Dynamic Tensor Shape Compilers

Yijia Diao

2022.11.15

Outline

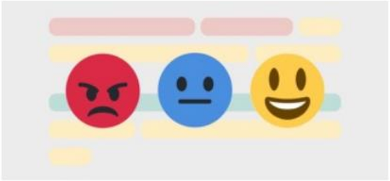
- Background
- Nimble
- DISC
- DietCode
- Summary

Dynamic-Shape Workloads

- Input sentences or audios have dynamic lengths.

Translation[1]

Speech Recognition[2]

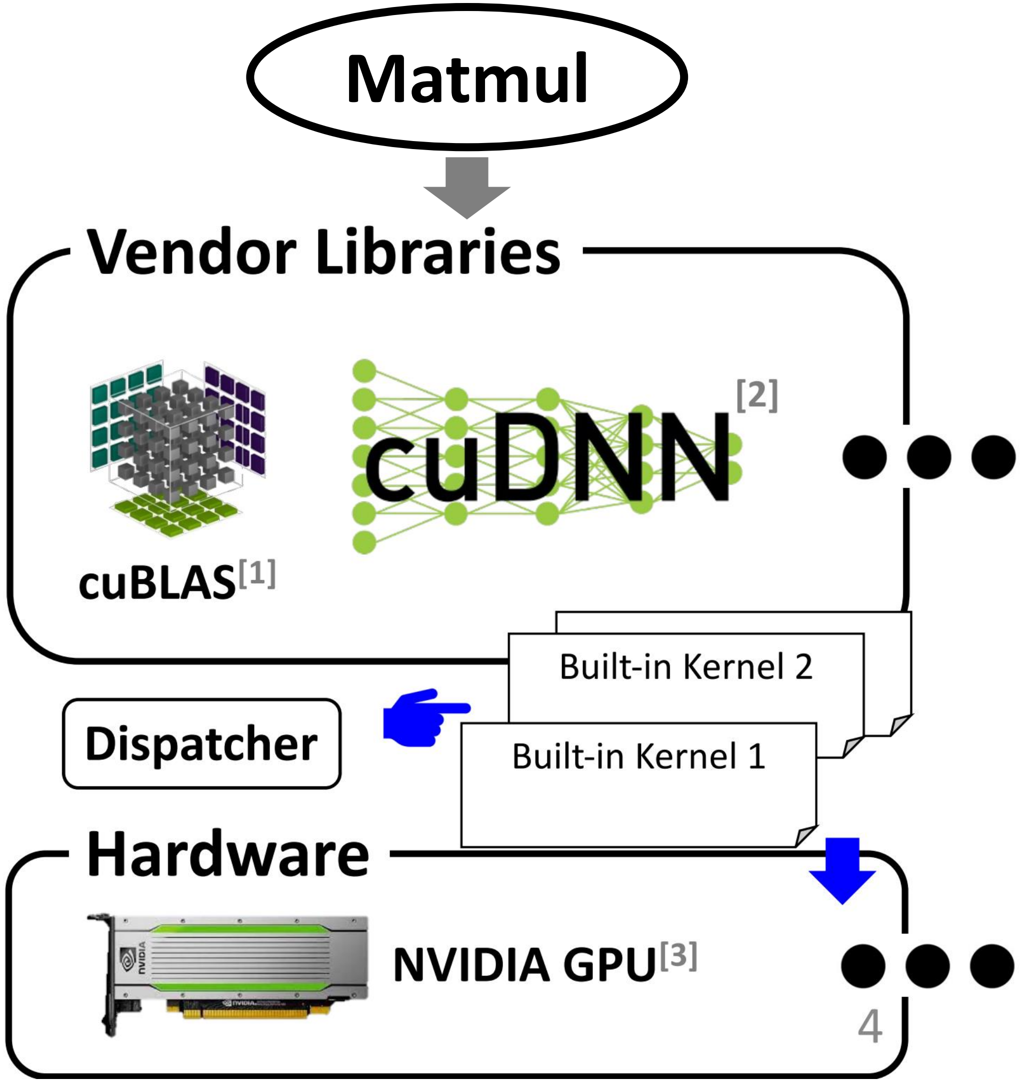


Sentiment Analysis[3]

Text Auto-Complete[4]

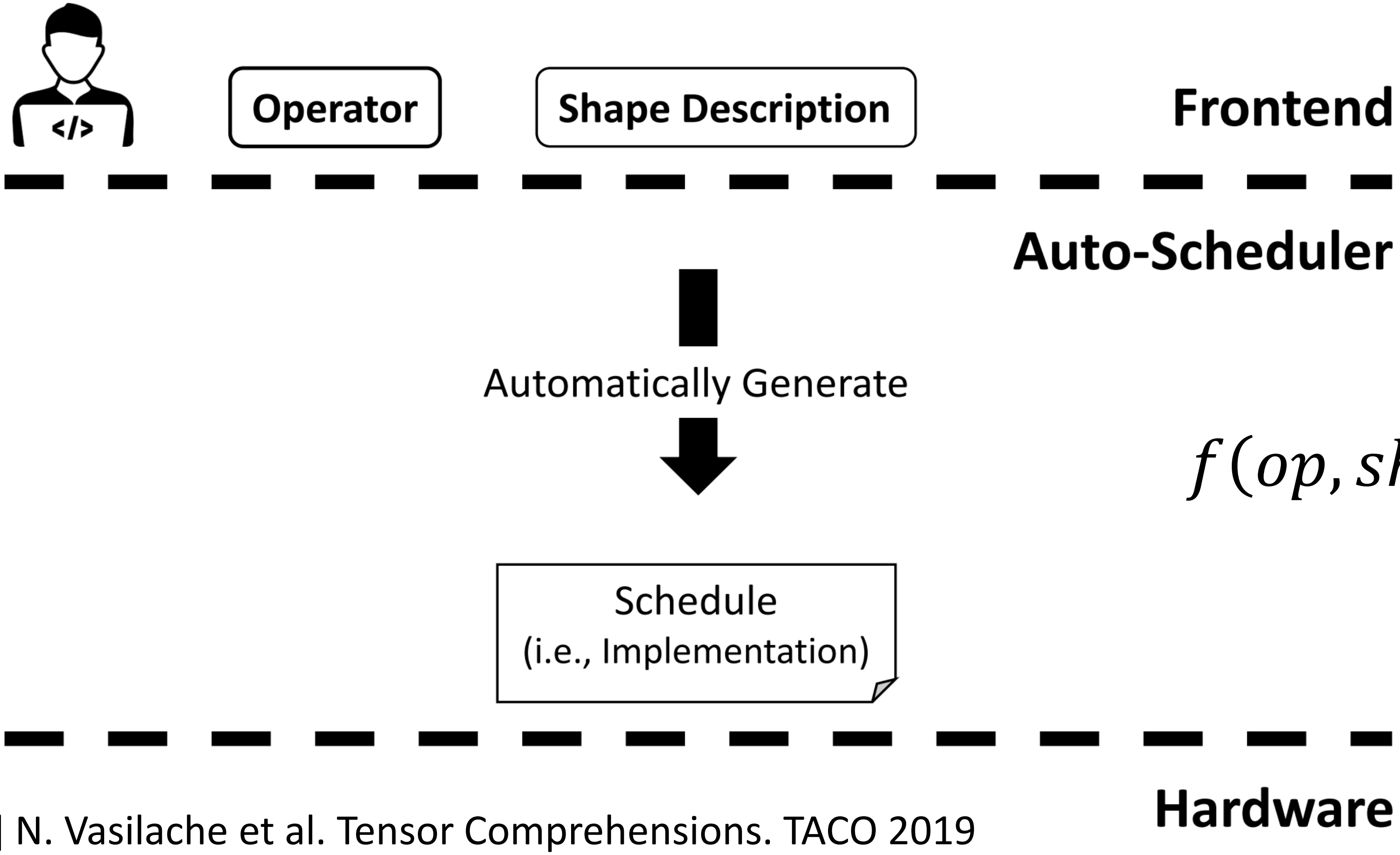
[1] https://translate.google.com/ [2] https://github.com/NVIDIA/NeMo
[3] J. Devlin et al. BERT. NAACL-HTL 2019 [4] A. Radford et al. GPT-2. 2019
https://mlsys.org/media/mlsys-2022/Slides/2175.pdf

Challenges: Vendor Libraries



- Huge engineering efforts
- Built-in kernels sometimes not optimal
  - If kernel for every shape optimal, the library would be huge 😨
- Unknown source code

Challenges: Tensor Compilers

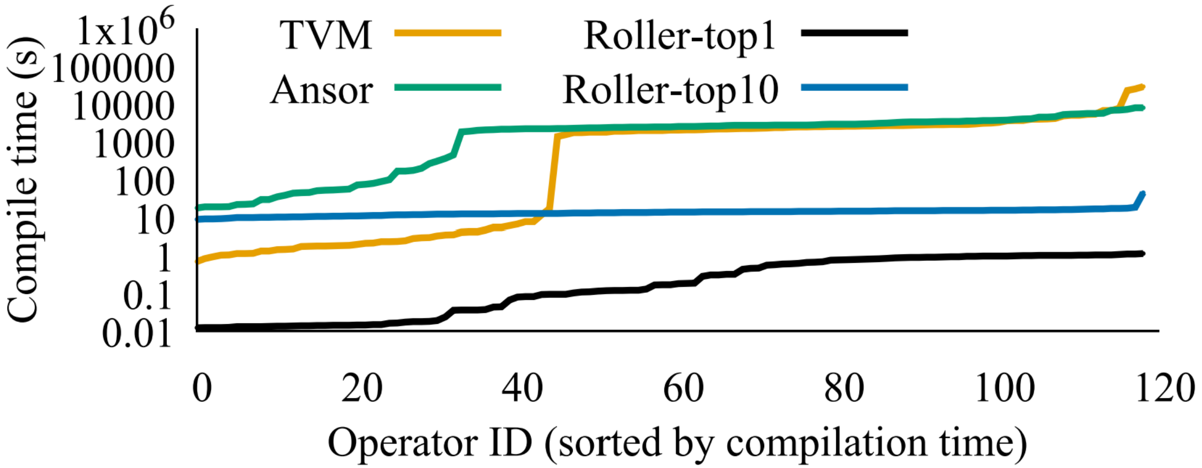


$$f(op, shape, hw) = Schedule$$

[1] N. Vasilache et al. Tensor Comprehensions. TACO 2019
[2] L. Zheng et al. Ansor. OSDI 2020
[3] S. Zheng et al. FlexTensor. ASPLOS 2020
[4] H. Zhu et al. Roller. OSDI 2022

Challenges: Tensor Compilers (Cont.)

- For tensor compilers like Ansor, it needs hour-level time to tune single shape for single op.
- The cost would be huge when input shape is dynamic.



	Tuning?	Complexity
Vendor Libraries	✘	-
Existing Auto-Schedulers	✓	$O(\|S\|)$

Outline

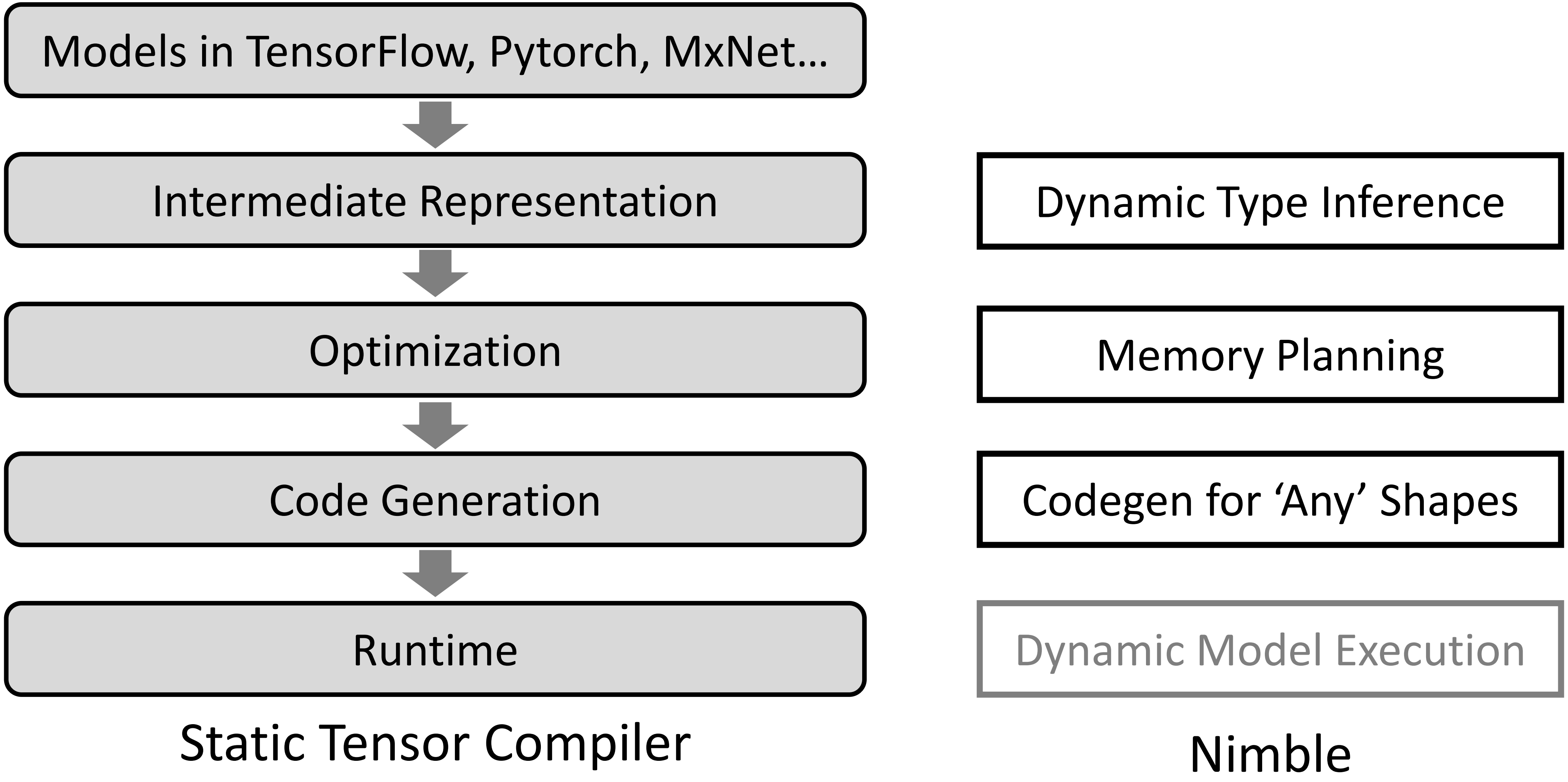
- Background
- Nimble
- DISC
- DietCode
- Summary

NIMBLE: EFFICIENTLY COMPILING DYNAMIC NEURAL NETWORKS FOR MODEL INFERENCE

Haichen Shen[*1] Jared Roesch[*2] Zhi Chen[1] Wei Chen[1] Yong Wu[1]
Mu Li[1] Vin Sharma[1] Zachary Tatlock[3] Yida Wang[1]

https://proceedings.mlsys.org/paper/2021/file/4e732ced3463d06de0ca9a15b6153677-Paper.pdf

Nimble: Overview



https://mlsys.org/virtual/2021/oral/1588

Nimble: Dynamic Type Inference

- Symbolic shape:
  - Tensor[(1, 10, Any), float32]
- New propagation rules:
  - broadcast rel(Any, 1) → Any
  - broadcast rel(Any, d) → d
  - broadcast rel(Any, Any) → Any

Nimble: Memory Planning

- Static compilers: implicit memory planning
- Dynamic shape needs explicit memory planning

```
fn (x: Tensor<?, 2>, y: Tensor<1, 2>)
     ->Tensor<?, 2> {
  let in_sh0 = shape_of(x);
  let in_sh1 = shape_of(y);  // shape calculating function
  let buf0 = alloc_storage(16, 64, ...);
  let out_sh0 = alloc_tensor(buf0, ...);  // allocate shape buffer & tensor
  invoke_shape_func(concat, // invoke shape function and get actual shape
       (in_sh0, in_sh1), (out_sh0,), ...);
------------------------------------------------------------ Host
------------------------------------------------------------ Device
  let buf1 = alloc_storage(...);
  let out0 = alloc_tensor(
       buf1, out_sh0, ...);  // allocate data buffer & tensor
  invoke_mut(concat, (x, y), (out0)); // invoke concat op
  out_0
}
```

Nimble: Codegen for ‘Any’ Shapes

- 3 Step to Codegen:
  - Any → 64, 128,… (Tensor[(1, 10, Any), float32])
  - Tune with static shapes, then pick TopK(100) results to test on other shapes
  - Pick the config: run best on other shapes
- Avoiding Boundary Check: duplicate kernels
  - Kernel tile = 8, then prepare 8 version kernels.
  - Automatically generate dispatch function.

Comments for Nimble

- End-to-End Dynamic Workload Compiler
  - New IR and primitive is necessary
- Compilation Overhead?
- Dynamic kernel performance?

Outline

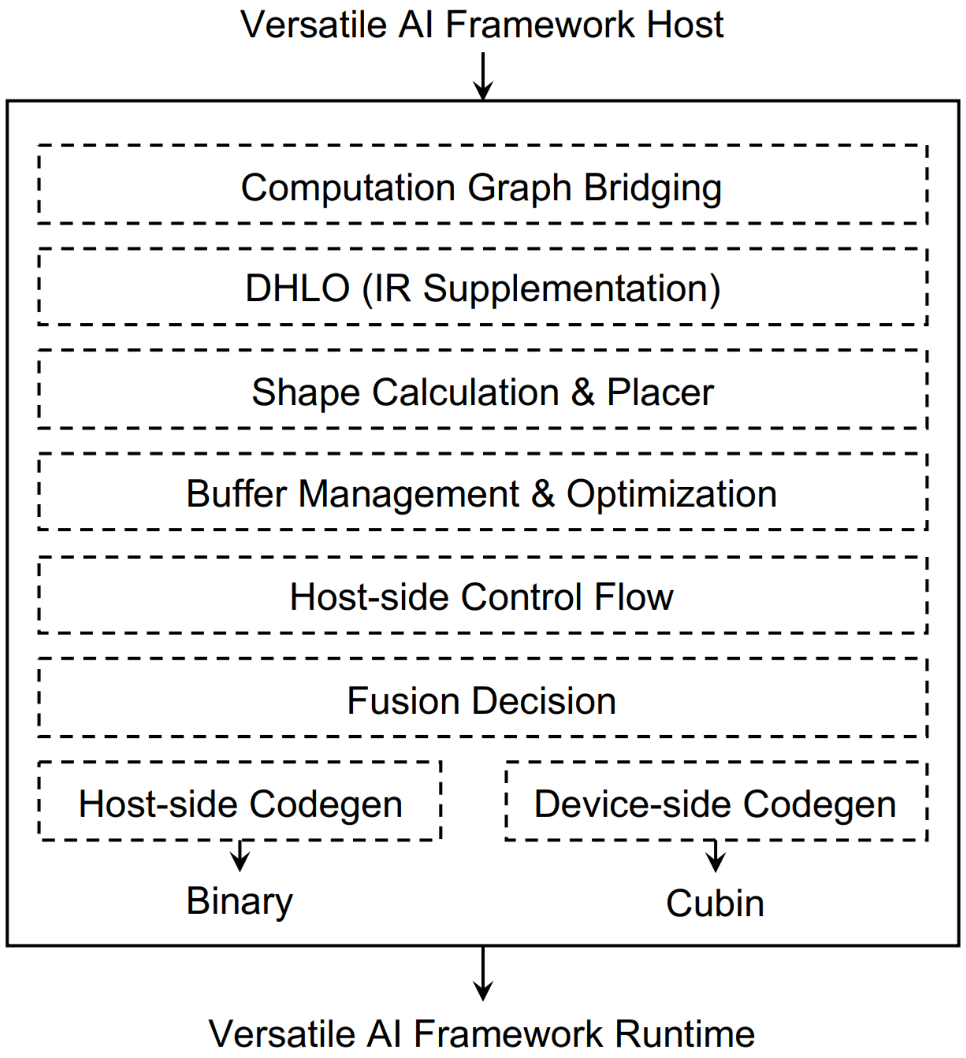
- Background
- Nimble
- DISC
- DietCode
- Summary

DISC **: A Dynamic Shape Compiler for Machine Learning Workloads**

Kai Zhu, Wenyi Zhao, Zhen Zheng, Tianyou Guo, Pengzhan Zhao, Feiwen Zhu
Junjie Bai, Jun Yang, Xiaoyong Liu, Lansong Diao, Wei Lin
Alibaba Group
{tashuang.zk, kevin.zwy, james.zz, tianyou.gty, pengzhan.zpz, feiwen.zfw
j.bai, muzhuo.yj, xiaoyong.liu, lansong.dls, weilin.lw}@alibaba-inc.com

https://arxiv.org/pdf/2103.05288

DISC Overview



- Based on MLIR
- Focus on Fusion of memory-bound OPs

DISC: Fusion of Dynamic Kernels

- Fusion requires same shape
- Shape Propagation
  - ADD op: input tensor shape = output tensor shape
- Shape Constraints
  - Dimension equality: reduction dim
  - Tensor size equality: transpose op

DISC: Which Kernel to Choose?

- Two Choices:
  - Choose 'Best' kernel from vendor libaries
  - Hand-tuned kernel for each shape
- Comments:
  - The offline workload is still huge for new ops & devices
  - Hand-tuned kernel is not a clear & general method

Outline

- Background
- Nimble
- DISC
- DietCode
- Summary

DIETCODE: AUTOMATIC OPTIMIZATION FOR DYNAMIC TENSOR PROGRAMS

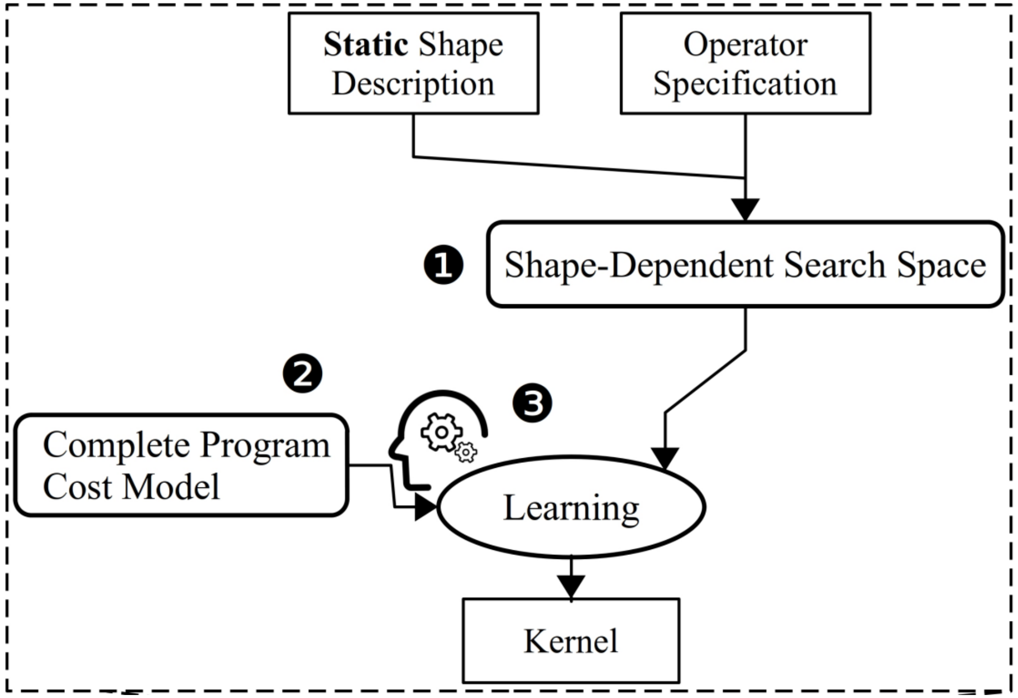
Bojian Zheng [*,1,2,3] **Ziheng Jiang** [*,4] **Cody Hao Yu** [2] **Haichen Shen** [5] **Josh Fromm** [6]
Yizhi Liu [2] **Yida Wang** [2] **Luis Ceze** [7,6] **Tianqi Chen** [8,6] **Gennady Pekhimenko** [1,2,3]

https://proceedings.mlsys.org/paper/2022/file/fa7cdfad1a5aaf8370ebeda47a1ff1c3-Paper.pdf

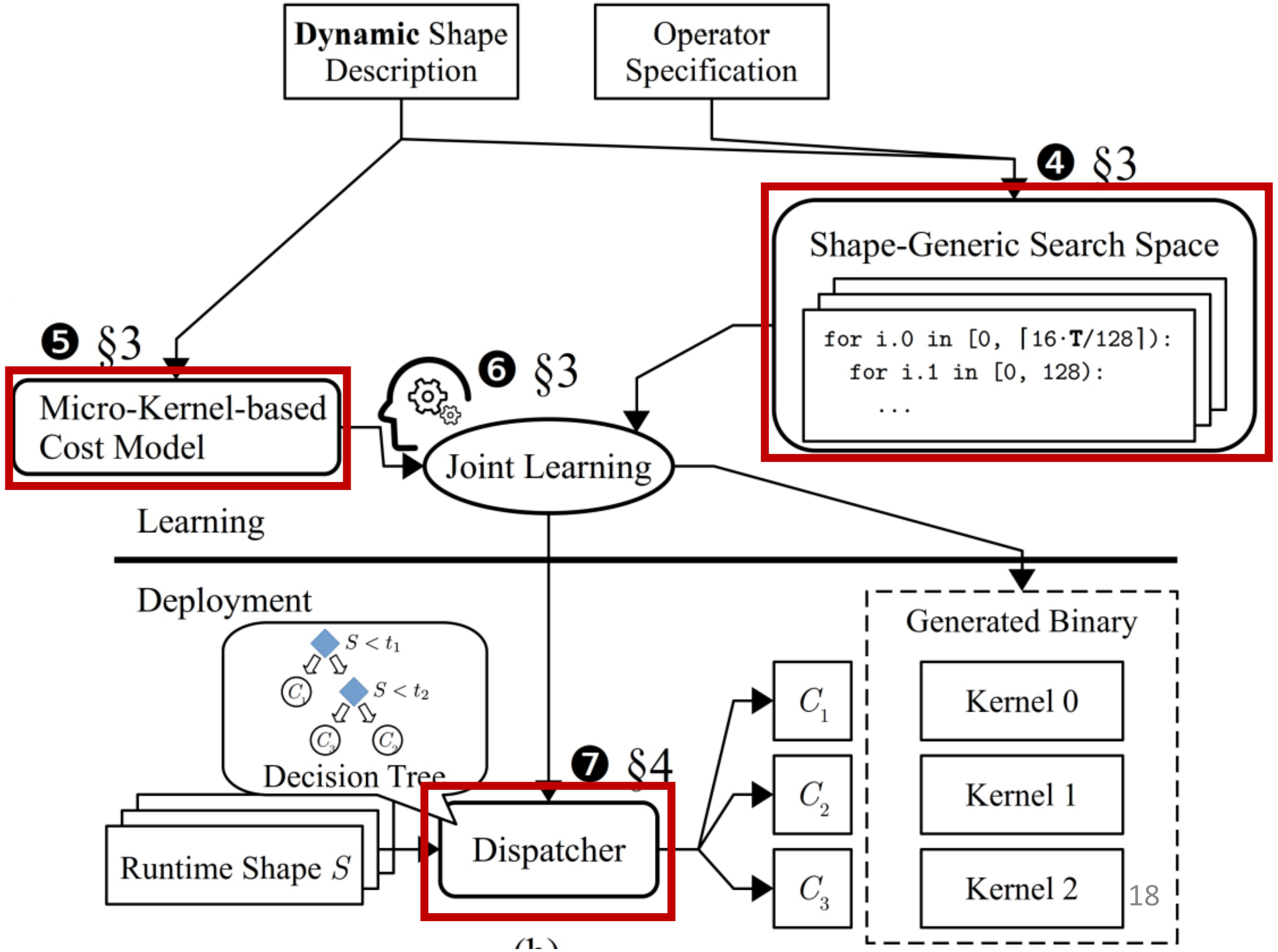
DietCode: Overview

- Codegen for a batch of shapes.
- Codebase: Ansor(TVM)

$f(op, shape, hw) = Schedule$



$f(op, shapes, hw) = Schedules, Dispatcher$



Dynamic Shape Description

- Dynamic shape interface in TVM

```python
def test_train_dynT():
    B = 16
    T = list(range(5, 128, 19))
    T.append(128)        ArmageddonKnight, 6个月前 • Initial commit
    I = 768
    H = 2304

    wkl_insts = cross_product(T, (I, H))
    logger.info("Workload Instances: {}".format(wkl_insts))

    DynT, DynI, DynH = tir.DynShapeVar('T'), tir.DynShapeVar('I'), tir.DynShapeVar('H')

    auto_scheduler.train(wkl_func=Dense,
                         wkl_func_args=(B * DynT, DynI, DynH),
                         shape_vars=[DynT, DynI, DynH], wkl_insts=wkl_insts,
                         wkl_inst_weights=[1. for _ in wkl_insts],
                         fcublas_fixture=cuBLASDenseFixture,
                         sched_func_name_prefix='dense_{}xTx{}x{}'.format(B, I, H)
                         )
```

DietCode: Search Space for a Batch Shape

- Full Search space: $t \in [2, +\infty)$
- Ansor Search space: $t \in \{2,5,10,25\}$
- DietCode: Any $t < T$ is valid.
  - Bring branch overhead 😥

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] = ...
  }
}
```

https://mlsys.org/media/mlsys-2022/Slides/2175.pdf

How to Reduce Branch Overhead?

Initial Code

```
for i.0 in [0, T):
  for i.1 in [0, t):
    if i.0*t+i.1 < T:
      X_local = X[...]
    if i.0*t+i.1 < T:
      Y_local = ...
    if i.0*t+i.1 < T:
      Y[...] = Y_local
```

Loop Partition

```
for i.0 in [0, T):
  for i.1 in [0, t):
    if i.0 < ⌊T/t⌋:
      X_local = X[...]
      Y_local = ...
      Y[...] = Y_local
    else:
      if i.0*t+i.1 < T:
        X_local = X[...]
      if i.0*t+i.1 < T:
        Y_local = ...
      if i.0*t+i.1 < T:
        Y[...] = Y_local
```

no branch block

Global Padding

```
X_pad = pad X to [⌈T/t⌉ · t]
for i.0 in [0, T):
  for i.1 in [0, t):
    X_local = X_pad[...]
    Y_local = ...
    Y_pad[...] = Y_local
slice Y_pad to [T]
```

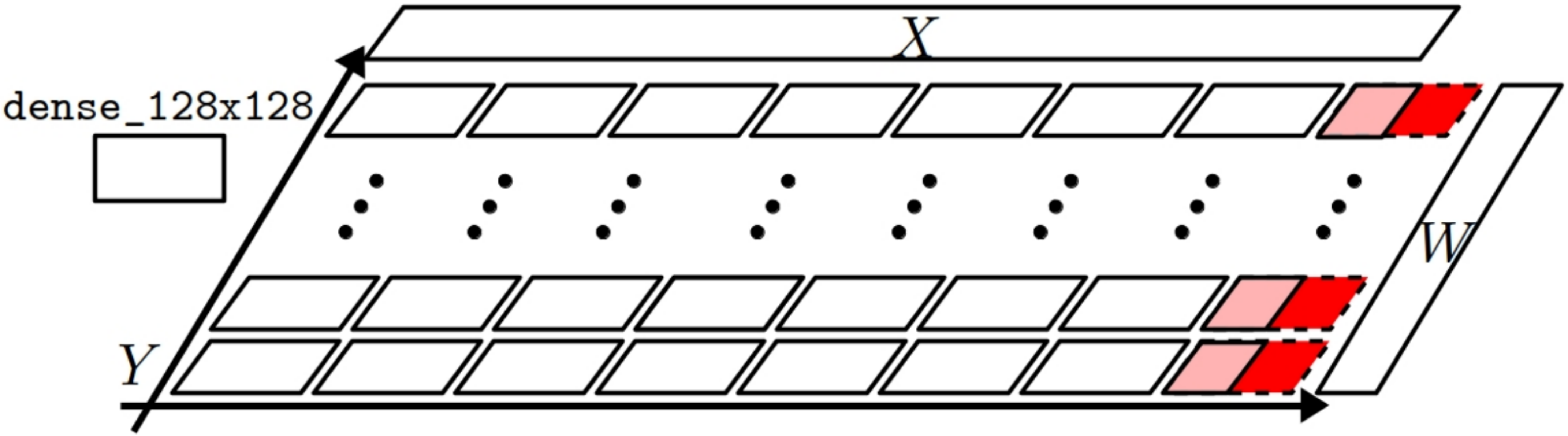
No branch

Local Padding

```
for i.0 in [0, T):
  for i.1 in [0, t):
    if i.0*t+i.1 < T:
      X_local = X[...]
    Y_local = ...
    if i.0*t+i.1 < T:
      Y[...] = Y_local
```

No branch Compute

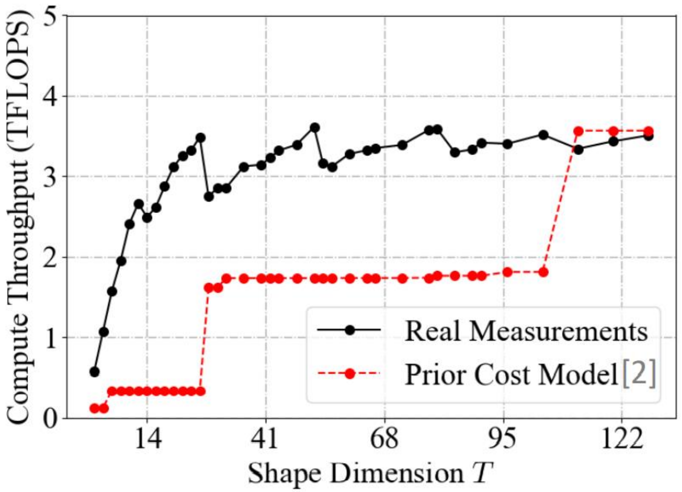
Micro-Kernel in DietCode

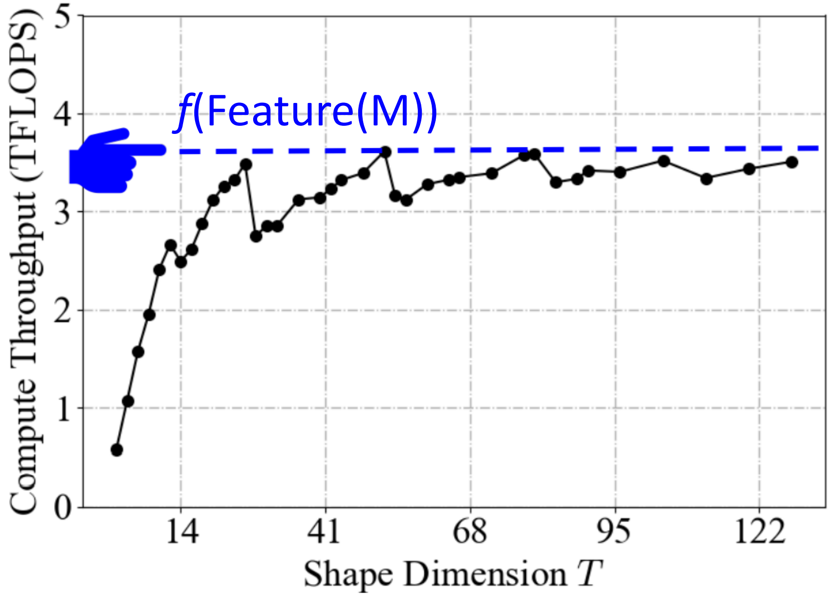


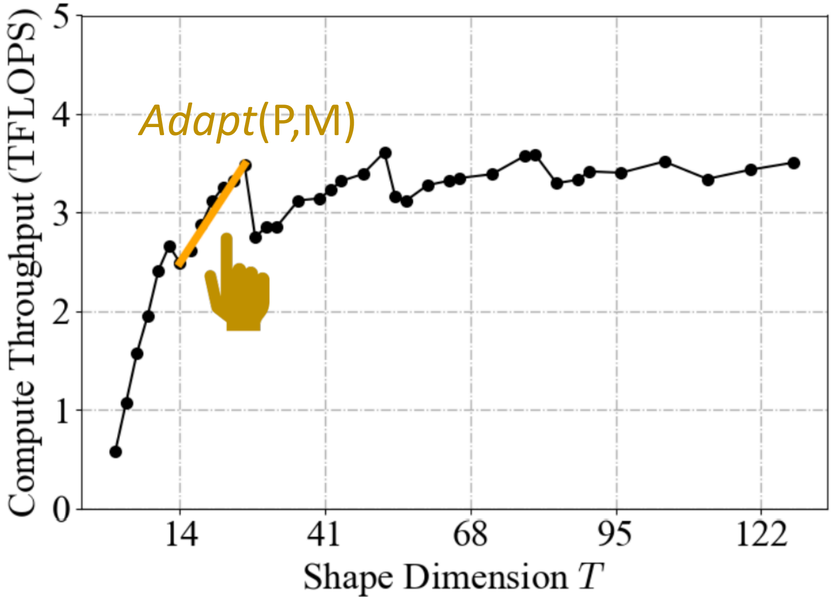
DietCode Cost Model: Micro-Kernel Based

- $Y = XW^T, X: [16 \times T, 768], W = [2304,768]$
- Ansor Cost Model: Cost(P) = *f*(Feature(P))
- DietCode Cost Model:

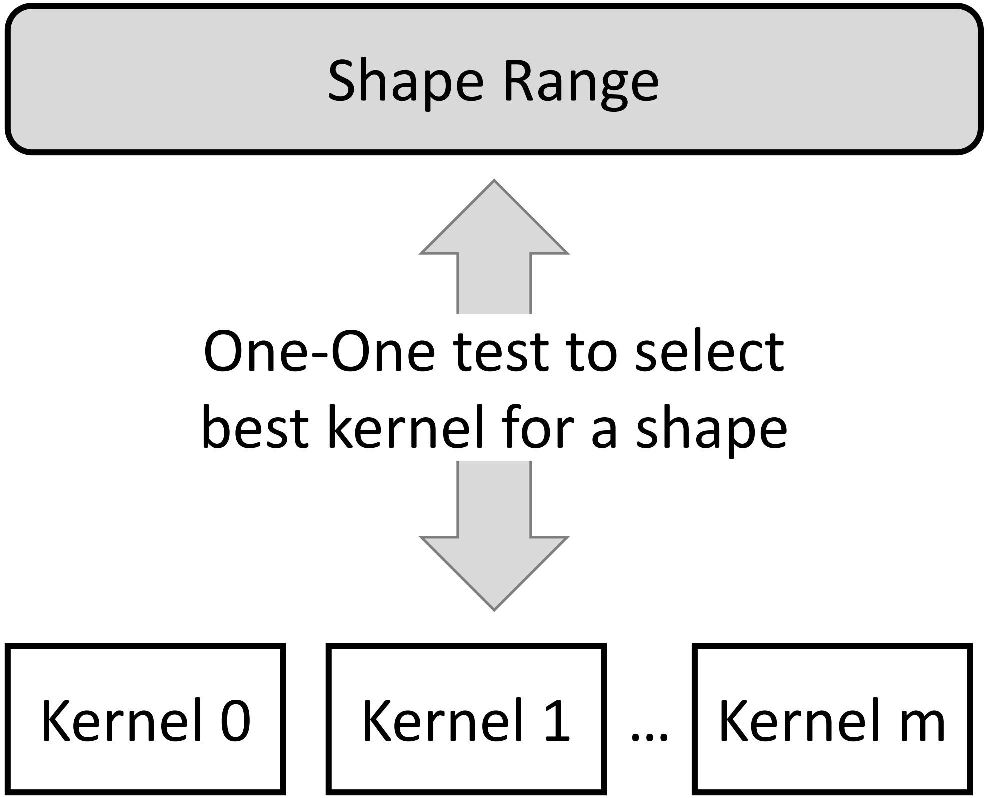
  Cost(P)=*f*(Feature(M))×*adapt*(P,M)







Dispatcher of DietCode



- DietCode could generate a batch of micro-kernels.
- Dispatcher: Decision Tree(?)

Summary

	Nimble	**DISC**	**DietCode**
End-To-End	Yes	Yes	No(TVM)
Backend	Virtual Machine	No VM	TVM
Dynamic Kernel	Off-line Tuning	Hand-tuned kernels + vendor	Search space + Cost Model
Dispatcher	Choose one kernel, duplicate, apply to all	?(maybe hardcode)	Decision Tree

Thanks!