Autotuning OpenCL Workgroup Size for Stencil Patterns

Chris Cummins

THE UNIVERSITY of EDINBURGH informatics

EPSRC Centre for Doctoral Training in Pervasive Parallelism

EPSRC Engineering and Physical Sciences Research Council

http://chriscummins.cc

Stencils & Workgroup size

Stencils & Workgroup size

input *stencil* output

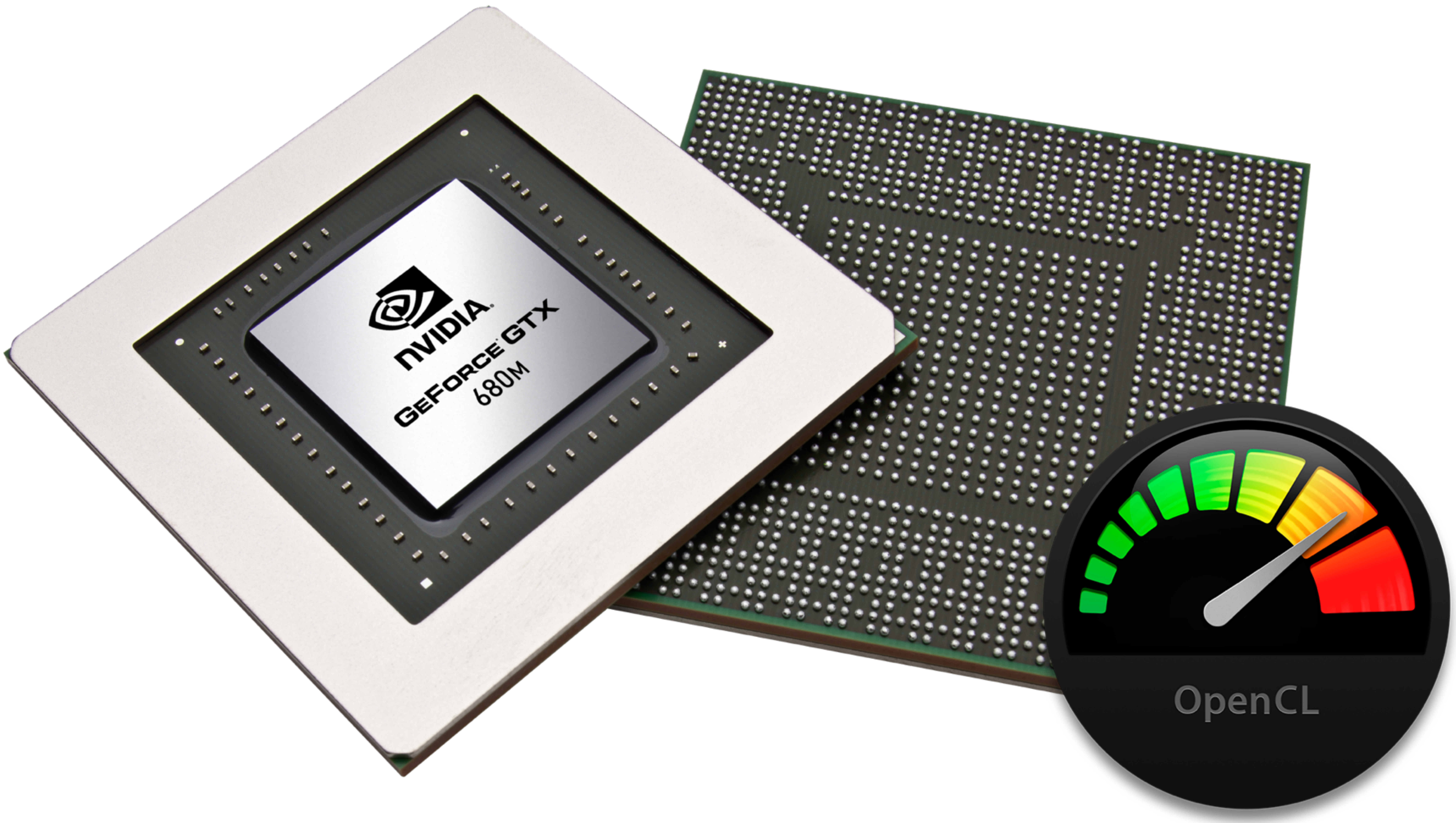

element
border region
input
stencil
output


10 ^ 6 elements
10 ^ 6 border regions
input
stencil
output


10 ^ 6 elements
10 ^ 6 border regions
input
stencil
output
Multiple independent computations

input ***stencil*** **output**

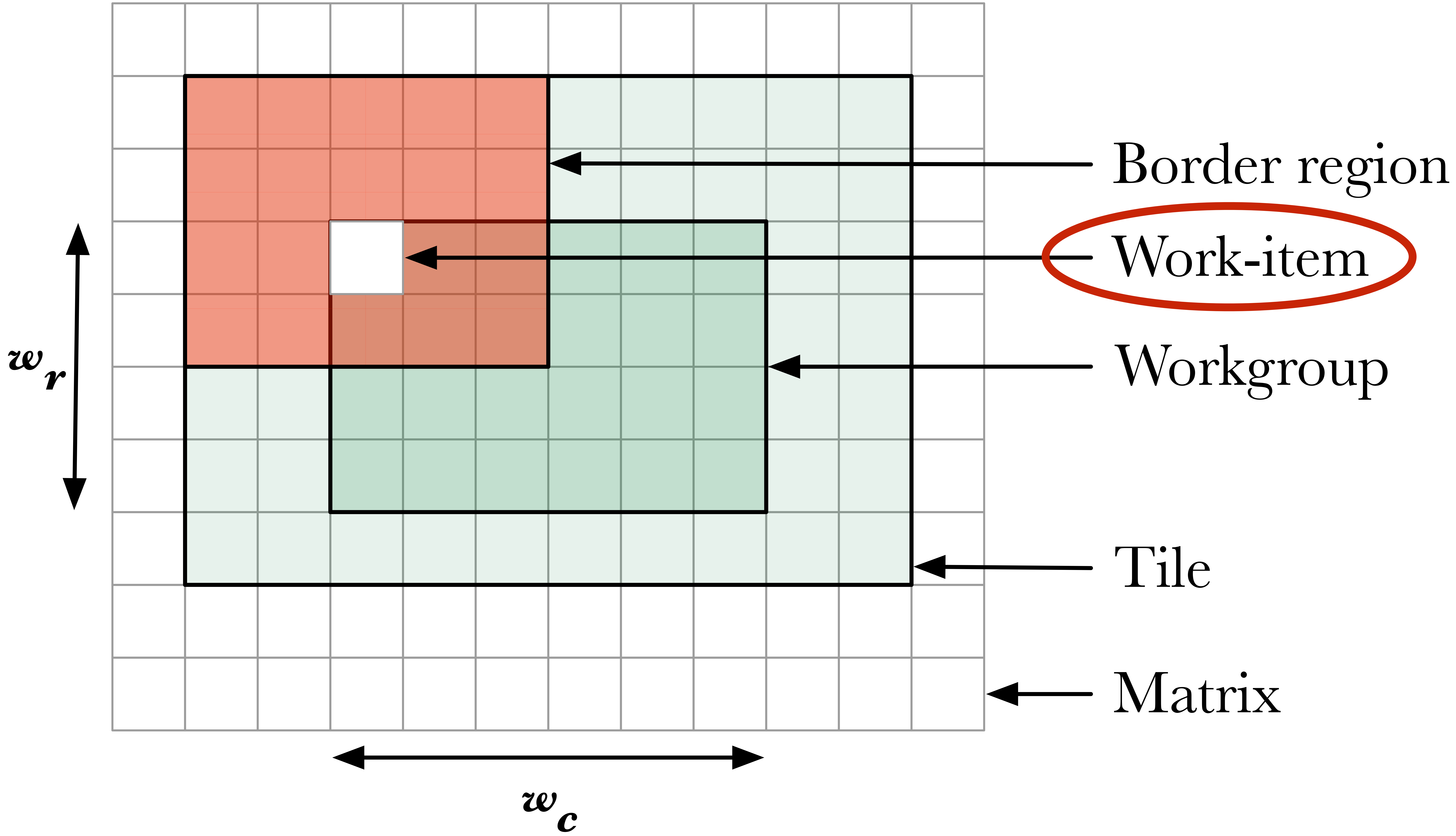
Multiple (overlapping) memory accesses


NVIDIA
GeForce GTX
680M
OpenCL


element
border region
input
stencil
output


element
border region
input
stencil kernel output


~~element~~ work-item
border region
input
stencil kernel output


Border region
Work-item
Workgroup
Tile
Matrix
w_r
w_c

Stencils & Workgroup size

Stencils & Workgroup size


Border region
Work-item
Workgroup
Tile
Matrix
w_r
w_c

Workgroup size affects

mapping to SIMD hardware.
device occupancy.
local memory utilisation.

Pop Quiz!

What is the best workgroup size for …

Gaussian blur, 512px x 512px, floats, on:

1. AMD HD7990?
2. Nvidia GTX Titan?
3. Intel i7-3820?

What is the best workgroup size for …

Gaussian blur, 512px x 512px, floats, on:

1. AMD HD7990? 64 x 4
2. Nvidia GTX Titan? 96 x 4
3. Intel i7-3820? 40 x 24

What is the best workgroup size for ...

Nvidia GTX 590, 4096 x 4096 elements running:

1. Sobel edge detection?
2. Heat equation?
3. Game of life?

What is the best workgroup size for ...

Nvidia GTX 590, 4096 x 4096 elements running:

1. Sobel edge detection? 256 x 2
2. Heat equation? 128 x 2
3. Game of life? 32 x 6

What is the best workgroup size for …

1. Intel i5-2430, game of life, 4096 x 4096?
2. Nvidia GTX 690, threshold, 512 x 512?
3. Intel i7-3820, NMS, 512 x 512?

What is the best workgroup size for …

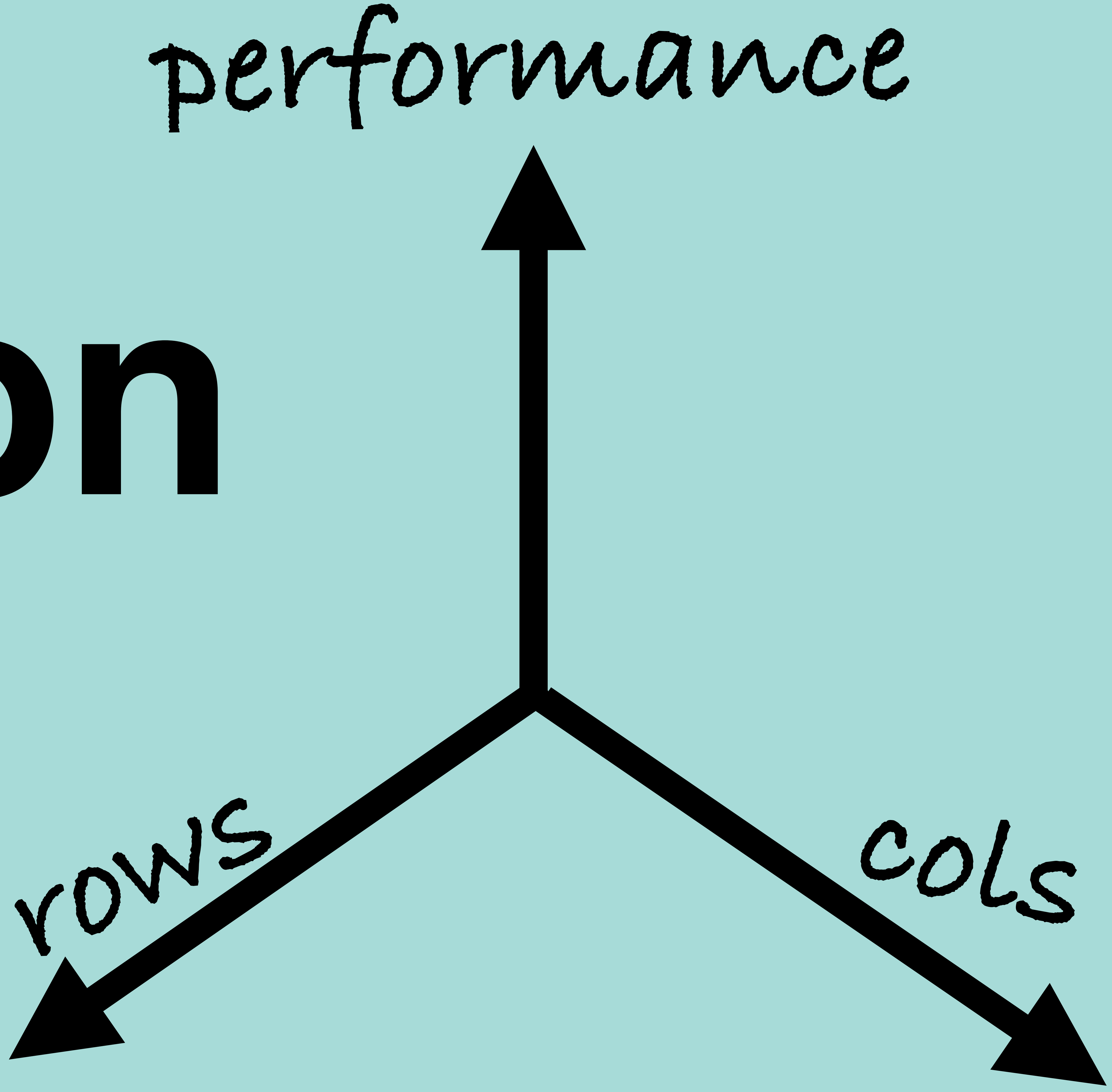
1. Intel i5-2430, game of life, 4096 x 4096? 196 x 20
2. Nvidia GTX 690, threshold, 512 x 512? 32 x 4
3. Intel i7-3820, NMS, 512 x 512? 88 x 8

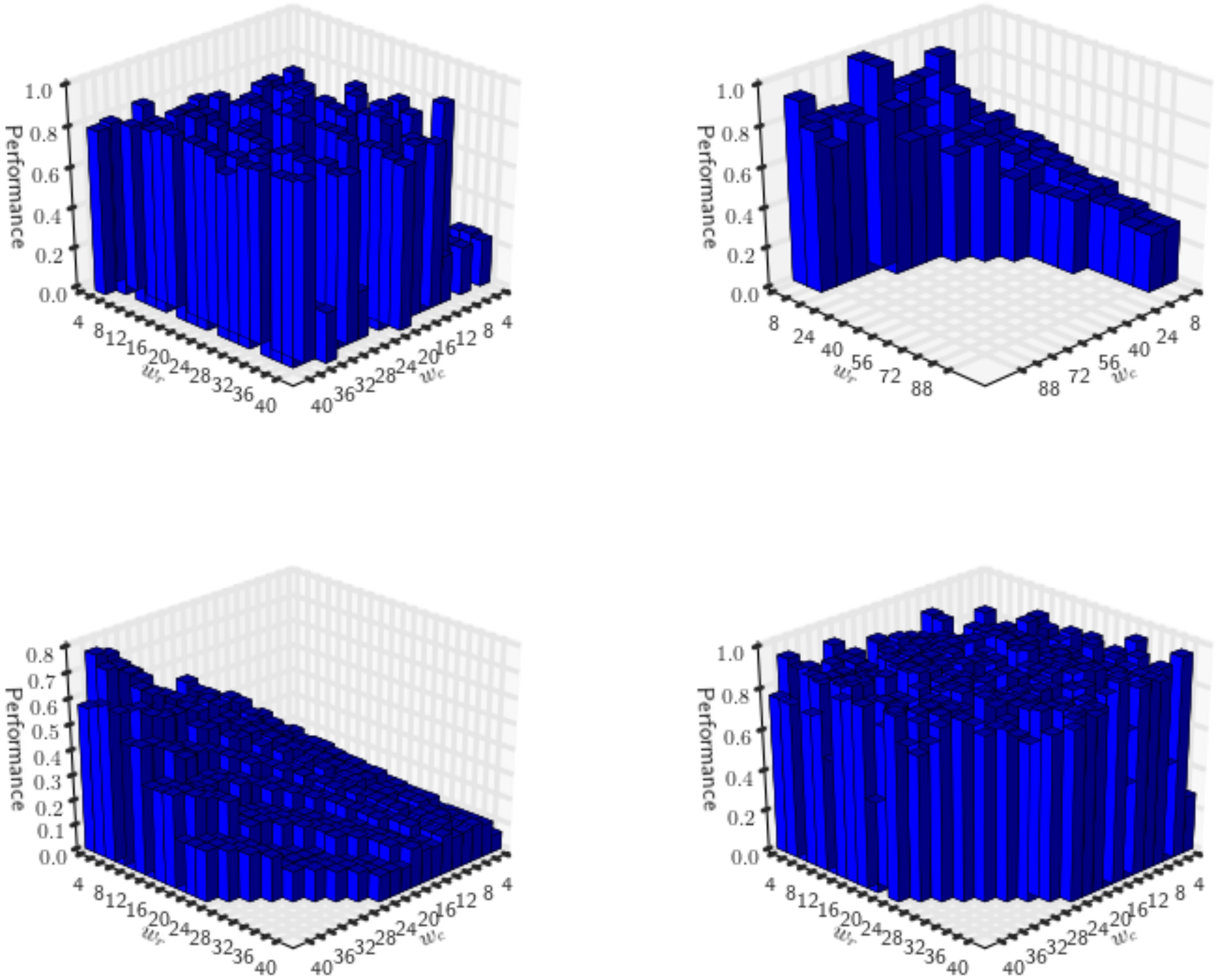
One size does not fit all!

Choosing workgroup size depends on:

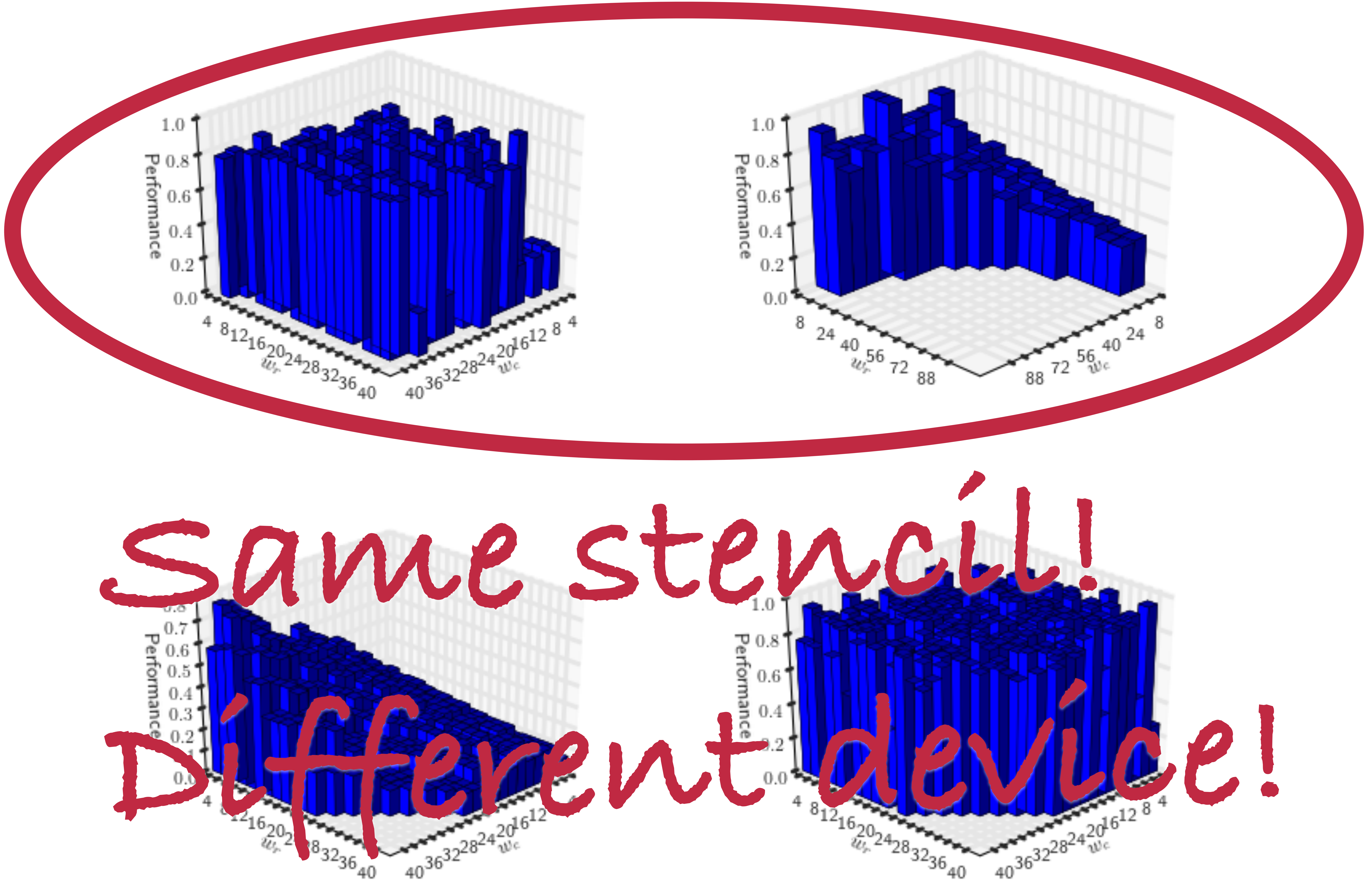
1. Device
2. Program
3. Dataset

Optimisation space

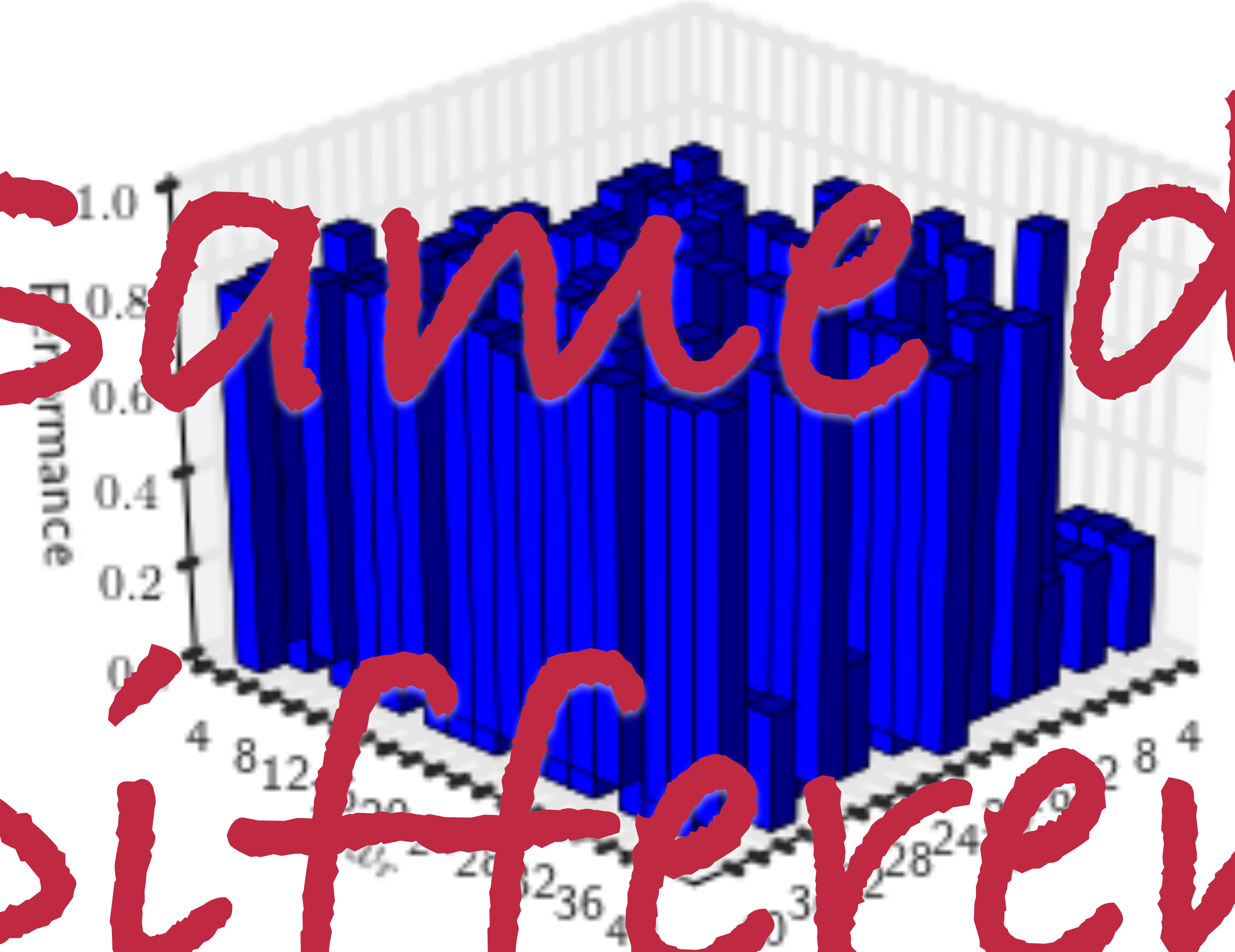


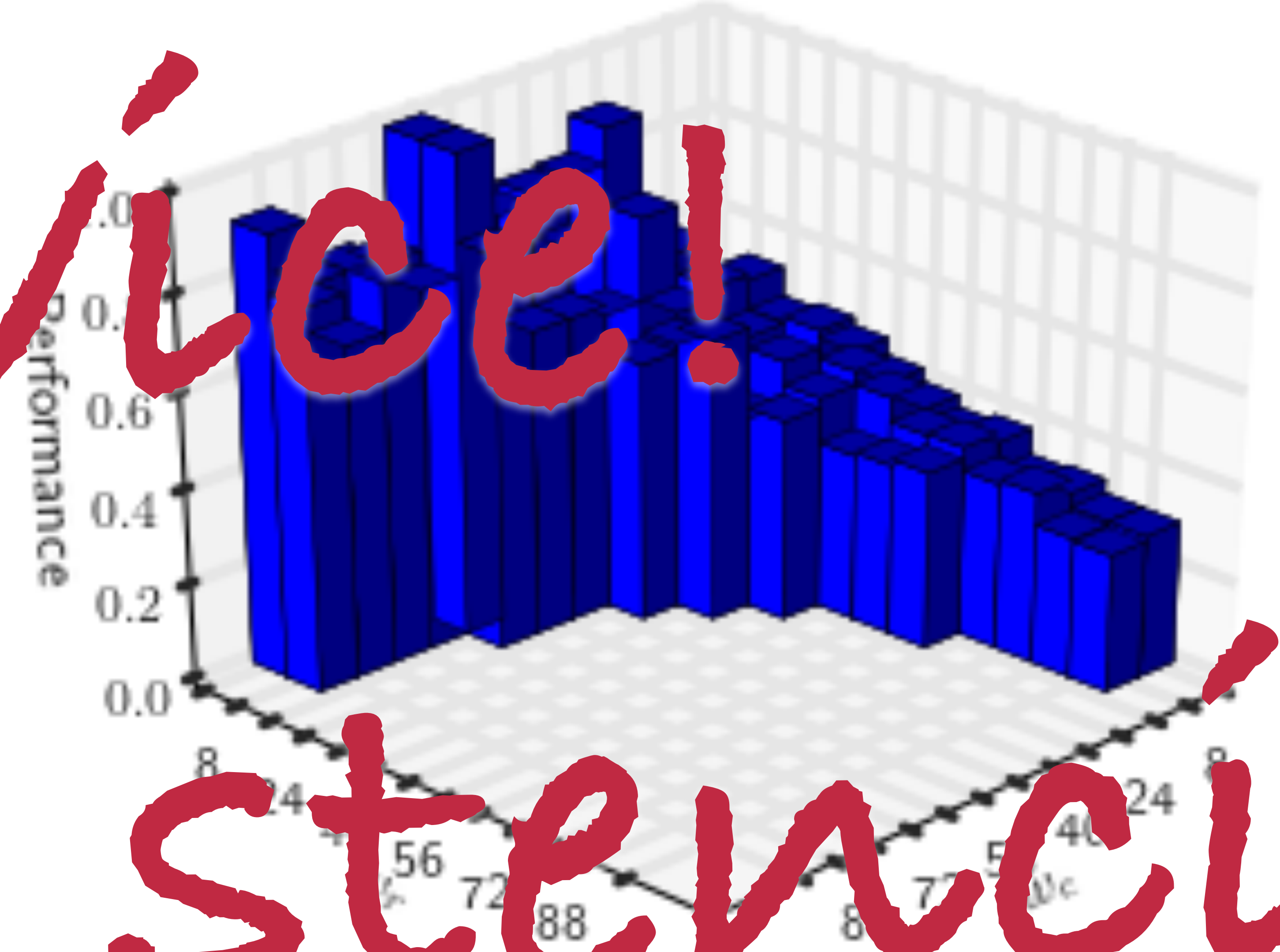


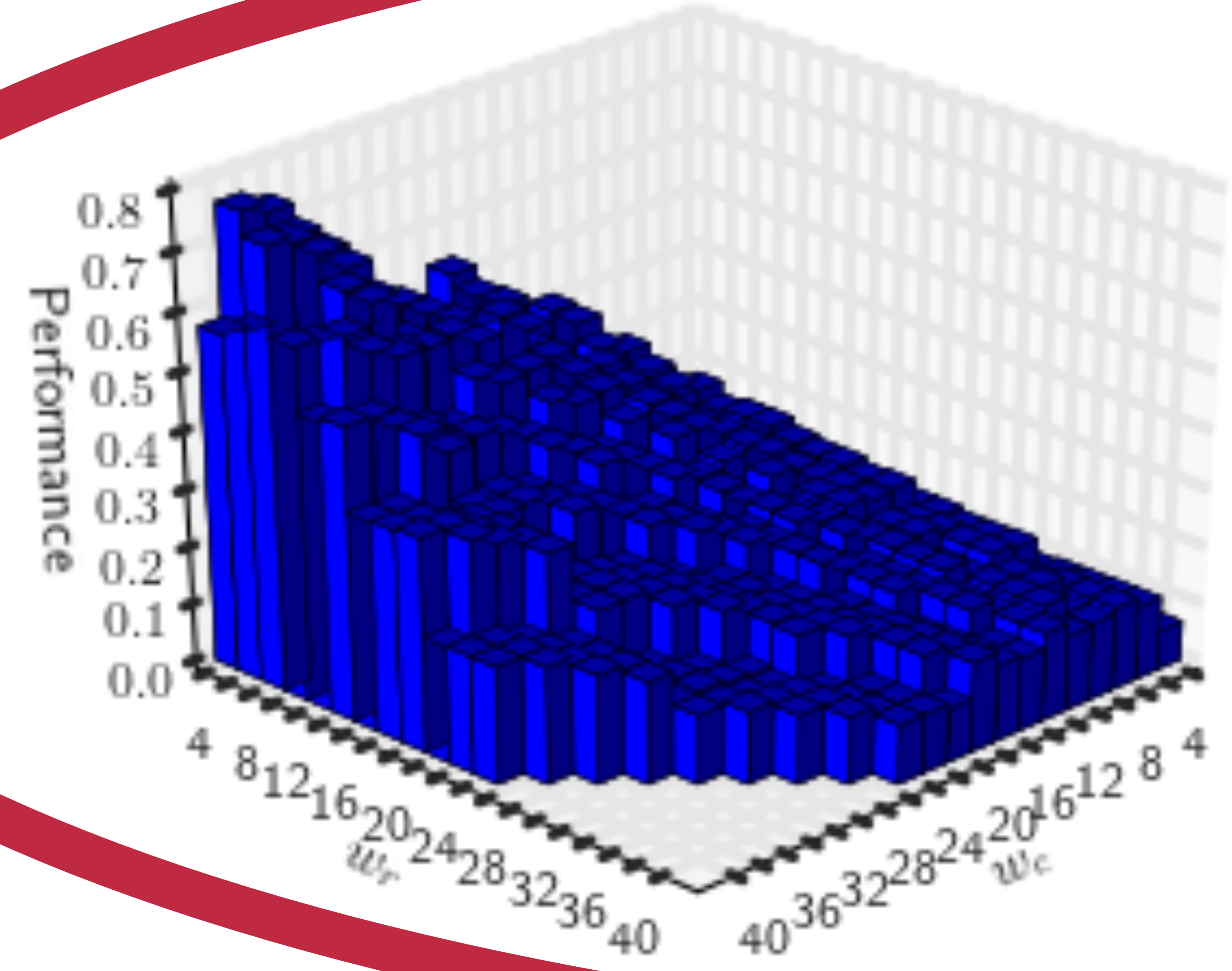
Performance
1.0 0.8 0.6 0.4 0.2 0.0
w_r 4 8 12 16 20 24 28 32 36 40
w_c 40 36 32 28 24 20 16 12 8 4
Performance
1.0 0.8 0.6 0.4 0.2 0.0
w_r 8 24 40 56 72 88
w_c 88 72 56 40 24 8
Performance
0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0
w_r 4 8 12 16 20 24 28 32 36 40
w_c 40 36 32 28 24 20 16 12 8 4
Performance
1.0 0.8 0.6 0.4 0.2 0.0
w_r 4 8 12 16 20 24 28 32 36 40
w_c 40 36 32 28 24 20 16 12 8 4

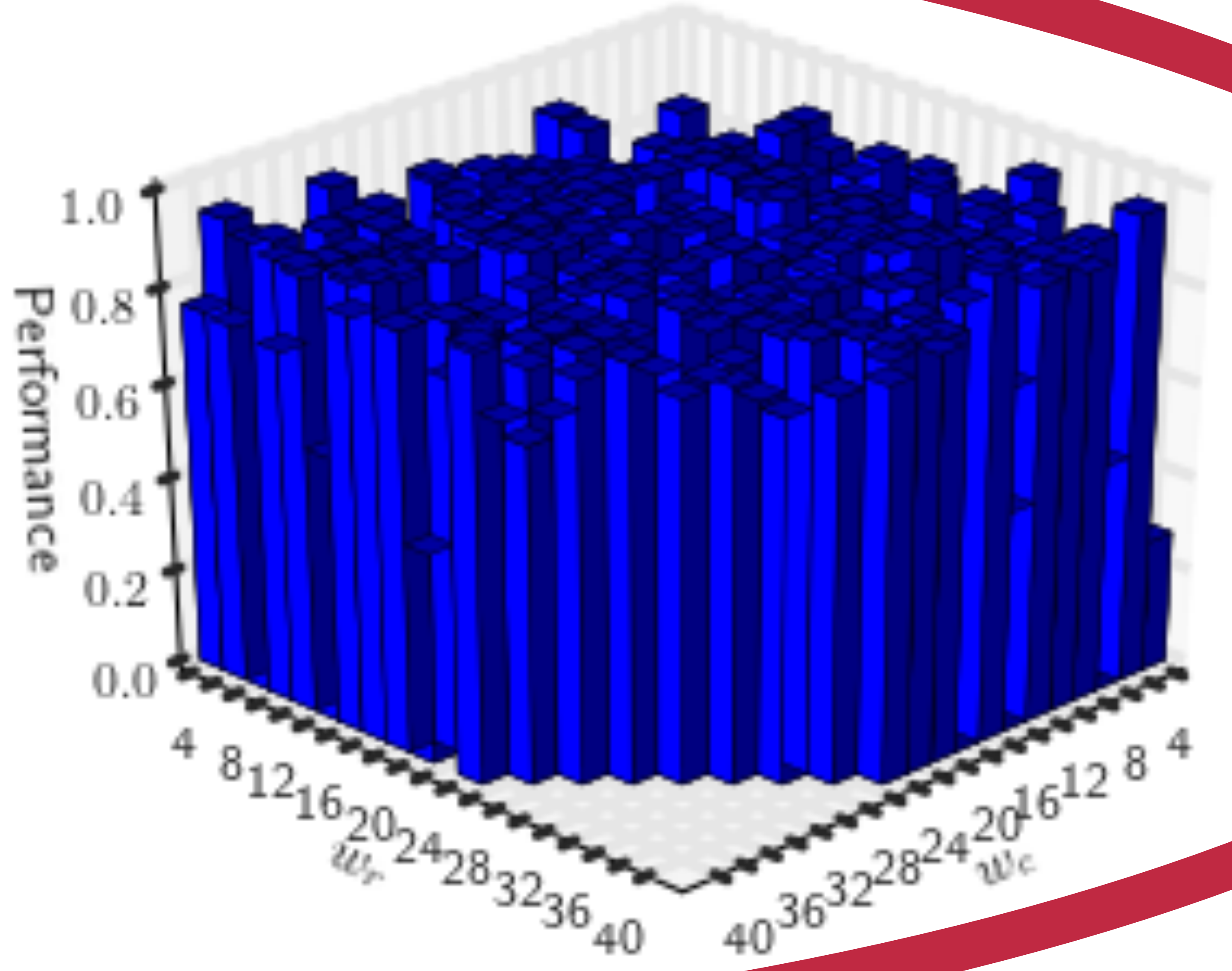

Performance
w_r
w_c
Same stencil!
Different device!

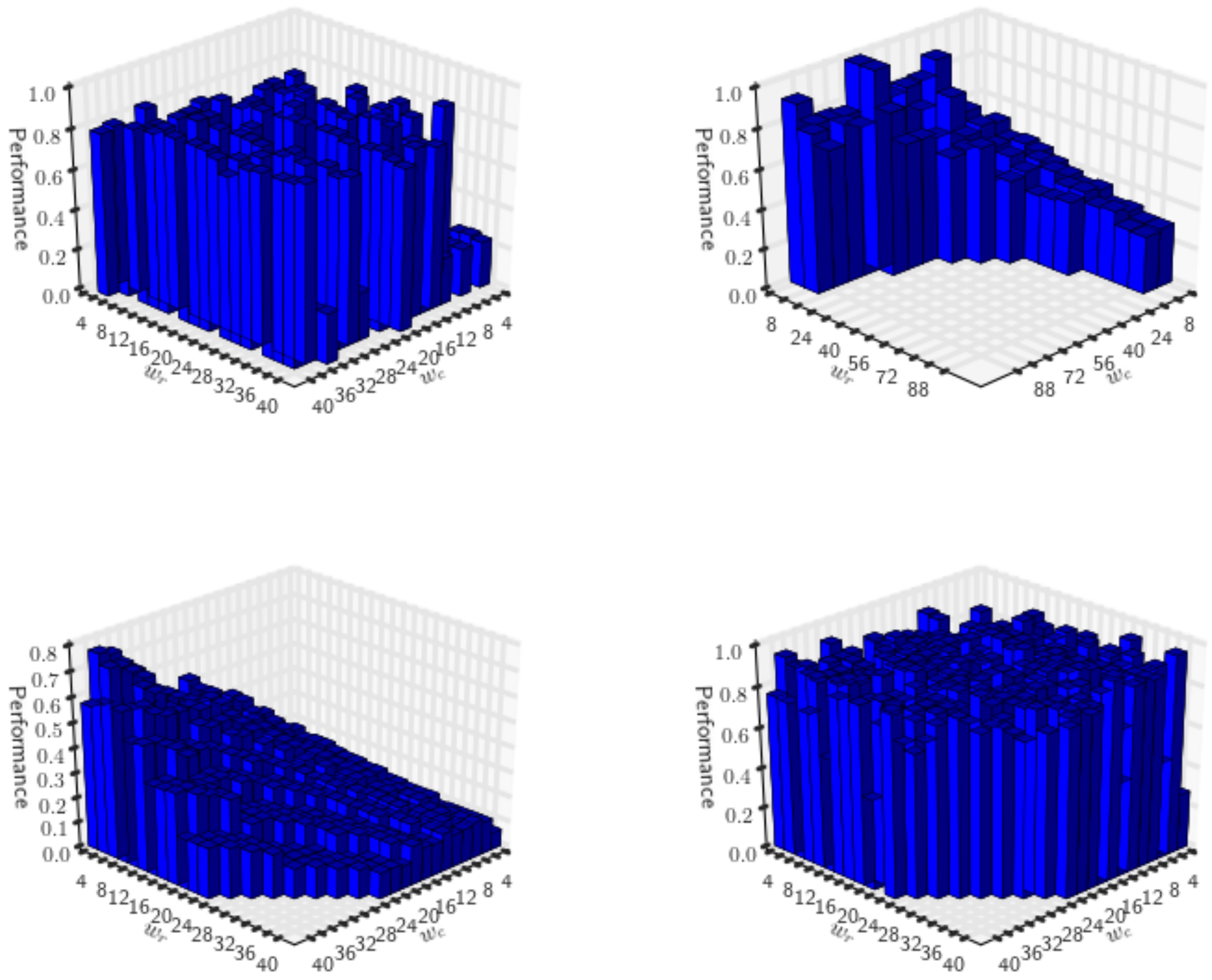
Same device!
Different stencil!











Performance
1.0 0.8 0.6 0.4 0.2 0.0
4 8 12 16 20 24 28 32 36 40
w_r
40 36 32 28 24 20 16 12 8 4
w_c
Performance
1.0 0.8 0.6 0.4 0.2 0.0
8 24 40 56 72 88
w_r
88 72 56 40 24 8
w_c
Performance
0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0
4 8 12 16 20 24 28 32 36 40
w_r
40 36 32 28 24 20 16 12 8 4
w_c
Performance
1.0 0.8 0.6 0.4 0.2 0.0
4 8 12 16 20 24 28 32 36 40
w_r
40 36 32 28 24 20 16 12 8 4
w_c

Workgroup Size + Stencils

1. Non-linear, non-continuous
2. Device, program, dataset
3. Not all values are legal

Autotuning

Set a workgroup size

Execute and time program

Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program

Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program

Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program
Set a workgroup size
Execute and time program

Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program

… (continue until done / bored)

Pick the best one you tried

Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program

… (continue until done / bored)

Pick the best one you tried

(iterative compilation)

BAD!

Takes a loooong time

BAD!

Takes a loooong time

BAD!

Must be repeated for every new "x"

device

program

dataset

Let’s improve

Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program

… (continue until done / bored)

Pick the best one you tried

Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program

Set a workgroup size

Execute and time program

… (continue until done / bored)

1 data point

Pick the best one you tried

Collect data points
Extract “features”
Train machine learning classifier

Extract “features”
Input to classifier

GOOD!

Can make *predictions* on unseen "x"

device

program

dataset

GOOD!

Can make *predictions* on unseen "x"

device

dataset

program

GOOD!

Many unanswered questions ...

Questions:

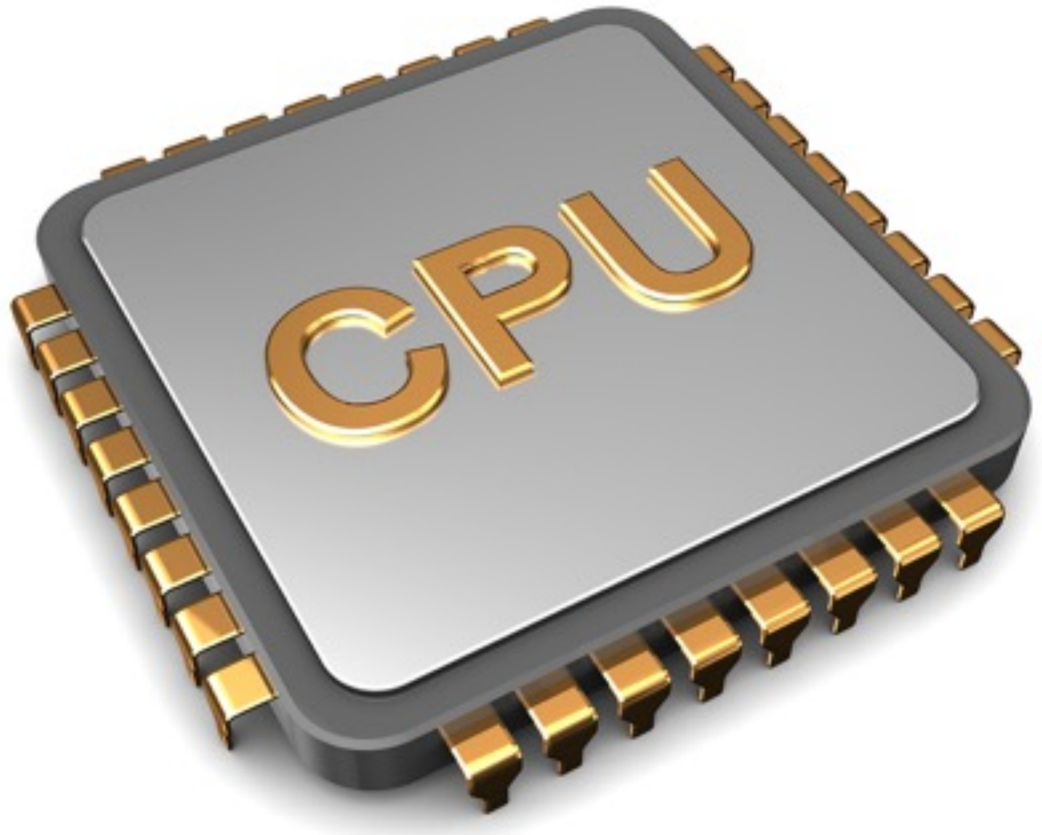
1. What features do we need?
2. What programs do we train on?
3. How do we make predictions?

Questions:

1. What features do we need?
2. What programs do we train on?
3. How do we make predictions?

1. Device
2. Kernel
3. Dataset

1. Device
2. Kernel
3. Dataset



or

How many compute units?

How much memory?

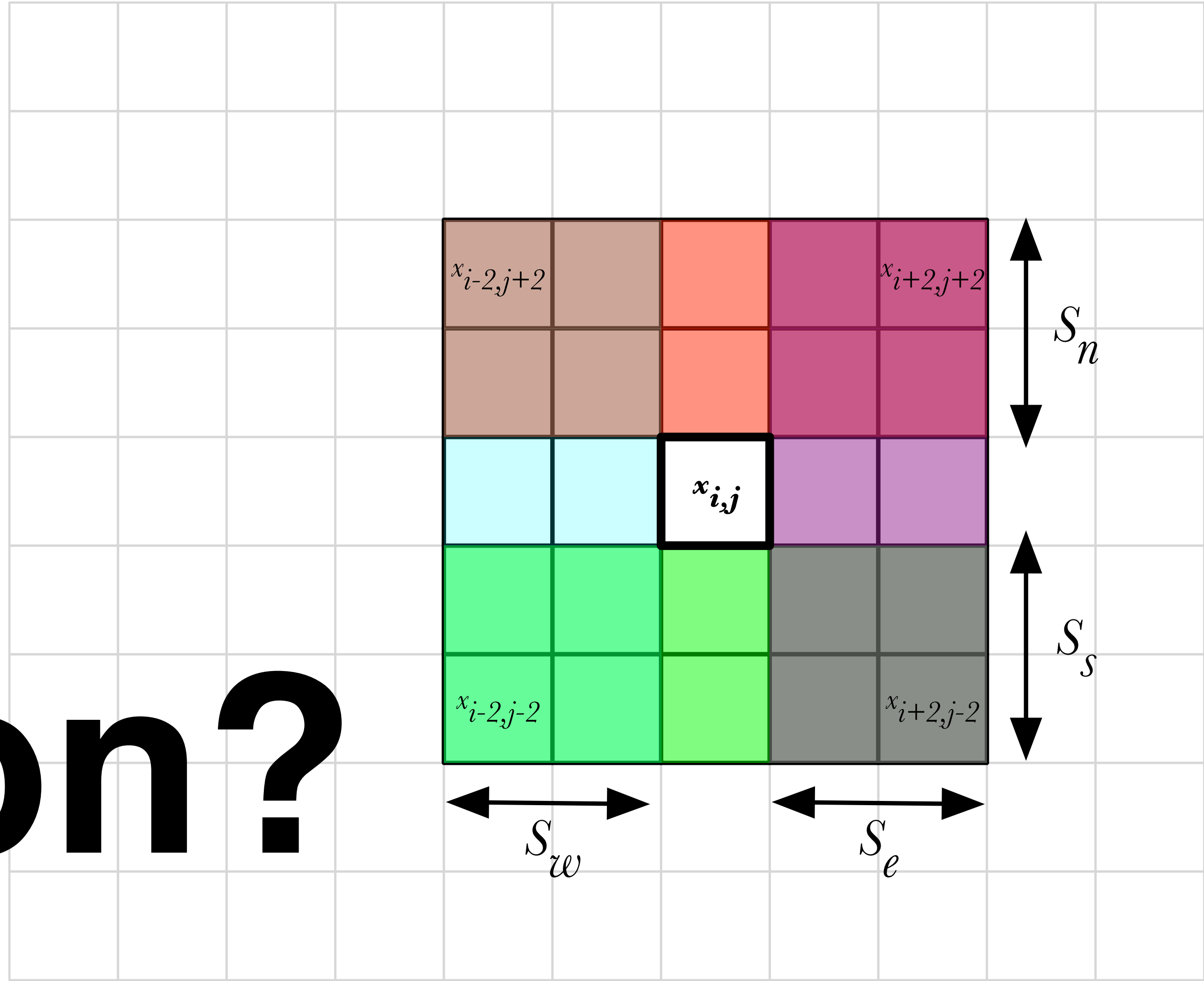
Cache size?

etc.

1. Device
2. Kernel
3. Dataset

1. Device
2. Kernel
3. Dataset

1. Device
2. Kernel
3. Dataset



How big is border region?

What shape is it?

How many instructions?

What type of instructions?

etc.

1. Device
2. Kernel
3. Dataset

1. Device
2. Kernel
3. Dataset

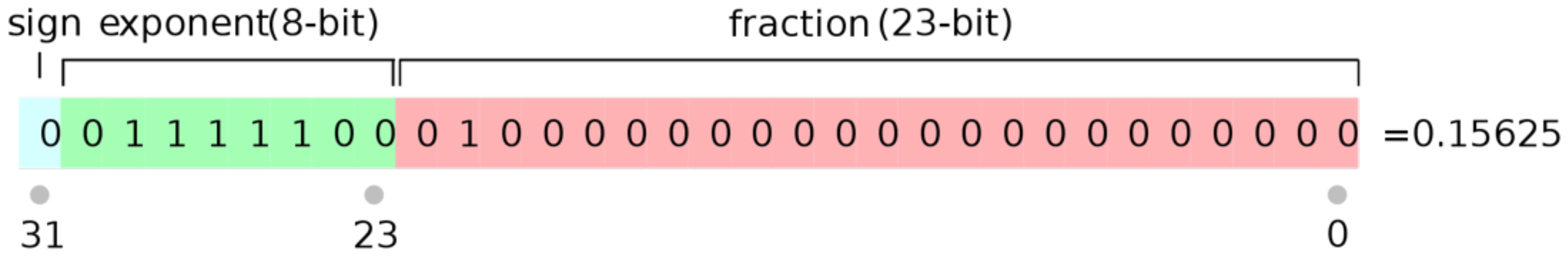
1. Device
2. Kernel
3. Dataset

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & & & A_{2n} \\ \vdots & & & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix}$$

How big is the data?

What type is the input?

What type is the output?



1. Device
2. Kernel
3. Dataset

1. Device
2. Kernel
3. Dataset

Questions:

1. What features do we need?
2. What programs do we train on?
3. How do we make predictions?

Questions:

1. What features do we need? ✓
2. What programs do we train on?
3. How do we make predictions?

1. **Learn by example**
2. **Learn by exploration**

Use benchmark programs
Hope that they are representative

1. Learn by example
2. Learn by exploration

1. **Learn by example**
2. **Learn by exploration**

1. Learn by example
2. Learn by exploration

   Create own benchmarks

   Explore (the huge!) program space

Questions:

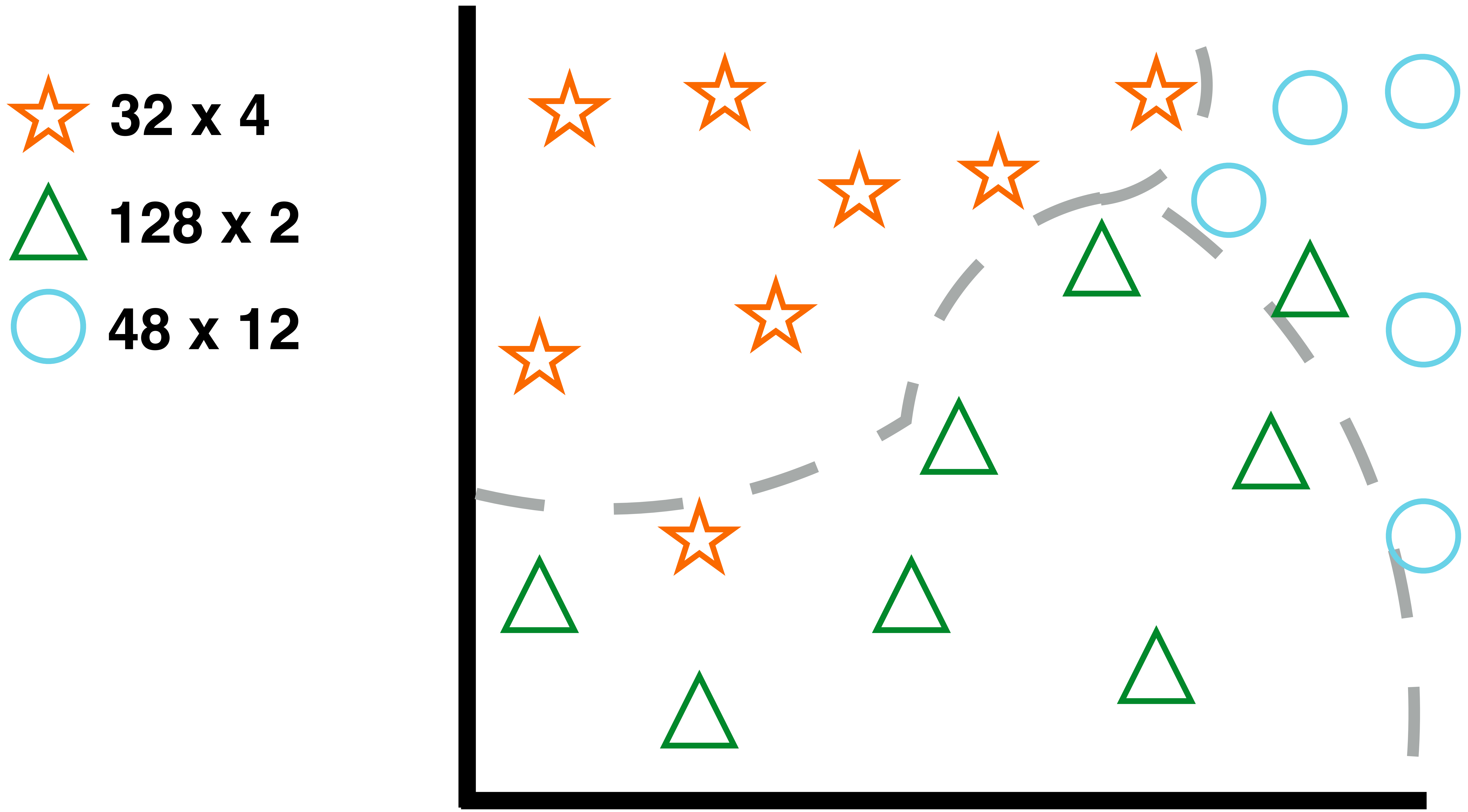
1. What features do we need? ✓
2. What programs do we train on?
3. How do we make predictions?

Questions:

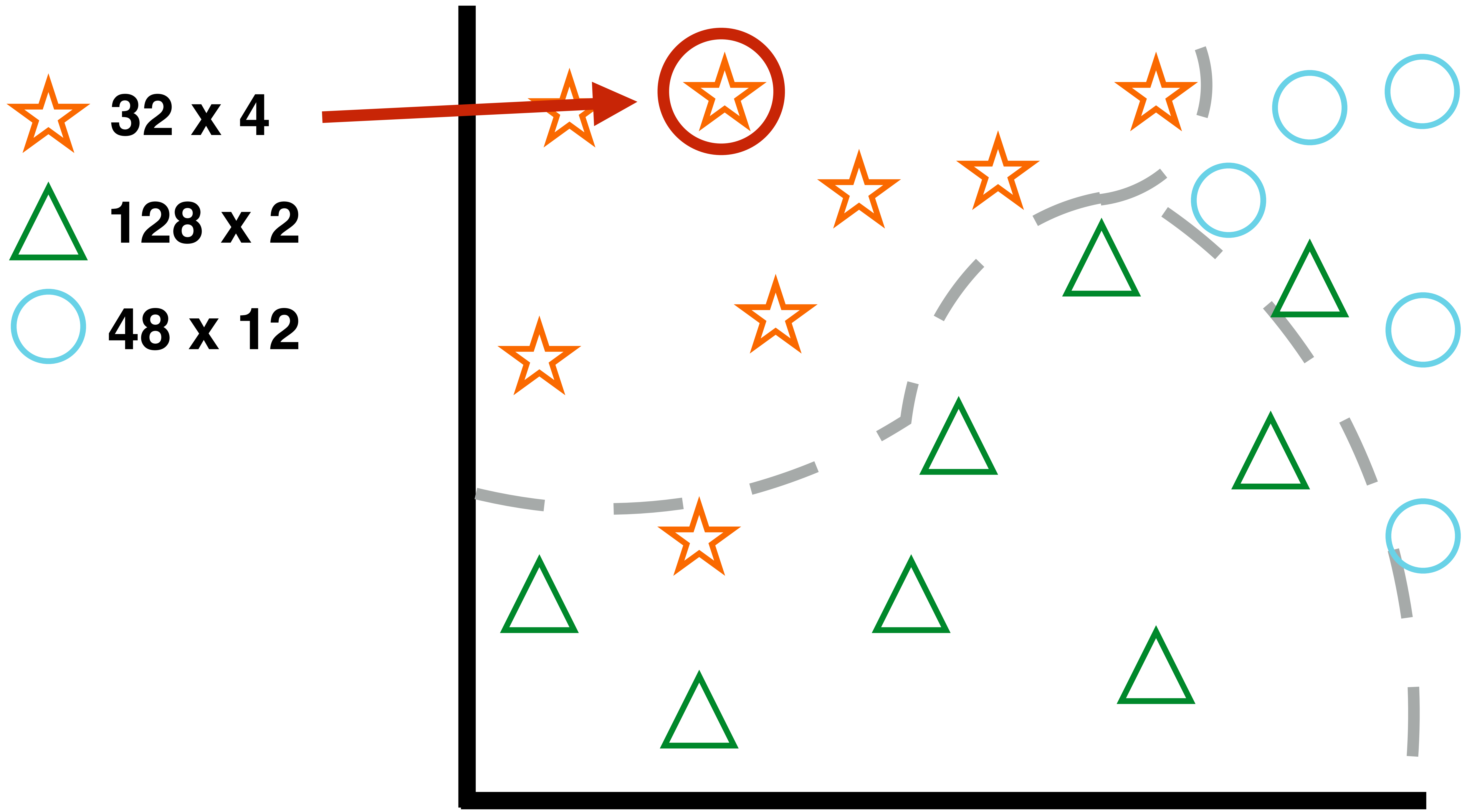
1. What features do we need? ✓
2. What programs do we train on? ✓
3. How do we make predictions?

1. Classifier
2. Runtime Regressor
3. Speedup Regressor

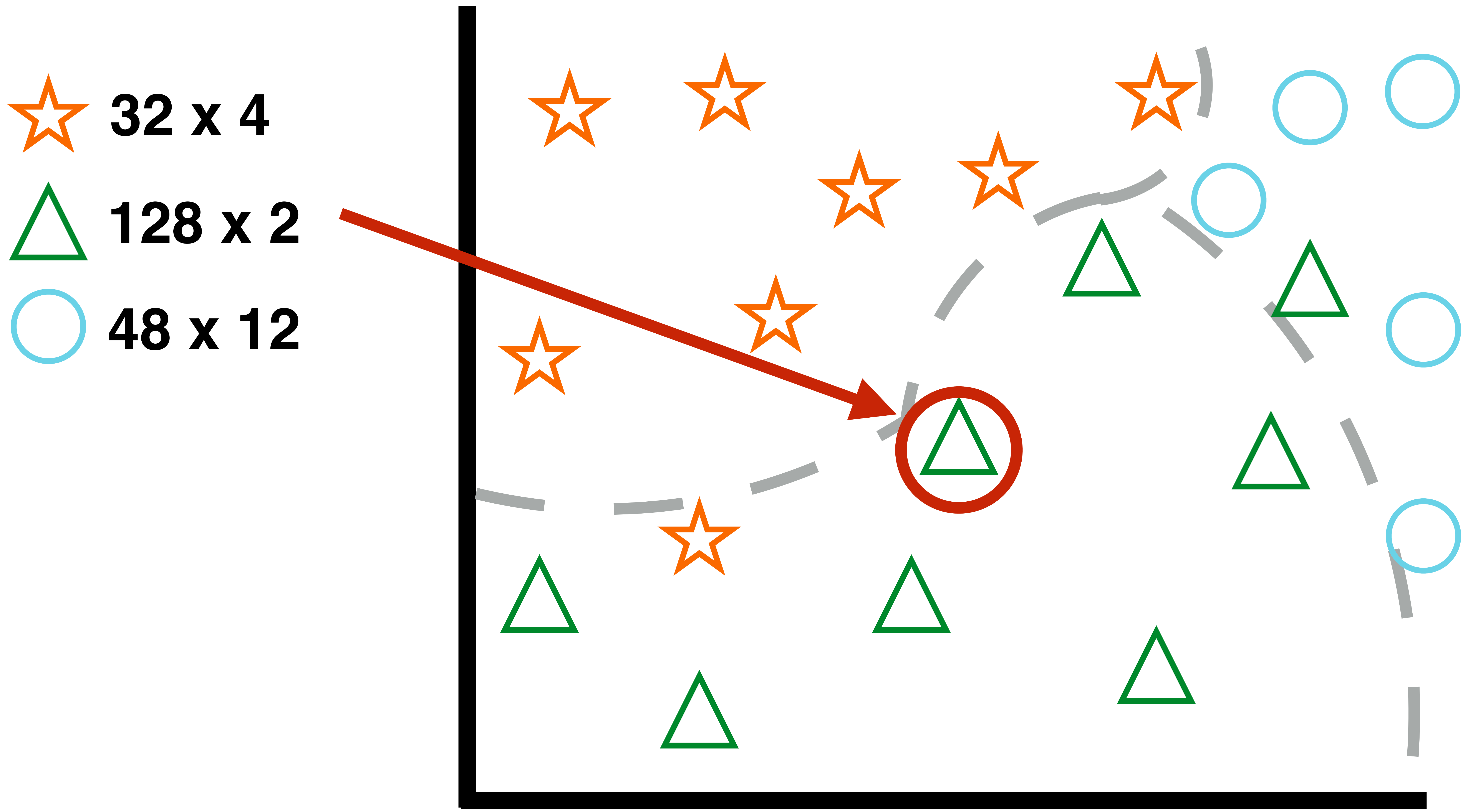
1. Classifier
2. Runtime Regressor
3. Speedup Regressor



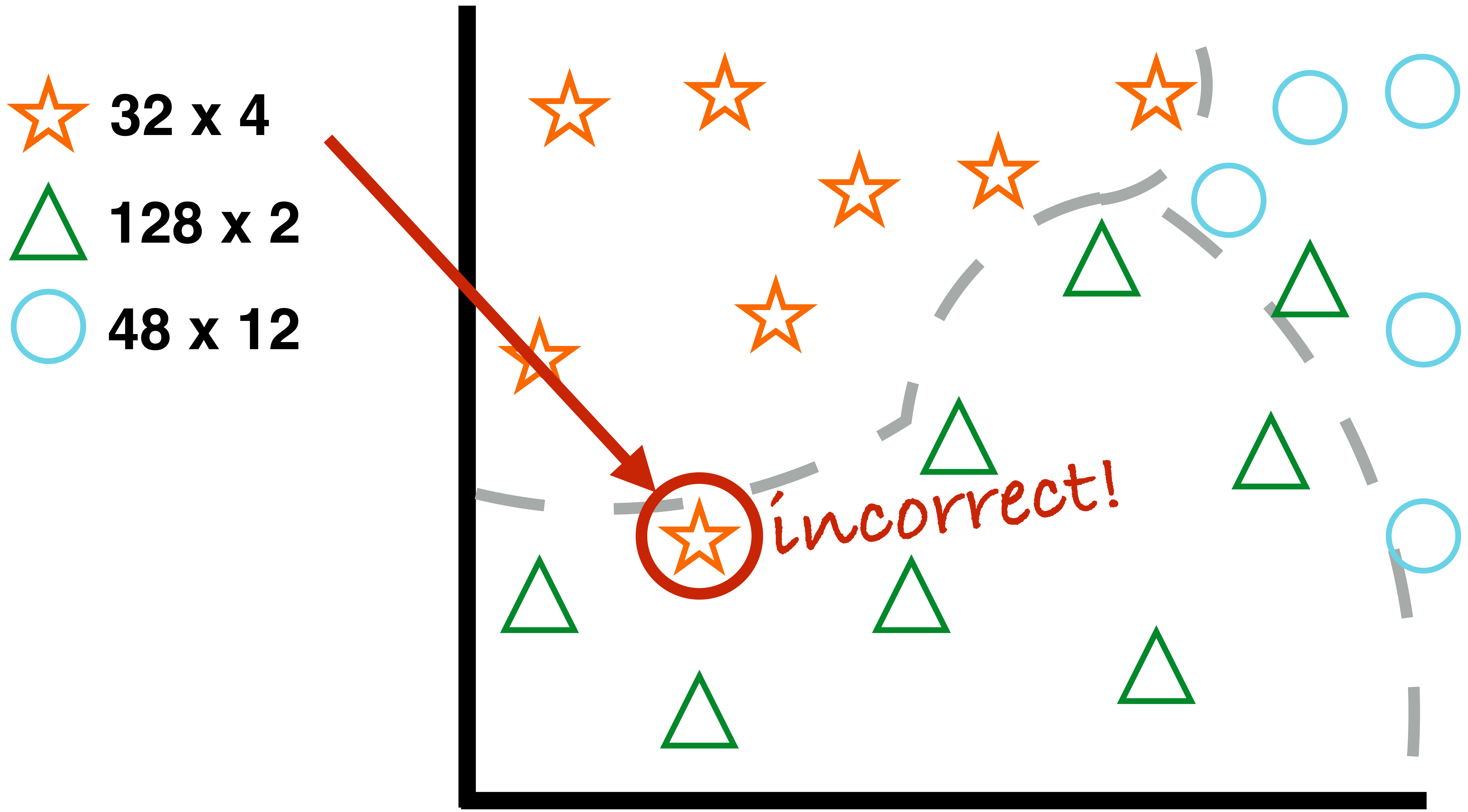
Predict category *(optimal workgroup size)* for scenario



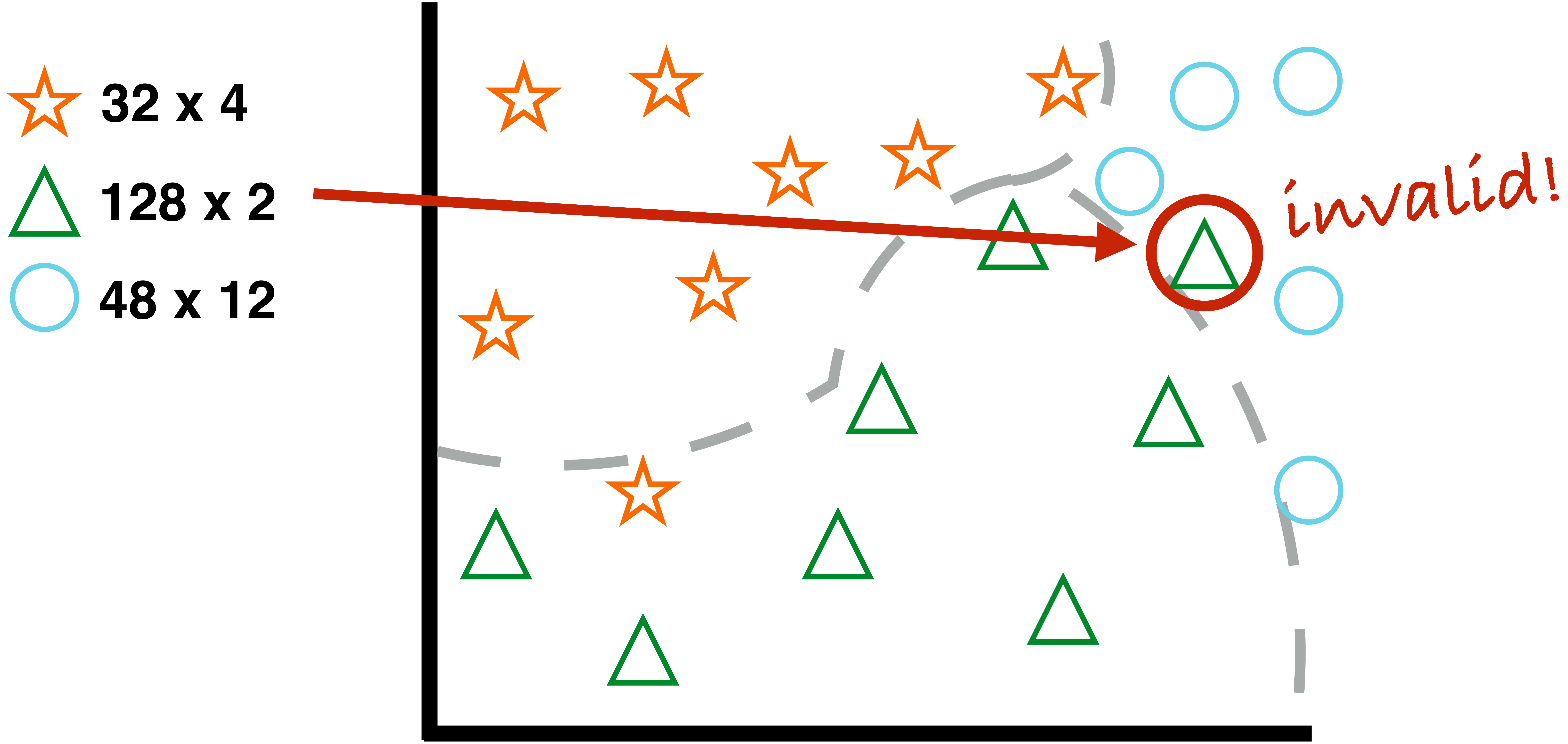
Predict category *(optimal workgroup size)* for scenario



Predict category *(optimal workgroup size)* for scenario



Predict category *(optimal workgroup size)* for scenario



Predict category *(optimal workgroup size)* for scenario

Fallback Handlers

1. Baseline
2. Random
3. Nearest Neighbour

Fallback Handlers

1. **Baseline**
2. Random
3. Nearest Neighbour

"pick something we know is safe"

Fallback Handlers

1. Baseline
2. Random
3. Nearest Neighbour

"pick a random value"

Fallback Handlers

1. Baseline
2. Random
3. **Nearest Neighbour**

"pick the closest value we think will work"

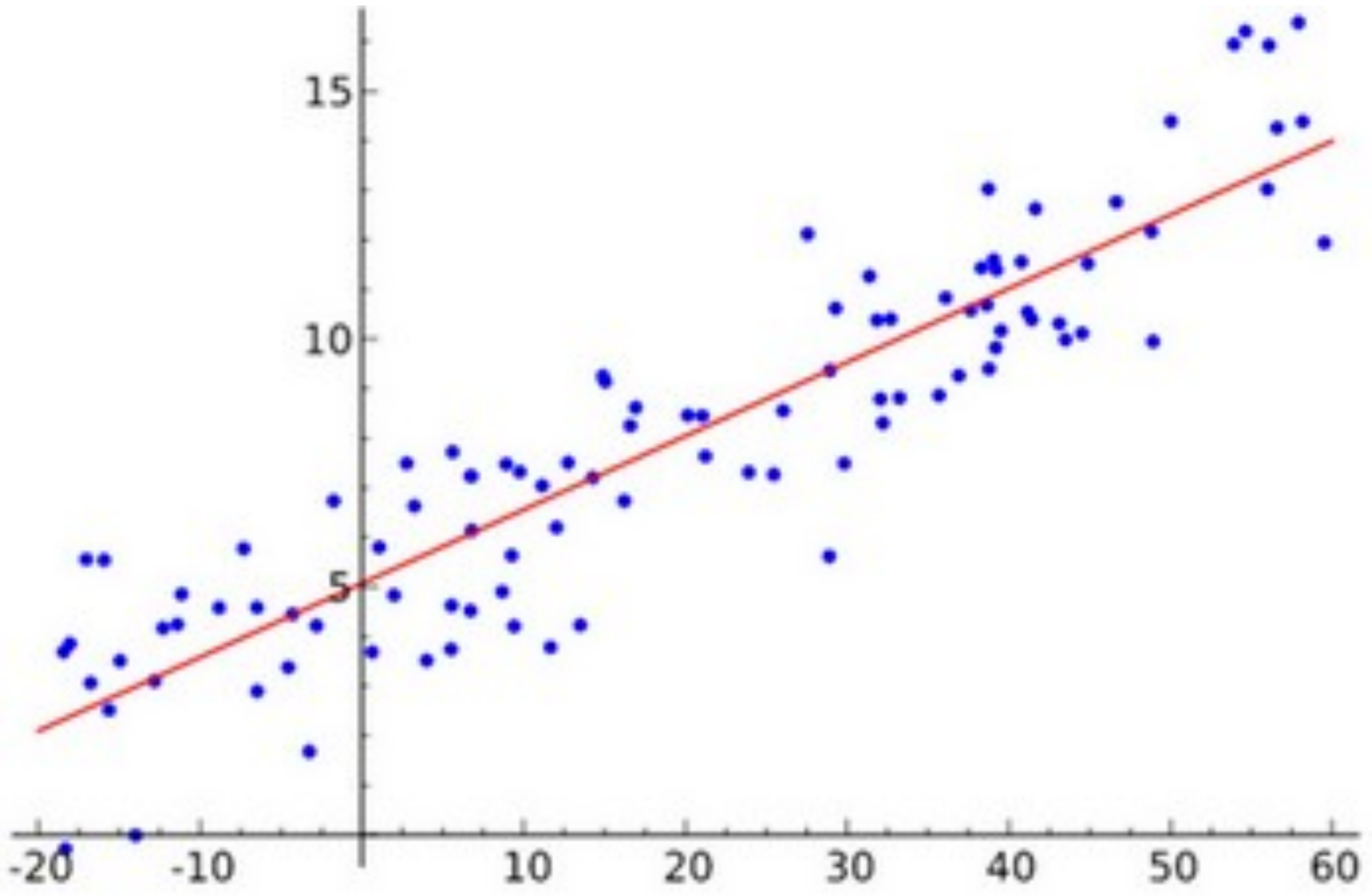
1. Classifier
2. Runtime Regressor
3. Speedup Regressor

1. Classifier
2. Runtime Regressor
3. Speedup Regressor

1. Classifier
2. Runtime Regressor
3. Speedup Regressor

Predict *runtime* of program for workgroup size

Search for *lowest runtime*



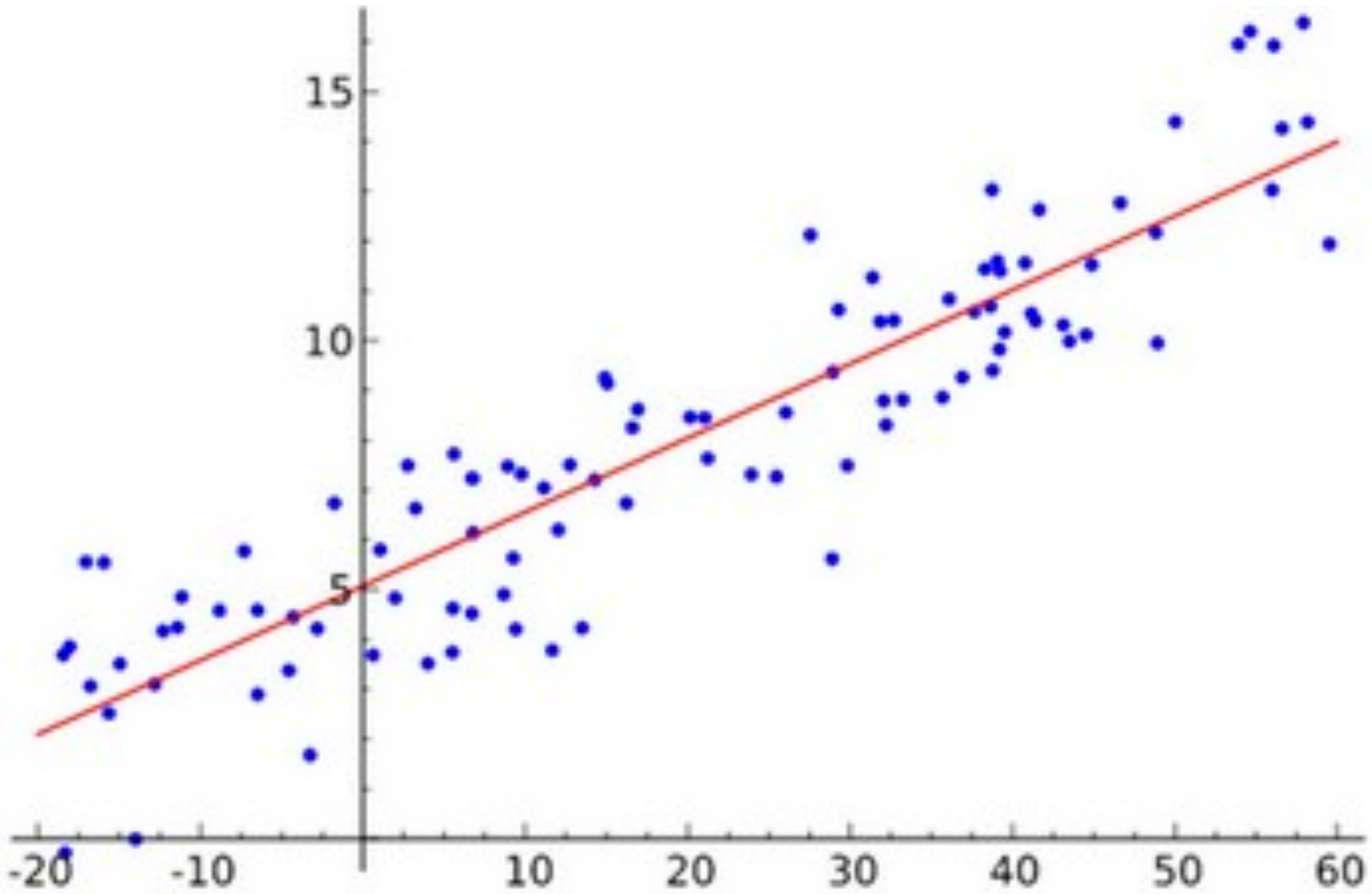
1. Classifier
2. Runtime Regressor
3. Speedup Regressor

1. Classifier
2. Runtime Regressor
3. Speedup Regressor

1. Classifier
2. Runtime Regressor
3. Speedup Regressor

Predict *speedup* of workgroup size *A* over *B* for program

Search for *highest speedup*



1. Classifier
2. Runtime Regressor
3. Speedup Regressor

1. Classifier
2. Runtime Regressor
3. Speedup Regressor

Questions:

1. What features do we need? ✓
2. What programs do we train on? ✓
3. How do we make predictions?

Questions:

1. What features do we need? ✓
2. What programs do we train on? ✓
3. How do we make predictions? ✓

Experiment

Implementation

Modified SkelCL stencil pattern

Python server process for autotuning

5 classifiers, random forest regressor

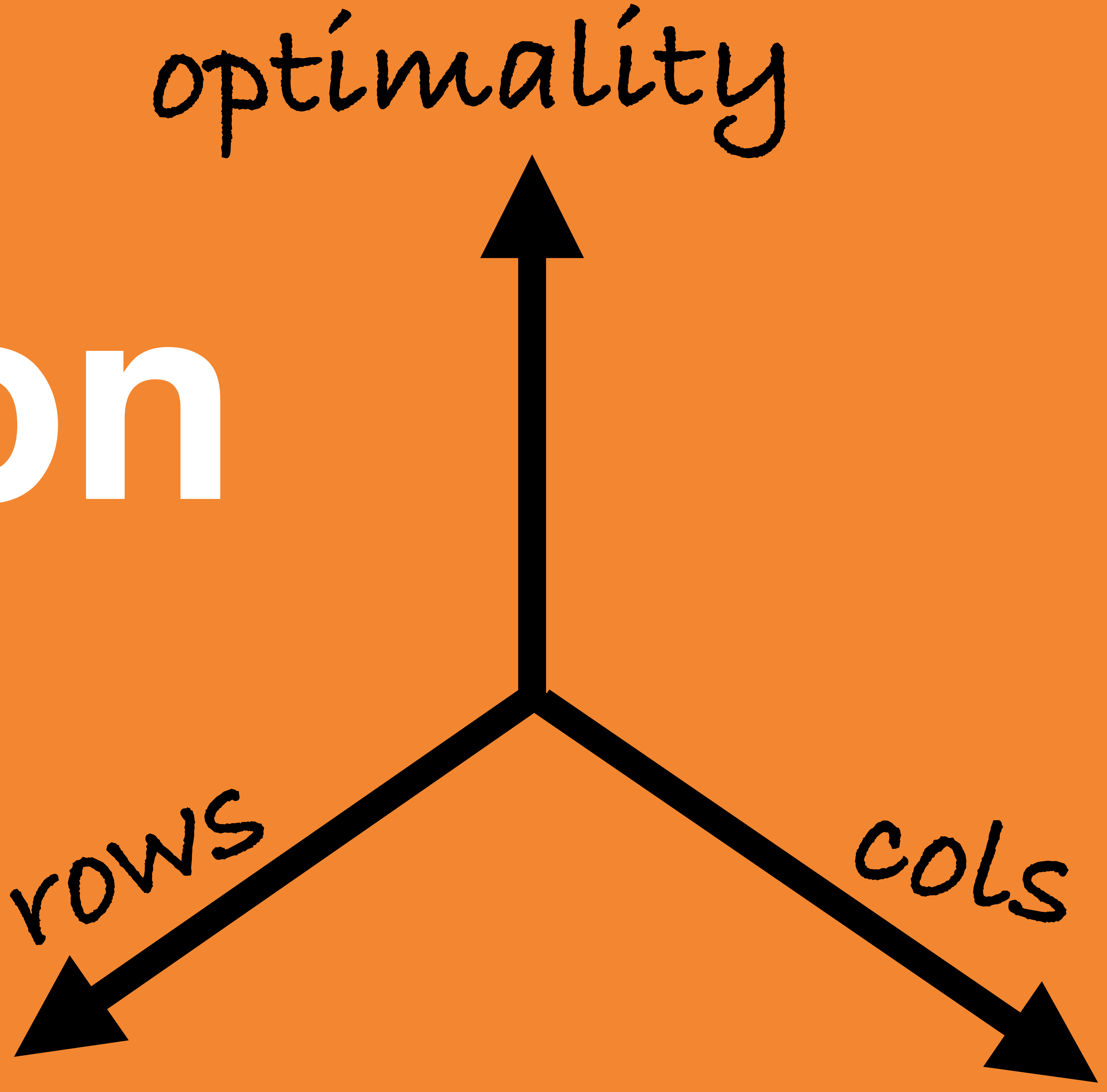
Experimental Setup

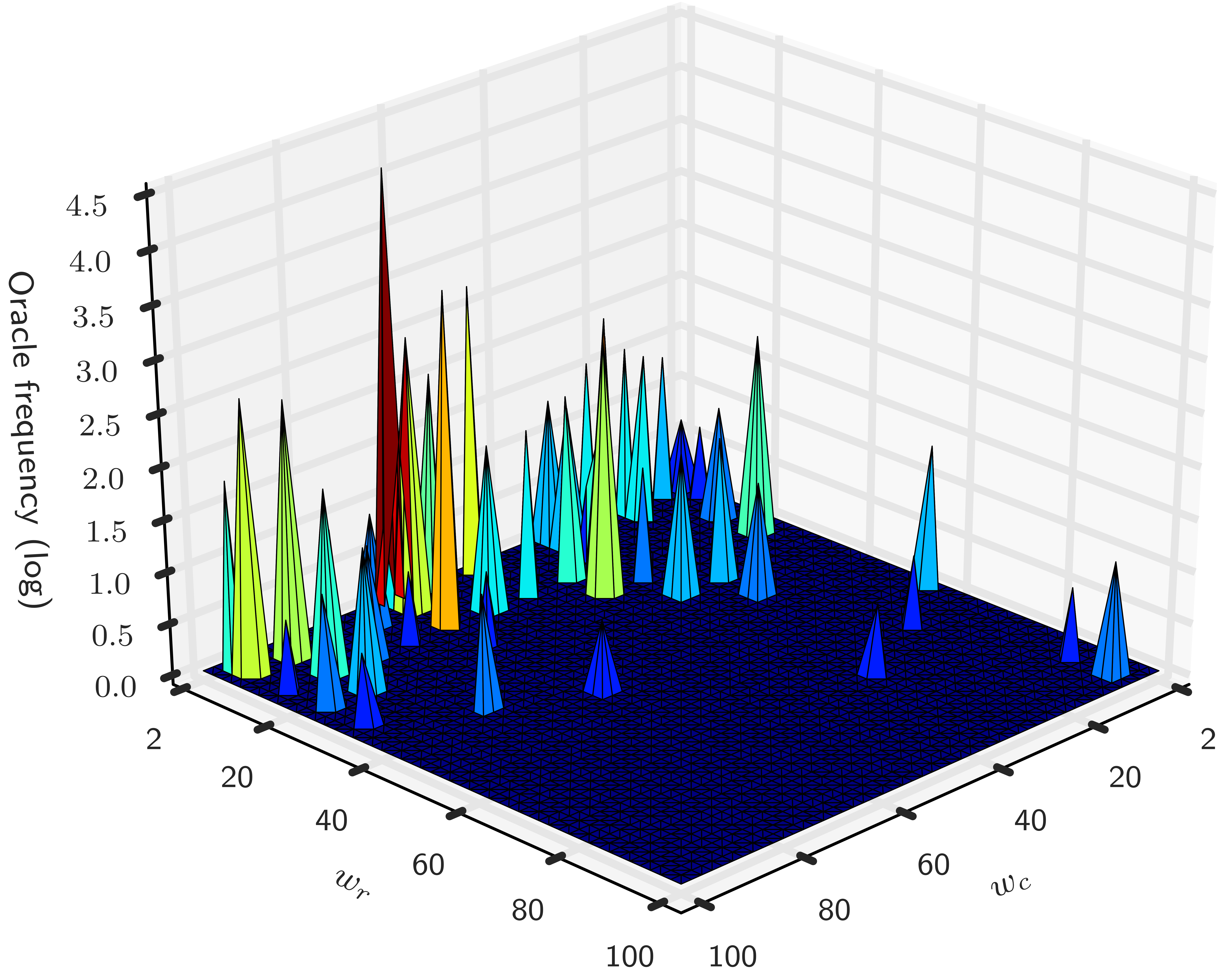
6 stencil benchmarks + synthetic.
7 different GPUs & CPUs.
4 dataset sizes.

Exhaustive search of workgroup size space for each

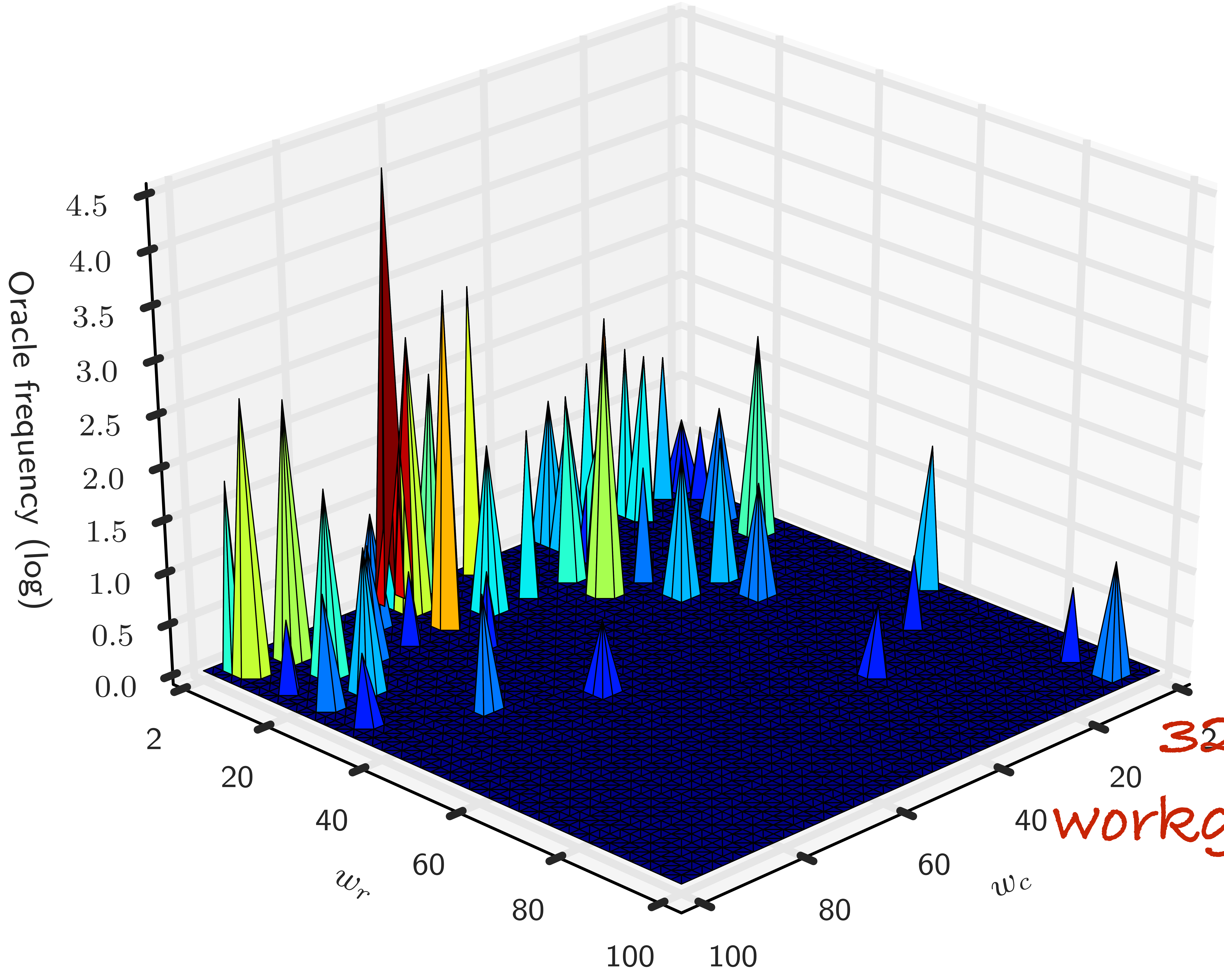
Results

Optimisation space

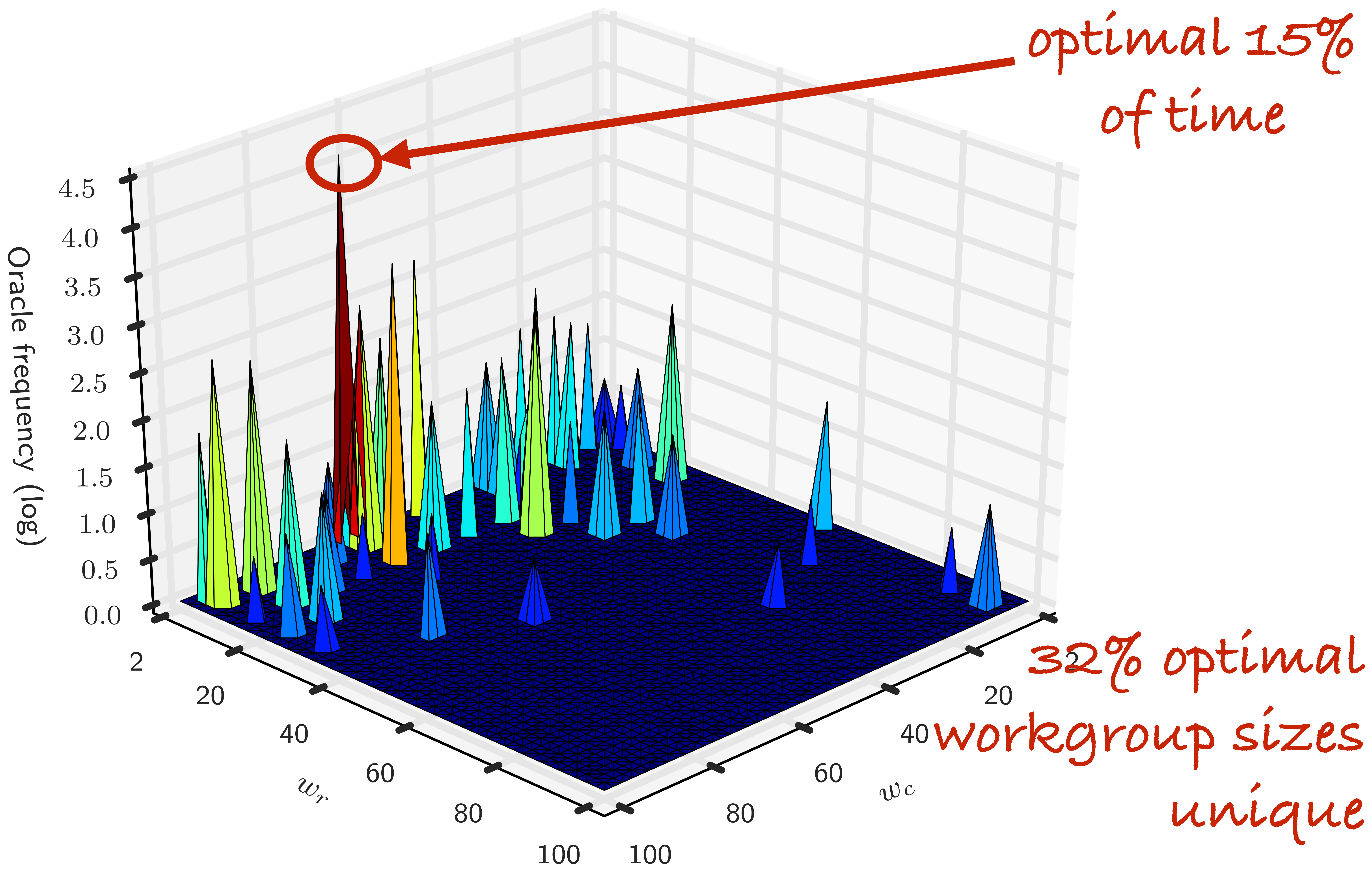




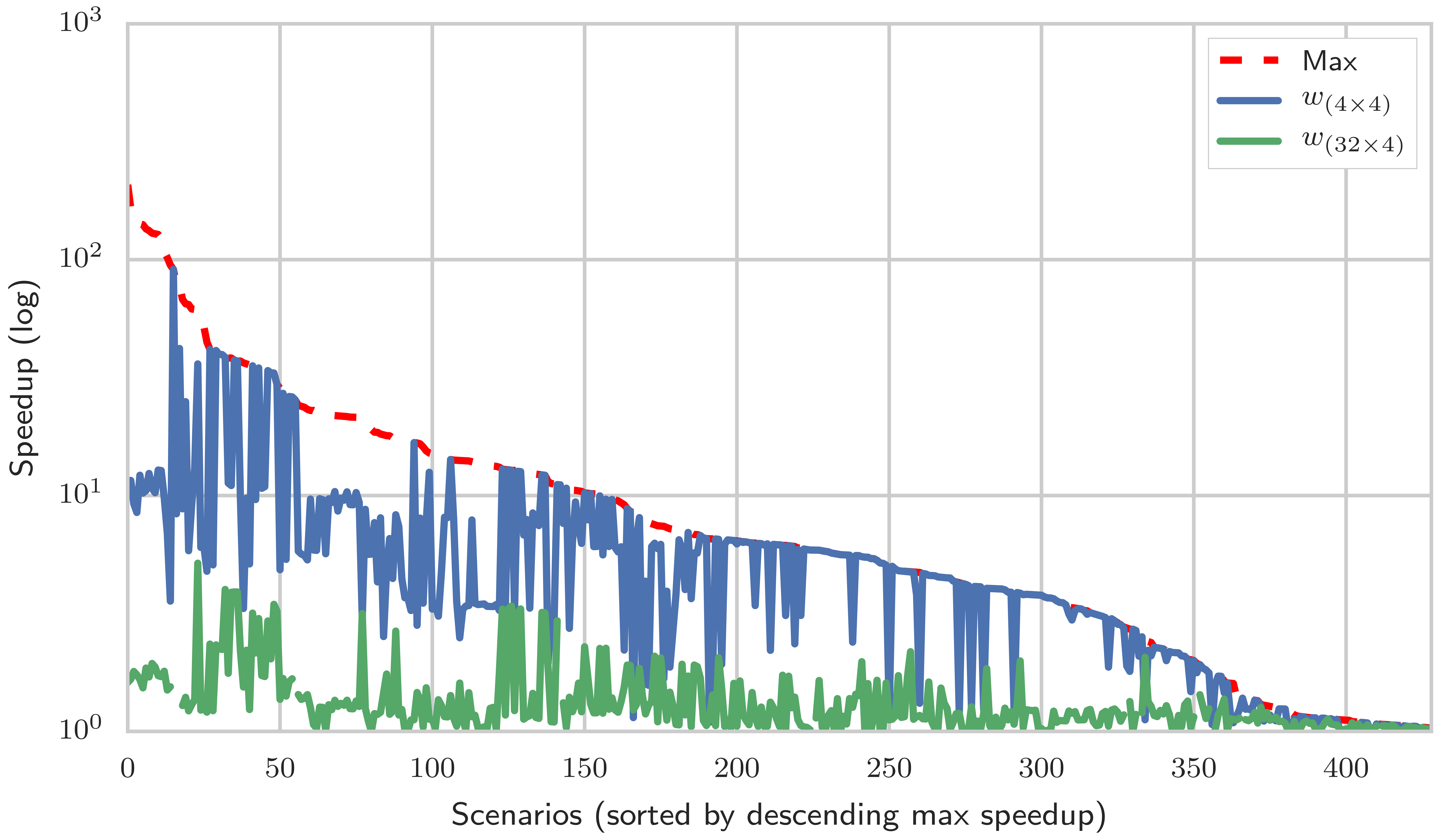
Oracle frequency (log)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
w_r
2
20
40
60
80
100
w_c
2
20
40
60
80
100



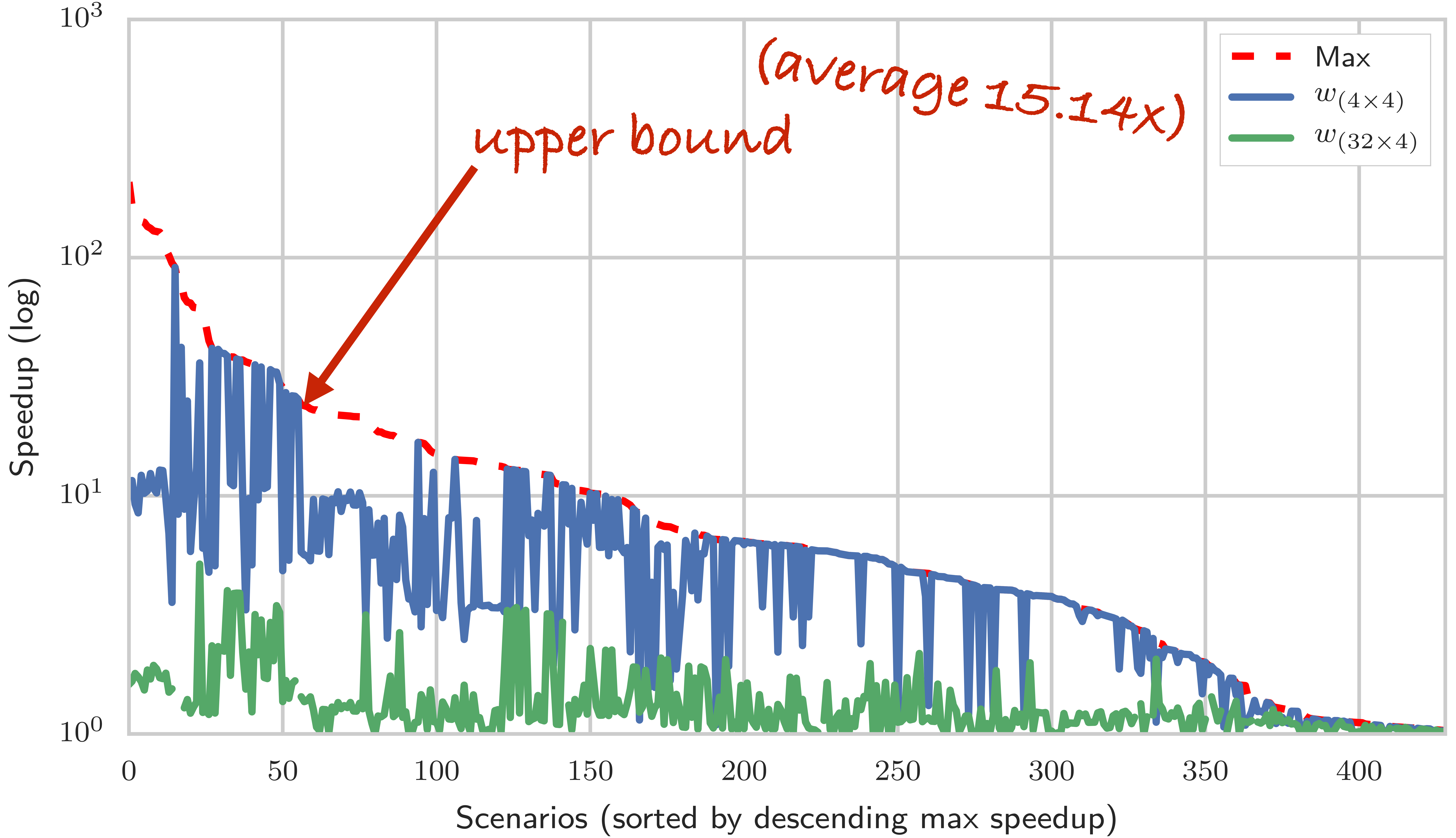
32% optimal workgroup sizes unique



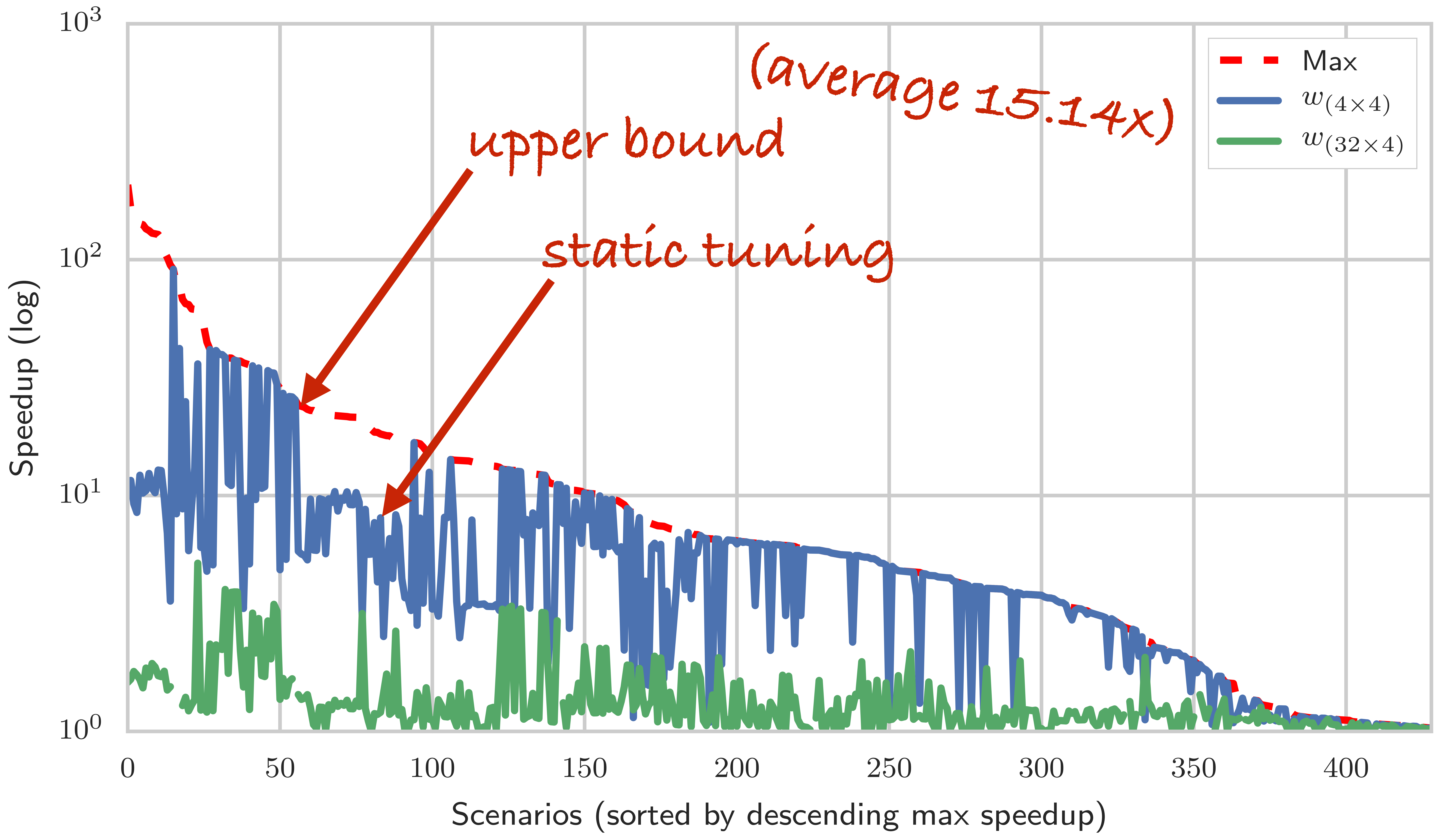
optimal 15% of time
32% optimal workgroup sizes unique
Oracle frequency (log)
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
w_r
2
20
40
60
80
100
w_c
2
20
40
60
80
100



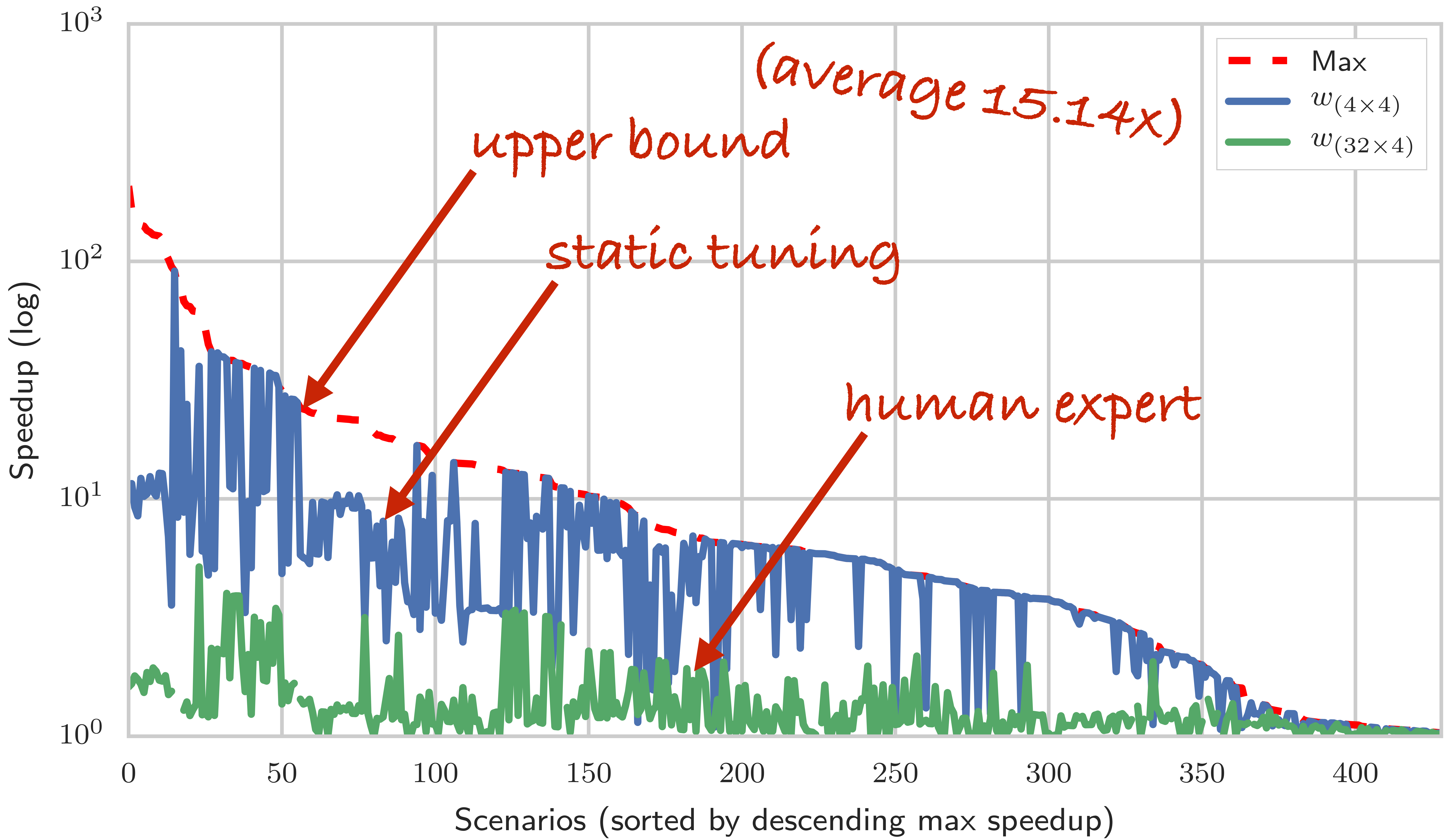
Speedup (log)
10^3
10^2
10^1
10^0
0
50
100
150
200
250
300
350
400
Scenarios (sorted by descending max speedup)
Max
$w_{(4\times4)}$
$w_{(32\times4)}$



(average 15.14x)
upper bound
Max
$w_{(4\times4)}$
$w_{(32\times4)}$
Speedup (log)
10^3
10^2
10^1
10^0
0
50
100
150
200
250
300
350
400
Scenarios (sorted by descending max speedup)



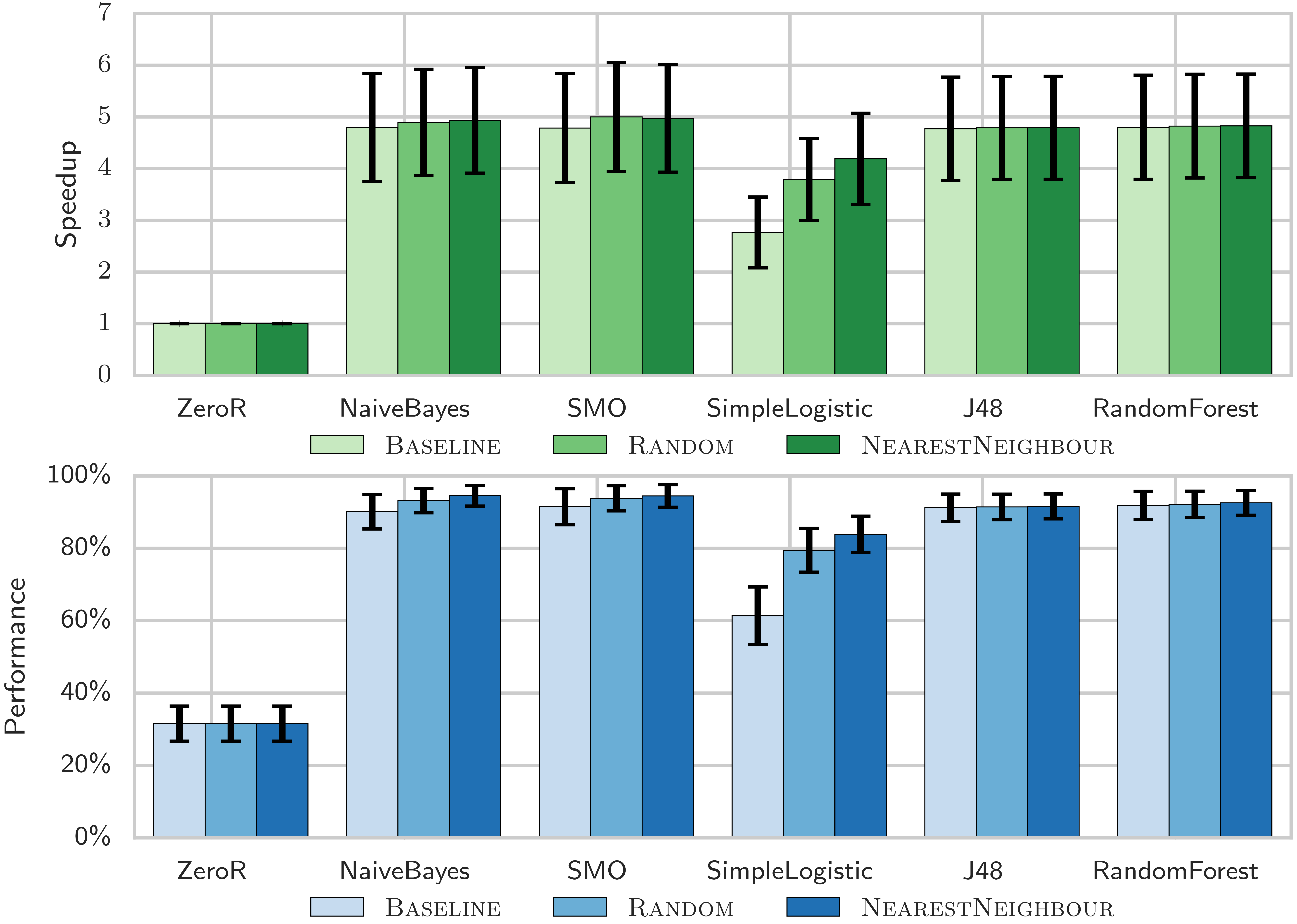
(average 15.14x)
upper bound
static tuning
Max
$w_{(4\times4)}$
$w_{(32\times4)}$
Speedup (log)
10^3
10^2
10^1
10^0
0
50
100
150
200
250
300
350
400
Scenarios (sorted by descending max speedup)



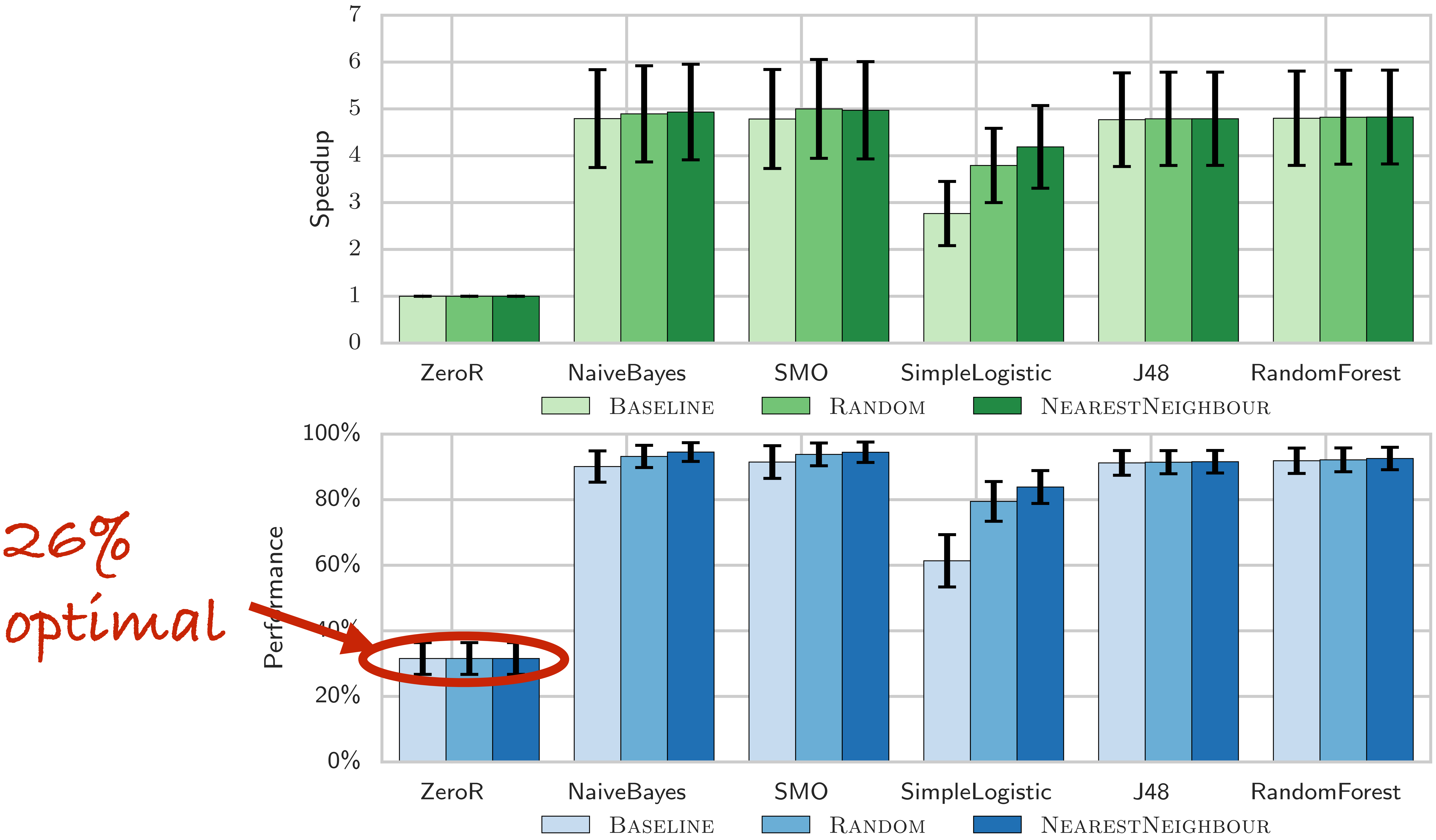
(average 15.14x)
upper bound
static tuning
human expert
Max
$w_{(4\times4)}$
$w_{(32\times4)}$
Speedup (log)
10^3
10^2
10^1
10^0
0
50
100
150
200
250
300
350
400
Scenarios (sorted by descending max speedup)

Autotuning

Classification



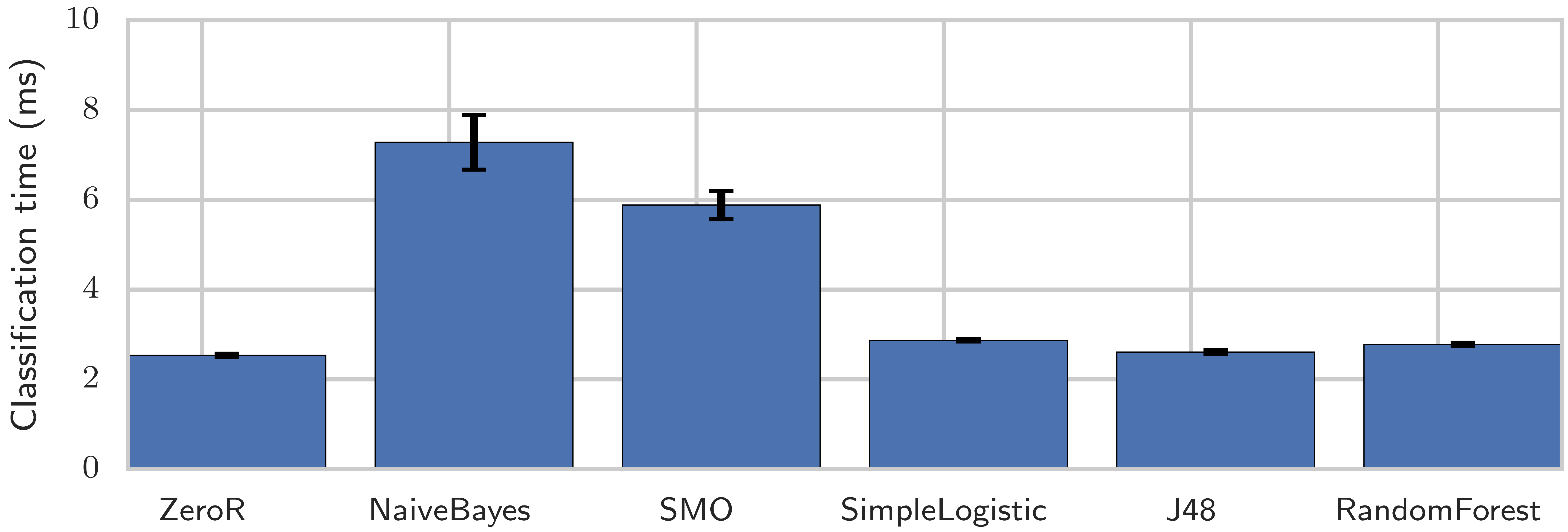
Speedup
7
6
5
4
3
2
1
0
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
Baseline
Random
NearestNeighbour
Performance
100%
80%
60%
40%
20%
0%
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
Baseline
Random
NearestNeighbour



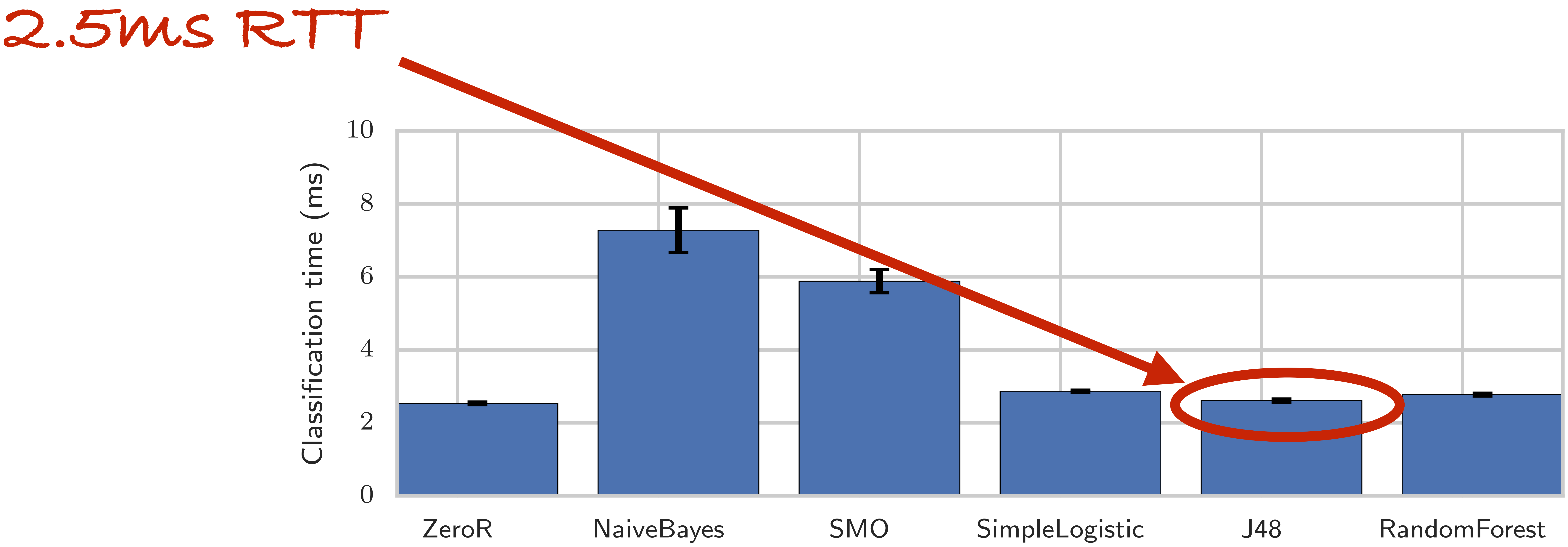
26% optimal
Speedup
7
6
5
4
3
2
1
0
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
Baseline
Random
NearestNeighbour
Performance
100%
80%
60%
40%
20%
0%
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
Baseline
Random
NearestNeighbour

Speedup
7
6
5
4
3
2
1
0
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
Baseline
Random
NearestNeighbour
Performance
100%
80%
60%
40%
20%
0%
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
Baseline
Random
NearestNeighbour
26% optimal
90% optimal

Nearest neighbour best
90% optimal
26% optimal
Speedup
7
6
5
4
3
2
1
0
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
BASELINE
RANDOM
NEARESTNEIGHBOUR
Performance
100%
80%
60%
40%
20%
0%
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest
BASELINE
RANDOM
NEARESTNEIGHBOUR



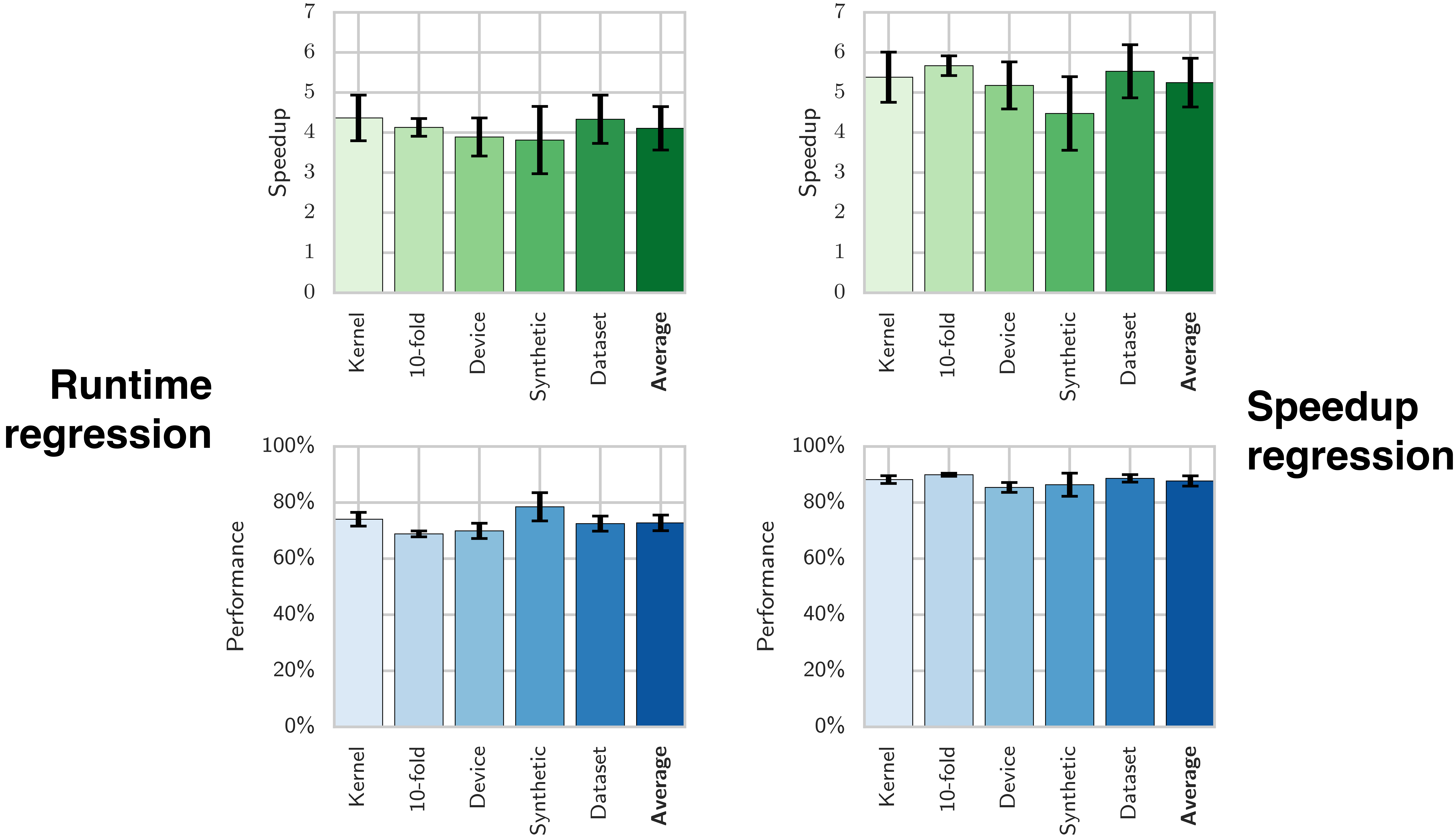
Classification time (ms)
0
2
4
6
8
10
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest



2.5ms RTT
Classification time (ms)
10
8
6
4
2
0
ZeroR
NaiveBayes
SMO
SimpleLogistic
J48
RandomForest

Autotuning

Regression

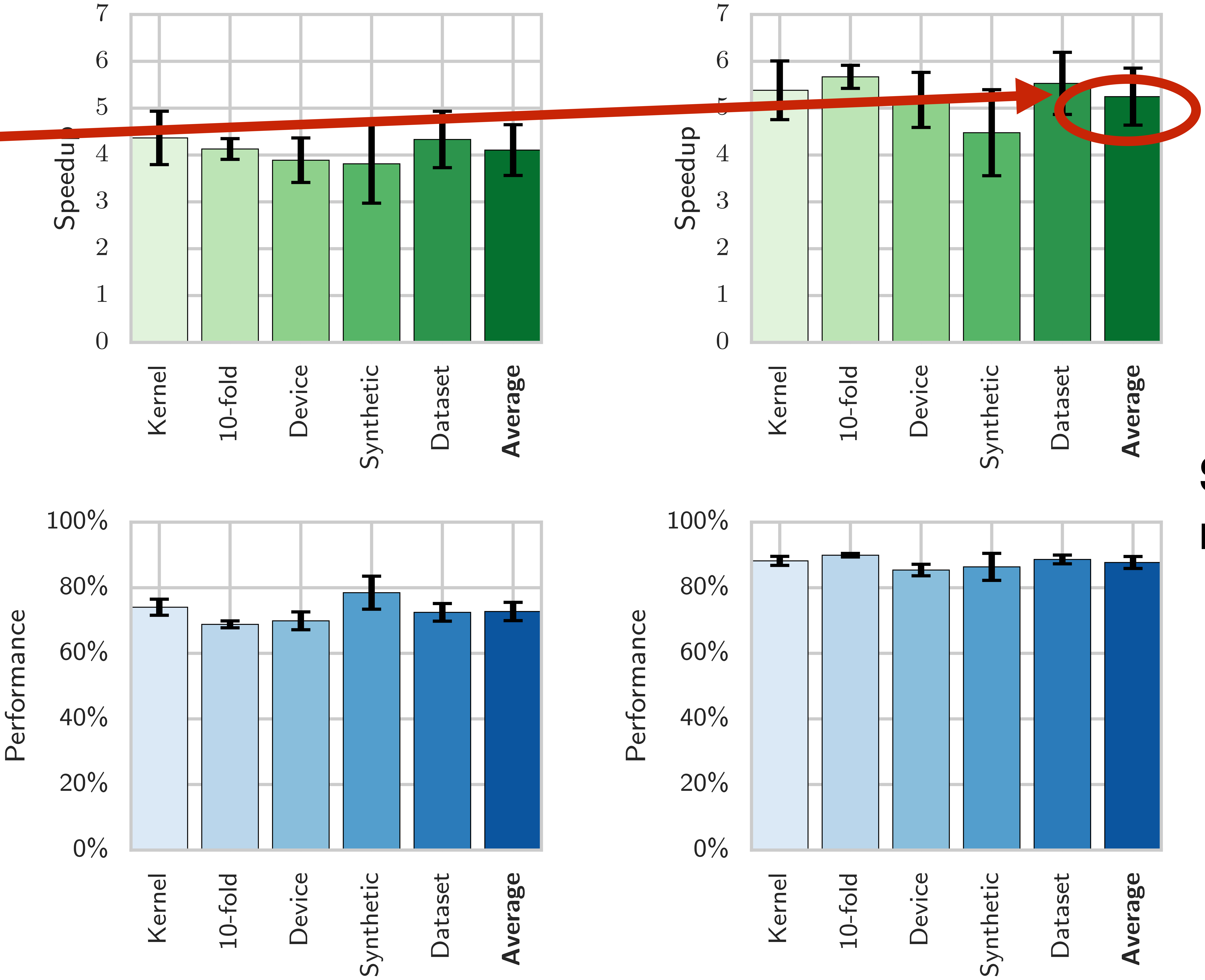


Runtime regression
Speedup regression
Speedup
Performance
Kernel
10-fold
Device
Synthetic
Dataset
Average

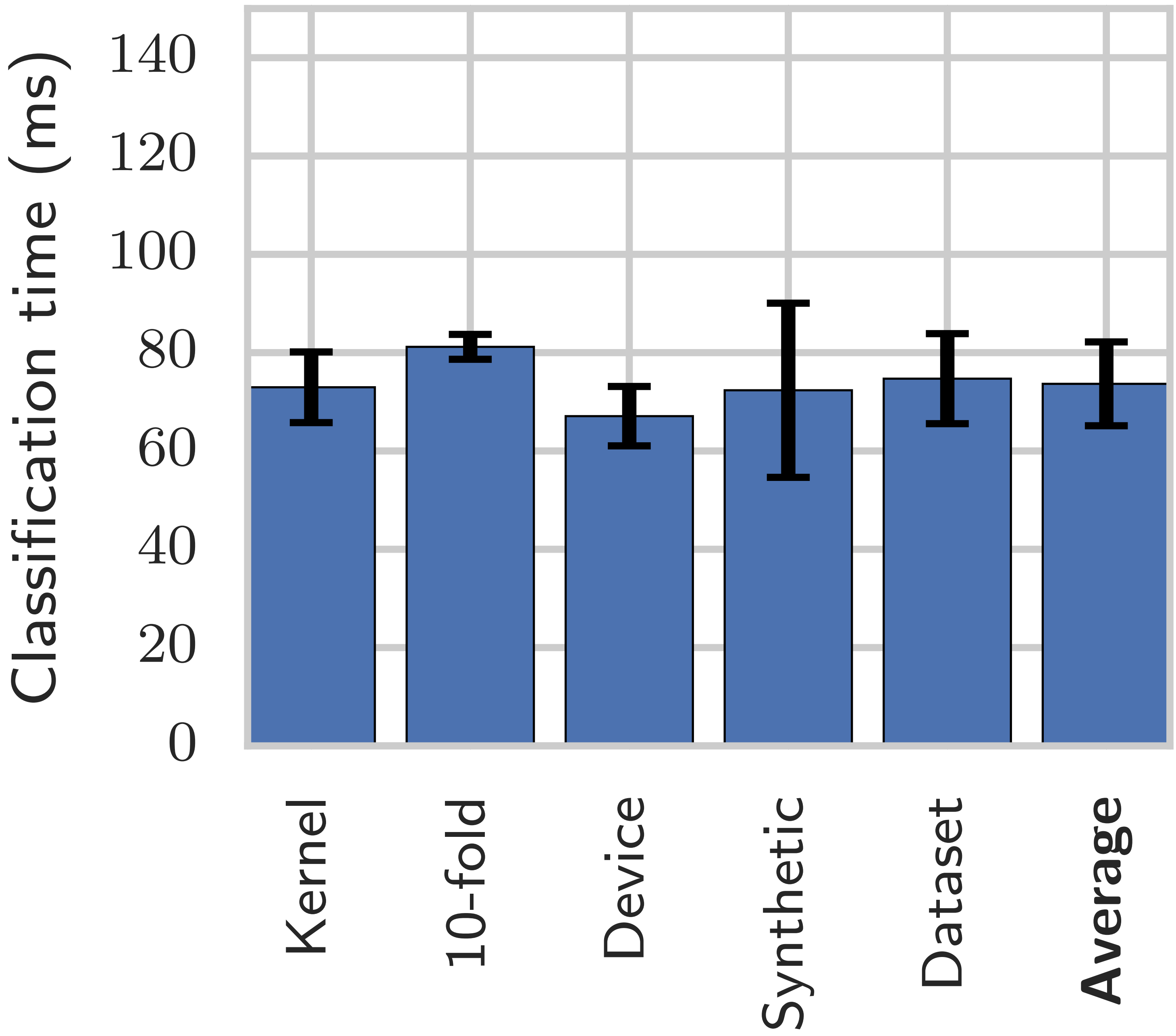
Highest speedup

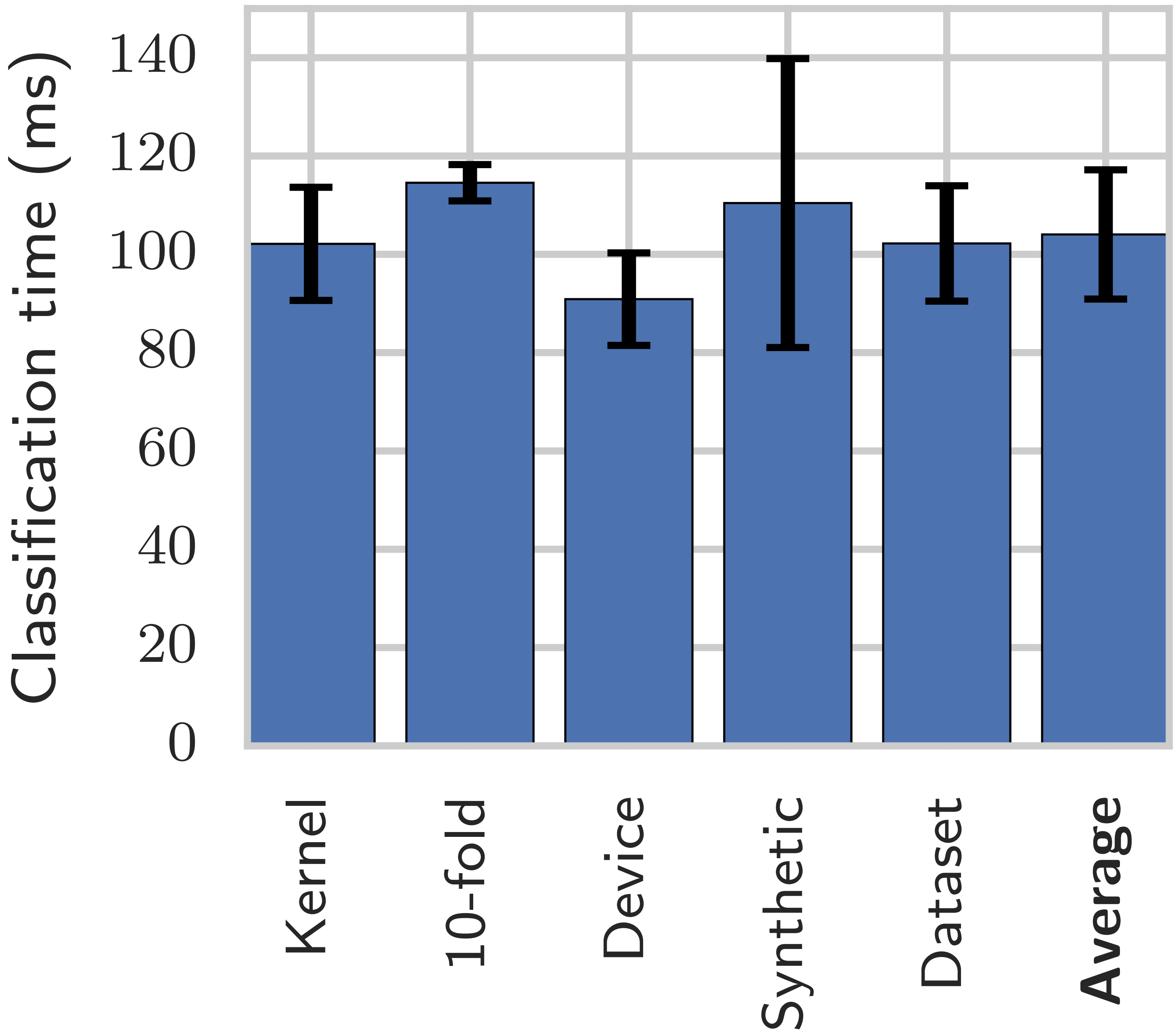
Runtime regression

Speedup regression



Runtime regression

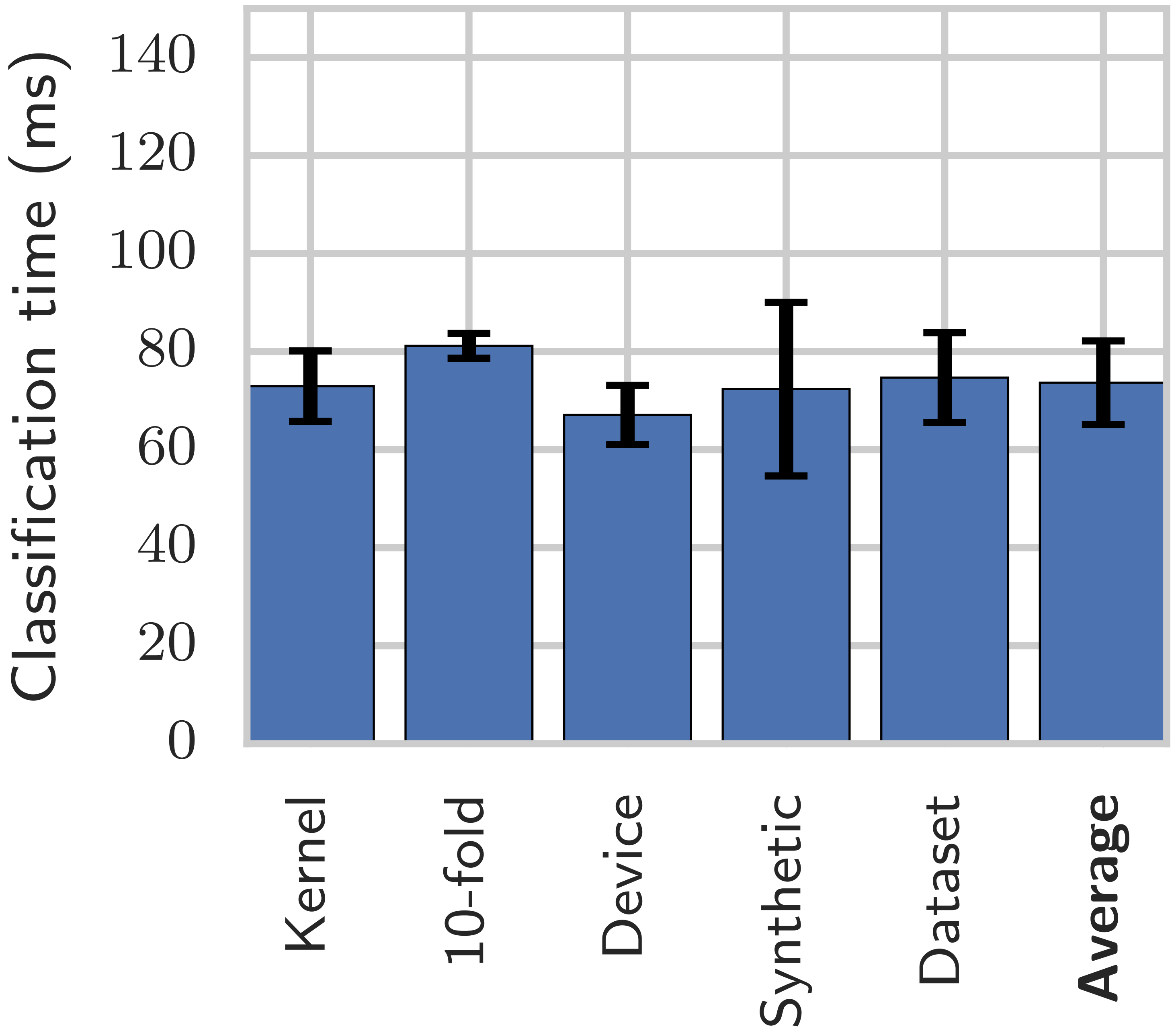


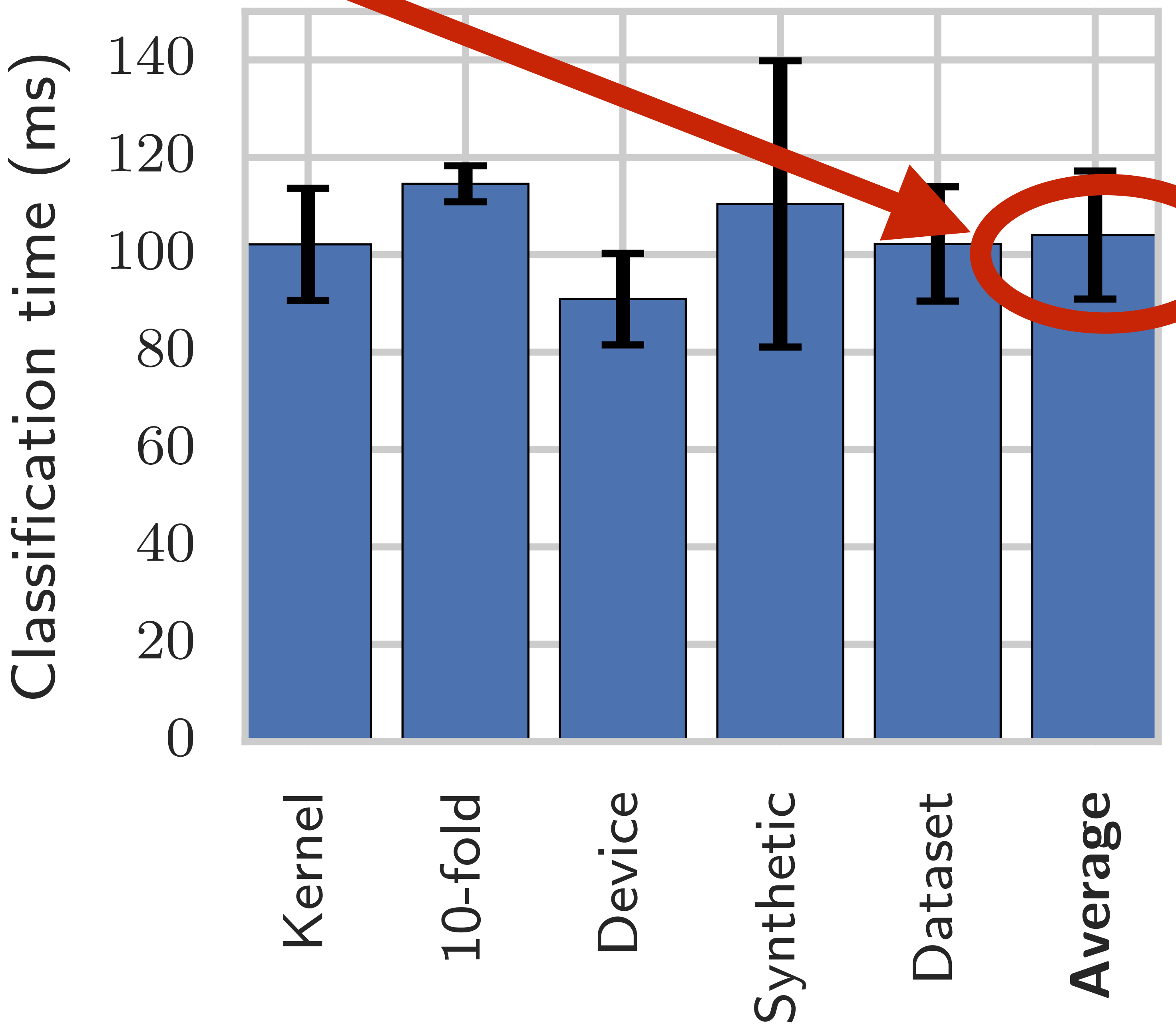


Speedup regression

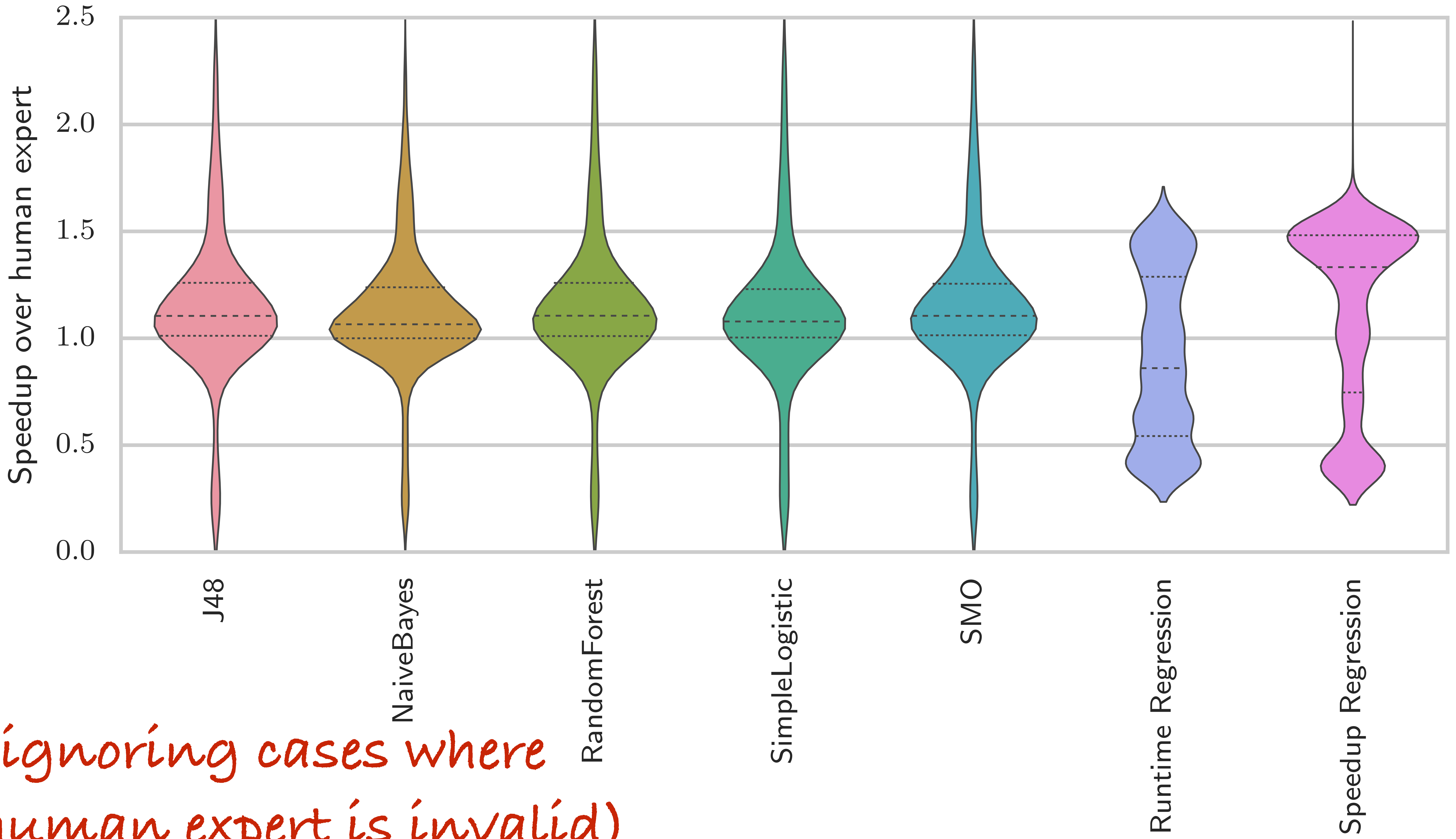
40x slower than J48

Runtime regression



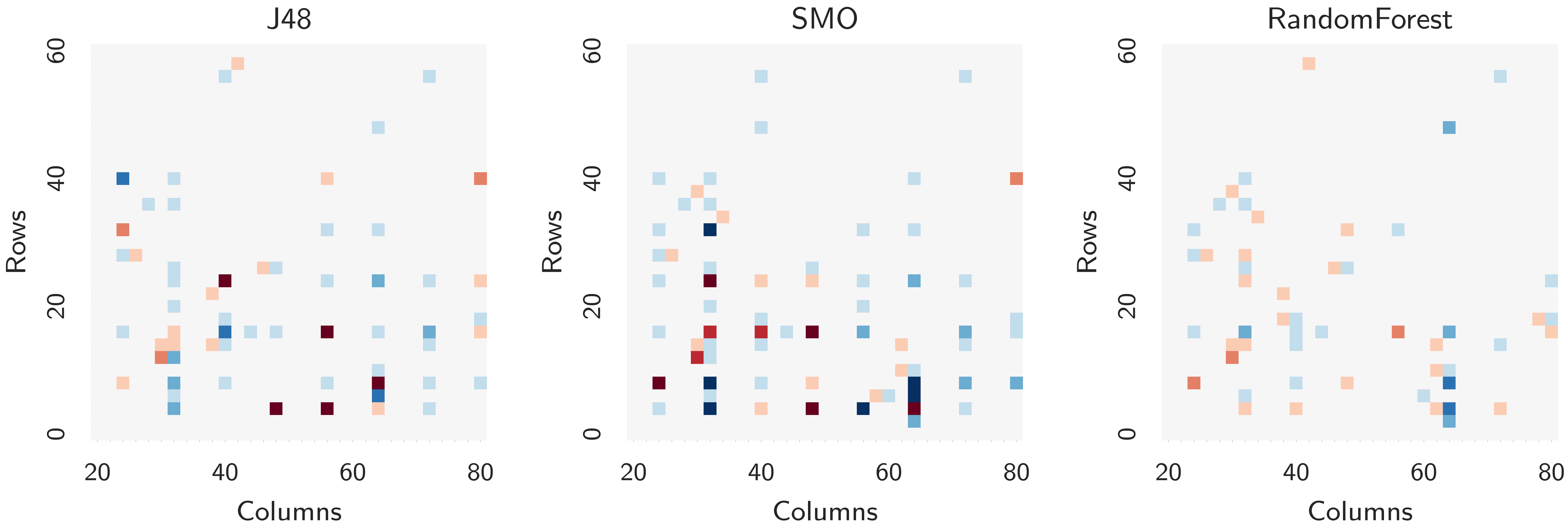


Speedup regression

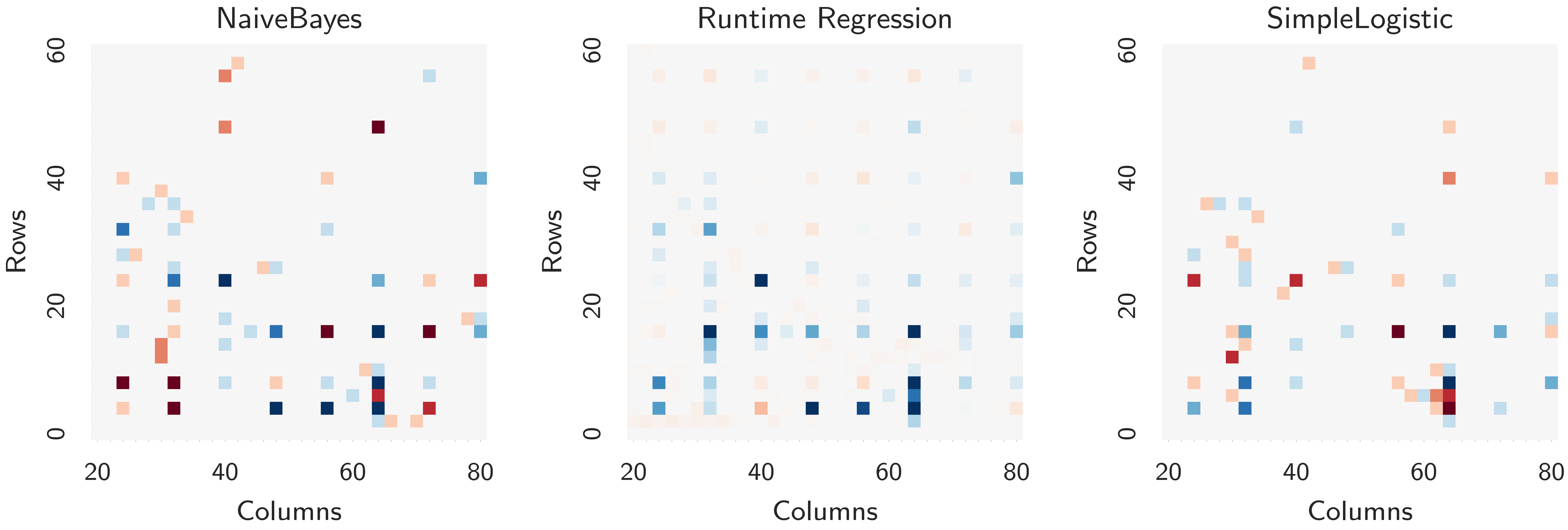


(ignoring cases where human expert is invalid)


Speedup over human expert
0.0
0.5
1.0
1.5
2.0
2.5
Appear similar
J48
NaiveBayes
RandomForest
SimpleLogistic
SMO
Runtime Regression
Speedup Regression
(ignoring cases where
human expert is invalid)



Very different prediction characteristics



Conclusions

Average 15x speedup best/worst workgroup size

Setting workgroup size depends on device, kernel, dataset

Static tuning achieves 26% of optimal performance

We present *three* methodologies for autotuning OpenCL workgroup size

Trade-offs between prediction cost and training cost

Achieving average 1.22x speedup over *human expert*, with increased reliability

Details in the paper!

Autotuning OpenCL Workgroup Size for Stencil Patterns

Chris Cummins Pavlos Petoumenos Michel Steuwer Hugh Leather
University of Edinburgh
c.cummins@ed.ac.uk, ppetoume@inf.ed.ac.uk, michel.steuwer@ed.ac.uk, hleather@inf.ed.ac.uk

Abstract

Selecting an appropriate workgroup size is critical for the performance of OpenCL kernels, and requires knowledge of the underlying hardware, the data being operated on, and the implementation of the kernel. This makes portable performance of OpenCL programs a challenging goal, since simple heuristics and statically chosen values fail to exploit the available performance. To address this, we propose the use of machine learning-enabled autotuning to automatically predict workgroup sizes for stencil patterns on CPUs and multi-GPUs.

We present three methodologies for predicting workgroup sizes. The first, using classifiers to select the optimal workgroup size. The second and third proposed methodologies employ the novel use of regressors for performing classification by predicting the runtime of kernels and the relative performance of different workgroup sizes, respectively. We evaluate the effectiveness of each technique in an empirical study of 429 combinations of architecture, kernel, and dataset, comparing an average of 629 different workgroup sizes for each. We find that autotuning provides a median 3.79× speedup over the best possible fixed workgroup size, achieving 94% of the maximum performance.

1. Introduction

Stencil codes have a variety of computationally demanding uses from fluid dynamics to quantum mechanics. Efficient, tuned stencil implementations are highly sought after, with early work in 2003 by Bolz et al. demonstrating the capability of GPUs for massively parallel stencil operations [1]. Since then, the introduction of the OpenCL standard has introduced greater programmability of heterogeneous devices by providing a vendor-independent layer of abstraction for data parallel programming of CPUs, GPUs, DSPs, and other devices [2]. However, achieving portable performance of OpenCL programs is a hard task — OpenCL kernels are sensitive to properties of the underlying hardware, to the implementation, and even to the dataset that is operated upon. This forces developers to laboriously hand tune performance on a case-by-case basis, since simple heuristics fail to exploit the available performance.

In this paper, we demonstrate how machine learning-enabled autotuning can address this issue for one such optimisation parameter of OpenCL programs — that of workgroup size. The 2D optimisation space of OpenCL kernel workgroup sizes is complex and non-linear, making it resistant to analytical modelling. Successfully applying machine learning to such a space requires plentiful training data, the careful selection of features, and an appropriate classification approach. The approaches presented in this paper use features extracted from the architecture and kernel, and training data collected from synthetic benchmarks to predict workgroup sizes for unseen programs.

2. The SkelCL Stencil Pattern

Introduced in [3], SkelCL is an Algorithmic Skeleton library which provides OpenCL implementations of data parallel patterns for heterogeneous parallelism using CPUs and multi-GPUs. Figure 1 shows the components of the SkelCL stencil pattern, which applies a user-provided *customising function* to each element of a 2D matrix. The value of each element is updated based on its current value and the value of one or more neighbouring elements, called the *border region*. The border region describes a rectangular region about each cell, and is defined in terms of the number of cells in the border region to the north, east, south, and west of each cell. Where elements of a border region fall outside of the matrix bounds, values are substituted from either a predefined padding value, or the value of the nearest cell within the matrix, determined by the user.

When a SkelCL stencil pattern is executed, each of the matrix elements are mapped to OpenCL work-items; and this collection of work-items is divided into *workgroups* for execution on the target hardware. A work-item reads the value of its corresponding matrix element and the surrounding elements defined by the border region. Since the border regions of neighbouring elements overlap, each element in the matrix is read multiple times. Because of this, a *tile* of elements of the size of the workgroup and the perimeter border region is allocated as a contiguous block in local memory. This greatly reduces the latency of repeated memory accesses

1

2016/1/6

Autotuning OpenCL Workgroup Size for Stencil Patterns

http://chriscummins.cc