Discovering High-Performance Tensor Programs with Bayesian Optimization

Felix Jonathan Rocke

Supervisors: Eiko Yoneki and Guoliang He
University of Cambridge

Motivation: Deploying an ML Application to Device

ML Application

Target Hardware (CPU, GPU, TPU)

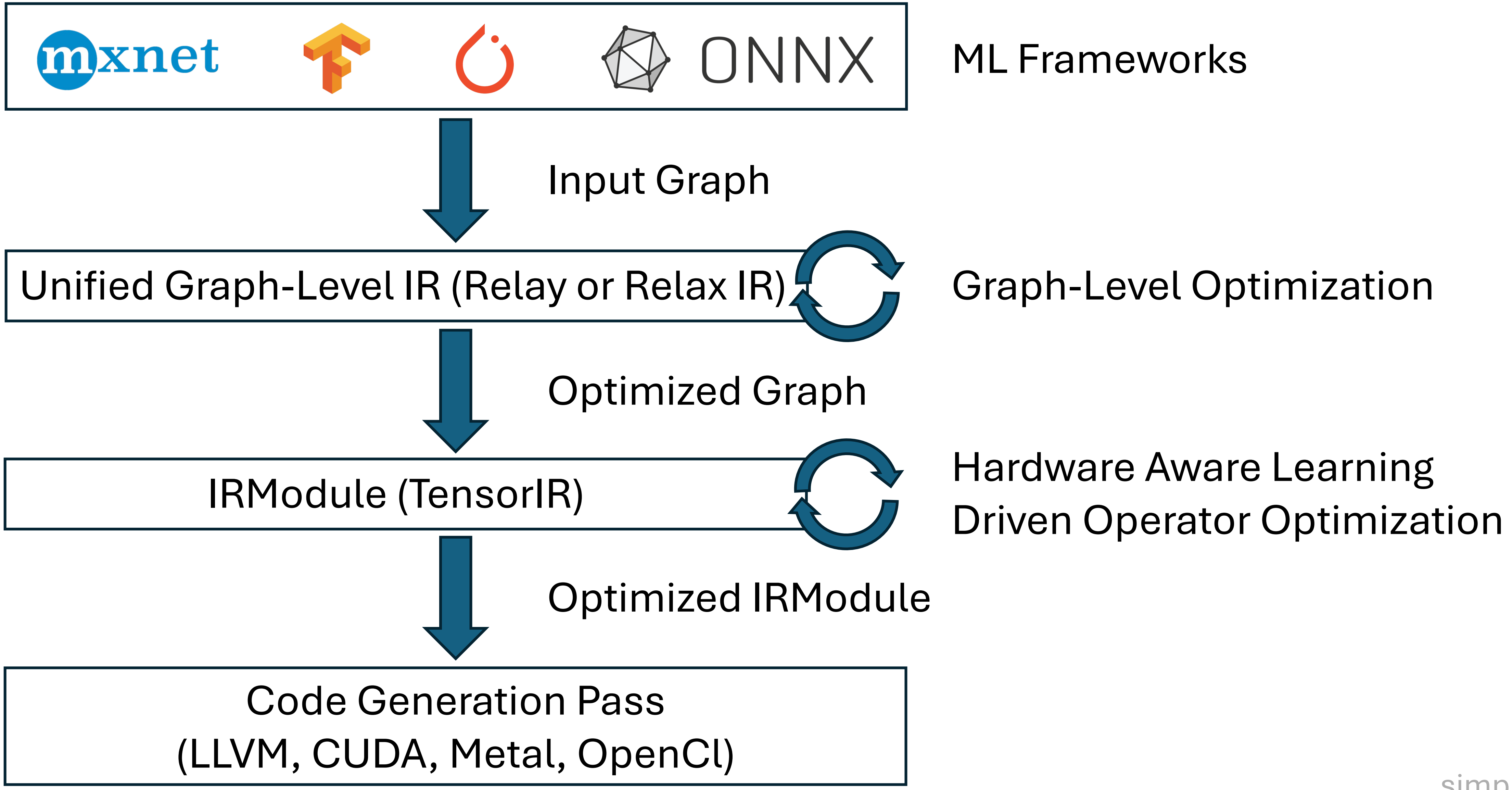
- Various Architectures
- Hundreds of Operators

- Various Microarchitectures
- Increasing Specialization (e.g., Tensor Cores)

Traditionally: Compile model using hardware-specific template libraries. However, high cost due to hardware and model variety

New Solution: A learning compiler that generates optimized kernels without templates

Overview Apache TVM



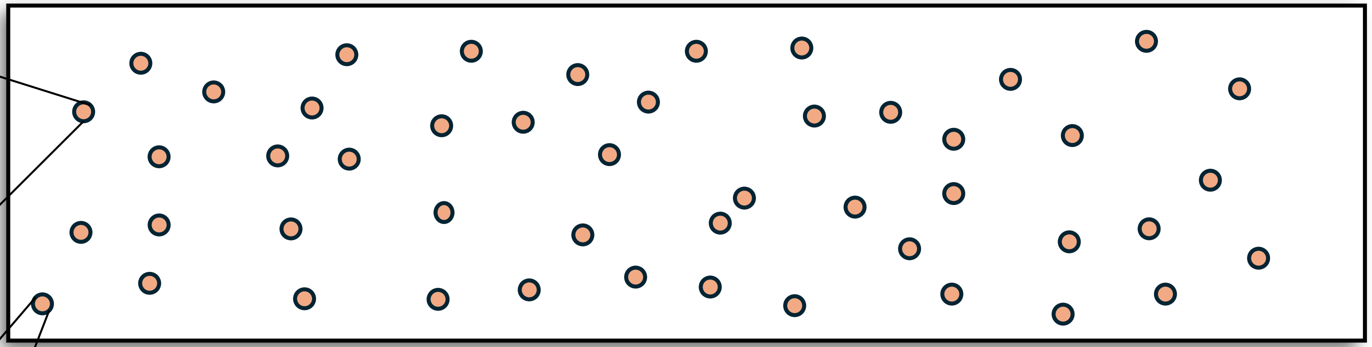
simplified

Problem Description

Possible Program 1

```
for i0, j0 in grid(16, 8):
  for i1, j1 in grid(8, 16):
    for k0 in range(1024):
      for i2, j2 in grid(8, 8):
        C[…] += …
```

Search Space of Equivalent Programs



Possible Program 2

```
for i0, j0 in grid(64, 8):
  for i1, j1 in grid(4, 32):
    for k0 in range(64):
      for i2, j2 in grid(4, 4):
        for k1 in range(16):
          C[…] += …
```

Challenge:

- Billions of possible equivalent programs
- Minimize the number of program evaluations required to find an efficient implementation

Proposed Solution:

Bayesian Optimization (BO) as it is a sample-efficient search strategy, with the potential to outperform TVM's ***Evolutionary Search*** (ES)

Code figures inspired by: https://doi.org/10.48550/arXiv.2205.13603

Rewriting Programs using Parameterized Transformations

Initial Program e_0

```
for i in range(1024):
   C[i] = A[i] + B[i]
```

$e_0 + ①$

Equivalent Program e_1

```
for i0 in range(32):
   for i1 in range(8):
      for i2 in range(4):
         i = i0 * 32 + i1 * 4 + i2
         C[i] = A[i] + B[i]
```

$e_1 + ② + ③$

Optimized Program e_2

```
parallel for i0 in range(32):
   for i1 in range(8):
      i = i0 * 32 + i1 * 4
      C[i : i + 4] =
         A[i : i + 4] + B[i : i + 4]
```

Transformation Trace

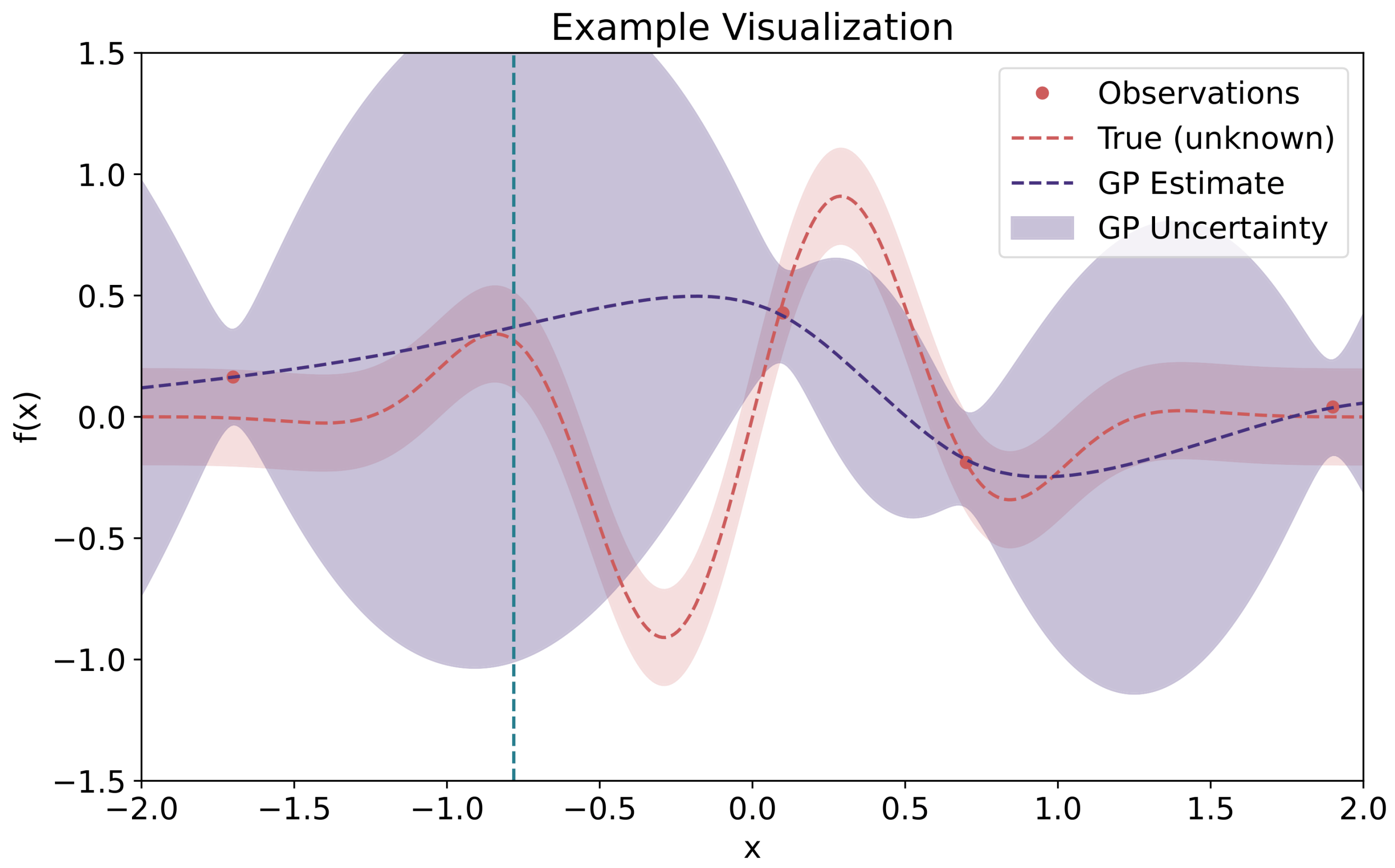
① `Split(loop=i, decision=[32, 8, 4])`

② `Parallelize(loop=i0)`

③ `Vectorize(loop=i2)`

Code figures inspired by: https://doi.org/10.48550/arXiv.2205.13603

Bayesian Optimization (BO)

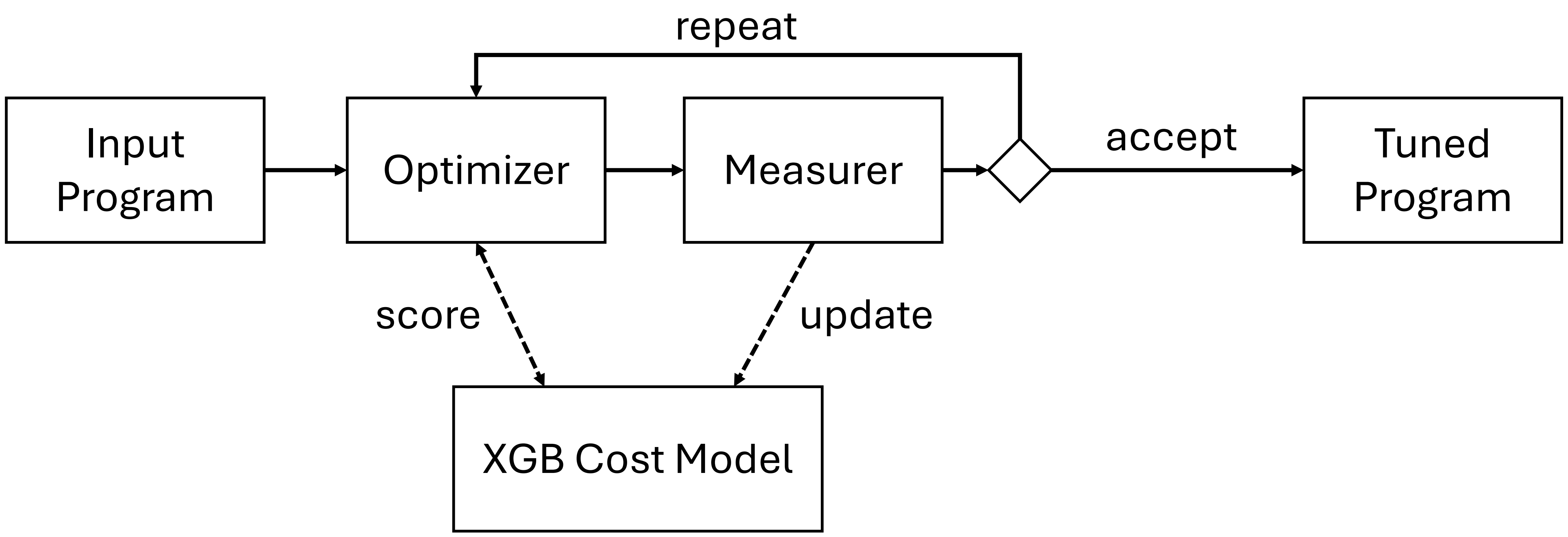


GP = Gausian Process
Evaluate f(x) at x = -0.7813

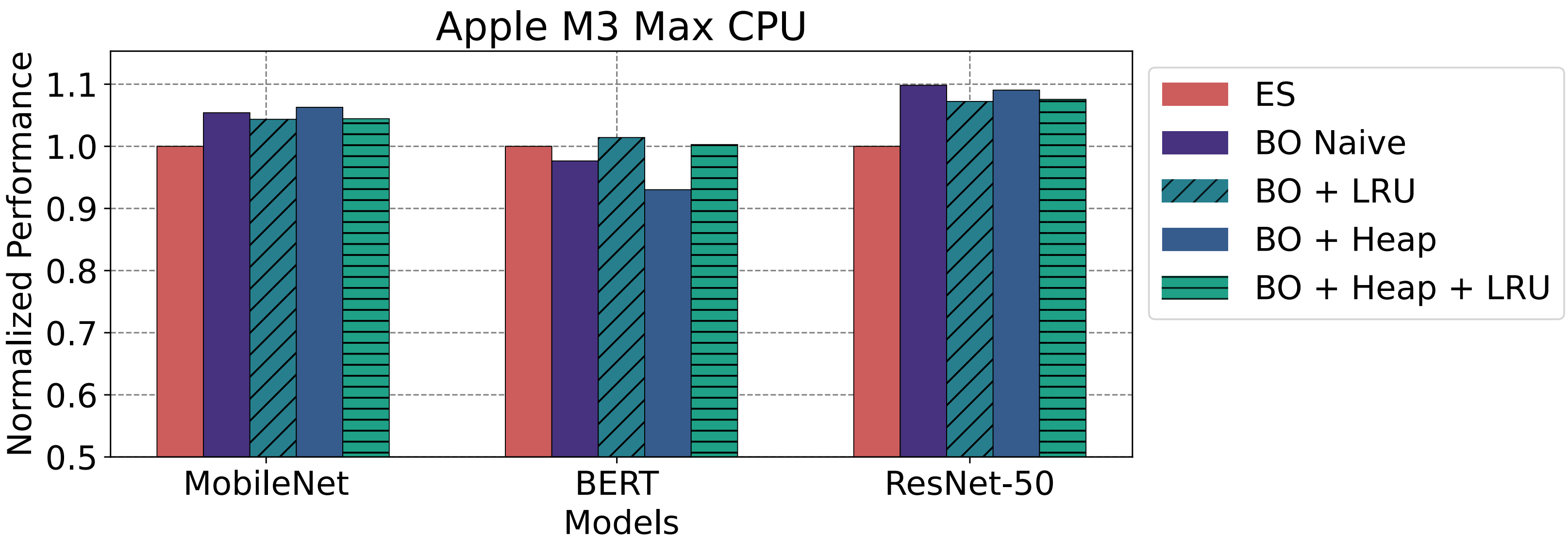
- Sequential design strategy
- Makes informed decisions based on previous observations
- Creates a probabilistic model of the black-box objective function
- Can manage exploration and exploitation
- Works best with continuous and smooth functions

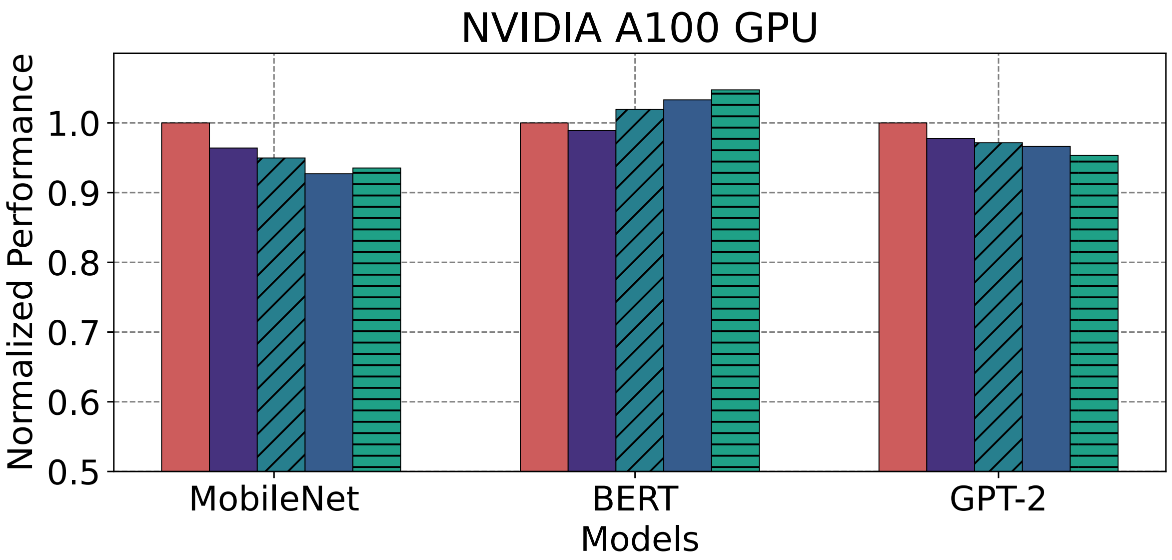
Code for figure adapted from: https://github.com/glouppe/talk-bayesian-optimisation/tree/master

Simplified Search Strategy Overview



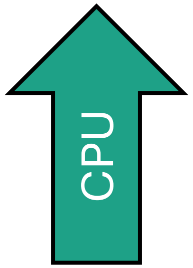
Benchmarking Results



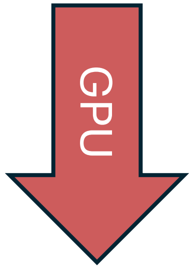


Normalized end-to-end inference latency after 6000 trials

The Best Configuration



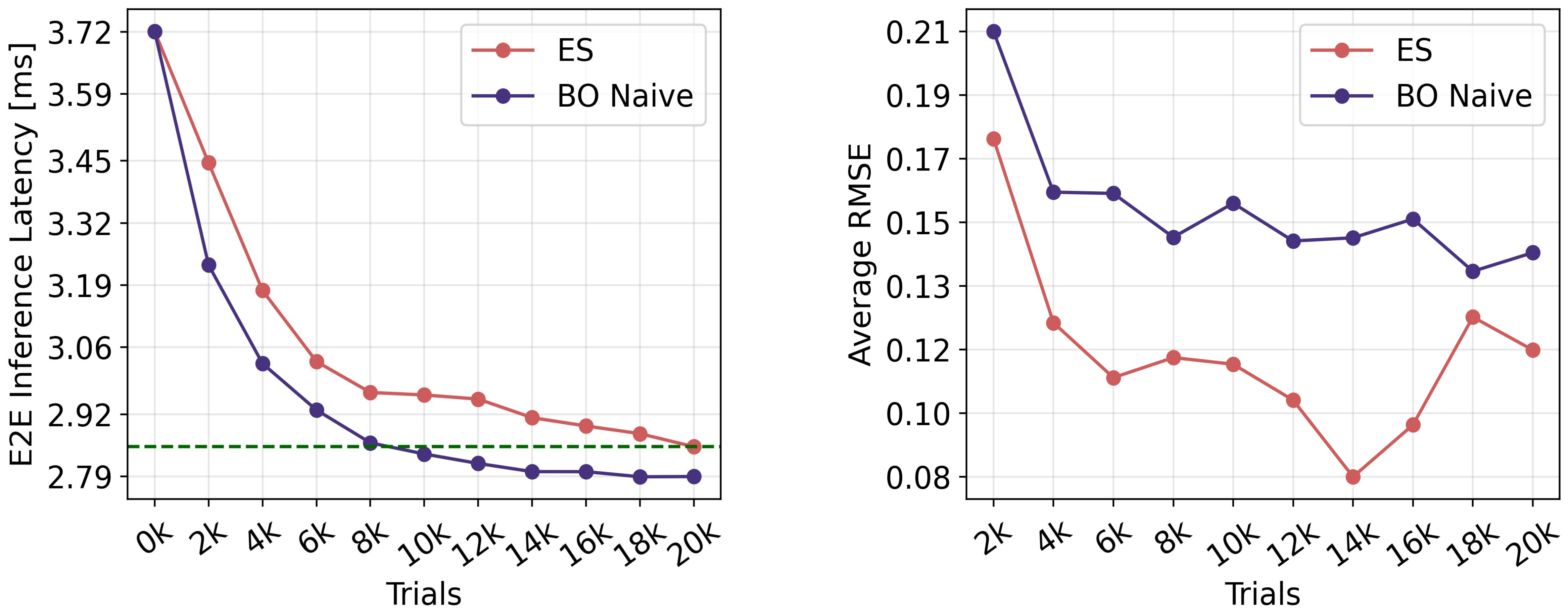
is on average 5.8% faster



is on average 0.4% slower

(all results produced with commit dffe78b)

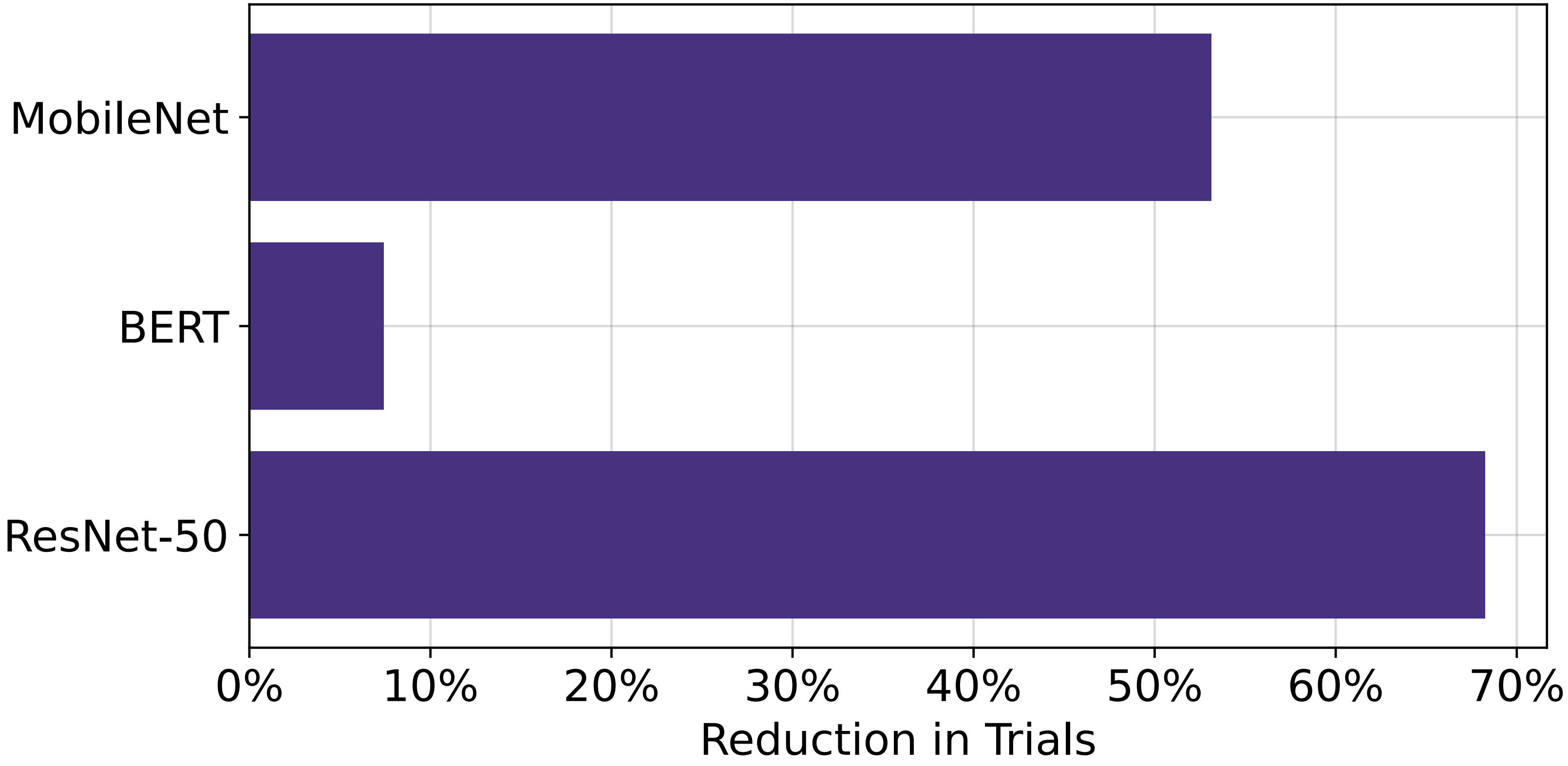
Compiling MobileNet with 20,000 Trials (CPU)



a) Latency after x-trials

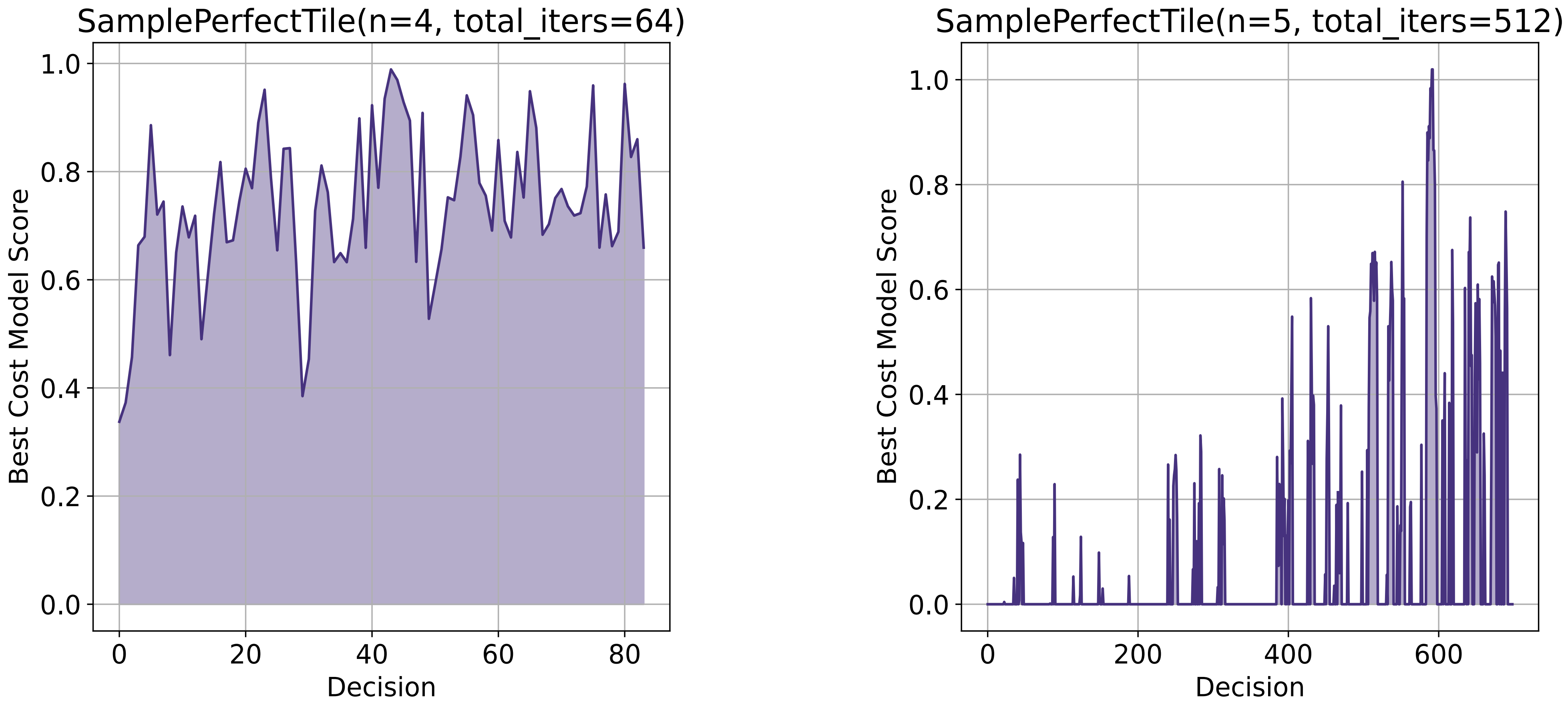
b) Cost Model RMSE

Reduction in Trials Compared to Evolutionary Search on CPUs



BO reduces the number of trials required to find a high-performance implementation by up to 68%

Sketch of One-Objective Function Dimension



a) CPU b) GPU

Problem: The jumps in the GPU's objective function make BO challenging

Summary

1. Implementation of Bayesian Optimization as a novel search strategy into Apache TVM and MetaSchedule
2. Notable performance improvement of up to 10% for CPU models when compared to ES at the same number of trials, resulting in a reduction of up to 68% in trials
3. For GPUs, BO's performance is more limited
   - ➢ However, our analysis of the black-box objective function in the CPU and GPU space will allow future research to be more targeted and efficient

Extended TVM Version: https://github.com/felix-ro/TVM-Bayesian-Optimization
Benchmarking Scripts: https://github.com/felix-ro/TVM-Benchmarking