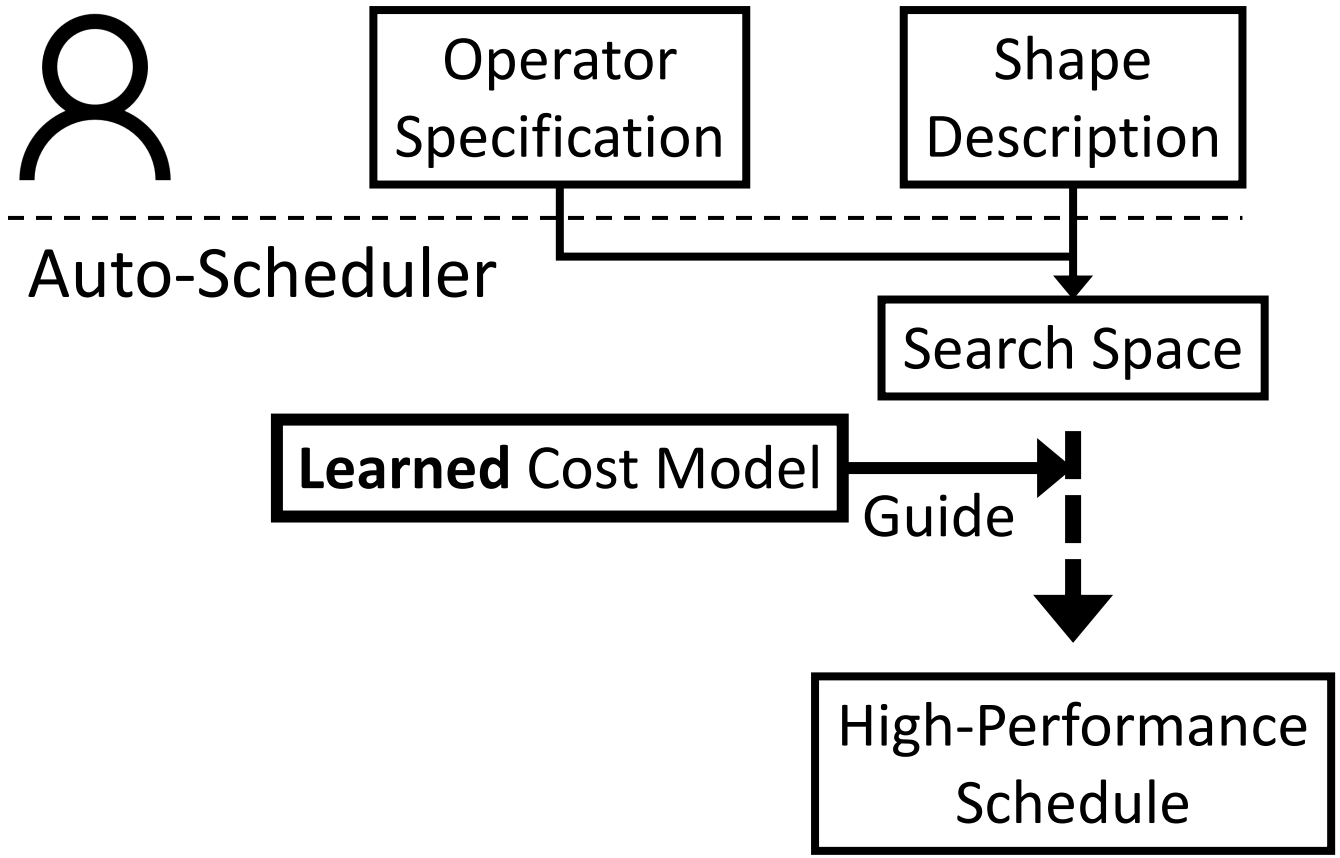
[2]

Measured and Predicted are strongly correlated.

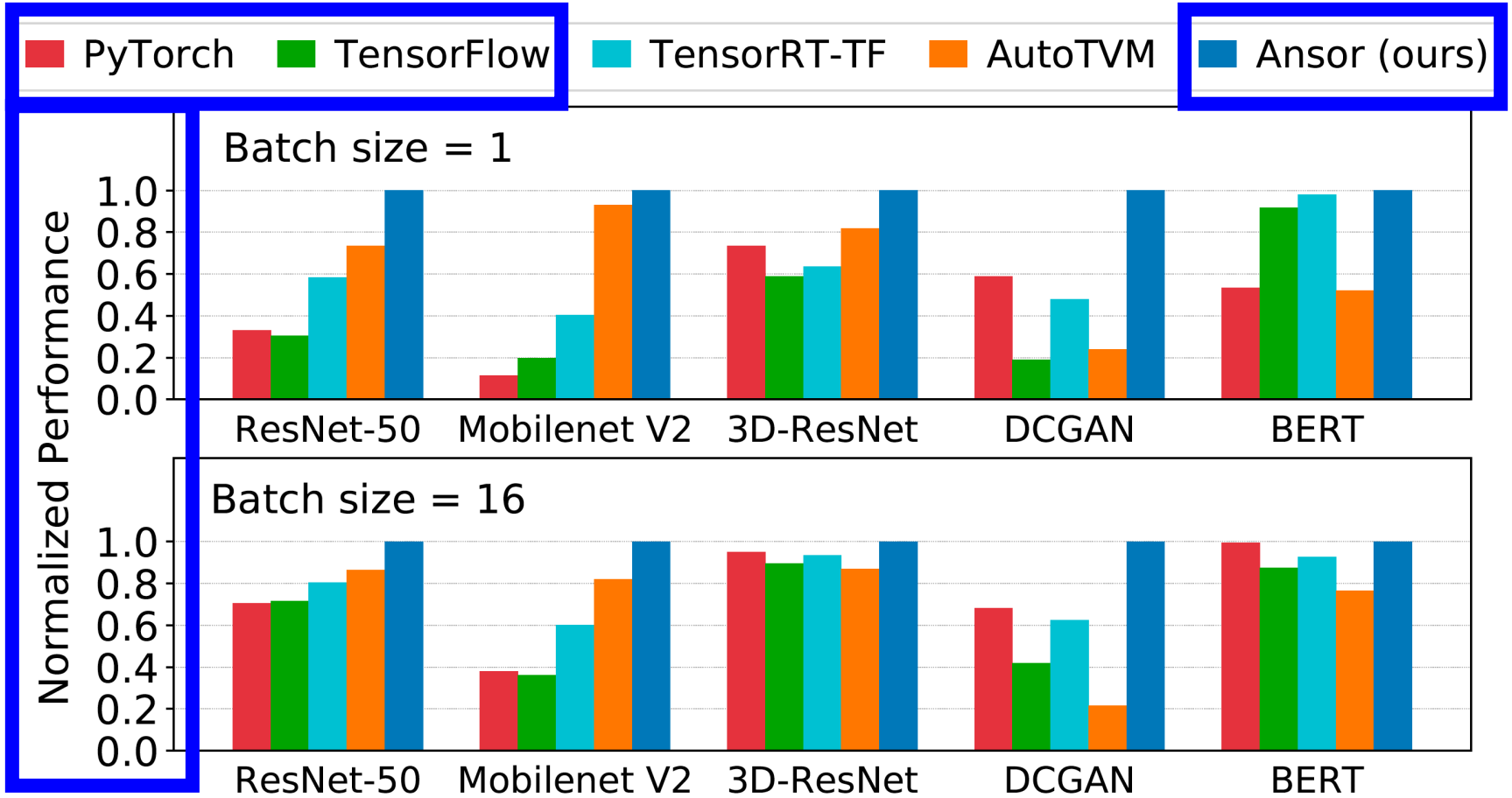
[1] A. Adams et al. *Learning to Optimize Halide with Tree Search and Random Programs*. ACM Transactions on Graphics (TOG) 2019

[2] L. Zheng et al. *Ansor: Generating High-Performance Tensor Programs for Deep Learning*. OSDI 2020

Auto-Scheduler System Overview



Auto-Scheduler Evaluation



Up to 1.7× better than the 2nd best alternative.

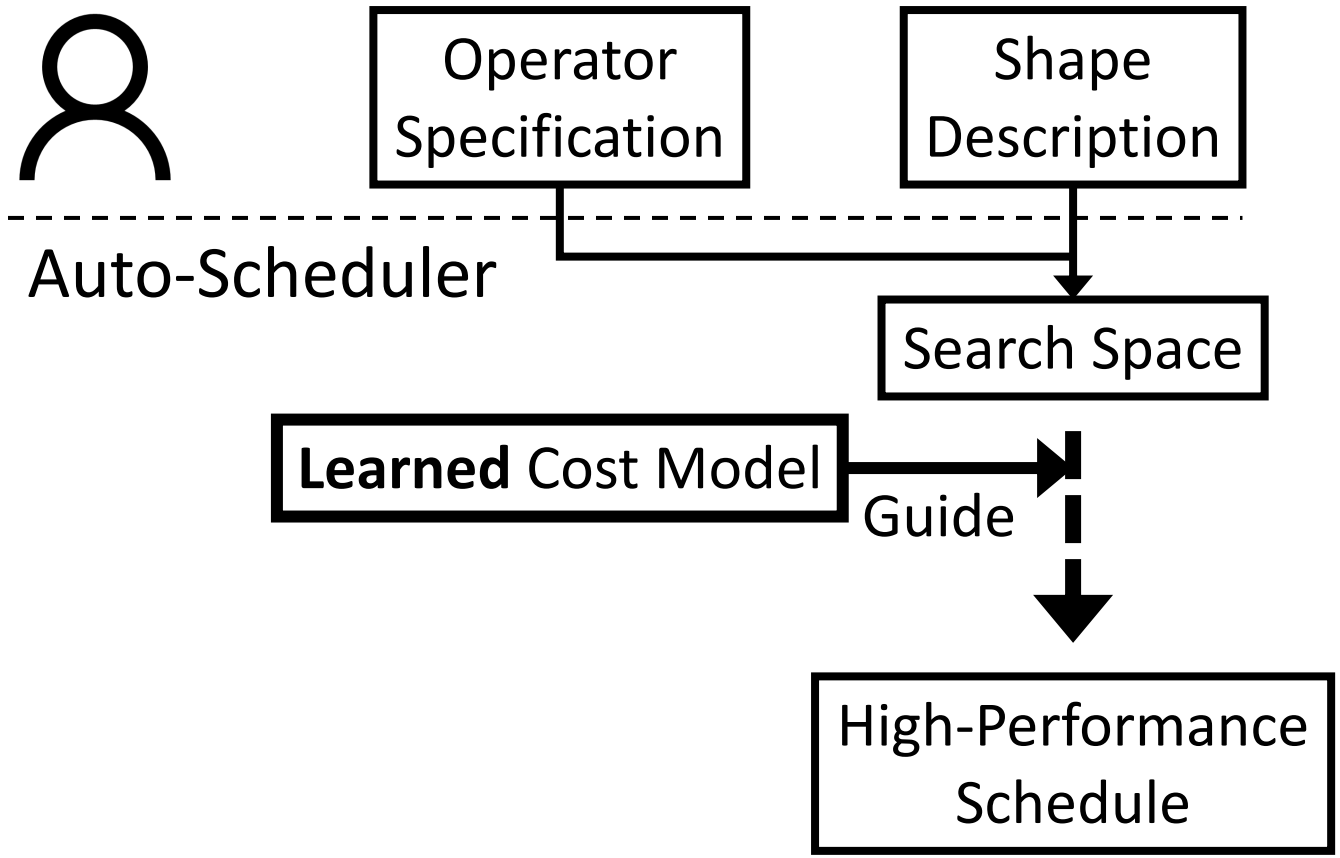
DietCode: Automatic Code Generation for Dynamic Tensor Programs

Bojian Zheng[*1, 2, 3], Ziheng Jiang[*4, 5], Cody Yu[2], Haichen Shen[2],
Josh Fromm[4], Yizhi Liu[2], Yida Wang[2],
Luis Ceze[4, 5], Tianqi Chen[4, 6], Gennady Pekhimenko[1, 2, 3]

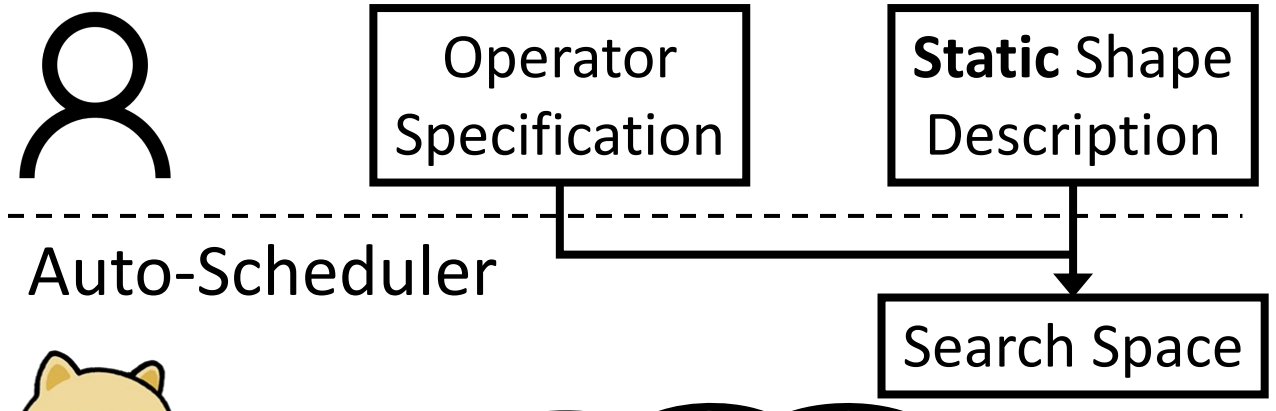
* Equal Contribution

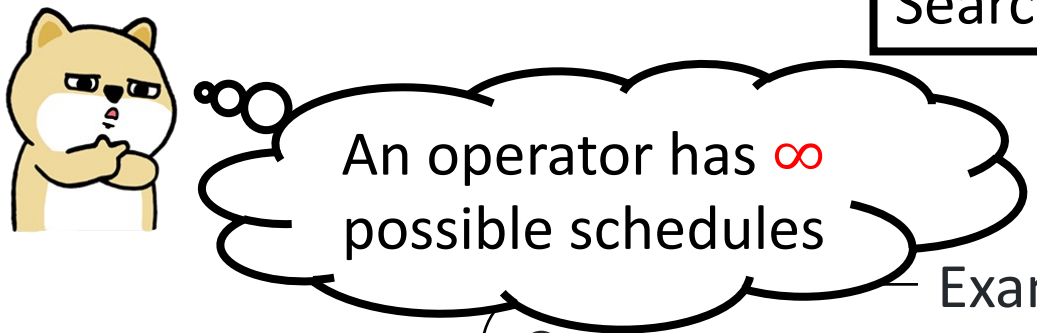
1 2 3 4 5 6

Auto-Scheduler System Overview



Auto-Scheduler System Overview



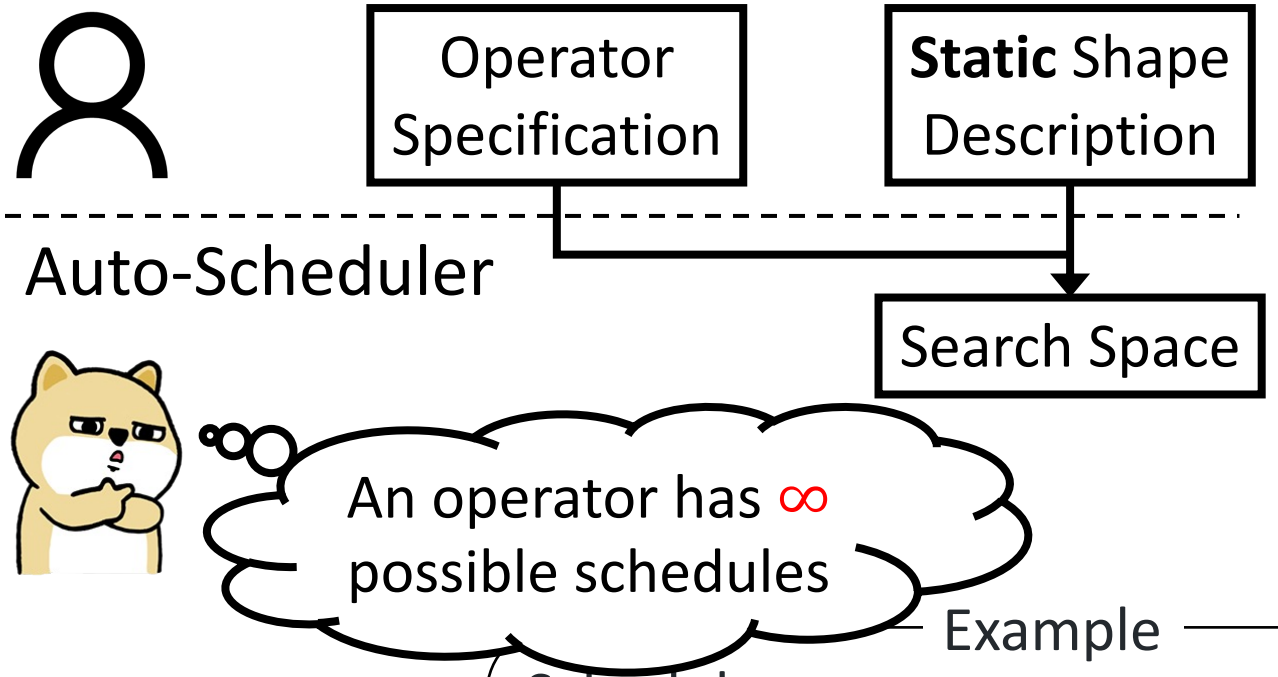


Example

Compute:

```
for (int i = 0; i < 50; ++i) {
  A[i] = ...
}
```

Auto-Scheduler System Overview



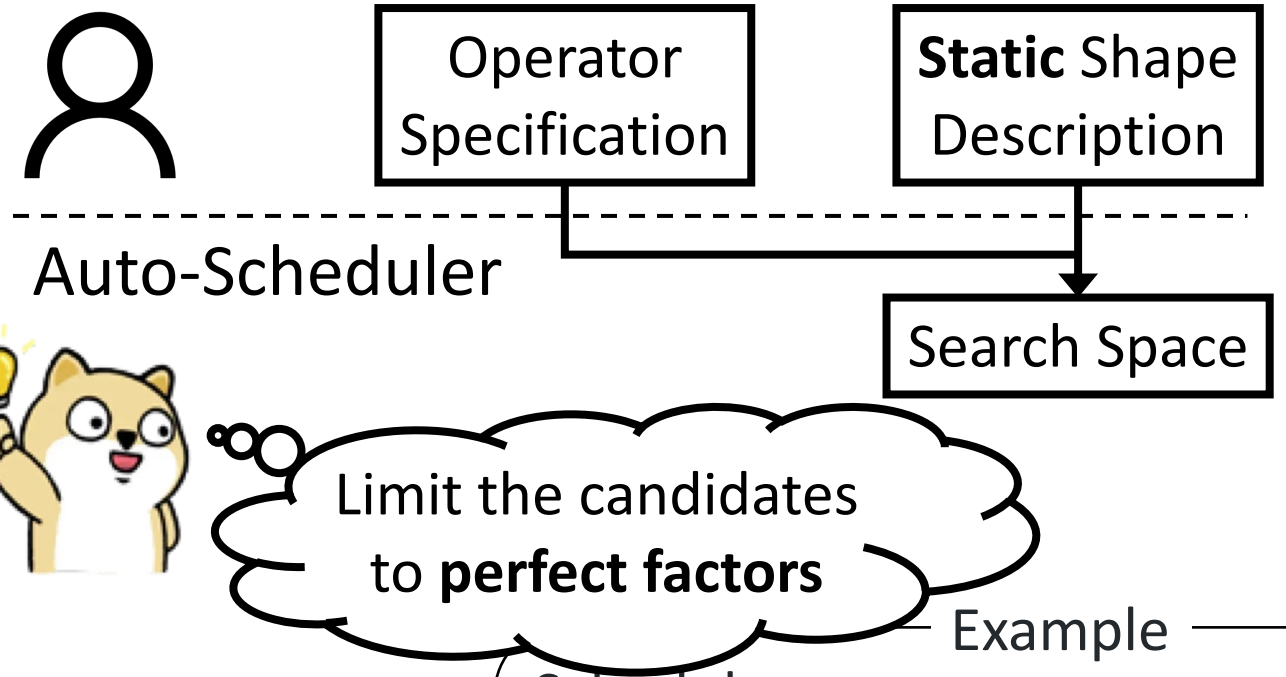
Example

Schedule:

```
for (int io = 0; io < ⌈50/t⌉; ++io) {
  for (int ii = 0; ii < t; ++ii) {
    if (io×t + ii < 50) A[io×t + ii] = ...
  }
}
```

$t \in [2, \infty)$!

Auto-Scheduler System Overview



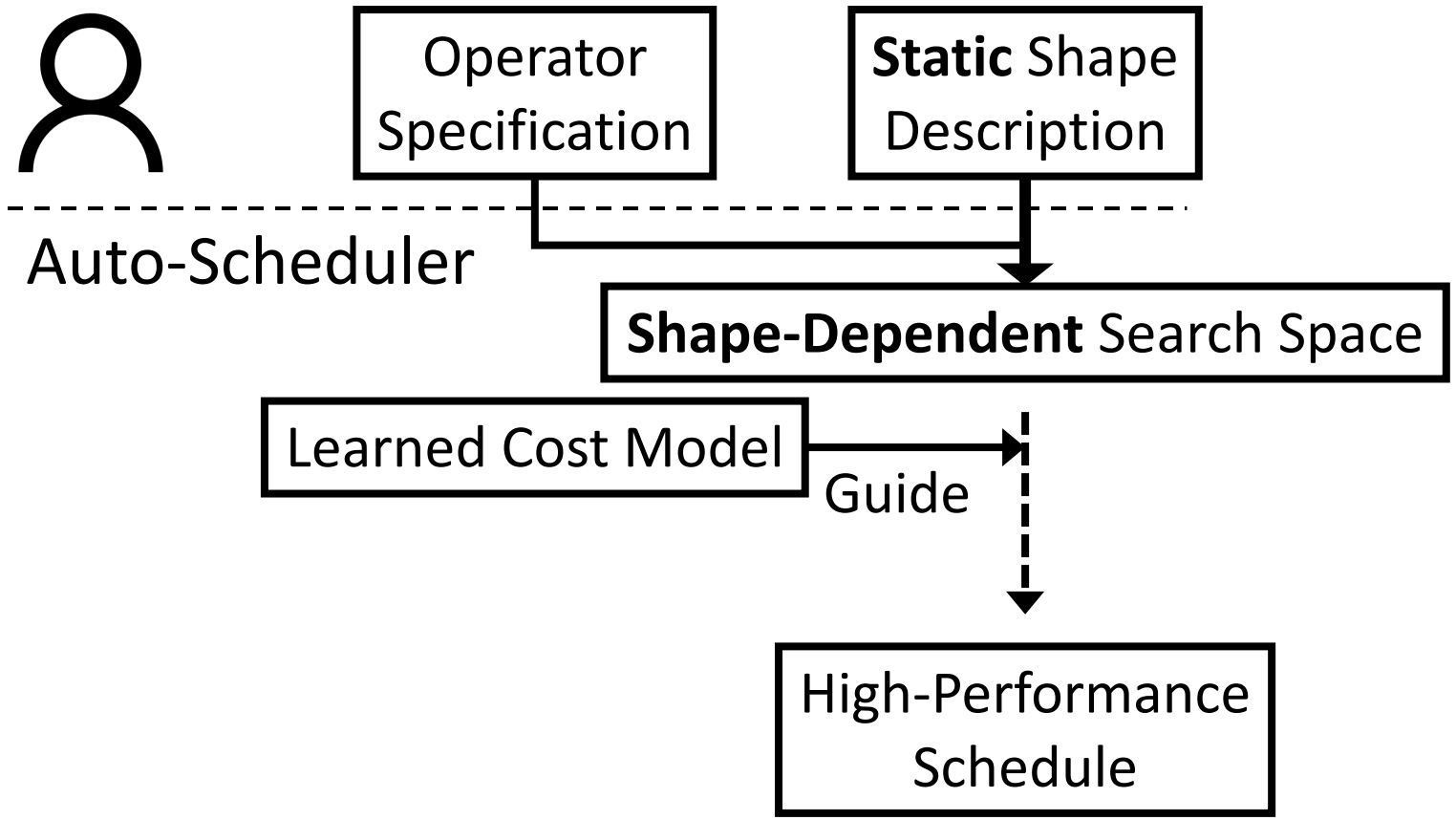
Example

Schedule:

```
for (int io = 0; io < ⌈50/t⌉; ++io) {
  for (int ii = 0; ii < t; ++ii) {
    if (io×t + ii < 50) A[i] = ...
  }
}
```

$t \in \{2, 5, 10, 25\}$

Auto-Scheduler System Overview



Challenges Faced by the Current System

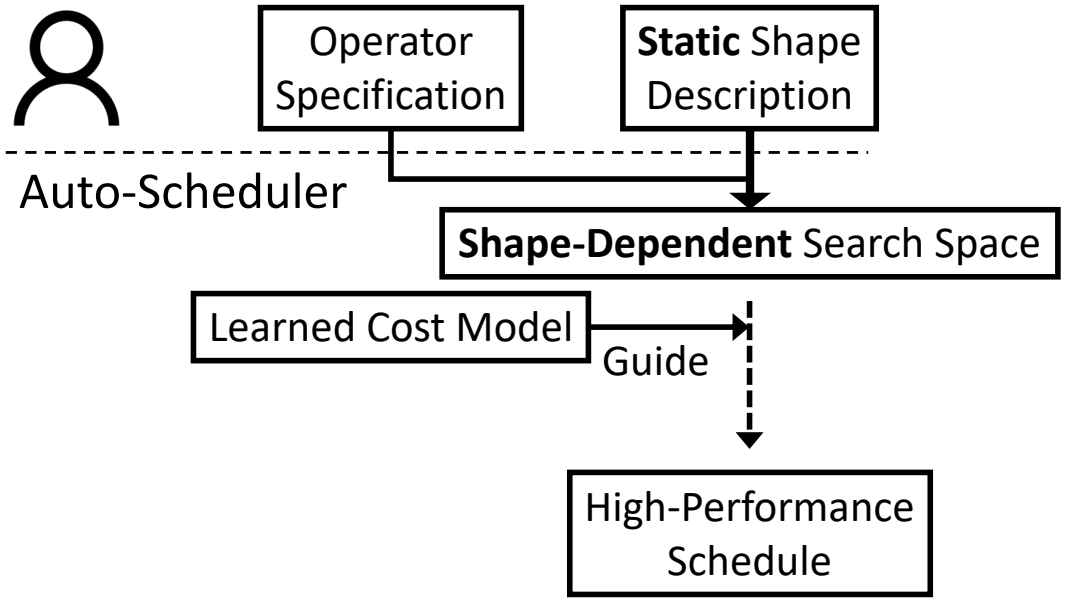
- Challenge #1:
  - Hard to share schedules across different shapes of the same operator.

Example

Schedule:

```
for (int io = 0; io < ⌈50/t⌉; ++io) {
  for (int ii = 0; ii < t; ++ii) {
    A[io×t + ii] = ...
  }
}
```

$t \in \{2, 5, 10, 25\}$



Challenges Faced by the Current System

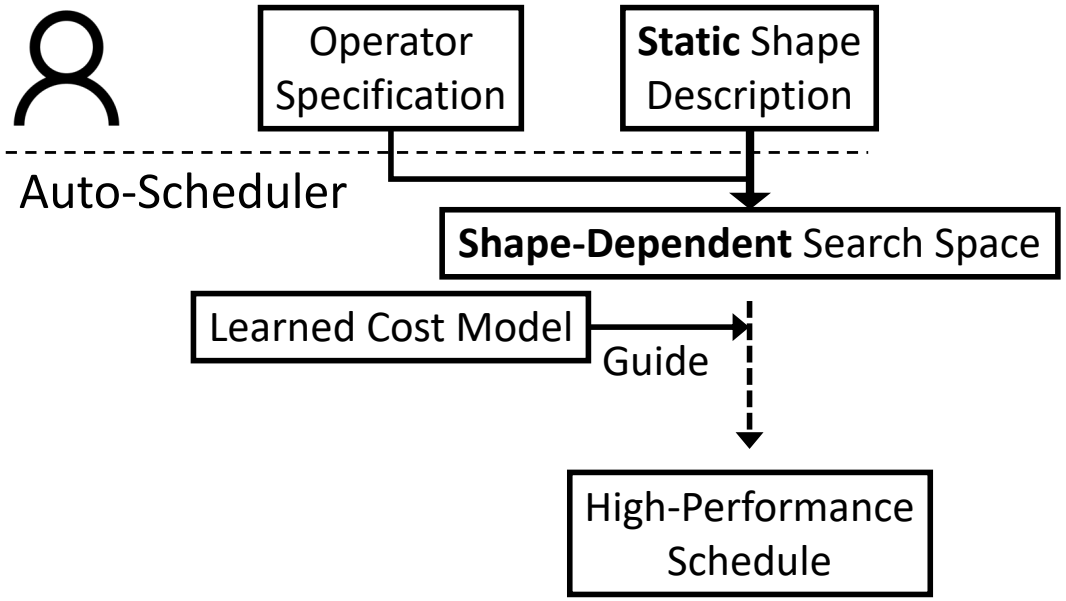
- Challenge #1:
  - Hard to share schedules across different shapes of the same operator.

Example

Schedule:

```
for (int io = 0; io < ⌈49/t⌉; ++io) {
  for (int ii = 0; ii < t; ++ii) {
    A[io×t + ii] = ...
  }
}
```

$t \in \{7\}$ $\cap\{2, 5, 10, 25\} = \emptyset$



Challenges Faced by the Current System

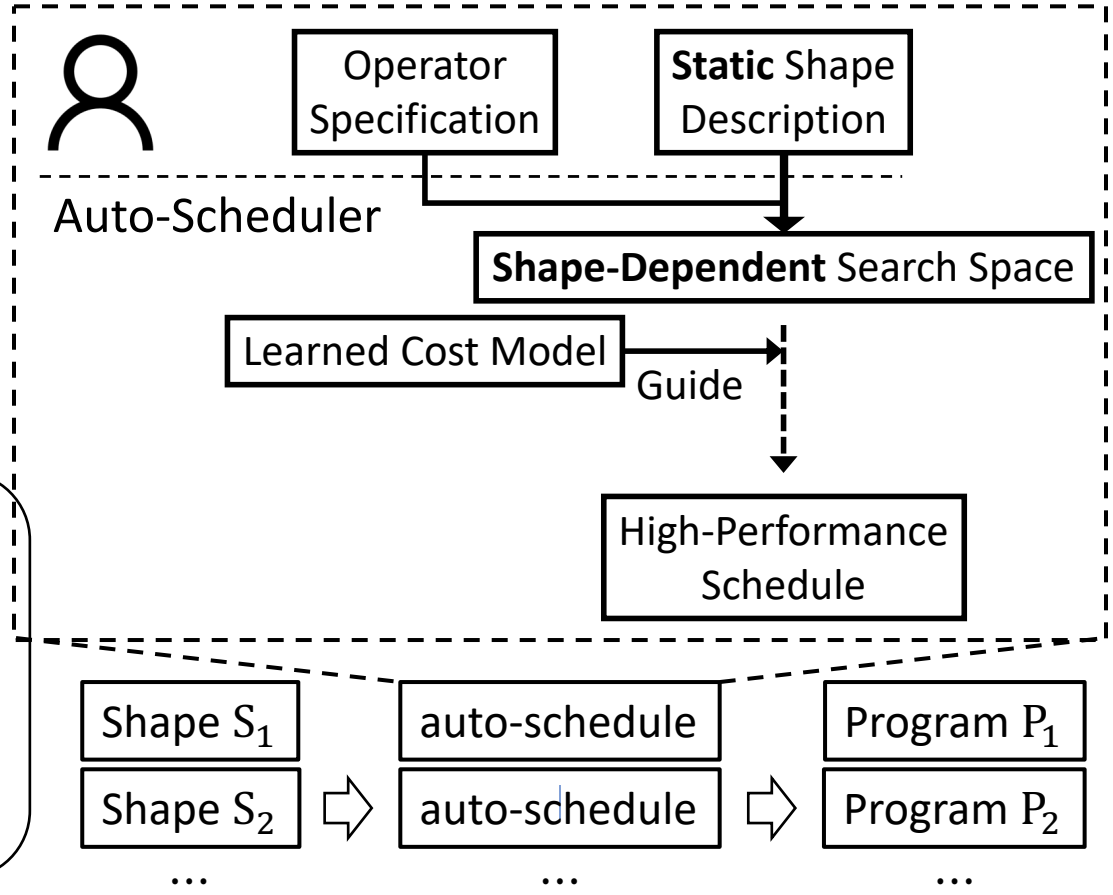
- Challenge #1:
  - Hard to share schedules across different shapes of the same operator.

Example

Schedule:

```
for (int io = 0; io < ⌈49/t⌉; ++io) {
  for (int ii = 0; ii < t; ++ii) {
    A[io×t + ii] = ...
  }
}
```

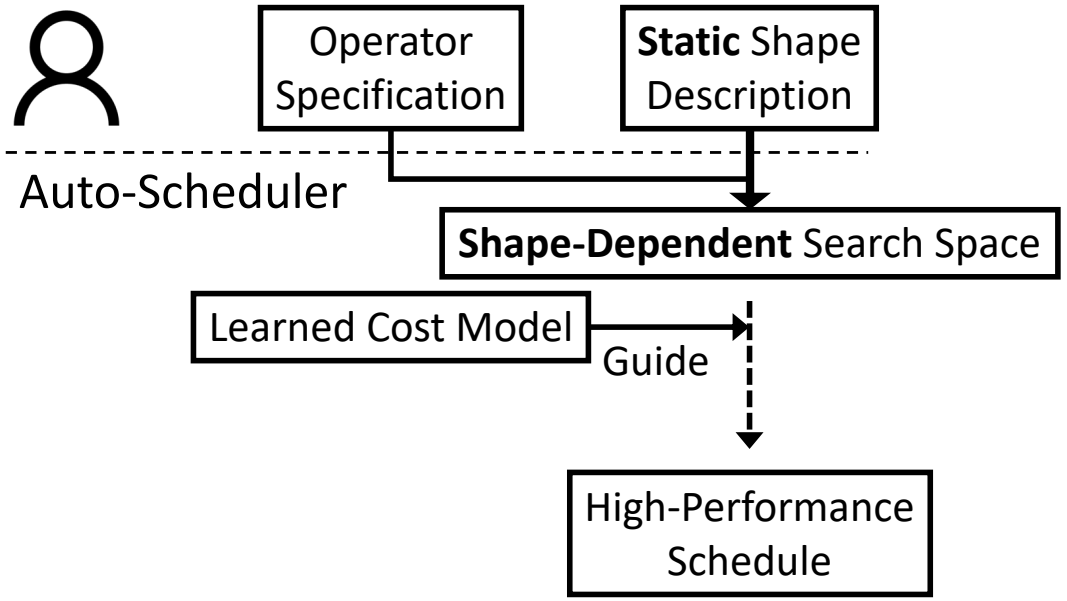
$t \in \{7\}$ $\cap\{2, 5, 10, 25\} = \emptyset$



Prohibitably expensive auto-scheduling time for dynamic-shape workloads.

Challenges Faced by the Current System

- Challenge #2:
  - Can deliver sub-optimal performance for not considering non-perfect candidates.



Example

Schedule:

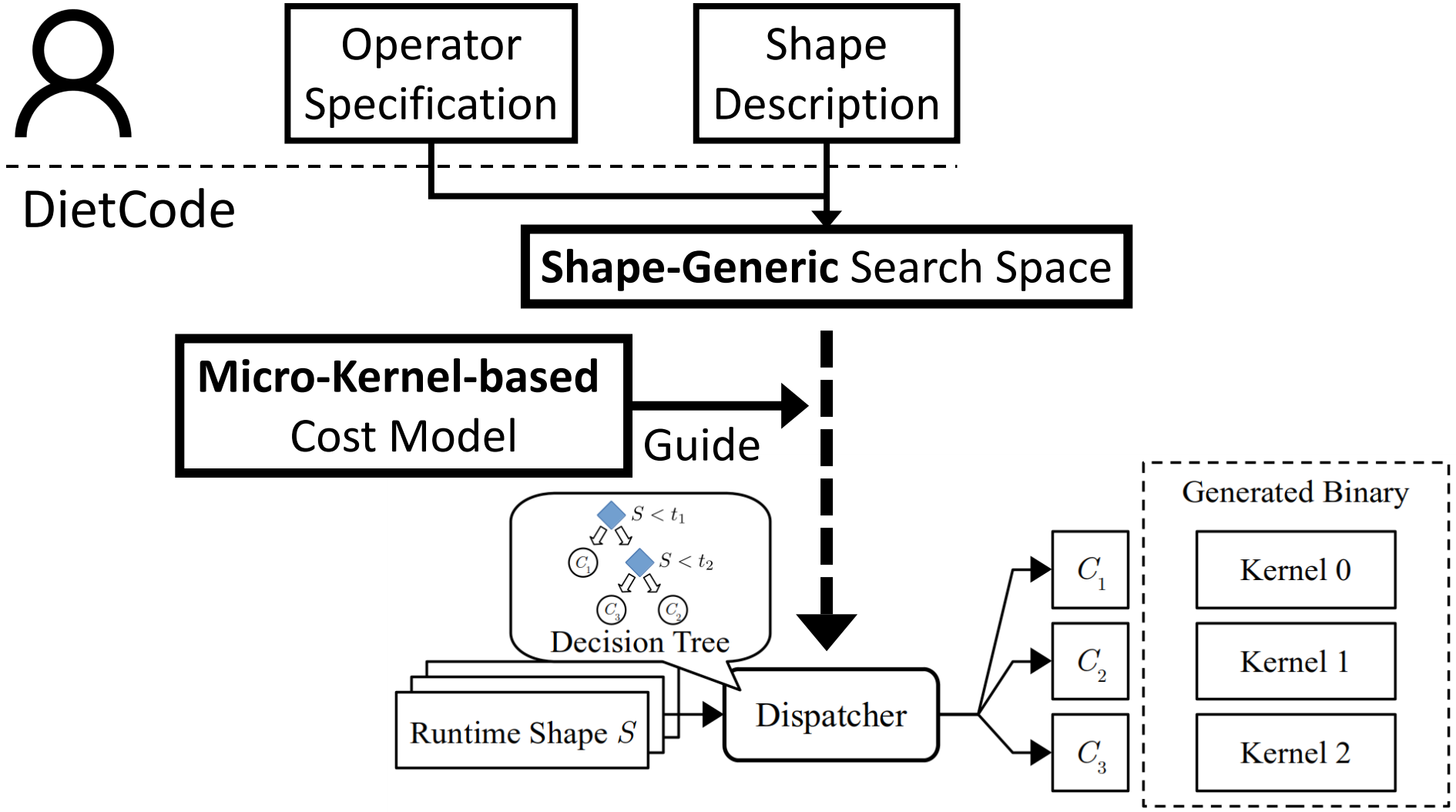
```
for (int io = 0; io < ⌈49/t⌉; ++io) {
  for (int ii = 0; ii < t; ++ii) {
    if (io×t + ii < 49) A[io×t + ii] = ...
  }
}
```

$t \in \{7\}$

$t = 2, 3, \ldots$ might be better candidates

Observation: Performance overhead of if-checks is negligible with **local padding** (i.e., pad tensors locally by the size of local and/or shared memory variables).

DietCode: A New Auto-Scheduler Framework



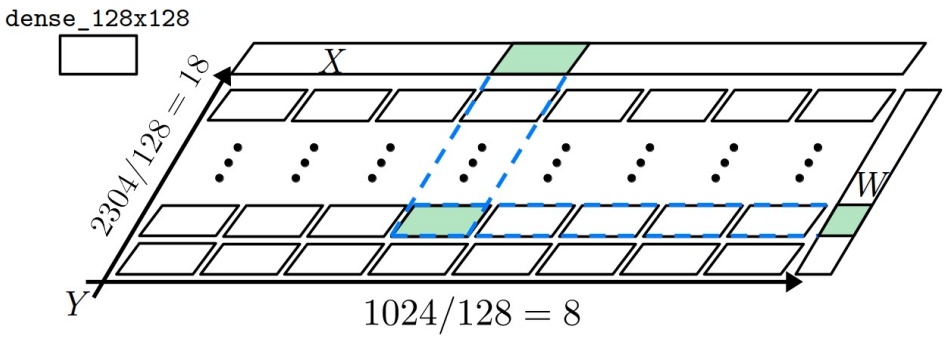
DietCode: Key Ideas

- Key Idea #1: **Shape-Generic** Search Space
  - Composed of micro-kernels. Each does a tile of the entire compute.
  - A micro-kernel can be ported to *all* shapes of the same operator.
  - Sampled from hardware constraints instead of shape factors (i.e., shape-generic).

Example:

$Y = XW^T$ X: $[1024, 768]$, W: $[2304, 768]$ with micro-kernel dense_128x128, which evaluates

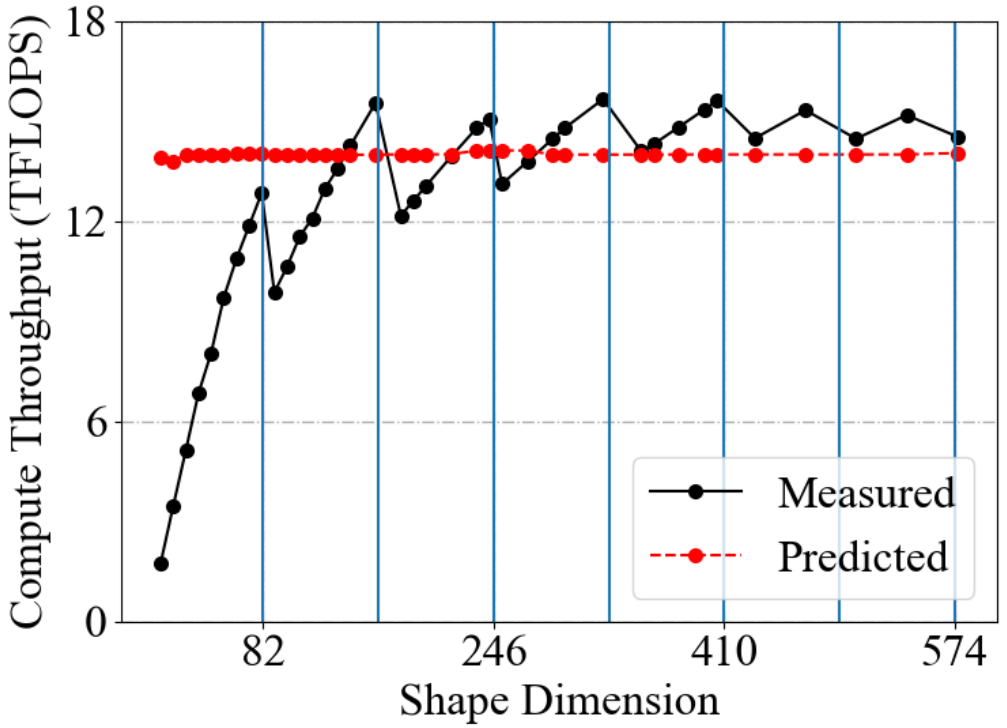
$Y = XW^T$ X: $[128, 768]$, W: $[128, 768]$



DietCode: Key Ideas

- Key Idea #2: **Micro-Kernel-based** Cost Model
  - Observation: A cost model trained on one shape can be inaccurate on other shapes.
  - Compute throughputs exhibit predictable linear trend w.r.t. shape dimensions.
  - Decompose the cost model into:

$$f_{\text{MK}} \cdot f_{\text{spatial}}$$

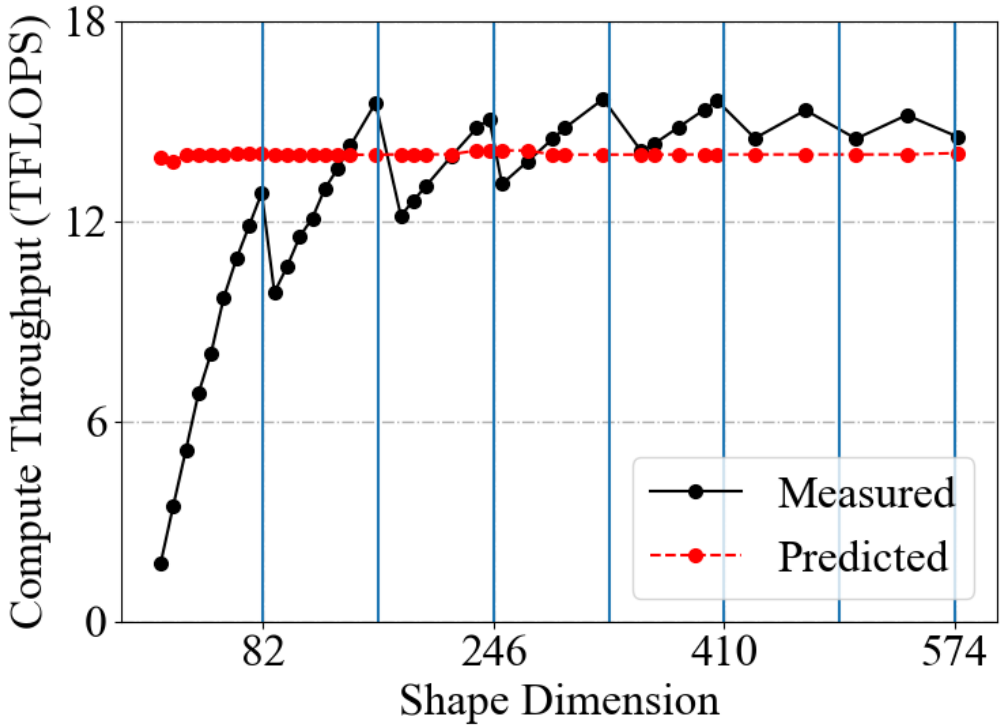


DietCode: Key Ideas

- Key Idea #2: **Micro-Kernel-based** Cost Model
  - Observation: A cost model trained on one shape can be inaccurate on other shapes.
  - Compute throughputs exhibit predictable linear trend w.r.t. shape dimensions.
  - Decompose the cost model into:

$$f_{\text{MK}} \cdot f_{\text{spatial}}$$

    - Trainable Micro-Kernel Cost

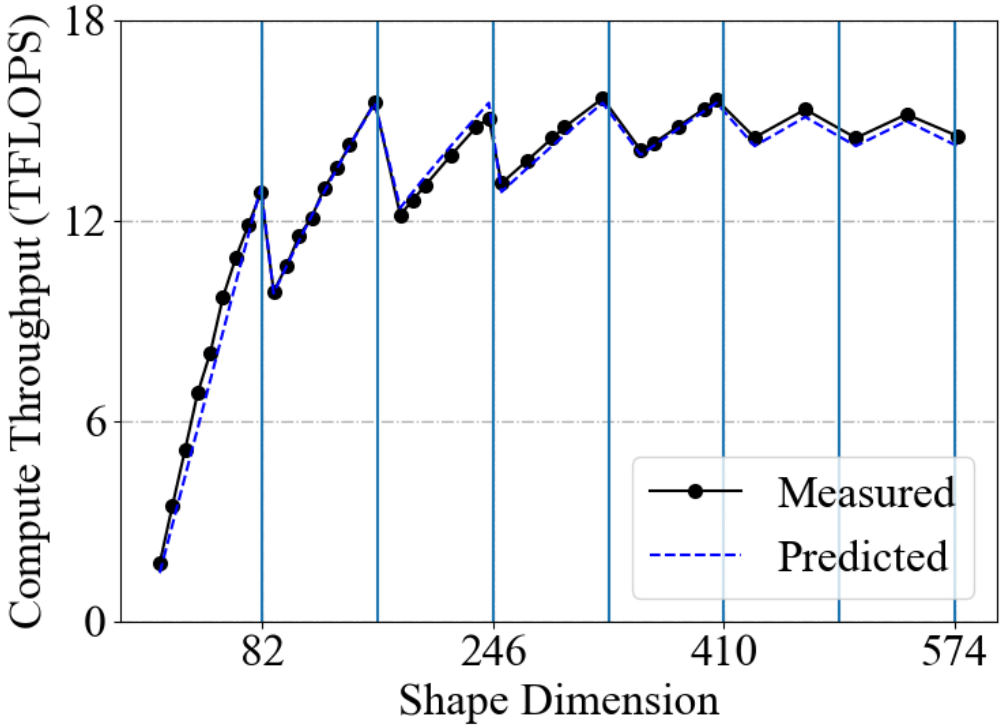


DietCode: Key Ideas

- Key Idea #2: **Micro-Kernel-based** Cost Model
  - Observation: A cost model trained on one shape can be inaccurate on other shapes.
  - Compute throughputs exhibit predictable linear trend w.r.t. shape dimensions.
  - Decompose the cost model into:

$$f_{\text{MK}} \cdot f_{\text{spatial}}$$

    - Trainable Micro-Kernel Cost
    - Analytical Spatial Generalization Cost (linear function)



Evaluation

Hardware: NVIDIA Tesla T4 GPU

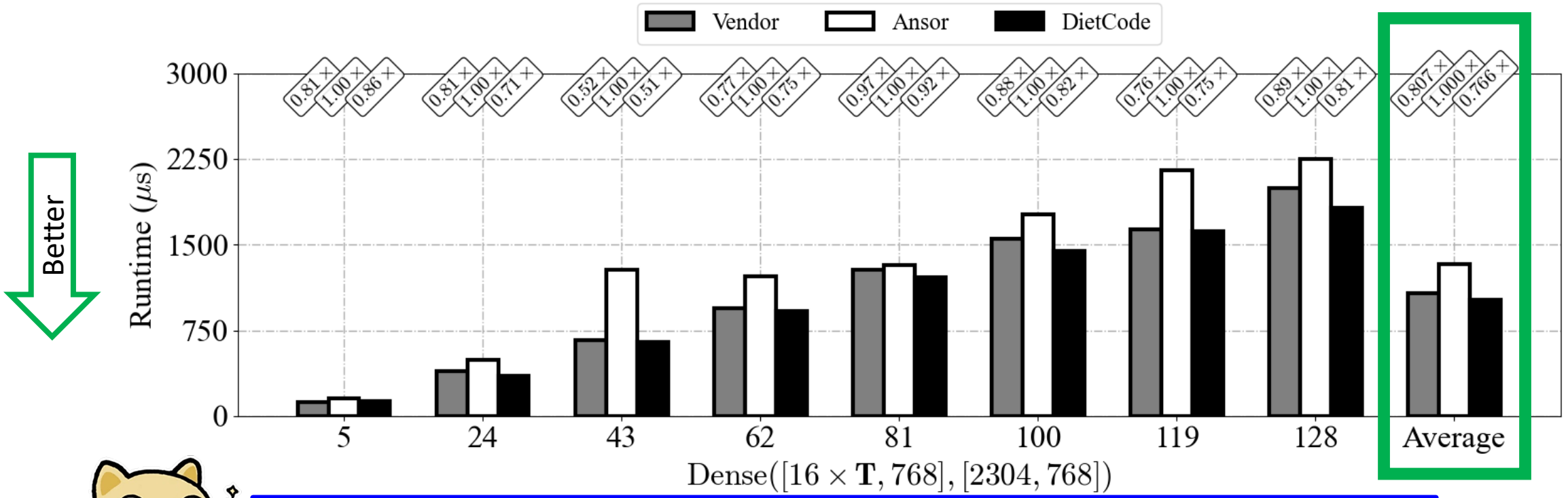
Software: TVM + CUDA + cuDNN

v0.8.dev0

NVIDIA CUDA v11.3

cuDNN v8.3

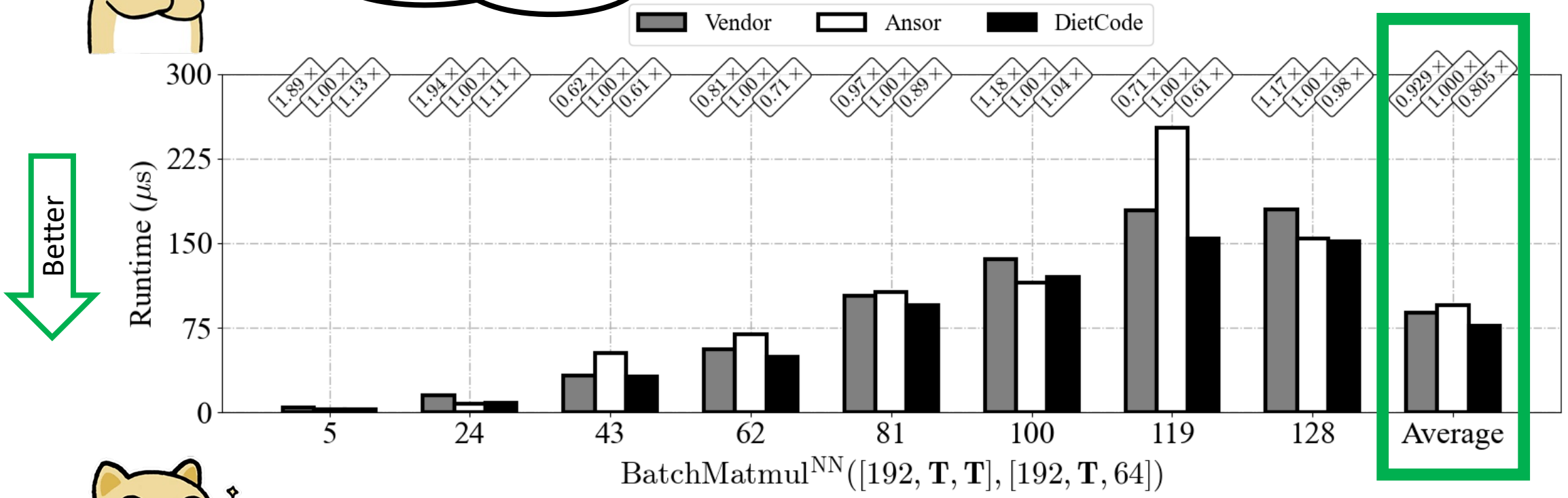
Evaluation



Performance: 30.5% better than Ansor; 5.3% better than Vendor
Auto-Scheduling Time: 5.6× less than Ansor

Evaluation

What about multiple dynamic axes?



Performance: 24.2% better than Ansor; 15.4% better than Vendor

Summary

- DietCode: An auto-scheduler for dynamic-shape workloads.
- Based on 2 key ideas:
  (1) Shape-Generic Search Space and
  (2) Micro-Kernel-based Cost Model
- Key Features:
  - **Auto-Schedule Once and For All Shapes**: Large reduction in the auto-scheduling time $5.6\times$ on dynamic-shape workloads.
  - **Better Performance**: Up to 30.5% speedup than Ansor, 15.4% than Vendor.

Last Time

- Why Machine Learning Compilers?
  - State-of-the-Art Machine Learning Frameworks Design, and Flaws:
  1. Vendor libraries not delivering the optimal performance.

- 2 Classes of Machine Learning Compilers:
  - **Halide[1]/TVM[2]**: Easier to write high-performance programs.

[1] J. Ragan-Kelley et al. *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines*. PLDI 2013
[2] T. Chen et al. *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*. OSDI 2018

Agenda for Today

- Why Machine Learning Compilers?
  - State-of-the-Art Machine Learning Frameworks Design, and Flaws:
  1. Vendor libraries not delivering the optimal performance.
  2. Distinct representations at different system levels.
- 2 Classes of Machine Learning Compilers:
  - **Halide[1]/TVM[2]**: Easier to write high-performance programs.
  - **MLIR[3]/TensorIR[4]**: Unified compiler infrastructure.

[1] J. Ragan-Kelley et al. *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines*. PLDI 2013

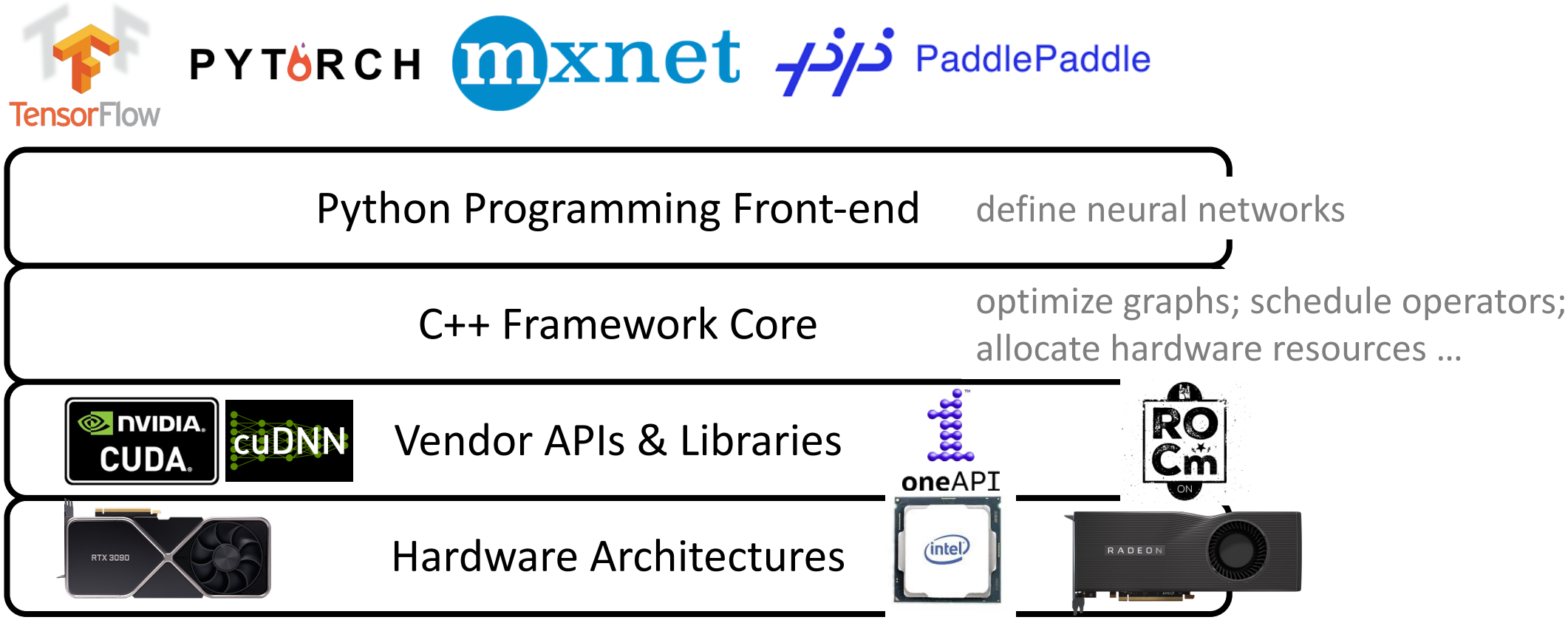
[2] T. Chen et al. *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*. OSDI 2018

[3] C. Lattner et al. *MLIR: Scaling Compiler Infrastructure for Domain Specific Computation*. CGO 2021

[4] S. Feng et al. *Understanding TensorIR: An Abstraction for Tensorized Program Optimization*. TVMCon 2021

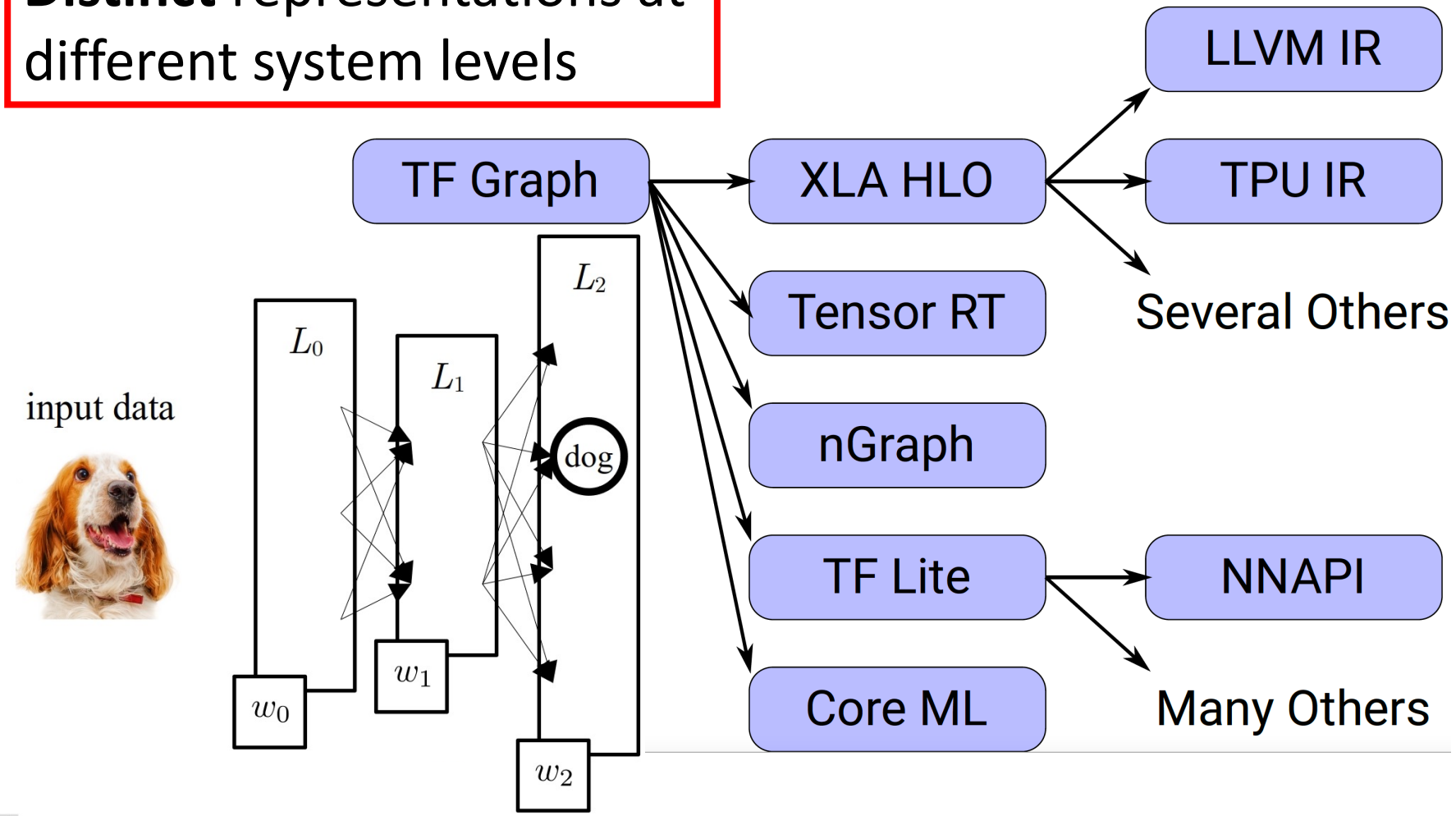
Why Unity?

- State-of-the-Art Machine Learning Frameworks Design:



TensorFlow System Overview

Distinct representations at different system levels



TVM System Overview

Distinct representations at different system levels

Relay

```
output = relay.dense(data, weight)
```

TE

```
for i in range(...):
  for j in range(...):
    for k in range(...):
      Y[i][j] += X[i][k] * W[j][k]
```

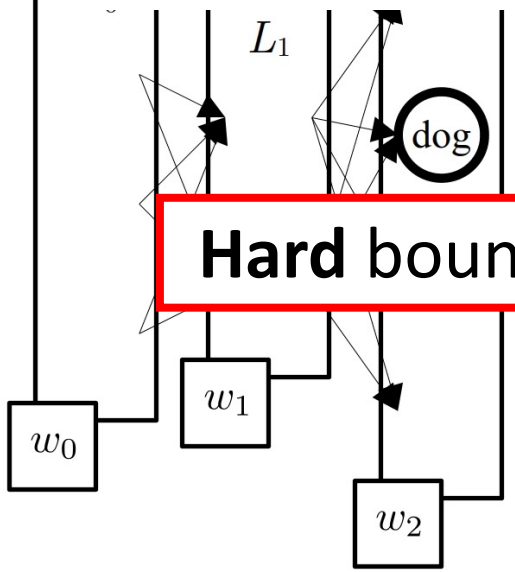
Schedule Tree

```
s = create_schedule(Y.op)
Y_local = s.cache_write([Y], "local")
i, j, k = tuple(Y_local.op.axis)
...
```

TIR

```
primfn(X_1: handle, W_1: handle, Y_1: handle) -> ()
attr [IterVar(blockIdx.x: int32)] "thread_extent" = 144;
allocate(Y.local: Pointer(local float32), [64]);
allocate(X.shared: Pointer(shared float32), [512]);
allocate(W.shared: Pointer(shared float32), [512]);
attr [IterVar(threadIdx.x: int32)] "thread_extent" = 256 {
  ...
}
```

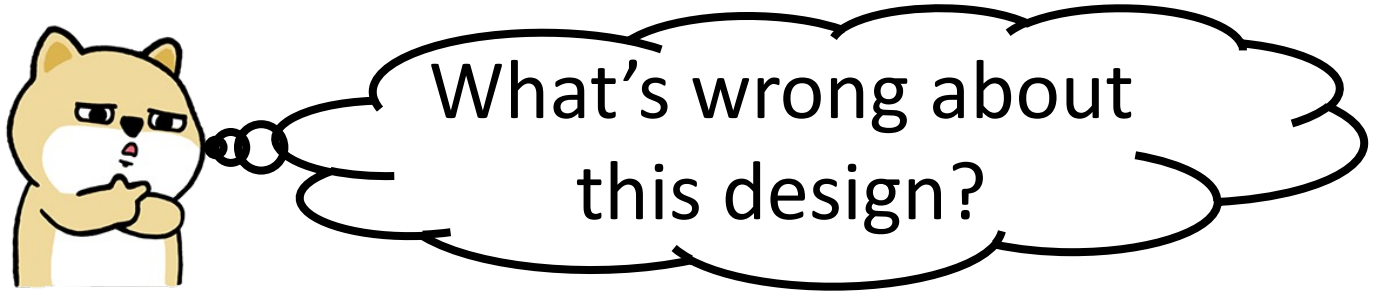
input data



Hard boundary between representations

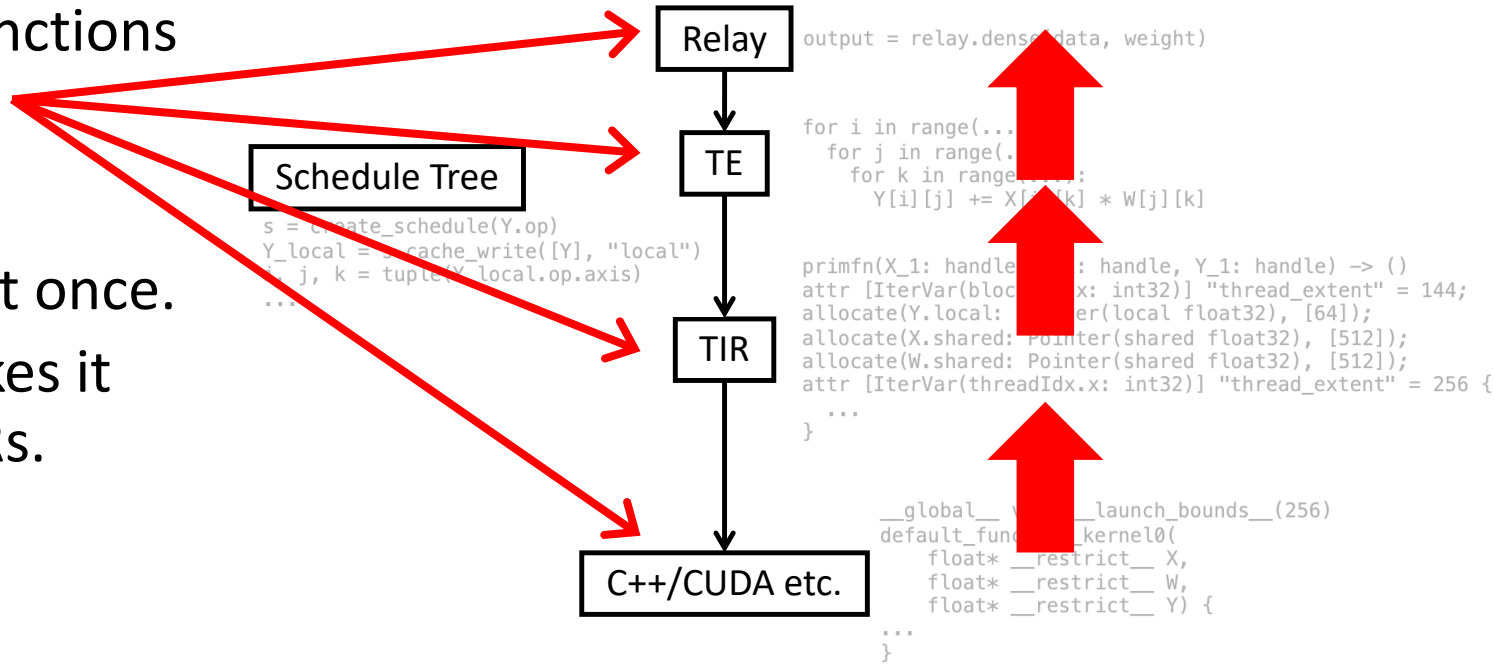
C++/CUDA etc.

```
__global__ void __launch_bounds__(256)
default_function_kernel0(
    float* __restrict__ X,
    float* __restrict__ W,
    float* __restrict__ Y) {
...
}
```



- Duplicated Infrastructure
  - Similar utility classes and functions for each IR.
- Premature Lowering
  - Need to lower ALL regions at once.
  - Unidirectional lowering makes it hard to recover high-level IRs.

TVM System Overview



Multi-Level Intermediate Representation[1]

- *abbrev*. MLIR
- Objective: Unified compiler infrastructure.
- Key Idea: Same infrastructure, distinct **dialects** at different levels.
- Each dialect is a set of operators.

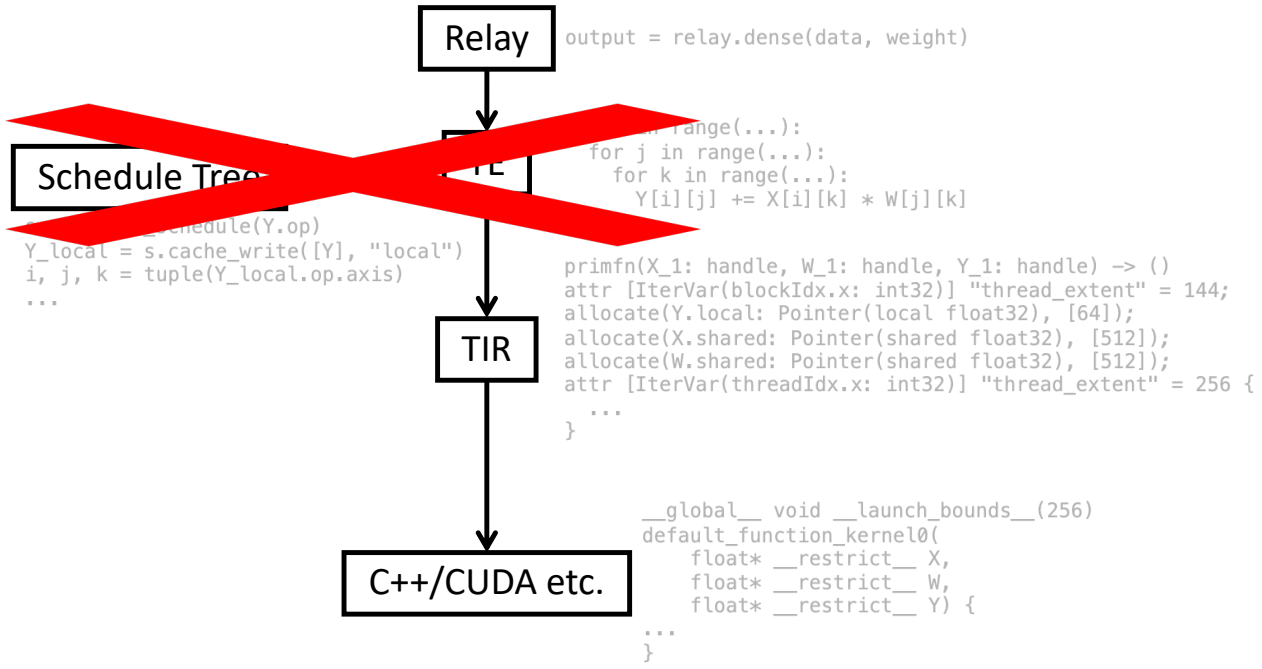
IRs	Operators
TensorFlow Graphs Affine For LLVM Instructions	

[1] C. Lattner. *MLIR: A Compiler Infrastructure for the End of Moore's Law*. arXiv 2020

TensorIR[1]

- Objective: Unified compiler infrastructure.
- Key Idea: Remove schedule tree representation.

TVM System Overview



[1] S. Feng et al. *Understanding TensorIR: An Abstraction for Tensorized Program Optimization*. TVMCon 2021

TensorIR[1]

Schedule Tree

```
s = te.create_schedule(Y)
Y_local = s.cache_write([Y], "local")
i, j, k = tuple(Y_local.op.axis)
i_o_i, i_i = s[Y_local].split(i, factor=8)
i_o_o_i, i_o_i = s[Y_local].split(i_o_i, factor=2)
i_o_o_o, i_o_o_i = s[Y_local].split(i_o_o_i, factor=1)
...
```

TensorIR

```
def MatMul(x, w, y):
  X = tir.match_buffer(x, (1024, 1024), "float32")
  W = tir.match_buffer(w, (1024, 1024), "float32")
  Y = tir.match_buffer(y, (1024, 1024), "float32")
  reducer = tir.comm_reducer(lambda x, y: x + y, tir.float32(0))

  with tir.block([1024, 1024, tir.reduce_axis(0, 1024)], "Y") \
       as [vi, vj, vk]:
    reducer.step(Y[vi, vj], X[vi, vk] * W[vk, vj])
```

(-) Not Intuitive

(-) Hard to support even existing hardware primitives.

TensorIR[1]

Schedule Tree

```
s = te.create_schedule(Y)
Y_local = s.cache_write([Y], "local")
i, j, k = tuple(Y_local.op.axis)
i_o_i, i_i = s[Y_local].split(i, factor=8)
i_o_o_i, i_o_i = s[Y_local].split(i_o_i, factor=2)
i_o_o_o, i_o_o_i = s[Y_local].split(i_o_o_i, factor=1)
...
```

(-) Not Intuitive

(-) Hard to support even existing hardware primitives.

TensorIR

```
def MatMul(x, w, y):
  X = tir.match_buffer(x, [1024, 1024])
  W = tir.match_buffer(w, [1024, 1024])
  Y = tir.match_buffer(y, [1024, 1024])
  reducer = tir.comm_reducer(lambda a, b: a + b, tir.float32(0))
  for i0_outer, i1_outer, i2_outer, i2_inner, \
      i0_inner, i1_inner in tir.grid(32, 32, 256, 4, 32, 32):
    with tir.block([1024, 1024, tir.reduce_axis(0, 1024)], "C") \
        as [vi, vj, vk]:
      tir.bind(vi, ((i0_outer*32) + i0_inner))
      tir.bind(vj, ((i1_outer*32) + i1_inner))
      tir.bind(vk, ((i2_outer*4) + i2_inner))
      reducer.step(Y[vi, vj], (X[vi, vk]*W[vk, vj]))
```

(+) Expressive

(+) Hierarchical `Block` structure makes it easier to map to hardware primitives.

Concluding Remarks

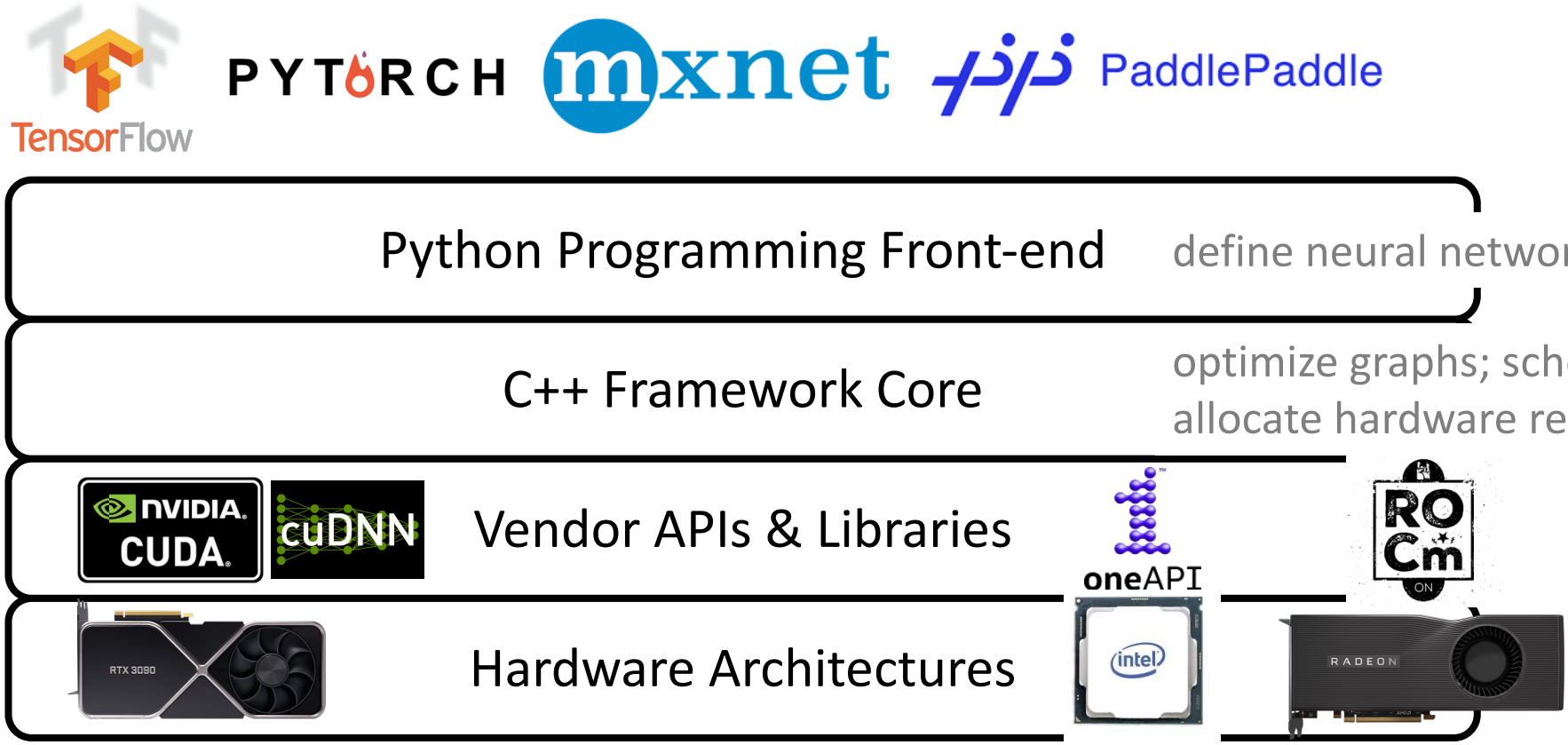
- Why Machine Learning Compilers?
  - State-of-the-Art Machine Learning Frameworks Design, and Flaws:
  1. Vendor libraries not delivering the optimal performance.
  2. Distinct representations at different system levels.
- 2 Classes of Machine Learning Compilers:
  - **Halide[1]/TVM[2]**: Easier to write high-performance programs.
  - **MLIR[3]/TensorIR[4]**: Unified compiler infrastructure.

[1] J. Ragan-Kelley et al. *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines*. PLDI 2013

[2] T. Chen et al. *TVM: An Automated End-to-End Optimizing Compiler for Deep Learning*. OSDI 2018

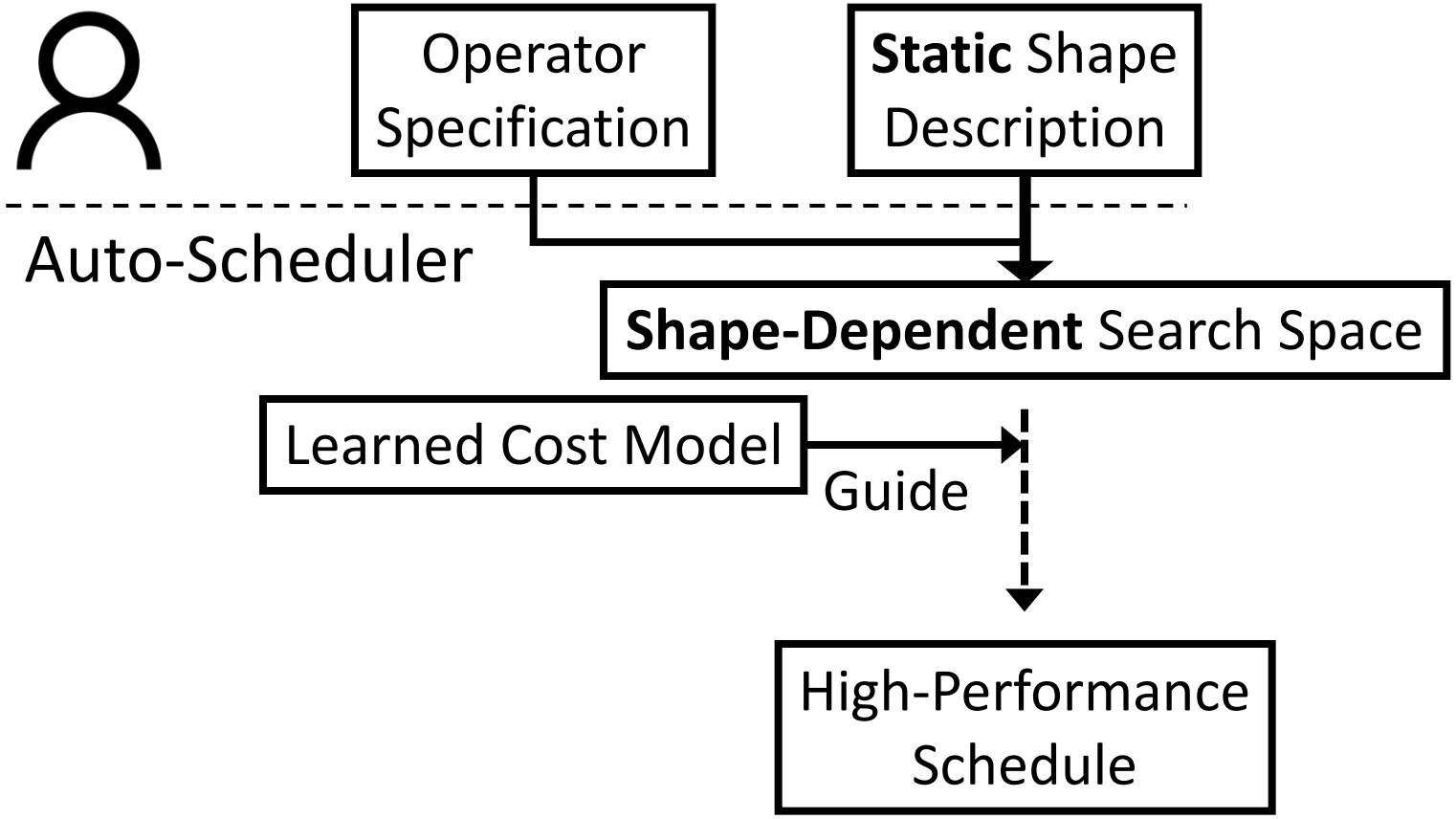
[3] C. Lattner et al. *MLIR: Scaling Compiler Infrastructure for Domain Specific Computation*. CGO 2021

[4] S. Feng et al. *Understanding TensorIR: An Abstraction for Tensorized Program Optimization*. TVMCon 2021


TensorFlow
PYTORCH
mxnet
PaddlePaddle
Python Programming Front-end
define neural networks
C++ Framework Core
optimize graphs; schedule operators;
allocate hardware resources …
NVIDIA
CUDA
cuDNN
Vendor APIs & Libraries
oneAPI
ROCm
RTX 3090
intel
RADEON
Hardware Architectures

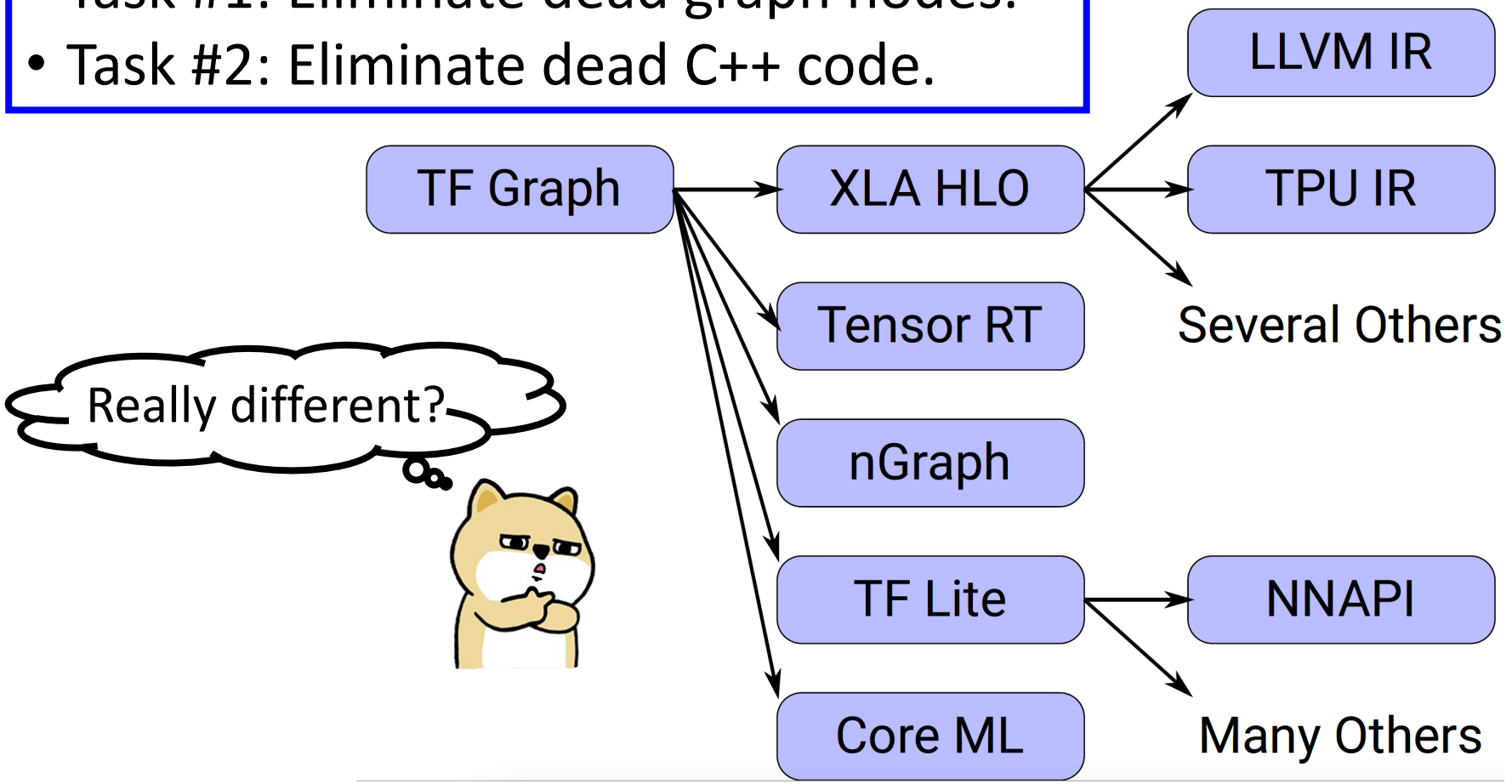
X: [{\color{red}\mathbf{129}}, K],
W: [{\color{red}\mathbf{129}}, K]?

$$
y=xw^\top\quad
\begin{cases}
  x: [32, *], w: [32, *] \\
  x: [64, *], w: [64, *] \\
  x: [128, *], w: [128, *]
\end{cases}
$$


Operator Specification
Static Shape Description
Auto-Scheduler
Shape-Dependent Search Space
Learned Cost Model
Guide
High-Performance Schedule

Current TensorFlow System

- Task #1: Eliminate dead graph nodes.
- Task #2: Eliminate dead C++ code.



TVM System Overview