TECHNISCHE UNIVERSITÄT WIEN

CDL-EML TU Wien
ICCAD TinyML Contest 2023

Daniel Schnöll

daniel.schnoell@tuwien.ac.at

Dominik Dallinger

dominik.dallinger@tuwien.ac.at

Bundesministerium Digitalisierung und Wirtschaftsstandort

Christian Doppler Laboratory

Team CDL-EML TU Wien

Christian Doppler Laboratory

Embedded Machine Learning

Dipl. Ing. Daniel Schnöll
daniel.schnoell@tuwien.ac.at

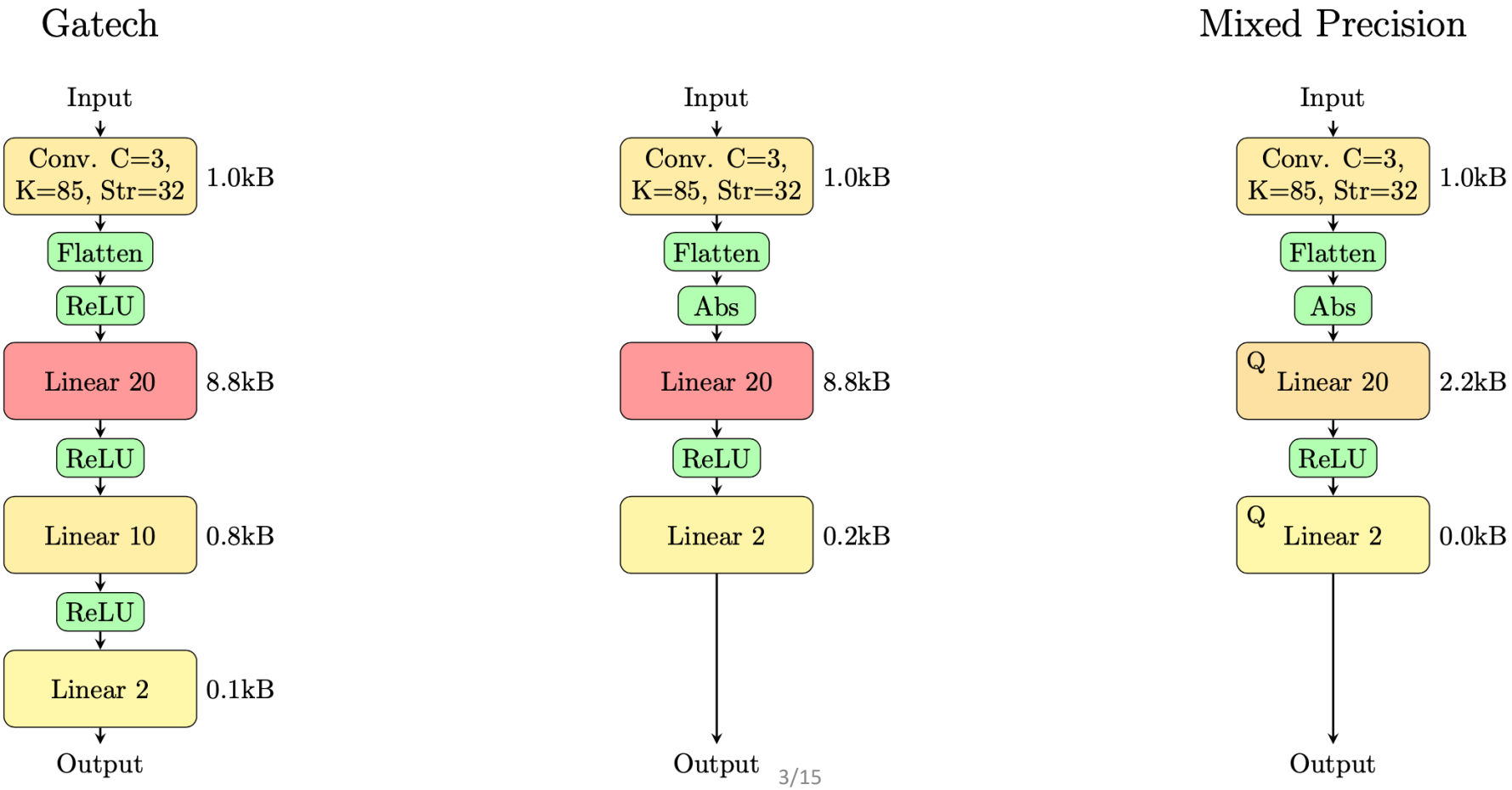
Doctoral student in Embedded Systems
(Electrical Engineering)

Dominik Dallinger, BSc
dominik.dallinger@tuwien.ac.at

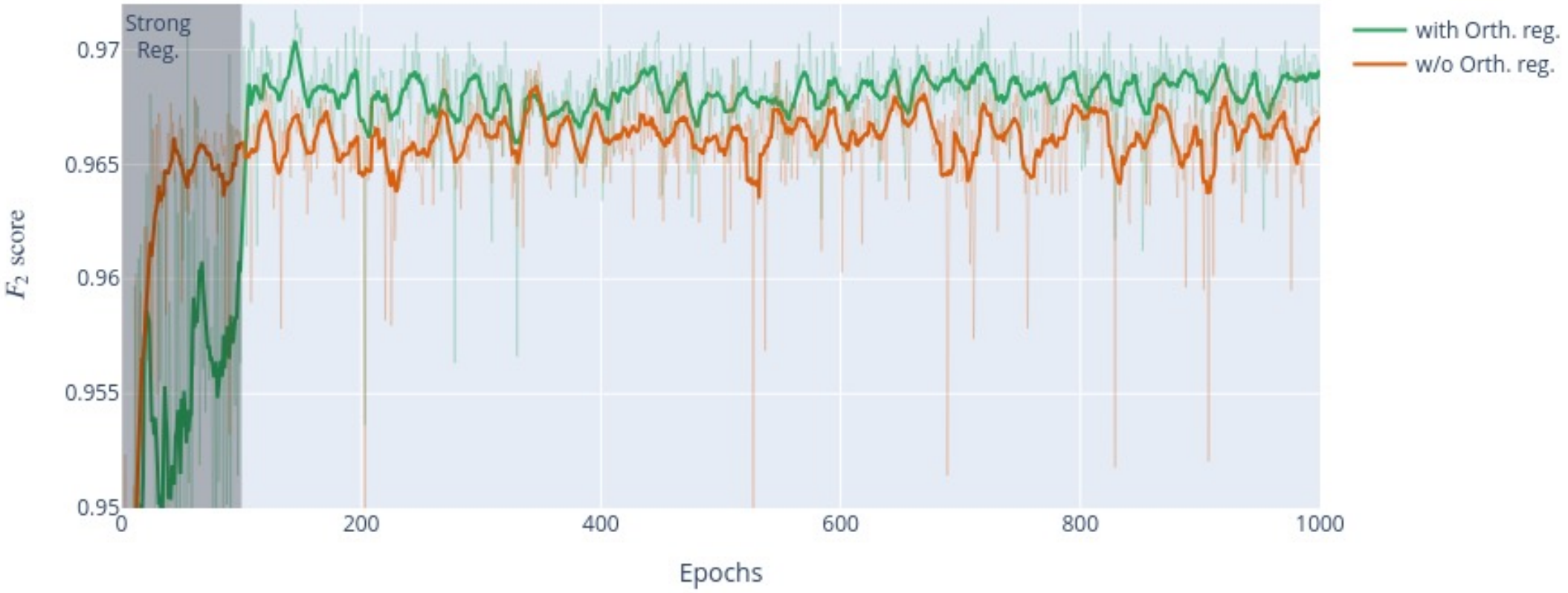
Master student in Embedded Systems
(Electrical Engineering)

TECHNISCHE UNIVERSITÄT WIEN

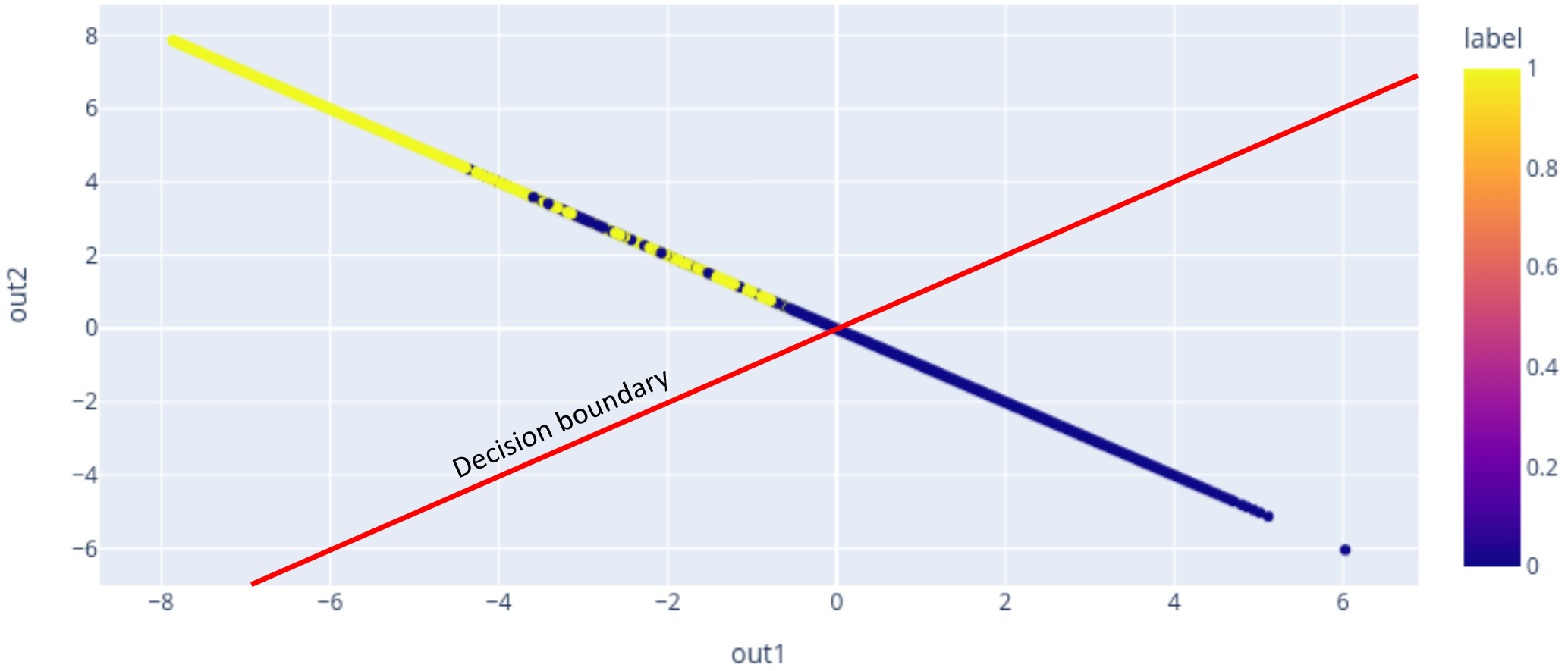
Model


Gatech
Input
Conv. C=3, K=85, Str=32
1.0kB
Flatten
ReLU
Linear 20
8.8kB
ReLU
Linear 10
0.8kB
ReLU
Linear 2
0.1kB
Output
Input
Conv. C=3, K=85, Str=32
1.0kB
Flatten
Abs
Linear 20
8.8kB
ReLU
Linear 2
0.2kB
Output
Mixed Precision
Input
Conv. C=3, K=85, Str=32
1.0kB
Flatten
Abs
Q
Linear 20
2.2kB
ReLU
Q
Linear 2
0.0kB
Output

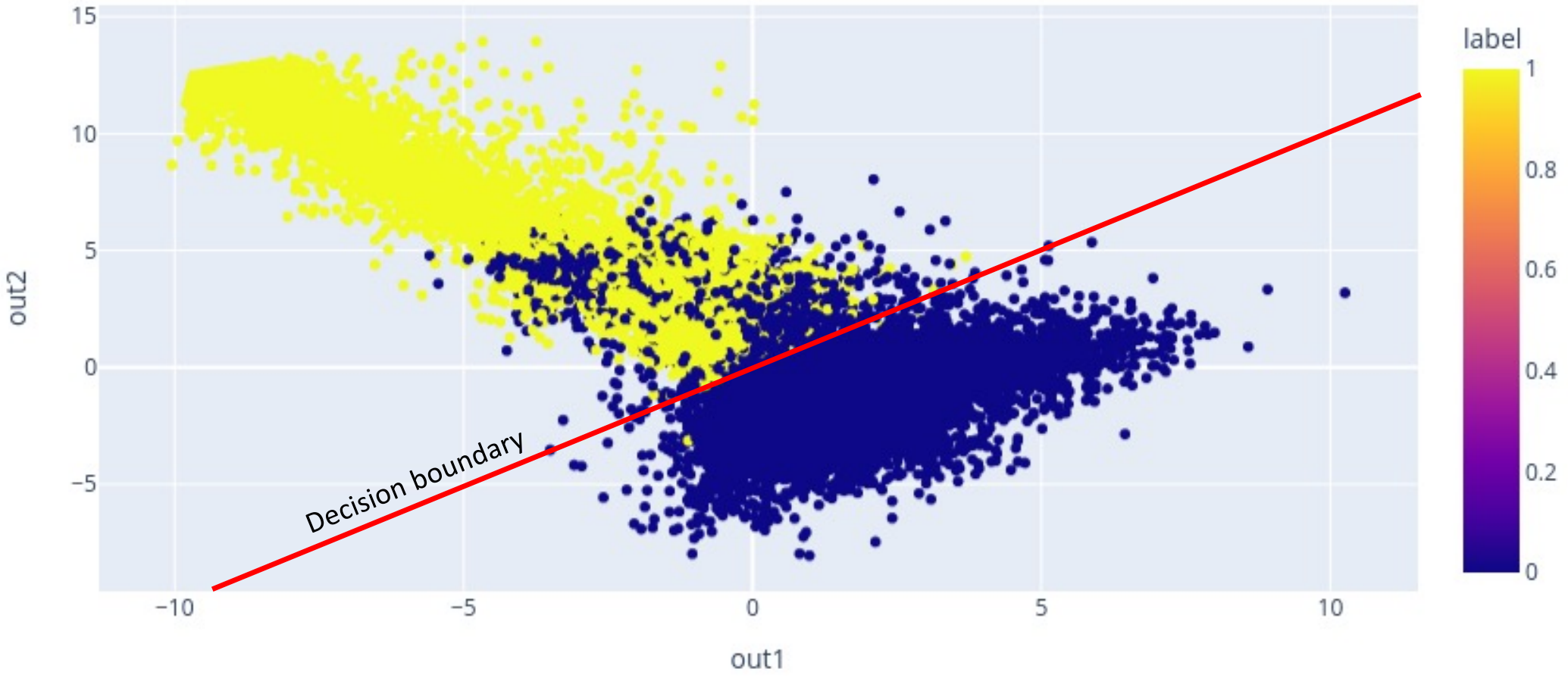
Orthogonal Regularization



Orthogonalization effect – Baseline



Orthogonalization effect – Regularized



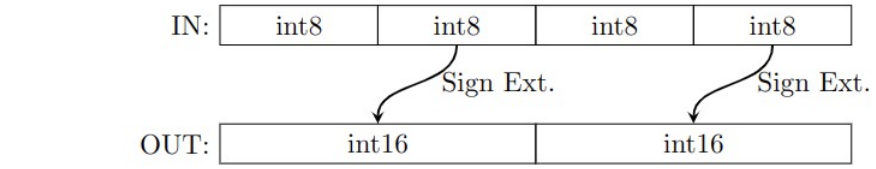
C++ Project

```
void aiRun(const float input[1][1250][1], float result[2])
{
    const Matrix<1, 1250> *IN = (Matrix<1, 1250> *)(void *)input;

    const auto I1 = ConvT<32>(*IN,Conv_weight_ram,Conv_bias,Conv_ACT);

    const auto I2 = Linear<int16_t> (I1,    fc2_weight_ram, fc2_bias,   fc2_right_shift,    fc2_ACT);
    const auto I3 = Linear<int32_t> (I2,    fc3_weight,     fc3_bias,   fc3_right_shift,    fc3_ACT);
    result[0] = float_conversion[0] * (float)I3.data[0][0];
    result[1] = float_conversion[1] * (float)I3.data[0][1];
}
```

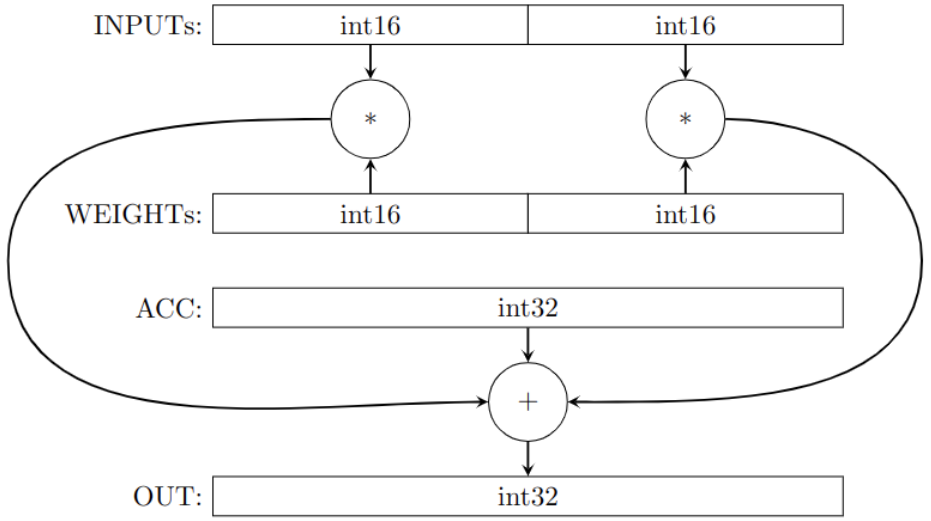
C++ Project – SIMD Instructions

```
for (; i1_2 < M1_2 - 3; i1_2 += 4)
{
    SIMD8 w{.true_data = {  //Weights
                B.data[i2_1][i1_2],
                B.data[i2_1][i1_2 + 1],
                B.data[i2_1][i1_2 + 2],
                B.data[i2_1][i1_2 + 3],
            }};
    SIMD16 in1{.true_data = { //First Input
                    A.data[i1_1][i1_2],
                    A.data[i1_1][i1_2 + 1]}};
    SIMD16 in2{.true_data = { //Second Input
                    A.data[i1_1][i1_2 + 2],
                    A.data[i1_1][i1_2 + 3]}};
    SIMD16 w1{.simd = __SXTB16(w.simd)};
    SIMD16 w2{.simd = __SXTB16_ROR8(w.simd)};

    sum.simd = __SMLAD(in1.simd, w1.simd, sum.simd);
    sum.simd = __SMLAD(in2.simd, w2.simd, sum.simd);
}
```

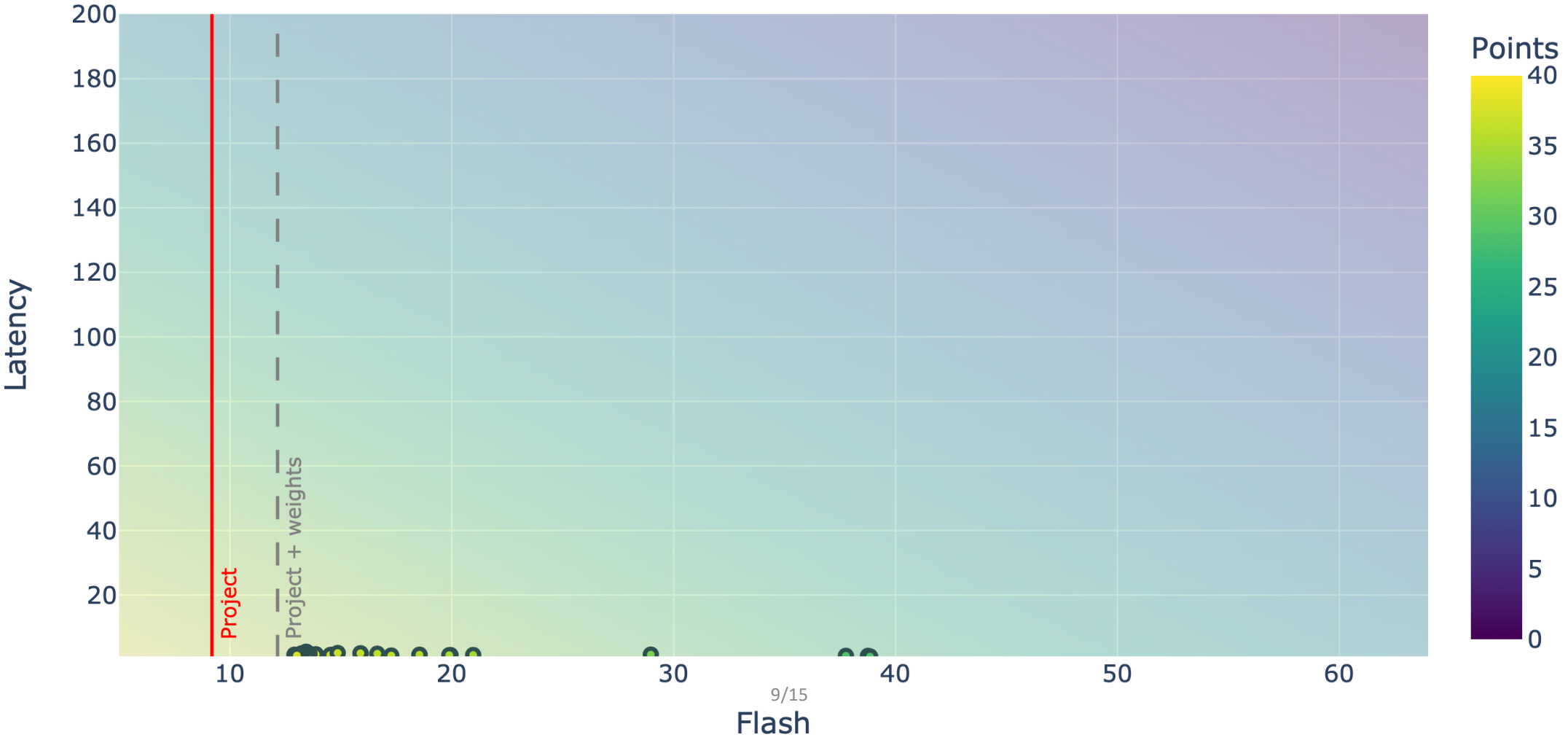
SXTB16:



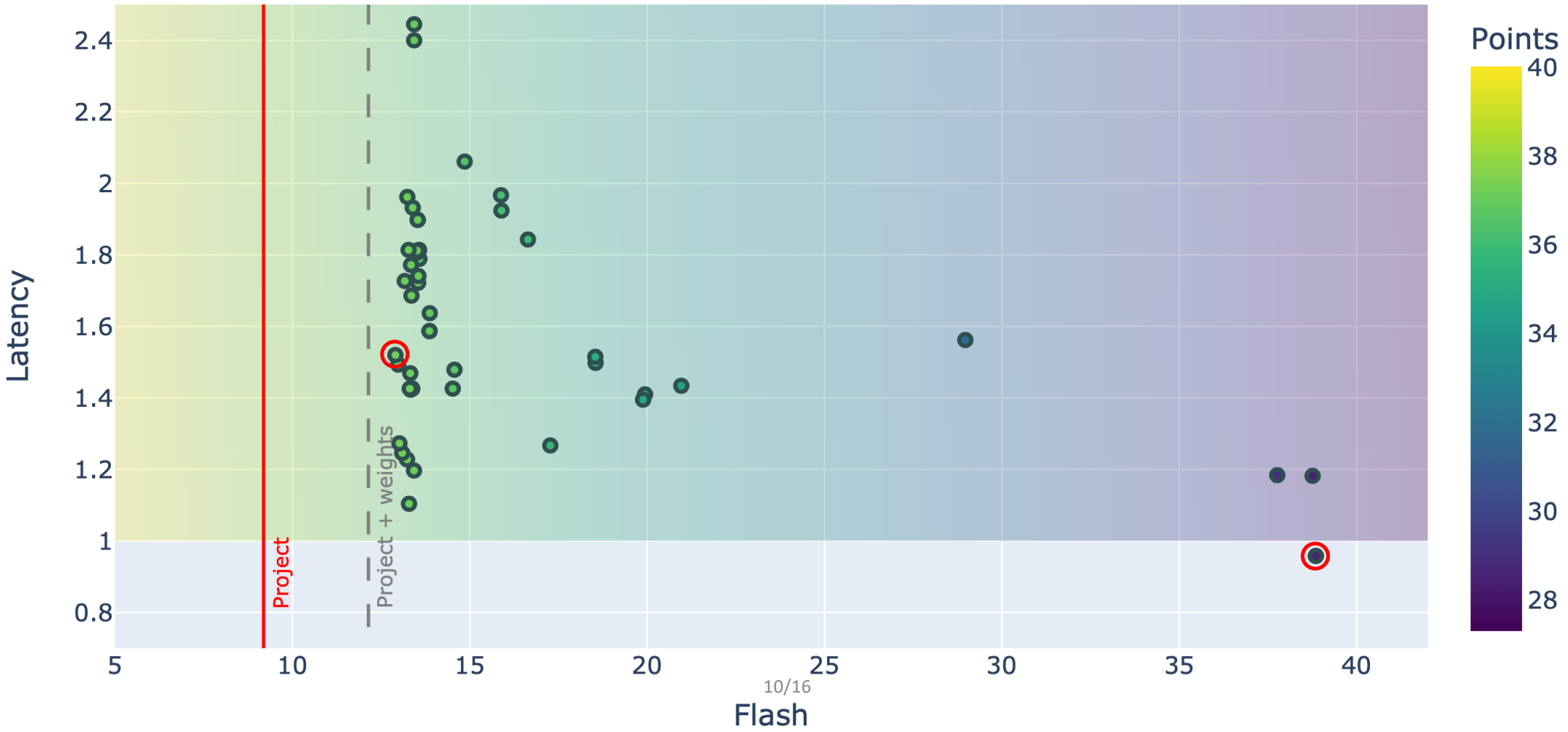
SMLAD:



Score


Latency
200
180
160
140
120
100
80
60
40
20
Project
Project + weights
10
20
30
40
50
60
Flash
Points
40
35
30
25
20
15
10
5
0

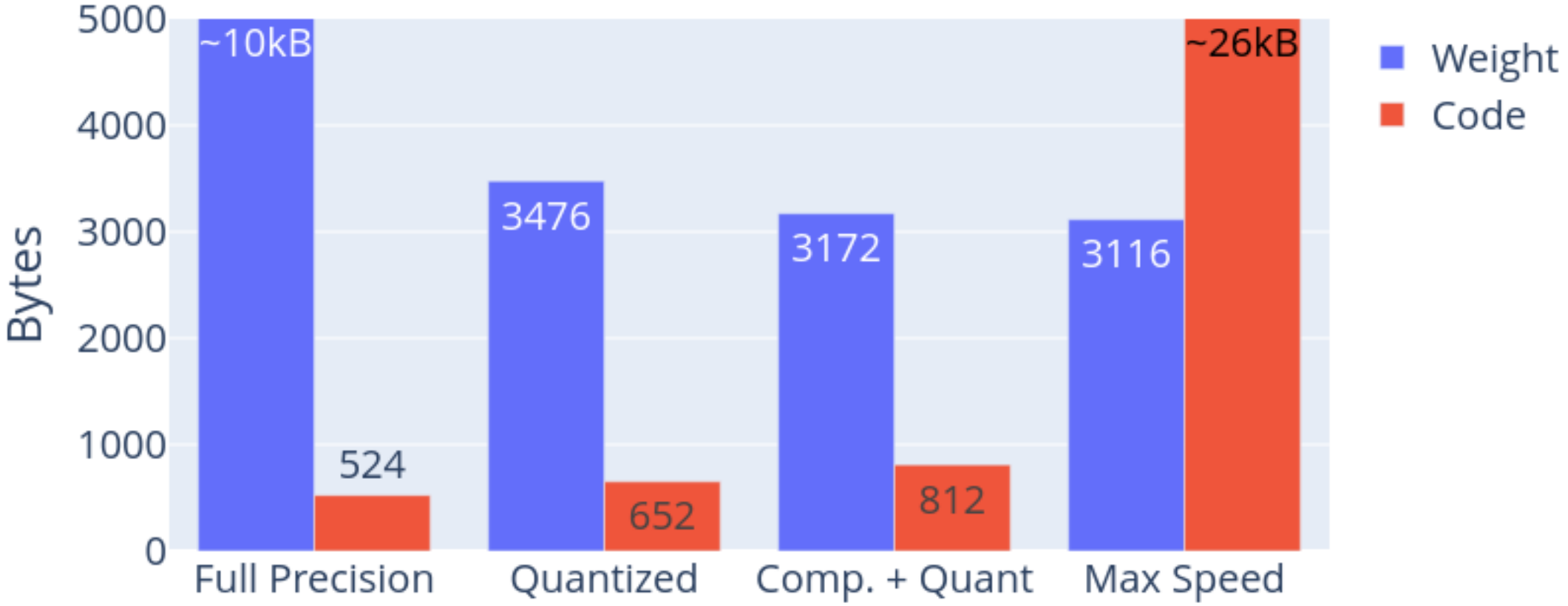
Score


Latency
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
Project
Project + weights
5
10
15
20
25
30
35
40
Flash
Points
40
38
36
34
32
30
28

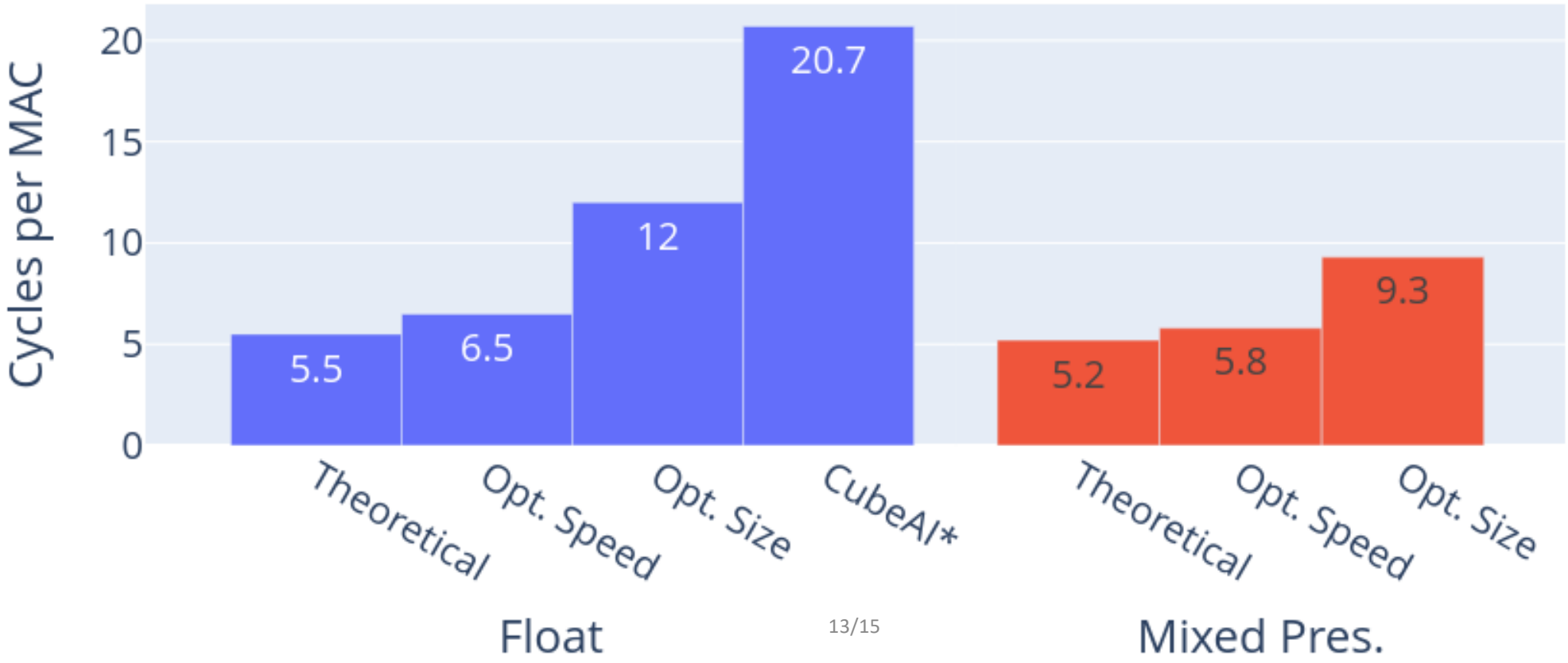
Model Compression

- Full Precision convolution:
  - 32 Bit Float Input
  - 32 Bit Float Weights
  - 32 Bit Float Bias
  - 16 Bit Integer Output
- Quantized Linear layer:
  - 16 Bit Integer Input
  - 8 Bit Integer Weights (Upper 4 Bits Huffman encoded for the big layer)
  - 32 Bit Integer Bias
  - 32 Bit Integer Shift values (Ups ¯_(ツ)_/¯)
  - 16-32 Integer Bit Output

Model Compression



Performance



Summary

Network

- Orthogonal Regularization
- Test different Activation Functions
- Quantization!

Implementation

- Compiler – Powerful tool
- Weight/Channel Reordering
- C++ for µC

Thank you for your attention!

Questions?