Deep Learning in Compilers

Chris Cummins
Pavlos Petoumenos
Zheng Wang
Hugh Leather

THE UNIVERSITY of EDINBURGH
informatics

EPSRC Centre for Doctoral Training in
Pervasive Parallelism

machine learning for compilers

$$y = f(x)$$

Decisions

CPU or GPU

Workgroup size

Cflags

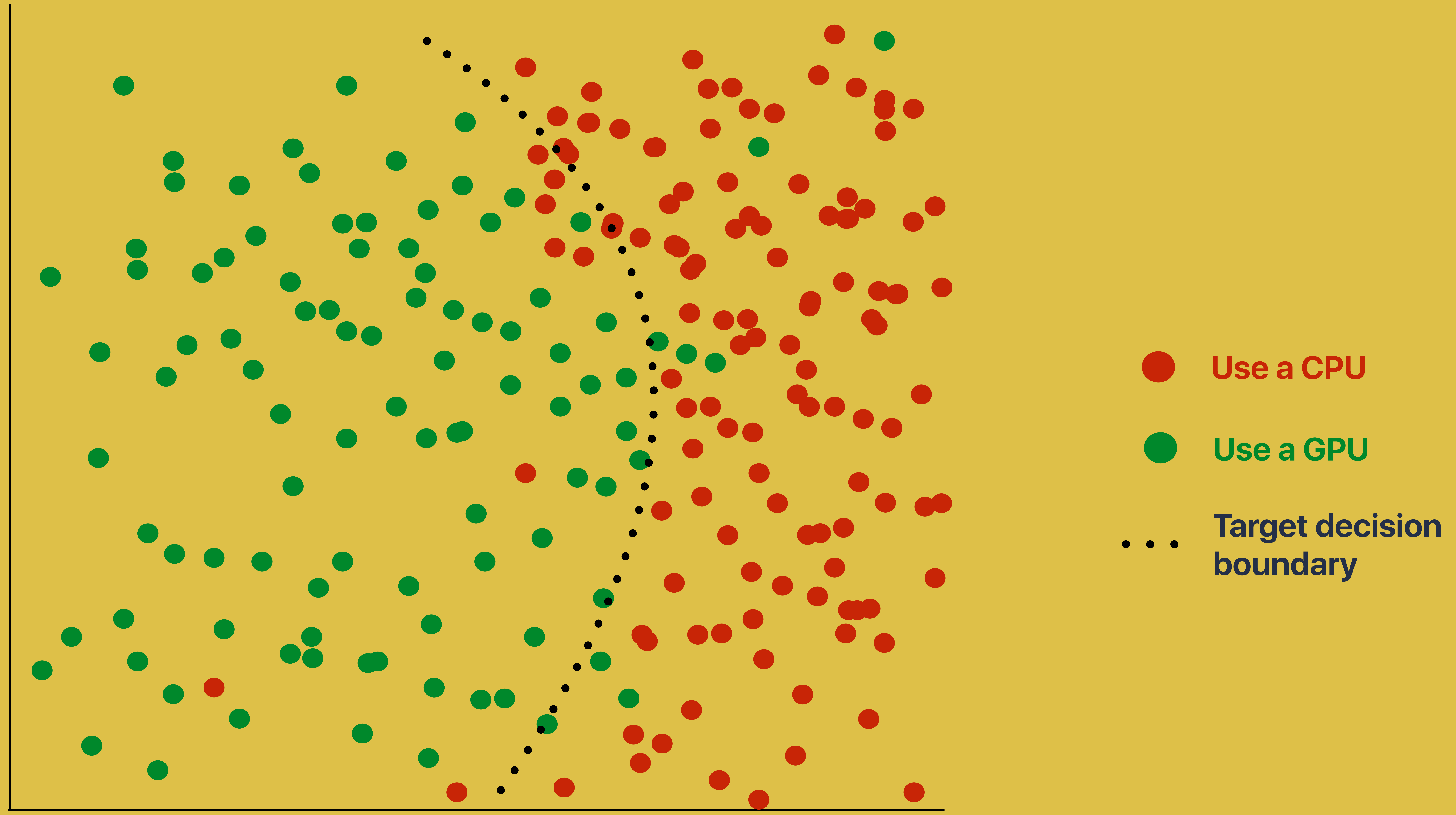
Features

instructions

Arithmetic density

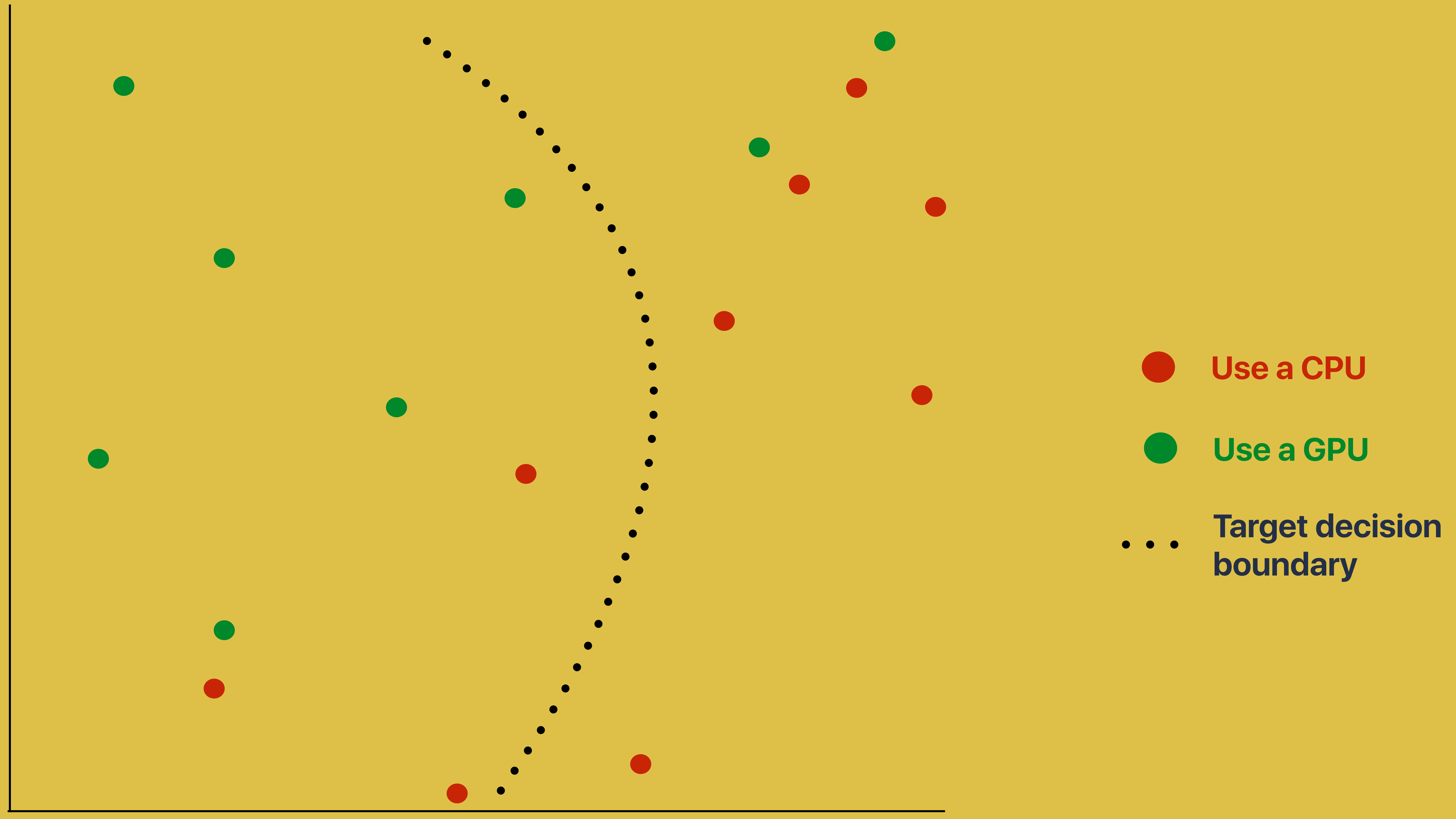
Dataset size

machine learning for compilers



The idea.

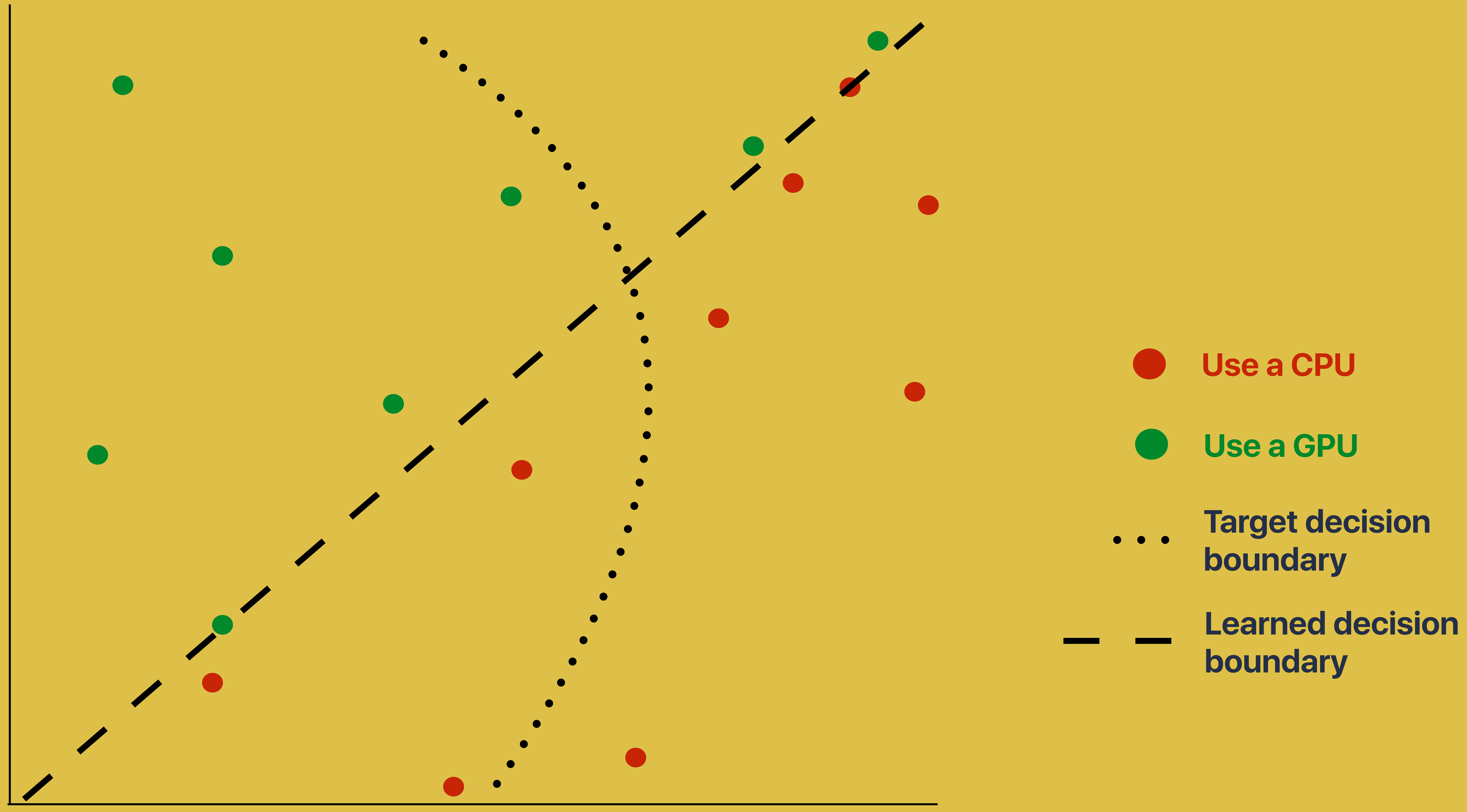
machine learning for compilers



The reality.

*17 benchmarks in avg compiler paper 2013-2016

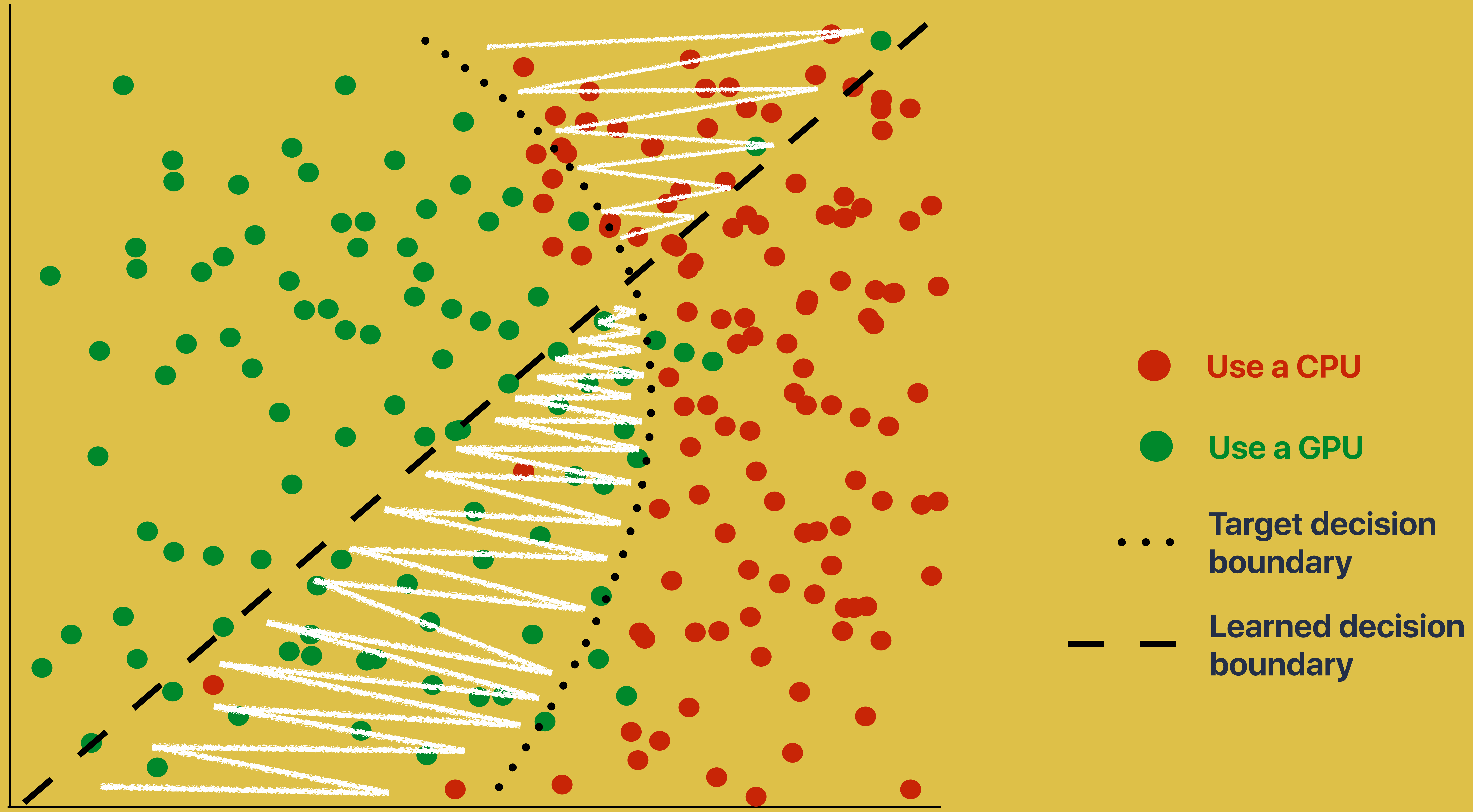
machine learning for compilers



The reality.

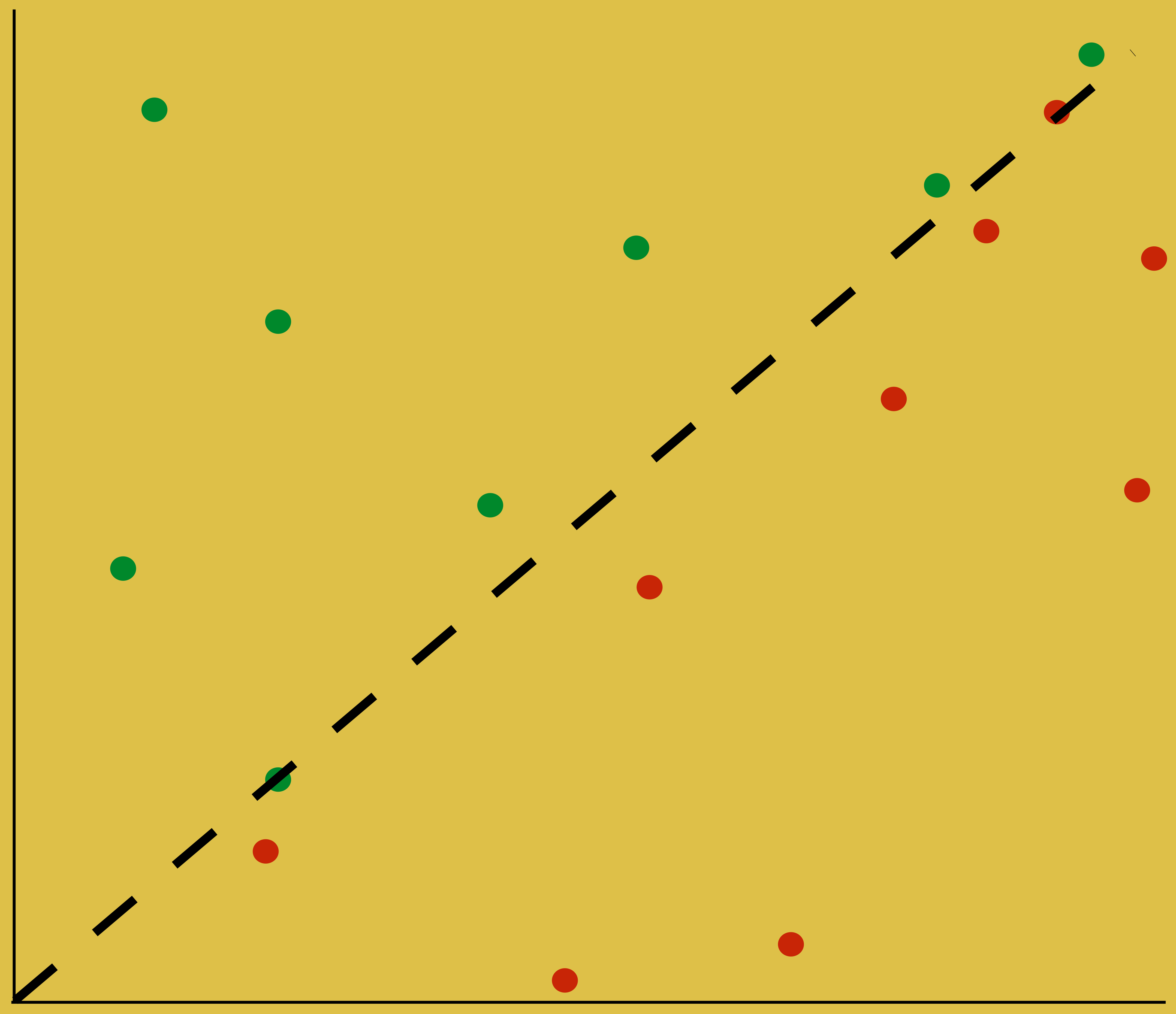
*17 benchmarks in avg compiler paper 2013-2016

machine learning for compilers

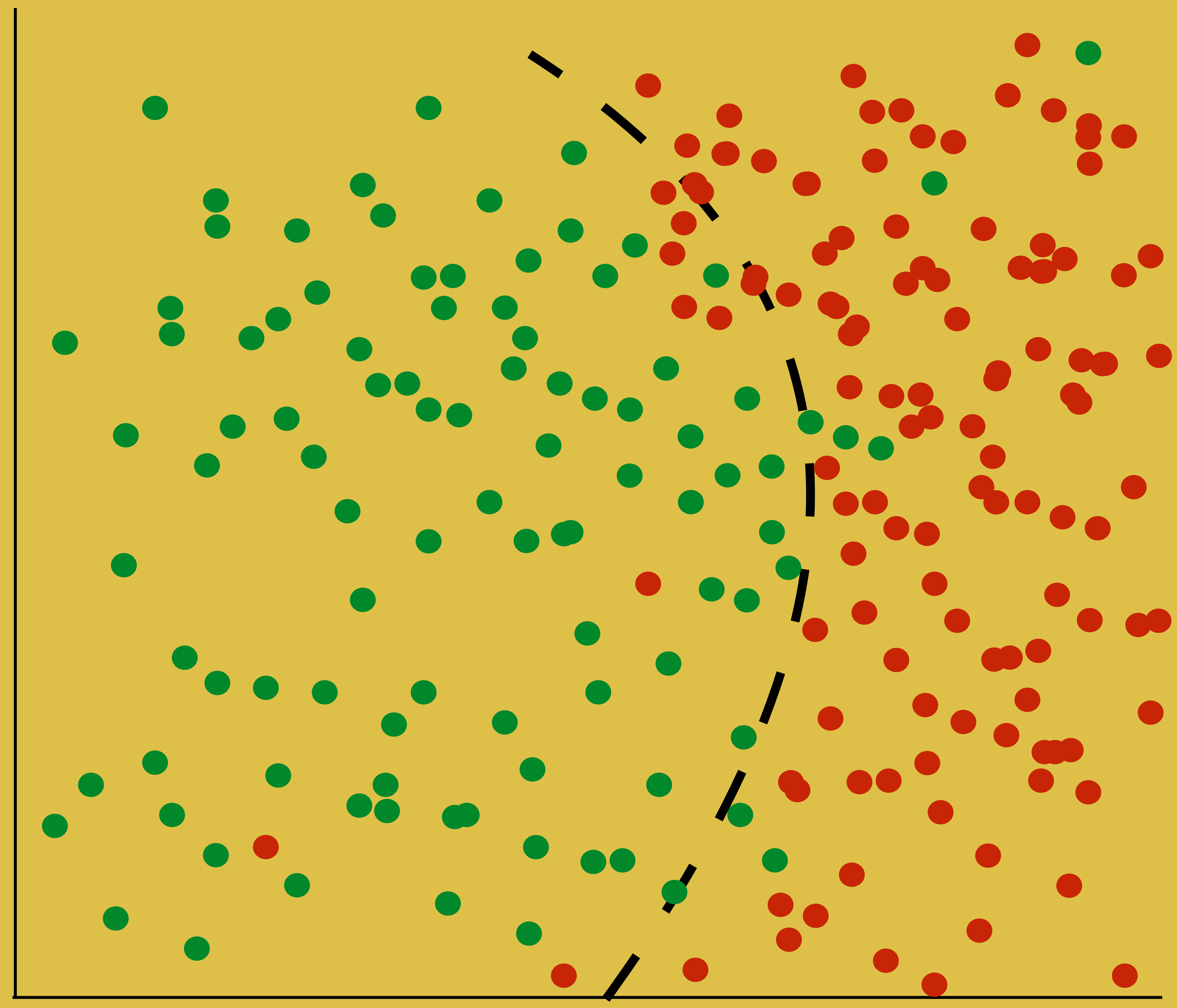


sparse data leads to inaccurate models!

what we need



from this



to this

our approach

mine code from web

our approach

model source distr.

our approach

sample lang. model

52% blind test

http://humanorrobot.uk

Listing 3: Sample 3

```
__kernel void A(__global int* a, __global int* b, __global int* c,
    __global int* d, const uint e) {
  const uint f = get_global_id(0);

  if (e == 0 && f == 0)
    *d = 0;
  else if (f < e) {
    int g = b[f];
    uint h = c[f];
    if (g > 0) {
      a[h] = f;
      h++;
    }
    if (f == e - 1)
      *d = h;
  }
}
```

Listing 4: Sample 4

```
__kernel void A(__global float* a, __global float* b, __global float* c,
    const int d) {
  int e = get_global_id(0);

  if (e < d) {
    float f = b[e];
    float g = a[e];
    a[e] = f * 3.141592f / (f + 1.0f + e * 1024 - f) + (0.5f * g * 1.0f
        / 18.0f + e / 2.0f);
  }

  for (e = 0; e < 30; e++) {
    c[e] = 0;
  }
}
```

Listing 4: Sample 4

```
__kernel void A(int a, __global float* b, __global int* c, __global int*
    d, __local int* e, int f) {
  int g = get_local_id(0);
  e[get_local_id(0)] = 0;
  barrier(1);
  while (g < f) {
    int h = c[g];

    if (h != -1) {
      __global float* i = b + g * a;
      float j = 0;
```

Listing 7: Sample 7

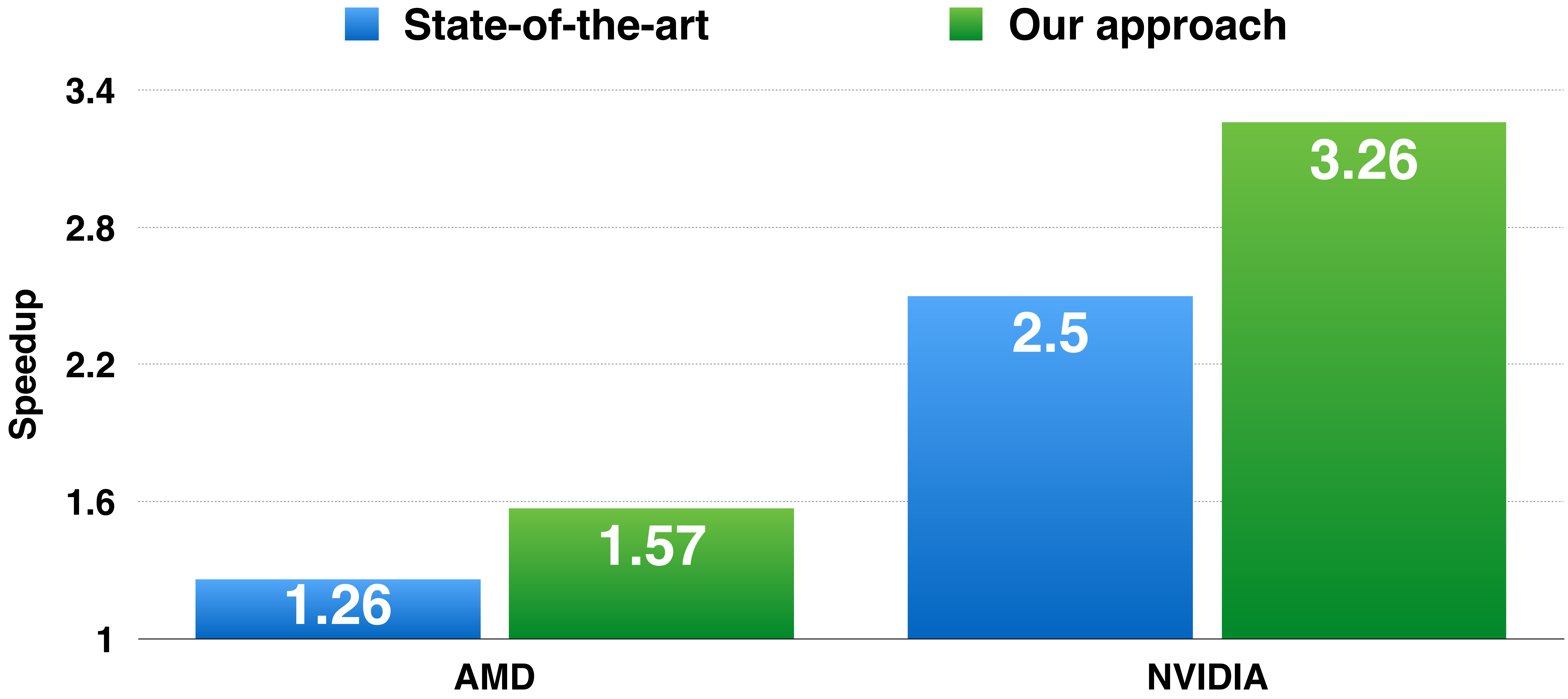
```
__kernel void A(int a, int b, int c, __global const float* d, __global
    const float* e, __global float* f, float g) {
  const int h = get_local_id(0);
  const int i = get_group_id(0);

  const int j = 4 * i + h;
  const int k = 4 * i + h + a;
  if (4 * i + h + a < c) {
    float l = 0.0;
    float m = 0.0;
    float n = 0.0;
    const float o = d[3 * (4 * i + h + a)];
    const float p = d[3 * (4 * i + h + a) + 1];
    const float q = d[3 * (4 * i + h + a) + 2];
    for (int r = 0; r < c; r++) {
      const float s = d[3 * r] - o;
      const float t = d[3 * r + 1] - p;
      const float u = d[3 * r + 2] - q;
      const float v = (d[3 * r] - o) * (d[3 * r] - o) + (d[3 * r + 1] -
          p) * (d[3 * r + 1] - p) + (d[3 * r + 2] - q) * (d[3 * r +
          2] - q) + g;
      const float w = e[r] / (((d[3 * r] - o) * (d[3 * r] - o) + (d[3 *
          r + 1] - p) * (d[3 * r + 1] - p) + (d[3 * r + 2] - q) * (d
          [3 * r + 2] - q) + g) * sqrt((d[3 * r] - o) * (d[3 * r] - o
          ) + (d[3 * r + 1] - p) * (d[3 * r + 1] - p) + (d[3 * r + 2]
          - q) * (d[3 * r + 2] - q) + g));
      l = l + (d[3 * r] - o) * w;
      m = m + (d[3 * r + 1] - p) * w;
      n = n + (d[3 * r + 2] - q) * w;
    }
    f[3 * (4 * i + h)] = l;
    f[3 * (4 * i + h) + 1] = m;
    f[3 * (4 * i + h) + 2] = n;
  }
}
```

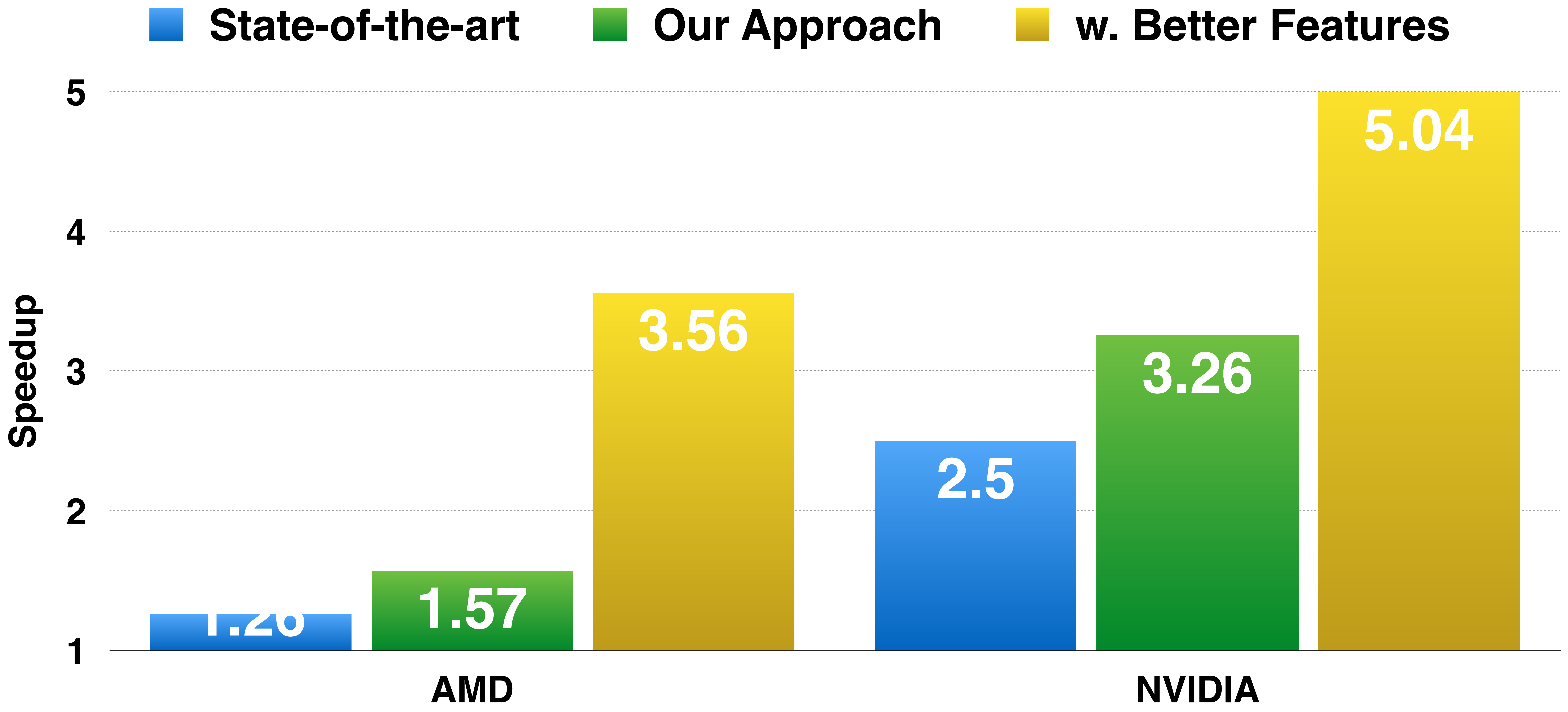
Listing 10: Sample 10

```
__kernel void A(__global ulong *a) {
    int i, j;
    struct S0 c_8;
    struct S0* p_7 = &c_8;
    struct S0 c_9 = {
        {{0x43250E6DL,2UL},{0x43250E6DL,2UL},{0x43250E6DL,2UL},
         {0x43250E6DL,2UL},{0x43250E6DL,2UL},{0x43250E6DL,2UL},
         {0x43250E6DL,2UL},{0x43250E6DL,2UL}},
        0x4BF90EDCAD2086BDL,
    };
    c_8 = c_9;
    barrier(0 | 1);
```

7 benchmarks, 1,000 synthetic benchmarks. 1.27x faster



71 benchmarks, 1,000 synthetic benchmarks. 4.30x faster



Synthesizing Benchmarks for Predictive Modeling

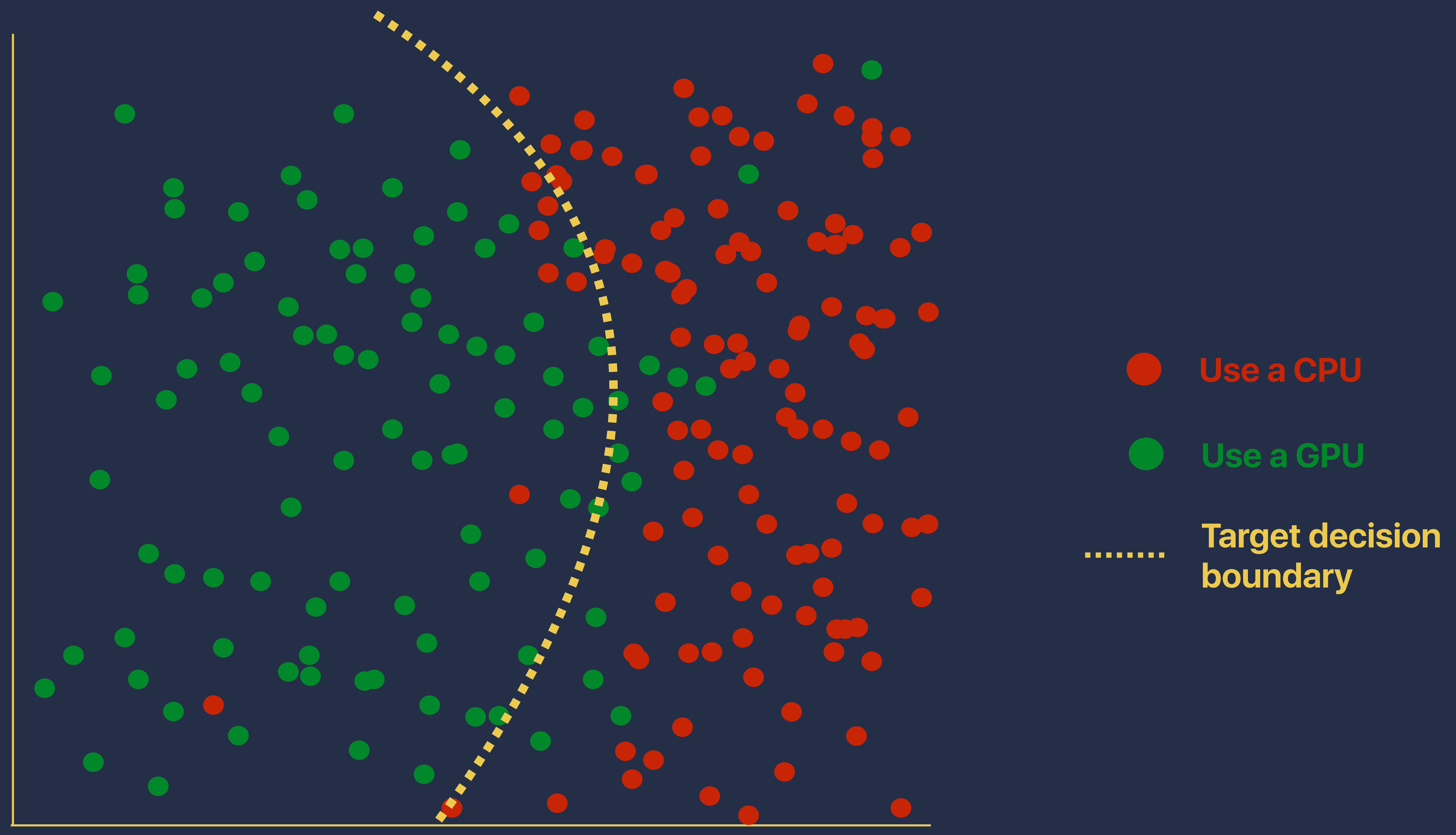
problem: insufficient benchmarks

first solution for general-purpose benchmark synthesis

turing tested! ;-)

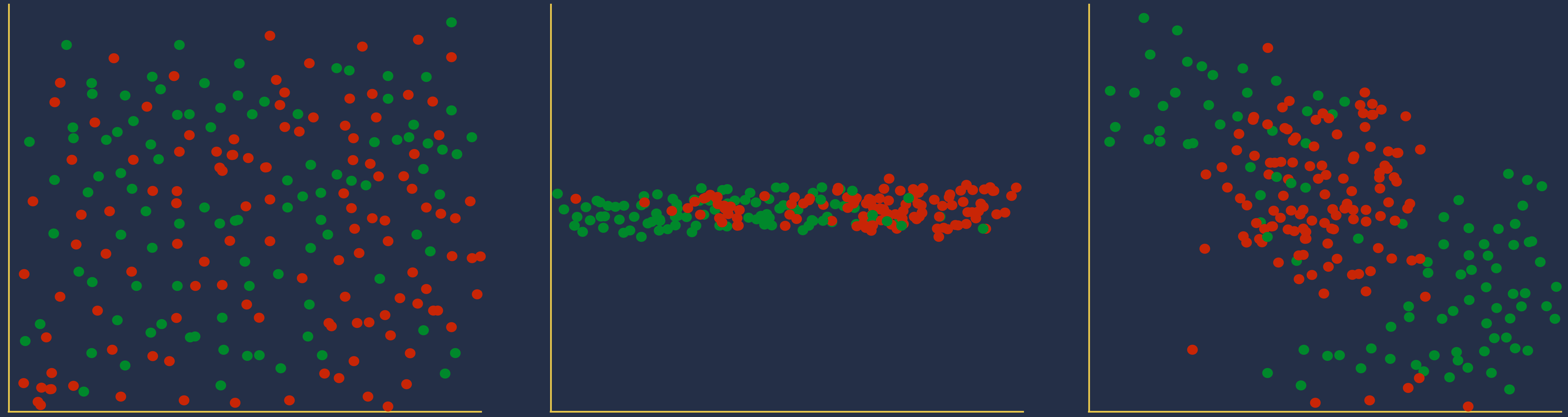
improved model performance and design

machine learning for compilers



The idea.

feature design is hard



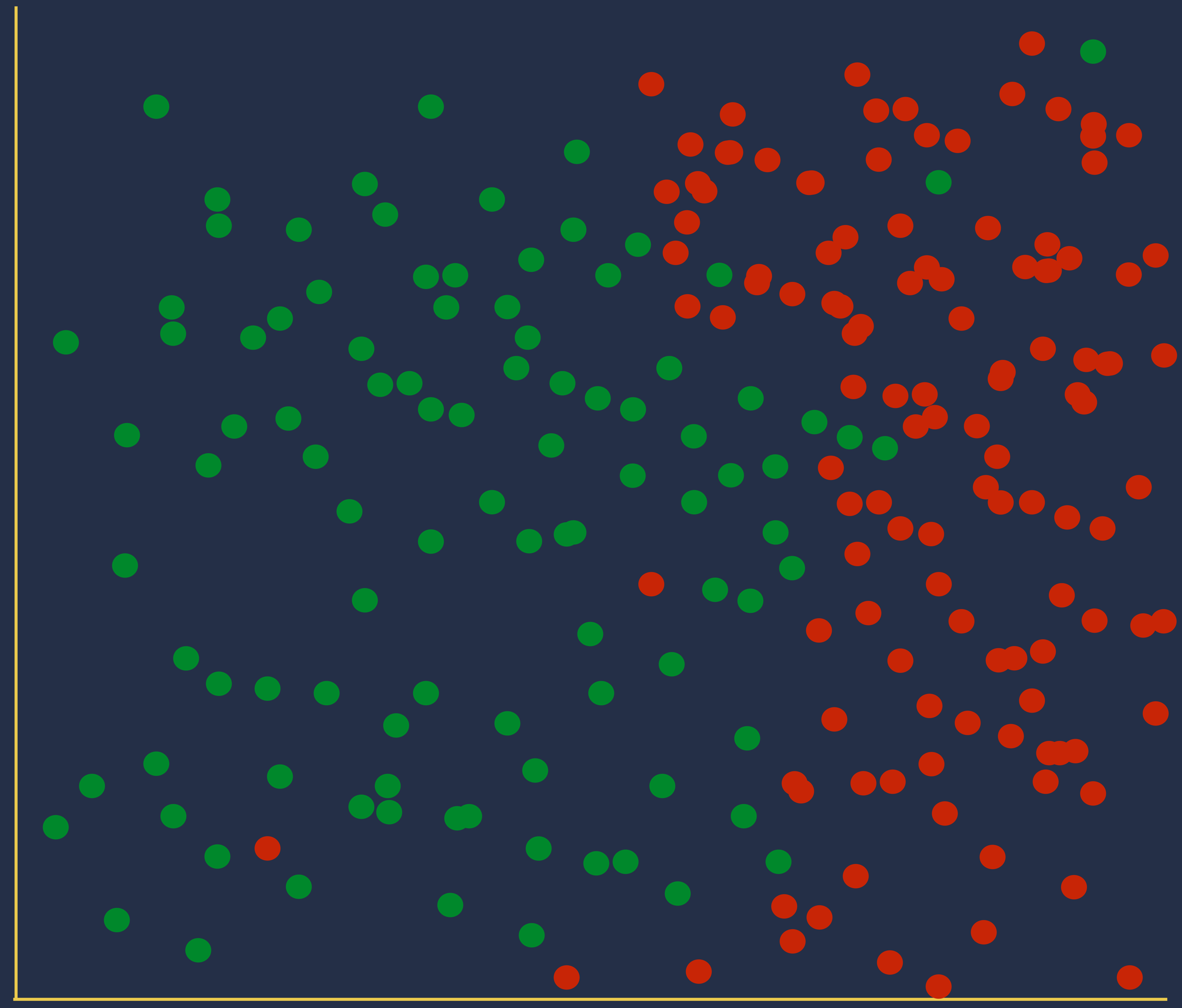
performance depends on good features!

what we need

program 1	CPU
program 2	CPU
program 3	CPU
program 4	CPU
...	
program *n*	CPU

from this

→



to this

our approach

```
__kernel void matrix_multiply(...
```

raw program code

↓

Source Transformations

↓

Deep Neural Networks

↓

Predicted Optimization

```
{CPU, GPU}
```

our approach

```
__kernel void matrix_multiply(...
```

raw program code

↓

Source Transformations

↓

Deep Neural Networks

↓

Predicted Optimization

```
{CPU, GPU}
```

Problem-agnostic

our approach

Optimisation Problem A

```
__kernel void matrix_multiply(...
```

raw program code

↓

Source Transformations

↓

Deep Neural Networks

↓

Predicted Optimization

```
{CPU, GPU}
```

Optimisation Problem B

```
__kernel void matrix_multiply(...
```

raw program code

↓

Source Transformations

↓

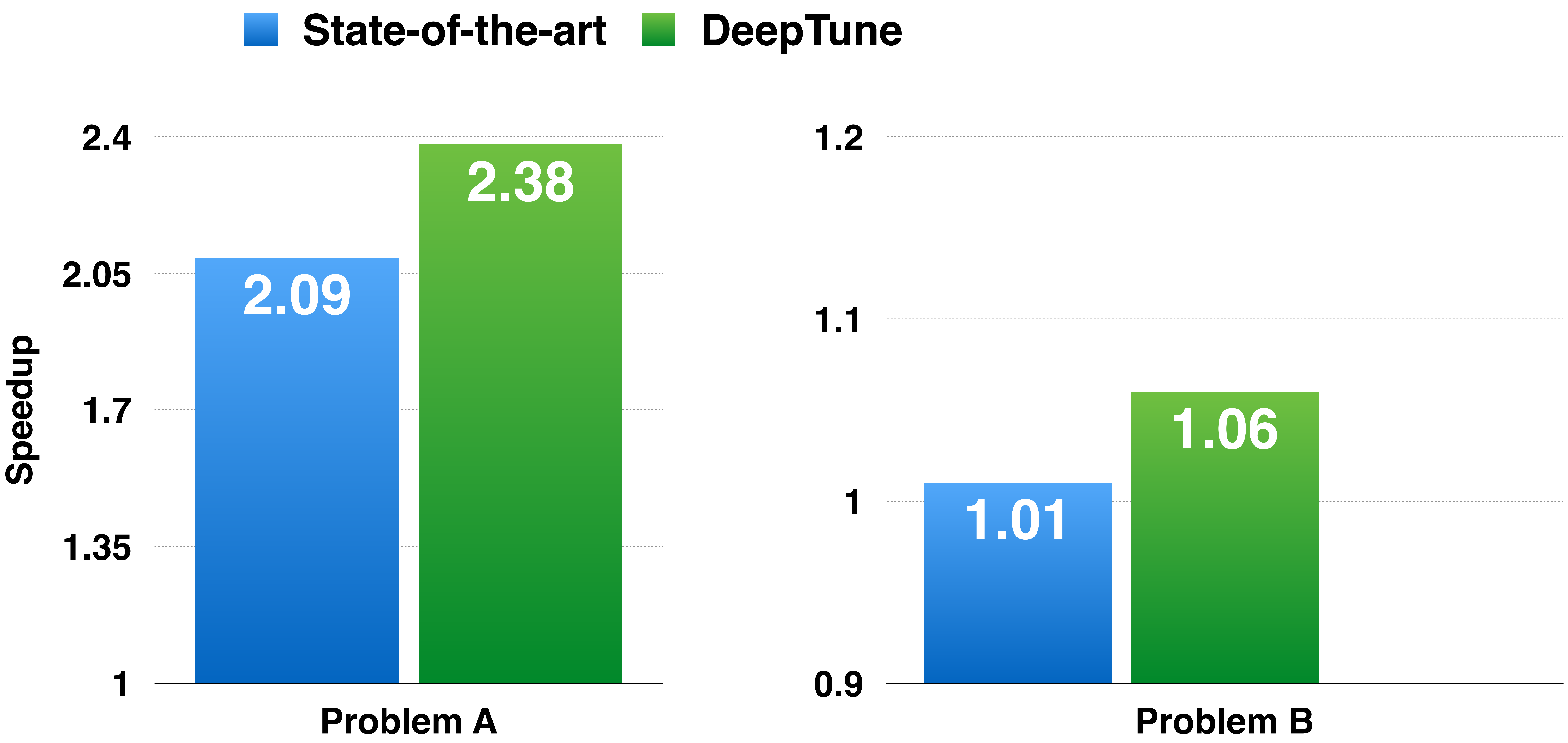
Deep Neural Networks

↓

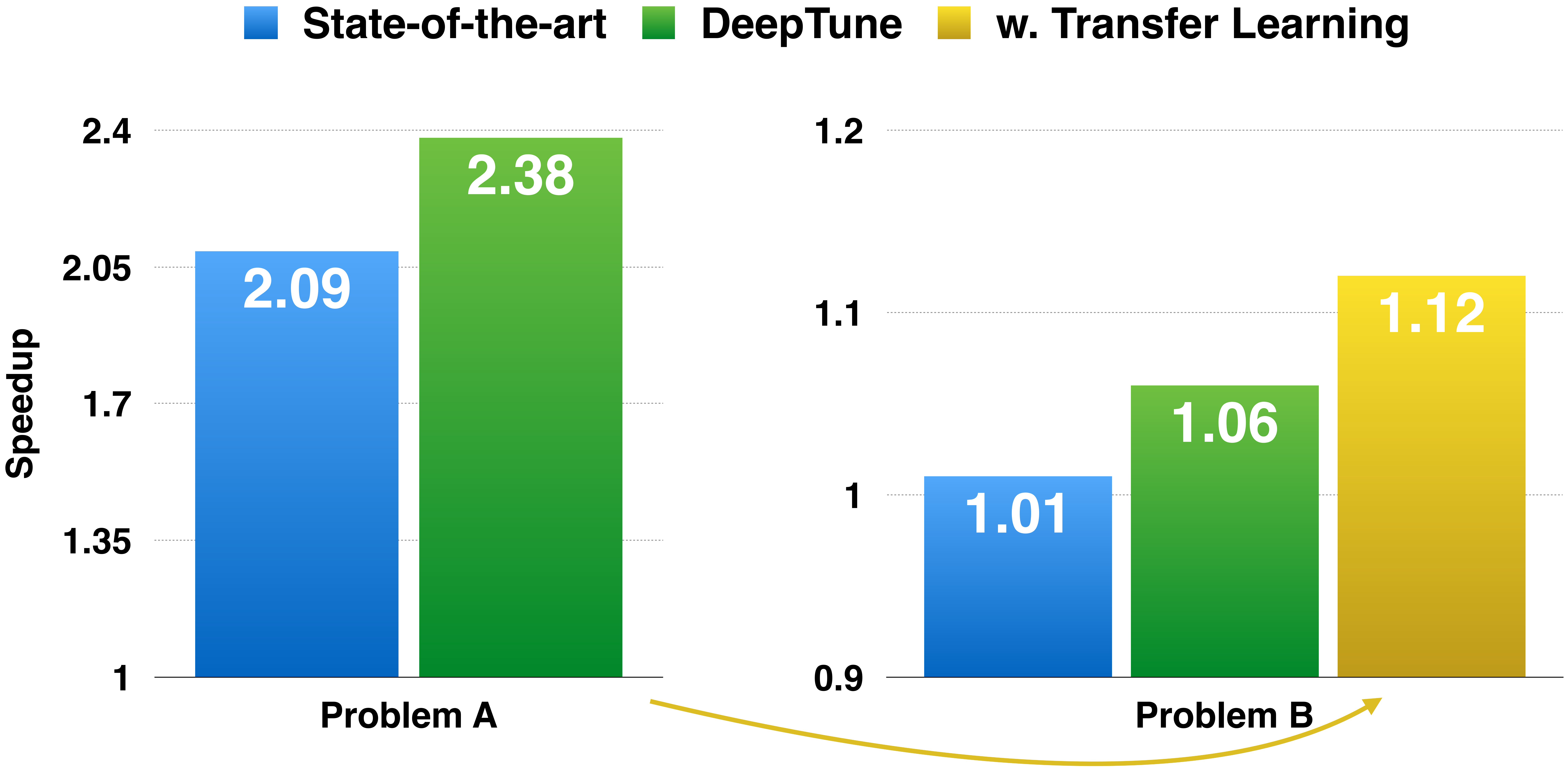
Predicted Optimization

```
{CF:1, CF:2, CF:4, CF:8, CF:16, CF:32}
```

14% and 12% improvements over state-of-the-art



14% and 12% improvements over state-of-the-art



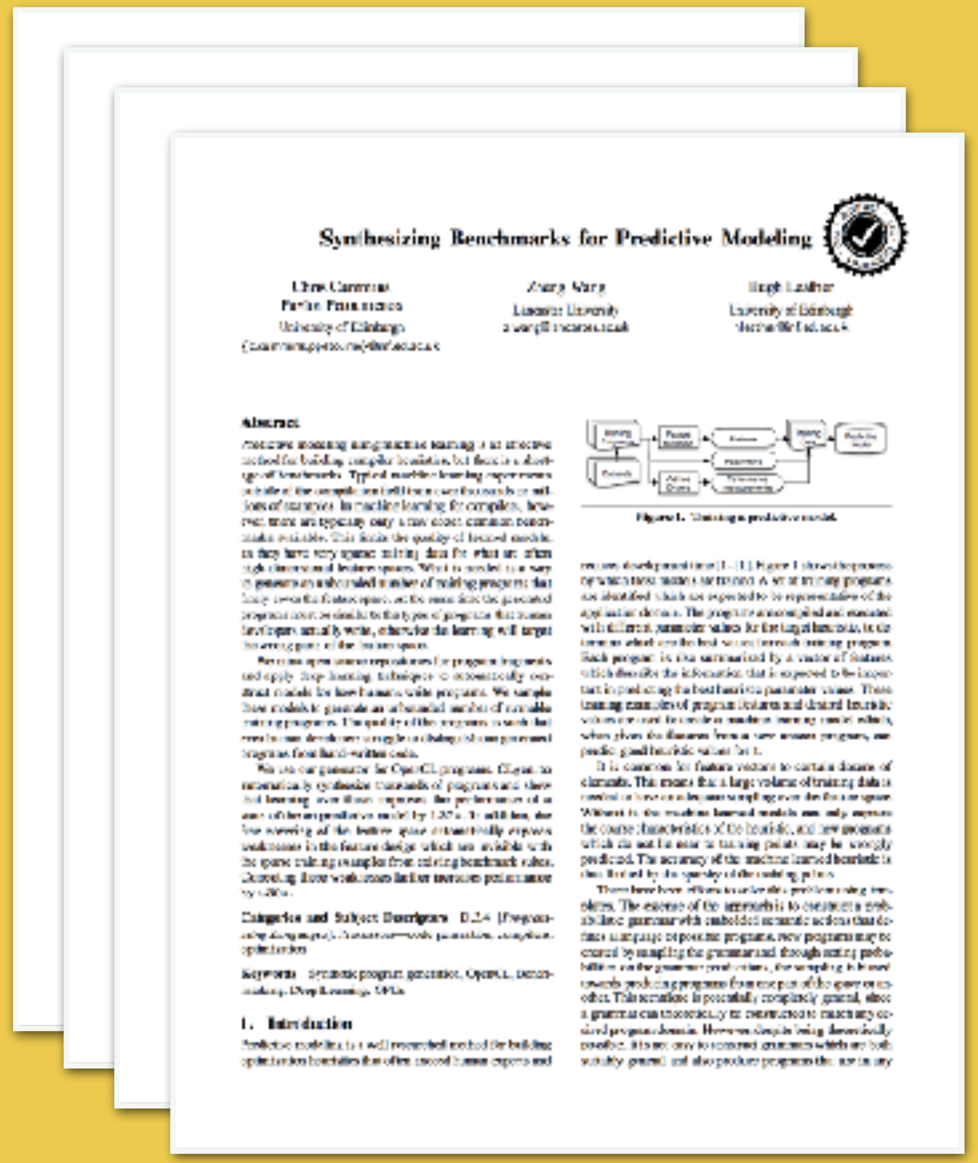
End-to-end Deep Learning of Optimisation Heuristics

problem: feature design is hard

“featureless” heuristic learning from raw program code

outperforms expert approach

first application of learning *across* optimisation domains


Synthesizing Benchmarks for Predictive Modeling

Synthesizing Benchmarks for Predictive Modeling

https://chriscummins.cc/cgo17


End-to-end Deep Learning of Optimization Heuristics

End-to-end Deep Learning of Optimisation Heuristics

https://chriscummins.cc/pact17