Multi-objective optimization for the CMS High Granularity Calorimeter Level 1 trigger

Hakimi Alexandre, LLR Ecole Polytechnique CNRS
Sauvan Jean-Baptiste, **LLR Ecole Polytechnique CNRS**

IN2P3/IRFU Machine Learning Workshop

Introduction

- ML: optimize internal parameters
- but **external hyperparameters**?
  - number of neurons, regularization etc. for DNN
  - tree depth, number of boosting rounds etc. for BDTs

- Finding the **optimal** values is **hard**
  - grid search, random search
  - bayesian optimisation ⇒ good at finding the optimum for one objective function (e.g. efficiency of the model)

- But when you have **multiple objectives**?

Multi objectives problems

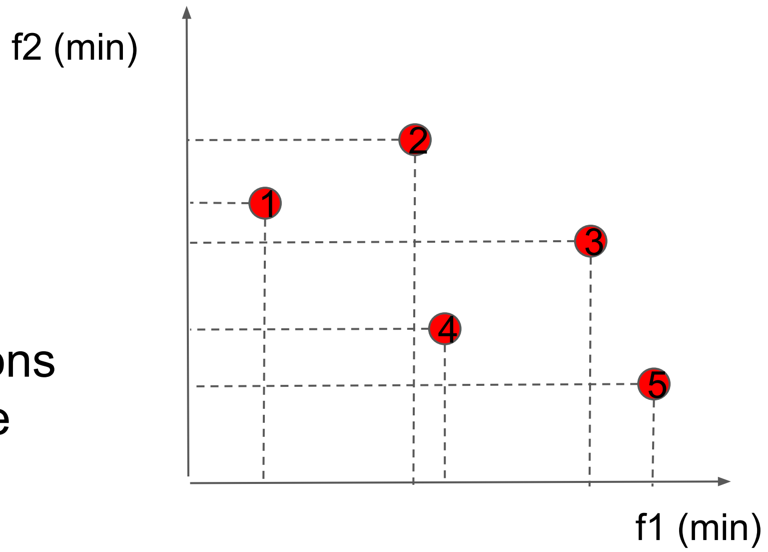
- ML problems may have other objectives:
  - minimizing resource usage (e.g. FPGA implementation)
  - efficiencies for different classes, etc.
  - often **competing** with each others
- "Easy" solution: combine them into one objective function:
  - $Obj = \alpha \times obj_1 + \beta \times obj_2 + \delta \times obj_3$
  - but the coefficient are arbitrary and there is a loss of information
- Solution: **MultiObjective Optimization (MOO)**
  - find the set of solutions that define the **best trade-off** between competing objectives

Dominance

Best solution?

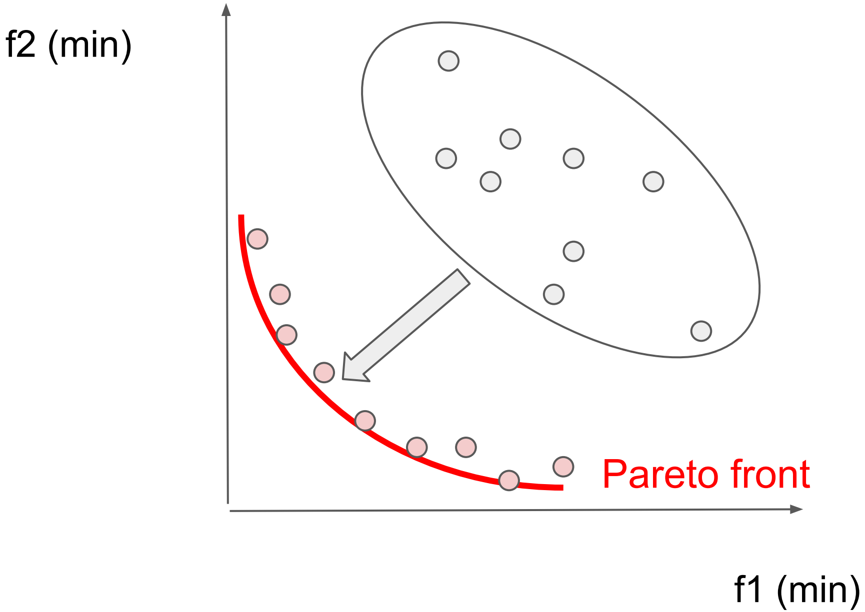
- single objective : easy
- multi-objective: **nondominated** solutions

⇨ **Nondominated** if can not improve one objective without degrading another one



- 2 is dominated by 1
- 3 is dominated by 4
- 1,4 and 5 are the non dominated solutions

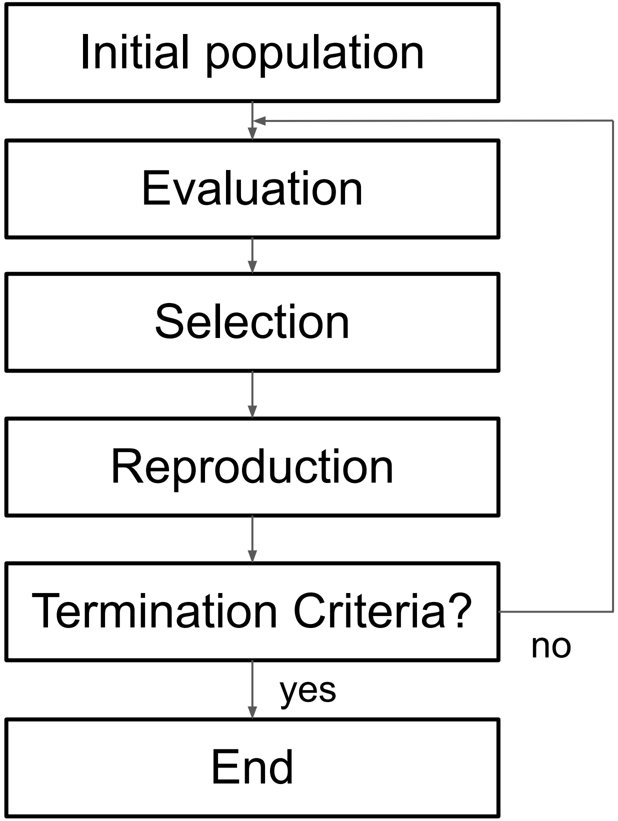
Pareto Front

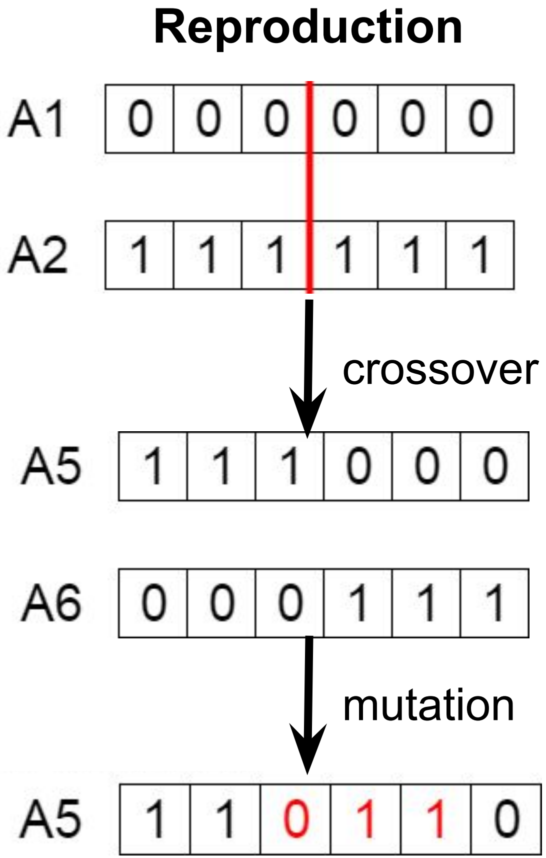


Go from initial…

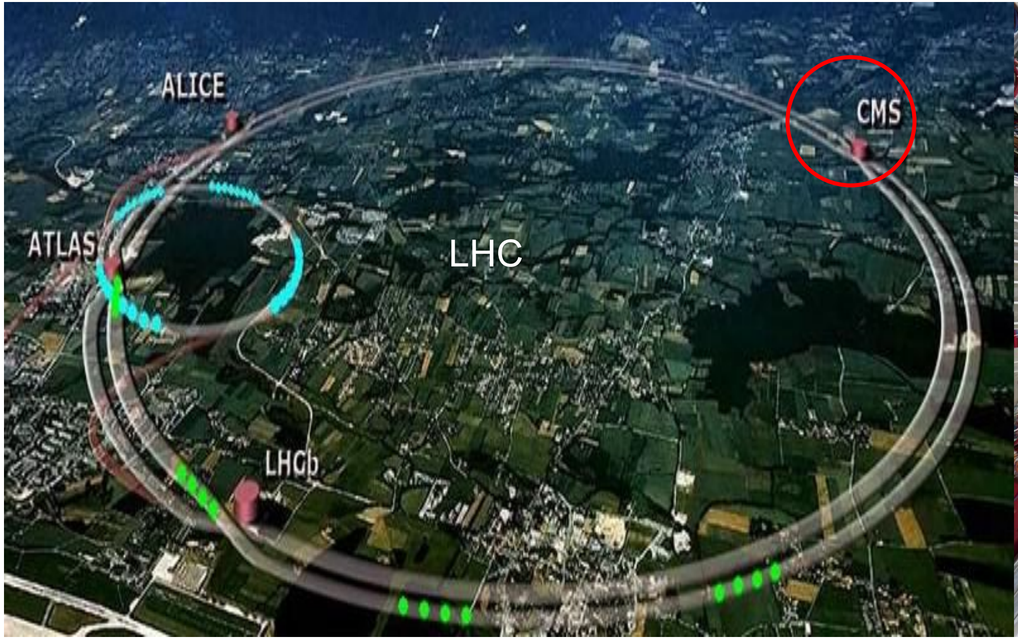
To optimal!

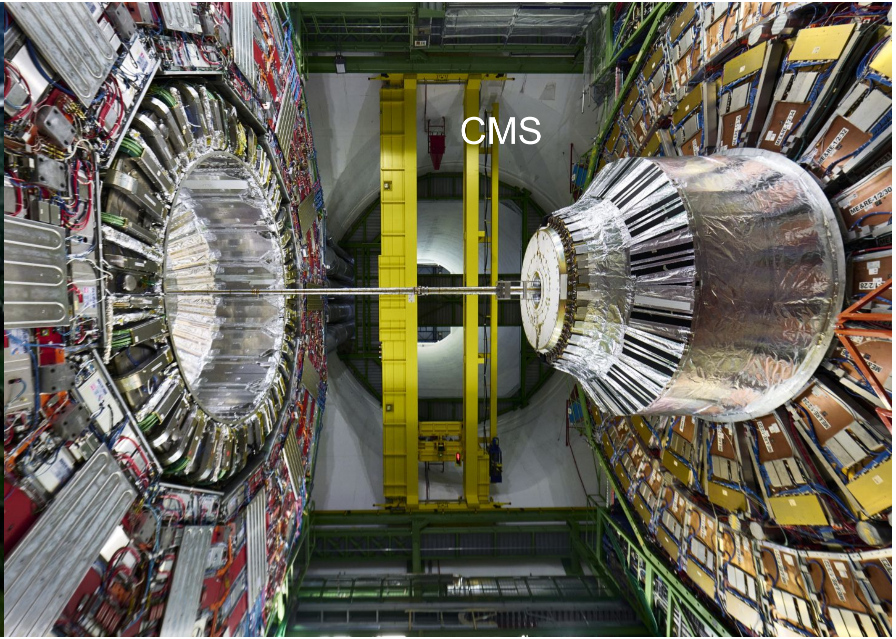
Genetic algorithm : NSGA-II





The CERN LHC and CMS experiment





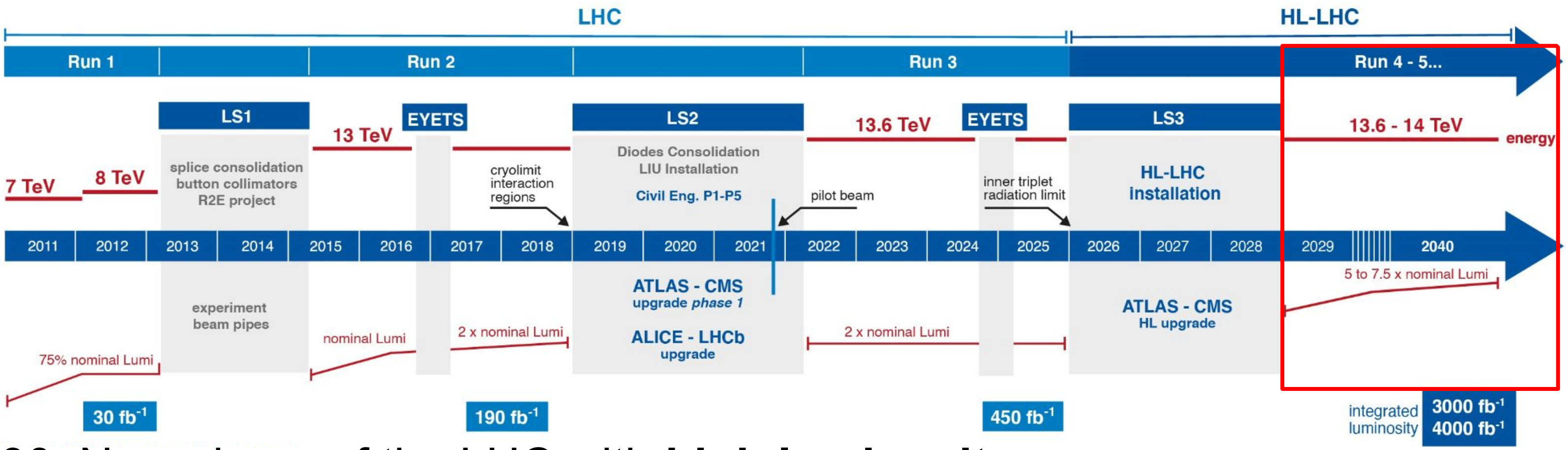
Collides protons at $\sqrt{s}$ = 13.6 TeV at very high frequency

- four main experiments: ALICE, ATLAS, CMS and LHCb

General purpose detector

- Higgs boson
- Physics at the TeV scale
- New physics?

High luminosity LHC



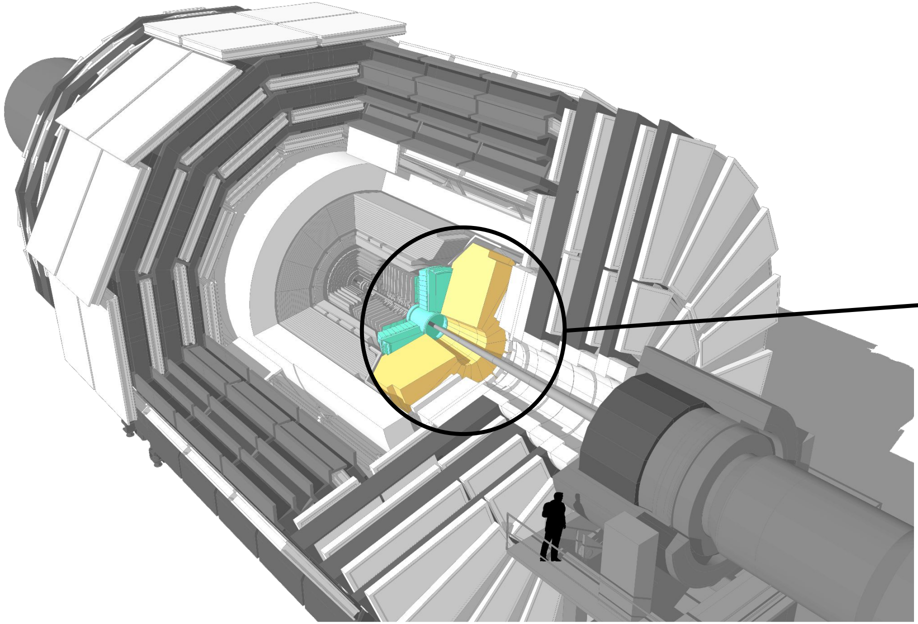
~2030: New phase of the LHC with **high luminosity**

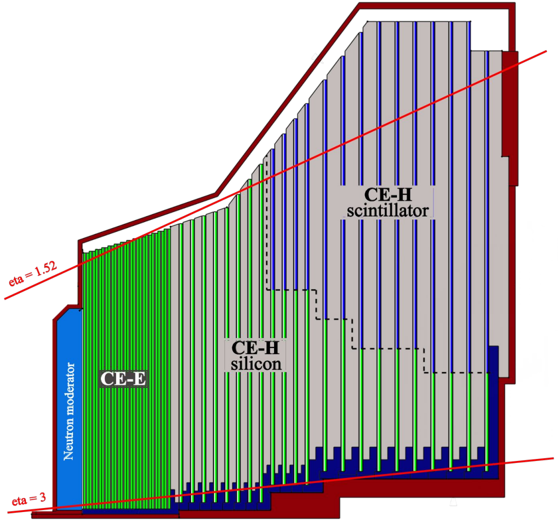
- more **collisions** and **high pileup** (multiplicity of the collisions in the detector)

⇨ **New challenges** : need detector upgrades!

- For **CMS**: Improved muon detector and tracker, more granular **endcap calorimeter** and **updated trigger system**

CMS Phase II endcap HGCAL



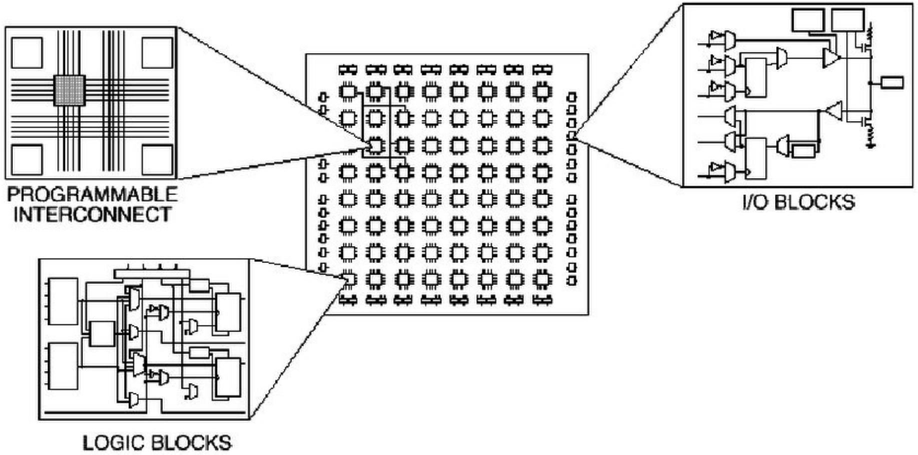


New calorimeter

- increased granularity
- 3D view of showers
- precise timing information

CMS Phase II L1 trigger

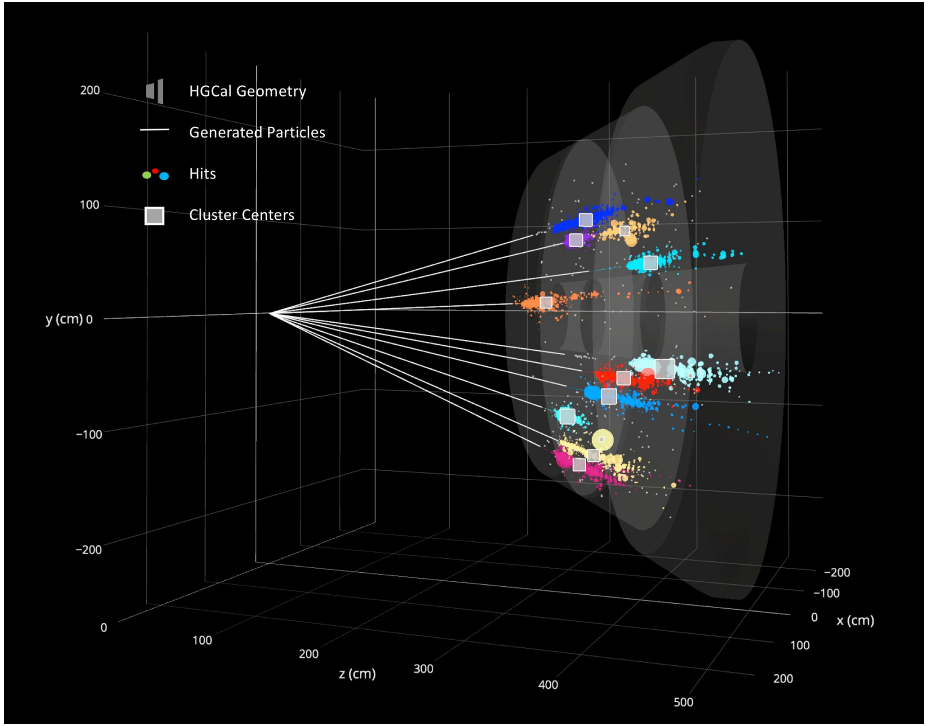
- **Trigger**: selects *interesting* events to be recorded
- Identify physics objects with **ML classifiers**
- In CMS: implemented on **FPGA** boards
    - Configurable logic
    - **Limited resources**, e.g. Lookup Tables (LUT) for logic
    - Integer or **fixed points operations**
- Uses inputs from the different CMS subdetectors
    - notably **trigger primitives from HGCAL**



HGCAL trigger primitives generation

Particles **shower** and deposit energy in the calorimeter

- This energy is reconstructed in **clusters**
- Variables describing the **shower shape** can be used to discriminate the type of shower
  - **transverse profile** and **longitudinal profile** help discriminate between electromagnetic and hadronic showers or low-energy pileup
- **Limited throughput** to the L1 trigger (128 bits)

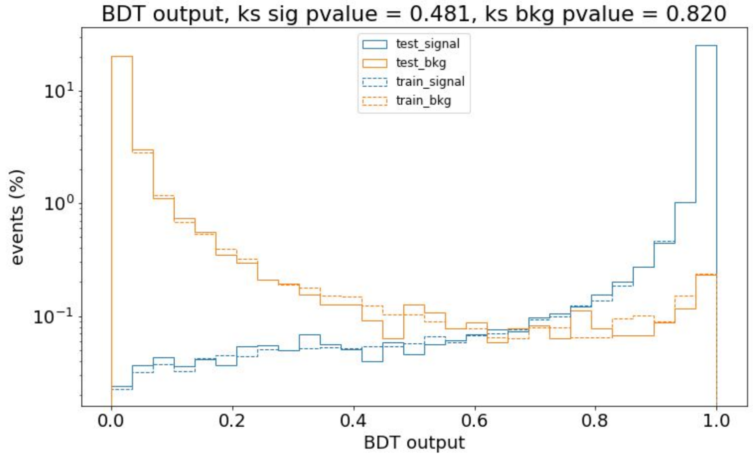


Electromagnetic shower discrimination

- **Goal**: Discriminate electrons clusters and PU clusters
  - using only HGCAL trigger primitives
- **Multi-objective optimization problem**
  - maximize performance
  - while minimizing the resource usage
  - and minimizing the throughput usage

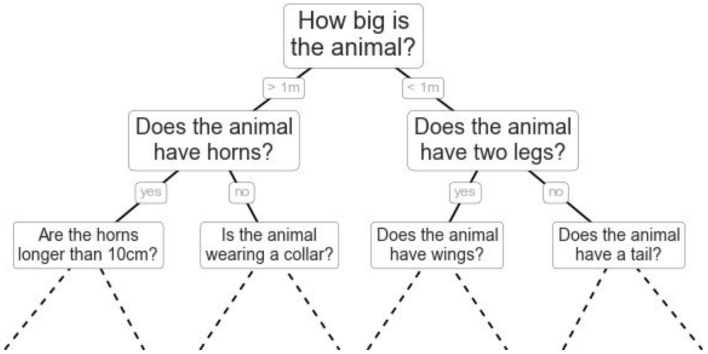
Chosen architecture: XGboost **BDTs**

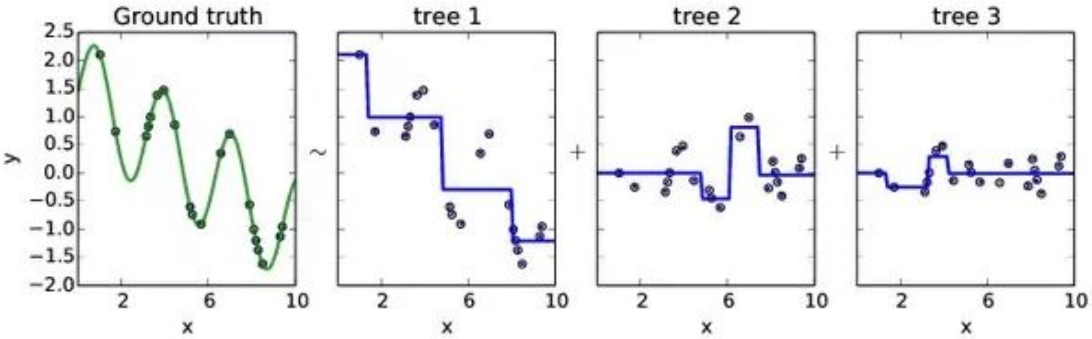
- DNNs can be used with similar performance
- Available conversion software for FPGA implementation: [conifer]



BDTs

- Boosted Decision Trees : combine **weak learners** (decision trees) into **strong classifier**
- Create a tree, measure it accuracy (loss function)
- **Boosting**: give more weight to misclassified events (**residuals**) and train new tree





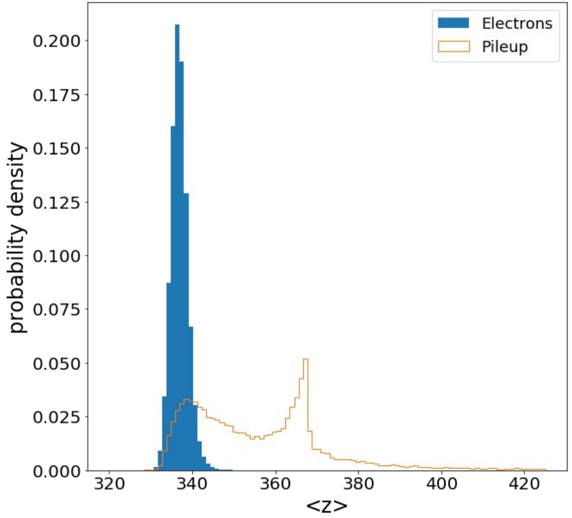
Hyperparameters:

- tree depth
- learning rate: how much change to the weights per iteration
- number of trees/boosting rounds

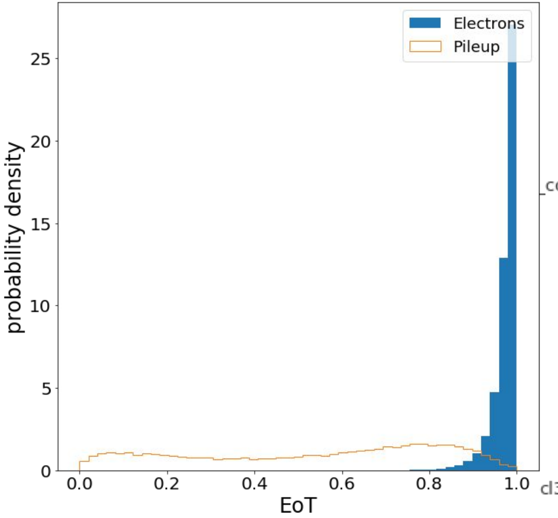
List of inputs

Most discriminating cluster shape variables in terms of SHAP values

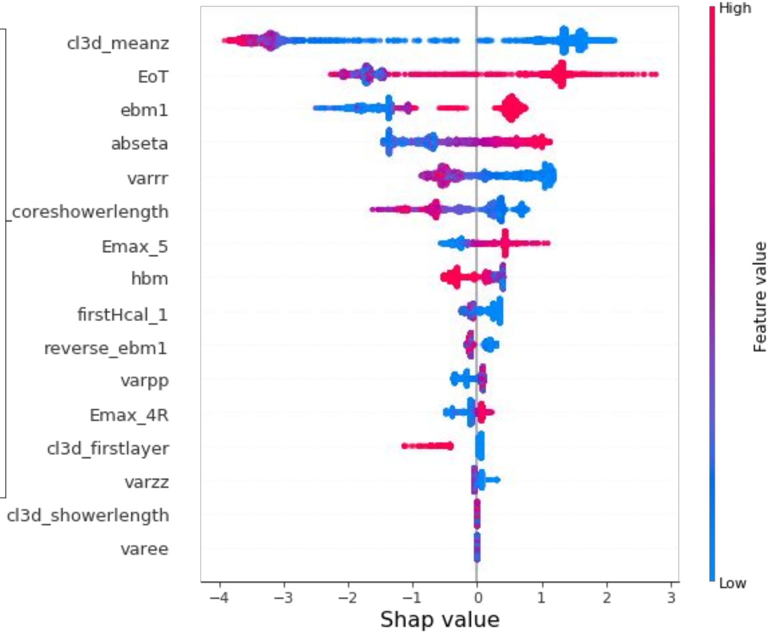
- game theory technique for explaining the output of ML models



Mean of the z-coordinates of trigger cells in cluster



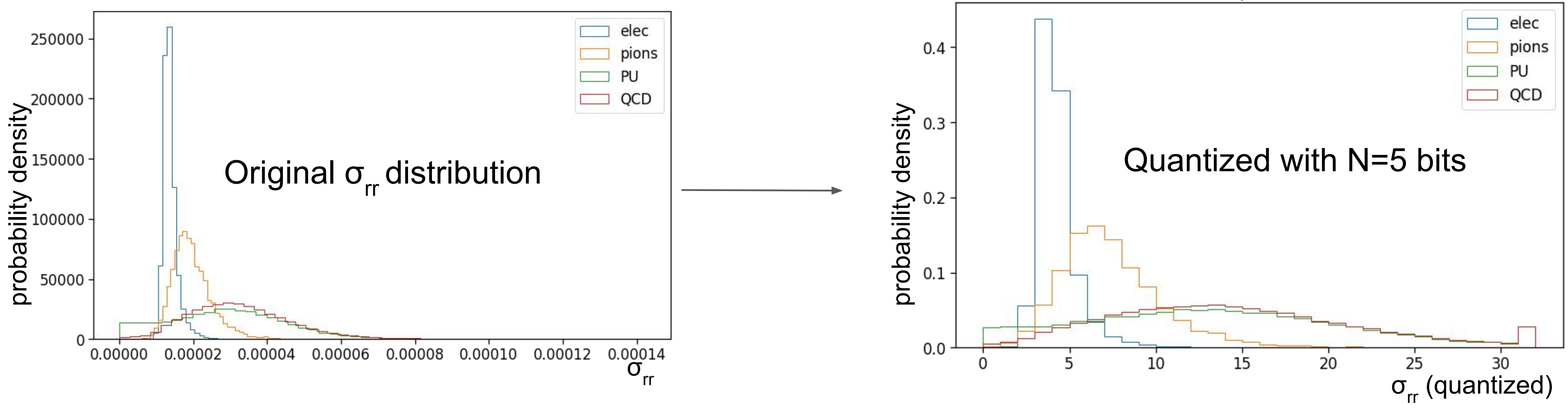
Fraction of energy in the electromagnetic part



background-like signal-like

Inputs quantization

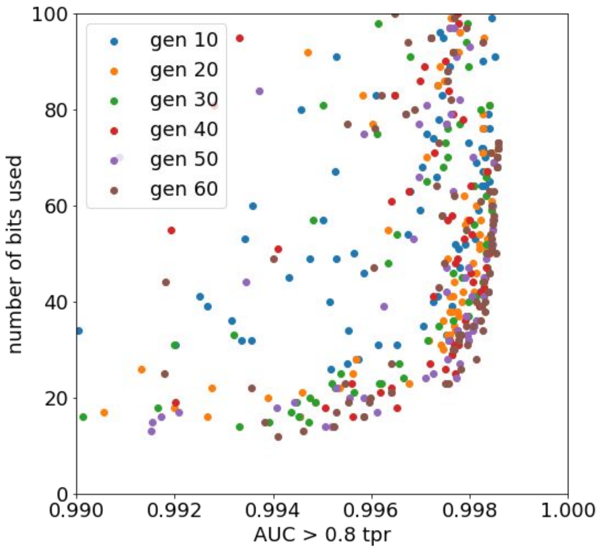
- Inputs need to be quantized into fixed points quantities for the FPGAs
    - between 0 and 16 bits precision (0 means dropped)

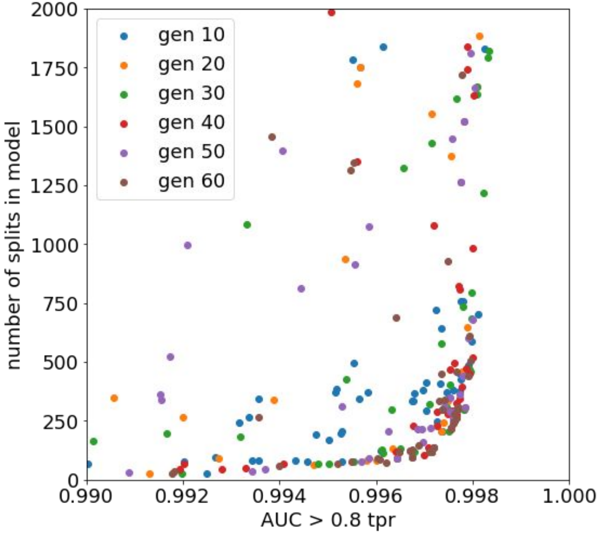


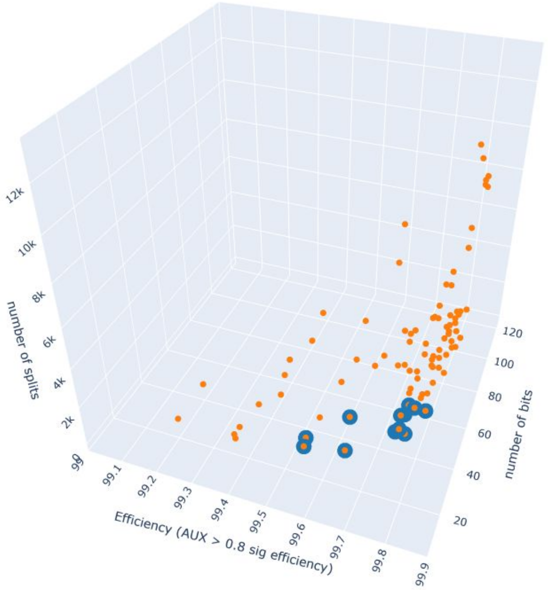
Model optimization

- **maximize** performance : ROC AUC (above a 80% signal efficiency threshold)
- **minimize** throughput: total number of bits used for input variables
  - also lowers the resources for their computation and BDT model
- **minimize** model size: number of “splits” in the BDT
  - reduce the amount of resource needed
- **Hyperparameters**:
  - number of bits for each input (16 params)
  - model complexity parameters (3 params)
    - max tree depth
    - number of boosting rounds
    - learning rate

Results





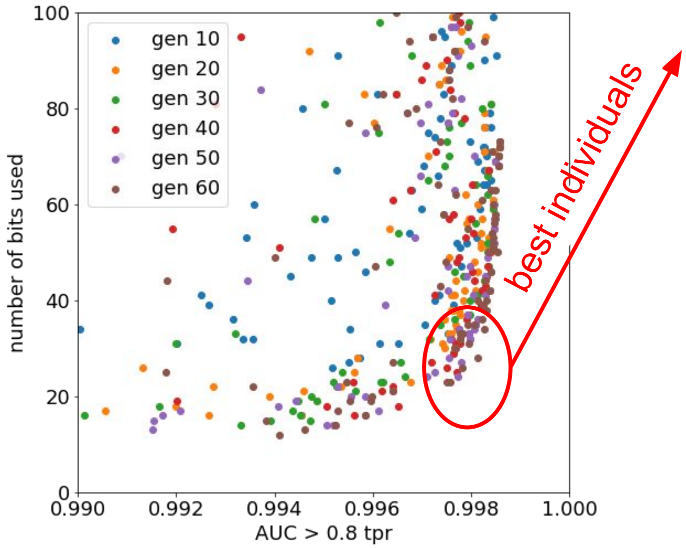


The optimal front is converging well across all dimensions

An interactive 3D visualisation is available here

On the right are highlighted the “solutions”: points in the last generations maximizing our objectives

Results (cont.)



	EoT	sho lgth	vzz	1st lay	firstH_5	Emx_2R	core lgth	1stH_1	meanz	Emx_5	Emx_4R	abseta	vee	vrr	vpp	ebm1	max boost rounds	max depth	eta	n_kept	bits	splits	eff
7	0	1	2	1	4	0	5	1	1	5	3	5	1	7	9	10	207	2	0.1	14	55	611	99.79
8	0	0	0	1	1	5	0	1	1	0	0	1	5	3	7	10	246	4	0.1	10	35	2945	99.79
5	0	1	0	2	1	0	1	3	1	0	4	2	0	4	3	11	146	4	0.1	11	33	1717	99.78
8	0	1	0	1	1	2	1	0	0	2	4	2	1	4	2	9	147	5	0.1	12	30	2973	99.77
4	2	0	0	1	1	5	1	1	1	2	0	0	1	3	2	10	120	5	0.1	12	30	2249	99.77
3	2	0	2	1	1	0	5	2	1	0	5	3	3	5	8	10	144	3	0.1	13	48	929	99.75
7	1	1	3	3	1	0	1	1	10	5	4	4	4	5	5	10	114	2	0.1	15	58	341	99.74
9	0	0	1	0	1	1	1	0	1	4	1	5	5	5	7	11	67	3	0.1	12	43	449	99.74

features bits   model params   objectives

- Each line correspond to an optimal solution
- can pick the one with the most interesting trade-off
- can look at the whole picture
  - some inputs are more ‘bit-effective’
  - some can be more ‘size-effective’

Best solutions

- Only keep ~12 inputs out of 16
- Number of bits used can be reduced a lot
- Between 20-60 bits
  - foreseen cluster data size of 128 bits
- Total size <3000 splits
  - Very small trees (90 splits) can perform well

	EoT	sho lgth	vzz	1st lay	firstH_5	Emx_2R	core lgth	1stH_1	meanz	Emx_5	Emx_4R	abseta	vee	vrr	vpp	ebm1	max boost rounds	max depth	eta	n_kept	bits	splits	eff
76	0	1	2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	9	10	207	2	0.1	14	55	611	99.79
83	0	0	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	7	10	246	4	0.1	10	35	2945	99.79
50	0	1	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	3	11	146	4	0.1	11	33	1717	99.78
86	0	1	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2	9	147	5	0.1	12	30	2973	99.77
42	2	0	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2	10	120	5	0.1	12	30	2249	99.77
37	2	0	2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	8	10	144	3	0.1	13	48	929	99.75
78	1	1	3	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	5	10	114	2	0.1	15	58	341	99.74
96	0	0	1	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	7	11	67	3	0.1	12	43	449	99.74
88	0	1	2	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	1	10	245	2	0.1	11	27	690	99.64
99	0	2	0	1	0	2	0	7	1	3	0	3	5	6	7	10	30	2	0.1	11	47	90	99.6
51	0	0	0	1	1	0	1	0	1	0	4	1	1	4	4	1	109	4	0.1	10	19	1346	99.55
34	0	2	0	0	0	1	1	1	1	1	0	3	4	3	5	1	221	3	0.1	11	23	1314	99.55

Best solutions

	EoT	sho lgth	vzz	1st lay	firstH_5	Emx_2R	core lgth	1stH_1	meanz	Emx_5	Emx_4R	abseta	vee	vrr	vpp	ebm1	max boost rounds	max depth	eta	n_kept	bits	splits	eff
76	0	1	2	1	4	0	5	1	1	5	3	5	1	7	9	10	207	2	0.1	14	55	611	99.79
83	0	0	0	1	1	5	0	1	1	0	0	1	5	3	7	10	246	4	0.1	10	35	2945	99.79
50	0	1	0	2	1	0	1	3	1	0	4	2	0	4	3	11	146	4	0.1	11	33	1717	99.78
86	0	1	0	1	1	2	1	0	0	2	4	2	1	4	2	9	147	5	0.1	12	30	2973	99.77
42	2	0	0	1	1	5	1	1	1	2	0	0	1	3	2	10	120	5	0.1	12	30	2249	99.77
37	2	0	2	1	1	0	5	2	1	0	5	3	3	5	8	10	144	3	0.1	13	48	929	99.75
78	1	1	3	3	1	0	1	1	10	5	4	4	4	5	5	10	114	2	0.1	15	58	341	99.74
96	0	0	1	0	1	1	1	0	1	4	1	5	5	5	7	11	67	3	0.1	12	43	449	99.74
88	0	1	2	0	1	0	1	1	1	0	1	1											
99	0	2	0	1	0	2	0	7	1	3	0	3											
51	0	0	0	1	1	0	1	0	1	0	4	1											
34	0	2	0	0	0	1	1	1	1	1	0	3											

- Low precision for most variables
  - even 1 bit can be enough!
- Even for important variables
  - <z>, EoT

Conclusion

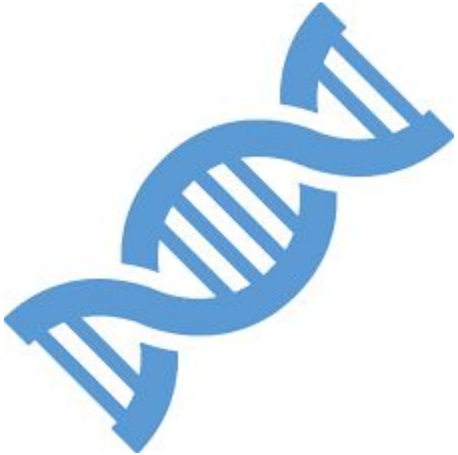
Can find models that satisfy the requirements:

- use a very **limited number of bit,**
  - can use less than **60 bits** for around **12 variables**
  - fit in 128 bit budget
- with **limited size** (need FPGA synthesis but expected to fit),
- **without sacrificing** the performance

- **Limits**:
  - high number of trainings (here: 60 gen * 100 pop)
    - fast for BDTs, but more complex models?
  - number of steps needed for convergence increases with the number of hyperparameters and/or objectives

Backup: Genetic/Evolutionary algorithm

- **Evolutionary algorithms** (EA) are optimization algorithms inspired by darwinian evolution
- Uses an **imitation of nature**'s tools to find optimal solution to a problem
  - mating
  - mutation
  - selection
- Different algorithms
  - multi-objective: Non-dominated sorting algorithm 2 (**NSGA-II)**
  - for high-dimensional problems: NSGA-III
  - multiple variations (see pymoo availables algorithms for examples)



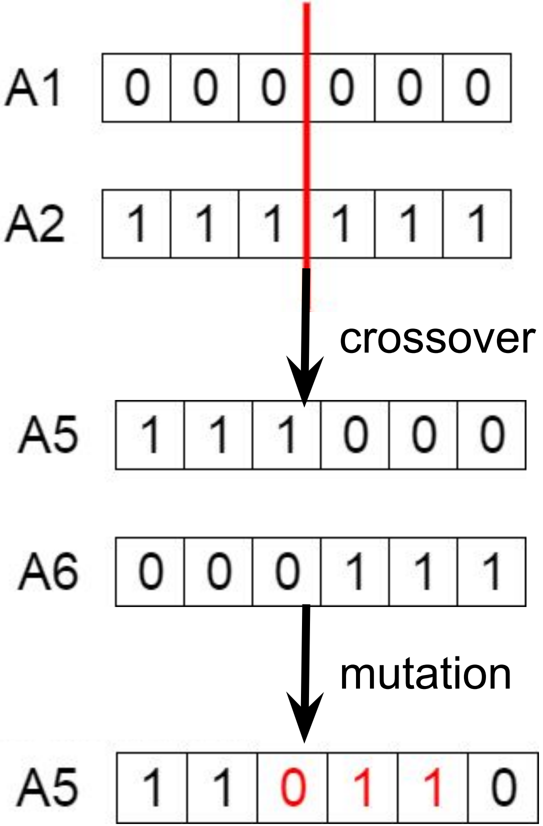
Backup: NSGA-II : initialization & evaluation

- 1st step: initialization
  - create a starting population
  - should ideally sample most of the phase space
    - random
    - hyperdiagonal
    - functional

- 2nd step: evaluation
  - determine which individuals are non dominated

Backup: NSGA-II: Reproduction, mating and crossover

- 3rd step: **reproduction**
  - new generation obtained by **reproducing pairs** of ND individuals
    - e.g. random, local, tournament selection pairing
  - **crossover**: mix the genes (parameter values)
    - tunable
  - **mutation**: genes can take new value
    - tunable
  - cycle to first step until **termination criterion** is met
    - e.g. a given number of generation for example



Backup: pymoo implementation

```python
from pymoo.factory import get_algorithm, get_crossover, get_mutation, get_sampling, get_selection
from pymoo.optimize import minimize
from pymoo.model.problem import Problem


class MyProblem(Problem):

    def __init__(self):
        super().__init__(n_var=nvar, #number of variables in the problem (ie hyperparameters)
                         n_obj=nobj, #number of objectives
                         n_constr=nconstr, #number of constraints
                         xl=xl, #lower bound for variables
                         xu=xu, #higher bound for variables
                         type_var=int,
                         elementwise_evaluation=True) #Only one solution is evaluated at a time,
                                                      #vectorized or functional evaluation supported

    def _evaluate(self, x, out, *args, **kwargs):
        out["F"] = train_quantized(x)[0] #black box function to evaluate, should return n_obj values
problem=MyProblem()
```

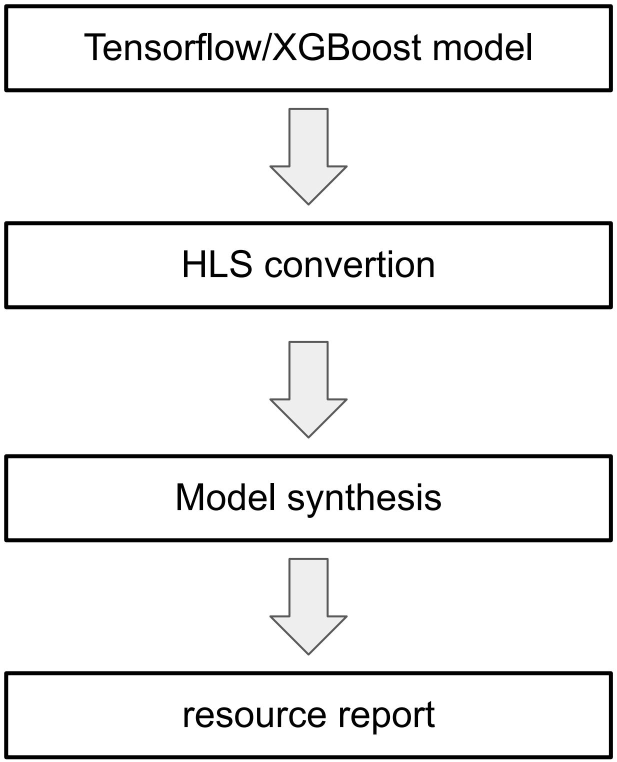
Backup: pymoo implementation (cont)

```
method = get_algorithm("nsga2",
                       pop_size=pop_size, #number of individual at each generation
                       sampling=sampling, #initial population
                       crossover=get_crossover("int_sbx", prob=1.0, eta=3.0), #crossover function
                       mutation=get_mutation("int_pm", eta=3.0), # def 3 #mutation functin
                       eliminate_duplicates=True,
                       )


res = minimize(problem,
               method,
               termination=('n_gen', 60), #end point
               seed=42,
               save_history=True,
               verbose=True
               )

```

FPGA resource usage

- Algorithms implemented on FPGA occupy a certain amount of ressource (for BDTs, in particular LUT)
- Synthesis software like VIVADO or VITIS (for XILINX boards) can simulate the implementation and provide a resource rapport
  - the models must be converted before synthesis
  - for NN: hls4ml library
  - for BDT: conifer
- But conifer has limitations:
  - can not allocate different precision to each input variable
  - theoretically possible, not implemented yet

⇨ Build a proxy to evaluate quickly a model resource usage



FPGA resource proxy

- Idea: estimate the amount of resource (LUT) used for a tree split with a given precision
- Train BDTs, synthesize them and measure ratio nb of LUT/nb of split
  - differentiate results by the precision but also max tree depth
- at high precision : linear behavior, easy to determine ratio
- at lower precision: plateauing effect
  - precision affect every quantity, even each tree coefficient
  - tree with low coeffs suppressed at low precision
  - extract ratio from linear part
- at very low precision: linear behavior hard to extract, close to 1 LUT per split

NSGA2 parameters

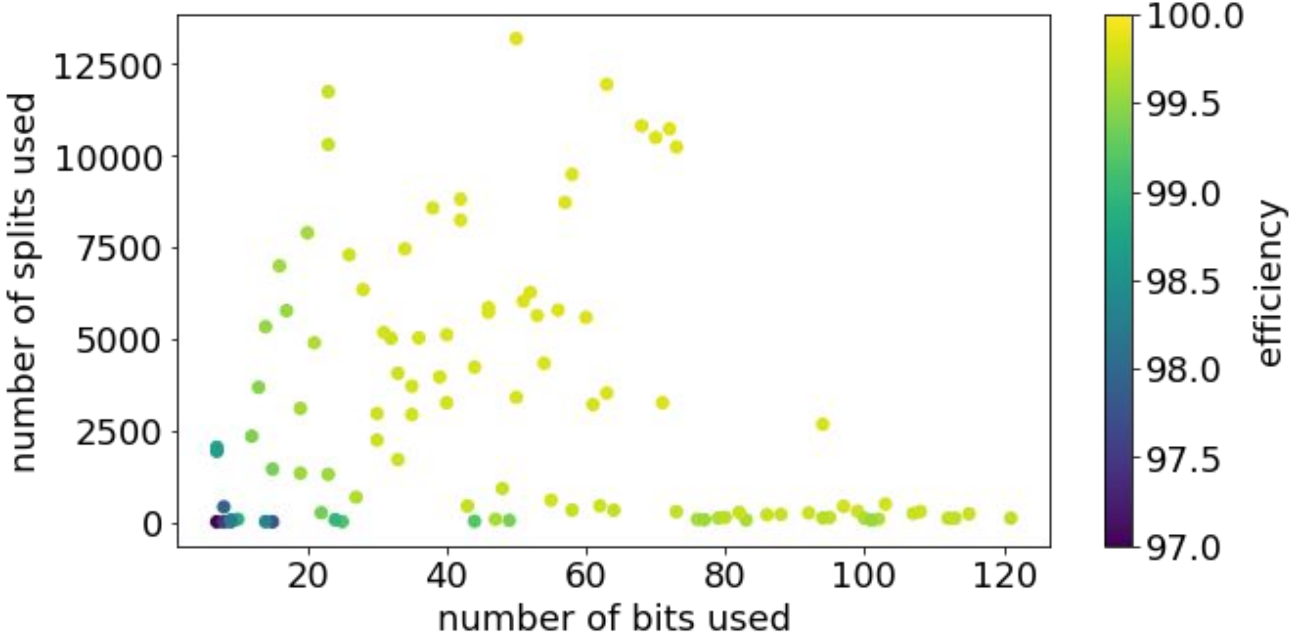
- Initialization:
  - hyperdiagonal + random (total = 100 individuals)
- Mating:
  - random pairing
  - binary crossover based on an exponential probability distribution
  - polynomial mutation (exponential distribution, $\eta = 3$)
  - 100 children per generation
- Termination at 60 cycles

Best solutions

	EoT	sho lgth	vzz	1st lay	firstH_5	Emx_2R	core lgth	1stH_1	meanz	Emx_5	Emx_4R	abseta	vee	vrr	vpp	ebm1	max boost rounds	max depth	eta	n_kept	bits	splits	eff
76	0	1	2	1	4	0	5	1	1	5	3	5	1	7	9	10	207	2	0.1	14	55	611	99.79
83	0	0							1	0	0	1	5	3	7	10	246	4	0.1	10	35	2945	99.79
50	0	1							1	0	4	2	0	4	3	11	146	4	0.1	11	33	1717	99.78
86	0	1							0	2	4	2	1	4	2	9	147	5	0.1	12	30	2973	99.77
42	2	0							1	2	0	0	1	3	2	10	120	5	0.1	12	30	2249	99.77
37	2	0							1	0	5	3	3	5	8	10	144	3	0.1	13	48	929	99.75
78	1	1							10	5	4	4	4	5	5	10	114	2	0.1	15	58	341	99.74
96	0	0							1	4	1	5	5	5	7	11	67	3	0.1	12	43	449	99.74
88	0	1							1	0	1	1	0	7	1	10	245	2	0.1	11	27	690	99.64
99	0	2							1	3	0	3	5	6	7	10	30	2	0.1	11	47	90	99.6
51	0	0							1	0	4	1	1	4	4	1	109	4	0.1	10	19	1346	99.55
34	0	2							1	1	0	3	4	3	5	1	221	3	0.1	11	23	1314	99.55

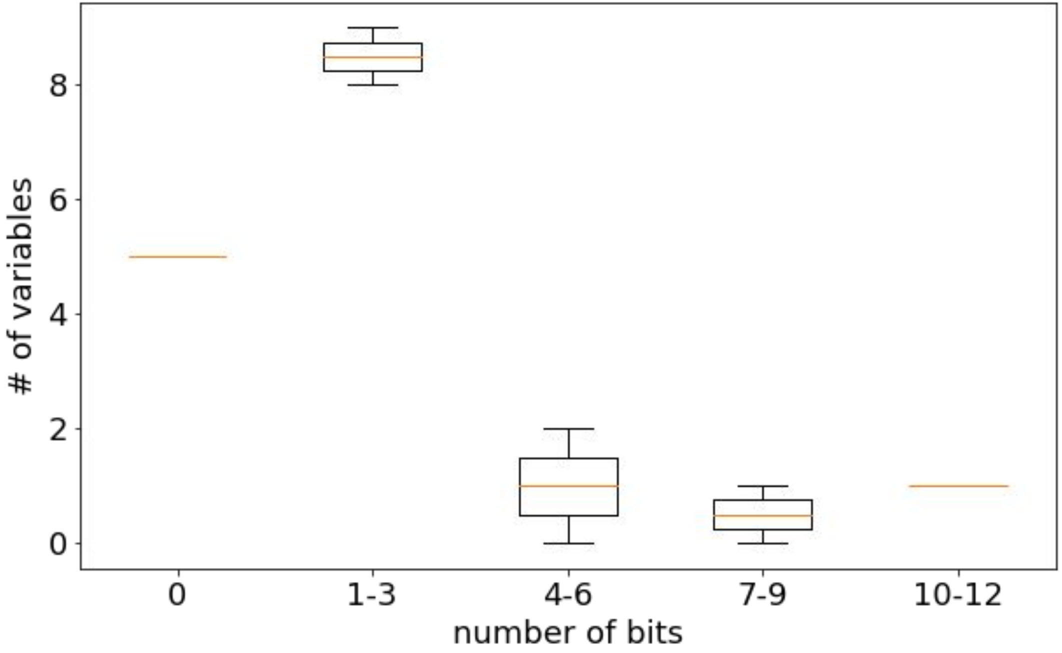
- ebm1 has the highest bit budget. Dropping the amount causes a significant drop in efficiency
- Different combination of medium budget for several features yields comparable results

Complexity vs efficiency



The best solutions seem to be the one balancing the size of the BDTs and the number of bits used

Distribution of number of bits per variable



- around ⅓ of the variables are dropped
- the majority are afforded a very small number of bits
- very few variables use more than 4 bits in most models