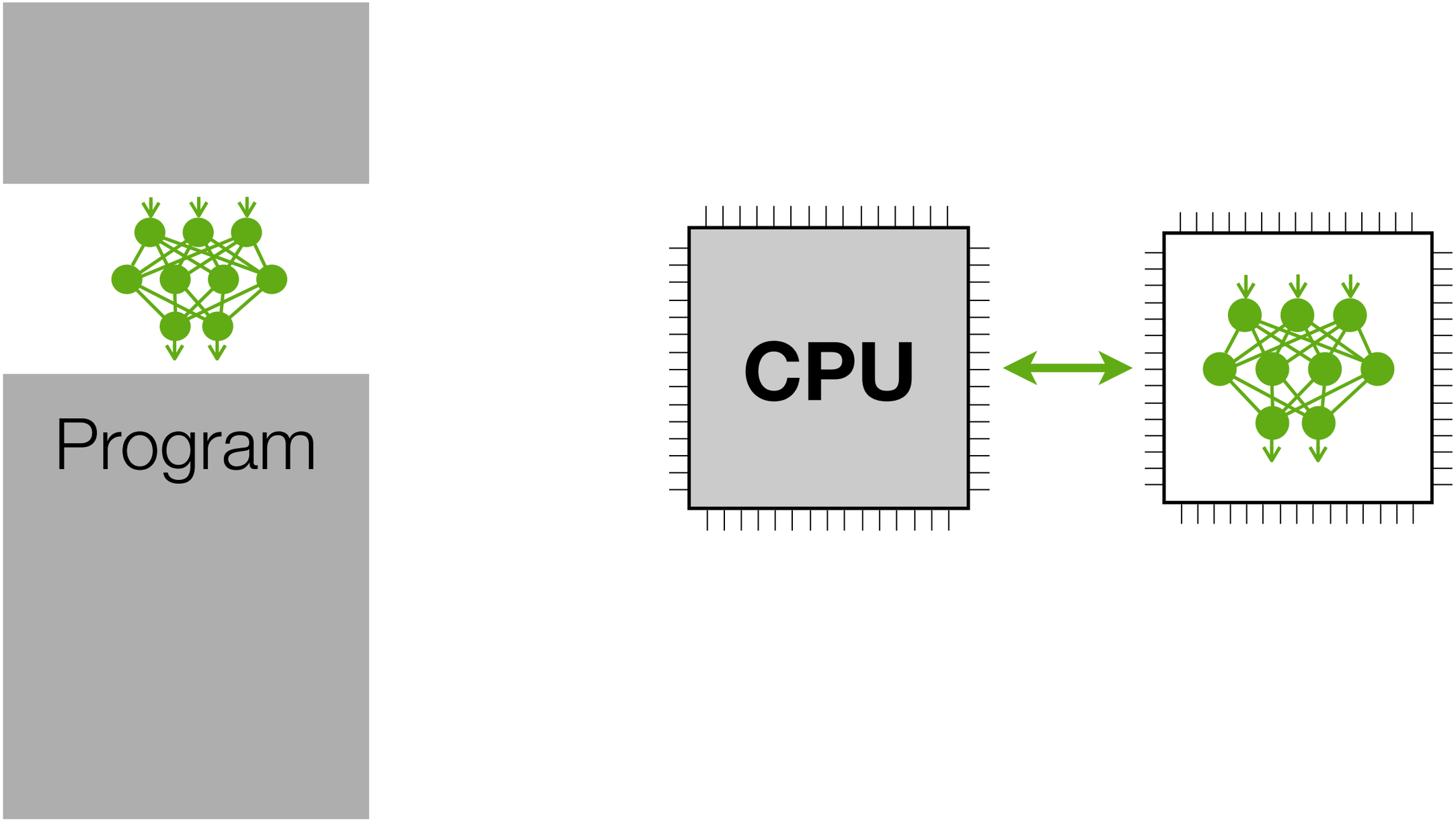
Neural Acceleration for General-Purpose Approximate Programs

Hadi Esmaeilzadeh
Adrian Sampson
Luis Ceze
Doug Burger

University of Washington
Microsoft Research


Program
CPU

computer vision

machine learning

sensory data

physical simulation

information retrieval

augmented reality

image rendering

JPL & Rob Hogg

NASA

thefrugalgirl.com

computer vision

machine learning

sensory data

physical simulation

information retrieval

augmented reality

image rendering

Approximate computing

Probabilistic CMOS designs
[Rice, NTU, Georgia Tech…]

Stochastic processors
[Illinois]

Code perforation transformations
[MIT]

Relax software fault recovery
[de Kruijf et al., ISCA 2010]

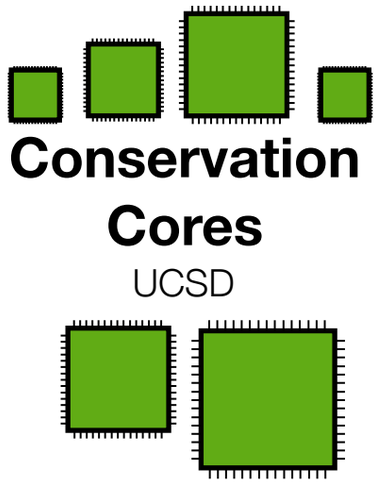
Green runtime system
[Baek and Chilimbi, PLDI 2010]

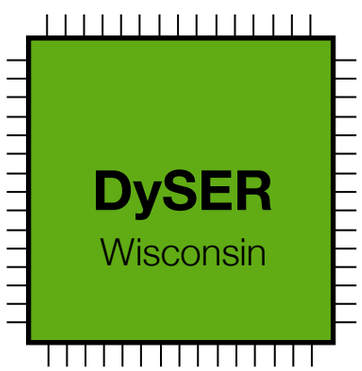
Flikker approximate DRAM
[Liu et al., ASPLOS 2011]

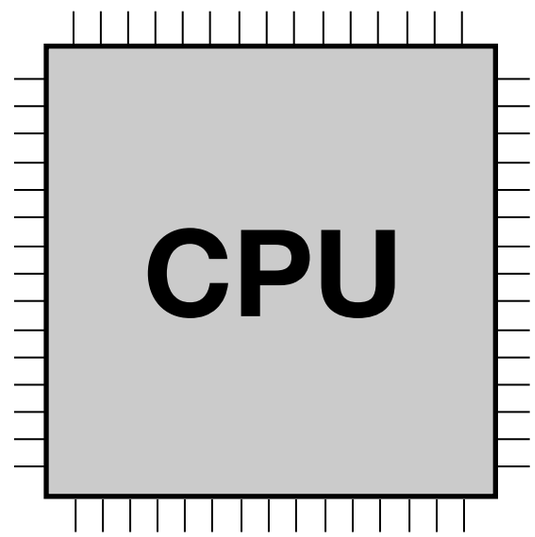
EnerJ programming language
[PLDI 2011]

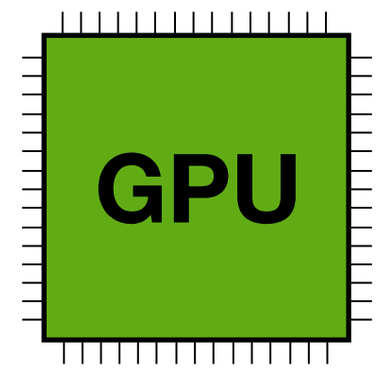
Truffle dual-voltage architecture
[ASPLOS 2012]

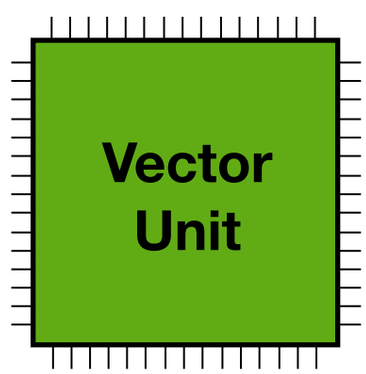
Accelerators

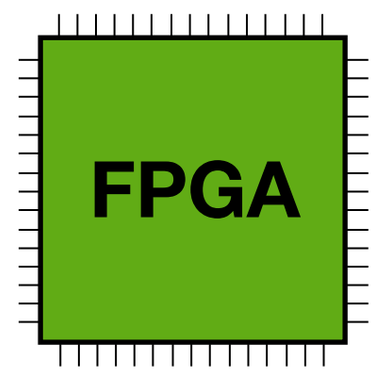




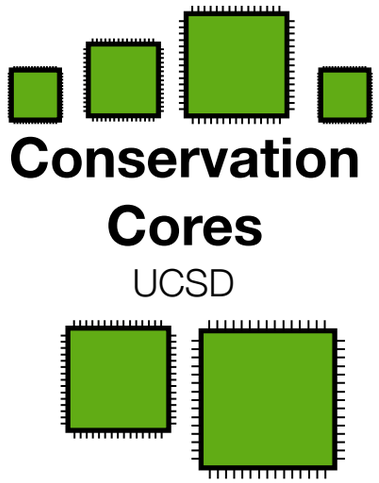


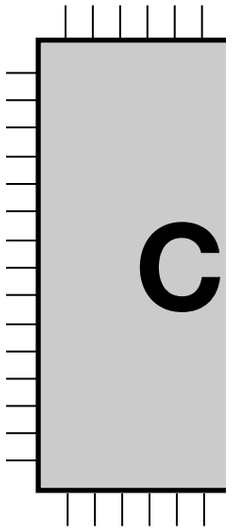


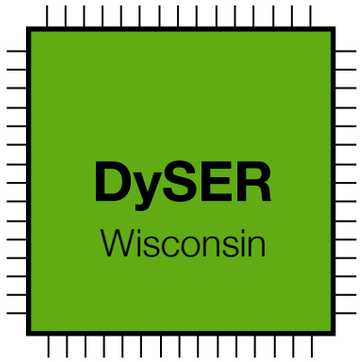


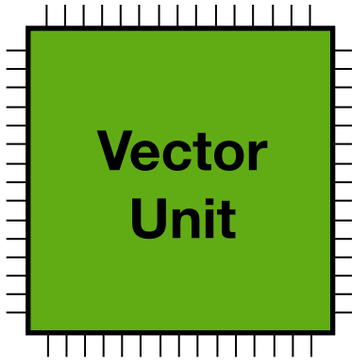


Accelerators









Approximate computing

computer vision

machine learning

sensory data

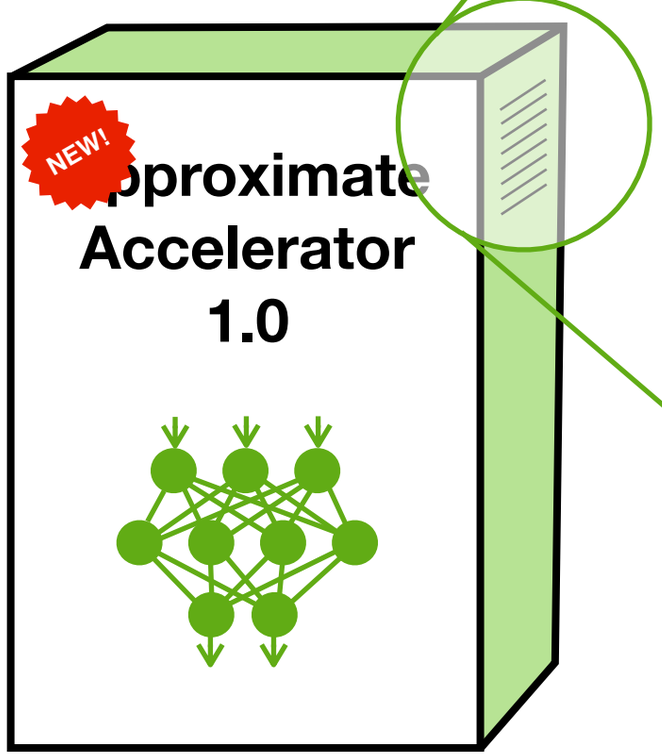
physical simulation

information retrieval

augmented reality

image rendering

An accelerator for approximate computations

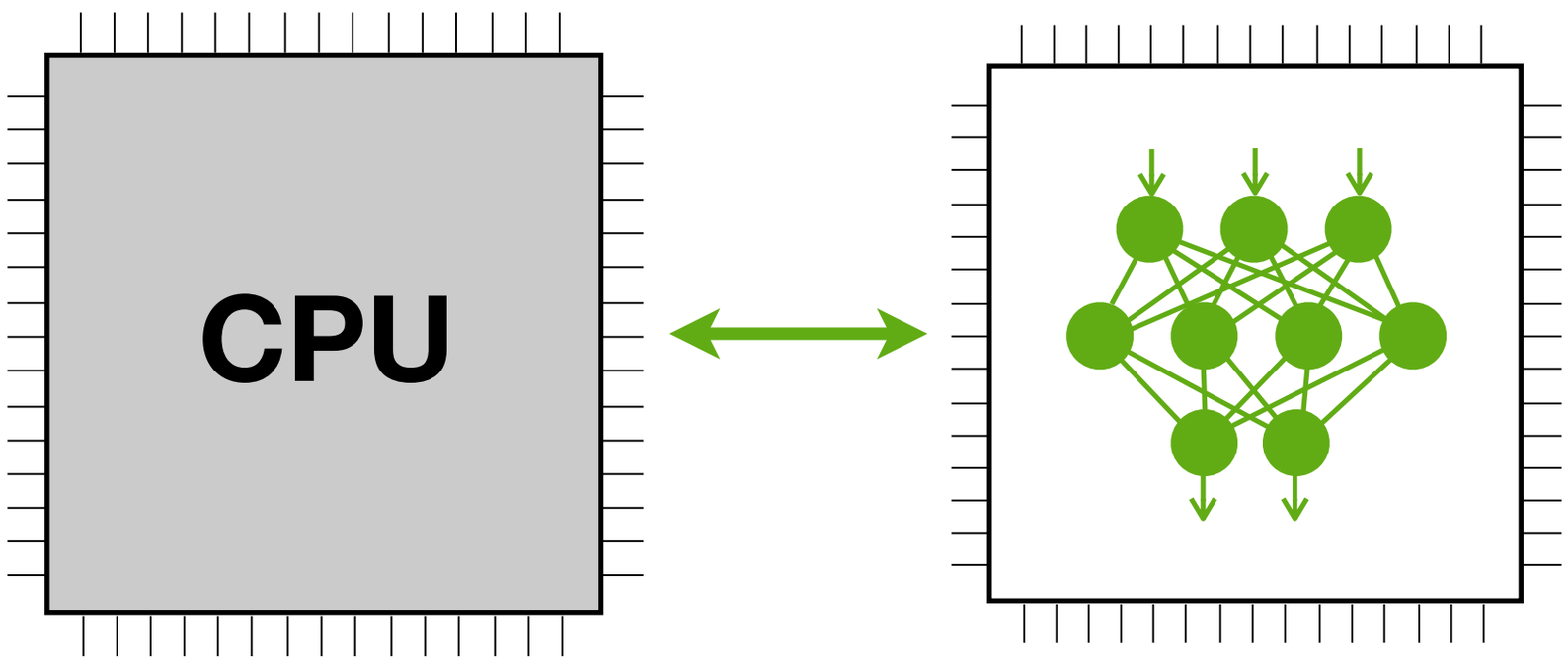


- ✓ Mimics functions written in traditional languages!
- ✓ Runs more efficiently than a CPU or a precise accelerator!
- ✓ May introduce small errors!

Neural networks are function approximators

Trainable: implements many functions

Highly parallel



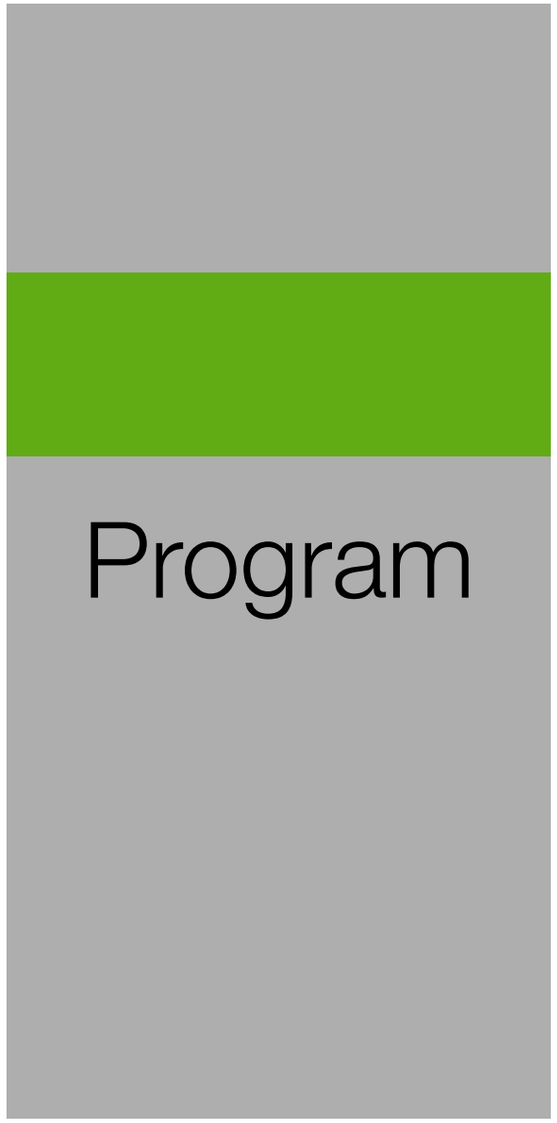
Very efficient hardware implementations

Fault tolerant

[Temam, ISCA 2012]

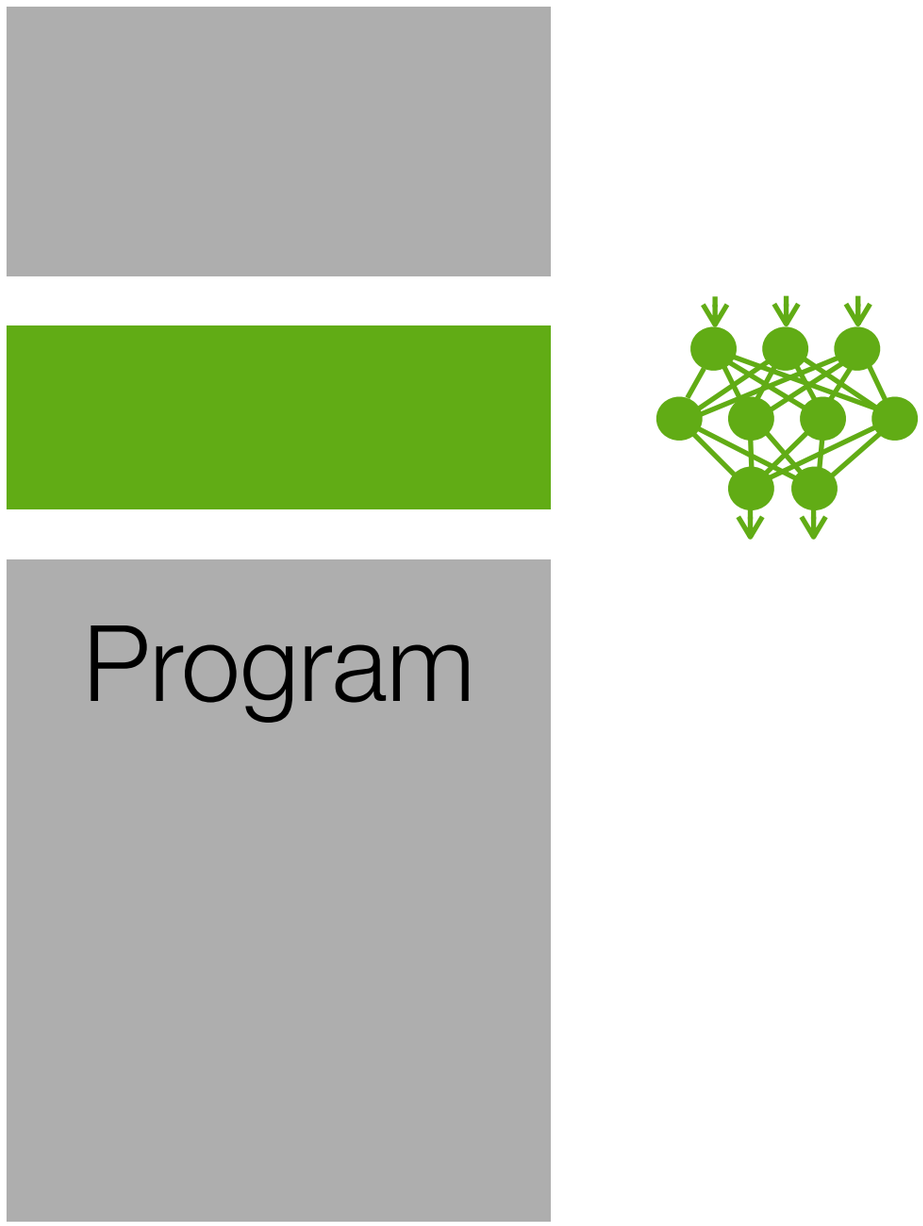
Neural acceleration

Neural acceleration



Annotate an approximate program component

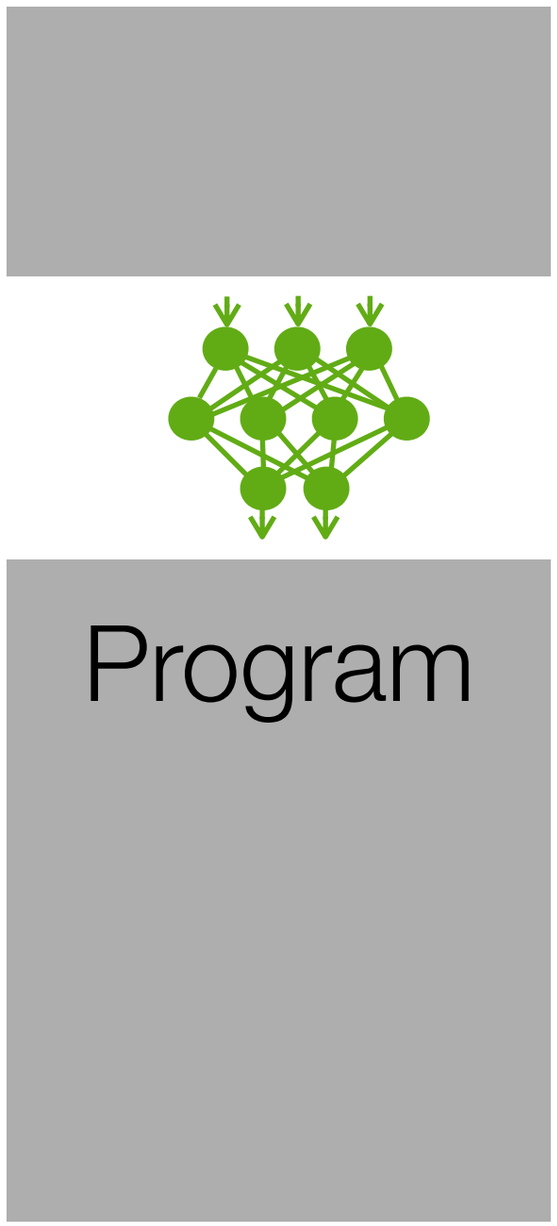
Neural acceleration



Annotate an approximate program component

Compile the program and train a neural network

Neural acceleration



Annotate an approximate program component

Compile the program and train a neural network

Execute on a fast Neural Processing Unit (NPU)

Neural acceleration

1 **Annotate** an approximate program component

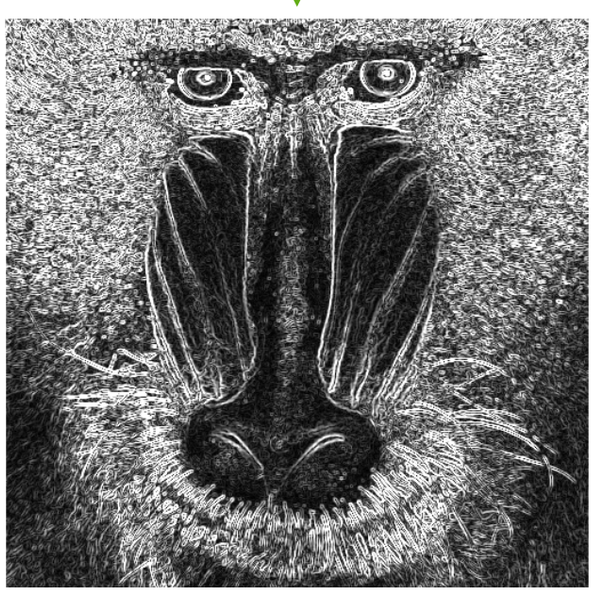
2 **Compile** the program and train a neural network

3 **Execute** on a fast Neural Processing Unit (NPU)

4 **Improve** performance 2.3x and energy 3.0x on average

Programming model

edgeDetection()



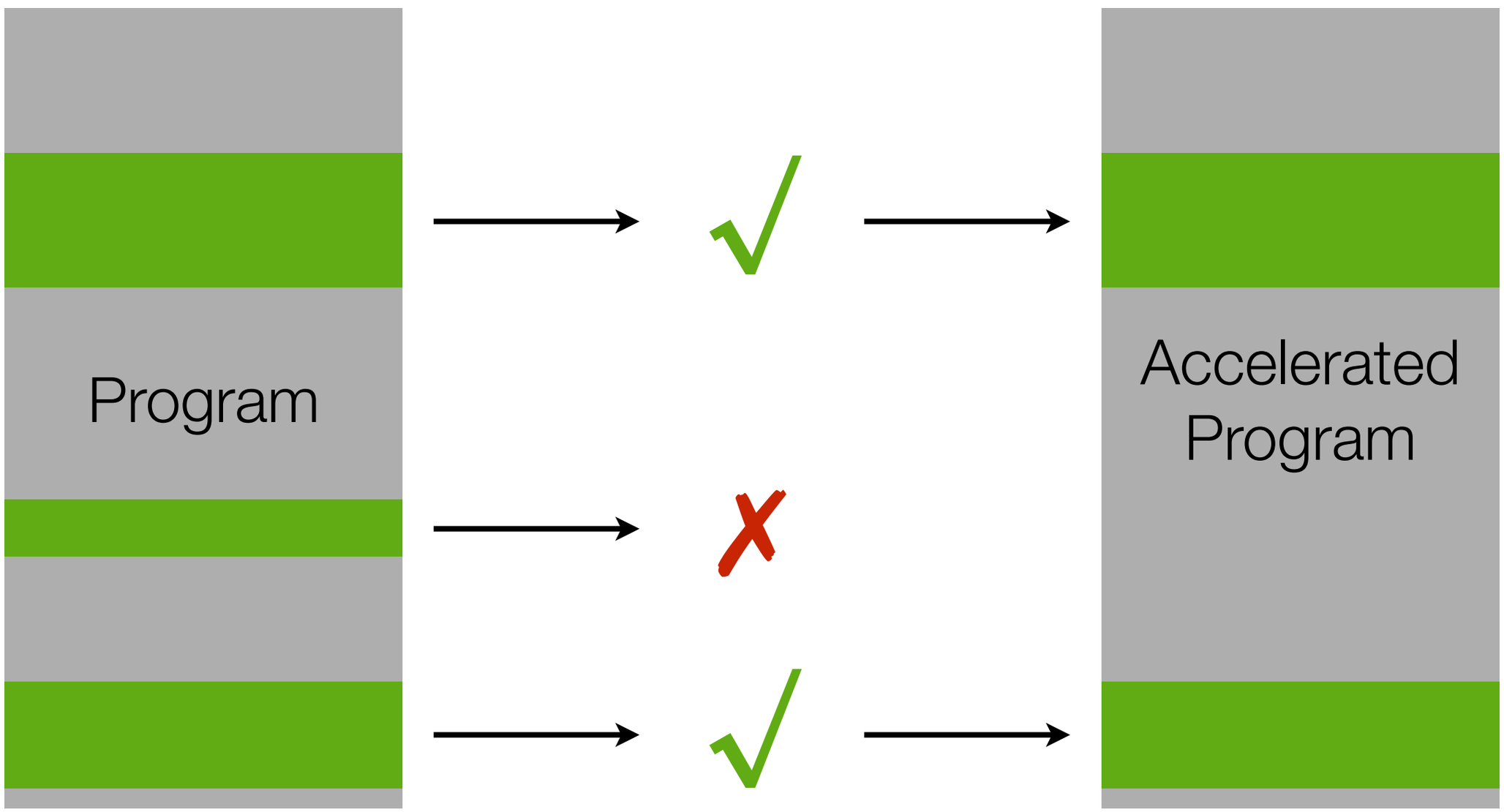
```
[[transform]]
float grad(float[3][3] p) {
  …
}

void edgeDetection(Image &src,
                   Image &dst) {
  for (int y = …) {
    for (int x = …) {
      dst[x][y] =
            grad(window(src, x, y));
    }
  }
}
```

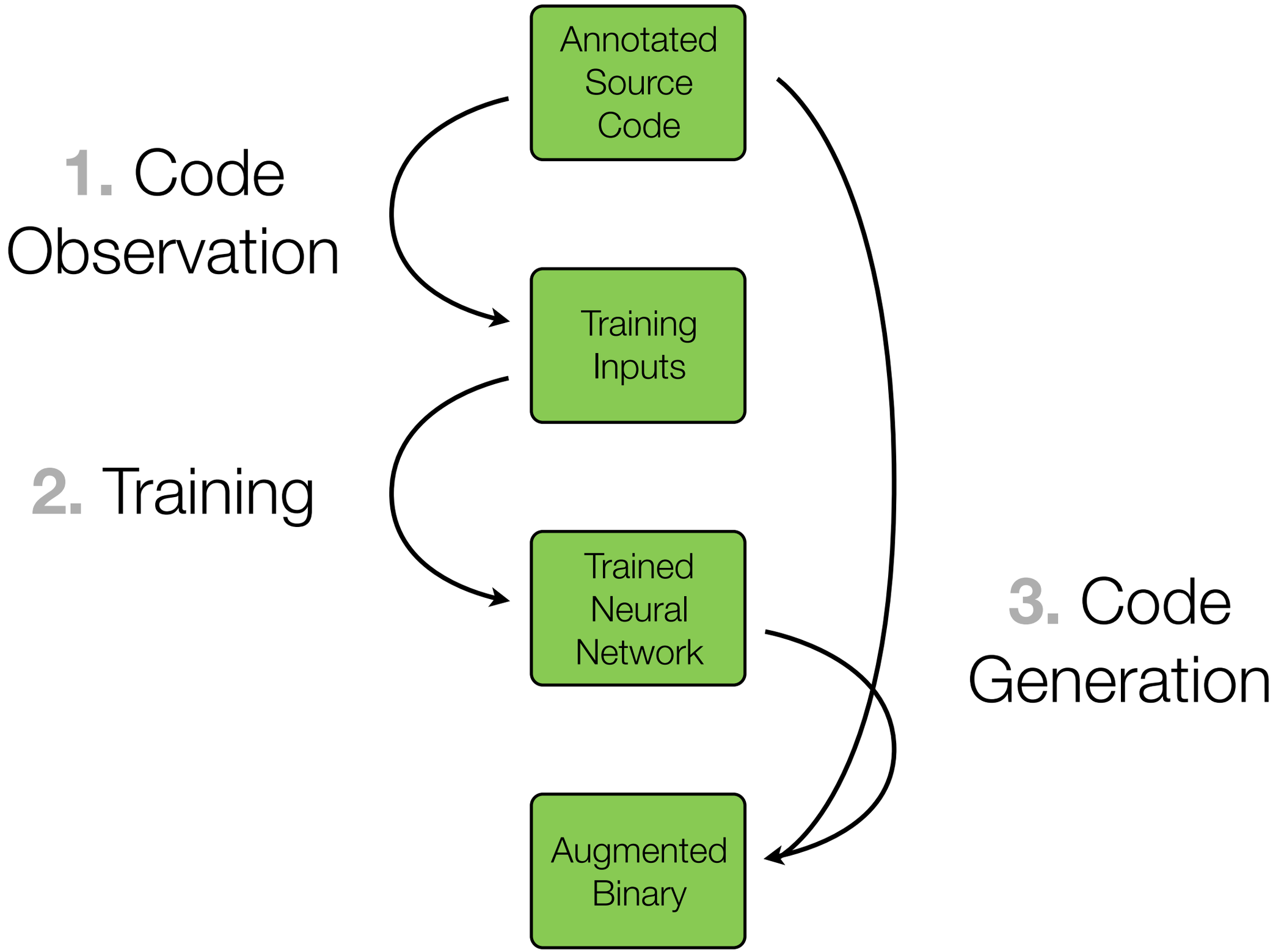
Code region criteria

	`grad()`
✓ Hot code	run on every 3x3 pixel window
✓ Approximable	small errors do not corrupt output
✓ Well-defined inputs and outputs	takes 9 pixel values; returns a scalar

Empirically selecting target functions



Compiling and transforming



Code observation

```
record(p);
```

```
record(result);
```

test cases

+

```
[[NPU]]
float grad(float[3][3] p) {
  …
}

void edgeDetection(Image &src,
                   Image &dst) {
  for (int y = …) {
    for (int x = …) {
      dst[x][y] =
           grad(window(src, x, y));
    }
  }
}
```

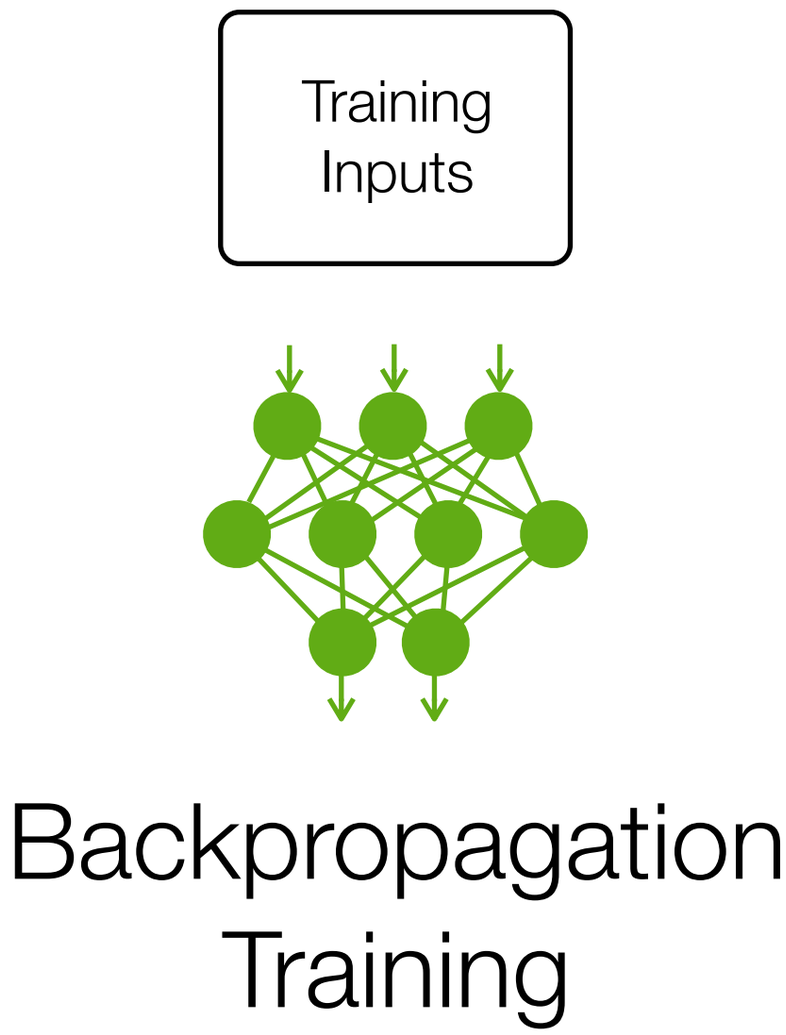
instrumented program

=

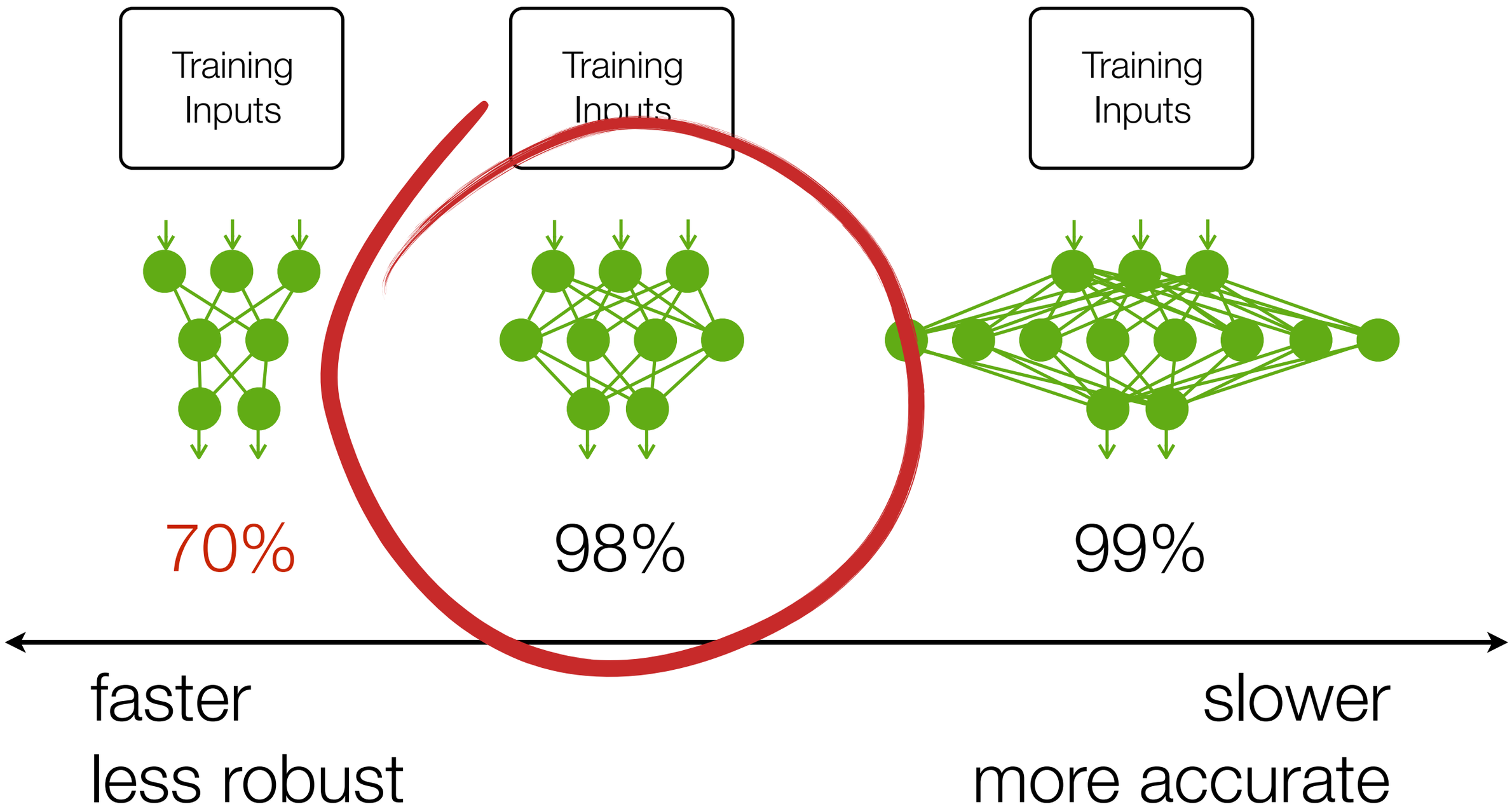
p		grad(p)
323, 231, 122, 93, 321, 49	→	53.2
49, 423, 293, 293, 23, 2	→	94.2
34, 129, 493, 49, 31, 11	→	1.2
21, 85, 47, 62, 21, 577	→	64.2
7, 55, 28, 96, 552, 921	→	18.1
5, 129, 493, 49, 31, 11	→	92.2
49, 423, 293, 293, 23, 2	→	6.5
34, 129, 72, 49, 5, 2	→	120
323, 231, 122, 93, 321, 49	→	53.2
6, 423, 293, 293, 23, 2	→	49.7

sample arguments & outputs

Training



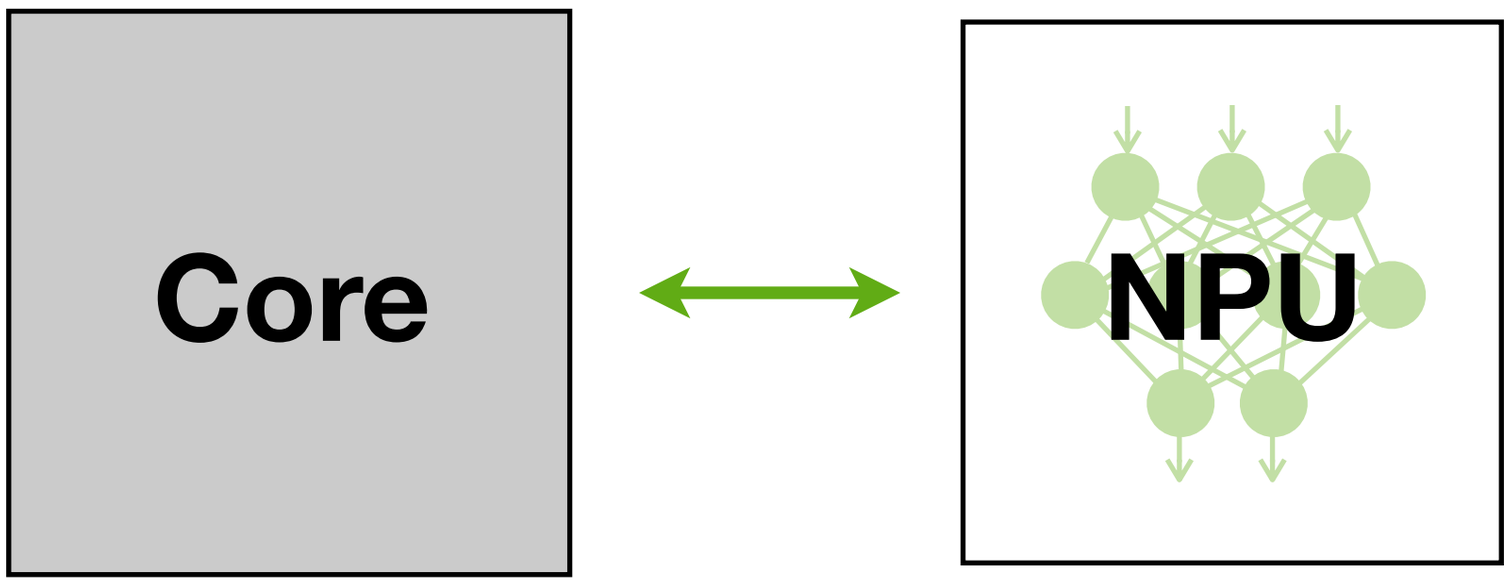
Training


Training Inputs
Training Inputs
Training Inputs
70%
98%
99%
faster
less robust
slower
more accurate

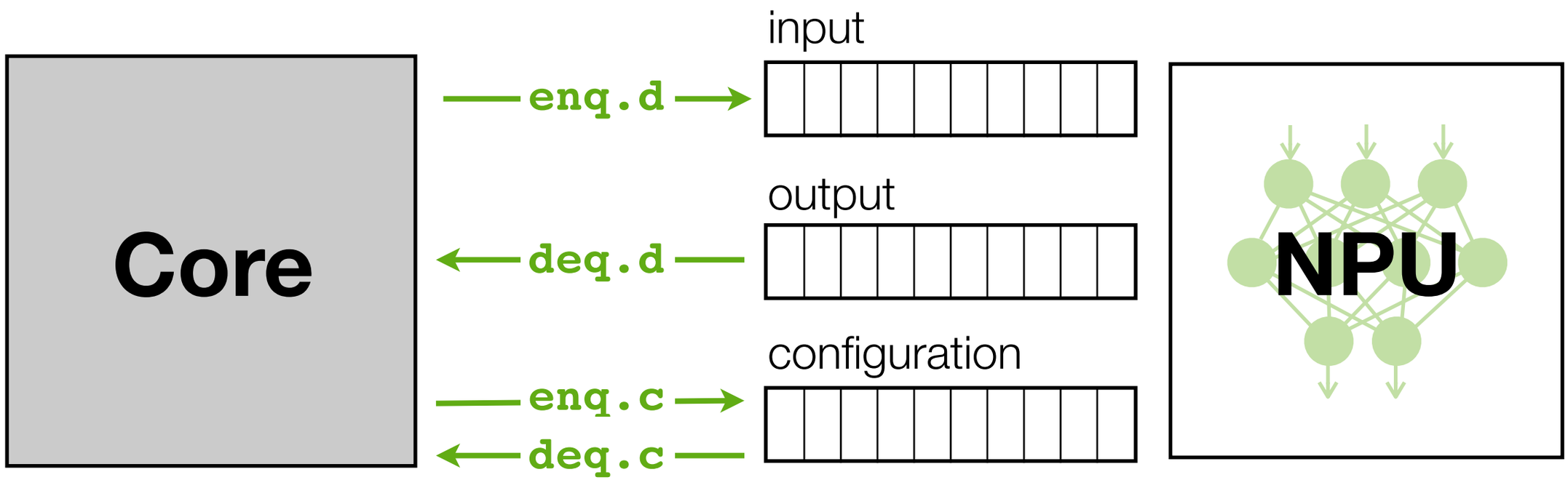
Code generation

```
void edgeDetection(Image &src,
                   Image &dst) {
  for (int y = …) {
    for (int x = …) {
      p = window(src, x, y);
      NPU_SEND(p[0][0]);
      NPU_SEND(p[0][1]);
      NPU_SEND(p[0][2]);
      …
      dst[x][y] = NPU_RECEIVE();
    }
  }
}
```

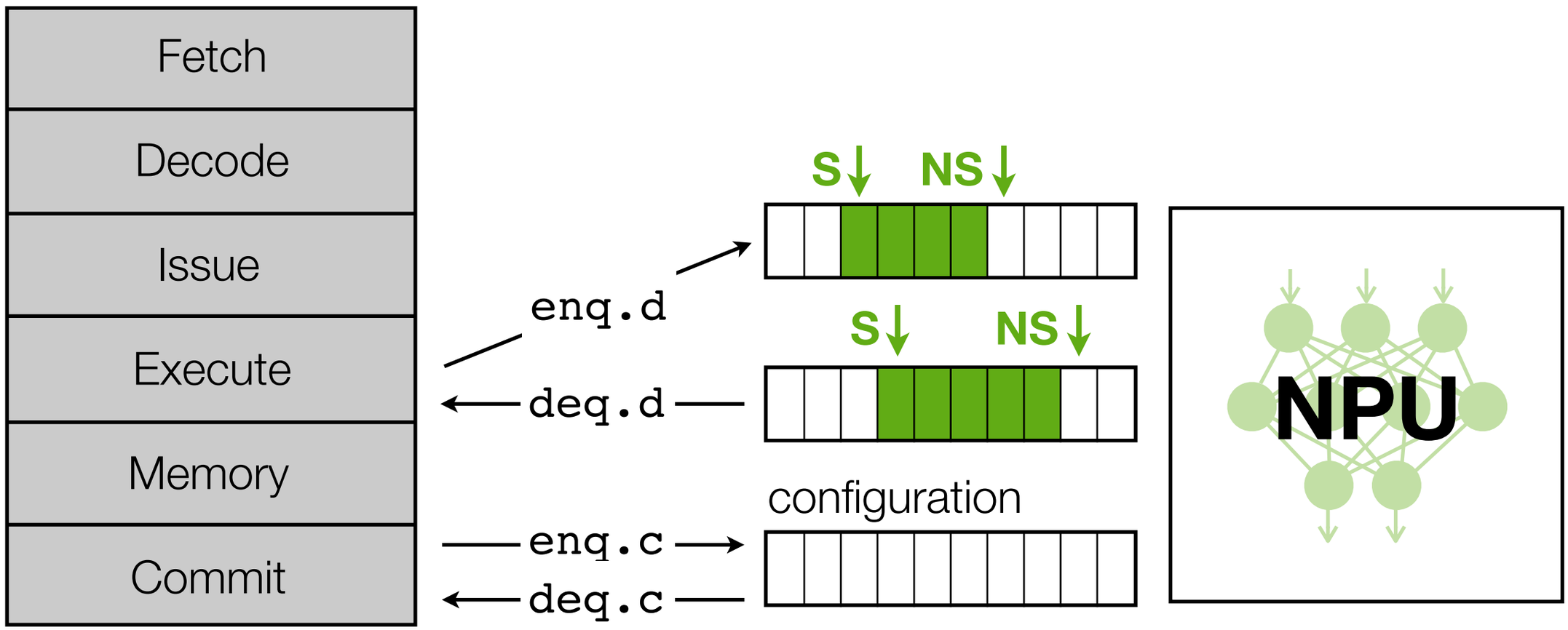
Neural Processing Unit (NPU)



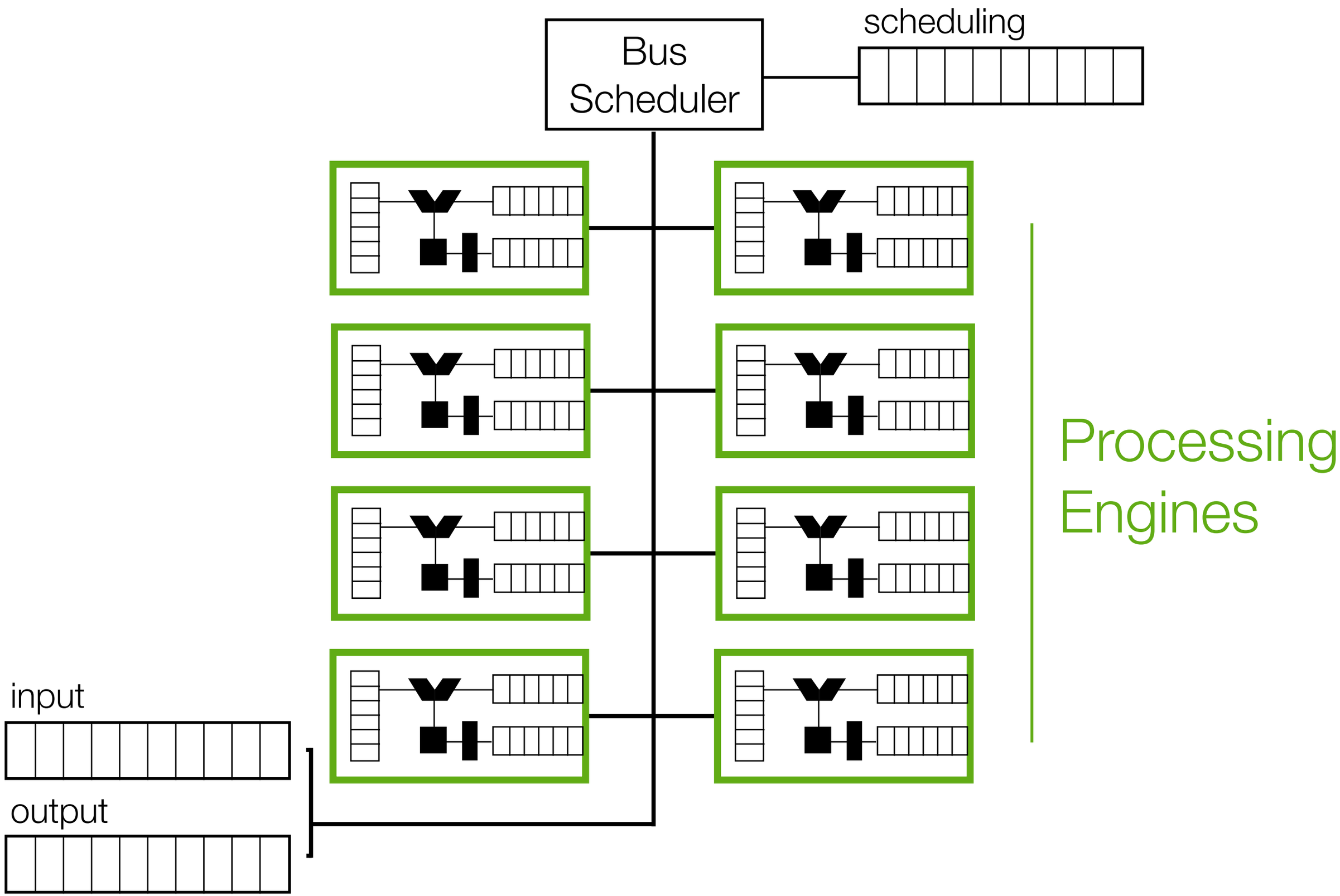
Software interface: ISA extensions



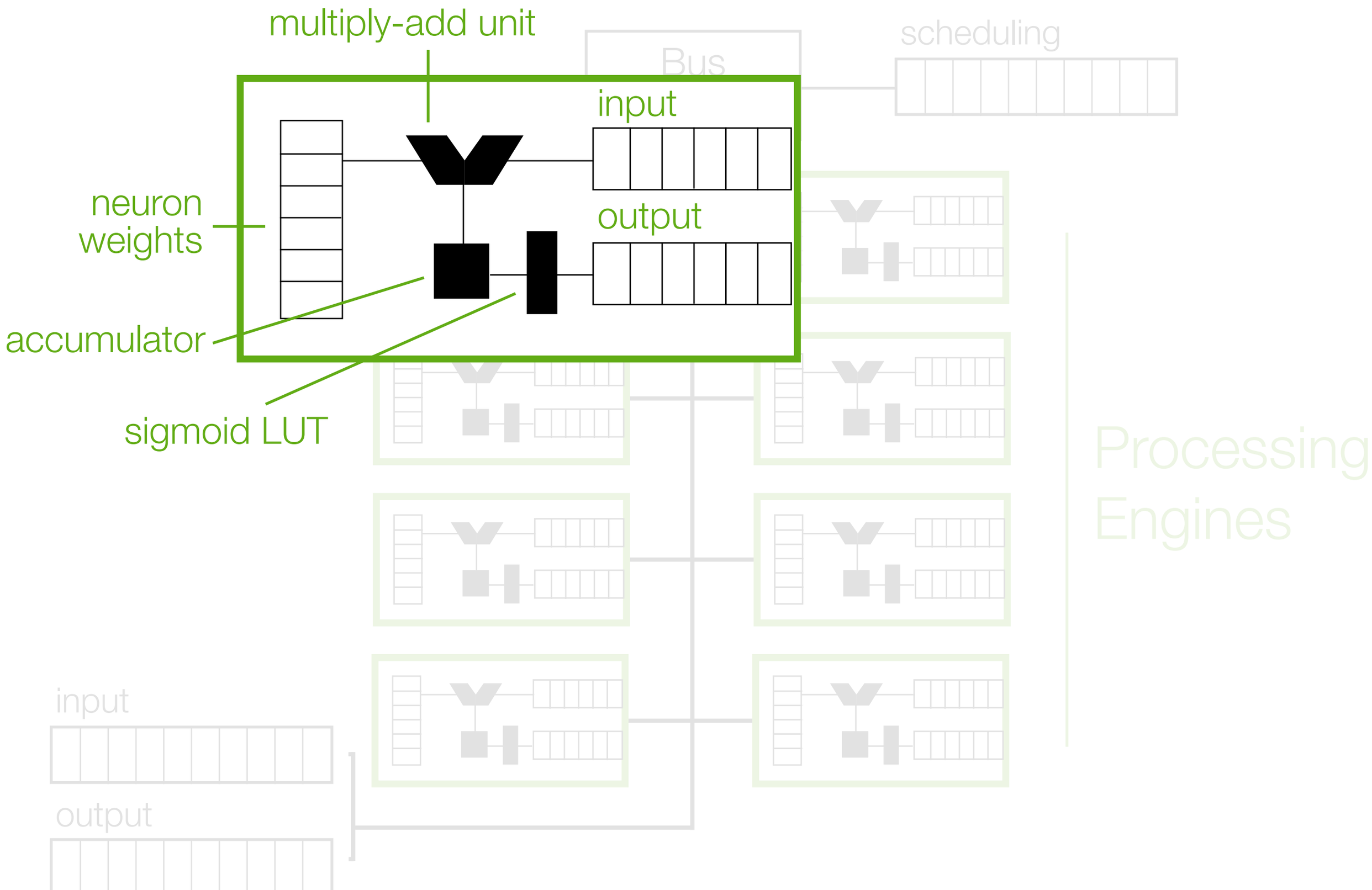
Microarchitectural interface



A digital NPU



A digital NPU



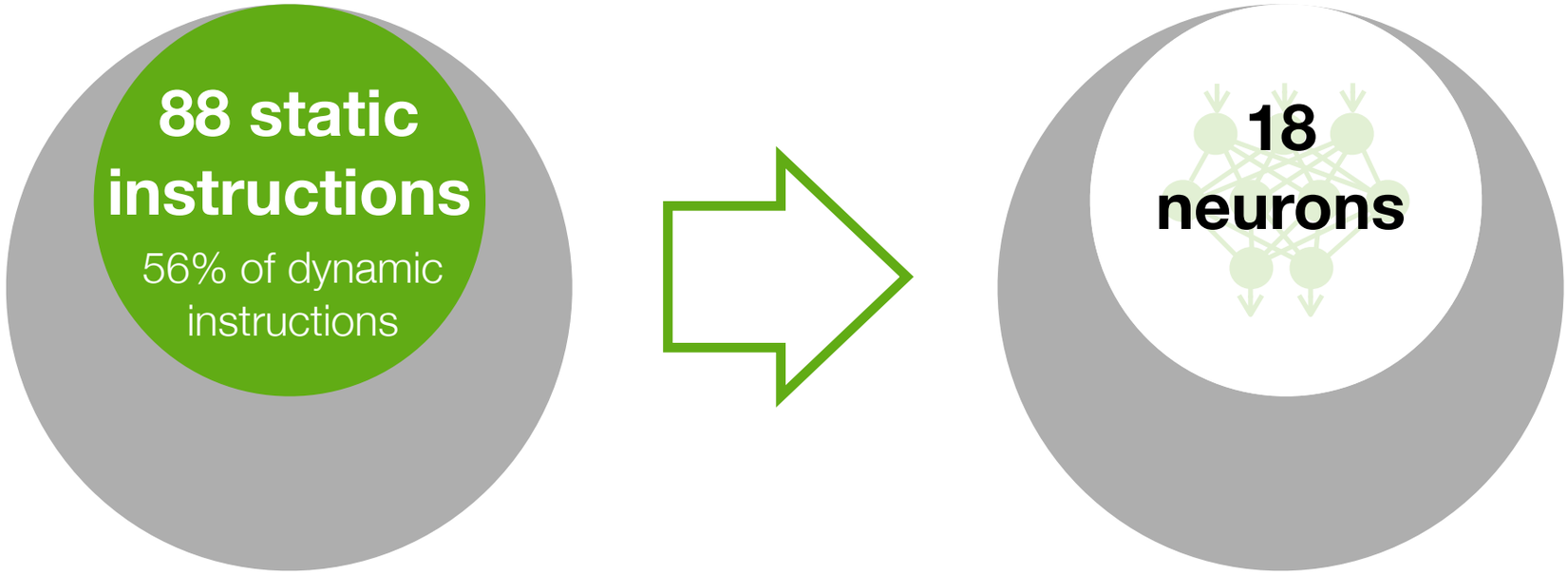
Experiments

Several benchmarks; annotated **one hot function** each
FFT, inverse kinematics, triangle intersection, JPEG, K-means, Sobel

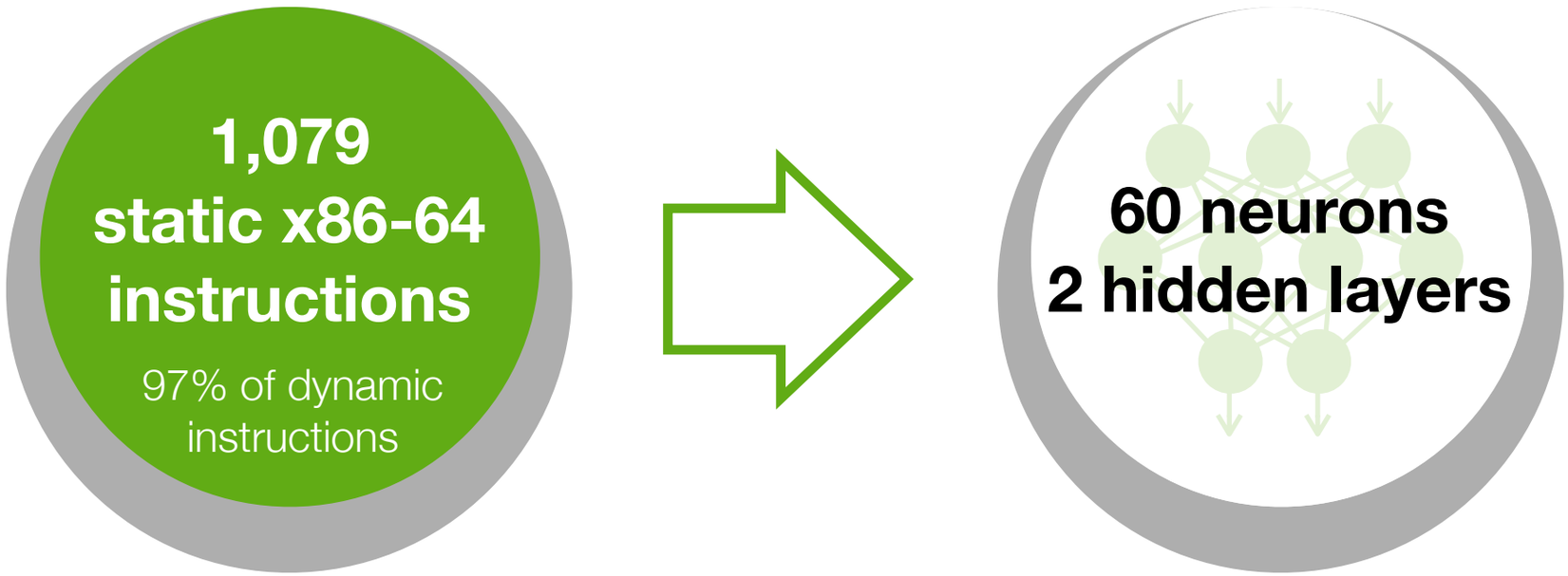
Simulated full programs on **MARSSx86**
Energy modeled with **McPAT** and **CACTI**
Microarchitecture like Intel Penryn: 4-wide, 6-issue
45 nm, 2080 MHz, 0.9 V

Two benchmarks

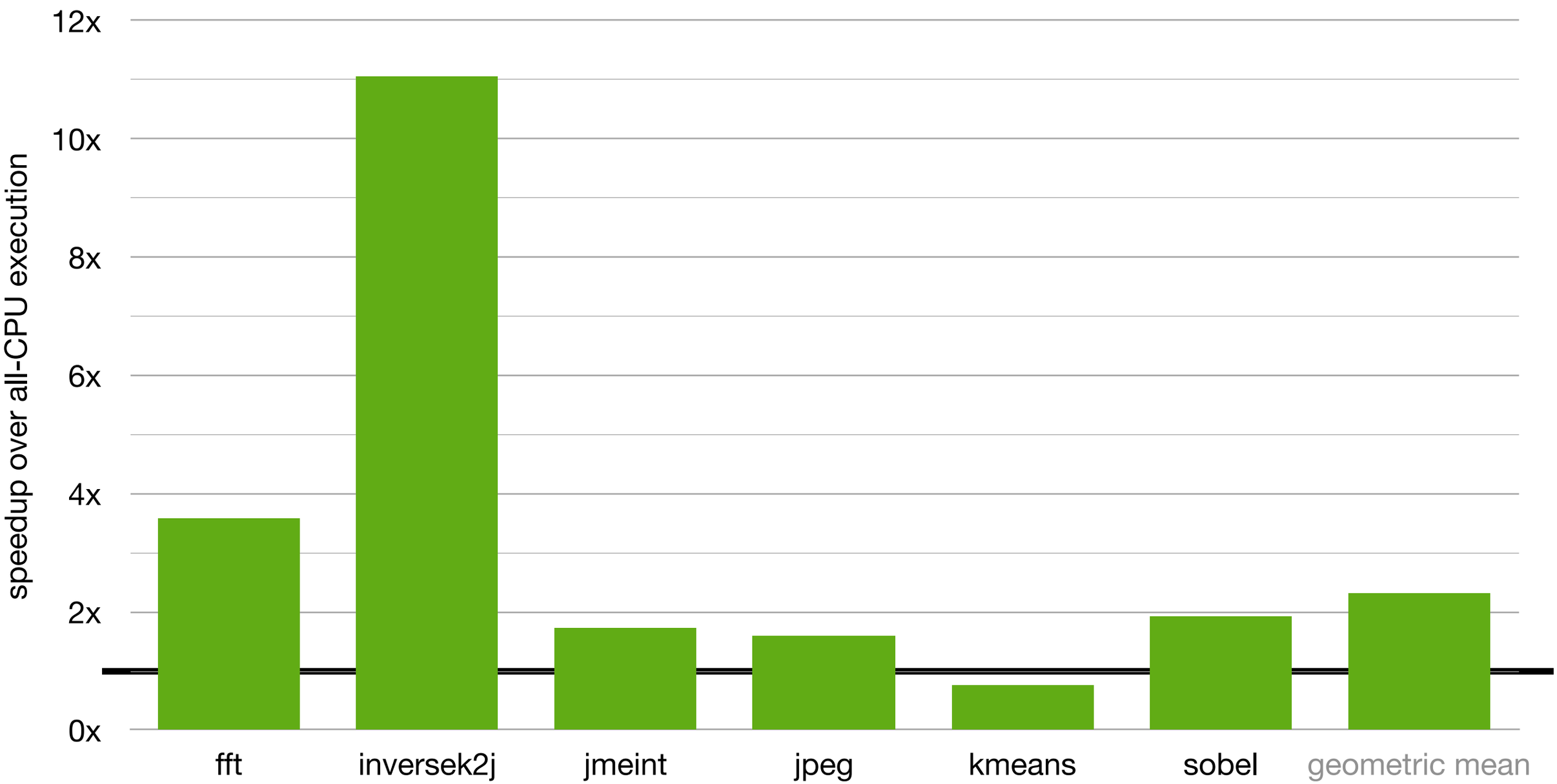
edge detection



triangle intersection



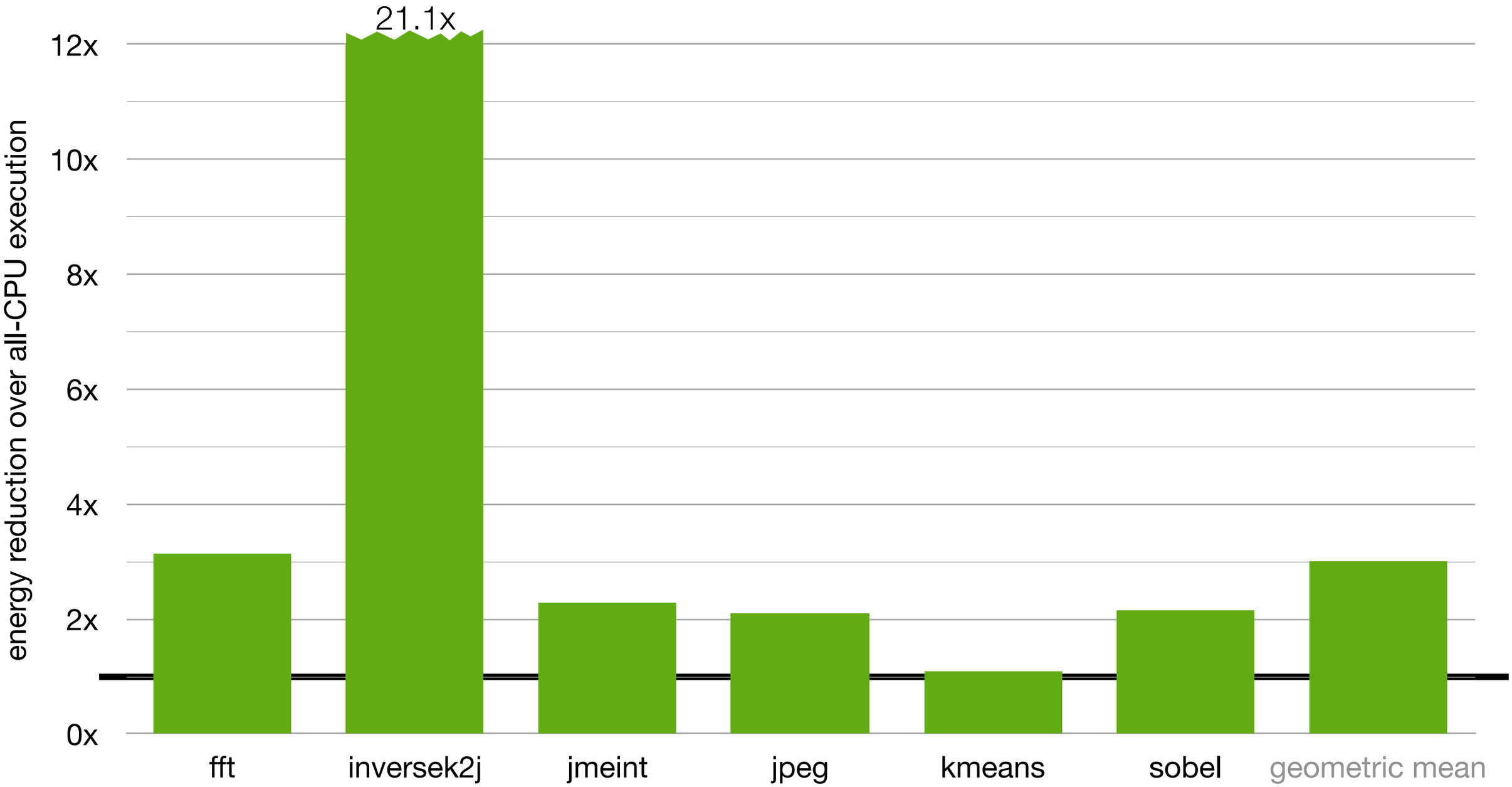
Speedup with NPU acceleration



2.3x average speedup

Ranges from 0.8x to 11.1x

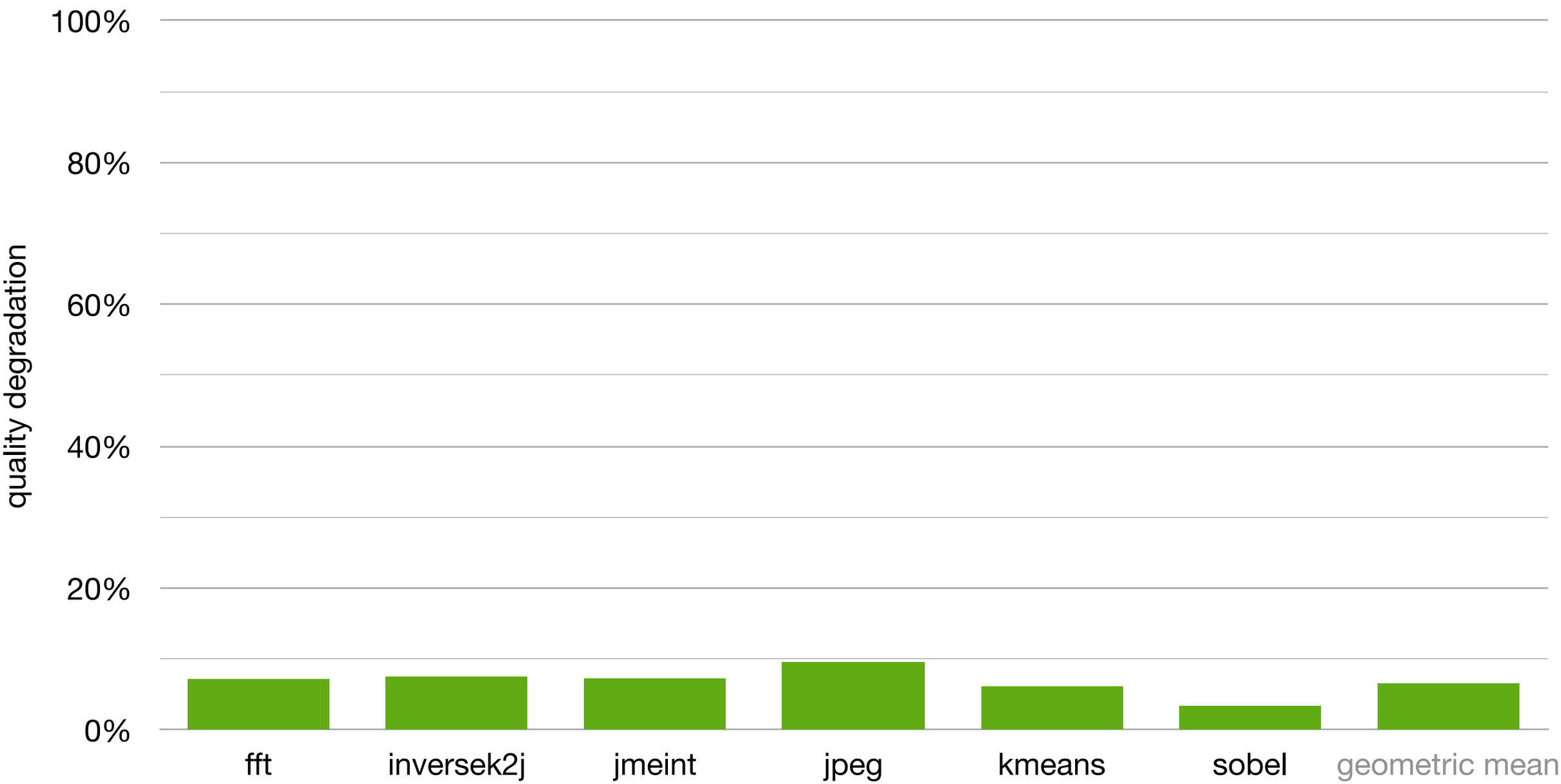
Energy savings with NPU acceleration



3.0x average energy reduction

All benchmarks benefit

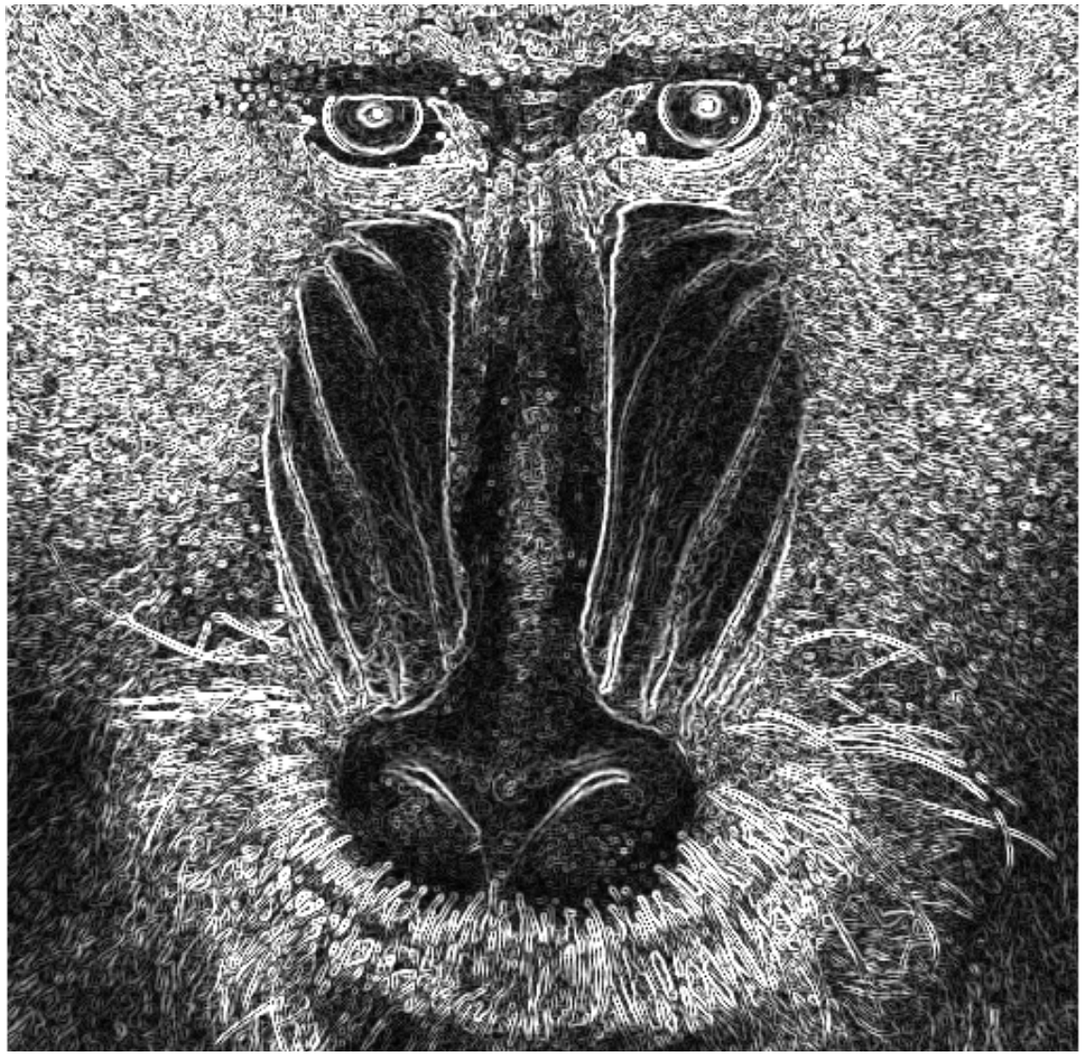
Application quality loss

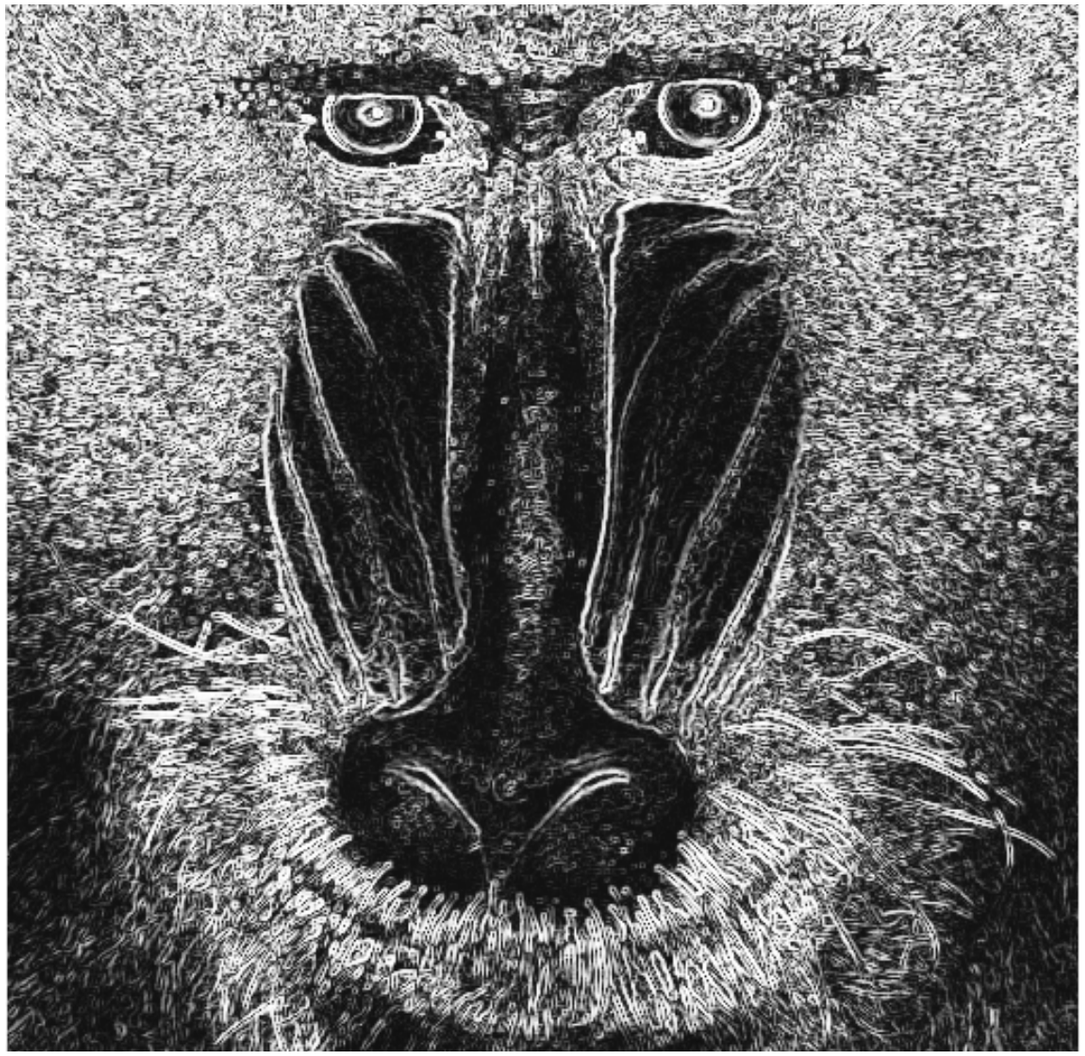


Quality loss below 10% in all cases

Based on application-specific quality metrics

Edge detection with gradient calculation on NPU





Also in the paper

Sensitivity to communication latency

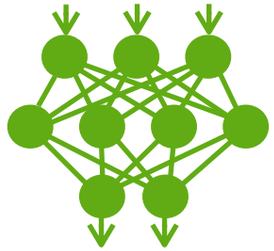
Sensitivity to NN evaluation efficiency

Sensitivity to PE count

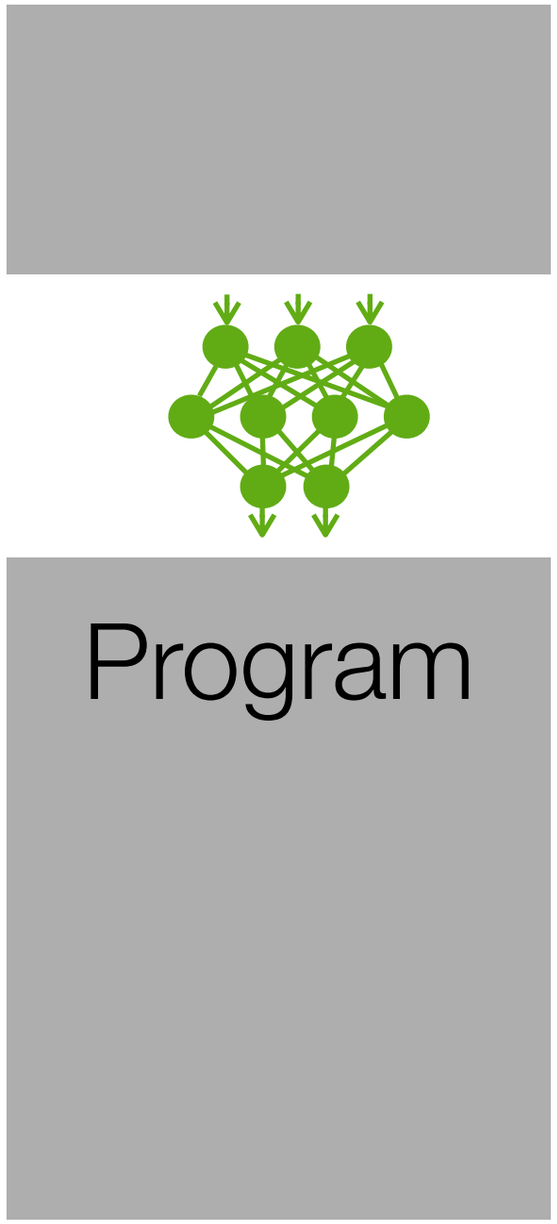
Benchmark statistics

All-software NN slowdown

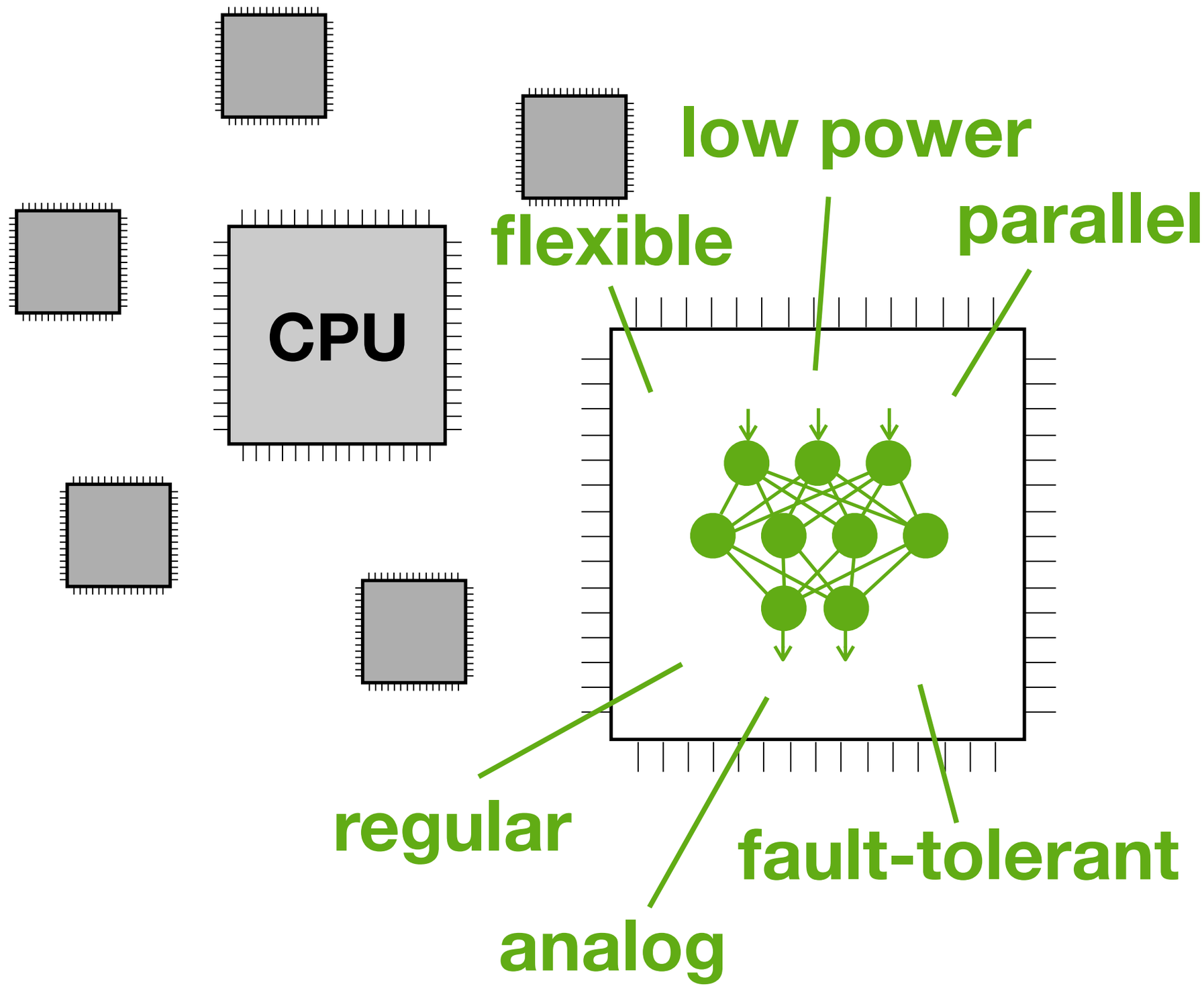
Program



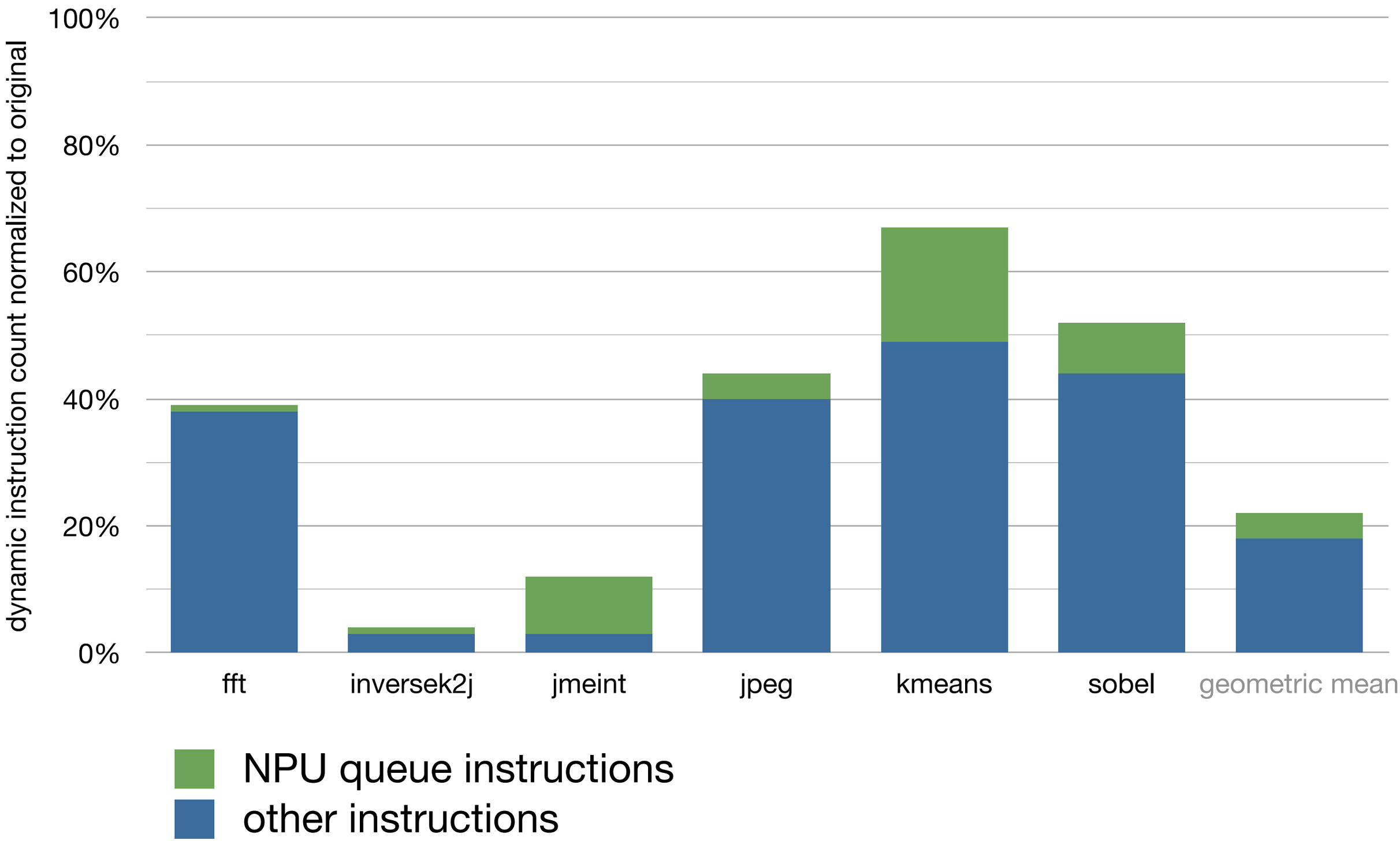
Program



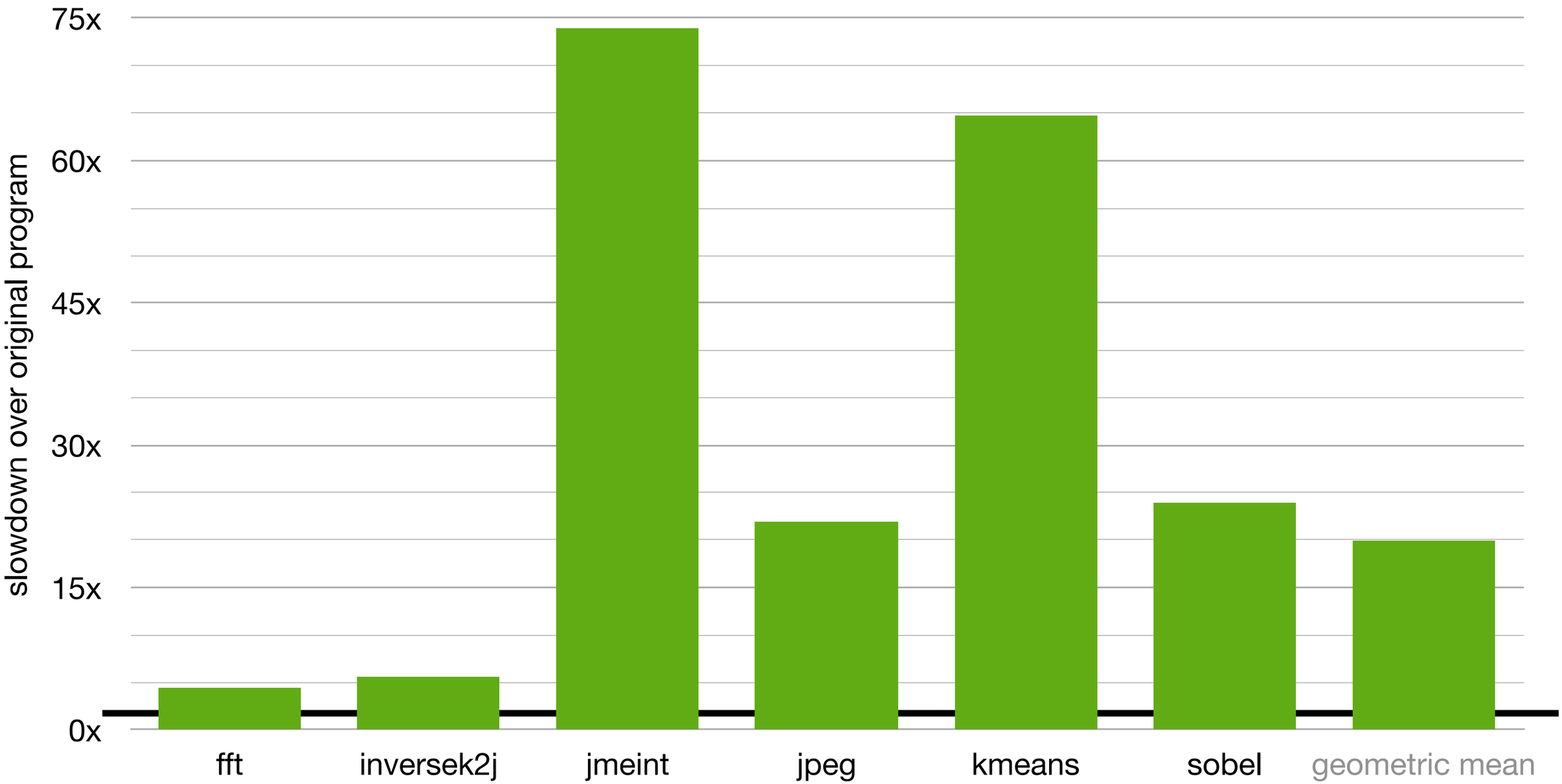
Neural networks can efficiently approximate functions from programs written in conventional languages.


CPU
low power
flexible
parallel
regular
fault-tolerant
analog

Normalized dynamic instructions



Slowdown with software NN



20x average slowdown

Using off-the-shelf FANN library