

Even Higher-Level Synthesis

An Exploration of AI Hardware Accelerators using HLS4ML



SPONSORED BY





Cameron Villone

HLS Technologist – Siemens EDA

Cameron Villone joined Siemens in August 2023 through the Atlas New Graduate Program. Cameron graduated from Rochester Institute of Technology with a Masters Degree in Electrical Engineering focusing on Robotics, Embedded Systems, and Computer Vision. Cameron has held previous student roles at General Motors and Texas Instruments. Cameron started his Siemens journey by working as a product marketer for Siemens's low power solution, PowerPro. Cameron then grew to his current role as part of the product management team as an HLS Technologist for Catapult AI/NN.

SIEMENS





Giuseppe Di Guglielmo

Senior ASIC Engineer – Fermilab

Giuseppe Di Guglielmo is a Senior Engineer at Fermilab focused on system-level design and AI/ML hardware acceleration. He develops intelligent, ultra-low-latency detectors for harsh environments, including ML-enabled, radiation-resistant chips for the LHC and quantum hardware for cryogenic systems. With a Ph.D. in Computer Science and over a decade of experience in high-level synthesis for ASIC/FPGA design, he previously held research roles at Columbia University and Tokyo University. He is an active contributor to open-source projects like ESP and hls4ml.



Why Customized Accelerators?



SPONSORED BY



Inferencing Will Be Everywhere

AI can make embedded devices:

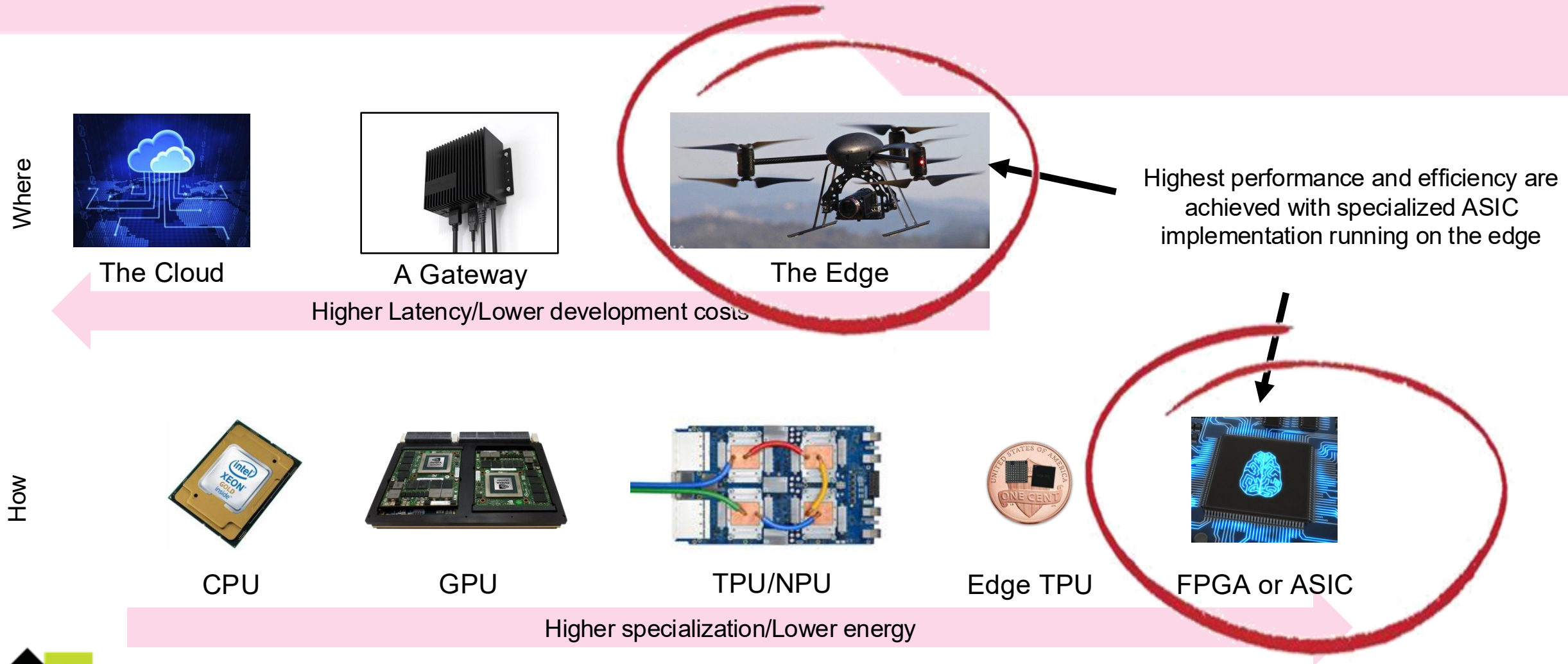
- More capable
- More secure
- Safer
- Faster



Automotive



Deploying AI in the Edge Systems

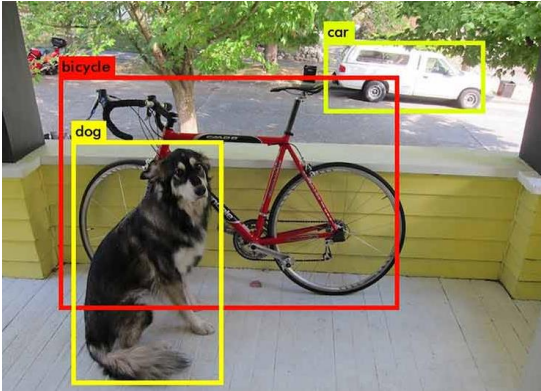


Hardware vs Software

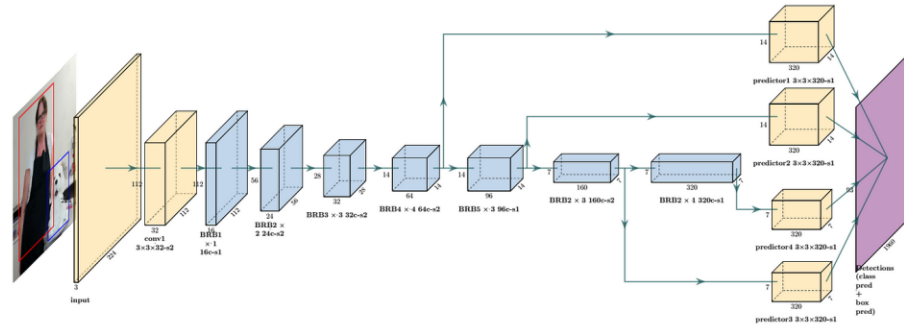
Pure Software Implementation		Software with generic hardware accelerator		Software with bespoke hardware accelerator	
Pros	Cons	Pros	Cons	Pros	Cons
Very Flexible	Performance and Timing issues for real-time applications	Retains Moderate Flexibility	Relies on standard HW	Very Low power and predictable timing	Requires development of custom hardware
Easy to update			Power consumption and timing issues-		Fixed to a limited set of network architectures



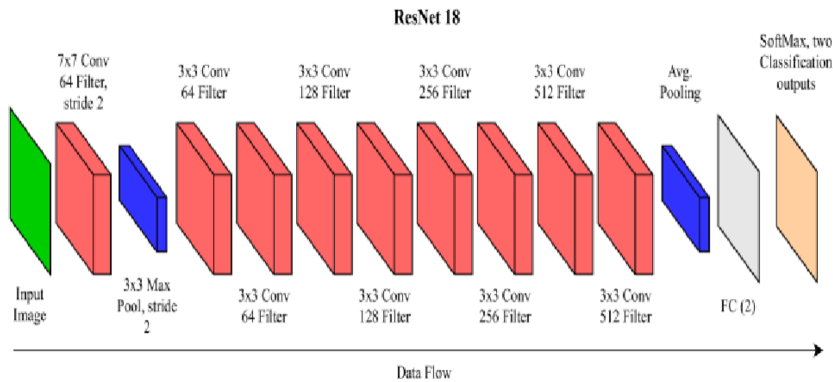
More and More Models



Yolo v1 – v8



MobileNet

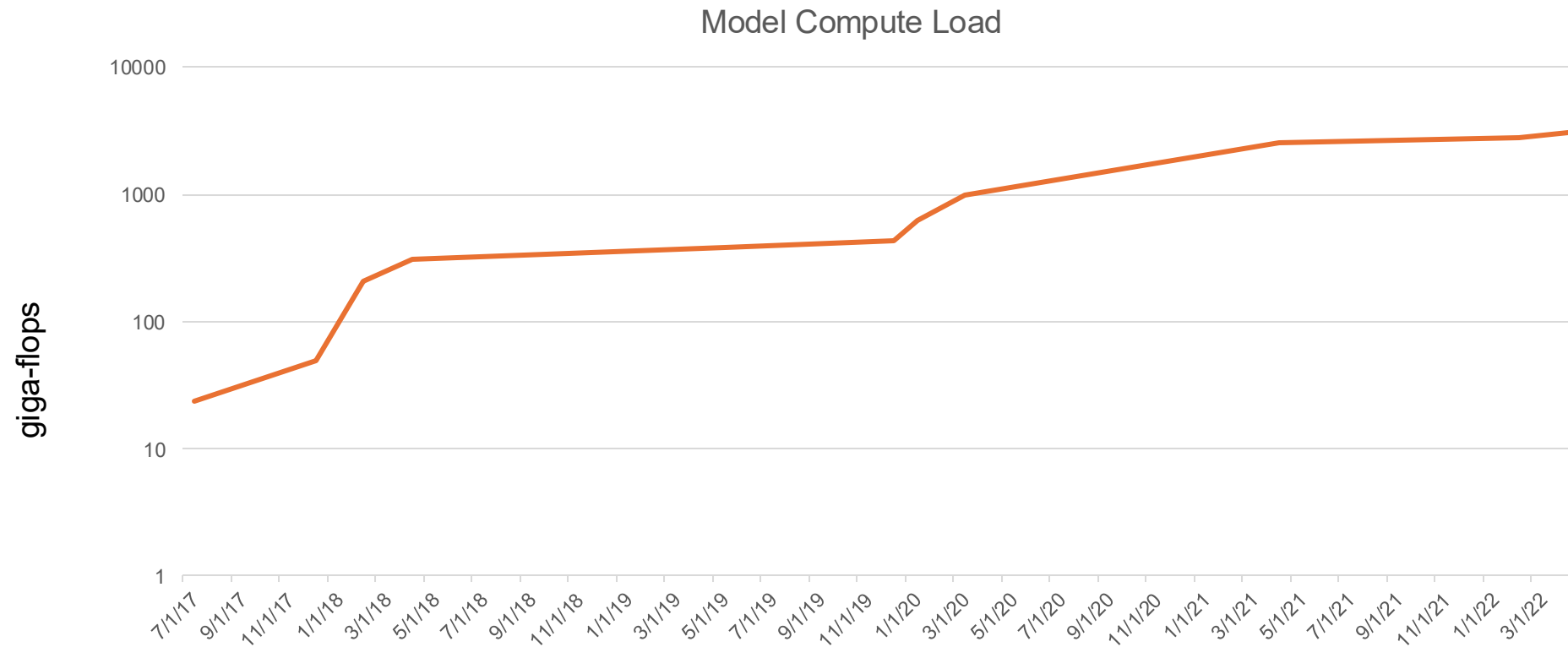


ResNet

Many Many More....

- DenseNet
- AlexNet
- EfficientNet
- SqueezeNet
- VGG
- Inception
- ResNeXt
- More and More.....

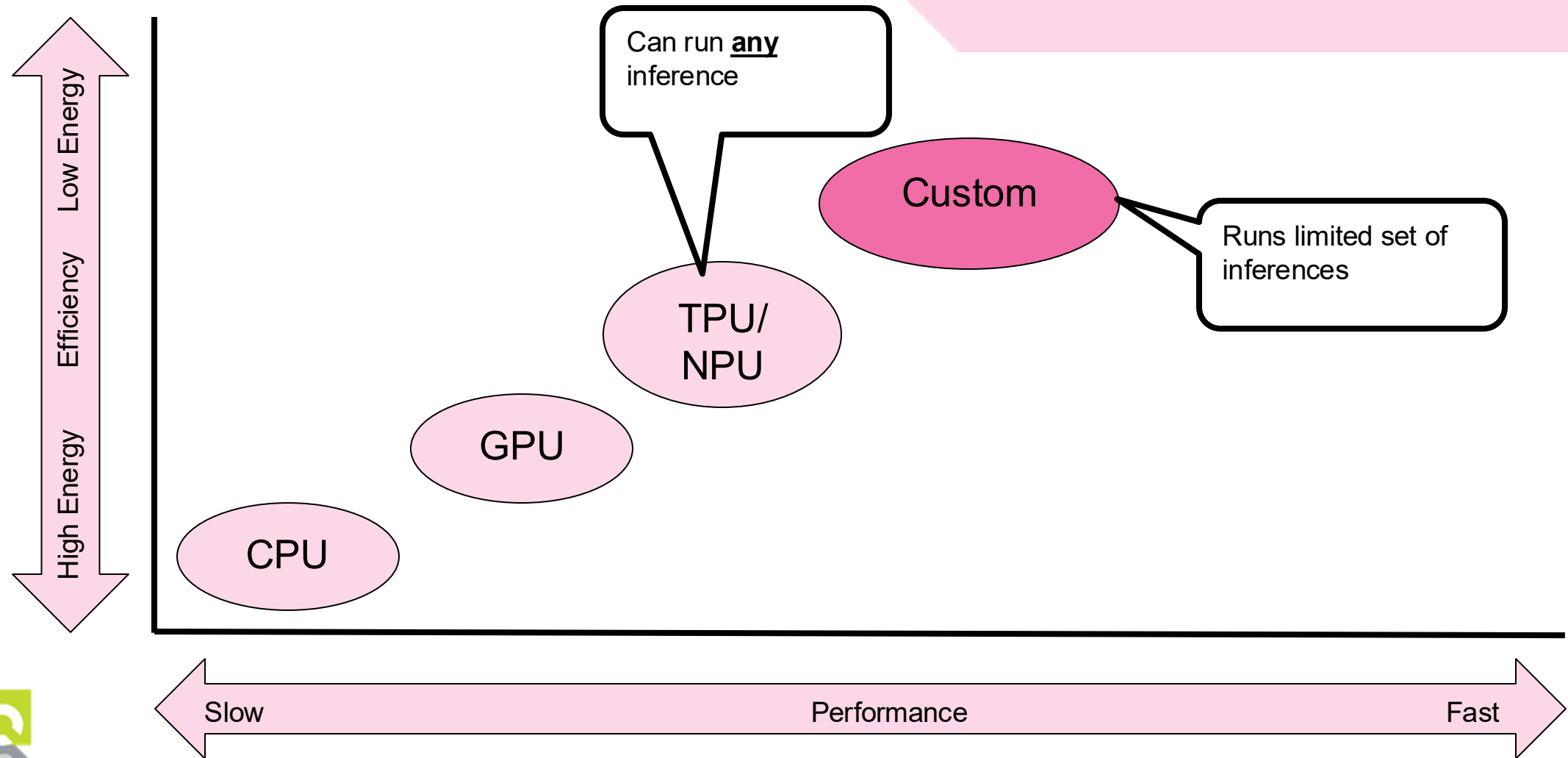
Model Size of Best ImageNet Algorithm



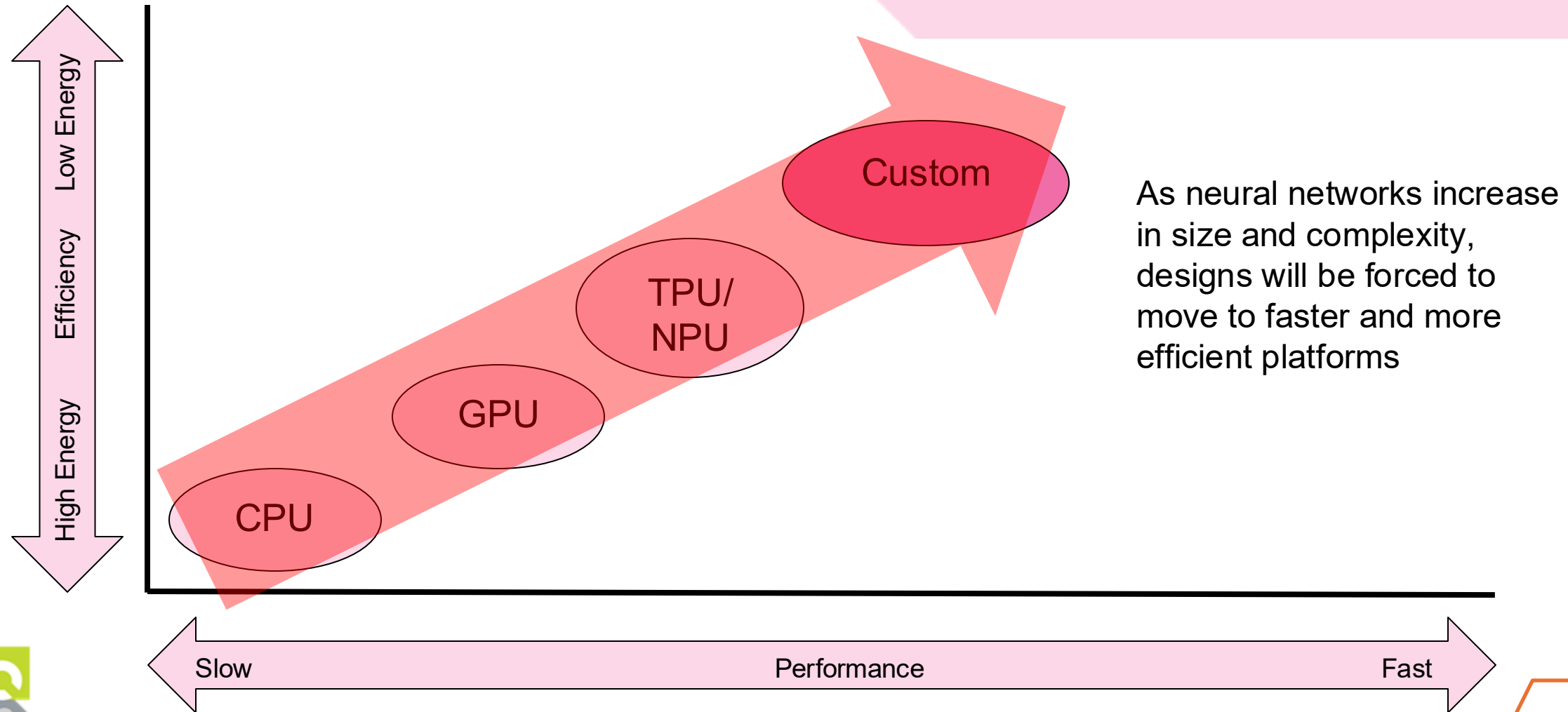
Models have increased in computational load by >100X in 5 years



Inference Execution



Complexity Drives Need for Customization



Drivers for ASIC Inferencing on the Edge

Drivers to the edge:

- Latency
- Security
- Privacy



Latency



Security



Privacy

Drivers to ASIC:

- Performance
- Efficiency



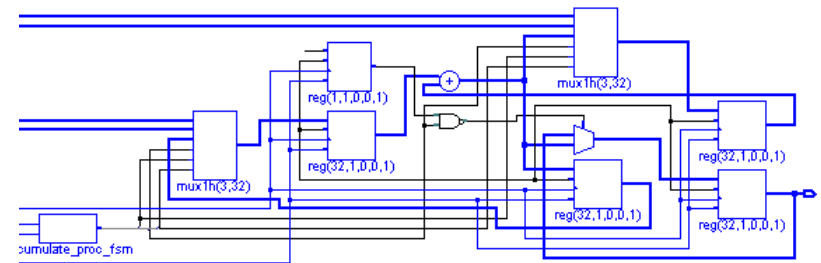
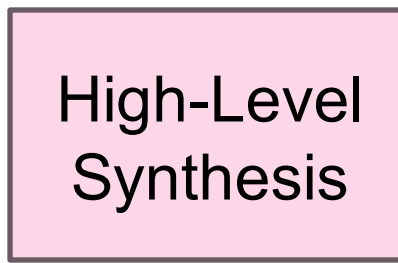
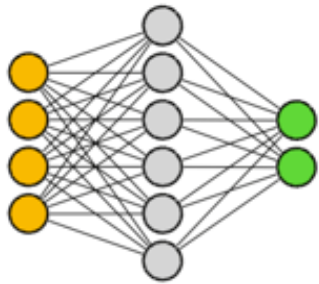
Performance



Efficiency

Inferencing on the Edge

- As AI algorithms get more complex, processors, software and off the shelf accelerators will struggle to meet design requirements
- Technology trends are driving edge inferencing to be done on device
- Designing a bespoke accelerator can deliver the highest performance and efficiency
- High-Level Synthesis delivers the fastest path from machine learning framework to RTL



What is High-Level Synthesis?



SPONSORED BY



What is High-Level Synthesis (HLS)?

```
361 void copy_to_regs(hw_cat_type *dst, index_type dst_offset, raw_memory_line *src, index_type src_offset, index_type size)
362 {
363     // read out of internal memories to an array of registers
364     // *should* make it easy for catapult to pipeline access to internal memories
365
366     index_type count;
367     index_type n;
368
369     static const index_type stride = STRIDE;
370
371     count = 0;
372     while (count < size) {
373         n = stride;
374         if ((size - count) < stride) n = size - count; // mis-aligned at the end of transfer
375         read_line(dst, dst_offset, src, src_offset, n);
376         count += n;
377         src_offset += n;
378         dst_offset += n;
379     }
380 }
```

C/C++ or SystemC

Automated path from C/C++ or SystemC
into technology optimized synthesizable RTL

High-Level
Synthesis

```
1 module timer (
2     input      clock,
3     input      resetn,
4     input [31:0] trans,
5     input [29:0] address,
6     input [31:0] bl,
7     input      we,
8     input      ce,
9     input [31:0] write_data,
10    output [31:0] read_data,
11    output [1:0] resp,
12    output      ready
13);
14
15 reg [31:0] timer_value;
16 reg [31:0] rd_reg;
17
18 //reg      ready_out = 1'b1;
19 reg      resp_out = 2'b00;
20
21 ready_gen #(1) rg (clock, resetn, ce, trans, ready_out);
22
23 assign ready      = ready_out;
24 assign resp       = resp_out;
25 assign read_data  = rd_reg;
26
27 always @(posedge clock or negedge resetn) begin
28     if (resetn == 1'b0) begin
29         timer_value <= 32'h00000000;
30     end else begin
31         timer_value <= timer_value + 32'h00000001;
32     end
33 end
34
35 always @(posedge clock or negedge resetn) begin
36     if (resetn == 1'b0) begin
37         rd_reg <= 32'h00000000;
38     end else begin
39         //if (trans[1] & ce) begin
40             if (ce & we) begin
41                 rd_reg <= timer_value;
42             end
43         end
44     end
45 end
46 endmodule
```

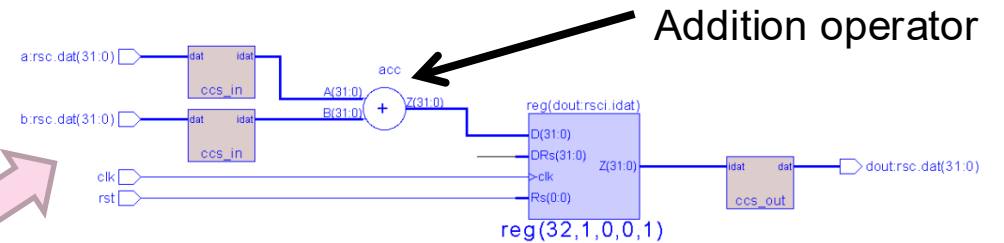
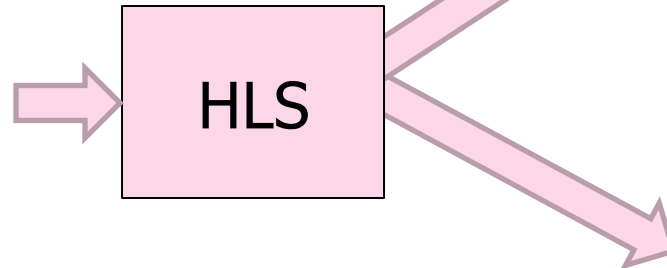
Synthesizable RTL

Generate Synthesizable RTL from C++

Optimized for a specific target technology or
FPGA device

Output in either VHDL or Verilog

```
void add(int a, int b, int &dout){  
    dout = a + b;  
}
```



```
16 module add_core (  
17     clk, rst, a_rsc_dat, b_rsc_dat, dout_rsc_dat  
18 );  
19     input clk;  
20     input rst;  
21     input [31:0] a_rsc_dat;  
22     input [31:0] b_rsc_dat;  
23     output [31:0] dout_rsc_dat;  
24  
25  
26     // Interconnect Declarations  
27     wire [31:0] a_rsci_idat;  
28     wire [31:0] b_rsci_idat;  
29     reg [31:0] dout_rsci_idat;  
30     wire [32:0] nl_dout_rsci_idat;  
31  
32     ...  
33     always @(posedge clk) begin  
34         if ( rst ) begin  
35             dout_rsci_idat <= 32'b0;  
36         end  
37         else begin  
38             dout_rsci_idat <= nl_dout_rsci_idat[31:0];  
39         end  
40         assign nl_dout_rsci_idat = a_rsci_idat + b_rsci_idat;  
41     endmodule
```

Clock and reset

Addition operator

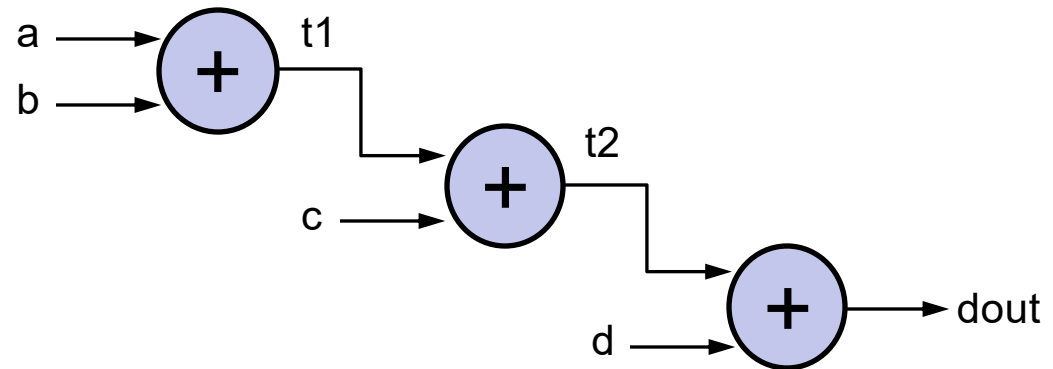
Analysis of C++ Descriptions

High-Level Synthesis analyzes the data dependencies between operations in the algorithm
Analysis produces a Data Flow Graph (DFG)

Each node on the DFG represents an operation in the algorithm

Connections between nodes represent data dependencies and indicate order of operations

```
void accumulate(int a, int b,  
               int c, int d,  
               int &dout) {  
    int t1, t2;  
    t1  = a  + b;  
    t2  = t1 + c;  
    dout = t2 + d;  
}
```



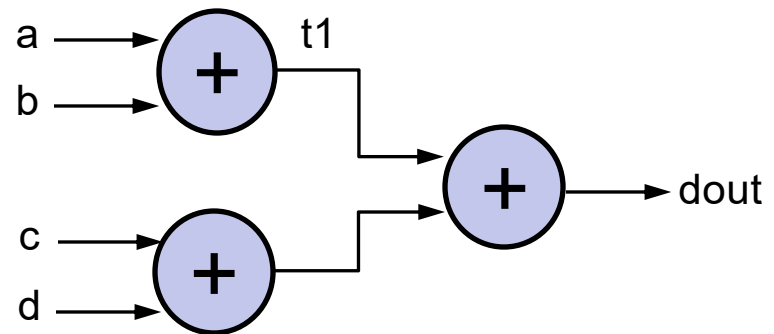
Analysis of C++ Descriptions

High-Level Synthesis analyzes the data dependencies between operations in the algorithm
Analysis produces a Data Flow Graph (DFG)

Each node on the DFG represents an operation in the algorithm

Connections between nodes represent data dependencies and indicate order of operations

```
void accumulate(int a, int b,  
               int c, int d,  
               int &dout) {  
    int t1, t2;  
    t1  = a  + b;  
    t2  = t1 + c;  
    dout = t2 + d;  
}
```



Parallelism

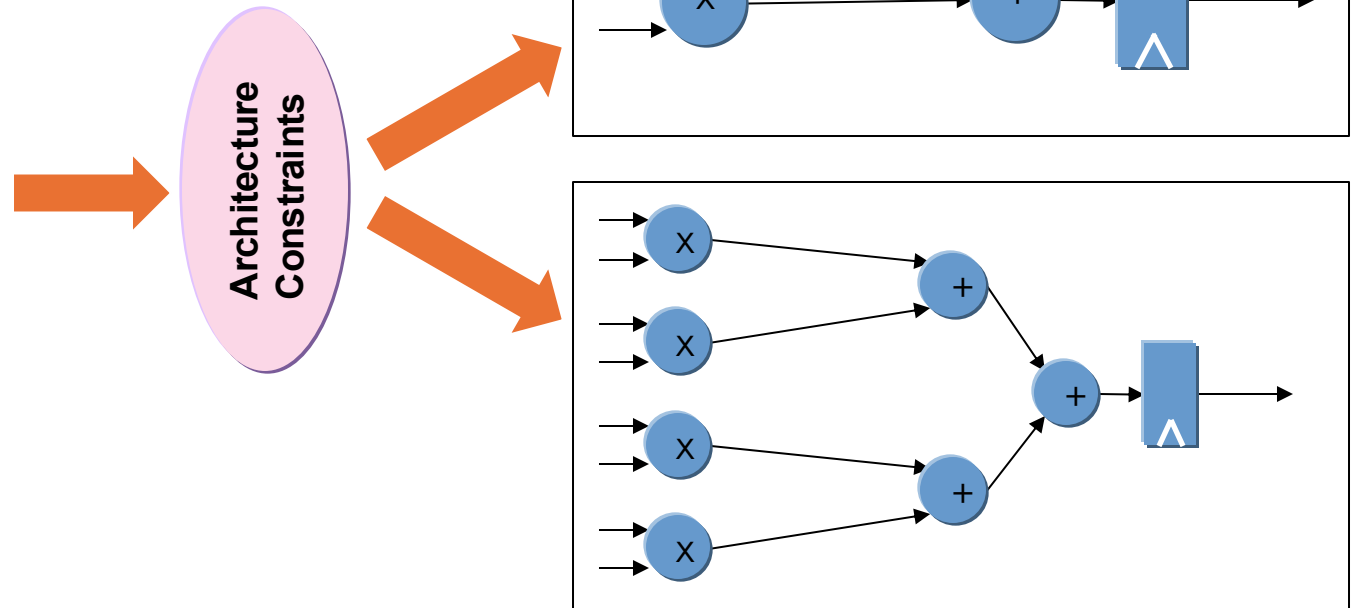
Parallelism is introduced using loop transformations

- Unrolling and pipelining

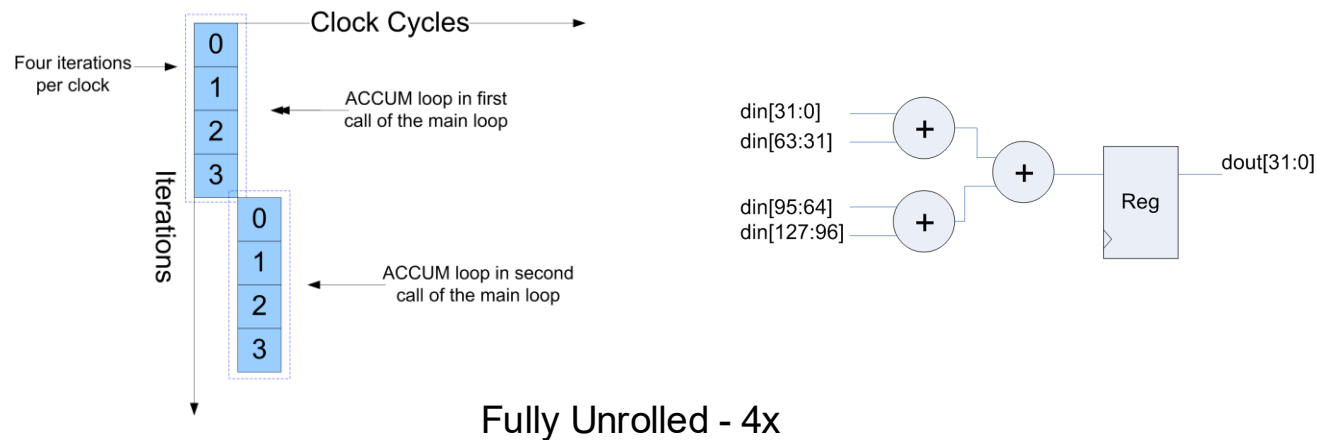
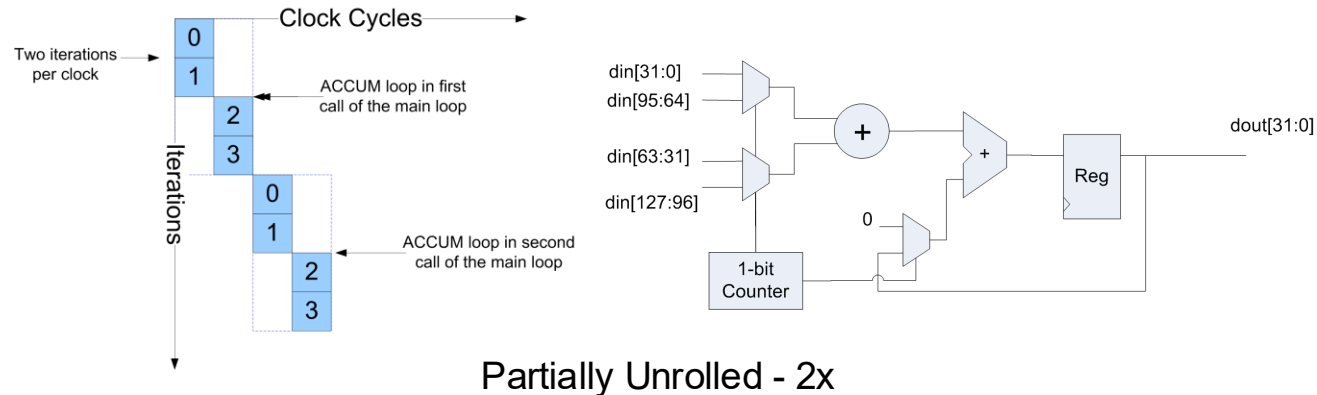
Unrolling drive parallelism

Pipelining also increases throughput and F_{\max}

```
data_t MAC (  
    data_t data_in[4],  
    coef_t coef_in[4]  
) {  
  
    accu_t acc = 0 ;  
  
    for (int i=0;i<4;i++) {  
        acc += data_in[i] * coef_in[i] ;  
    }  
    return acc ;  
}
```



Loop Unrolling



Loop styles

- “for...”
- “while...”
- “do ... while”

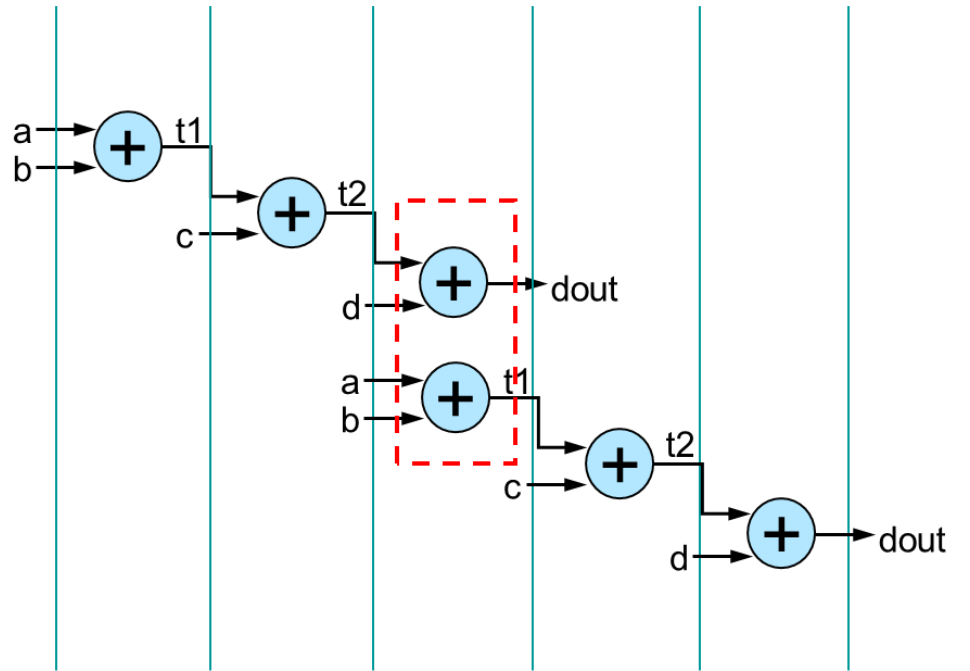
```
unsigned int sum_fn (int d[4])
{
    unsigned int sum = 0;
    for (int i=0; i<4; i++) sum += d[i];
    return sum;
}
```

Loop unrolling provides a way to explore several micro-architectures for a given design

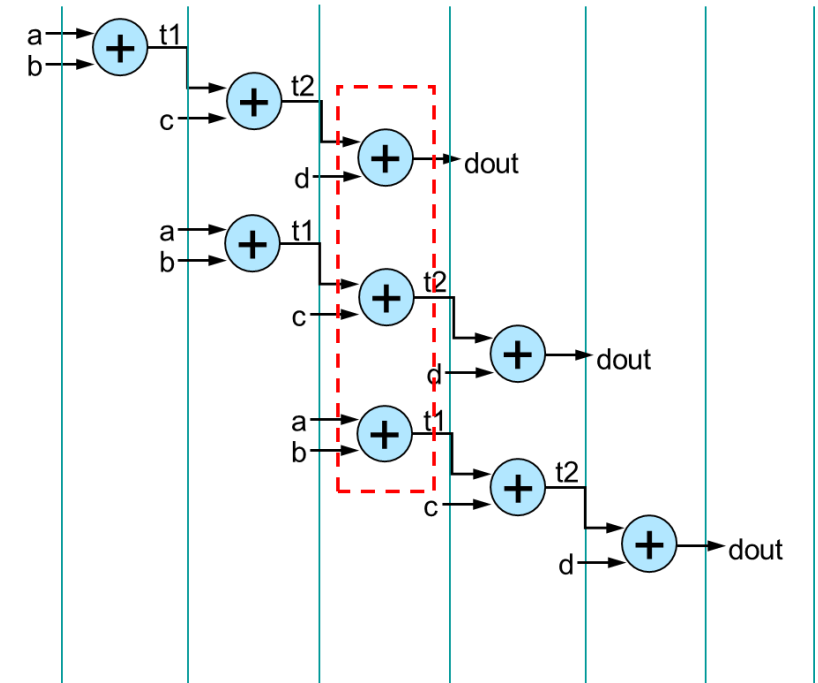
Loops can be fully or partially unrolled

Loop Pipelining

- A single stage pipeline, i.e. no pipelining, has no overlap between loop executions
- Results in data being written every 4 clock cycles
- With no overlap, the resources (the adder) can be shared between all C-Steps



Pipelining with II=2



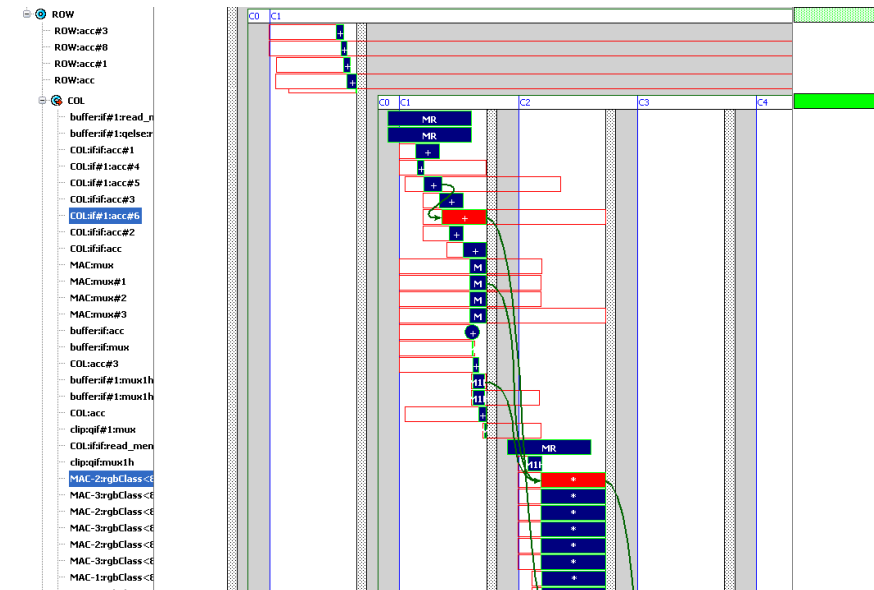
Pipelining with II=1

Pipelining or Loop Unrolling

What is the optimal architecture? What makes the most sense for your design?

Considerations:

- Data arrival and departure rates
 - Do not create more compute capacity than the communication channels can support
- Throughput vs. latency
 - Is it lower latency or greater throughput more important
- Performance vs. area
 - Smaller *usually* means slower
- HLS can give the data needed to make these decisions
 - Gantt Chart
 - Reports



Modeling Arbitrary Precision

Hardware design requires being able to specify any bit-width for variables, registers, etc.

Need to model true hardware behavior and precision to meet specification and save power/area

- Not limited to power-of-two bit-widths (1, 8, 16, 32, 64 bits)
- Integer, fixed-point, and floating-point support

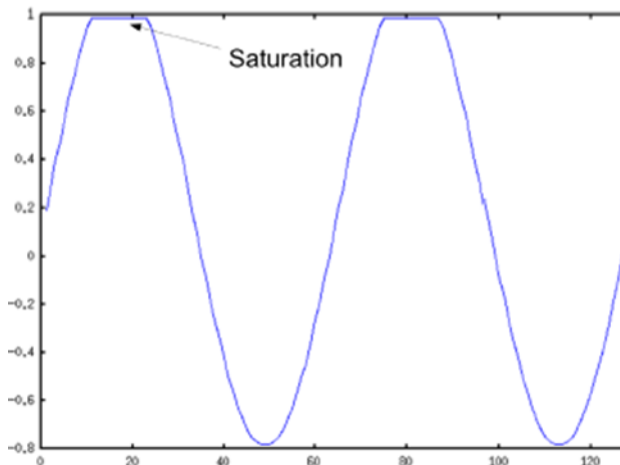
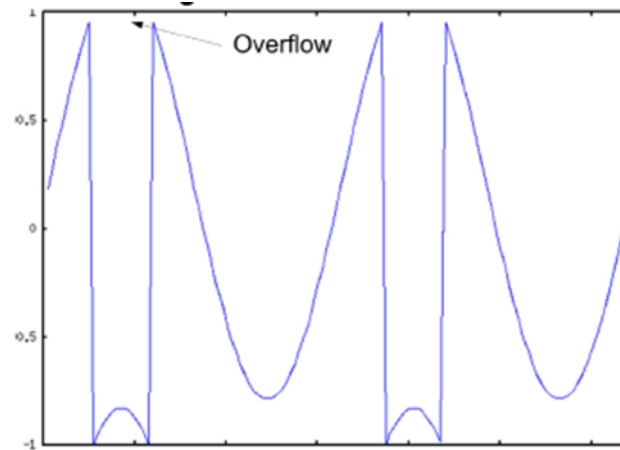
Algorithmic C (AC) data types are C++ classes defined to provide storage for precise hardware mapping in HLS



Saturating Math

```
#include <ac_fixed.h>
const double pi = 3.14;
const double OFFSET = 0.2;
int main(){
    fstream fptr;
    fptr.open("tmp.txt", fstream::out);
    ac_fixed<7,1,true,AC_TRN,AC_SAT> x[128];

    for(int i=0;i<128;i++){
        x[i] = OFFSET + 0.98*sin(2*pi*i/(double)64);
        fptr << x[i] <<endl;
    }
    fptr.close();
}
```



Overflow:

62.5	0 1 1 1 1 1 1 . 1 0 0
+ 2.0	1 0 . 0 0 0
<hr/>	
- 1.5	1 1 1 1 1 0 1 . 1 0 0

REALLY WRONG!

Saturation:

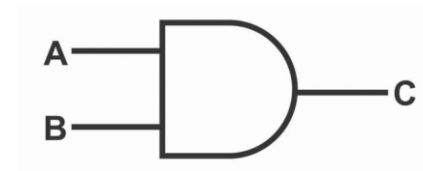
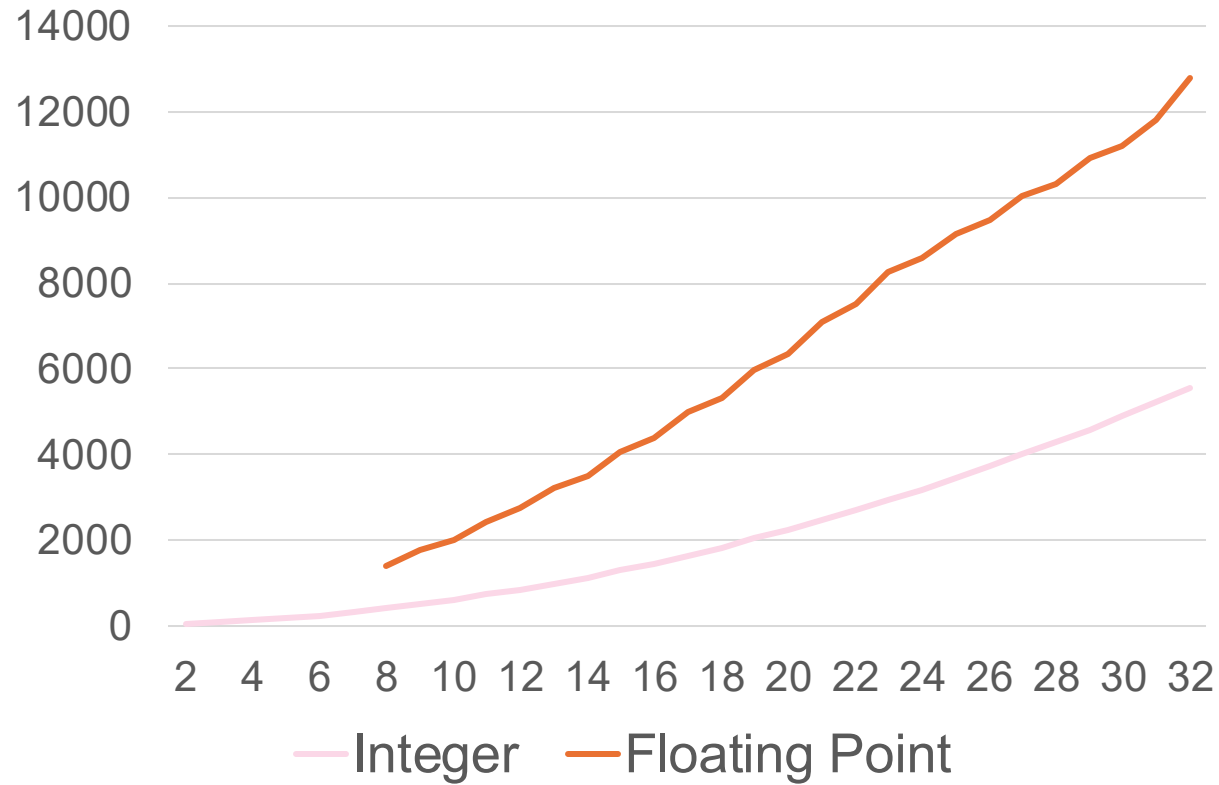
62.5	0 1 1 1 1 1 1 . 1 0 0
+ 2.0	1 0 . 0 0 0
<hr/>	
63.875	0 1 1 1 1 1 1 . 1 1 1

Close to correct



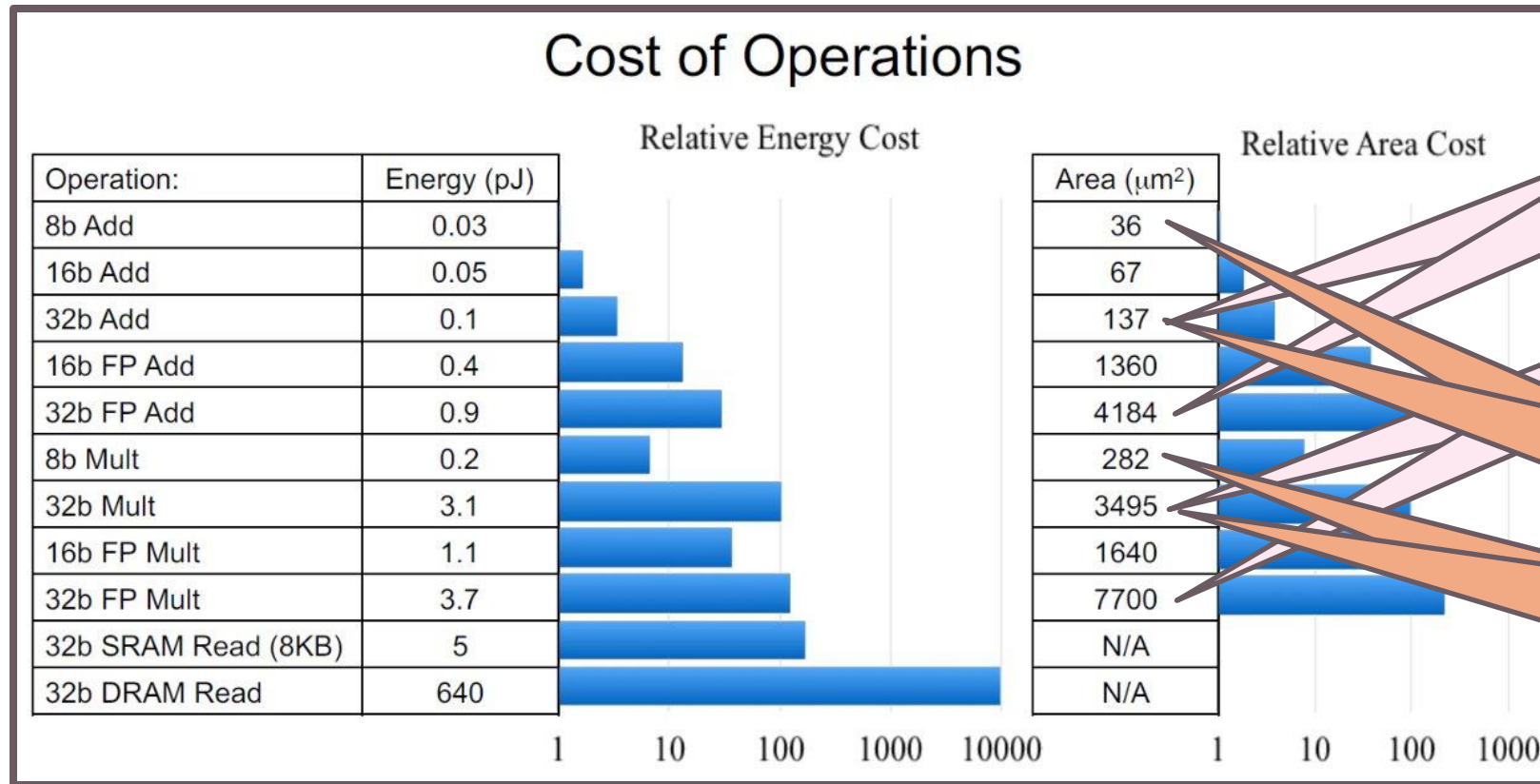
Smaller is Better

Operand Size vs Multiplier Area



A one-bit integer multiplier is an
“and” gate

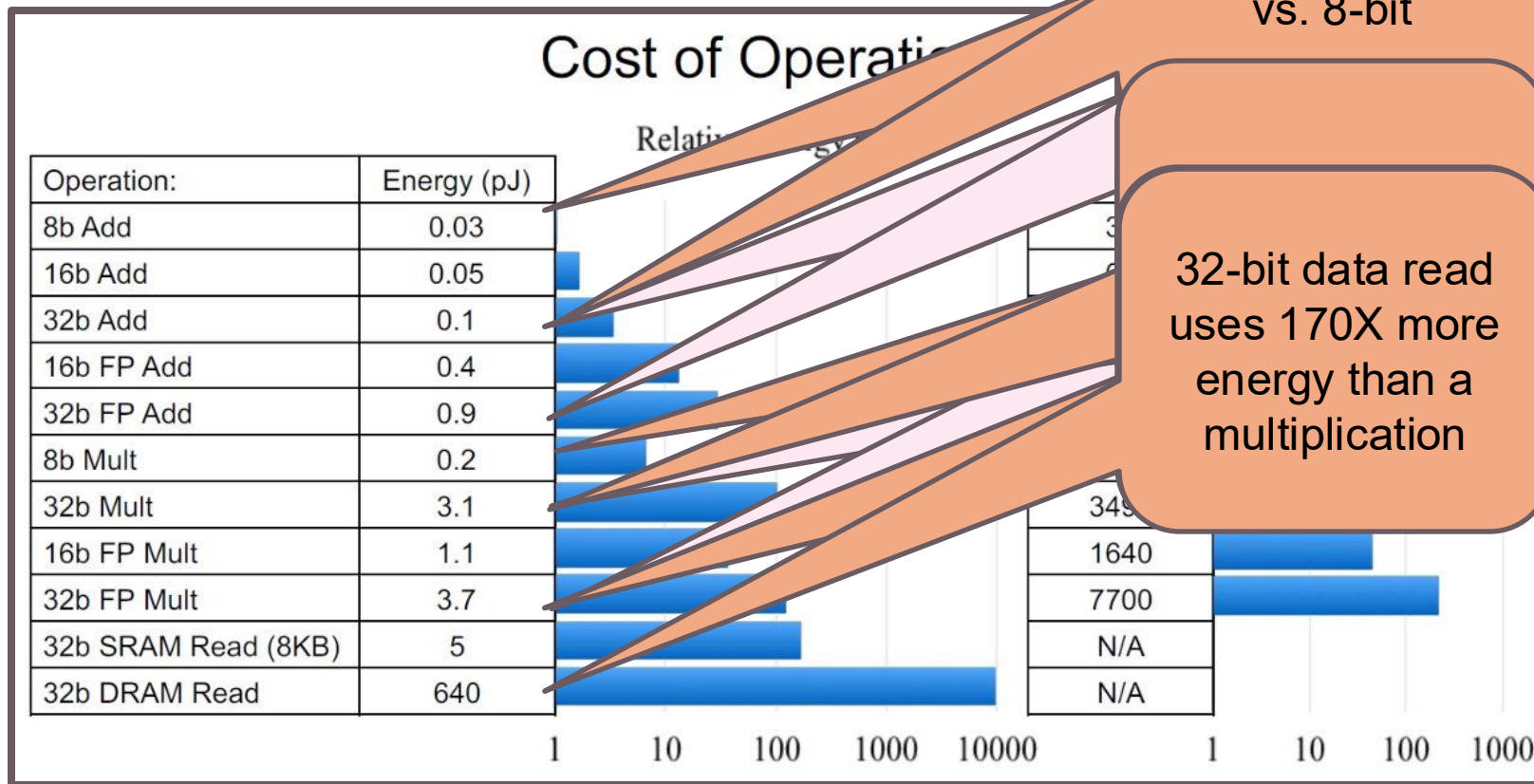
Data Sizes and Operators



Source: Nvidia DAC2017



Energy and Operators



Source: Nvidia DAC2017

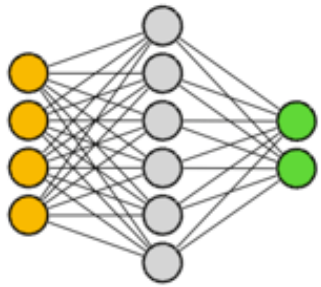


Benefits of High-Level Synthesis

High-Level Synthesis can help make this process easier, quicker, and flexible

Exploration through design constraints and synthesis settings, not manual re-coding

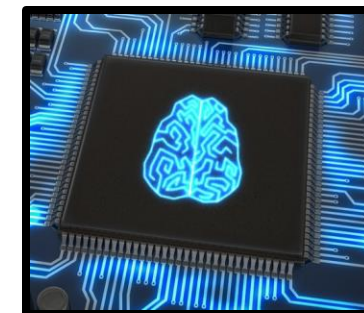
- Evaluate more options than possible with a manual RTL design process
- Automated path from C/C++ or SystemC into technology optimized synthesizable RTL



High-Level
Synthesis

```
1 module counter_1
2   input clk;
3   input reset;
4   input [1:0] count;
5   input [1:0] mode;
6   output [1:0] out;
7   output [1:0] out_data;
8   output [1:0] out_data;
9   output [1:0] ready;
10  output [1:0] ready;
11
12  //reg [1:0] count_value;
13  reg [1:0] out_reg;
14
15  //reg [1:0] count_value;
16  reg [1:0] out_reg;
17  reg [1:0] out_reg;
18  reg [1:0] out_reg;
19  reg [1:0] out_reg;
20  reg [1:0] out_reg;
21  reg [1:0] out_reg;
22  reg [1:0] out_reg;
23  reg [1:0] out_reg;
24  reg [1:0] out_reg;
25  reg [1:0] out_reg;
26  reg [1:0] out_reg;
27  reg [1:0] out_reg;
28  reg [1:0] out_reg;
29  reg [1:0] out_reg;
30  reg [1:0] out_reg;
31  reg [1:0] out_reg;
32  reg [1:0] out_reg;
33  reg [1:0] out_reg;
34  reg [1:0] out_reg;
35  reg [1:0] out_reg;
36  reg [1:0] out_reg;
37  reg [1:0] out_reg;
38  reg [1:0] out_reg;
39  reg [1:0] out_reg;
40  reg [1:0] out_reg;
41  reg [1:0] out_reg;
42  reg [1:0] out_reg;
43  reg [1:0] out_reg;
44  reg [1:0] out_reg;
45  reg [1:0] out_reg;
46  reg [1:0] out_reg;
47  reg [1:0] out_reg;
48  reg [1:0] out_reg;
49  reg [1:0] out_reg;
50  reg [1:0] out_reg;
51  reg [1:0] out_reg;
52  reg [1:0] out_reg;
53  reg [1:0] out_reg;
54  reg [1:0] out_reg;
55  reg [1:0] out_reg;
56  reg [1:0] out_reg;
57  reg [1:0] out_reg;
58  reg [1:0] out_reg;
59  reg [1:0] out_reg;
60  reg [1:0] out_reg;
61  reg [1:0] out_reg;
62  reg [1:0] out_reg;
63  reg [1:0] out_reg;
64  reg [1:0] out_reg;
65  reg [1:0] out_reg;
66  reg [1:0] out_reg;
67  reg [1:0] out_reg;
68  reg [1:0] out_reg;
69  reg [1:0] out_reg;
70  reg [1:0] out_reg;
71  reg [1:0] out_reg;
72  reg [1:0] out_reg;
73  reg [1:0] out_reg;
74  reg [1:0] out_reg;
75  reg [1:0] out_reg;
76  reg [1:0] out_reg;
77  reg [1:0] out_reg;
78  reg [1:0] out_reg;
79  reg [1:0] out_reg;
80  reg [1:0] out_reg;
81  reg [1:0] out_reg;
82  reg [1:0] out_reg;
83  reg [1:0] out_reg;
84  reg [1:0] out_reg;
85  reg [1:0] out_reg;
86  reg [1:0] out_reg;
87  reg [1:0] out_reg;
88  reg [1:0] out_reg;
89  reg [1:0] out_reg;
90  reg [1:0] out_reg;
91  reg [1:0] out_reg;
92  reg [1:0] out_reg;
93  reg [1:0] out_reg;
94  reg [1:0] out_reg;
95  reg [1:0] out_reg;
96  reg [1:0] out_reg;
97  reg [1:0] out_reg;
98  reg [1:0] out_reg;
99  reg [1:0] out_reg;
100 reg [1:0] out_reg;
```

Synthesizable RTL



Custom Hardware

Introduction to HLS4ML



SPONSORED BY



History of AI/ML Designs w/HLS

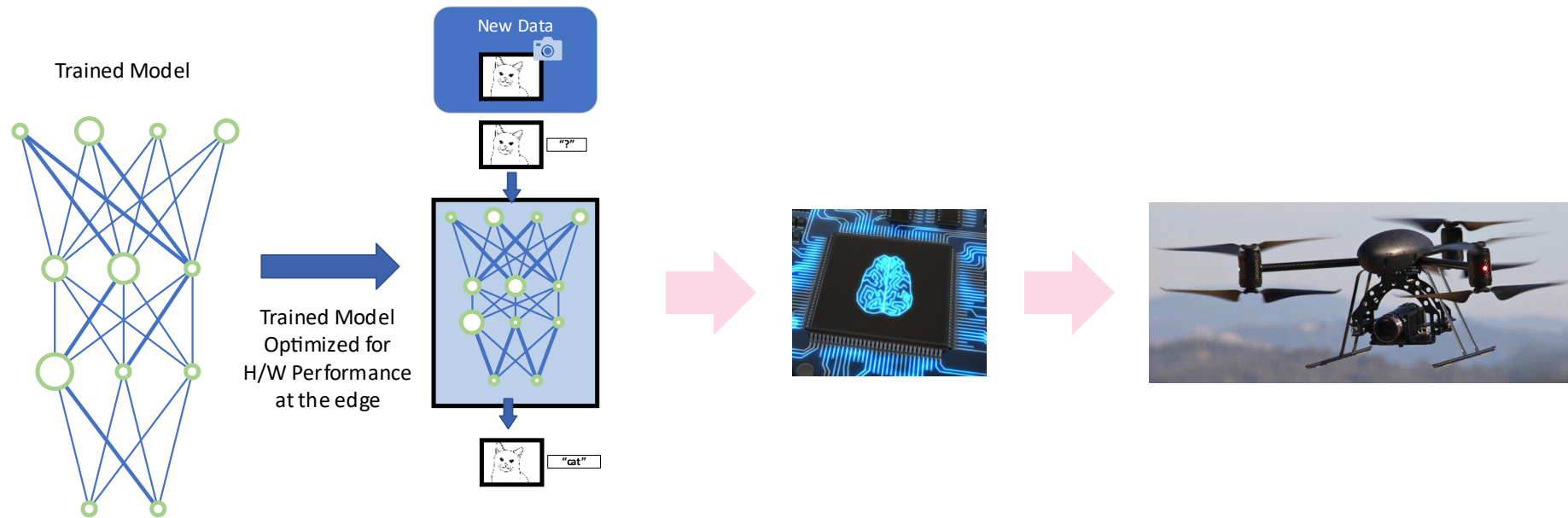
Customers have been using HLS for AI/ML designs since 2017

Mostly for **Convolutional Neural Networks** customized in ASIC for **Inferencing** at the edge

Manually optimized bit-widths for lowest area and power

Manually designed custom C++ IP for HLS and adjusted constraints to meet PPA target

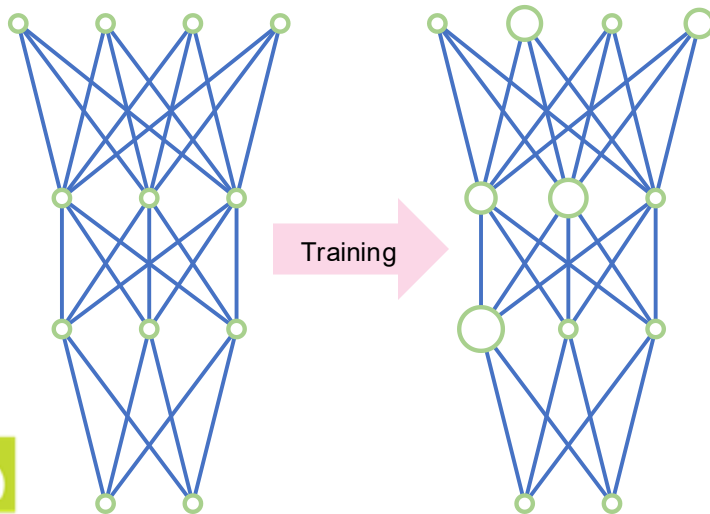
Mixture of pure dataflow layer connections and PE-Array architectures



Meeting designers where they are

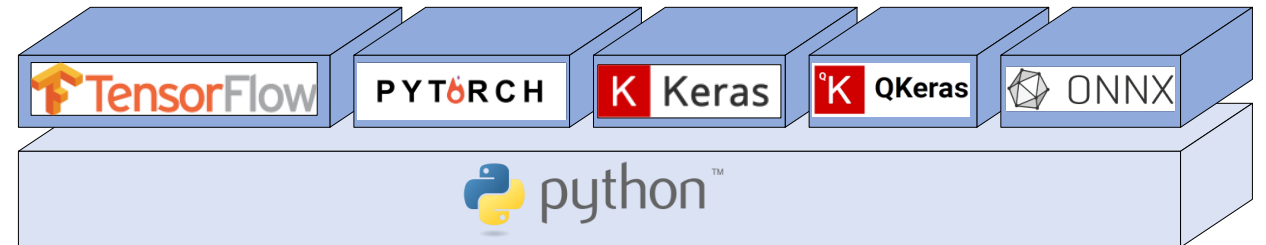
Motivation

- A Python env is the de facto standard development platform for AI/ML neural network models
- Generating an efficient hardware implementation from a Python model is tedious and error-prone
- Validation of the accuracy and PPA at the end is often too late
- Recent advances have allowed quantized-aware training using the Python model...
 - ... but those precision details must be manually (re)coded into HDL model



```
[8]: test_loss, test_acc = model.evaluate(x_test, y_test)
      print(f"Test Accuracy: {test_acc}")

313/313 [=====] - 1s 2ms/step - loss: 0.0858 - accuracy: 0.9723
Test Accuracy: 0.9722999930381775
```



Introduction

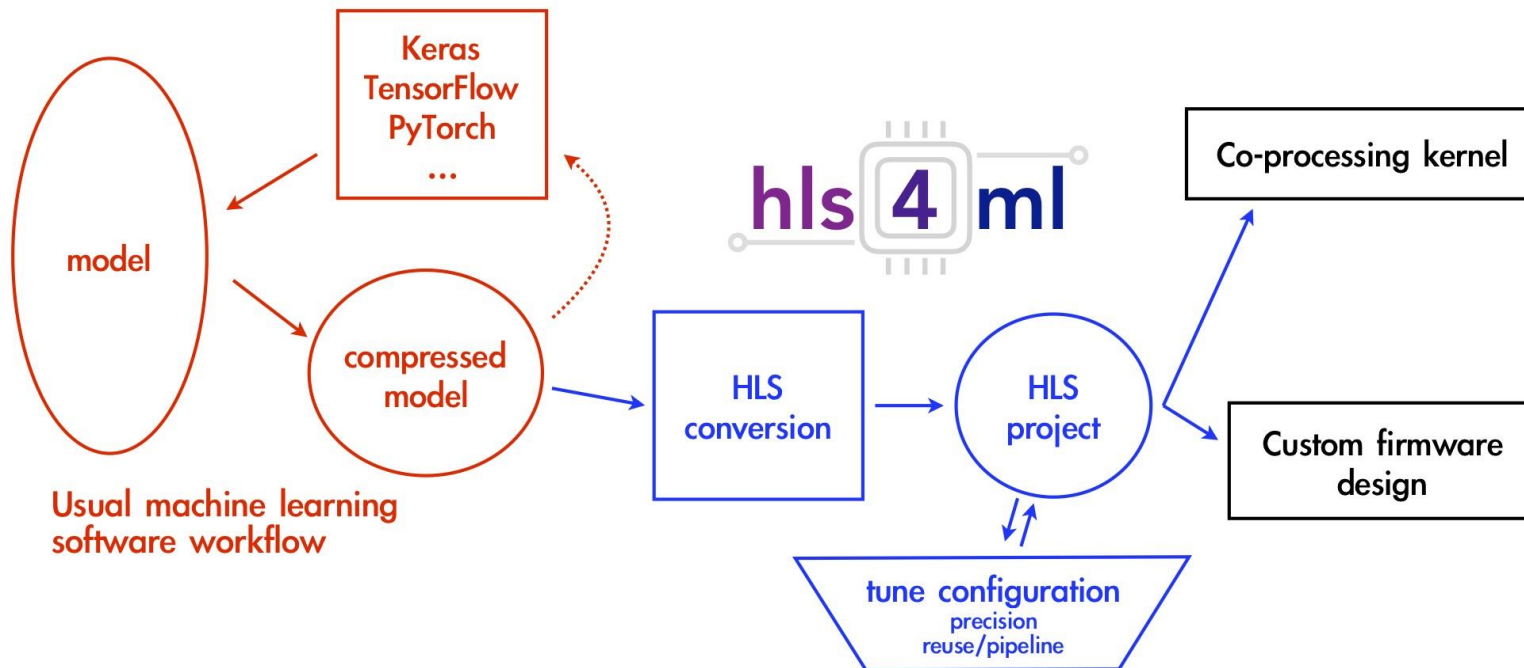
- Provide an efficient and fast translation of machine learning models from open-source packages for training machine learning algorithms to High-Level Synthesis

Inspiration

- Originally inspired by the CERN Large Hadron Collider (LHC)
- ML applications have proven extremely useful for large dataset analysis.
- Taking data offline will allow for data to be calculated faster along with sorting data for storage
- Lower Latency, Realtime Detections



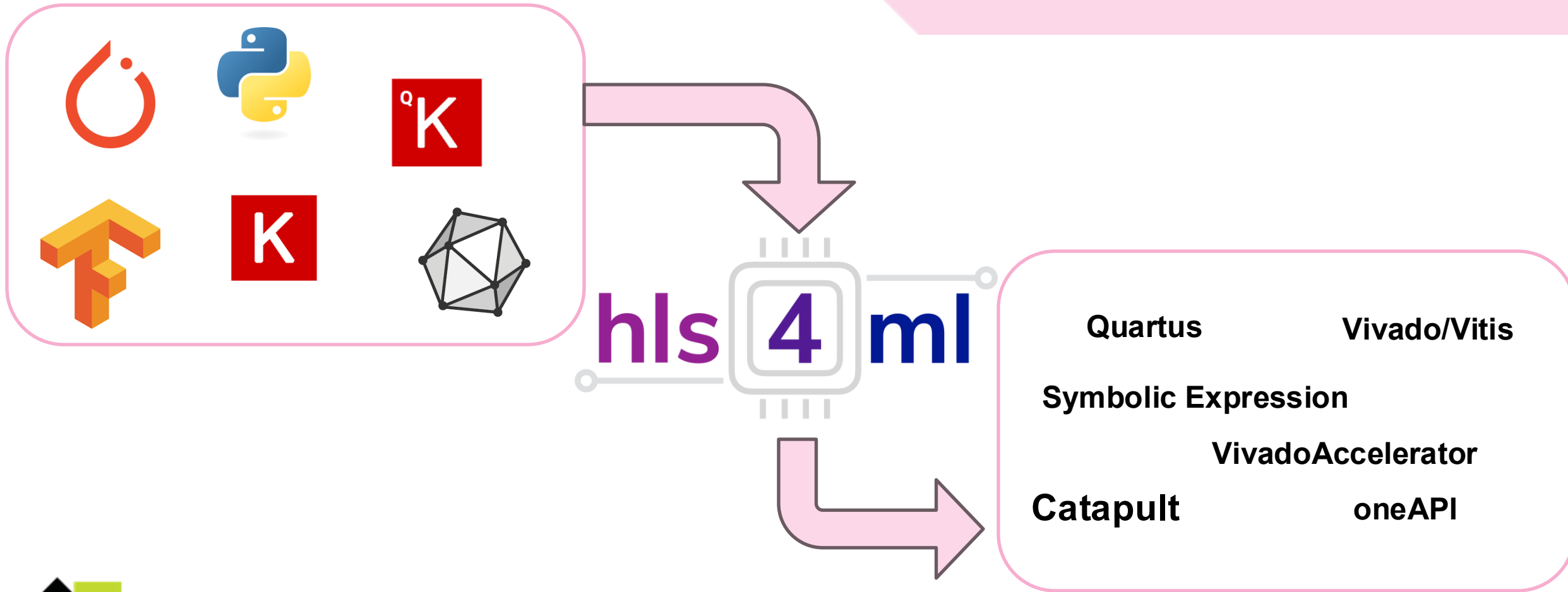
HLS4ML



Solution:

- ASIC and FPGAs have specialized architecture compared to CPUs and GPUs
- Specialized hardware is always able to help with design constraints
- Specialized hardware tend to have lower-power and faster results.

Frontends & Backends



The Full Flow

```
#####  
# MNIST CNN definition  
#####  
#  
# Can be modified to increase or decrease number of layer, number of channels  
# supported layers are Conv2D, Dense, and Flatten. Supported kernel sizes are  
# 3, 5, and 7, square kernels only  
#####  
  
def mnist_model():  
    # create model  
    model = Sequential()  
    model.add(Input(shape=(28,28,1)))  
    model.add(Conv2D(20, (5,5), use_bias=True, padding='same', activation='relu'))  
    model.add(MaxPooling2D(pool_size=(2,2)))  
    # was 10  
    model.add(Conv2D(1, (3,3), use_bias=True, padding='same', activation='relu'))  
    model.add(MaxPooling2D(pool_size=(2,2)))  
    model.add(Flatten())  
    model.add(Dense(500, use_bias=True, kernel_initializer='normal', activation='relu'))  
    model.add(Dense(20, use_bias=True, kernel_initializer='normal', activation='relu'))  
    model.add(Dense(10, use_bias=True, kernel_initializer='normal', activation='softmax'))  
    # Compile model  
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])  
    return model
```

Python



```
361 void copy_to_regs(hw_cat_type *dst, index_type dst_offset, raw_memory_line *src, index_type src_offset, index_type size)  
362 {  
363     // read out of internal memories to an array of registers  
364     // *should* make it easy for catapult to pipeline access to internal memories  
365  
366     index_type count;  
367     index_type n;  
368  
369     static const index_type stride = STRIDE;  
370  
371     count = 0;  
372     while (count < size) {  
373         n = stride;  
374         if ((size - count) < stride) n = size - count; // mis-aligned at the end of transfer  
375         read_line(dst, dst_offset, src, src_offset, n);  
376         count += n;  
377         src_offset += n;  
378         dst_offset += n;  
379     }  
380 }  
381
```

C/C++

High-Level
Synthesis

```
1 module timer (  
2     input      clock,  
3     input      resetn,  
4     input [31:0] trans,  
5     input [20:0] address,  
6     input [31:0] bl,  
7     input      we,  
8     input      ce,  
9     input [31:0] write_data,  
10    output [31:0] read_data,  
11    output [31:0] resp,  
12    output      ready  
13 );  
14  
15 reg [31:0] timer_value;  
16 reg [31:0] rd_reg;  
17  
18 //reg      ready_out = 1'b1;  
19 reg      resp_out = 2'b00;  
20  
21 ready_gen #(1) rg (clock, resetn, ce, trans, ready_out);  
22  
23 assign ready      = ready_out;  
24 assign resp      = resp_out;  
25 assign read_data  = rd_reg;  
26  
27 always @(posedge clock or negedge resetn) begin  
28     if (resetn == 1'b0) begin  
29         timer_value = 32'h00000000;  
30     end else begin  
31         timer_value = timer_value + 32'h00000001;  
32     end  
33 end  
34  
35 always @(posedge clock or negedge resetn) begin  
36     if (resetn == 1'b0) begin  
37         rd_reg = 32'h00000000;  
38     end else begin  
39         //if (trans[1] & ce) begin  
40             if (ce & !we) begin  
41                 rd_reg = timer_value;  
42             end  
43         end  
44     end  
45 end  
46 endmodule
```

Synthesizable RTL



An Example



SPONSORED BY



MNIST Dataset

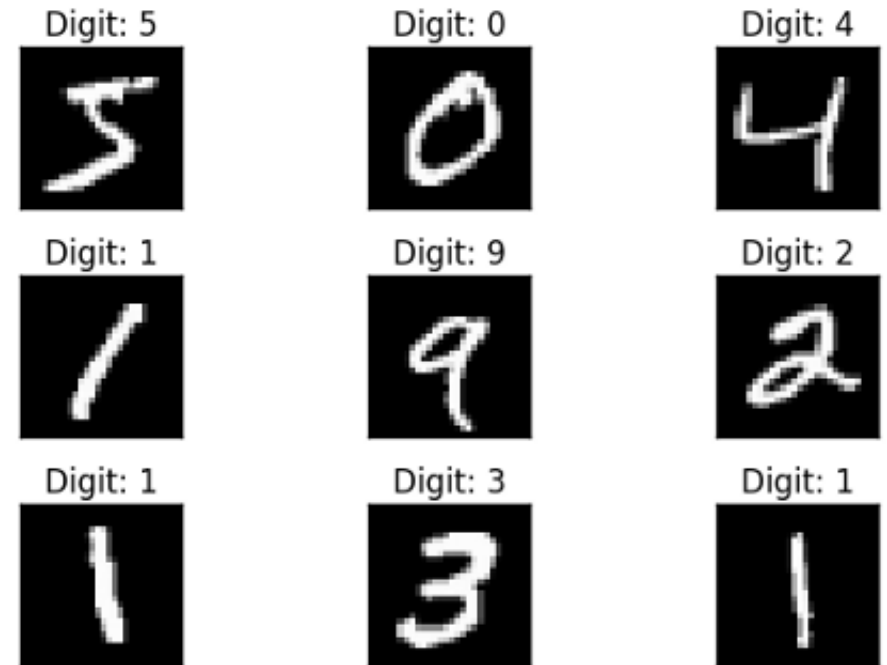
The MNIST dataset is included in several popular machine learning packages

Contains 70,000 images:

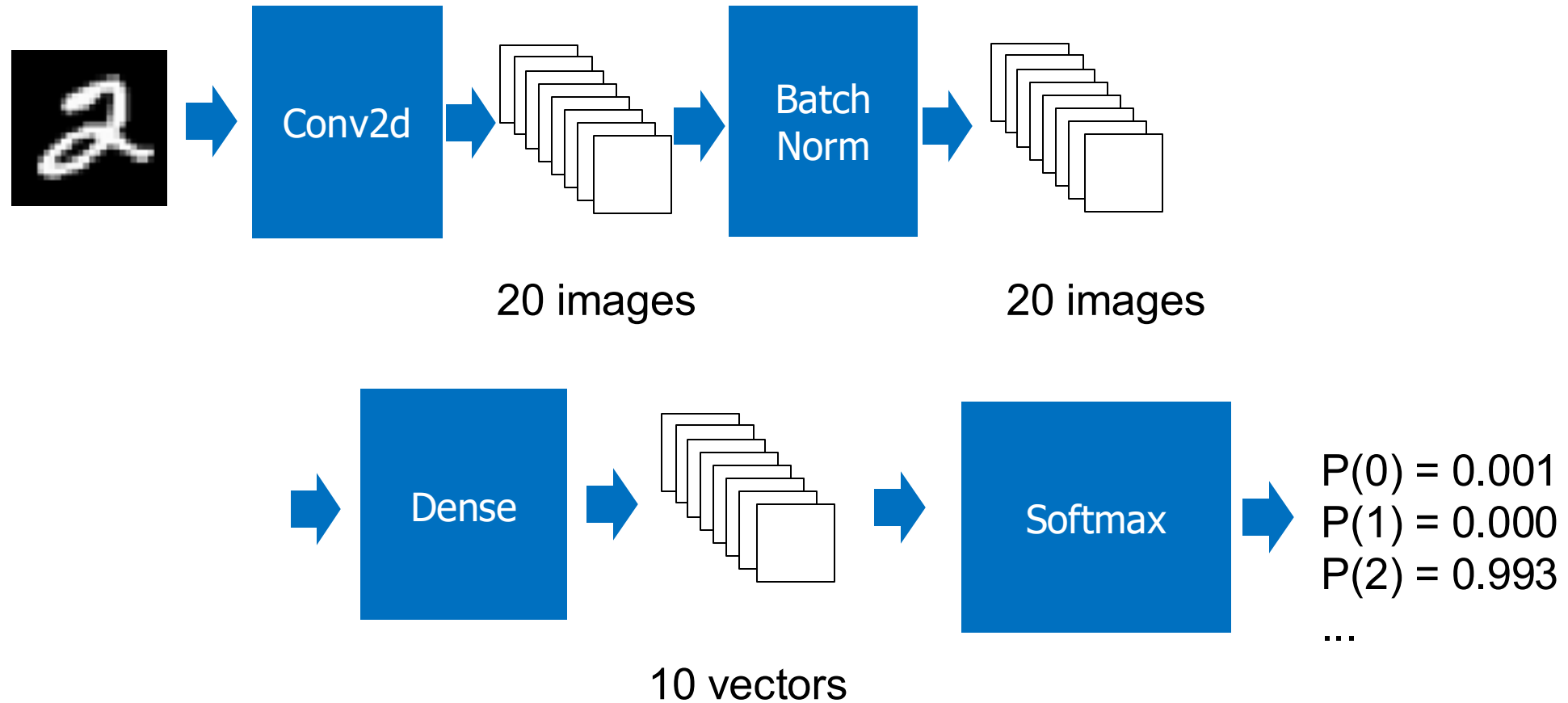
- Images are 28 x 28 pixels
- Pixels are 8-bit greyscale (1 color plane)

Typically separated training and validation:

- 60,000 images for training
- 10,000 images for verification



MNIST Neural Network



Accelerator Development

* System performance and power measured for 64-bit Rocket Core RISC-V

Profile the execution to determine functions that need acceleration

Software Profile

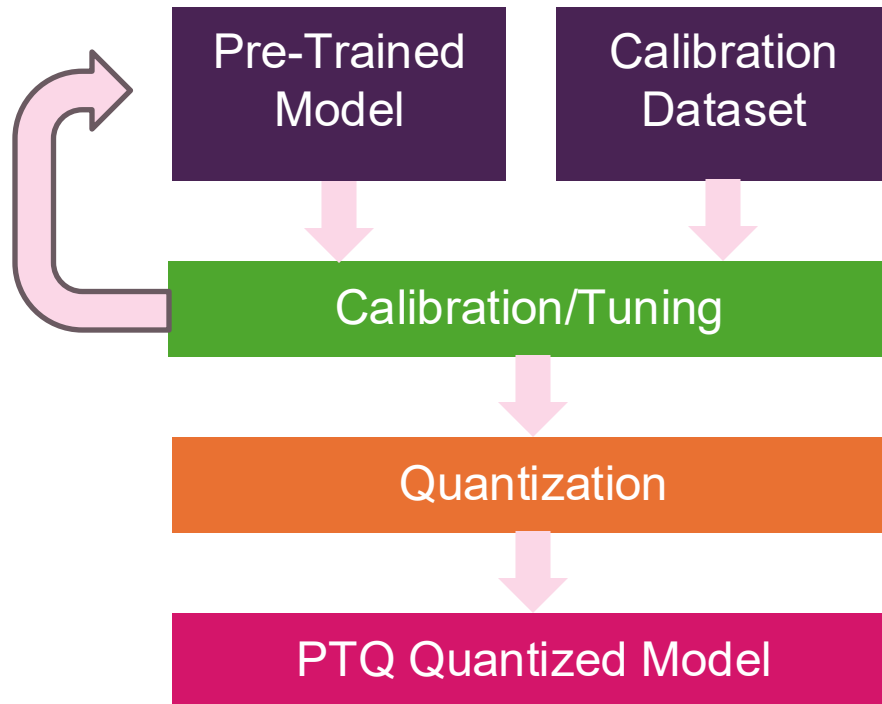
	Weight	Self Weight	Symbol Name
1995.00 ms	100.0%		mnist(85781)
1995.00 ms		33.00 ms	Main Thread 0x1af672
1962.00 ms		0 s	start
1962.00 ms		0 s	main (int, char *)
1962.00 ms		210.00 ms	test_mnist(int, float*, float*)
1752.00 ms	100.0%	70.00 ms	sw_inference(float*, float *, float*)
1682.00 ms	100.0%	4.00 ms	load_image(int, float*)
1678.00 ms	100.0%	1.00 ms	load_weights(int, float*)
1677.00 ms	100.0%	18.00 ms	sw_auto_infer(int, float *, float*)
922.00 ms	55.3%	922.00 ms	dense_sw(float*, float*, float *, float*, int, int, int, int)
738.00 ms	44.2%	738.00 ms	conv2d_sw(float*, float*, float*, float*, float*, int, int, int, int, int, int)
35.00 ms	0.5%	3.00 ms	softmax(int, float*)
17.00 ms		2.00 ms	check_results(int, float*, float*)
15.00 ms		15.0 ms	exit()

Convolution and dense layers consume 99.5% of the computational load (excluding test overhead)
These will benefit from acceleration

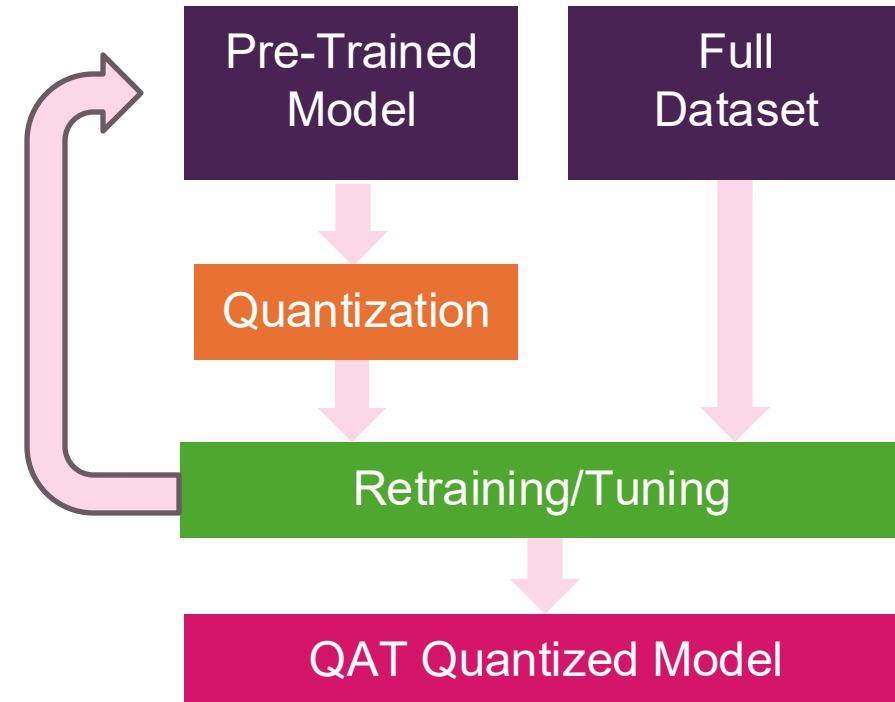


Feature and Weight Quantization

Post Training Quantization



Quantized Aware training



Higher levels of abstraction

Catapult AI NN has a simplified Python API for configuring the project and generating the RTL

- Use `config_for_dataflow` to configure the project – using only the model and dataset variables
- Use `generate_dataflow` to generate the Catapult HLS C++ model, C++ testbench and build scripts
- Use `build` to generate the RTL

```
# Configure the project - passing in the TF model, test dataset and reference output
config_ccs = catapult_ai_nn.config_for_dataflow(model=model, x_test=x_test, y_test=y_test, num_samples=50, tech='asic',
                                                asiclibs='saed32rvt_tt0p78v125c_beh', clock_period=10, io_type='io_stream')

# Generate the C++ HLS model
hls_model_ccs = catapult_ai_nn.generate_dataflow(model=model, config_ccs=config_ccs)

# Use Catapult Ultra to generate the RTL (batch mode)
Hls_model_ccs.build()
```

This example is available using the Catapult AI/NN Frontend for HLS4ML



Reports

Layer Report:

- *HLS4ML Layer Summary* – report shows python description of each layer
- *nnet layer results* – report shows PPA for each network layer

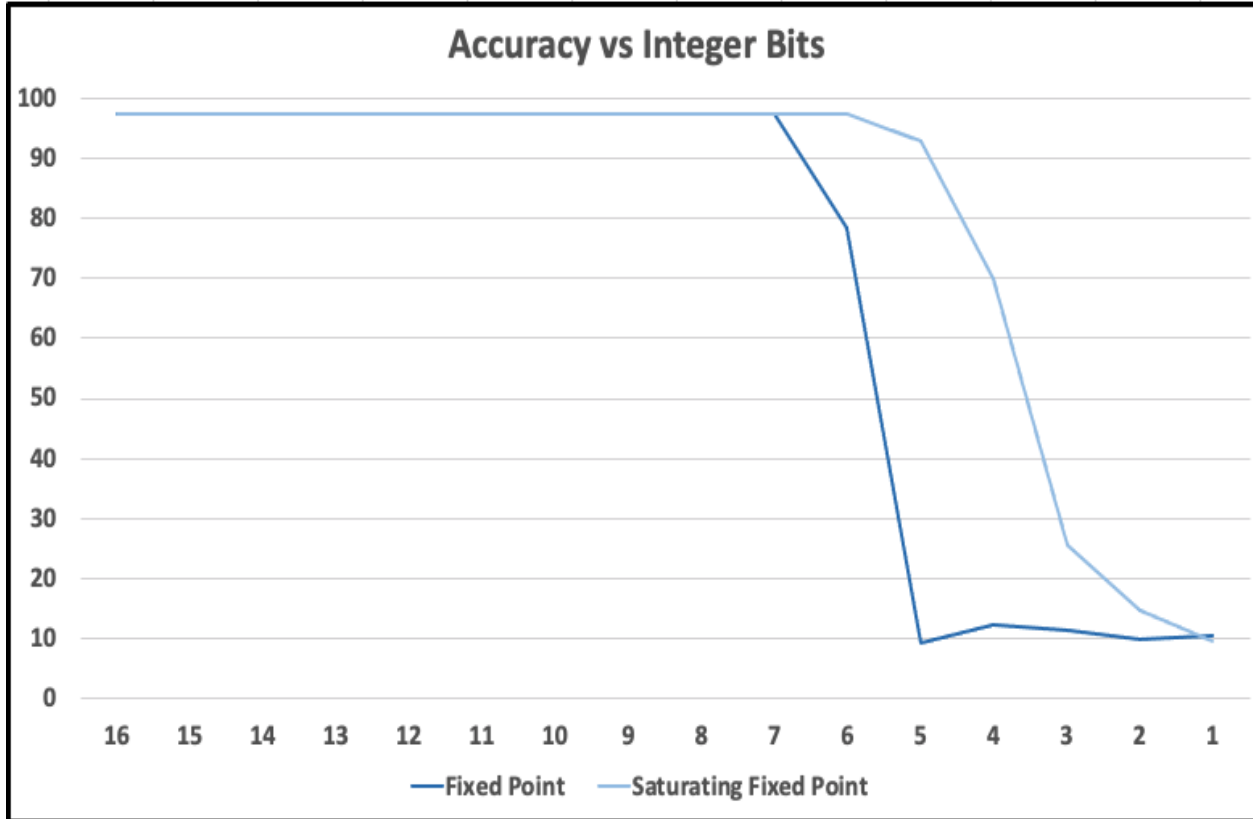
Layer Name	Layer Class	Input Type	Input Shape	Output Type	Output Shape
conv2d1	Conv2D	ac_fixed<8,1,true>	[14][14][1]	ac_fixed<16,6,true>	[4][4][5]
relu1	relu	ac_fixed<16,6,true>	[4][4][5]	ac_fixed<16,6,true>	[4][4][5]
flatten1	Reshape	ac_fixed<16,6,true>	[4][4][5]	ac_fixed<16,6,true>	[80]
dense1	Dense	ac_fixed<16,6,true>	[80]	ac_fixed<16,6,true>	[10]
softmax1	Softmax	ac_fixed<16,6,true>	[10]	ac_fixed<16,6,true>	[10]

Weight Type	Bias Type
ac_fixed<16,6,true>	ac_fixed<16,6,true>
ac_fixed<16,6,true>	ac_fixed<16,6,true>

This report is available using the Catapult AI/NN Frontend for HLS4ML



Understanding Precision



High-water mark of data and intermediate values showed range of values was -37 to 56

- Float32 ($\pm 10^{38}$ is excessive)

Sensitivity analysis performed across varying fixed-point representations

Value Range Analysis

For this example, a fixed-point precision of `ac_fixed<16,6>` resulted in 3 numerically different results compared to the floating-point Python output (after quantization)

```
catapult_ai_nn.run_testbench(hls_model_ccs,0.005)
```

```
Weights directory: ./firmware/weights
Test Feature Data: ./tb_data/tb_input_features.dat
Test Predictions : ./tb_data/tb_output_predictions.dat
Processing input 0
Predictions
0 0 1.5e-05 9.2e-05 0 1e-06 0 0.99989 1e-06 1e-06
Quantized predictions
0 0 0 0 0 0 .9990234375 0 0
Ref 0.885848 Ref(quantized) .8857421875 DUT 0.879883 <- MISMATCH
Ref 0.379353 Ref(quantized) .37890625 DUT 0.366211 <- MISMATCH
Ref 0.616644 Ref(quantized) .6162109375 DUT 0.628906 <- MISMATCH
INFO: Saved inference results to file: tb_data/csim_results.log
Error: A total of 3 differences detected between golden Python prediction and C++ prediction using threshold of 0.005
```

```
from sklearn.metrics import accuracy_score
print('Python Model Accuracy : {}'.format(accuracy_score))
print('C++ Model Accuracy : {}'.format(accuracy_score))
```

```
313/313 [=====] - 0s 988us/step
Python Model Accuracy : 0.9576
C++ Model Accuracy : 0.9497
```



This tool is available using the Catapult AI/NN Frontend for HLS4ML

Customization

Add refinements by layer

```
# Configure the project - passing in the TF model, test dataset and reference output
config_ccs = catapult_ai_nn.config_for_dataflow(model=model, x_test=x_test, y_test=y_test, num_samples=50, tech='asic',
                                                asiclibs='saed32rvt_tt0p78v125c_beh', clock_period=10, io_type='io_stream')

# Refinements per layer - Precision
config_ccs['HLSConfig']['LayerName']['input1']['Precision'] = 'ac_fixed<8,1,true>'
```

Measuring the accuracy of this model shows a slight improvement

Python Model Accuracy : 0.9576
C++ Model Accuracy : 0.9497

AREA SCORE: 70125



Python Model Accuracy : 0.9576
C++ Model Accuracy : 0.9498

AREA SCORE: 72275

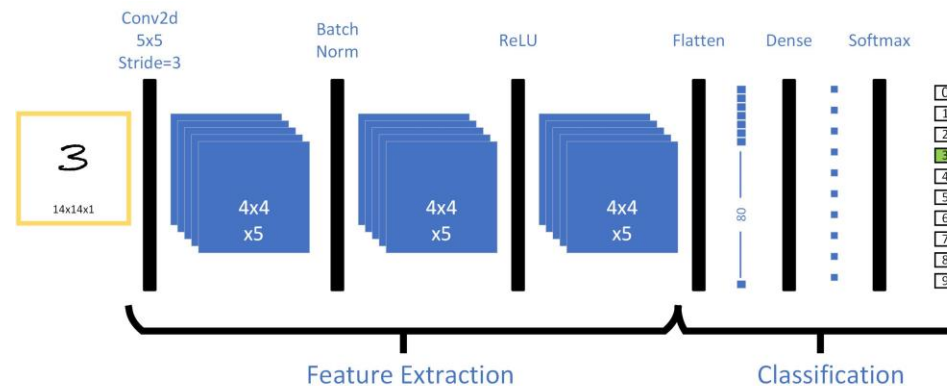
Does the accuracy increase of 0.0001 warrant and increase in size?



Rethinking the Approach - QAT

Going back to the Python model, you can use QKeras to model the quantization affects at the interfaces of the layers during training

Note that even though QKeras is applying quantization at the interfaces (feature, weights and biases), the internal math operations are still performed as double precision whereas the fixed-point C++ model will use bit-precise fixed-point operations



Transferring Your Network

```
model = Sequential()
model.add(layers.Input(shape=(Fw,Fw, 1), name='input1'))
model.add(layers.Conv2D(filters=5,
                        kernel_size=5, strides=3, name='conv2d1'))
model.add(layers.BatchNormalization(name='batchnorm1'))
model.add(layers.Activation('relu', name='relu1'))
model.add(layers.Flatten(name='flatten1'))
model.add(layers.Dense(10, name='dense1'))
model.add(layers.Activation('softmax', name='softmax1'))
```



Transferring Your Network

```
model = Sequential()
model.add(layers.Input(shape=(Fw,Fw, 1), name='input1'))
model.add(QConv2D(filters=5, kernel_size=5, strides=3,
                  kernel_quantizer=quantized_bits(8, 1, 1, alpha=1),
                  bias_quantizer=quantized_bits(8, 1, alpha=1),
                  name='conv2d1'))
model.add(layers.BatchNormalization(name='batchnorm1'))
model.add(layers.Activation('relu', name='relu1'))
model.add(layers.Flatten(name='flatten1'))
model.add(QDense(
    units=10,
    kernel_quantizer=quantized_bits(8, 1, alpha=1),
    bias_quantizer=quantized_bits(8, 1, alpha=1),
    kernel_regularizer=tf.keras.regularizers.L1L2(0.0001),
    activity_regularizer=tf.keras.regularizers.L2(0.0001),
    name='dense1',
))
model.add(layers.Activation('softmax', name='softmax1'))
```



Model Accuracy – Quantizer Bits

		Integer Bit								
Fractional Bits		8	7	6	5	4	3	2	1	0
	8	0.9557	0.9537	0.9583	0.9509	0.953	0.9421	0.907	0.8966	0.098
	7	0.9565	0.9552	0.9569	0.9576	0.9552	0.9459	0.941	0.9308	0.098
	6	0.9497	0.952	0.9556	0.9496	0.9579	0.9495	0.9469	0.9133	0.2298
	5	0.9608	0.957	0.9565	0.9532	0.952	0.9405	0.9238	0.9211	0.098
	4	0.9537	0.9567	0.9519	0.9605	0.9539	0.9492	0.9344	0.9016	0.5703
	3	0.9512	0.9549	0.9553	0.951	0.9513	0.9515	0.9408	0.9212	0.8202
	2	0.953	0.915	0.9559	0.9576	0.9555	0.9501	0.9413	0.9099	0.7048

Design Exploration and Optimizing

Conv 2D 5x5 filter	Model Accuracy	Area – u^2	Bias bits In ROM	Weight bits In ROM
8int 5p	0.9608	133255	65	1625
7int 4p	0.9567	115933	55	1375
7int 2p	0.915	99520	45	1125
4int 6p	0.9579	99550	50	1250
0int 3p	0.8202	37591	15	375

Discover the optimal design

- Make informed choices
- Find the smallest design with an optimal accuracy

Key Points

- As the number of bits decrease the size decreases
- The less bits moving through ROM the less energy used

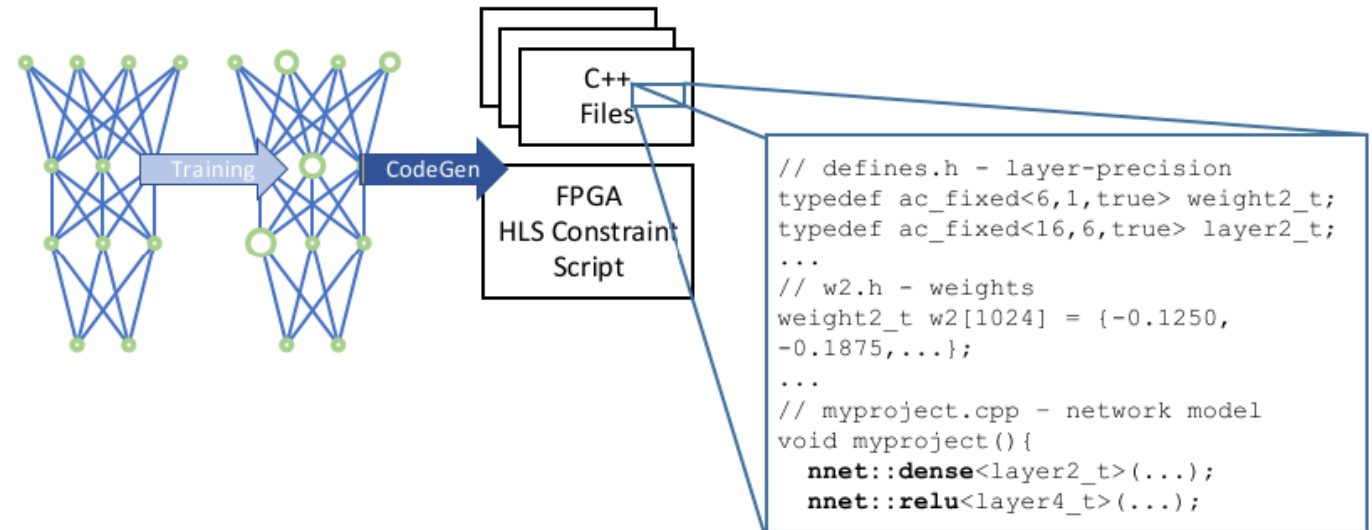
Dense 10 Ch	Model Accuracy	Area – u^2	Bias bit In ROM	Weight bits In ROM
8int 5p	0.9608	813888	130	10400
7int 4p	0.9567	703025	110	8800
7int 2p	0.915	597973	90	7200
4int 6p	0.9579	596609	100	8000
0int 3p	0.8202	200393	30	2400



Meeting designers where they are

Ease of Use and Optimization

- High-Performance C++ IP Libraries for better hardware
- Enhanced analysis and reporting
- Complete low-power design w/power estimation and optimization
- Integrated Value-Range Analysis (VRA) for detection of quantization/overflow in C++
- C++ Testbench options to measure numerical differences vs Python



This example is available using the Catapult AI/NN Frontend for HLS4ML

How can we use HLS4ML to make our lives easier



SPONSORED BY



What is hls4ml?

- **hls4ml** is a Python package for machine learning inference as custom hardware
 - Translate traditional open-source ML models into an HLS project
- Easy to install
 - `pip install hls4ml`
- Open source
 - <https://github.com/fastmachinelearning/hls4ml>
 - <https://fastmachinelearning.org/hls4ml>
- Community
 - Research laboratories, universities, and companies

SIEMENS EDA

Catapult® Synthesis Release Notes

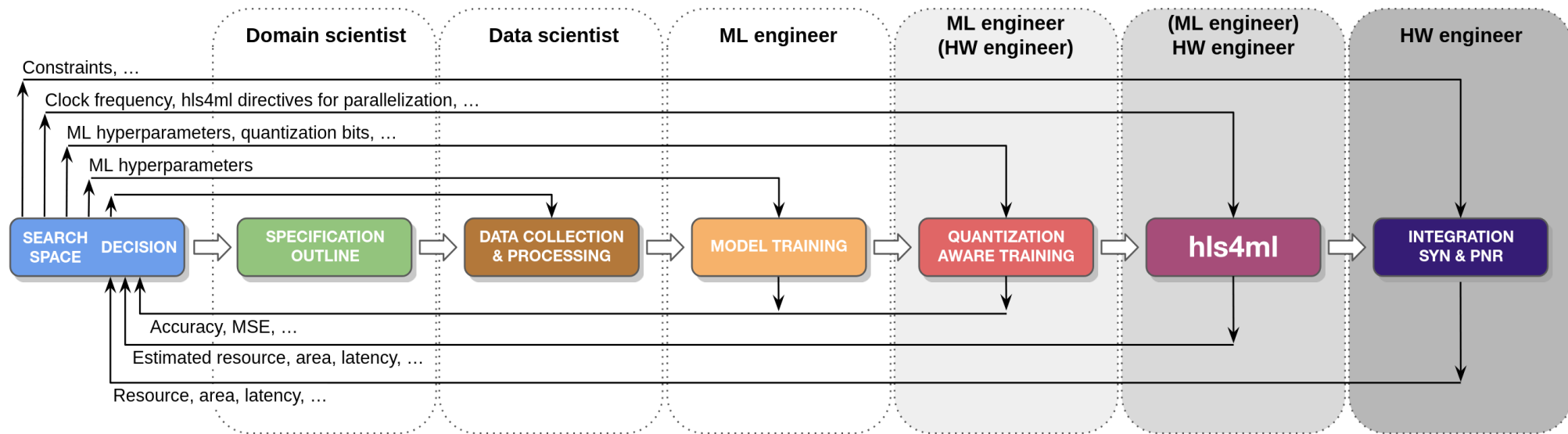
Software Version v2024.1
February 2024

Support for HLS4ML flow (beta)



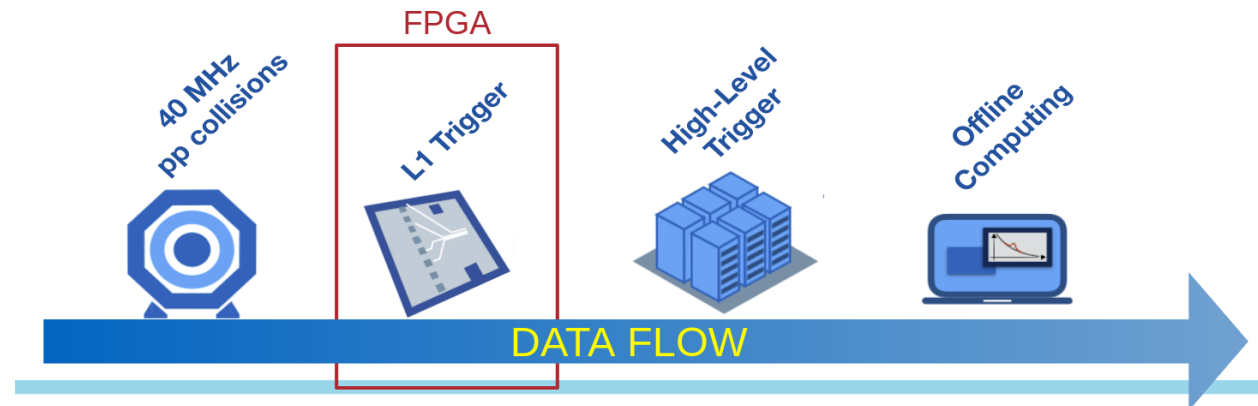
Co-design with hls4ml

- Co-design = development loop between algorithm design, data collection, training, and hardware implementation
 - Large design search space
 - Scientists and engineers with different expertise



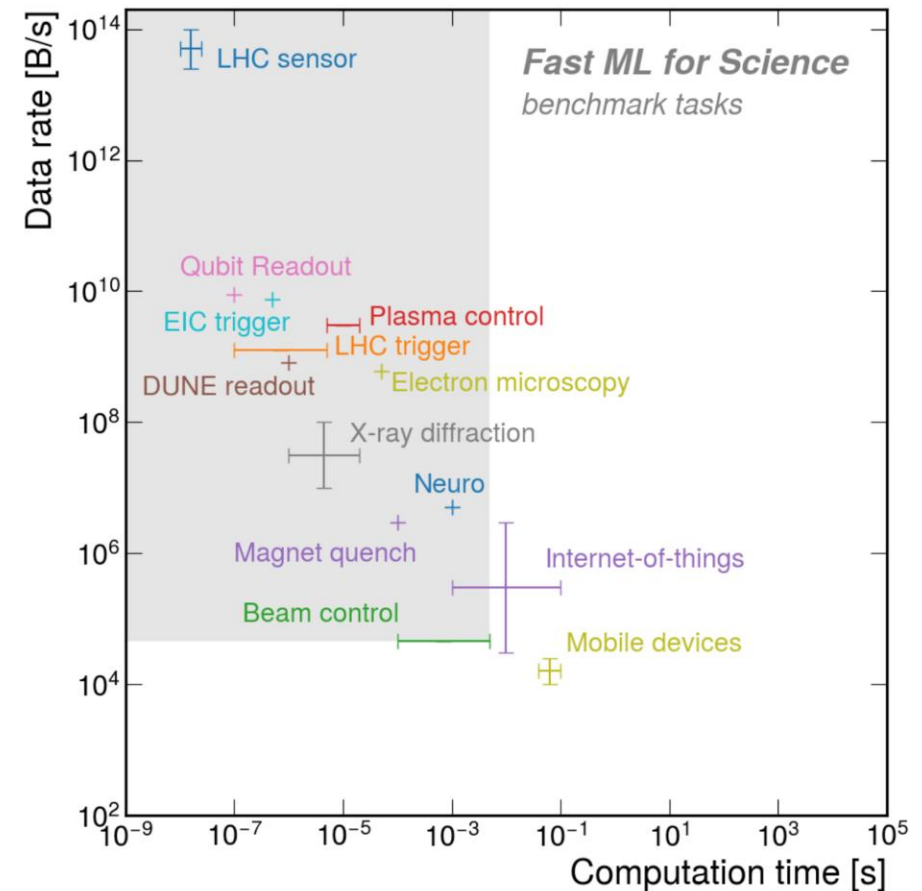
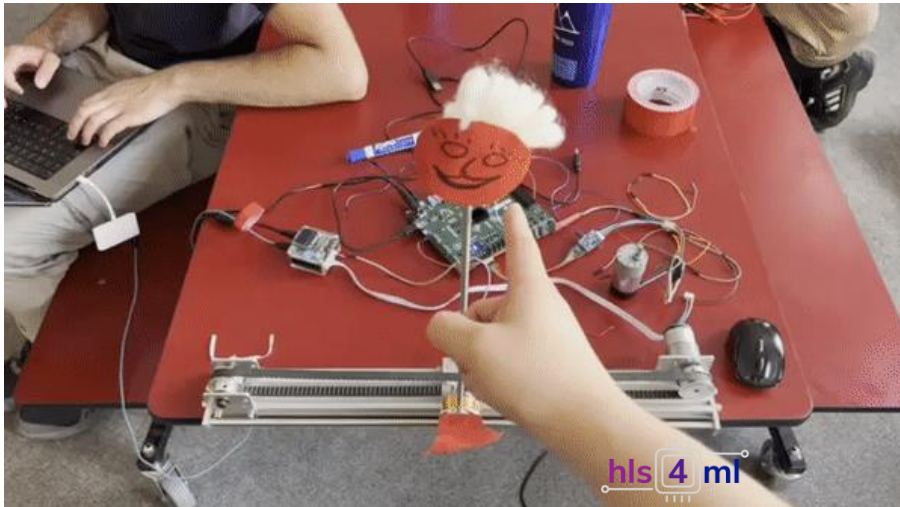
hls4ml origins

- High energy physics
 - Large Hadron Collider (LHC) at CERN
 - Extreme collision frequency of 40 MHz → extreme data rates $O(100 \text{ TB/s})$
 - Most collision “events” don’t produce interesting physics
 - “Triggering” = filter events to reduce data rates to manageable levels



hls4ml has grown

- To a large variety of scientific applications
 - Low latencies (ms \rightarrow ns)
 - High throughput O(100TB/s)
- ... including teaching material



hls4ml community

GitHub

Watch 54 Fork 375 Starred 1.1k



README

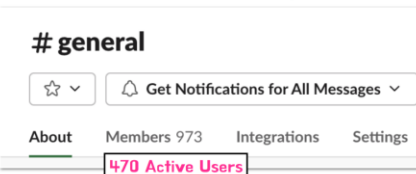
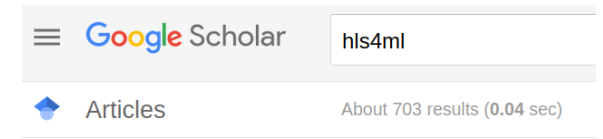
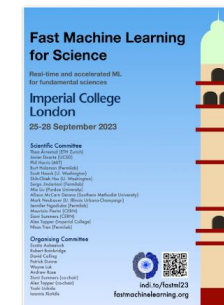
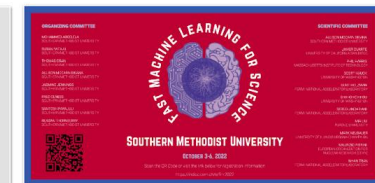
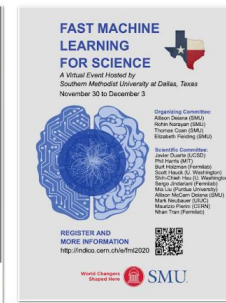
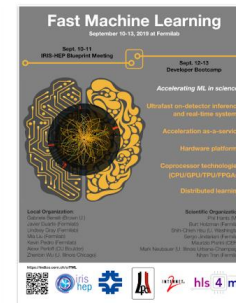
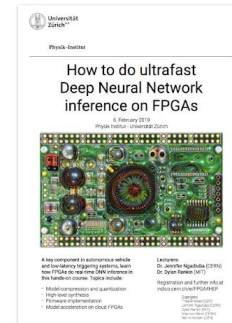
hls4ml-tutorial: Tutorial notebooks for hls4ml

Jupyter book deploy-book passing code style black pre-commit enabled launch binder

There are several ways to run the tutorial notebooks:

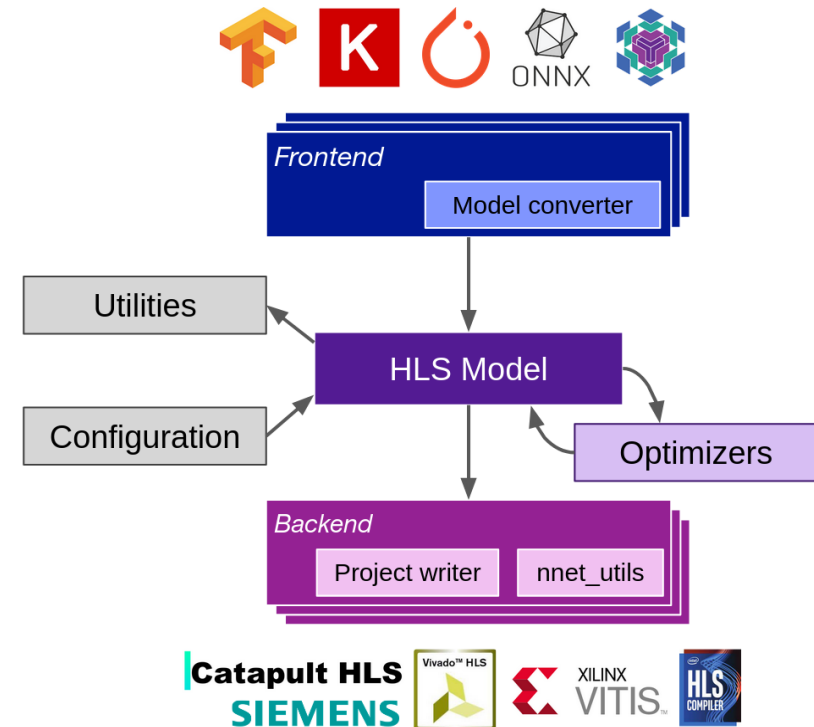
Online

launch binder

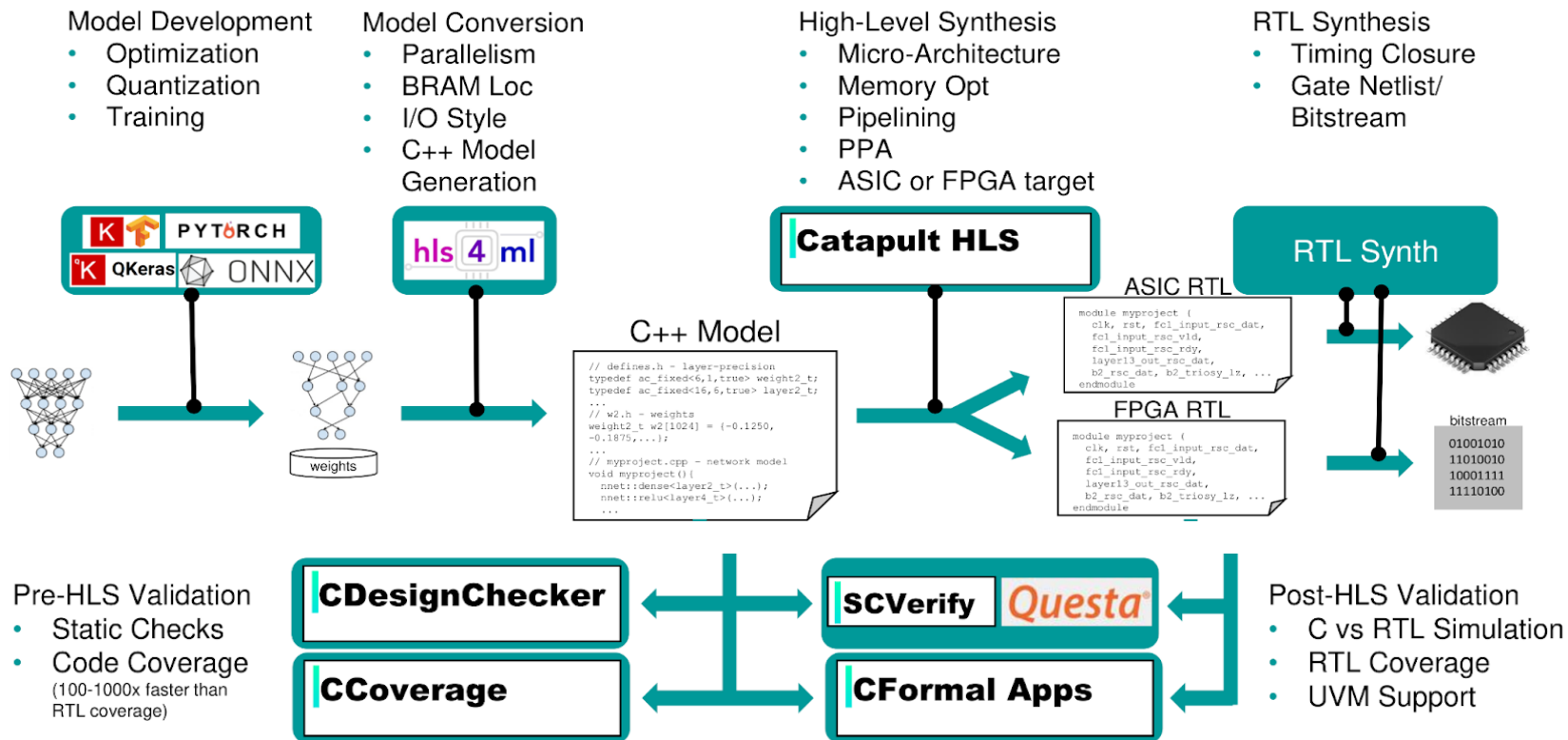


hls4ml architecture

- Converts from ML frameworks
- Internal representation
- Configuration to tune latency vs. resources, bit precision
 - | hls4ml knobs | << | HLS knobs |
- Optimizers, e.g. merging layers
- Backends to HLS tools
- nnet_utils = C++ library of ML functionalities optimized for HLS

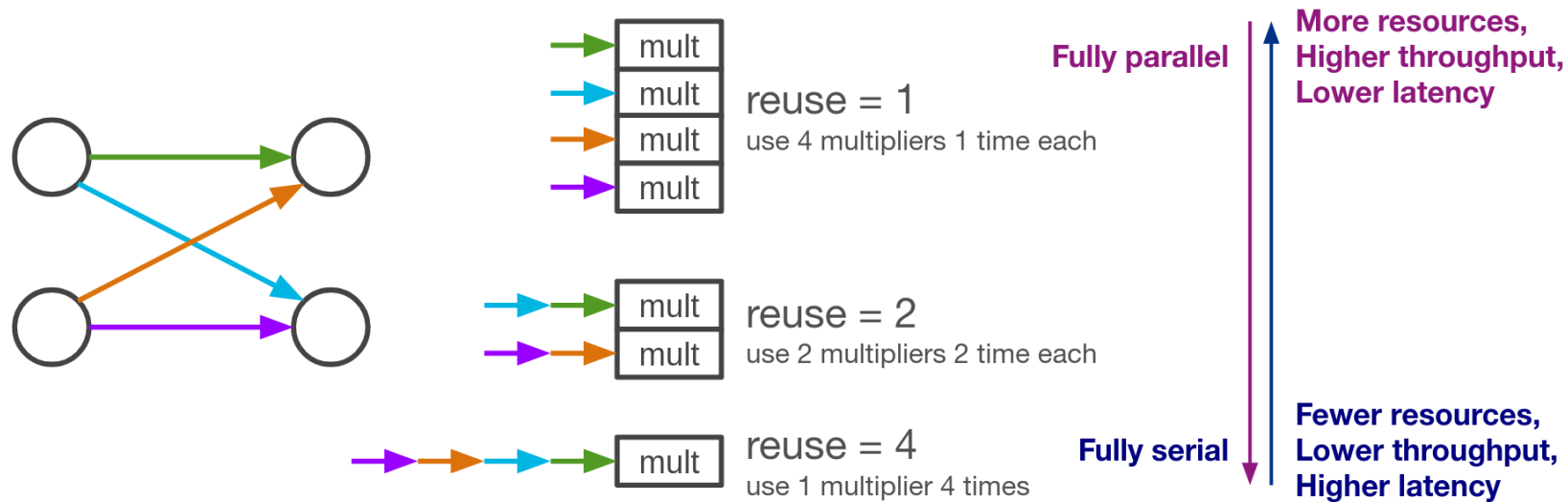


hls4ml supports Catapult HLS



hls4ml – Parallelization

- Trade-off between latency and resource usage determined by the parallelization of the logic in each layer
- `ReuseFactor` = number of times a multiplier is used to do a computation



Design space exploration via reuse factor

- ReuseFactor = 1, 2, 4
- Other configurations (ignore for now)
 - Streaming Input, On-chip Weights, 32nm ASIC, 10ns Clock, Latency mode

Layer	Area	Latency	TotalPwr	DynPwr	LeakPwr
nnet::zeropad2d_cl<input_t,layer5_t,config5>	631	868	113	22	91
nnet::conv_2d_cl<layer5_t,layer2_t,config2>	75855	842	5787	688	5099
nnet::normalize<layer2_t,result_t,config4>	4924	196	434	34	400

Layer	Area	Latency	TotalPwr	DynPwr	LeakPwr
nnet::zeropad2d_cl<input_t,layer5_t,config5>	631	868	108	17	91
nnet::conv_2d_cl<layer5_t,layer2_t,config2>	49916	1682	6942	704	6238
nnet::normalize<layer2_t,result_t,config4>	4924	391	421	20	401

Layer	Area	Latency	TotalPwr	DynPwr	LeakPwr
nnet::zeropad2d_cl<input_t,layer5_t,config5>	631	868	102	11	92
nnet::conv_2d_cl<layer5_t,layer2_t,config2>	40815	3363	5453	456	4997
nnet::normalize<layer2_t,result_t,config4>	4942	781	416	13	403

RF = 1

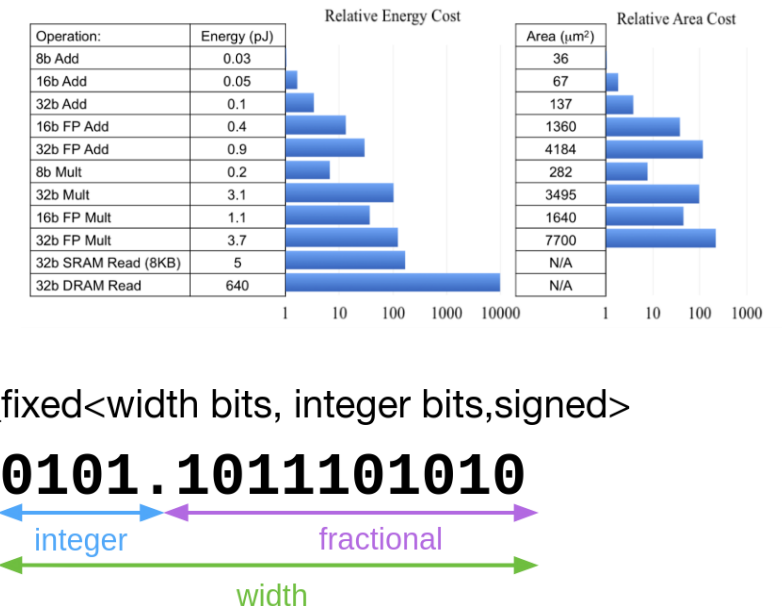
RF = 2

RF = 4

Latency increases by factor of 2 while area decreases accordingly

hls4ml – Quantization

- As “customary” in custom hardware, we use quantized representation
 - Floating-point computation is too resource intensive
- Precision = fixed point types
 - `ac_fixed`, **Algorithmic C Datatypes**
 - https://github.com/hlslibs/ac_types
- Operations are integer ops, but we can represent fractional values
- But we have to make sure we’ve used the correct data types!
 - Post training quantization
 - Quantization aware training



1.1 computing's energy problem (and what we can do about it), M. Horowitz 2014

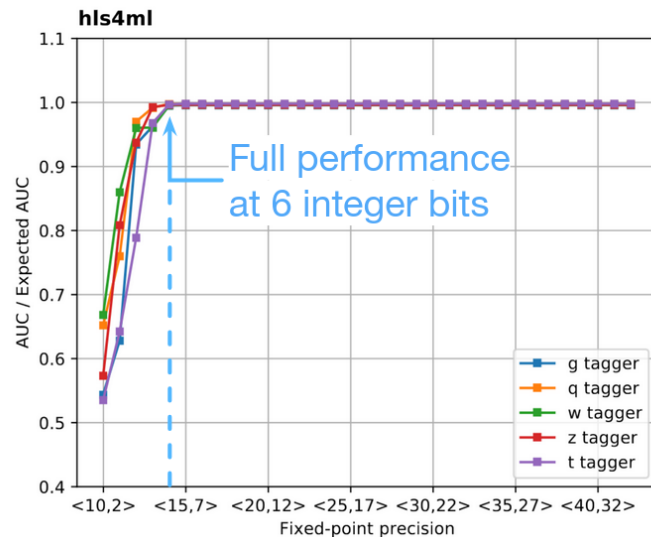
High-performance hardware for machine learning, W. Dally 2015

Design space exploration via (post-training) quantization

- Post-training quantization (PTQ) = turning weights from float to fixed (or other quantized format)

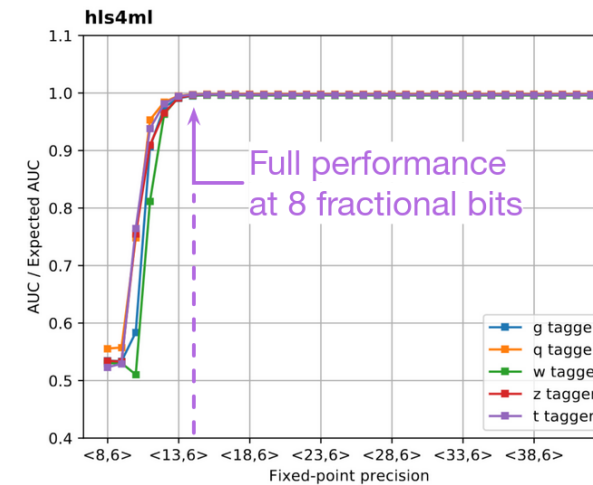
Scan integer bits

Fractional bits fixed to 8



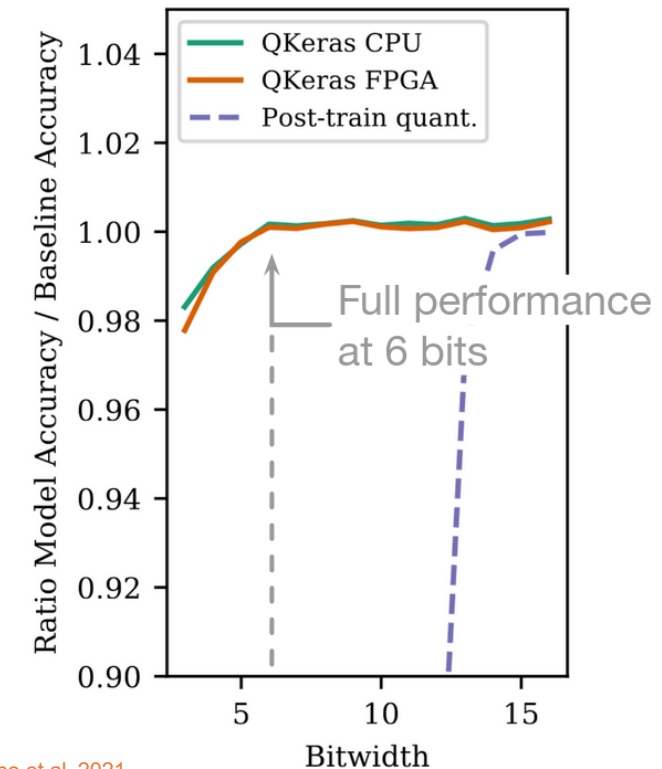
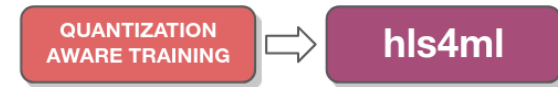
Scan fractional bits

Integer bits fixed to 6



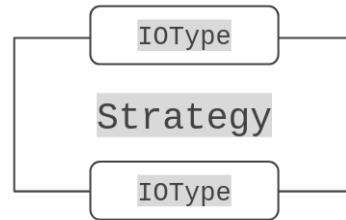
Quantization-aware training (QAT)

- QAT improves on PTQ
 - Taking into account quantization numerics and learning around them
 - More compact bit representation → Reduction area, power, and latency
 - QKeras <https://github.com/google/qkeras>, Brevitas <https://github.com/Xilinx/brevitas>
 - Easy to use, e.g. drop-in replacements for Keras layers
 - Dense → QDense
 - Conv2D → QConv2D



hls4ml – Layer implementations and interfaces

- **hls4ml** is a specialized compiler or transpiler
 - Translate a high-level specification of a model into HLS-ready code that implements the same algorithms
- User can choose
 - `Strategy` for the implementation of the layers
 - “Latency” for smaller model where likely the goal is high-parallelism, i.e. low reuse factor
 - “Resource” for larger model and higher reuse factor
 - `IOType` for the interfaces of layers and overall module
 - “io_parallel” for data passed as arrays
- “io_stream” for data passed as latency-insensitive channels, e.g. `ac_channels` **Algorithmic C Datatypes**

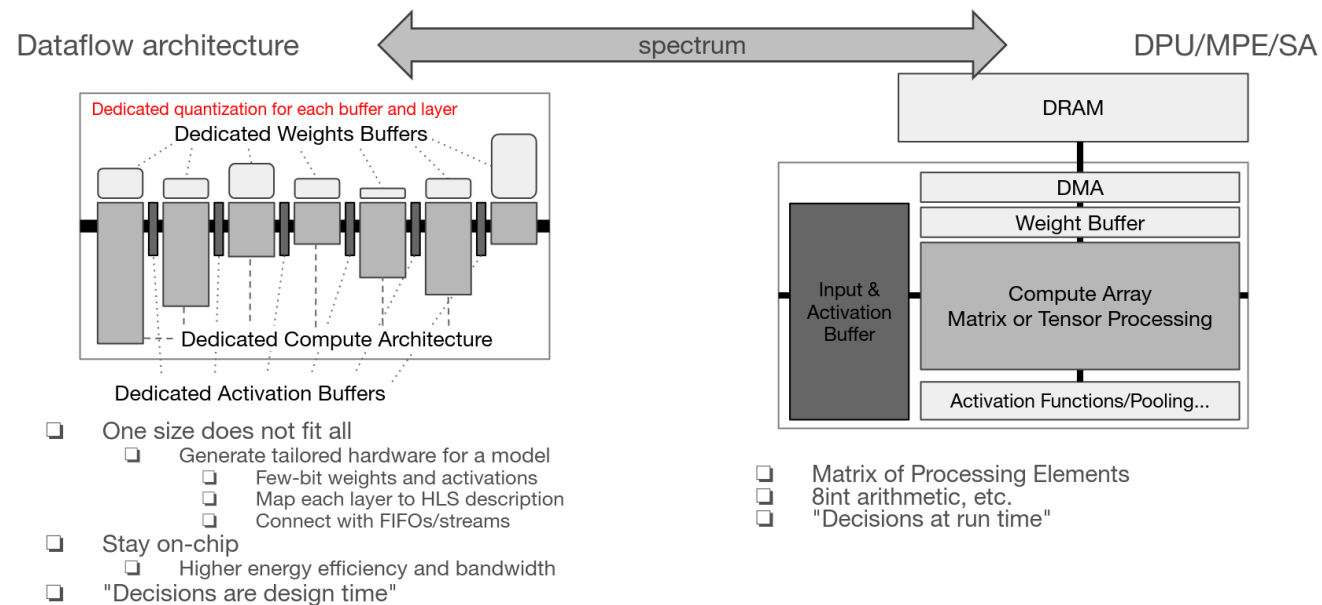


hls4ml configuration in summary

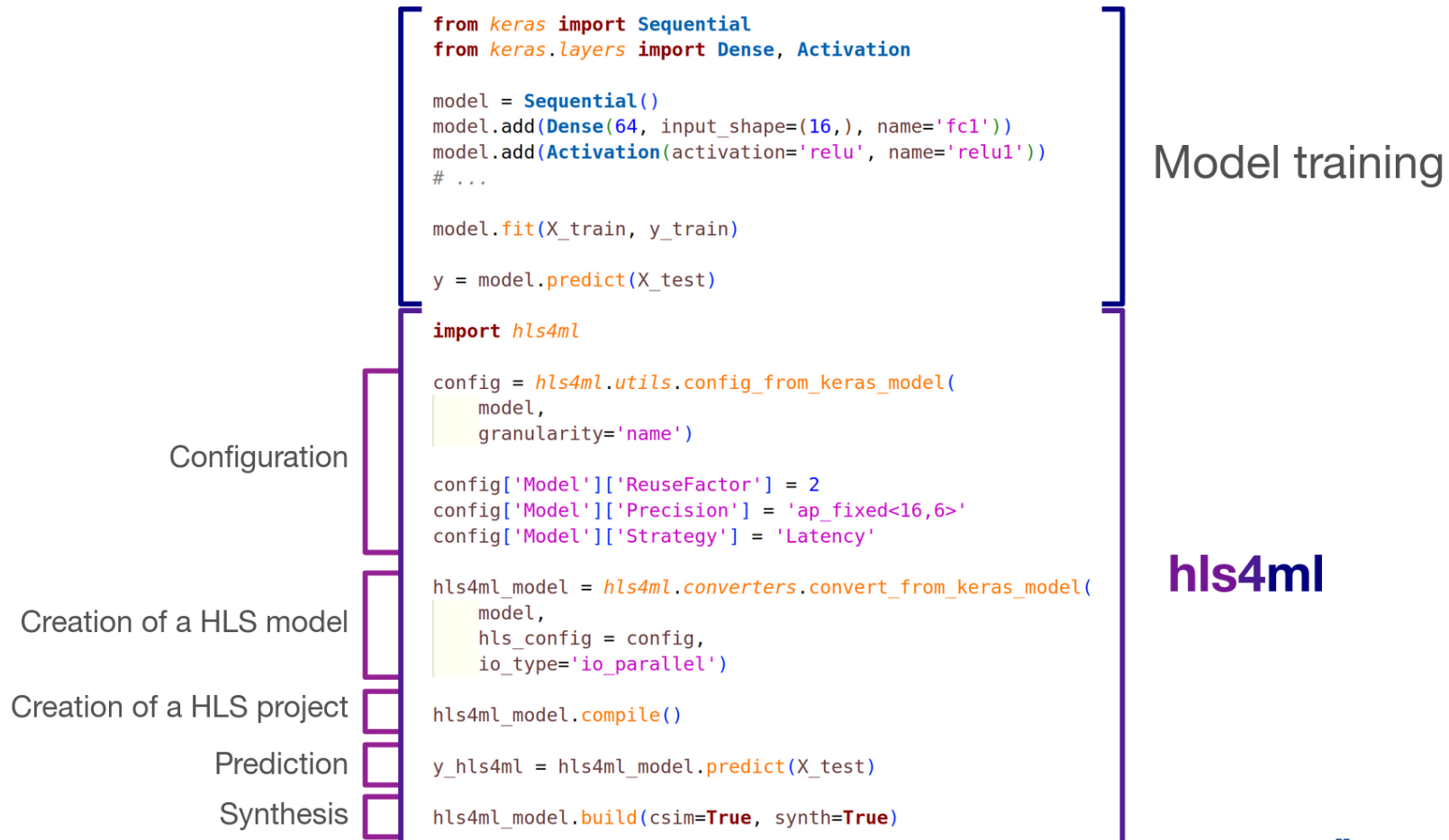
- `ReuseFactor` : <integer value>
 - Controls the level of parallelism – 1 is the most parallel (smallest latency), 2 is half that...
- `Precision` : <fixed-point data type>
 - Global or per-layer option configuring the precision for feature, weight and bias values
- `Strategy` : "latency" or "resource"
 - Selects different C++ architectures for the layer implementations
- `IOType` : "io_parallel" or "io_stream"
 - Passes data either as arrays or latency-insensitive channels, e.g.
- `Part` : <FPGA part>
 - Identifies the specific FPGA family/part is used in downstream RTL synthesis
- `ClockPeriod` : <period in ns>
 - Specifies the clock period for HLS

hls4ml – Heterogenous dataflow architecture

- **hls4ml** instantiates and configures layers of a model in a data flow architecture



hls4ml – Example



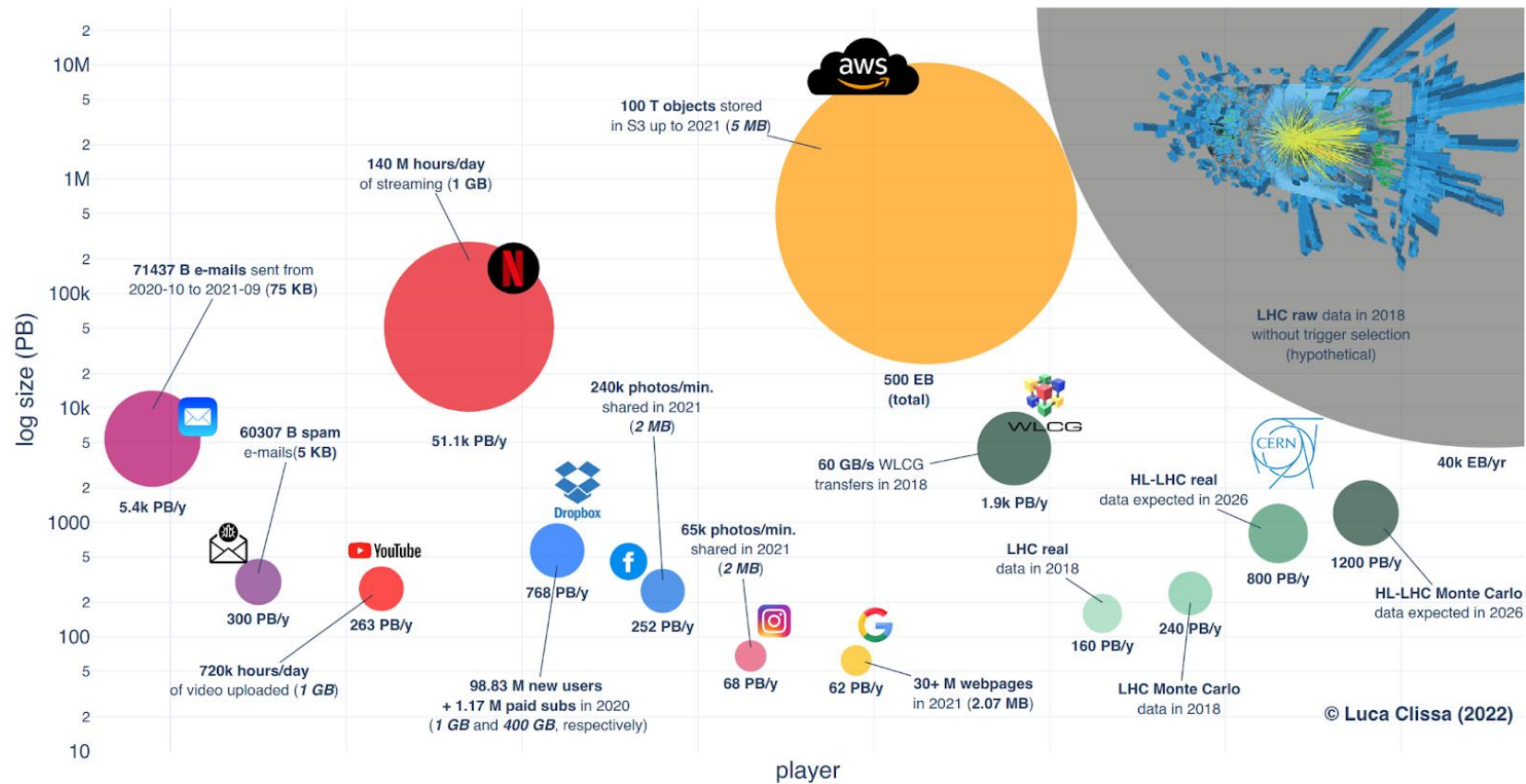
Applications



SPONSORED BY



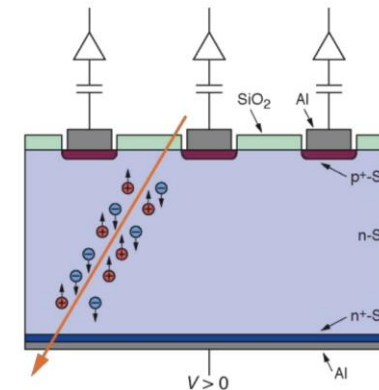
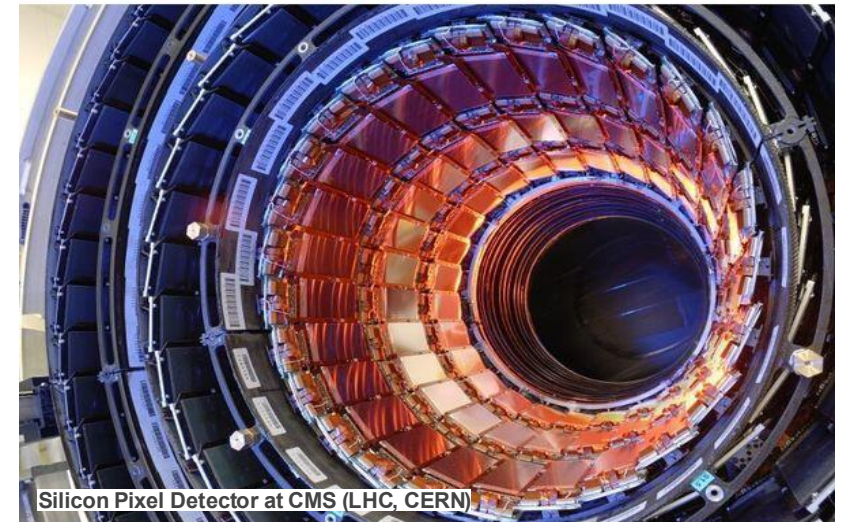
Survey of Big Data sizes in 2021



Silicon pixel detectors

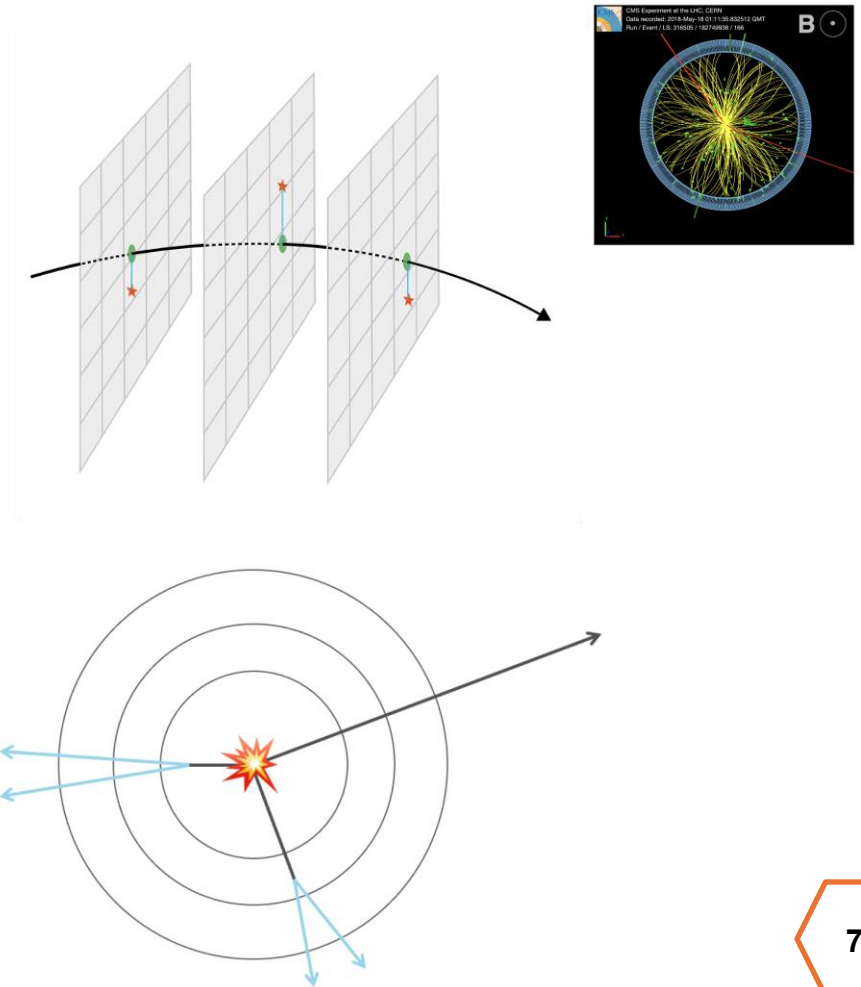
- Experiments at colliders typically have a silicon pixel detector at the center
 - Concentric rings tiled with sensors
- Silicon sensors are depleted of charge carriers by high voltage
- When a charged particle from a collision passes through, it creates e/h pairs
- Charge is read out and transferred off-detector
 - Charge cluster information is used for physics analysis offline

<https://cms.cern/detector>



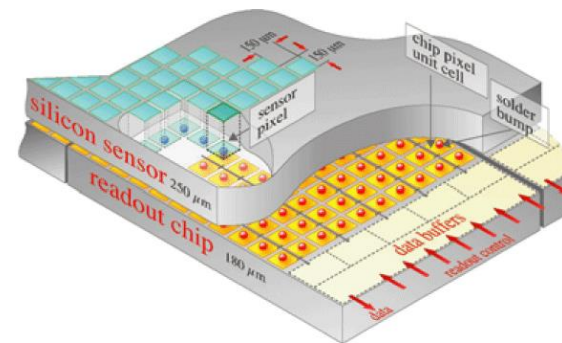
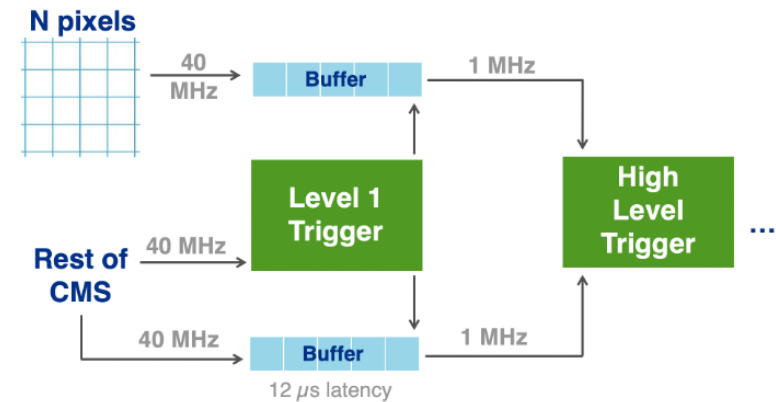
Particle tracks and vertices

- Connecting the dots between charge collected in different pixel layers creates a particle track
 - Detector should be low-mass so interactions in inactive material doesn't disrupt this trajectory
- Solenoid magnet immerses the pixel detector in a magnetic-field, causing tracks to curve
 - Very curved \rightarrow low transverse momentum (low- p_T)
 - Almost straight \rightarrow high transverse momentum (high- p_T)
- Reconstructing vertices is critical
 - Secondary vertices help identify particles: long, short, medium lifetime?



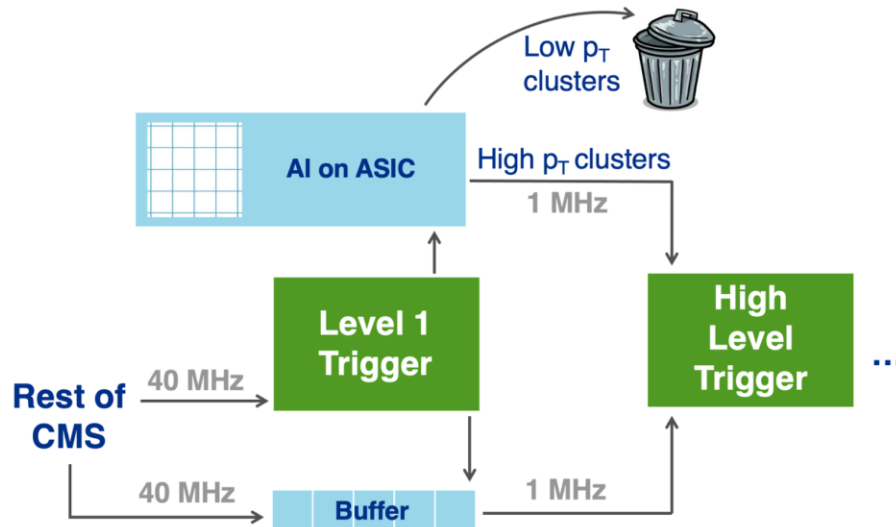
Designing hardware for the LHC is challenging

- LHC/CMS produces a lot of data
 - New data every 25 ns (p-p collision)
 - Physicists have to throw most of it away
 - Physically and financially challenging
 - Risk to throw away significant information
- Detector is continuously being sprayed with particles
 - Need radiation tolerant on-detector electronics
- High voltage and low temperature requirements
 - Up to -800 V, -35 C



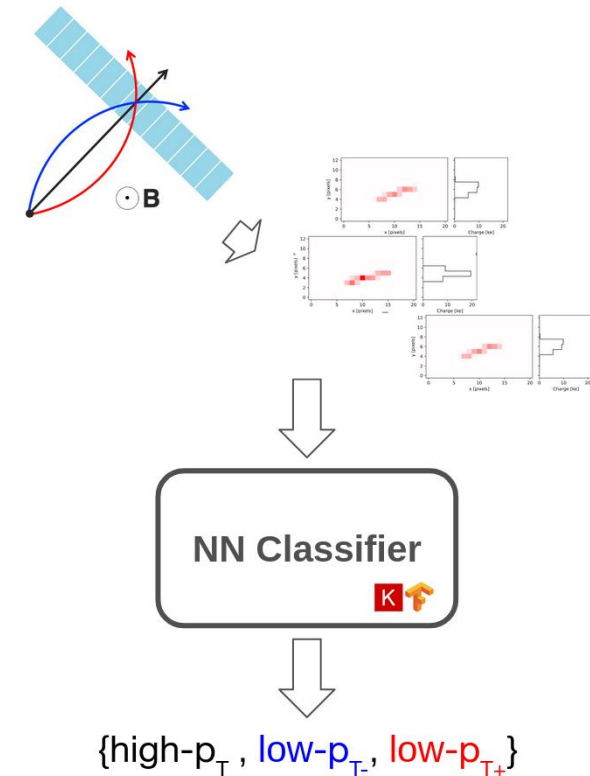
Goal of the Smart Pixel team

- On-chip data filtering at rate (40 MHz)
- AI algorithms
- Reconfigurable algorithms
- Hybrid pixel detector
 - Silicon sensor
 - Pixelated ROIC
 - Analog front-end + ADC
 - AI in digital logic



Neural network classifier (filter)

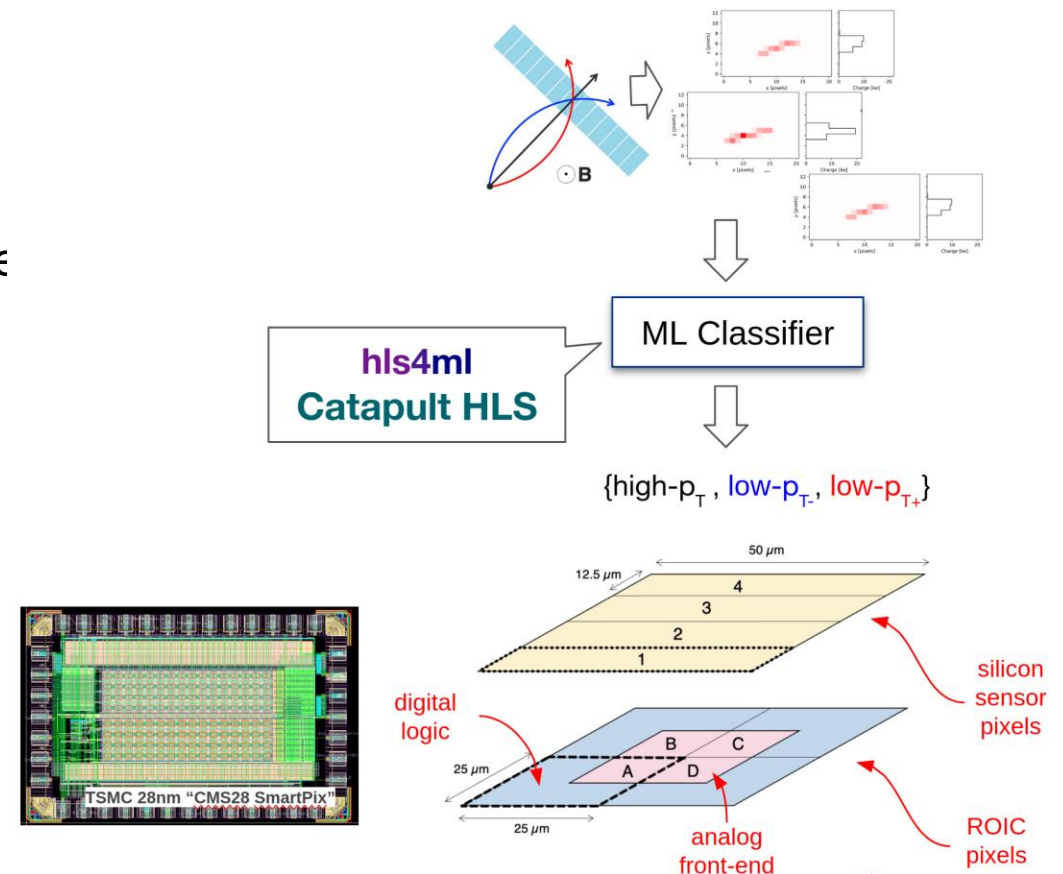
- Inputs are cluster images projected onto y-axis and the associated y_0
- Three output categories
 - high-momentum (> 200 MeV)
 - low-momentum, negatively charged
 - low-momentum, positively charged
- Simulated dataset of 800,000 clusters
- Classical training and testing set split 80%-20%
- Tensorflow/Keras, 200 epochs for training, 20 epochs of early stopping, 1024 batch size, Adam optimizer



Filtering in ASIC at LHC

- On-chip data reduction at BX rate
 - R&D for phase III CMS experiments
 - pp-collision 40 MHz
- Integration of the ML algorithm as digital logic with the analog front-end into the in the pixelated area
- Low-power 28nm CMOS
- Total power < 1 W/cm²
 - Analog ~5 μ W/pixel
 - Digital ~1 μ W/pixel
- Bandwidth saving
 - **54.4% - 75.4%**

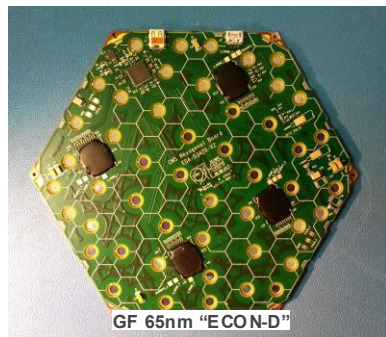
[Smart pixel sensors: towards on-sensor filtering of pixel clusters with deep learning, J. Yoo et al. 2023](#)



Data compression in ASIC at LHC

- Autoencoder (ML) on the detector front-end for data compression
 - ASIC required due to radiation tolerance, handled through triple modular redundancy, and power requirements
- Reconfigurable ASIC to address: evolving LHC conditions (beam related), detector performance (noise, dead channels), and updated performance metric (resolution, new physics signatures)

8'' hexagonal silicon module
(1 out of ~27,000)



Metric / requirement	Value
Rate	40 MHz
Total ionizing dose	200 Mrad
High energy hadron flux	$10^7 \text{ cm}^2/\text{s}$
Tech. node	65 nm LP CMOS
Power	48 mW
Energy / inf.	1.2 nJ
Area	2.88 mm^2
Gates	780k
Latency	50 ns

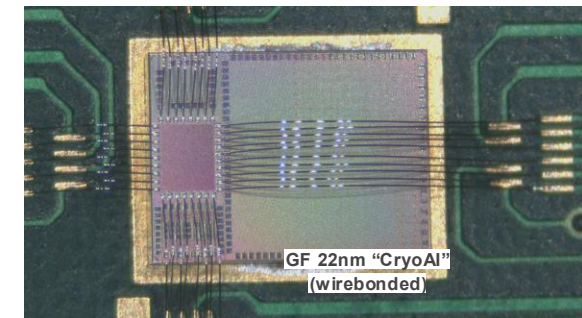
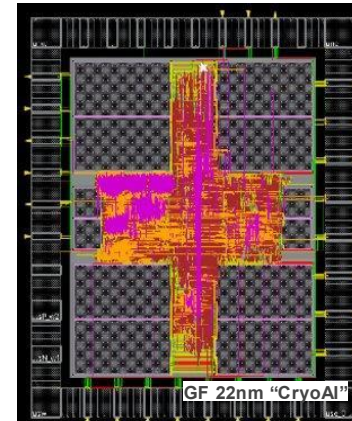
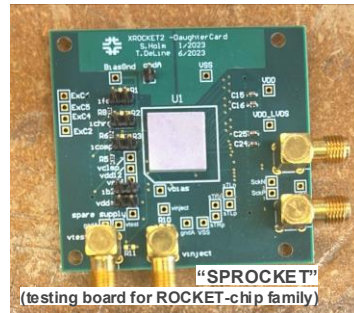
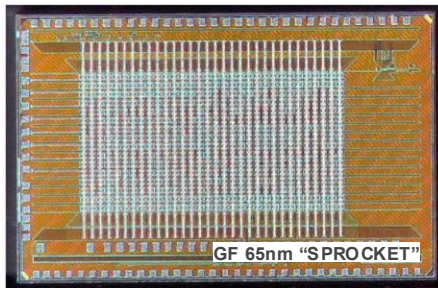
Using QKeras, **hls4ml**, and **Catapult HLS**

- reduced **power by 50%**, **area by 80%**, and achieved **2x better performance** reference solutions by optimizing compression and quantization
- Faster design cycle!

More ASIC applications with hls4ml and Catapult HLS

- Data compression for X-ray microscopy (ptychography)
- Testing chip at GF 65nm
- Evaluation of algorithms
- PCA vs. Autoencoder

Up to **70x data compression** at source with a **20% increase** in pixel area

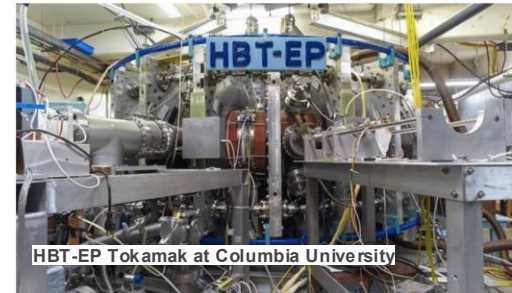


- Quantum readout at cryogenic temperatures (4 Kelvin)
- Testing chip at GF 22nm
- SoC with ML accelerator
- Under testing

A recent application for FPGA: Plasma control

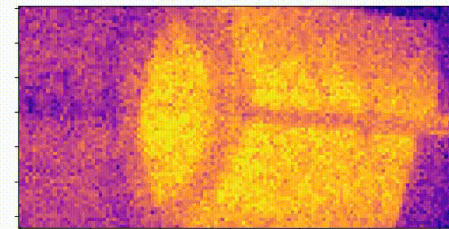
- Plasma instabilities when magnetic field lines become distorted
 - μ -seconds constraints
- Confinement loss \rightarrow damage to the reactor
- One of the major roadblocks preventing lasting thermonuclear fusion

Model Name	PPCF23 Baseline	QAT+Pruning	Optimized
Image Resolution	128×64	128×64	32×32
Conv layer filters	{8,8,16}	{8,8,16}	{16,16,24}
Dense layer widths	{256,64}	{256,64}	{42,64}
Total parameters	362,730	362,730	12,910
Parameter precision	PTQ, 18 bits	QAT, 8 bits	QAT, 7 bits
Sparsity	none	80%	50%
Bit Operations	$6.74e13$	x	$4.52e11$

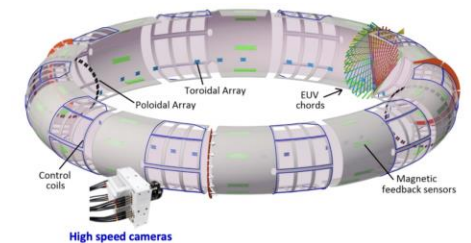
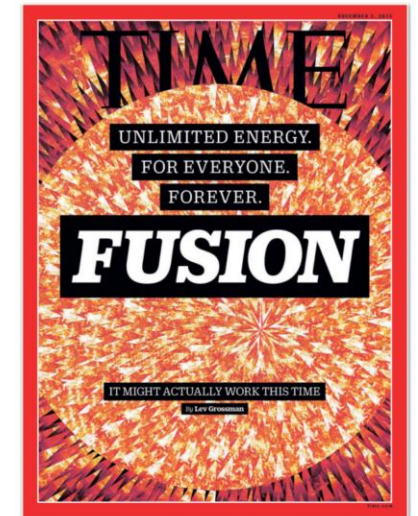
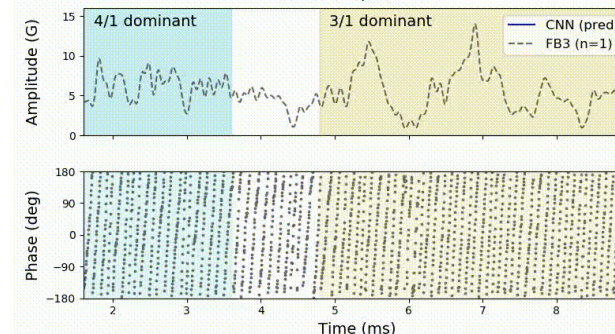


<http://sites.apam.columbia.edu/HBT-EP>

Shot 114467 Camera 1

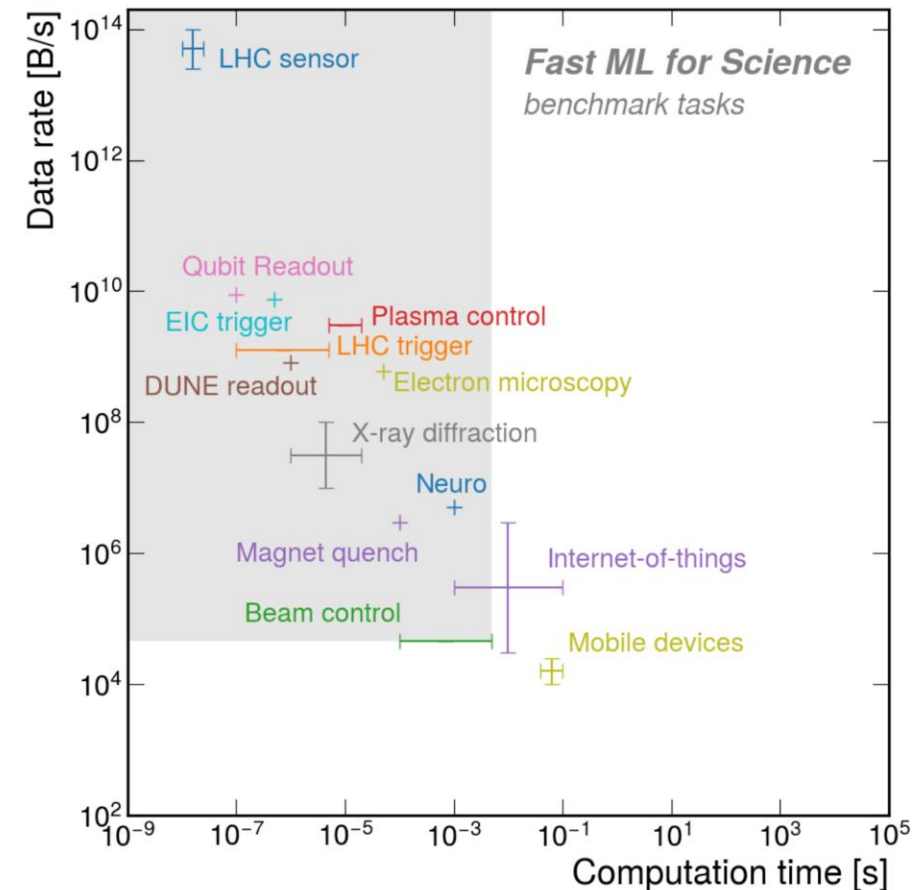


frame 0000, 1.604 ms



hls4ml in summary

- Open source + community
- Python ML package
 - Reads and optimizes ML networks
 - Library of optimized HLS-ready ML functions
 - Dataflow pipeline of hardened layers
 - Easier design space explore for ML implementation
 - Support of **Catapult HLS**
- Successful for both ASIC and FPGA applications





AI



Security



Systems



EDA



Design

Thank you!



THE CHIPS
TO SYSTEMS
CONFERENCE

SPONSORED BY



Breaking Down a Neural Network

```
# Create NN Model
model = Sequential(name='mnist_conv2d_ptq')
model.add(layers.Input(shape=(28,28, 1), name='input1'))
model.add(layers.Conv2D(filters=5, kernel_size=5, strides=3,
                        name='conv2d1'))
model.add(layers.BatchNormalization(name='batchnorm1'))
model.add(layers.Activation('relu', name='relu1'))
model.add(layers.Flatten(name='flatten1'))
model.add(layers.Dense(10, name='dense1'))
model.add(layers.Activation('softmax', name='softmax1'))

# Compile and Train
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=10)

# Measure the accuracy of the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc}")
```


Headline for a section divider



SPONSORED BY



Content Headline

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam at iaculis tortor. Donec in feugiat erat. Ut dapibus, quam quis suscipit condimentum, risus enim aliquam quam, eu convallis sapien enim a tellus. In hac habitasse platea dictumst. Nunc eu interdum odio. Proin porta felis vitae nibh lobortis auctor. Praesent ut orci ex. Pellentesque eu dui ex. Aenean ac ultricies massa. In blandit tortor elementum tortor ullamcorper venenatis.

Content Page Headline

- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam at iaculis tortor. Donec in feugiat erat.
- Ut dapibus, quam quis suscipit condimentum, risus enim aliquam quam, eu convallis sapien enim a tellus.
- In hac habitasse platea dictumst. Nunc eu interdum odio. Proin porta felis vitae nibh lobortis auctor. Praesent ut orci ex. Pellentesque eu dui ex. Aenean ac ultricies massa. In blandit tortor elementum tortor ullamcorper venenatis.

Headline Only





Speaker Name

Speaker Company and
Title



Speaker Name

Speaker Company and
Title



Speaker Name

Speaker Company and
Title



Speaker Name

Speaker Company and
Title



AI



Security



Systems



EDA



Design



THE CHIPS
TO SYSTEMS
CONFERENCE

SPONSORED BY

