Logic Neural Networks: Ultra-Fast ML for Triggers and Beyond

RAL seminar 2026

28.01.26

This presentation contains animations - best viewed in .pptx or .key

Lino Gerlach, Thore Gerlach, Elliott Kauffman, Liv Våge

Who I am

Teaching ML since 2017

PhD at Imperial College

New directions for particle tracking at the High-Luminosity LHC

GNNs and reinforcement learning for tracking

Working with IRIS-HEP, based in Geneva

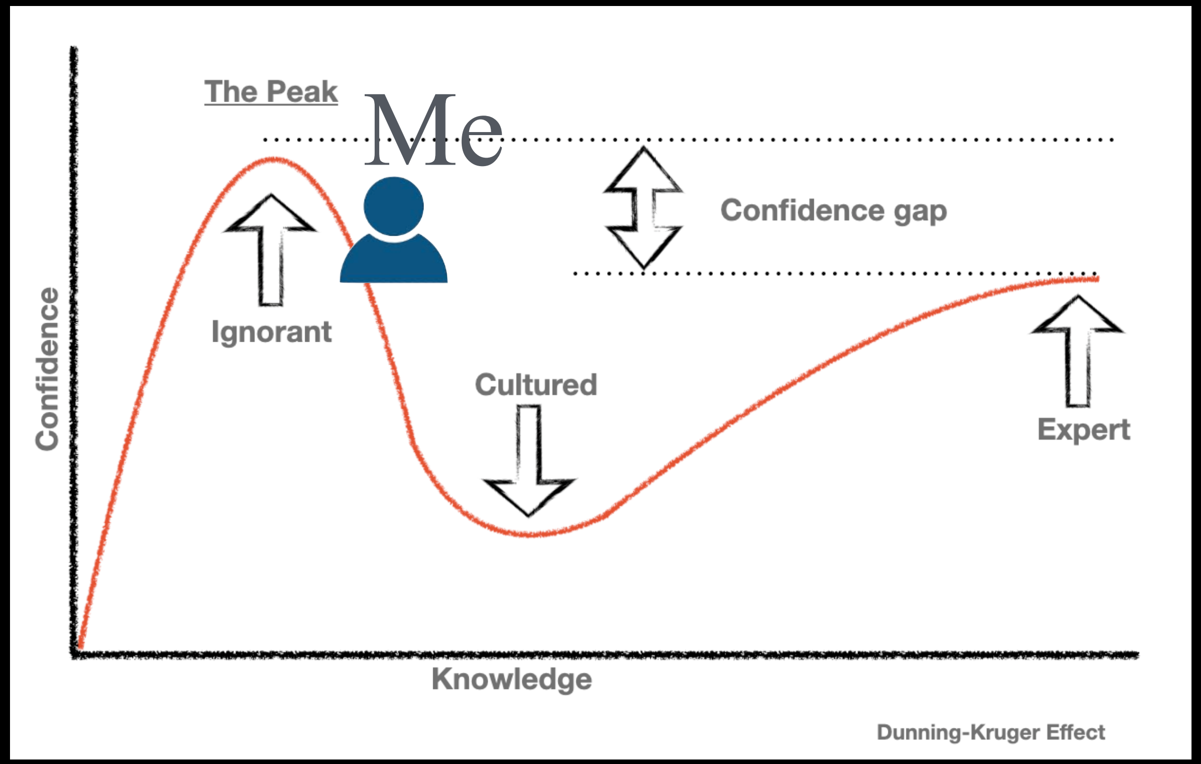
Innovative algorithms

GNNs for tracking
Rapid inference

ML infrastructures in HEP

Who I am not

An FPGA expert



Introduction

Low latency inference - evaluating ML fast

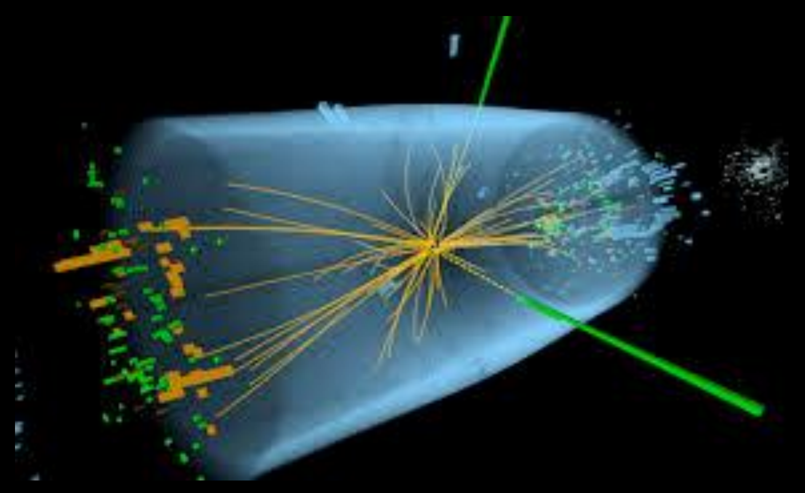
When we want ML that is:

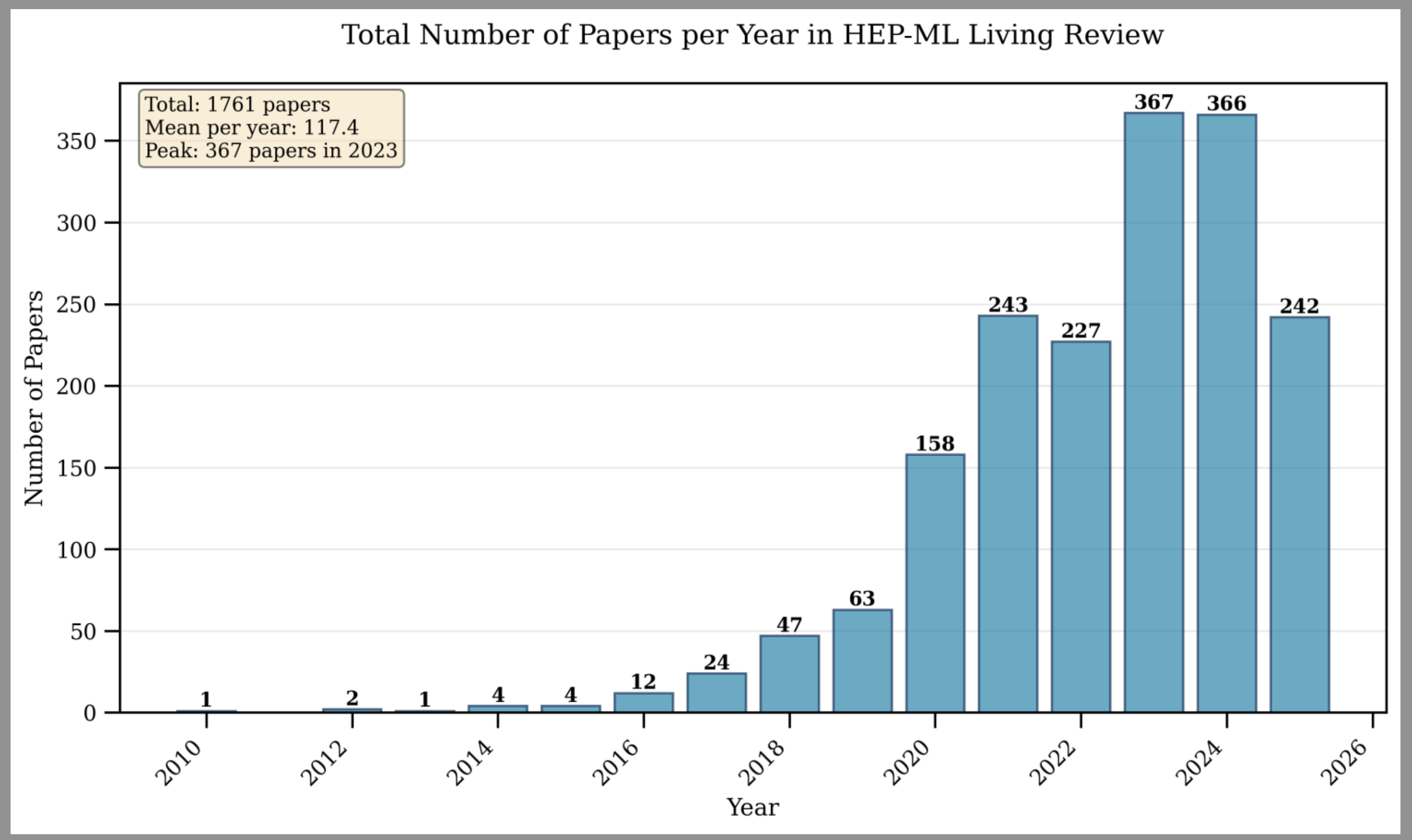
Close to host
Edge computing is increasing:
now at ~ 21 billion IoT devices

Cheap and efficient to run
LLMs, medical devices, space devices etc.
Pacemakers need ~ 50 microwatts to run for 7 years

Super fast
Financial trading (Spread networks e.g. spent 300 M on a cable to save 13 ms), self driving cars and lots more

HEP





* This plot was made around October 2025

ML use
ML use is increasing in HEP. We have huge and increasing amounts of data

Triggers
The Phase 2 CMS Level 1 Trigger will for instance need to process 63 Tb/s with a latency of 12.5 μs

How much of the energy consumption and cost does inference account for in an ML pipeline?

10-30%	0%
30-50%	0%
50-70%	0%
70-90%	0%

Inference devices

Lower latency and power, harder to program →

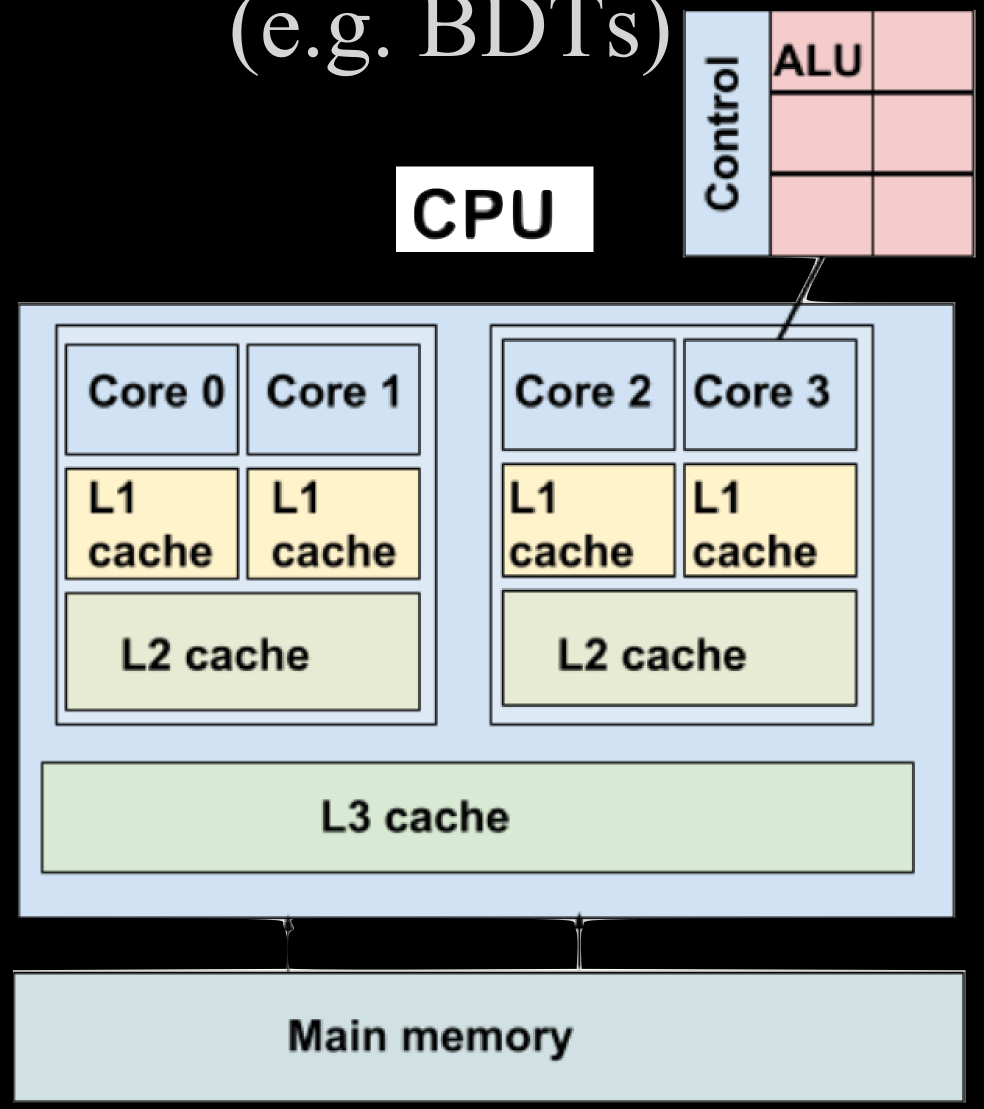
With high luminosity LHC and FCC, we are moving towards the right to handle our data quickly

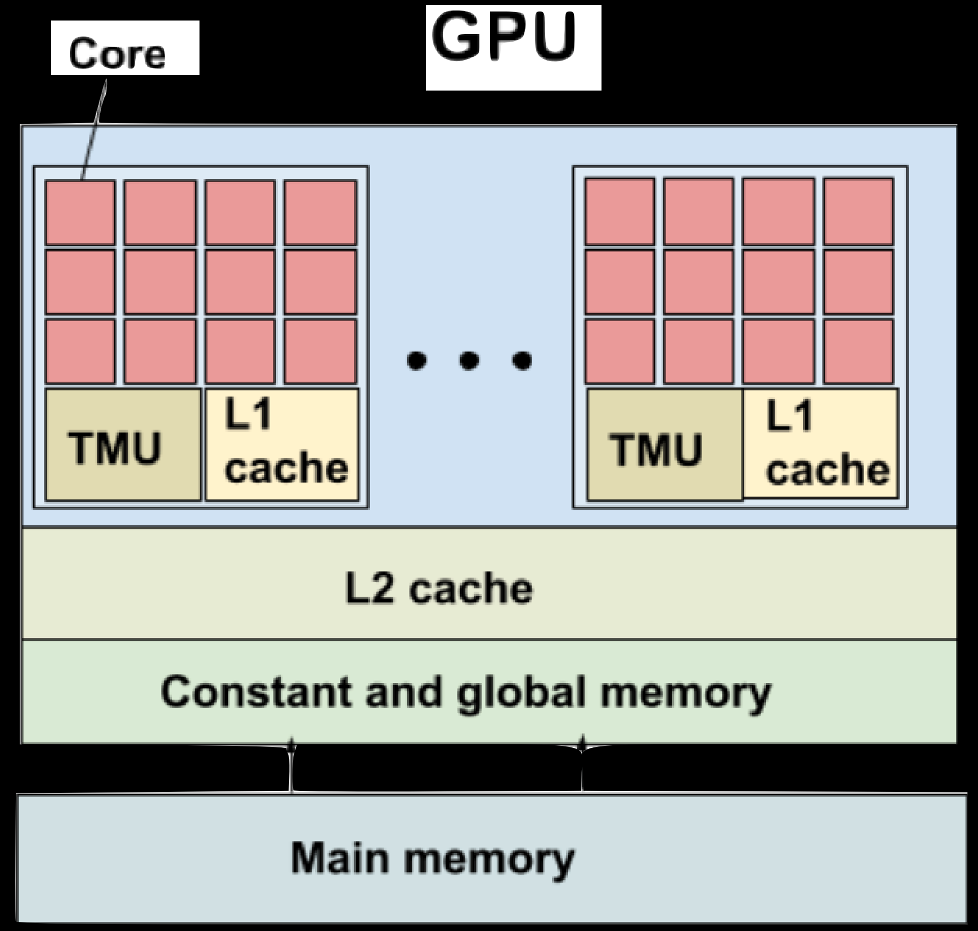
	CPU	GPU	TPU/NPU	FPGA	ASIC
Latency	ms-s	µs–ms	µs–ms	ns-µs	ns
Strength	Flexible easy to program	Parallelism, ML optimised	ML-specific	Deterministic Low power	Ultimate speed/power efficiency
Weakness	Slow, high power	High power	Bespoke	Hard to program	Expensive Not flexible

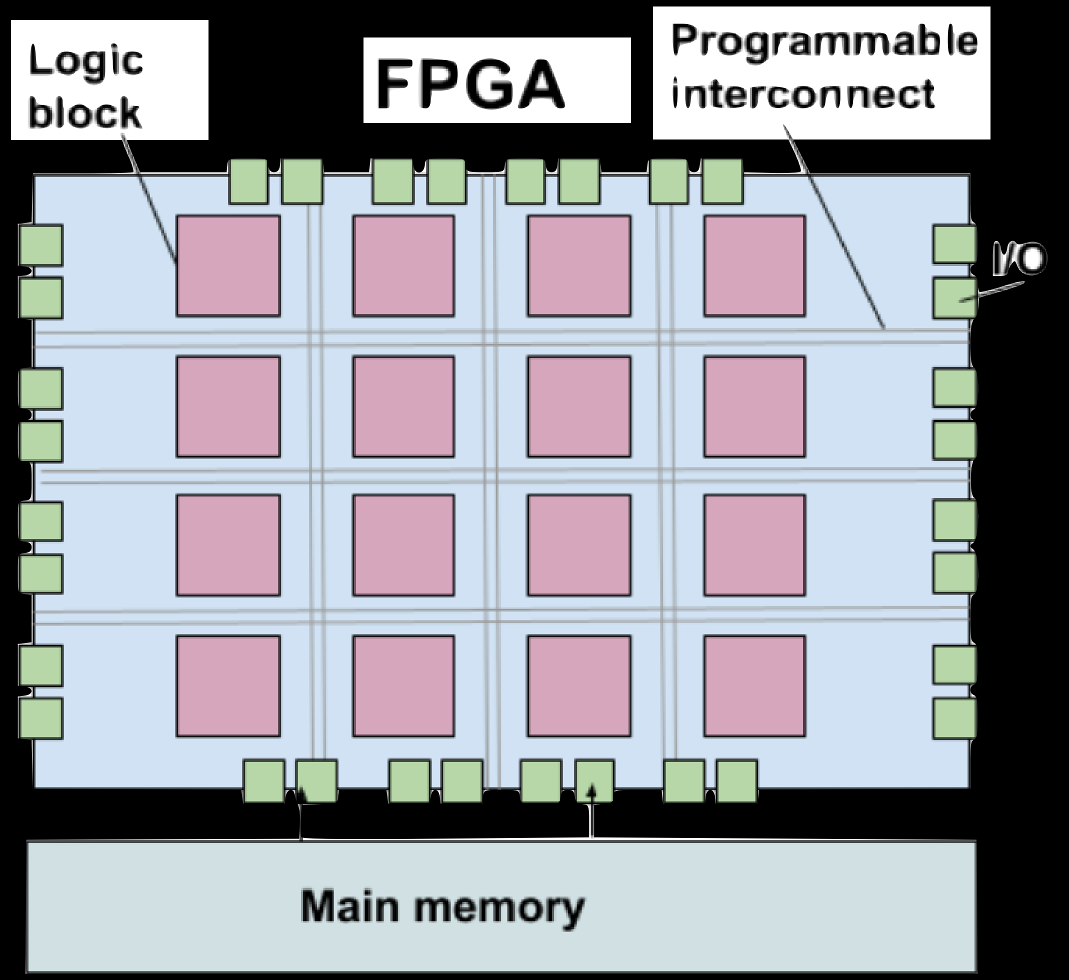
Small ML models (e.g. BDTs)

Most ML

Low inference ML

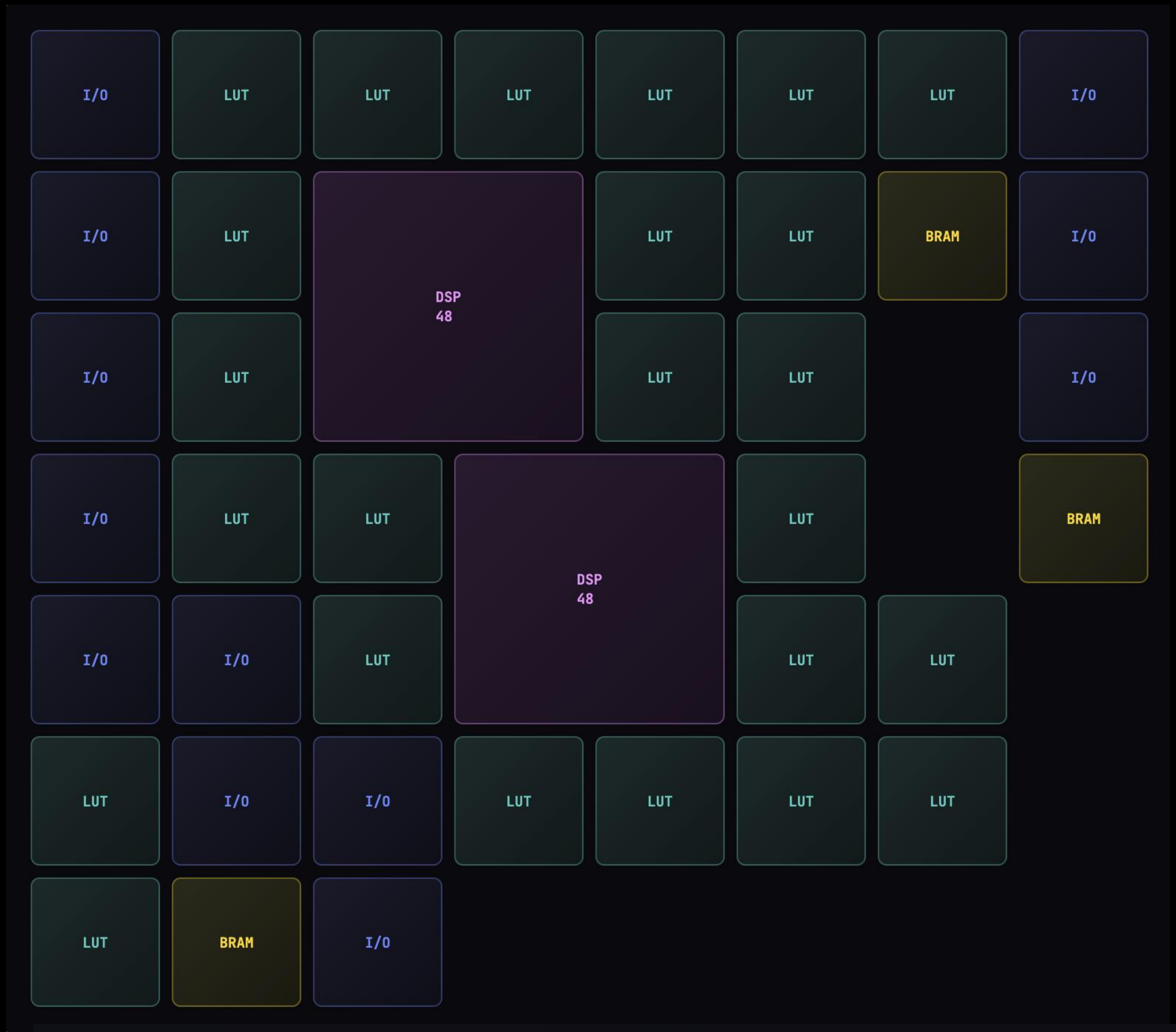






No matter what chip we use or how we program it - it is ultimately translated to binary logic

Field Programmable Gate Arrays (FPGAs)



🟩 **LUT** — Lookup Tables (thousands!) 🟪 **DSP48** — Digital Signal Processors 🟨 **BRAM** — Block RAM 🟦 **I/O** — Input/Output Blocks

Hardware you can reprogram after manufacturing

Massively parallel → very fast for specific tasks

Deterministic timing (no caches, no surprises)

Programmed using Verilog / VHDL

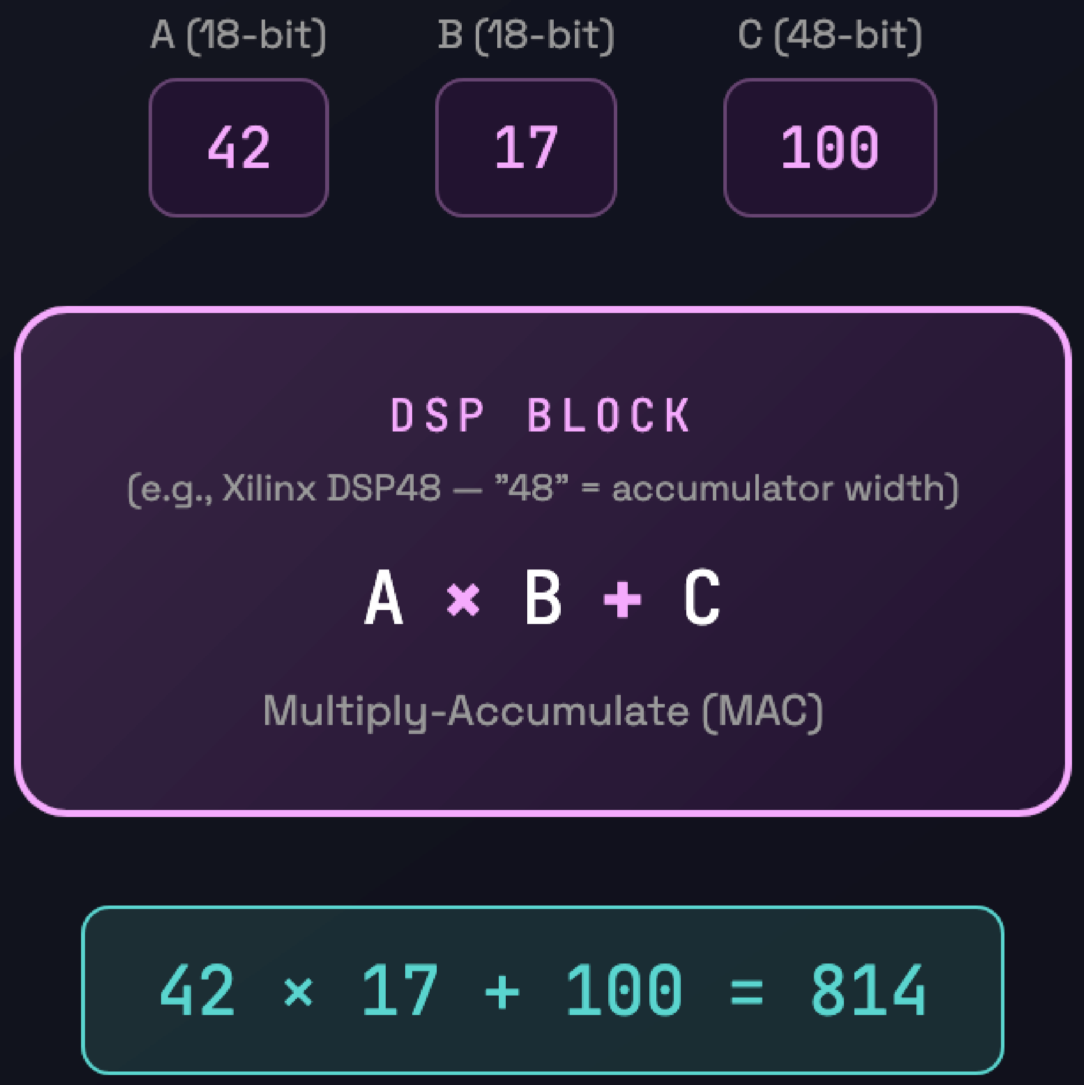
Can also use high-level tools (e.g. HLS, hls4ml)

Available in radiation-tolerant versions for space or HEP detectors

DSPs and LUTs

DSP — Digital Signal Processor

Hardwired arithmetic unit for fast math



⚡ Speed
1 cycle throughput (pipelined), ~3-4 cycle latency

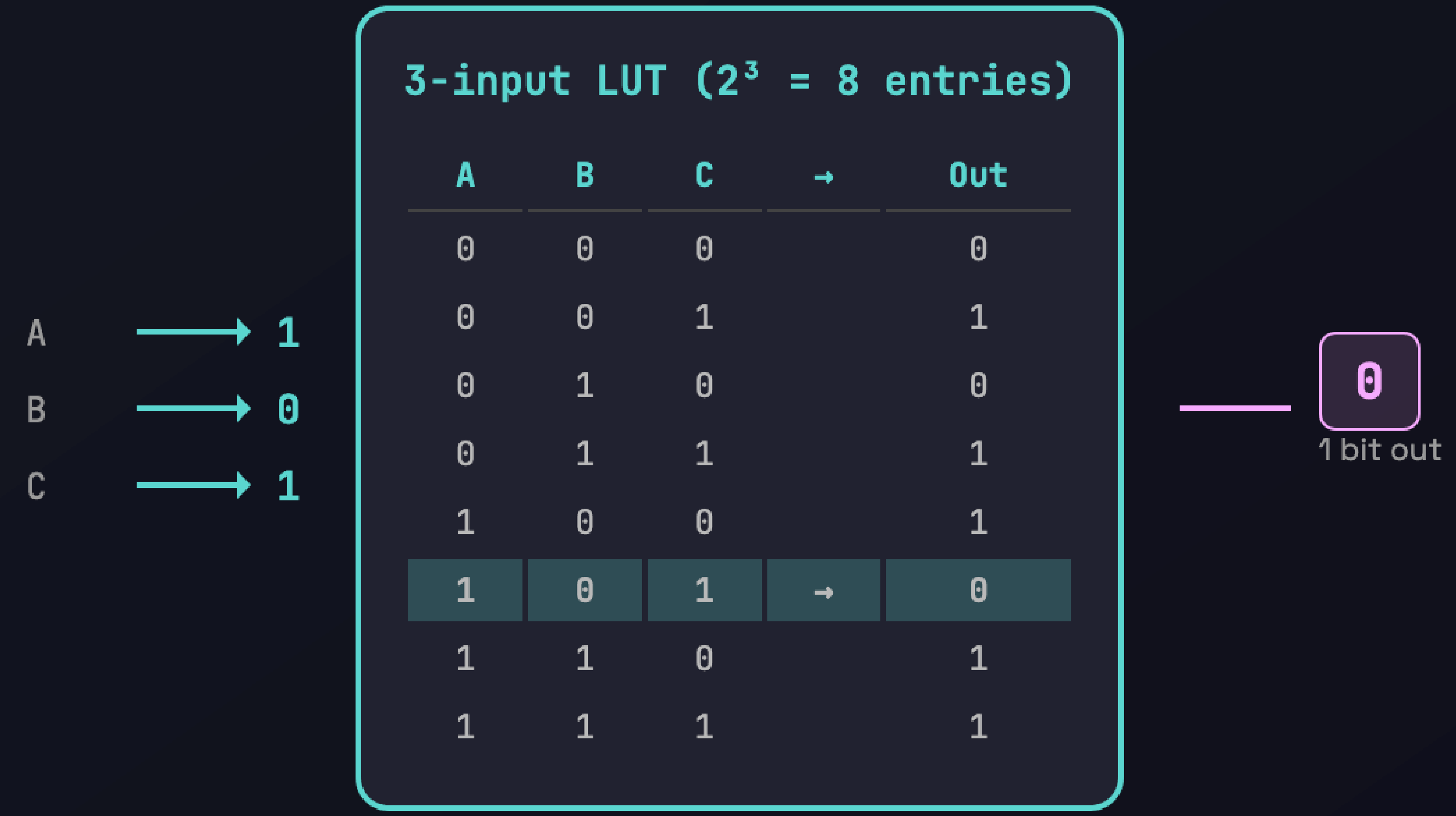
🎯 Purpose
General arithmetic: multiply, add, accumulate, shifts, pattern detect

📦 Capacity
18×18 bit multiply → 48-bit result

🔢 Key constraint
Limited count (~2-6k per FPGA) — they're big!

LUT — Lookup Table

A tiny programmable truth table that outputs 1 bit



3-input LUT (2^3 = 8 entries)

A	B	C	→	Out
0	0	0		0
0	0	1		1
0	1	0		0
0	1	1		1
1	0	0		1
1	0	1	→	0
1	1	0		1
1	1	1		1

⚡ Speed
Single clock cycle — just a memory read

🎯 Flexibility
Can implement ANY boolean function of N inputs

📦 Size
Modern FPGAs have 6-LUTs (64 entries each)

🔢 Key constraint
Each LUT outputs only 1 bit — need multiple for wider outputs

`Computing: y = (3 × x) + 1 where x is a 4-bit input (0-15)`

DSP Using a DSP Block

1. Configure DSP
   `A = 3, C = 1 (constants)`
2. Feed input each cycle
   `B = x (your 4-bit input)`
3. DSP computes
   `y = A × B + C = 3x + 1`

RESOURCES	LATENCY
1 DSP (precious!)	**1 cycle*** *or 3-4 if pipelined

LUT Using LUTs

1. Pre-compute ALL 16 answers
   `x=0→1, x=1→4, x=2→7, ... x=15→46`
2. Store each output BIT in a separate LUT
   `Output range 1-46 needs 6 bits`
3. Look up using x as address
   $y = \{LUT_5[x], LUT_4[x], \ldots, LUT_0[x]\}$

6 LUTs working in parallel (one per output bit)

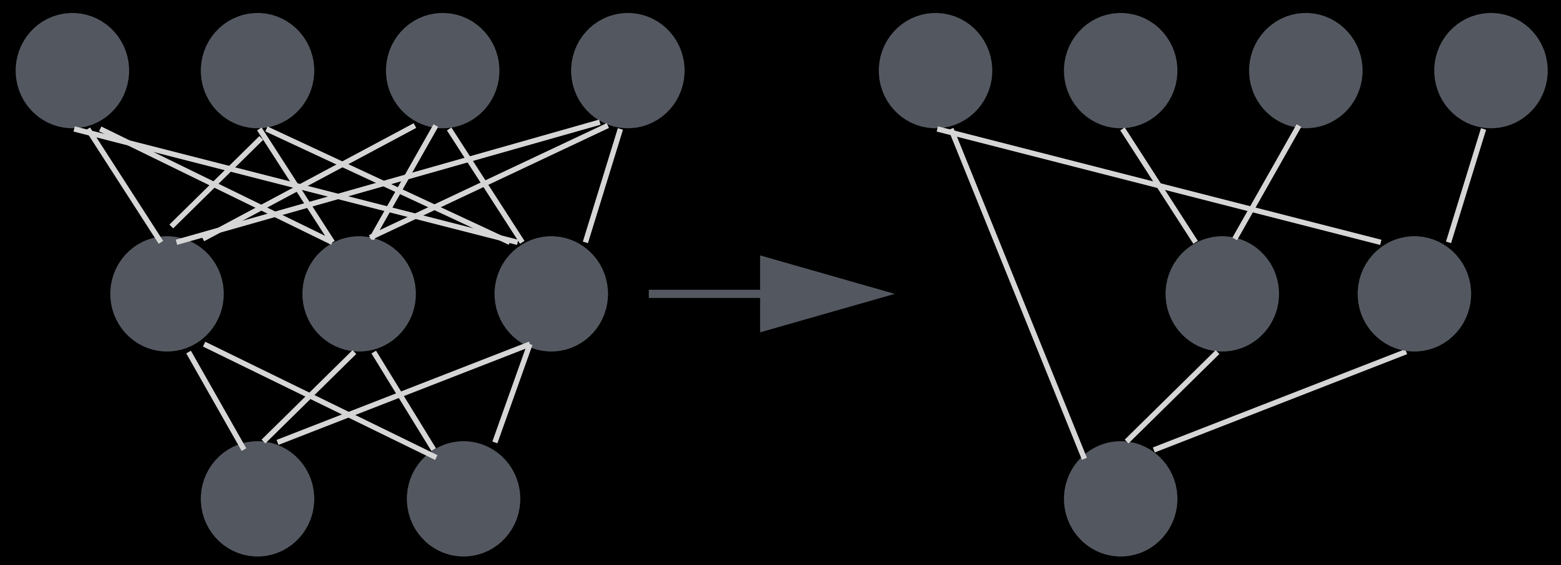
bit 5
bit 4
bit 3
bit 2
bit 1
bit 0
LUT
LUT
LUT
LUT
LUT
LUT

RESOURCES	LATENCY
6 LUTs (abundant!)	**1 cycle** (always)

Current fast inference methods

Pruning

Remove unnecessary weights & neurons

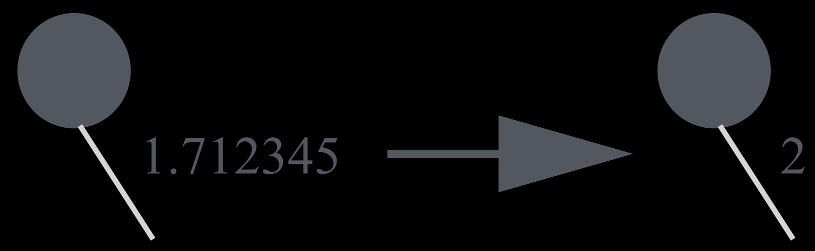


Can often reduce the number of parameters by 3-10x

Often you get better performing retraining the pruned network

Quantisation

Reduce numerical precision

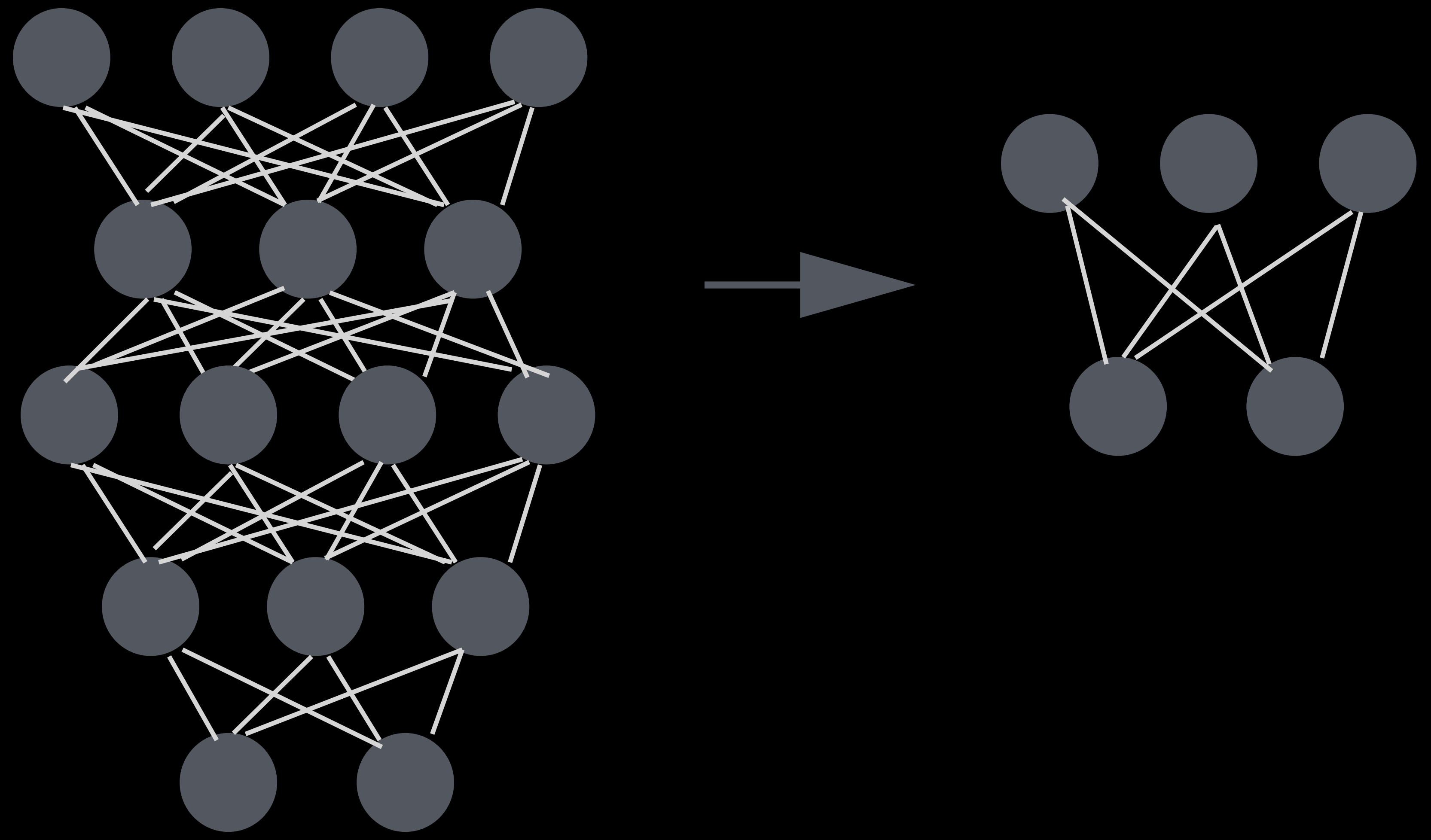


Makes it better for FPGA

Accuracies suffer below int4

Knowledge distillation

Small model learns behaviour of a large one



Used for anomaly detection in CMS

You have to start with a large model and the tuning can be hard

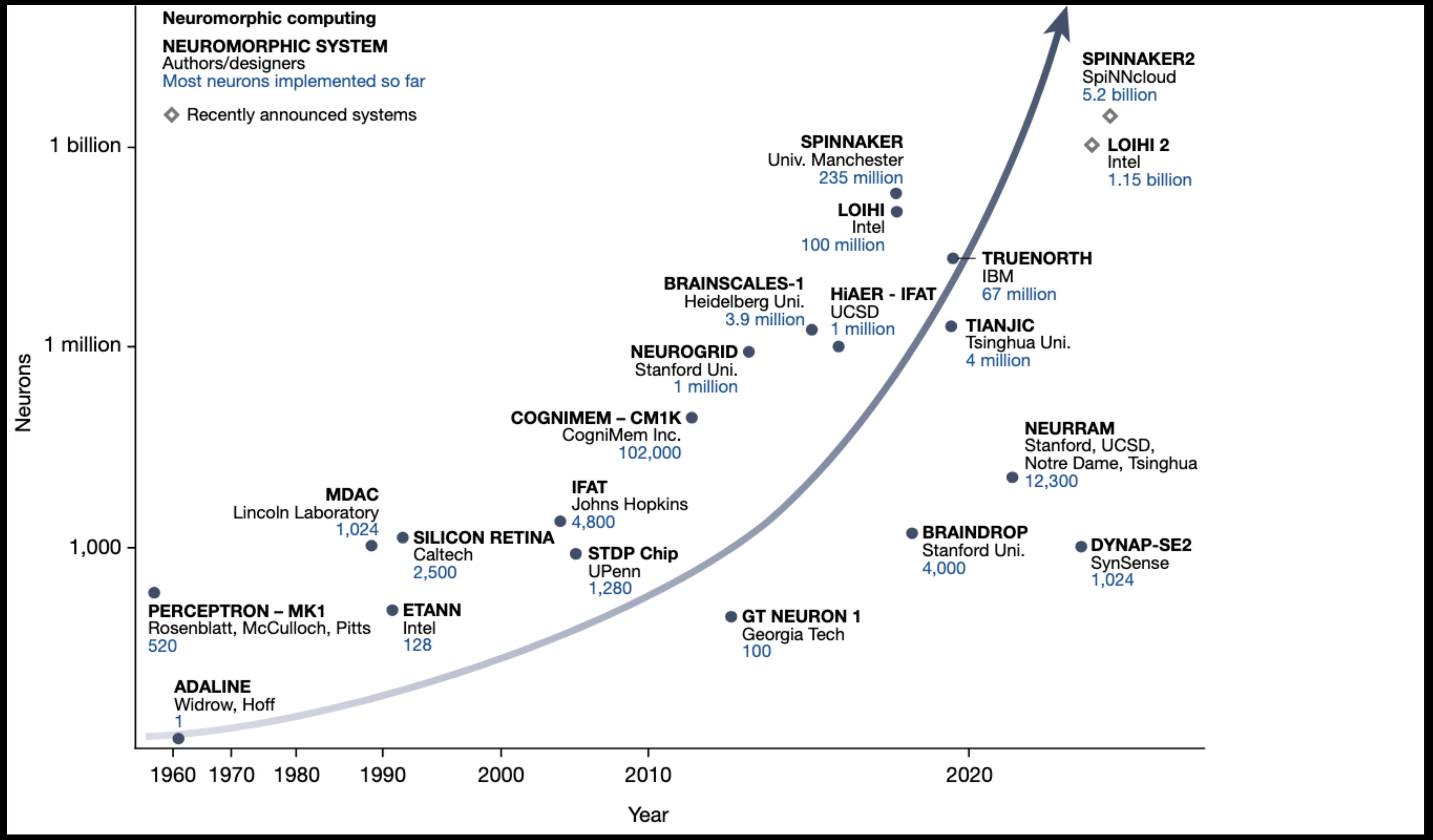
Scaling wall

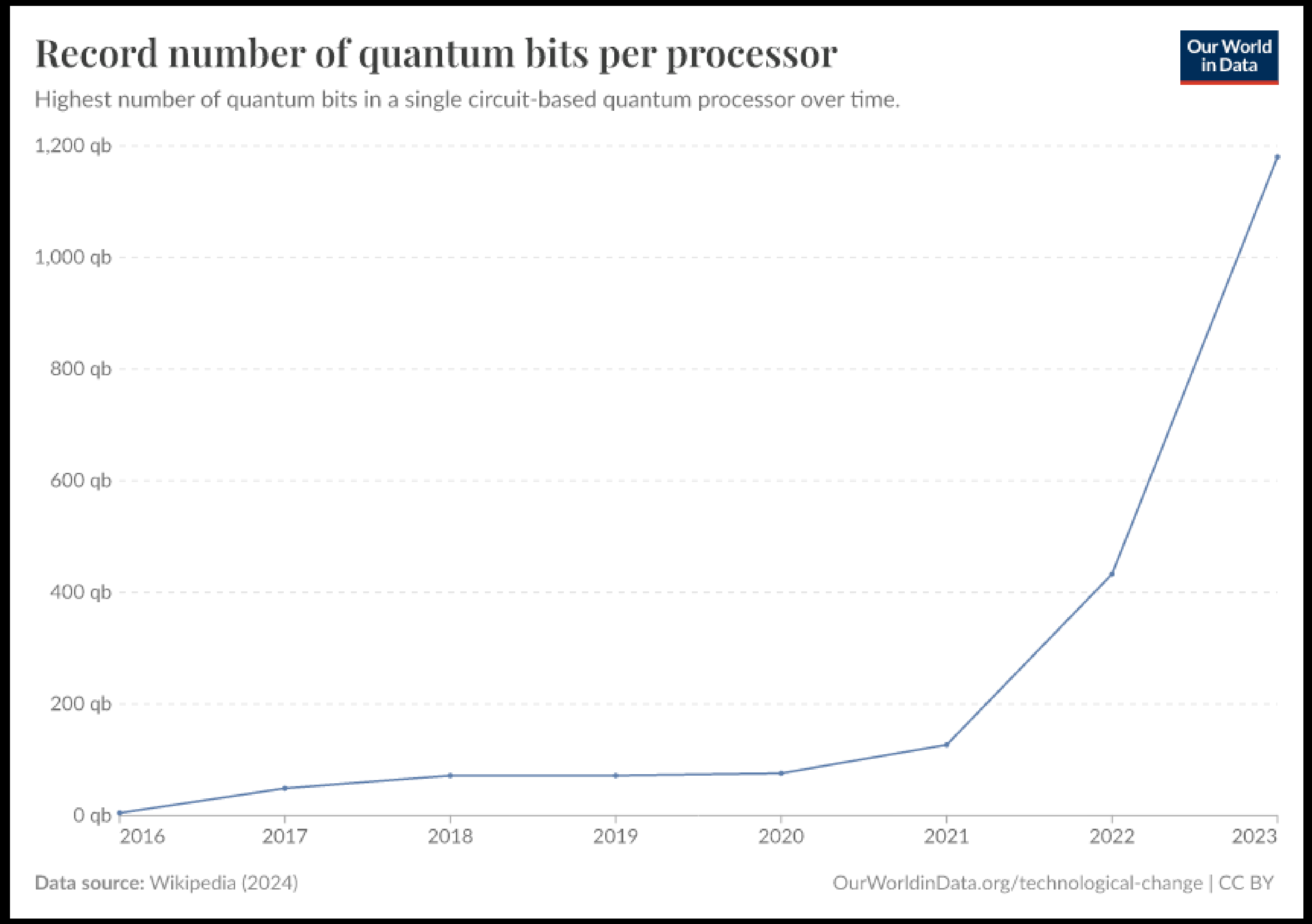
All these approaches aim to learn something big and then reduce - maybe we could learn small from the start

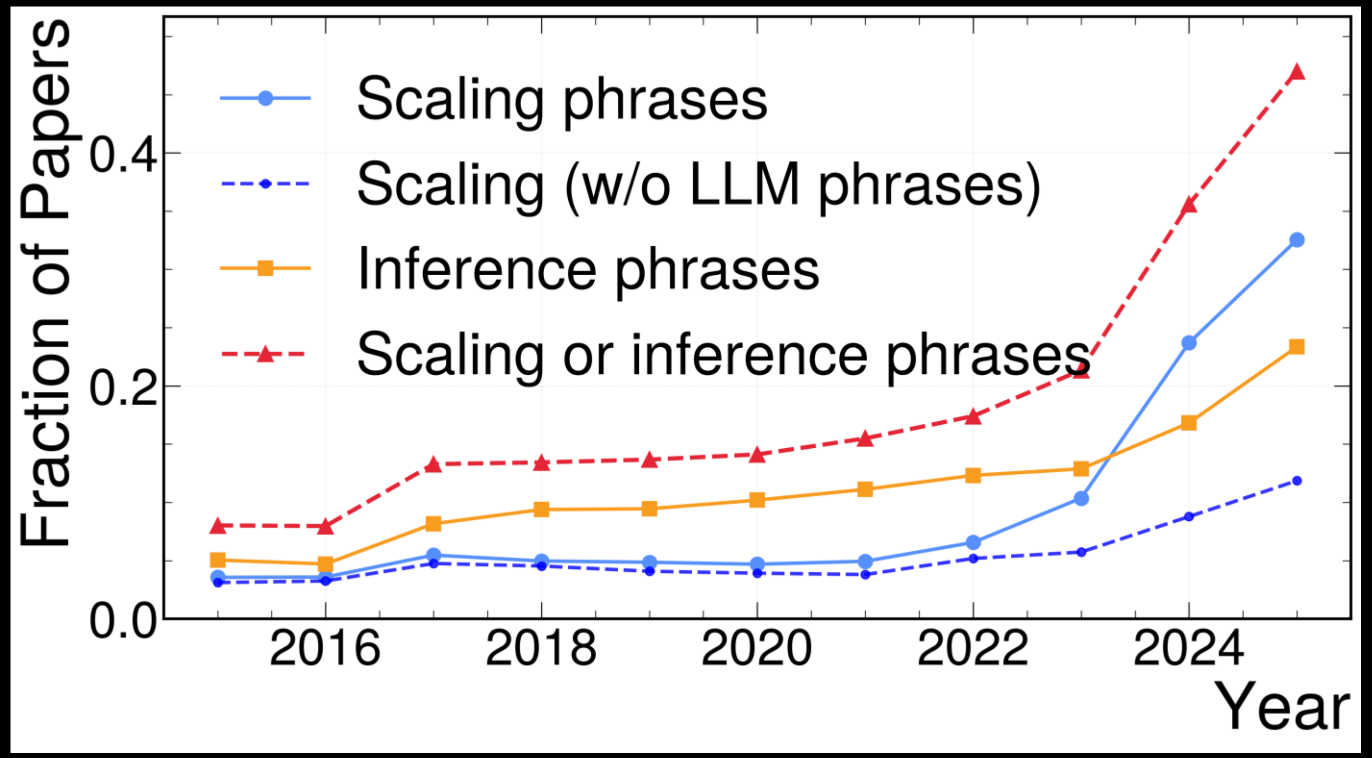
Could we learn ML suited directly for our hardware?

There are a few paradigm shifting potentials

[Neuromorphic computing](#) [Quantum computing](#)







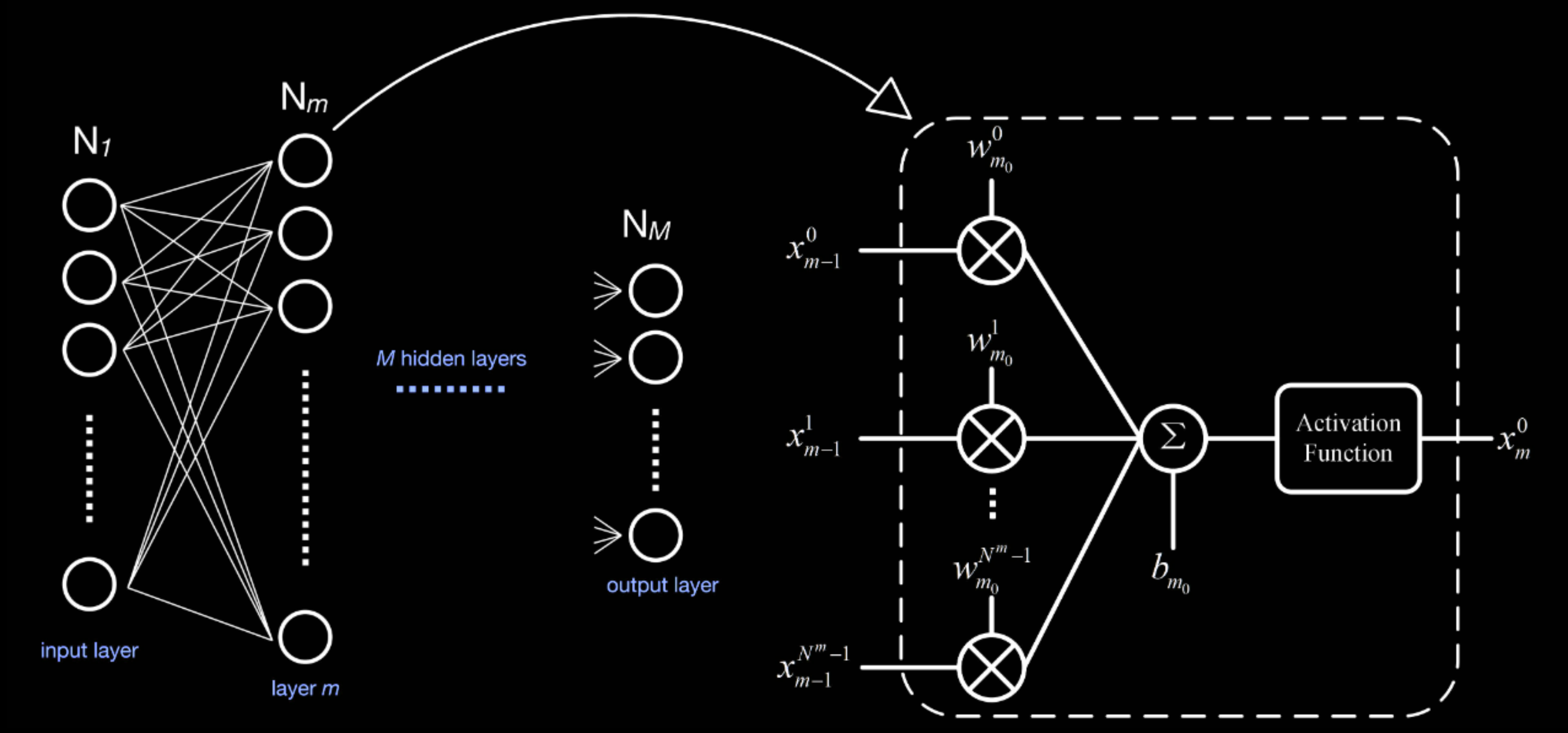
Abstracts from NeurIPS and ICML that mention keywords related to scaling or inference

Fast inference needs are increasing

We are moving to more specialised hardware

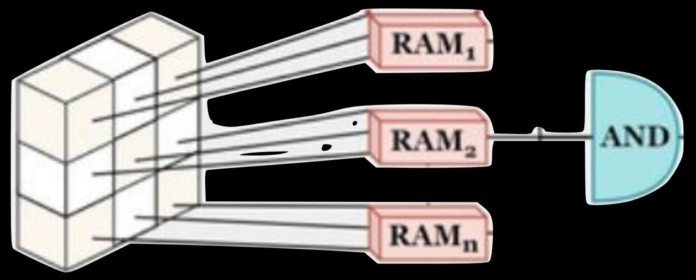
Our techniques for making ML fast are not enough

Logic neural networks



Evolution of the field

Weightless learning



1959

Ntuple

Early character recognition
by storing states in RAM

1984

Wisard

Similar to Ntuple, but with multiple classes
One of the first commercial real-time image recognition systems

2022

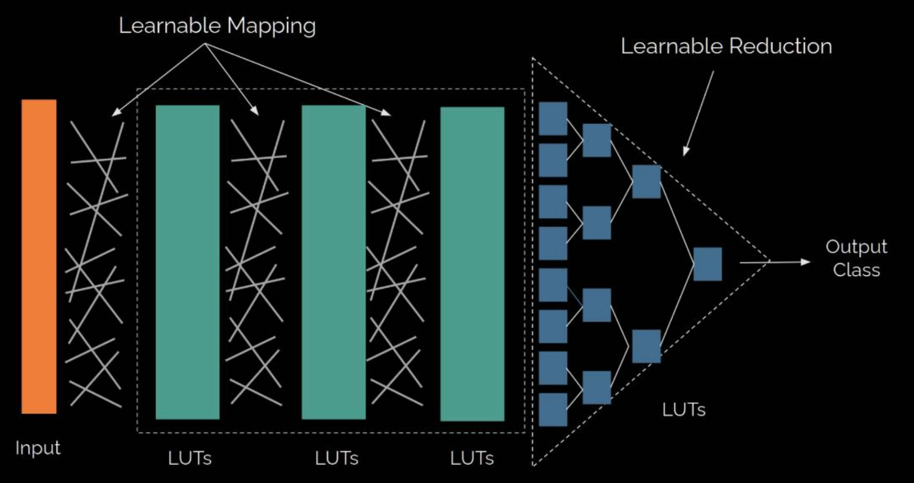
Weightless NN

Same as Wisard, but with several clever tricks

2024

Differentiable weightless NN

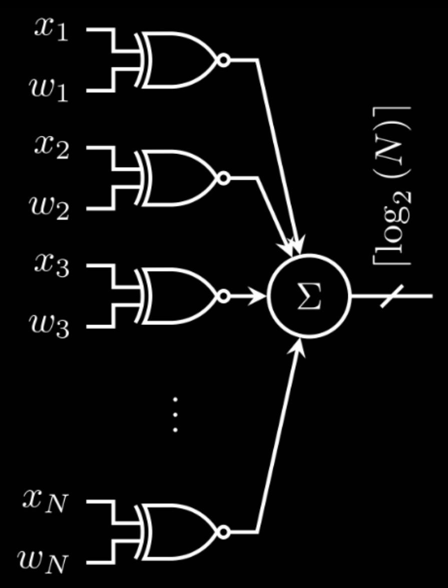
EFD to estimate *gradients* of LUT mapping



Quantised learning

2016

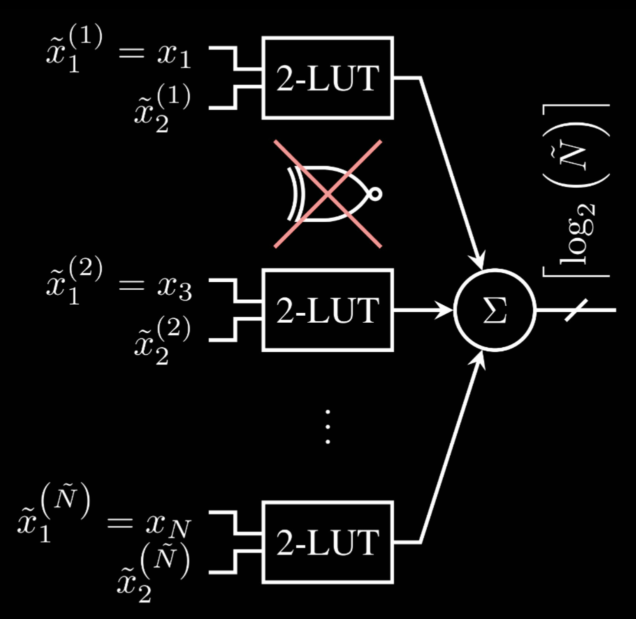
Binary neural net



Weights are 0 or 1 (i.e. XNOR)
Accumulate result with pop count

2019

LUTNet
NullaNet



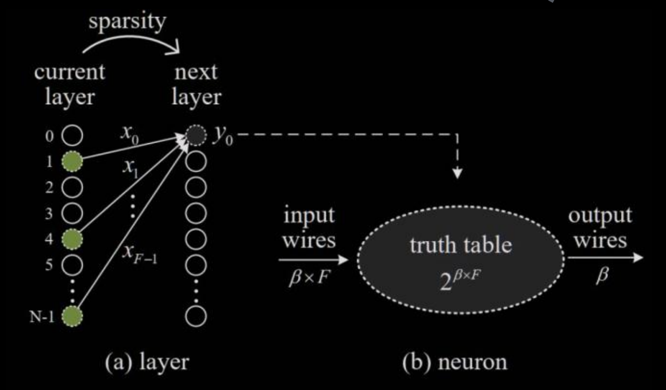
$\tilde{x}_1^{(1)} = x_1$
$\tilde{x}_2^{(1)}$
2-LUT
$\tilde{x}_1^{(2)} = x_3$
$\tilde{x}_2^{(2)}$
2-LUT
$\tilde{x}_1^{(\tilde{N})} = x_N$
$\tilde{x}_2^{(\tilde{N})}$
2-LUT
Σ
$\lceil \log_2(\tilde{N}) \rceil$

Compile a BNN efficiently in LUTs

2020

Logic net

Design for LUTs from the start



2022

Differentiable logic gate NN

No weights at all, just learn which logic gates to use

Newer developments

PolyLUT

Tree LUT

AmigoLUT

NeuraLUT

More details on different methods

Method	Year	Key Innovation	Advance	Limitation
N-tuple	1959	LUT pattern matching	First weightless	No generalization
WiSARD	1984	Commercial RAM-NN	Hardware, multi-class	Exponential memory
BNNs	2016	±1 weights, XNOR	Trainable with backprop	Still arithmetic (popcount)
FINN	2017	FPGA dataflow	Practical deployment	Arithmetic-based
LUTNet	2019	Arbitrary Boolean ops	2× area efficiency	Exponential params with K
LogicNets	2020	Neurons = truth tables	Direct LUT mapping	Needs high sparsity
DLGNs	2022	Learn gate type (not weights)	Zero arithmetic	Training difficulty
PolyLUT	2023	Polynomial neurons	Fewer layers	Diminishing returns D>2
DWN	2024	Differentiable WNNs	135× energy efficiency	Tabular focus
NeuraLUT	2024	MLPs inside LUTs	Better accuracy	Complex training

Logic gate neural networks

Construct neural network from logic gates
instead of nodes, we have logic gates

Convert the input to a binary representation
different representations can produce different results

Each node receives two inputs
The connections are randomly initialised

Outputs are summed so we can classify or regress

$$\hat{y}_i = \sum_{j=i\cdot n/k+1}^{(i+1)\cdot n/k} a_j/\tau + \beta$$

Final output

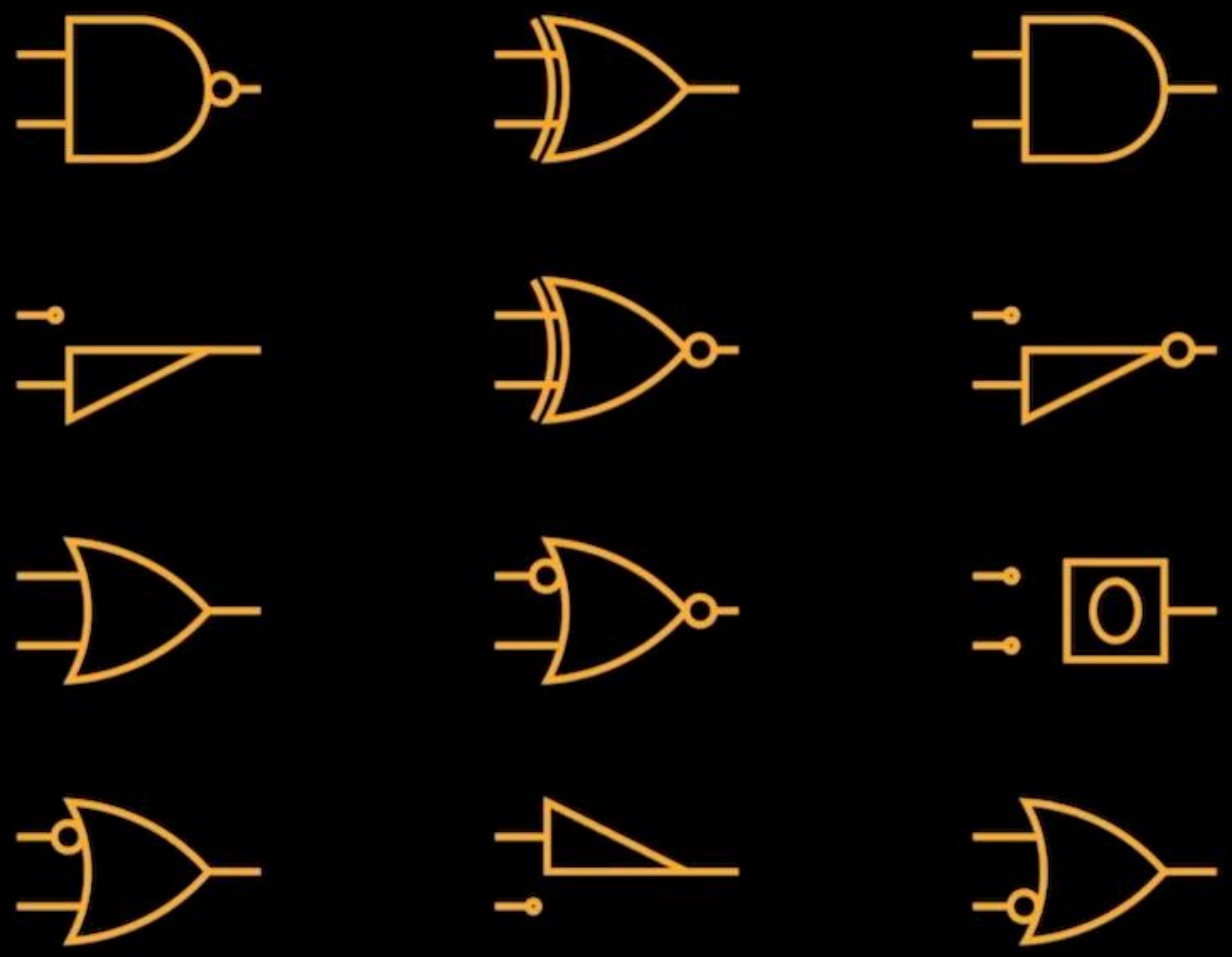
Output dimension

Neuron output

Optional offset

Number of output neurons

Normalisation temperature



Video adapted from [4], made by Felix Petersen et al.

Why it is fast

At inference each 16 gate block is replaced by most probable gate

Binary computations are fast

Compiler can optimise the binary logic

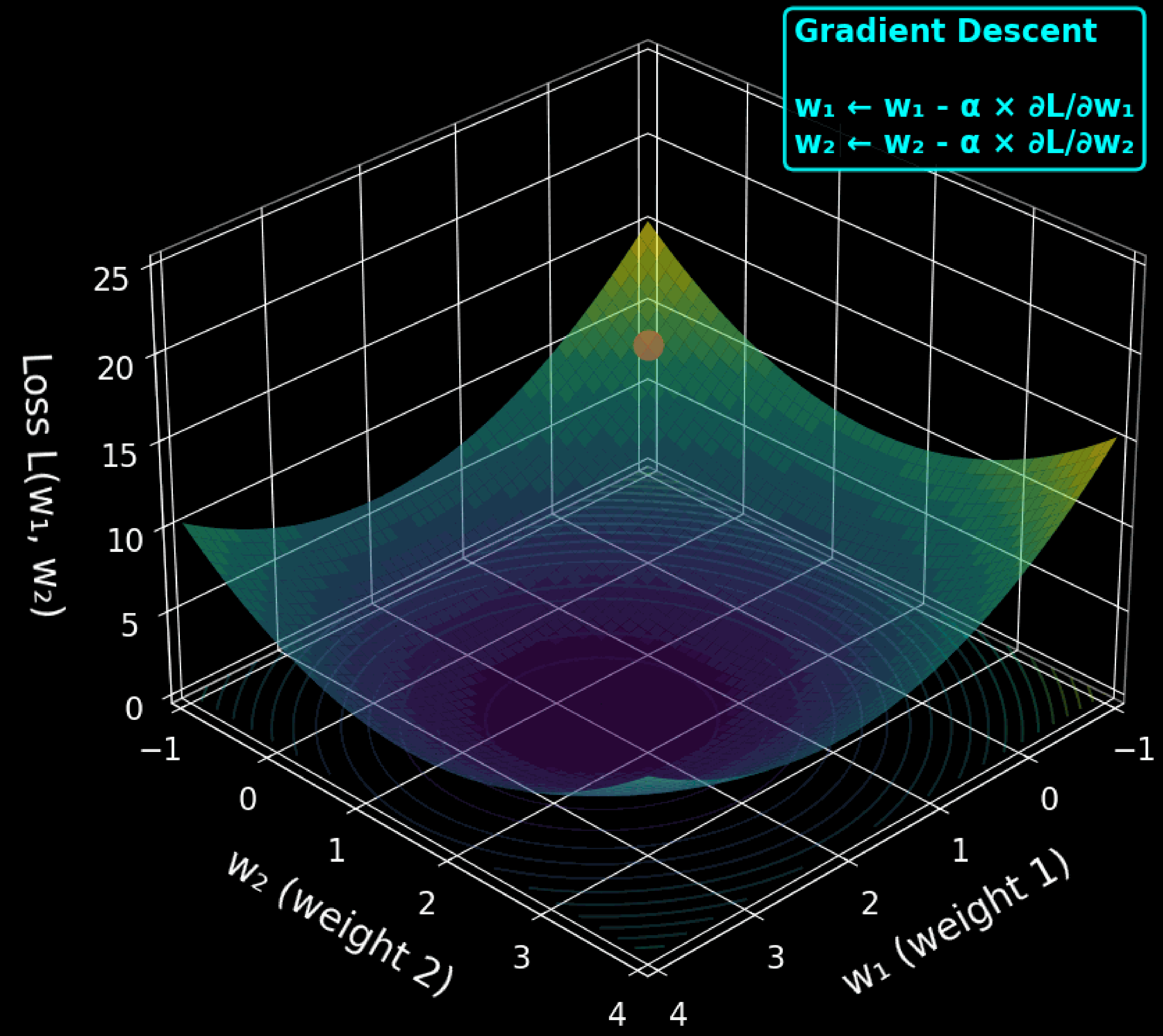
No matrix multiplications!

Problem: logic gates aren’t differentiable

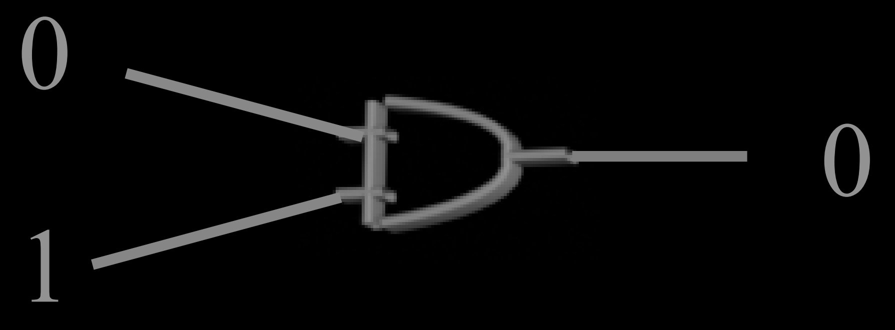
Most ML is done with gradient descent

Gradient descent needs differentiable variables

Logic gates aren’t differentiable



AND gate



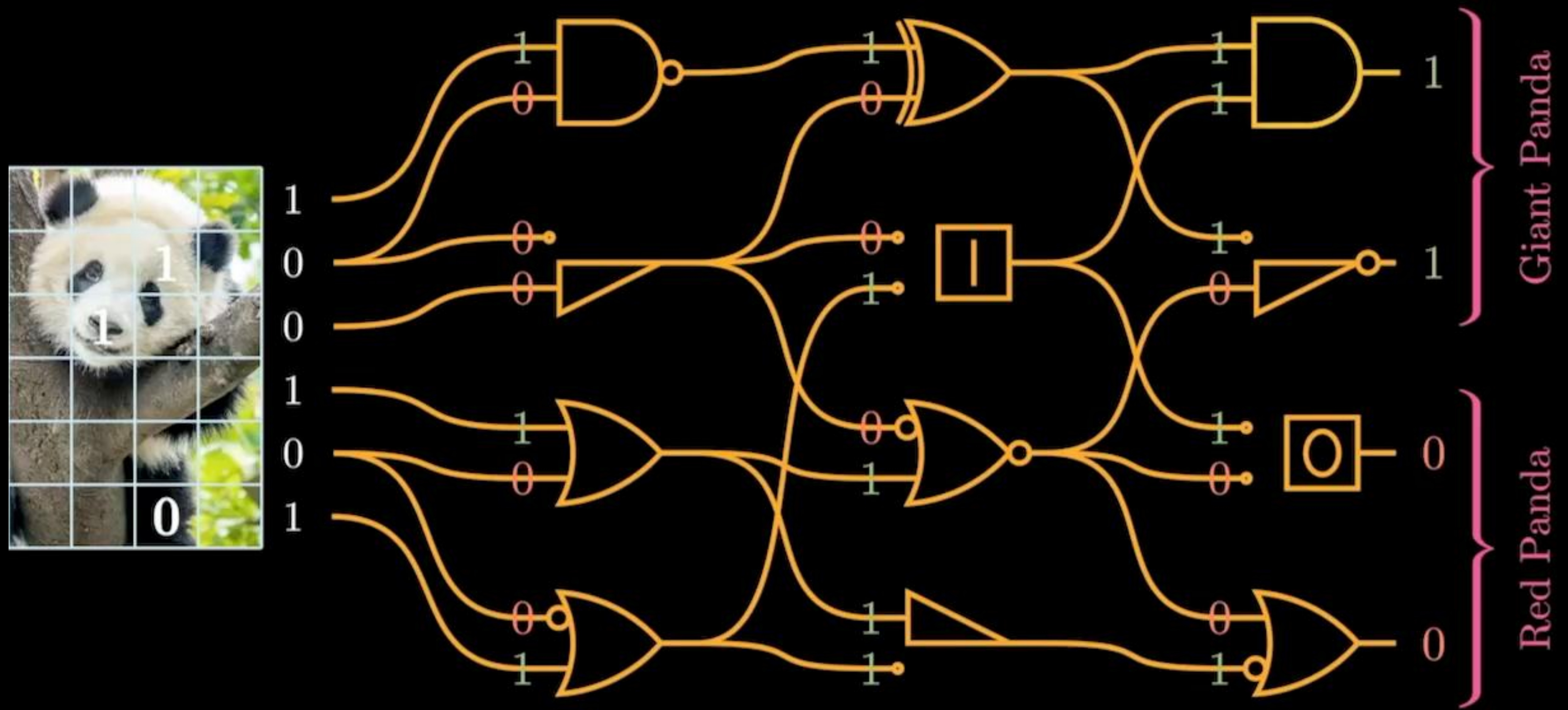
$$f(A,B) = \begin{cases} 1, & A = 1 and B = 1 \\ 0, & otherwise \end{cases} \Rightarrow \frac{\partial f}{\partial A} and \frac{\partial f}{\partial B} are not defined$$

Making it differentiable

Gate operations are continuous approximations
This is how we make it differentiable

Softmax of the 16 gate blocks
This is how we learn which gate is best for inference

During training evaluate 16 gates for each “neuron”
slow training, but quick inference when we replace each block of 16 with the most probable gate



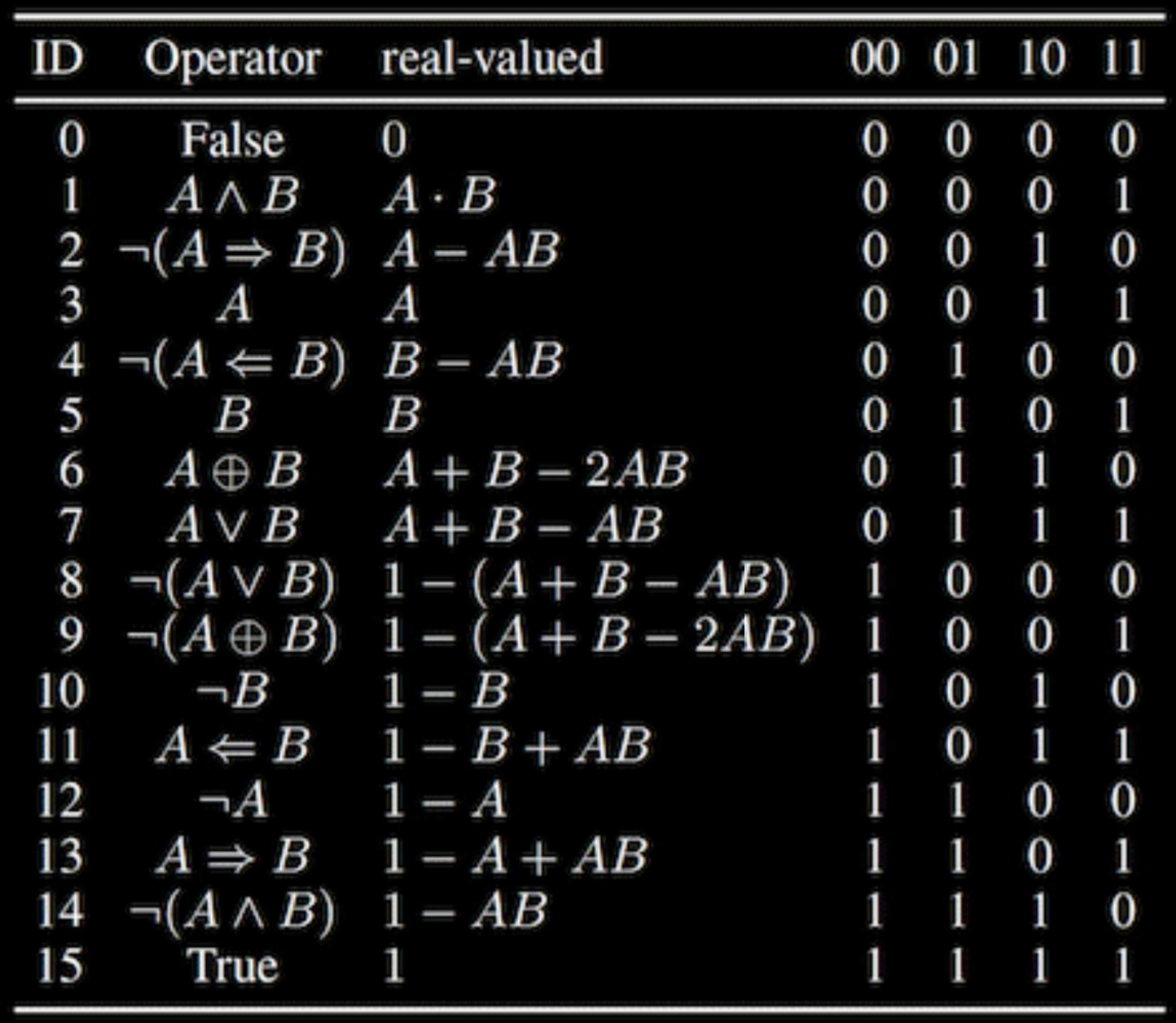
Video adapted from [4] , made by Felix Petersen et al.

Learnable weights

Logic gate operation

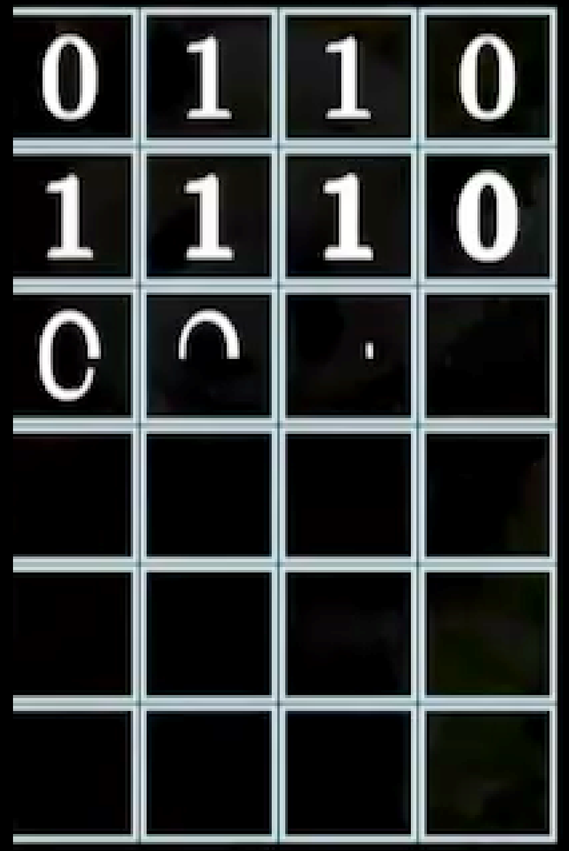
Neuron output

$$a' = \sum_{i=0}^{15} \frac{e^{\mathbf{w_i}}}{\sum_j e^{\mathbf{w_j}}} \cdot f_i(a_1, a_2)$$



ID	Operator	real-valued	00	01	10	11
0	False	0	0	0	0	0
1	$A \wedge B$	$A \cdot B$	0	0	0	1
2	$\neg(A \Rightarrow B)$	$A - AB$	0	0	1	0
3	A	A	0	0	1	1
4	$\neg(A \Leftarrow B)$	$B - AB$	0	1	0	0
5	B	B	0	1	0	1
6	$A \oplus B$	$A + B - 2AB$	0	1	1	0
7	$A \vee B$	$A + B - AB$	0	1	1	1
8	$\neg(A \vee B)$	$1 - (A + B - AB)$	1	0	0	0
9	$\neg(A \oplus B)$	$1 - (A + B - 2AB)$	1	0	0	1
10	$\neg B$	$1 - B$	1	0	1	0
11	$A \Leftarrow B$	$1 - B + AB$	1	0	1	1
12	$\neg A$	$1 - A$	1	1	0	0
13	$A \Rightarrow B$	$1 - A + AB$	1	1	0	1
14	$\neg(A \wedge B)$	$1 - AB$	1	1	1	0
15	True	1	1	1	1	1

Convolutional differentiable logic gate neural networks



Linear layers is not enough for image tasks
empirically struggled to train over 6 layers

-> Replace normal CNN kernel by a binary tree
aggregates information while keeping expressivity

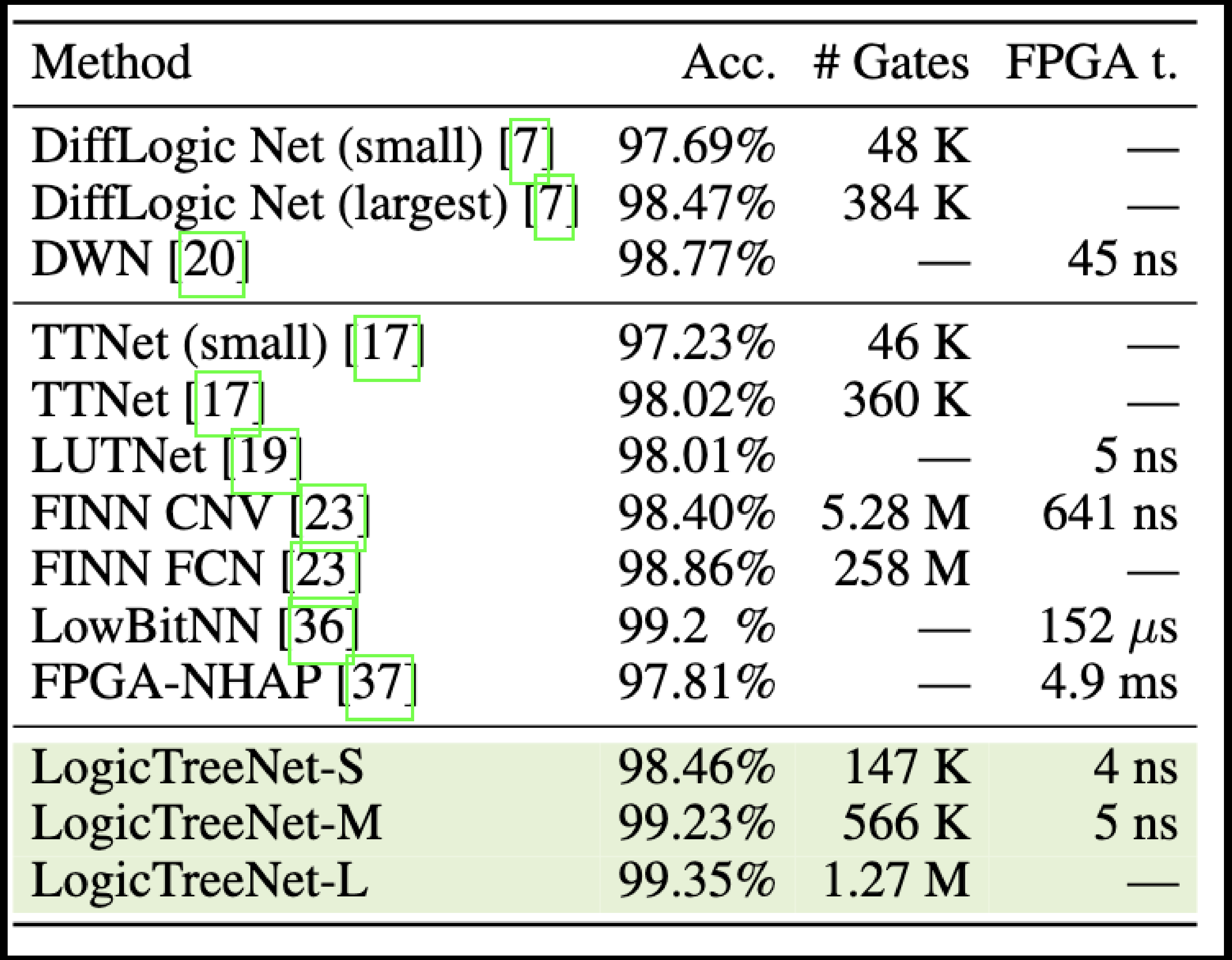
One channel for each input bit
We learn the significance of each bit

Create special *Or* pooling layers
fast and only need to propagate through the maximum activations

$$\perp_{max}(a,b) = max(a,b)$$

Video adapted from [4] , made by Felix Petersen et al.

Current state of DLGN research



Method	Acc.	# Gates	FPGA t.
DiffLogic Net (small) [7]	97.69%	48 K	—
DiffLogic Net (largest) [7]	98.47%	384 K	—
DWN [20]	98.77%	—	45 ns
TTNet (small) [17]	97.23%	46 K	—
TTNet [17]	98.02%	360 K	—
LUTNet [19]	98.01%	—	5 ns
FINN CNV [23]	98.40%	5.28 M	641 ns
FINN FCN [23]	98.86%	258 M	—
LowBitNN [36]	99.2 %	—	152 μs
FPGA-NHAP [37]	97.81%	—	4.9 ms
LogicTreeNet-S	98.46%	147 K	4 ns
LogicTreeNet-M	99.23%	566 K	5 ns
LogicTreeNet-L	99.35%	1.27 M	—

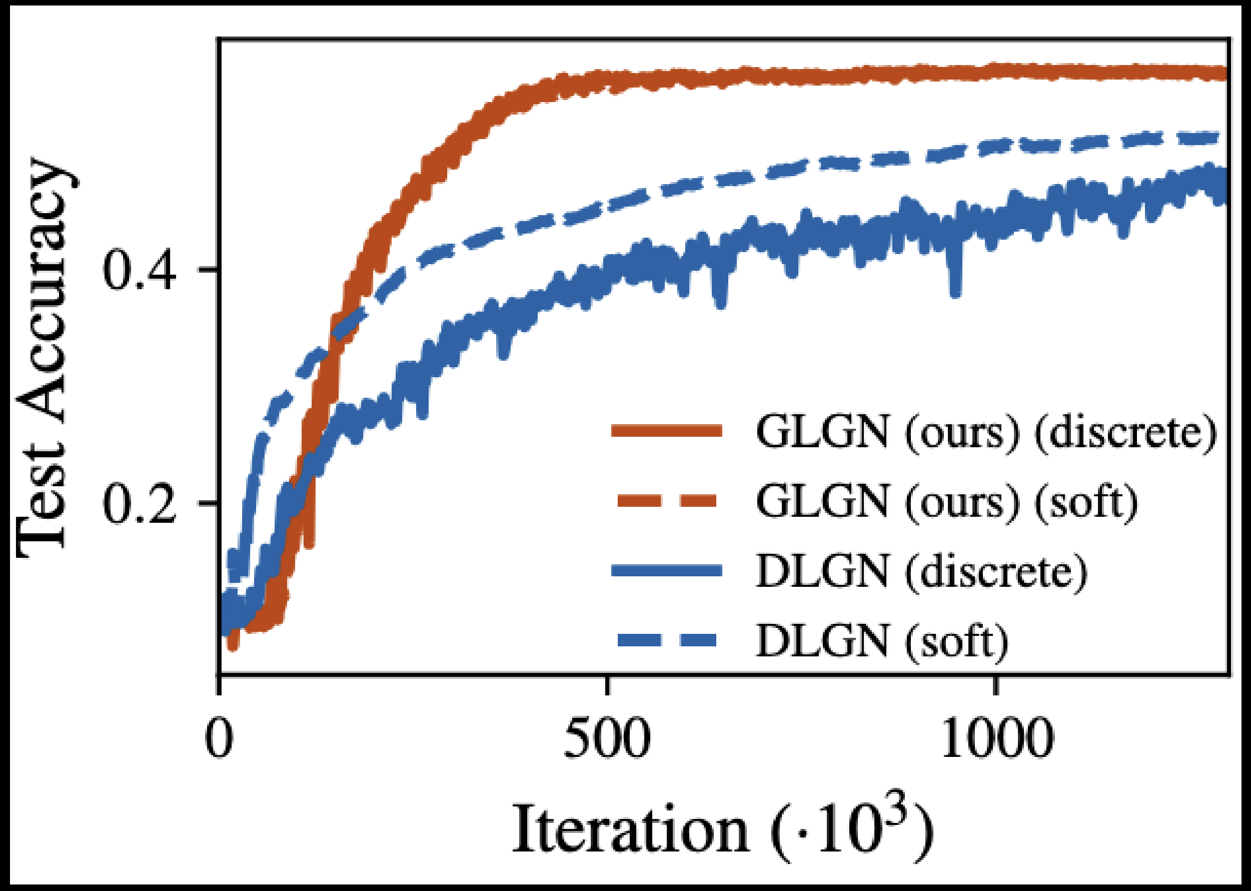
Speed records: 1 million+ MNIST inferences/second on a single CPU core

86.3% on CIFAR-10 using only 61 million logic gates

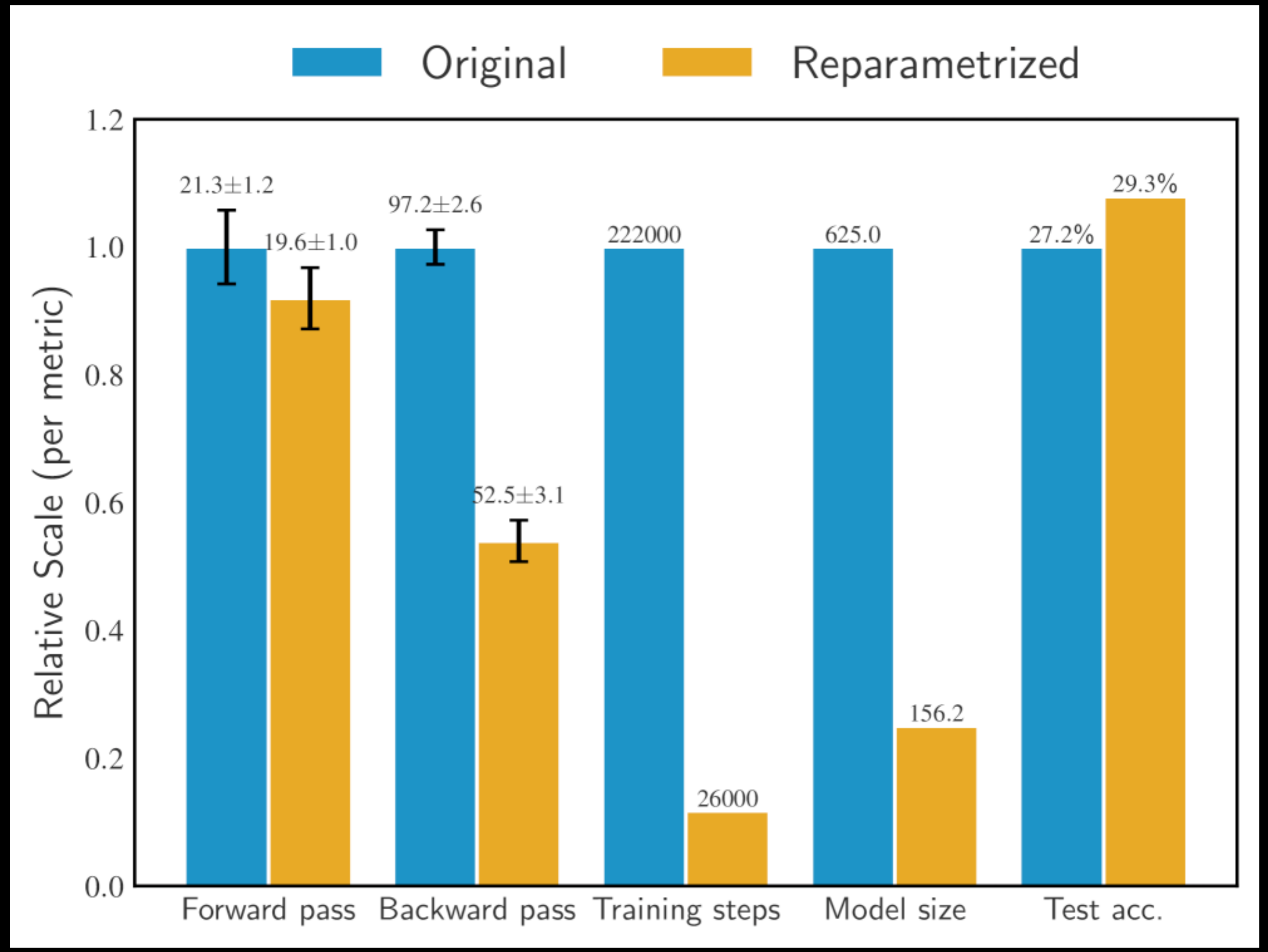
O(10-100) times smaller than state-of-the-art models at comparable accuracy

NeurIPS talk

Reducing discretisation gap



Speeding up training



New architectures: Cellular automata Recurrent DLGN

Reducing the number of parameters

There are also several startups in this area

Verifiability and interpretability

Normal neural networks

Interpretability and verifiability remains an open issue

Current methods often rely on approximations or is NP hard

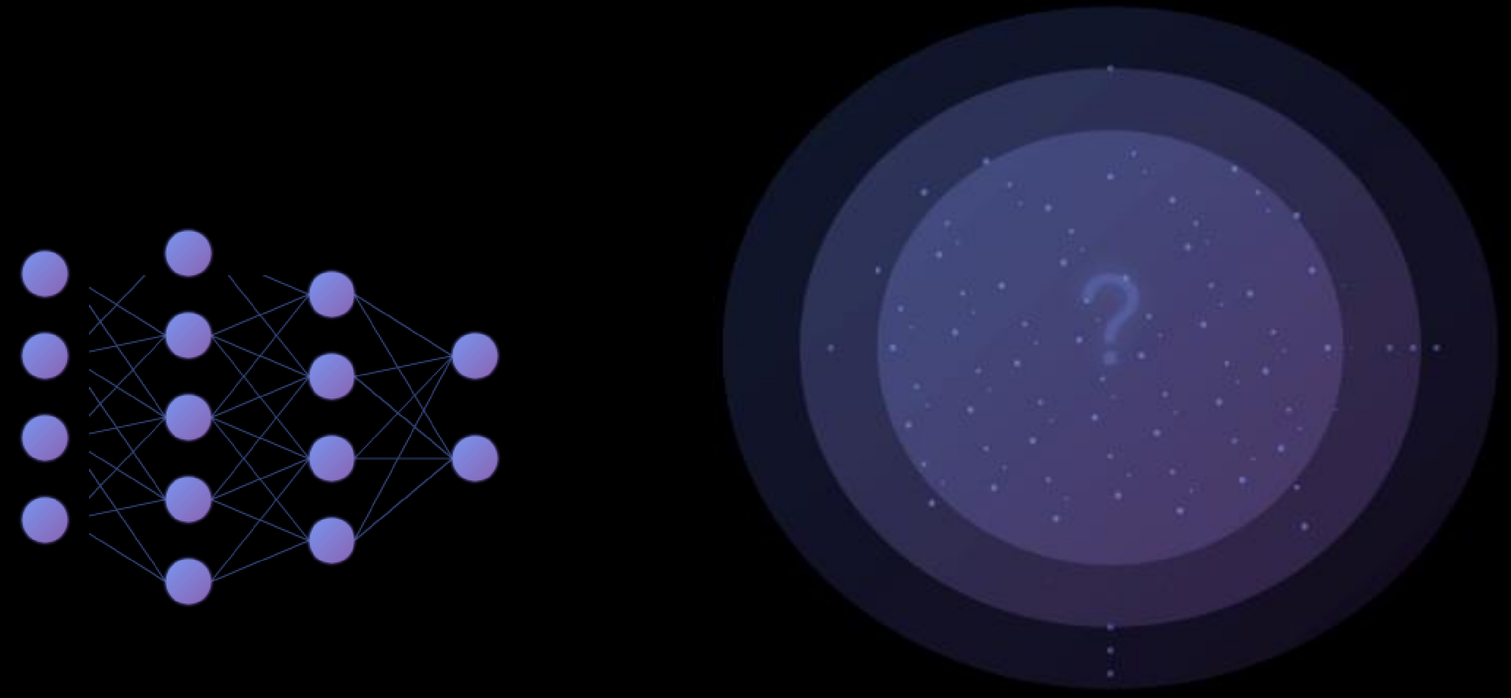
Best tools fall short in numbers of parameters they can handle with orders of magnitude

Lots of work goes into testing LLM behaviour [Anthropic, ChatGPT]

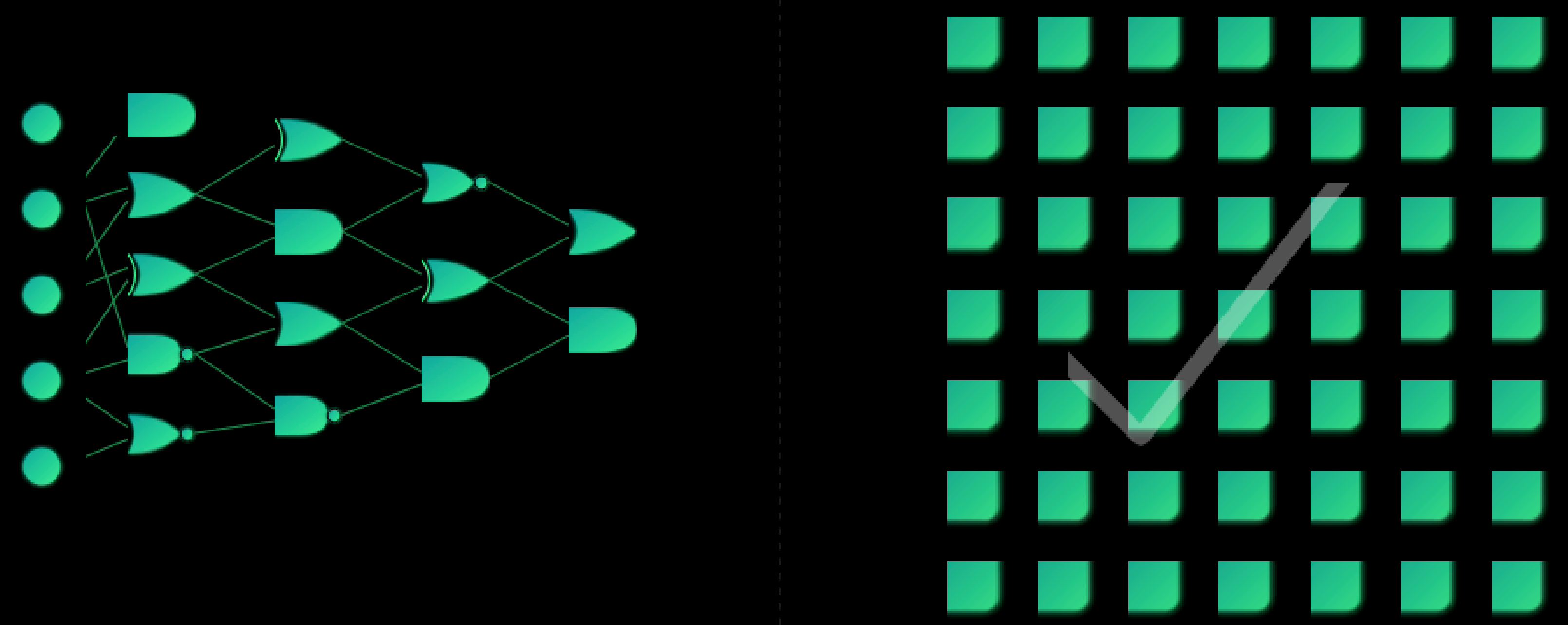
LNNs

Because the states are limited, you can perform mathematical proofs (Kresse et al)

You don't need to enumerate all possibilities, SAT solvers can help



Continuous, infinite state space



Discrete, finite state space

When does ChatGPT give shorter responses?

Monday mornings 0%

When you’re rude to it 0%

In December as compared to spring 0%

After midnight 0%

Which of these prompting tricks actually improves LLM performance?

Take a deep breath and think step by step

0%

I have no fingers, please write the full code

0%

I'll tip you $200 for a better answer

0%

All of the above

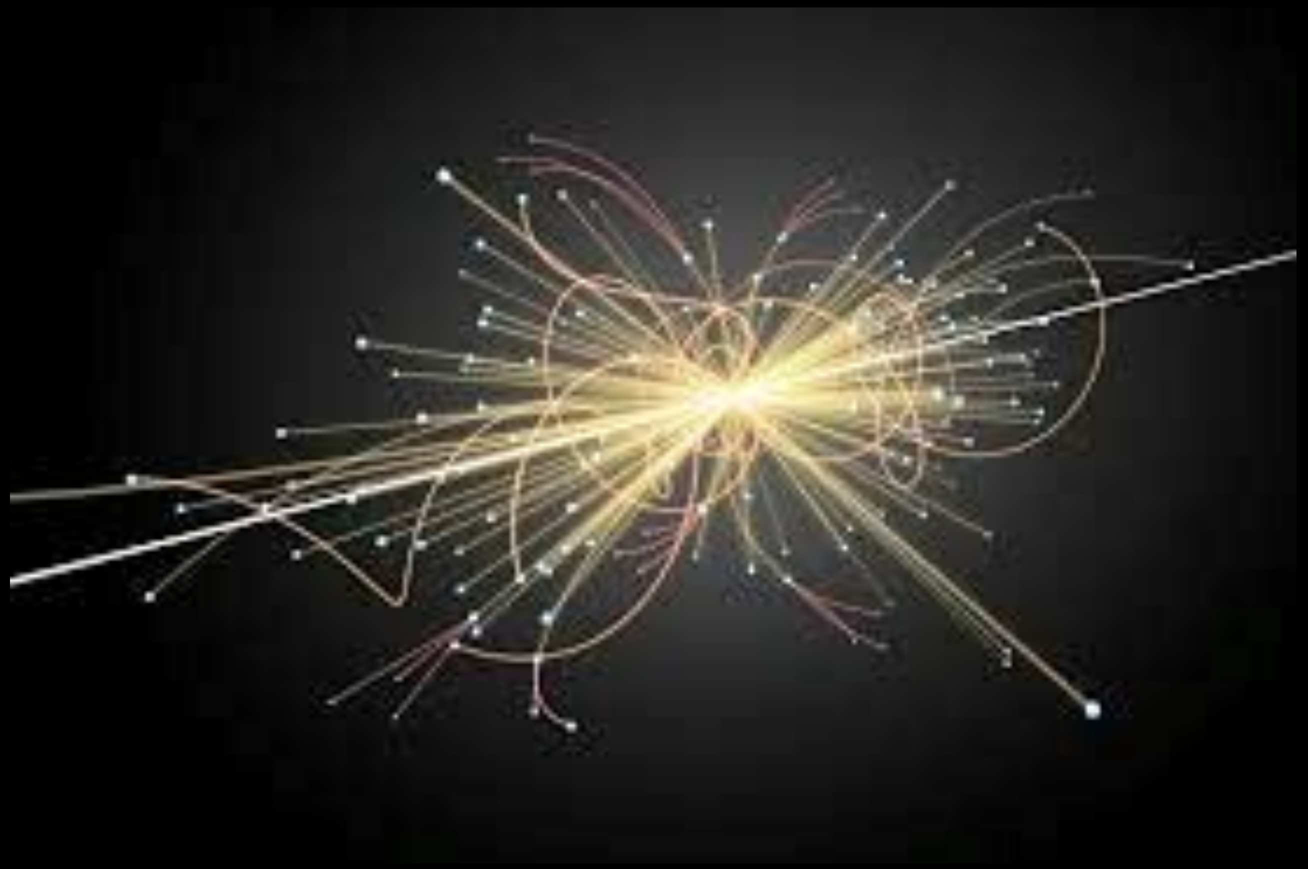
0%

The model IS a Boolean circuit. Training learns the circuit structure/contents

DLGNs are very fast and show good accuracies

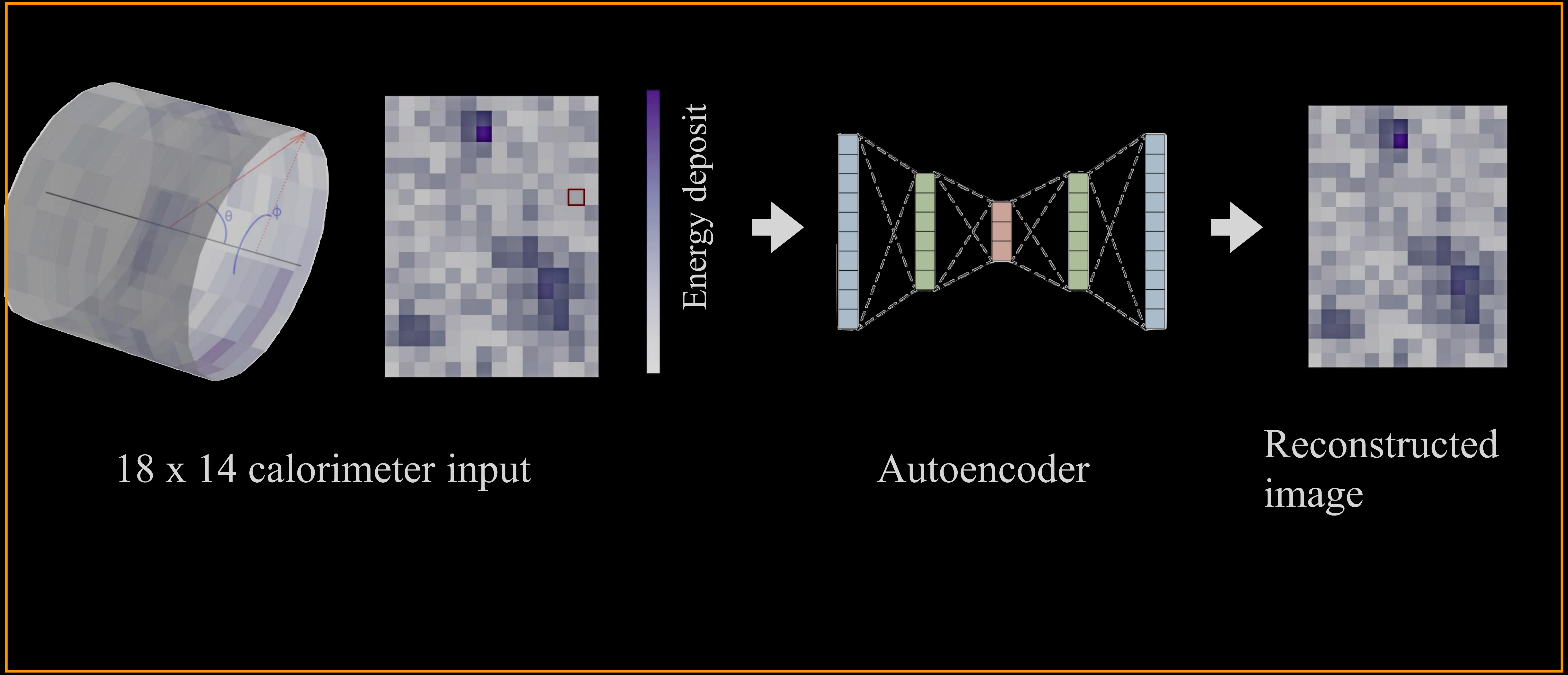
It has the potential to be verifiable and explainable

HEP applications

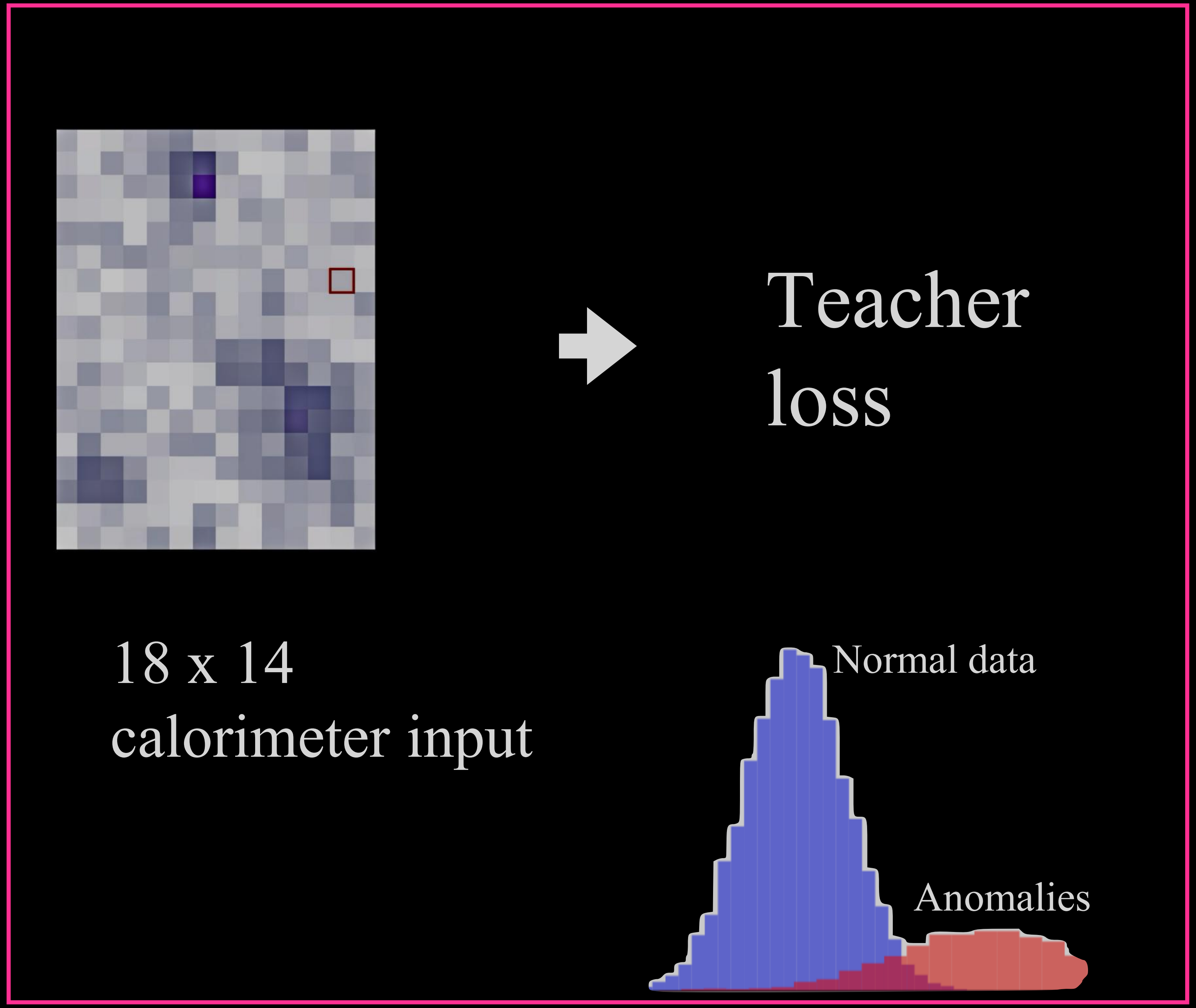


CICADA anomaly detection at the CMS Level-1 Trigger

Teacher



Student



Student is implemented on FPGA in the L1T which has output rate of 100 kHz

Can we achieve better physics performance or latency?

[5] CICADA collaboration

Results

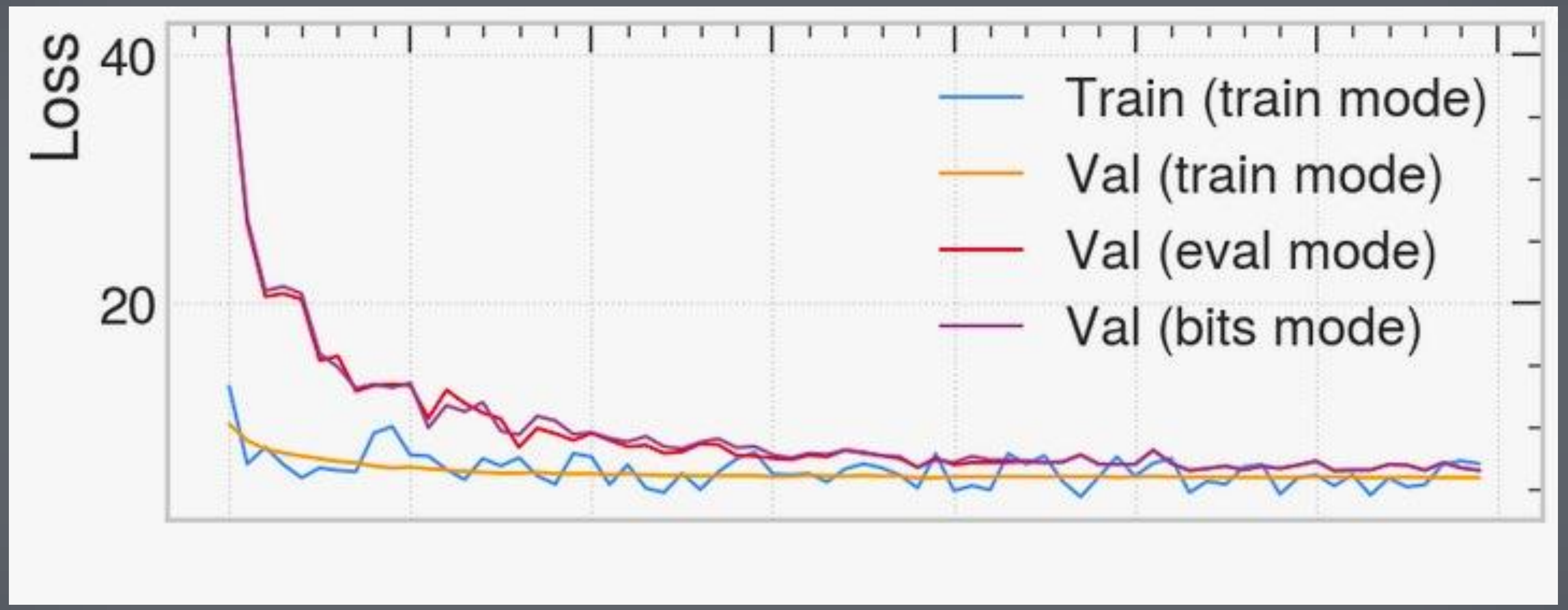
Convolutional logic gate neural net outperforms baseline implementation (QKeras)

Background is Zero Bias, $\bar{t}t$ is outlier

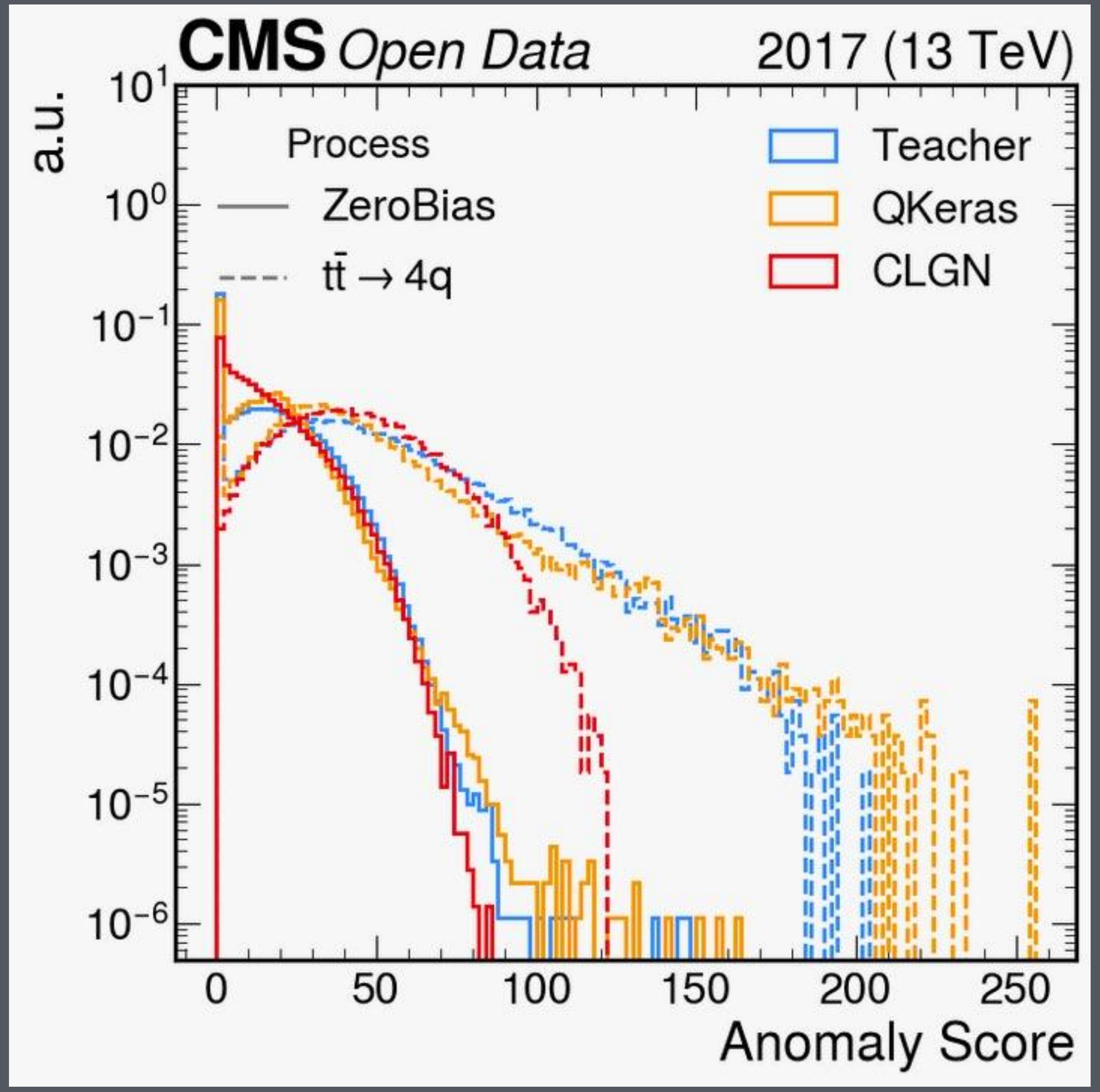
There are three possible outlier final states, two are reserved for validation and testing

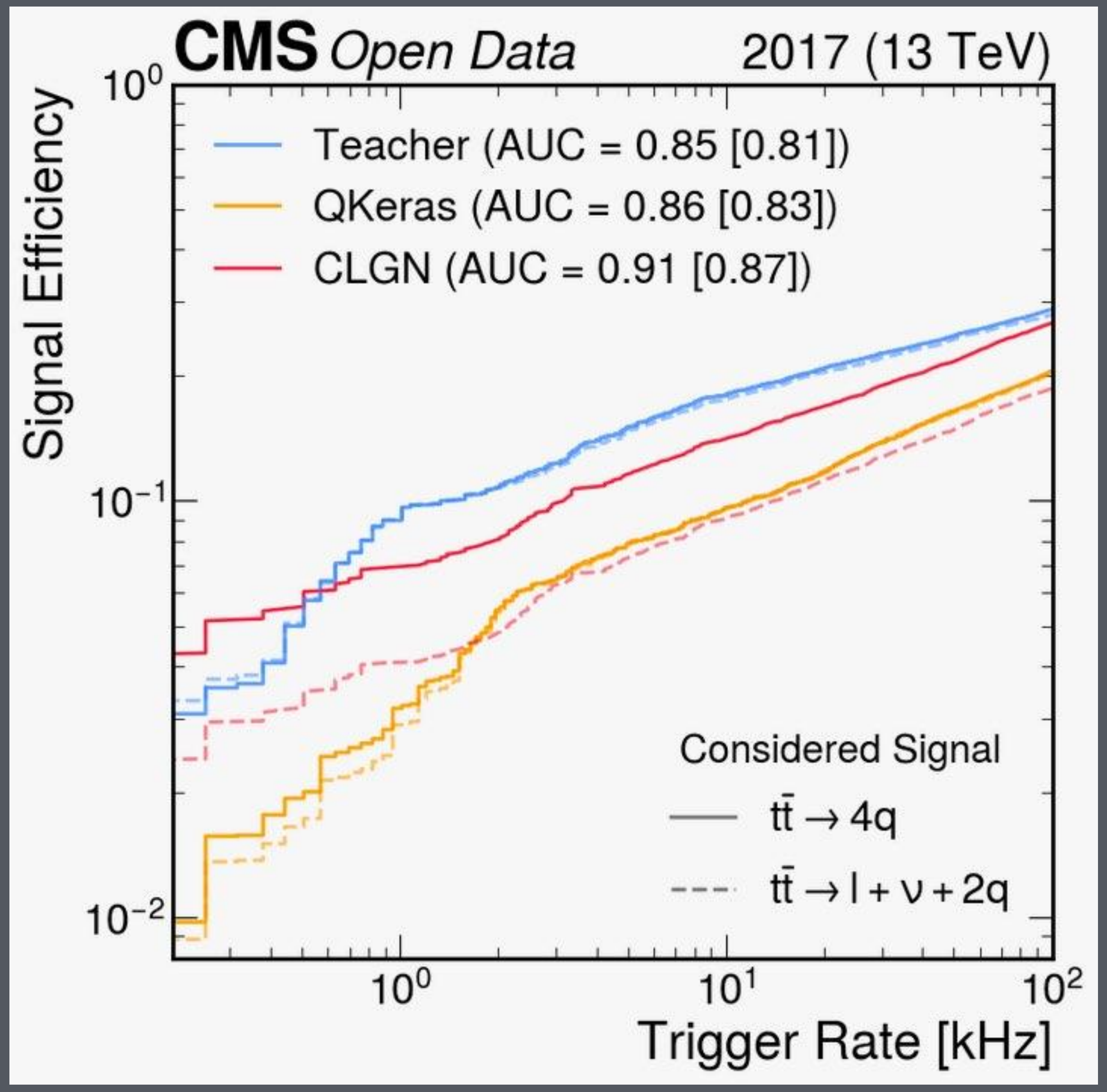
Very high anomaly scores lower since binarised inputs

This is okay, we can set the trigger threshold



A low gap between training and validation in bit mode is a sign of a good model





Results using Using 2017 CMS Open data - will be openly available very soon

Uses thermometer encodings to convert inputs to binary, described further in [6]

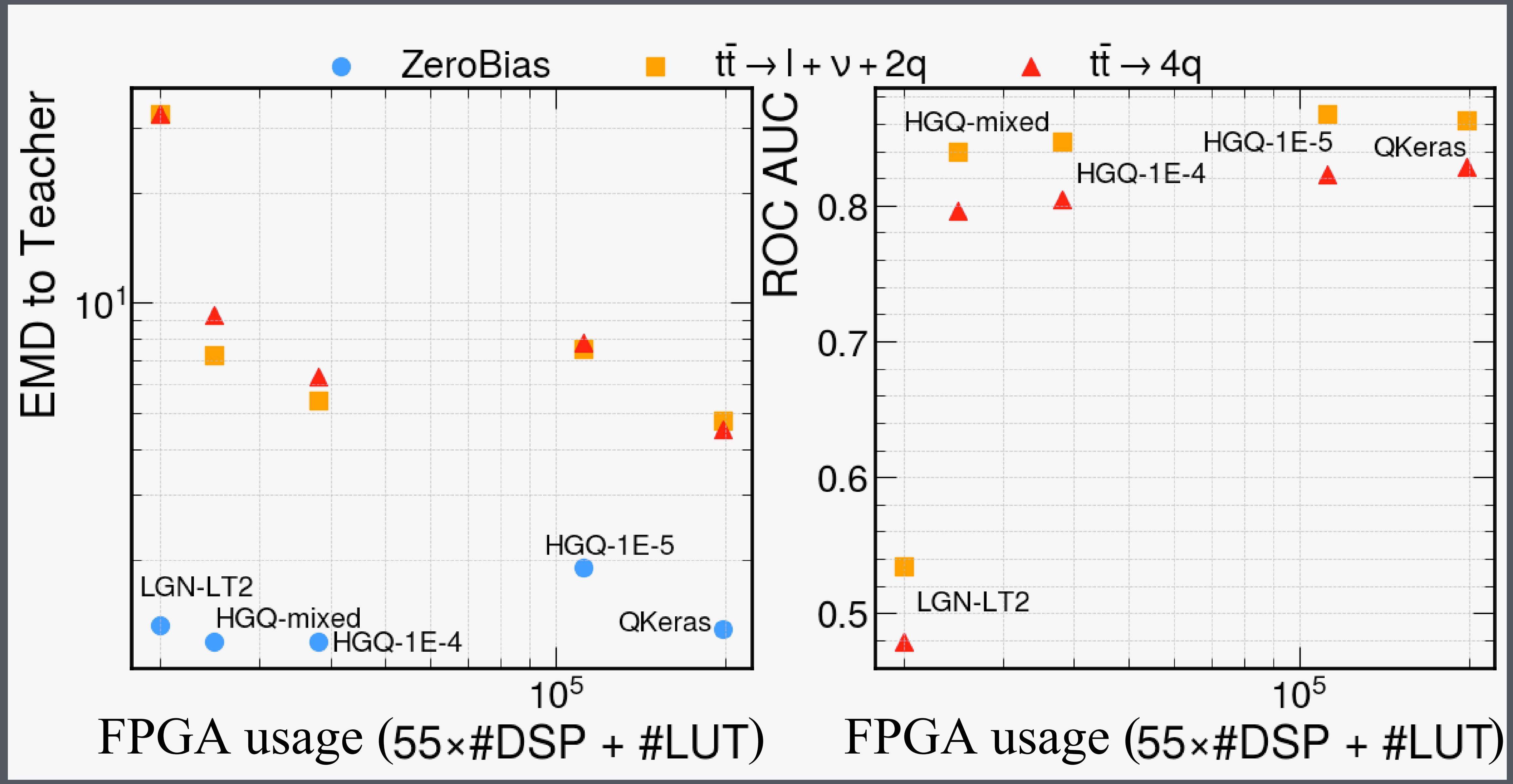
EMD: earth's mover distance

FPGA synthesis

Synthesised a logic gate deep neural net *(not convolutional)* for FPGA

Latency outperforms QKeras and HGQ* implementations
details of those can be found in [5]

We have no DSP usage since we don't have matrix multiplications



This model has half the trainable parameters as the previous slide, so the physics performance suffers a bit - working on a CNN FPGA implementation underway

Vitis HLS re-place-and-route hardware cost comparison
Synthesised for AMD Virtex-7

Model Label	Quantization Library	Latency (in cc)	DSPs	FFs	LUTs	HGQ EBOPs
QKeras	QKeras	16	697	50368	159447	-
HGQ-1E-5	HGQ	17	4	27776	111848	39170
HGQ-1E-4	HGQ	11	1	6229	38111	7570
HGQ-mixed	HGQ	8	0	3019	24947	3301
LGN-LT2	LGN	3	0	856	19977	-

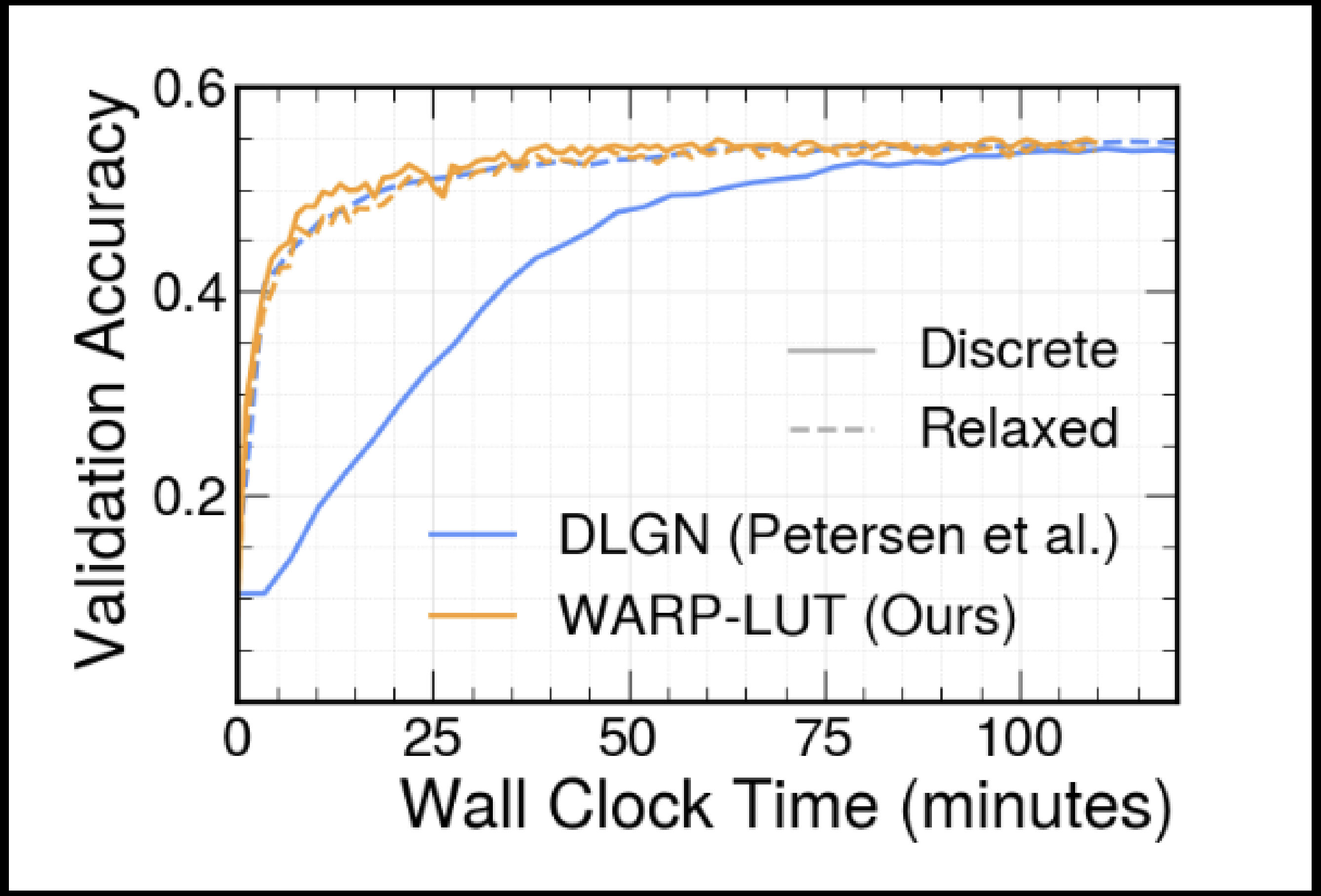
Vitis HLS estimates

* QKeras is a quantisation package where you change precision of the weights as you are training,
High Granularity Quantisation (HGQ) lets you have different precision in each layer

Our work so far

torchlogix

We are maintaining and developing a python library called torchlogix

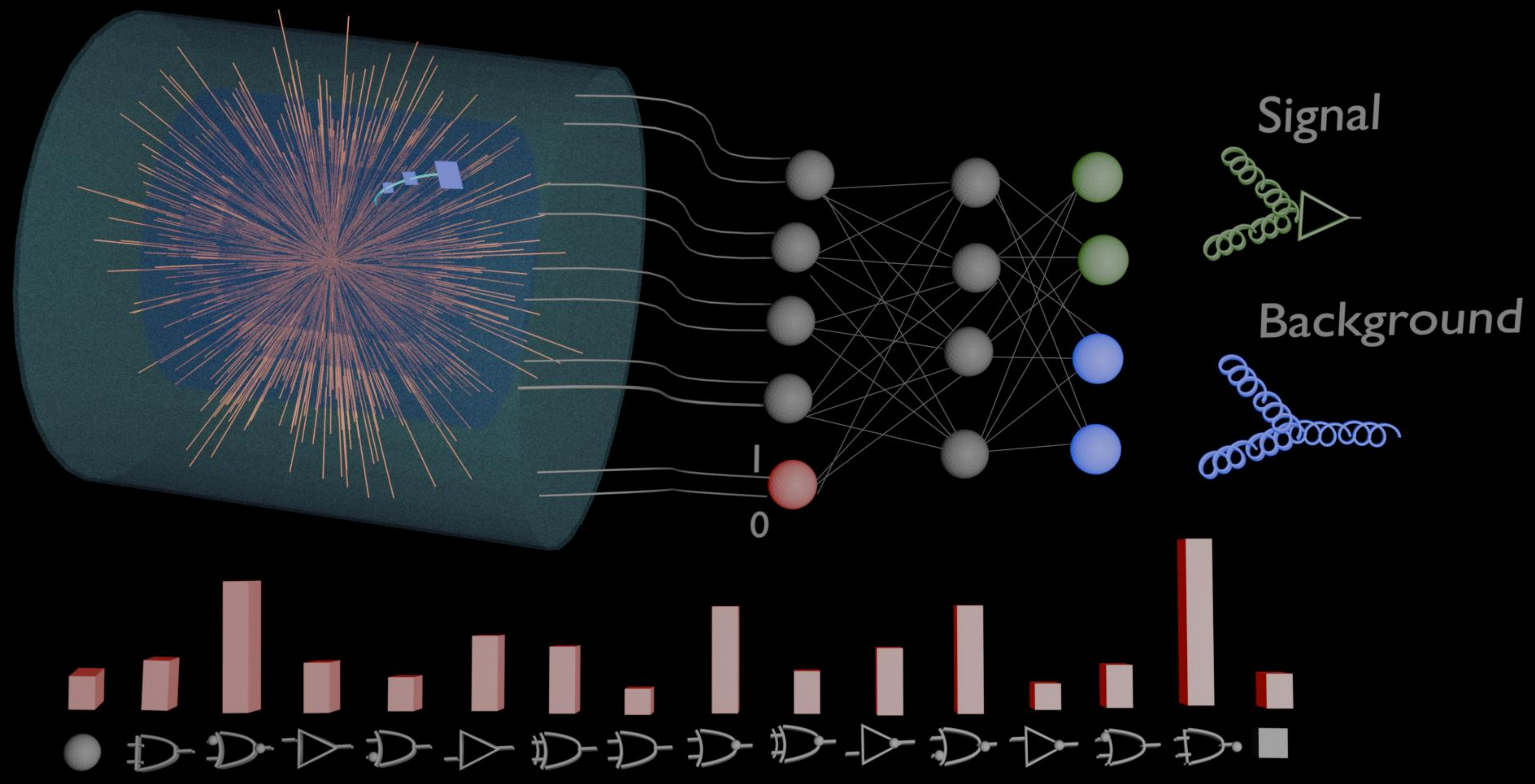


We have increased the scalability by changing binary gate representation, presented in this paper - which was shown at a NeurIPS workshop last year

Submitting an ICML paper on this topic

Also submitting an ICML position paper on the need to focus on inference speed

Potential applications in HEP



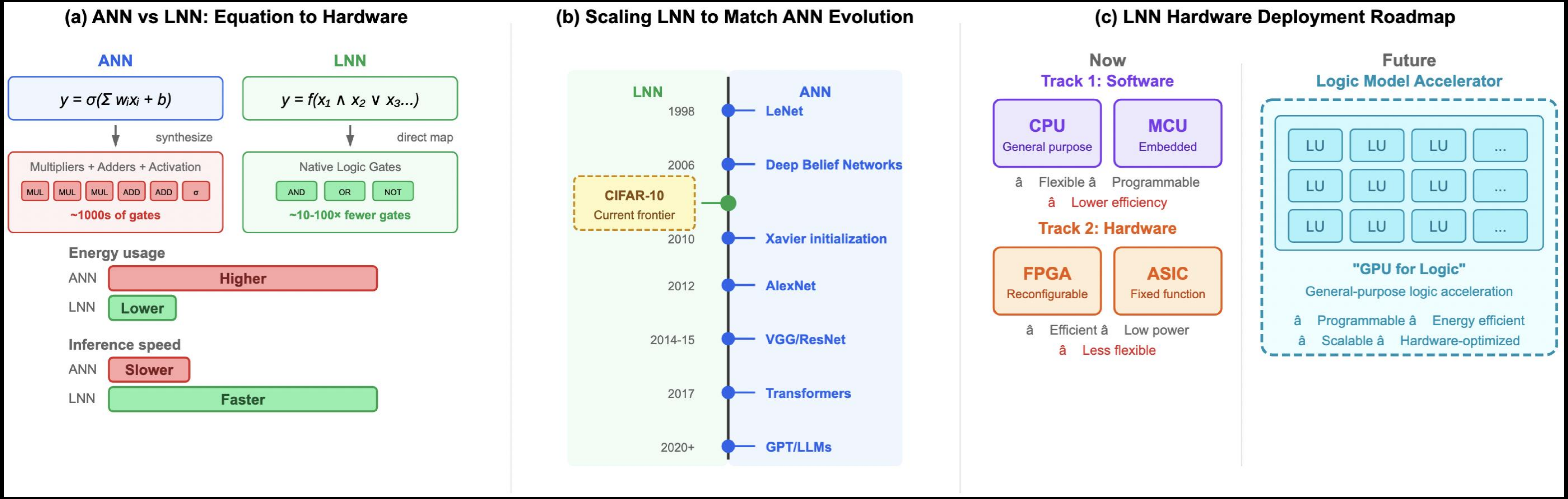
Many trigger applications - timing/accuracy tradeoff could shift

Could potentially be used directly on ASICS for FCC

Could be used for fast inference offline

Opens opportunities for physics verifiability in ML

Future of LGNs



Scaling up is the main issue:

Training can take a long time

There are gradient issues - increasing depth can harm accuracy

LNNs are fairly parameter sensitive

The FPGA tools are generally not being shared openly

Conclusion

CICADA

3 cc	FPGA LATENCY
0	DSP USAGE
~ 0.91	AUC (CICADA)
30-60%	MODEL COMPRESSION

STRENGTHS

No Matrix Multiply → Fast
Binary logic operations only, no expensive floating-point arithmetic

Maps Directly to FPGA LUTs
The trained model IS a Boolean circuit — no translation needed

Deterministic Latency
Fixed timing, no cache misses, no surprises — ideal for triggers

Interpretable
You can read the logic — potential for mathematical verification

LIMITATIONS (Current)

Harder to Train
Training takes longer than standard NNs; gradient issues with depth

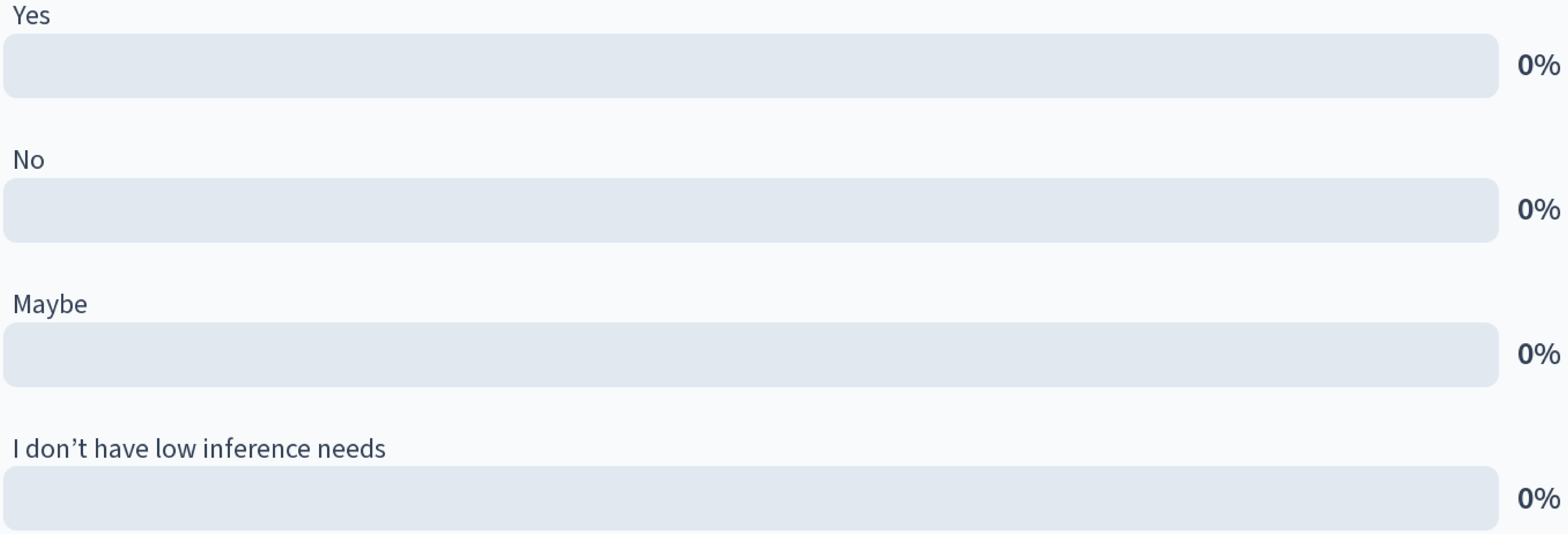
Scaling Still in Progress
Current frontier is CIFAR-10; larger problems need more research

Input Encoding Matters
Thermometer thresholds need tuning; representation affects results

Less Explored
Smaller research community than quantization/pruning methods

LGNs offer a **paradigm shift** *for ultra-fast ML inference — trading training complexity for* **massive deployment efficiency**.

Would you consider logic gate nerual nets for your low inference needs?



Yes	0%
No	0%
Maybe	0%
I don’t have low inference needs	0%

How to get started

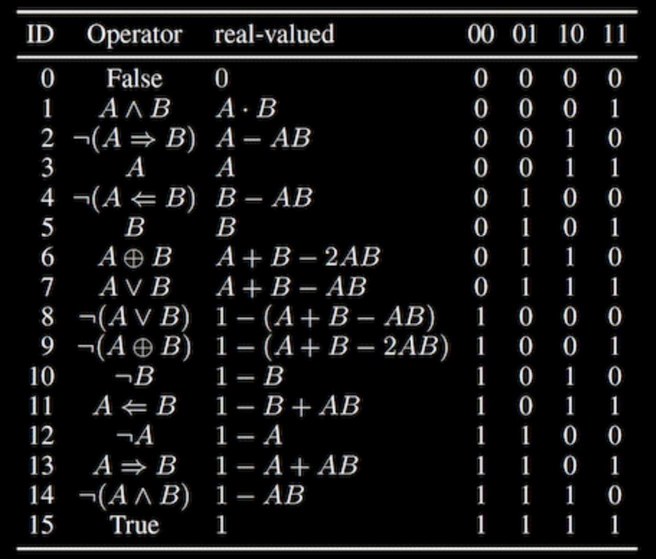
A very good overview paper

Play around with torchlogix

Get in touch if you have questions: liv.helen.vage@cern.ch

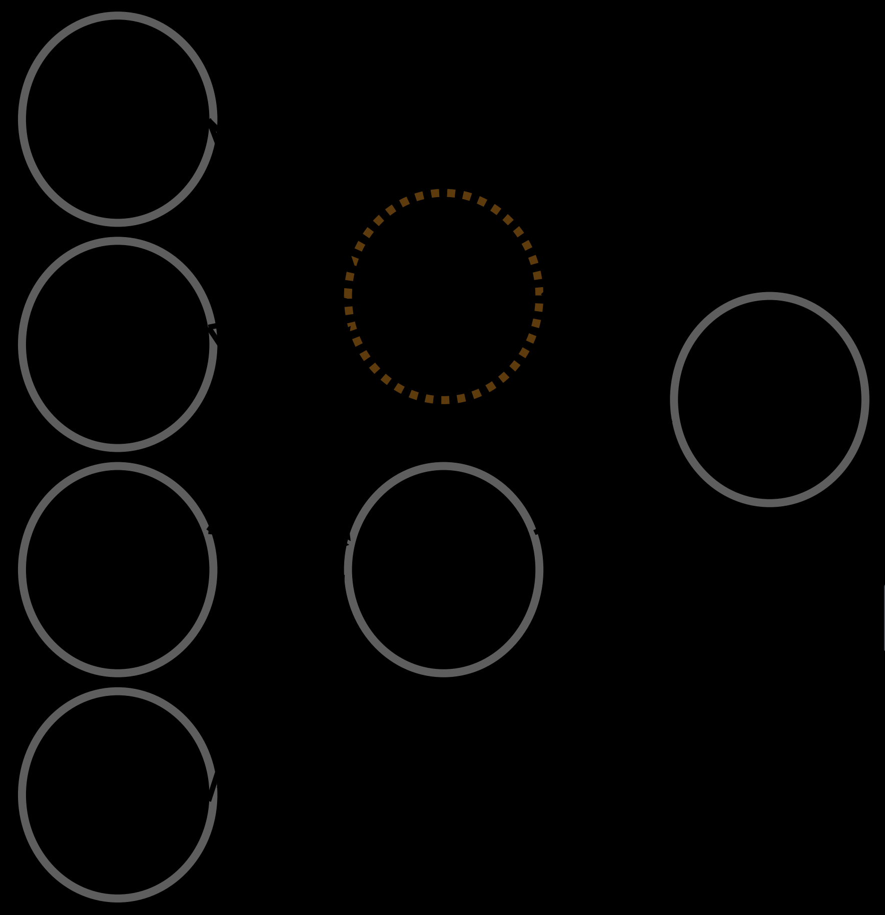
Backup

MLP vs difflog MLP



ID	Operator	real-valued	00	01	10	11
0	False	0	0	0	0	0
1	$A \wedge B$	$A \cdot B$	0	0	0	1
2	$\neg(A \Rightarrow B)$	$A - AB$	0	0	1	0
3	A	A	0	0	1	1
4	$\neg(A \Leftarrow B)$	$B - AB$	0	1	0	0
5	B	B	0	1	0	1
6	$A \oplus B$	$A + B - 2AB$	0	1	1	0
7	$A \vee B$	$A + B - AB$	0	1	1	1
8	$\neg(A \vee B)$	$1 - (A + B - AB)$	1	0	0	0
9	$\neg(A \oplus B)$	$1 - (A + B - 2AB)$	1	0	0	1
10	$\neg B$	$1 - B$	1	0	1	0
11	$A \Leftarrow B$	$1 - B + AB$	1	0	1	1
12	$\neg A$	$1 - A$	1	1	0	0
13	$A \Rightarrow B$	$1 - A + AB$	1	1	0	1
14	$\neg(A \wedge B)$	$1 - AB$	1	1	1	0
15	True	1	1	1	1	1

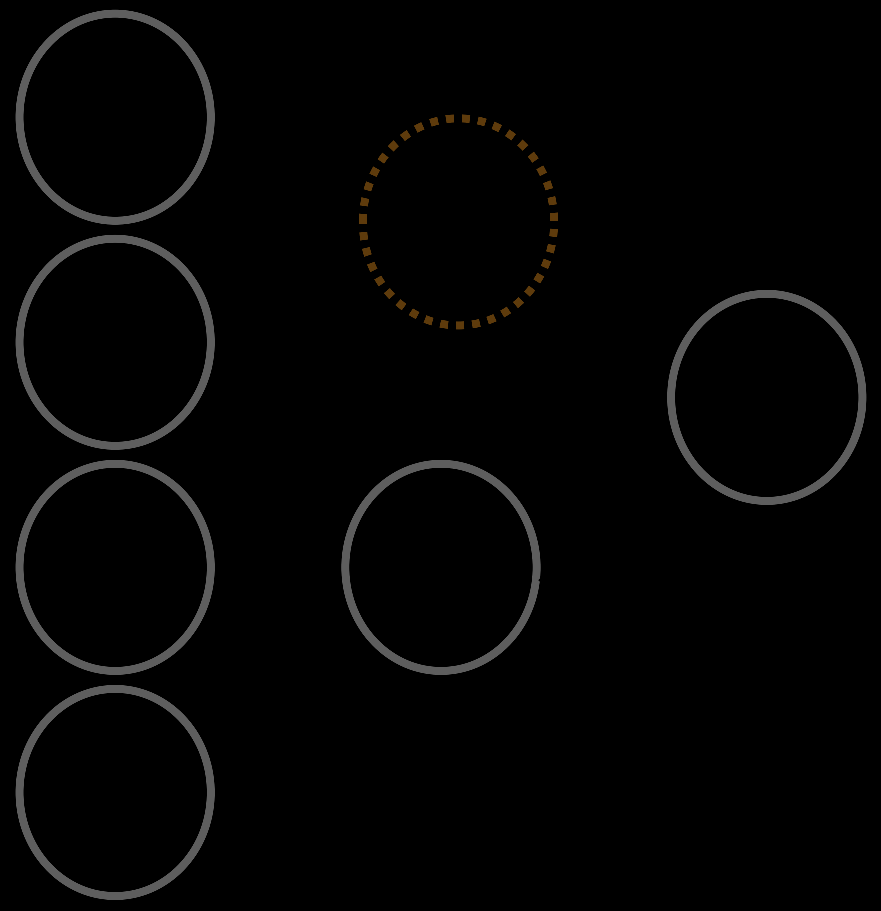
Normal neural net



ReLu

$$\begin{array}{cc} x & w \\ \begin{bmatrix} 121 \\ 244 \\ 31 \\ 121 \end{bmatrix} & \begin{bmatrix} 0.1 & -0.5 & 0.9 & 2 \end{bmatrix} \end{array}$$

Logic gate neural net



X	f(x1, x2)	w
		-0.1
0.21	0	0.2
0.63	0.21 * 0.63	2
	0.21 - 0.21 * 0.63	….
	….	

3%	10%	40 % …
false	and	or

Fully connected

Matrix multiplication in nodes

Same in training as inference

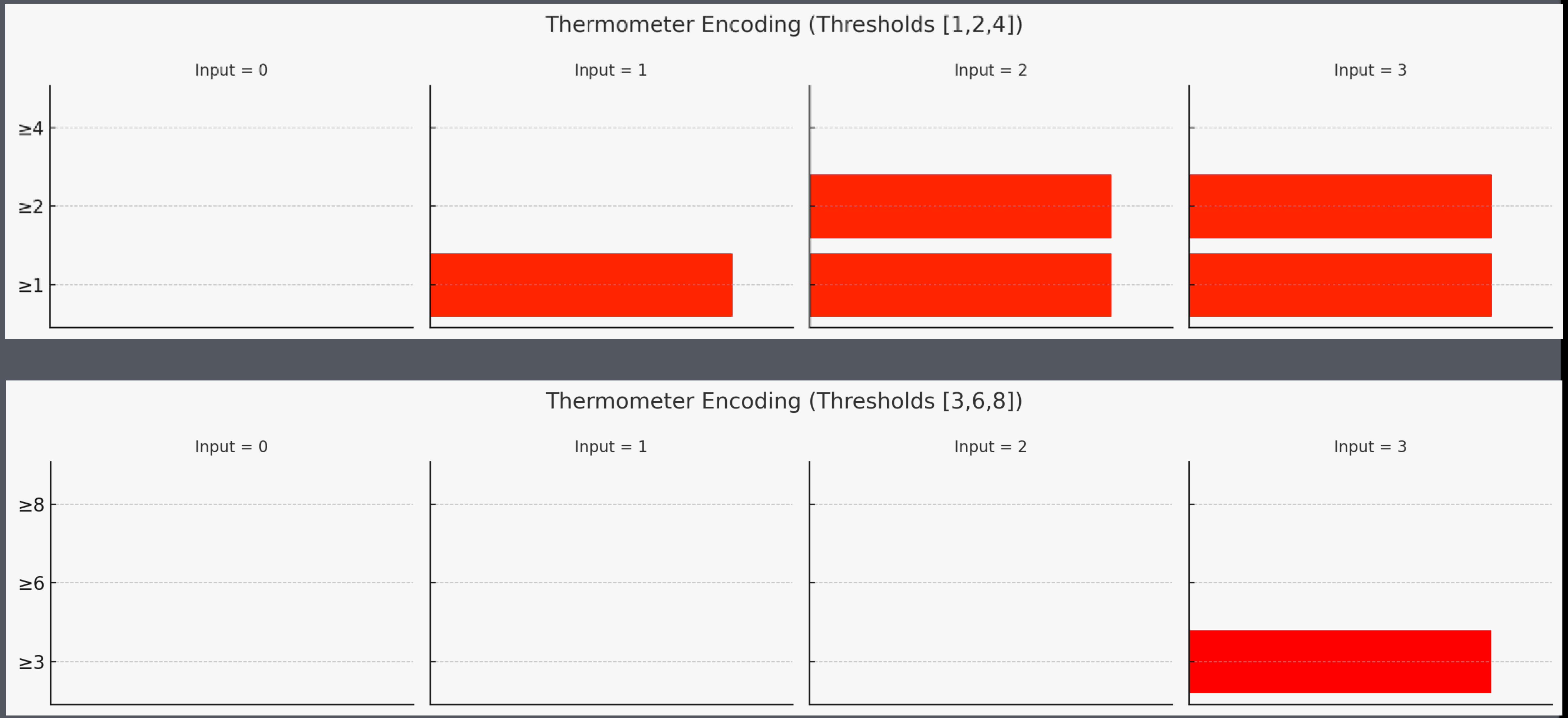
Randomly connected; each node has two inputs

Nodes evaluate logic gates

At inference, inputs are binary and only use most probable logic gate for each node

No matrix multiplication, binary logic can be simplified by compiler, binary computations faster

Thermometer thresholding



We need to convert continuous numbers to binary

We do this based on thresholds, as implemented in [4]

We learn the bin edges as we train

each bin checks whether the number ≥ threshold

1 if it is, 0 otherwise

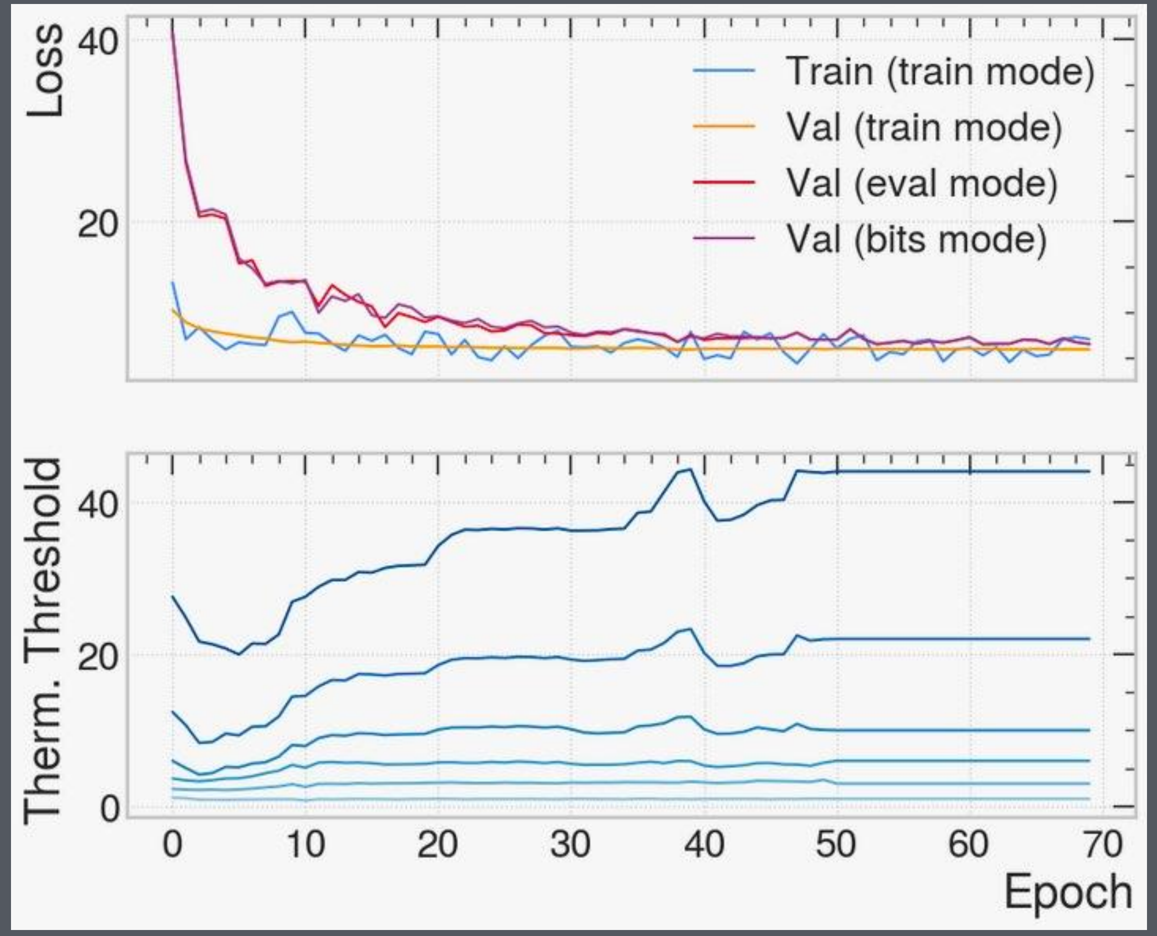
Our representation matters

the best edges will depend on the problem

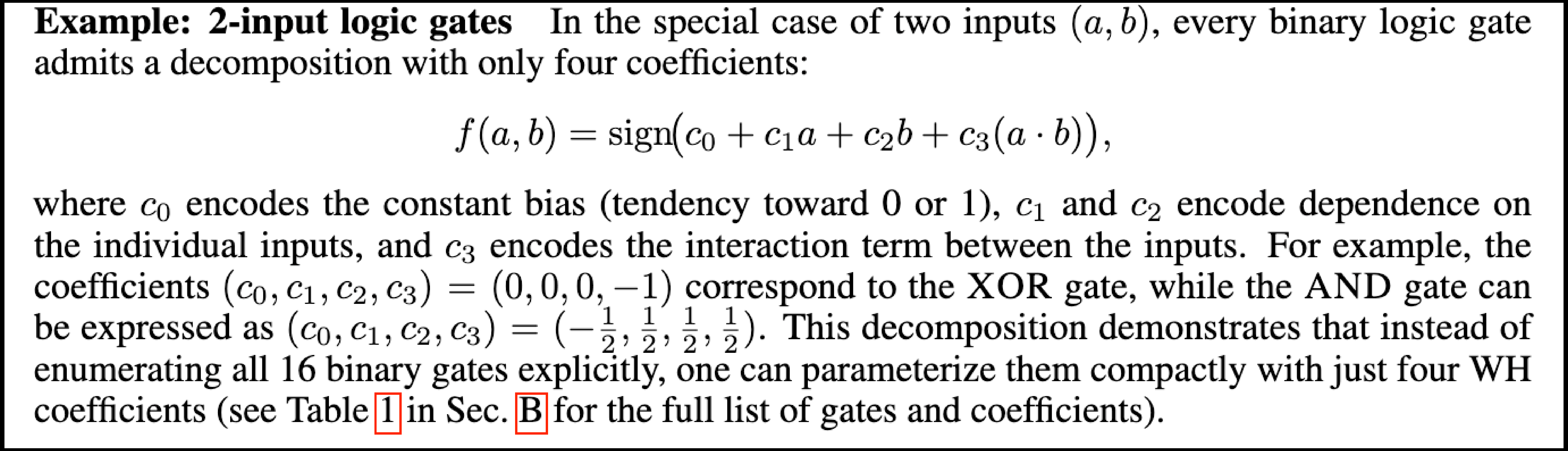
In our case, we let it change, then freeze it

we use a 6 bit representation

one of the bins settles on a high threshold - likely to capture high anomaly scores



WARP-LUTS



Example: 2-input logic gates In the special case of two inputs (a, b), every binary logic gate admits a decomposition with only four coefficients:

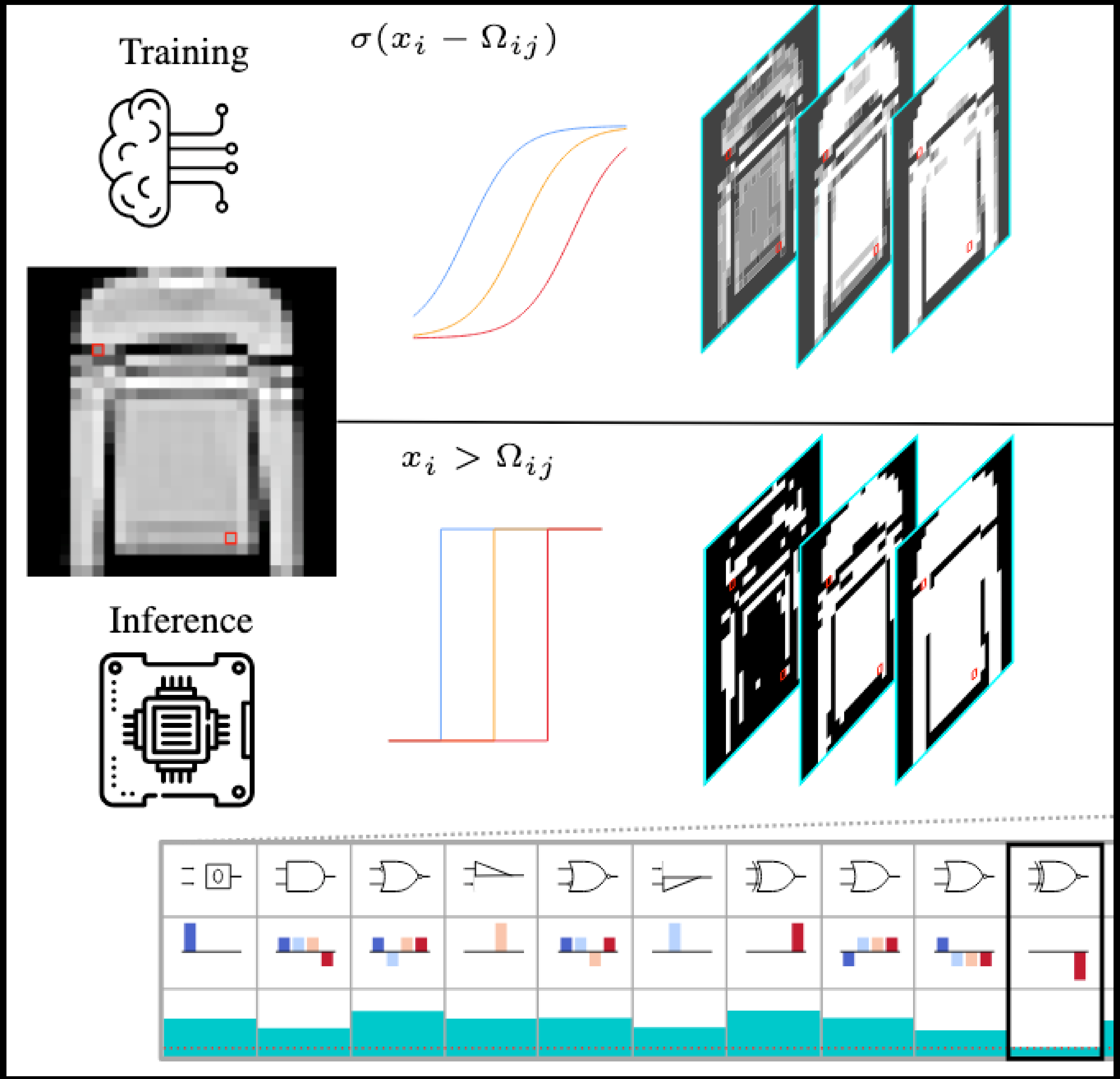
$$f(a, b) = \text{sign}\big(c_0 + c_1 a + c_2 b + c_3 (a \cdot b)\big),$$

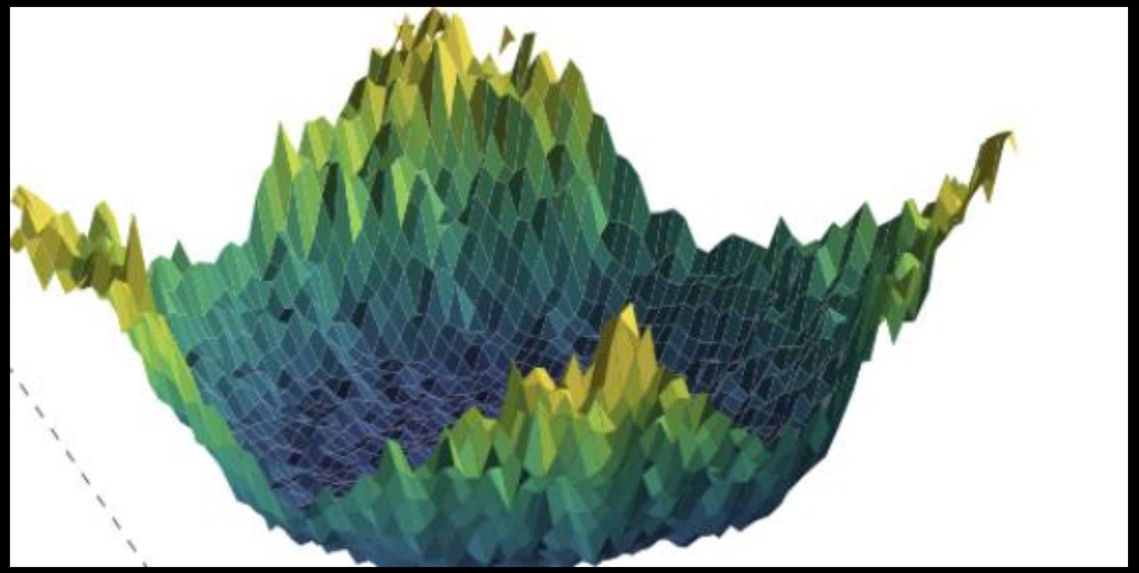
where c_0 encodes the constant bias (tendency toward 0 or 1), c_1 and c_2 encode dependence on the individual inputs, and c_3 encodes the interaction term between the inputs. For example, the coefficients $(c_0, c_1, c_2, c_3) = (0, 0, 0, -1)$ correspond to the XOR gate, while the AND gate can be expressed as $(c_0, c_1, c_2, c_3) = (-\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$. This decomposition demonstrates that instead of enumerating all 16 binary gates explicitly, one can parameterize them compactly with just four WH coefficients (see Table 1 in Sec. B for the full list of gates and coefficients).

Straight through estimators

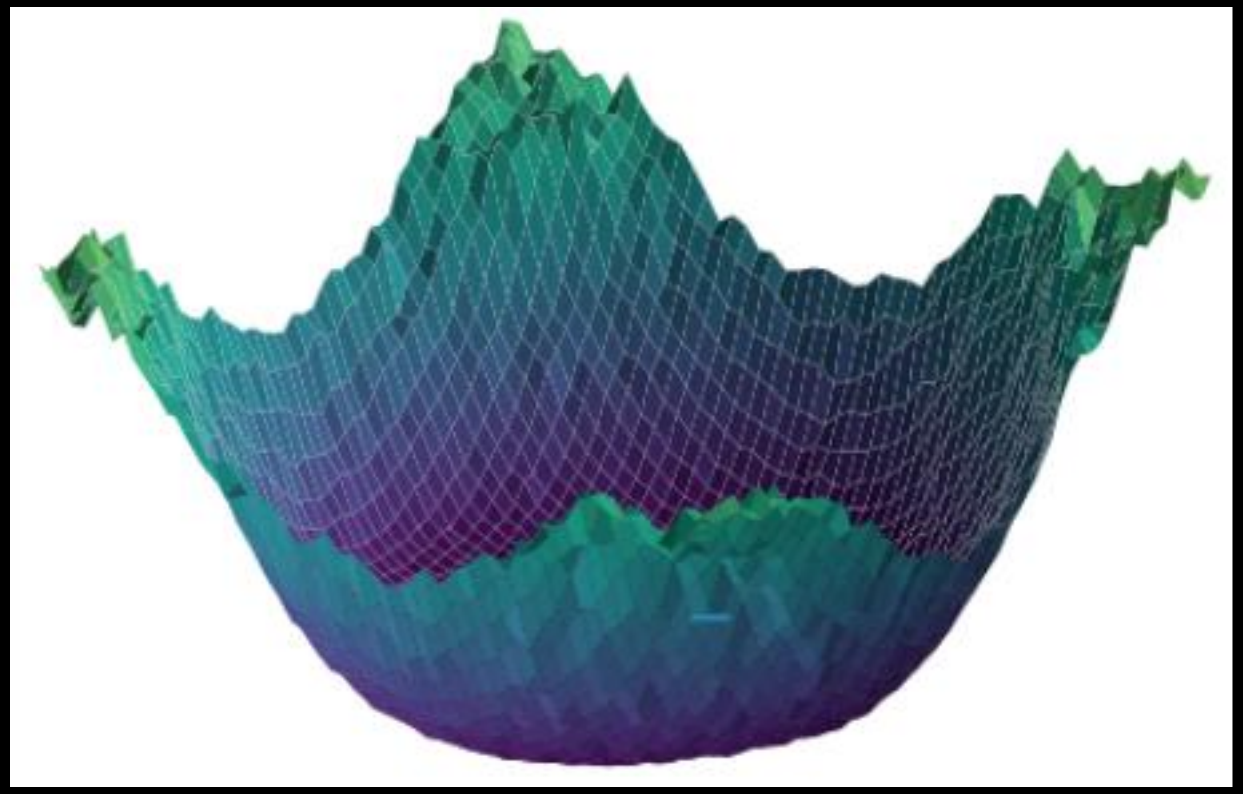
```
Forward:  w_binary = sign(w_real)     # discrete

Backward: ∂L/∂w_real = ∂L/∂w_binary   # pretend sign() was identity
```

Discretisation gap





DLGN



DLGN with Gumbel noise

Reducing discretisation gap

Scaling issue with LUTs

If you have a computation like:

Y = A AND B AND C AND D AND E AND F AND G AND H

Most LUTs take 4 or 6 inputs, so it can’t all be contained in one LUT ->

$(A \wedge B \wedge C \wedge D \wedge E \wedge F) \rightarrow \text{LUT1}$

$(G \wedge H) \rightarrow \text{LUT2}$

$(\text{LUT1} \wedge \text{LUT2}) \rightarrow \text{LUT3}$

Most LUTs take 4 or 6 inputs, so it can’t all be contained in one LUT -> superlunar scaling with input

Gradient issues with LNN

DLGNs are sensitive to the initialisation - part of the reason is that logic gates can symmetrically cancel out

Each logic gate reduces the gradient strength, so deep neural networks struggle

DWNs only estimate gradients - can scale badly with parameters and be slow to converge