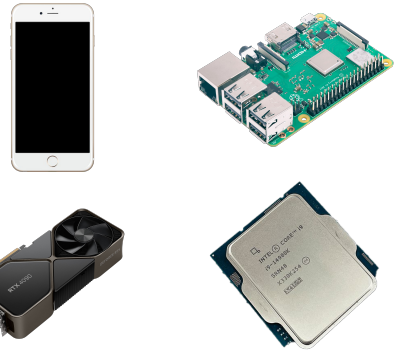
TVM: An Automated End-to-End Optimizing Compiler for Deep Learning

Group 6: Abhinav Tondapu & Hang Yeung
November 10, 2025

Problem: Deploying ML Models

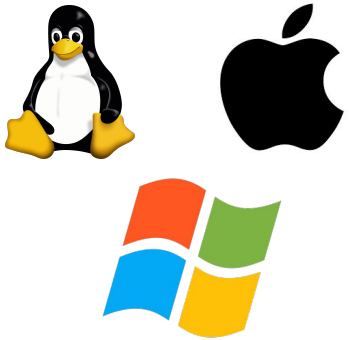
1

Device?



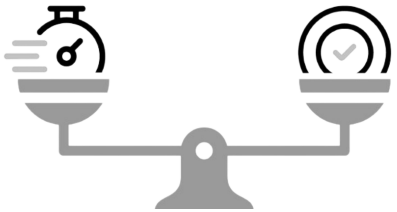
2

Platform?



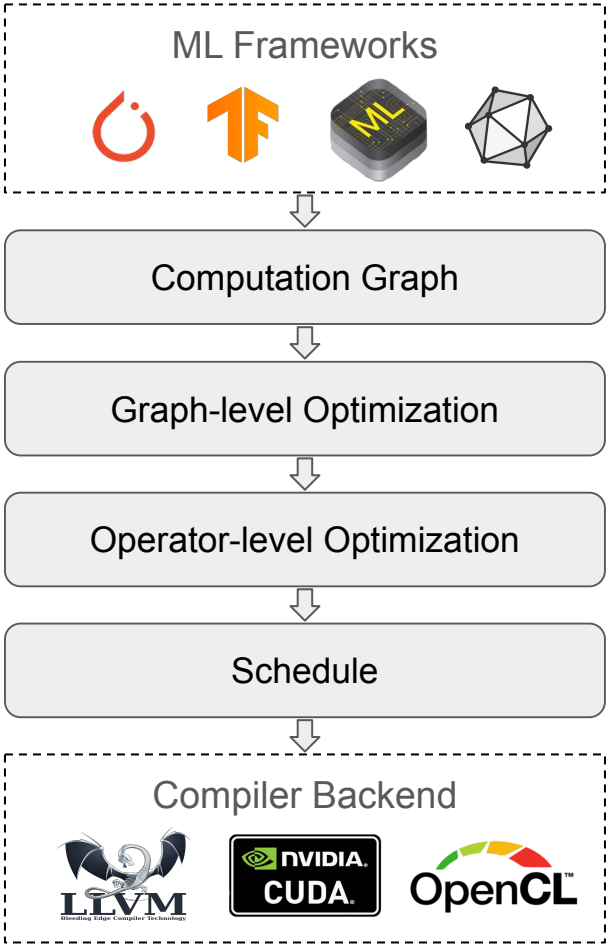
3

Fast vs accurate?



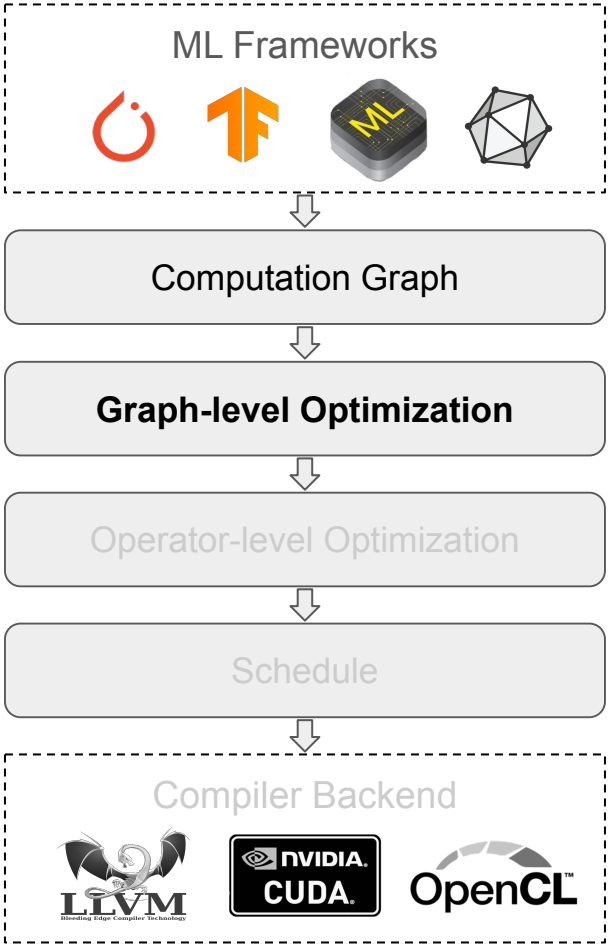
TVM Architecture

- Graph-level optimization
  - Operator fusion
  - Data layout
- Operator-level optimization
  - Tensorization
  - Nested parallelism
  - Latency hiding
- Schedule Search
  - ML Cost Model

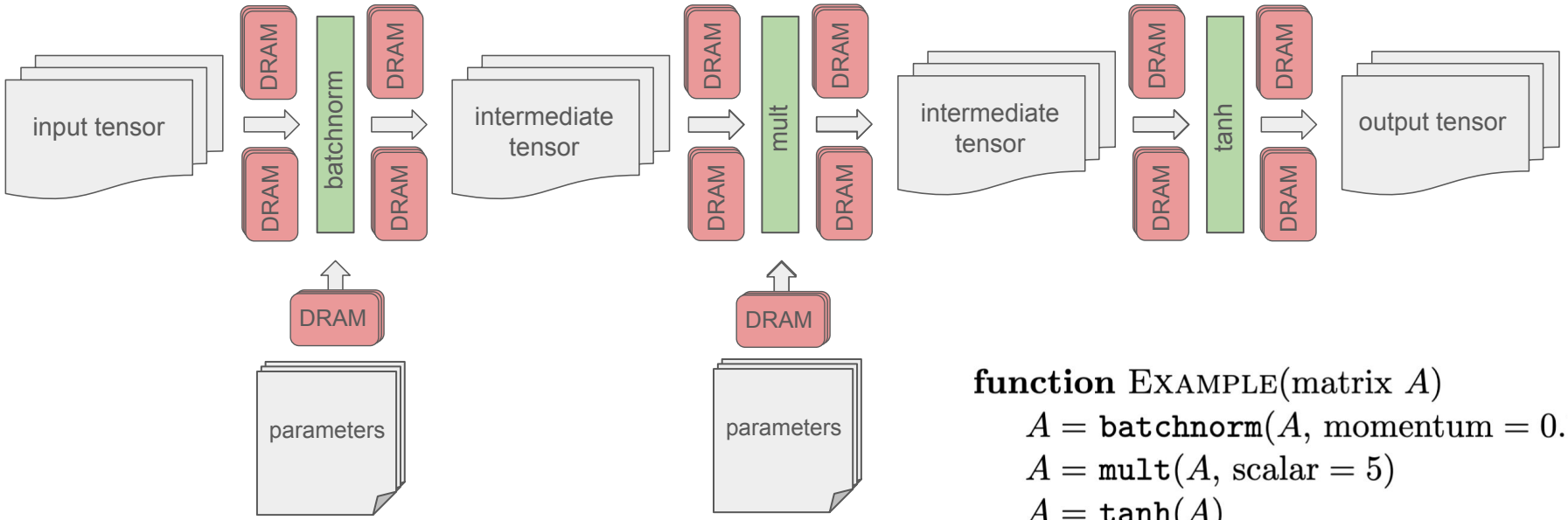


TVM Architecture

- **Graph-level optimization**
  - Operator fusion
  - Data layout
- Operator-level optimization
  - Tensorization
  - Nested parallelism
  - Latency hiding
- Schedule Search
  - ML Cost Model

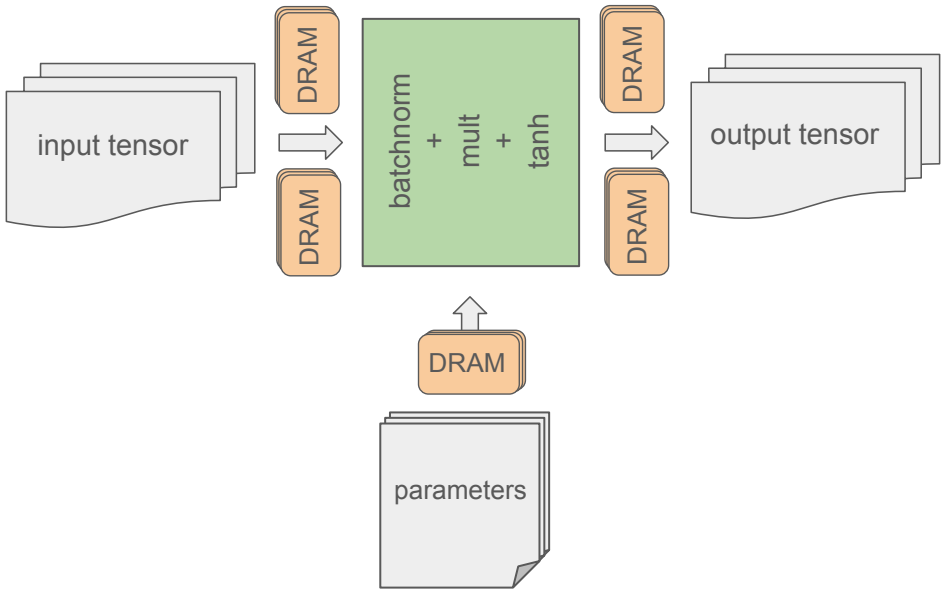


Graph-level Optimization - Operator Fusion



function EXAMPLE(matrix A)
$A = \texttt{batchnorm}(A, \text{momentum} = 0.1)$
$A = \texttt{mult}(A, \text{scalar} = 5)$
$A = \texttt{tanh}(A)$
return A

Graph-level Optimization - Operator Fusion (*con.*)



- Minimize DRAM access
  - ~ proportional to layer count
- Limitations
  - Shape mismatch
  - Register pressure (regression)
  - Stateful/random operators
- Manual fused implementations?
  - TVM code generation, on demand

Graph-level Optimization - Data Layout

$$\texttt{AvgPool2d}(\text{matrix } A, \text{ size} = 2, \text{ stride} = 2)$$

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \Rightarrow \begin{pmatrix} 3.5 & 5.5 \\ 11.75 & 13.5 \end{pmatrix}$$

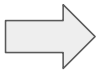
Simple in math but…

Strided access is slow!

Graph-level Optimization - Data Layout *(con.)*

Reorganize the underlying memory

$$A' = \begin{pmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \\ 9 & 10 & 13 & 15 \\ 11 & 12 & 15 & 16 \end{pmatrix}$$



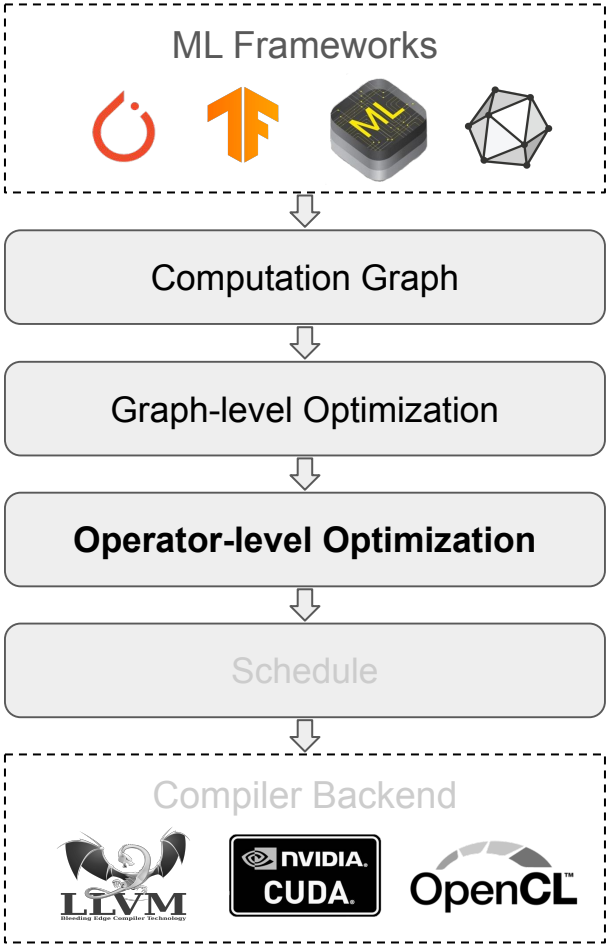
$$\begin{pmatrix} 3.5 & 5.5 \\ 11.75 & 13.5 \end{pmatrix}$$

Much better cache performance

- Reduce memory latency
  - Coalesce memory access
- Limitations
  - Operators want different layouts
  - Producer-consumer complexity
- Write data layout?
  - TVM code generation on demand

TVM Architecture

- Graph-level optimization
  - Operator fusion
  - Data layout
- **Operator-level optimization**
  - Tensorization
  - Nested parallelism
  - Latency hiding
- Schedule Search
  - ML Cost Model



Operator-level Optimization - Tensorization

```
for y, x, k in grid(64, 64, 64):
  C[y, x] += A[y, k] * B[k, x]
```

```
for yo, xo, ko in grid(16, 16, 16):
  for y, x, k in grid(4, 4, 4):
    C[by*4+y, bx*4+x] +=
      A[by*4+y, bk*4+k] * B[bk*4+k, bx*4+x]
```

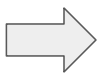
block

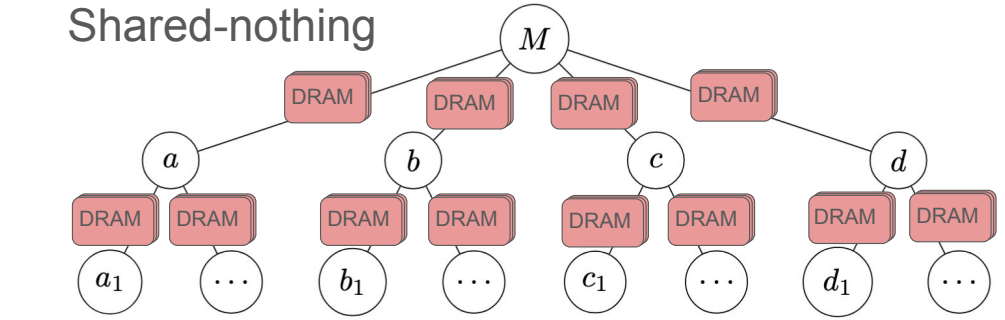
- Divide Into Tensors
  - DLP
- Portable abstraction
- Reduces search noise

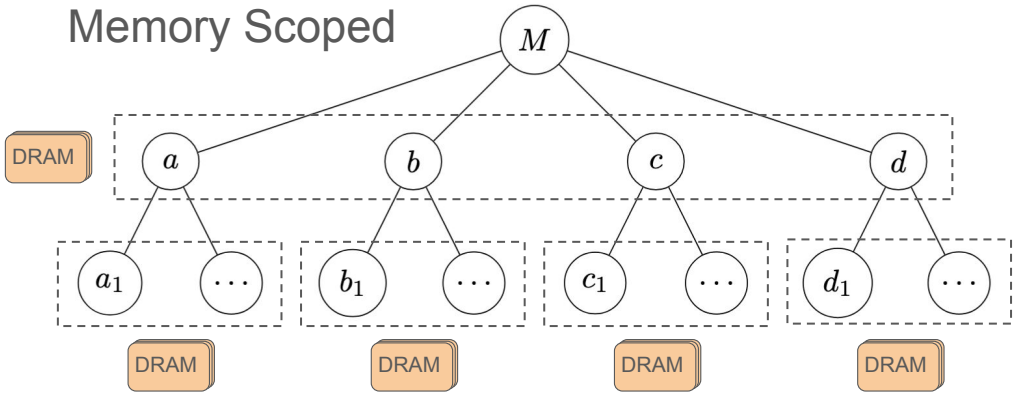
Operator-level Optimization - Nested Parallelism

Consider a parallelized operation

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$







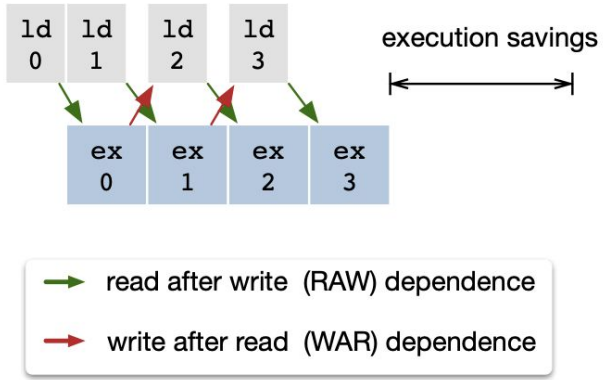
Operator-level Optimization - Latency Hiding

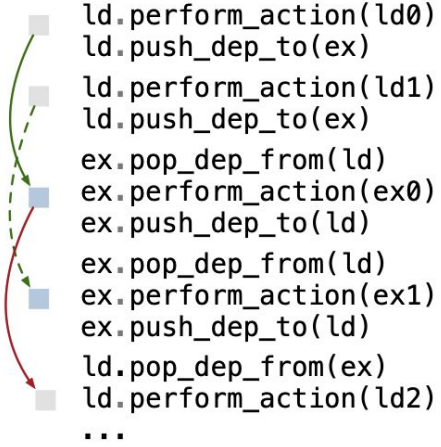
Monolithic Pipeline

Instruction Stream

```
ld.perform_action(ld0)
ex.perform_action(ex0)
ld.perform_action(ld1)
ex.perform_action(ex1)
...
```

Decoupled Access-Execute Pipeline

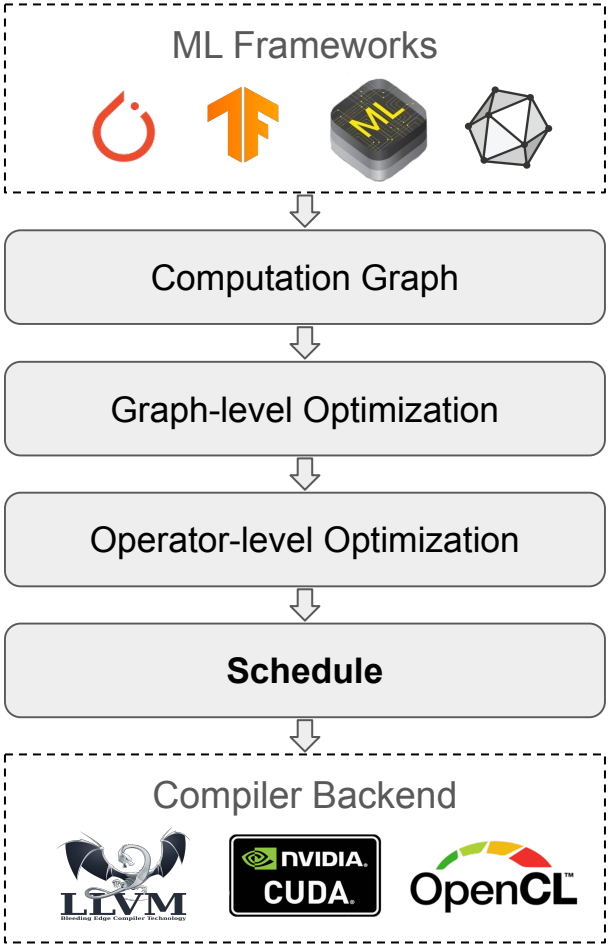




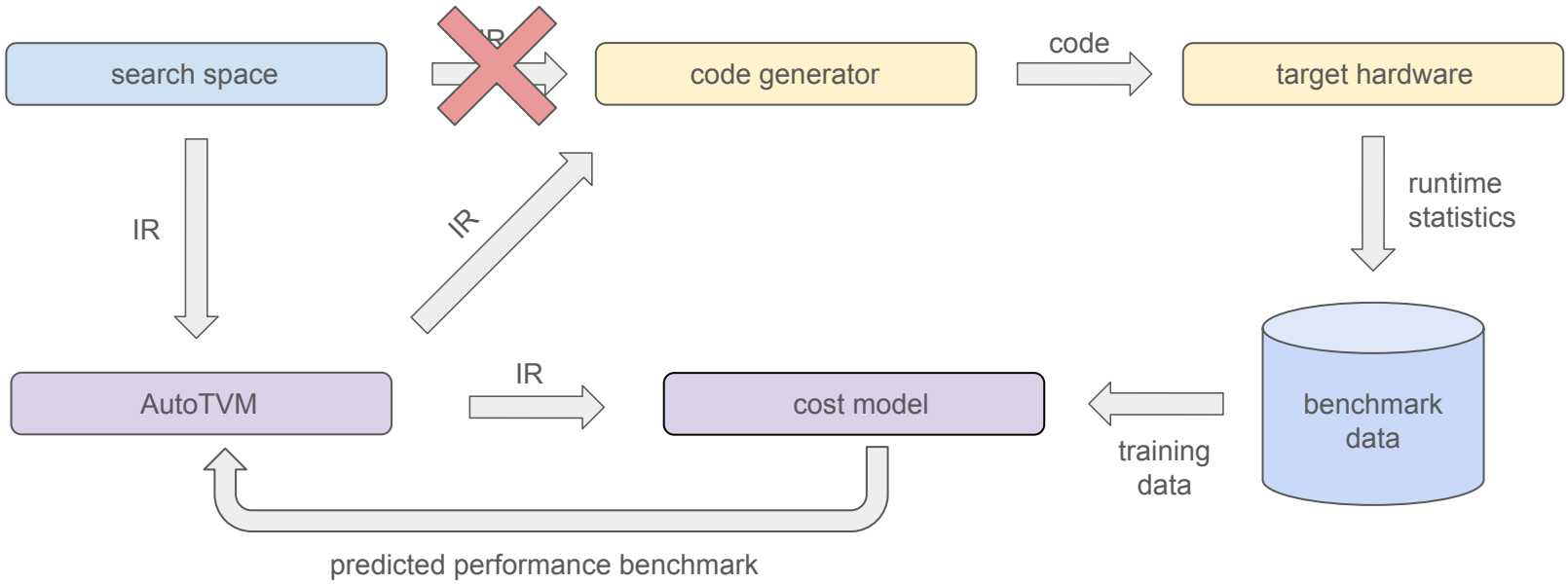
```
ld.perform_action(ld0)
ld.push_dep_to(ex)
ld.perform_action(ld1)
ld.push_dep_to(ex)
ex.pop_dep_from(ld)
ex.perform_action(ex0)
ex.push_dep_to(ld)
ex.pop_dep_from(ld)
ex.perform_action(ex1)
ex.push_dep_to(ld)
ld.pop_dep_from(ex)
ld.perform_action(ld2)
...
```

TVM Architecture

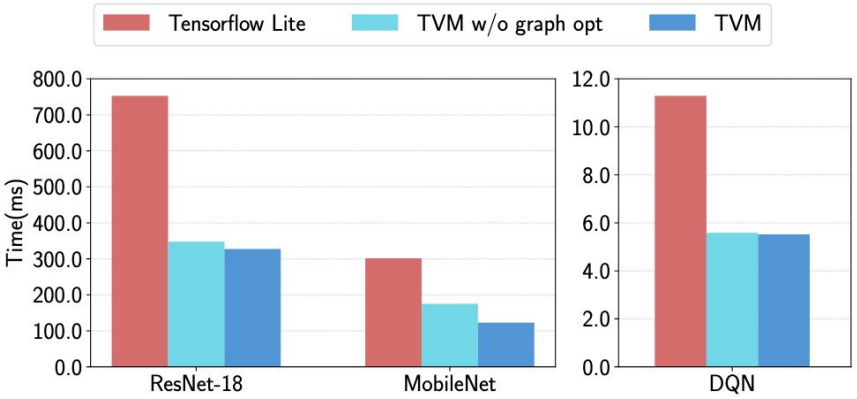
- Graph-level optimization
  - Operator fusion
  - Data layout
- Operator-level optimization
  - Tensorization
  - Nested parallelism
  - Latency hiding
- **Schedule Search**
  - ML Cost Model



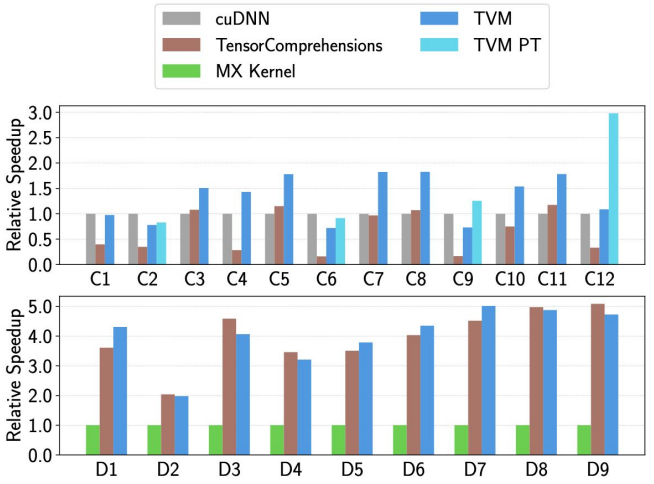
Schedule Search



Results



End-to-end latency
(Lower is better)



Conv2d speedup in ResNet-18
and MobileNet
(Higher is better)

Our thoughts

- Limited hardware introspection
- More thorough benchmarks
- Gap between cost model and tensor selection
- Hard to generalize cost model to different hardware