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 Contributions

1 

2 

3 

4 

5 

6 

7 

Executive Summary

- Challenges posed by dynamic-shape workloads:
  - Vendor Libraries: Hard to be Engineered for Efficiency
  - Existing Auto-Schedulers: Long Compilation Time (**days** for a single operator)
- *DietCode* addresses the challenges with
  ① shape-generic search space
  ② micro-kernel-based cost model.
- Key Results:
  - Compilation Time: $\mathbf{5.88}\times$ saving vs. Ansor.
  - Performance: Up to $\mathbf{1.70}\times$ better vs. Ansor and $\mathbf{1.19}\times$ vs. the vendor library on modern GPUs.

Background: ML Framework Stack

Application

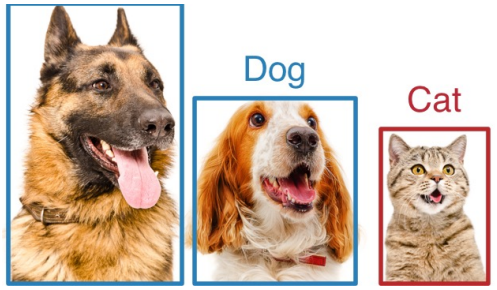
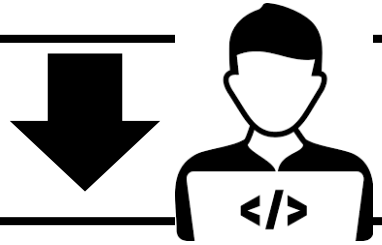


Image Classification[1]

Machine Translation[2]

Speech Recognition[3]



Framework

TensorFlow [4]

PyTorch [5]

[6]

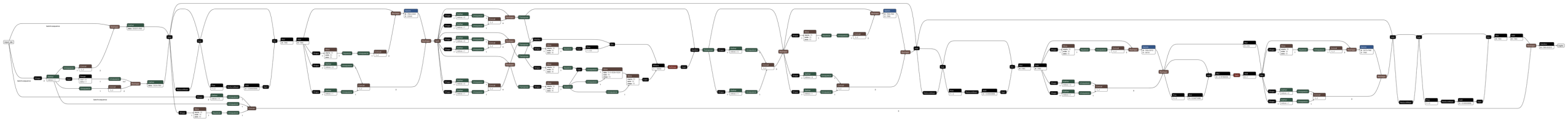
[1] J. Guo et al. *GluonCV and GluonNLP*. JMLR 2020
[2] https://translate.google.com/
[3] https://github.com/NVIDIA/NeMo
[4] M. Abadi et al. *TensorFlow*. OSDI 2016
[5] A. Paszke et al. *PyTorch*. NeurIPS 2019
[6] https://github.com/google/jax

Background: ML Framework Stack

Framework

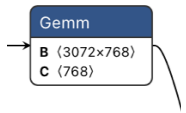
TensorFlow [4] PyTorch [5] [6]

Interpretation: Graph of Operators[7]



[4] M. Abadi et al. *TensorFlow*. OSDI 2016
[5] A. Paszke et al. *PyTorch*. NeurIPS 2019
[6] https://github.com/google/jax
[7] https://netron.app/

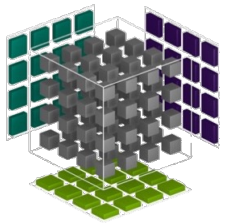
Background: Vendor Libraries



Operator
(Matrix Multiply)

Invoke

Vendor Libraries



cuBLAS[1]

[2]

Dispatcher

Built-in Kernel 2

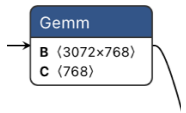
Built-in Kernel 1

Hardware

NVIDIA GPU[3]

[1] https://developer.nvidia.com/cublas
[2] https://developer.nvidia.com/cudnn
[3] https://www.nvidia.com/en-us/data-center/tesla-t4/

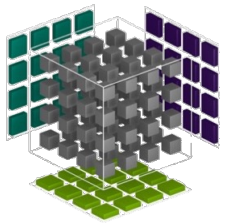
Background: Vendor Libraries



Operator
(Matrix Multiply)

Invoke

Vendor Libraries



cuBLAS[1]

[2]

●●●

●●●

Built-in Kernel 2

Built-in Kernel 1

Dispatcher

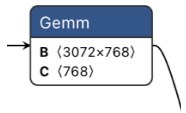
Hardware

NVIDIA GPU[3]

●●●

[1] https://developer.nvidia.com/cublas
[2] https://developer.nvidia.com/cudnn
[3] https://www.nvidia.com/en-us/data-center/tesla-t4/

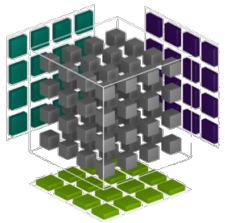
Background: Vendor Libraries



Operator
(Matrix Multiply)

Invoke

Vendor Libraries



cuBLAS[1]

[2]

Built-in Kernel 2

Built-in Kernel 1

Dispatcher

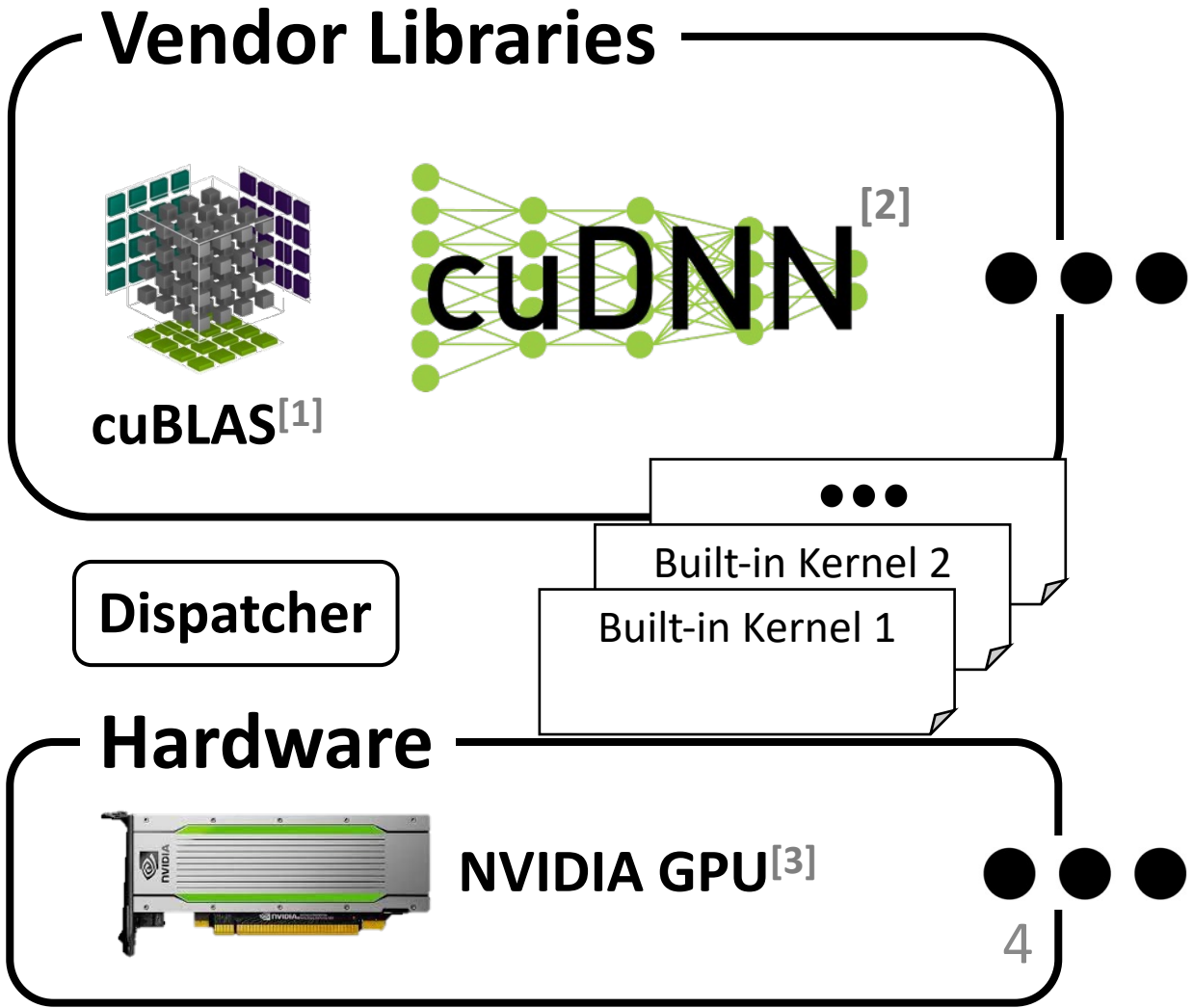
Hardware

NVIDIA GPU[3]

[1] https://developer.nvidia.com/cublas
[2] https://developer.nvidia.com/cudnn
[3] https://www.nvidia.com/en-us/data-center/tesla-t4/

Background: Vendor Libraries

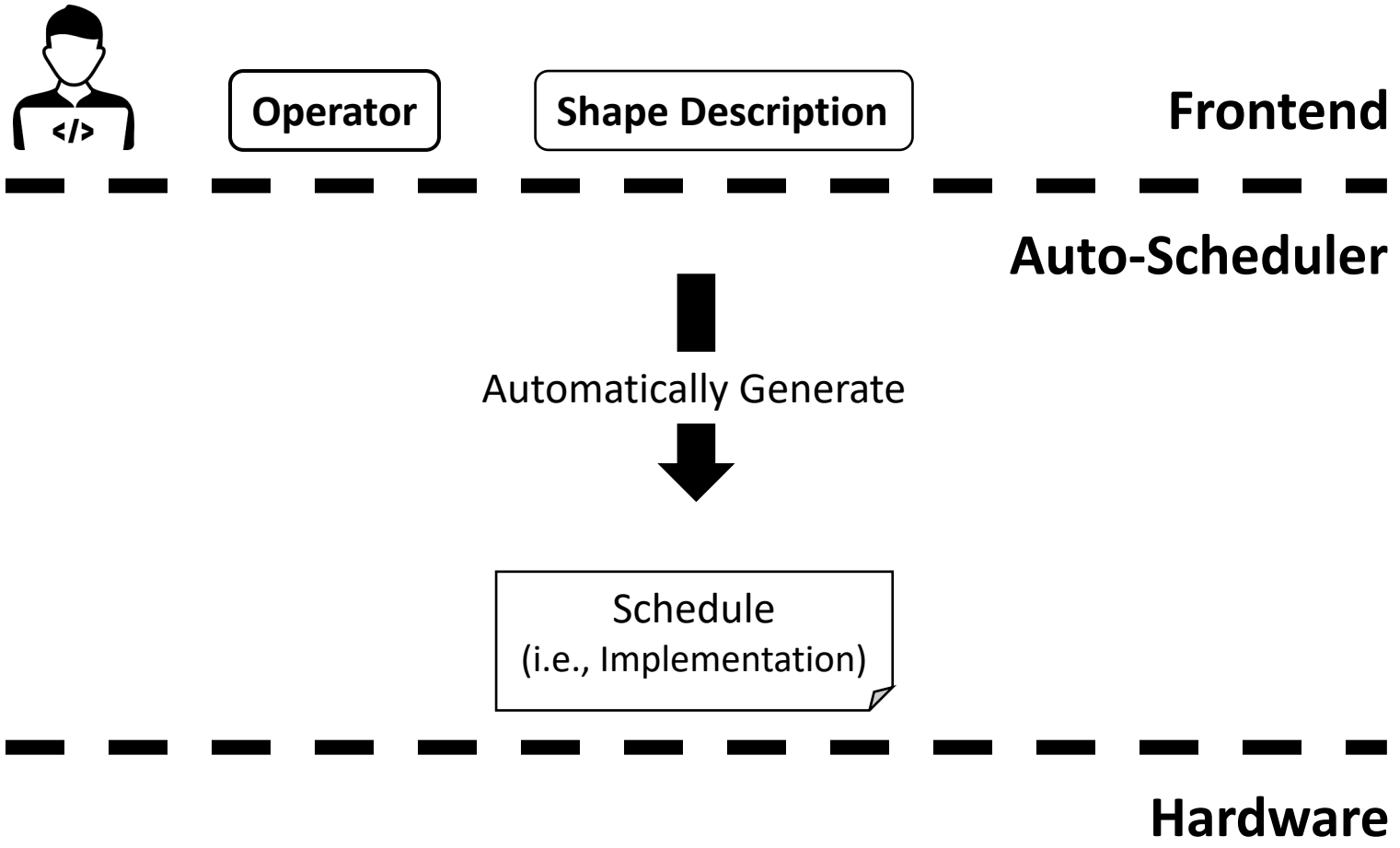
- Challenges
  - Performance of built-in kernels can be **suboptimal** on the given shapes or hardware[4, 5, 6, 7, 8, 9, …].
  - **Huge** engineering efforts and expertise required to tune for specific use cases.



[1] https://developer.nvidia.com/cublas
[2] https://developer.nvidia.com/cudnn
[3] https://www.nvidia.com/en-us/data-center/tesla-t4/
[4] T. Chen et al. *TVM*. OSDI 2018
[5] N. Vasilache et al. *Tensor Comprehensions*. TACO 2019
[6] L. Zheng et al. *Ansor*. OSDI 2020
[7] F. Yu et al. *Towards Latency-aware DNN Optimization with GPU Runtime Analysis and Tail Effect Elimination*. arXiv 2020
[8] S. Feng, B. Hou et al. *TensorIR*. arXiv 2022
[9] https://tvm.apache.org/2018/03/23/nmt-transformer-optimize

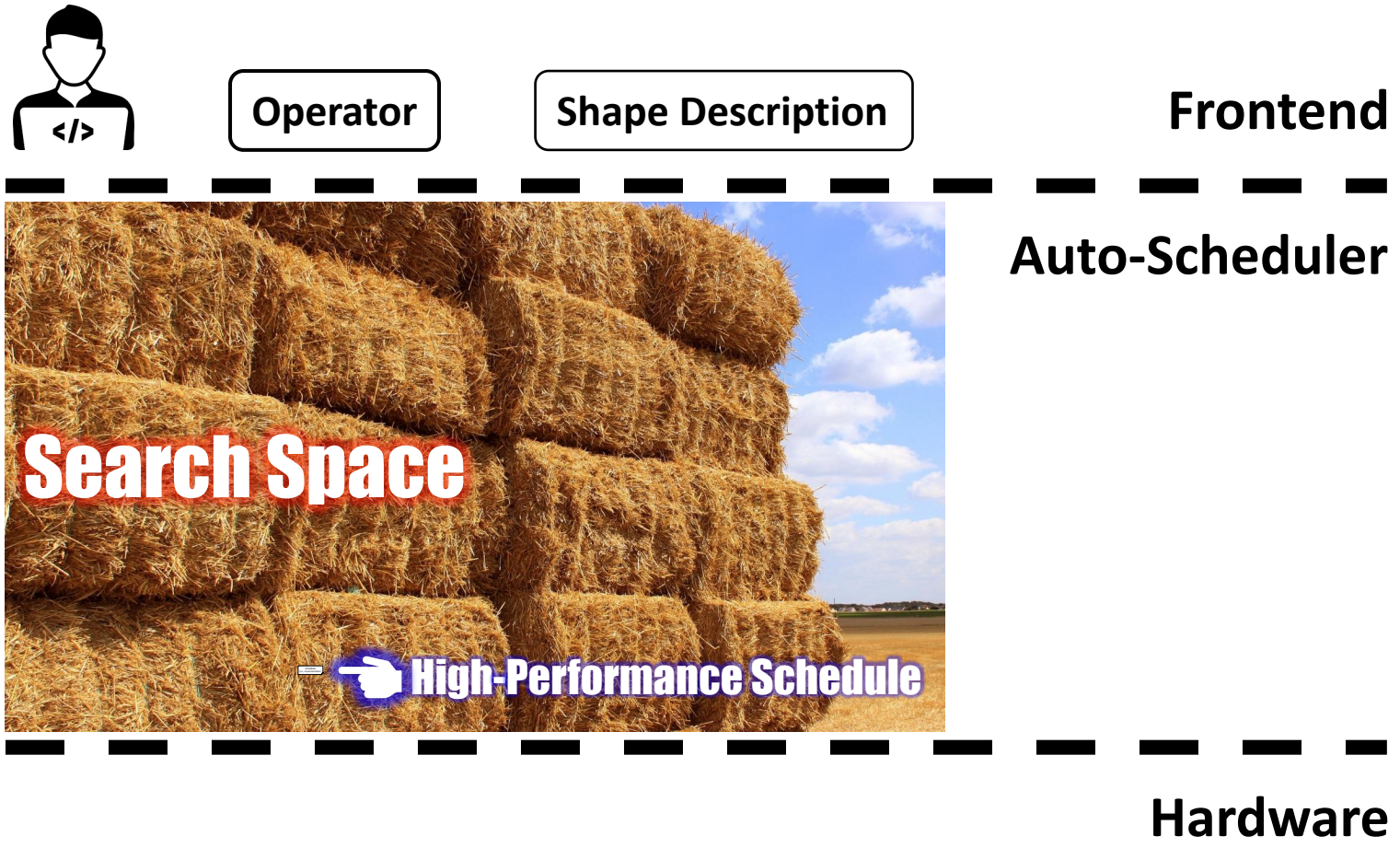
Background: Auto-Scheduler [1, 2, 3, 4]

- **Objective**:



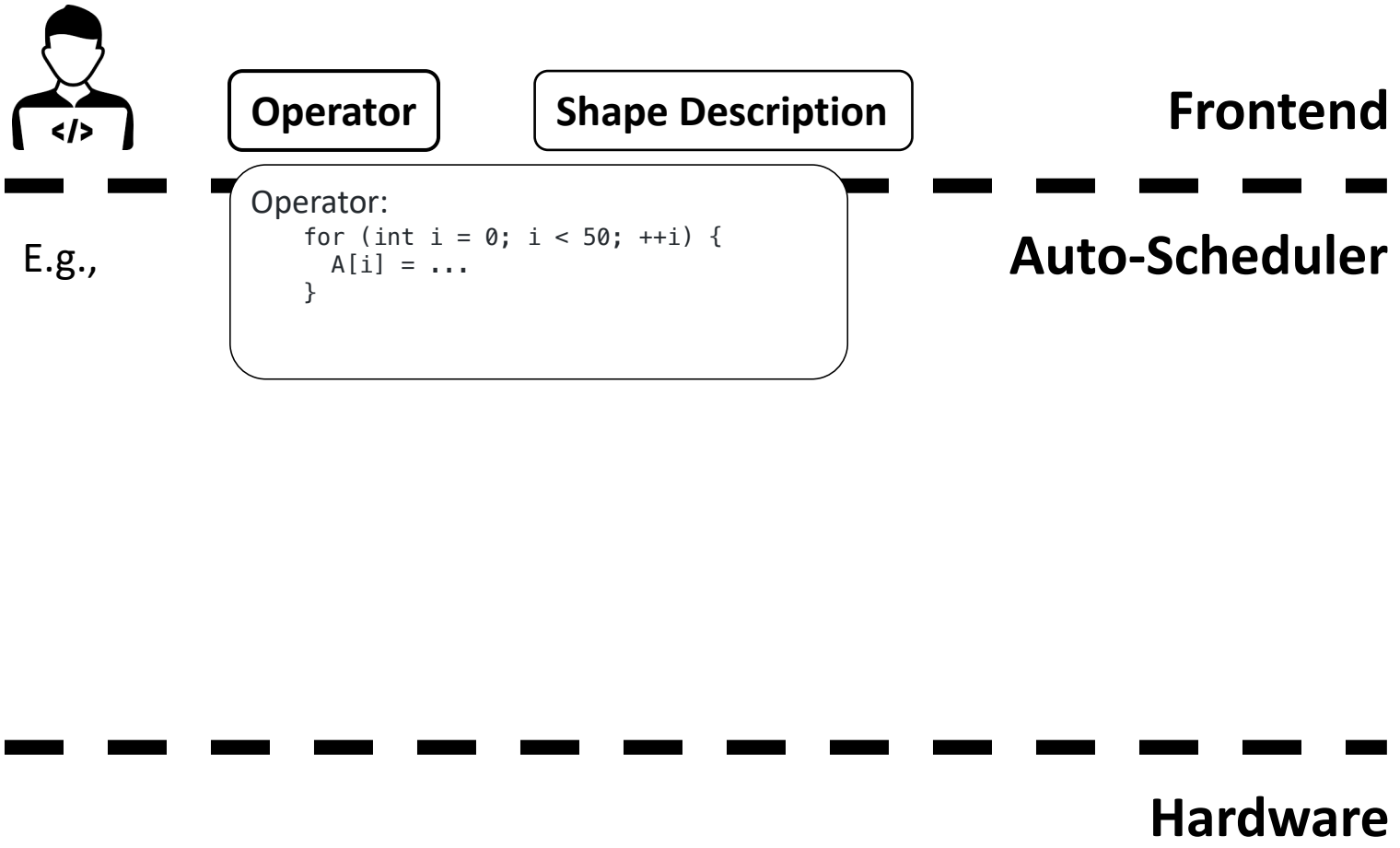
[1] A. Adams et al. *Halide Auto-Scheduler*. SIGGRAPH 2019
[2] N. Vasilache et al. *Tensor Comprehensions*. TACO 2019
[3] L. Zheng et al. *Ansor*. OSDI 2020
[4] S. Feng, B. Hou et al. *TensorIR*. arXiv 2022

Background: Auto-Scheduler [1, 2, 3, 4]



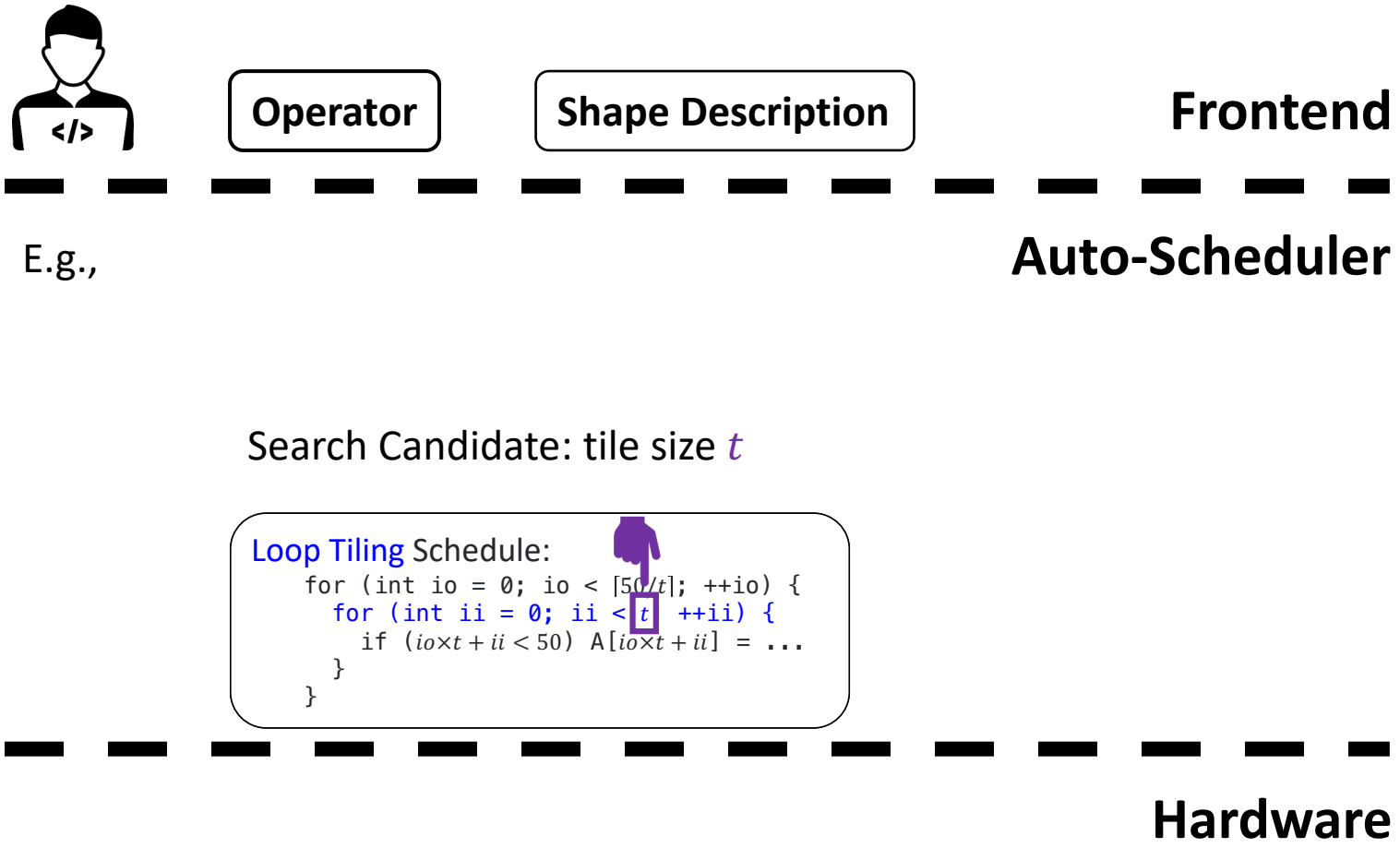
[1] A. Adams et al. *Halide Auto-Scheduler*. SIGGRAPH 2019
[2] N. Vasilache et al. *Tensor Comprehensions*. TACO 2019
[3] L. Zheng et al. *Ansor*. OSDI 2020
[4] S. Feng, B. Hou et al. *TensorIR*. arXiv 2022

Background: Auto-Scheduler [1, 2, 3, 4]



[1] A. Adams et al. *Halide Auto-Scheduler*. SIGGRAPH 2019
[2] N. Vasilache et al. *Tensor Comprehensions*. TACO 2019
[3] L. Zheng et al. *Ansor*. OSDI 2020
[4] S. Feng, B. Hou et al. *TensorIR*. arXiv 2022

Background: Auto-Scheduler [1, 2, 3, 4]



[1] A. Adams et al. *Halide Auto-Scheduler*. SIGGRAPH 2019
[2] N. Vasilache et al. *Tensor Comprehensions*. TACO 2019
[3] L. Zheng et al. *Ansor*. OSDI 2020
[4] S. Feng, B. Hou et al. *TensorIR*. arXiv 2022

Background: Auto-Scheduler [1, 2, 3, 4]

Operator **Shape Description** **Frontend**

E.g., **Auto-Scheduler**

Search Candidate: tile size $t \in [2, \infty)$

Search Techniques:
1. Shape-Dependent Search Space
2. Complete Program Cost Model

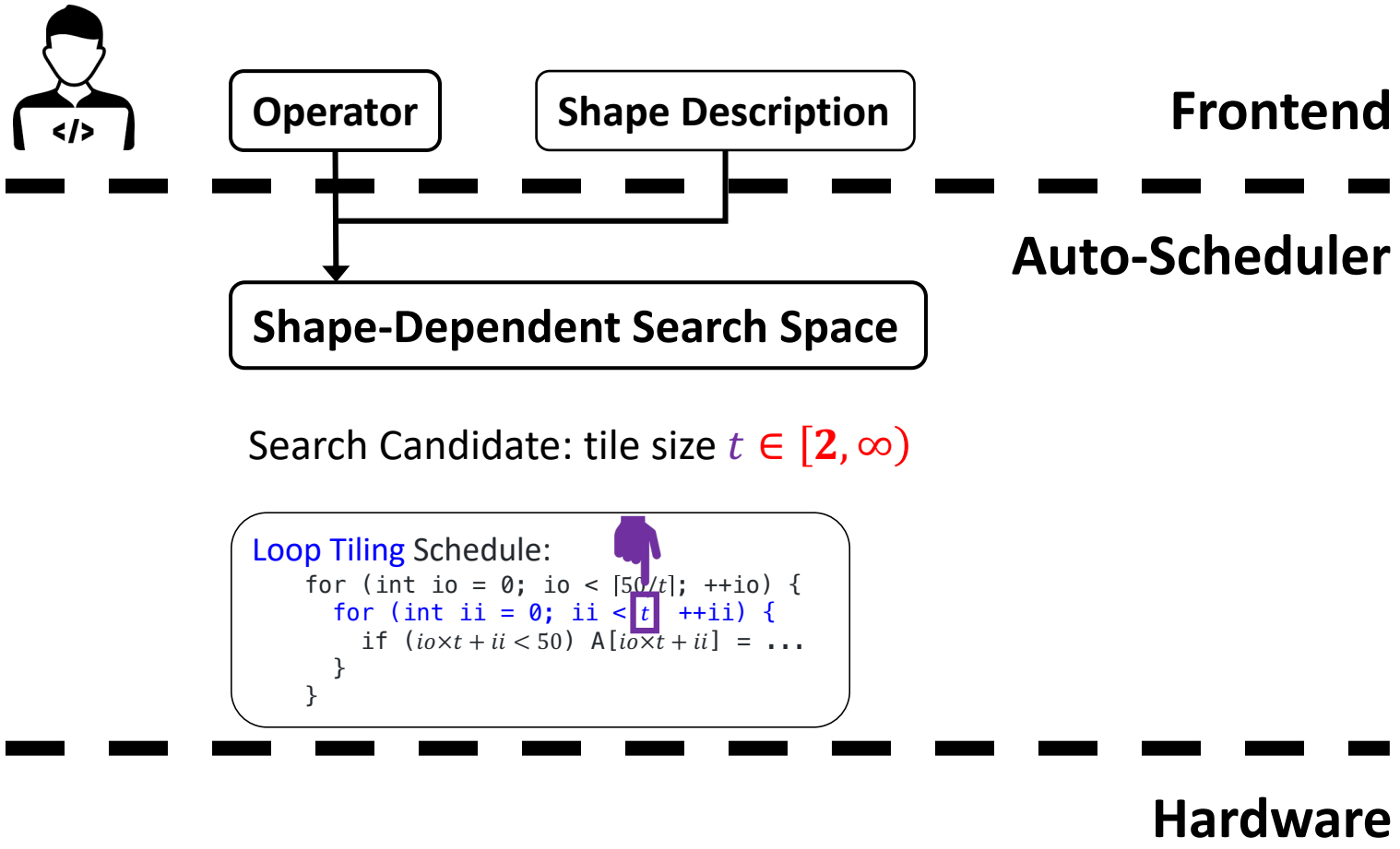
Loop Tiling Schedule:

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

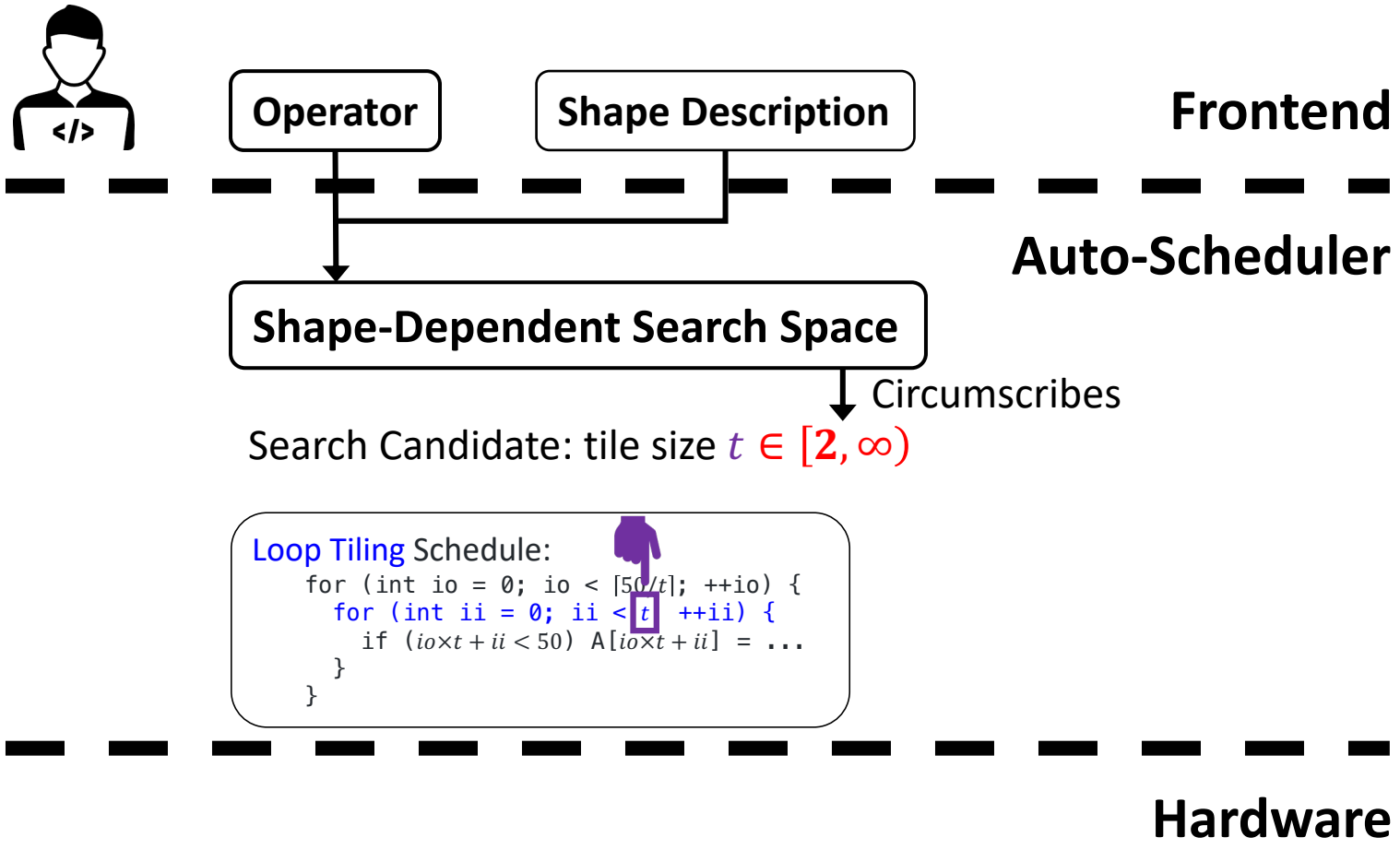
Hardware

[1] A. Adams et al. *Halide Auto-Scheduler*. SIGGRAPH 2019
[2] N. Vasilache et al. *Tensor Comprehensions*. TACO 2019
[3] L. Zheng et al. *Ansor*. OSDI 2020
[4] S. Feng, B. Hou et al. *TensorIR*. arXiv 2022

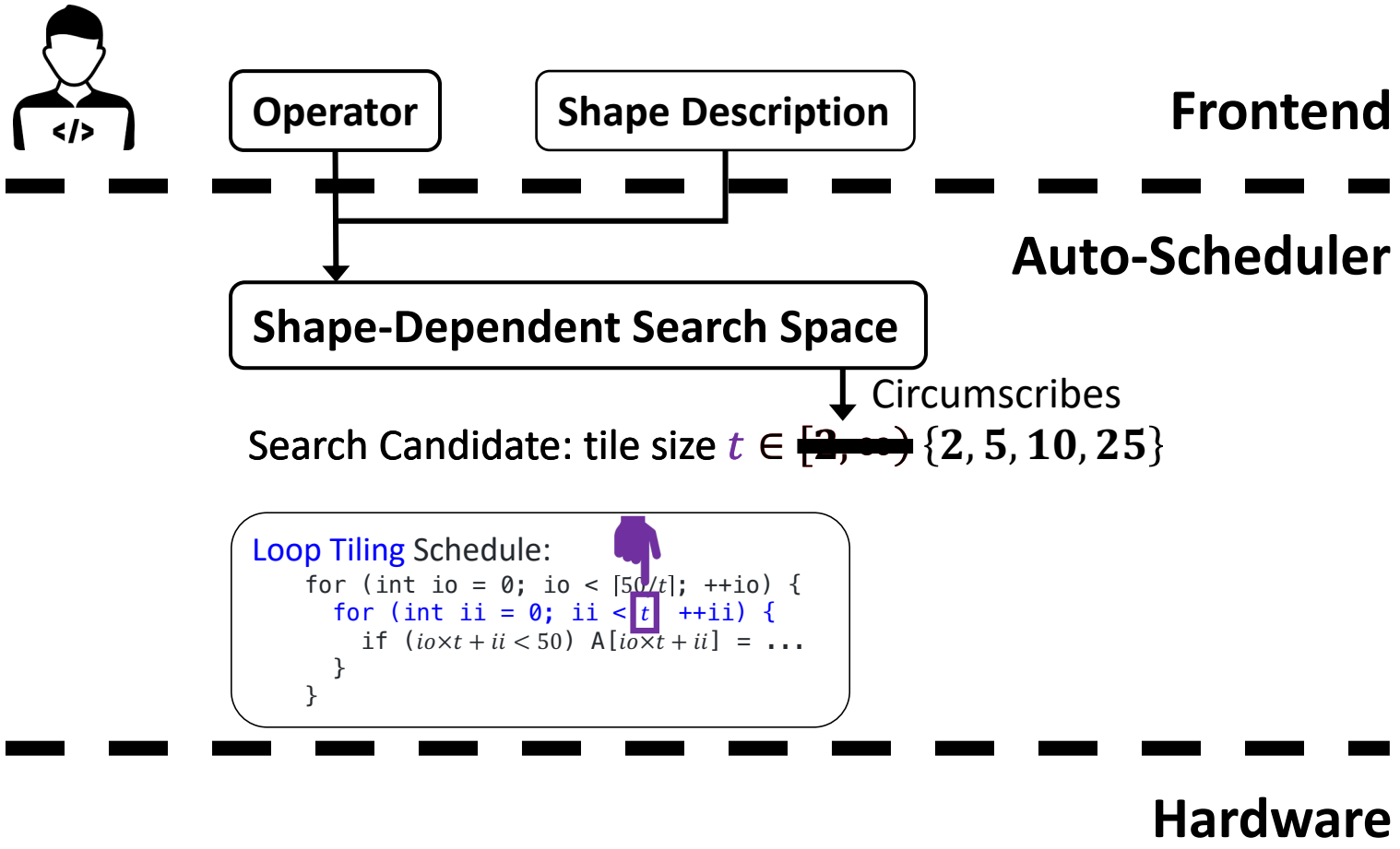
1. Shape-Dependent Search Space



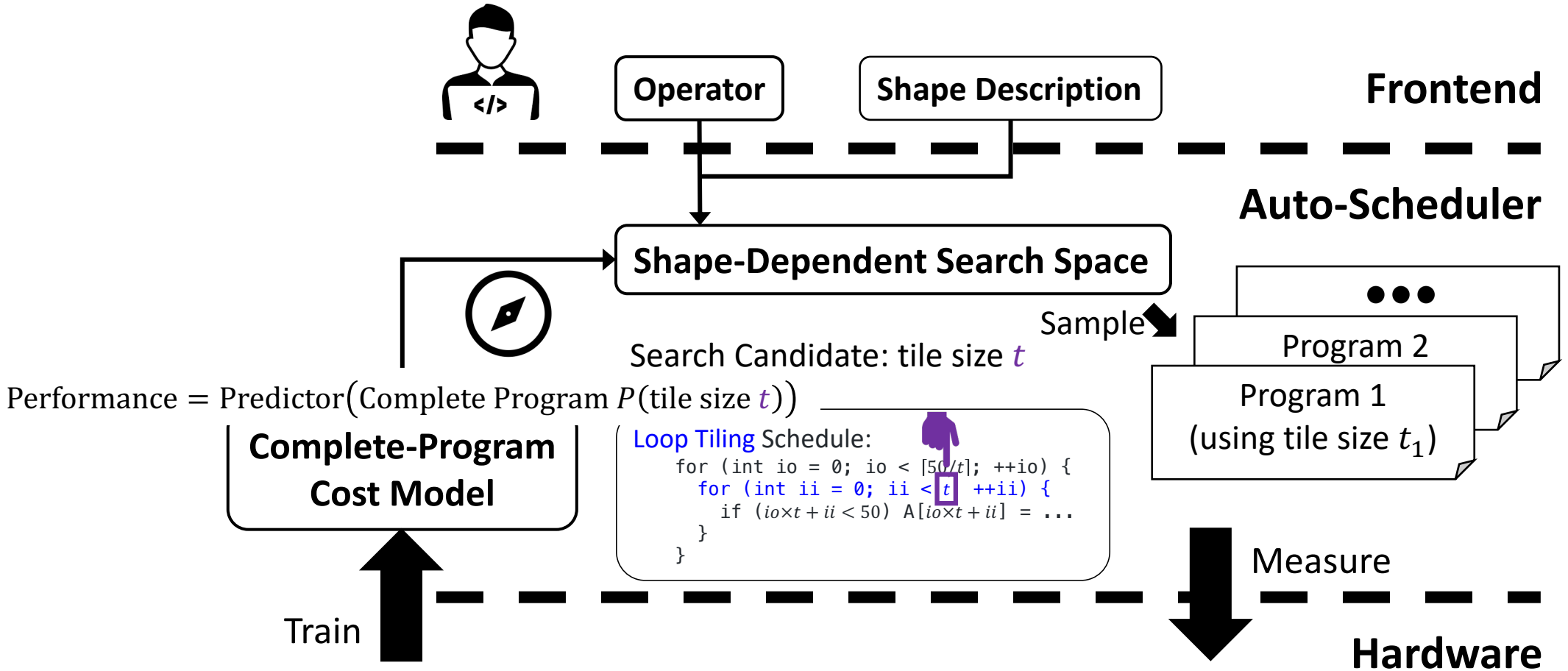
1. Shape-Dependent Search Space



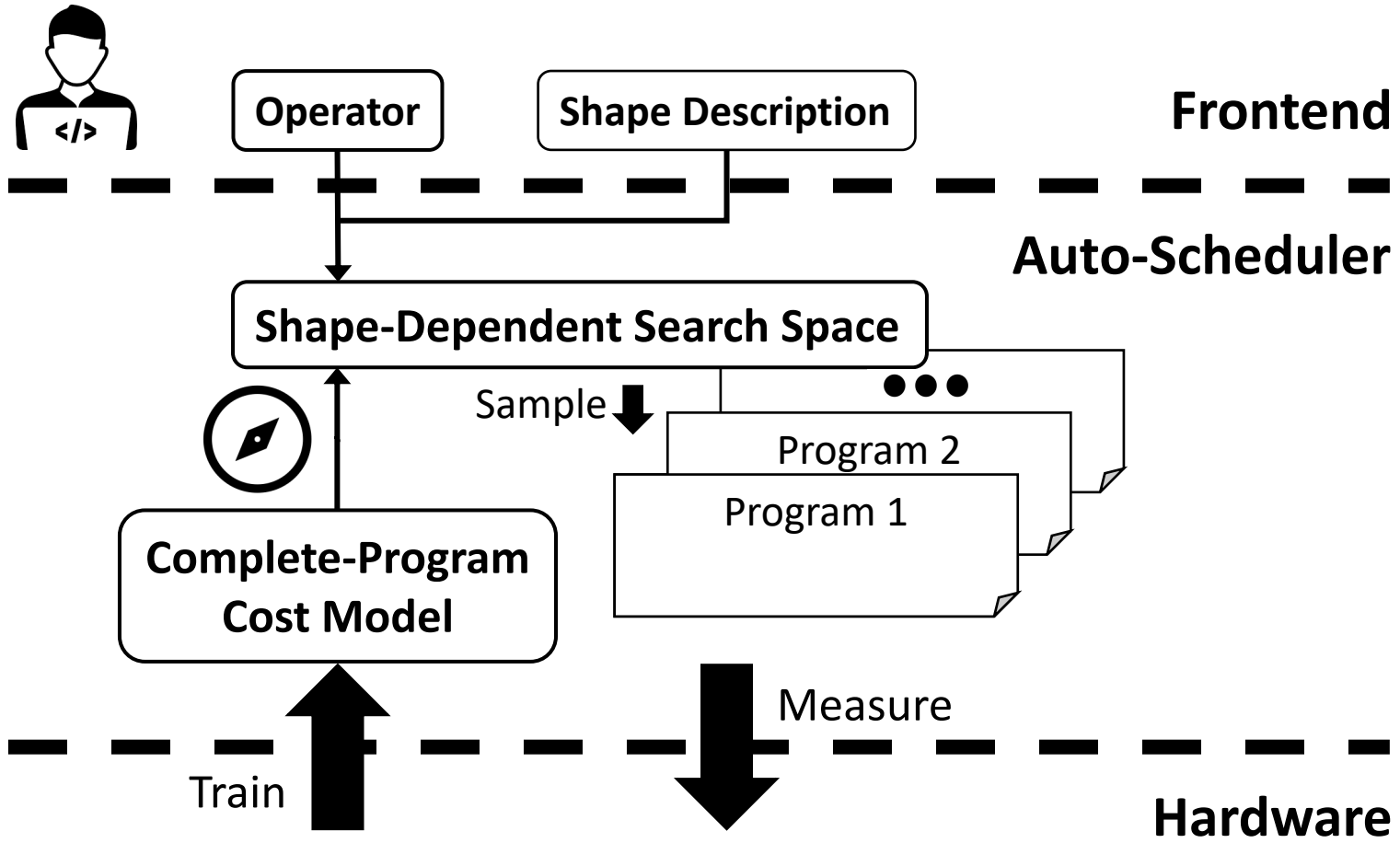
1. Shape-Dependent Search Space



2. Complete Program Cost Model

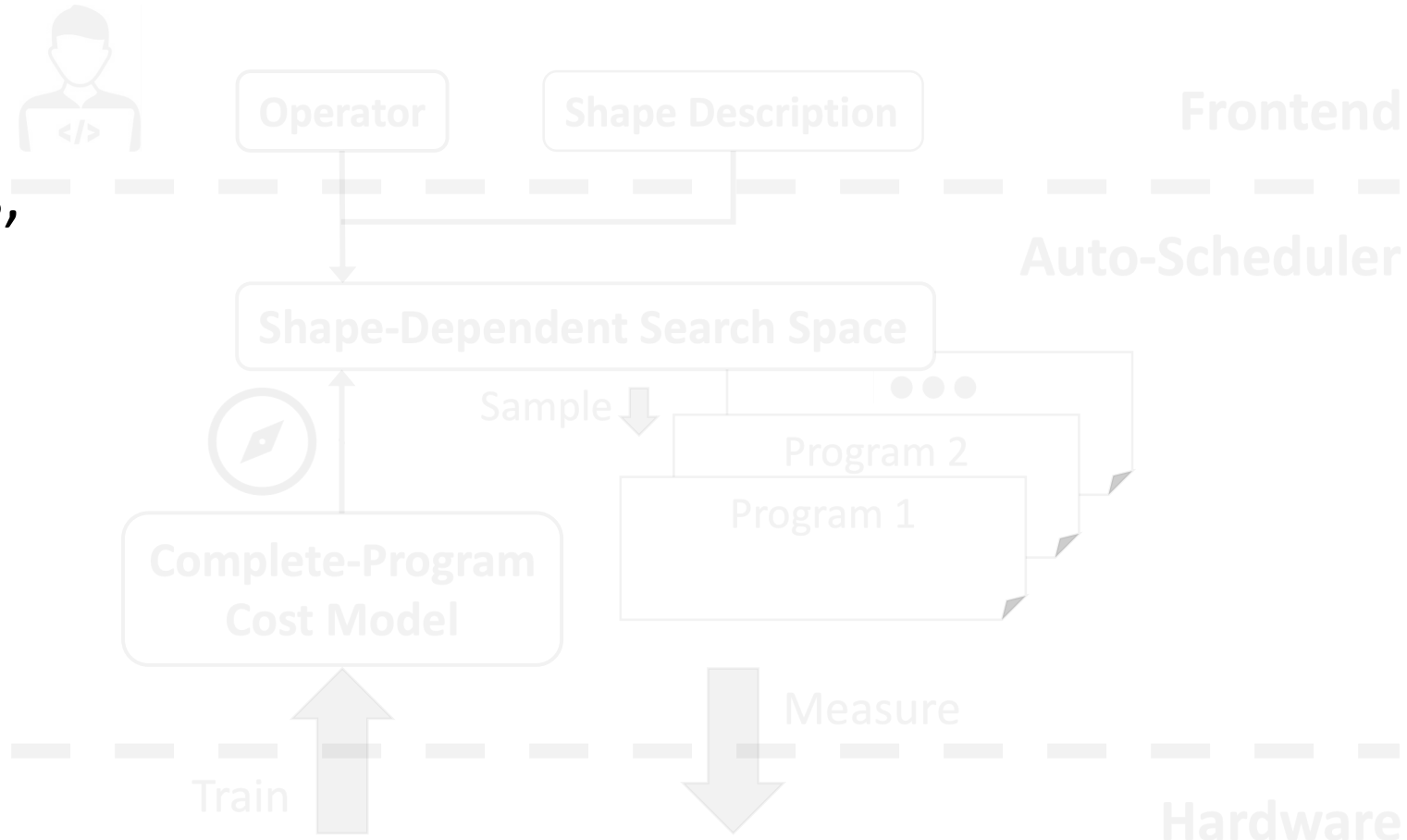


Challenges from Dynamic-Shape Workloads



Challenges from Dynamic-Shape Workloads

- **Cannot** efficiently handle **dynamic-shape** operators, common in

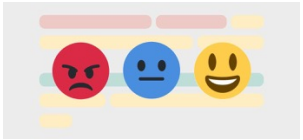


Challenges from Dynamic-Shape Workloads

- **Cannot** efficiently handle **dynamic-shape** operators, common in

Translation[1]

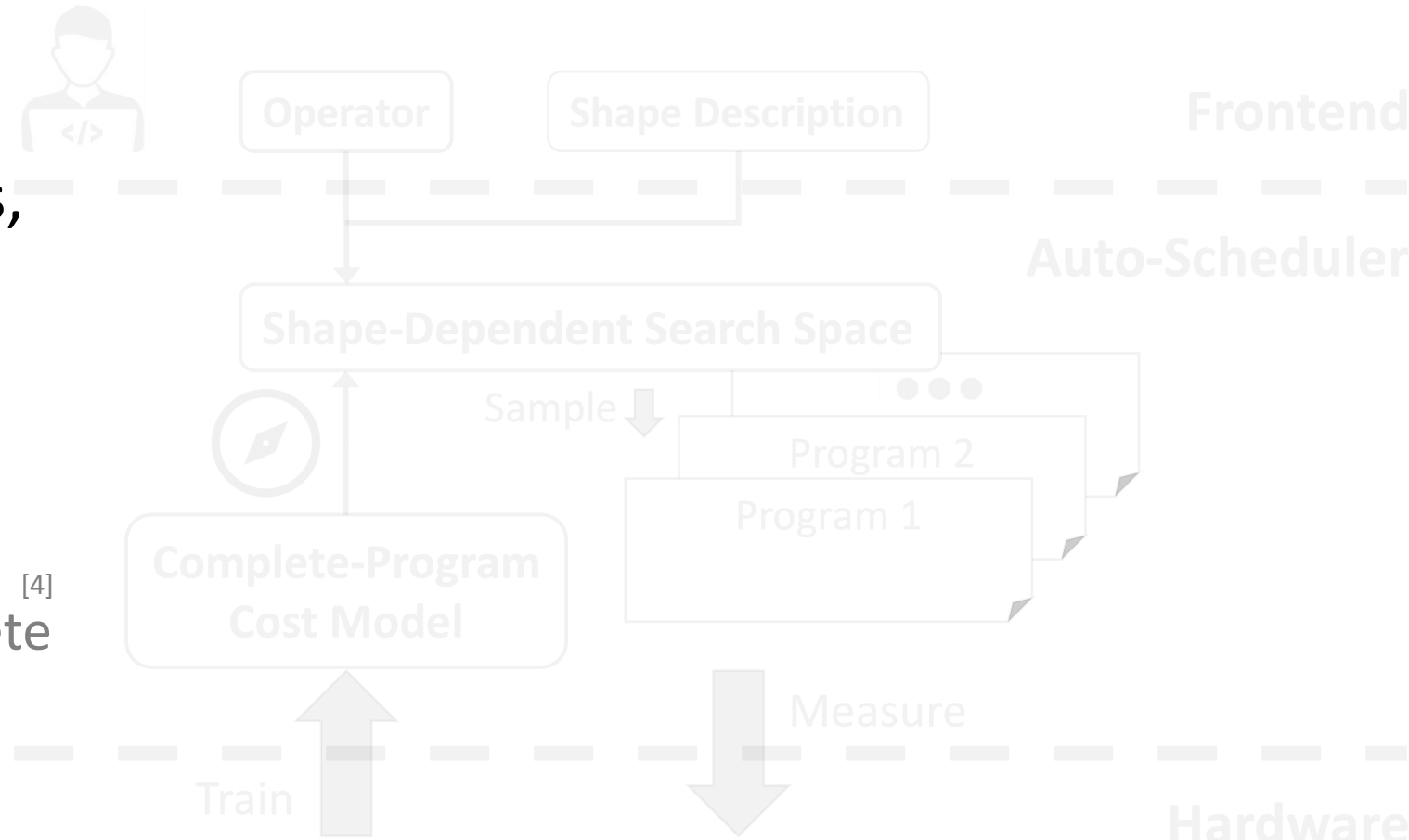
Speech Recognition[2]



Sentiment Analysis[3]

Text Auto-Complete[4]

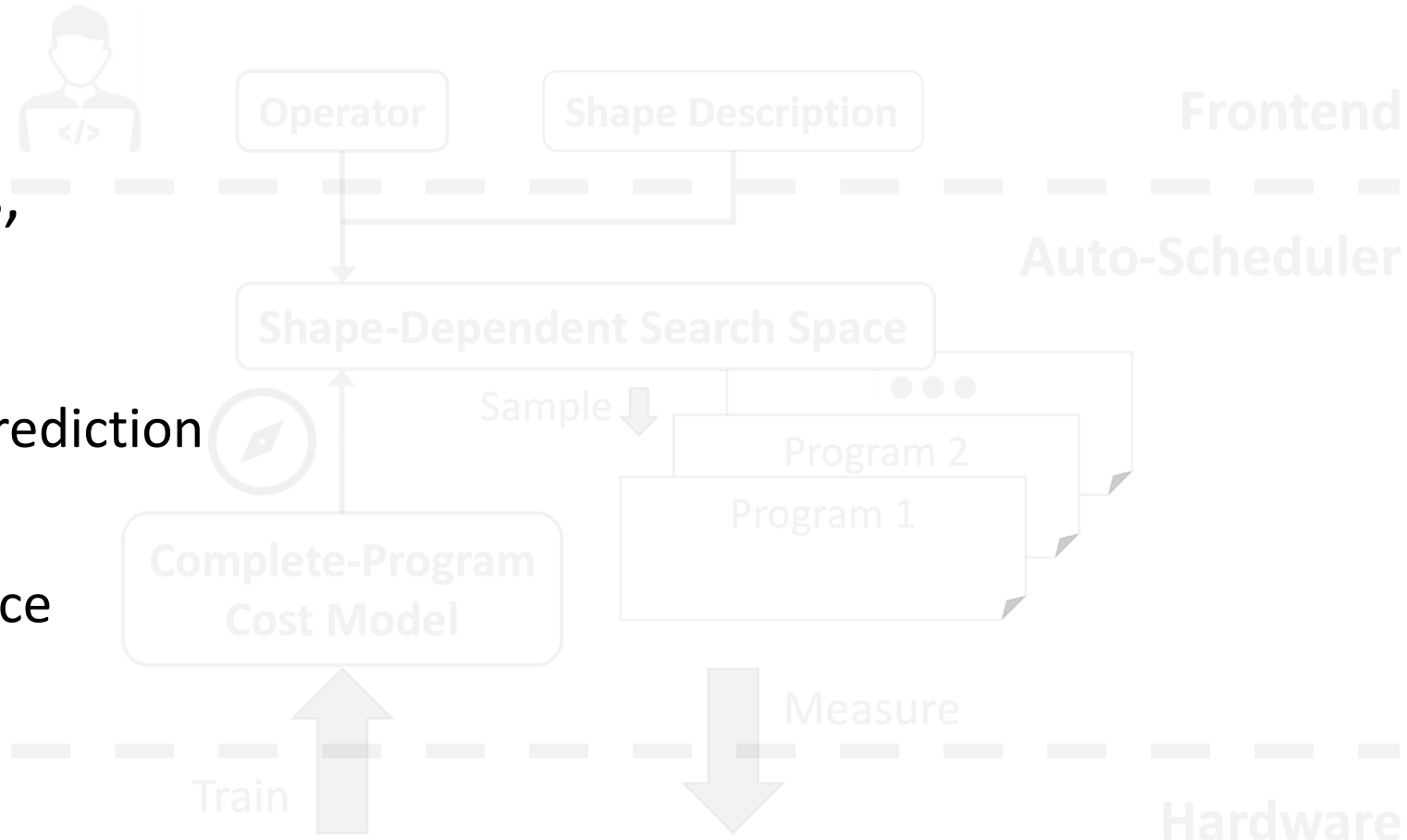
whose input sentences/audios have dynamic lengths.



[1] https://translate.google.com/
[2] https://github.com/NVIDIA/NeMo
[3] J. Devlin et al. *BERT*. NAACL-HTL 2019
[4] A. Radford et al. *GPT-2*. 2019

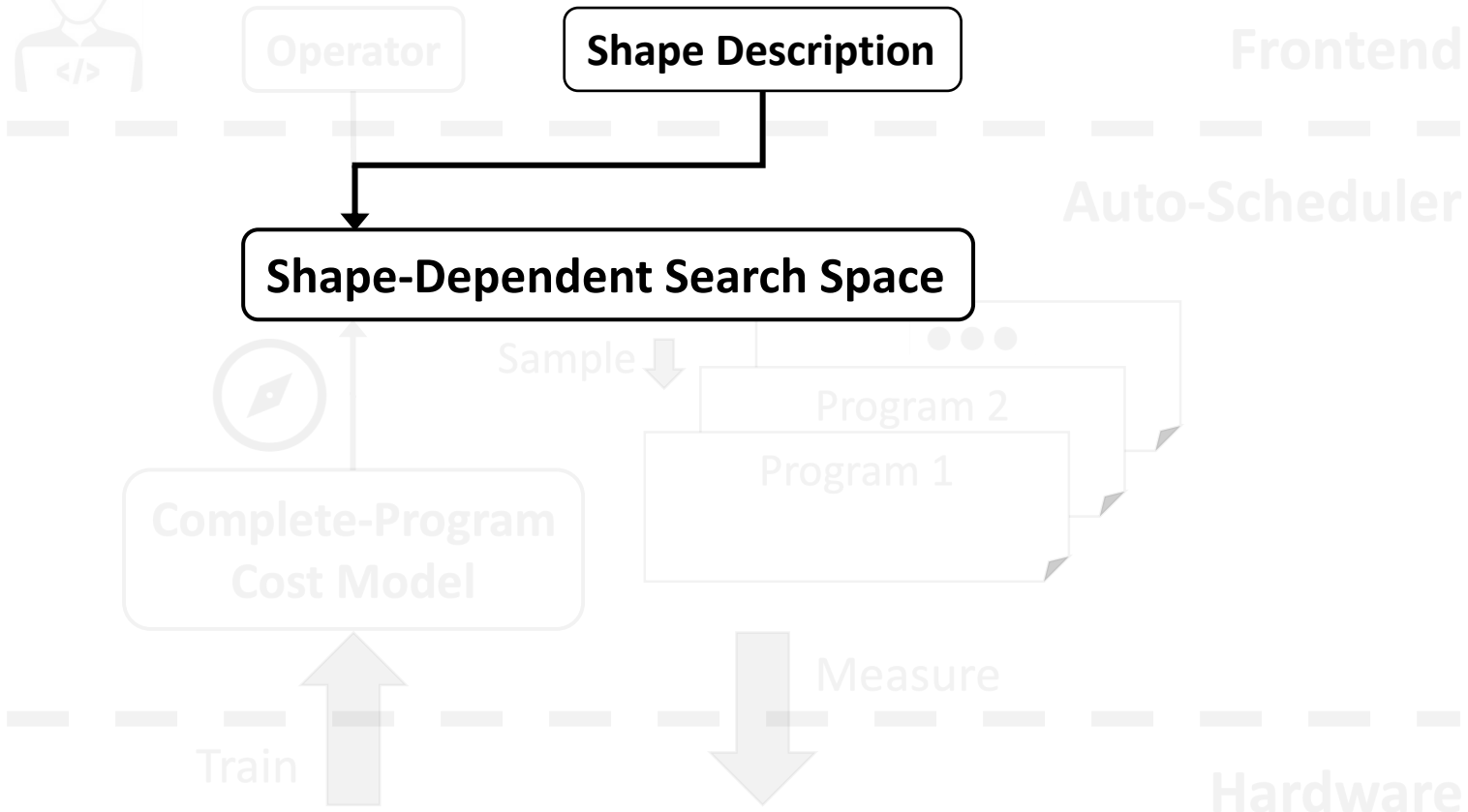
Challenges from Dynamic-Shape Workloads

- **Cannot** efficiently handle **dynamic-shape** operators, due to
  - Humongous Search Space
  - Inaccurate Performance Prediction
- *DietCode*'s Key Ideas:
  - Shape-Generic Search Space
  - Micro-Kernel-based Cost Model



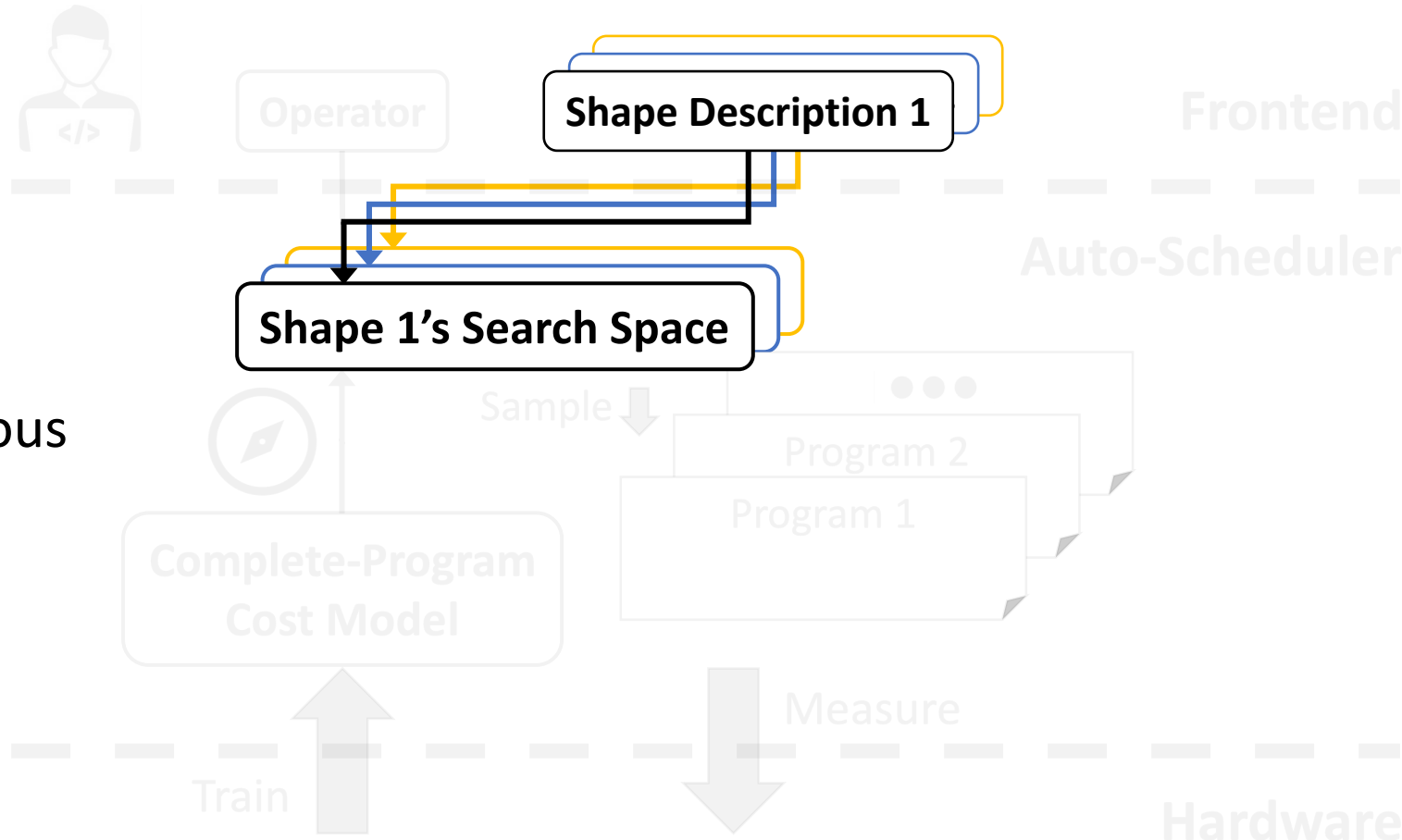
Challenge #1. Humongous Search Space

- **Hard** to share search space between operators of different shapes.



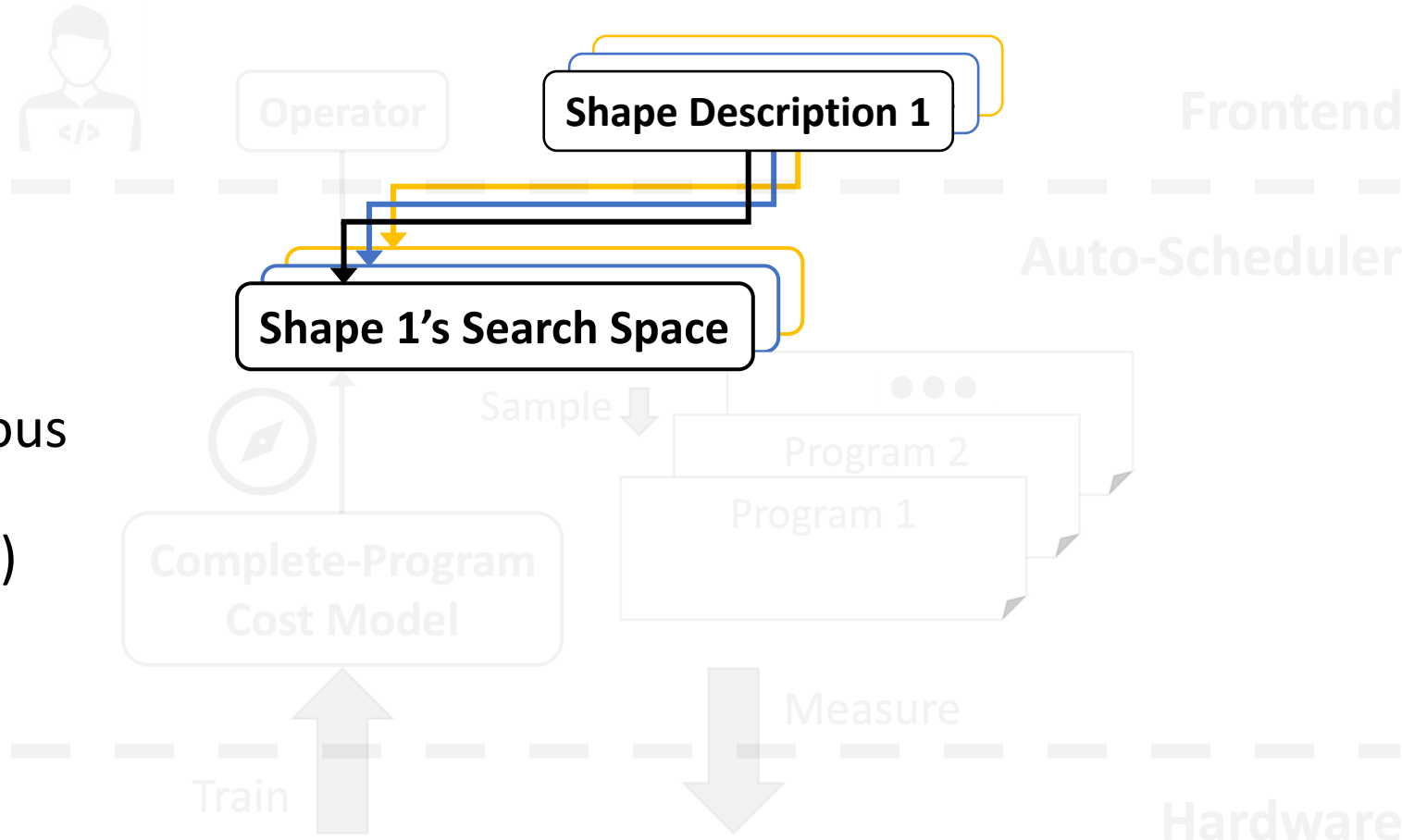
Challenge #1. Humongous Search Space

- **Hard** to share search space between operators of different shapes.
  - ⋂ search space: Tiny
  - ⋃ search space: Humongous



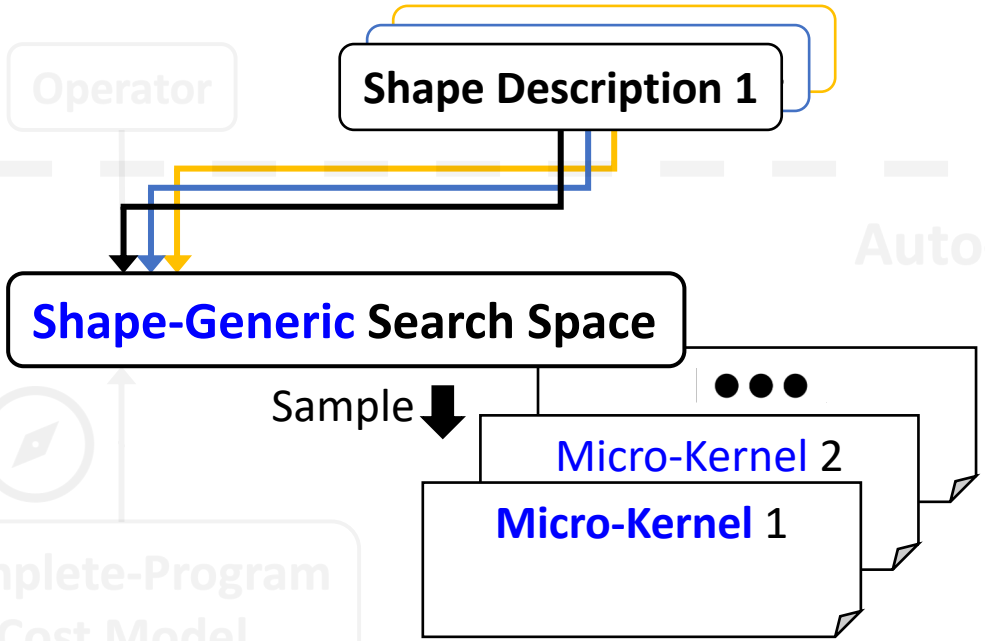
Challenge #1. Humongous Search Space

- **Hard** to share search space between operators of different shapes.
  - ⋂ search space: Tiny
  - ⋃ search space: Humongous
    ⇒ Huge Compilation Time
    (**days** for a single operator)



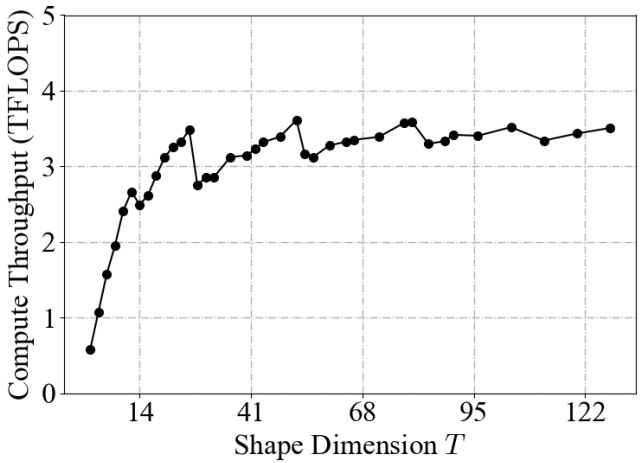
Key Idea #1. Shape-Generic Search Space

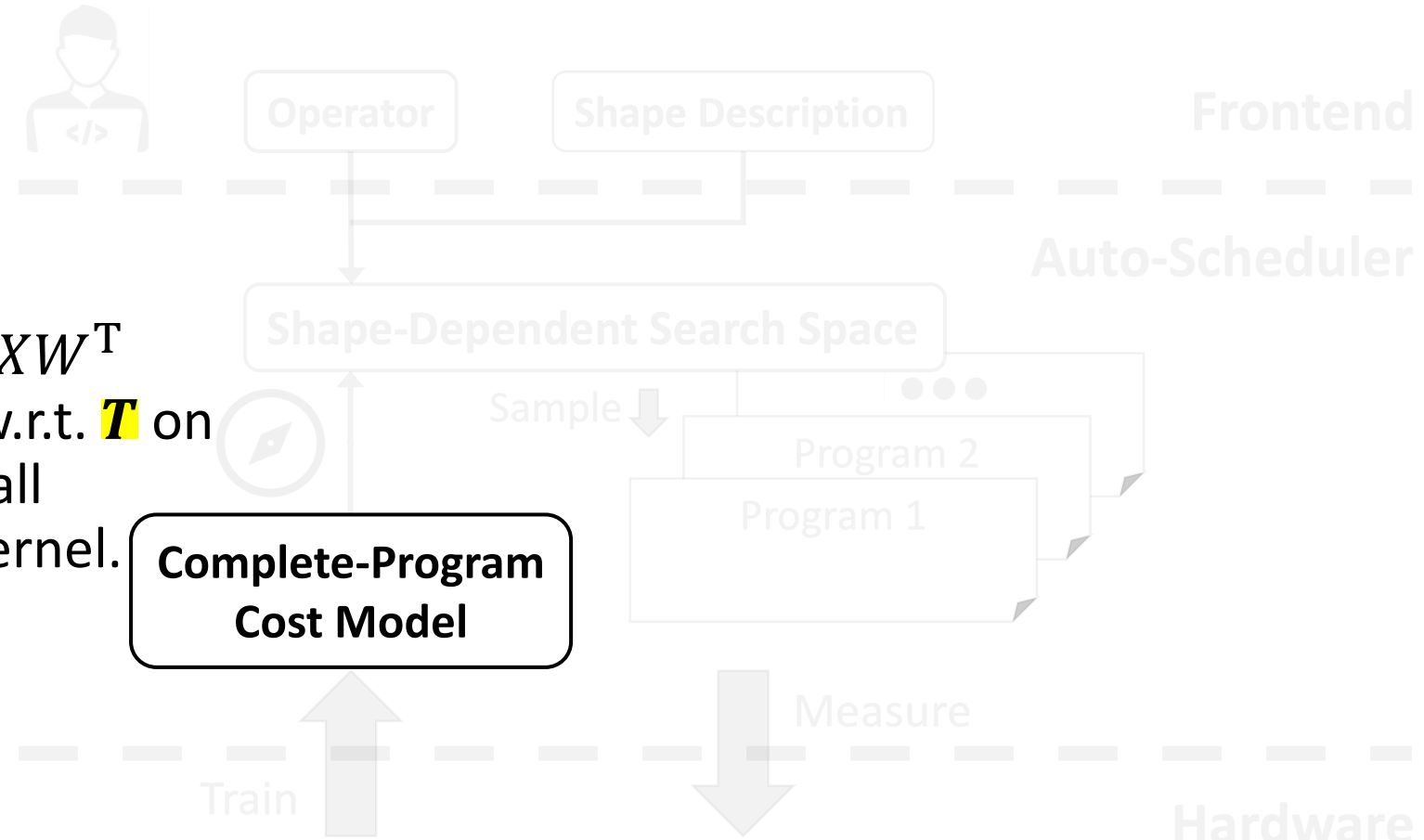
- Composed of **micro-kernels**, each
  - Does a tile of the entire compute.
  - Sampled uniformly from maximum shapes and constrained by hardware parameters.
  - Can be ported to **all** shapes of the same operator.



Challenge #2. Inaccurate Performance Prediction

- Cost model trained on one shape can be **inaccurate** on others.
  - E.g., Performance of $Y = XW^{\mathrm{T}}$ X: $[16\times\boldsymbol{T}, 768]$, W: $[2304, 768]$ w.r.t. $\boldsymbol{T}$ on a NVIDIA Tesla T4 GPU[1], all sharing the same micro-kernel.

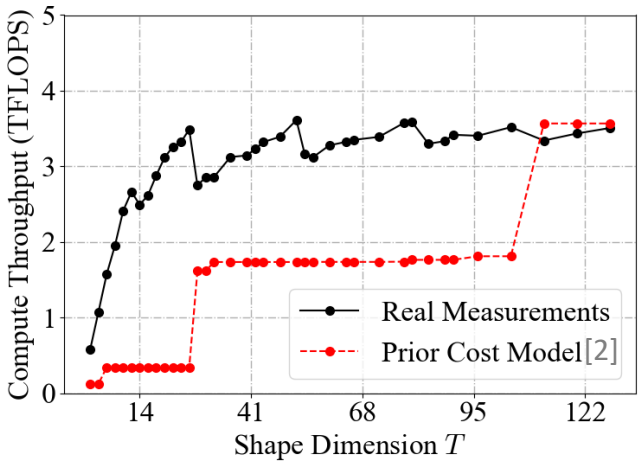




[1] https://www.nvidia.com/en-us/data-center/tesla-t4/

Challenge #2. Inaccurate Performance Prediction

- Cost model trained on one shape can be **inaccurate** on others.
  - E.g., Performance of $Y = XW^{\mathrm{T}}$ X: $[16\times\boldsymbol{T}, 768]$, W: $[2304, 768]$ w.r.t. $\boldsymbol{T}$ on a NVIDIA Tesla T4 GPU[1], all sharing the same micro-kernel.



Complete-Program Cost Model

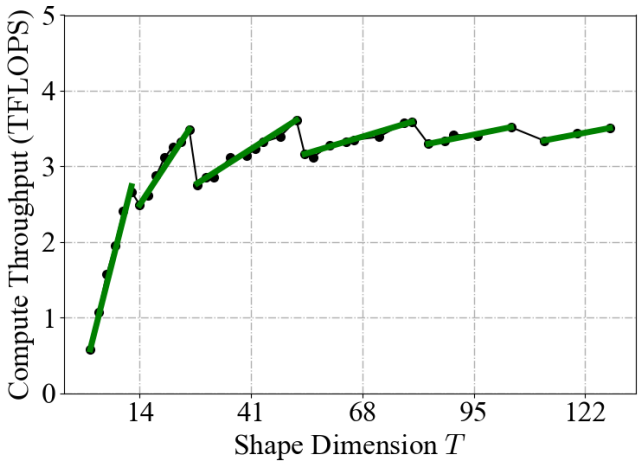
😕 Predictions are **inaccurate** on other shapes.

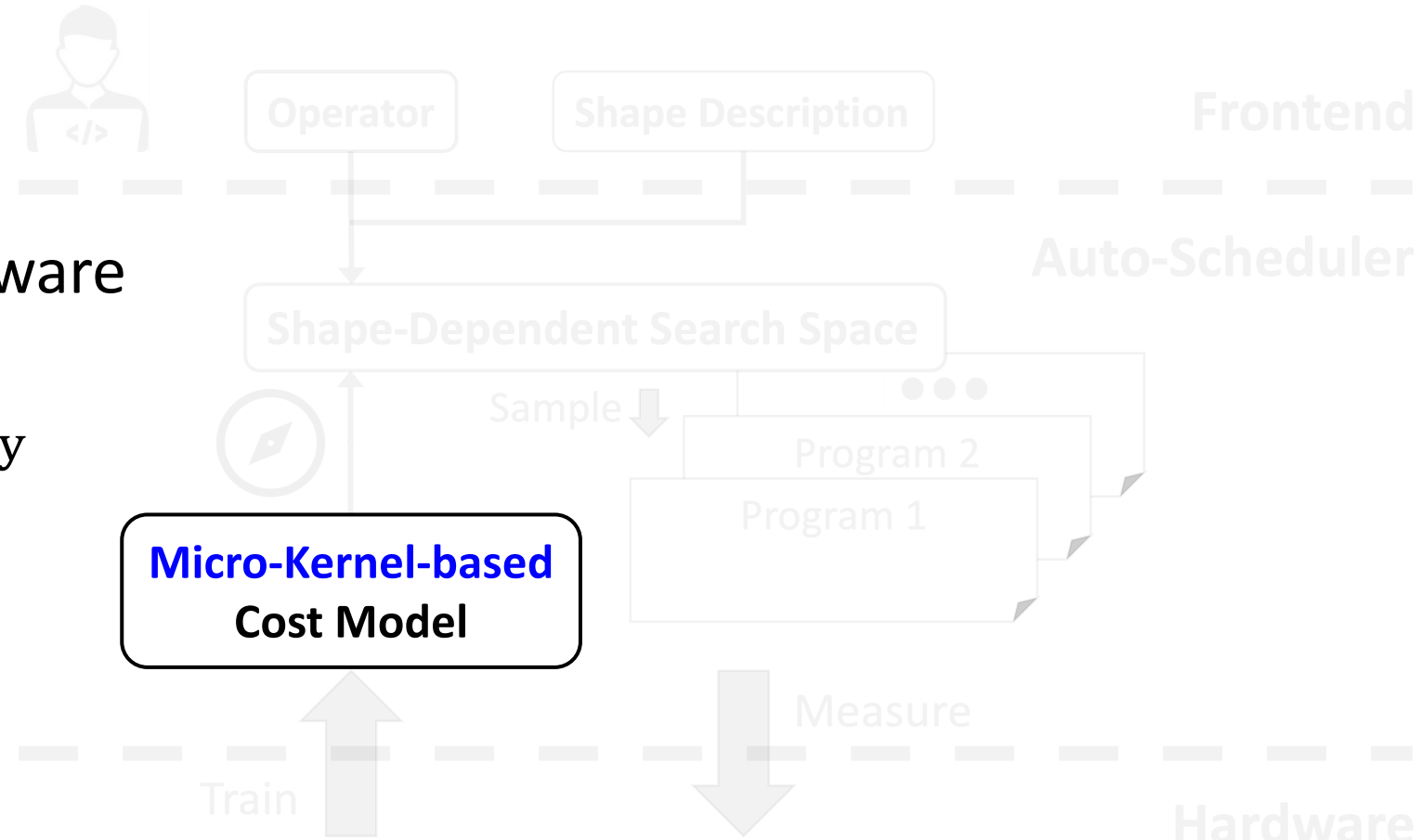
[1] https://www.nvidia.com/en-us/data-center/tesla-t4/
[2] L. Zheng et al. *Ansor*. OSDI 2020

Key Idea #2. Micro-Kernel-based Cost Model

- Key Observation: Performance scales **proportionally** with hardware core occupancy.

$$f_{\text{MicroKernel}} \cdot f_{\text{Penalty}}$$



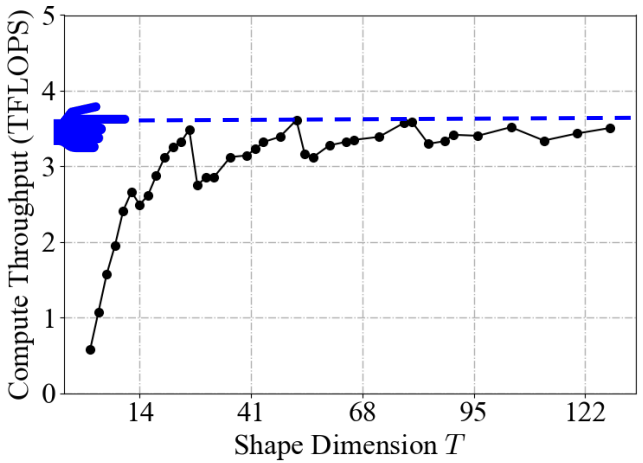


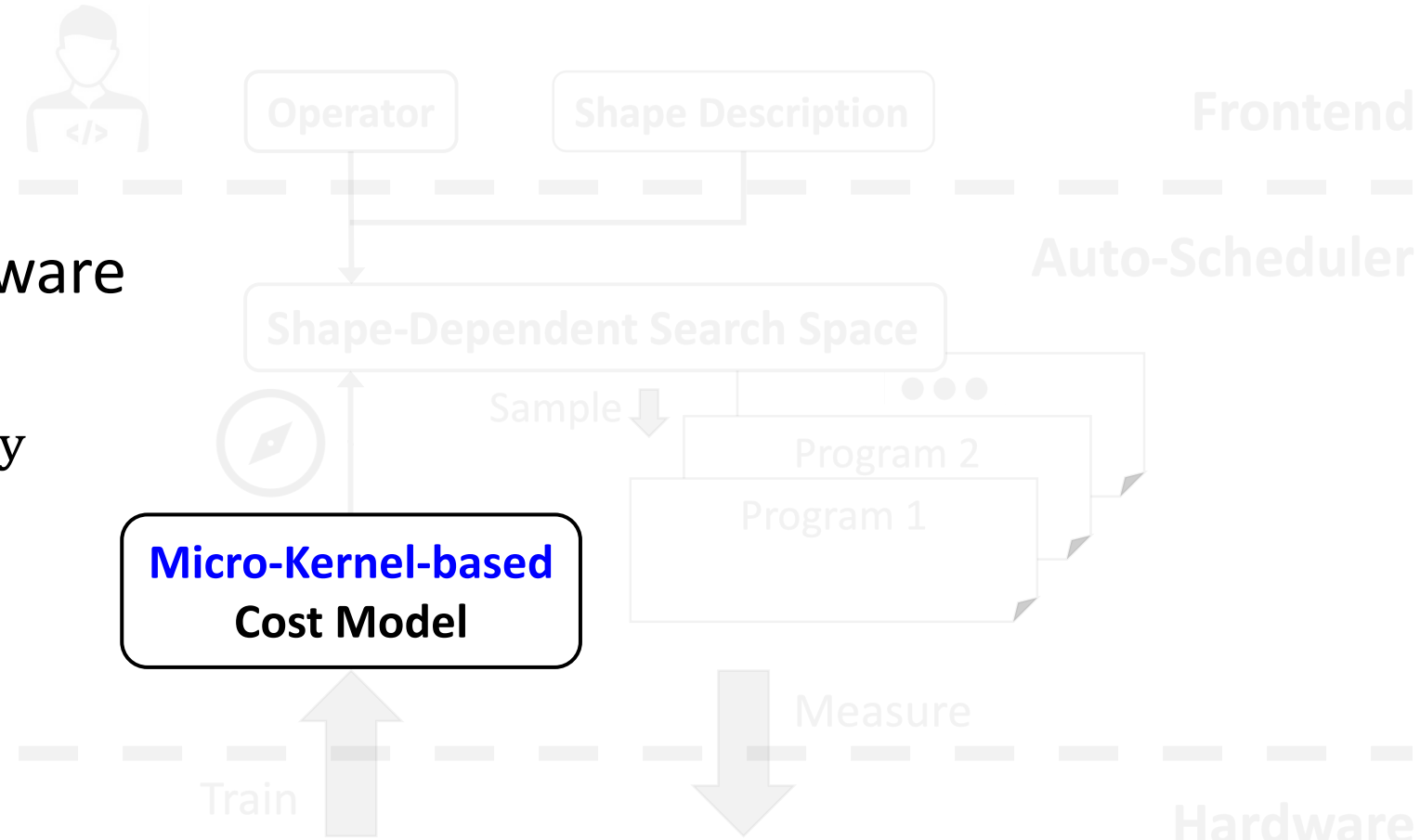
Key Idea #2. Micro-Kernel-based Cost Model

- Key Observation: Performance scales **proportionally** with hardware core occupancy.

$$f_{\text{MicroKernel}} \cdot f_{\text{Penalty}}$$

Trainable function for peak prediction



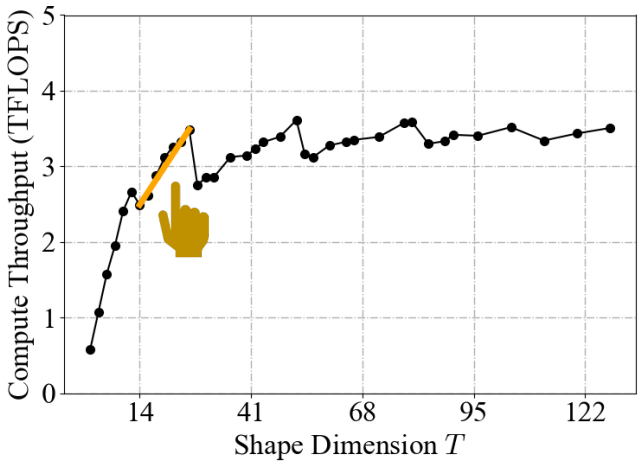


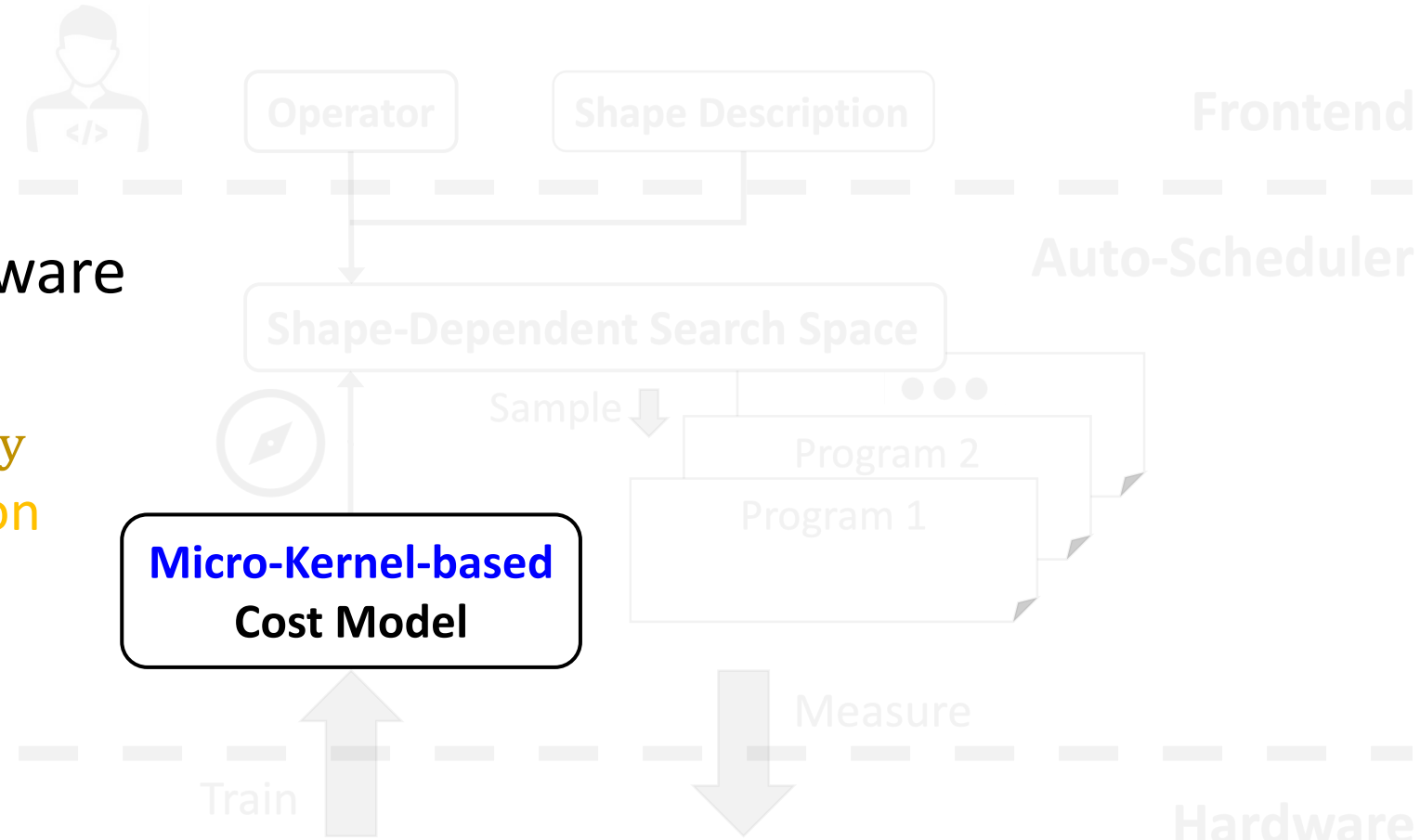
Key Idea #2. Micro-Kernel-based Cost Model

- Key Observation: Performance scales **proportionally** with hardware core occupancy.

$$f_{\text{MicroKernel}} \cdot f_{\text{Penalty}}$$

Analytical **linear** function of the core occupancy



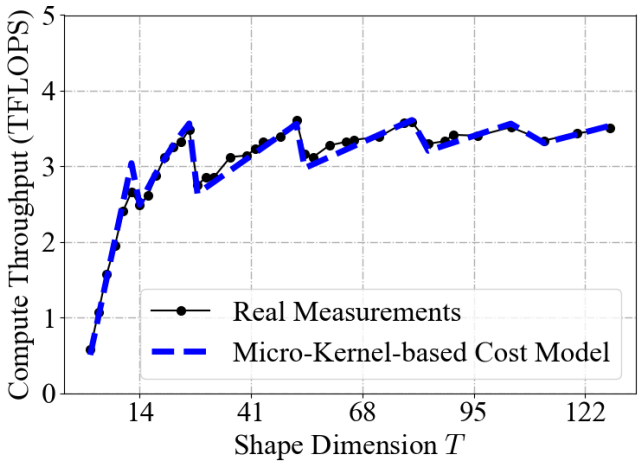


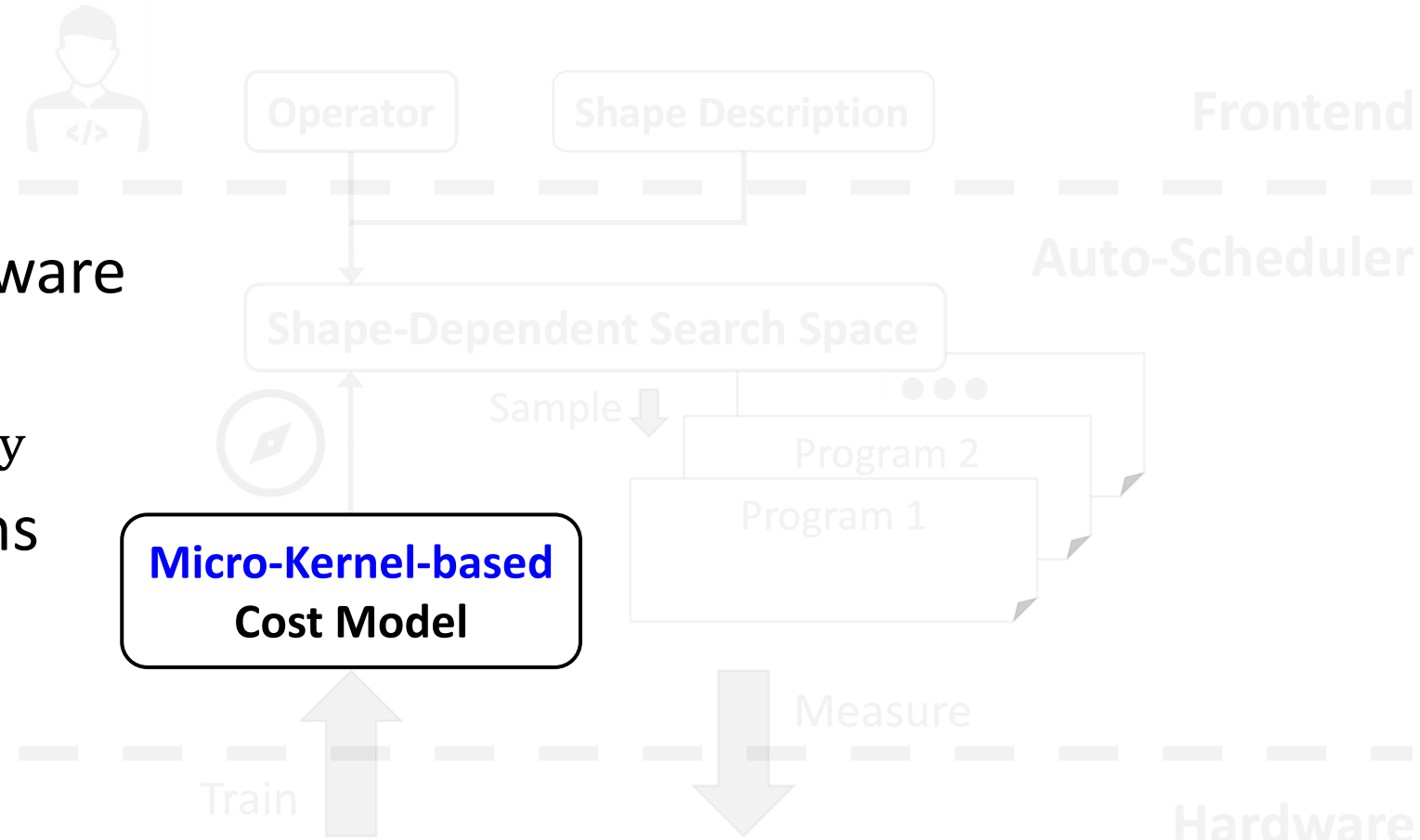
Key Idea #2. Micro-Kernel-based Cost Model

- Key Observation: Performance scales **proportionally** with hardware core occupancy.

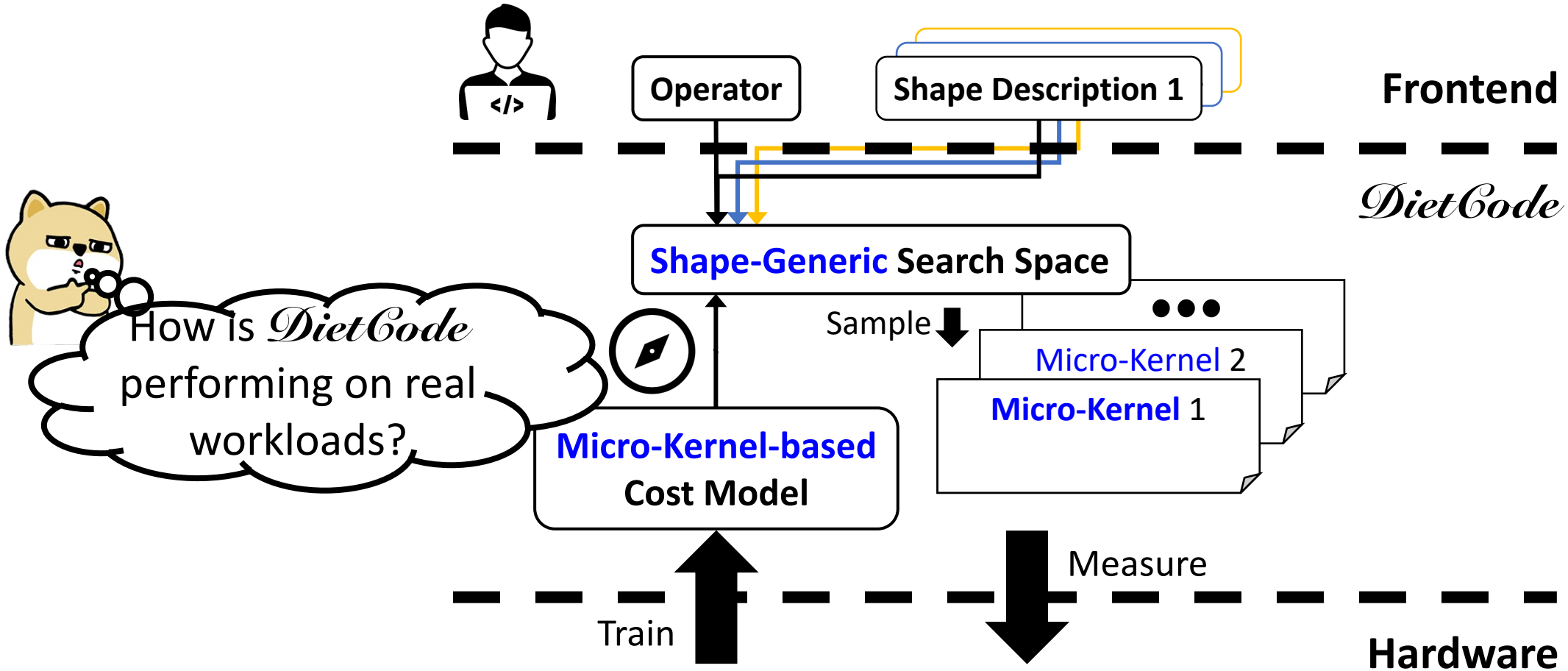
$$f_{\text{MicroKernel}} \cdot f_{\text{Penalty}}$$

- More **Accurate** Predictions





DietCode System Overview



Evaluation

[1] https://www.nvidia.com/en-us/data-center/tesla-t4/
[2] https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/

Hardware

NVIDIA Tesla T4 GPU[1]

NVIDIA RTX 3090 GPU[2]

Evaluation

[1] https://www.nvidia.com/en-us/data-center/tesla-t4/
[2] https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/

Hardware	 NVIDIA Tesla T4 GPU[1]  NVIDIA RTX 3090 GPU[2]
Software	TVM[3] v0.8.dev0 NVIDIA CUDA[4] v11.3 cuDNN[5] v8.3

[3] T. Chen et al. *TVM*. OSDI 2018
[4] https://docs.nvidia.com/cuda/archive/11.3.0/
[5] https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html

Evaluation

[1] https://www.nvidia.com/en-us/data-center/tesla-t4/
[2] https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/

Hardware	 NVIDIA Tesla T4 GPU[1]  NVIDIA RTX 3090 GPU[2]
Software	TVM[3] v0.8.dev0 NVIDIA CUDA [4] v11.3 cuDNN [5] v8.3
Application	 Google BERT [6] Dynamic sequence lengths uniformly sampled within the range $[1, 128]$

[3] T. Chen et al. *TVM*. OSDI 2018
[4] https://docs.nvidia.com/cuda/archive/11.3.0/
[5] https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html
[6] J. Devlin et al. *BERT*. NAACL-HTL 2019

Evaluation

[1] https://www.nvidia.com/en-us/data-center/tesla-t4/
[2] https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/

Hardware	 NVIDIA Tesla T4 GPU[1]  NVIDIA RTX 3090 GPU[2]
Software	TVM[3] v0.8.dev0 NVIDIA CUDA[4] v11.3 cuDNN[5] v8.3
Application	 Google BERT[6] Dynamic sequence lengths uniformly sampled within the range $[1, 128]$
Baselines	PyTorch[7] with the **Vendor** Library ’s Auto-Scheduler **Ansor**[8]

[3] T. Chen et al. *TVM*. OSDI 2018
[4] https://docs.nvidia.com/cuda/archive/11.3.0/
[5] https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html
[6] J. Devlin et al. *BERT*. NAACL-HTL 2019
[7] A. Paszke et al. *PyTorch*. NeurIPS 2019
[8] L. Zheng et al. *Ansor*. OSDI 2020

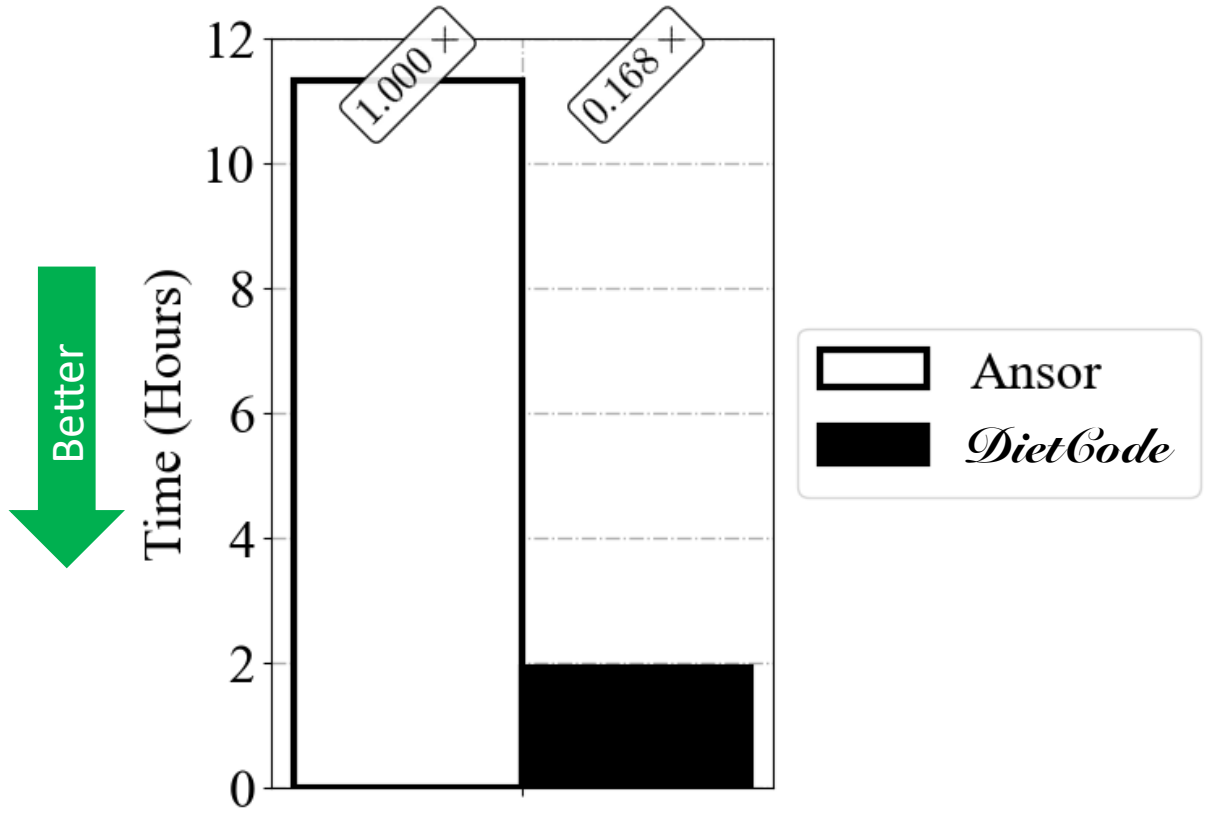
Evaluation

[1] https://www.nvidia.com/en-us/data-center/tesla-t4/
[2] https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/

Hardware	 NVIDIA Tesla T4 GPU[1] NVIDIA RTX 3090 GPU[2]
Software	TVM[3] v0.8.dev0 NVIDIA CUDA[4] v11.3 cuDNN[5] v8.3
Application	 Google BERT[6] Dynamic sequence lengths uniformly sampled within the range $[1, 128]$
Baselines	PyTorch[7] with the **Vendor** Library ’s Auto-Scheduler **Ansor**[8]

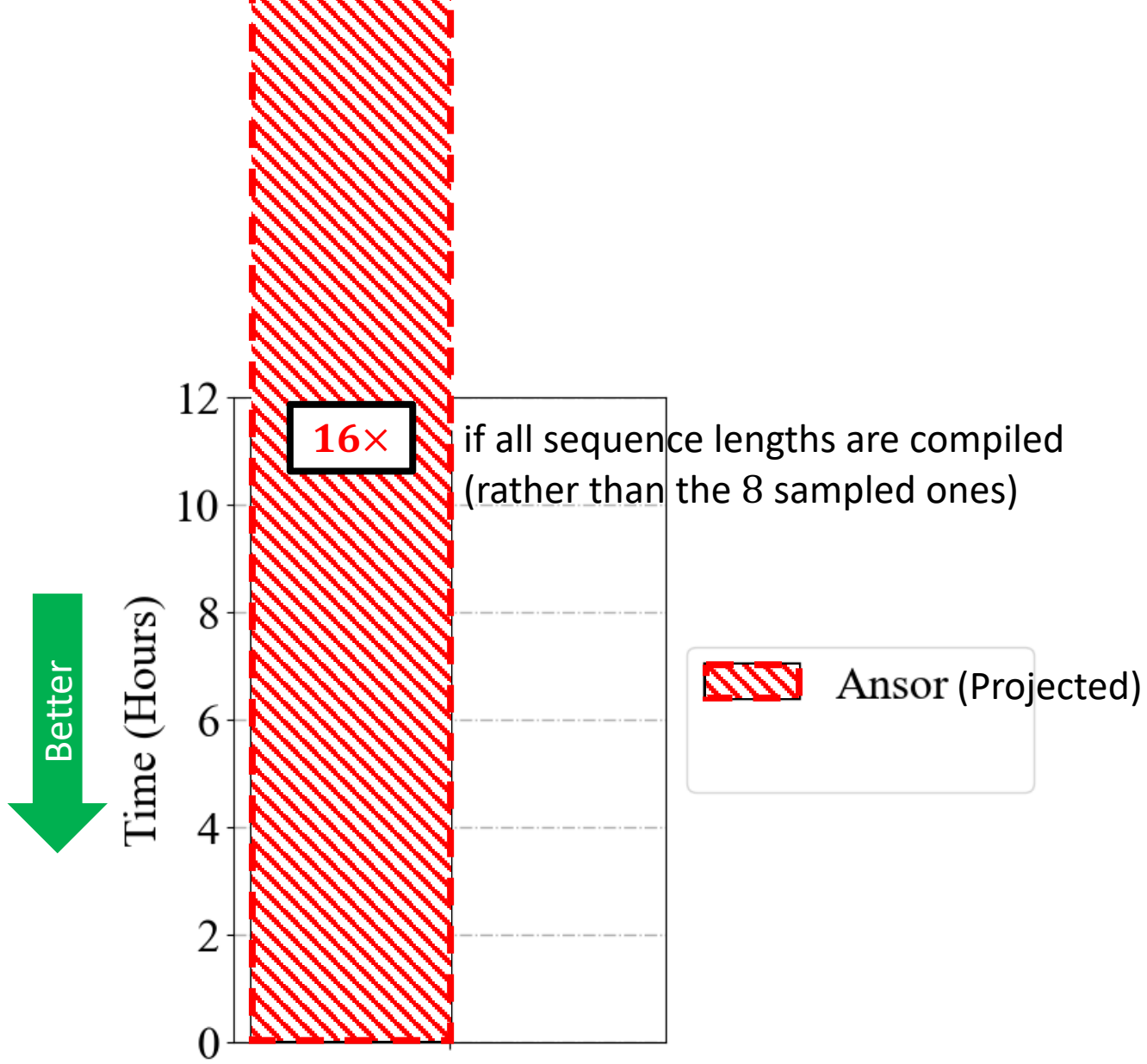
[3] T. Chen et al. *TVM*. OSDI 2018
[4] https://docs.nvidia.com/cuda/archive/11.3.0/
[5] https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html
[6] J. Devlin et al. *BERT*. NAACL-HTL 2019
[7] A. Paszke et al. *PyTorch*. NeurIPS 2019
[8] L. Zheng et al. *Ansor*. OSDI 2020

Compilation Time vs. Ansor



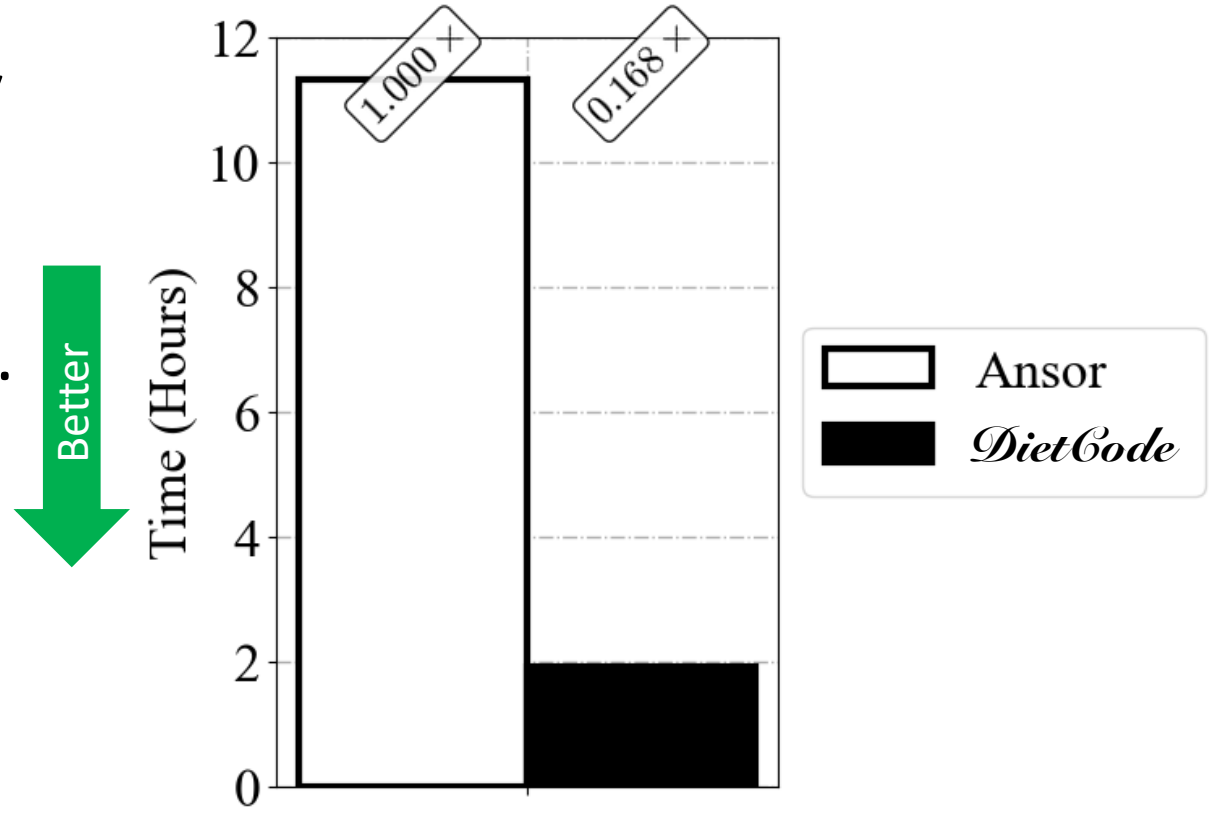
Compilation Time

- Ansor: Time estimated to be more than **1 week** for only key operators.



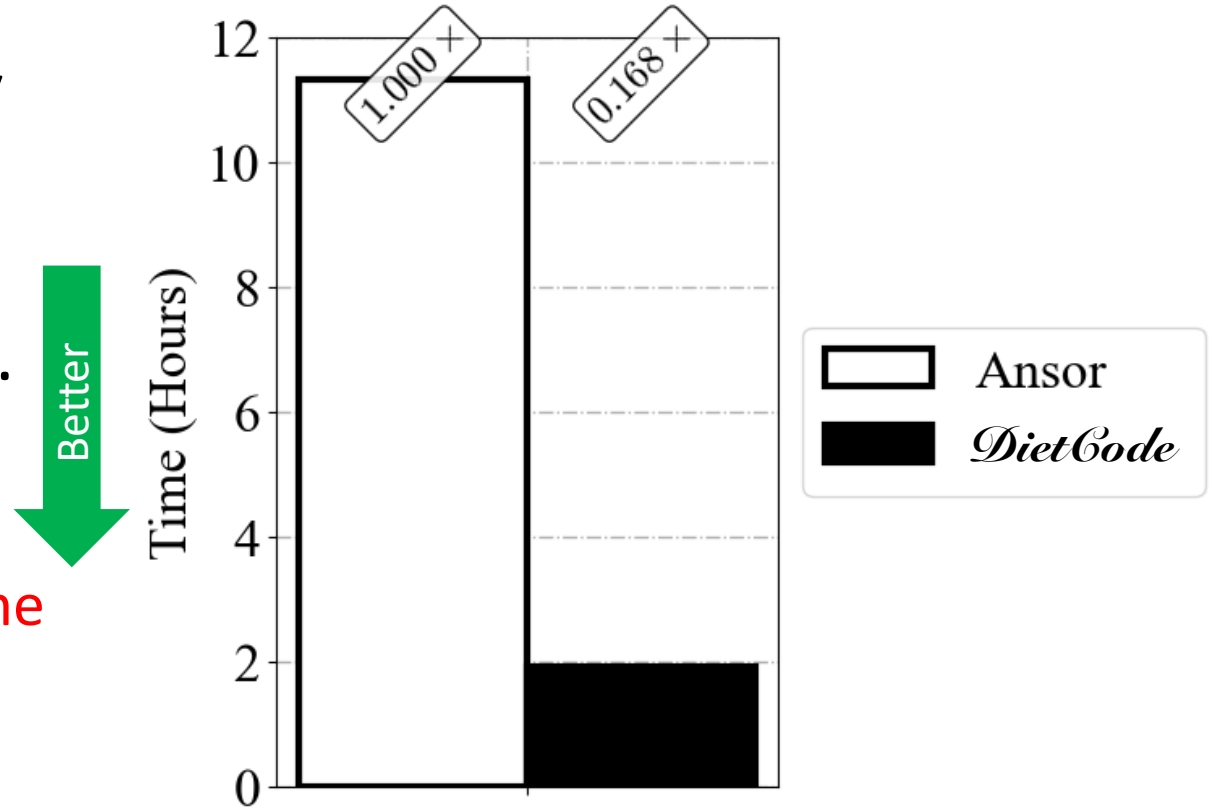
Compilation Time vs. Ansor

- Ansor: Time estimated to be more than **1 week** for only key operators.
- *DietCode* reduces the compilation time by **5.88×** vs. Ansor



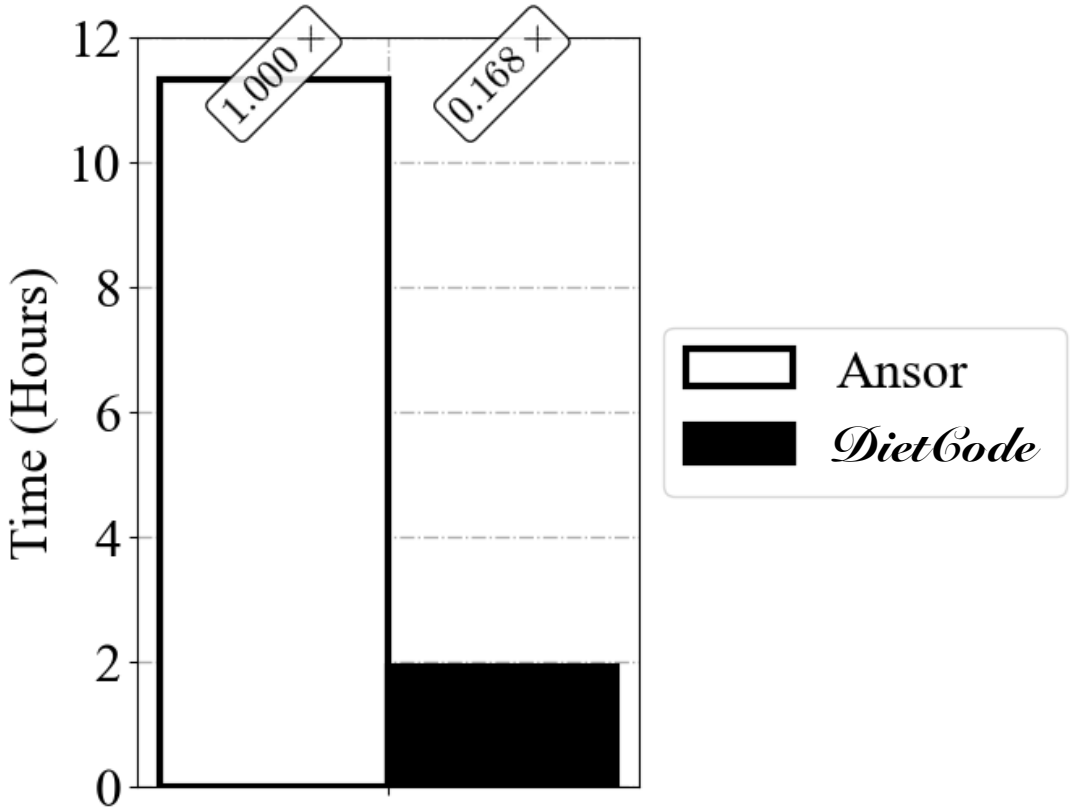
Compilation Time vs. Ansor

- Ansor: Time estimated to be more than **1 week** for only key operators.
- *DietCode* reduces the compilation time by **5.88×** vs. Ansor, as it only needs to **compile once for all shapes**.
  - 16× increase in compilation time

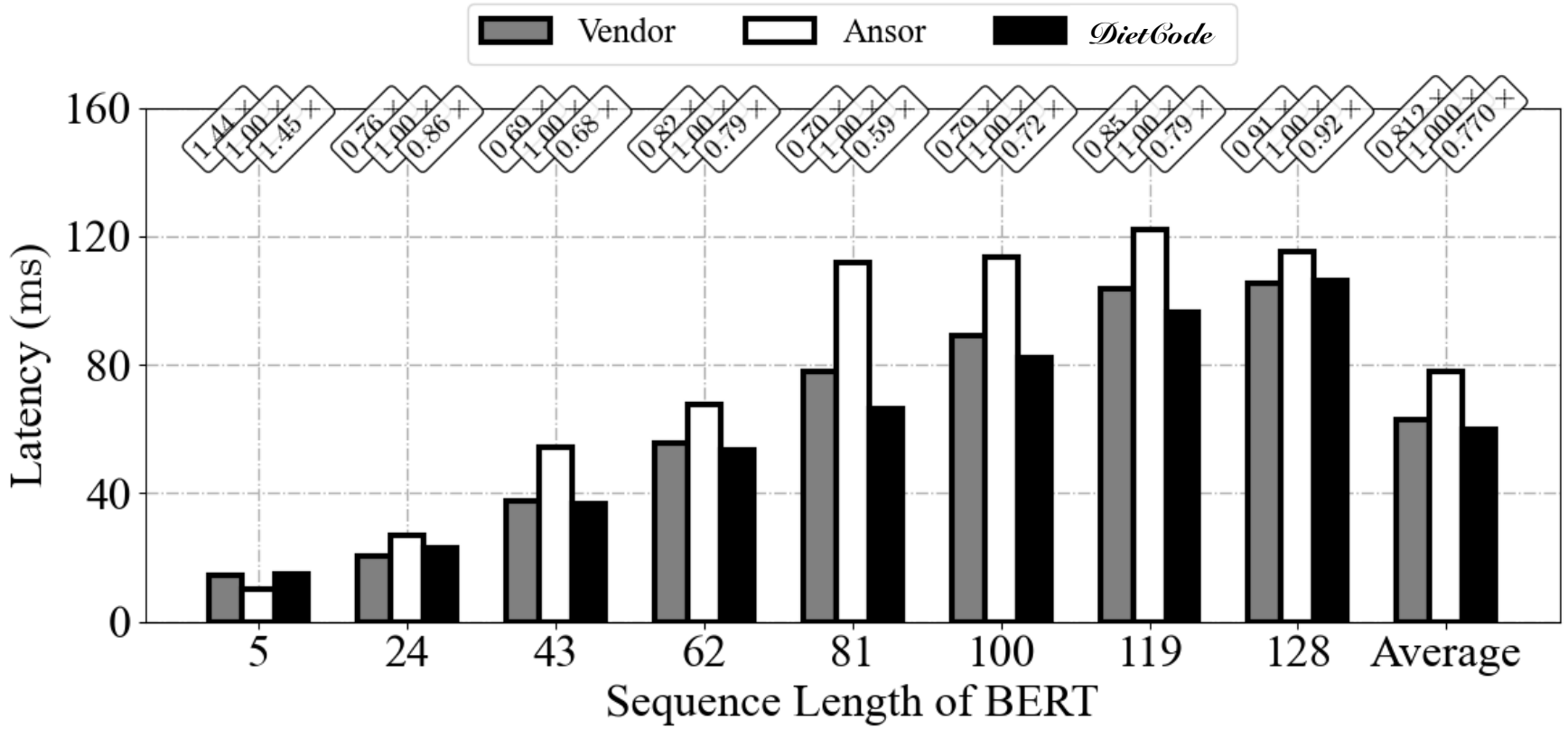


Compilation Time vs. Ansor

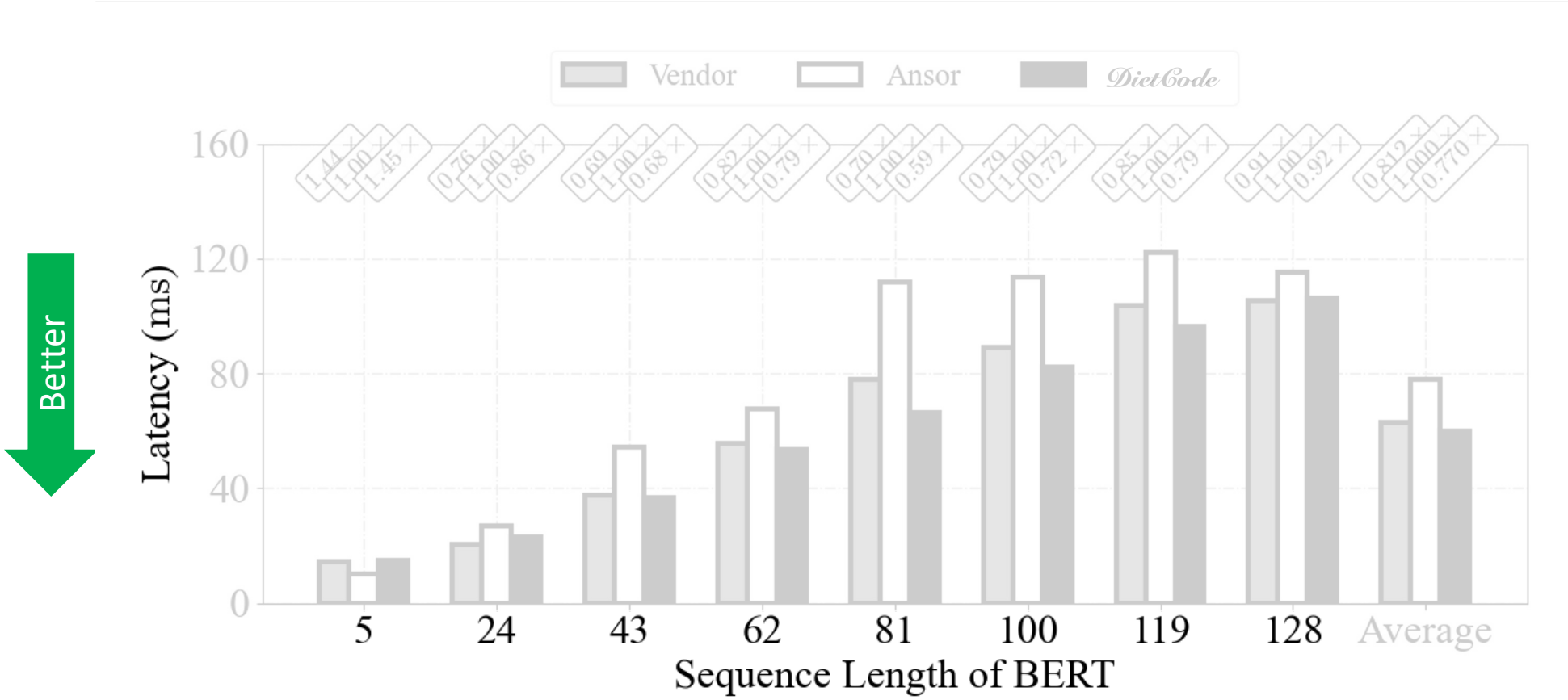
- Ansor: Time estimated to be more than **1 week** for only key operators.
- *DietCode* reduces the compilation time by **5.88×** vs. Ansor, as it only needs to **compile once for all shapes**.
  - ~~16× increase in compilation time~~



Latency vs. Vendor/Ansor

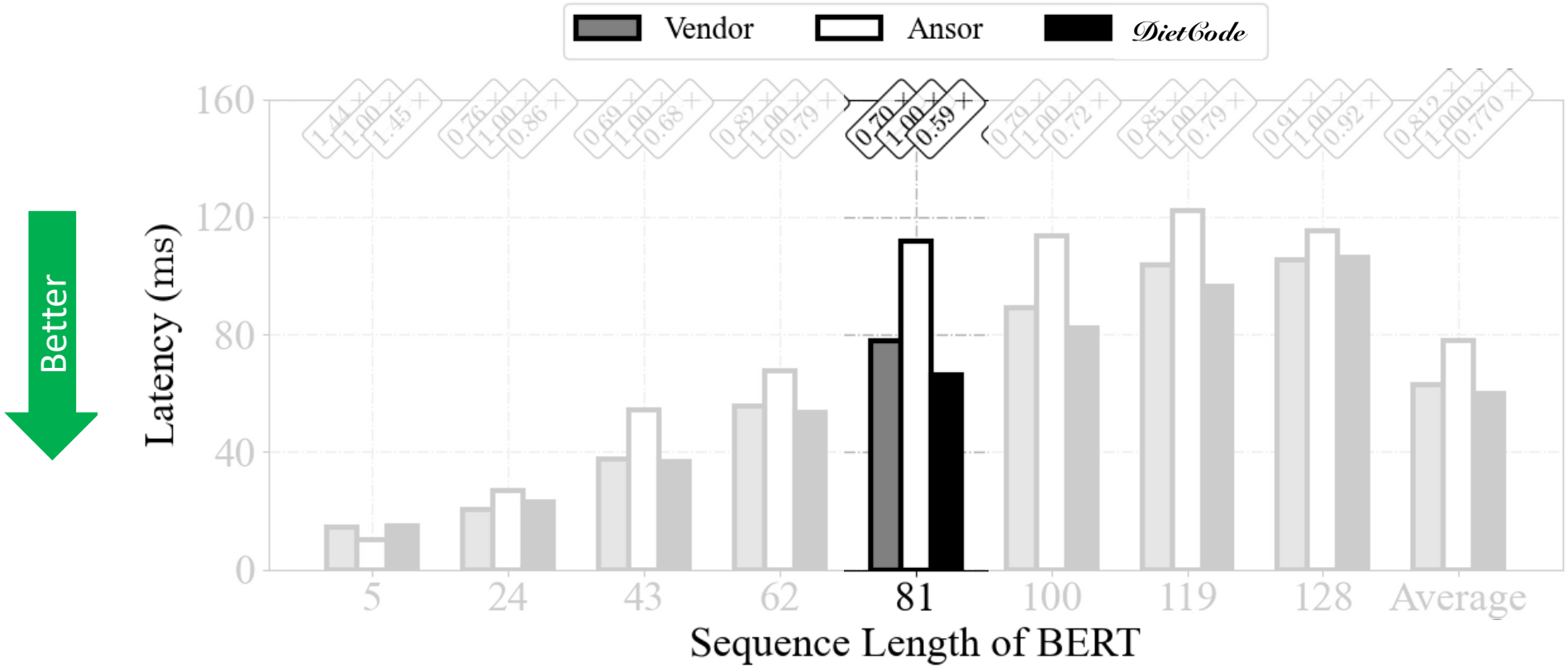


Latency vs. Vendor/Ansor



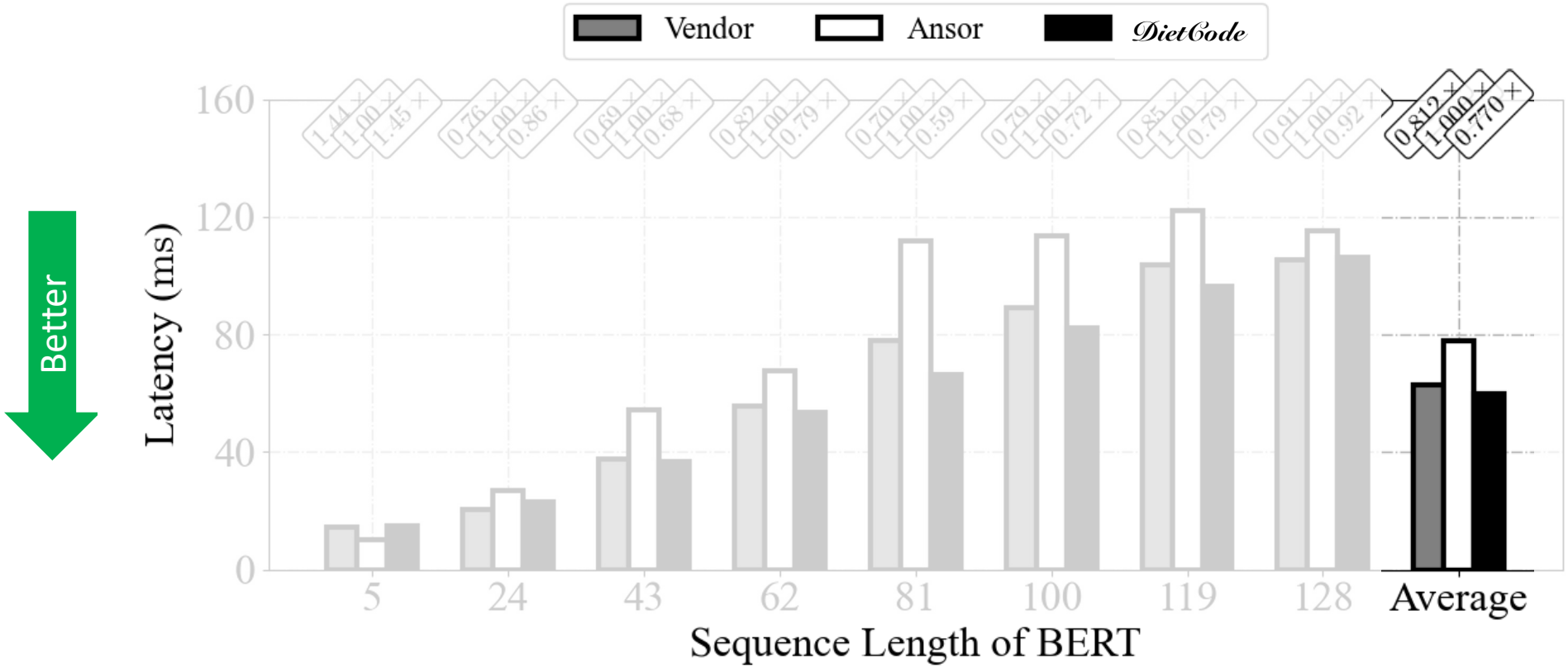
Uniformly sampled and includes composite and **prime** numbers.

Latency vs. Vendor/Ansor



√ Up to $\mathbf{1.70}\times/\mathbf{1.19}\times$ better than Ansor/Vendor.

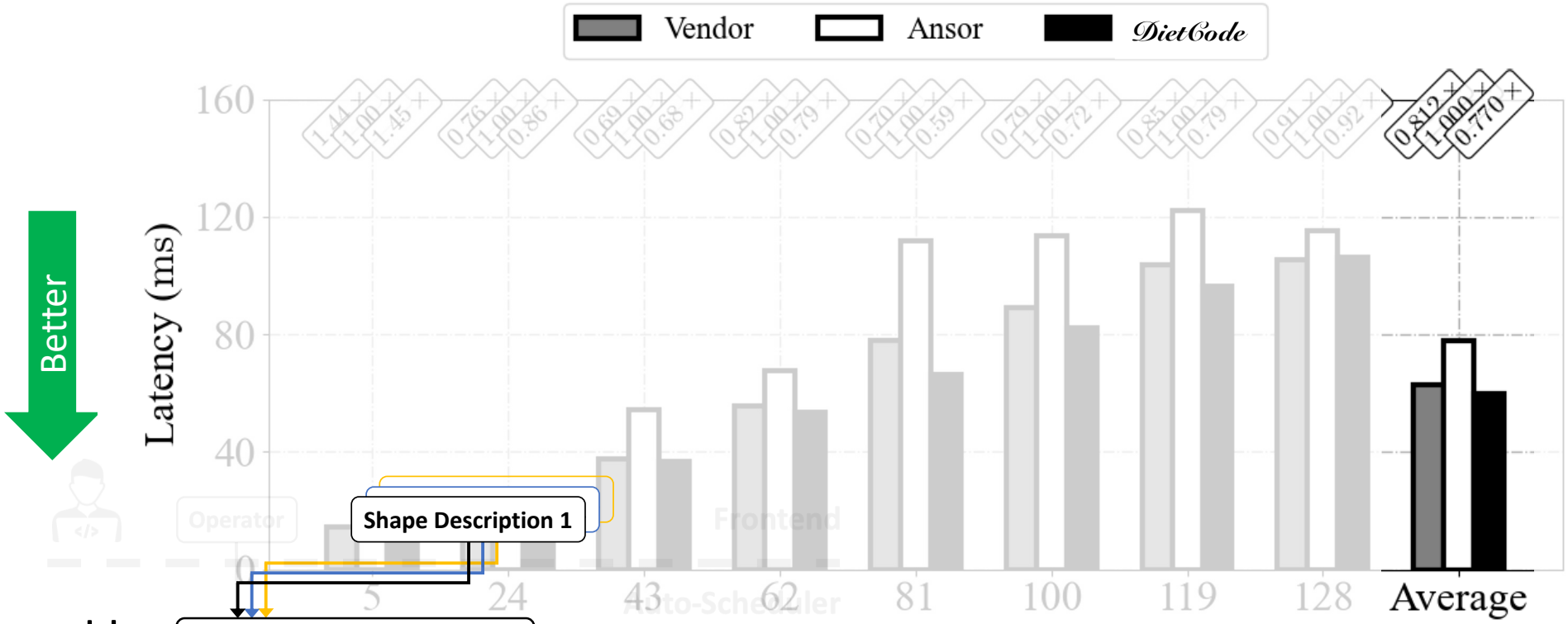
Latency vs. Vendor/Ansor



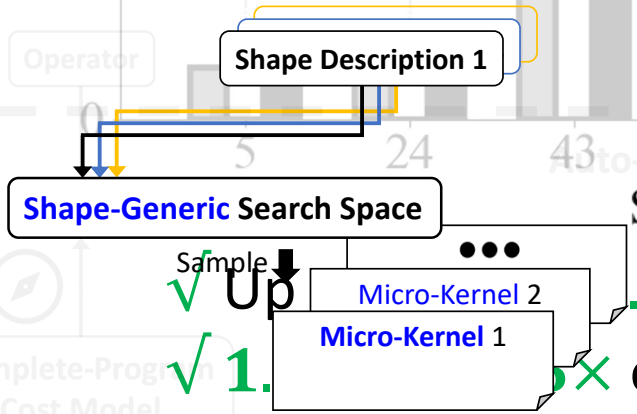
√ Up to $\mathbf{1.70\times}$/$\mathbf{1.19\times}$ better than Ansor/Vendor.

√ $\mathbf{1.30\times}$/$\mathbf{1.05\times}$ on average.

Latency vs. Vendor/Ansor



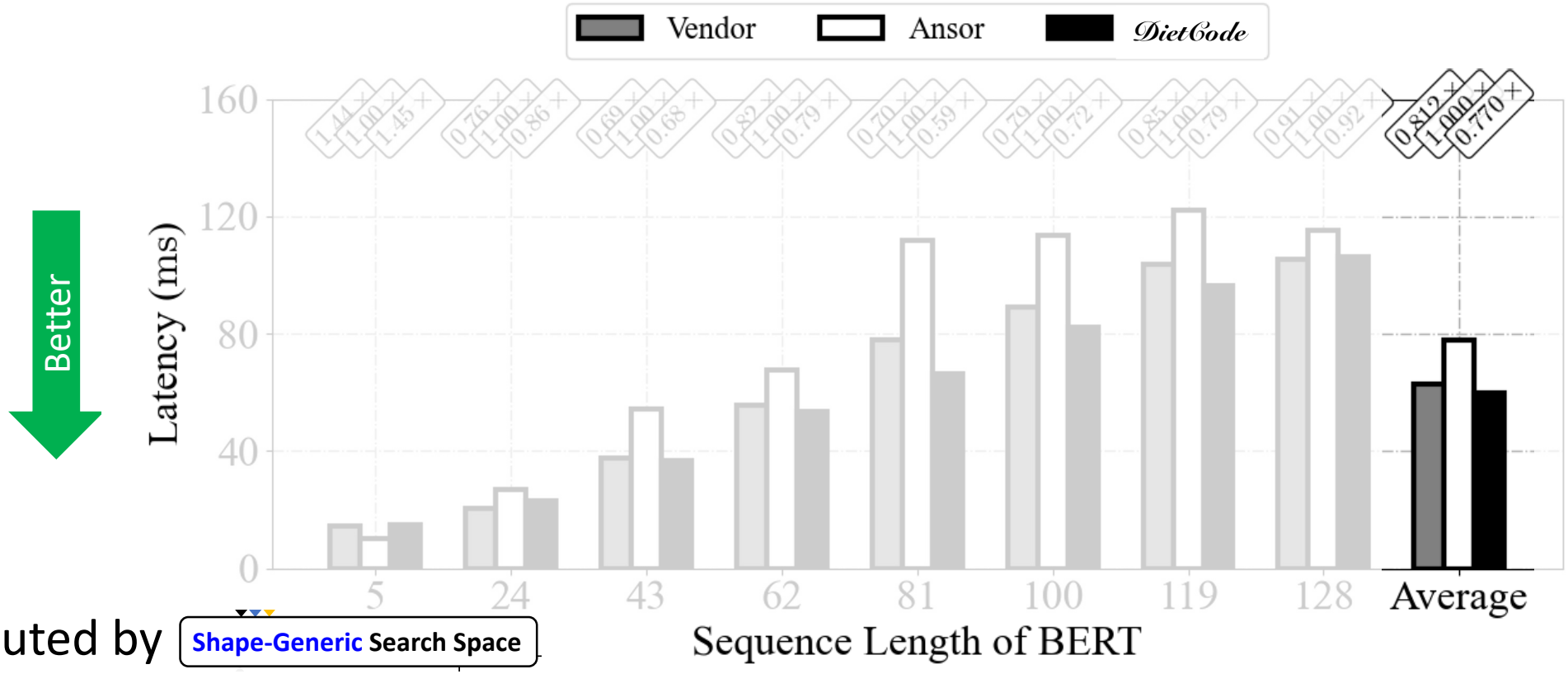
Contributed by



√ Up to **1.19×** better than Ansor/Vendor.

√ **1.**[illegible]**×** on average.

Latency vs. Vendor/Ansor



Contributed by Shape-Generic Search Space

√ Up to $\mathbf{1.70\times}$/$\mathbf{1.19\times}$ better than Ansor/Vendor.

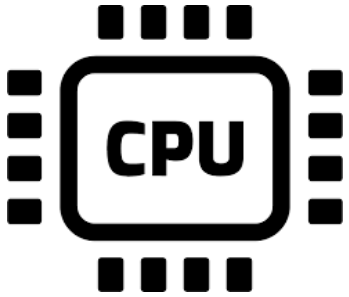
√ $\mathbf{1.30\times}$/$\mathbf{1.05\times}$ on average.

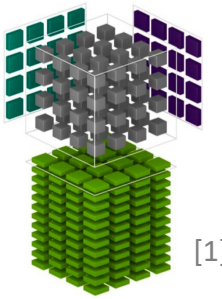
Future Directions

- We are working on upstreaming *DietCode* to the TVM main branch: https://github.com/apache/tvm-rfcs/pull/72, together with improvement of the tuning algorithms.

  Many thanks to the  community!

- Evaluations on more hardware platforms (CPUs and NVIDIA GPUs using tensor core operations)





[1]

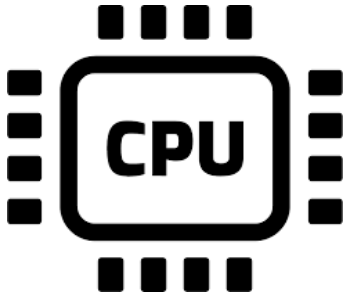
[1] https://www.nvidia.com/en-us/data-center/tensor-cores/

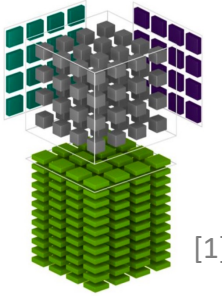
Future Directions

- We are working on upstreaming *DietCode* to the TVM main branch: https://github.com/apache/tvm-rfcs/pull/72, together with improvement of the tuning algorithms.

  Many thanks to the  community!

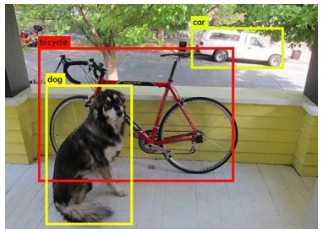
- Evaluations on more hardware platforms (CPUs and NVIDIA GPUs using tensor core operations) and workloads.





[1]

Speech Recognition[2]



Object Detection[3]

[1] https://www.nvidia.com/en-us/data-center/tensor-cores/
[2] https://github.com/NVIDIA/NeMo
[3] K. He et al. Mask R-CNN. ICCV 2017

Conclusion

- Challenges posed by dynamic-shape workloads:
  - Vendor Libraries: Hard to be Engineered for Efficiency
  - Existing Auto-Schedulers: Long Compilation Time
- *DietCode* addresses the challenges with
  ① shape-generic search space
  ② micro-kernel-based cost model.
- Key Results:
  - Compilation Time: $\mathbf{5.88}\times$ saving vs. Ansor.
  - Performance: Up to $\mathbf{1.70}\times$ better vs. Ansor and $\mathbf{1.19}\times$ vs. the vendor library on modern GPUs.

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 Contributions

1 

2 

3 

4 

5 

6 

7 

Optimization Strategy

- **Objective**: End-to-End Latency

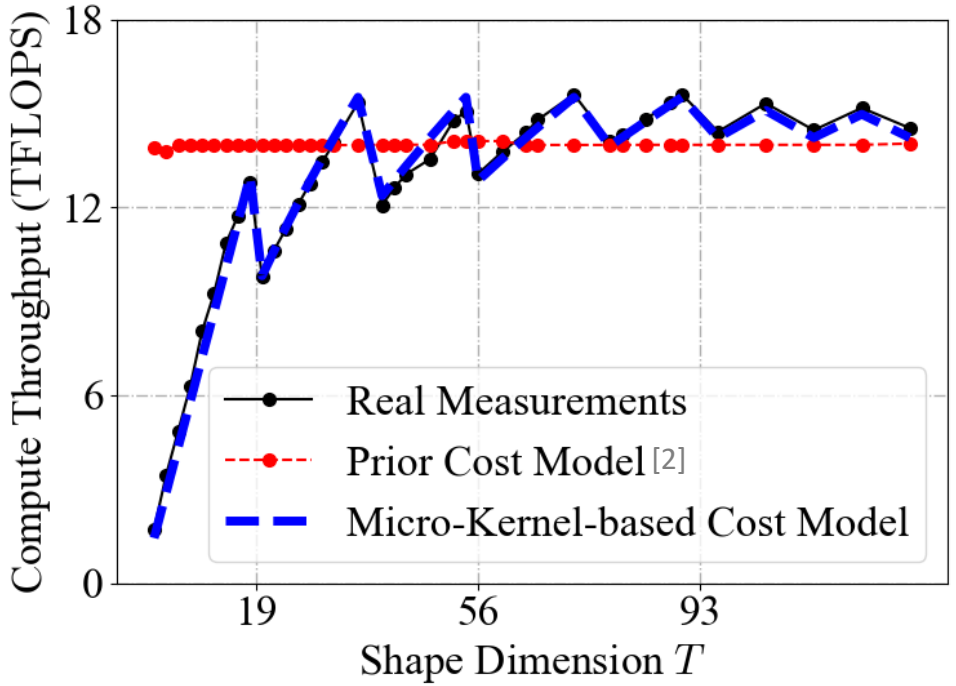
- All workloads have “optimization priority” as

$$\text{FLOPs} \times \text{weight (user-defined, 1 by default)}$$

- Consequently, workloads with higher FLOPs will be given more attention compared with smaller ones.

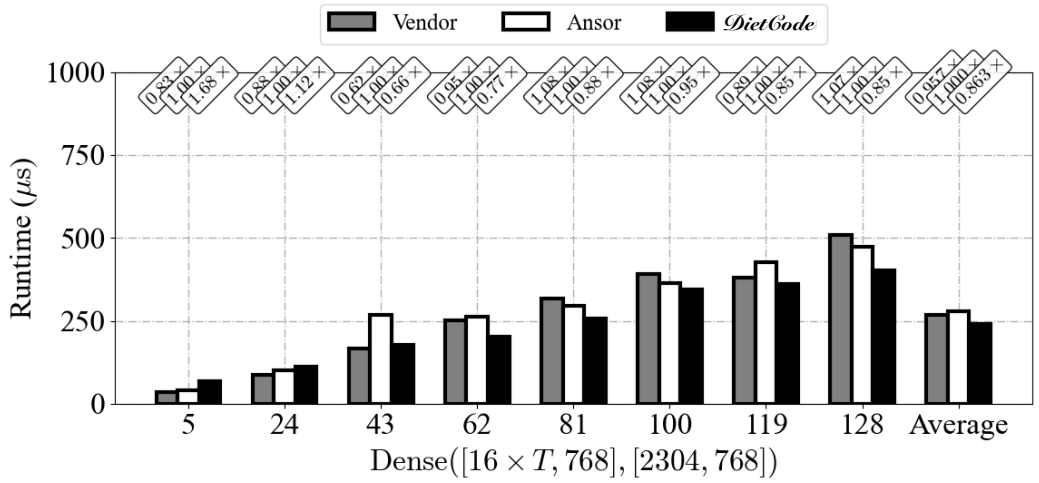
NVIDIA RTX 3090[1] Preliminary Results

Micro-Kernel-based Cost Model



Performance

Better

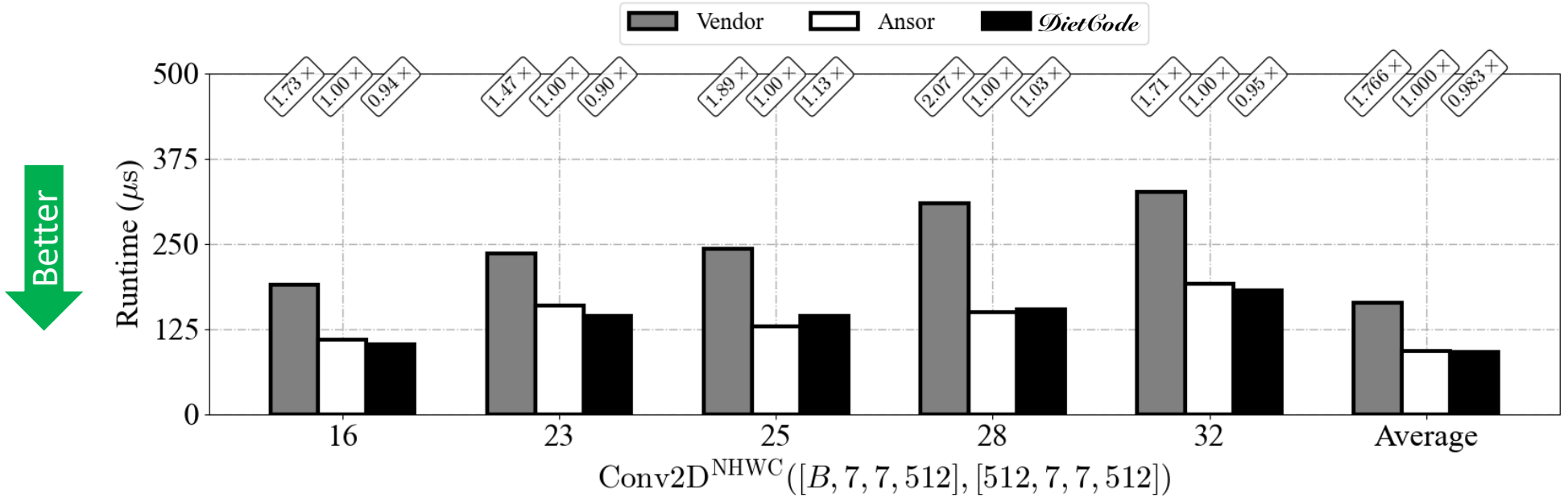


Up to $\mathbf{1.52\times}$/$\mathbf{1.26\times}$ better than Ansor/Vendor ($\mathbf{1.16\times}$/$\mathbf{1.11\times}$ on average).

[1] https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/
[2] L. Zheng et al. *Ansor*. OSDI 2020

Conv2D

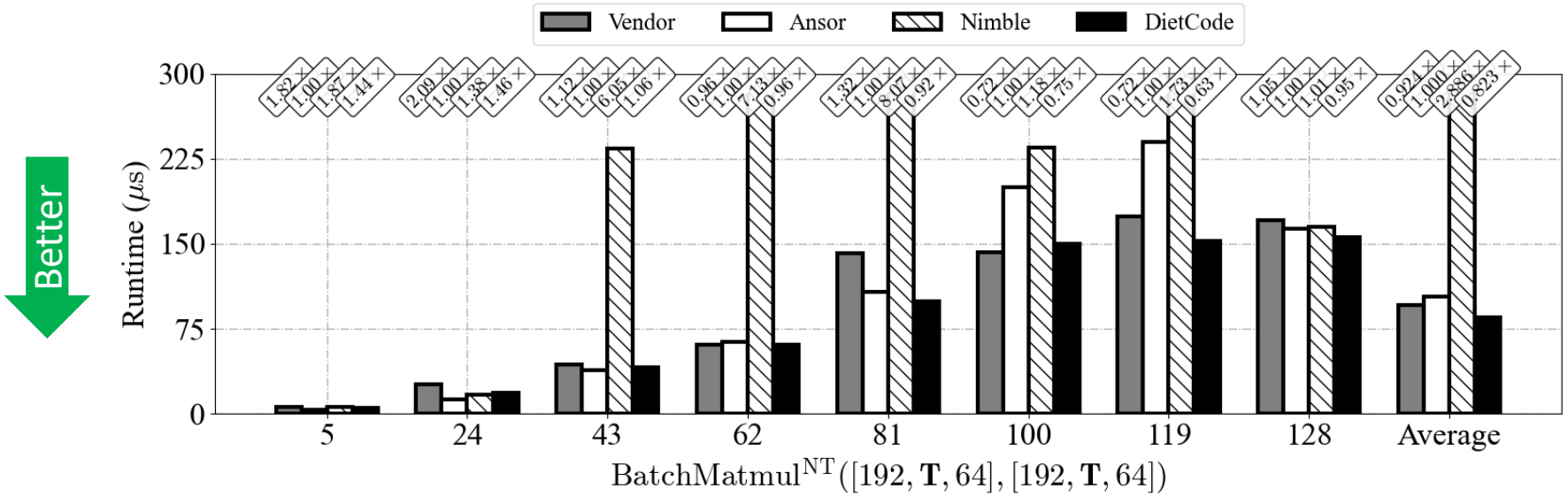
- NCHW is usually implemented using the Winograd algorithm, which is in essence batched matrix multiplies.
- NHWC:



Up to $\mathbf{1.11}\times/\mathbf{2.01}\times$ better than Ansor/Vendor ($\mathbf{1.02}\times/\mathbf{1.80}\times$ on average).

vs. Nimble [1]

- Focuses on the **runtime system** for dynamic-shape workloads, with one section (i.e., Section 3.5) discussing about the code generation.



Nimble cannot cover all shapes efficiently.

[1] H. Shen, J. Roesch et al. *Nimble*. MLSys 2021

Local Padding vs. Loop Partitioning

- Common:
  - Key Observation: Out-of-boundary checks in the **compute** stage are what negatively affect performance the most.
  - Performance difference is usually less than 5%.

Local Padding vs. Loop Partitioning

Local Padding

- Key Idea: Pads tensors by the size of the local workspace when loading from the off-chip device memory.

(-) Redundant computations

Loop Partitioning

- Key Idea: Partition the regions that have predicates and regions that do not.

(-) Cannot remove overheads in some pathological cases.

(-) Cannot support compute intrinsics such as the tensor core operations.