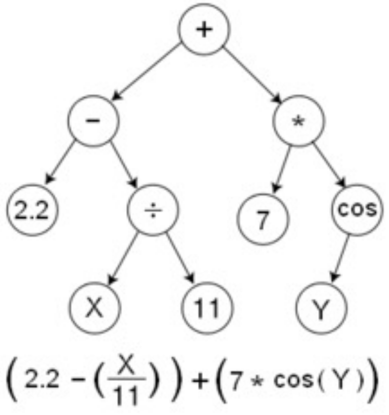
Symbolic Regression on FPGAs for Fast Machine Learning Inference



Ho Fung Tsoi, Adrian Alan Pol, Vladimir Loncar, Ekaterina Govorkova,
Miles Cranmer, Sridhara Dasu, Peter Elmer, Philip Harris, Isobel Ojalvo, Maurizio Pierini

26th International Conference on Computing in High Energy & Nuclear Physics (CHEP 2023)
Norfolk, VA, USA
May 8-12, 2023

Background

arXiv > cs > arXiv:2305.04099

Search...

Help | Advai

Computer Science > Machine Learning

[Submitted on 6 May 2023]

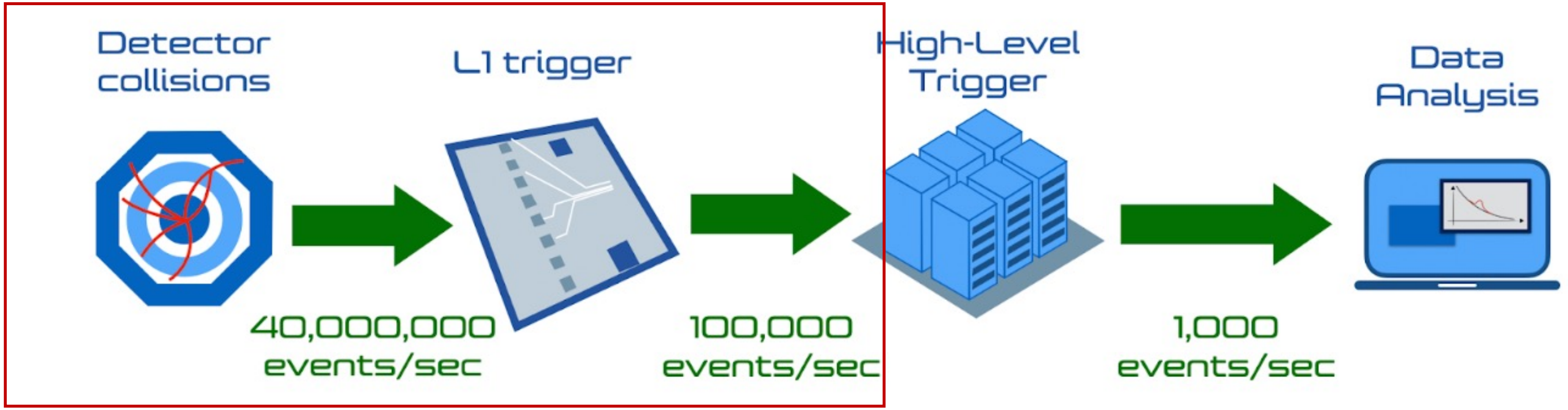
Symbolic Regression on FPGAs for Fast Machine Learning Inference

Ho Fung Tsoi, Adrian Alan Pol, Vladimir Loncar, Ekaterina Govorkova, Miles Cranmer, Sridhara Dasu, Peter Elmer, Philip Harris, Isobel Ojalvo, Maurizio Pierini

The high-energy physics community is investigating the feasibility of deploying machine-learning-based solutions on Field-Programmable Gate Arrays (FPGAs) to improve physics sensitivity while meeting data processing latency limitations. In this contribution, we introduce a novel end-to-end procedure that utilizes a machine learning technique called symbolic regression (SR). It searches equation space to discover algebraic relations approximating a dataset. We use PySR (software for uncovering these expressions based on evolutionary algorithm) and extend the functionality of hls4ml (a package for machine learning inference in FPGAs) to support PySR-generated expressions for resource-constrained production environments. Deep learning models often optimise the top metric by pinning the network size because vast hyperparameter space prevents extensive neural architecture search. Conversely, SR selects a set of models on the Pareto front, which allows for optimising the performance-resource tradeoff directly. By embedding symbolic forms, our implementation can dramatically reduce the computational resources needed to perform critical tasks. We validate our procedure on a physics benchmark: multiclass classification of jets produced in simulated proton-proton collisions at the CERN Large Hadron Collider, and show that we approximate a 3-layer neural network with an inference model that has as low as 5 ns execution time (a reduction by a factor of 13) and over 90% approximation accuracy.

This talk is based on the paper [2305.04099](2305.04099)

Background

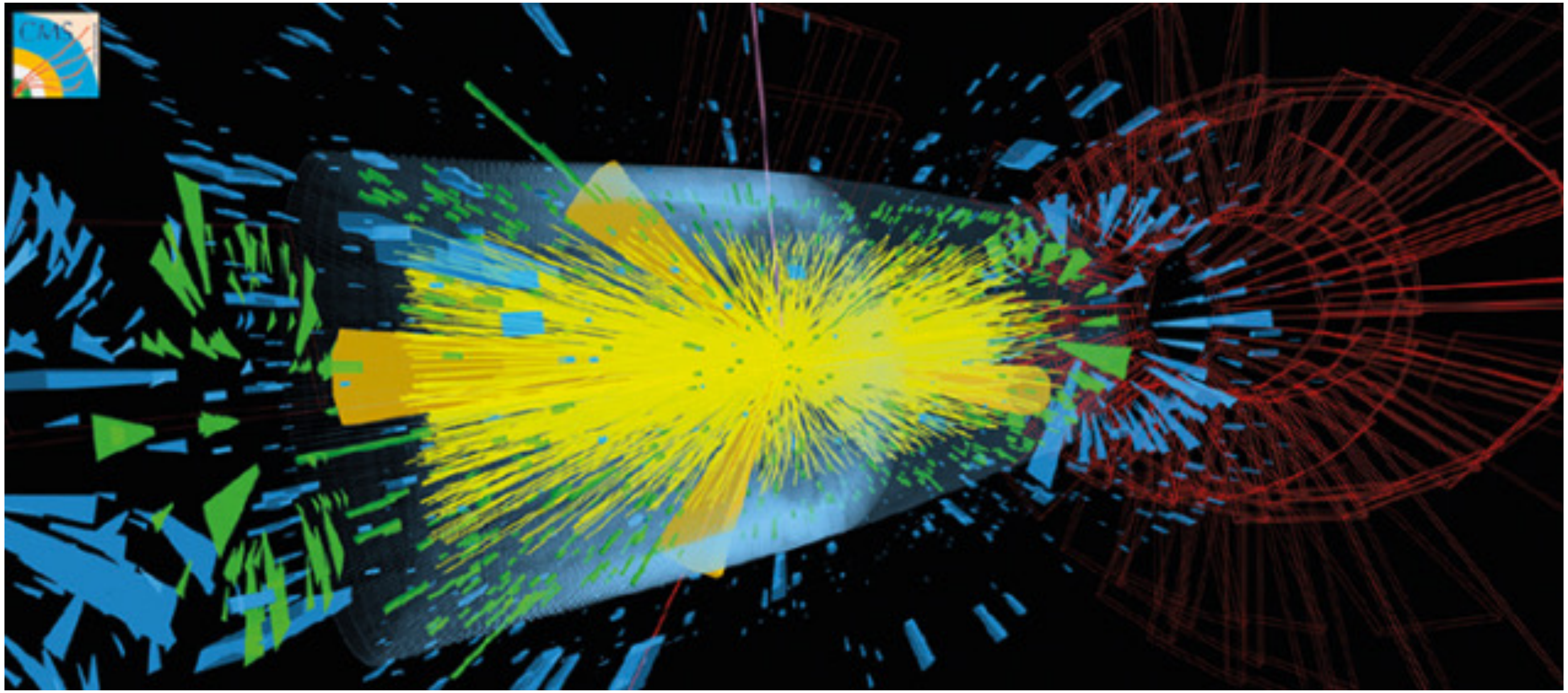


L1 trigger at the LHC reduces extreme data rates of $O(10)$ TB/s to a manageable level

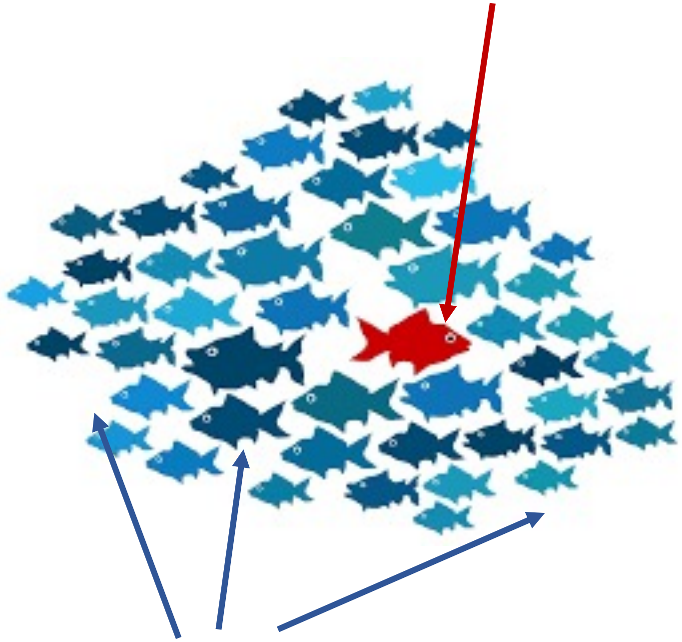
- It discards events forever! So we need very precise selection to keep interesting physics events
  - ML algorithms which can improve sensitivity to rare/new physics
- Strict computing resource constraints and ultra low-latency < $O(1)$ μs
  - Algorithms need to be extremely lightweight
  - Need to run on custom hardware such as FPGAs to achieve nanosecond inference

But always a performance-resource trade-off!

Background



“Rare” physics
(Higgs, BSM, etc.)



“Background” physics
(QCD, etc.)

Ongoing ML developments at the L1 trigger for LHC Run3 include the *anomaly triggers*

- Anomaly detection as an unsupervised learning that targets all “rare” events at once
- Search interesting physics events at the trigger level in a model-agnostic way

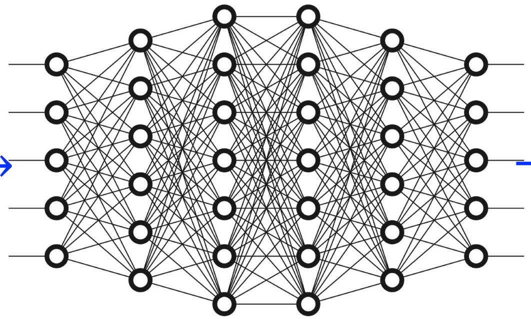
➢ Challenges: NN-based models can hardly fit to resource/latency constraints without largely compromising accuracy

Symbolic regression

Symbolic Regression (SR): a ML technique that seeks to discover analytic functions that approximate a dataset

NN

Dataset → input



→ Output

Big black box

vs.

SR

Dataset → input $f(x) = x_0^3 - \sin(x_1 x_3) + x_2 \log(5 + x_4) + \cdots$ → Output

One-line closed form solution

Offer interpretable results for the underlying problem

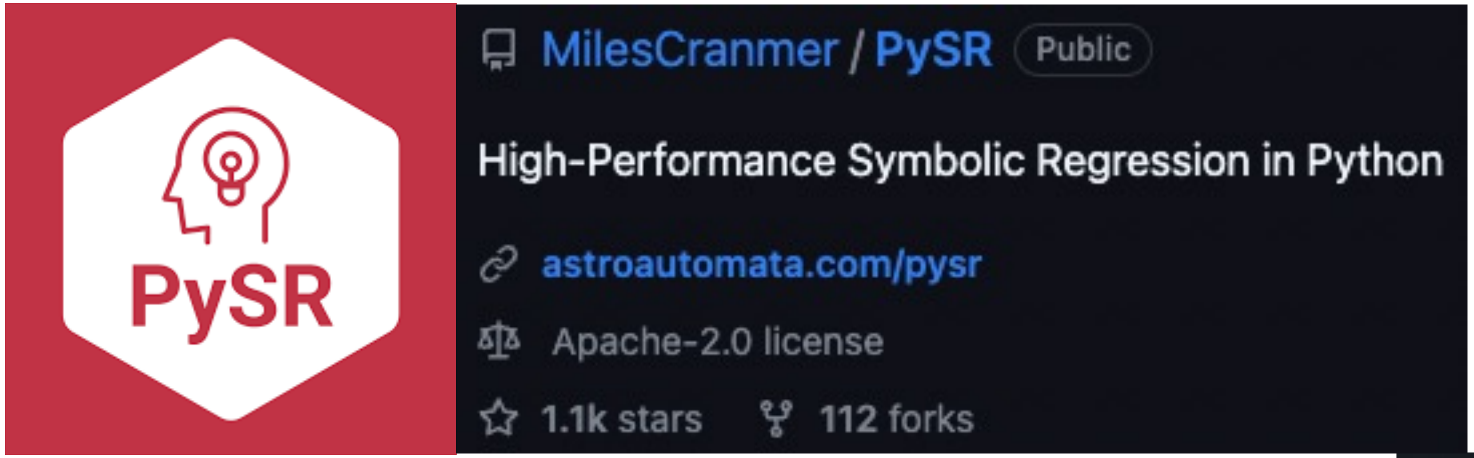
- E.g., rediscovering Newton's law of gravitation from observables 2202.02306

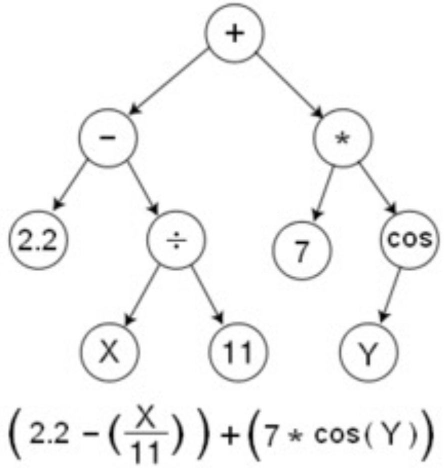
Unlike deep learning models, SR easily generates a set of models on the Pareto front, which allows for optimizing the performance-resource tradeoff directly

Potential to be highly efficient for resource-constrained production environments

- For low-dim problems: use SR as master model
- For high-dim problems: intermediate compression (e.g., distill a big NN)

High-performance symbolic regression in Python: *PySR*





Open-source and user-friendly

Based on genetic programming

- Create expression trees
- Trees grow and fittest ones can evolve to next generation
- Mutation and crossbreeding can happen to explore more expressions

Simple and flexible configuration for different use case

- Custom operators, loss, complexity definition, etc.

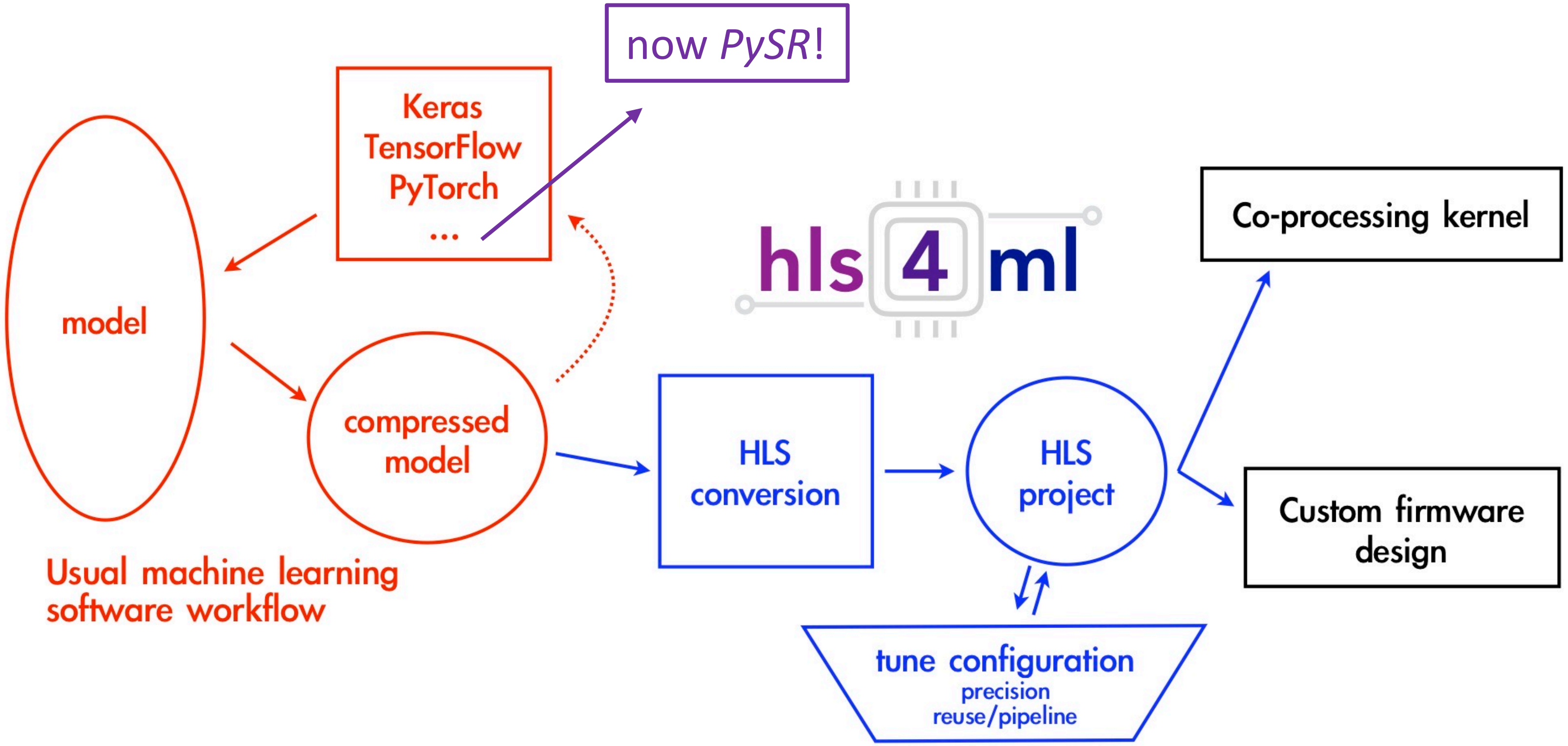
```python
from pysr import PySRRegressor

model = PySRRegressor(
    niterations=40,  # < Increase me for better results
    binary_operators=["+", "*"],
    unary_operators=[
        "cos",
        "exp",
        "sin",
        "inv(x) = 1/x",
        # ^ Custom operator (julia syntax)
    ],
    extra_sympy_mappings={"inv": lambda x: 1 / x},
    # ^ Define operator for SymPy as well
    loss="loss(prediction, target) = (prediction - target)^2",
    # ^ Custom loss function (julia syntax)
)
```

high level synthesis for machine learning

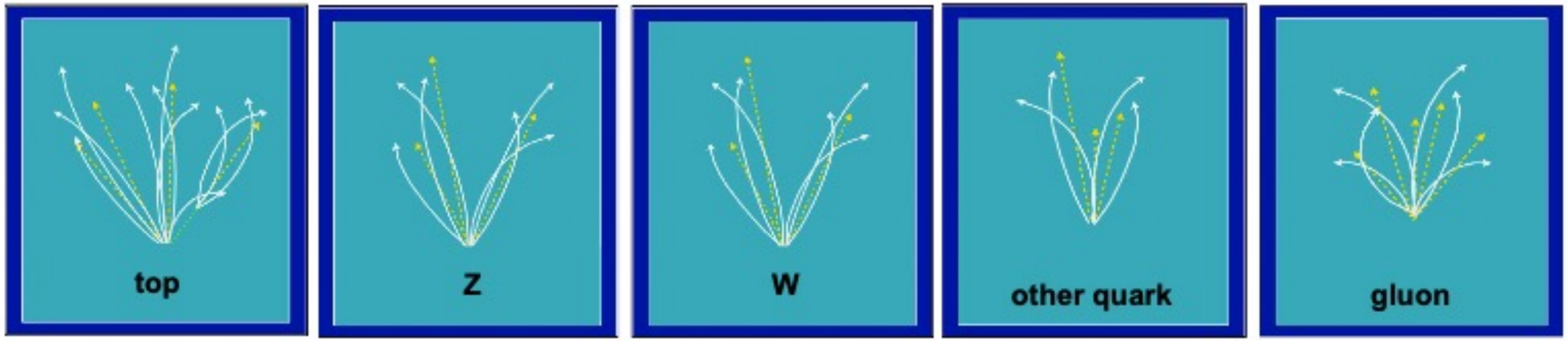
hls4ml: a user-friendly open-source Python package for fast ML inference in FPGAs

- ➢ Input trained models from standard libraries such as (*Q*)*Keras*, *PyTorch*,… and *PySR*
- ➢ Provide an efficient and fast translation to HLS code
- ➢ User can control model aspects for optimal performance on FPGAs
- ❖ Necessary for extreme environments such as LHC L1 trigger where resources are strictly constrained and a max latency of $O(1)$ μs is imposed



https://fastmachinelearning.org/hls4ml/

Dataset: LHC jet tagging



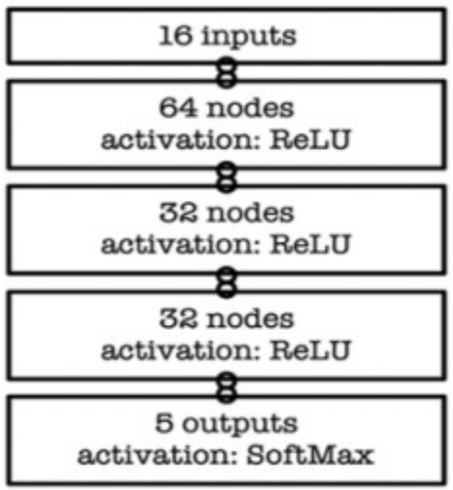
A physics benchmark: HLS4ML LHC Jet dataset publicly available at Zenodo, generated for FastML/HLS4ML studies

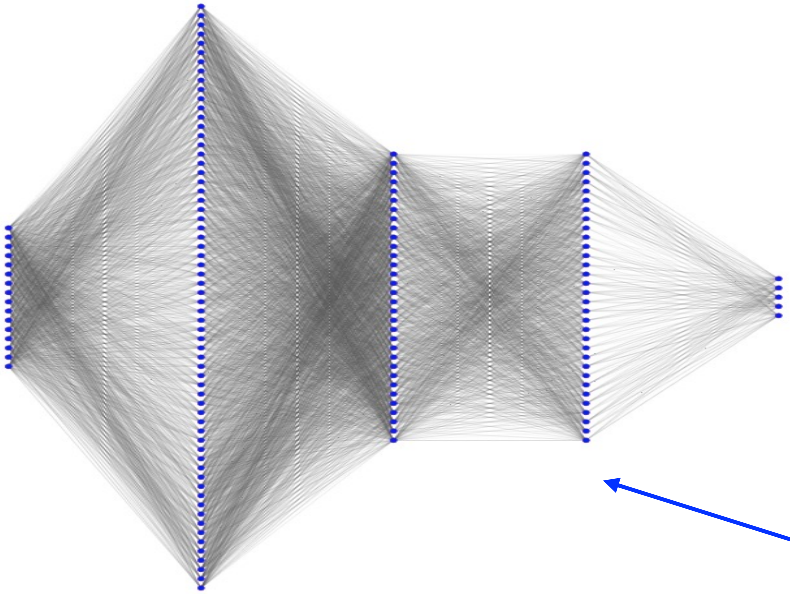
- Dataset of boosted jets from simulations of LHC proton-proton collisions (~1M simulated jets)
- Each jet represented by 16 high-level physics-motivated features
- Multiclass classification → {gluon, light-quark, W, Z, top}

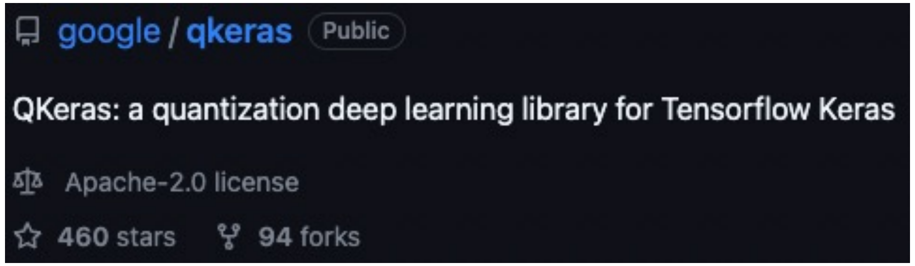
Observables
m_{mMDT}
$N_2^{\beta=1,2}$
$M_2^{\beta=1,2}$
$C_1^{\beta=0,1,2}$
$C_2^{\beta=1,2}$
$D_2^{\beta=1,2}$
$D_2^{(\alpha,\beta)=(1,1),(1,2)}$
$\sum z \log z$
Multiplicity

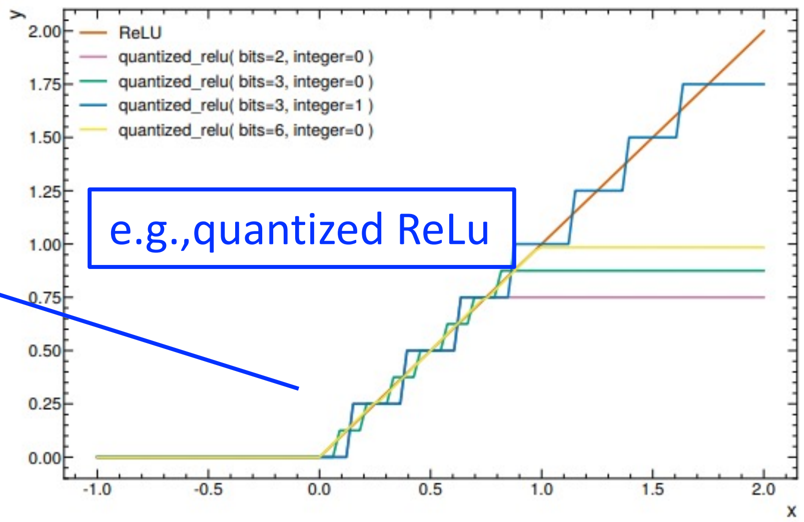
Description of each of the 16 input variables at 1709.08705

Baseline model







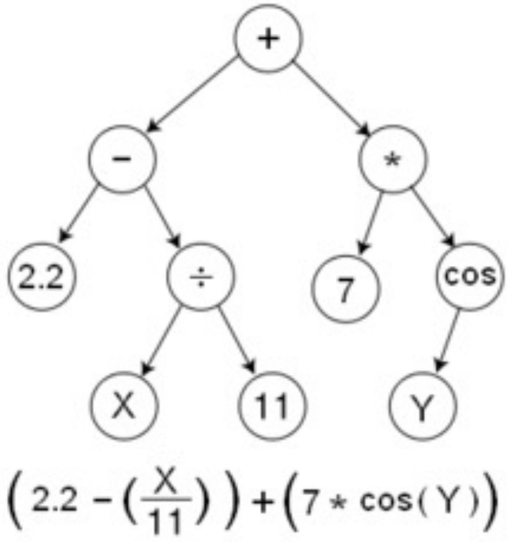


Model parameters in fixed numerical precision and constrained during weight optimization

- Standard baseline architecture for FastML studies chosen to yield reasonable performance while being lightweight (sub-100 ns latency)
- Trained quantization aware with *QKeras* 2006.10159
- Convert to HLS firmware with *hls4ml*
- Serve as baseline for accuracy and FPGA resource utilization to be compared with SR

Plain SR implementation

- Models with single class of math functions
  - Polynomial: +, -, x
  - Trigonometric: +, -, x, sin(·)
  - Exponential: +, -, x, Gauss(·)=exp(-(·)²)
  - Logarithmic: +, -, x, log(abs(·))



- Model size can be quantified by a measure called complexity
- ➢ Default complexity for every operator = 1
- ➢ All operators are equally penalized
- ➢ Search is configured to find expressions having a complexity up to a max value c_{max}

- Reduce input dimensions by random forest regressor (*PySR* built-in functionality). We select 6 out of 16 here

Model	Expression for the t tagger with $c_{\max} = 40$	AUC
Polynomial	$C_1^{\beta=2} + 0.09m_{\text{mMDT}}(2C_1^{\beta=1} + M_2^{\beta=2} - m_{\text{mMDT}} - \text{Multiplicity} - (1.82C_1^{\beta=1} - M_2^{\beta=2})(C_1^{\beta=2} - 0.49m_{\text{mMDT}}) - 3.22) - 0.53$	0.914
Trigonometric	$\sin(0.06(\sum z\log z)M_2^{\beta=2} - 0.25C_1^{\beta=2}(-C_1^{\beta=1} + 2C_1^{\beta=2} - M_2^{\beta=2} + \text{Multiplicity} - 8.86) - m_{\text{mMDT}} + 0.06\text{Multiplicity} - 0.4)$	0.925
Exponential	$0.23C_1^{\beta=1}(-m_{\text{mMDT}} + \text{Gauss}(0.63\text{Multiplicity}) + 1) - \text{Gauss}(C_1^{\beta=1}) + 0.45C_1^{\beta=2} - 0.23m_{\text{mMDT}}$ $+0.23\text{Gauss}((4.24 - 1.19C_2^{\beta=1})(C_1^{\beta=2} - m_{\text{mMDT}})) + 0.15$	0.920
Logarithmic	$C_1^{\beta=2} - 0.1m_{\text{mMDT}}(\text{Multiplicity} \times \log(\text{abs}(\text{Multiplicity})) + 2.2) - 0.02\log(\text{abs}(\text{Multiplicity}))$ $-0.1(C_1^{\beta=2}(C_1^{\beta=1} - 1.6M_2^{\beta=2} + m_{\text{mMDT}} + 1.28) - m_{\text{mMDT}} - 0.48)\log(\text{abs}(C_1^{\beta=2})) - 0.42$	0.923

Table 2. Expressions generated by PySR for the t tagger in different models with $c_{\max} = 40$. Operator complexity is set to 1 by default. Constants are rounded to two decimal places here.

Math function approximation with lookup table

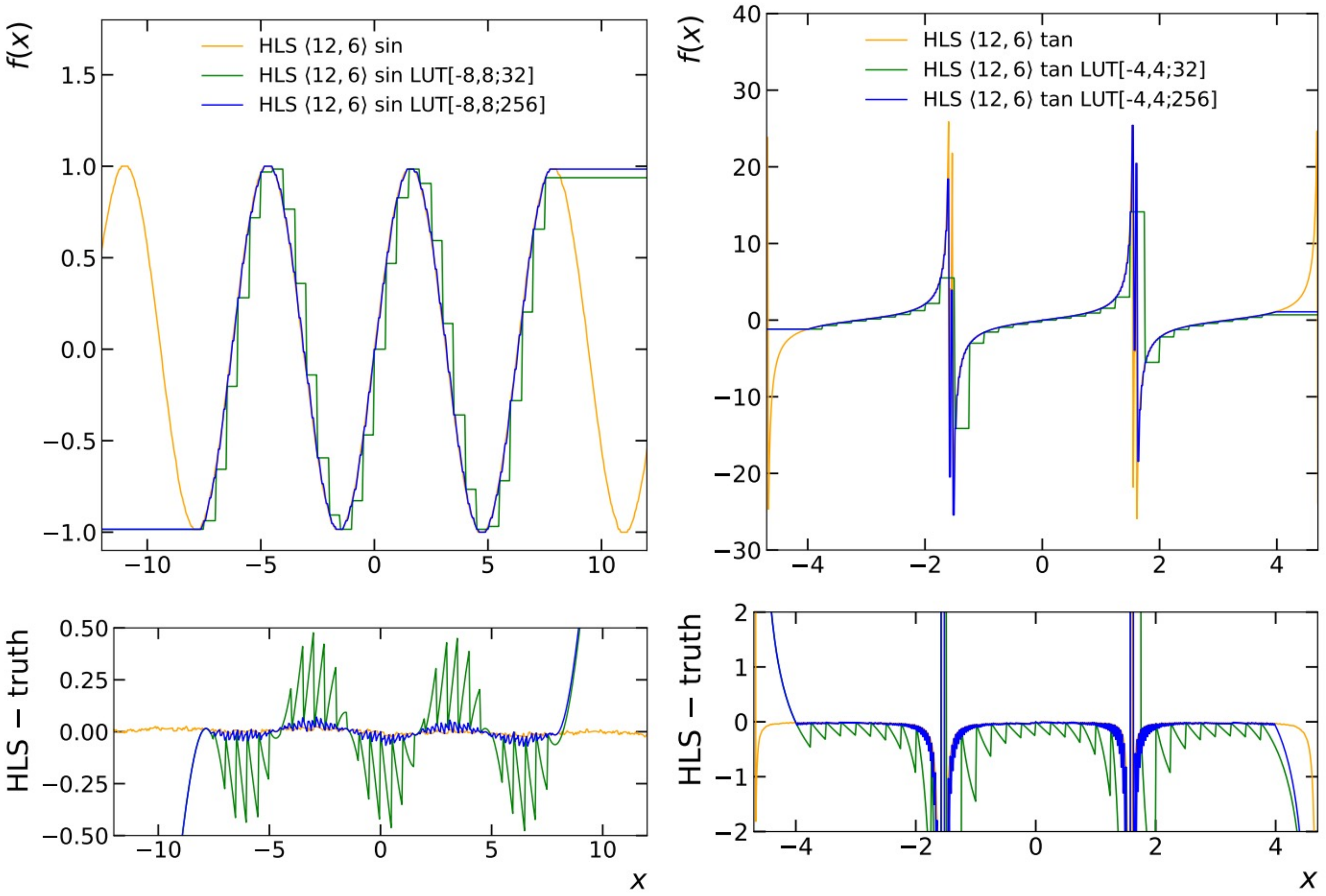


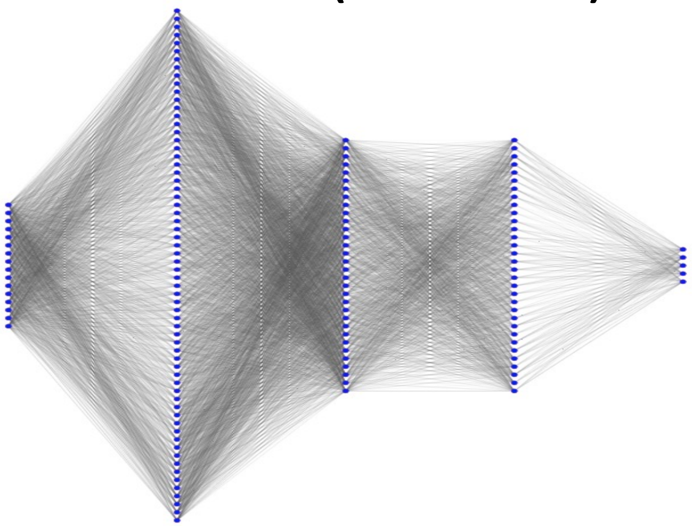
Figure 1. The sine (left) and tangent (right) functions evaluated with and without the use of LUTs, implemented in HLS with precision ⟨12, 6⟩, i.e. 12 bits variable with 6 integer bits. The LUT notation reads: [range start, range end; table size] for table definition. The lower panel shows the function deviation from the truth.

- Use an array that maps input to output, thus runtime computation is replaced by array indexing operation
- Custom table range and size
  - No DSPs allocated if both are 2^n
- Every LUT operation requires only 1 clock cycle

Benchmark

To compare

Baseline (QAT NN)



vs.

SR (5-line expressions)

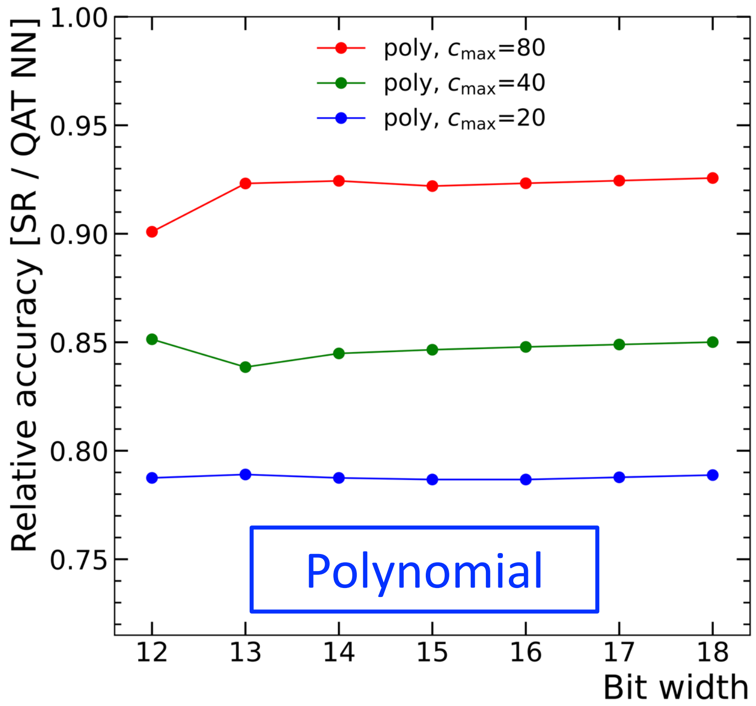
Tagger	Expression for the trigonometric model with $c_{\max} = 20$	AUC
g	$\sin(-2C_1^{\beta=1} + 0.31C_1^{\beta=2} + m_{\mathrm{mMDT}} + \mathrm{Multiplicity} - 0.09\mathrm{Multiplicity}^2 - 0.79)$	0.897
q	$-0.33(\sin(m_{\mathrm{mMDT}}) - 1.54)(\sin(-C_1^{\beta=1} + C_1^{\beta=2} + \mathrm{Multiplicity}) - 0.81)\sin(m_{\mathrm{mMDT}}) - 0.81$	0.853
t	$\sin(C_1^{\beta=1} + C_1^{\beta=2} - m_{\mathrm{mMDT}} + 0.22(C_1^{\beta=2} - 0.29)(-C_1^{\beta=2} + C_2^{\beta=1} - \mathrm{Multiplicity}) - 0.68)$	0.920
W	$-0.31(\mathrm{Multiplicity} + (2.09 - \mathrm{Multiplicity})\sin(8.02C_1^{\beta=2} + 0.98)) - 0.5$	0.877
Z	$(\sin(4.84m_{\mathrm{mMDT}}) + 0.59)\sin(m_{\mathrm{mMDT}} + 1.14)\sin(C_1^{\beta=2} + 4.84m_{\mathrm{mMDT}}) - 0.94$	0.866

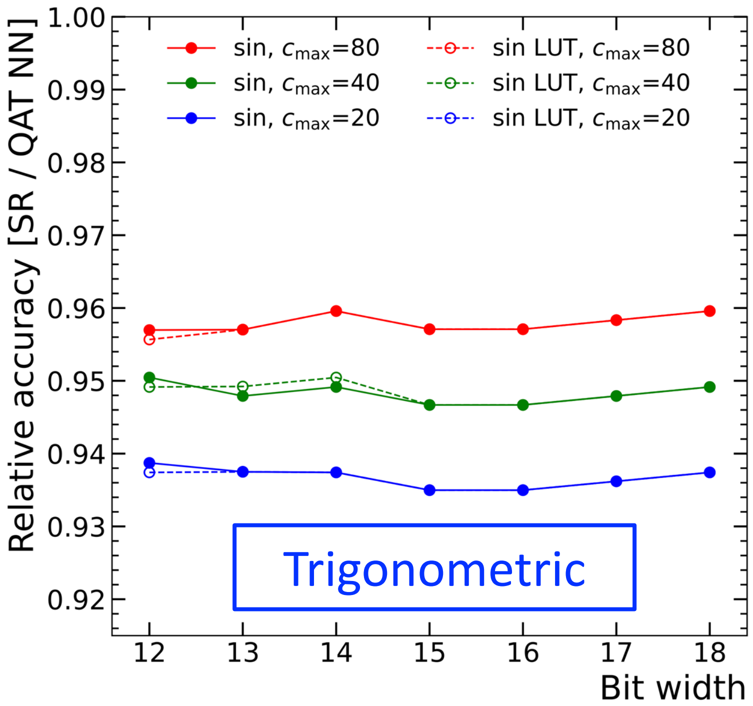
Table 1. Expressions generated by PySR for the trigonometric model with $c_{\max} = 20$. Operator complexity is set to 1 by default. Constants are rounded to two decimal places for readability. Area under the receiver operating characteristic (ROC) curve, or AUC, is reported.

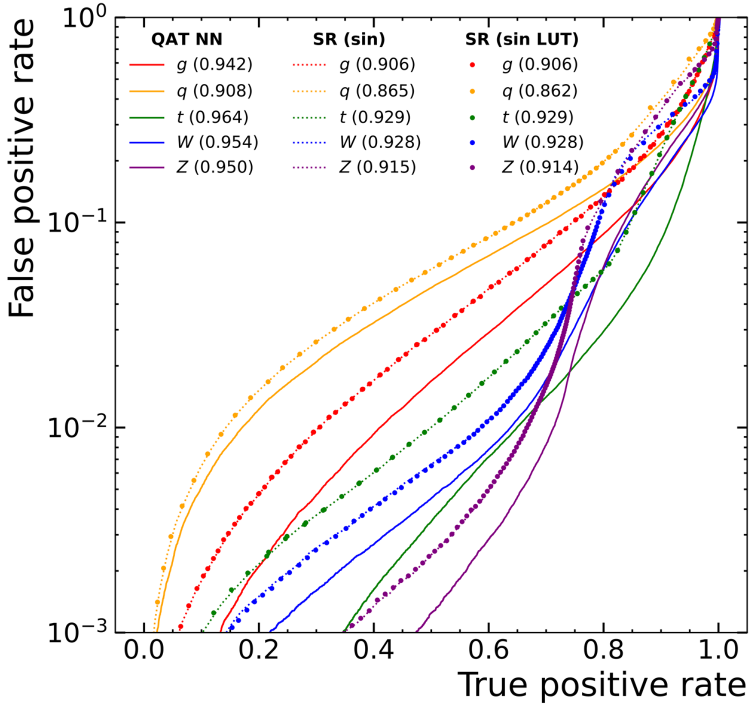
On

- Classification accuracy
- FPGA resource utilization
  - DSPs
  - LUTs
  - Inference latency

Accuracy

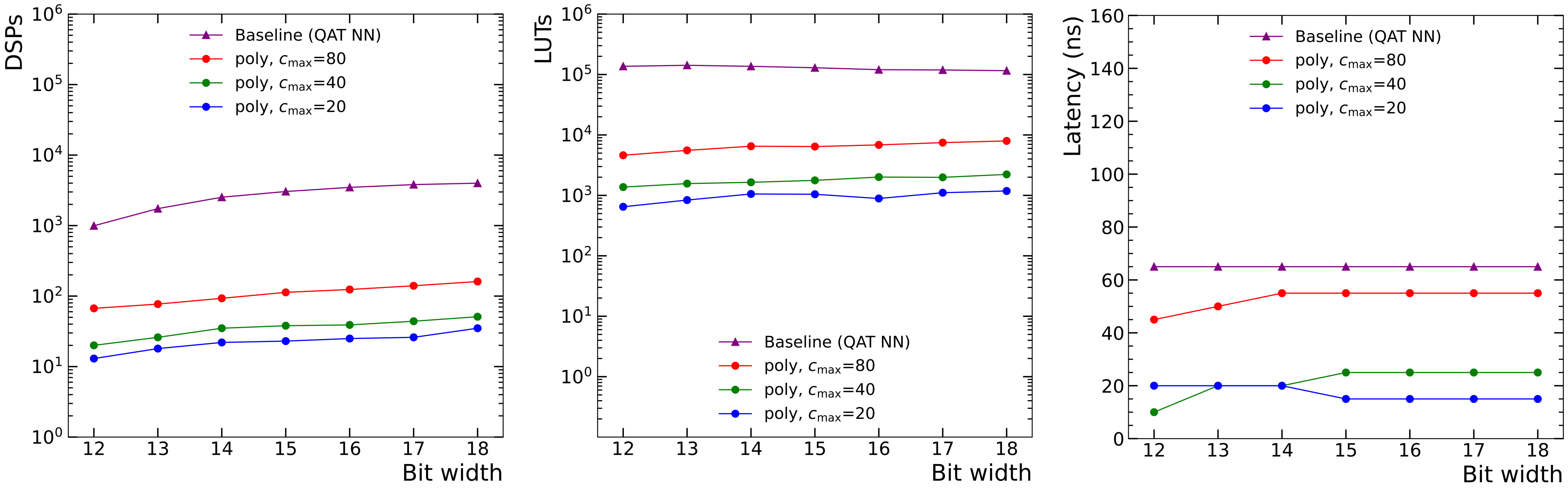






- Trigonometric equations, and others, perform very close to NN
- Models with single math class at relatively low complexity can give comparable accuracy
- *S*ensitive to function choice, e.g. trigonometric vs. polynomial
- Approximation with lookup tables does not downgrade performance

Resource utilization



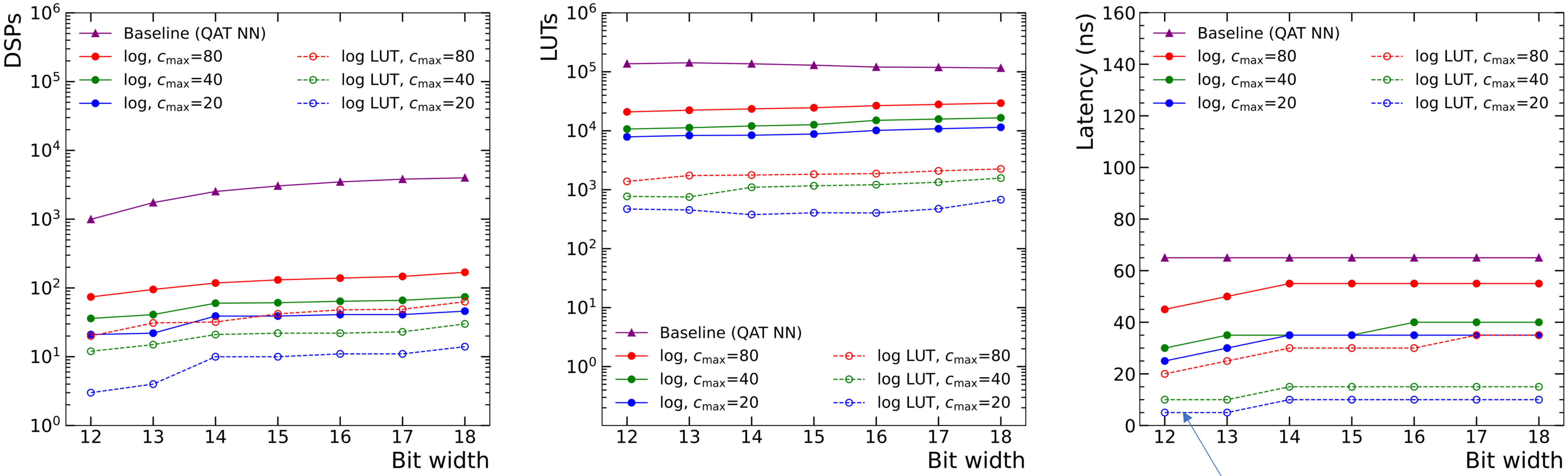
Polynomial

II=1

SR models dramatically reduce latency and resources compared to NN

- Several orders of magnitude improvement in resource usage
- Several times faster

Resource utilization



Logarithmic

II=1

SR models dramatically reduce latency and resources compared to NN

5 ns!

- Several orders of magnitude improvement in resource usage
- Several times faster

Latency-aware training

- By default, *PySR* assigns every operator complexity to 1, this means all operators are equally penalized when being added
- Not optimal for FPGA deployment since there is difference in number of clock cycles (cc) required
- We can re-define operator complexity to the corresponding no. of cc
- One can also specify a latency budget
- Note that function approximation with lookup tables is not relevant here since every array indexing operation needs 1 cc only

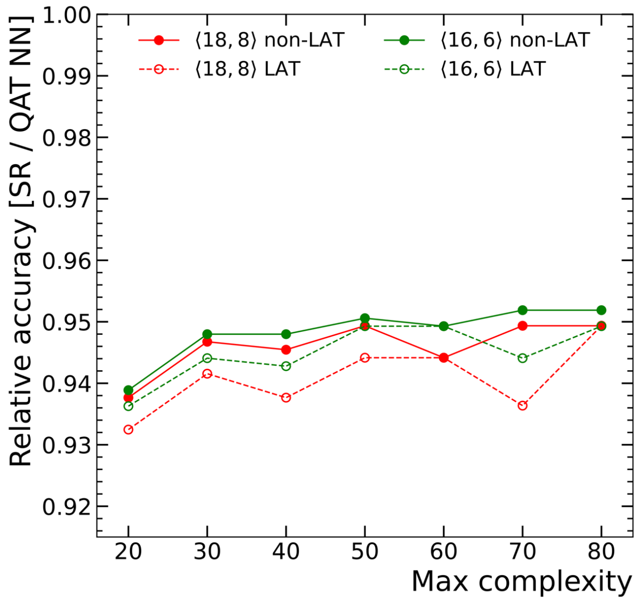
Operator	No. of cc
$+$	1
$-$	1
$\times$	1
log(abs(·))	4
sin(·)	8
tan(·)	48
cosh(·)	8
sinh(·)	9
exp(·)	3

Evaluated with <16,6> on a Xilinx VU9P FPGA (xccu9p-flga2577-2-e)

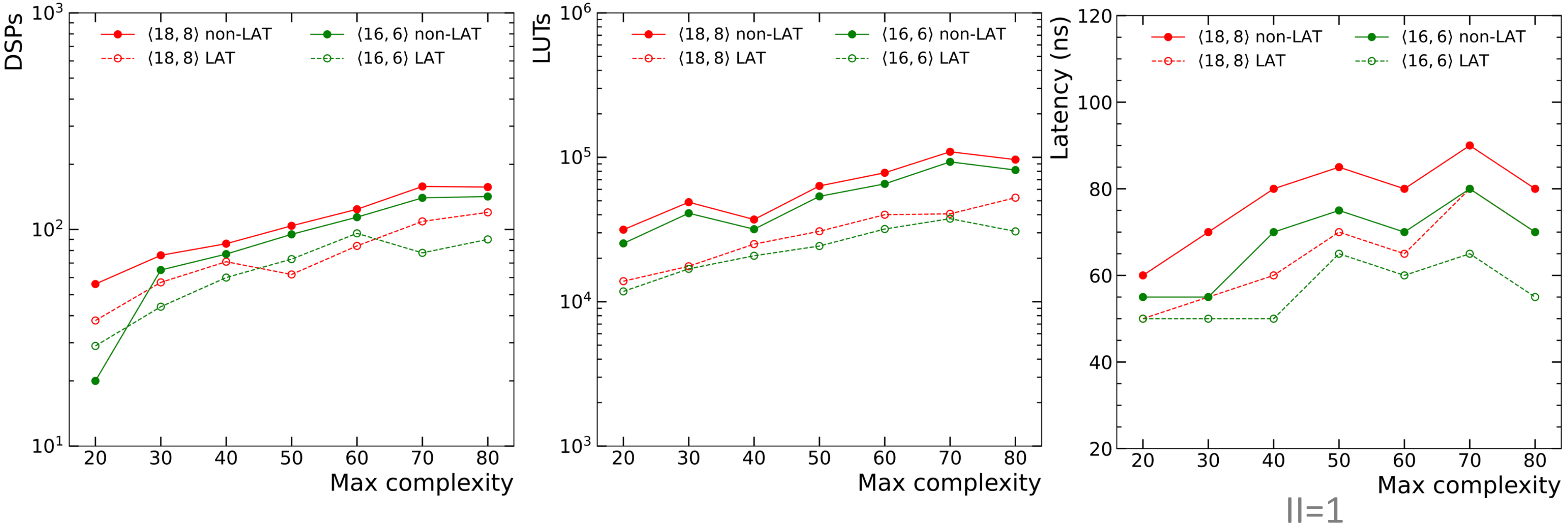
Latency-aware training

Operator complexity	Expression for the t tagger with $c_{\max} = 40$	AUC
All 1's (PySR default)	$0.11(C_1^{\beta=1} + C_1^{\beta=2} + \log(\mathrm{abs}(C_1^{\beta=2}))) - 0.48m_{\mathrm{mMDT}} - 0.05\mathrm{Multiplicity}(\mathrm{Multiplicity} + \log(\mathrm{abs}(m_{\mathrm{mMDT}})))$ $-\sin(-C_1^{\beta=2} + 0.14C_2^{\beta=1}m_{\mathrm{mMDT}}) + 0.11\sinh(C_1^{\beta=1}) - 0.24$	0.930
No. of clock cycles at $\langle 16, 6\rangle$	$0.04((\sum z\log z) + C_1^{\beta=1} + C_2^{\beta=1} - m_{\mathrm{mMDT}} - (\mathrm{Multiplicity} - 0.2)(\mathrm{Multiplicity} + \log(\mathrm{abs}(C_1^{\beta=2}))))$ $-\sin(-C_1^{\beta=1} - C_1^{\beta=2} + 1.23m_{\mathrm{mMDT}} + 0.58)$	0.924
No. of clock cycles at $\langle 18, 8\rangle$	$0.04\mathrm{Multiplicity}(C_1^{\beta=2}(C_1^{\beta=2} - m_{\mathrm{mMDT}}) - \mathrm{Multiplicity} - \log(\mathrm{abs}(C_1^{\beta=2}((\sum z\log z) + 0.23))))$ $-\sin(-C_1^{\beta=1} - C_1^{\beta=2} + 1.19m_{\mathrm{mMDT}} + 0.61)$	0.926

Table 3. Expressions generated by PySR for the t tagger with $c_{\max} = 40$, implemented with and without LAT. Constants are rounded to two decimal places for readability.



Accuracy



Resources and latency

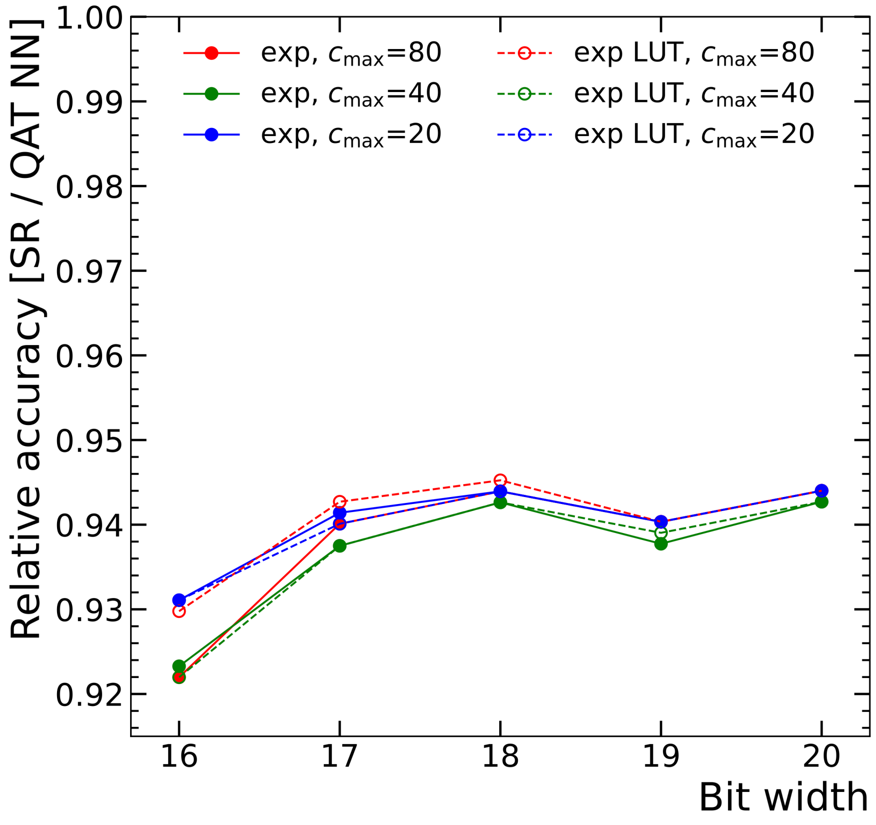
Summary

- We presented a novel approach of SR utilization in the context of FPGAs
  - ✓ Integrated SR into *hls4ml*
  - ✓ Proposed 3 implementation strategies
  - ✓ Demonstrated SR can achieve comparable accuracy while using significantly less resources (by orders of magnitude) and inferring faster (by few multiples), as compared to NN-based model

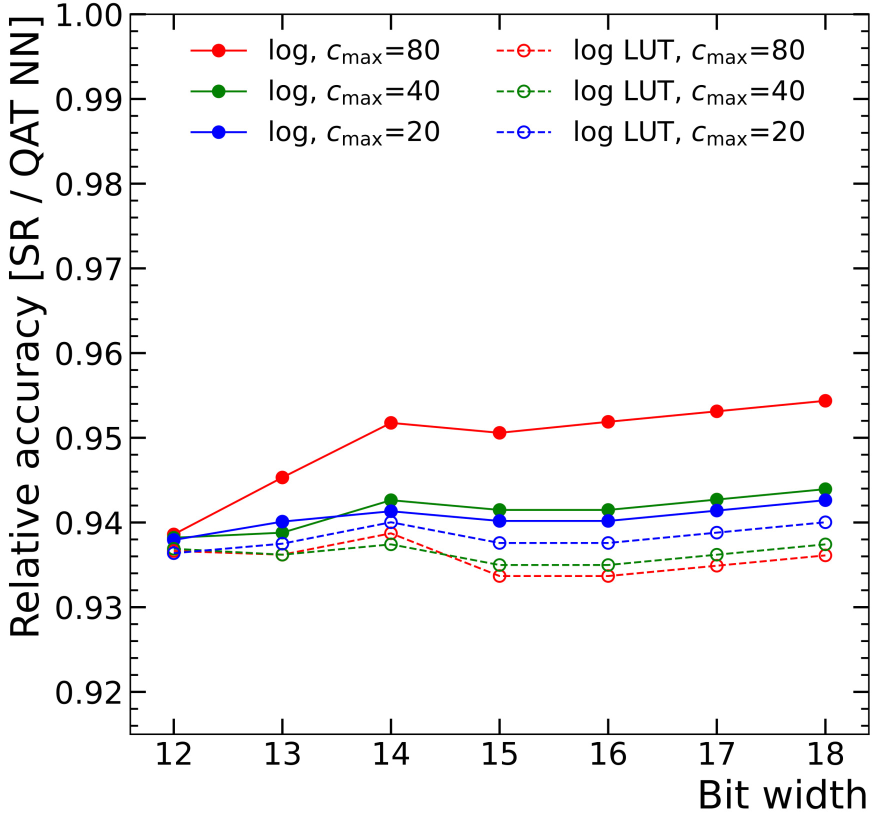
- Future works (naming only a few)
  - ❑ NN-based SR to enable quantization-aware training: start from a (sparse) NN with math operations as activations, trained with *QKeras*, then prune to yield final expressions
  - ❑ Use SR as distillation
    - Distill intermediate layers of big models
    - Regress outputs directly
  - ❑ Investigate SR in problems with high input dimensions
    - Feature engineering
    - Break it into lower dimensions, feed to a hierarchy of localized NNs, then do SR

Backup

Accuracy

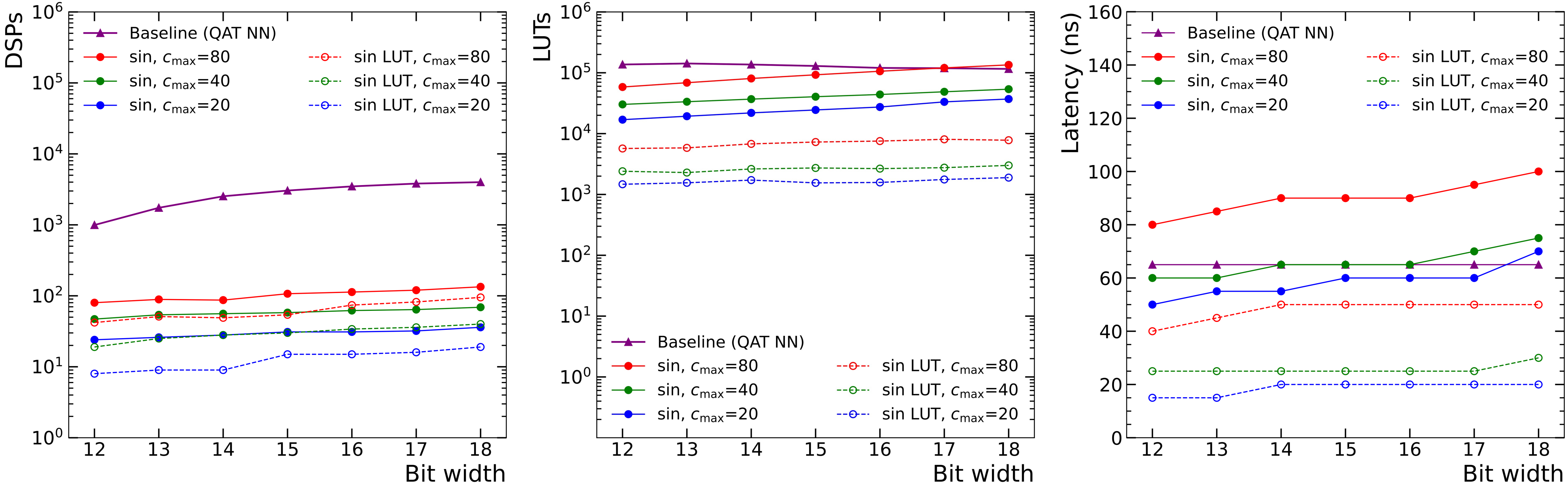


Exponential



Logarithmic

Resource utilization



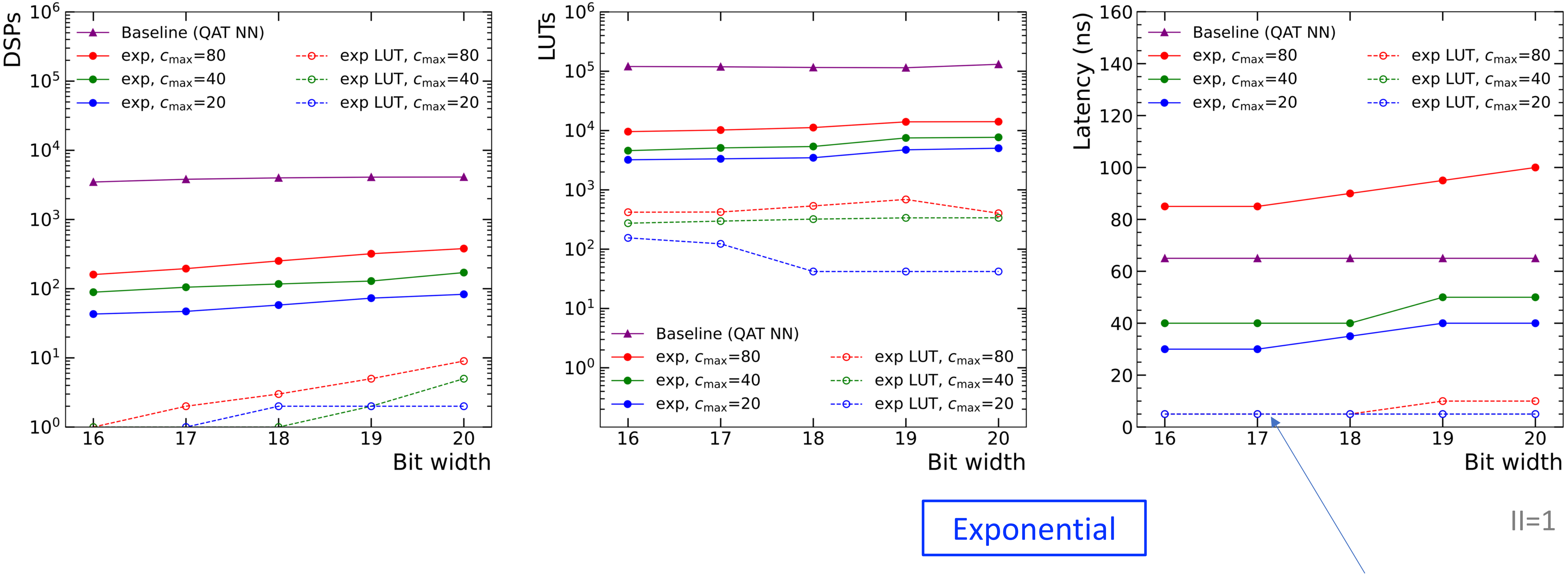
Trigonometric

II=1

SR models dramatically reduce latency and resources compared to NN

- Several orders of magnitude improvement in resource usage
- Several times faster

Resource utilization



SR models dramatically reduce latency and resources compared to NN

- Several orders of magnitude improvement in resource usage
- Several times faster