15-442/15-642: Machine Learning Systems

Machine Learning Compilation

Spring 2024

Tianqi Chen and Zhihao Jia
Carnegie Mellon University

Outline

Overview of Machine Learning Compilation

Example ML Compilation Flow

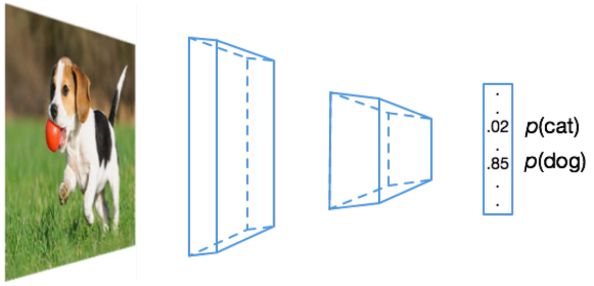
Outline

Overview of Machine Learning Compilation

Example ML Compilation Flow

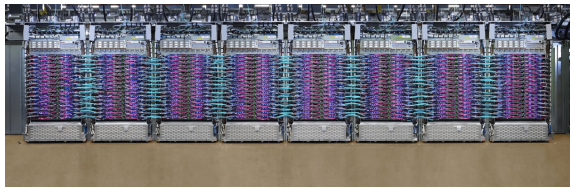
ML System Optimization Problem

ML Models



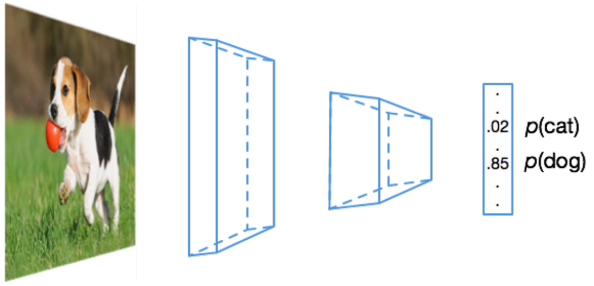
- Specialized libraries for each backend (labor intensive)
- Non-automatic optimizations

MKL-DNN | cuDNN | ARM-Compute | TPU Backends



Machine Learning Compilation

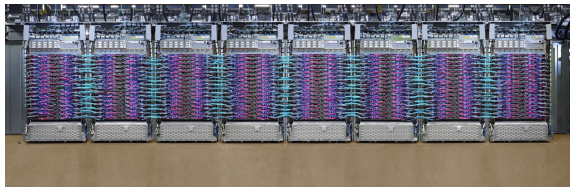
ML Models



High-level IR Optimizations and Transformations

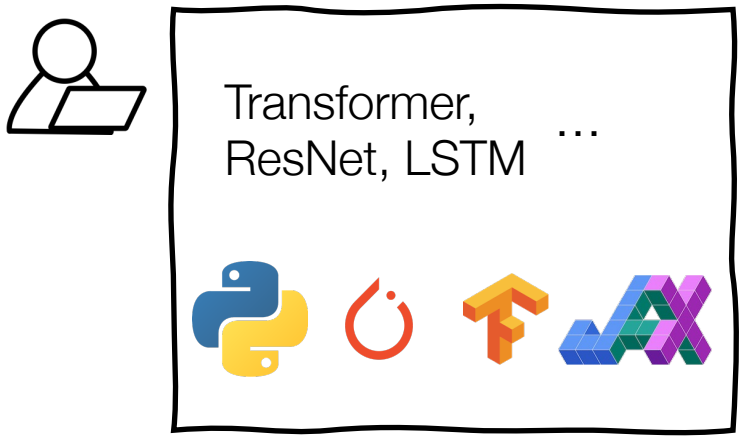
Tensor Operator Level Optimization

Direct code generation



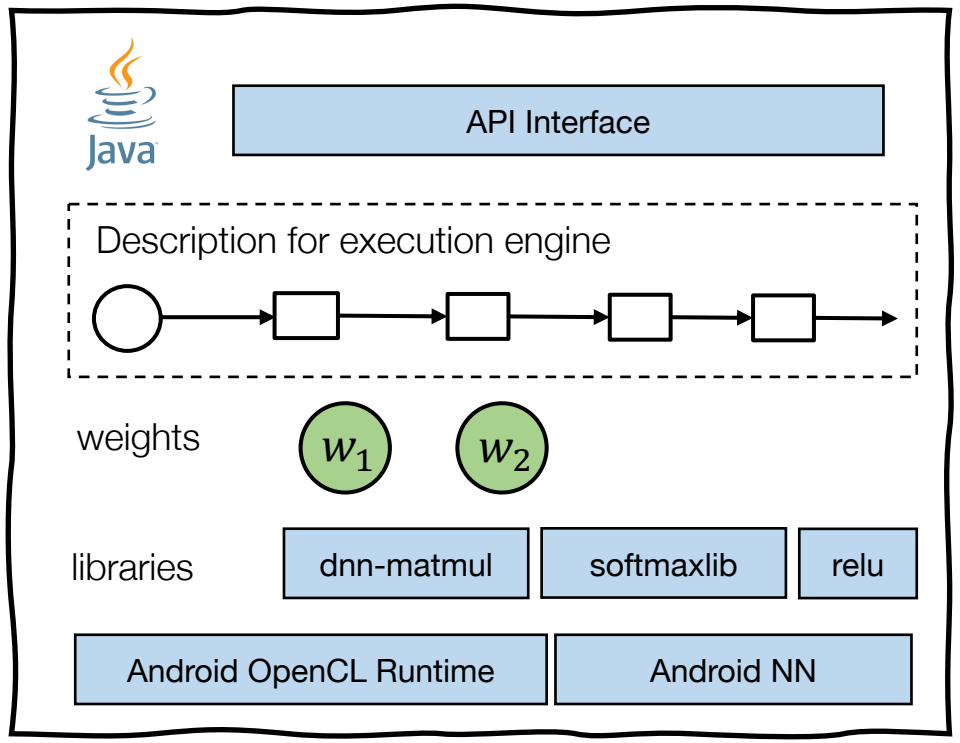
Machine Learning Compilation

Development Form



MLC Process

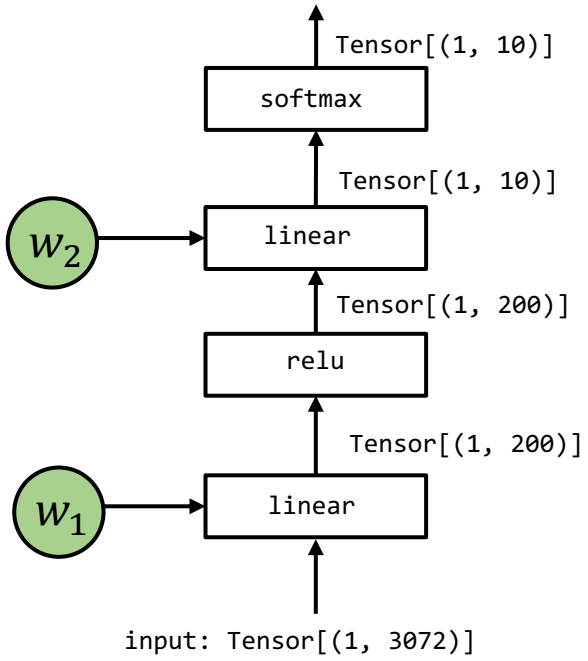
Deployment Form



Machine learning compilation (MLC) is the process to transform and optimize machine learning execution from its development form to deployment form.

An example instance of deployment form

Key Elements in Machine Learning Compilation



Tensor multi-dimensional array that stores the input, output and intermediate results of model executions.

Tensor Functions that encodes computations among the input/output. Note that a tensor function can contain multiple operations

ML Compilation Goals

There are many equivalent ways to run the same model execution. The common theme of MLC is optimization in different forms:

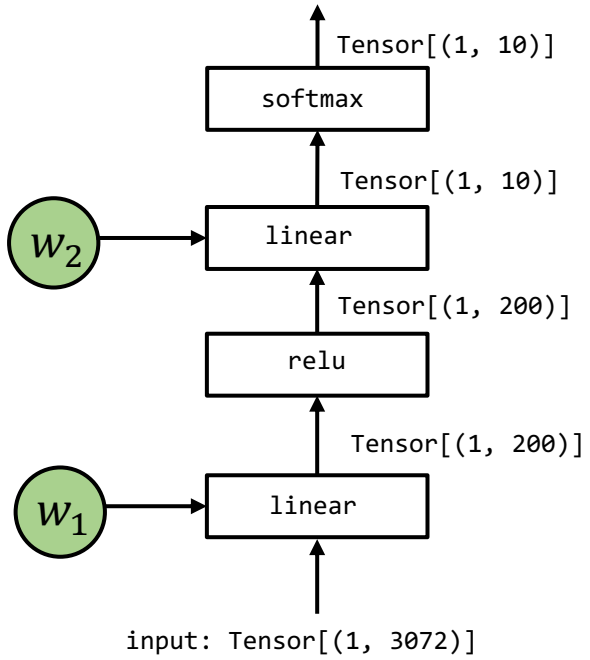
Minimize memory usage.

Improve execution efficiency.

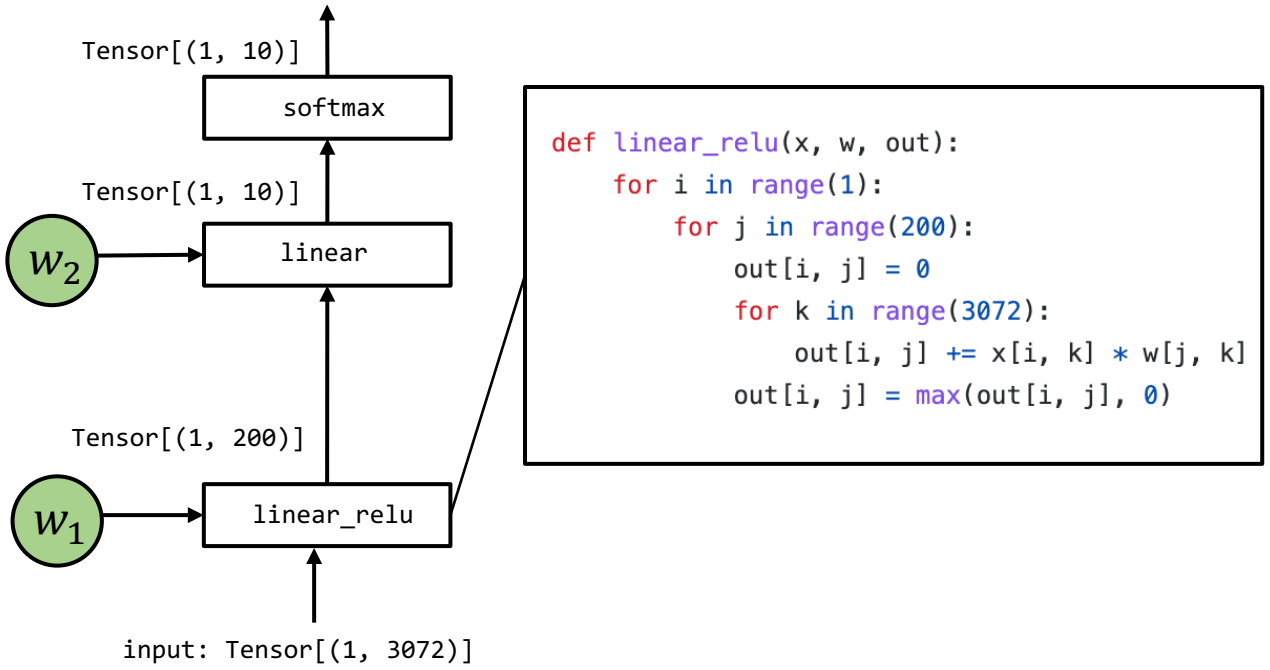
Scaling to multiple heterogeneous nodes.

Example Compilation Process

Development



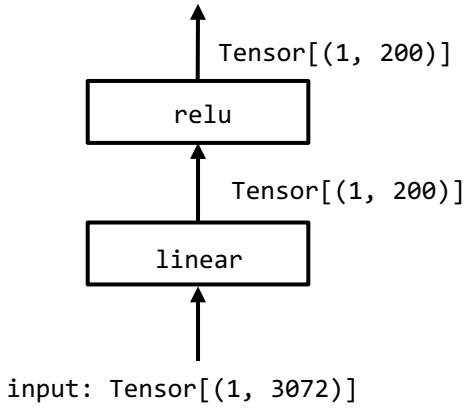
Deployment

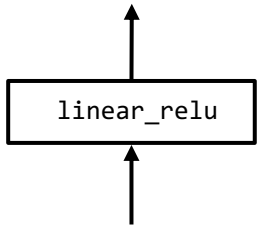


In this particular example, two tensor functions are folded into one (linear-relu). With a specialized implementation (in reality, they will be implemented using low-level primitives).

Abstraction and Implementation

Abstraction refers to different ways to represent the same system interface (tensor function)

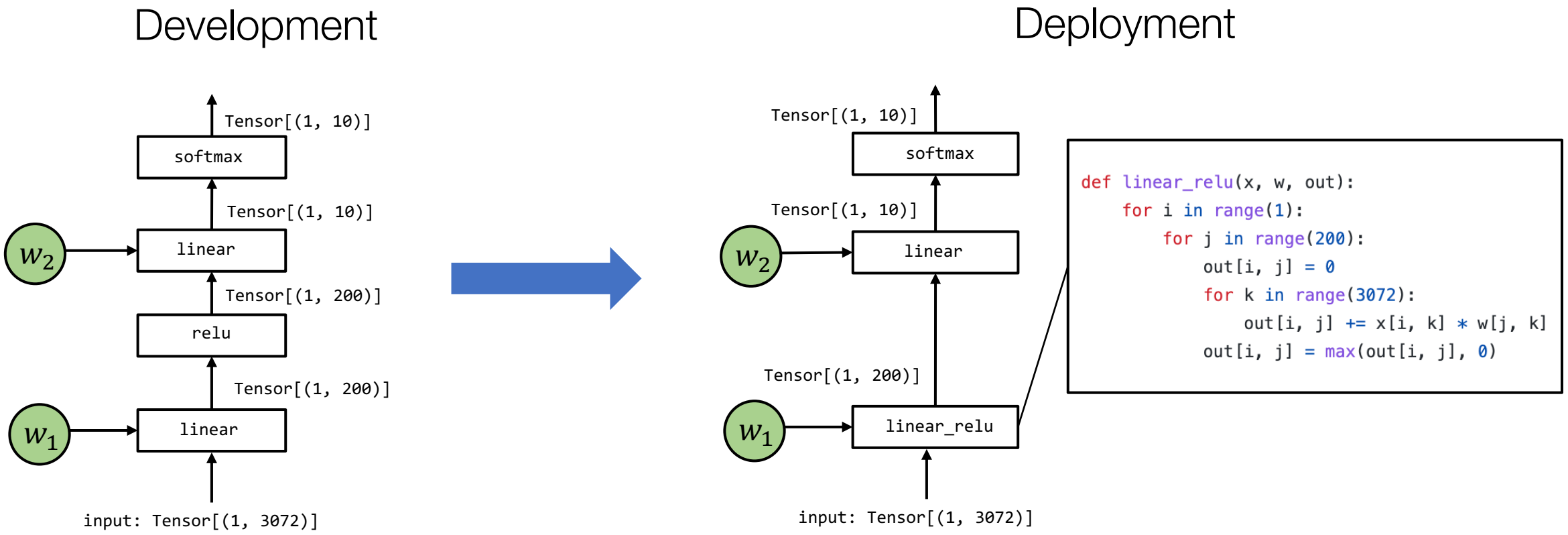





```
def linear_relu(x, w, out):
    for i in range(1):
        for j in range(200):
            out[i, j] = 0
            for k in range(3072):
                out[i, j] += x[i, k] * w[j, k]
            out[i, j] = max(out[i, j], 0)
```

Three abstraction ways to represent the same tensor function (linear_relu), each providing a different level of details. In practice, we usually say that the more specialized version is an **implementation** of higher-level abstraction.

MLC as Tensor Function Transformation (with different abstractions)



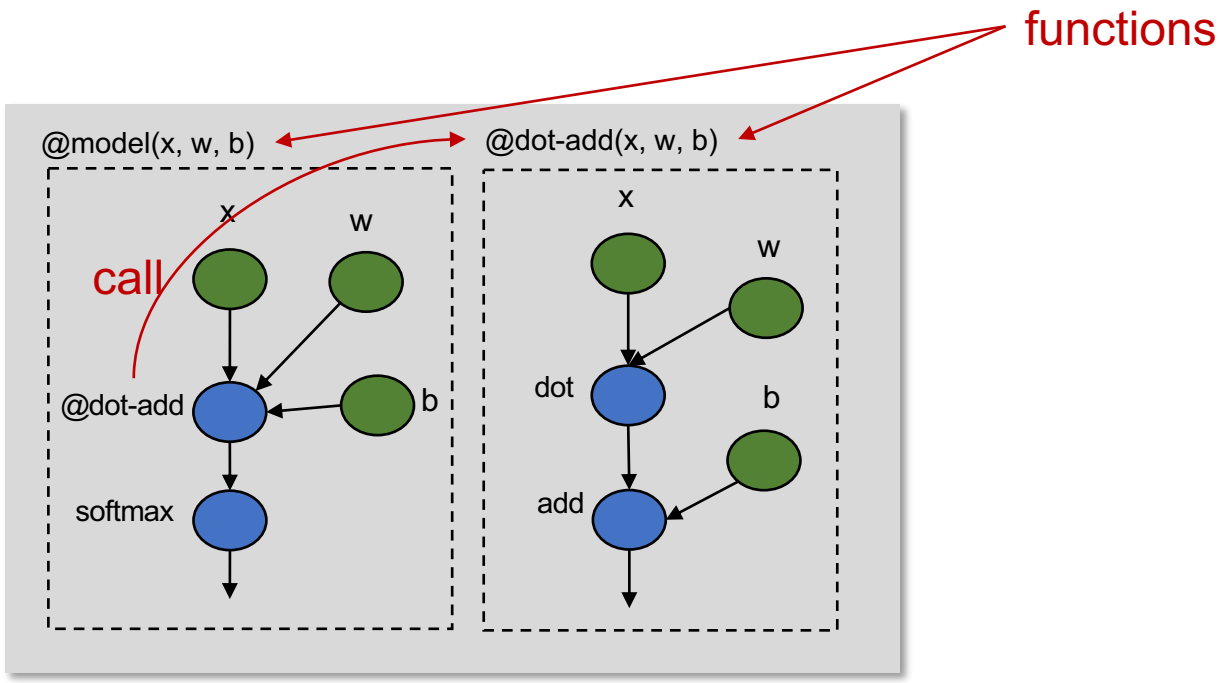
Most MLC process can be viewed as transformation among tensor functions (that can be represented with different abstractions).

Outline

Overview of Machine Learning Compilation

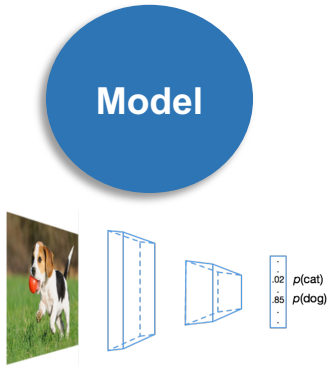
Example ML Compilation Flow

Compiler Representation of a ML Model

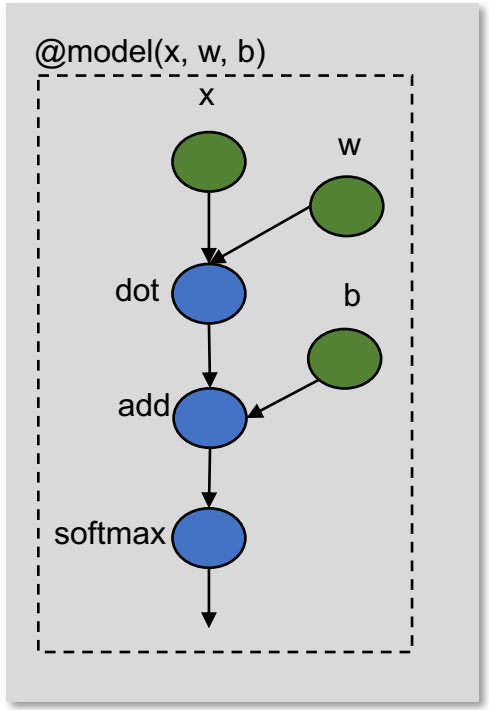


IRModule: a collection if interdependent functions

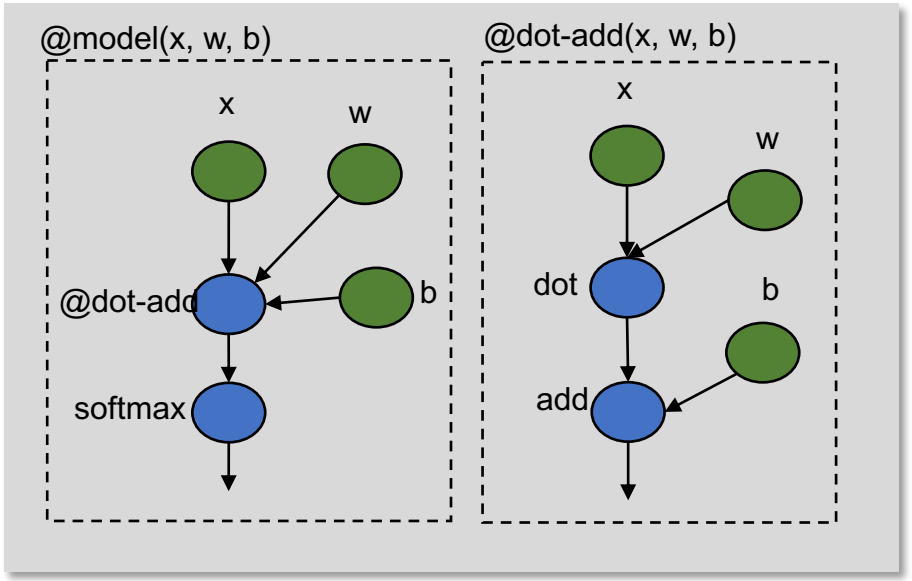
Example Compilation Flow: High-Level Transformations



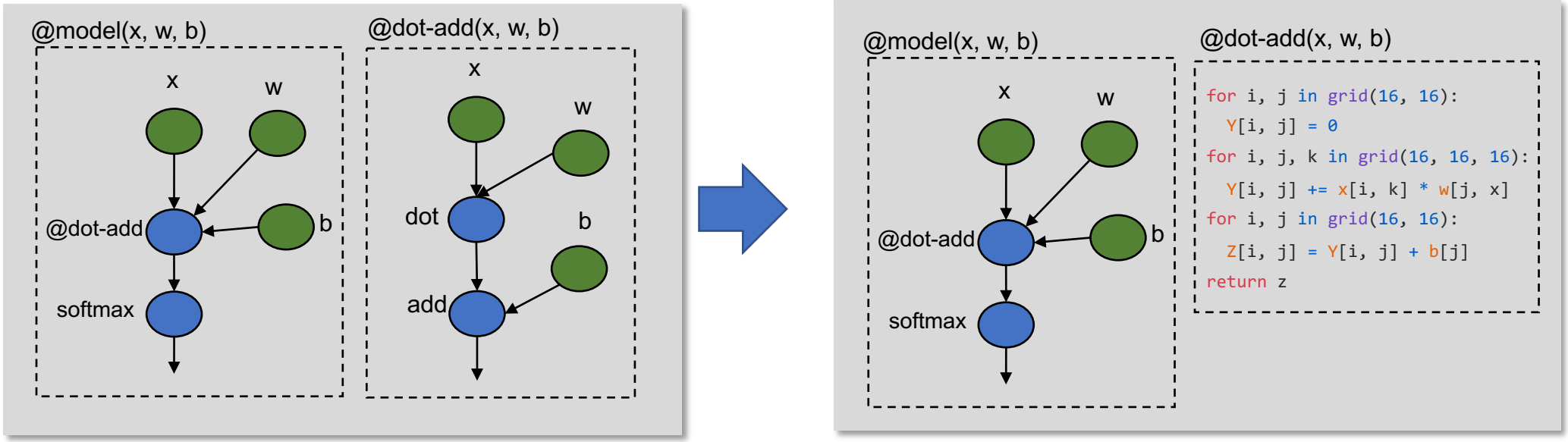
import



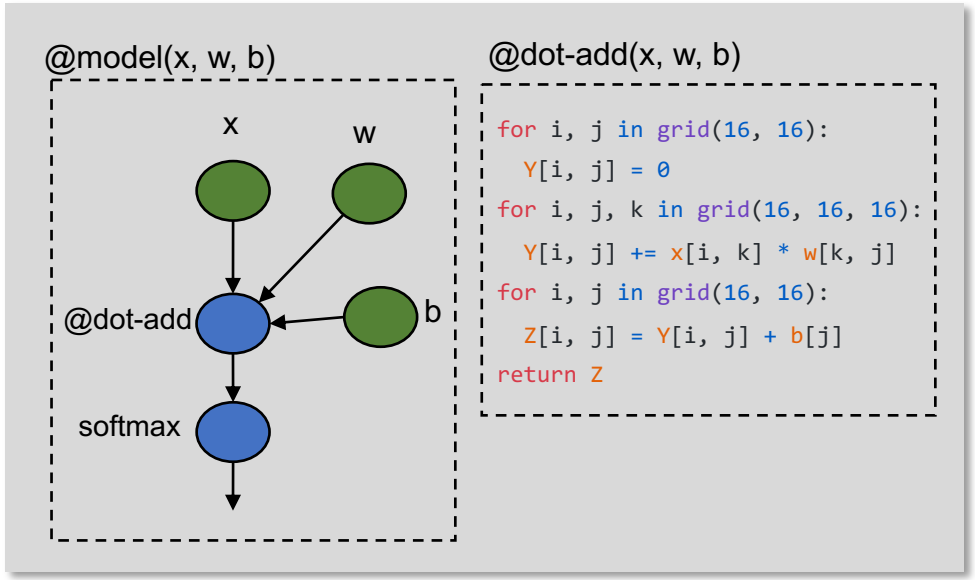
High-level transformations

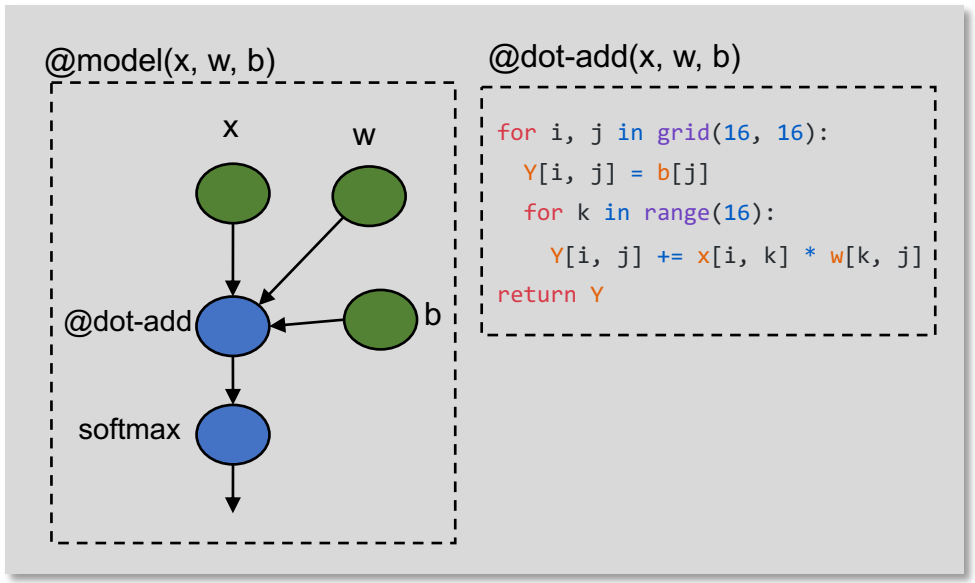


Example Compilation Flow: Lowering to Loop IR



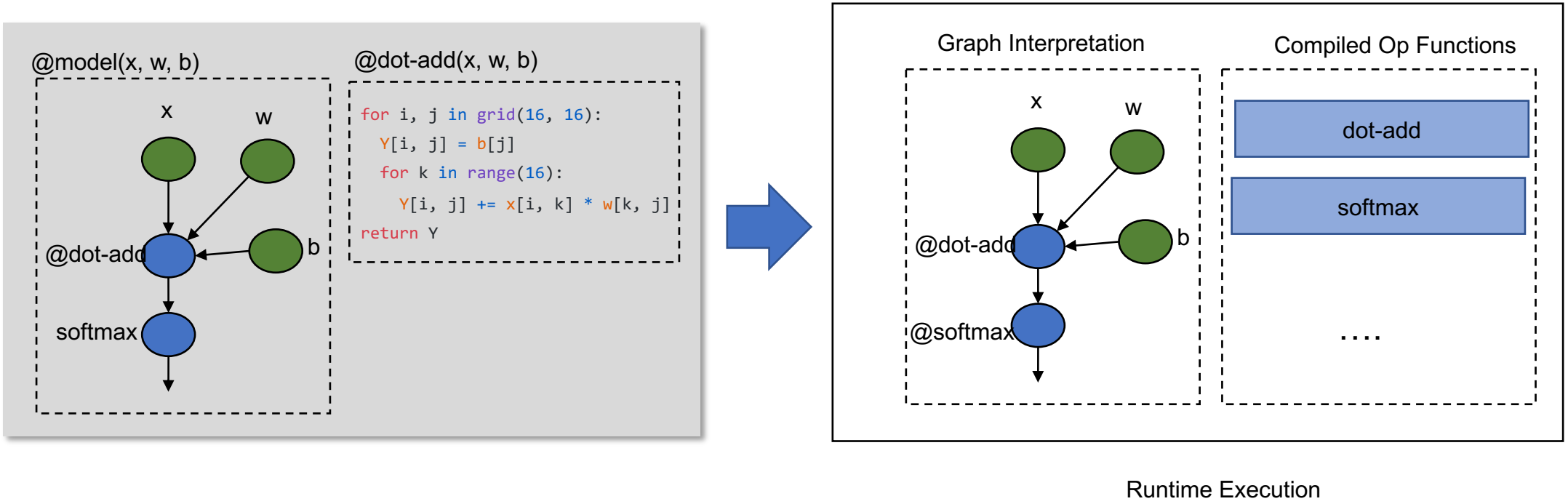
Example Compilation Flow: Low Level Transformations





Low-level transformations

Example Compilation Flow: CodeGen and Execution

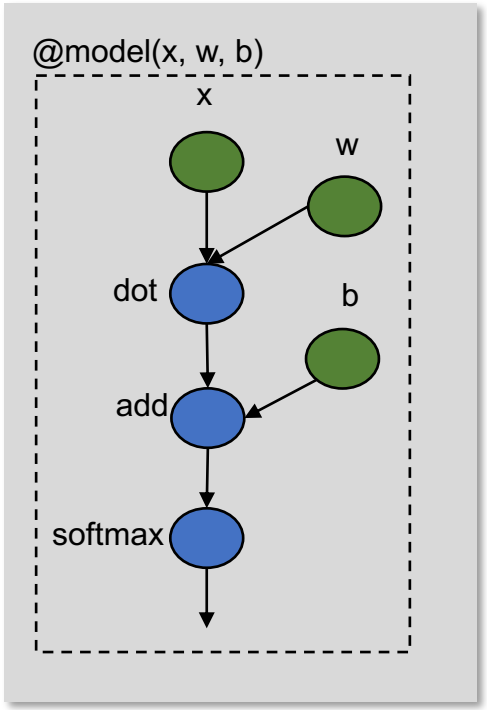


Runtime Execution

Discussion

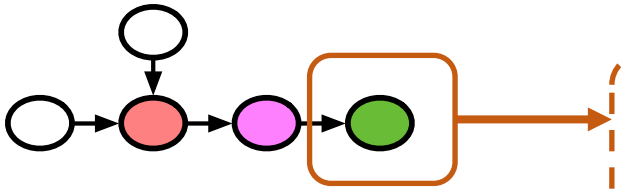
- What are possible ways to represent a function in ML
- The possible set of optimizations we can perform in each type of representations.

High-level IR and Optimizations



- Computation graph(or graph-like) representation
- Each node is a tensor operator(e.g. convolution)
- Can be transformed (e.g. fusion) and annotated (e.g. device placement)
- Most ML frameworks have this layer

Low-level Code Optimization



Specification

```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

Search Space of Possible Program Optimizations

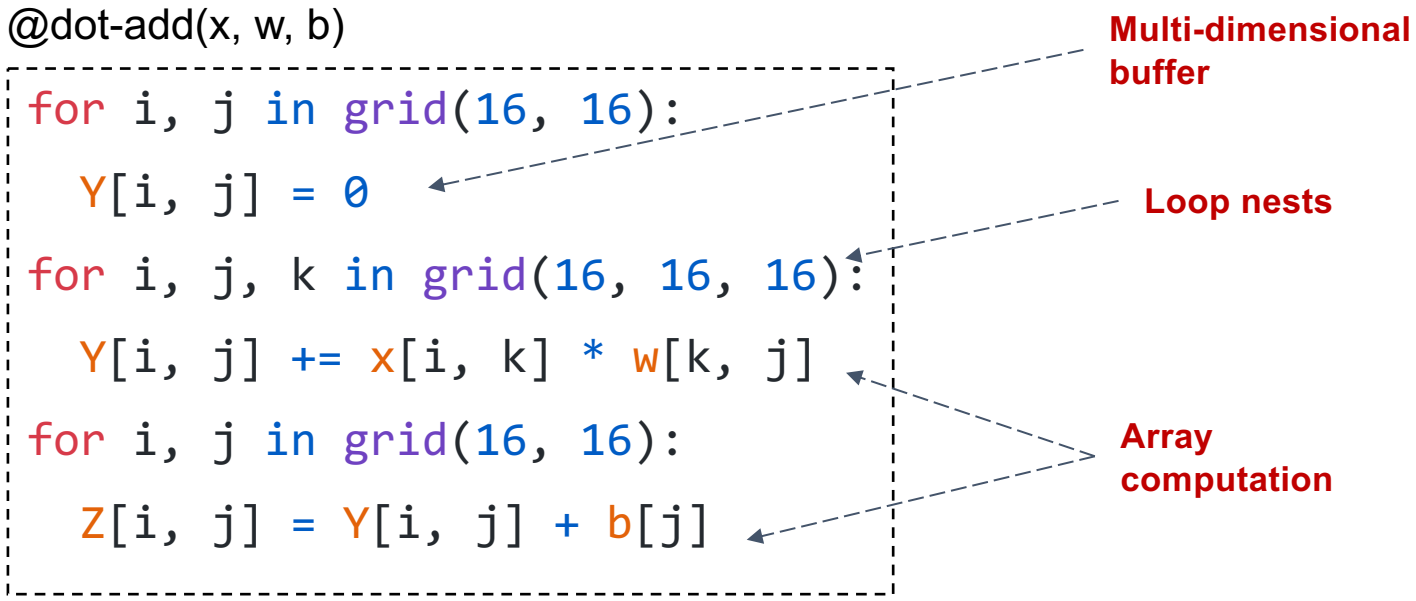
Low-level Program Variants

```
inp_buffer AL[8][8], BL[8][8]
acc_buffer CL[8][8]
for yo in range(128):
  for xo in range(128):
    vdla.fill_zero(CL)
    for ko in range(128):
      vdla.dma_copy2d(AL, A[ko*8:ko*8+8][yo*8:yo*8+8])
      vdla.dma_copy2d(BL, B[ko*8:ko*8+8][xo*8:xo*8+8])
      vdla.fused_gemm8x8_add(CL, AL, BL)
    vdla.dma_copy2d(C[yo*8:yo*8+8,xo*8:xo*8+8], CL)
```

```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      for yi in range(8):
        for xi in range(8):
          for ki in range(8):
            C[yo*8+yi][xo*8+xi] +=
               A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```

```
for y in range(1024):
  for x in range(1024):
    C[y][x] = 0
    for k in range(1024):
      C[y][x] += A[k][y] * B[k][x]
```

Elements of Low-level Loop Representation



Transforming Loops: Loop Splitting

Code

```
for x in range(128):
  C[x] = A[x] + B[x]
```

```
for xo in range(32):
  for xi in range(4):
     C[xo * 4 + xi]
      = A[xo * 4 + xi] + B[xo * 4 + xi]
```

Transformation

```
x = get_loop("x")
xo, xi = split(x, 4)
```

Transforming Loops: Loop Reorder

Code

```
for xo in range(32):
  for xi in range(4):
    C[xo * 4 + xi]
     = A[xo * 4 + xi] + B[xo * 4 + xi]
```

```
for xi in range(4):
  for xo in range(32):
    C[xo * 4 + xi]
     = A[xo * 4 + xi] + B[xo * 4 + xi]
```

Transformation

```
x = get_loop("x")
xo, xi = split(x, 4)
reorder(xi, xo)
```

Transforming Loops: Thread Binding

Code

```
for xi in range(4):
  for xo in range(32):
    C[xo * 4 + xi]
      = A[xo * 4 + xi] + B[xo * 4 + xi]
```

```
def gpu_kernel():
  C[threadId.x * 4 + blockIdx.x] = . . .
```

Transformation

```
x = get_loop("x")
xo, xi = split(x, 4)
reorder(xi, xo)
bind_thread(xo, "threadIdx.x")
bind_thread(xi, "blockIdx.x")
```

Search via Learned Cost Model