25TH International Symposium on
VLSI Design and Test (VDAT 2021)

LightFPGA: Scalable and Automated FPGA Acceleration of LightGBM for Machine Learning Applications

Alish Kanani*
kanani.1@iitj.ac.in

Swar Vaidya*
bhavarth.1@iitj.ac.in

Harshit Agarwal
agarwalh@iitj.ac.in

Department of Electrical Engineering
Indian Institute of Technology Jodhpur

PAPER ID: 113

* Both authors contributed equally to this research.

Motivation

- Machine learning is progressing at a rate that is outpacing Moore's Law [1] and now are evolving faster than silicon can be designed.

- With the growth in usage of Machine Learning in real time applications, the need to accelerate inference of algorithms is emergent.

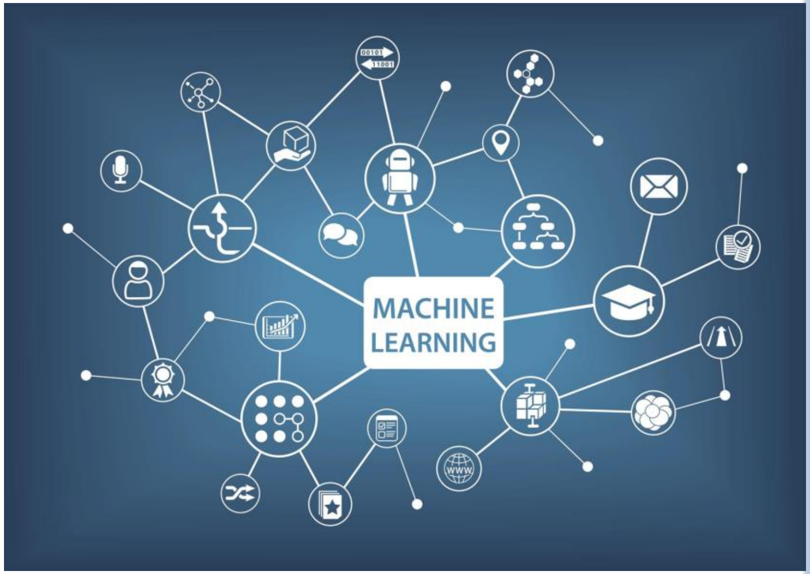
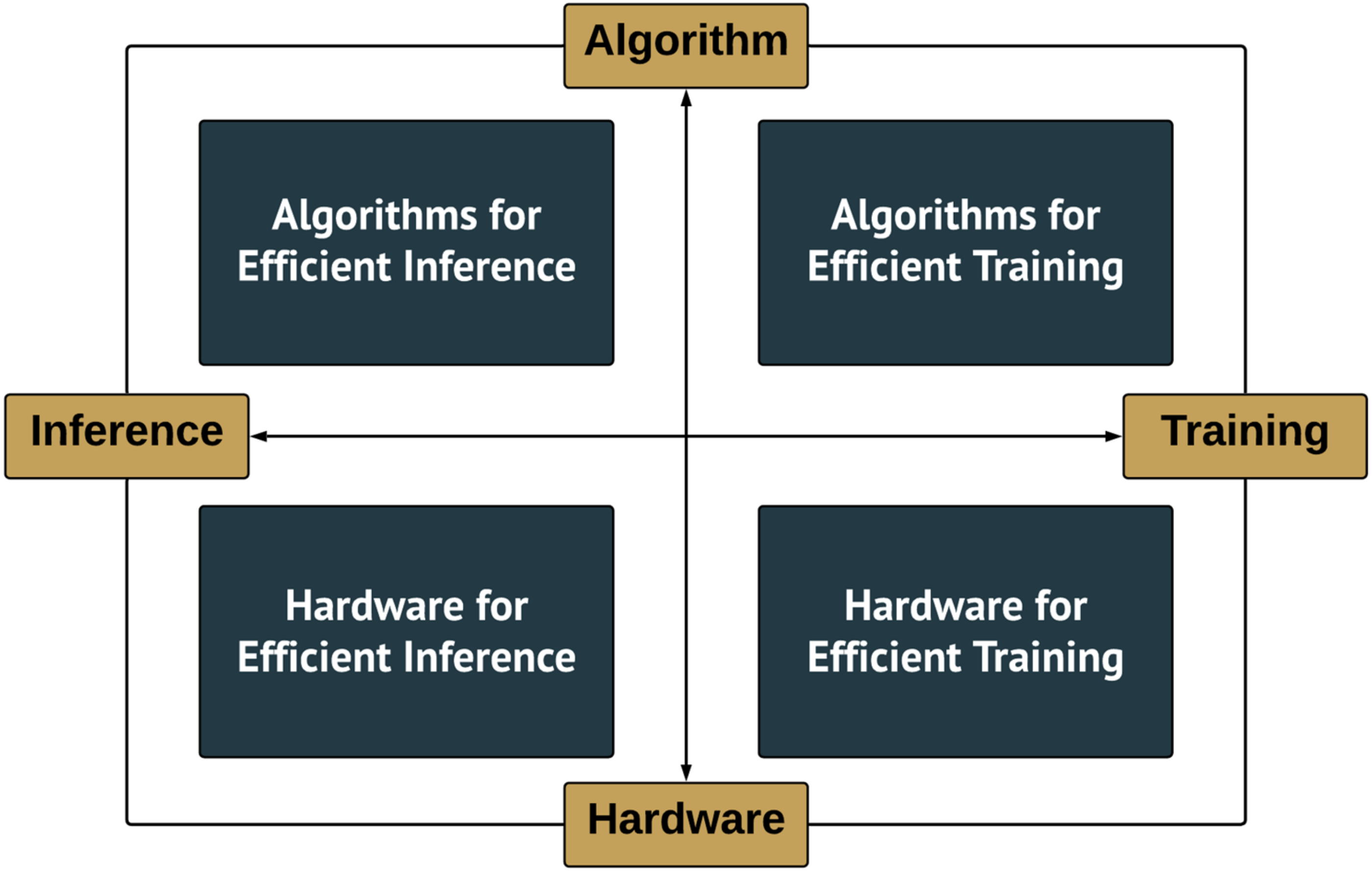


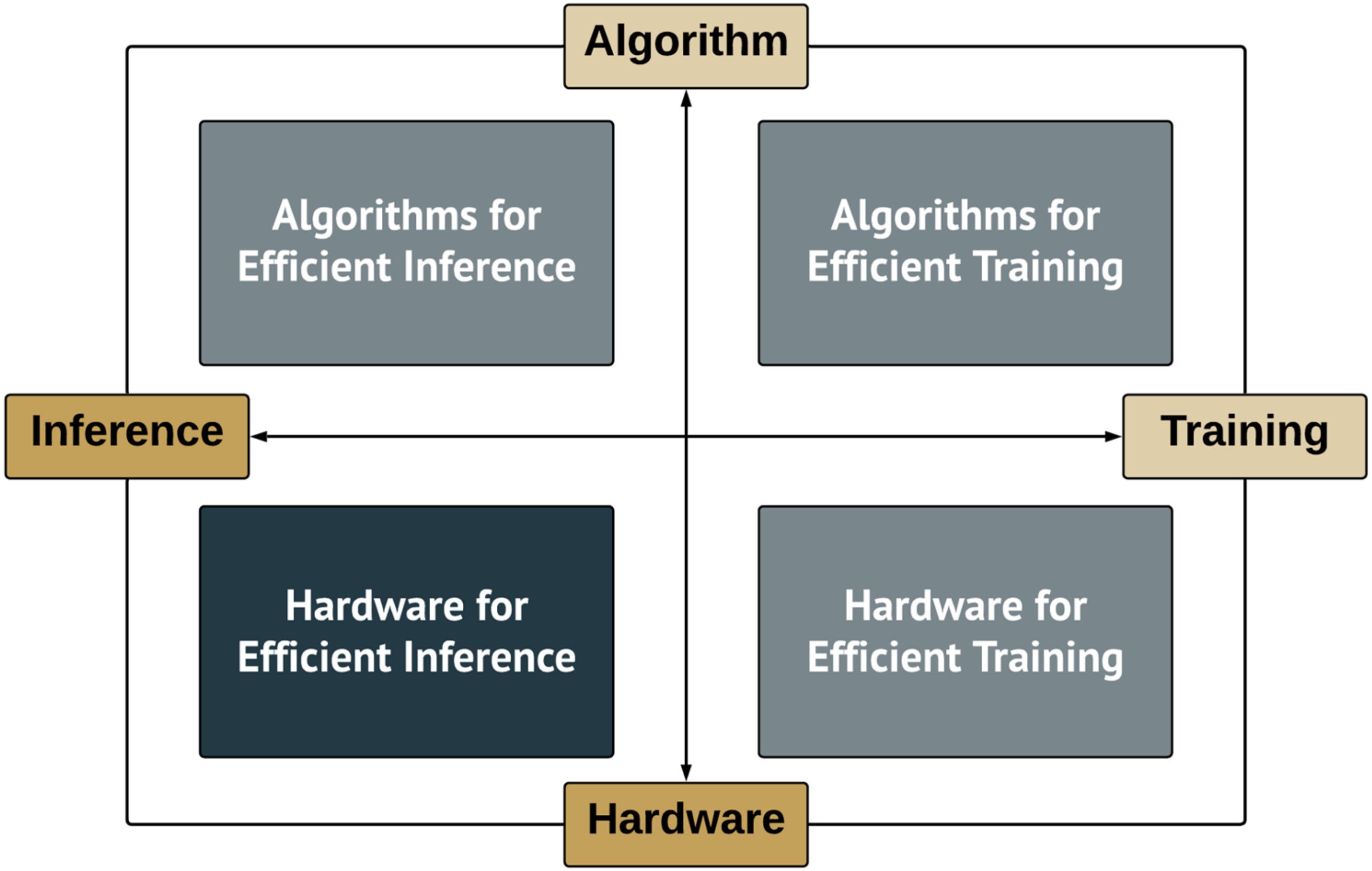
Image Source: Internet

[1] Shalf, John. (2020). "The future of computing beyond Moore's Law. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences." 378.

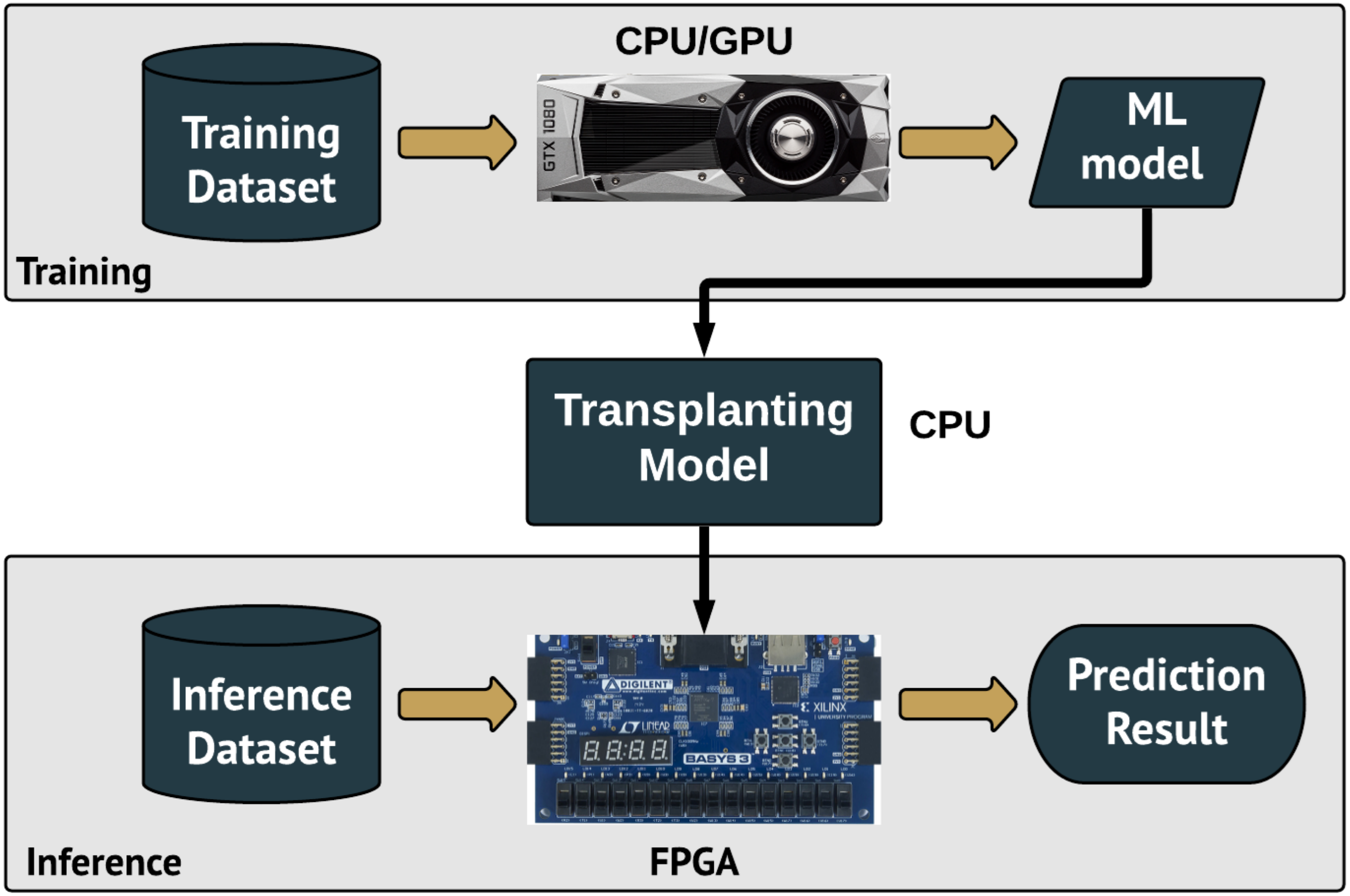
Methods to accelerate Machine Learning


Algorithm
Algorithms for Efficient Inference
Algorithms for Efficient Training
Inference
Training
Hardware for Efficient Inference
Hardware for Efficient Training
Hardware

Methods to accelerate Machine Learning


Algorithm
Algorithms for Efficient Inference
Algorithms for Efficient Training
Inference
Training
Hardware for Efficient Inference
Hardware for Efficient Training
Hardware

Flow for FPGA Inference



But why LightGBM?

- Gradient Boosting and Decision tree based ensemble learning algorithm.
- Popular and relatively more accurate algorithm compared to similar state of the art algorithms.
- Classification and regression for small scale applications.
- Good accuracy even with small dataset.
- Well developed & open-source library by Microsoft.

Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. "Lightgbm: A highly efficient gradient boosting decision tree." In Advances in neural information processing systems, pp. 3146-3154. 2017.

LightGBM vs State of the art Algorithms

Accuracy Score		
	MNIST digits	Iris
CatBoost [2]	97.35%	93.33%
LightGBM	97.59%	95.55%
XGBoost [1]	96.48%	95.55%

Timing Results				
	MNIST Digits		Iris	
	Train (s)	Test (ms)	Train (s)	Test (ms)
CatBoost	1.53	14.37	1.58	1.62
LightGBM	1.10	9.33	0.11	0.58
XGBoost	1.12	11.18	0.22	1.31

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 785–794. ACM, 2016

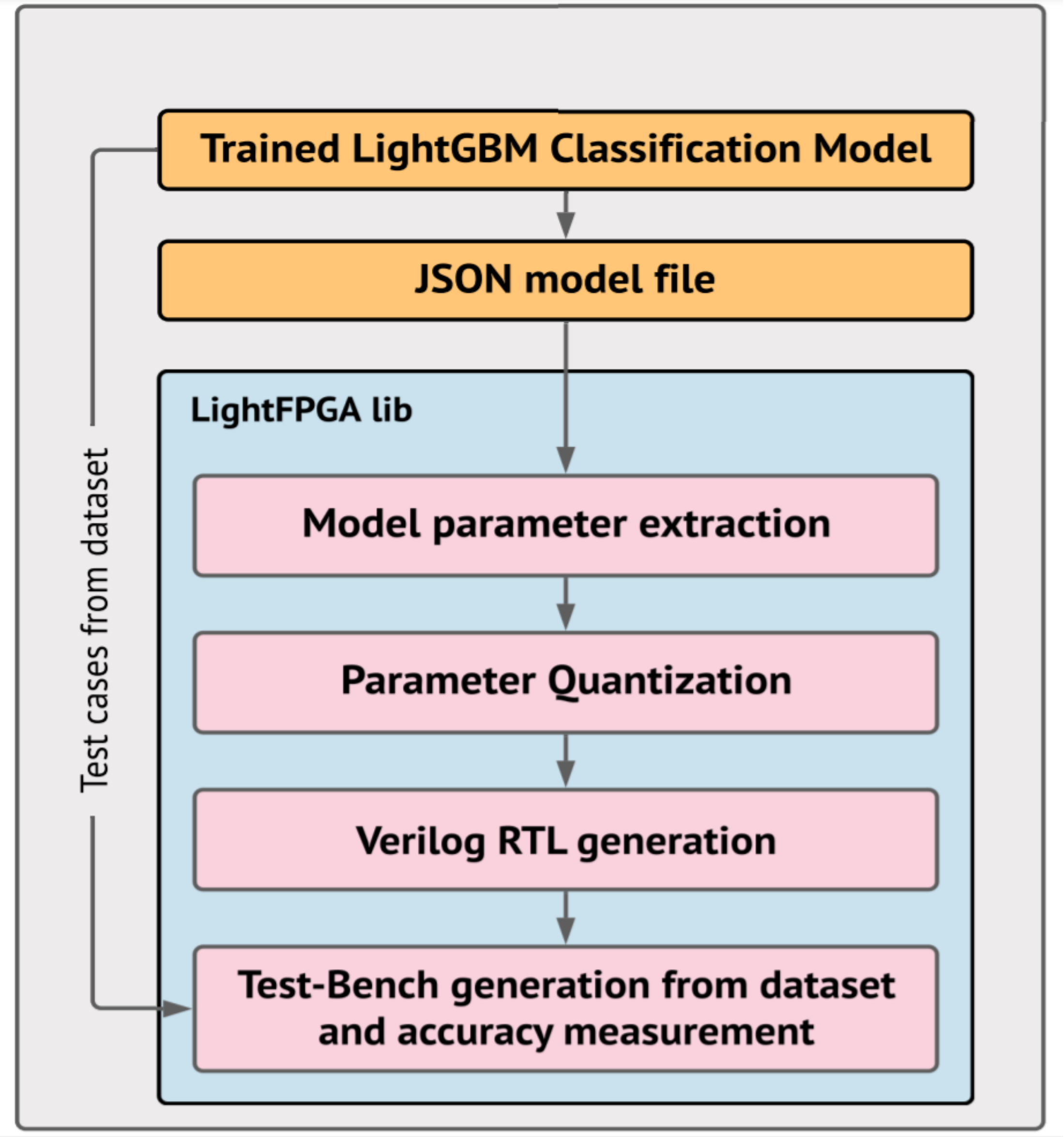
[2] Prokhorenkova, Liudmila, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. "CatBoost: unbiased boosting with categorical features." In Advances in neural information processing systems, pp. 6638-6648. 2018

Methodology

Need of Automation : LightFPGA

- Hundreds/Thousands of decision trees, each with hundreds/thousands of comparisons.
- Manually RTL implementation is a very overwhelming task.
- Needs to be re-implemented for different datasets.
- The Verilog code which is to be written, is fundamentally very repetitive.
- Change is only in some of the numerical values, main logic remains the same. Thus, it is very suitable for automation.

LightFPGA : Library Flow

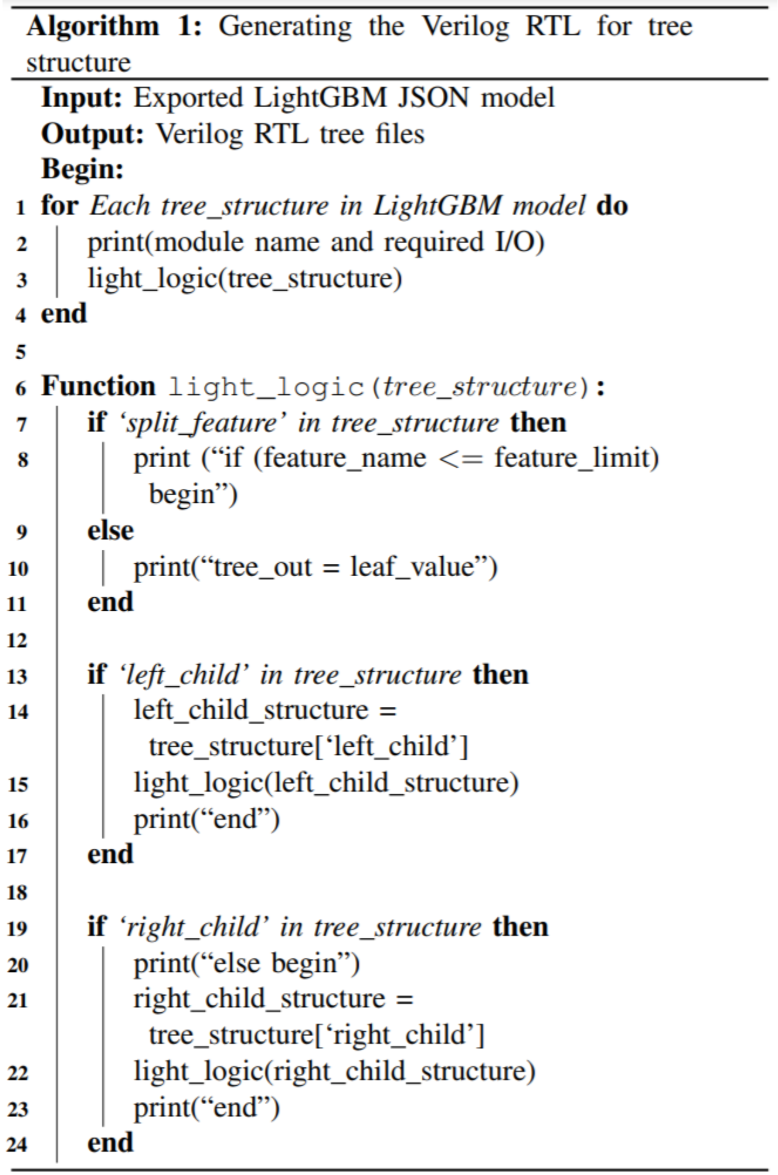

Trained LightGBM Classification Model
JSON model file
LightFPGA lib
Model parameter extraction
Parameter Quantization
Verilog RTL generation
Test-Bench generation from dataset and accuracy measurement
Test cases from dataset

VLSI Society of India

MeitY
Government of India

cadence

LightFPGA: RTL of trees



Algorithm 1: Generating the Verilog RTL for tree structure

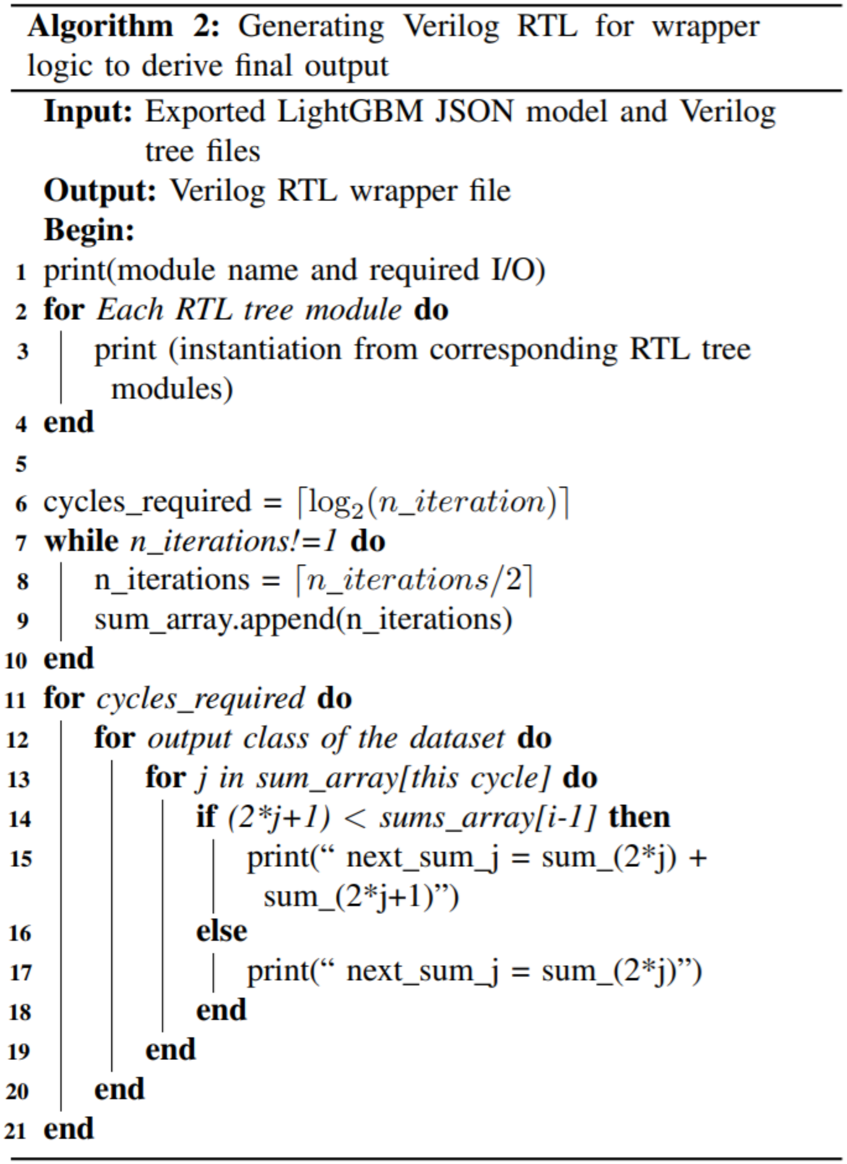
```
Input: Exported LightGBM JSON model
Output: Verilog RTL tree files
Begin:
for Each tree_structure in LightGBM model do
    print(module name and required I/O)
    light_logic(tree_structure)
end

Function light_logic(tree_structure):
    if 'split_feature' in tree_structure then
        print ("if (feature_name <= feature_limit)
          begin")
    else
        print("tree_out = leaf_value")
    end

    if 'left_child' in tree_structure then
        left_child_structure =
          tree_structure['left_child']
        light_logic(left_child_structure)
        print("end")
    end

    if 'right_child' in tree_structure then
        print("else begin")
        right_child_structure =
          tree_structure['right_child']
        light_logic(right_child_structure)
        print("end")
    end
```

LightFPGA: RTL of Wrapper over Trees



Algorithm 2: Generating Verilog RTL for wrapper logic to derive final output

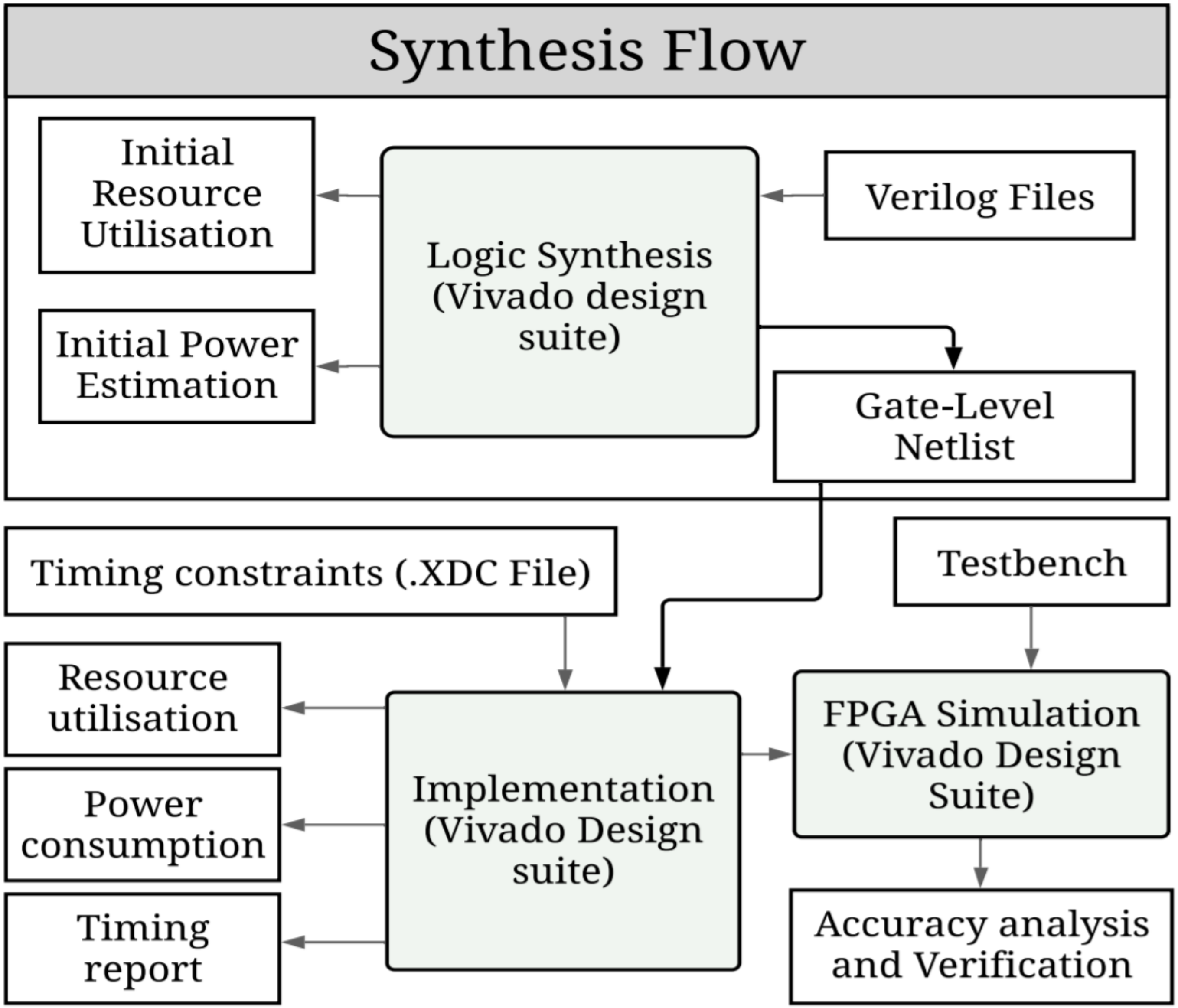
```
Input: Exported LightGBM JSON model and Verilog
       tree files
Output: Verilog RTL wrapper file
Begin:
print(module name and required I/O)
for Each RTL tree module do
    print (instantiation from corresponding RTL tree
      modules)
end

cycles_required = $\lceil \log_2(n\_iteration) \rceil$
while n_iterations!=1 do
    n_iterations = $\lceil n\_iterations/2 \rceil$
    sum_array.append(n_iterations)
end
for cycles_required do
    for output class of the dataset do
        for j in sum_array[this cycle] do
            if (2*j+1) < sums_array[i-1] then
                print(" next_sum_j = sum_(2*j) +
                  sum_(2*j+1)")
            else
                print(" next_sum_j = sum_(2*j)")
            end
        end
    end
end
```

LightFPGA: Testing and Accuracy Verification

- Testbench is generated automatically from the dataset.
- LightFPGA is integrated with a Verilog simulation and synthesis tool called *iverilog*.
- Testing is performed in *iverilog* software automatically.
- The outputs from verilog are compared with the outputs of the CPU implementation.

Synthesis, Implementation and Simulation


Synthesis Flow
Initial Resource Utilisation
Initial Power Estimation
Logic Synthesis (Vivado design suite)
Verilog Files
Gate-Level Netlist
Timing constraints (.XDC File)
Testbench
Resource utilisation
Power consumption
Timing report
Implementation (Vivado Design suite)
FPGA Simulation (Vivado Design Suite)
Accuracy analysis and Verification

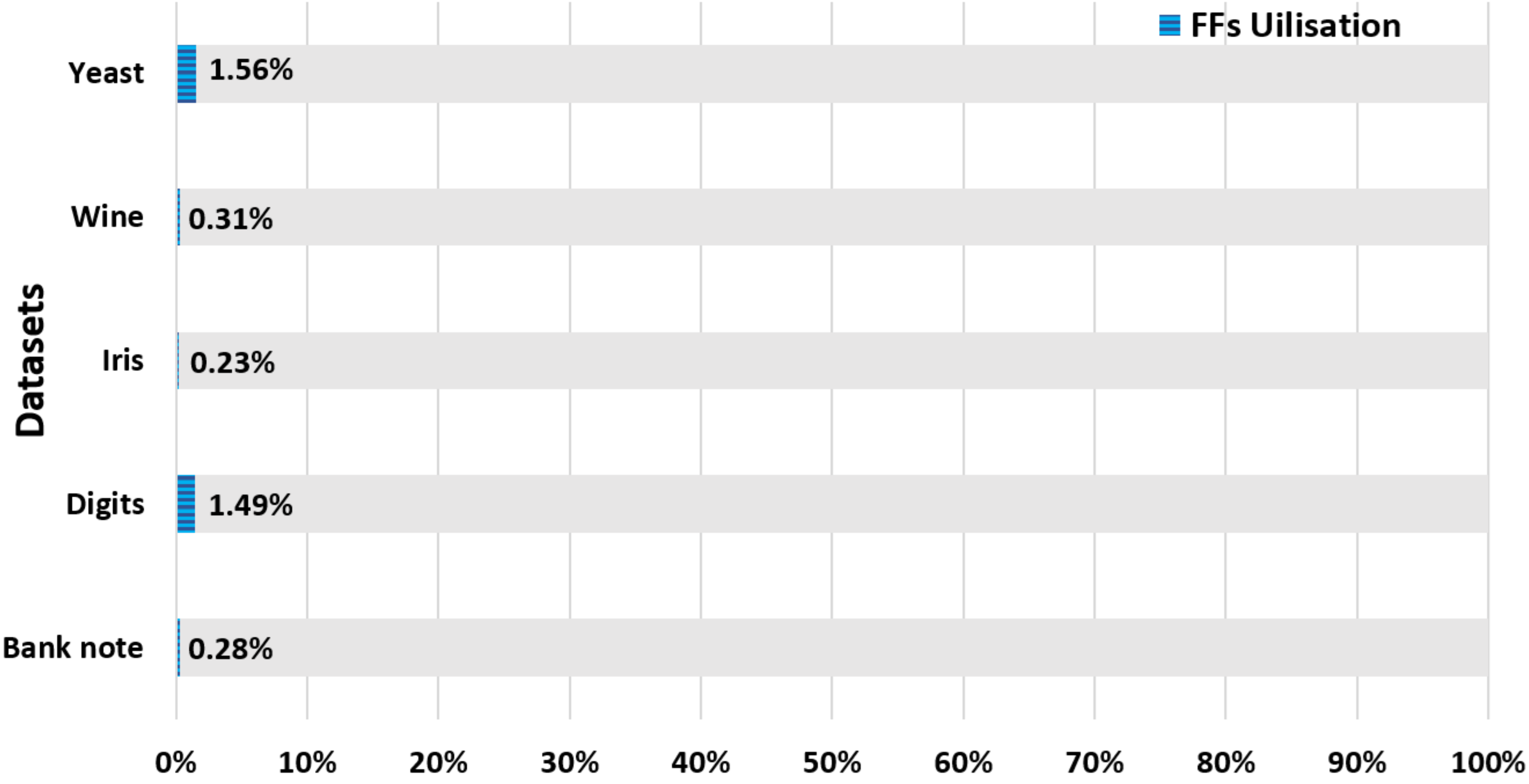
Results & Discussion

- The synthesis is performed in Xilinx Vivado for *Alveo U280* data center accelerator card FPGA.
- For comparison with CPU, Intel i5 processor with 2.40 GHz is used.
- Optimum clock frequency of FPGA is found to be 25MHz
- BRAM, URAM, and DSPs were not used in the implementation.

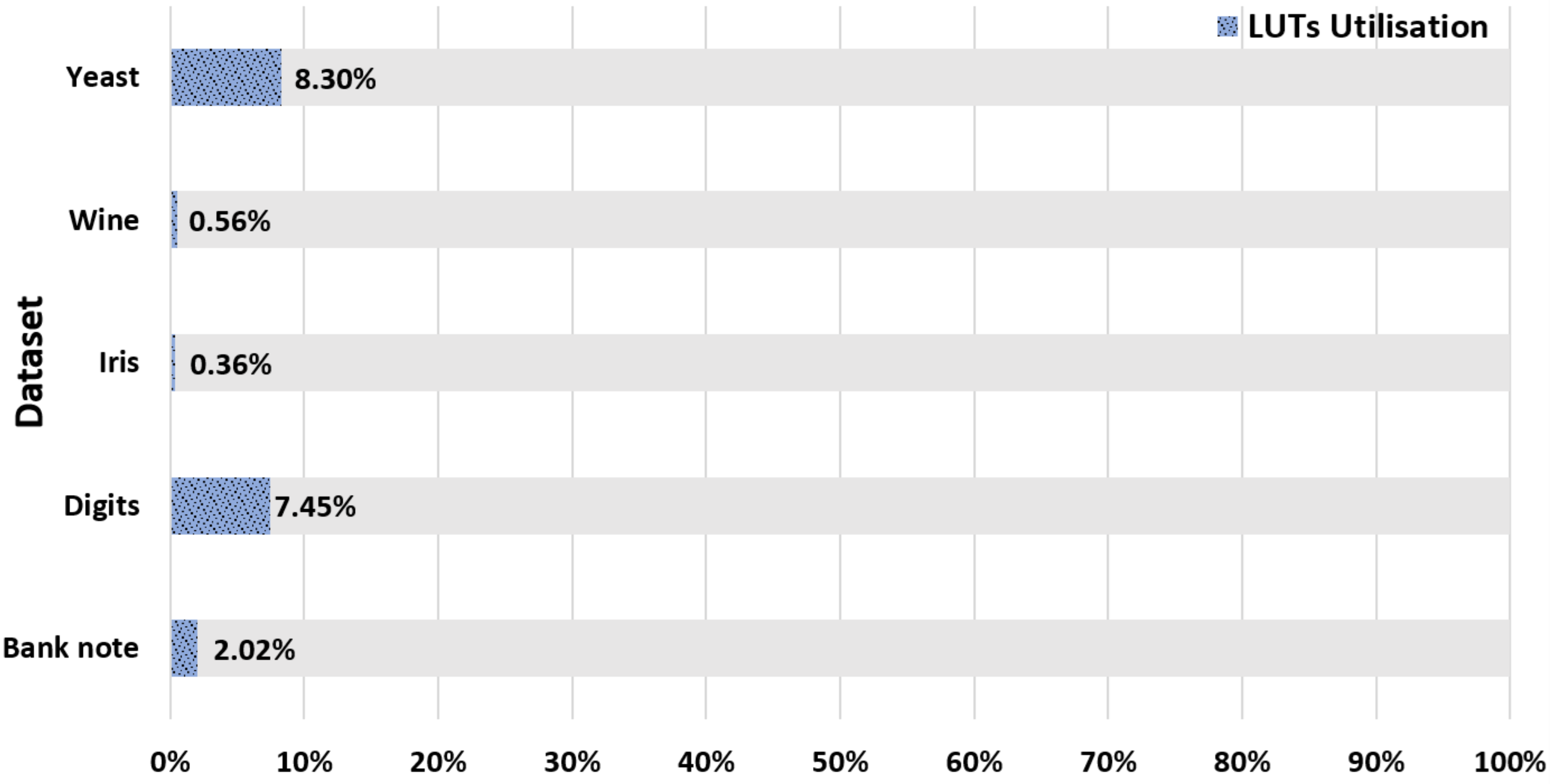
Pre-Training on CPU

- Models trained on 5 datasets:
- MNIST dataset : 64 x 1797 images of handwritten digits.
- Wine dataset : 13 x 178 inputs of wine constituents.
- Bank Note dataset : 5 x 1372 bank notes
- Iris dataset : 4 x 150 species of Iris
- Yeast dataset : 9 x 1484 cellular localization sites of proteins
- Train-Test split: 40%-60%

Resource Utilisation : FFs



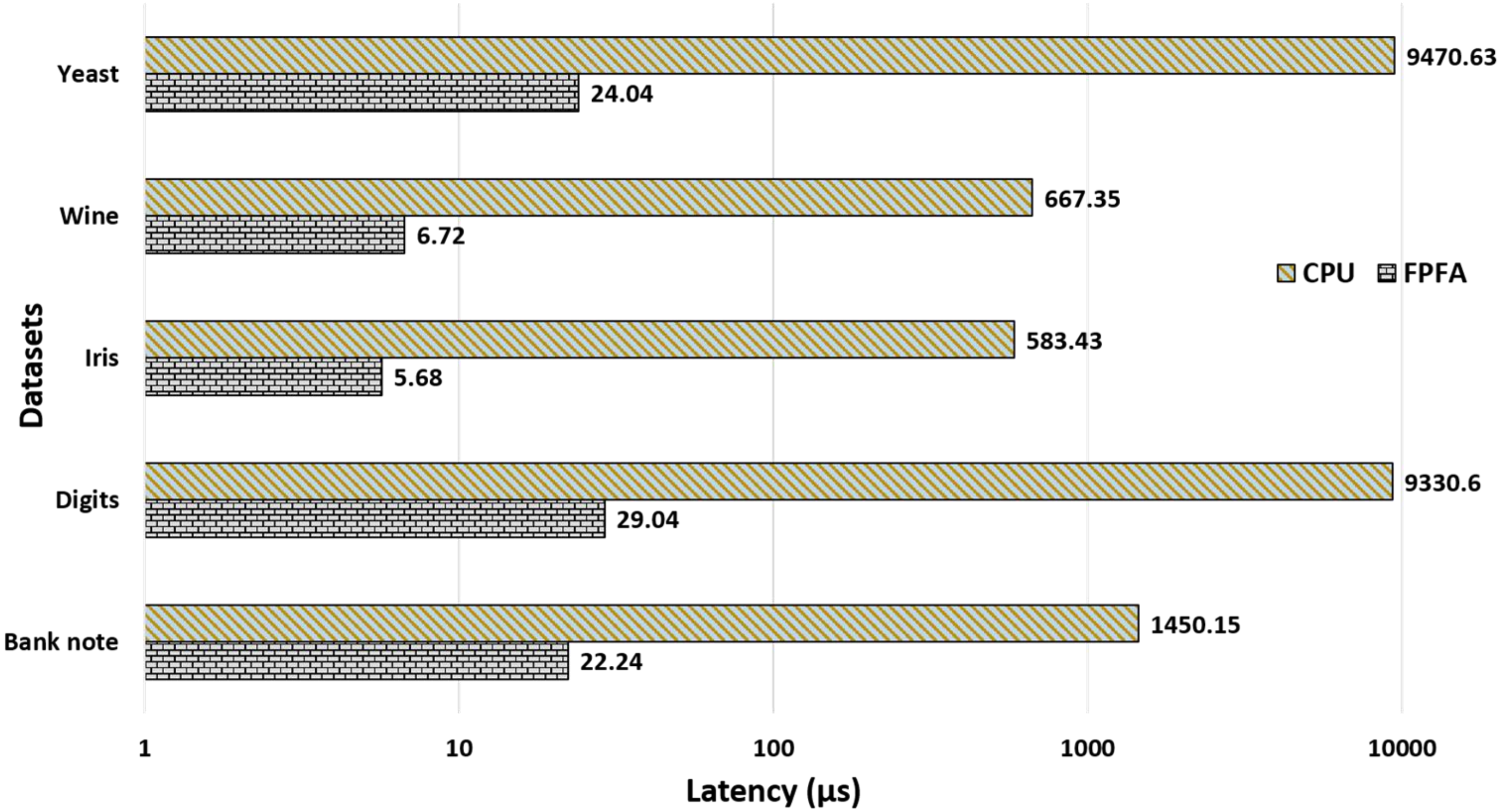
Resource Utilisation : LUTs



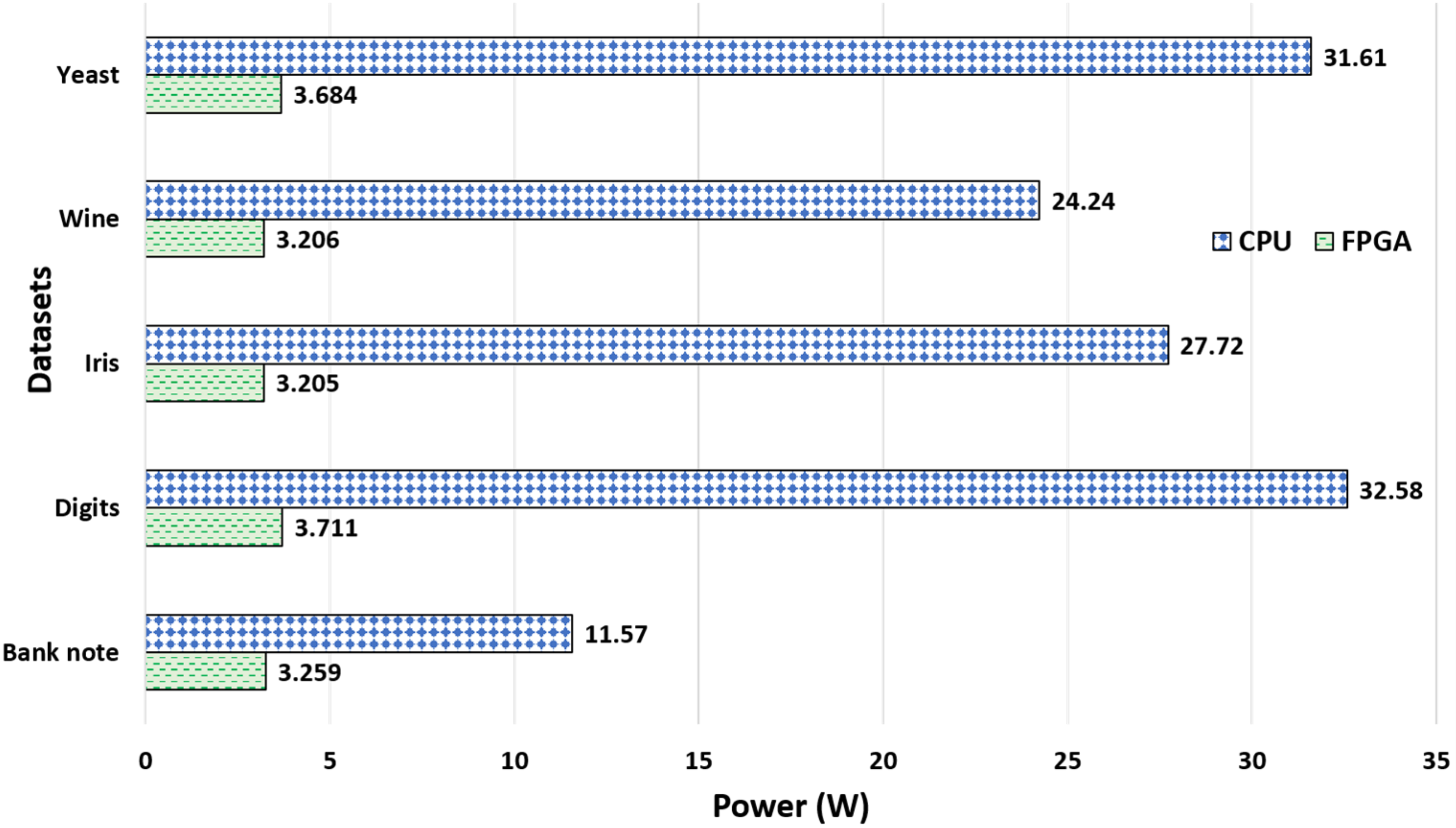
Latency & Throughput

- End to end latency is 8 clock cycles. Each tree generates output in 1 clock cycle, and since there are 100 iterations, $\log_2 100$ clock cycles, i.e. 7 clock cycles are required in the wrapper.
- It is fully pipelined, so new input can be fetched by the model at every clock cycle. Thus, throughput = clock frequency.

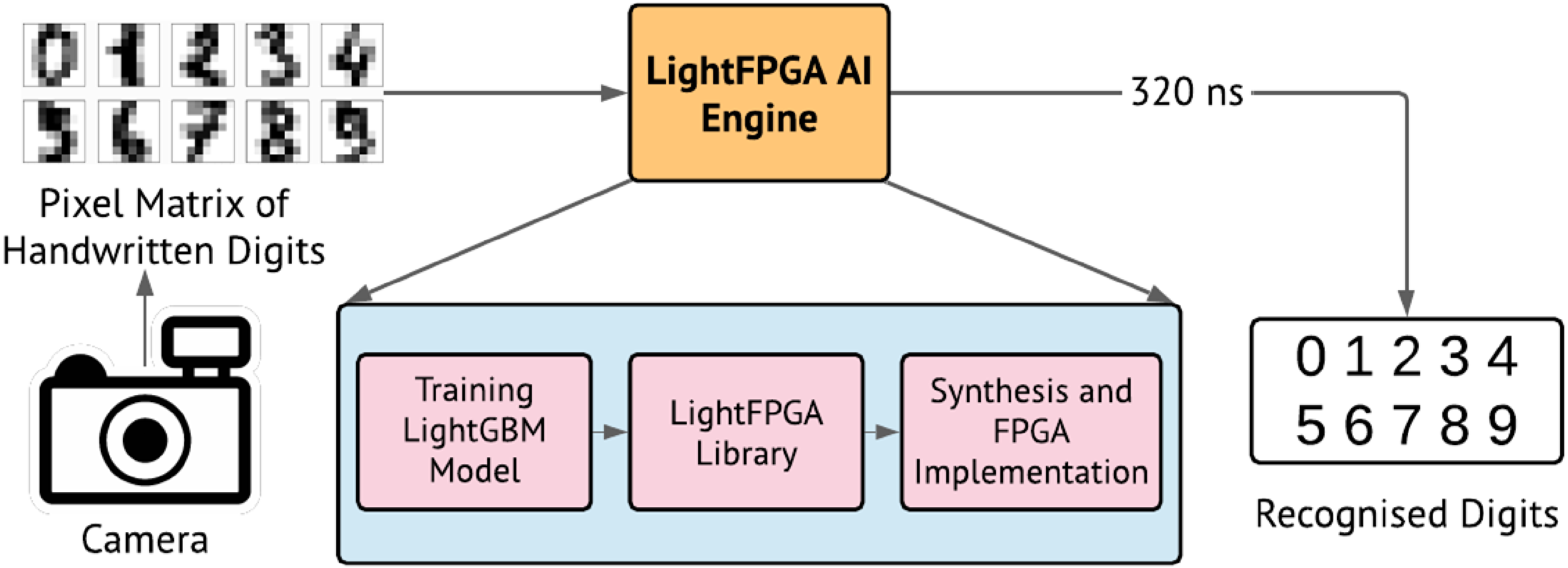
FPGA vs CPU: Latency Comparison



FPGA vs CPU: Power Comparison



LightFPGA : End to End Use-case



Thank You!

Q&A...