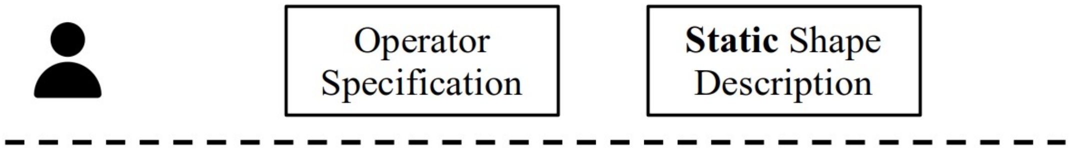
DietCode: Automatic Code Generation for Dynamic Tensor Programs

Bojian Zheng*[1, 2, 3], Ziheng Jiang*[4], Cody Yu[2], Haichen Shen[2],
Josh Fromm[5], Yizhi Liu[2], Yida Wang[2],
Luis Ceze[5, 6], Tianqi Chen[5, 7], Gennady Pekhimenko[1, 2, 3]

* Equal Contribution

1 2 aws 3 VECTOR INSTITUTE 4 ByteDance 5 OctoML 6 7

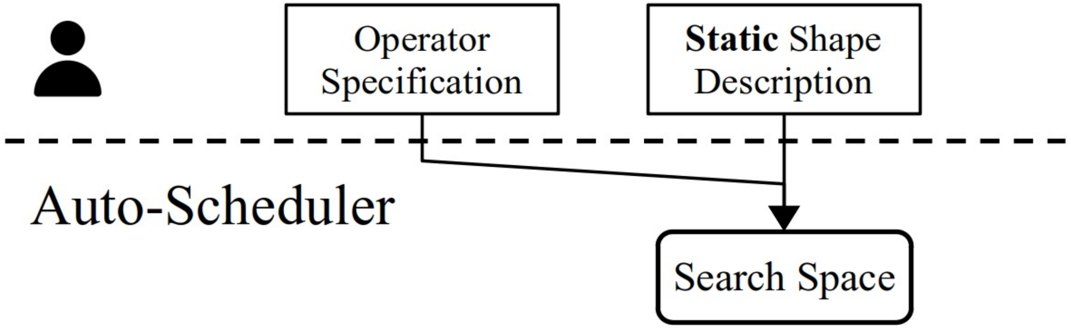
Background: Current Auto-Scheduler Design



Operator Specification

Static Shape Description

Background: Current Auto-Scheduler Design



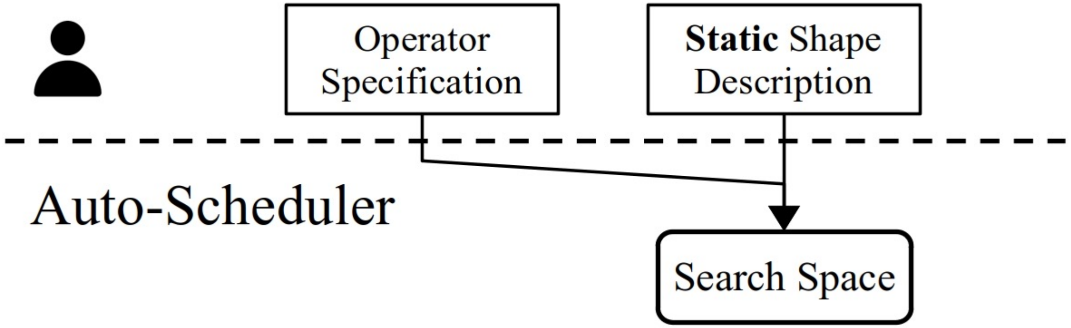
An operator can have infinitely many possible schedules

Example

Compute:

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

Background: Current Auto-Scheduler Design



Operator Specification

Static Shape Description

Auto-Scheduler

Search Space

An operator can have infinitely many possible schedules

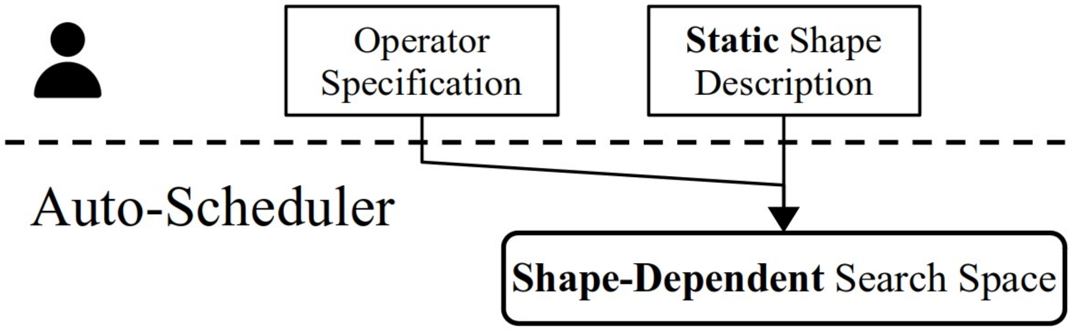
Example

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] = ...
  }
}
```

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

Background: Current Auto-Scheduler Design



Operator Specification

Static Shape Description

Auto-Scheduler

Shape-Dependent Search Space

Limit the candidates to **perfect factors**

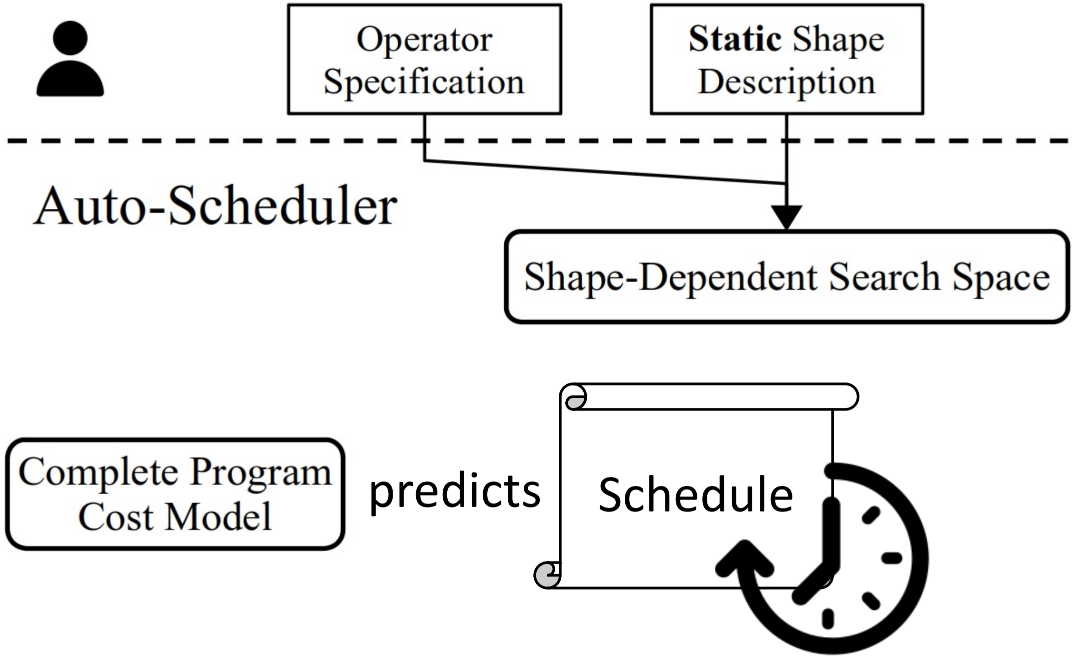
Example

Schedule:

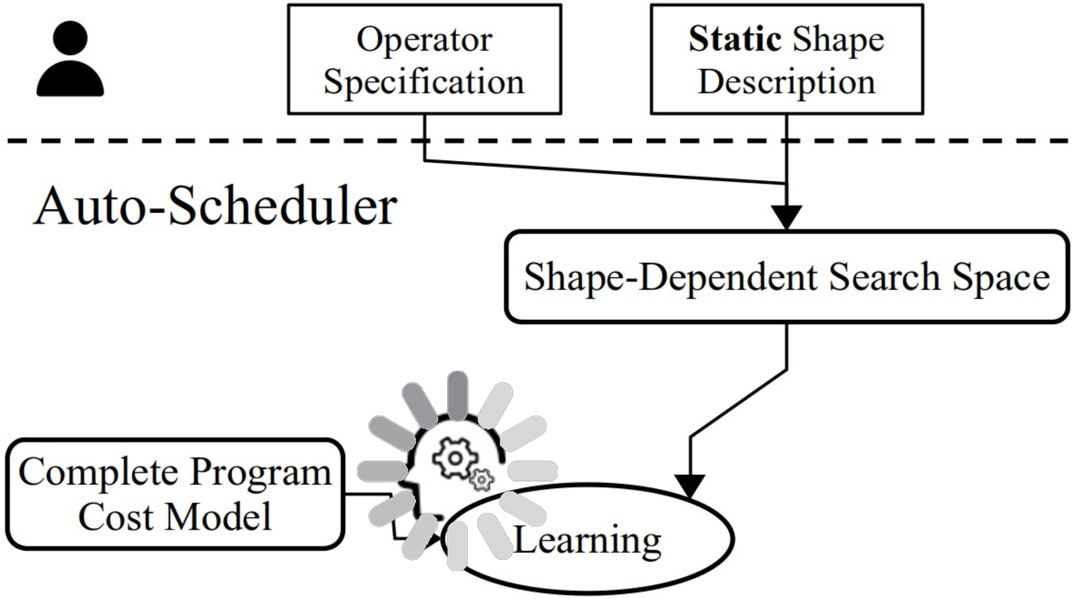
```
for (int io = 0; io < ⌈50/t⌉; ++io) {
  for (int ii = 0; ii < t; ++ii) {
    ~~if (io×t + ii < 50)~~ A[io×t + ii] = ...
  }
}
```

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

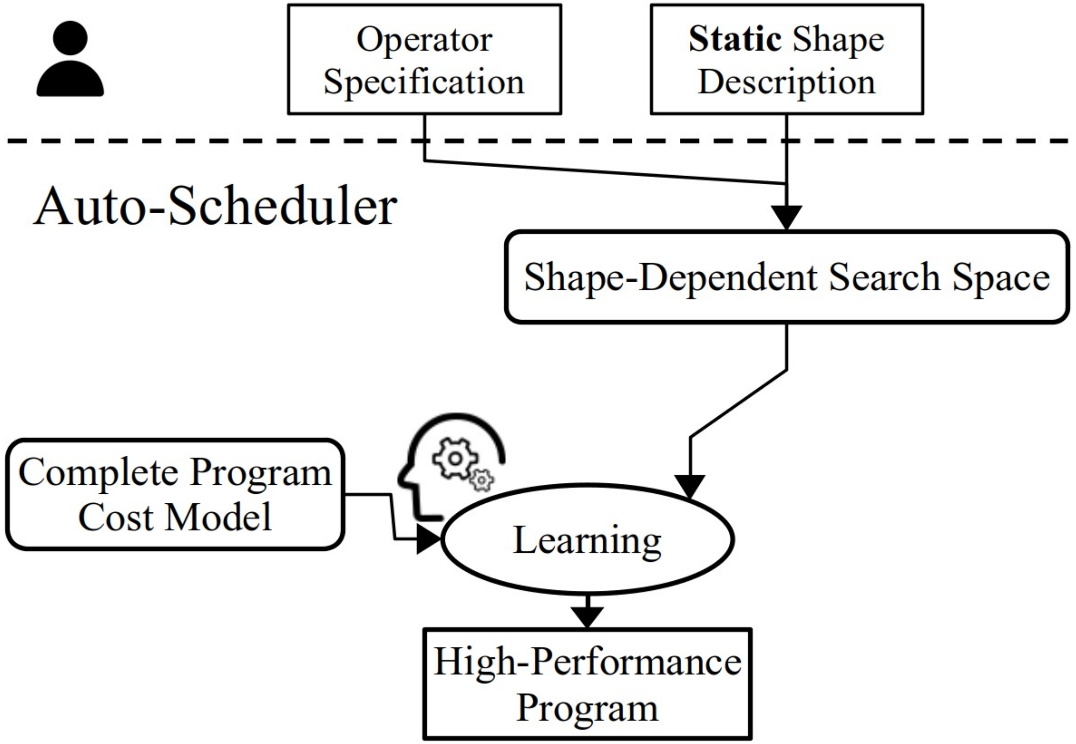
Background: Current Auto-Scheduler Design


Operator Specification
Static Shape Description
Auto-Scheduler
Shape-Dependent Search Space
Complete Program Cost Model
predicts
Schedule

Background: Current Auto-Scheduler Design


Operator Specification
Static Shape Description
Auto-Scheduler
Shape-Dependent Search Space
Complete Program Cost Model
Learning

Background: Current Auto-Scheduler Design


Operator Specification
Static Shape Description
Auto-Scheduler
Shape-Dependent Search Space
Complete Program Cost Model
Learning
High-Performance Program

Challenges Faced by the Current Design

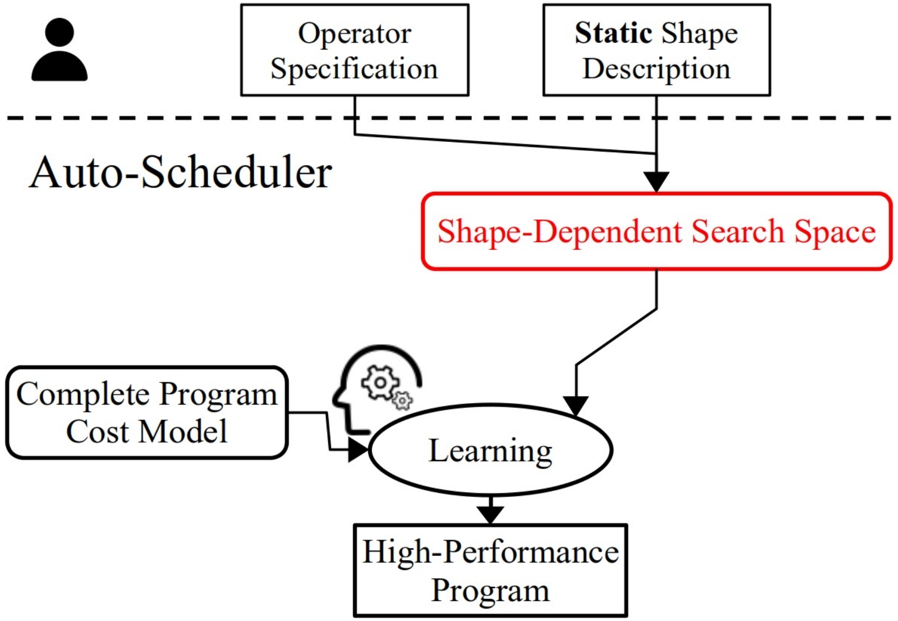
- Challenge #1:
  - Hard to share schedules across different shapes of the same operator.

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 Design

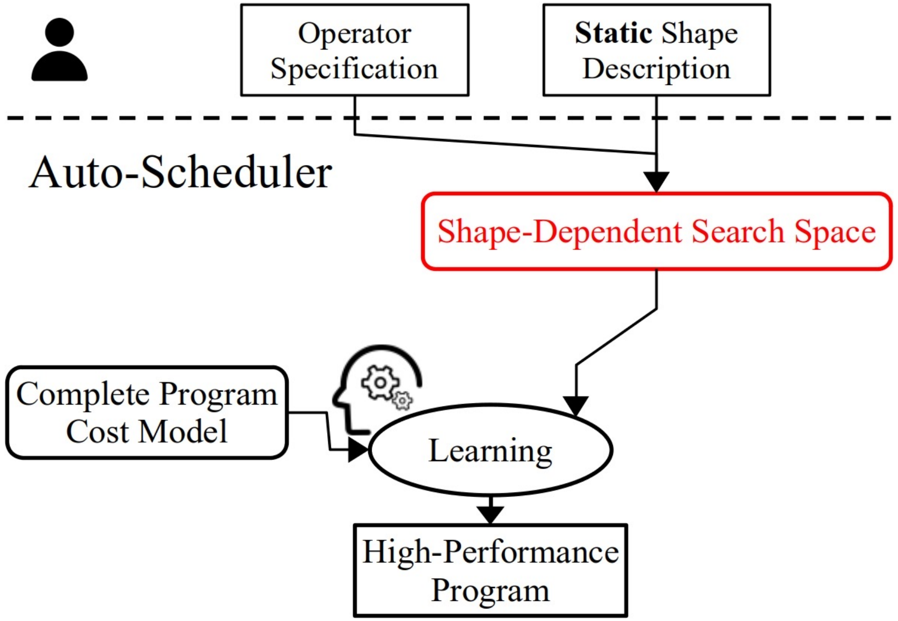
- Challenge #1:
  - Hard to share schedules across different shapes of the same operator.

Example

Schedule:

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

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



Challenges Faced by the Current Design

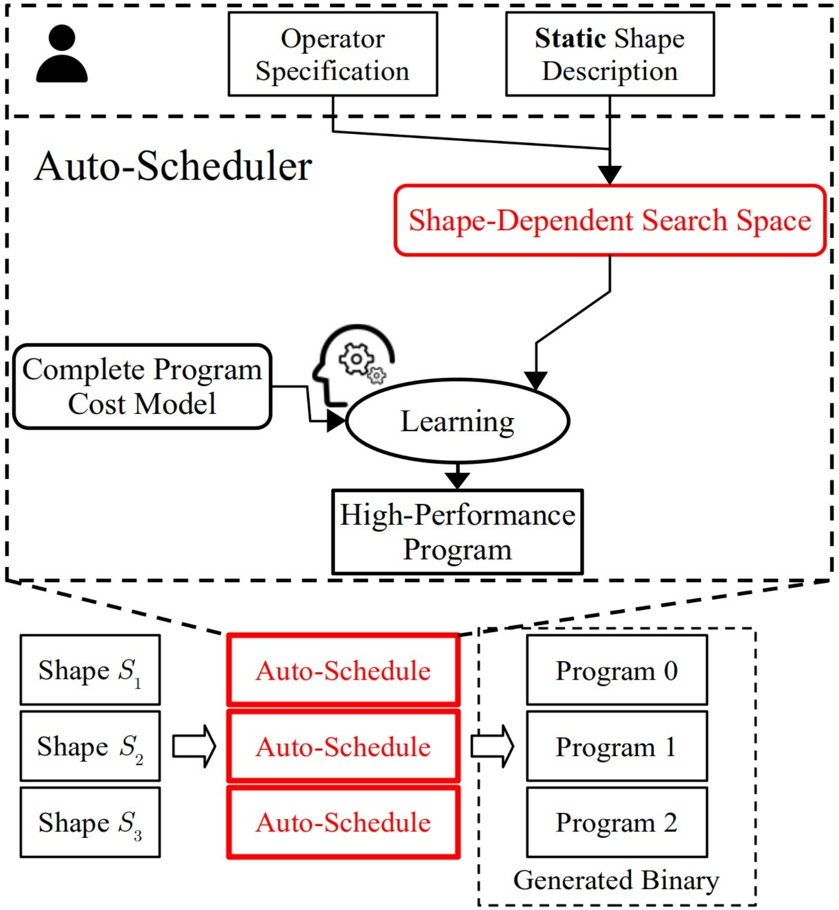
- Challenge #1:
  - Hard to share schedules across different shapes of the same operator.

Example

Schedule:

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

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



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

Challenges Faced by the Current Design

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

Example

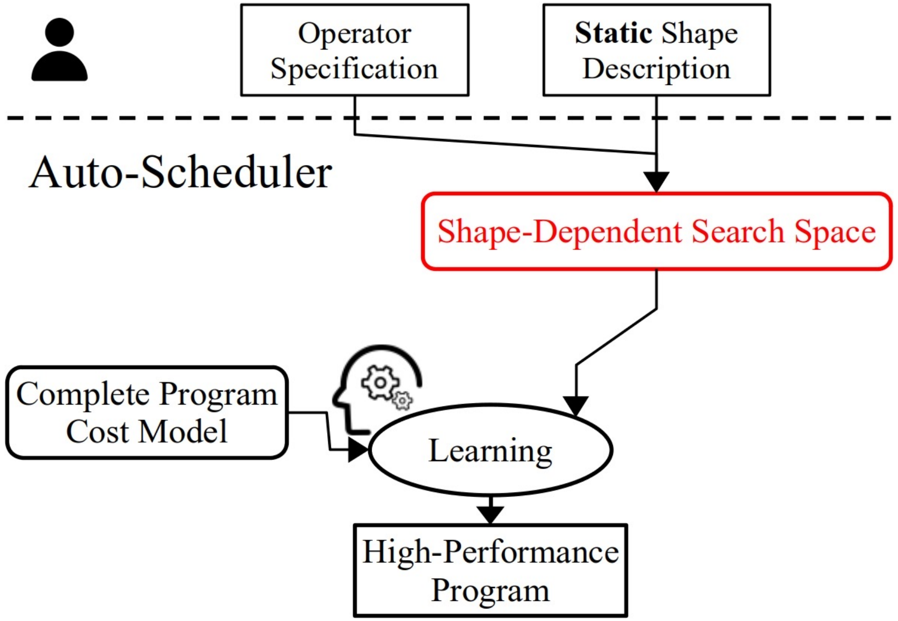
Schedule:

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

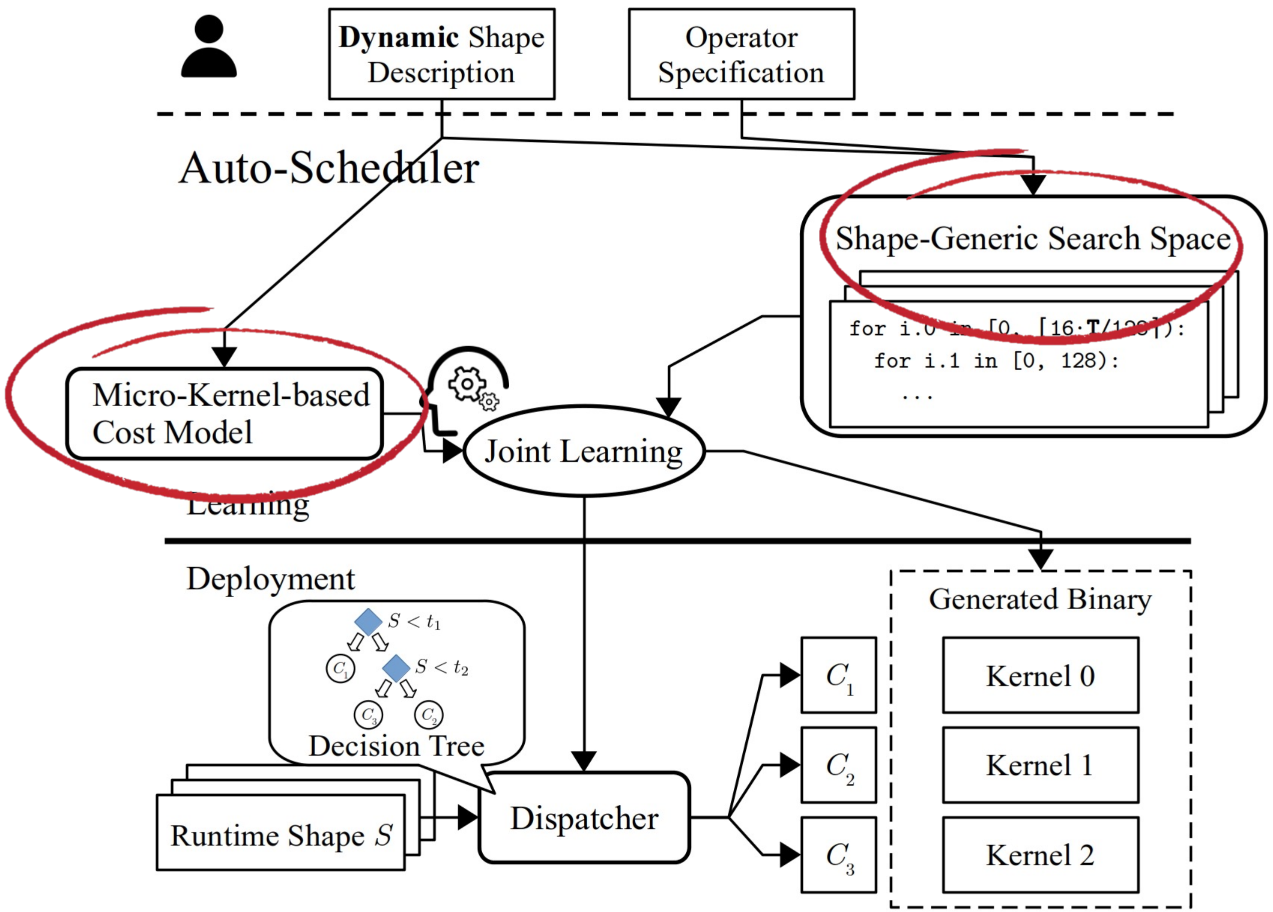
$\boxed{t \in \{7\}}$

$t = 2, 3, \ldots$ 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).


Operator Specification
Static Shape Description
Auto-Scheduler
Shape-Dependent Search Space
Complete Program Cost Model
Learning
High-Performance Program

DietCode: A New Auto-Scheduler Framework



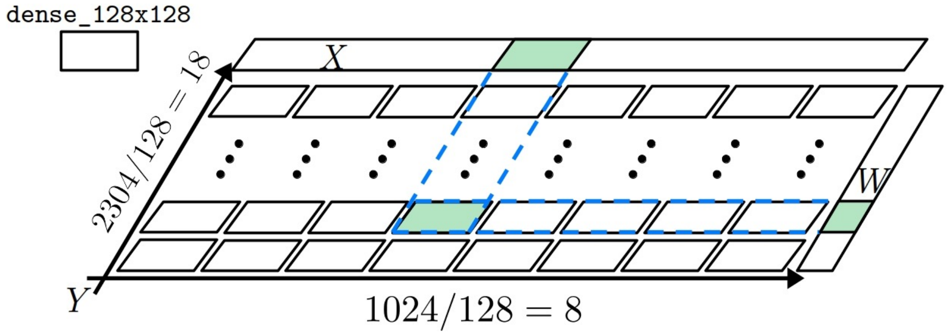
DietCode: Key Ideas

- 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:

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

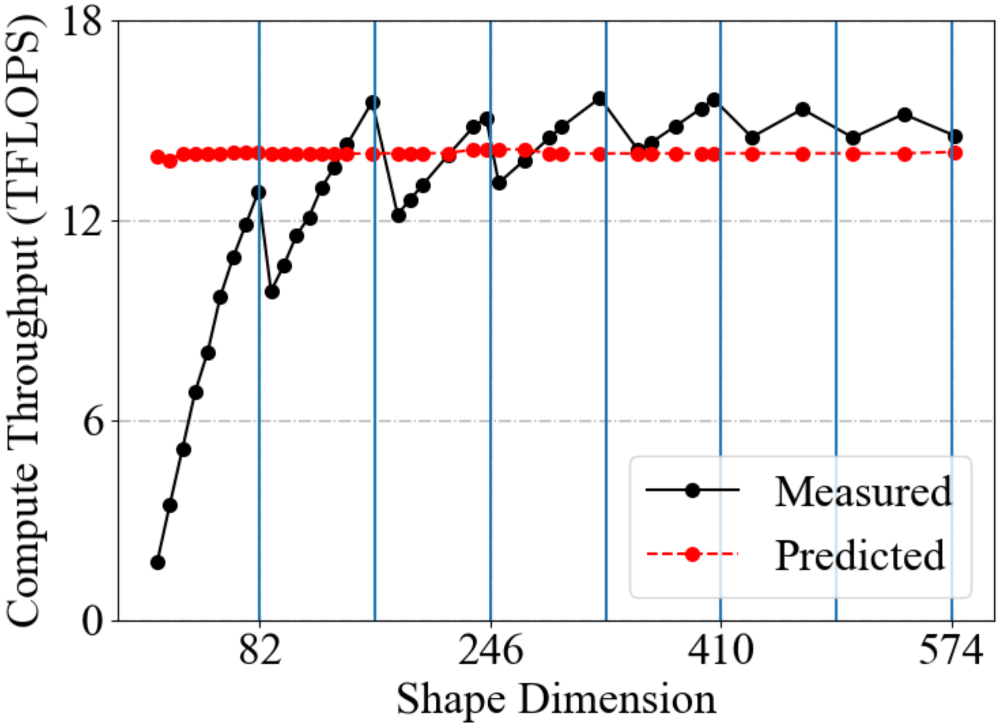
$Y = XW^T \;\; X\colon [128, 768], W\colon [128, 768]$



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:

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

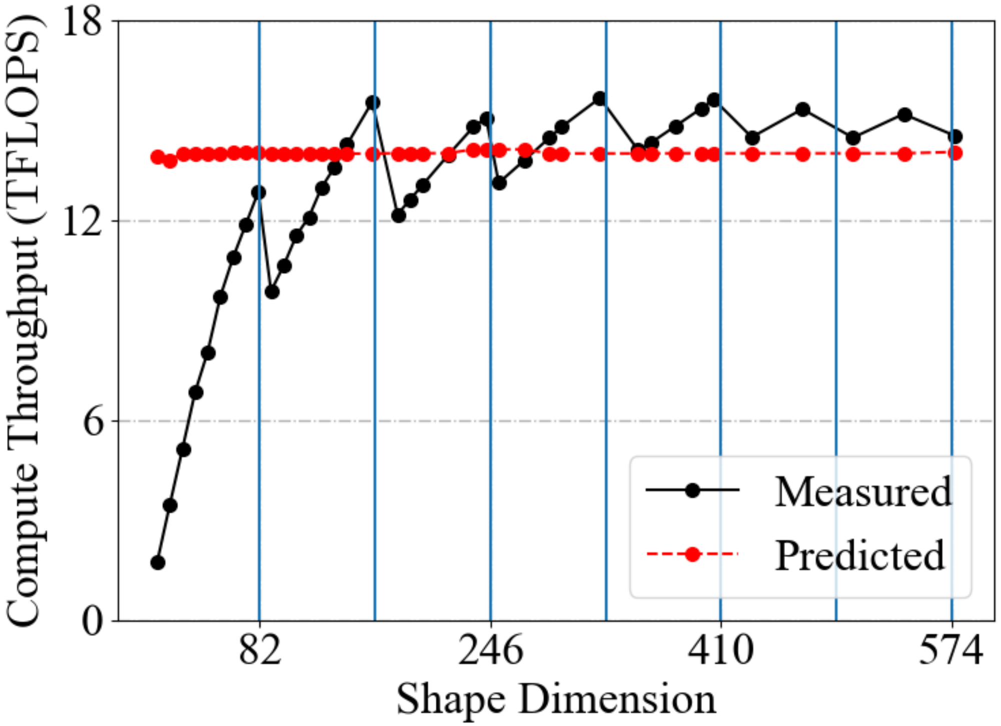


DietCode: Key Ideas

- 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:

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

    - Trainable Micro-Kernel Cost

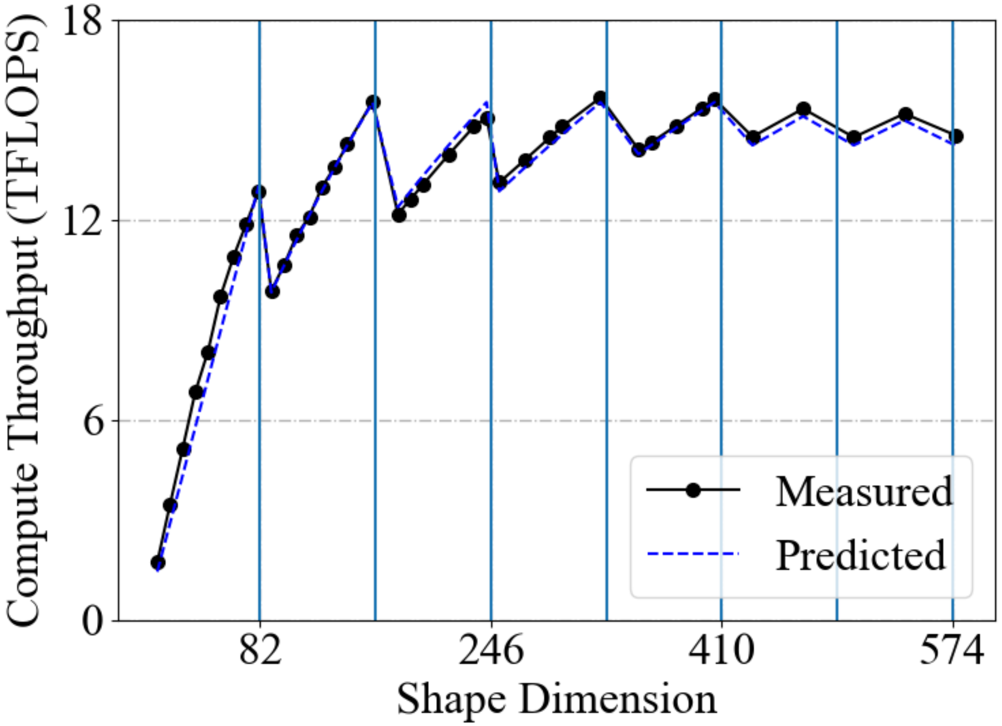


DietCode: Key Ideas

- 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)



DietCode: A New Interface

- Supports dynamic-shape workloads with its new interface.
  - E.g., $Y = XW^T$ X: $[16\times\mathbf{T}, 768]$, W: $[2304, 768]$, $T \in [1, 128]$

```
T, T_vals = tir.ShapeVar('T’), list(range(1, 128))

task = SearchTask(func=Dense, args=(16*T, 768, 2304),
                  shape_vars=(T,), wkl_insts=(T_vals,)
                  wkl_inst_weights=([1. for _ in T_vals],)
                  )
```

DietCode: A New Interface

- Supports dynamic-shape workloads with its new interface.
  - E.g., $Y = XW^T$ X: $[16\times\mathbf{T}, 768], W: [2304, 768], T \in [1, 128]$

```
T, T_vals = tir.ShapeVar('T’), list(range(1, 128))

task = SearchTask(func=Dense, args=(16*T, 768, 2304),
                  shape_vars=(T,), wkl_insts=(T_vals,)
                  wkl_inst_weights=([1. for _ in T_vals],)
                  )
```

  - Define a dynamic shape variable T and its instances.

DietCode: A New Interface

- Supports dynamic-shape workloads with its new interface.
  - E.g., $Y = XW^T$ X: $[16\times\mathbf{T}, 768], W: [2304, 768], T \in [1, 128]$

```
T, T_vals = tir.ShapeVar('T'), list(range(1, 128))

task = SearchTask(func=Dense, args=(16*T, 768, 2304),
                  shape_vars=(T,), wkl_insts=(T_vals,)
                  wkl_inst_weights=([1. for _ in T_vals],)
                  )
```

  - Define a dynamic shape variable T and its instances.
  - Pass the variable and its instances to the workload function.

DietCode: A New Interface

- Supports dynamic-shape workloads with its new interface.
  - E.g., $Y = XW^T$ X: $[16\times\mathbf{T}, 768]$, W: $[2304, 768]$, $T \in [1, 128]$

```
T, T_vals = tir.ShapeVar('T’), list(range(1, 128))

task = SearchTask(func=Dense, args=(16*T, 768, 2304),
                  shape_vars=(T,), wkl_insts=(T_vals,)
                  wkl_inst_weights=([1. for _ in T_vals],)
                  )
```

  - Define a dynamic shape variable T and its instances.
  - Pass the variable and its instances to the workload function.
  - [Optional] Assign weight to each shape instance.

Evaluation

Hardware: NVIDIA Tesla T4 GPU

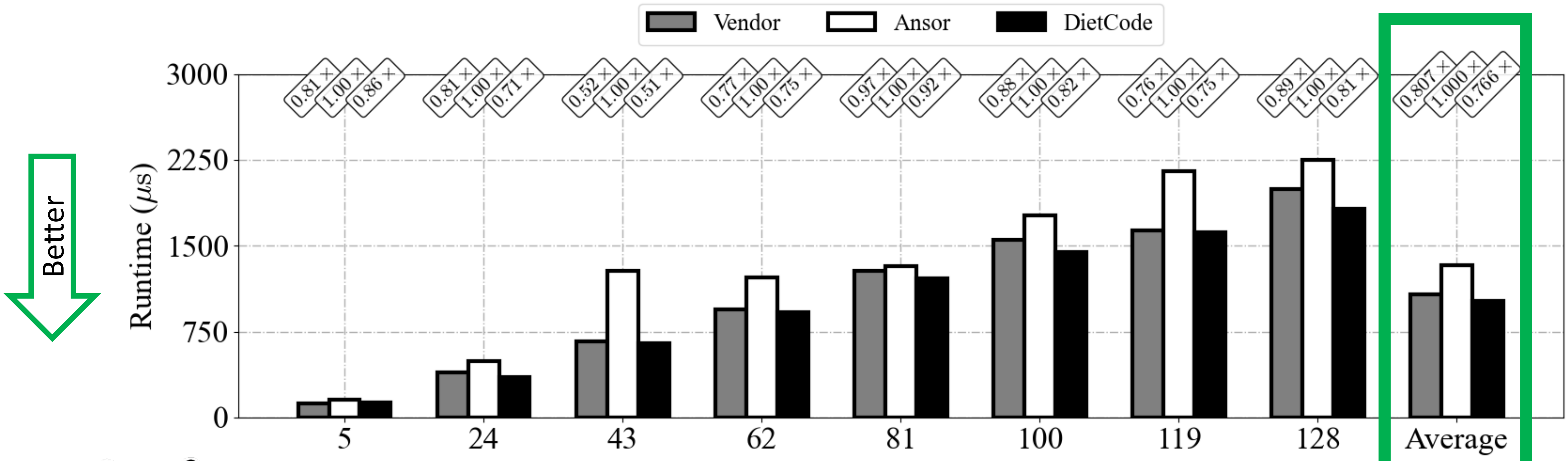
Software: TVM + CUDA + cuDNN

v0.8.dev0

NVIDIA CUDA v11.3

cuDNN v8.3

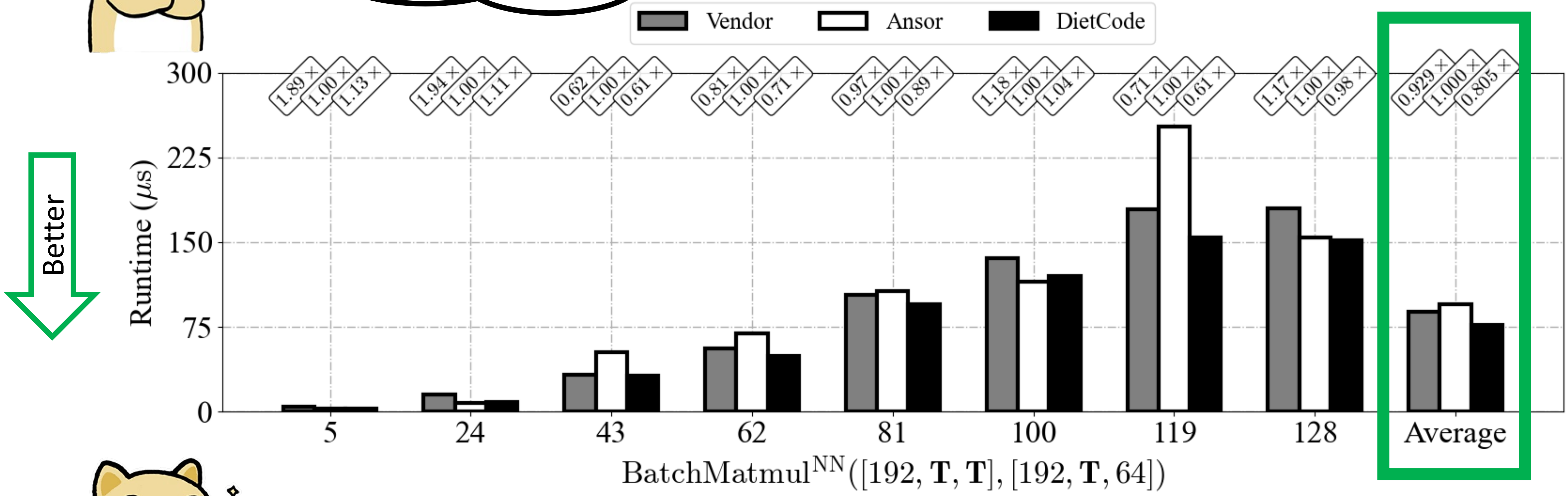
Evaluation

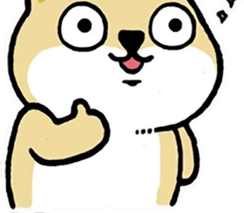


Performance: 30.5% better than Ansor; 5.3% better than Vendor
Auto-Scheduling Time: 5.6× less than Ansor

Evaluation

What about multiple dynamic axes?





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.
  - **Better Performance**
    - Up to 30.5% speedup than Ansor, up to 15.4% than Vendor.
- Working on integrating into the TVM main branch …

DietCode: Automatic Code Generation for Dynamic Tensor Programs

Bojian Zheng*[1, 2, 3], Ziheng Jiang*[4], Cody Yu[2], Haichen Shen[2],
Josh Fromm[5], Yizhi Liu[2], Yida Wang[2],
Luis Ceze[5, 6], Tianqi Chen[5, 7], Gennady Pekhimenko[1, 2, 3]

* Equal Contribution

1

2 aws

3 VECTOR INSTITUTE

4 ByteDance

5 OctoML

6

7

Backup

Scratchpad

DietCode: Key Ideas

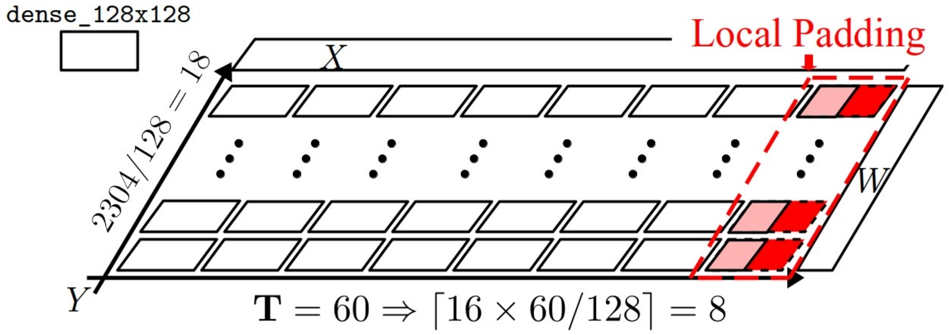
- Key Idea #1: **Shape-Generic** Search Space
  - Composed of *micro-kernels*. Each does a tile of the entire compute.
  - A micro-kernel can be ported to *all* shapes of the same operator.
  - Sampled from *hardware* constraints instead of shape factors (i.e., shape-generic).

Example:

$Y = XW^T$ X: $[16\times\mathbf{T}, 768], W: [2304, 768]$

with micro-kernel dense_128x128, which evaluates

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


dense_128x128
X
Local Padding
$2304/128 = 18$
W
Y
$\mathbf{T} = 60 \Rightarrow \lceil 16 \times 60/128 \rceil = 8$

Challenges Faced by the Current Design

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

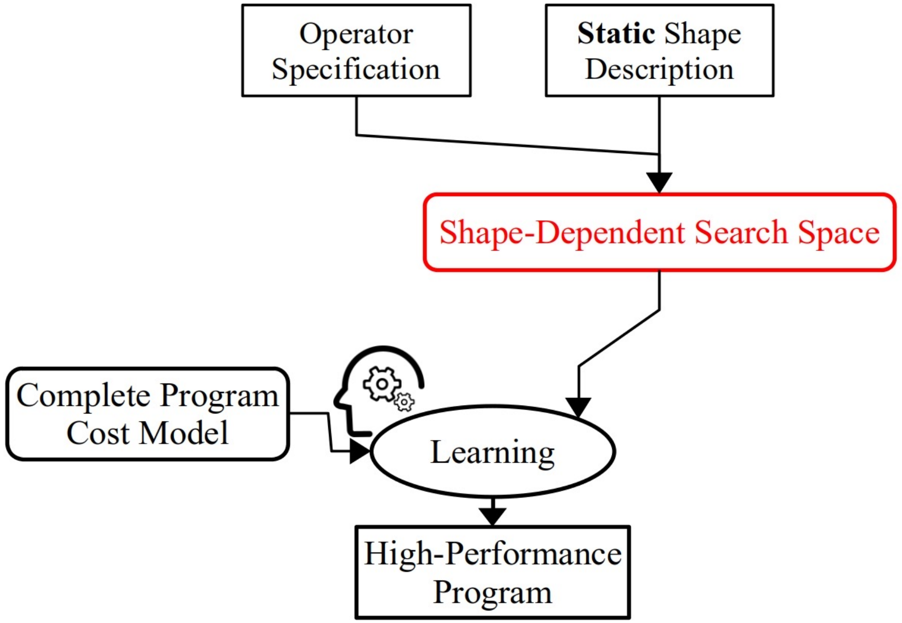
Example

Schedule (Loop Tiling):

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

$\boxed{t \in \{7\}}$

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


Operator Specification
Static Shape Description
Shape-Dependent Search Space
Complete Program Cost Model
Learning
High-Performance Program