Fast Recurrent Neural Networks on FPGAs with hls4ml

Elham E Khoda
University of Washington, Seattle

Elham E Khoda[1]*, Dylan Rankin[2]†, Rafael Teixeira de Lima[3]‡
Philip Harris[2], Scott Hauck[1], Shih-Chieh Hsu[1], Michael Kagan[3], Vladimir Loncar[4],
Chaitanya Paikara[1], Richa Rao[1], Sioni Summers[4], Caterina Vernieri[3], Aaron Wang[1]

[1]University of Washington, Seattle, WA 98195, USA
[2]Massachusetts Institute of Technology Cambridge, MA 02139, USA
[3]SLAC National Accelerator Laboratory, Menlo Park, CA 94025, USA
[4]European Organization for Nuclear Research (CERN), Geneva, Switzerland

Fast Machine Learning for Science Workshop 2022
October 4, 2022

Accelerated AI Algorithms for Data-Driven Discovery

UNIVERSITY of WASHINGTON

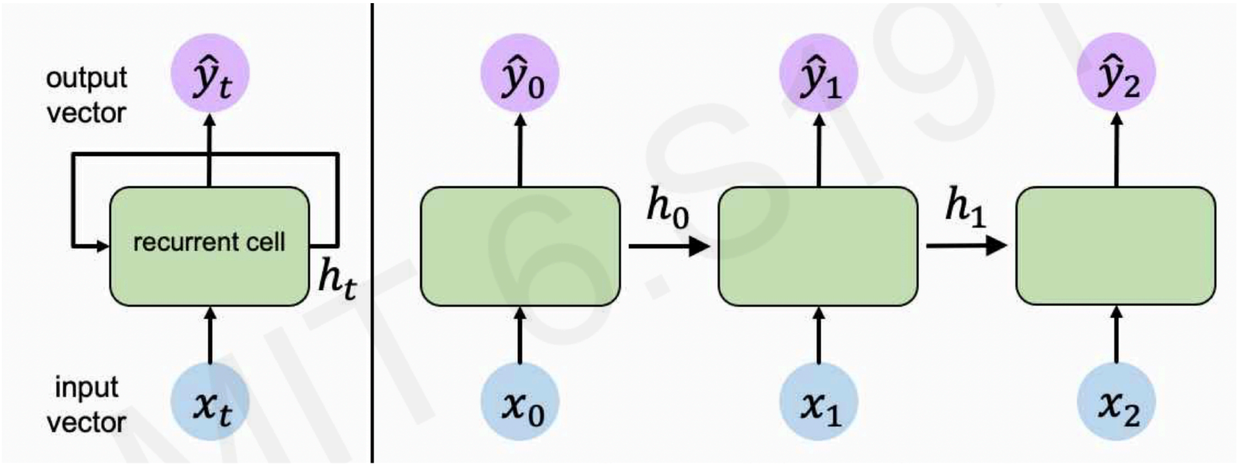
Recurrent Neural Network (RNN)

Image source

- **Recurrent Neural Networks**

  - Designed to work with sequential data
    - Text, audio, video, strokes, etc

  - RNNs have a state, h_t , that is updated at each time step as the sequence is processed
  - Apply a recurrence relation at every time step to process a sequence



$$\hat{y} = f(x_t, h_{t-1})$$

Output Input past memory

$$h_t = f_W(x_t, h_{t-1})$$

cell state

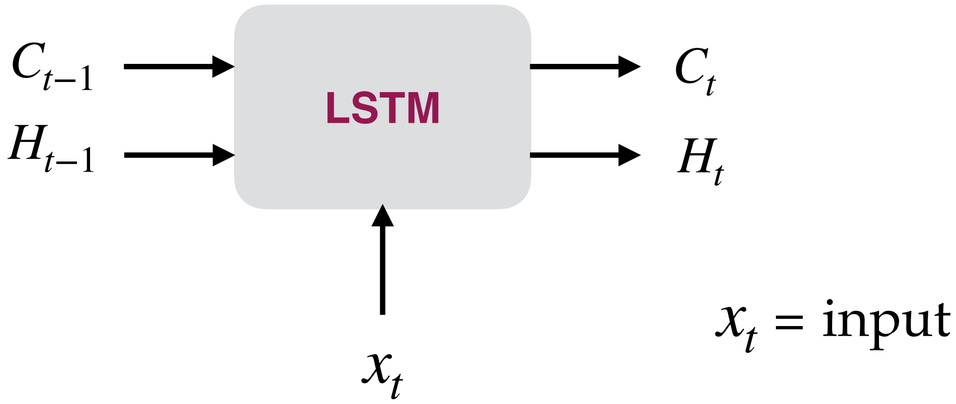
Function with weights W

Input old state

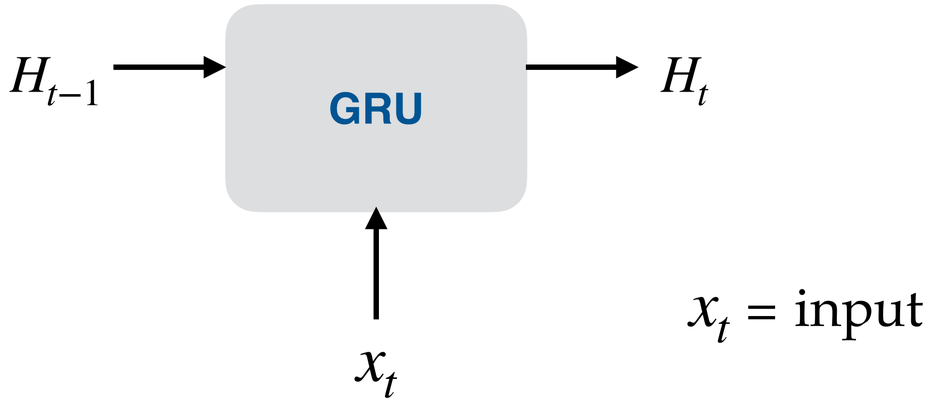
Implementation of RNN models:

- LSTM (Long Short-Term Memory)
- GRU (Gated Recurrent Unit)

LSTM vs GRU



- **3 gates:** Input, Output, Forget
- **2 States:** Cell state (C_t) and Hidden state (H_t)

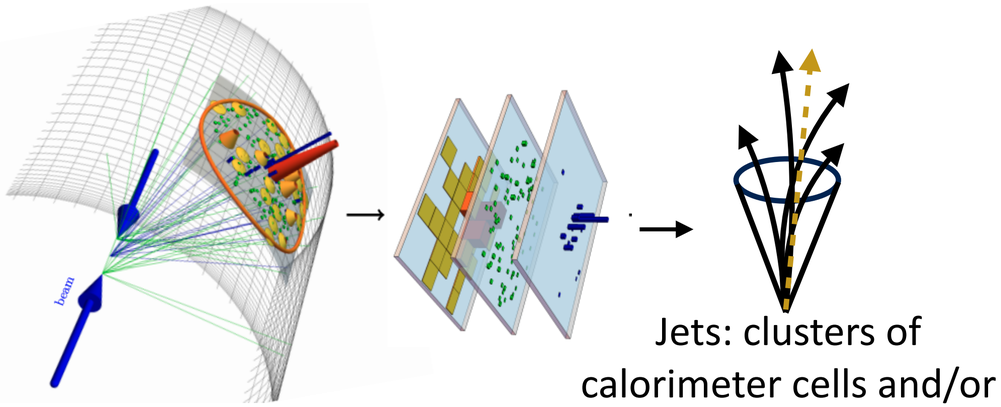


- **2 gates:** Update and Reset
- **Single** Hidden state (H_t)

- Faster to train
- Less number of matrix multiplications

Particle jets at the LHC

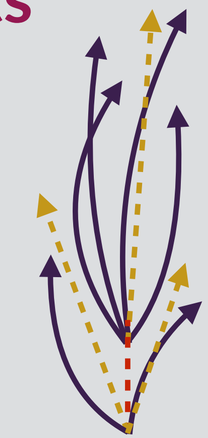
Jets are experimental signature of quarks and gluons



Jets: clusters of calorimeter cells and/or tracks

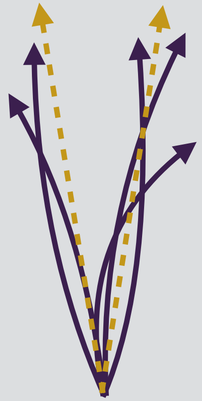
b or c jets

- Jets coming from b- or c-quark
- Displaces vertex



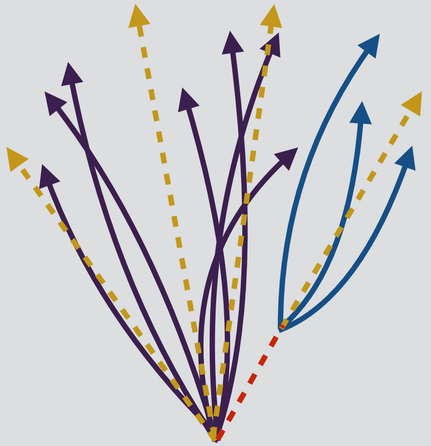
light-jets

- Jets coming from u/d/s-quarks



top jet

- Jets coming from top-quark
- 3-prong structure



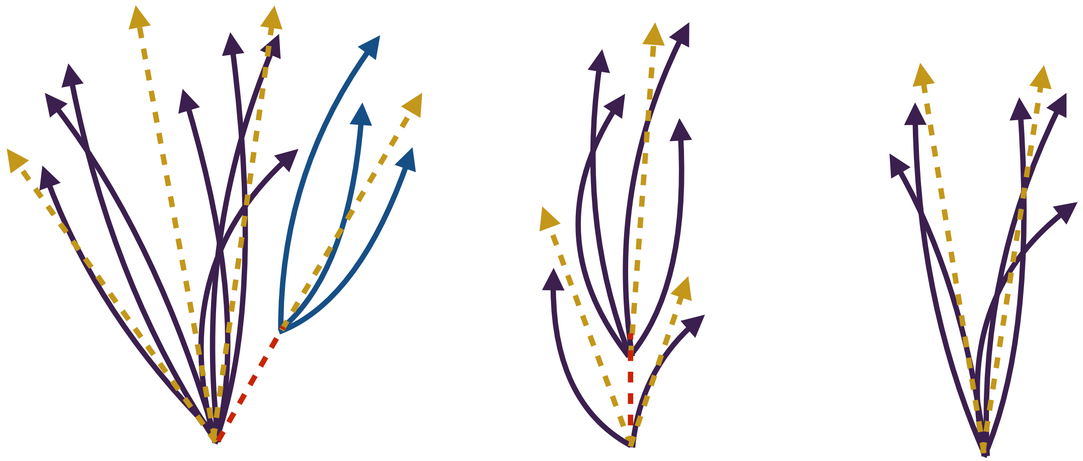
Benchmark Examples

Three benchmark cases

1. Binary classifier: ~4k parameters
   Identify top-quarks

2. 3-class classifier: ~60k parameters
   Classify b / c / light jets

3. 5-class classifier: ~130k parameters
   QuickDraw dataset: differentiate between Bees, Butterflies, Mosquitos, Snails, Ants



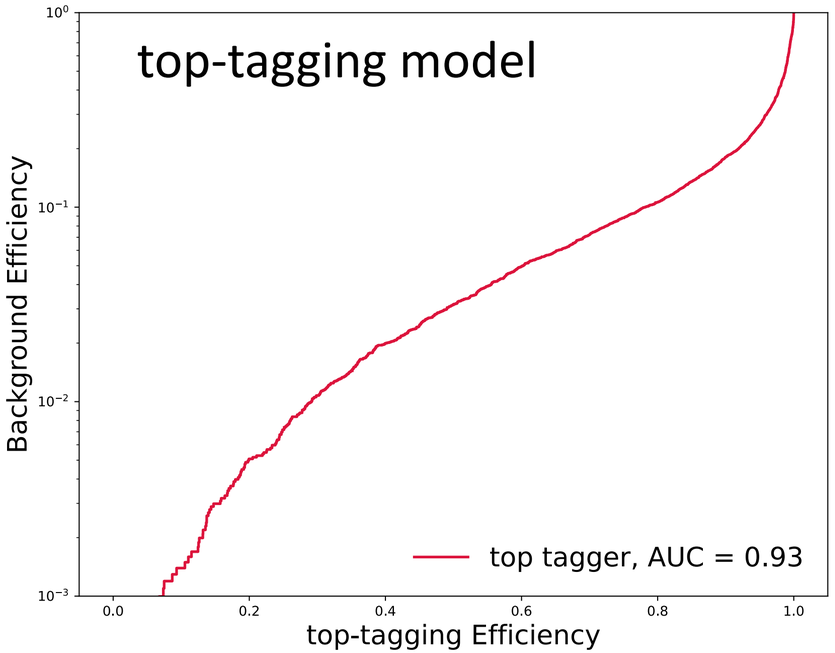
top **b/c** **light**

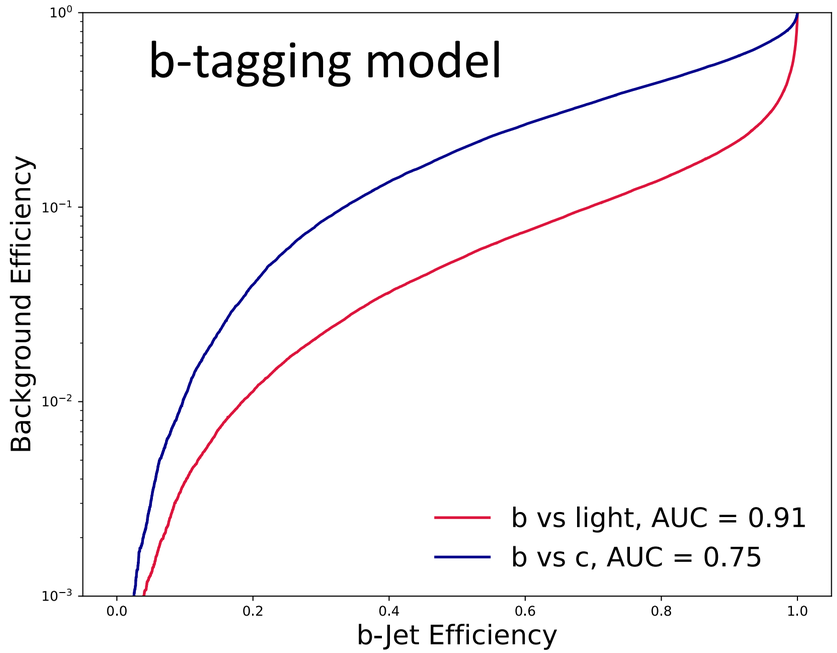
QuickDraw dataset

See Max's talk for more use cases at the LHC

Model performance: ROC

- All the benchmark models are trained using **Keras + TensorFlow**
- Weights and biases are represented by 32 bit floating point numbers





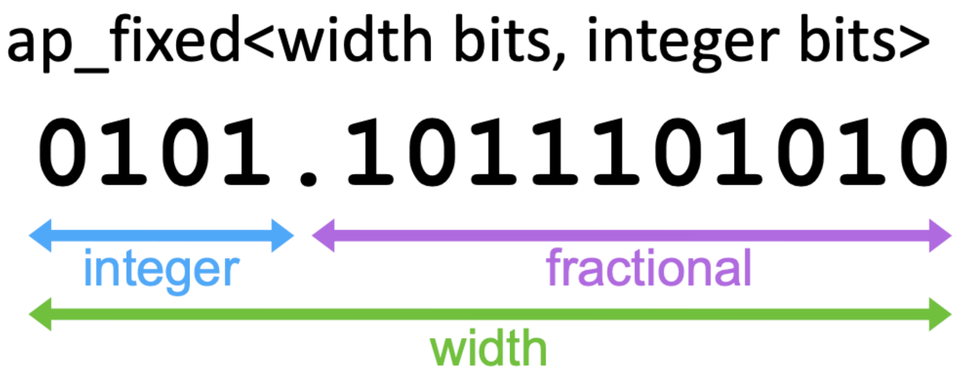
Quantization

Quantization – Reducing the bit precision used for NN arithmetic

Why this is necessary?

- Floating-point operations (32 bit numbers) on an FPGA consumes large resources
- Not necessary to do it for desired performance

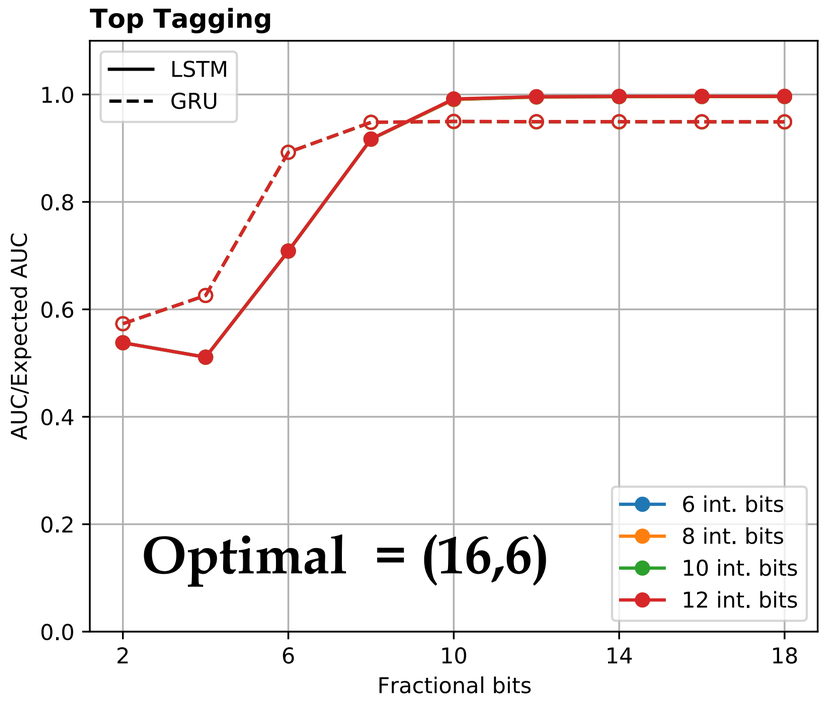
- hls4ml uses fixed-point representation for all computations
  - Operations are integer ops, but we can represent fractional values

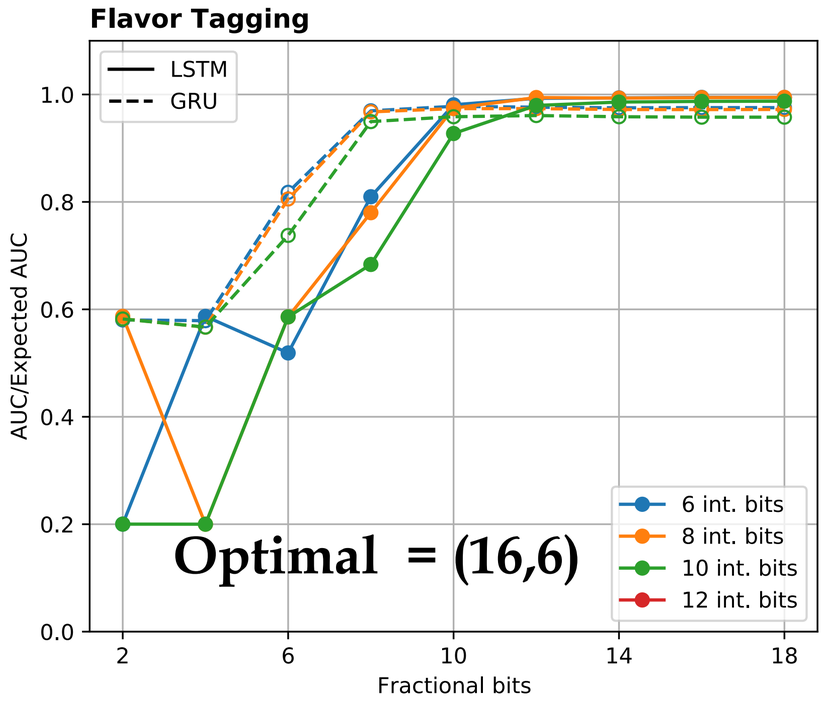


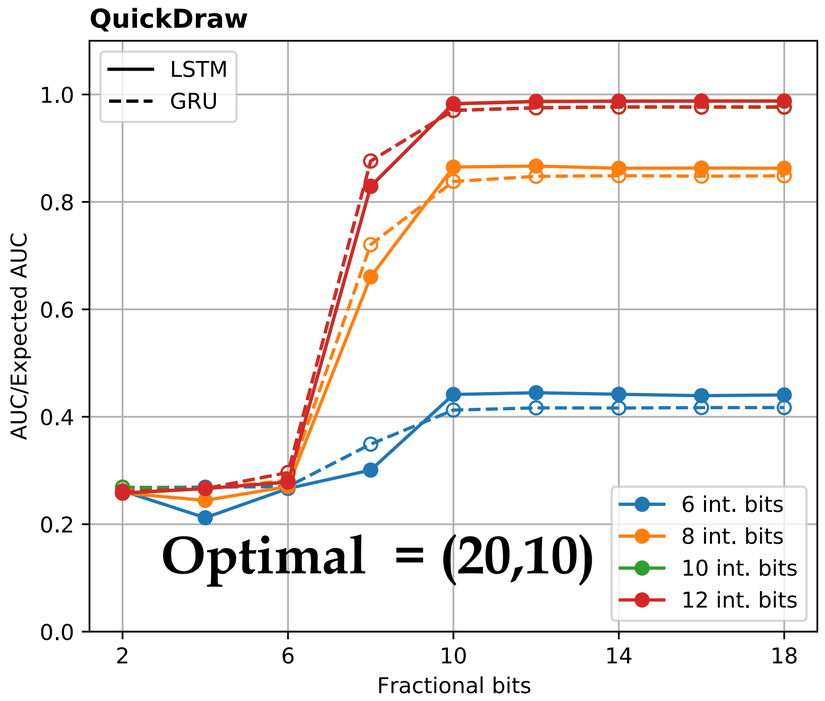
AUC after HLS conversion

$$\text{Relative AUC} = \frac{\text{AUC}_{\text{HLS}}}{\text{AUC}_{\text{Keras}}}$$

- Post-training quantized **LSTM models** (with optimal precision) performs similar to the floating-point models
- Small performance degradation (< 5%) in the **GRU models** after quantization





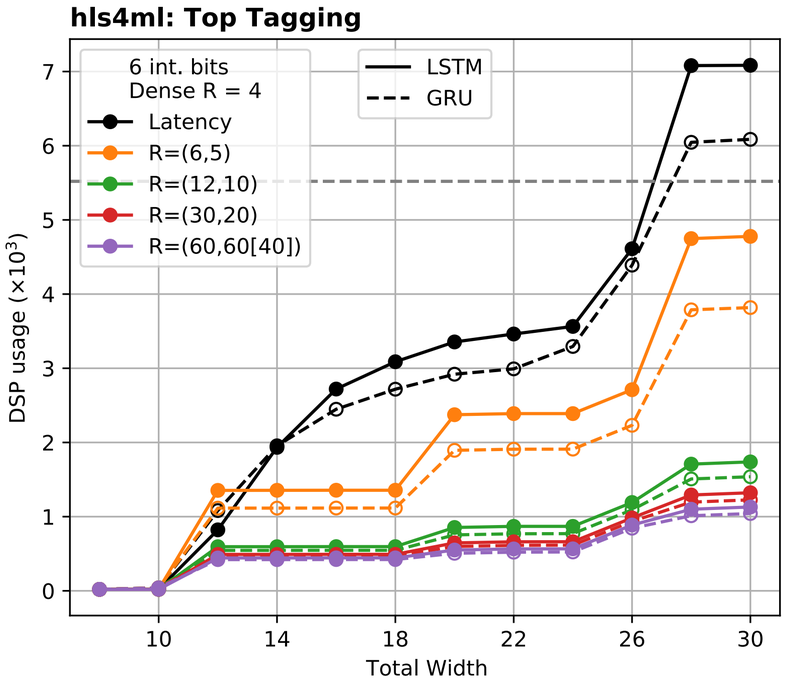


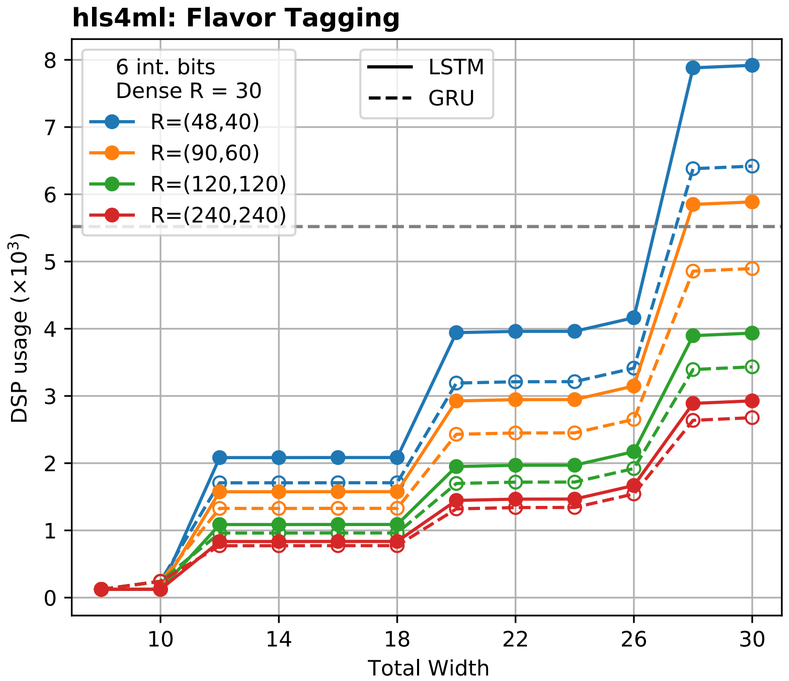
HLS synthesis: DSP usage

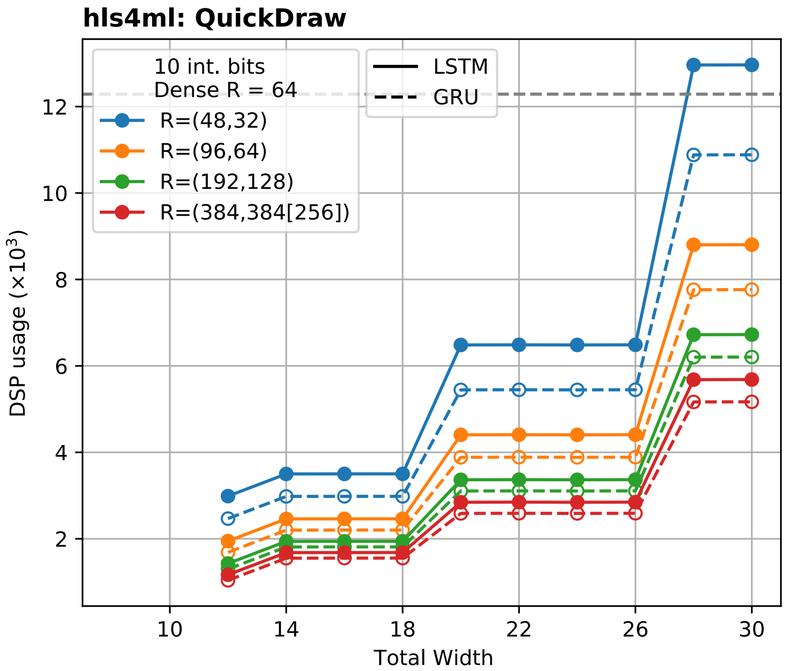
- **DSP usage** as a function of **Total bit width** after HLS synthesis
- The Jumps correspond to DSP input width

Synthesized using Xilinx Kintex UltraScale and Alveo U250 FPGA

FPGA part: xcku115-flvb2104-2-i, xcu250-figd2104-2-e





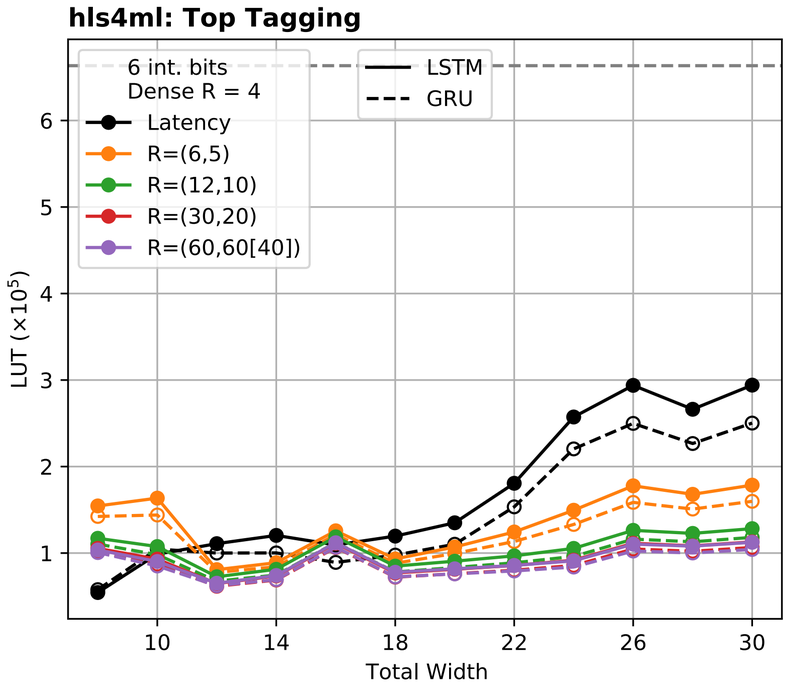


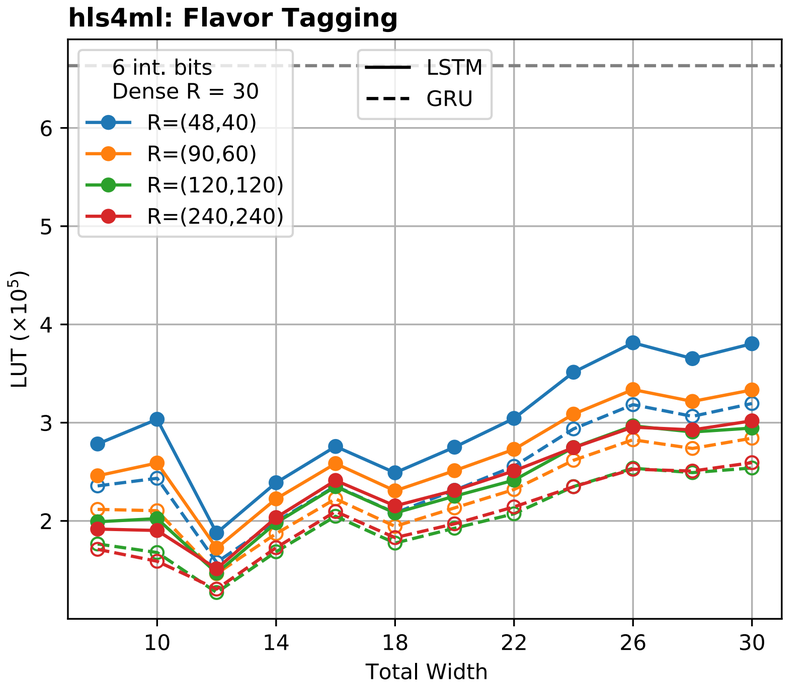
HLS synthesis: Lookup table usage

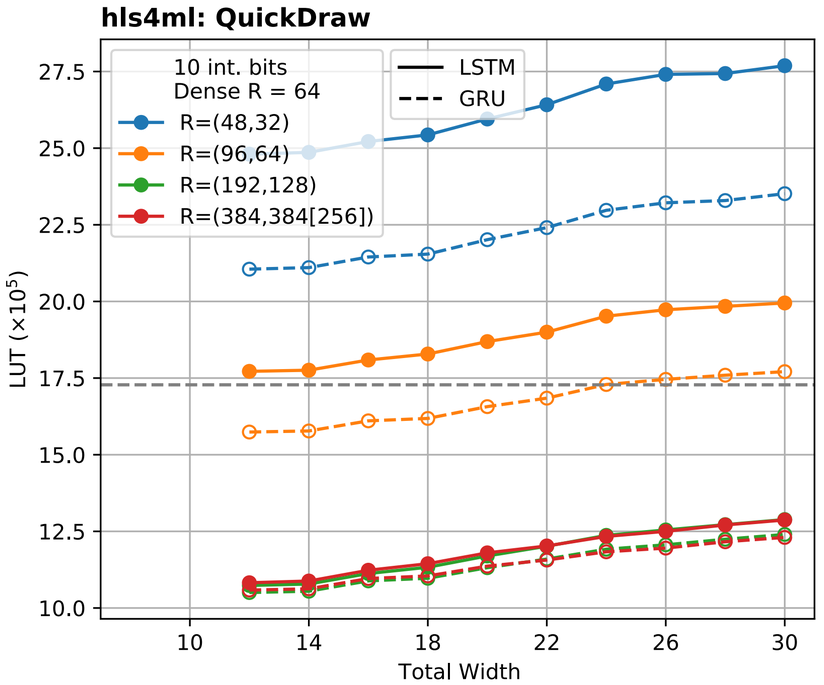
- **Lookup table usage** as a function of **Total bit width** after HLS synthesis
- Utilizations increases linearly with total width

Synthesized using Xilinx Kintex UltraScale and Alveo U250 FPGA

FPGA part: xcku115-flvb2104-2-i, xcu250-figd2104-2-e





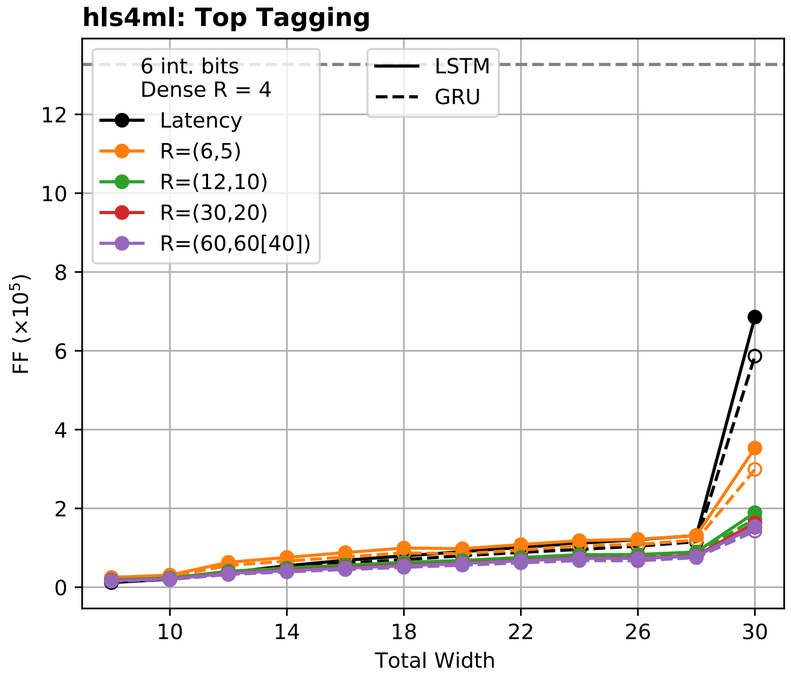


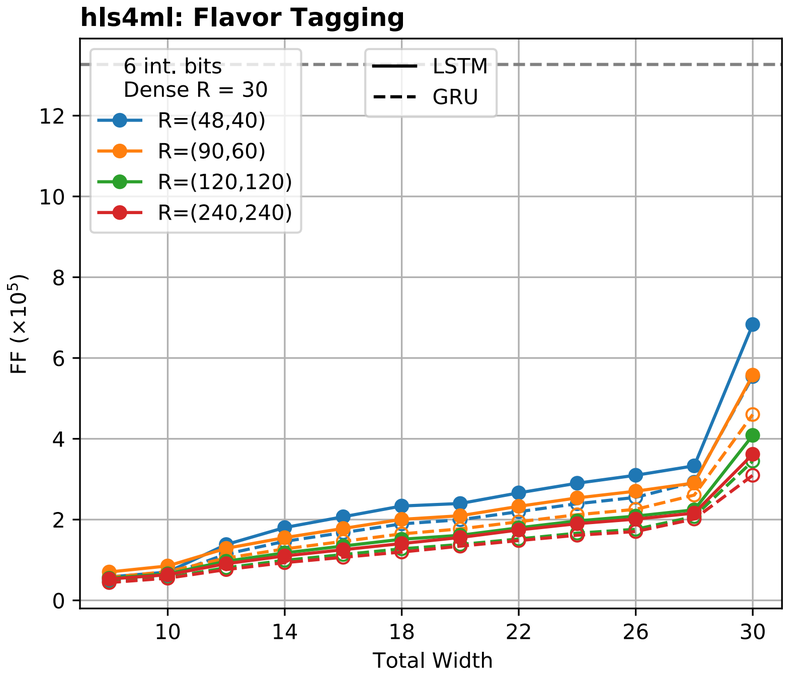
HLS synthesis: Flip-Flop usage

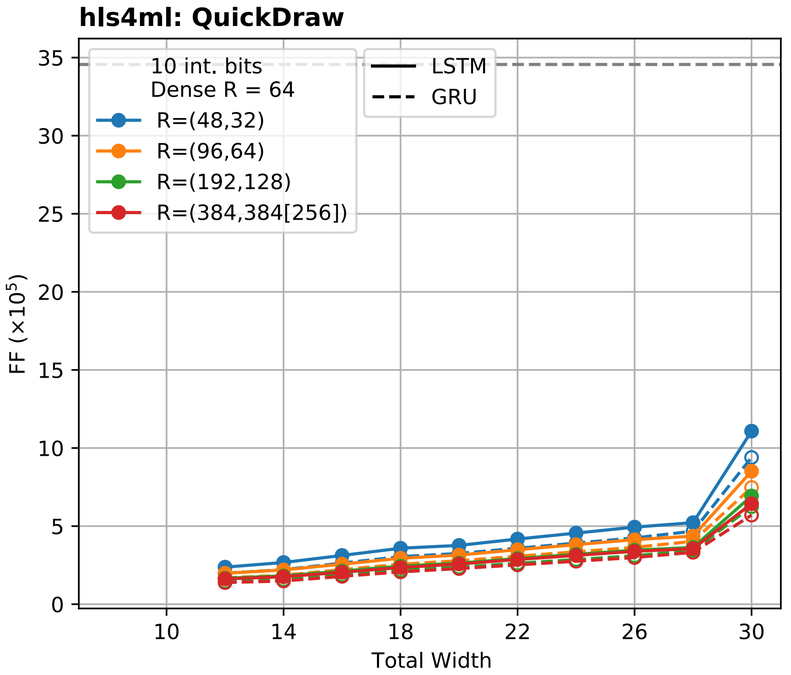
- **Flip-Flop usage** as a function of **Total bit width** after HLS synthesis
- Utilizations increases linearly with total width

Note we use Alveo U250 for the QuickDraw model

Synthesized using Xilinx Kintex UltraScale and Alveo U250 FPGA
FPGA part: xcku115-flvb2104-2-i, xcu250-figd2104-2-e

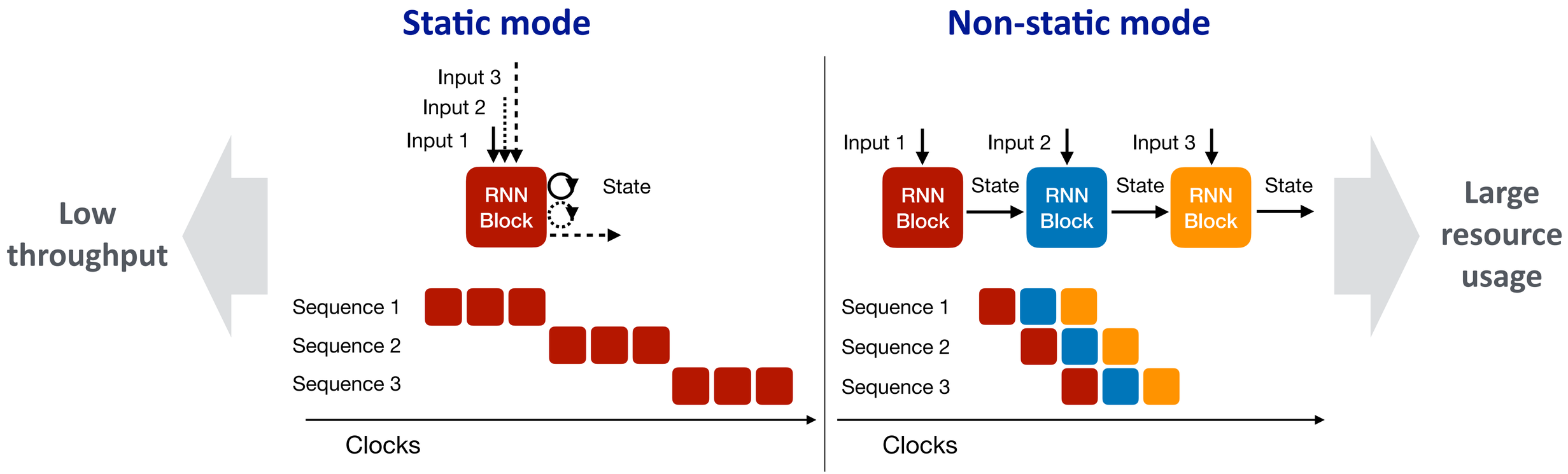






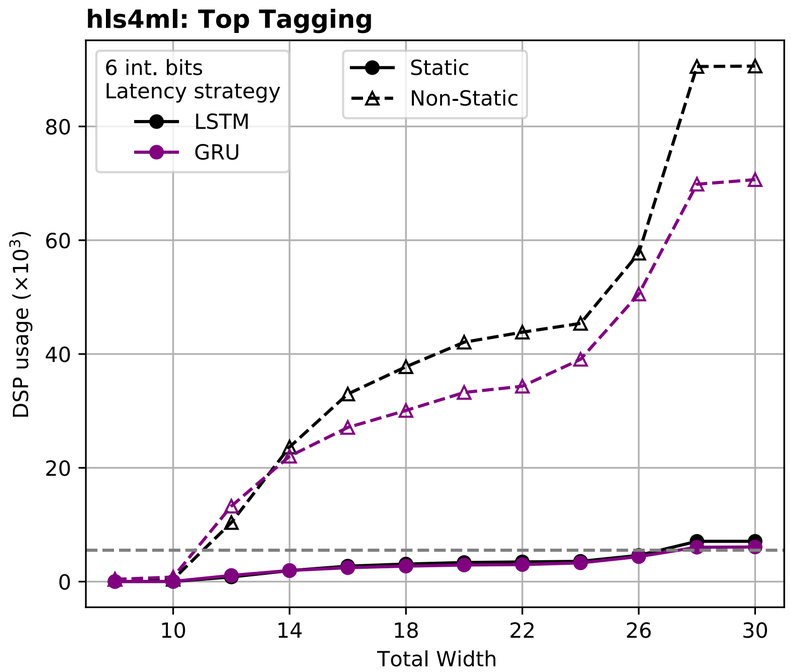
Static Vs Non-static Implementation

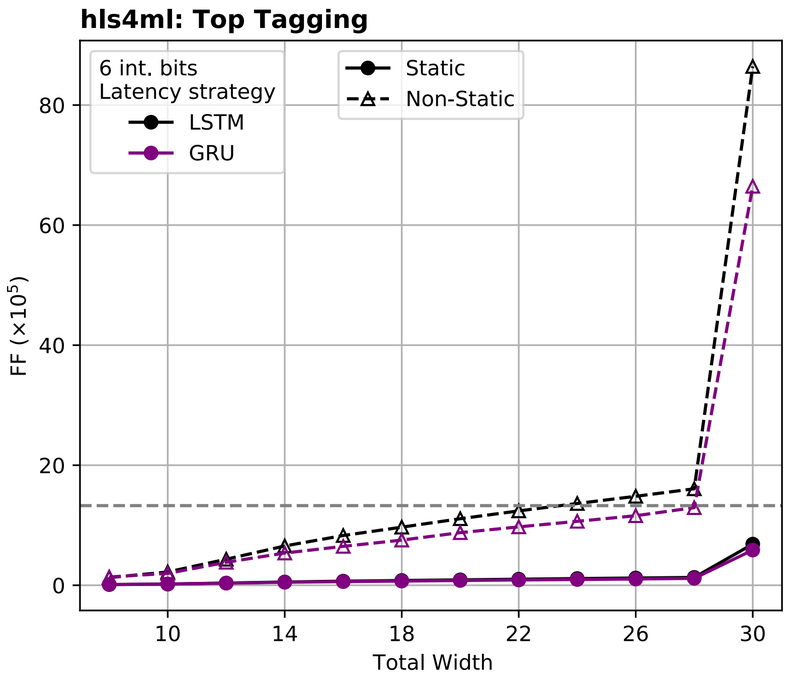
- **Static mode:** a single RNN block processes every input for every sequence
- **Non-Static mode:** RNN blocks for each input in the sequence
  - Necessary state vectors are passed from one RNN block to another
- Non-Static mode is expected to give larger throughput

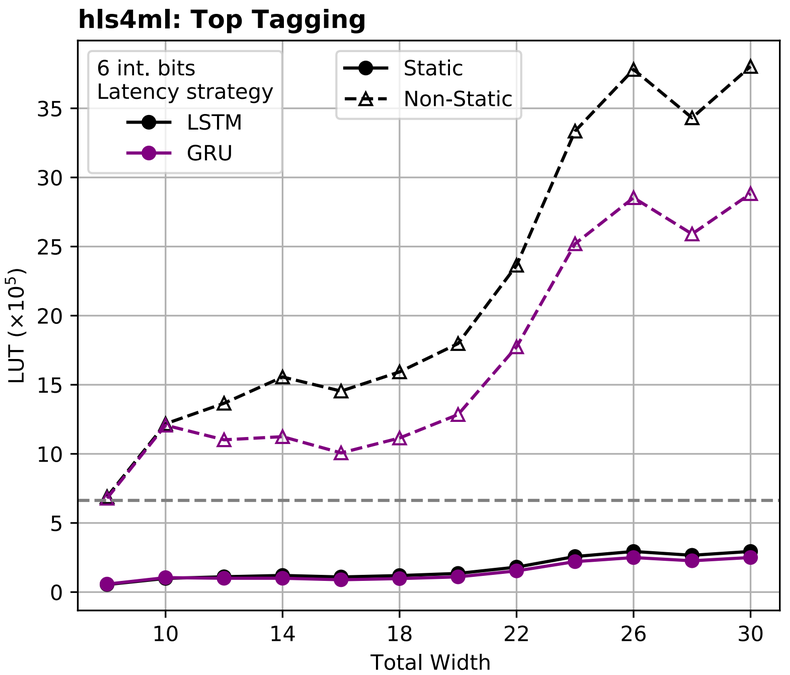


Resource utilization: Static Vs Non-static

- Only the Top Tagging models are synthesized using non-static mode
- Resource usage for non-static mode increases dramatically compared to static mode
- Non-static mode is able to fit within the available resources of the chip only for very small bit widths







Throughput: Static Vs Non-static

- The overall latency is similar in the two modes
- Significant reduction in Initiation Interval (II) for the non-static case
  - II reduced to 1 from 315 for the LSTM model
- Throughput increases in the non-static mode by a factor of 315 for the LSTM model

Table 5: Minimum and maximum latencies and initiation intervals for the Top Tagging model in both static and non-static mode.

Model	Static latency [μs]	Non-static latency [μs]	Static II	Non-static II
GRU	1.7 - 1.7	1.6 - 1.6	315	1
LSTM	1.6 - 1.6	1.5 - 1.5	314	1

Summary

- Support Keras/TensorFlow LSTM and GRU models with wide range of trainable parameters (3k - 130k)
- The implementation uses the existing matrix multipliers and activation functions in hls4ml
- **Two different modes:** static and non-static

- Minimal or no performance degradation after post training quantization
- Able to fit best modes on the target FPGA boards

Paper link: arXiv:2207.00559

LSTM and GRU layers are available on hls4ml tool

Future directions:

- Even a small RNN model can require a substantial amount of resources to implement
  - Quantization-aware training could be helpful to reduce resource usage
- More detailed studies for non-static mode of synthesis

Thank You

BACKUP

Latency Estimate

Table 2: Minimum and maximum latencies for the Top Tagging model.

Model	Latency [μs]	R=(6,5) [μs]	R=(12,10) [μs]	R=(30,20) [μs]	R=(60,60 [40]) [μs]
GRU	1.7 - 1.7	2.4 - 6.5	3.2 - 7.3	5.0 - 9.1	8.0 - 12.1
LSTM	1.7 - 1.7	2.7 - 6.8	3.5 - 7.6	5.3 - 9.4	8.3 - 12.4

Top Tagging model

Table 3: Minimum and maximum latencies for the Flavor Tagging model.

Model	R=(48,40) [μs]	R=(90,60) [μs]	R=(120,120) [μs]	R=(240,240) [μs]
GRU:	6.7 - 24.8	9.8 - 27.9	11.5 - 29.6	20.5 - 38.6
LSTM:	6.9 - 25.0	10.1 - 28.2	11.7 - 29.8	20.7 - 38.8

Flavor Tagging model

Table 4: Minimum and maximum latencies for the QuickDraw model.

Model	R=(48,32) [μs]	R=(96,64) [μs]	R=(192,128) [μs]	R=(384,384 [256]) [μs]
GRU	35.4 - 164.0	59.4 - 188.0	107.0 - 235.0	203.0 - 331.0
LSTM	35.9 - 164.0	59.9 - 188.0	107.0 - 236.0	203.0 - 332.0

QuickDraw model

Network Parameters

Top tagging: 3-4k

Flavor tagging: 50-65k

QuickDraw: 115-130k

Table 1: Network hyperparameters and total number of trainable parameters for different benchmark models.

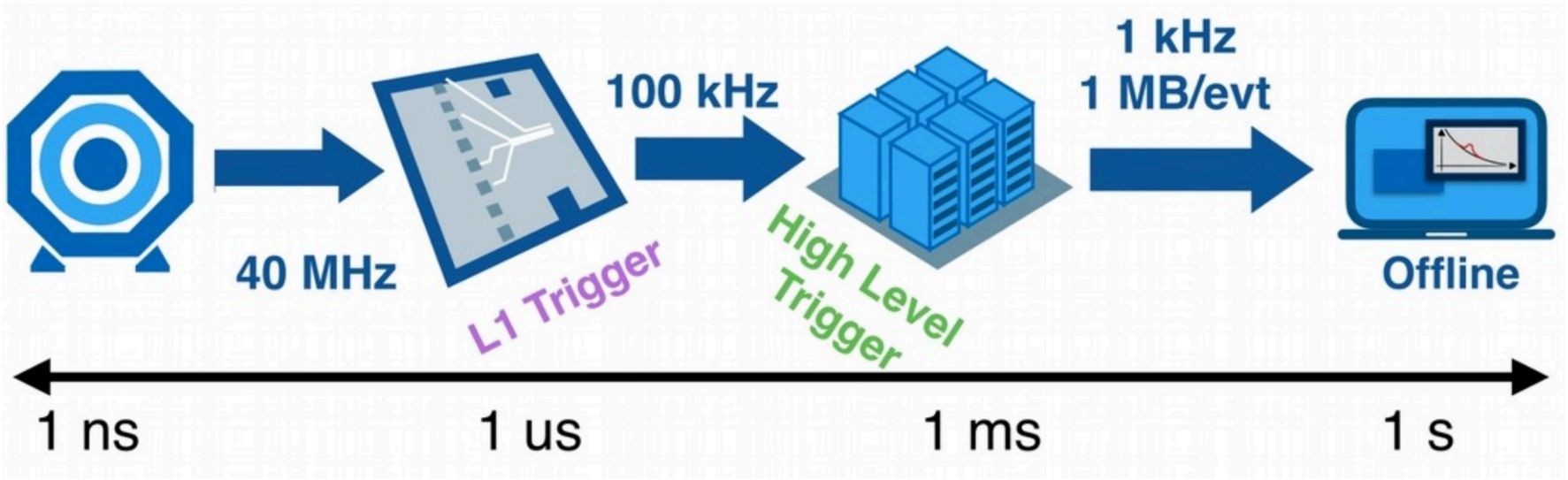
Benchmark	Sequence length	Input vector size	Hidden vector size	Dense layer sizes	output vector size	Trainable parameters		
						Non-RNN layers	LSTM	GRU
Top tagging	20	6	20	64	1	1,409	2,160	1,680
Flavor tagging	15	6	120	50/10	3	6,593	60,960	46,080
QuickDraw	100	3	128	256/128	5	66,565	67,584	51,072

QuickDraw dataset

- **Types of images:**
  - Bees
  - Butterflies
  - Mosquitos
  - Snails
  - Ants

- **Input features (per stroke):**
  - Pixel coordinates(x)
  - Pixel coordinates(y)
  - Time (t)

LHC Data processing



L1 Trigger (hardware: FPGAs) – *O(μs) hard latency*

- Typically coarse selections are applied

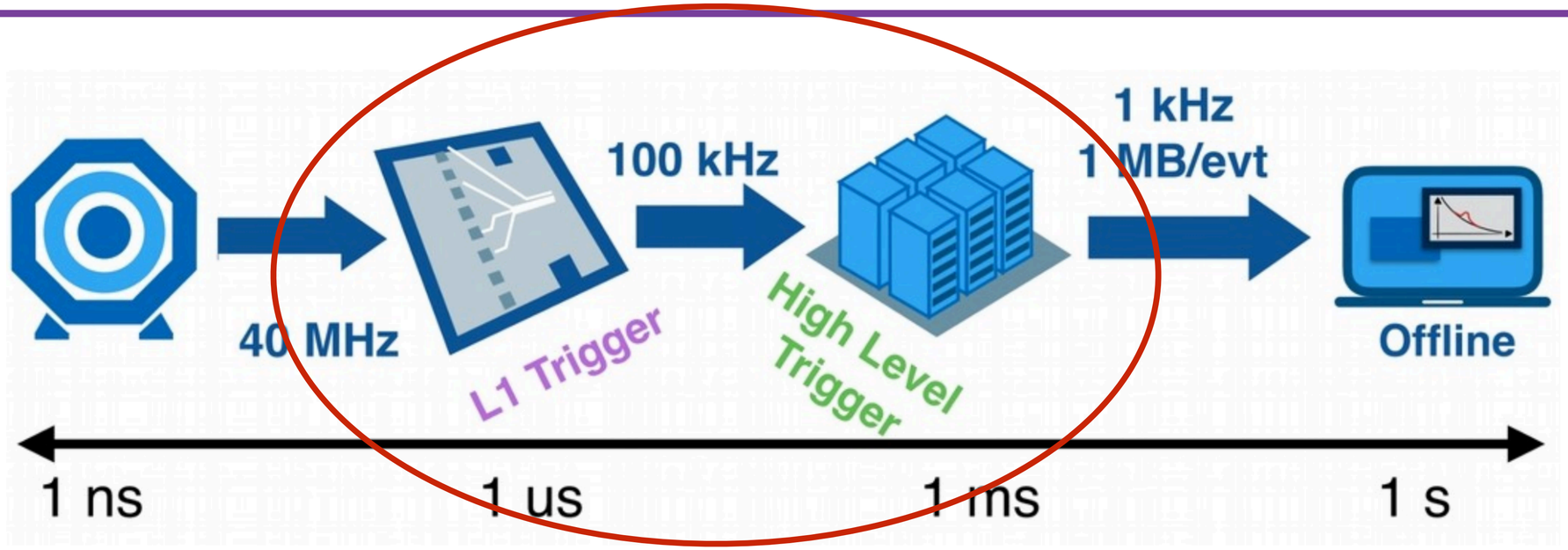
HLT (software: CPUs) – *O(100 ms) soft latency*

- More complex algorithms (full detector information available), some BDTs and DNNs used

Offline (software: CPUs)

- Full event reconstruction, bulk of machine learning usage in ATLAS/CMS

LHC Data processing

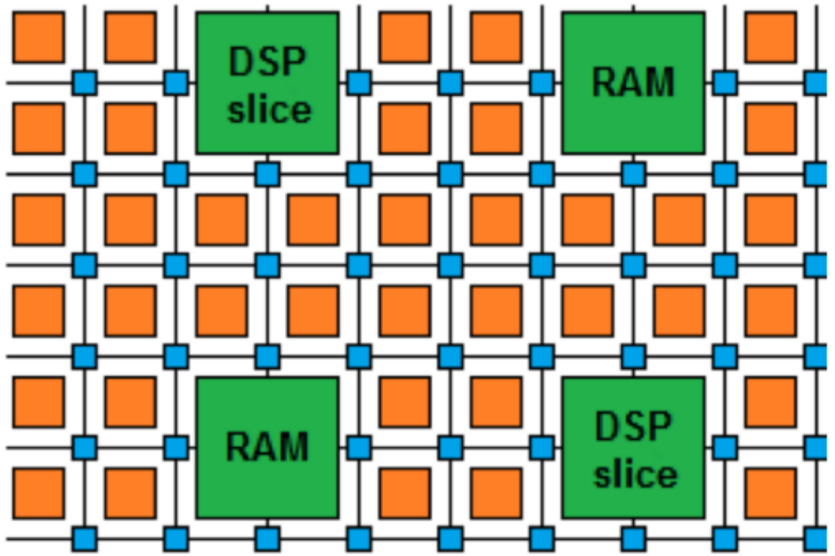


- DNNs have the potential to greatly improve physics performance in the trigger system
- In order to implement an algorithm, need to ensure inference latencies of μs (ms) for L1 (HLT)
  - For L1, this means we must use FPGAs

How can we run neural network inference quickly on an FPGA?

What are FPGAs?

Field **P**rogrammable **G**ate **A**rrays (FPGAs) are reprogrammable integrated circuits



- Contain many different building blocks (‘resources’) which are connected together as you desire

- Originally popular for prototyping ASICs, but now also for high performance computing

Building blocks:

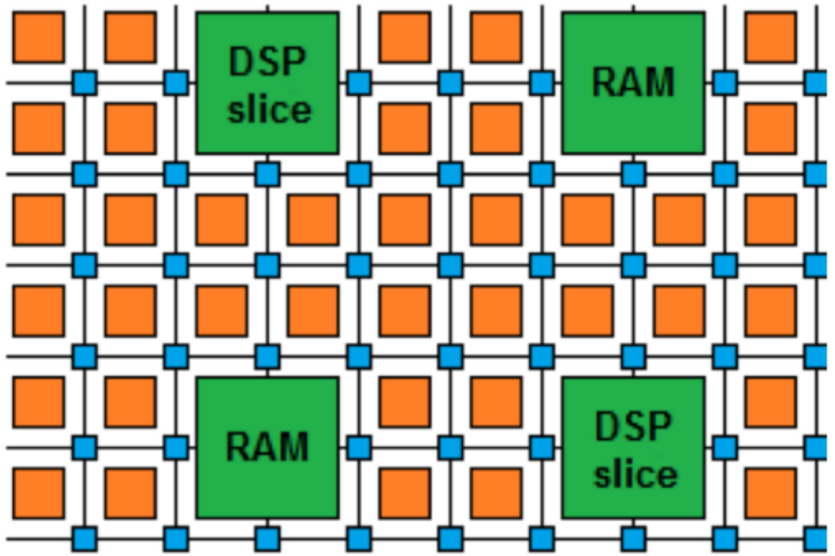
– **Multiplier units (DPSs)** [arithmetic]
– **Look Up Tables (LUTs)** [logic]
– **Flip-flops (FFs)** [registers]
– **Block RAMs (BRAMs)** [memory]

What are FPGAs?

- Algorithms are wired onto the chip

- Run at high frequency - *O(100 MHz)*
  - Can compute outputs in O(ns)

- Programming traditionally done in Verilog/VHDL
  - Low-level hardware languages
- Possible to translate C to Verilog/VHDL using High Level Synthesis (HLS) tools

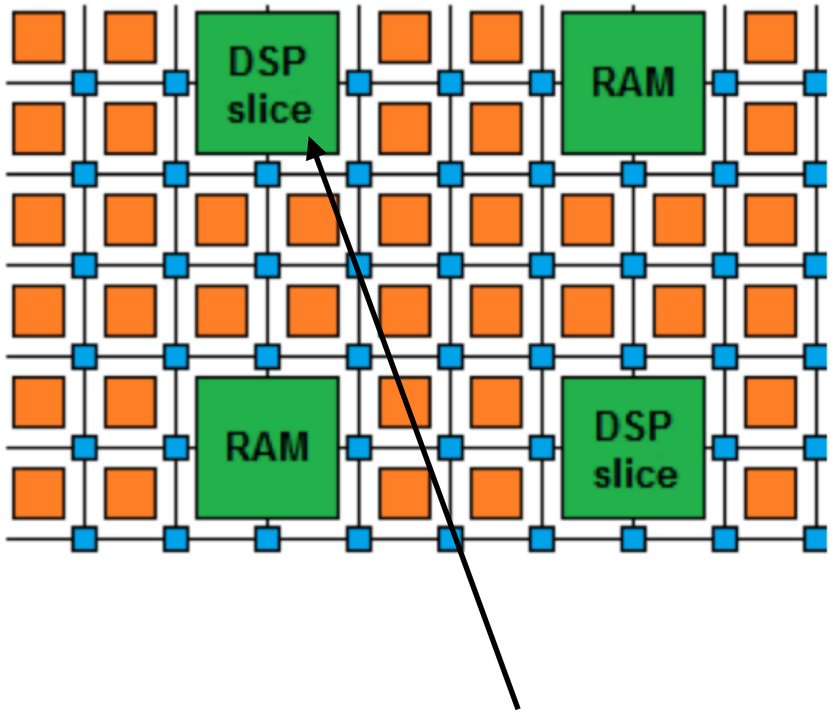


Building blocks:

– **Multiplier units (DPSs)** [arithmetic]
– **Look Up Tables (LUTs)** [logic]
– **Flip-flops (FFs)** [registers]
– **Block RAMs (BRAMs)** [memory]

What are FPGAs?

- **DSPs** (Digital Signal Processor) are specialized units for multiplication and arithmetic
- Faster and more efficient than using LUTs for these types of operations
- And for Neural Nets, DSPs are often the most scarce



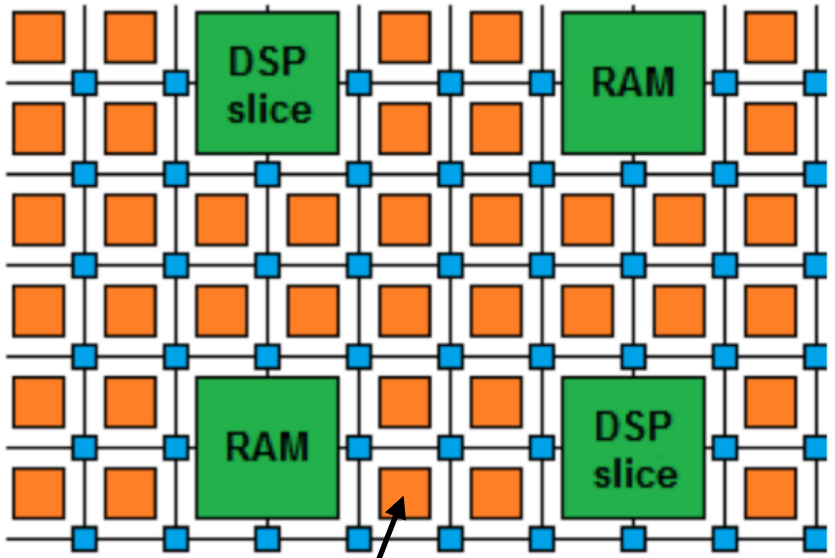
DSP
(multiplication)

Building blocks:

- **Multiplier units (DPSs)** [arithmetic]
- **Look Up Tables (LUTs)** [logic]
- **Flip-flops (FFs)** [registers]
- **Block RAMs (BRAMs)** [memory]

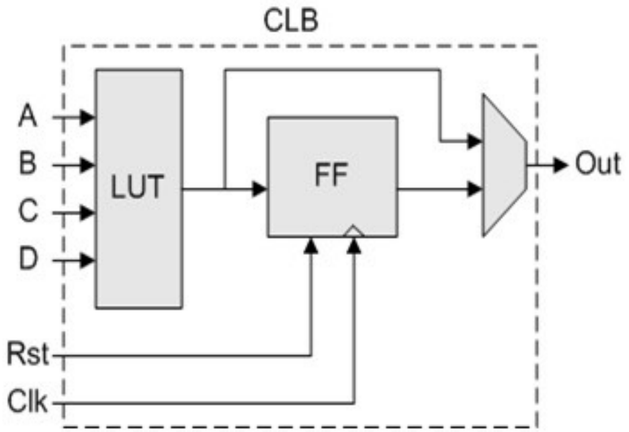
What are FPGAs?

- **Logic cells / Look Up Tables** perform arbitrary functions on small bitwidth inputs (2-6)
- These can be used for boolean operations, arithmetic, small memories
- **Flip-Flops** register data in time with the clock pulse

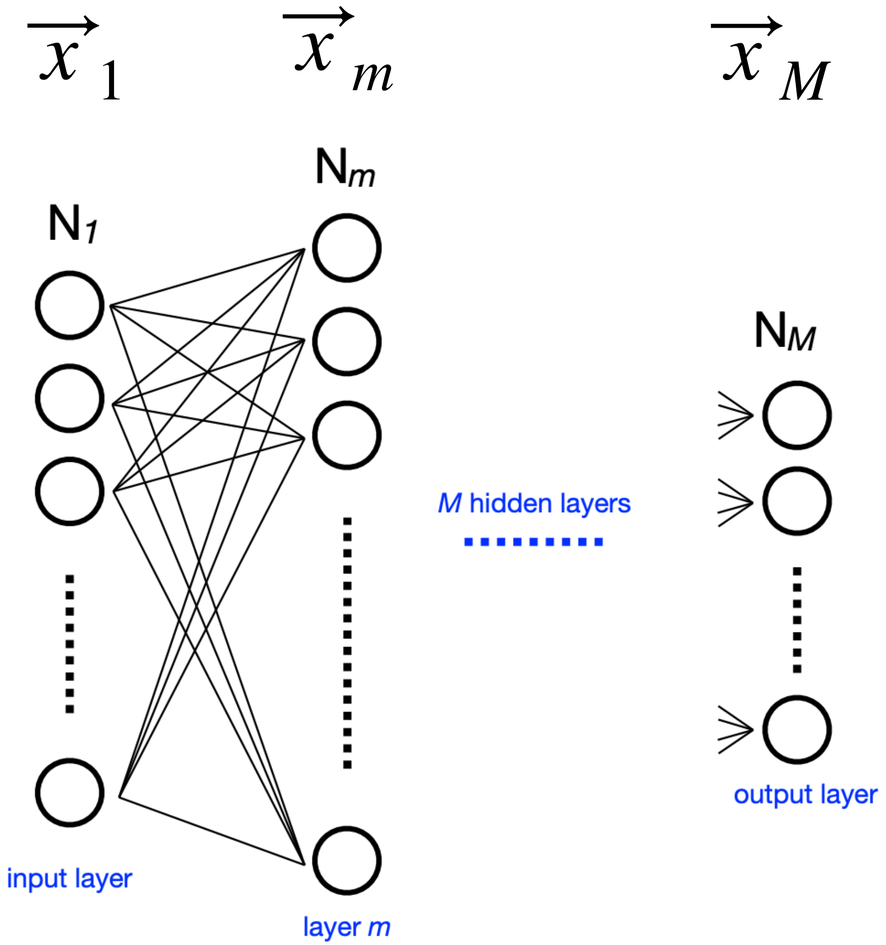


Building blocks:

- **Multiplier units (DPSs)** [arithmetic]
- **Look Up Tables (LUTs)** [logic]
- **Flip-flops (FFs)** [registers]
- **Block RAMs (BRAMs)** [memory]



Inference on an FPGA



$$\vec{x}_m = g_m\left(W_{m,m-1}\vec{x}_{m-1} + \vec{b}_m\right)$$

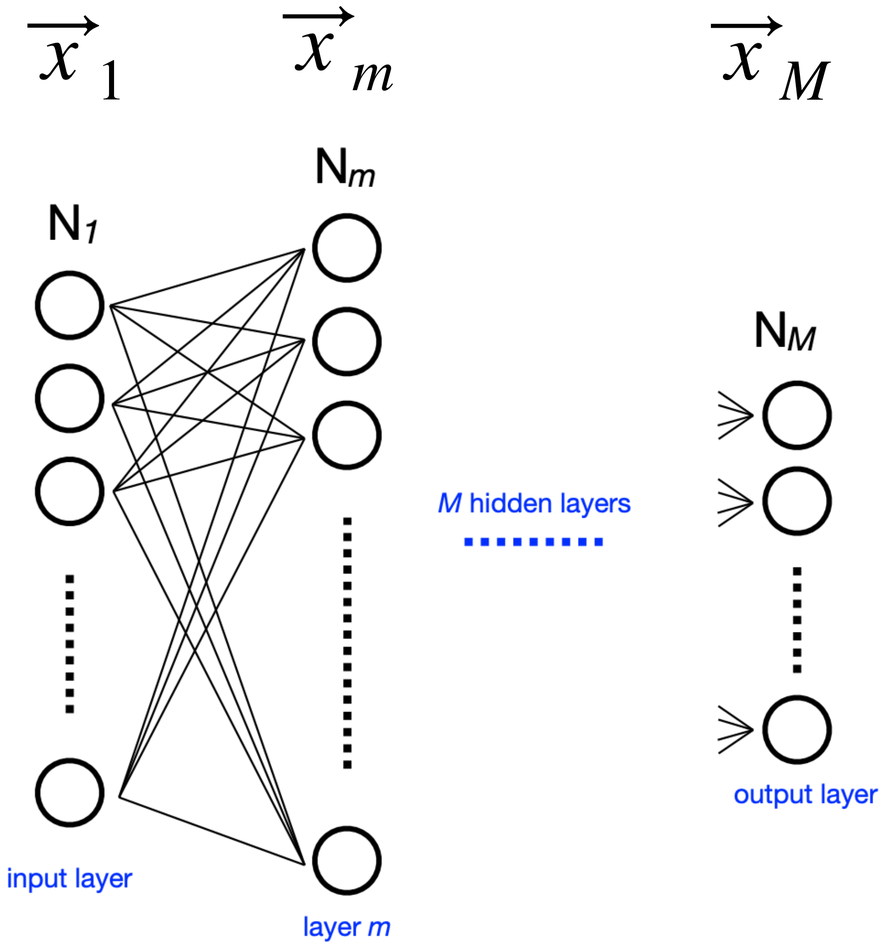
multiplication

activation function

addition

Credit: Dylan Rankin

Inference on an FPGA



$$\vec{x}_m = g_m\left(W_{m,m-1}\vec{x}_{m-1} + \vec{b}_m\right)$$

multiplication

activation function

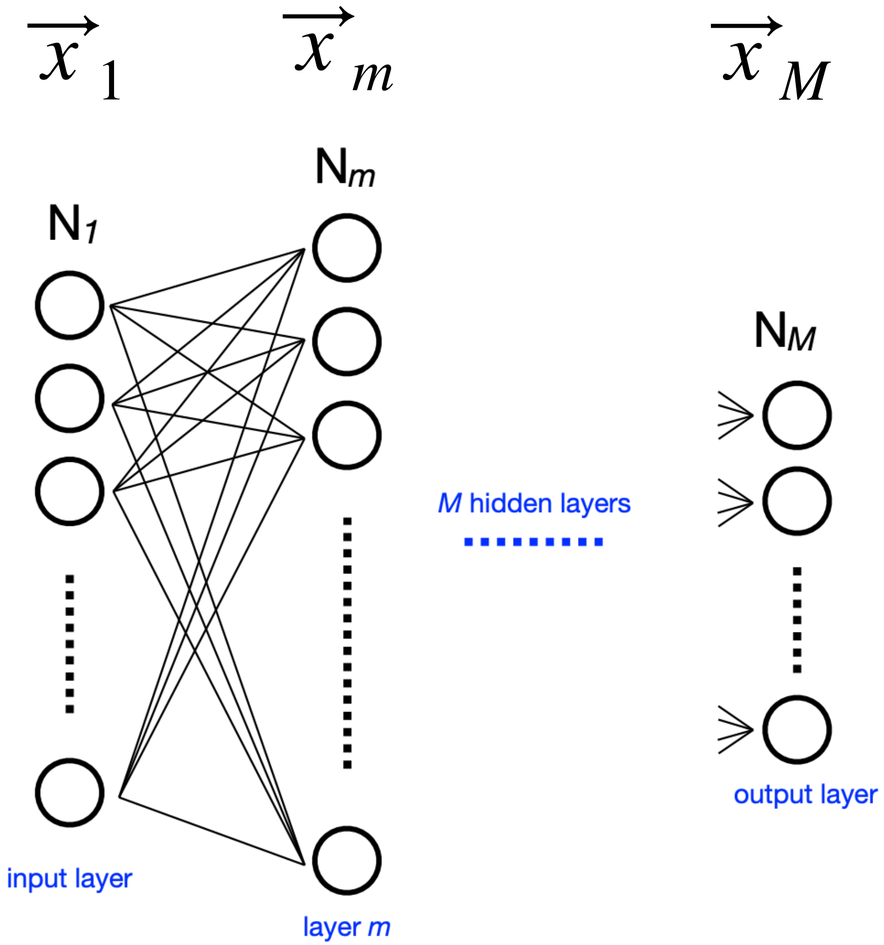
addition

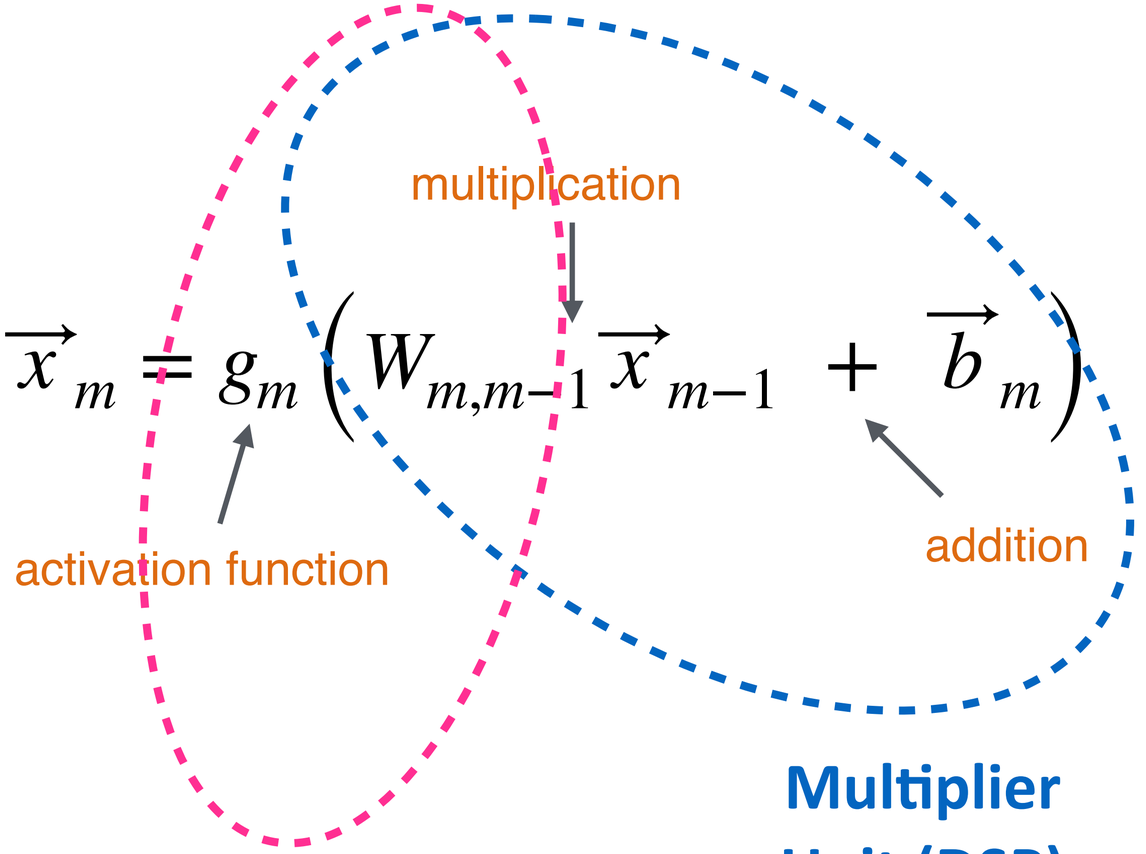
Multiplier Unit (DSP)

up to ~6k parallel operation (VU9P)

Credit: Dylan Rankin

Inference on an FPGA



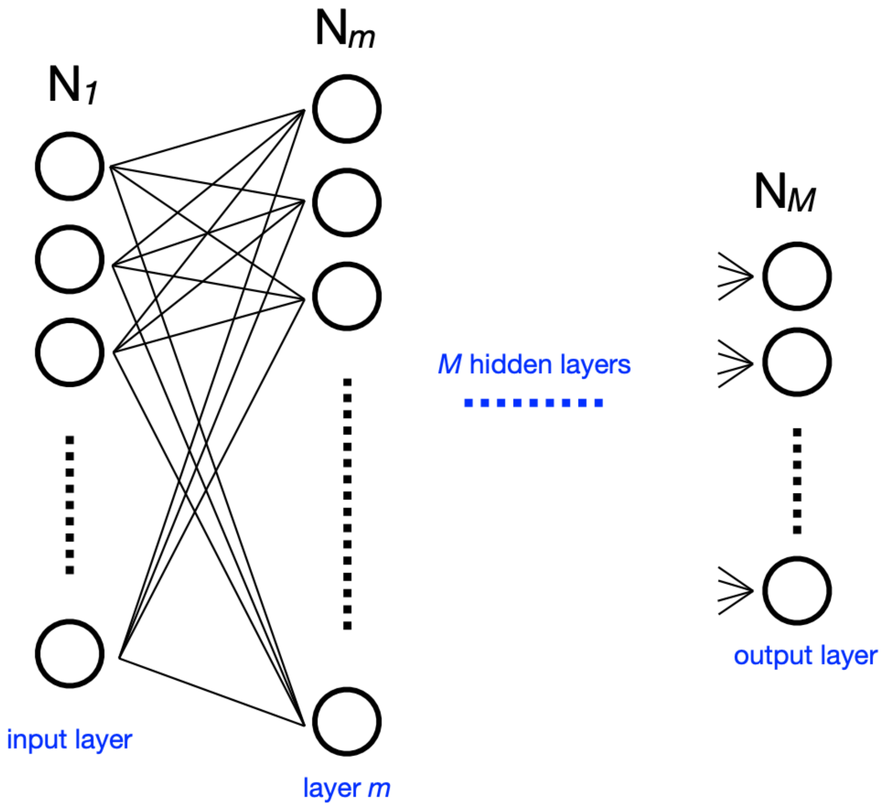


Credit: Dylan Rankin

Inference on an FPGA

Every clock cycle
(all layer operations can be
performed simultaneously)

$$\vec{x}_1 \rightarrow \vec{x}_m \rightarrow \vec{x}_M$$



multiplication

$$\vec{x}_m = g_m\left(W_{m,m-1}\vec{x}_{m-1} + \vec{b}_m\right)$$

activation function

addition

Credit: Dylan Rankin

High Level Synthesis with Machine Learning (hls4ml)

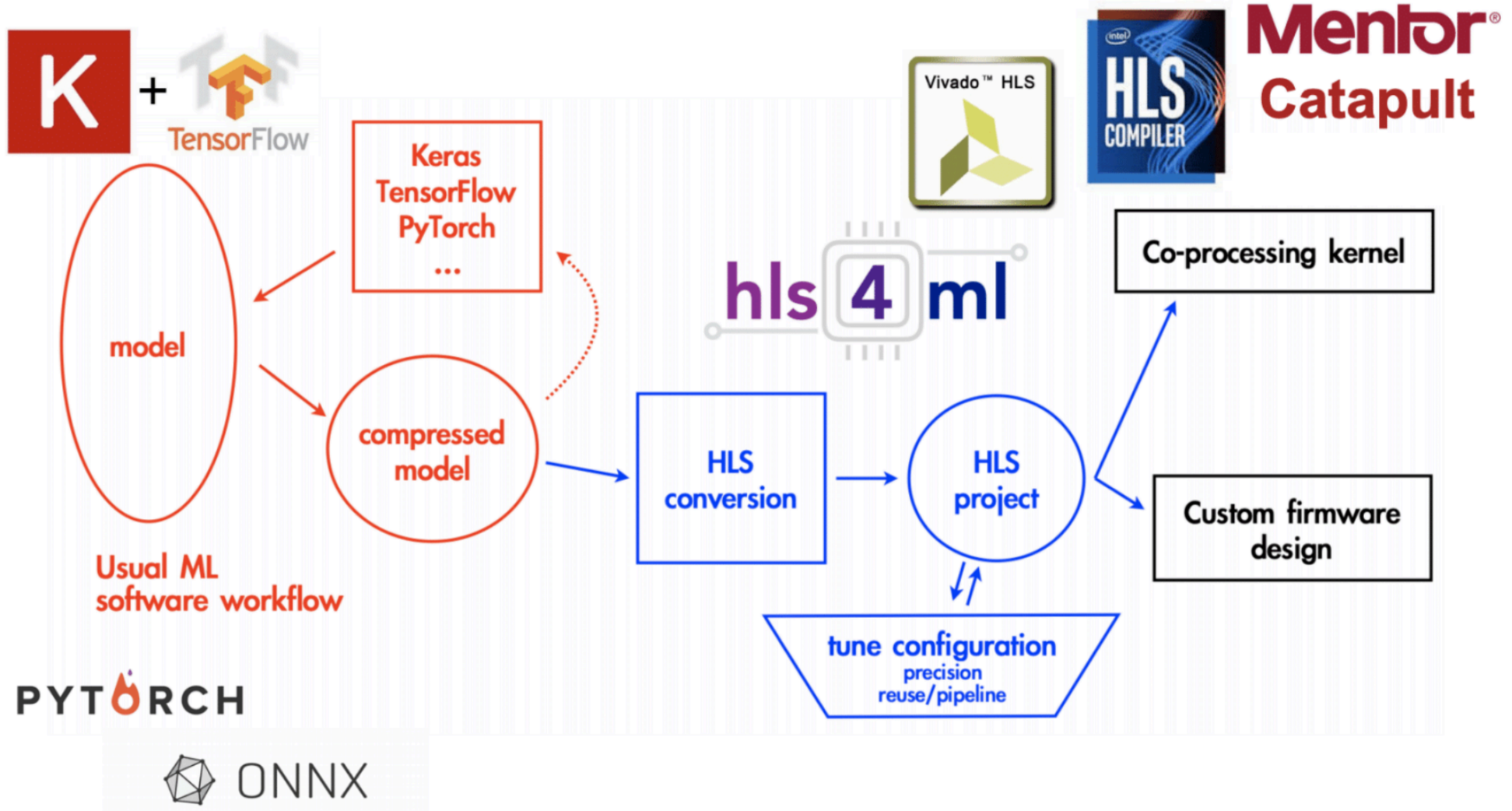
hls4ml is a software package for creating implementations of neural networks for FPGAs and ASICs

https://fastmachinelearning.org/hls4ml/
arXiv:2103.05579

- Supports common layer architectures and model software, options for quantization/pruning

- Output is a fully ready high level synthesis (HLS) project
- Customizable output

hls4ml Work Flow



Quantization

Quantization – Reducing the bit precision used for NN arithmetic

Why this is necessary?

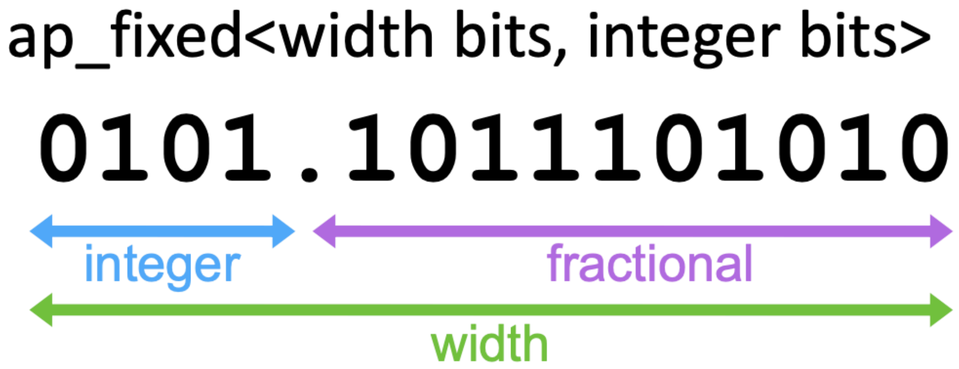
- Software assumes all computations performed with floating-point (32 bit) arithmetic
- Not always necessary for desired performance

- hls4ml uses fixed-point representation for all computations
  - Operations are integer ops, but we can represent fractional values
- Bitwidth can be adjusted as needed (impacts **performance**, resources)

ap_fixed<width bits, integer bits>

Quantization

- Reduction of precision automatically zeros very small weights ($w < 2^{-\text{fractional}}$)
- Also reduces resources needs to compute/store multiplications and intermediate layers

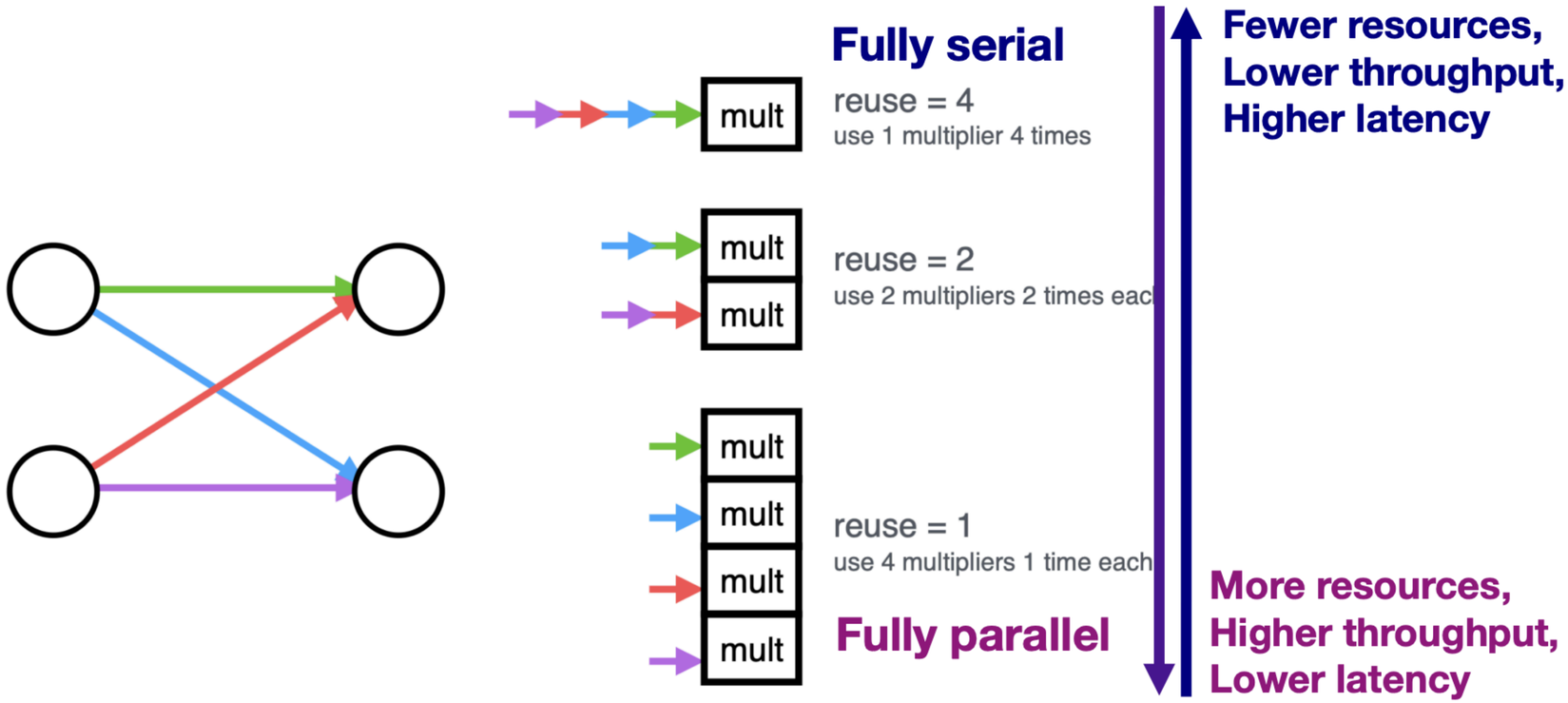


- Can greatly reduce size of network by training with knowledge of quantization
- Quantization-aware training - QKeras + support in hls4ml: [arXiv: 2006.10159]

Parallelization

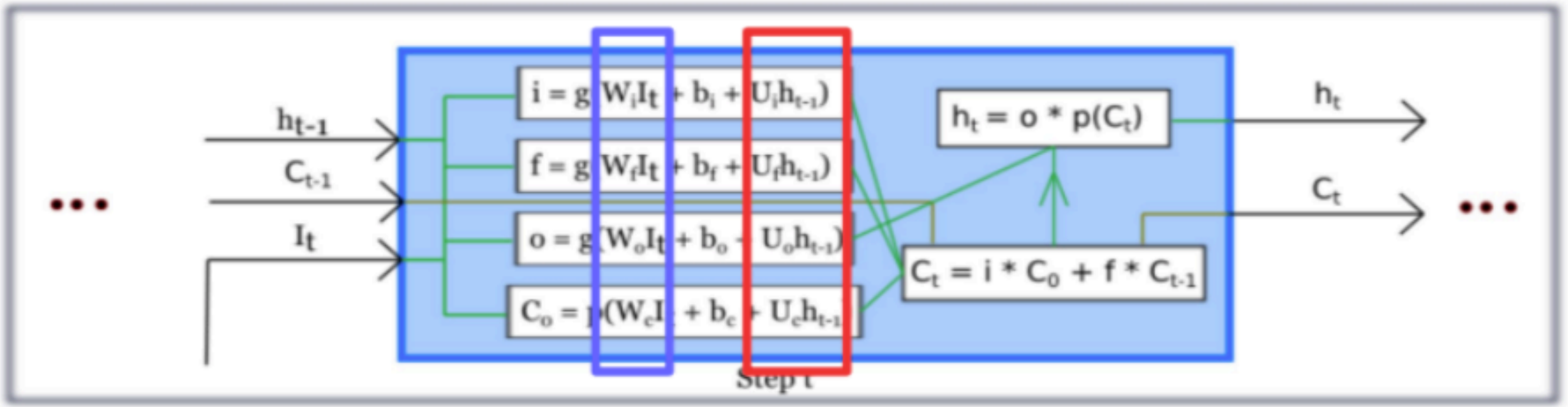
- Trade-off between latency and FPGA resource usage determined by the parallelization of the calculations in each layer

- Configure the "reuse factor" = number of times a multiplier is used to do a computation



LSTM

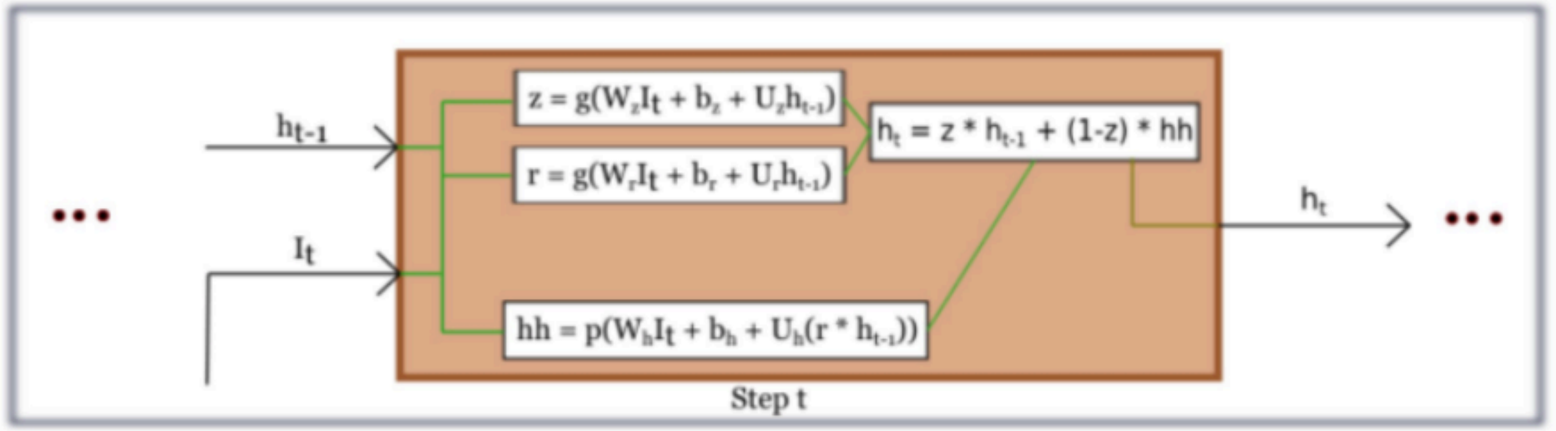
The structure of the LSTM cell is the following:



The **g** is the recurrent activation, **p** is the activation **W**s are the kernels, **U**s are the recurrent kernels, **h** is the hidden variable which is the output too and the notation * is an element-wise multiplication.

GRU

The structure of the GRU cell is the following:



The meaning of the notations are the same as in case of LSTM. 1-z means an element-wise subtraction.