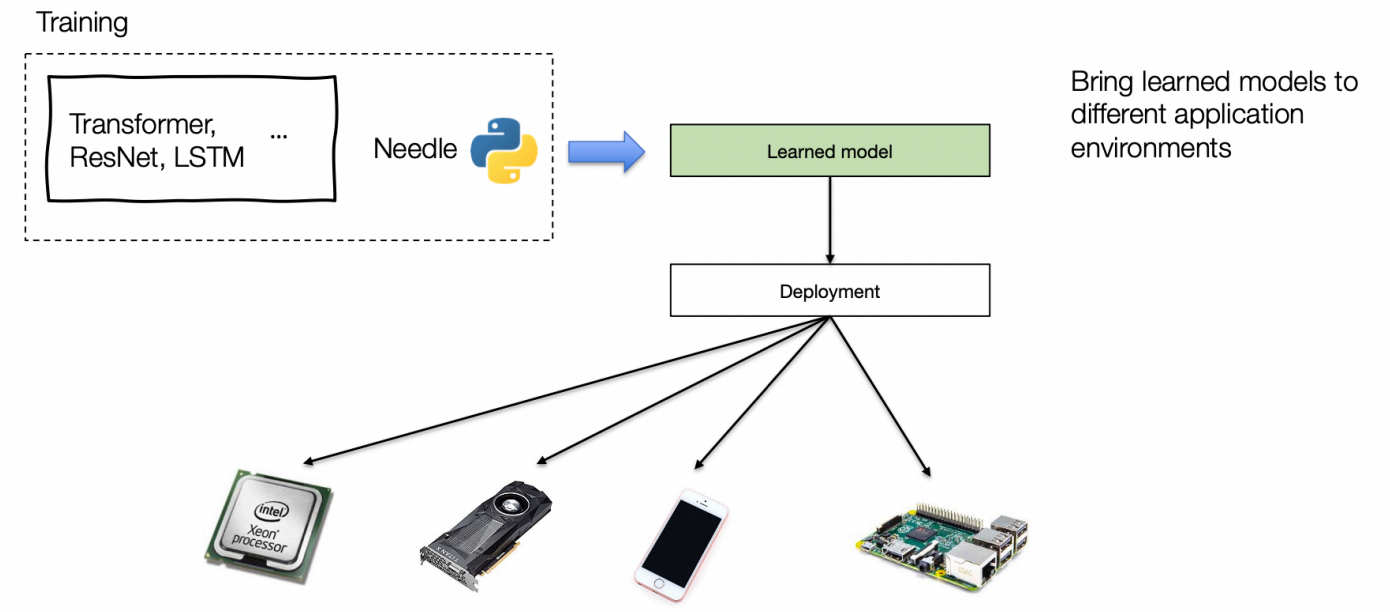
Model Deployment

将训练好的模型部署在不同的应用环境下



Model Deployment Overview

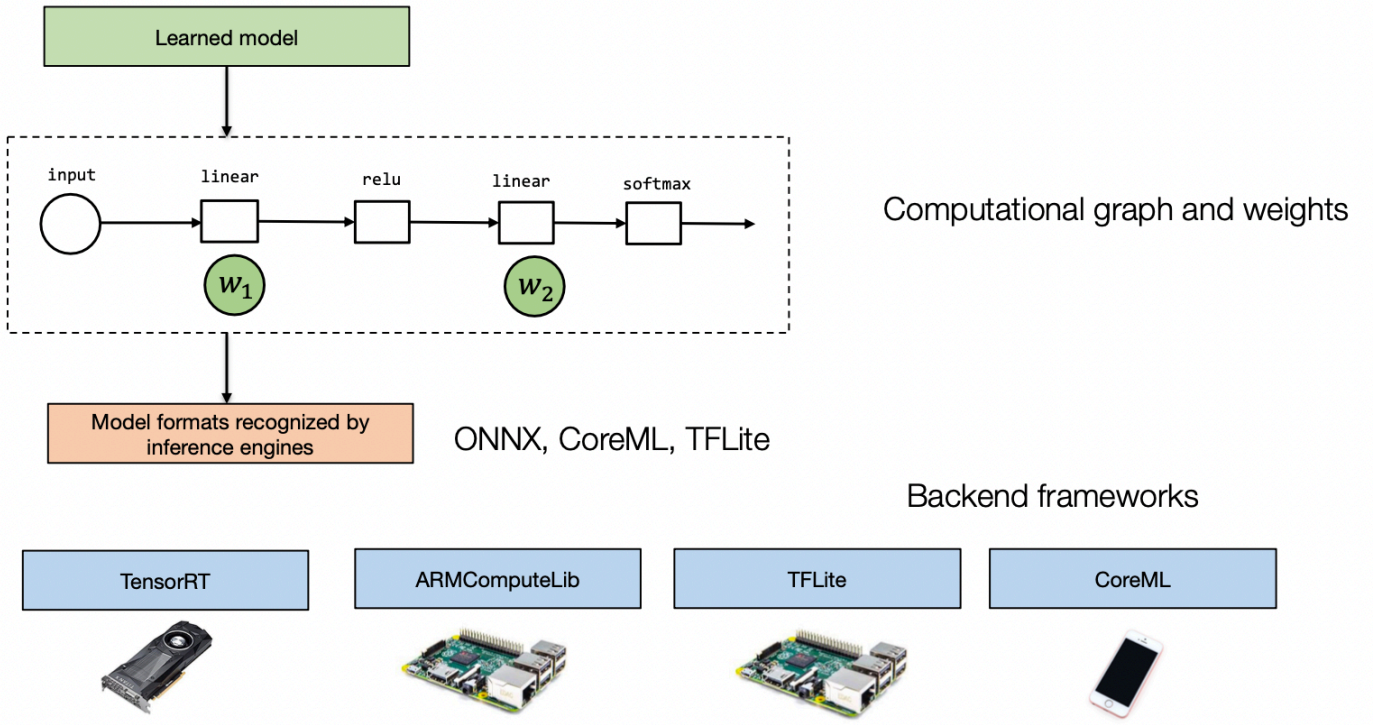
Model Deployment Considerations

模型部署主要考虑以下三个方面：

- 应用环境的限制（模型大小, no-python）
- 如何利用本地硬件进行加速 (mobile GPUs, accelerated CPU instructsions, NPUs)
- 如何与应用集成结合 (data preprocessing, post processing)

Model Exportation

将训练好的模型转换成后端推理引擎 (inference engine) 支持的计算图格式



Inference Engine Internals

很多推理引擎本质上是计算图的解释器 (interpreter)

推理引擎为中间激活函数分配内存，遍历并执行计算图中的算符，并且只支持一部分算符和编程模型

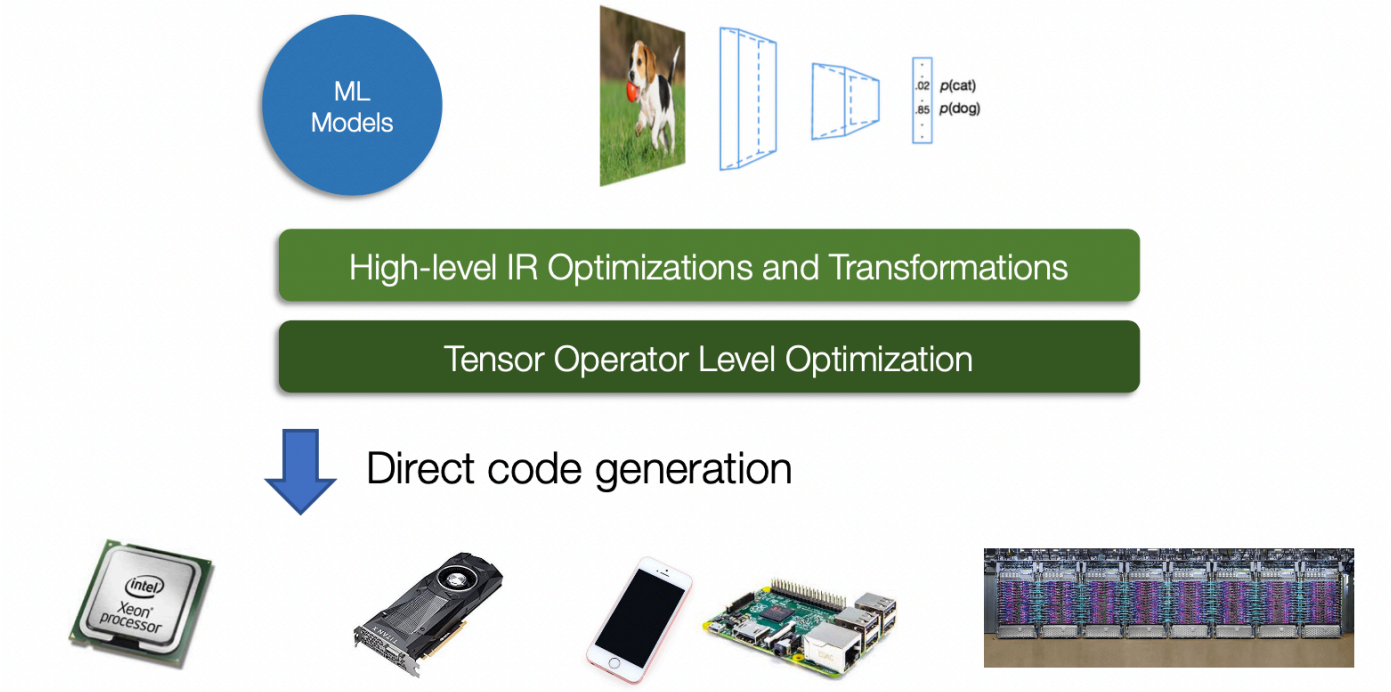
Machine Learning Compilation

Limitation of Library Driven Inference Engine Deployment

Library driven 的推理引擎需要为每个硬件后端建立特制的库，需要很多人力对库进行优化

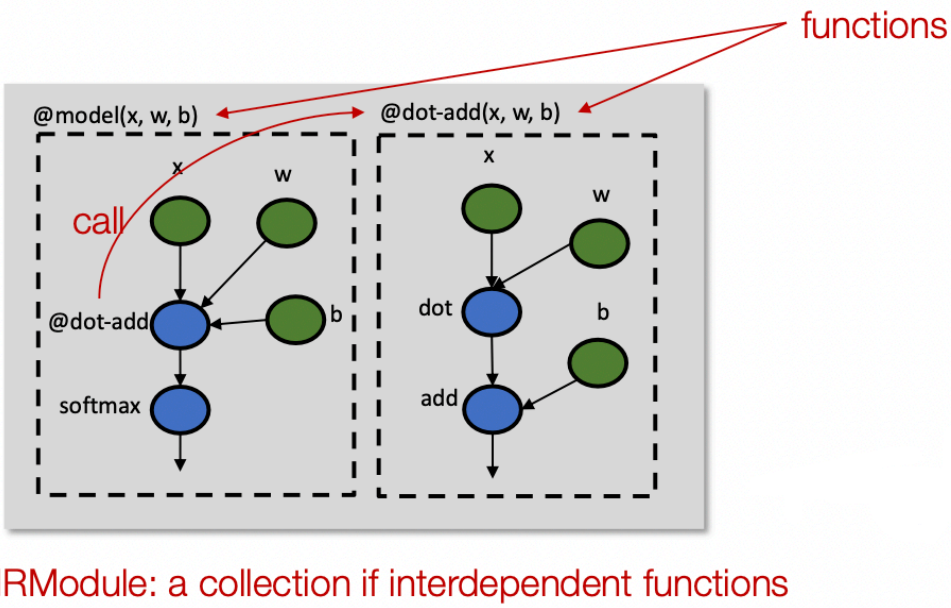
ML Compilation

机器学习编译希望将机器学习模型转化成 high-level IR 并对其进行优化与转换，再进行算符层面的优化，最后生成能在硬件上执行的代码



Compiler Representation of a Model

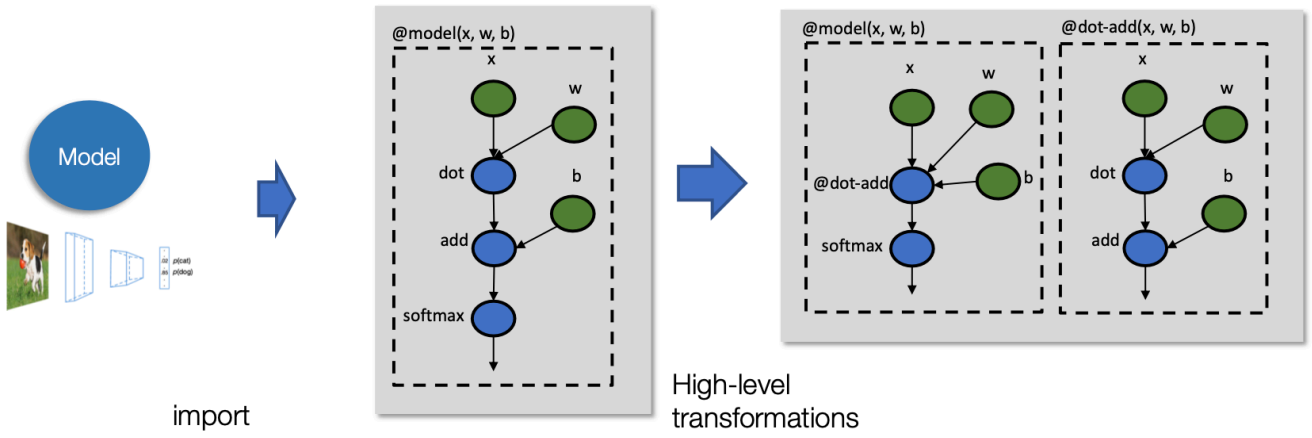
编译器通常将模型表示为 IRModule, 其中包含了所有互相依赖的函数，该模块也被称为 IR.



Example Compilation Flow

High-level Transformations

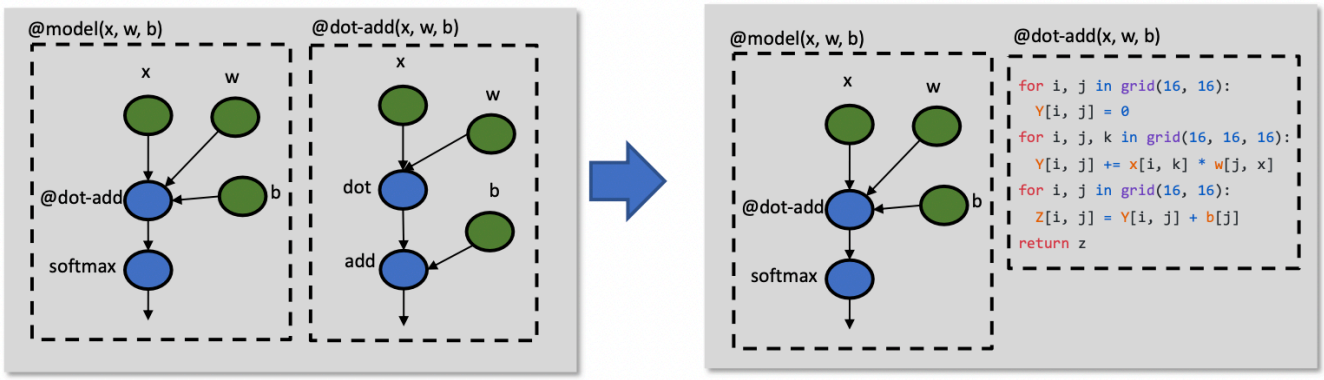
对模型的计算图进行转换，对多个基本算符进行合并 (operator fusion) 得到新算符，形成上文所述的 IRModule.



operator fusion 能够节省计算内存

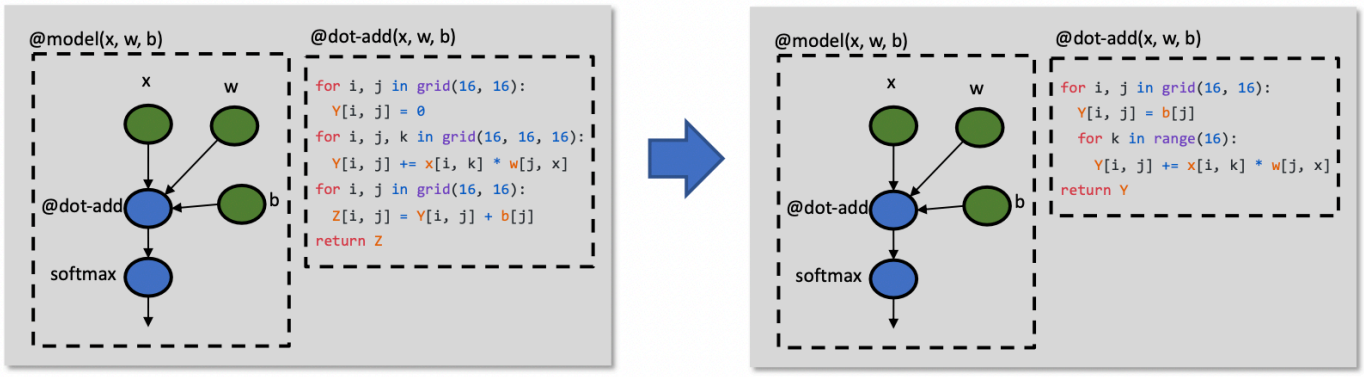
Lowering to Loop IR

将函数计算图转换成循环形式的 IR



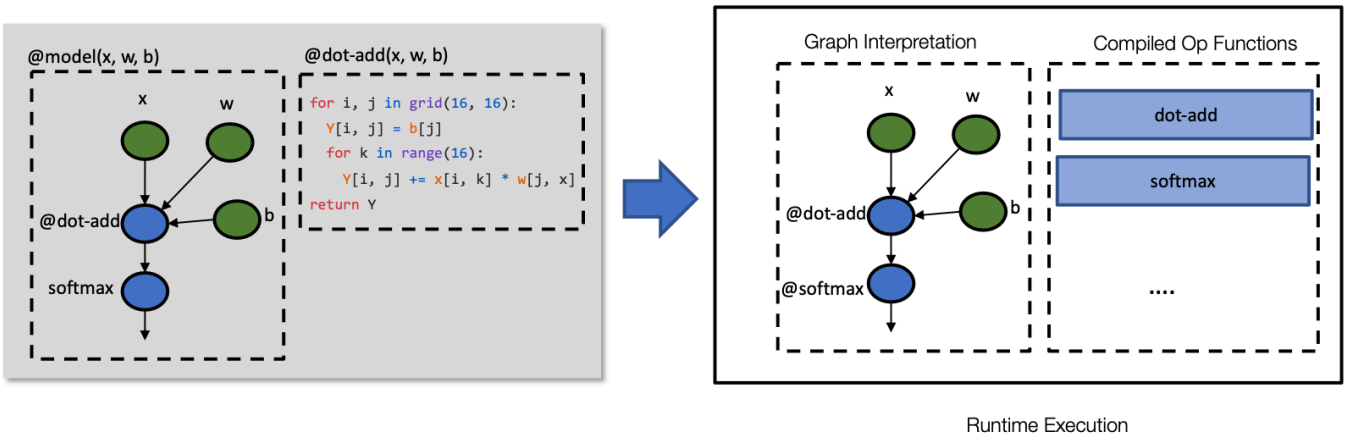
Low-level Transformations

对 Loop IR 进行相应转换，简化循环过程



Code Generation and Execution

根据 Loop IR 生成函数的代码，在计算图中被调用



Runtime Execution

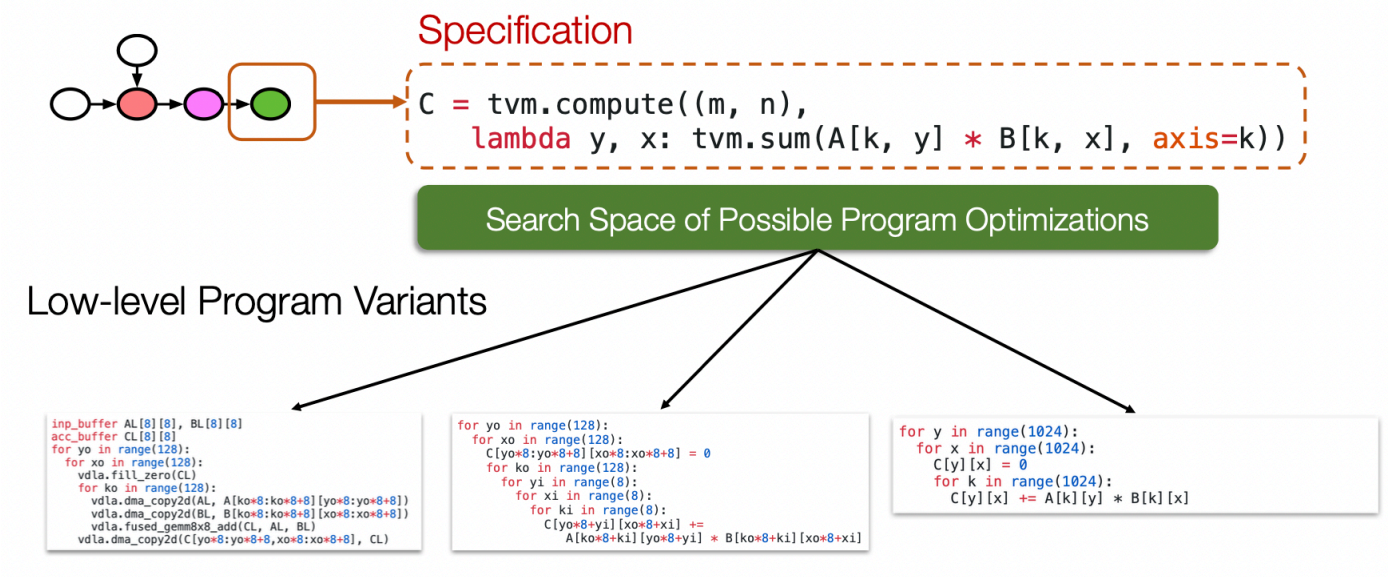
High-level IR and Optimizations

High-level IR 指的是用计算图来表示模型，每个节点都是一个张量算符，可以通过 fusion 进行转换

大多数 ML 框架含有这一层表示

Low-level Code Optimizations

底层代码优化需要在搜索空间中搜索可能的程序优化



Transforming Loops

底层循环主要包含三个部分：多维的 buffer, 循环结构, 数组计算

Splitting

对循环进行拆分

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)
```

Reorder

对循环嵌套结构进行重排

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)
```

Thread Binding

将循环变量绑定到线程上，从 CPU 执行代码转成 GPU 执行代码

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

在循环优化过程中需要设置很多参数，采取的优化手短多，搜索空间大，因此可引入衡量计算开销的模型使得尽快搜索到最优的生成代码