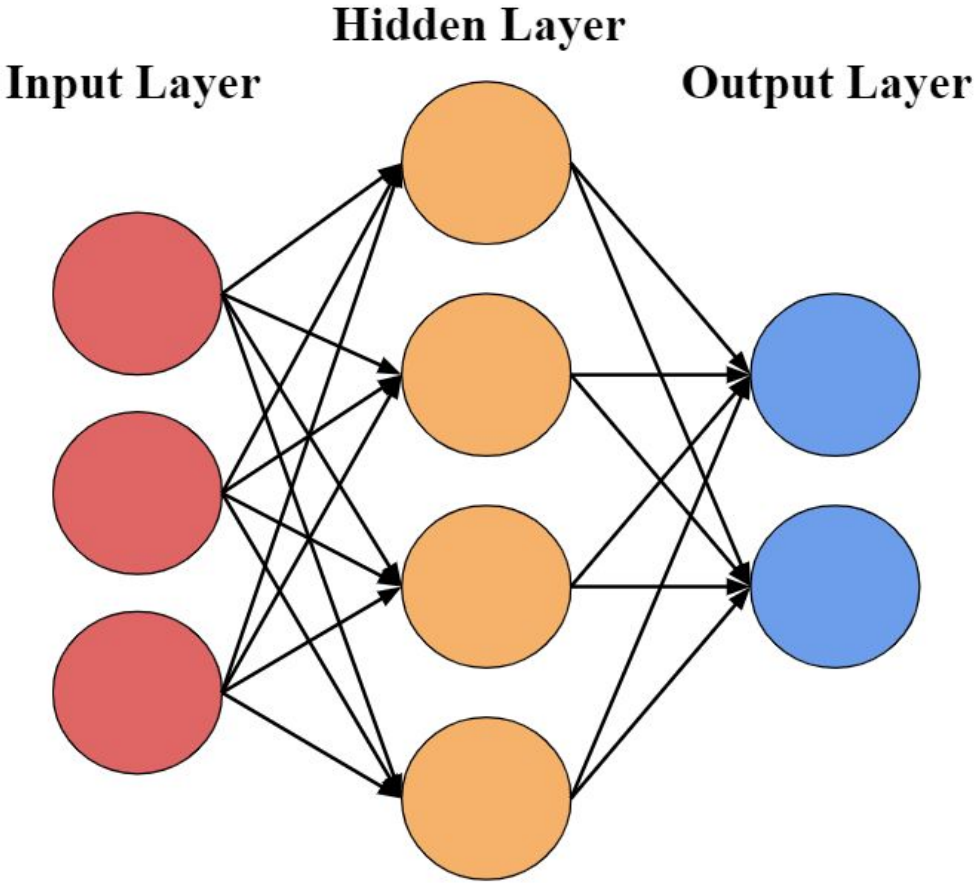
Implementation of Long Short-Term Memory Neural Networks in High-Level Synthesis Targeting FPGAs

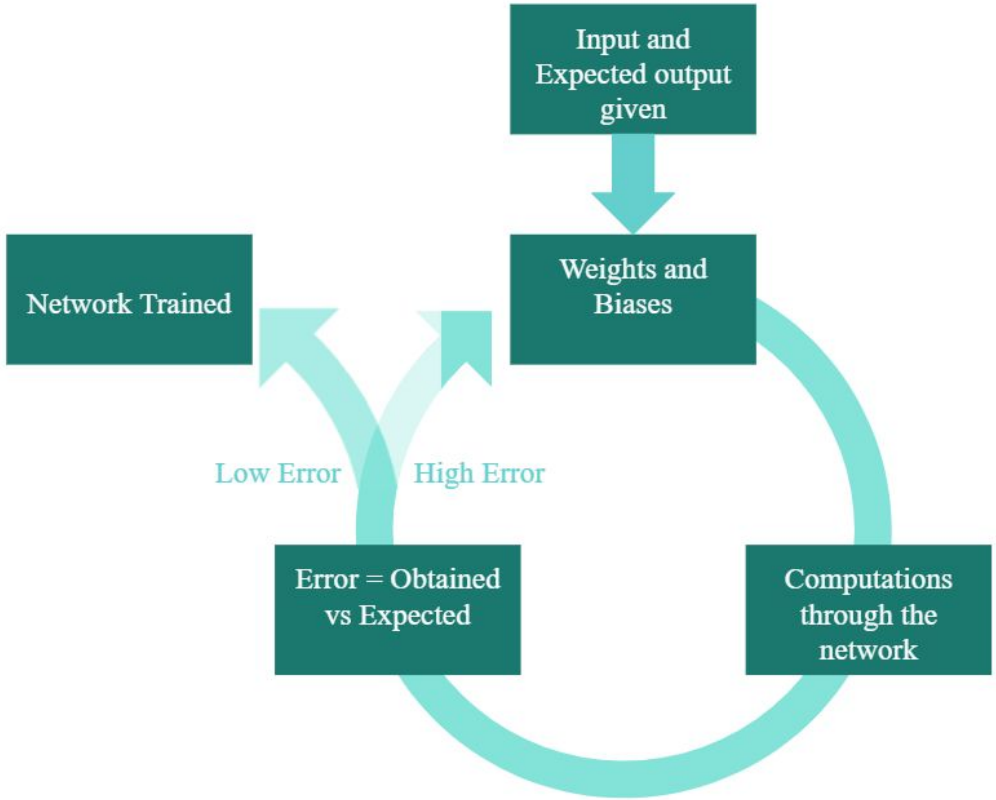
Richa Rao
University of Washington
June 9th, 2020

Neural Networks



Training and Inference

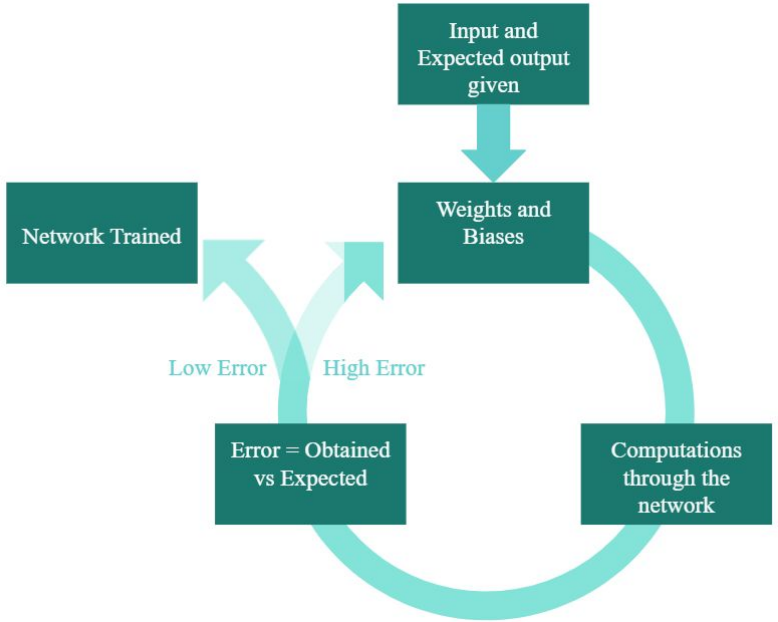
- **Training:** Process by which neural networks learn



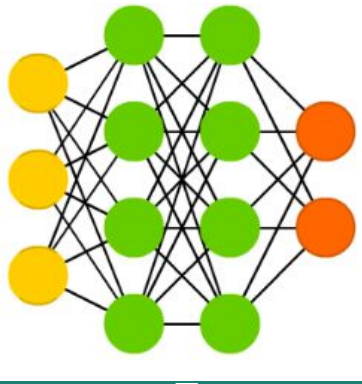
Training and Inference

- **Training:** Process by which neural networks learn

- **Inference:** Using trained networks for prediction



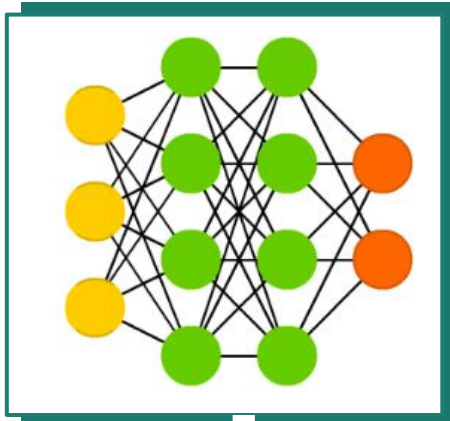
Types of Neural Networks



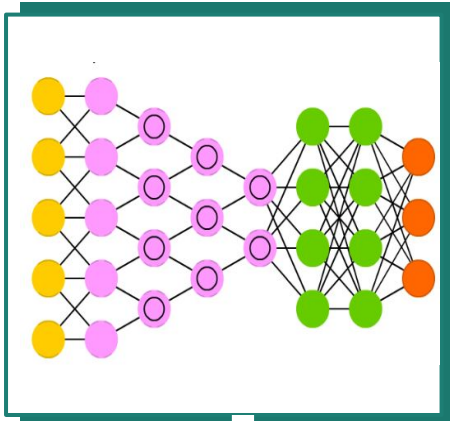
- Deep Neural Networks
- More than 1 hidden layer

Image Source

Types of Neural Networks



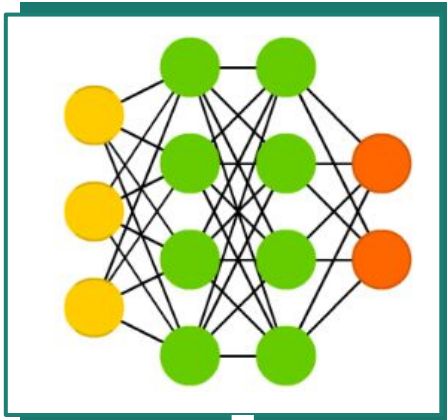
- Deep Neural Networks
- More than 1 hidden layer



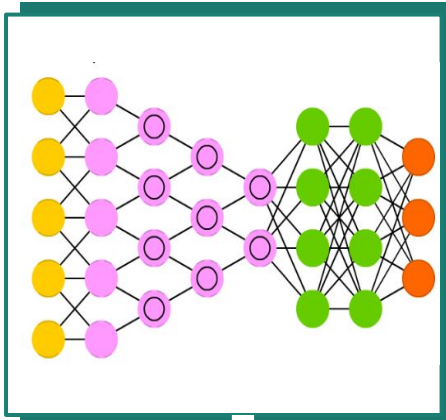
- Convolutional Neural Networks
- Convolution layers
- Work well with image as input

Image Source

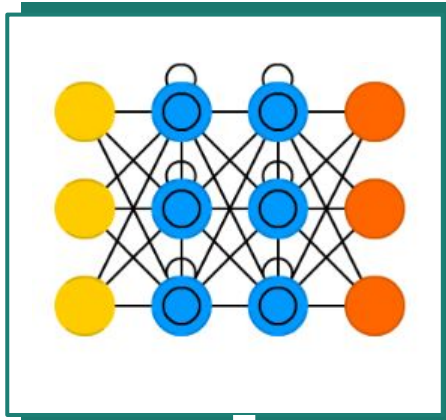
Types of Neural Networks



- Deep Neural Networks
- More than 1 hidden layer



- Convolutional Neural Networks
- Convolution layers
- Work well with image as input

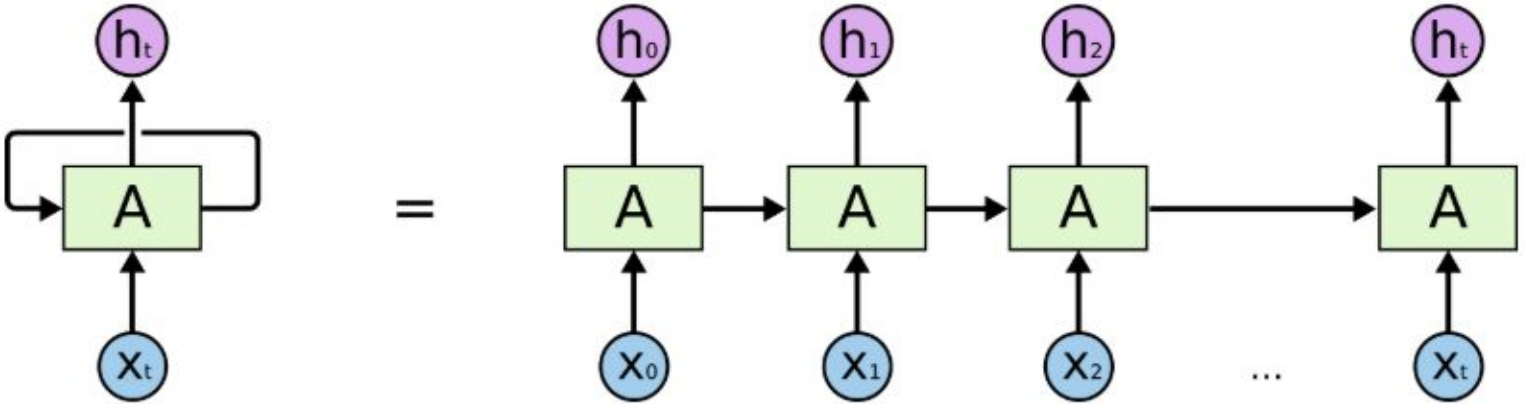


- Long Short-Term Memory NN
- Memory Cell
- Work well with sequence of data as input

Image Source

Long Short-Term Memory Neural Networks

- $x_t \rightarrow$ Input
- A $\rightarrow$ LSTM Cell
- $H_t \rightarrow$ Output



Unrolled LSTM network

Image Source

Long-Term dependencies

I grew up in France. I speak _____

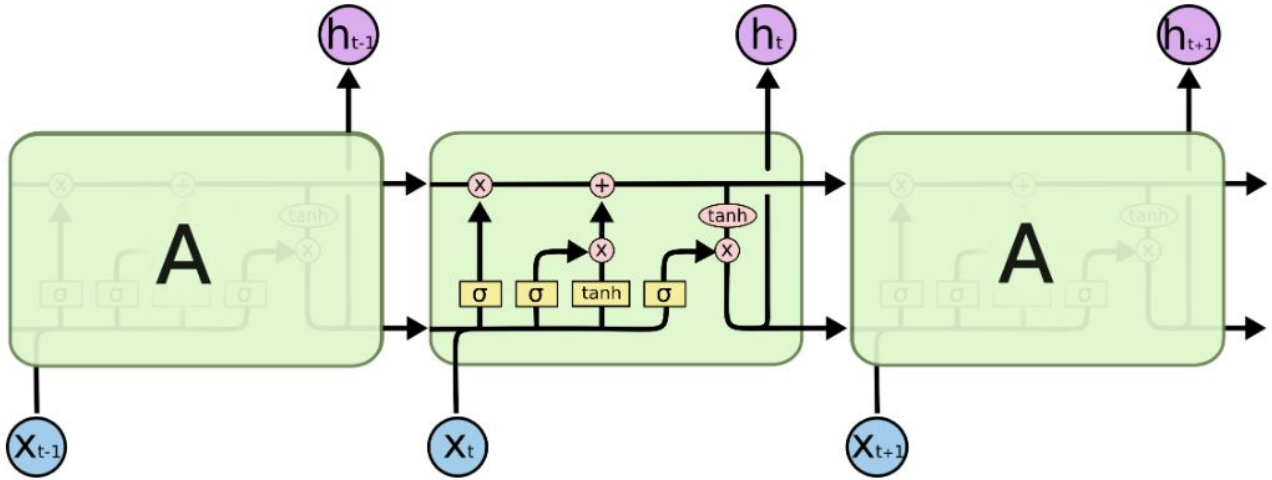
Long-Term dependencies

I grew up in France. I speak *french*

Long-Term dependencies

LSTM Cell

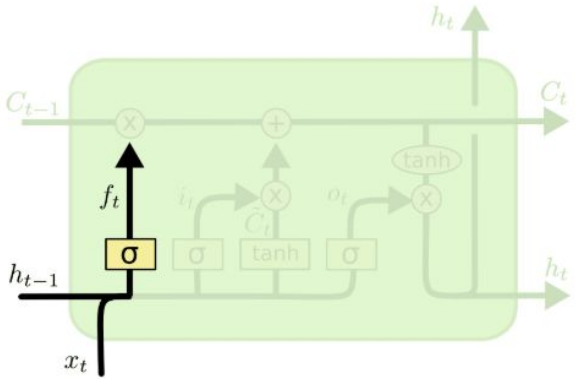
- Allow long-term dependencies between data by deciding-
  - Information to get of at each timestep
  - Information to carry to the next timestep
  - Output at each timestep



Unrolled LSTM network with contents of LSTM Cell

Image Source

I. Information to get rid of at each timestep



Working of Forget Gate

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

x_t → Input

h_{t-1} → Output from previous timestep

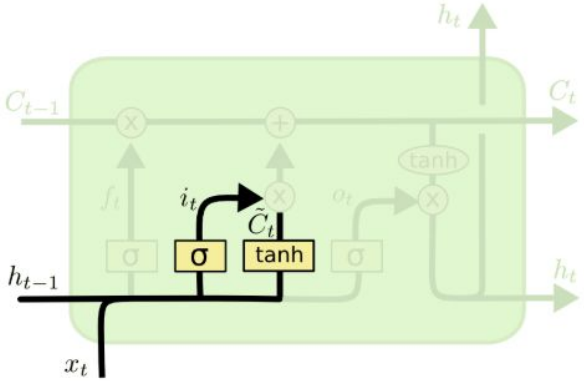
W_f, b_f → Weights and Biases

f_t → Forget gate

$\boldsymbol{\sigma}$ → Sigmoid Activation

Image Source

II. Information to carry to the next timestep



Working of Input Gate

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

x_t → Input

h_{t-1} → Output from previous timestep

W_i, b_i , W_C, b_C → Weights and Biases

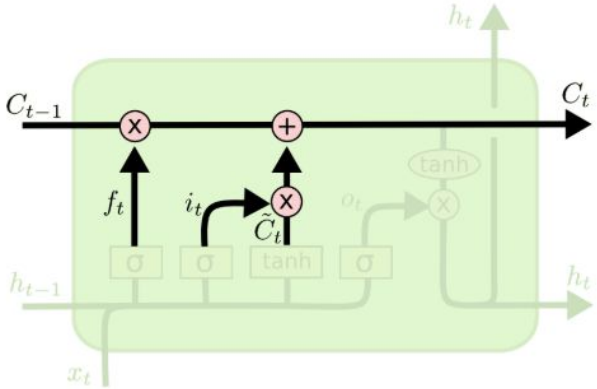
i_t → Input gate

$\boldsymbol{\sigma}$, tanh → Sigmoid, Tanh Activation

$\tilde{C}_t$ → New vector values

Image Source

II. Information to carry to the next timestep



Cell State

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

C_t → Cell State

C_{t-1} → Cell state from previous timestep

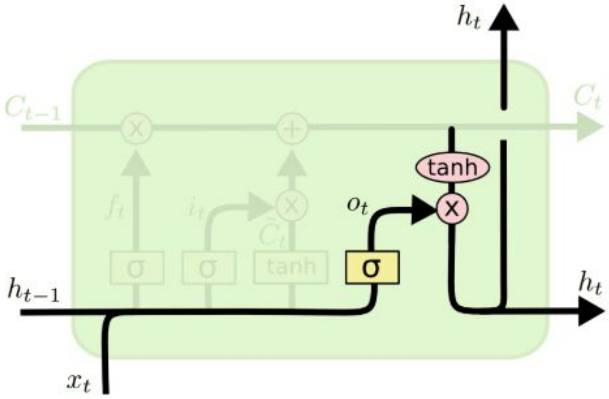
i_t → Input gate

f_t → Forget gate

$\tilde{C}_t$ → New vector values

Image Source

III. Output at each timestep



Working of Output Gate

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

x_t , h_t → Input, Output

h_{t-1} → Output from previous timestep

W_o, b_o → Weights and Biases

o_t → Output gate

σ , tanh → Sigmoid, Tanh Activation

C_t → Cell state

Image Source

FPGA for ML Applications

- Adaptable architecture
- Low power consumption
- Flexible
- Preferred for the task of inference due to low latency

FPGA for ML Applications

- Adaptable architecture
- Low power consumption
- Flexible
- Preferred for the task of inference due to low latency
- Need to code in HDL

High-Level Synthesis

- Automated design process that converts an algorithm in high-level language to low-level language.

High-Level Synthesis

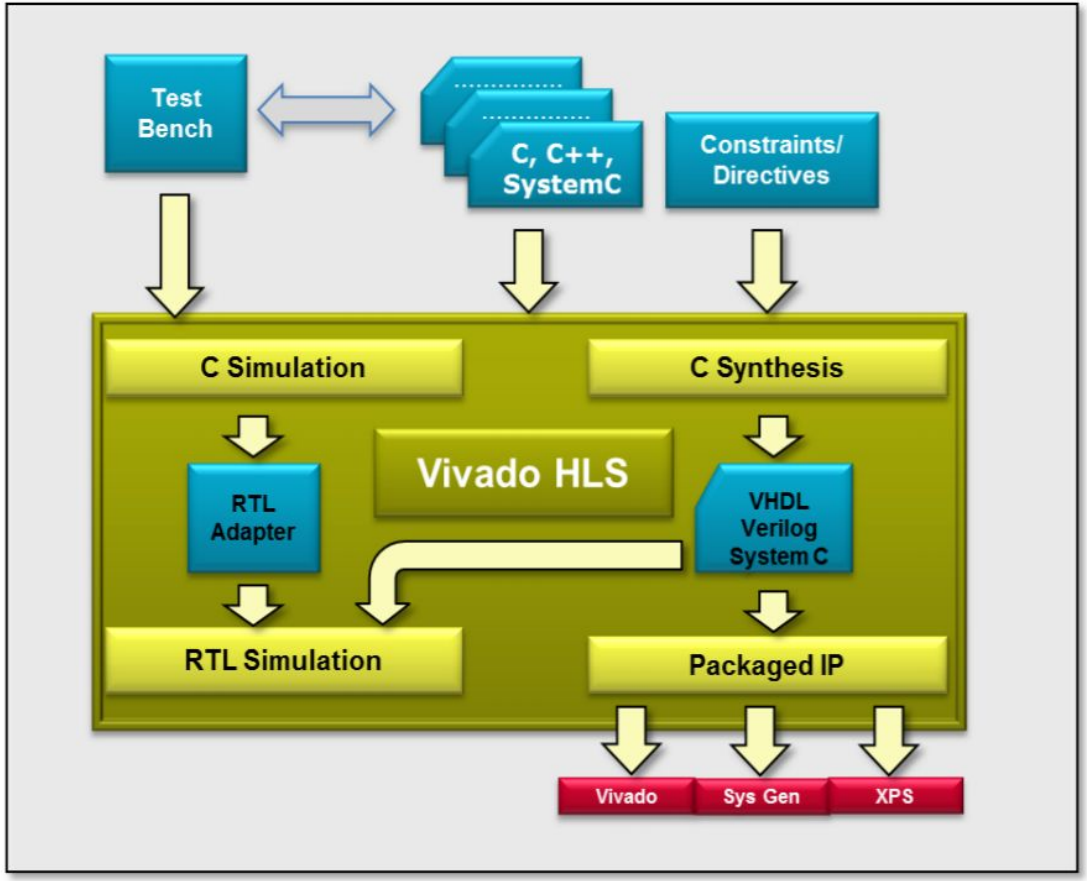


Image Source

HLS4ML Framework

- High-Level Synthesis for Machine Learning
- Tunable parameters
  - Quantization
  - Parallelization

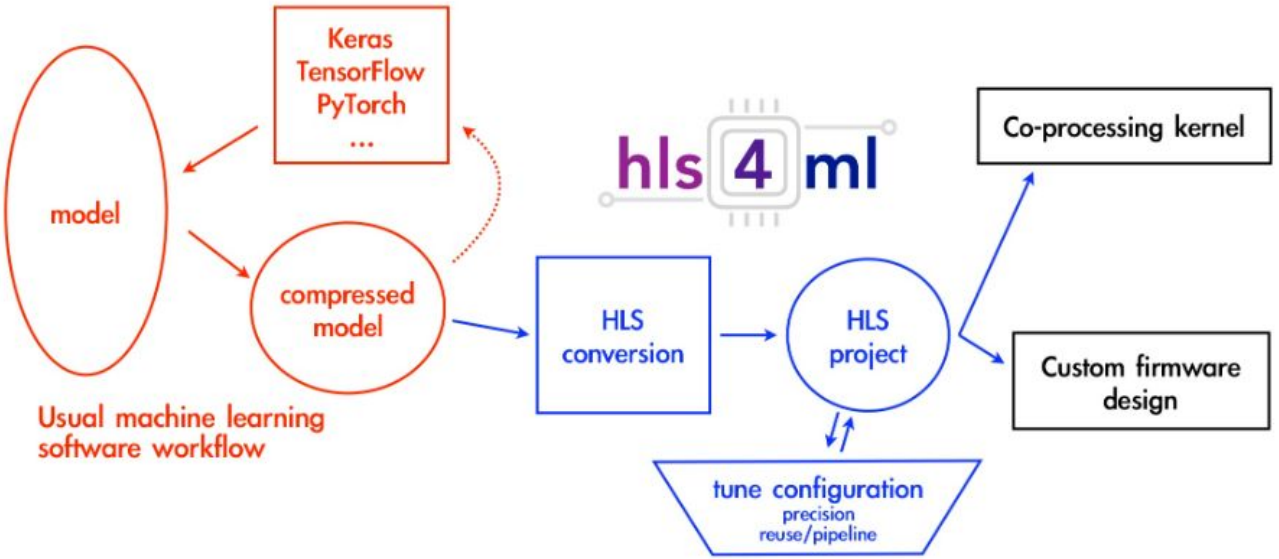


Image Source

HLS4ML Framework

- High-Level Synthesis for Machine Learning
- Tunable parameters
  - Quantization
  - Parallelization

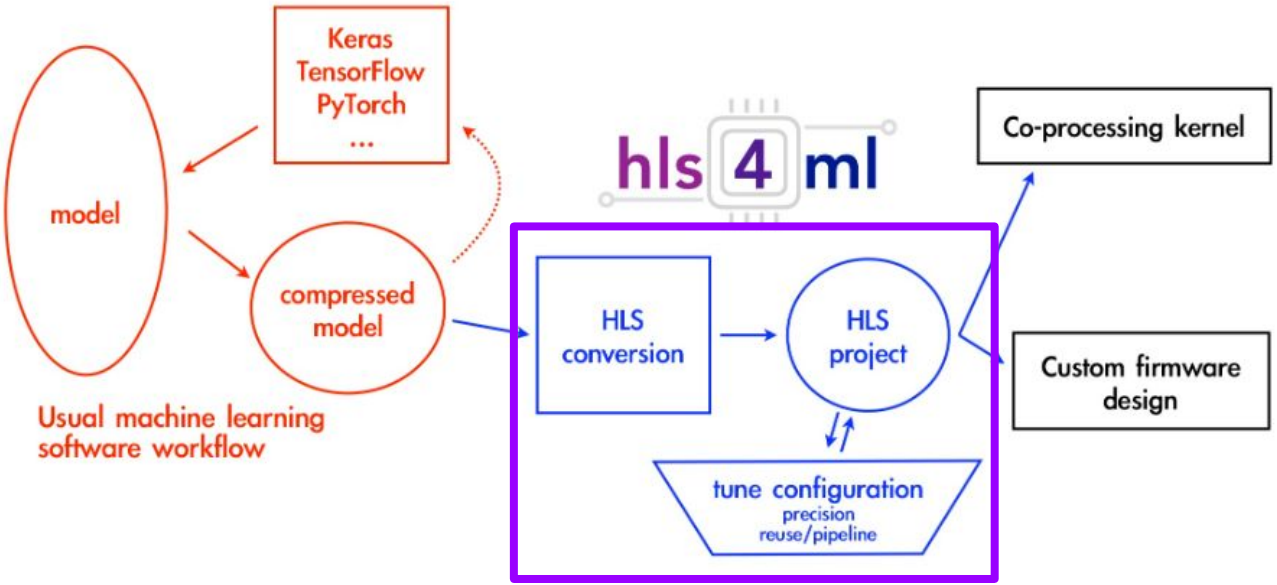


Image Source

Neural Networks and ML packages supported by HLS4ML

- ML Packages
  - Keras
  - Tensorflow
  - PyTorch
  - Onnx
- Neural networks supported
  - Fully connected NNs
  - Convolutional NNs
  - Boosted Decision Trees

Neural Networks and ML packages supported by HLS4ML

- ML Packages
  - Keras
  - Tensorflow
  - PyTorch
  - Onnx
- Neural networks supported
  - Fully connected NNs
  - Convolutional NNs
  - Boosted Decision Trees
  - **Long Short-Term Memory NNs**

Application of HLS4ML

- If the neural network model and ML package is supported
  - Any ML application

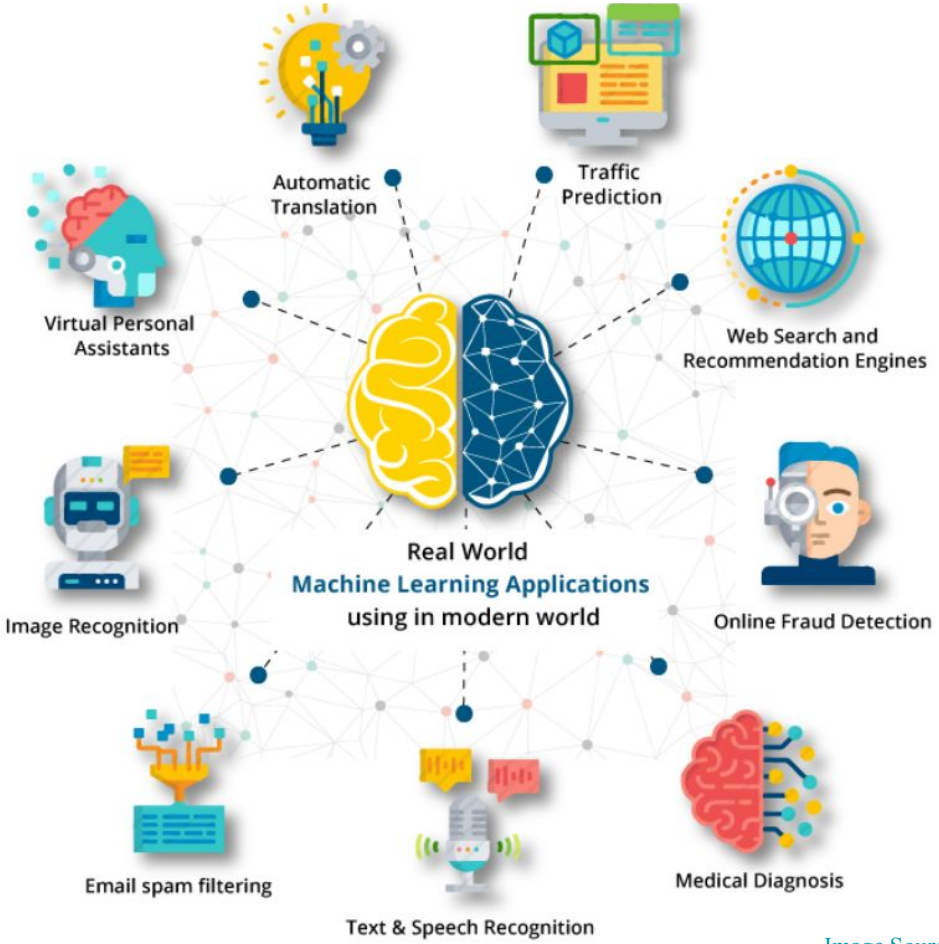


Image Source

Application of HLS4ML

- If the neural network model and ML package is supported
  - Any ML application

- Focus of HLS4ML group
  - **Application in Physics - Top Tagging**

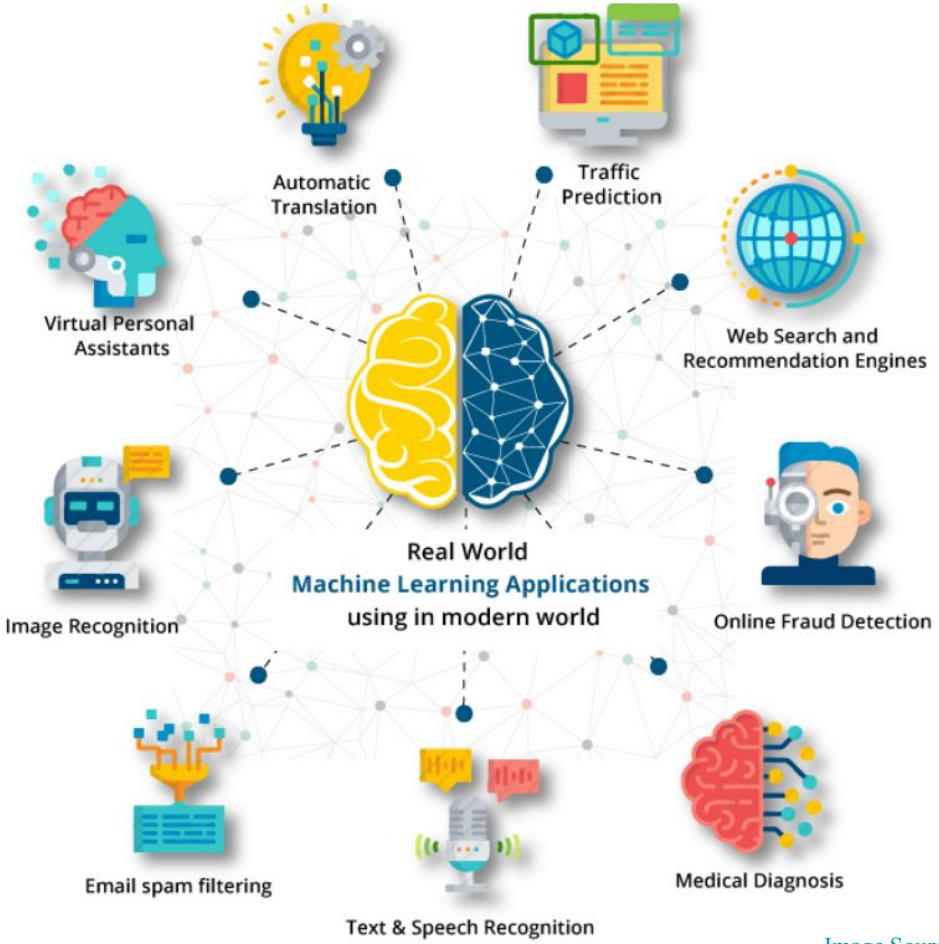


Image Source

A quick detour into the world of physics...

Top Tagging

- Types of jets
  - light quark (q)
  - gluon (g)
  - W boson (W)
  - Z boson (Z)
  - top quark (t)

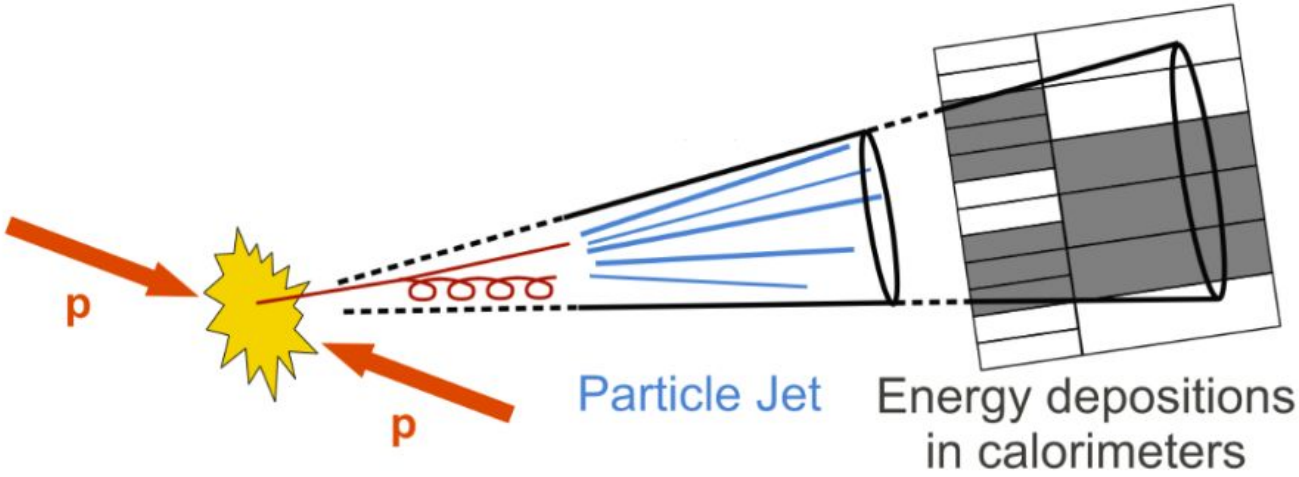
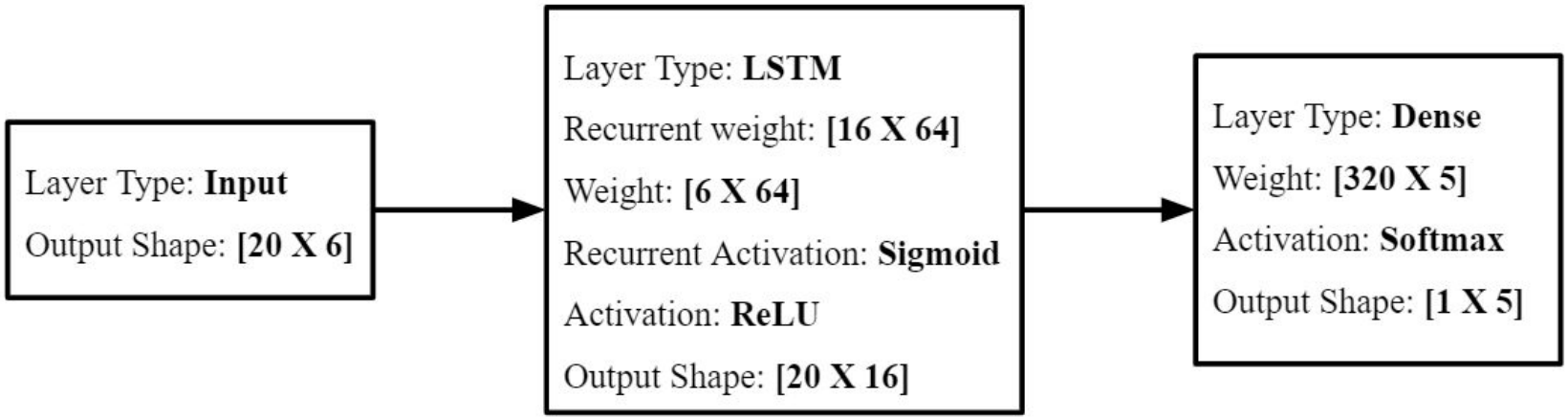


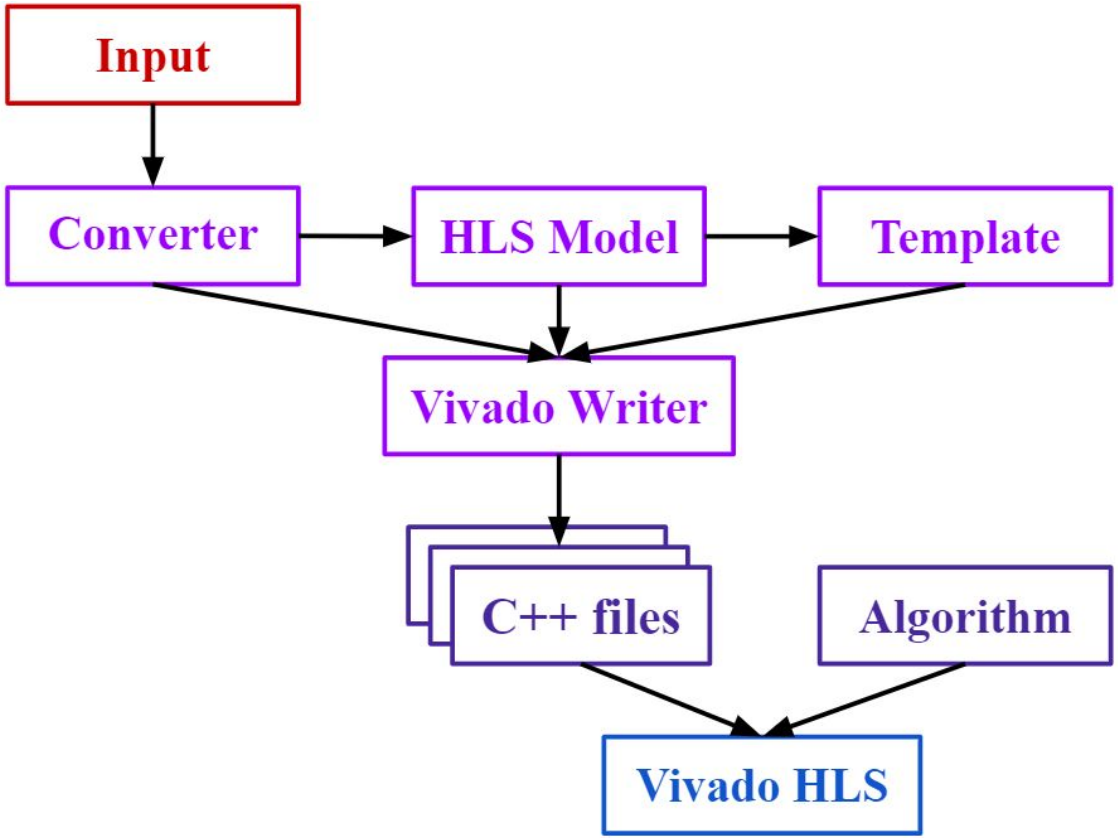
Image Source

KERAS LSTM top tagging model

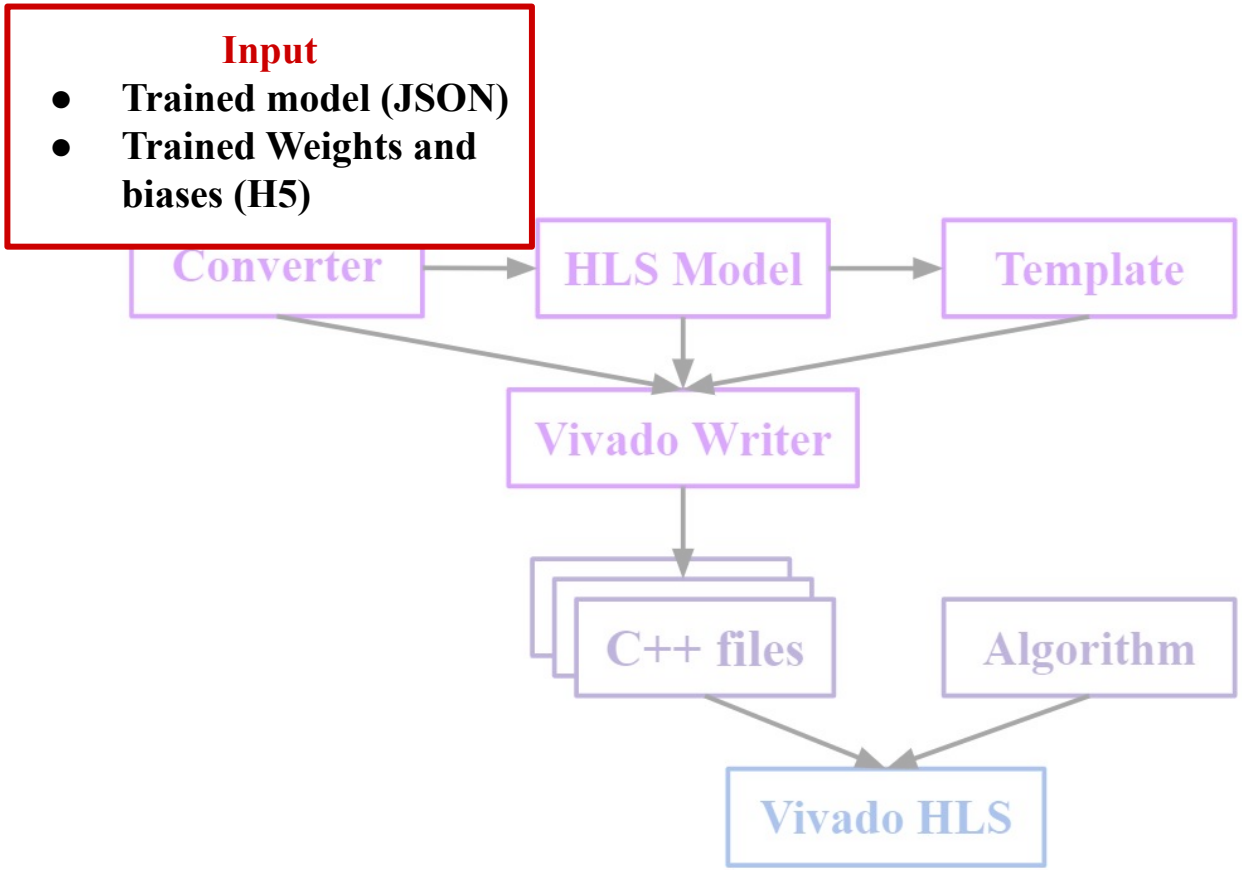
- Input:
  - Sequence of 20 particles with 6 features each
- Output:
  - Probability of 5 jet classes (q,g,W,Z,t)



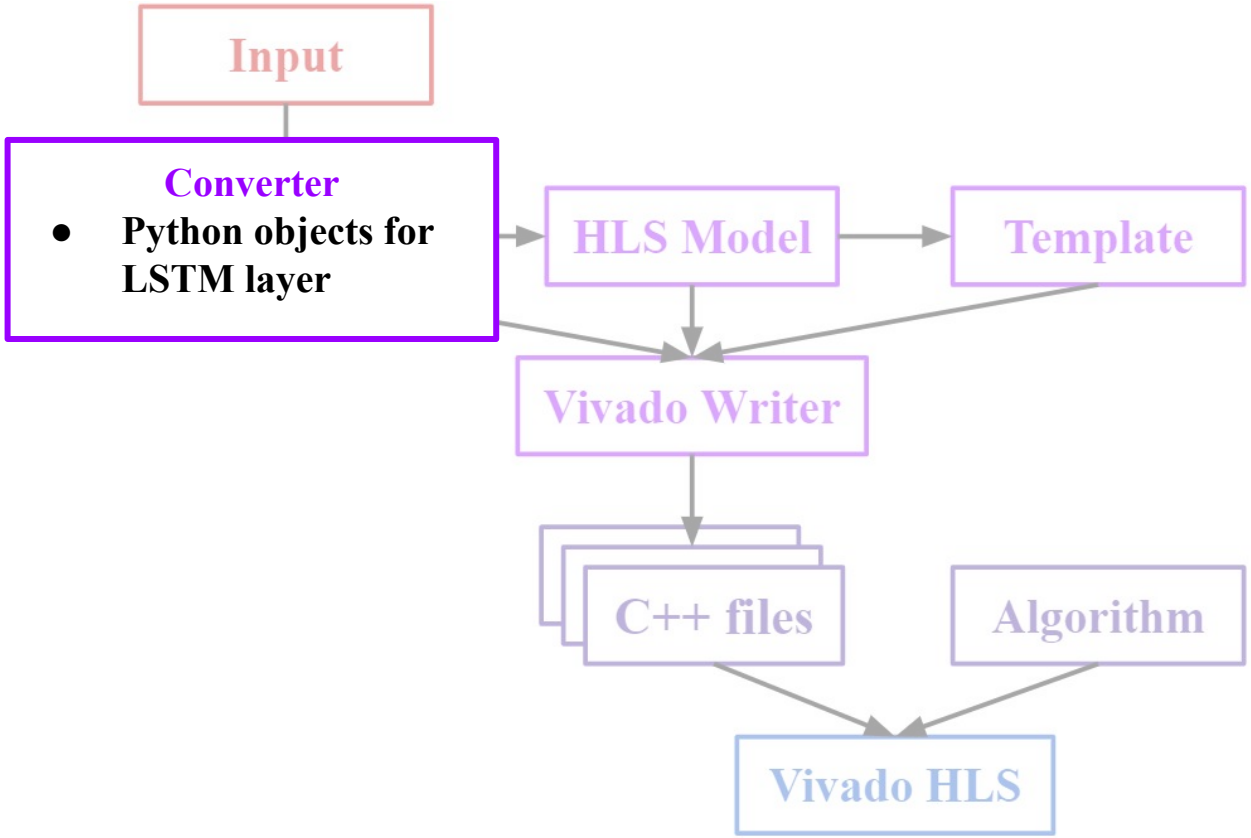
Block diagram to add LSTM into HLS4ML



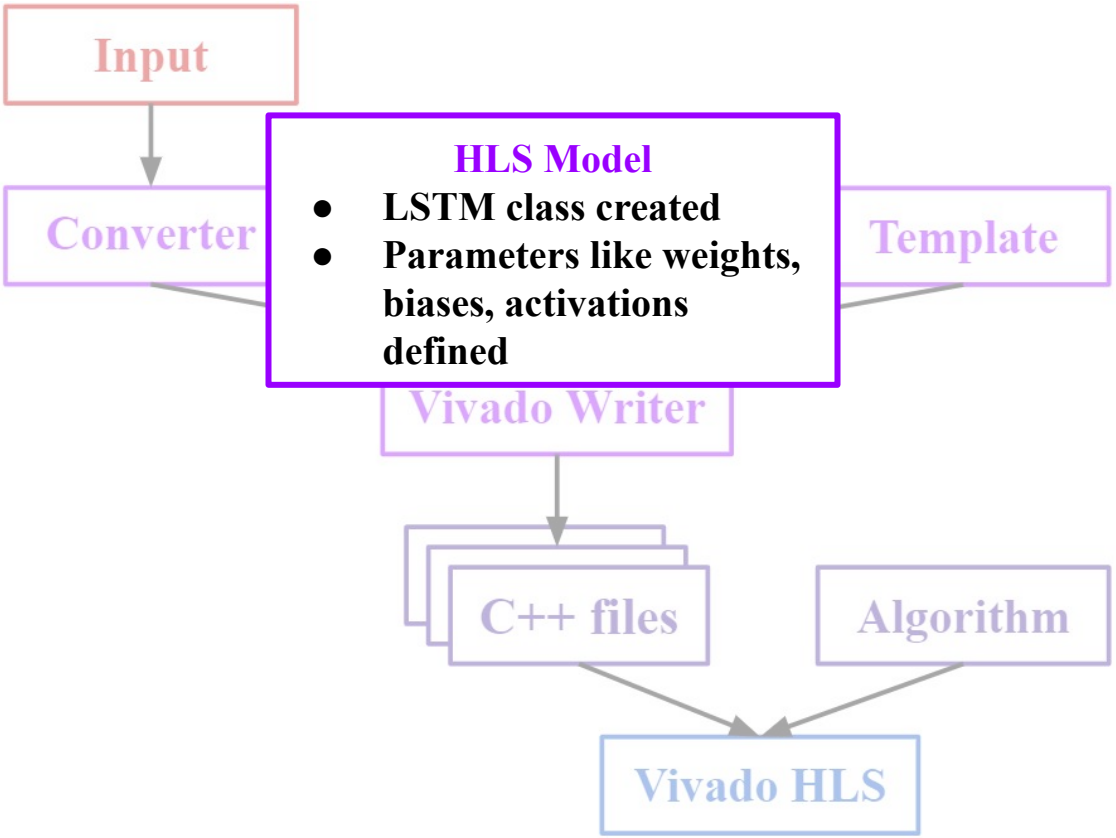
Block diagram to add LSTM into HLS4ML



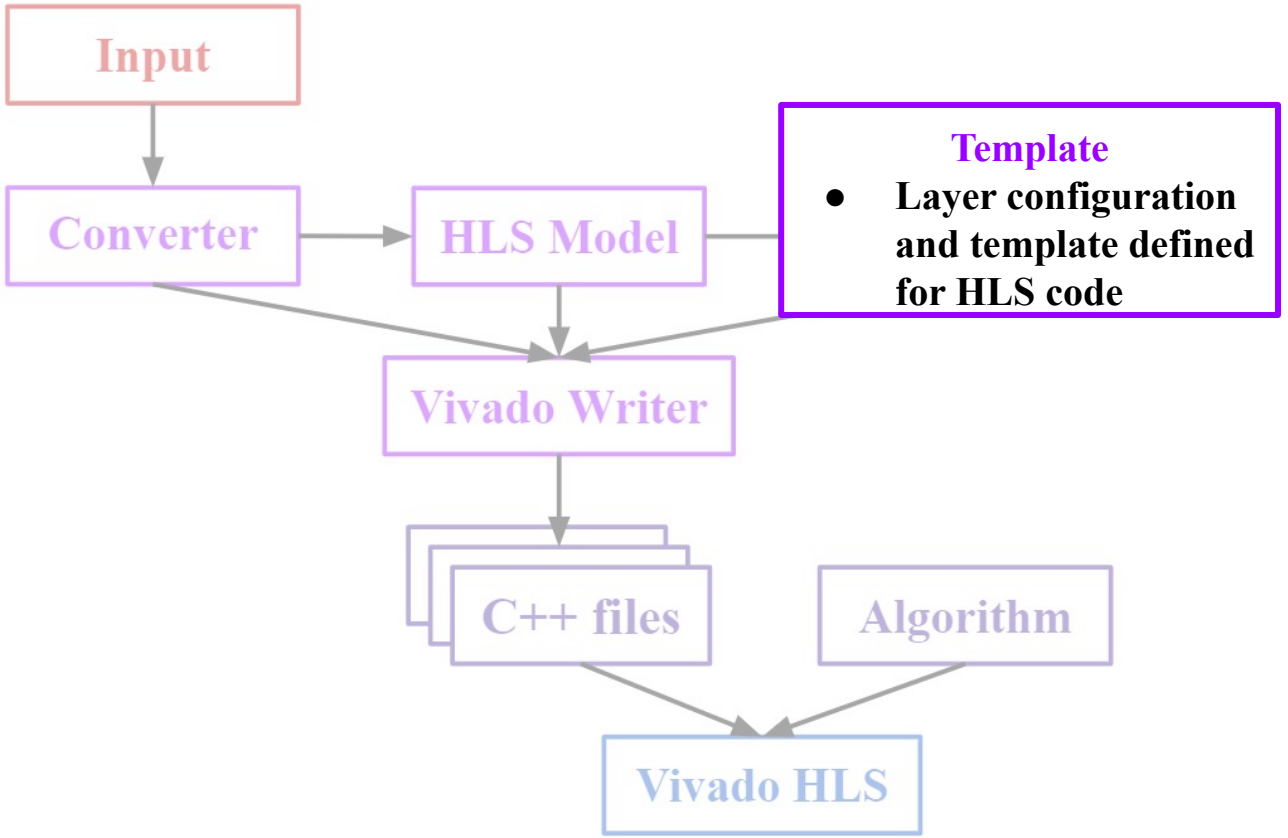
Block diagram to add LSTM into HLS4ML



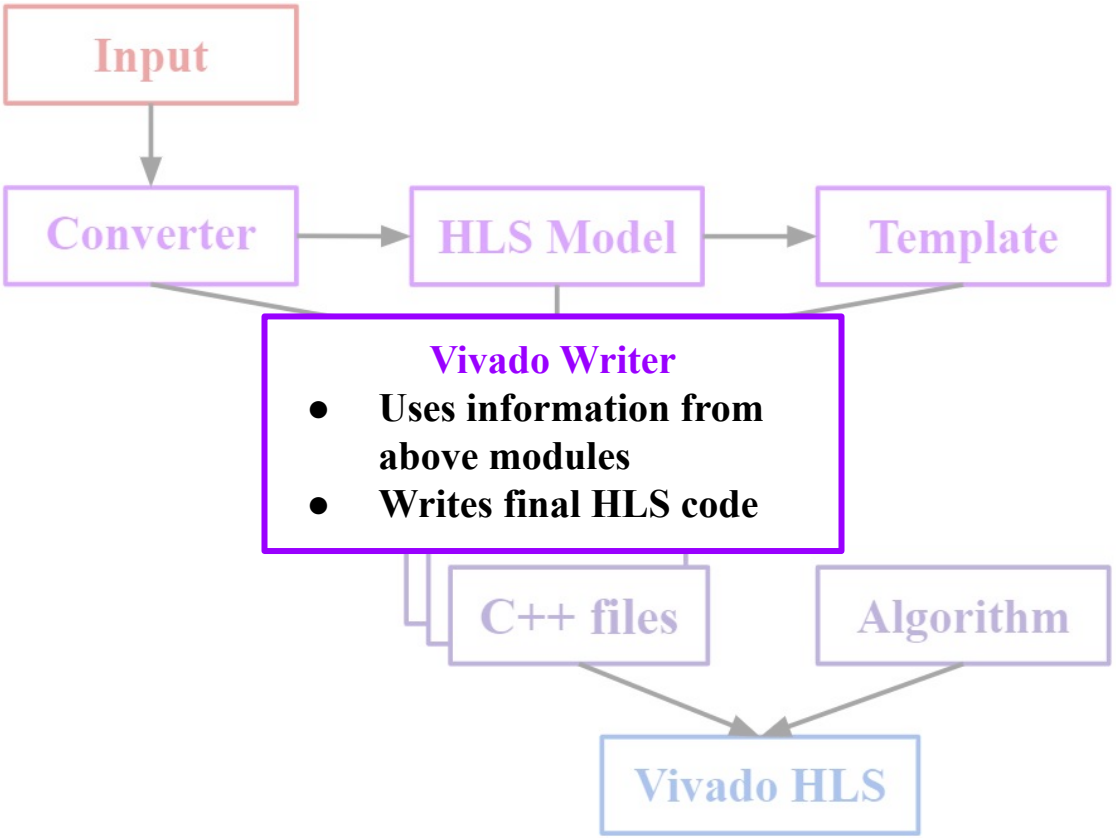
Block diagram to add LSTM into HLS4ML



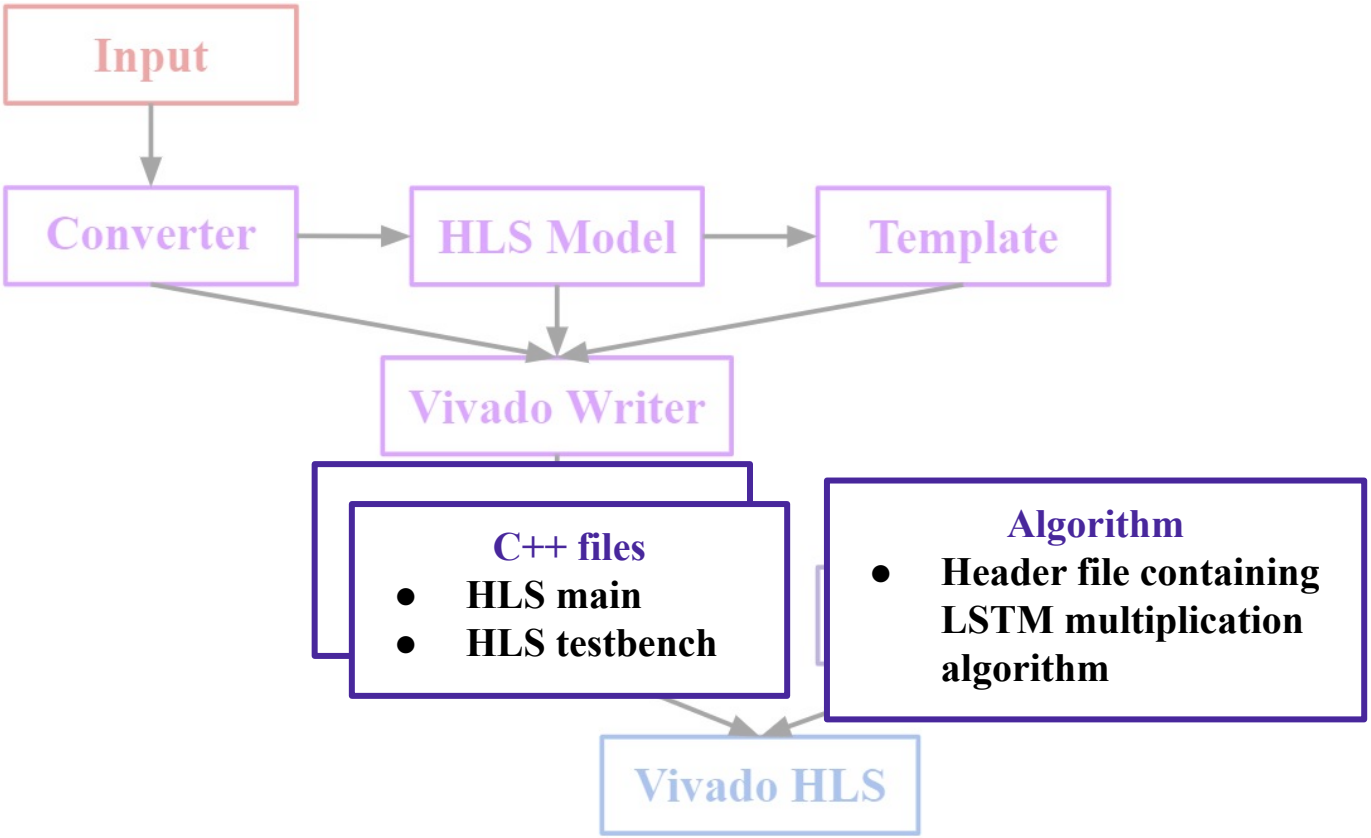
Block diagram to add LSTM into HLS4ML



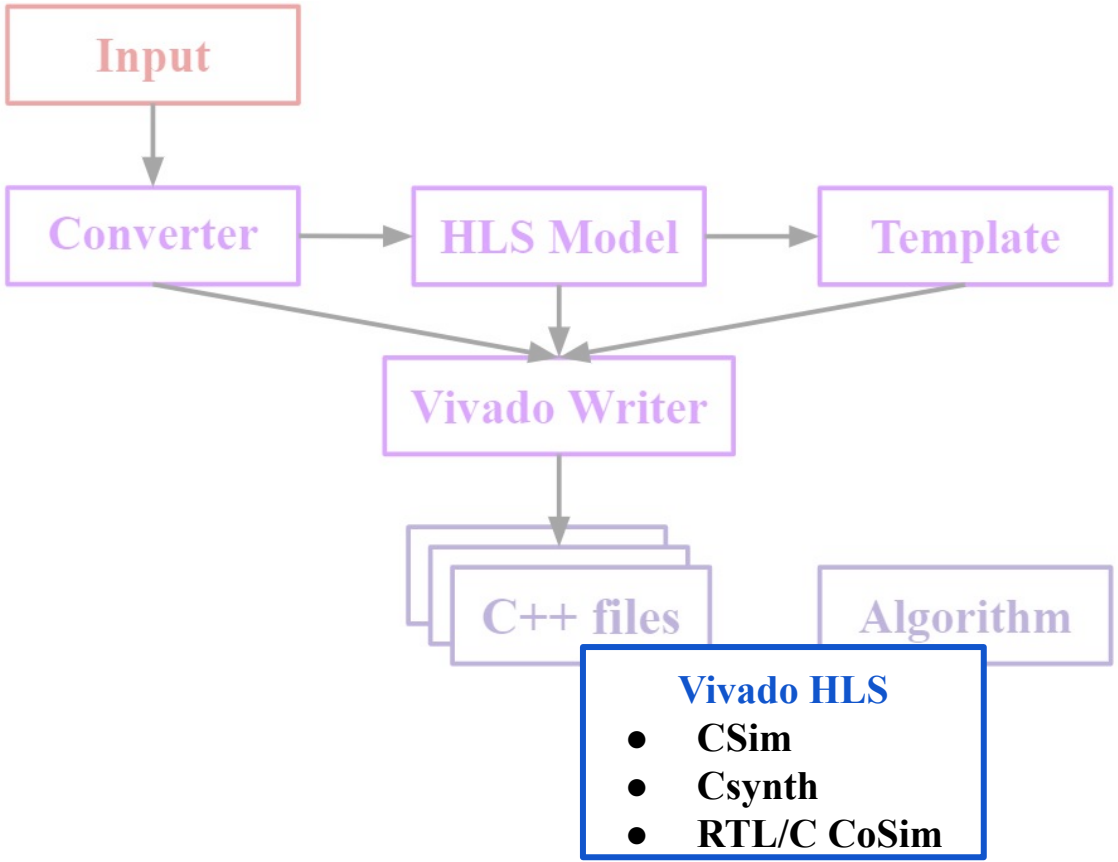
Block diagram to add LSTM into HLS4ML



Block diagram to add LSTM into HLS4ML

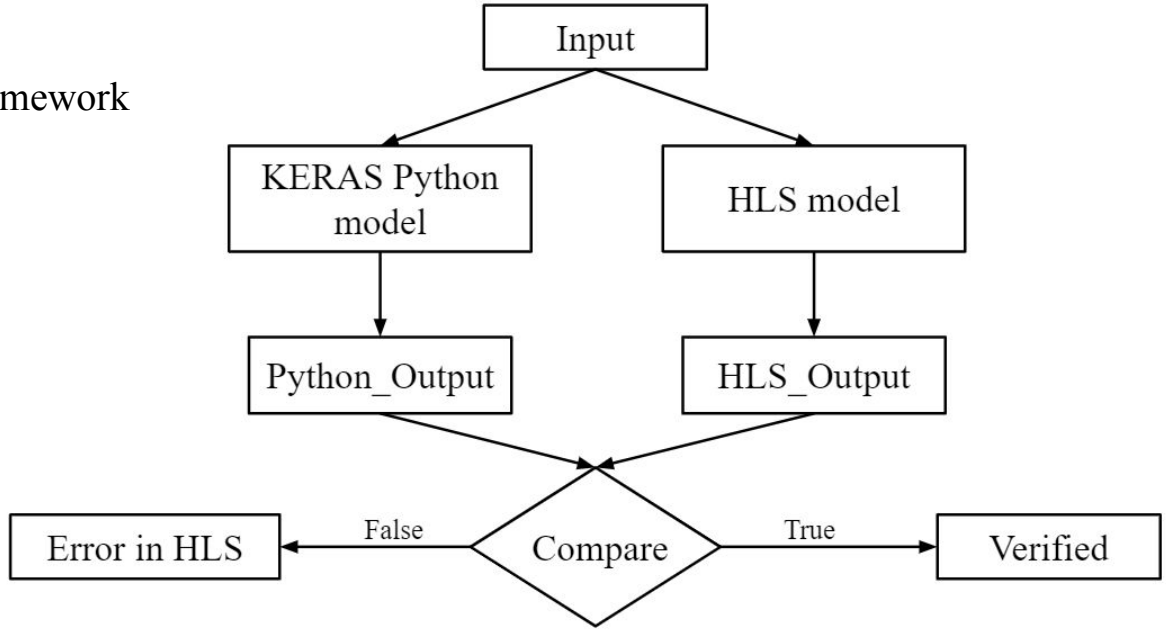


Block diagram to add LSTM into HLS4ML



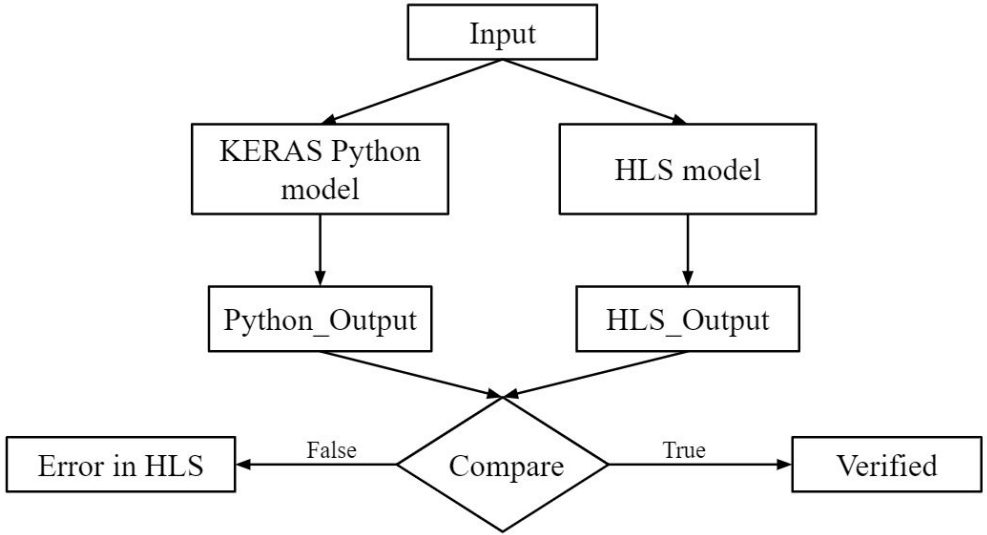
Two step verification

- **Step 1:** KERAS vs HLS
  - Not a part of HLS4ML framework



Two step verification

- **Step 1:** KERAS vs HLS
  - Not a part of HLS4ML framework
- **Step 2:** HLS vs RTL
  - Verified by Vivado HLS



Resource Utilization and Latency as per HLS compiler

- FPGA Targeted: Xilinx Kintex Ultrascale FPGA (xcku115-flvb2104-2-i)
- Reuse factor: 1
- Precision: <16,6> (6 integer bits, 10 fractional bit)

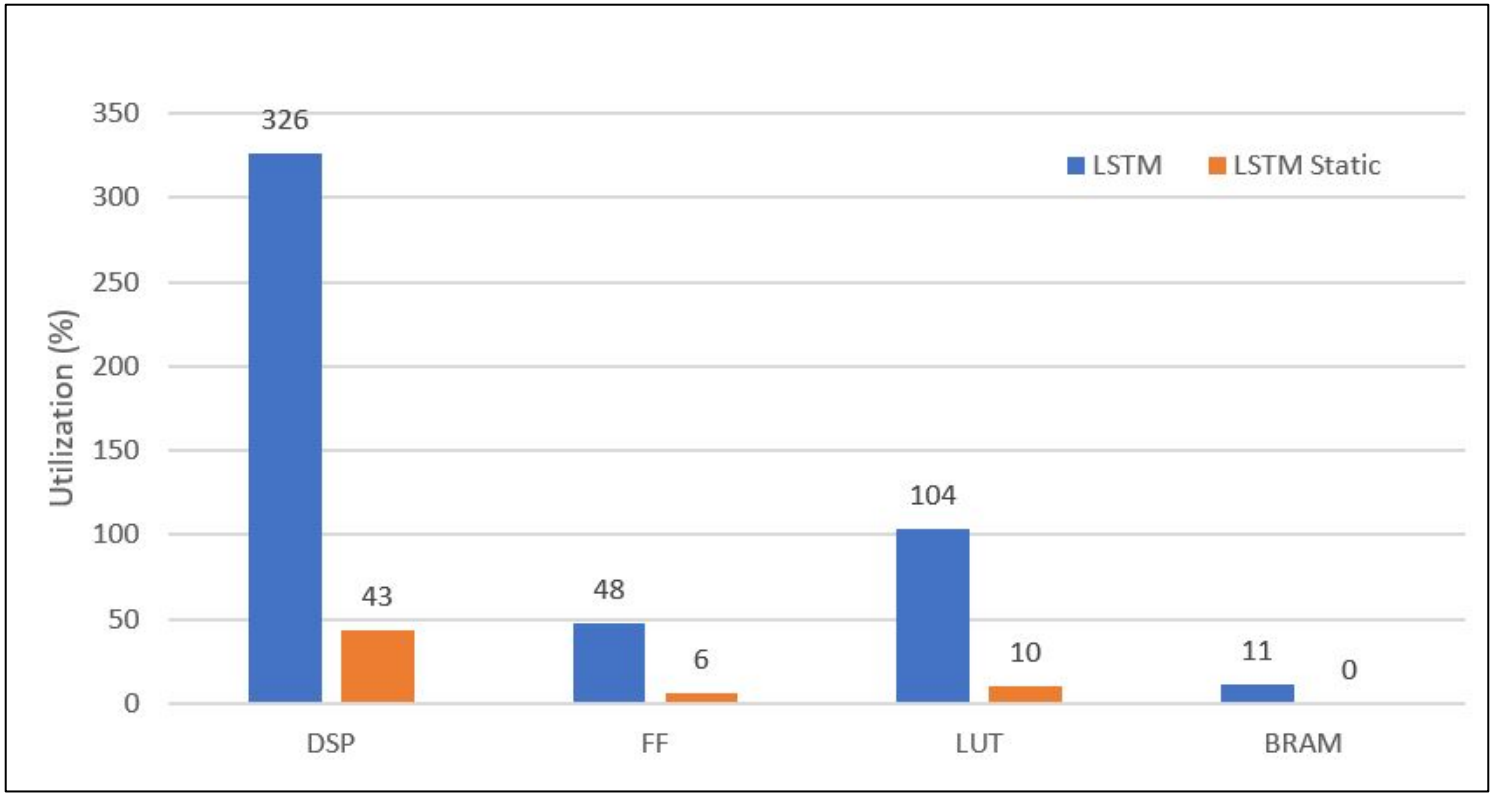
Model	Latency(μs)	Iteration Interval(ns)	DSP(%)	FF(%)	LUT(%)	BRAM(%)
LSTM	1.35	5	326	48	104	11

Resource Utilization and Latency as per HLS compiler

- FPGA Targeted: Xilinx Kintex Ultrascale FPGA (xcku115-flvb2104-2-i)
- Reuse factor: 1
- Precision: <16,6> (6 integer bits, 10 fractional bit)

Model	Latency(μs)	Iteration Interval(ns)	DSP(%)	FF(%)	LUT(%)	BRAM(%)
LSTM	1.35	5	326	48	104	11
LSTM Static	1.35	1350	43	6	10	~0

Resource Utilization as per HLS compiler



Conclusion

- LSTM was implemented in the HLS4ML framework
- LSTM implementation was optimized to reduce resource utilization
- Github link: https://github.com/richarao/hls4ml/tree/keras-lstm

References

[1] Duarte, J., Han, S., Harris, P., Jindariani, S., Kreinar, E., Kreis, B., Ngadiuba, J., Pierini, M., Rivera, R., Tran, N. and Wu, Z., 2018. Fast inference of deep neural networks in FPGAs for particle physics. *Journal of Instrumentation*, 13(07), pp.P07027-P07027

[2] Fastmachinelearning.org. 2020. *HLS4ML · Gitbook*. [online] Available at: <https://fastmachinelearning.org/hls4ml/>.

Thank You Scott, Shih-Chieh

Thank You HLS4ML team!