SURF SARA
HPC center of the Netherlands

blippAR
Augmented Reality Technology

Better Than All the Rest:
Finding Maximum-Performance GPU Kernels Using Auto-Tuning

GPU Technology Conference
April 7, 2016

Cedric Nugteren

How to find the best flight ticket?

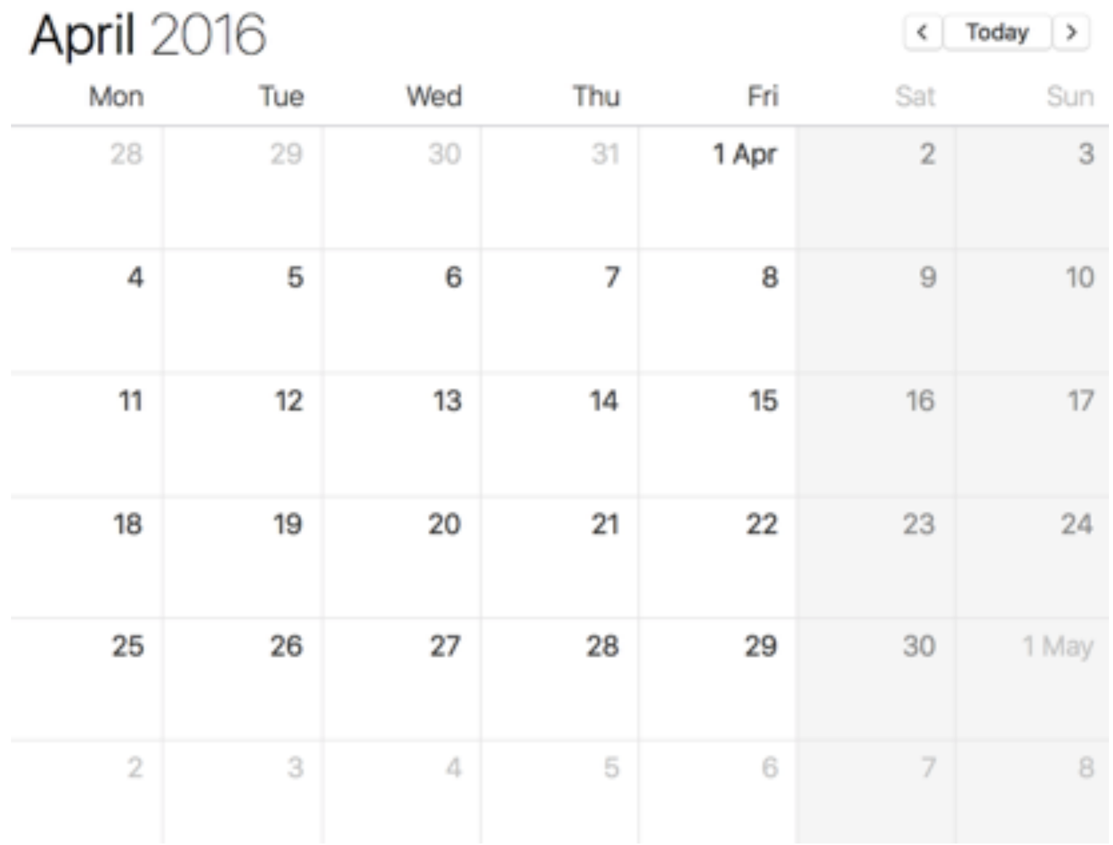
SOUTHWEST AIRLINES

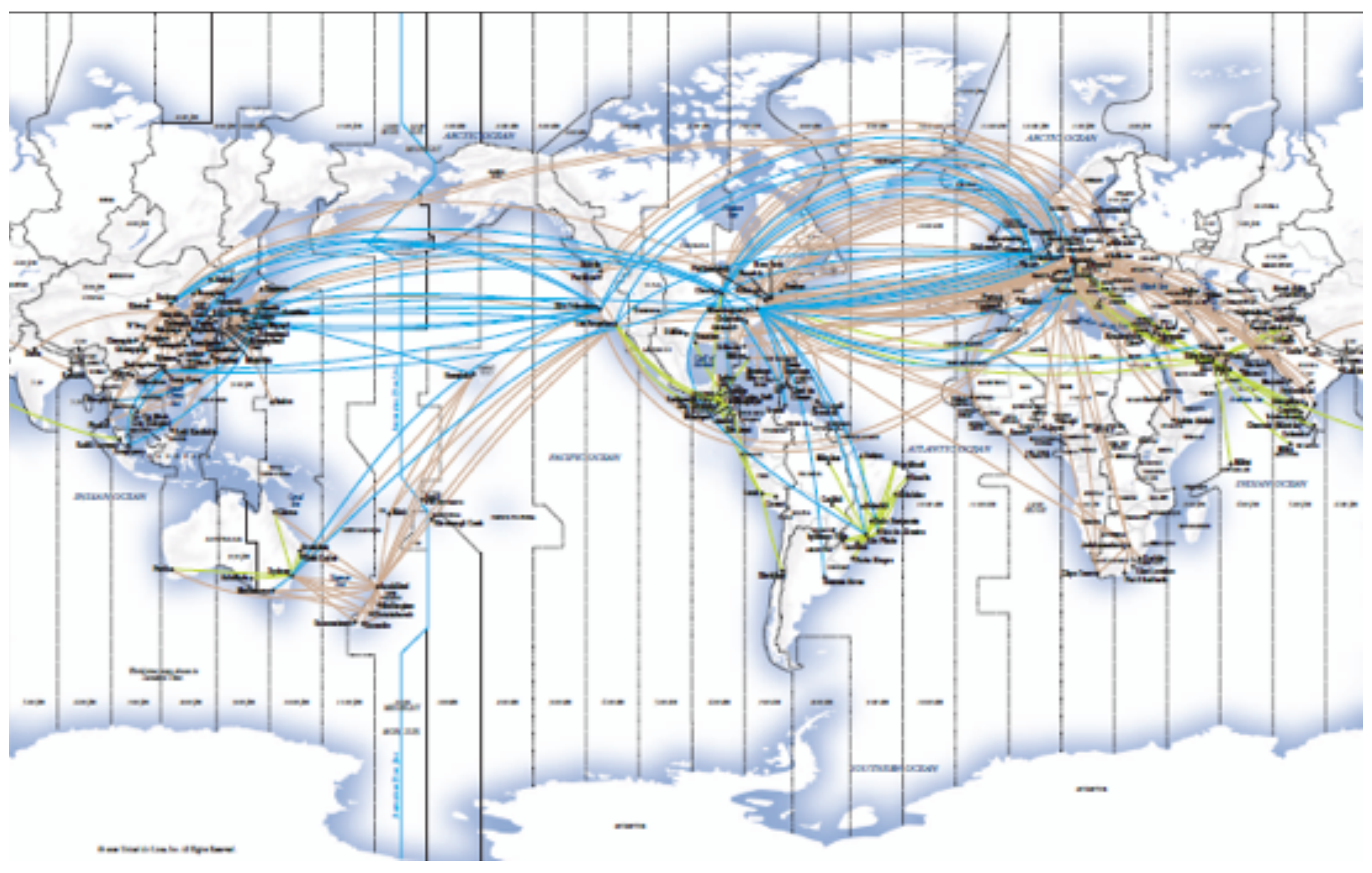
UNITED

Continental Airlines

Which airline?

Fly a day earlier or later?





Which route?

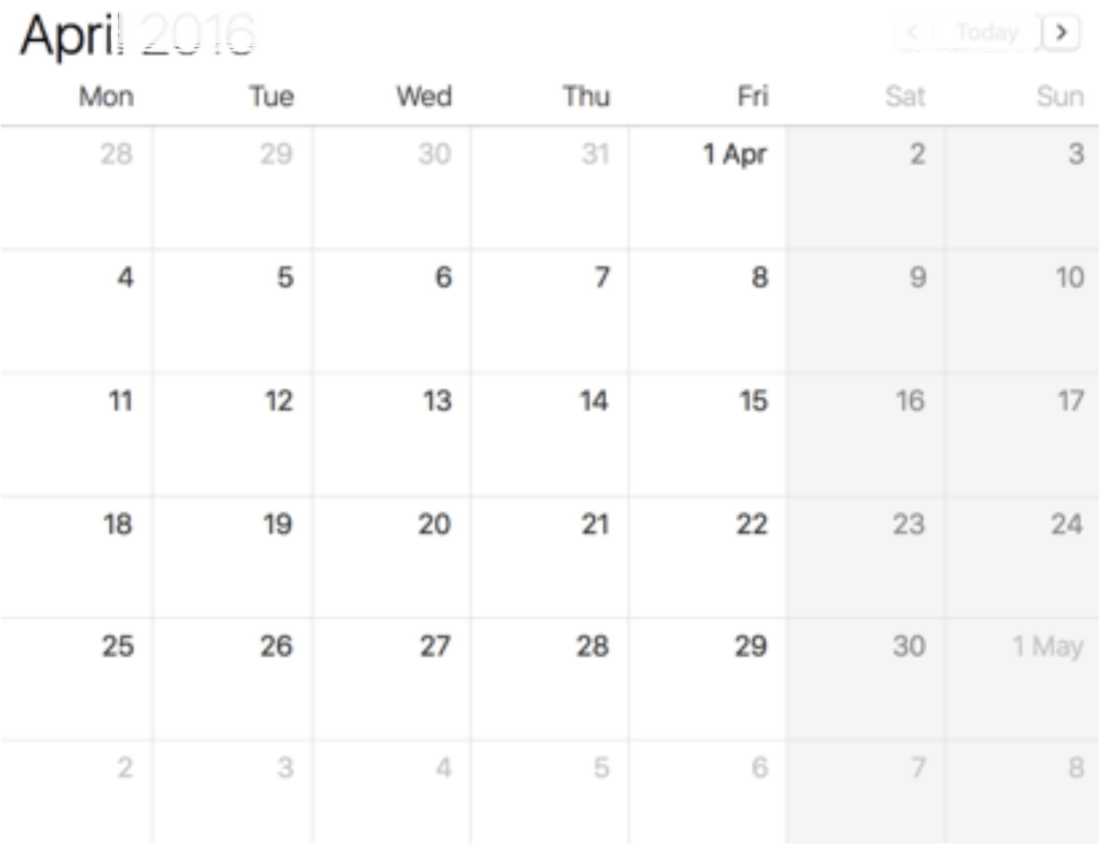
Solution:

- Evaluate all combinations manually
- … or use a flight comparison website

How to find the best flight ticket?

Thread block sizes?

Cache in shared memory?



More work per thread or less?

Solution:

- Evaluate all combinations manually
- … or use **the CLTune auto-tuner**

Example: convolution

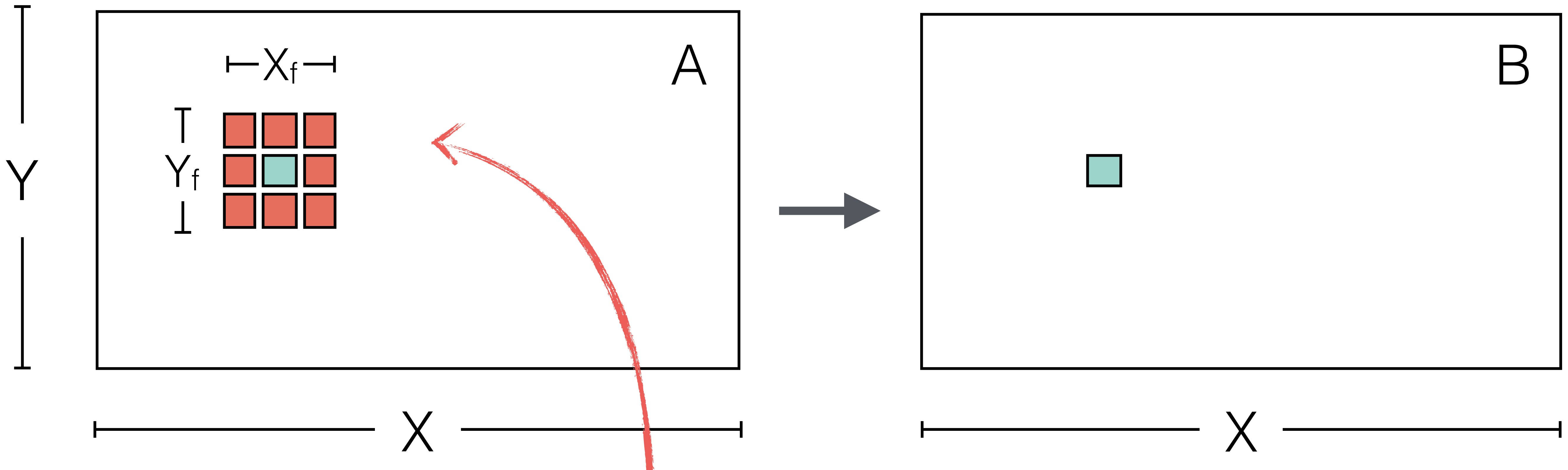
Example: blur filter

input

output

input

output



filter size: X_f by Y_f

filter coefficients

$$B_{x,y} = w \cdot \sum_{i=-1}^{i\leq 1} \sum_{j=-1}^{j\leq 1} F_{i,j} A_{x+i,y+j}$$

example: 3 by 3 filter

Targets:

- GPUs (CUDA & OpenCL)
- Multi-core CPUs
- Other OpenCL-capable devices

OpenCL 2D convolution

each thread: one output pixel

Thread coarsening (2D)?

$$B_{x,y} = w \cdot \sum_{i=-1}^{i\leq 1} \sum_{j=-1}^{j\leq 1} F_{i,j} A_{x+i,y+j}$$

double for-loop

Unroll loops?

```

#define HFS (3)          // Half filter size
#define FS (HFS+HFS+1)  // Filter size

__kernel void conv_reference(const int size_x, const int size_y,
                             const __global float* src,
                             __constant float* coeff,
                             __global float* dest) {

  const int tid_x = get_global_id(0);
  const int tid_y = get_global_id(1);

  float acc = 0.0f;

  // Loops over the neighbourhood
  for (int fx=-HFS; fx<=HFS; ++fx) {
    for (int fy=-HFS; fy<=HFS; ++fy) {
      const int index_x = tid_x + HFS + fx;
      const int index_y = tid_y + HFS + fy;

      // Performs the accumulation
      float coefficient = coeff[(fy+HFS)*FS + (fx+HFS)];
      acc += coefficient * src[index_y*size_x + index_x];
    }
  }

  // Stores the result
  dest[tid_y*size_x + tid_x] = acc;
}
```

work-group / thread-block size?

Vector data-types?

Cache in local / shared memory?

Search-space explosion

Thread coarsening (2D)?

Work-group / thread-block size?

Cache in local / shared memory?

16 x 2 x 16 x 4 x 5 = 10240

Unroll loops?

Vector data-types?

filter illegal configurations

3424 configurations

Large search-space:

- Not feasible to explore manually
- Perhaps not even feasible automatically?

Why do we need an auto-tuner?

Large search-space:

- Not feasible to explore manually
- Perhaps not even feasible automatically?

Cache in local / shared memory?

Work-group / thread-block size?

Thread coarsening (2D)?

Unroll loops?

Vector data-types?

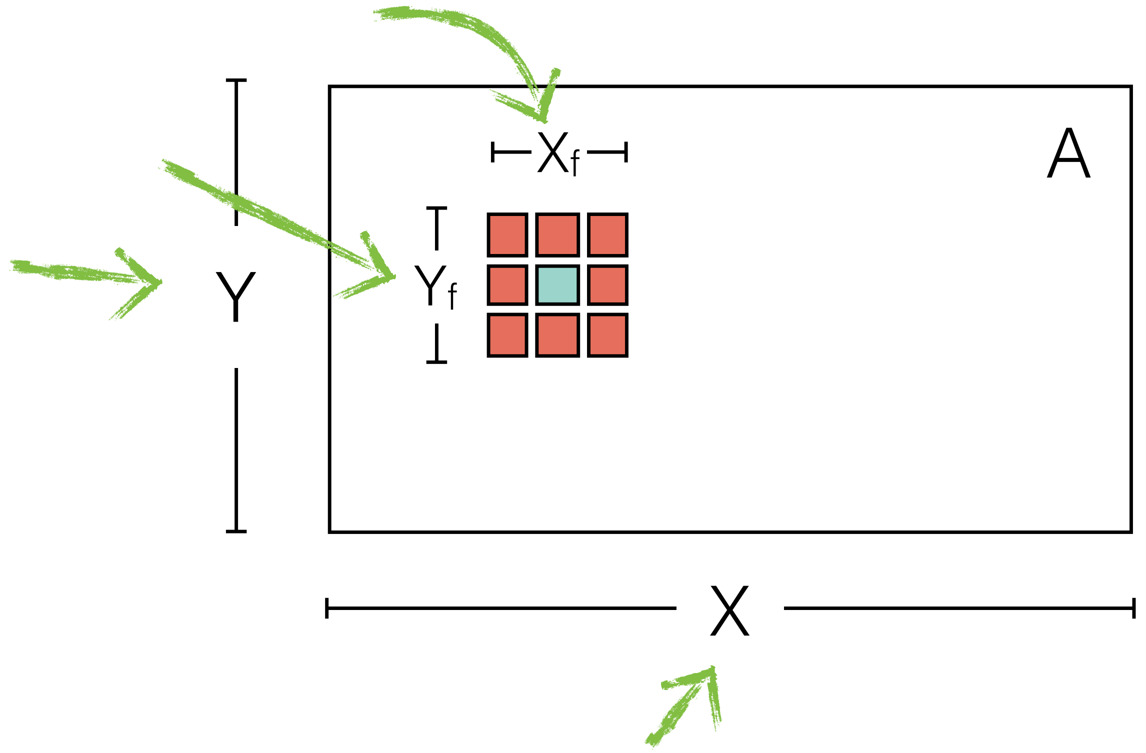
Wide variety of devices:

- Different optimal kernels
- Even from the same vendor

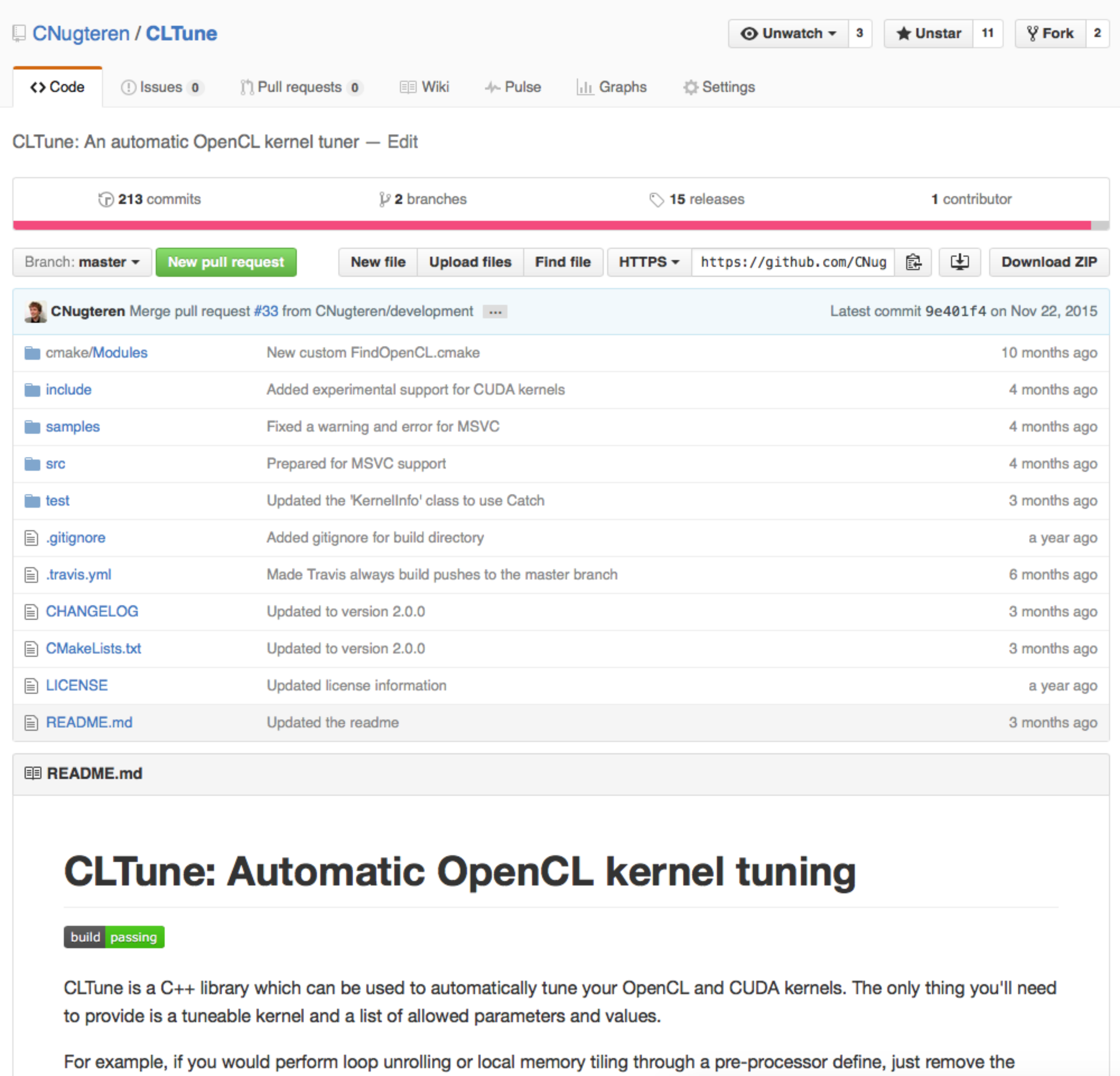
vendor and device name	archi-tecture	peak GFLOPS	peak GB/s	GFLOPS per GB/s
NVIDIA Tesla K40m	Kepler	4291	288	14.9
NVIDIA GeForce GTX480	Fermi	1345	177	7.6
AMD Radeon HD7970	Tahiti	4368	288	15.1
Intel Iris 5100	Iris	832	26	32.5

User-parameter dependent:

- Examples: matrix sizes, image size, filter sizes, etc.



Why ... uner?



Large search-s

- Not feasible t
- Perhaps not e

Wide variety of

- Different opti
- Even from the

User-paramete

- Examples: m
  image size, fi

?

Thread

rsening (2D)?

-types?

OPS
GB/s
.9
.6
.1
.5

CLTune in action

Example: matrix-vector multiplication (y = Ax) with tiling

Example input data

Kernel to tune

Tile size (TS): 4 possible values

Verifies output based on optional reference kernel

```
#include "cltune.h"

void main() {
  constexpr auto kSizeM = 2048;
  constexpr auto kSizeN = 4096;

  std::vector<float> mat_a(kSizeN*kSizeM);
  std::vector<float> vec_x(kSizeN);
  std::vector<float> vec_y(kSizeM);

  cltune::Tuner tuner(0, 1); // platform 0, device 1

  // Adds a kernel which supports tiling by a factor TS
  auto id = tuner.AddKernel({"../tiled.opencl"}, "matvec_tiled", {kSizeM}, {1});
  tuner.AddParameter(id, "TS", {32, 64, 128, 256});
  tuner.MulLocalSize(id, {"TS"});

  // Sets the function's arguments
  tuner.AddArgumentScalar(static_cast<int>(kSizeM));
  tuner.AddArgumentScalar(static_cast<int>(kSizeN));
  tuner.AddArgumentInput(mat_a);
  tuner.AddArgumentInput(vec_x);
  tuner.AddArgumentOutput(vec_y);

  // Runs the tuner
  tuner.Tune();
  tuner.PrintToScreen();
}
```

CLTune in action

```
#include "cltune.h"

void main() {
  constexpr auto kSizeM = 2048;
  constexpr auto kSizeN = 4096;

  std::vector<float> mat_a(kSizeN*kSizeM);
  std::vector<float> vec_x(kSizeN);
  std::vector<float> vec_y(kSizeM);

  cltune::Tuner tuner(0, 1); // platform 0, device 1

  // Adds a kernel which supports tiling by a factor TS
  auto id = tuner.AddKernel({"../tiled.opencl"}, "matvec_tiled", {kSizeM}, {1});
  tuner.AddParameter(id, "TS", {32, 64, 128, 256});
  tuner.MulLocalSize(id, {"TS"});

  // Sets the function's arguments
  tuner.AddArgumentScalar(static_cast<int>(kSizeM));
  tuner.AddArgumentScalar(static_cast<int>(kSizeN));
  tuner.AddArgumentInput(mat_a);
  tuner.AddArgumentInput(vec_x);
  tuner.AddArgumentOutput(vec_y);

  // Runs the tuner
  tuner.Tune();
  tuner.PrintToScreen();
}
```

Handles all the host - device data transfers!

```
__kernel void matvec_tiled(const int M, const int N,
                           const __global float* mat_a,
                           const __global float* vec_x,
                           __global float* vec_y) {
  const int i = get_global_id(0); // From 0 to M - 1
  const int tile_id = get_local_id(0); // From 0 to TS - 1

  // Initializes the local/shared memory
  __local float tile_x[TS];

  // Loops over the tiles
  float result = 0.0f;
  for (int t=0; t<N/TS; ++t) {

    // Loads a tile of the vector x into the local memory
    tile_x[tile_id] = vec_x[t*TS + tile_id];
    barrier(CLK_LOCAL_MEM_FENCE);

    // Computes the partial result
    for (int j=0; j<TS; ++j) {
      result += mat_a[(t*TS + j)*M + i] * tile_x[j];
    }
    barrier(CLK_LOCAL_MEM_FENCE);
  }

  // Stores the result
  vec_y[i] = result;
}
```

Tuneable tile-size TS

CLTune in action

```
#include "cltune.h"

void main() {
  constexpr auto kSizeM = 2048;
  constexpr auto kSizeN = 4096;

  std::vector<float> mat_a(kSize
  std::vector<float> vec_x(kSize
  std::vector<float> vec_y(kSize

  cltune::Tuner tuner(0, 1); //

  // Adds a kernel which support
  auto id = tuner.AddKernel({".
  tuner.AddParameter(id, "TS",
  tuner.MulLocalSize(id, {"TS"}

  // Sets the function's argume
  tuner.AddArgumentScalar(stati
  tuner.AddArgumentScalar(stati
  tuner.AddArgumentInput(mat_a)
  tuner.AddArgumentInput(vec_x)
  tuner.AddArgumentOutput(vec_y

  // Runs the tuner
  tuner.Tune();
  tuner.PrintToScreen();
}
```

```
[==========] Initializing on platform 0 device 2
[==========] Device name: 'AMD Radeon R9 M370X Compute Engine' (OpenCL 1.2 )

[----------] Testing reference matvec_reference
[ RUN      ] Running matvec_reference
[       OK ] Completed matvec_reference (11 ms) - 1 out of 1

[----------] Testing kernel matvec_tiled
[ RUN      ] Running matvec_tiled
[       OK ] Completed matvec_tiled (10 ms) - 1 out of 4
[ RUN      ] Running matvec_tiled
[       OK ] Completed matvec_tiled (9 ms) - 2 out of 4
[ RUN      ] Running matvec_tiled
[       OK ] Completed matvec_tiled (8 ms) - 3 out of 4
[ RUN      ] Running matvec_tiled
[       OK ] Completed matvec_tiled (8 ms) - 4 out of 4

[----------] Printing results to stdout
[ RESULT   ] matvec_tiled;     10 ms;    TS 32;
[ RESULT   ] matvec_tiled;      9 ms;    TS 64;
[ RESULT   ] matvec_tiled;      8 ms;   TS 128;
[ RESULT   ] matvec_tiled;      8 ms;   TS 256;

[----------] Printing best result to stdout
[     BEST ] matvec_tiled;      8 ms;   TS 128;

[----------] Printing results to file in JSON format

[==========] End of the tuning process
```

```
t M, const int N,
global float* mat_a,
global float* vec_x,
 float* vec_y) {
/ From 0 to M - 1
0); // From 0 to TS - 1

emory

 into the local memory
tile_id];

+ i] * tile_x[j];
```

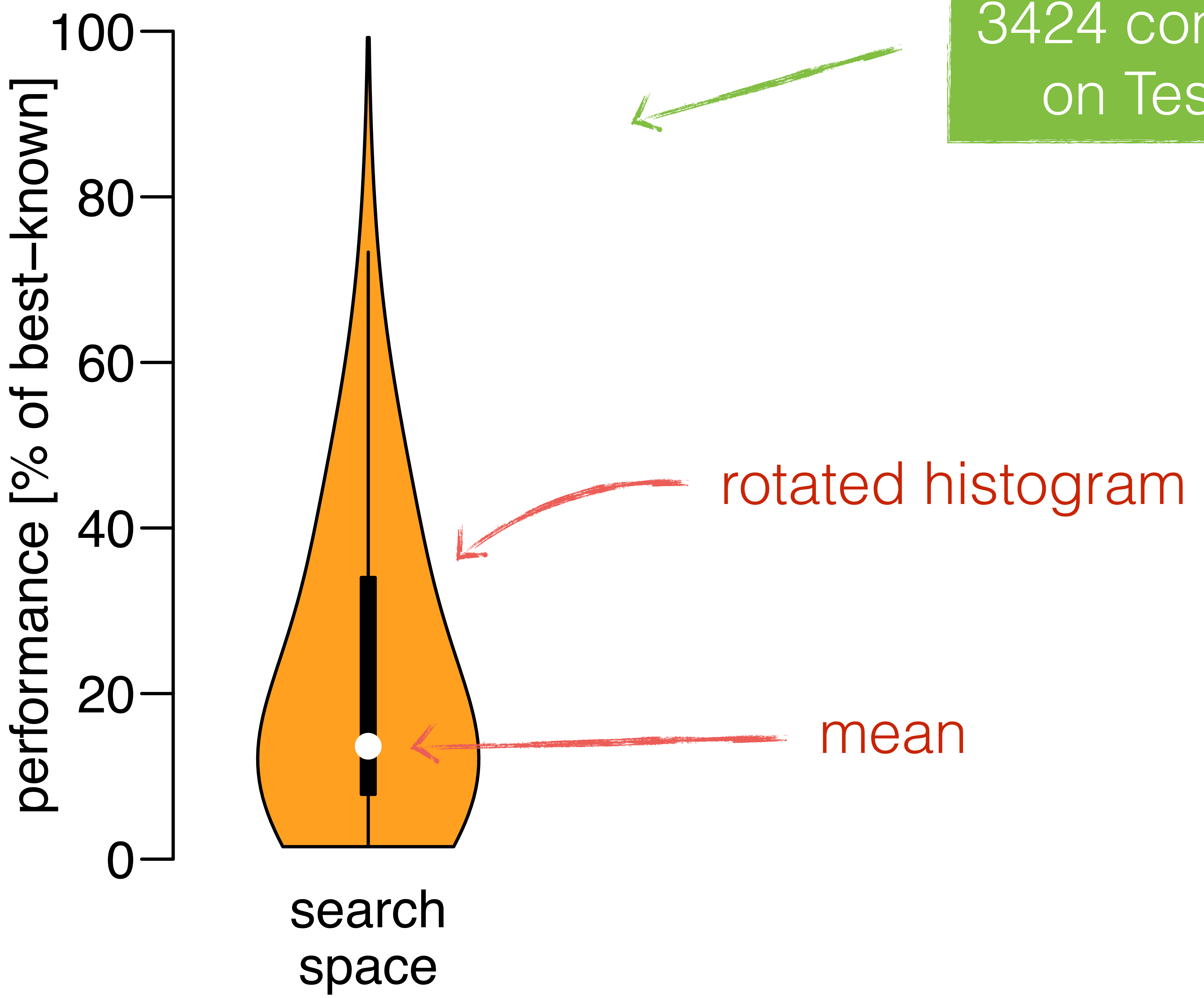
Tuneable tile-size TS

Search strategies

Option 0: Full search

☺ Finds optimal solution

✗ Explores all options



3424 convolution kernels on Tesla K40m GPU

Search strategies

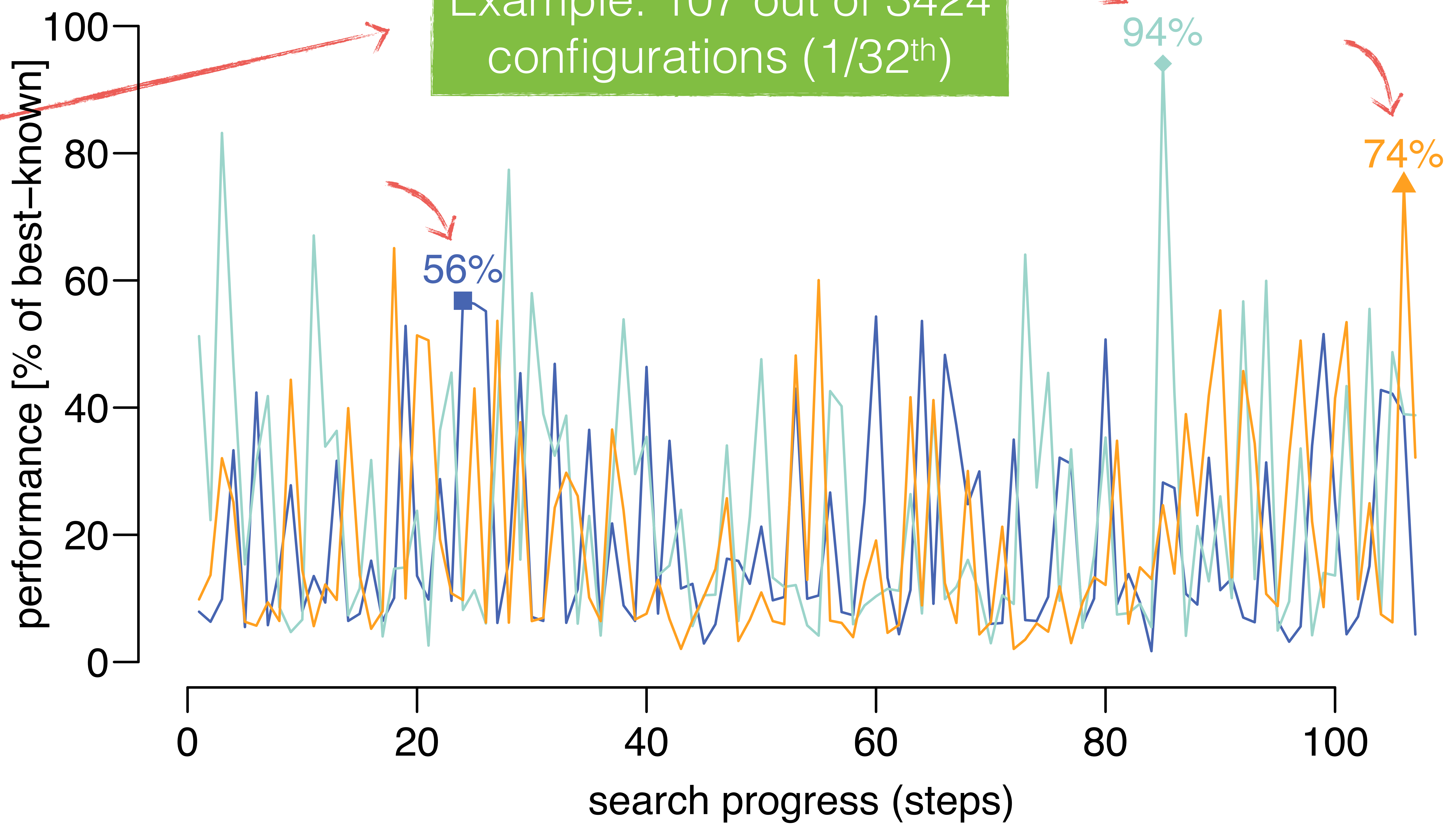
Option 0: Full search

Option 1: Random search

☺ Explores arbitrary fraction

✗ Performance varies

Example: 107 out of 3424 configurations (1/32th)



Colours: 3 example runs

Search strategies

Option 0: Full search

Option 1: Random search

Option 2: Simulated annealing

☺ Explores arbitrary fraction

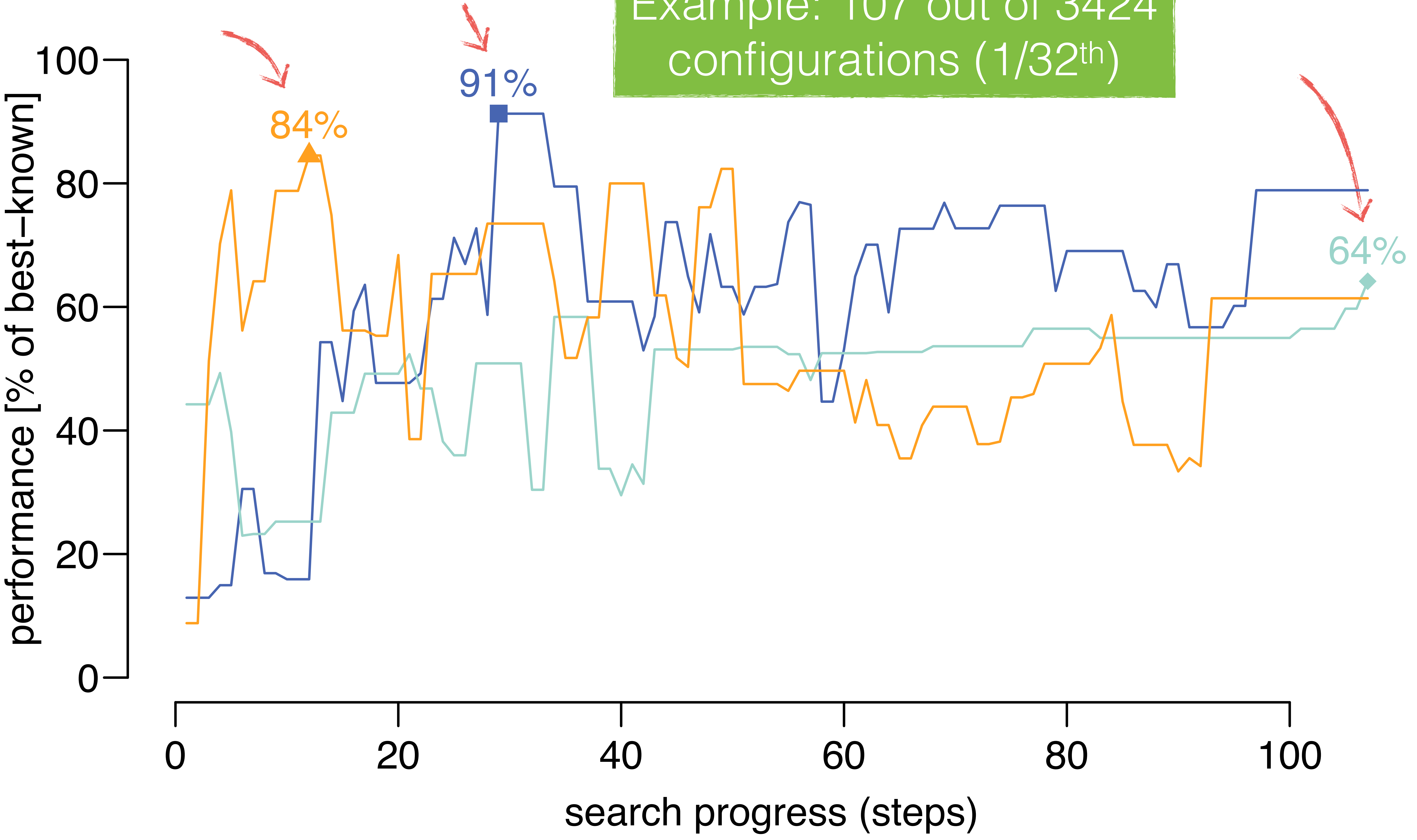
✗ Performance varies

✗ Meta-parameter

✗ Local optima

Colours: 3 example runs

Example: 107 out of 3424 configurations (1/32th)



Search strategies

Option 0: Full search

Option 1: Random search

Option 2: Simulated annealing

Option 3: Particle swarm optim.

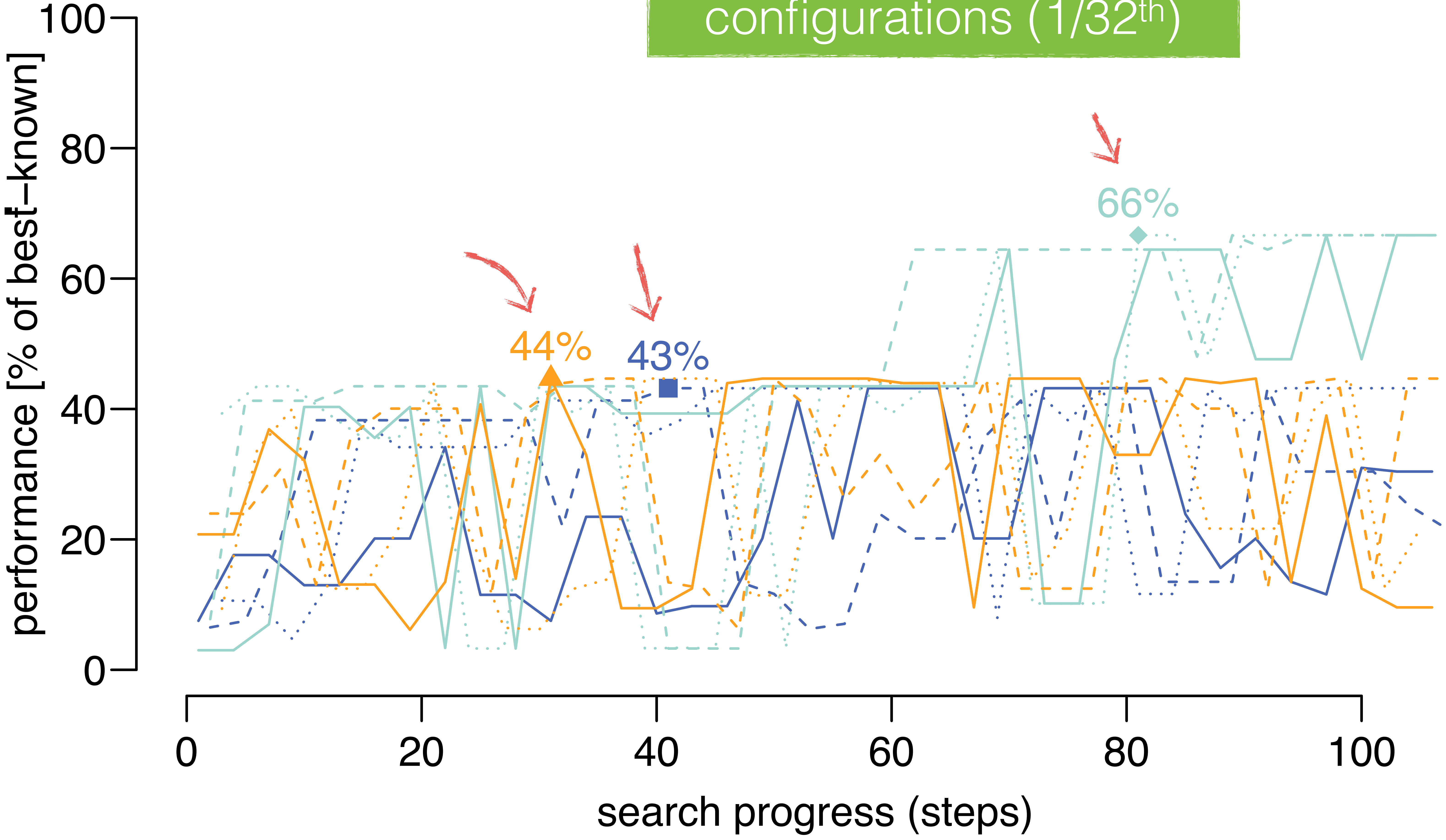
☺ Explores arbitrary fraction

✗ Performance varies

✗ Meta-parameter

✗ Local optima

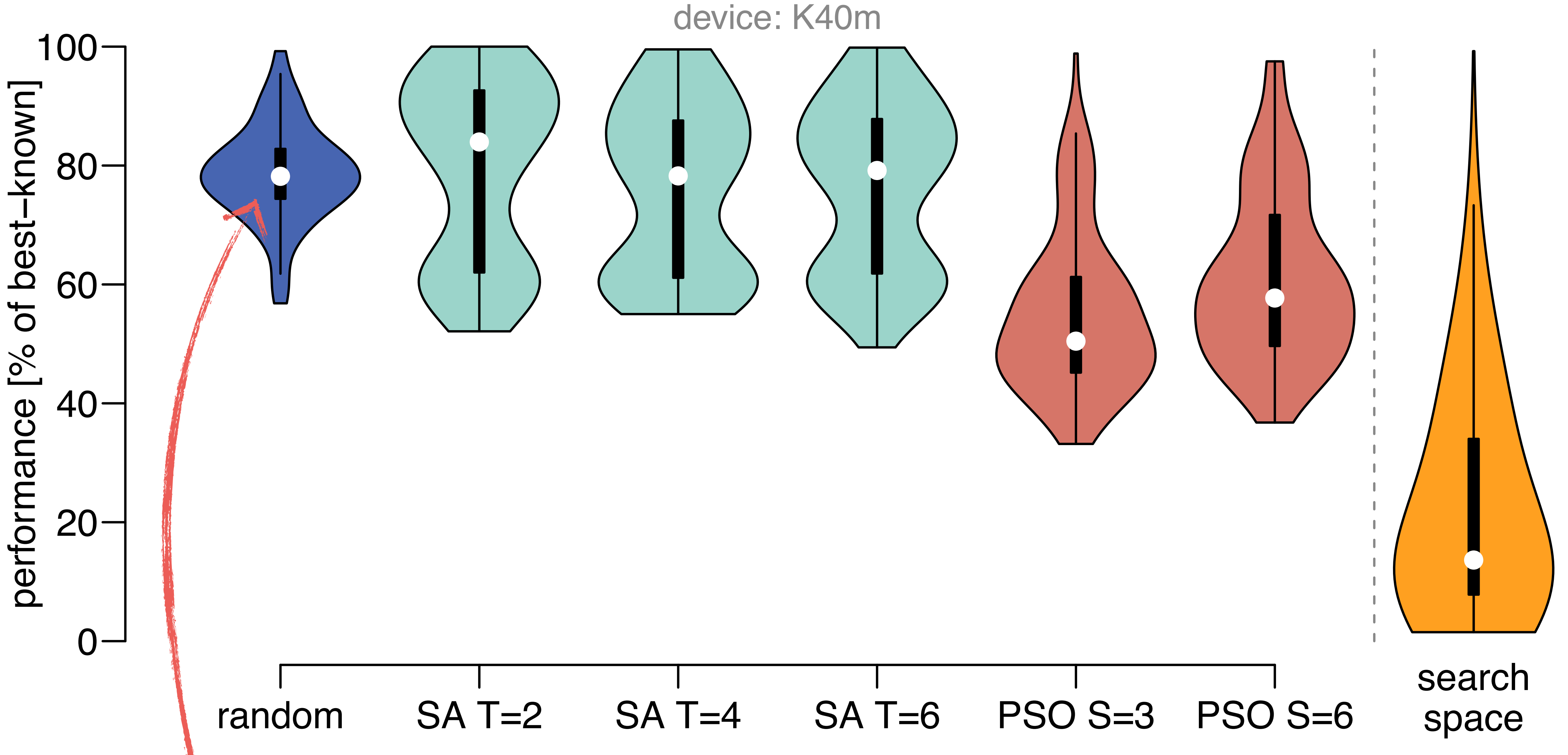
Example: 107 out of 3424 configurations (1/32th)



Colours: 3 example runs

Line-types: 3 swarms

Search strategies evaluation



average best result of 128 searches

meta-parameters for SA and PSO

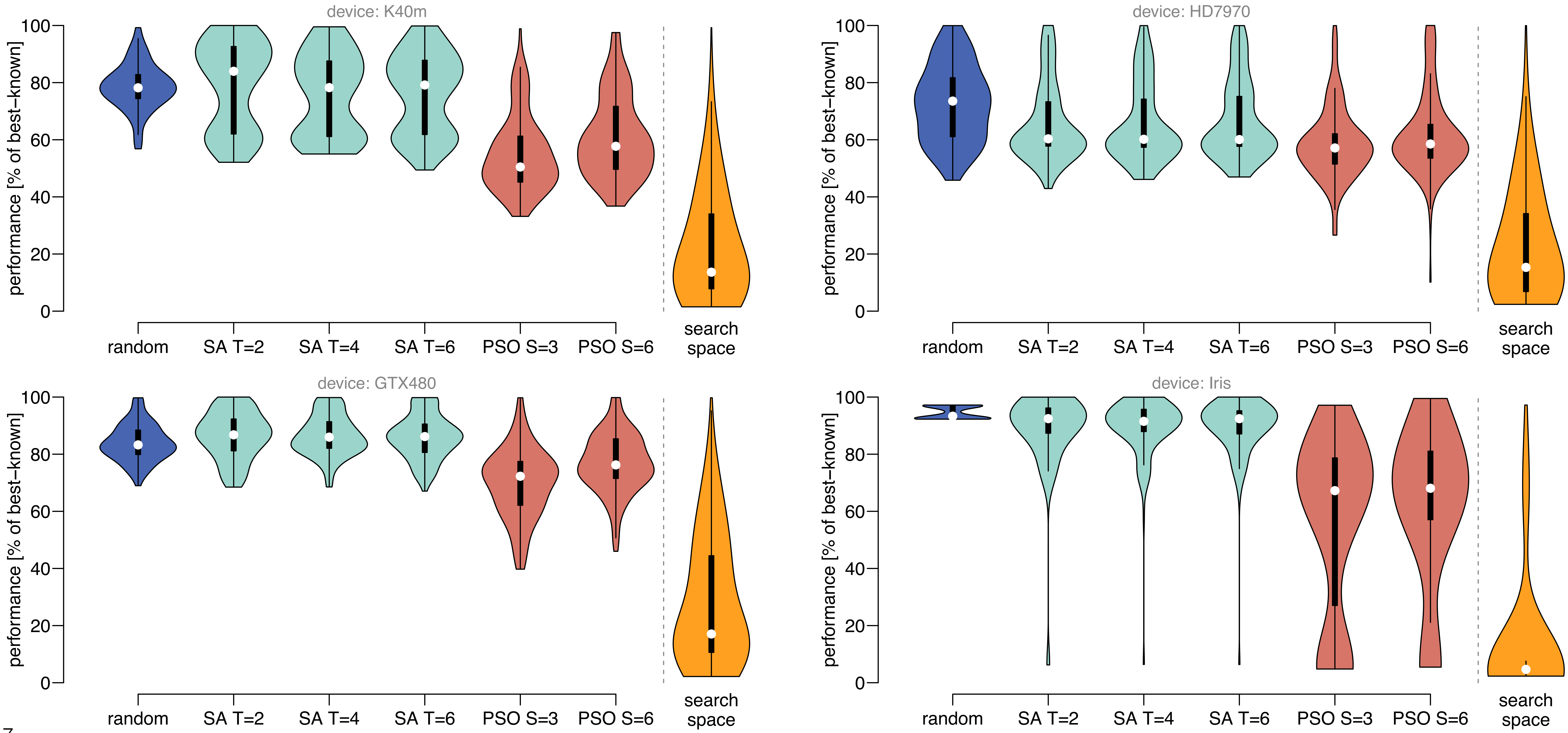
Conclusions:

- PSO performs poorly
- SA perform good, but not much better than random search

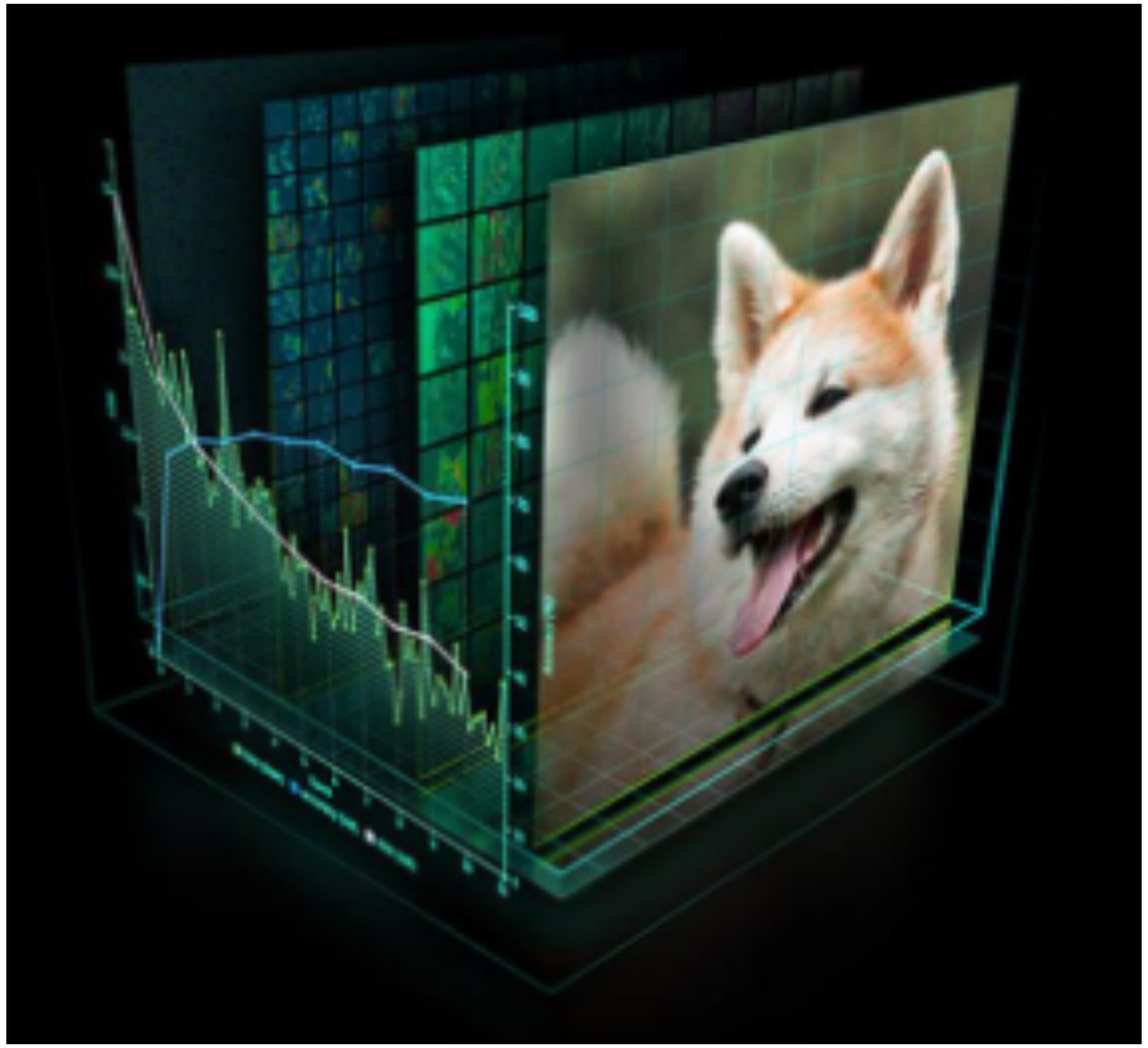
Each search: 107 out of 3424 configurations (1/32th)

[1]: C. Nugteren and V. Codreanu. CLTune: A Generic Auto-Tuner for OpenCL Kernels. IEEE MCSoC '15.

Search strategies evaluation



Solution: machine learning?

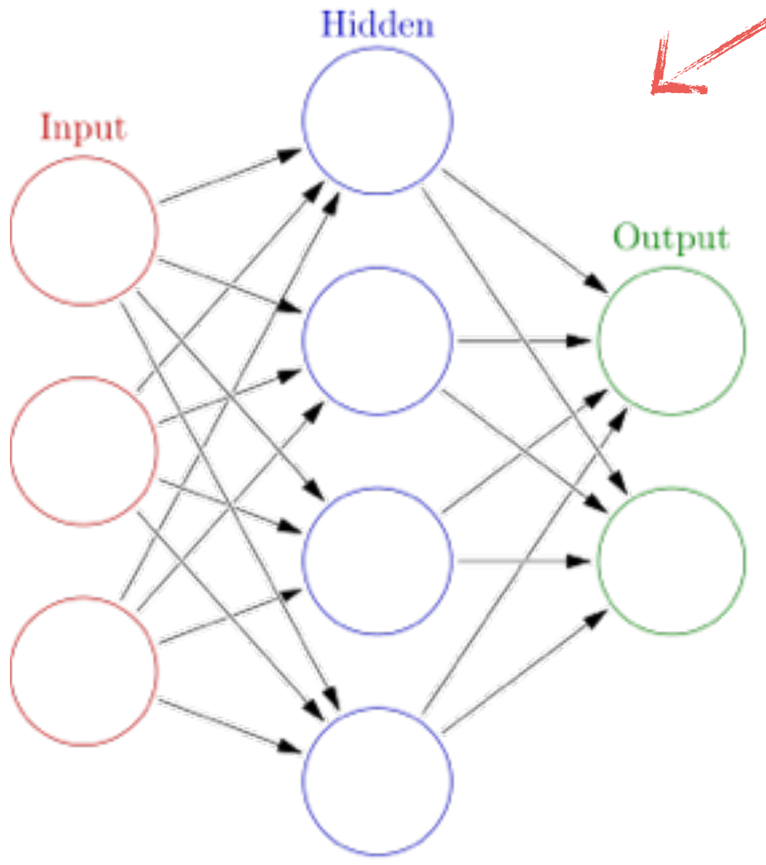


Machine learning an auto-tuner

Evaluate subset of all configurations (e.g. 100 out of 3424) → Train model with the examples → Predict execution time for all other configurations

↓

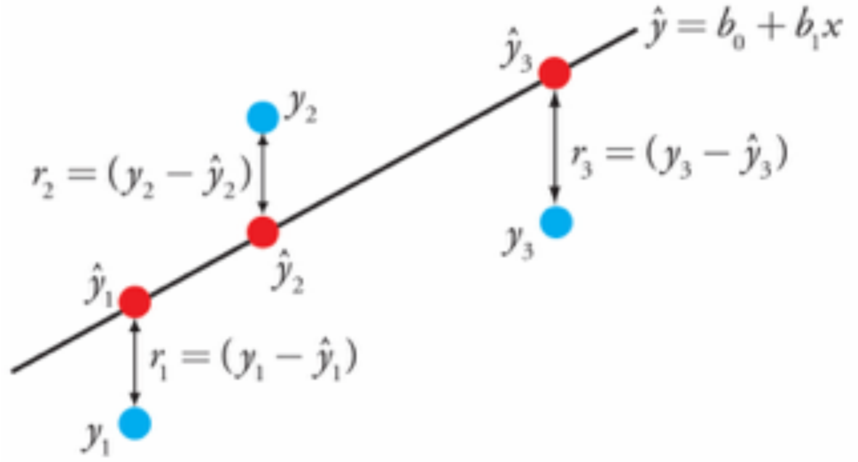
(take best 10 and evaluate them on actual hardware)



Neural network
(3-layer fully connected)

```
input:
parameter
configuration

output:
execution time
```

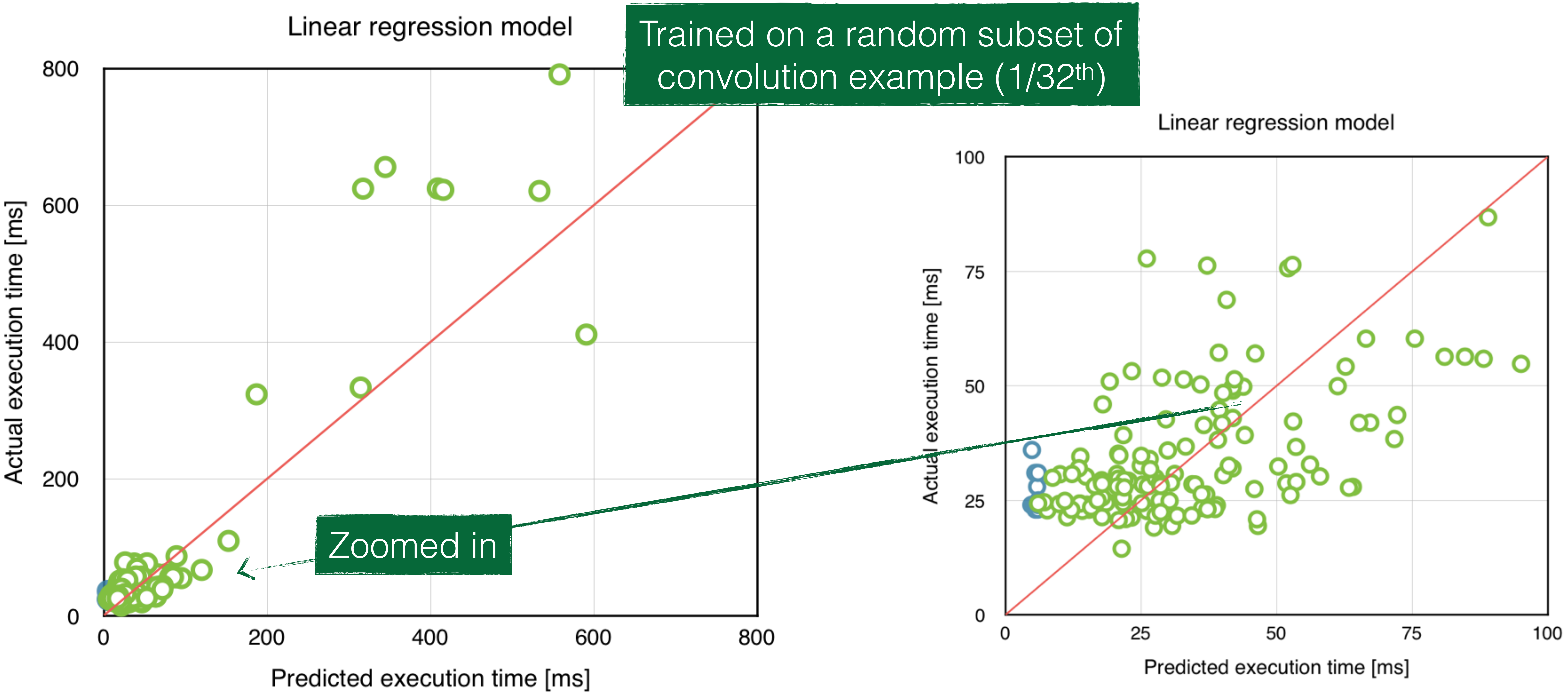


Linear regression

[2]: T.L. Falch and A.C. Elster. Machine Learning Based Auto-tuning for Enhanced OpenCL Performance Portability.

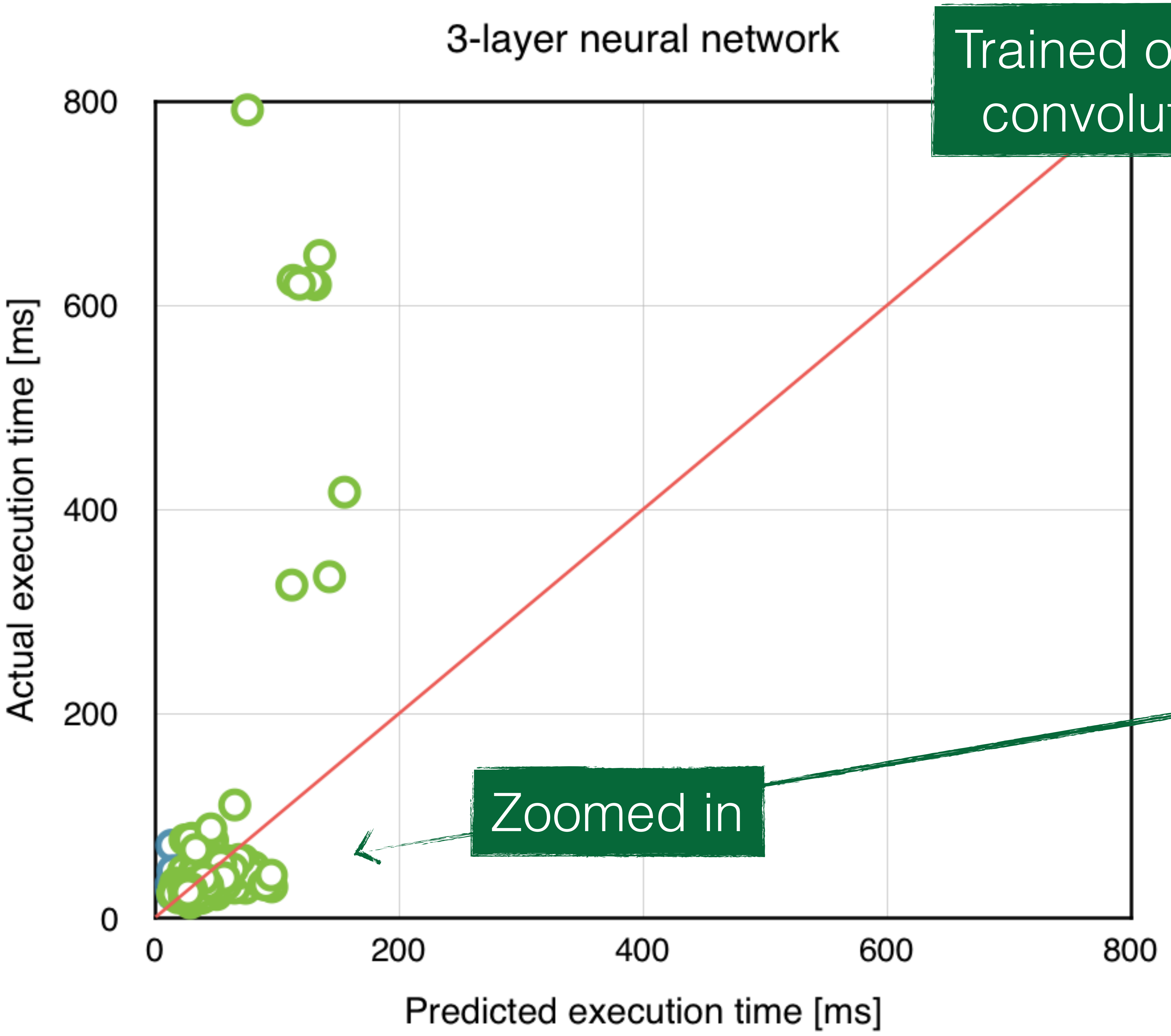
Machine learning an auto-tuner

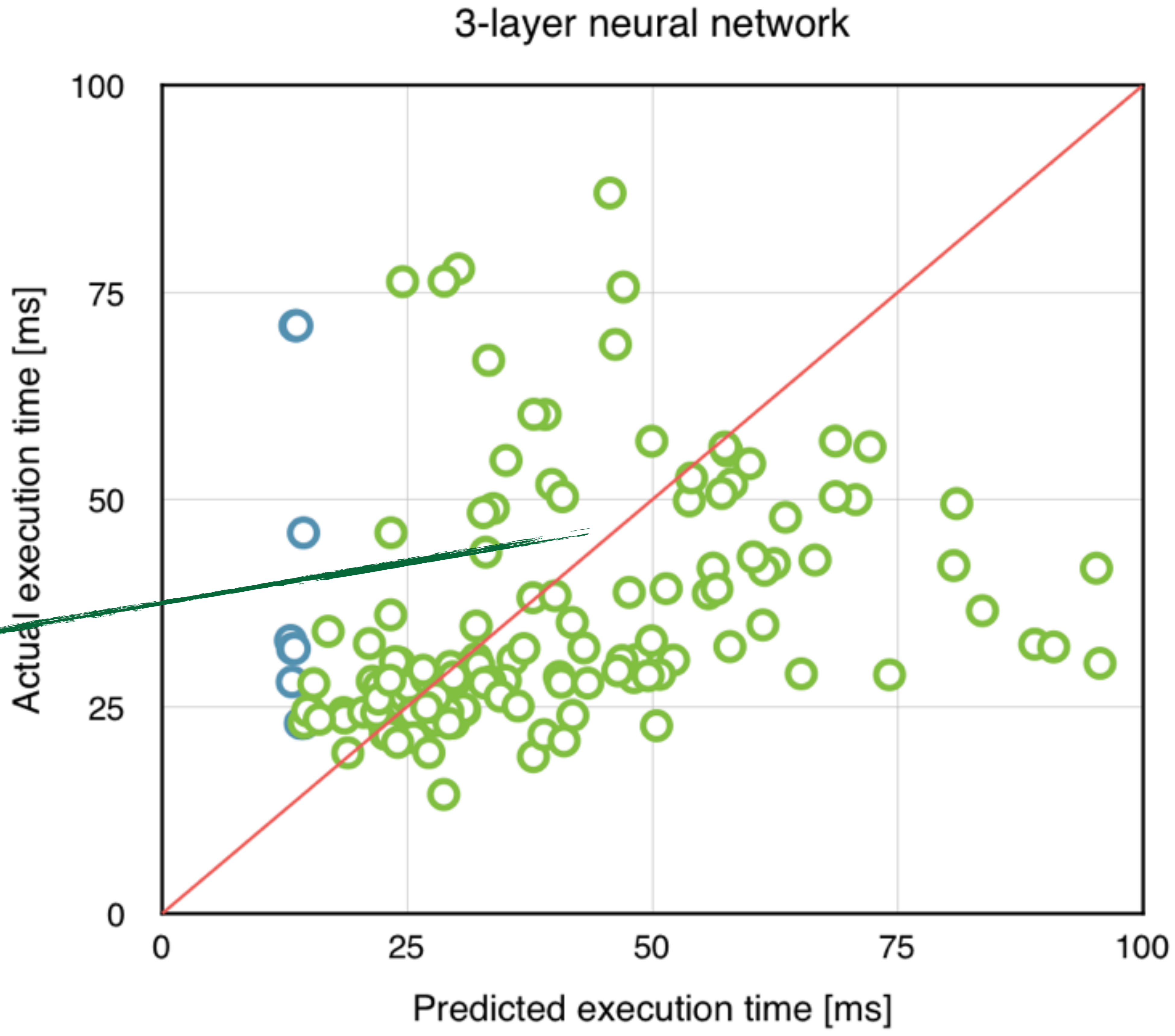
Trained on a random subset of convolution example (1/32th)



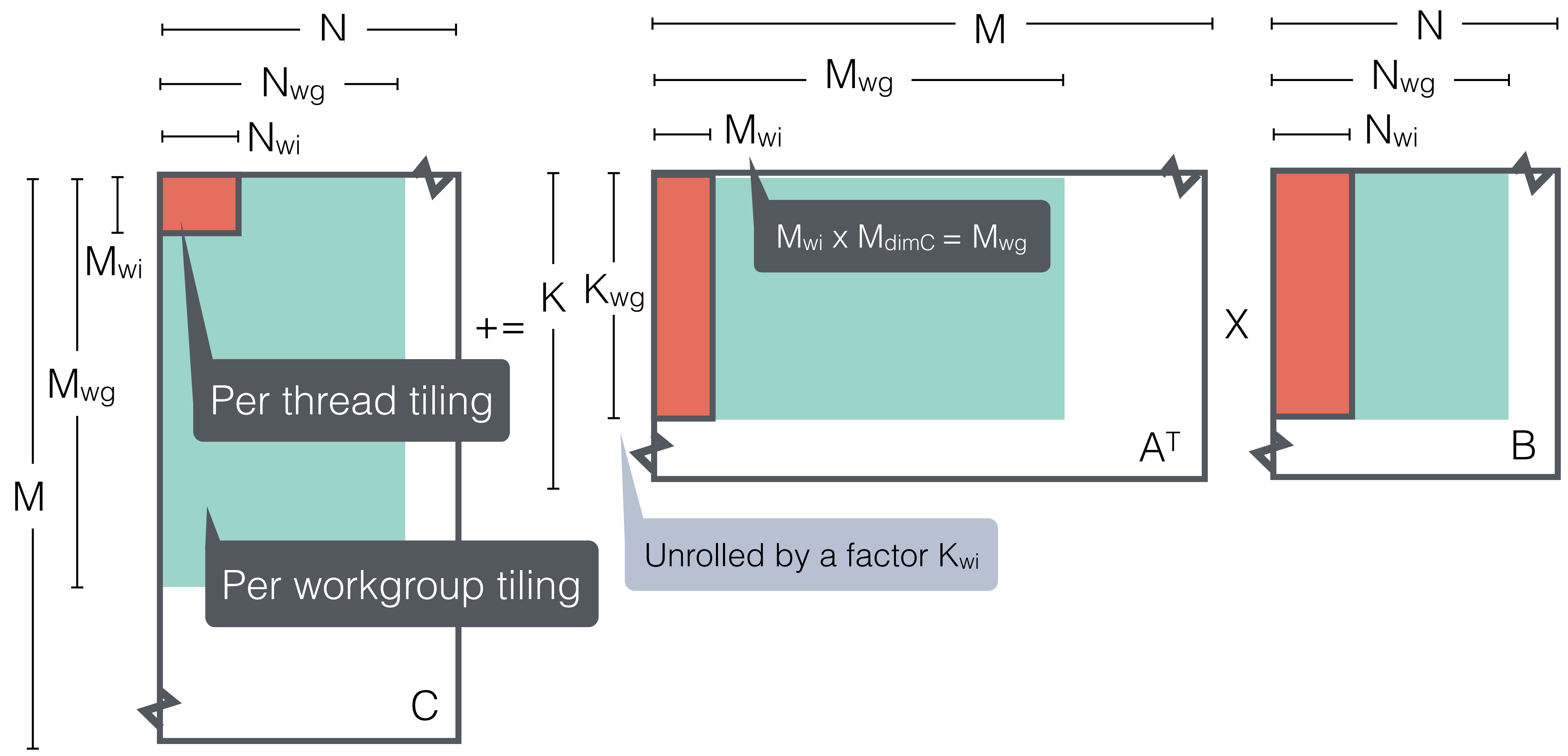
Machine learning an auto-tuner

Trained on a random subset of convolution example (1/32th)





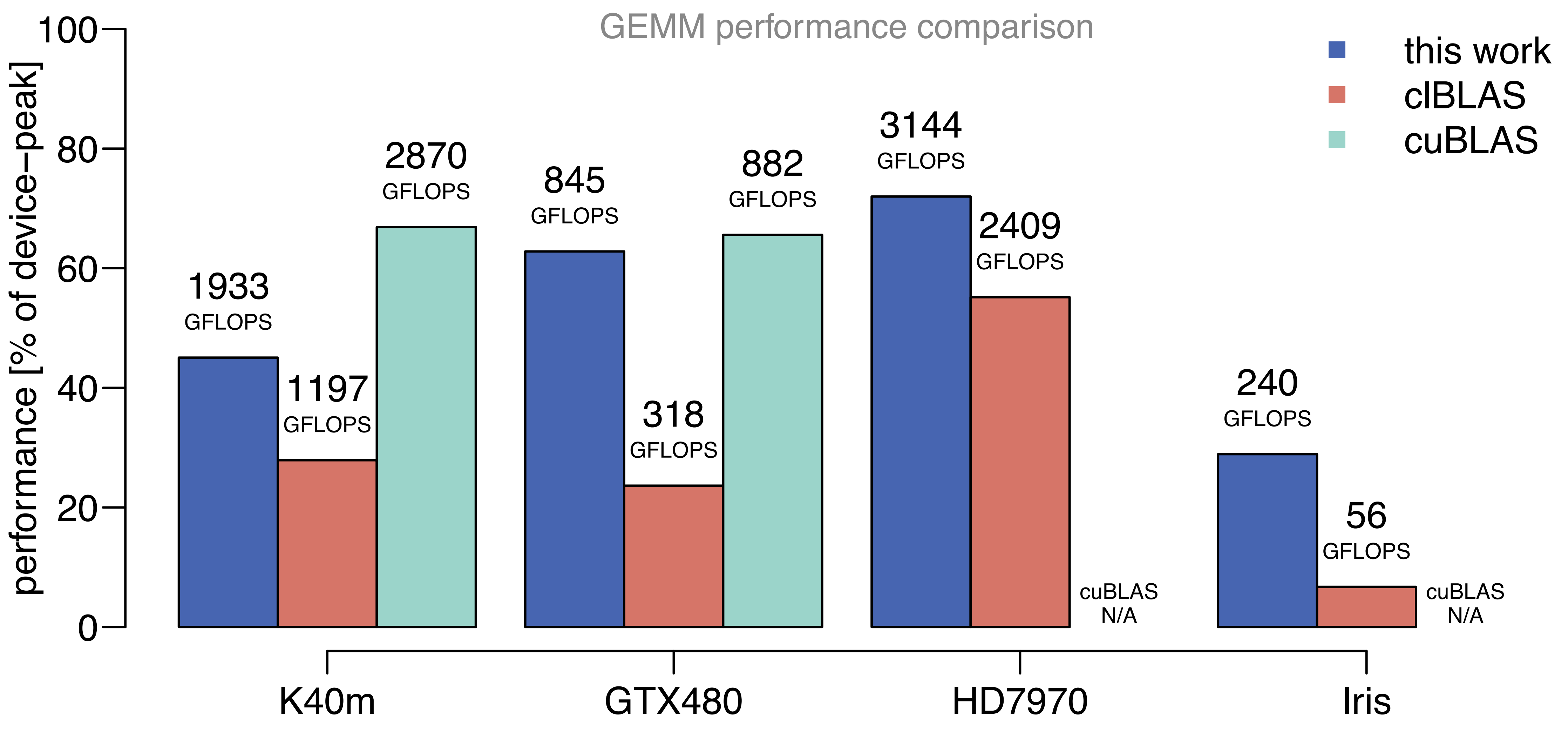
GEMM case-study

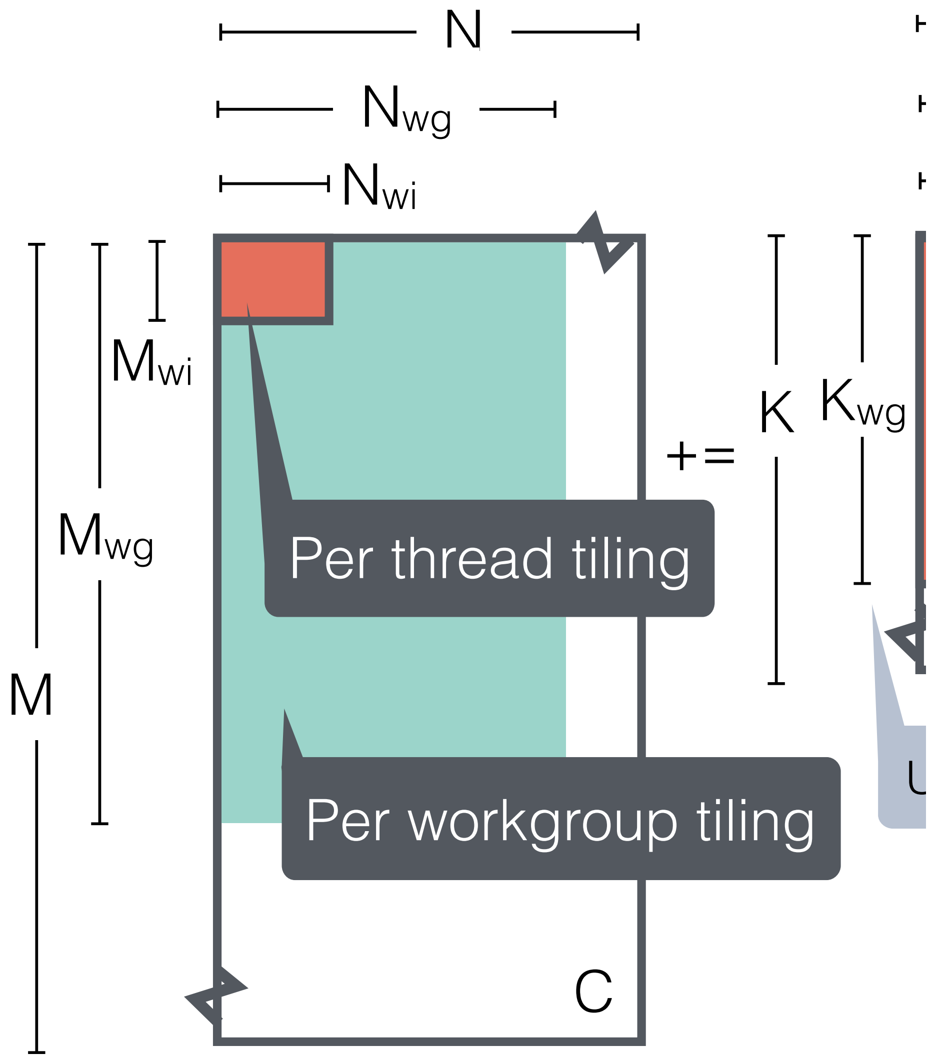


Conclusions:

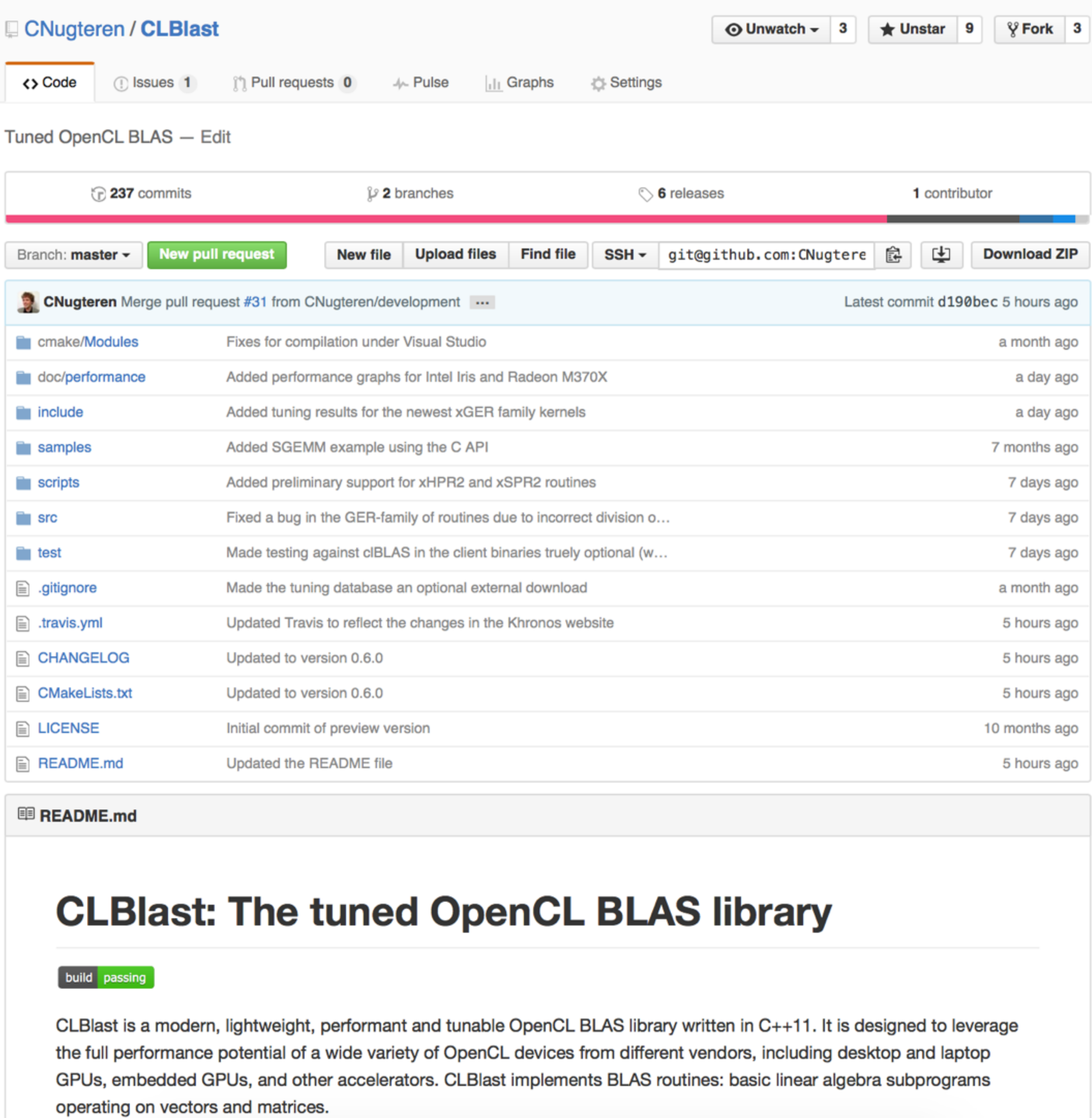
- Different best parameters for different devices
- Performance better than clBLAS, but not as good as assembly-tuned cuBLAS

parameter(s)	allowed values	best parameters per device K40m	GTX480	HD7970	Iris
M_{wg}, N_{wg}, K_{wg}	{16,32,64,128}	128,128,16	64,64,32	128,128,32	64,64,16
M_{dimC}, N_{dimC}	{8,16,32}	16,16	8,16	16,16	8,8
$L\$_A, L\$_B$	{yes,no}	yes, yes	yes, yes	yes, yes	yes, yes
M_{dimA}, N_{dimB}	{8,16,32}	32,16	32,32	32,32	8,16
M_{stride}, N_{stride}	{yes,no}	yes, no	yes, no	no, yes	yes, yes
M_{vec}, N_{vec}	{1,2,4,8}	2,1	2,2	4,4	4,4
K_{wi}	{2,8}	8	8	2	8





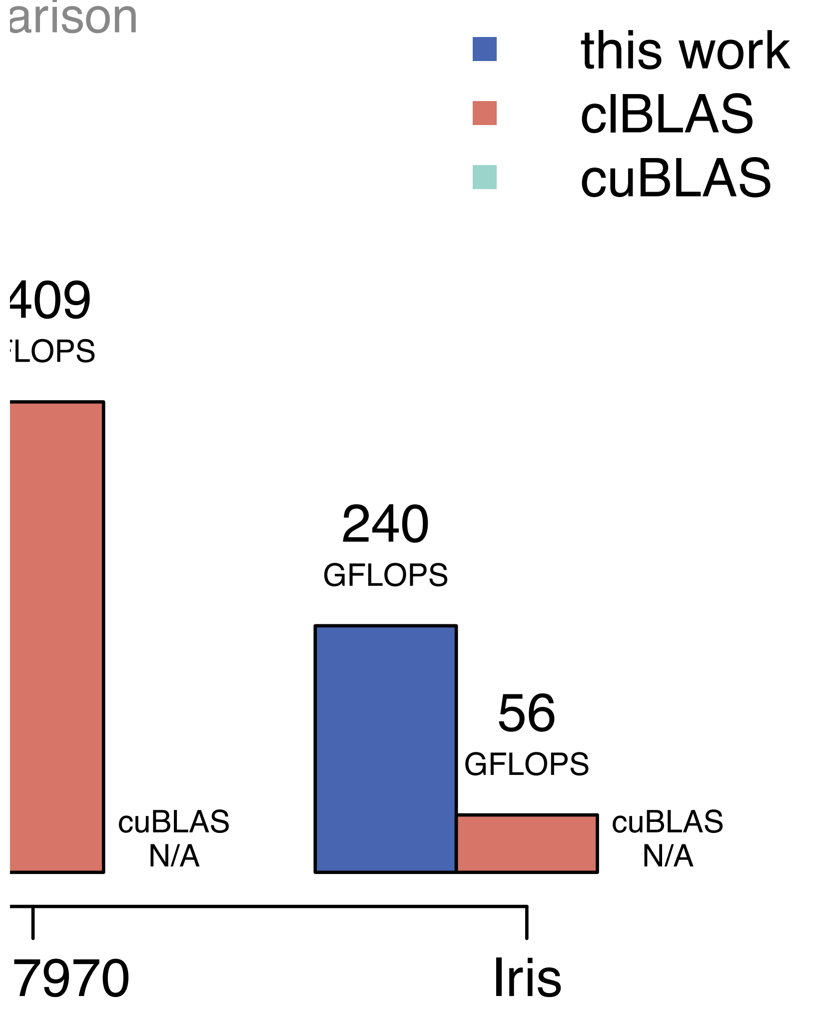
parameter(s)	allowed values
M_{wg}, N_{wg}, K_{wg}	{16,32,64,128}
M_{dimC}, N_{dimC}	{8,16,32}
$L\$_A, L\$_B$	{yes,no}
M_{dimA}, N_{dimB}	{8,16,32}
M_{stride}, N_{stride}	{yes,no}
M_{vec}, N_{vec}	{1,2,4,8}
K_{wi}	{2,8}



ameters for

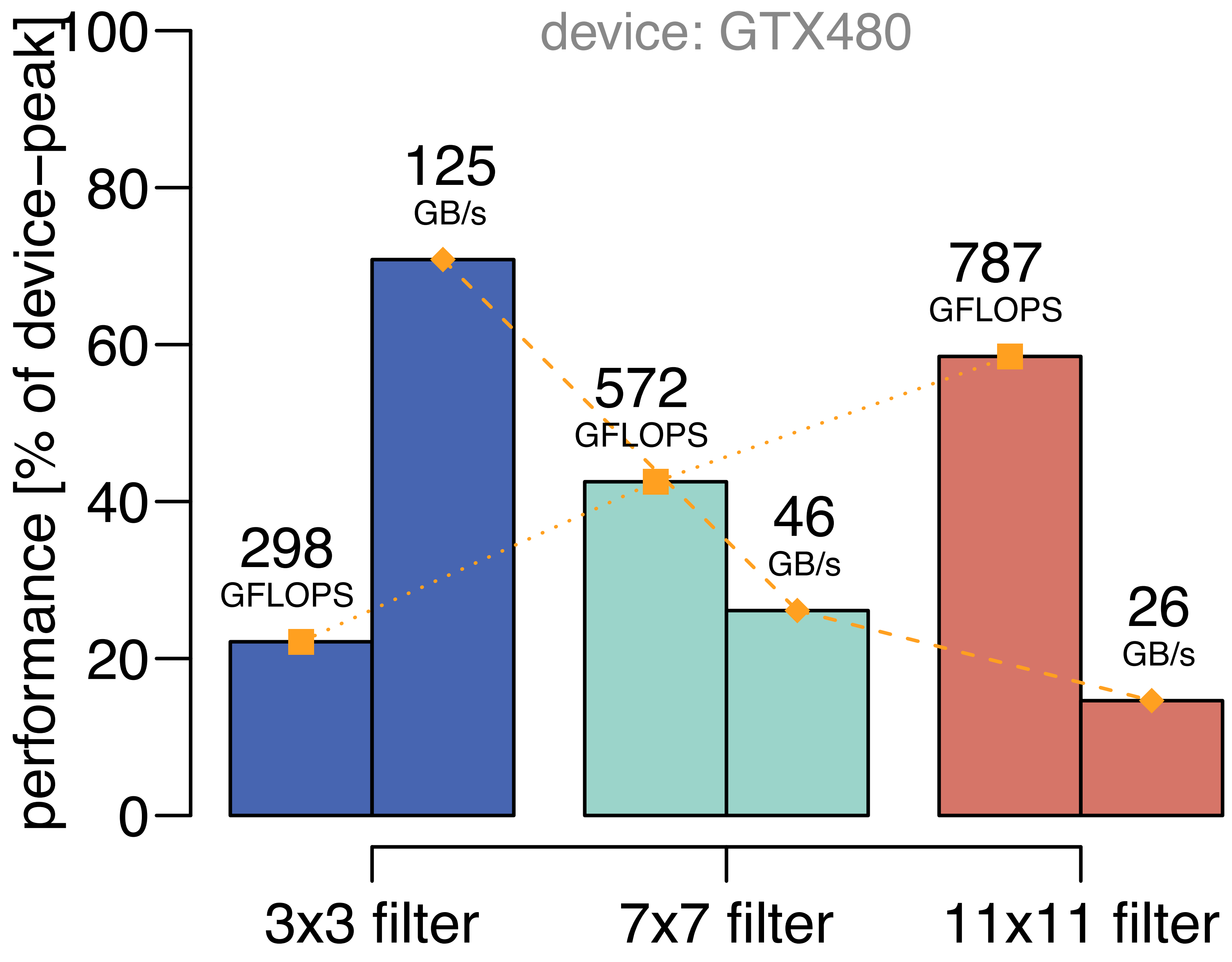
er than clBLAS,
s assembly-

arison



Convolution case-study

applied to a filter of size	best parameters for 3x3	7x7	11x11
3x3	100%	82%	64%
7x7	65%	100%	83%
11x11	66%	75%	100%



Conclusions:

- Different best parameters for different:
  - devices (see paper)
  - filter-sizes
- Performance equal or better than the state-of-the-art [3]

[3]: B. Van Werkhoven, J. Maassen, H.E. Bal, and F.J. Seinstra. Optimizing Convolution Operations on GPUs Using Adaptive Tiling.

Wha ort?

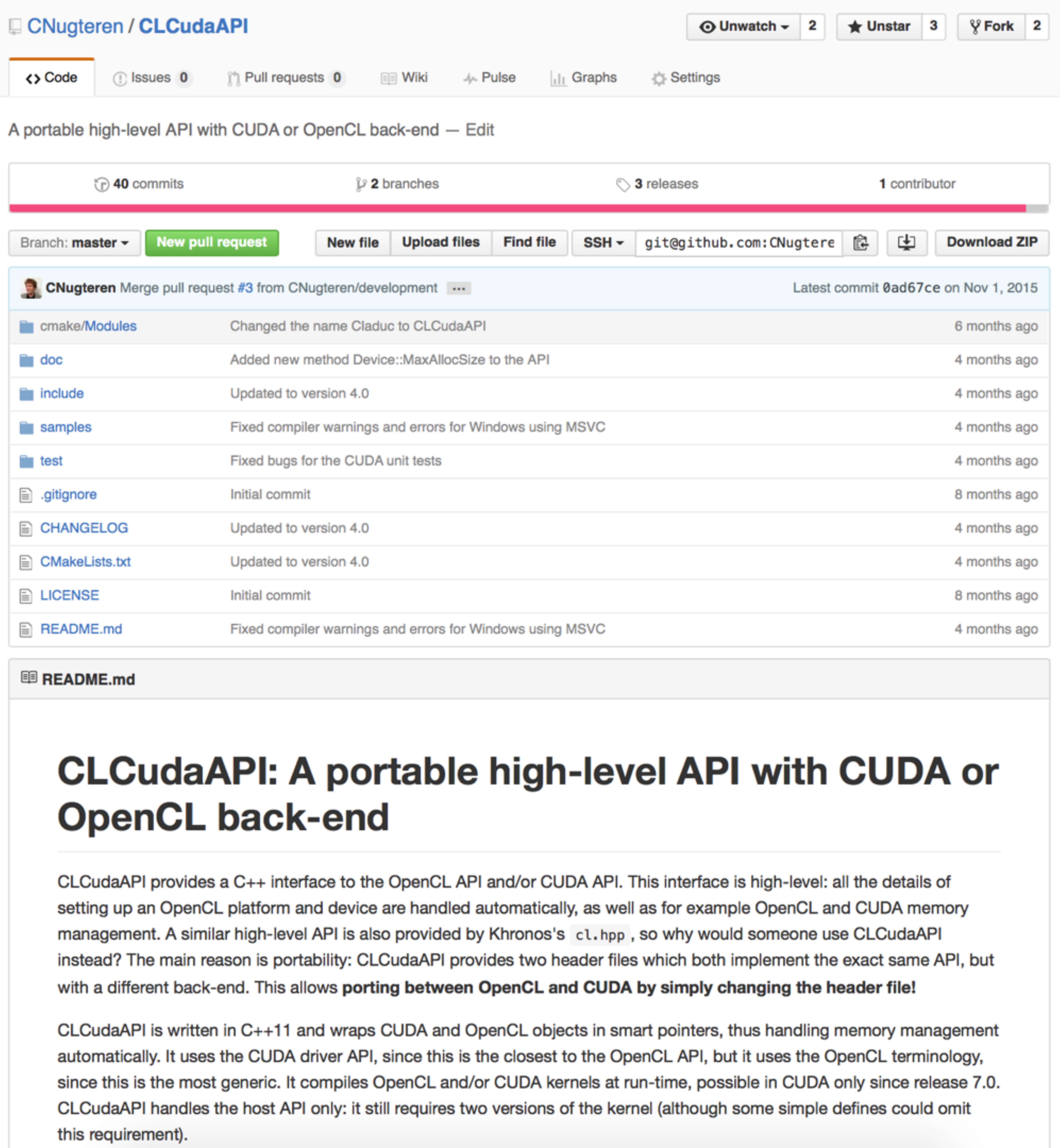
Written for

- Uses the

```
// Copy buf
auto gpuMem
gpuMemory.W
```

- And **CLC**

- Switch b
- Uses the vrtc)


CNugteren / CLCudaAPI

Unwatch 2 | Unstar 3 | Fork 2

Code | Issues 0 | Pull requests 0 | Wiki | Pulse | Graphs | Settings

A portable high-level API with CUDA or OpenCL back-end — Edit

40 commits | 2 branches | 3 releases | 1 contributor

Branch: master | New pull request | New file | Upload files | Find file | SSH | git@github.com:CNugtere | Download ZIP

CNugteren Merge pull request #3 from CNugteren/development ... Latest commit 0ad67ce on Nov 1, 2015

cmake/Modules	Changed the name Claduc to CLCudaAPI	6 months ago
doc	Added new method Device::MaxAllocSize to the API	4 months ago
include	Updated to version 4.0	4 months ago
samples	Fixed compiler warnings and errors for Windows using MSVC	4 months ago
test	Fixed bugs for the CUDA unit tests	4 months ago
.gitignore	Initial commit	8 months ago
CHANGELOG	Updated to version 4.0	4 months ago
CMakeLists.txt	Updated to version 4.0	4 months ago
LICENSE	Initial commit	8 months ago
README.md	Fixed compiler warnings and errors for Windows using MSVC	4 months ago

README.md

CLCudaAPI: A portable high-level API with CUDA or OpenCL back-end

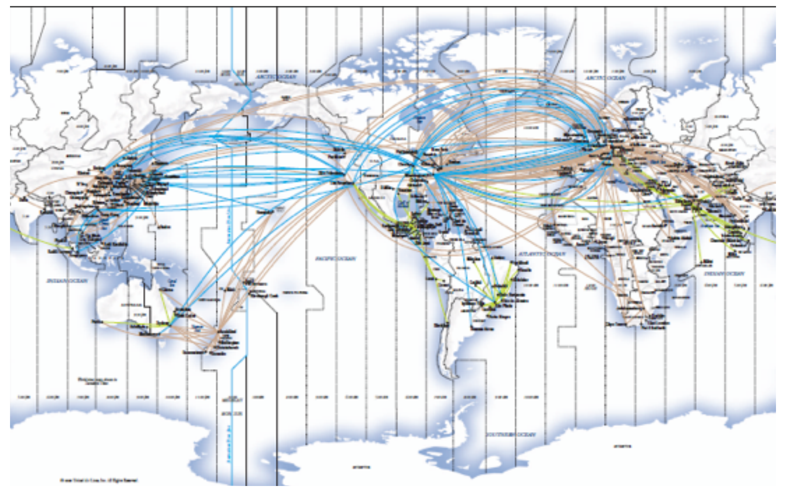
CLCudaAPI provides a C++ interface to the OpenCL API and/or CUDA API. This interface is high-level: all the details of setting up an OpenCL platform and device are handled automatically, as well as for example OpenCL and CUDA memory management. A similar high-level API is also provided by Khronos's `cl.hpp`, so why would someone use CLCudaAPI instead? The main reason is portability: CLCudaAPI provides two header files which both implement the exact same API, but with a different back-end. This allows **porting between OpenCL and CUDA by simply changing the header file!**

CLCudaAPI is written in C++11 and wraps CUDA and OpenCL objects in smart pointers, thus handling memory management automatically. It uses the CUDA driver API, since this is the closest to the OpenCL API, but it uses the OpenCL terminology, since this is the most generic. It compiles OpenCL and/or CUDA kernels at run-time, possible in CUDA only since release 7.0. CLCudaAPI handles the host API only: it still requires two versions of the kernel (although some simple defines could omit this requirement).

Better Than All the Rest:
Finding Maximum-Performance GPU Kernels Using Auto-Tuning

Auto-tuning GPU kernels:

- Large search-space
- Wide variety of devices
- User-parameter dependent

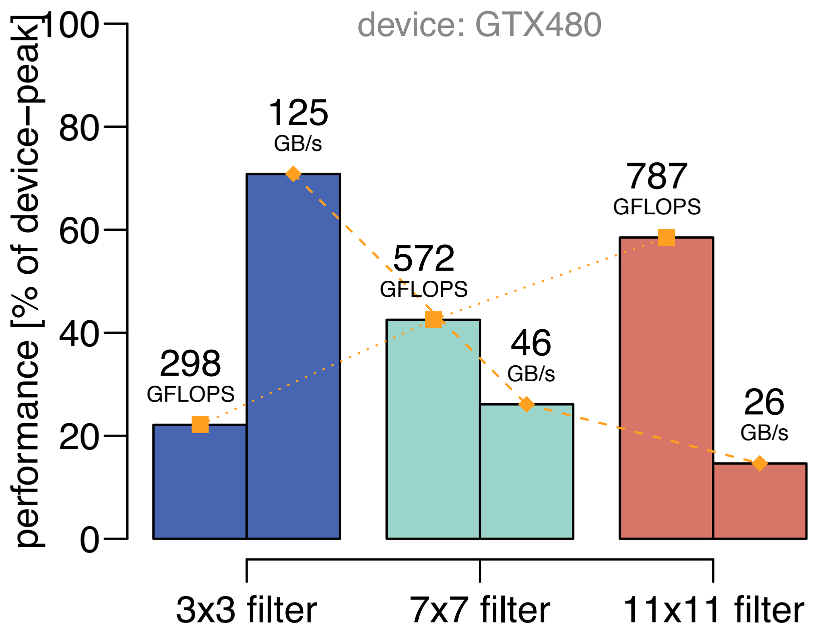


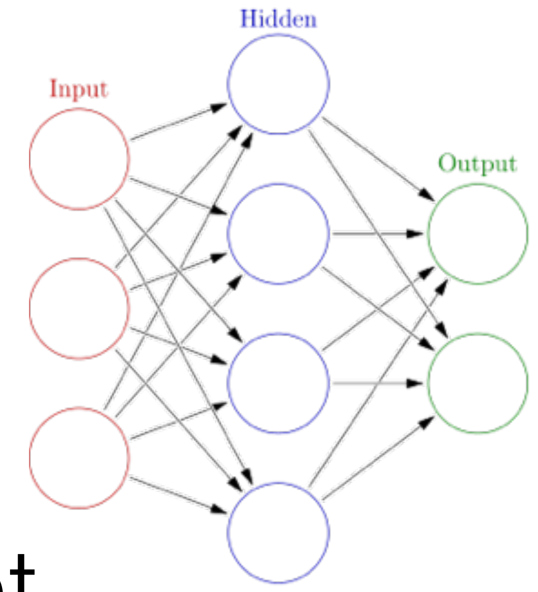
Advanced search strategies:

- Simulated annealing
- Particle swarm optimisation

Case-studies:

- Fastest 2D convolution
- CLBlast matrix-multiplication library





Machine-learning:

- Train a model on a small subset
- Use the model to predict the remainder

Source-code on GitHub:
https://github.com/CNugteren/CLTune

[More info]: C. Nugteren and V. Codreanu. CLTune: A Generic Auto-Tuner for OpenCL Kernels. IEEE MCSoC '15.

Slides available @ www.cedricnugteren.nl