Training Neural Networks on Embedded Devices

Targeting Embedded Environments

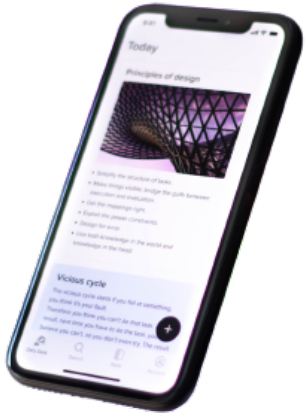
Prasanth Shaji, Deepak Venkataram

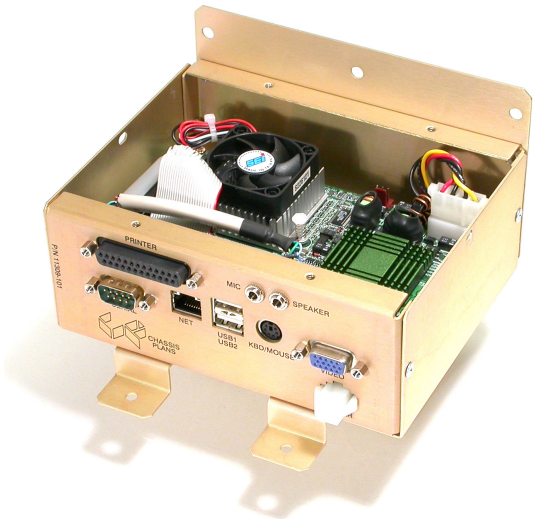
Master's Thesis

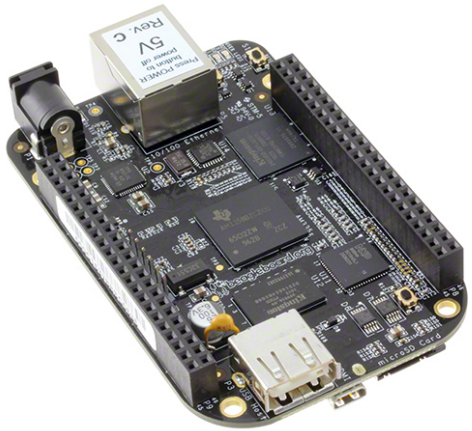
Uppsala University

Outline

Embedded Devices



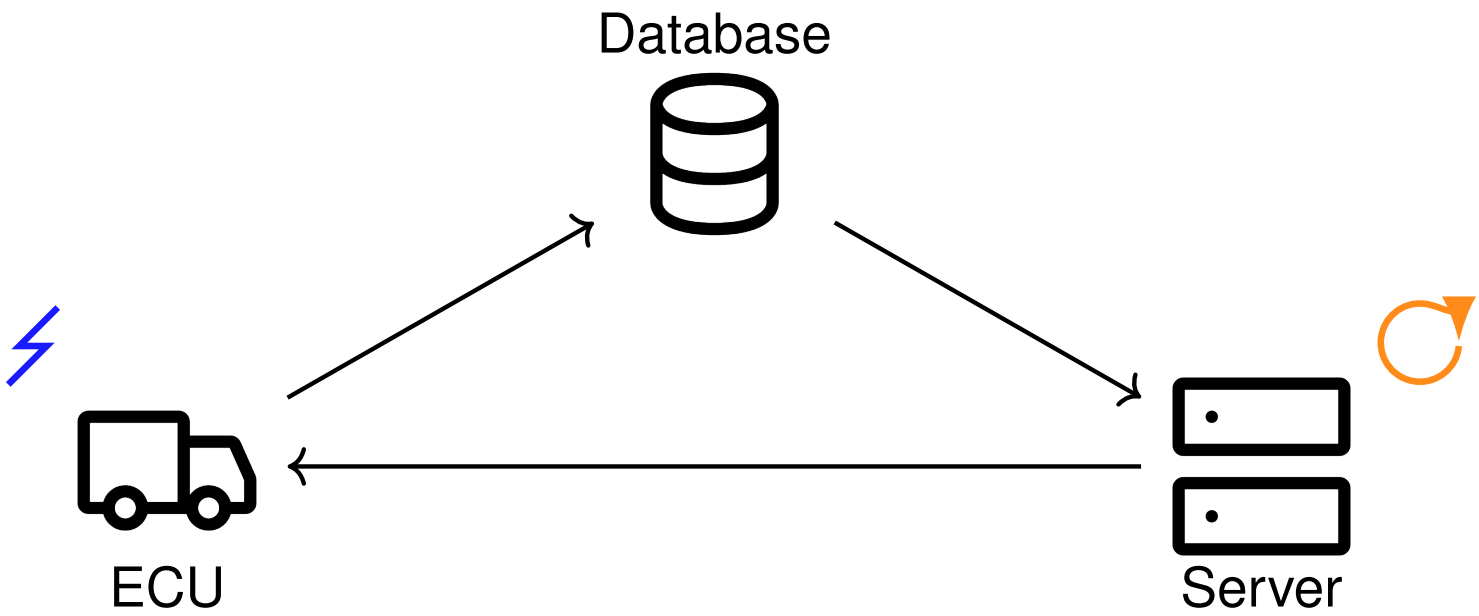




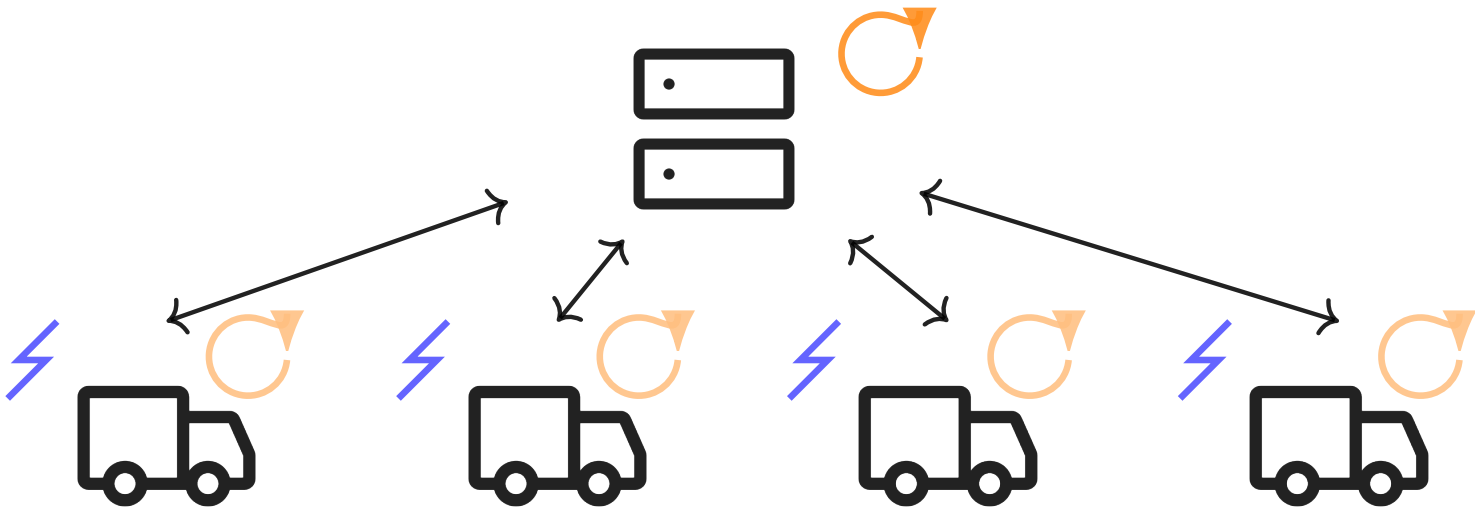
Neural Network Applications on Embedded Devices

Goal : Neural Network Training on Scania ECU

Traditional Paradigm



Federated Learning



UPPSALA UNIVERSITET

Goal : Neural Network Training on Scania ECU

Predictive vehicle maintenance using data from a fleet of connected trucks

Why Training on the ECU?

- Training on devices reduces network usage, ensures data privacy, etc.
- Repurpose existing ECUs to perform neural network related tasks

Scope *Repurpose Scania ECU, train neural network model, assess training performance*

Goal : Neural Network Training on Scania ECU

Predictive vehicle maintenance using data from a fleet of connected trucks

Why Training on the ECU?

- Training on devices reduces network usage, ensures data privacy, etc.
- Repurpose existing ECUs to perform neural network related tasks

Scope *Repurpose Scania ECU, train neural network model, assess training performance*

Goal : Neural Network Training on Scania ECU

Predictive vehicle maintenance using data from a fleet of connected trucks

Why Training on the ECU?

- Training on devices reduces network usage, ensures data privacy, etc.
- Repurpose existing ECUs to perform neural network related tasks

Scope *Repurpose Scania ECU, train neural network model, assess training performance*

Goal : Neural Network Training on Scania ECU

Predictive vehicle maintenance using data from a fleet of connected trucks

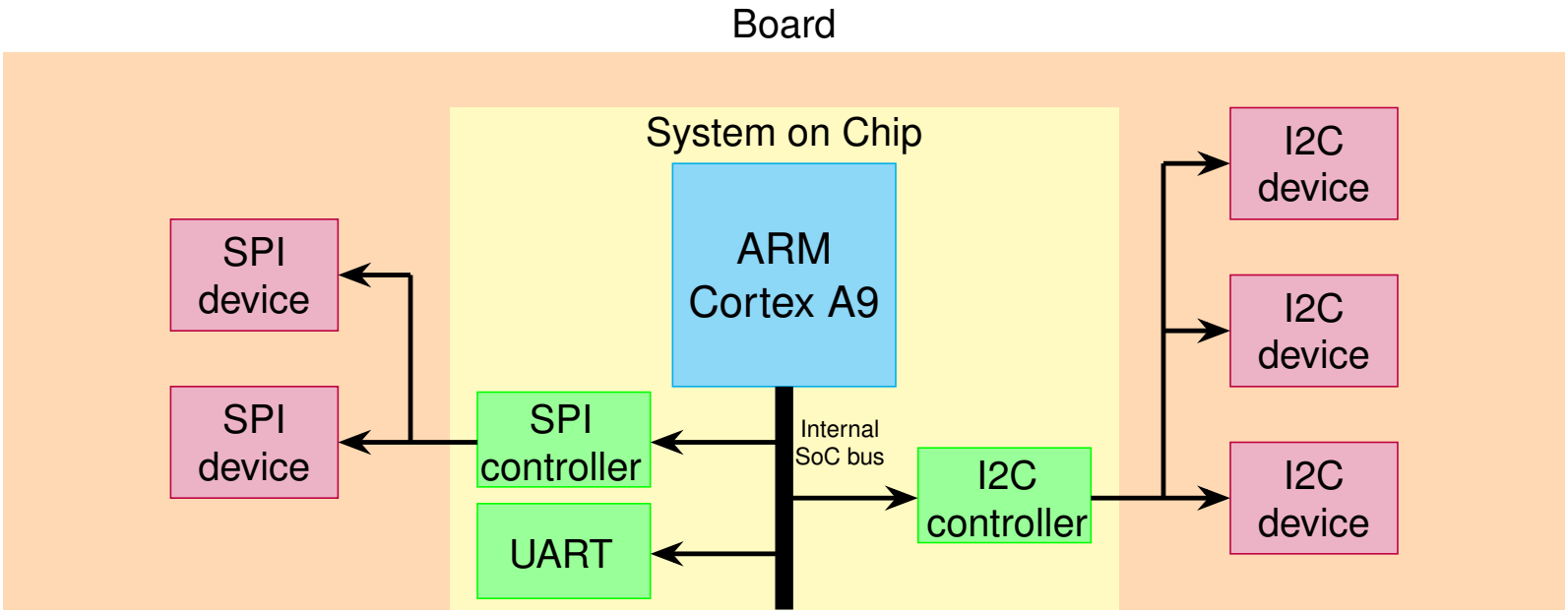
Why Training on the ECU?

- Training on devices reduces network usage, ensures data privacy, etc.
- Repurpose existing ECUs to perform neural network related tasks

Scope *Repurpose Scania ECU, train neural network model, assess training performance*

Embedded Devices : Hardware Outline

ARM | Silicon Vendor | System Maker



ARM Boards : Examples

SoC	Application Domain	Architecture	Processor Core
nRF51822	Ultra low power, BLE	ARM v6-M	ARM Cortex M0
AM2732	Automotive	ARM v7-R	ARM Cortex R5
AM3358	Industrial / IoT	ARM v7-A	ARM Cortex A8
i.MX6S	Multimedia applications	ARM v7-A	ARM Cortex A9
BCM2711	Educational / IoT	ARM v8-A	ARM Cortex-A72
Snapdragon 460	Smartphones	ARM v8-A	ARM Cortex-A73, A53
Google Tensor	Pixel 6	ARM v8.2-A	ARM Cortex-X1, A76, A55
Apple A16 Bionic	iPhone 14 Pro	ARM v8.6-A	APL1W10
Apple M1	Macbooks / Mac Mini / iPads	ARM v8.5-A	APL1102

Embedded Linux

Operating System for Embedded Devices
- Customized for Specific Hardware

Kernel, Bootloaders, Device Trees
Cross Compiling Toolchains, Root Filesystems

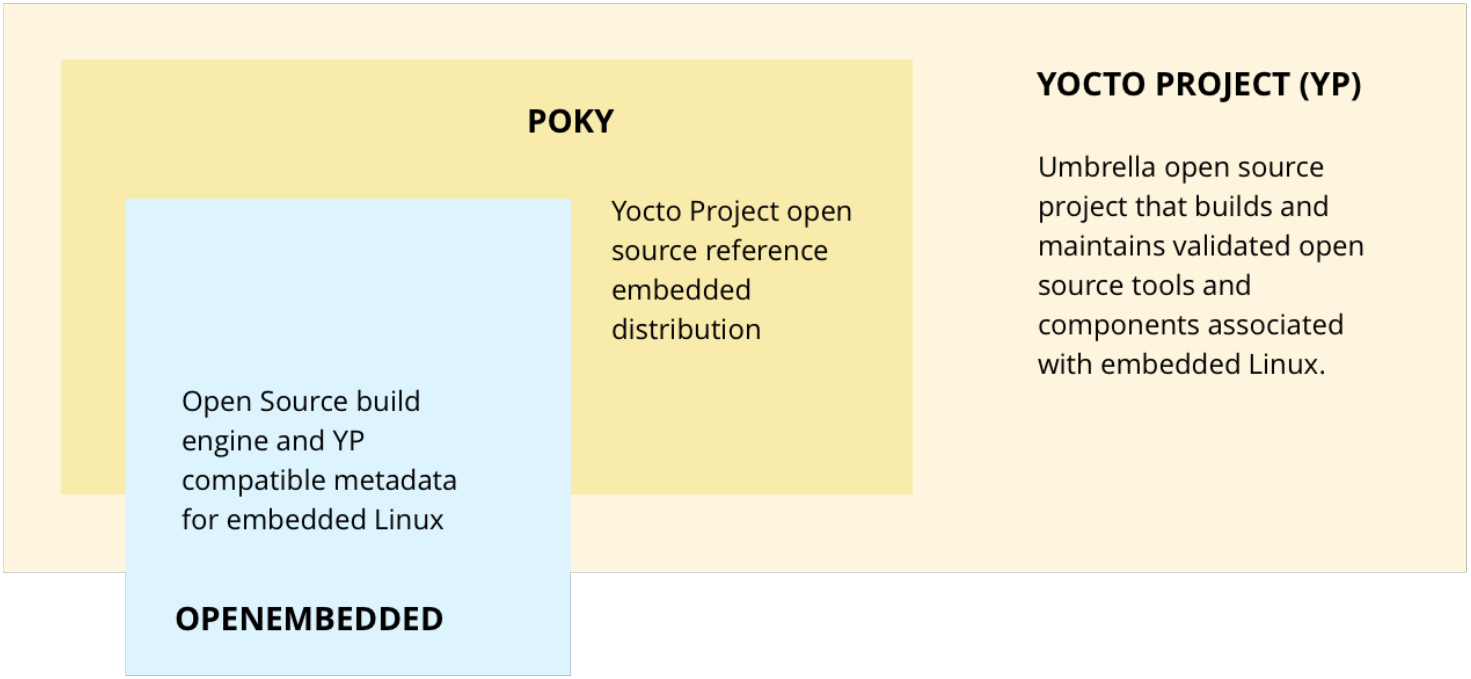
Embedded Linux

Operating System for Embedded Devices
- Customized for Specific Hardware

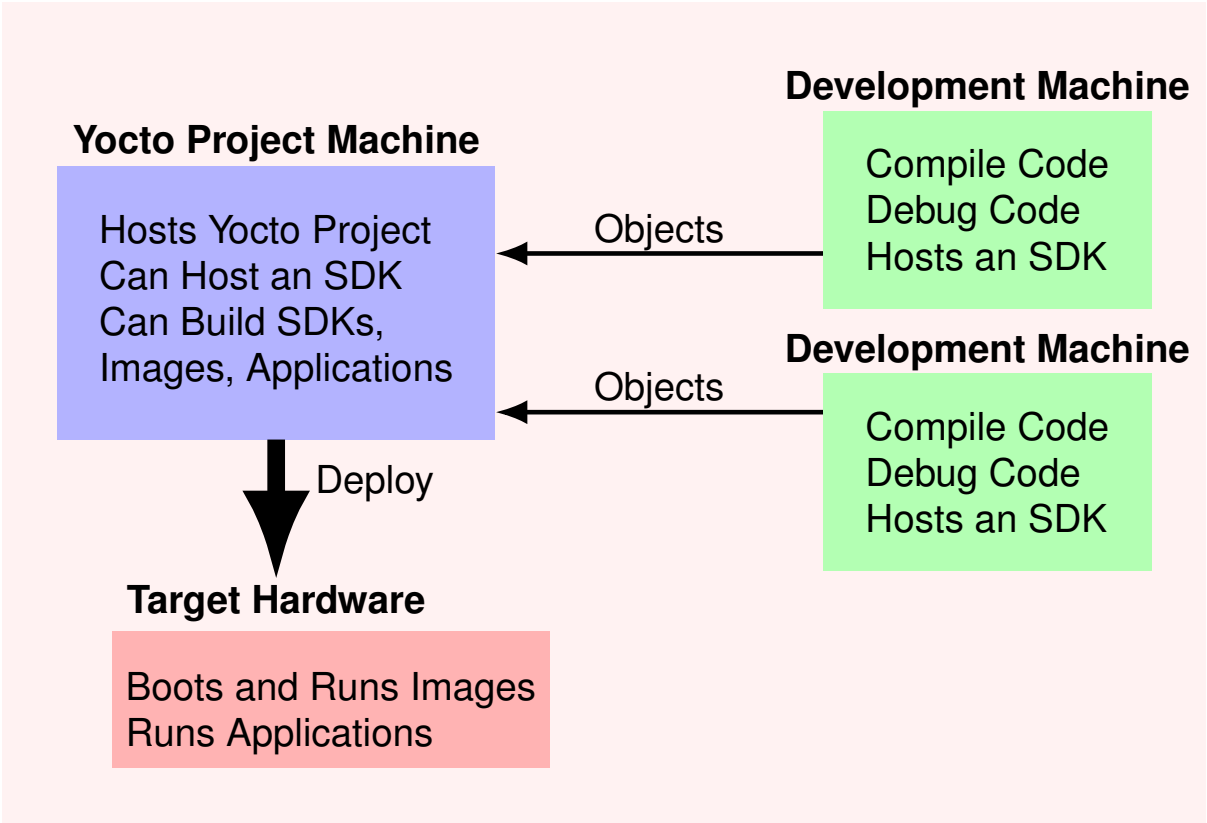
Kernel, Bootloaders, Device Trees
Cross Compiling Toolchains, Root Filesystems

Build Systems

Yocto Project

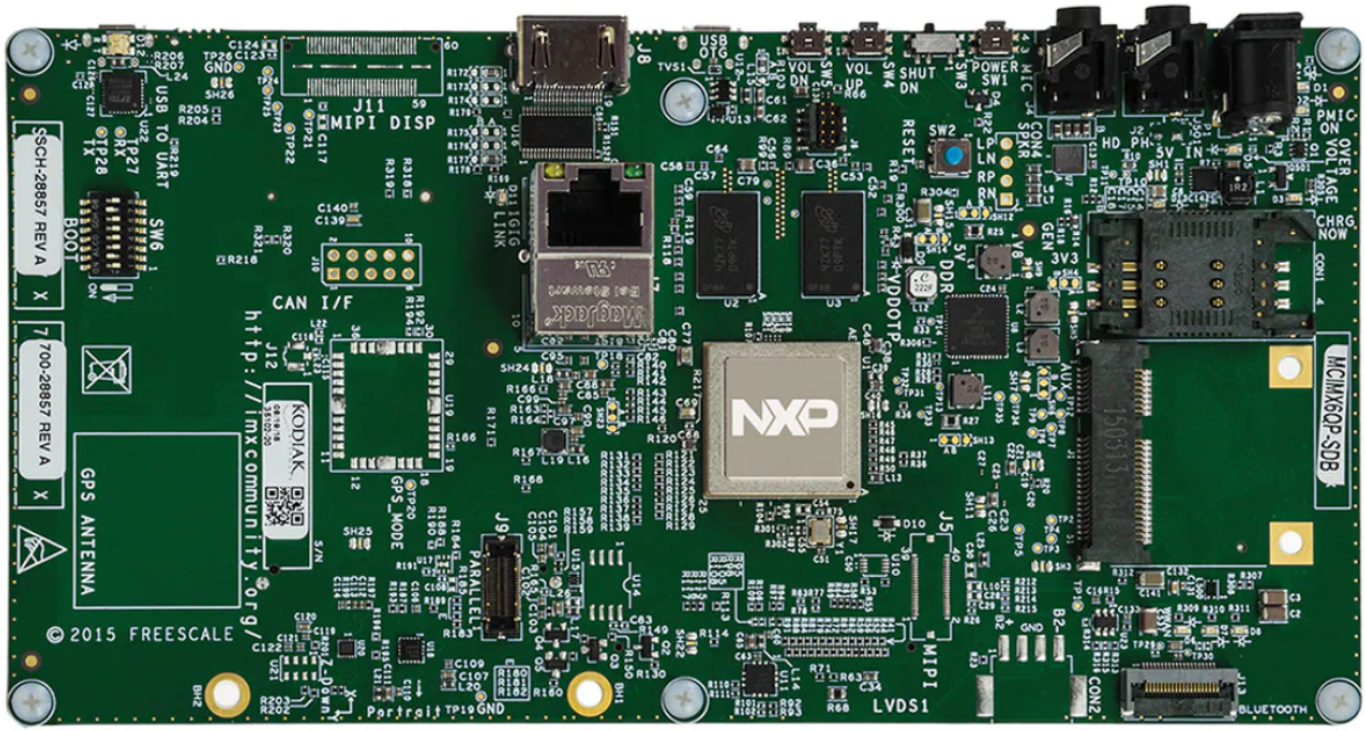


Development Setup

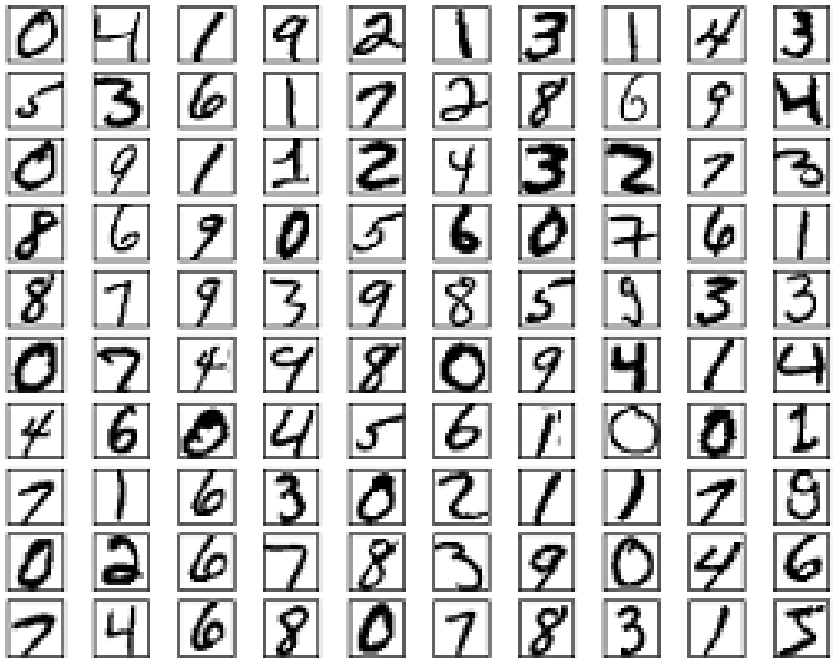


Software Versions

Scania ECU to MCIMX6Q-SDB

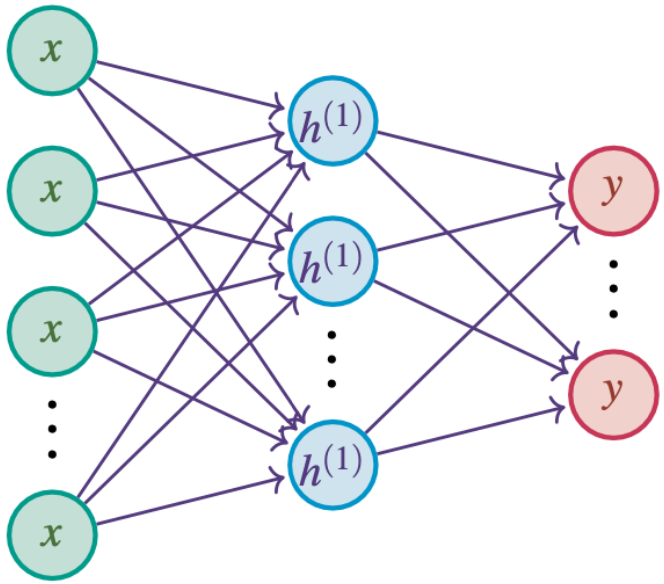


Handwritten Digit Recognition Neural Network



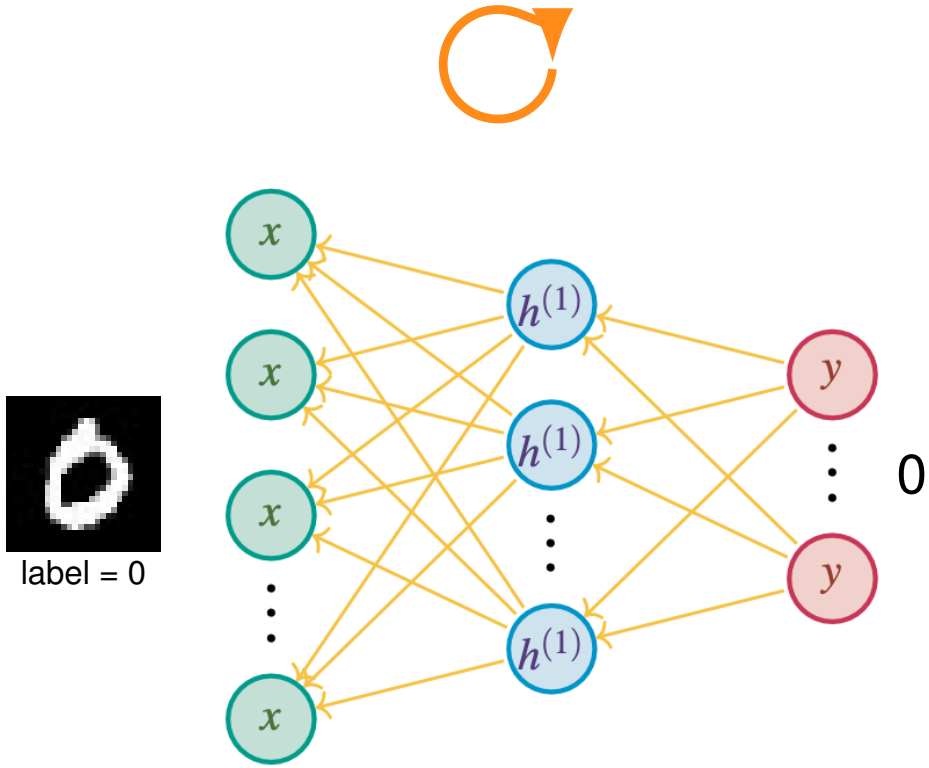
MNIST

Neural Network



$$y_k = \sigma\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

Neural Network Training



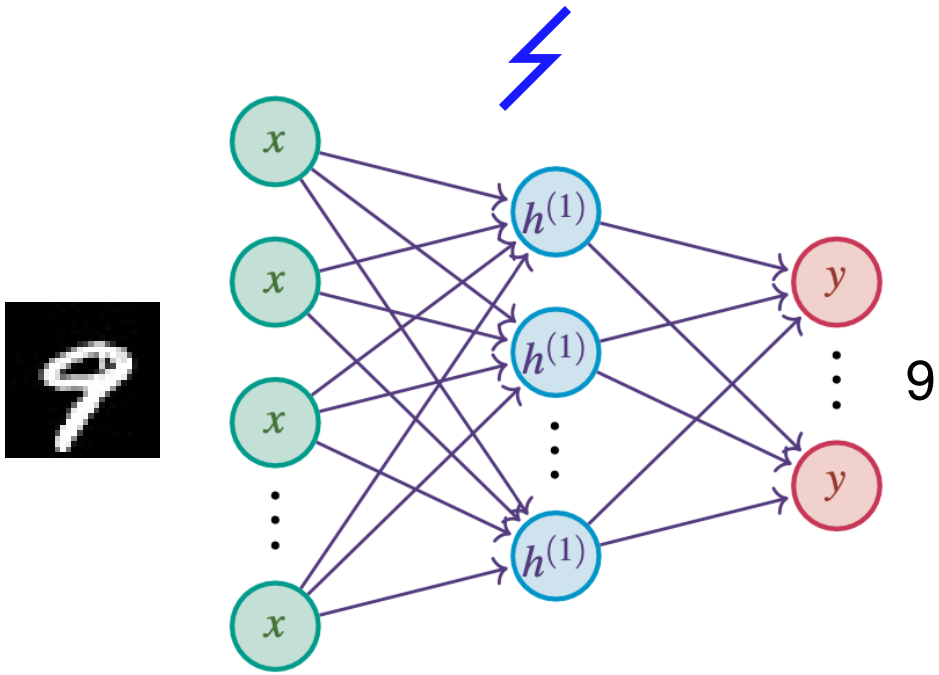
Learning Algorithm

Algorithm 1 Mini Batch Gradient Descent with learning rate η and the Mean Squared Error (MSE) cost function

Require: initial weights $w^{(0)}$, number of epochs E, batch size B, training data with T entries
Ensure: final weights $w^{(E*T)}$

```
for e = 0 → E − 1 do
    for b = 0 → T/B do
        for t = b ∗ B → (b + 1) ∗ B do
            estimate ∇ℒ(w^(t))                    ▷ ℒ here is MSE
            compute Δw^(b) += −∇ℒ(w^(t))
        end for
        w^(e+1) := w^(e) + η Δw^(e)
    end for
end for
return w^(T)
```

Neural Network Inference



HDR-NN Implementations

github.com/paperbound/hdrnn-arm

- `C` based HDR-NN
- `C++` / `Eigen`
- `C++` / `libtorch` (PyTorch)
- `Python` / `Numpy`

C based HDR-NN

```c
/* HDR Neural Network */
typedef struct
{
    float bias;
    float *weights;
    float *nabla_w;
} Neuron;

typedef struct LayerT
{
    int size;
    int incidents;
    Neuron *neurons;
    float *activations;
    float *z_values;
    float *nabla_b;
    struct LayerT *next;
    struct LayerT *previous;
} Layer; // Network layers except for input

typedef struct
{
    Layer *layers;
    int depth;
} Network; // HDRNN
```

```c
static void feed_forward(Network *network, float *image)
{
    // First layer is the image
    float *activations = image;

    Layer *layer = network->layers;
    while (layer != NULL)
    {
        for (int i = 0; i < layer->size; i++)
        {
            Neuron *neuron = &layer->neurons[i];
            float zvalue = 0;
            for (int j = 0; j < layer->incidents; j++)
            {
                zvalue += neuron->weights[j] * activations[j];
            }
            layer->z_values[i] = zvalue + neuron->bias;
            layer->activations[i] = sigmoid(layer->z_values[i]);
        }
        activations = layer->activations;
        layer = layer->next;
    }
}
```

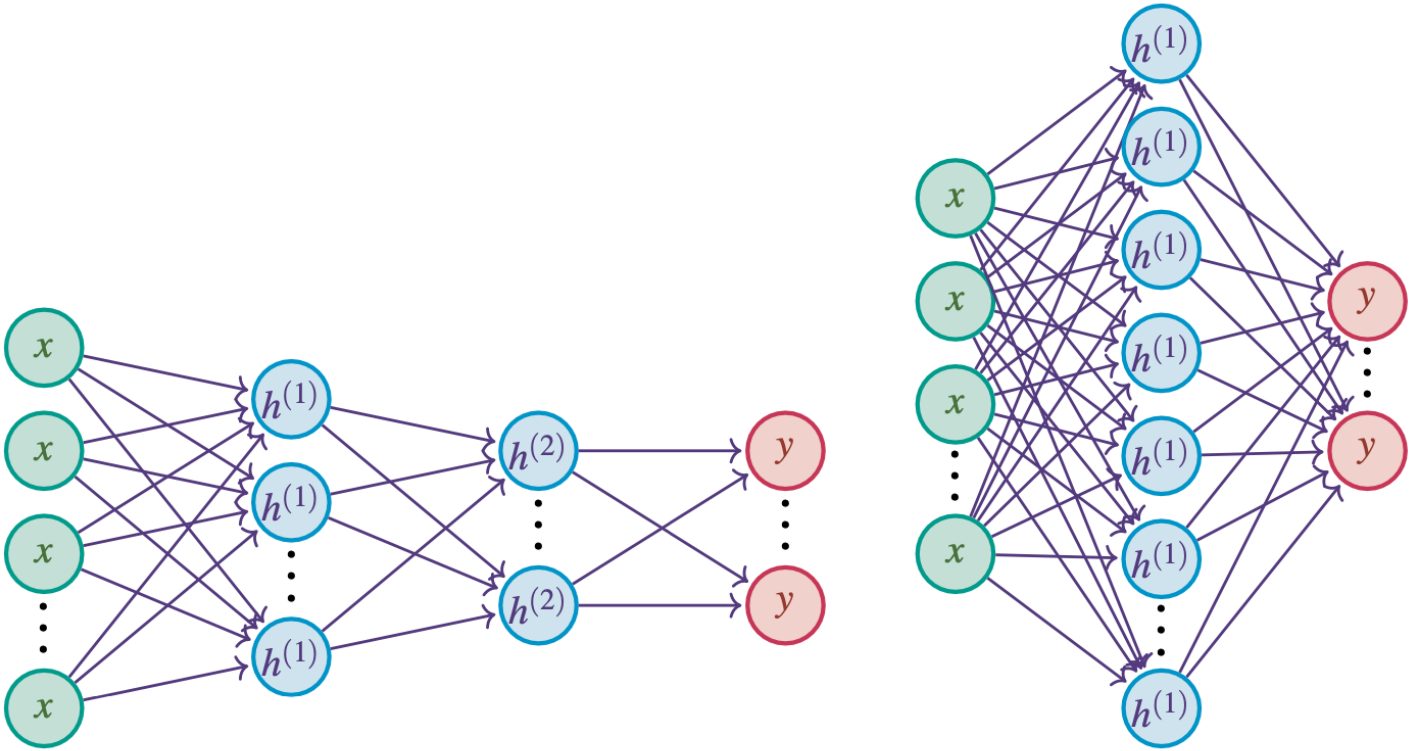
C++ / Eigen based HDR-NN

- `Eigen` is a `C++` template library for linear algebra
- Analogous to `Python` / `Numpy`

C++ / libtorch based HDR-NN

- PyTorch doesn't support MCIMX6Q
- Porting `libtorch` using QEMU

HDR-NN configurability



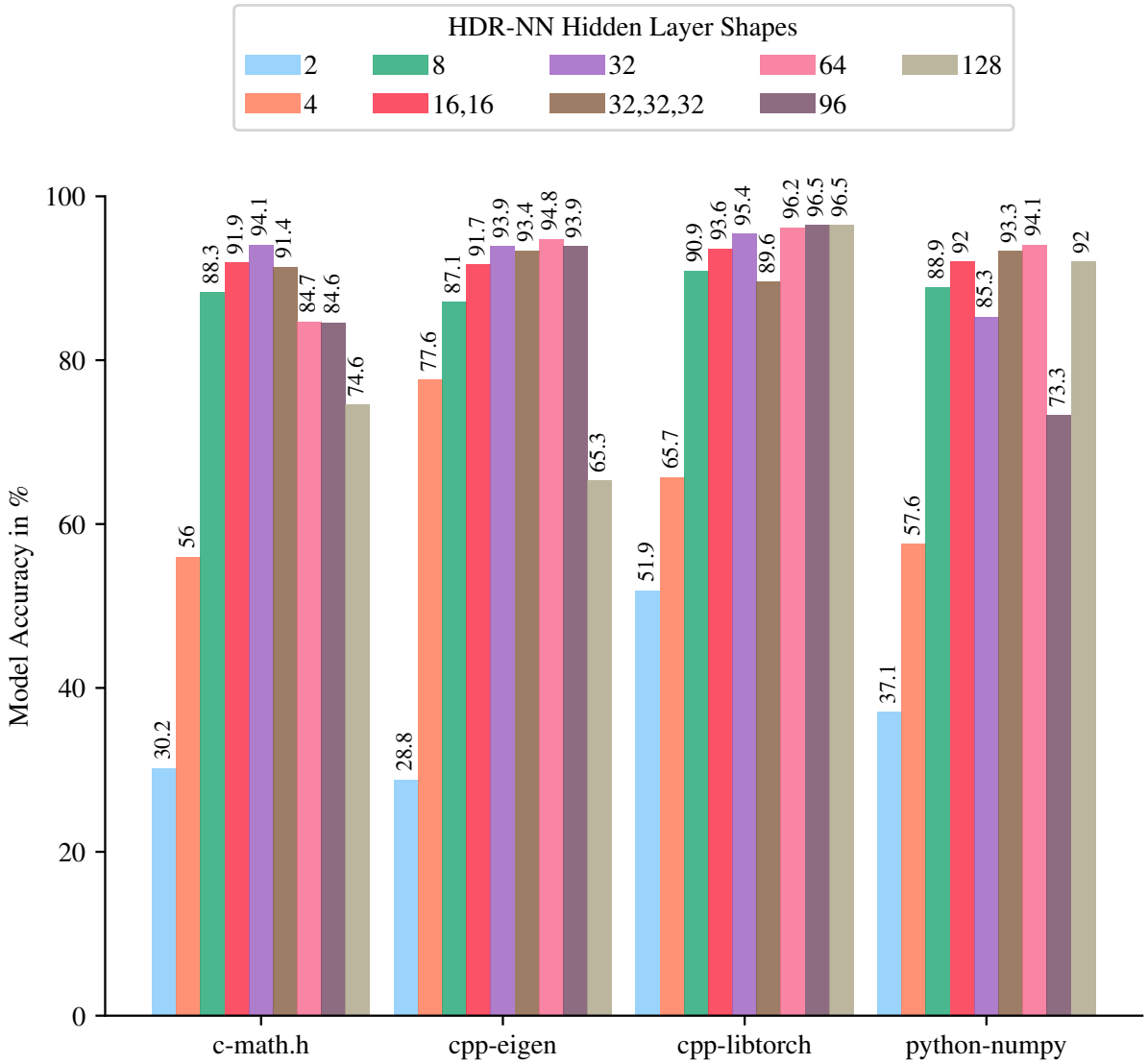
HDR-NN configurability

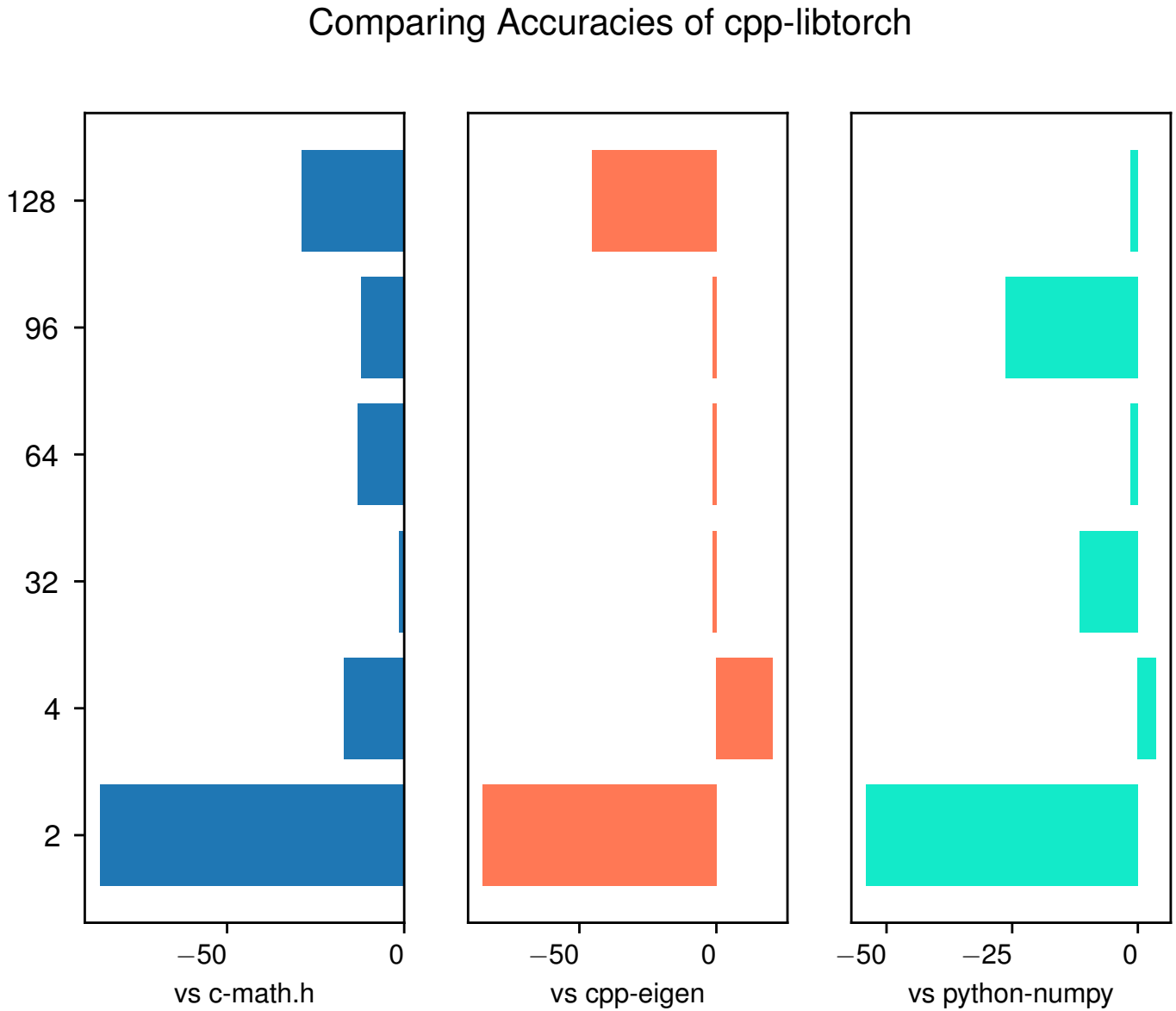
```
→ ./bin/hdrnn
nothing to do
Usage: ./bin/hdrnn

        <command> [<args>]

Commands:
        infer [-i, --image IMAGE_PATH] [-n, --net c-math.nn]
        train [-s, --shape 32] [-e, --epochs 30]
         [-q, --quiet] [-bs, --batch_size=10]
         [-lr, --learning_rate 3] [-n, --net c-math.nn]
```

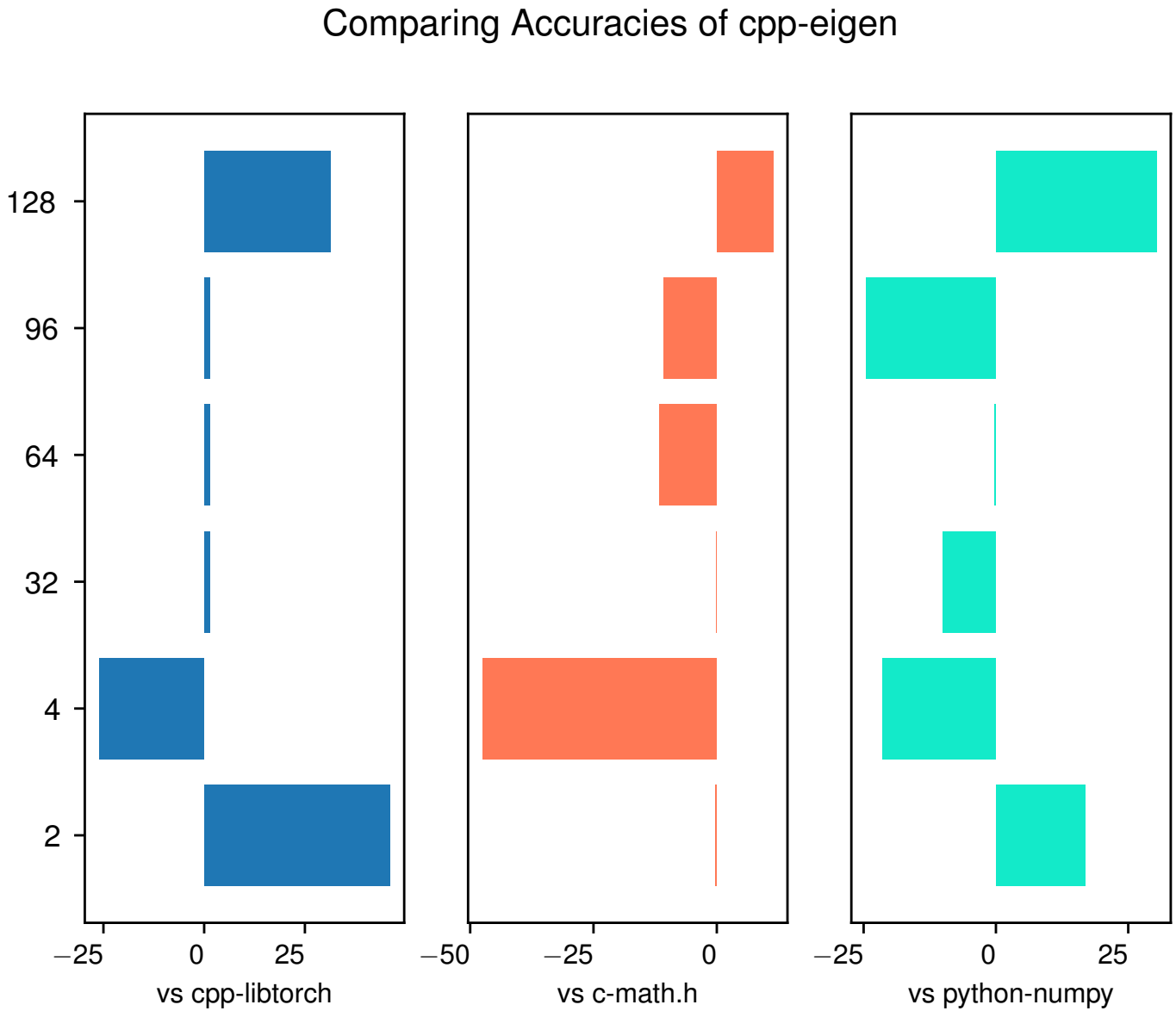
Experiment Considerations

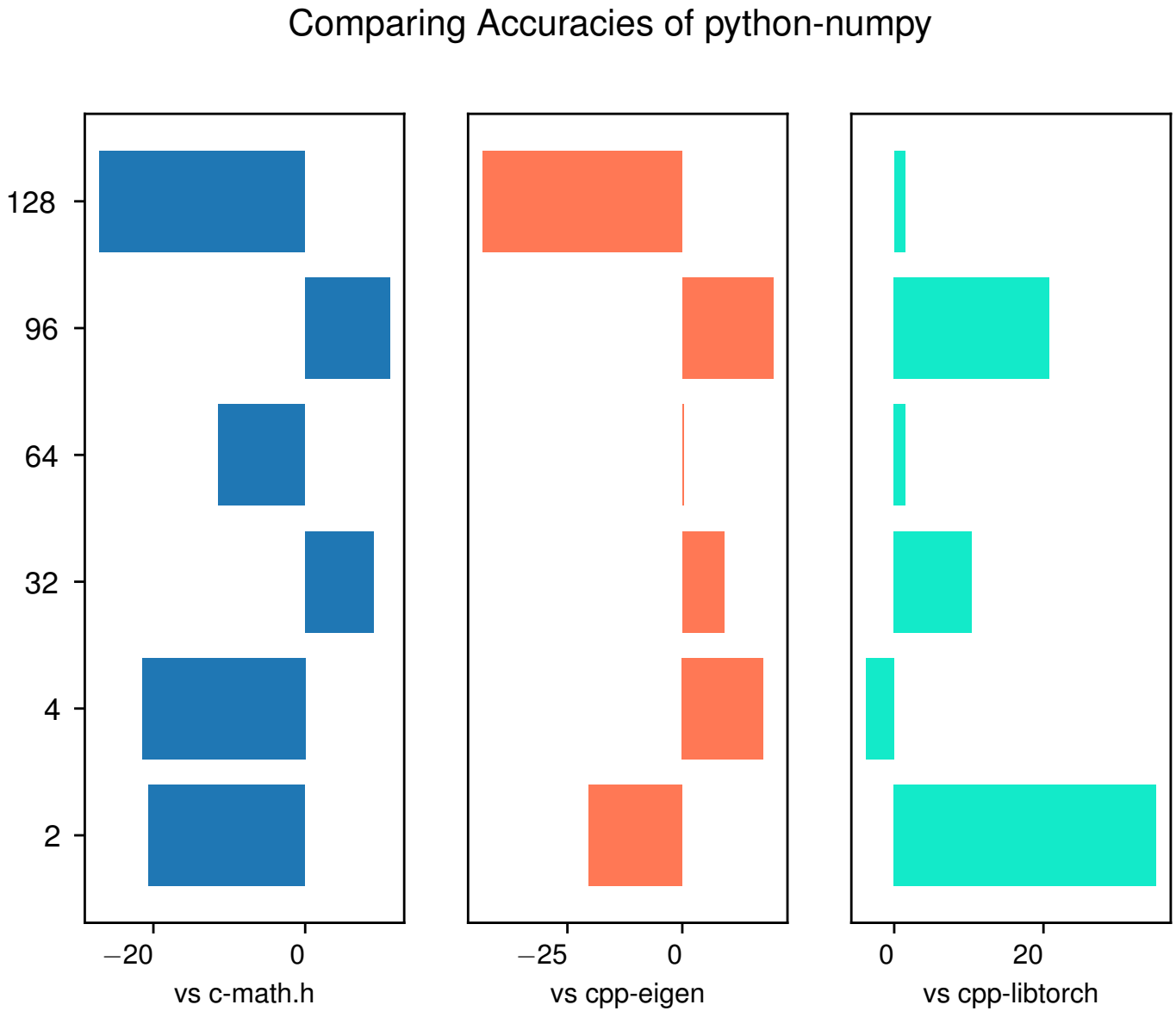
- Try multiple shapes m
- Compare Accuracies
- Compare Execution Time

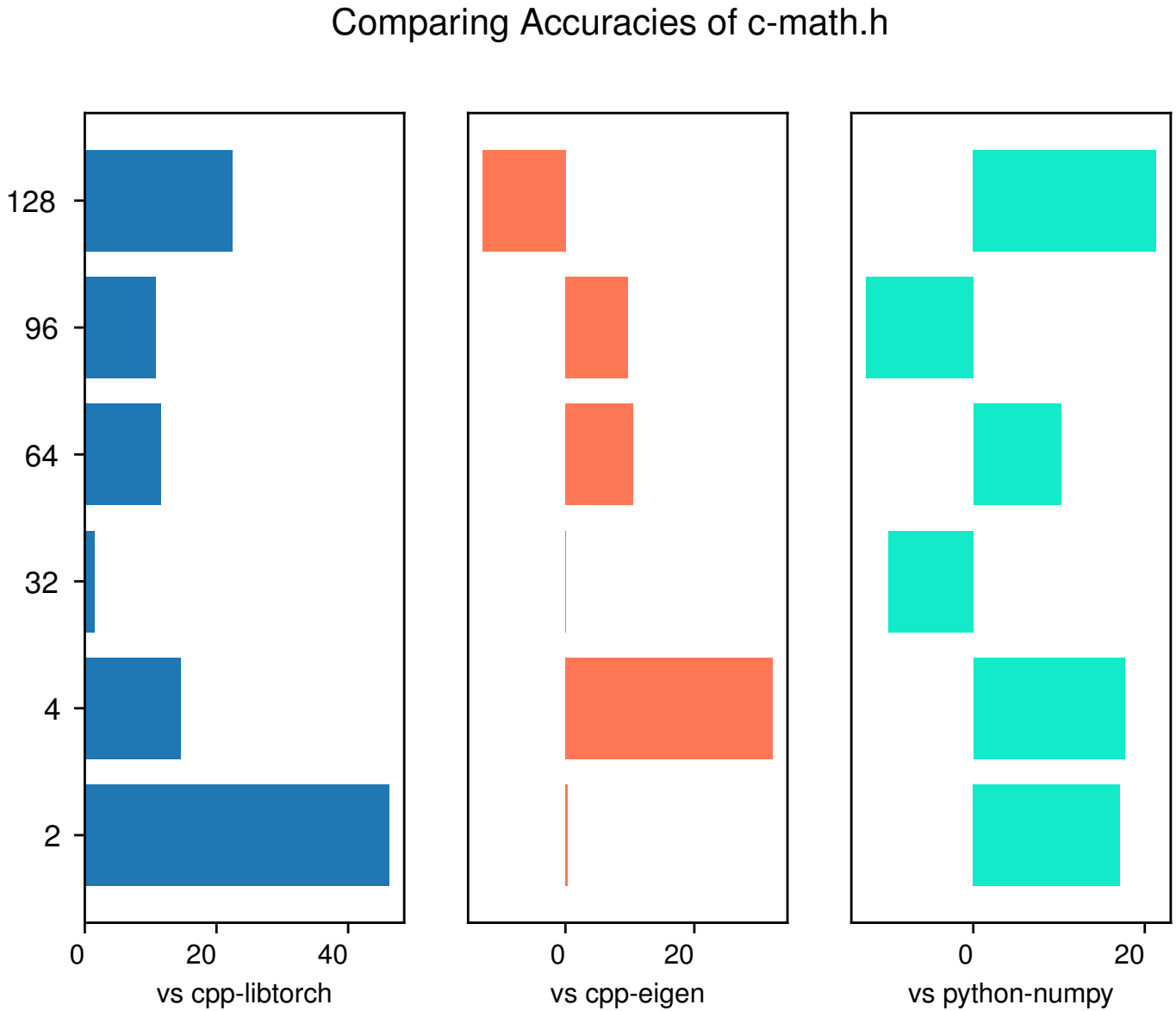
Accuracy


HDR-NN Hidden Layer Shapes
2
4
8
16,16
32
32,32,32
64
96
128
Model Accuracy in %
0
20
40
60
80
100
30.2
56
88.3
91.9
94.1
91.4
84.7
84.6
74.6
28.8
77.6
87.1
91.7
93.9
93.4
94.8
93.9
65.3
51.9
65.7
90.9
93.6
95.4
89.6
96.2
96.5
96.5
37.1
57.6
88.9
92
85.3
93.3
94.1
73.3
92
c-math.h
cpp-eigen
cpp-libtorch
python-numpy

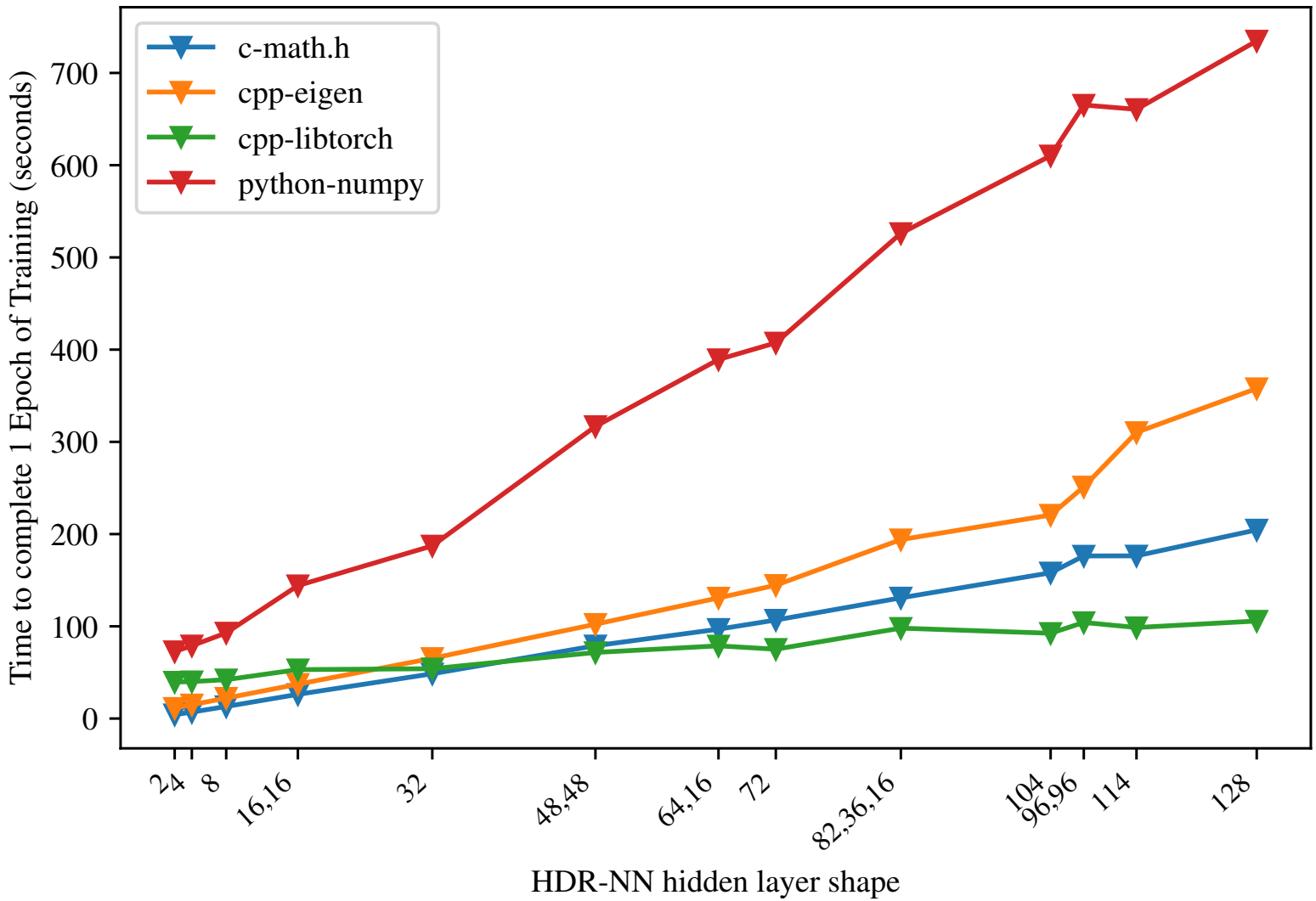

Comparing Accuracies of cpp-libtorch
128
96
64
32
4
2
−50
0
vs c-math.h
−50
0
vs cpp-eigen
−50
−25
0
vs python-numpy

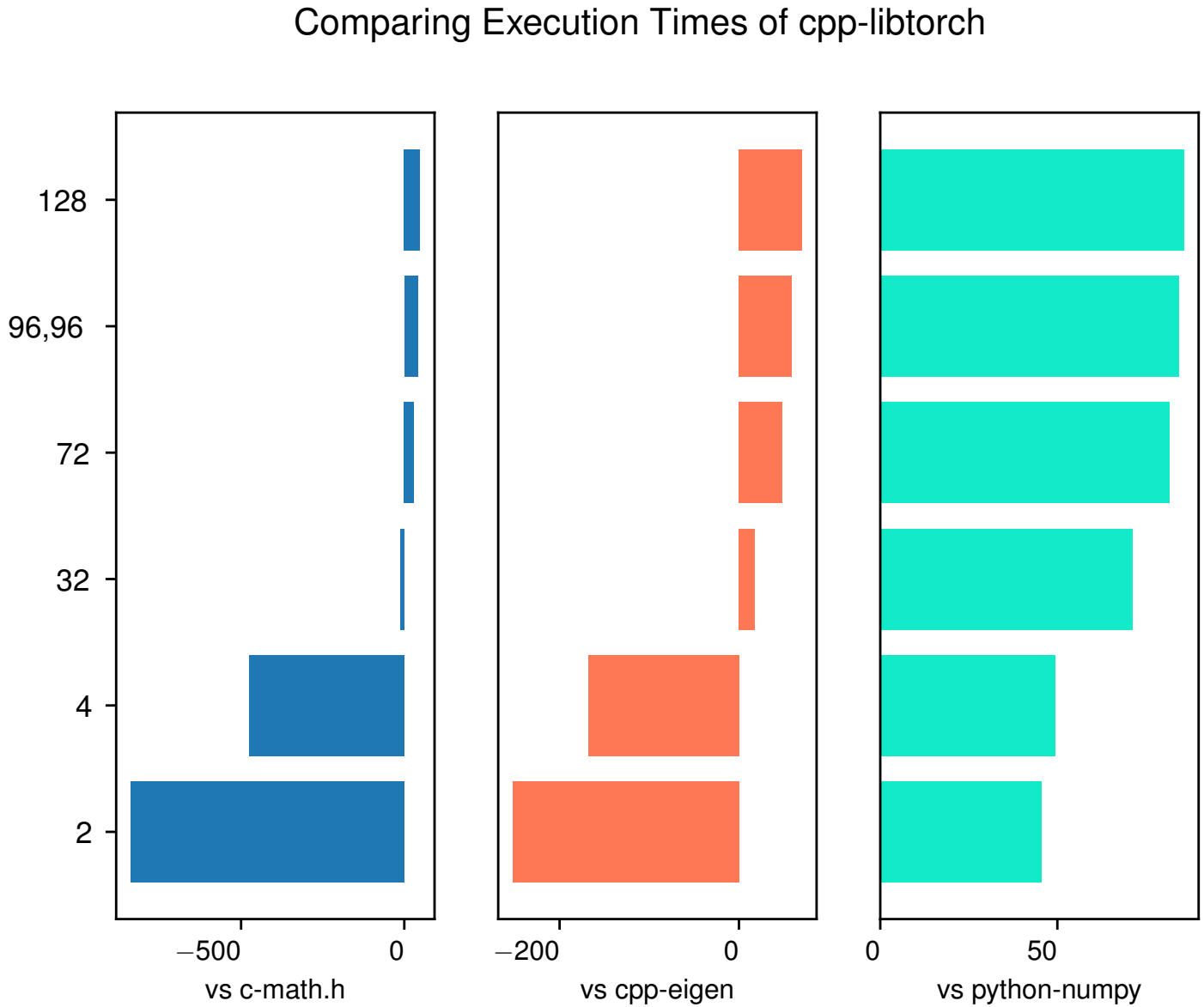

Comparing Accuracies of cpp-eigen
128
96
64
32
4
2
−25
0
25
vs cpp-libtorch
−50
−25
0
vs c-math.h
−25
0
25
vs python-numpy

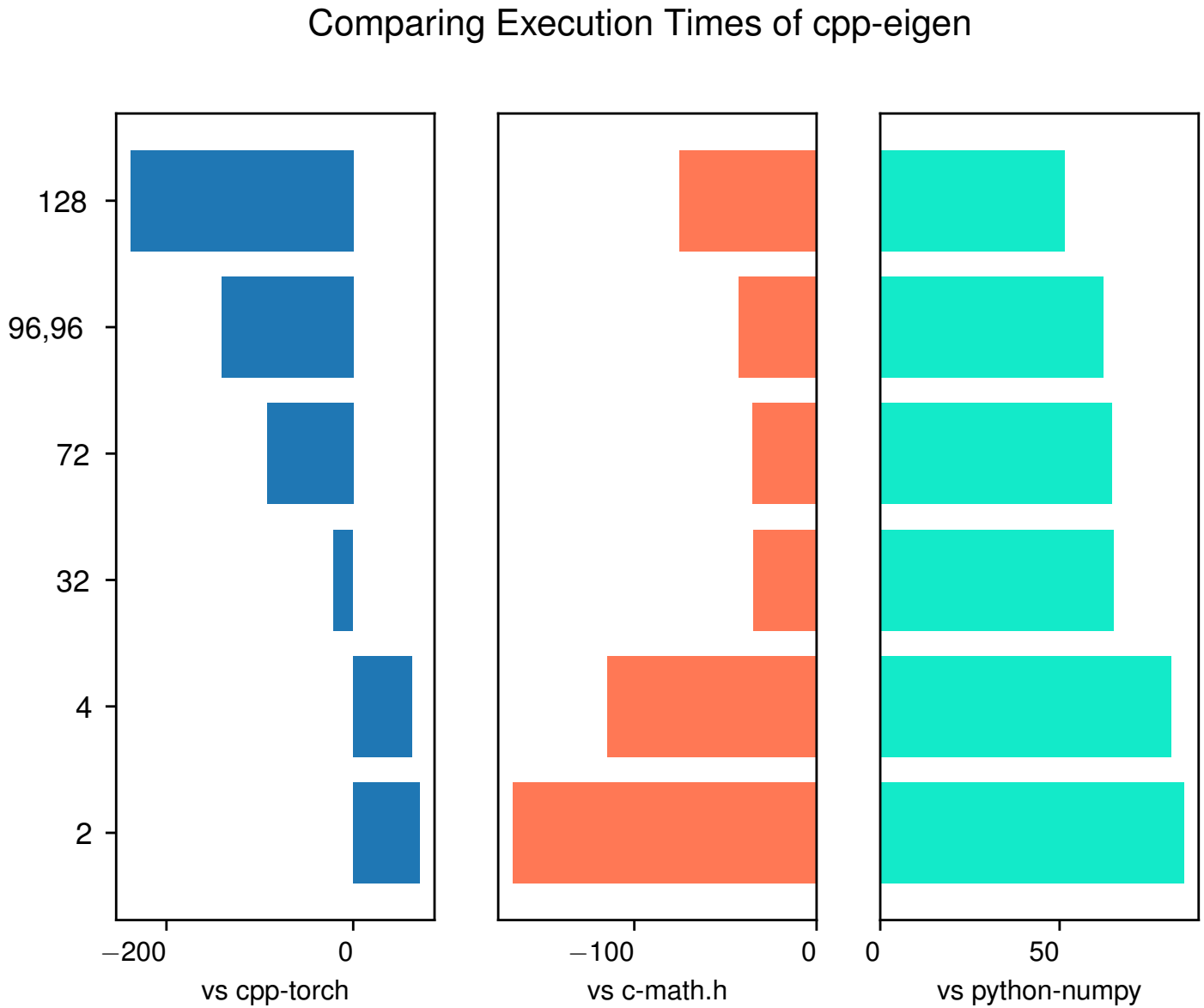

Comparing Accuracies of python-numpy
128
96
64
32
4
2
−20
0
vs c-math.h
−25
0
vs cpp-eigen
0
20
vs cpp-libtorch

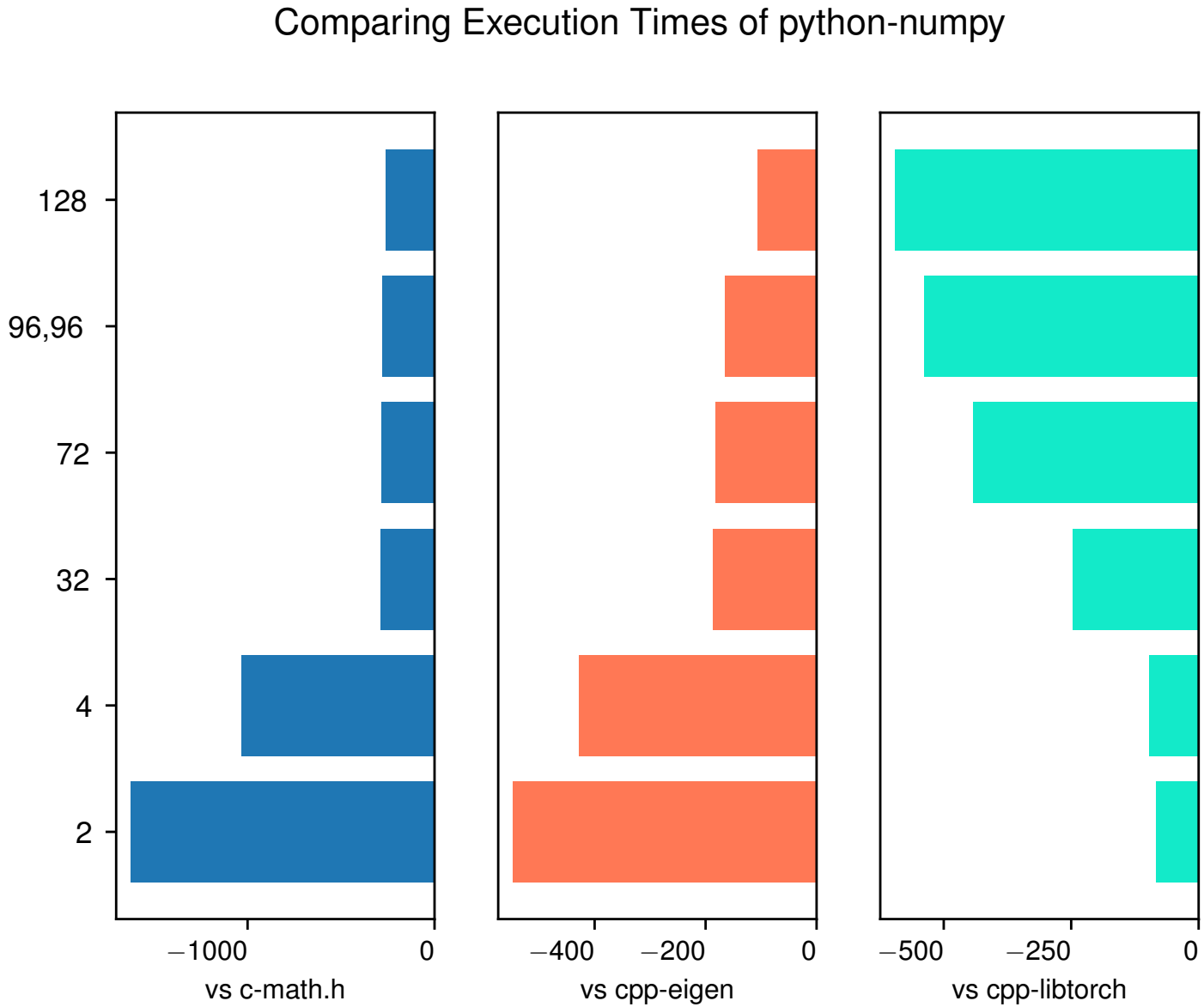

Comparing Accuracies of c-math.h
128
96
64
32
4
2
0
20
40
vs cpp-libtorch
0
20
vs cpp-eigen
0
20
vs python-numpy

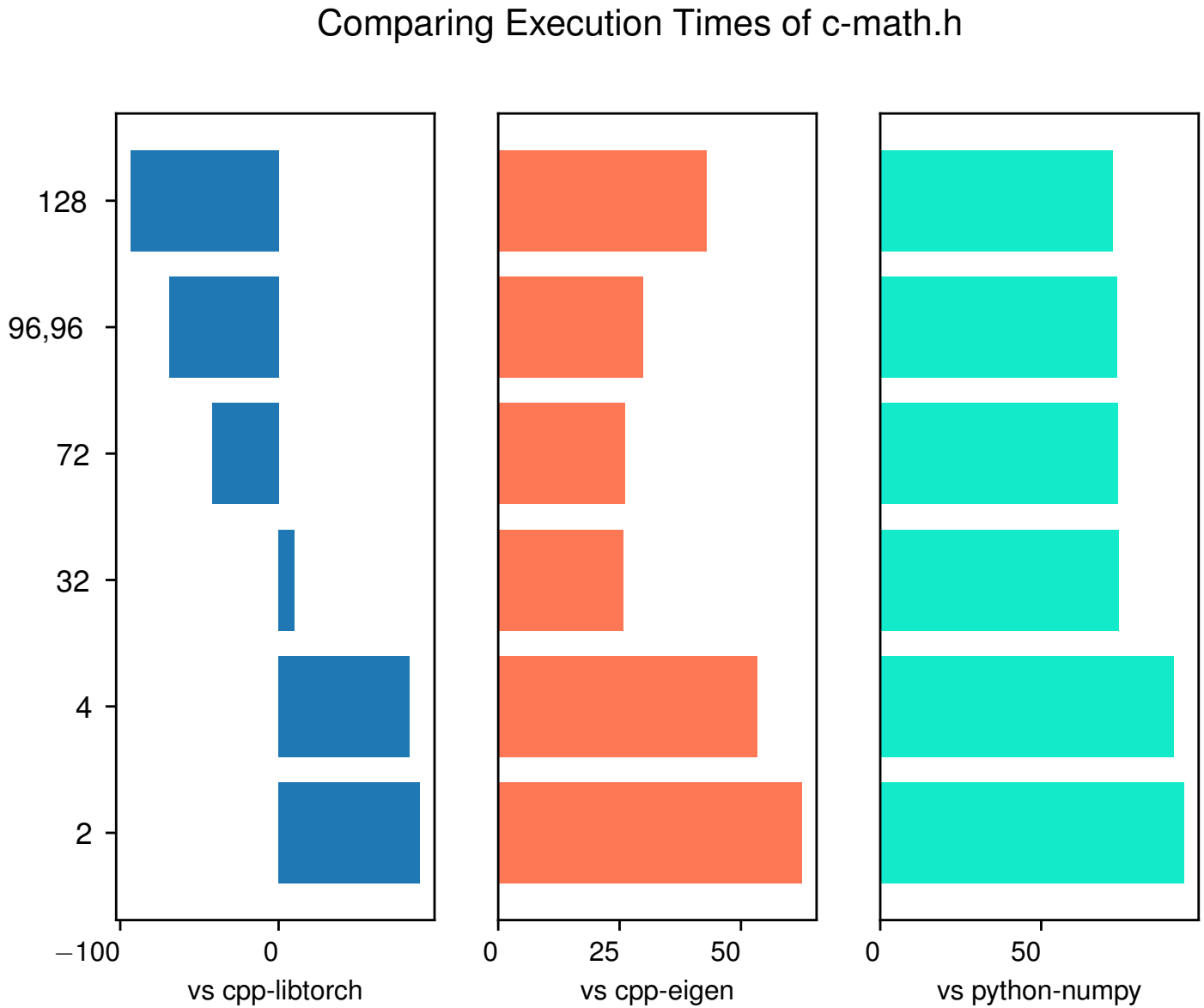
Training Time




Comparing Execution Times of cpp-libtorch
128
96,96
72
32
4
2
−500
0
vs c-math.h
−200
0
vs cpp-eigen
0
50
vs python-numpy


Comparing Execution Times of cpp-eigen
128
96,96
72
32
4
2
−200
0
vs cpp-torch
−100
0
vs c-math.h
0
50
vs python-numpy


Comparing Execution Times of python-numpy
128
96,96
72
32
4
2
−1000
0
vs c-math.h
−400
−200
0
vs cpp-eigen
−500
−250
0
vs cpp-libtorch


Comparing Execution Times of c-math.h
128
96,96
72
32
4
2
−100
0
vs cpp-libtorch
0
25
50
vs cpp-eigen
0
50
vs python-numpy

Reverse Engineering Scania ECU

- Goal : Get a vanilla U-Boot running on ECU
- Extract device tree information
- Silent bootloader, No visibility into Board/SoM fork

UPPSALA UNIVERSITET

Neural Network Software Development on Embedded Devices

- Embedded Platforms are fragmented in both *hardware* and *software*
- C is easier to target harder to Implement
- Python commonly found in Neural Network Research, harder to port

Future Work

- Reverse Engineering Efforts on the ECU
- Performance engineering the `C` based implementation
- Adding more target devices
- Adding more implementations (Tensorflow, TTE)

UPPSALA
UNIVERSITET

End

Thank you

Questions?