ETH zürich

Systems@ETH Zürich

FPGA-accelerated Dense Linear Machine Learning: A Precision-Convergence Trade-off

Kaan Kara, Dan Alistarh, Gustavo Alonso, Onur Mutlu, Ce Zhang

What is the most efficient way of training dense linear models on an FPGA?

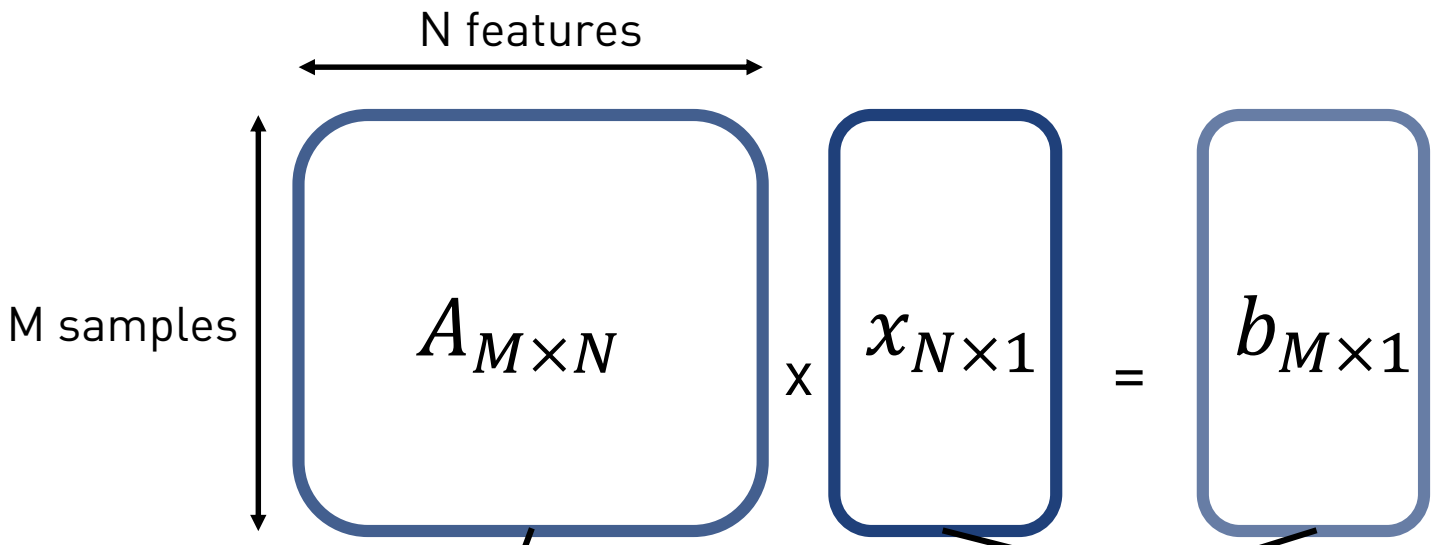
- FPGAs can handle floating-point, but they are certainly not the best choice. Yet!

  ⟶ We have tried: On par with a 10-core Xeon, not a clear win.

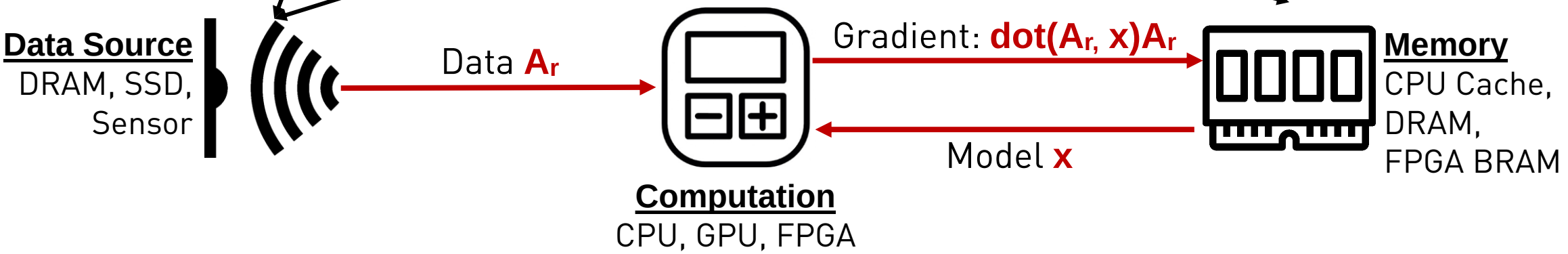
- How about some recent developments in the machine research?

Low-precision data —**In theory / leads to**→ EXACTLY the same end-result.

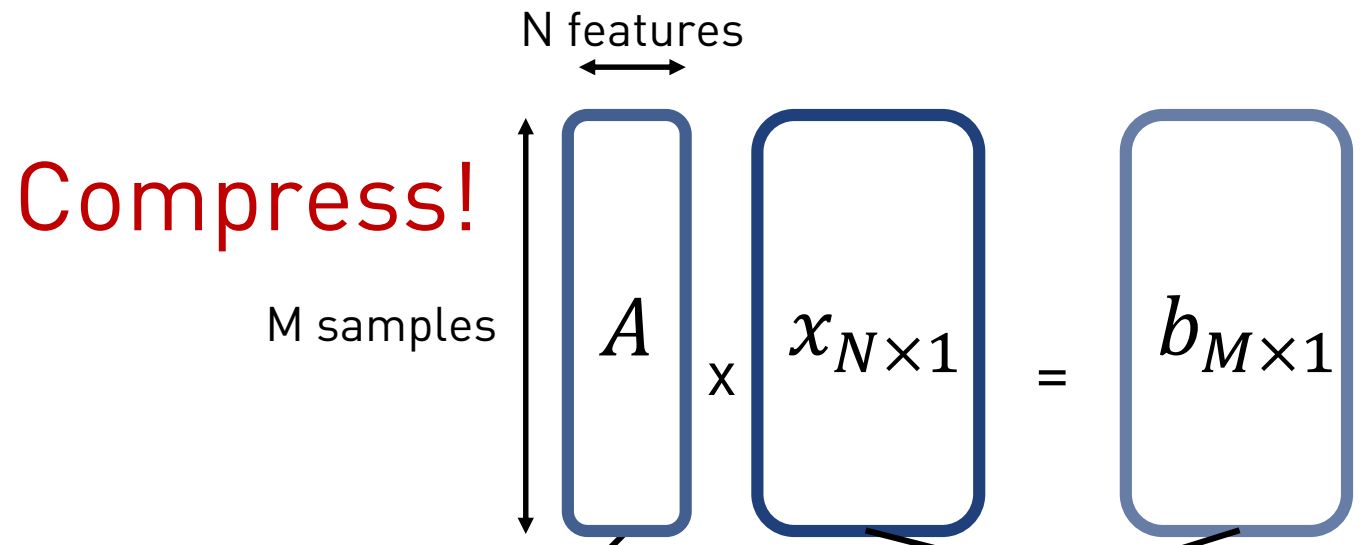
Stochastic Gradient Descent (SGD)



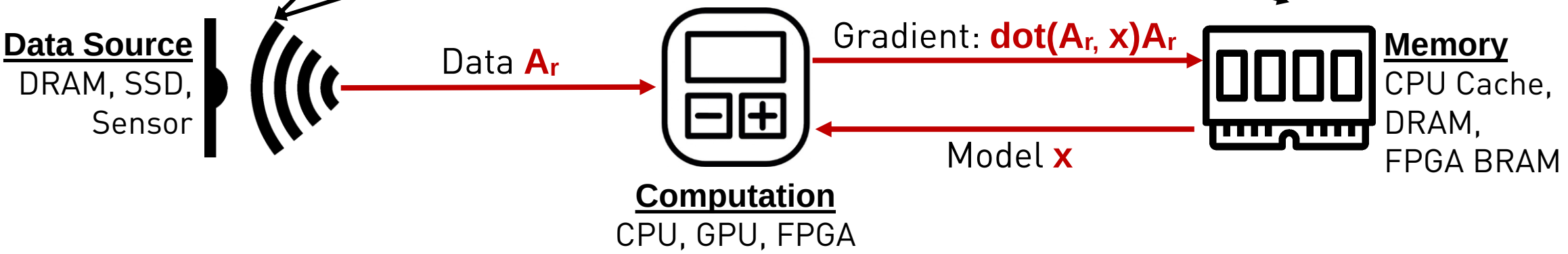
while not converged **do**
  for i from 1 to M **do**
    $g_i = (a_i \cdot x - b_i)a_i$
    $x \leftarrow x - \gamma g_i$
  end
end



Advantage of Low-Precision Data + FPGA



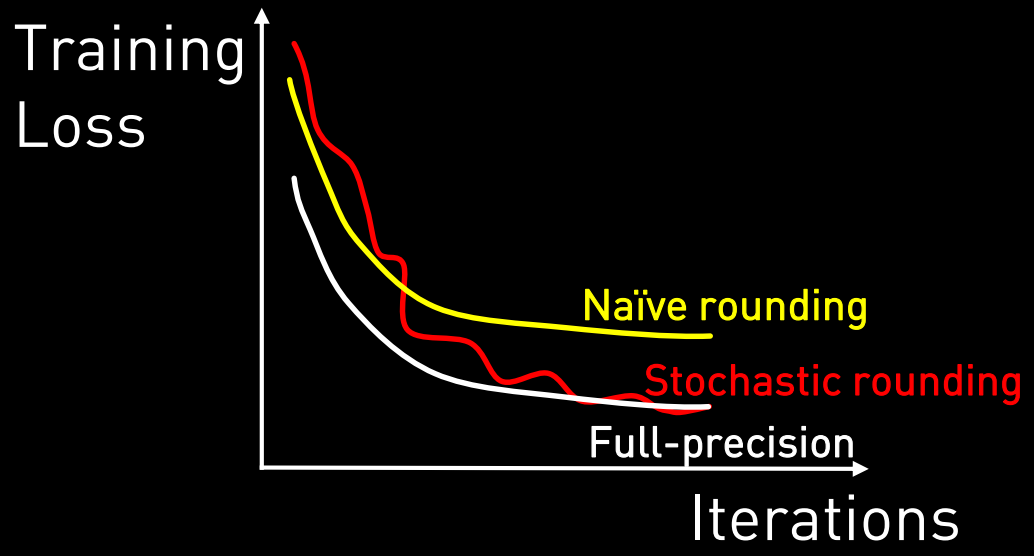

```
while not converged do
    for i from 1 to M do
        g_i = (a_i · x − b_i) a_i
        x ← x − γ g_i
    end
end
```

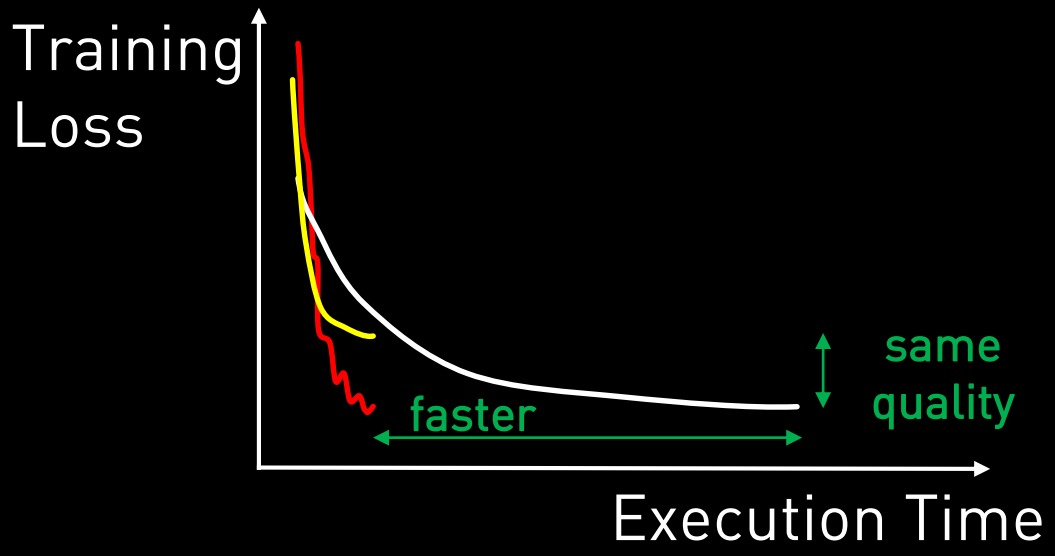


ETH zürich

Key Takeaway

- Using an FPGA, we can increase the **hardware efficiency** while maintaining the **statistical efficiency** of SGD for dense linear models.





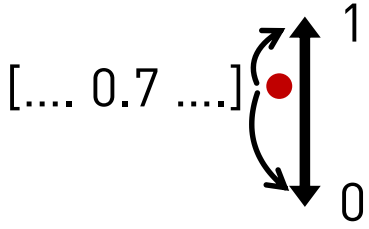
- In practice, the system opens up a multivariate trade-off space:

⟶ Data properties, FPGA implementation, SGD parameters…

Outline for the Rest…

1. Stochastic quantization. Data layout
2. Target platform: Intel Xeon+FPGA
3. Implementation: From 32-bit to 1-bit SGD on FPGA.
4. Evaluation: Main results. Side effects.
5. Conclusion

Stochastic Quantization [1]



Naive solution: Nearest rounding (=1)
=> Converge to a different solution.

Stochastic rounding: 0 with probability 0.3, 1 with probability 0.7
=> Expectation remains the same. Converge to the same solution.

$$\min_x \frac{1}{2}\sum_r (A_r \cdot x - b_r)^2 \xrightarrow{\text{Gradient}} \boldsymbol{g_i = (a_i \cdot x - b_i)a_i}$$

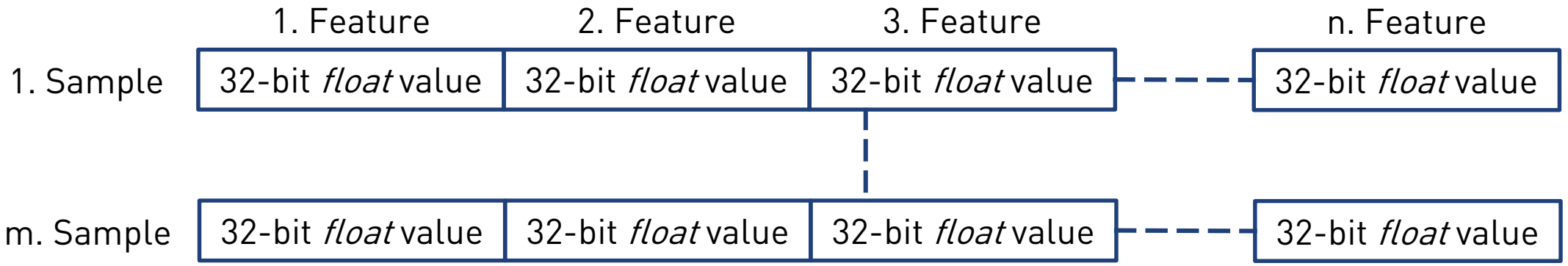
We need 2 independent samples

$$\boldsymbol{g_i = (Q_1(a_i) \cdot x - b_i)Q_2(a_i)}$$

...and each iteration of the algorithm needs fresh samples.

[1] H Zhang, K Kara, J Li, D Alistarh, J Liu, C Zhang: ZipML: An End-to-end Bitwise Framework for Dense Generalized Linear Models, arXiv:1611.05402

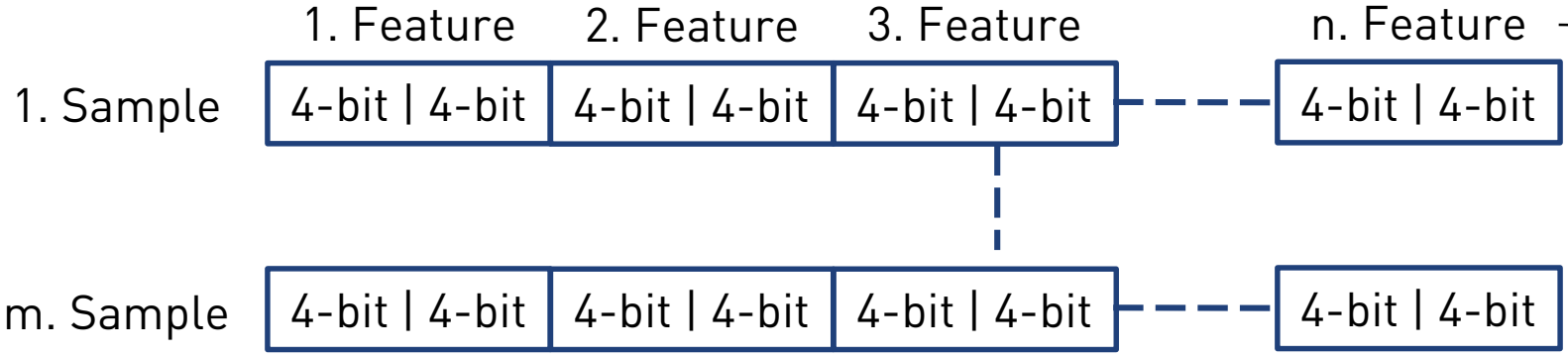
Before the FPGA: The Data Layout



Currently slow!

4x compression!

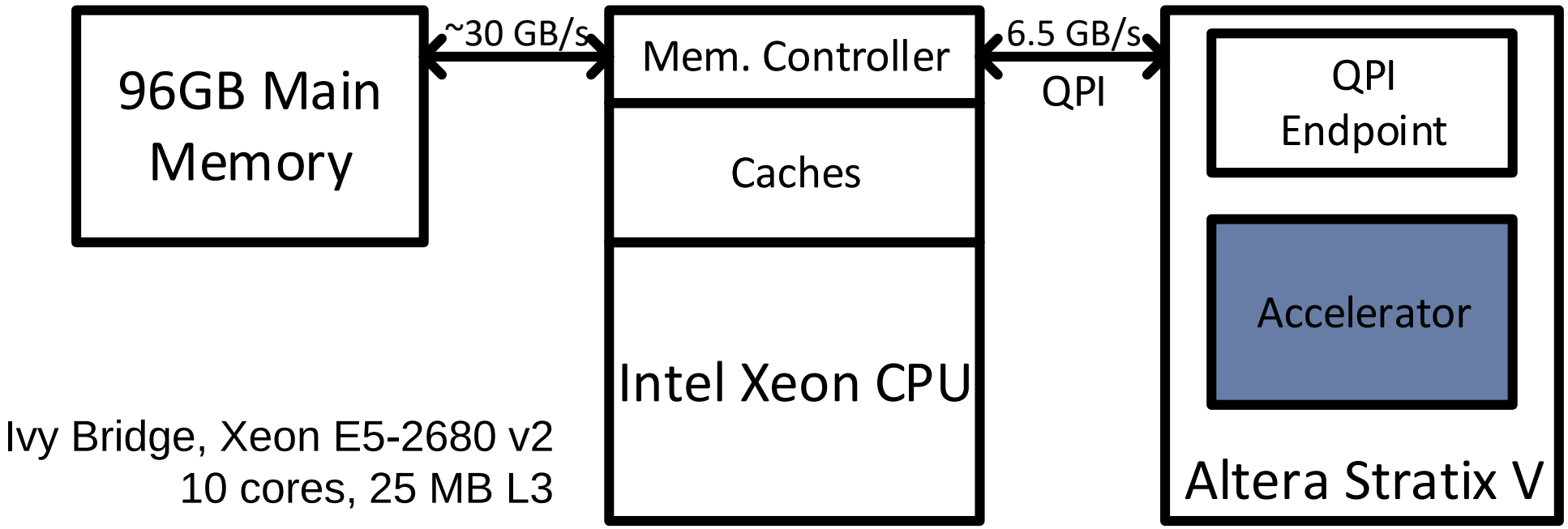
Quantize to 4-bit.
We need 2 independent quantizations!



1 quantization index

For each iteration
A new index!

Target Platform

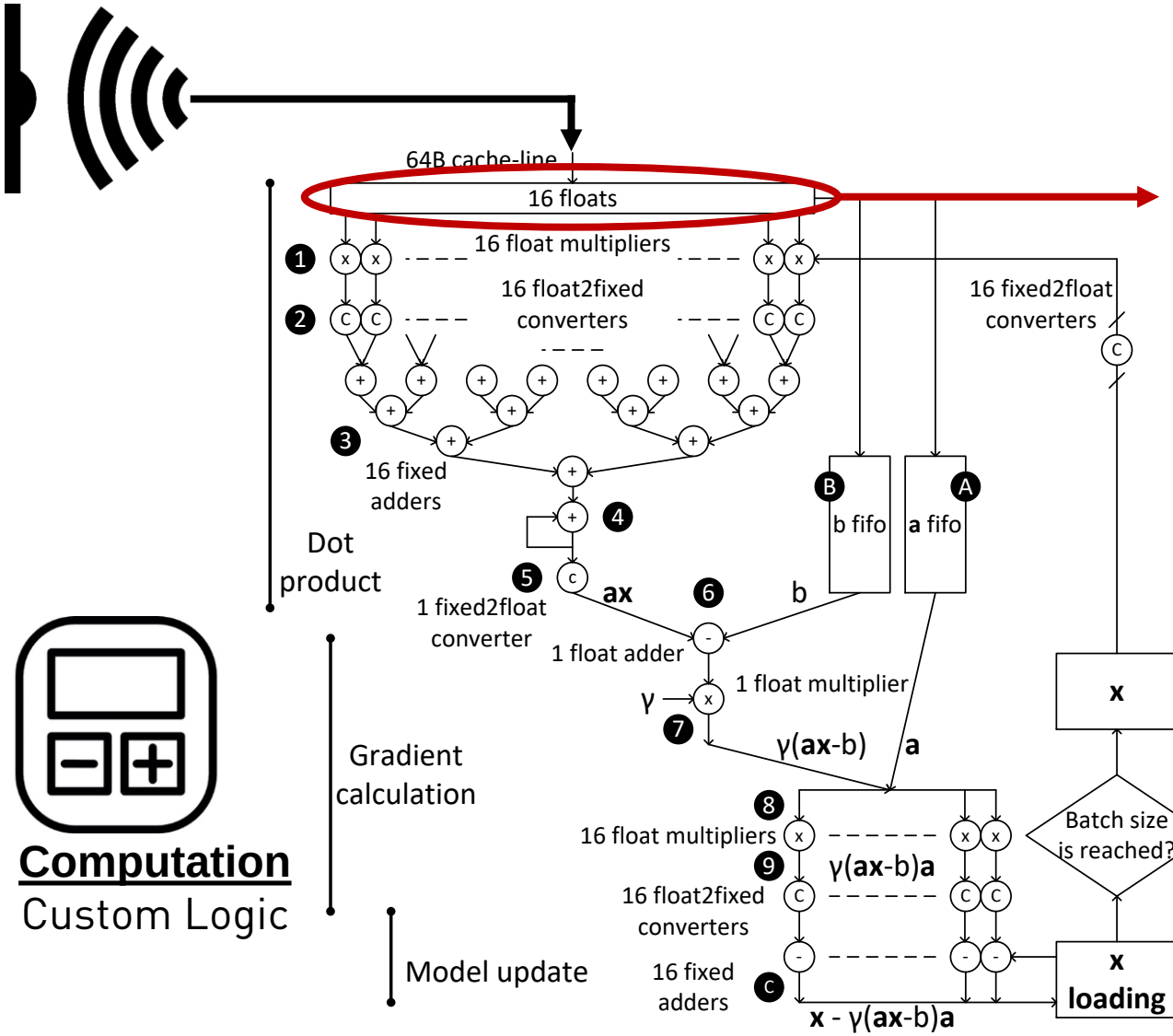


Ivy Bridge, Xeon E5-2680 v2
10 cores, 25 MB L3

Intel Xeon+FPGA

Full Precision SGD on FPGA

Data Source
DRAM, SSD,
Sensor



32-bit floating-point: 16 values

Processing rate: 12.8 GB/s

Computation
Custom Logic

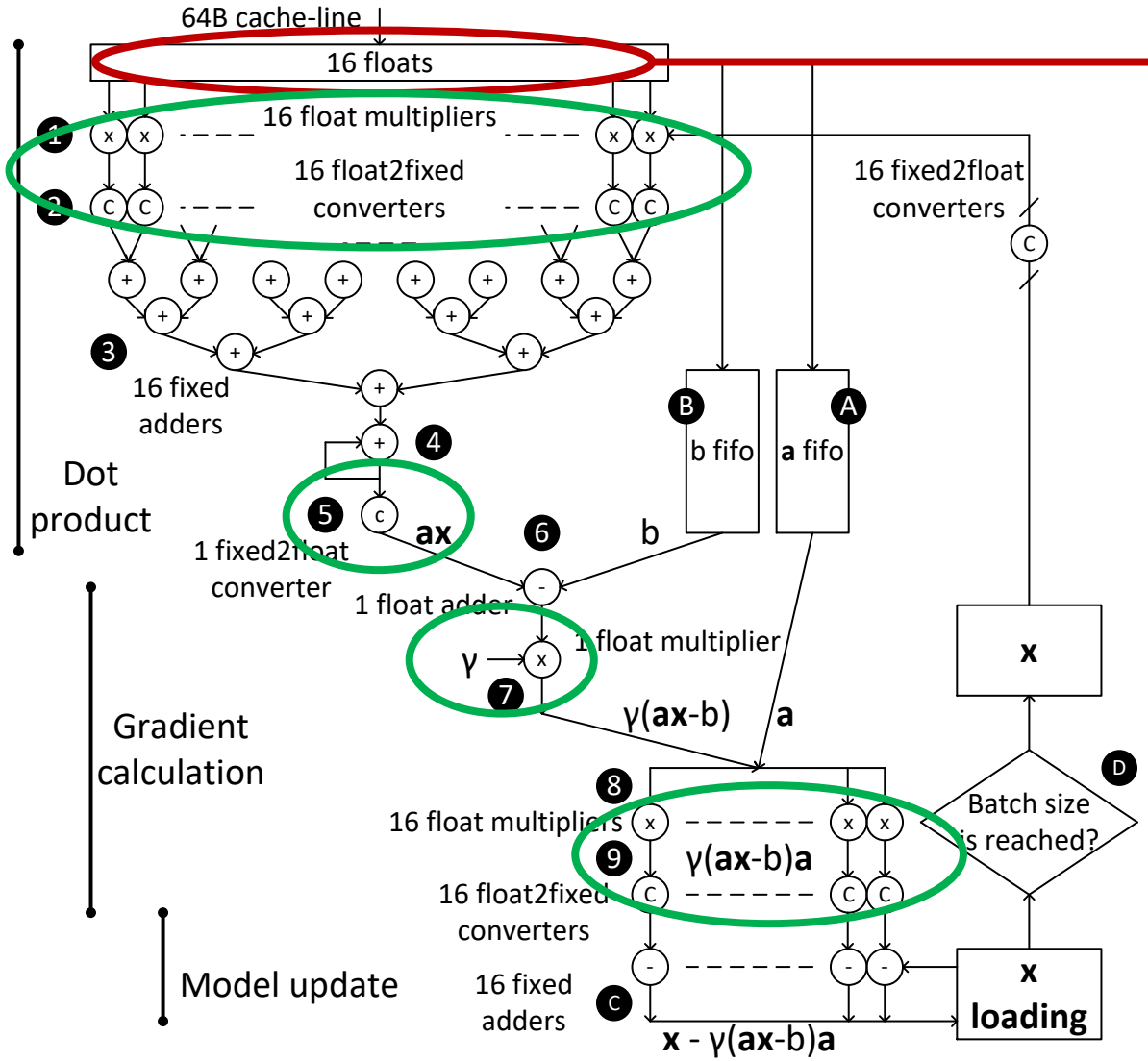
Storage Device
FPGA BRAM

ETH zürich

Scale out the design for low-precision data!

How can we increase the internal parallelism while maintaining the processing rate?

Challenge + Solution

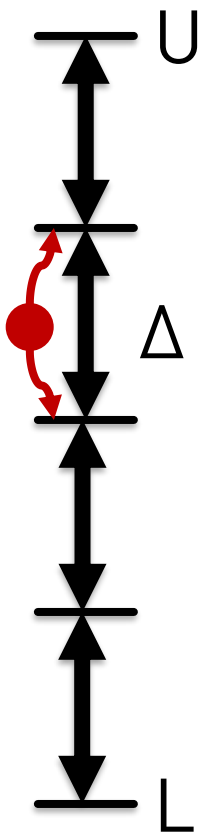


8-bit: 32 values Q8
4-bit: 64 values Q4
2-bit: 128 values Q2
1-bit: 256 values Q1

=> Scaling out is not trivial!

1) We can get rid of floating-point arithmetic.
2) We can further simplify integer arithmetic for lowest precision data.

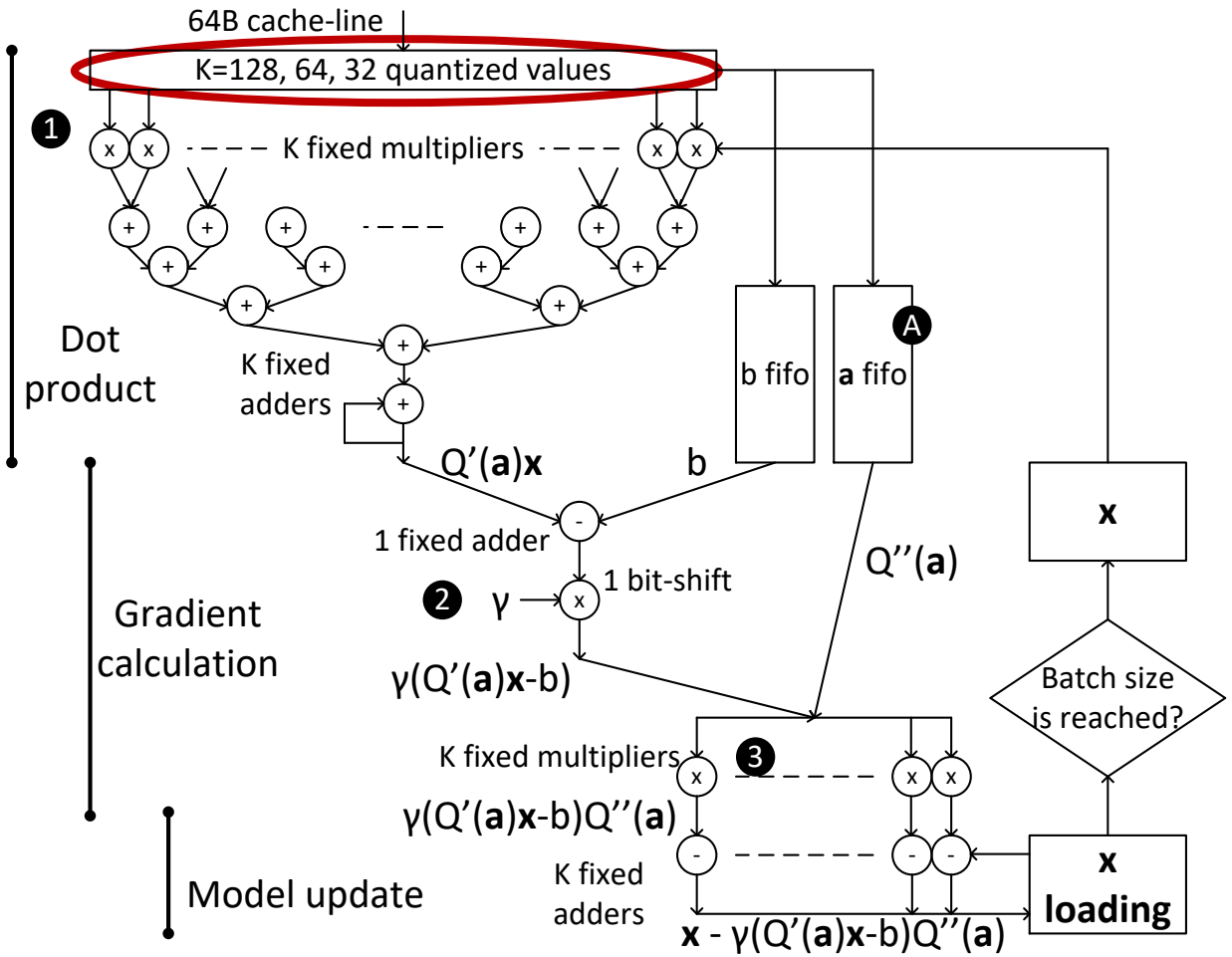
Trick 1: Selection of quantization levels



1. Select the lower and upper bound [L,U]
2. Select the size of the interval Δ

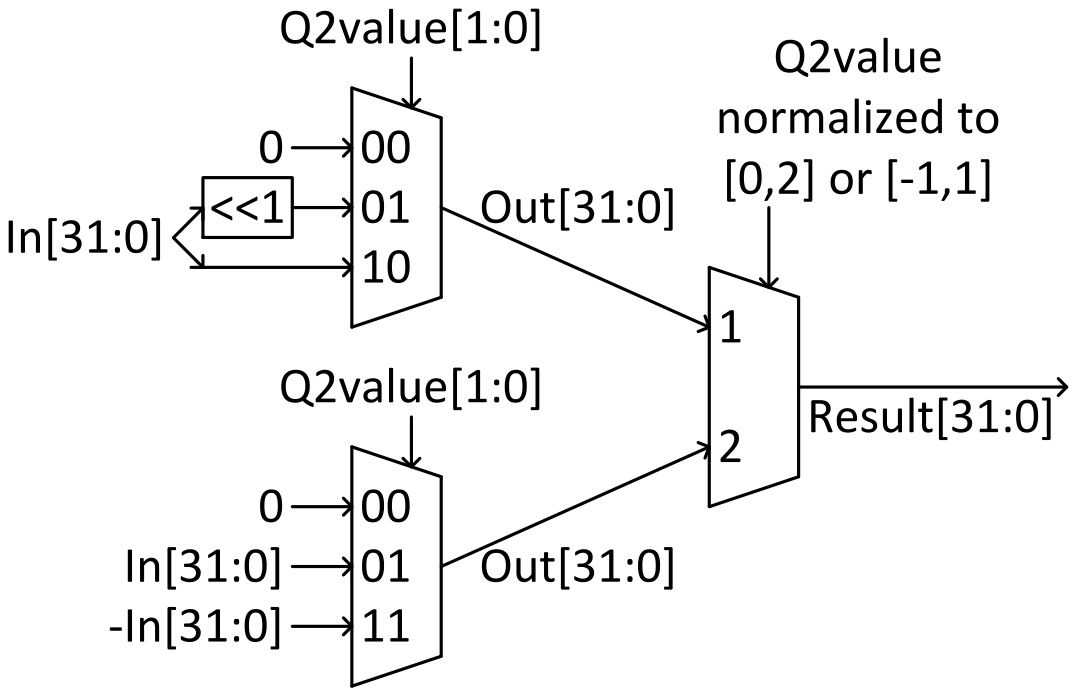
➡ All quantized values are integers!

This is enough for Q4 and Q8

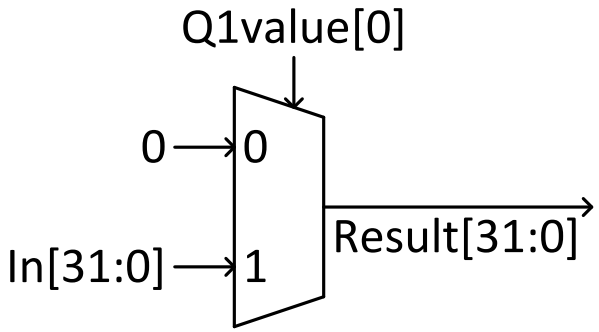


8-bit: 32 values Q8 ✔
4-bit: 64 values Q4 ✔
2-bit: 128 values Q2
1-bit: 256 values Q1

Trick 2: Implement Multiplication using Multiplexer

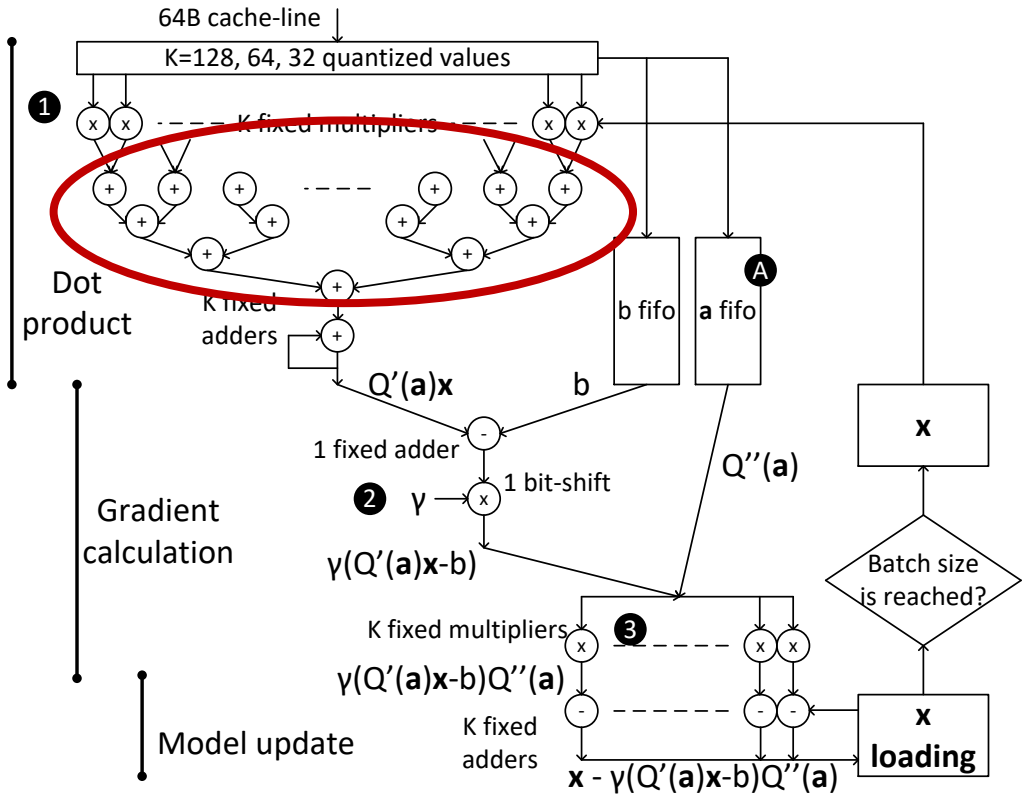


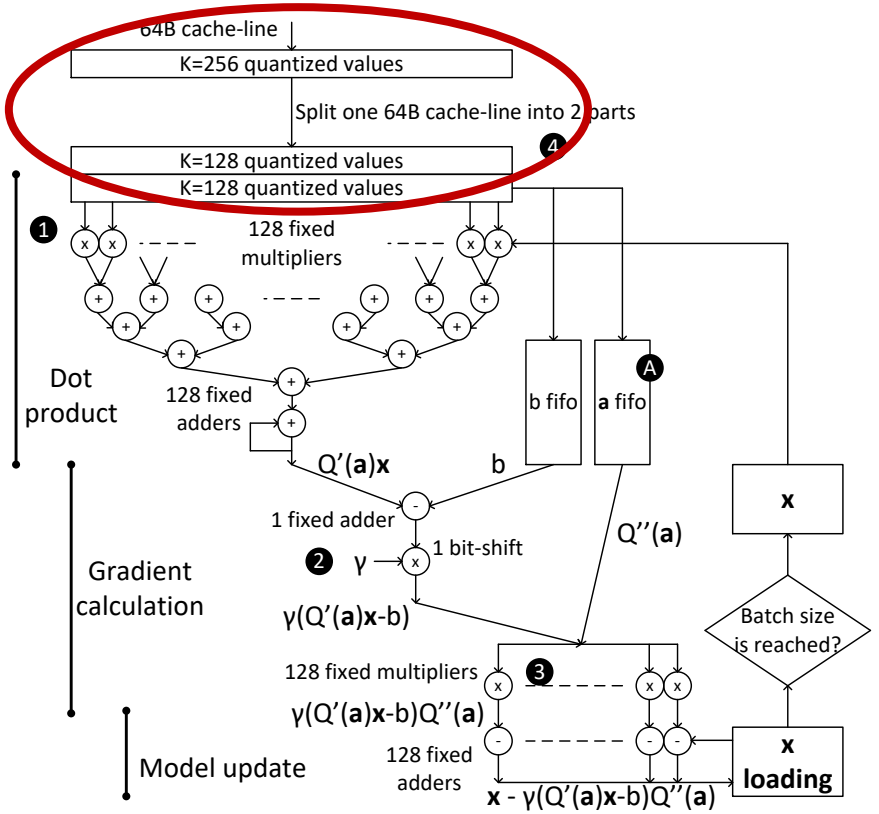
Q2 multiplier



Q1 multiplier

Support for all Qx





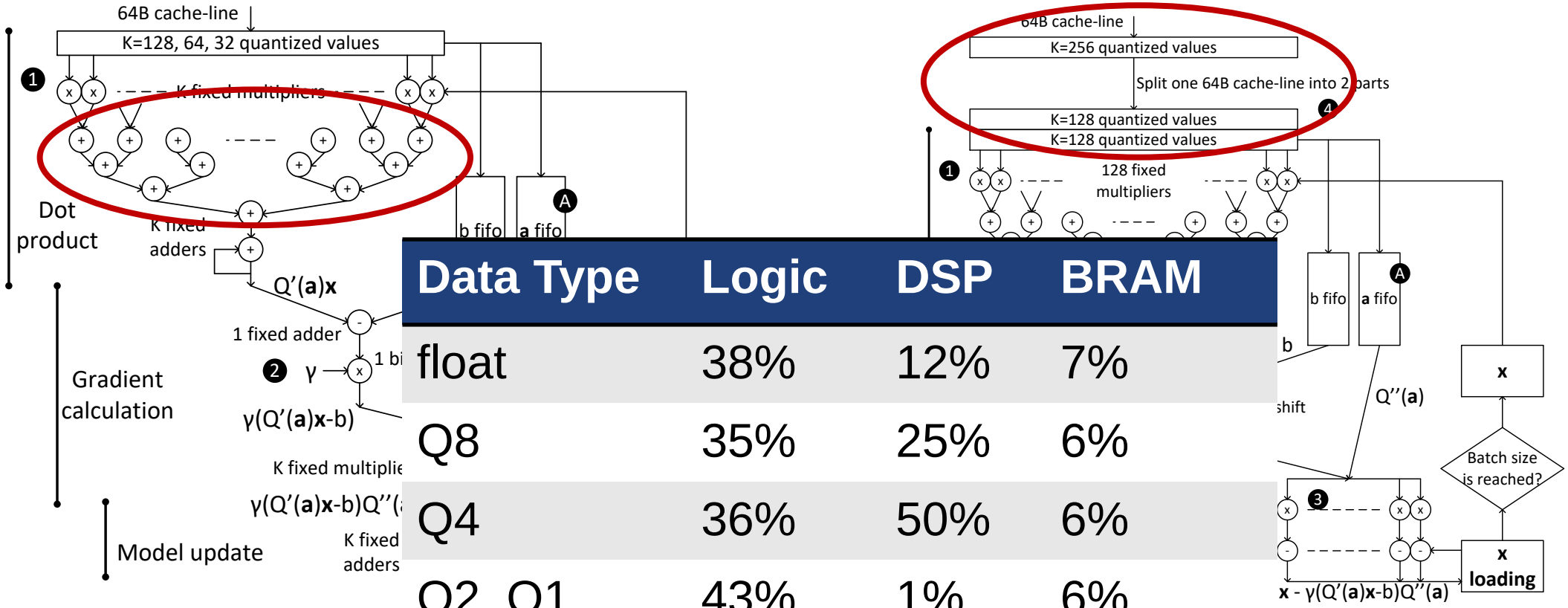
Circuit for Q8, Q4 and Q2 SGD

Processing rate: 12.8 GB/s

Circuit for Q1 SGD

Processing rate: 6.4 GB/s

Support for all Qx



Data Type	Logic	DSP	BRAM
float	38%	12%	7%
Q8	35%	25%	6%
Q4	36%	50%	6%
Q2, Q1	43%	1%	6%

Circuit for Q8, Q4 and Q2 SGD

Processing rate: 12.8 GB/s

Circuit for Q1 SGD

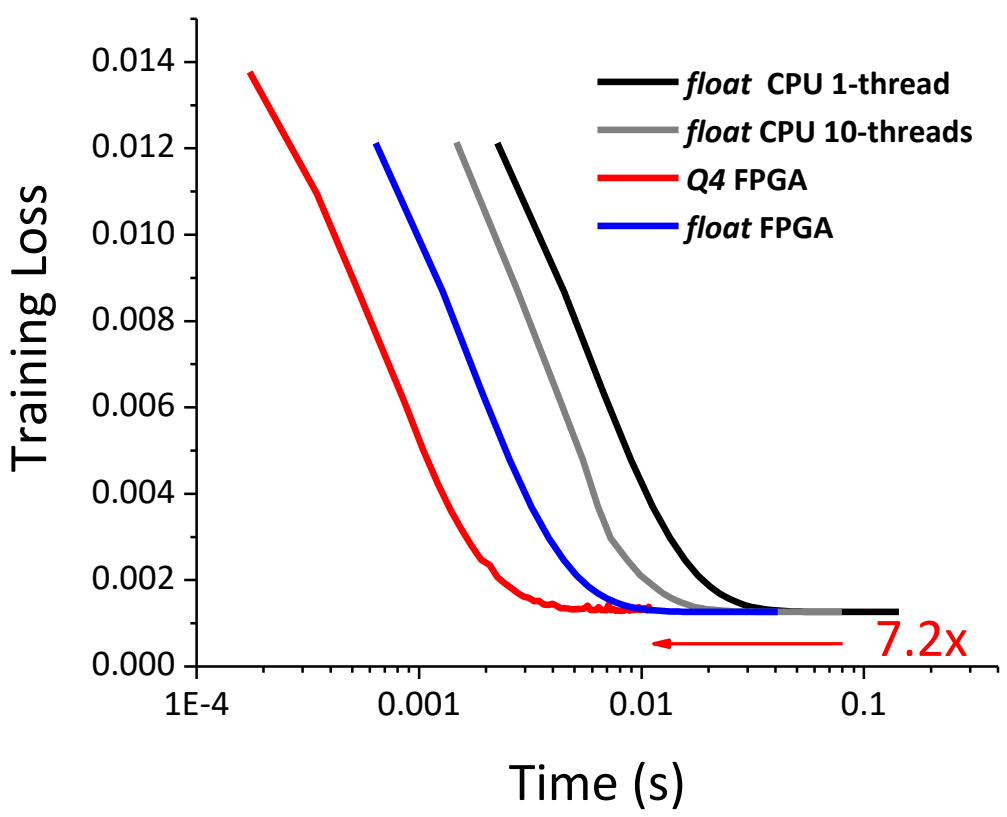
Processing rate: 6.4 GB/s

Data sets for evaluation

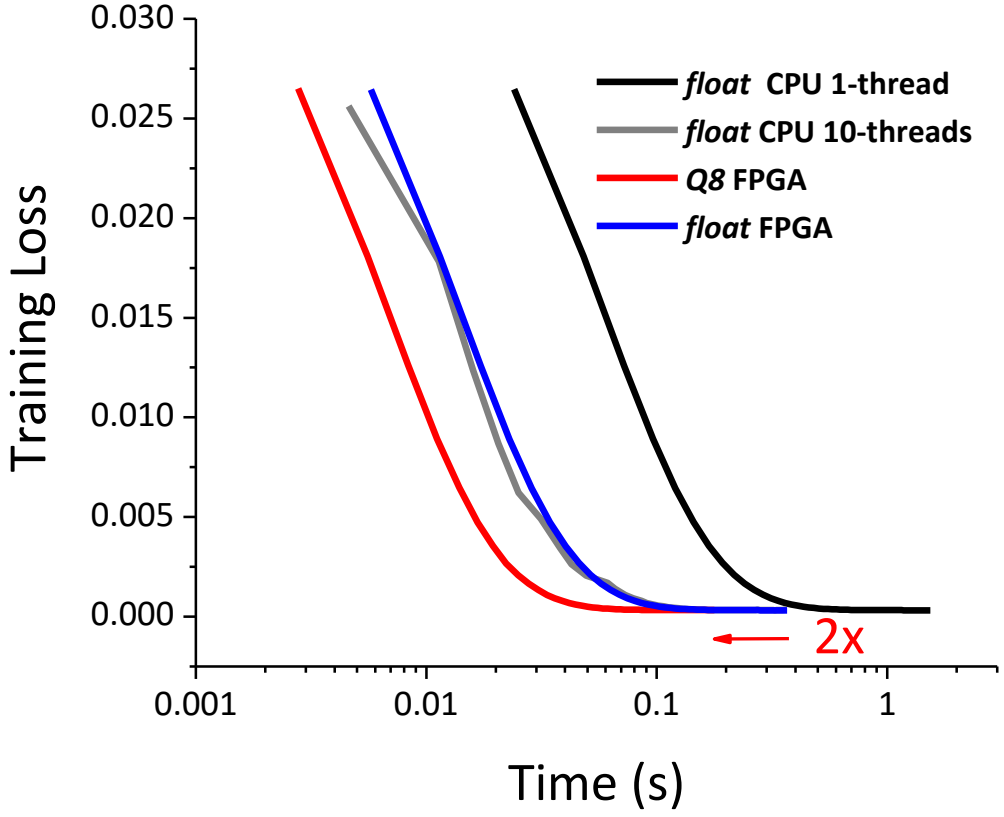
Name	Size	# Features	# Classes
MNIST	60,000	780	10
GISETTE	6,000	5,000	2
EPSILON	10,000	2,000	2
SYN100	10,000	100	Regression
SYN1000	10,000	1,000	Regression

Following the Intel legal guidelines on publishing performance numbers, we would like to make the reader aware that results in this publication were generated using preproduction hardware and software, and may not reflect the performance of production or future systems.

SGD Performance Improvement

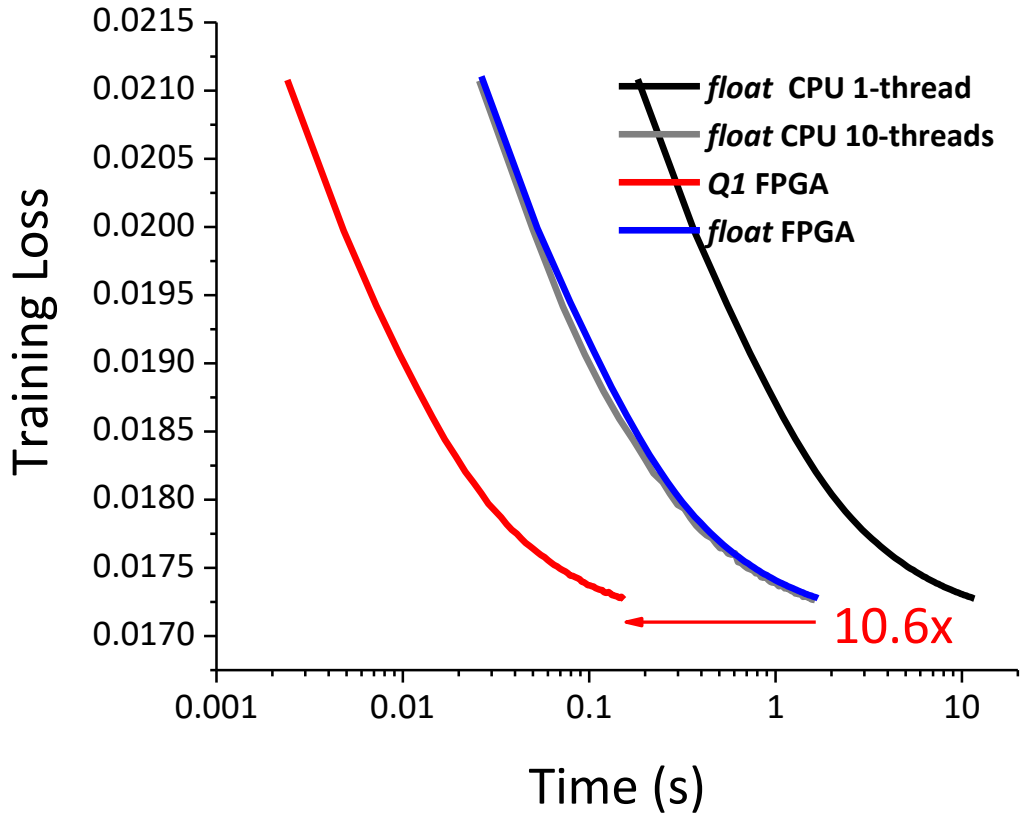


SGD on Synthetic100,
4-bit works

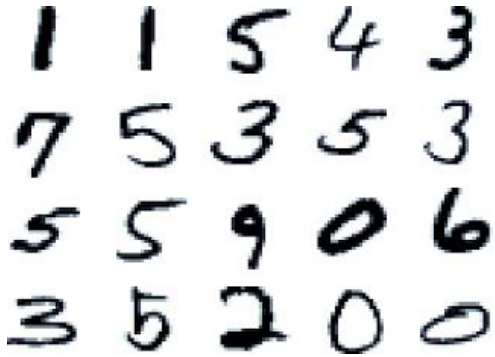


SGD on Synthetic1000,
8-bit works

1-bit also works on some data sets



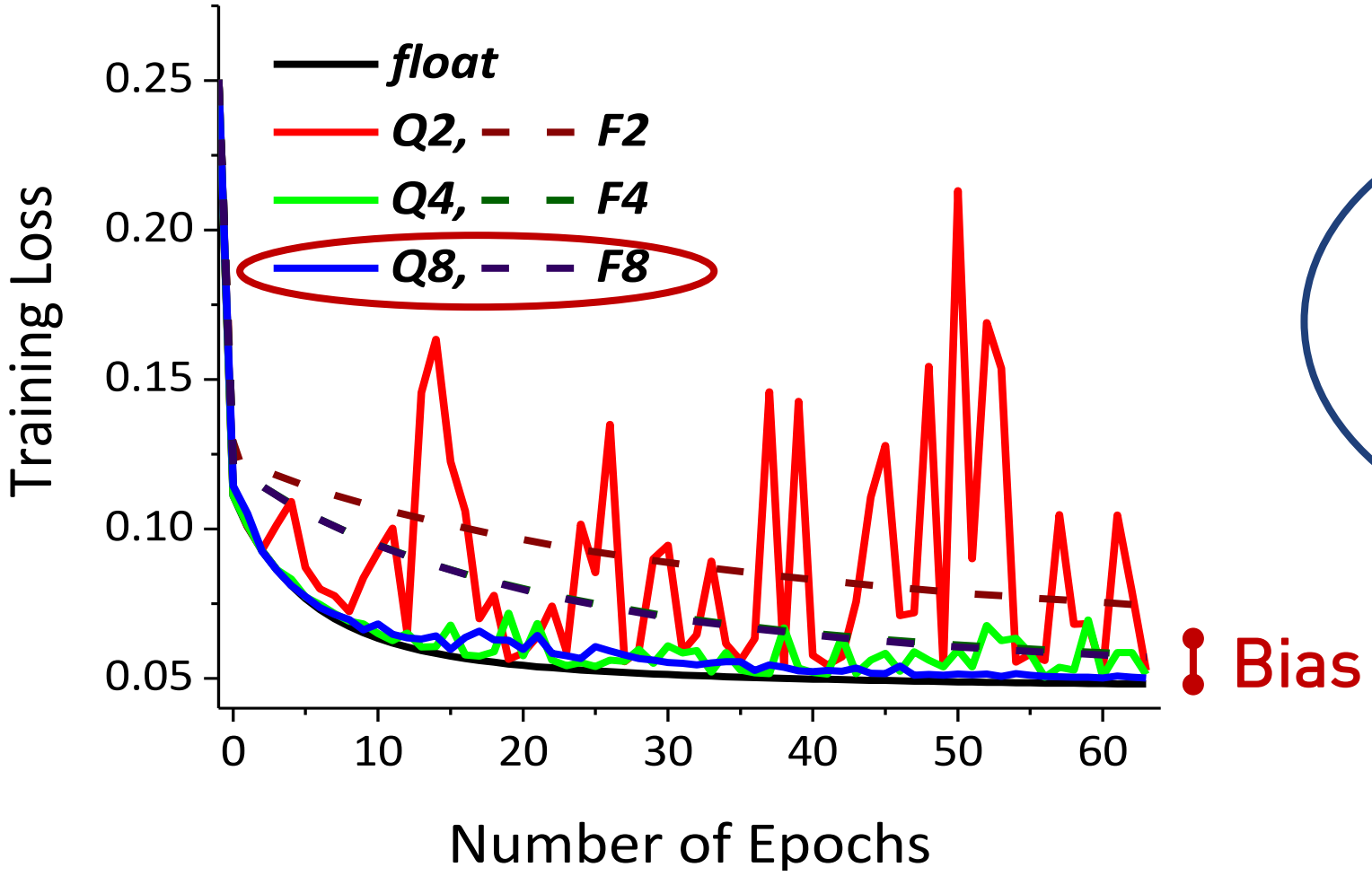
SGD on MNIST, Digit 7
1-bit works



MNIST
multi-classification accuracy

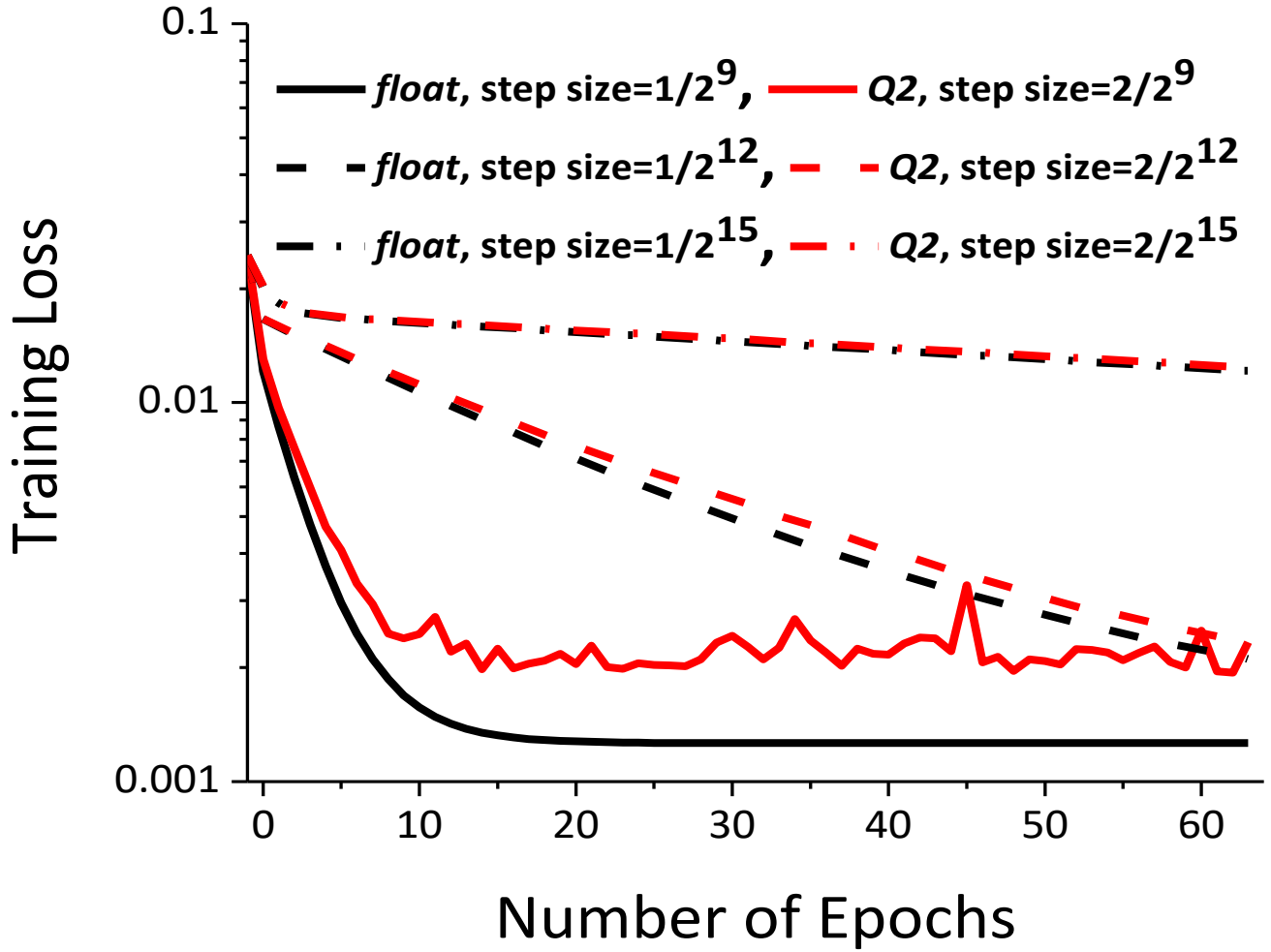
Precision	Accuracy	Training Time
float CPU-SGD	85.82%	19.04 s
1-bit FPGA-SGD	85.87%	2.41 s

Naïve Rounding vs. Stochastic Rounding



Stochastic quantization results in unbiased convergence.

Effect of Step Size



Large step size +
Full precision
vs.
Small step size +
Low precision

ETH zürich

Conclusion

Systems@ETH zürich

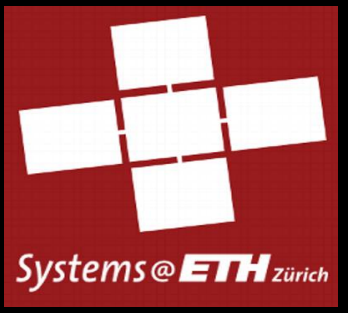
- Highly scalable, parametrizable FPGA-based stochastic gradient descent implementations for doing linear model training.
- Open source: www.systems.ethz.ch/fpga/ZipML_SGD

ZipML

Key Takeaways:

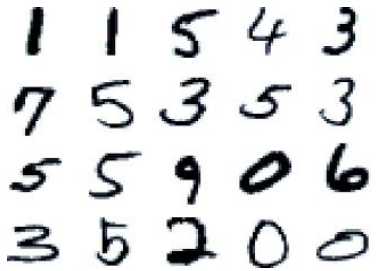
1. *The way to train linear models on FPGA should be through the usage of stochastically rounded, low-precision data.*
2. *Multivariate trade-off space: Precision vs. end-to-end runtime, convergence quality, design complexity, data and system properties.*

ETH zürich



Backup

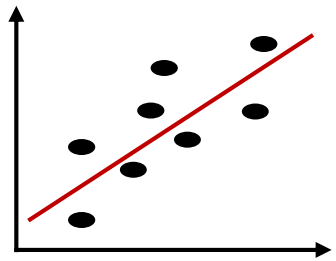
Why do dense linear models matter?



- Workhorse algorithm for regression and classification.
- Sparse, high dimensional data sets can be converted into dense ones.

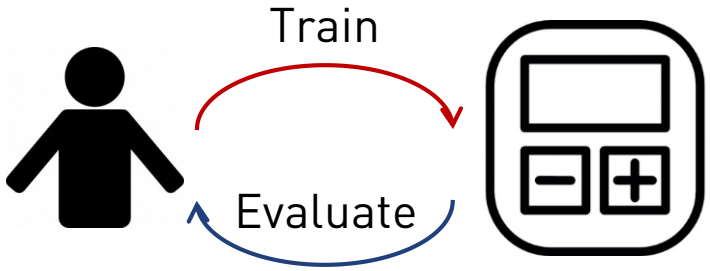
Transfer Learning

Random Features

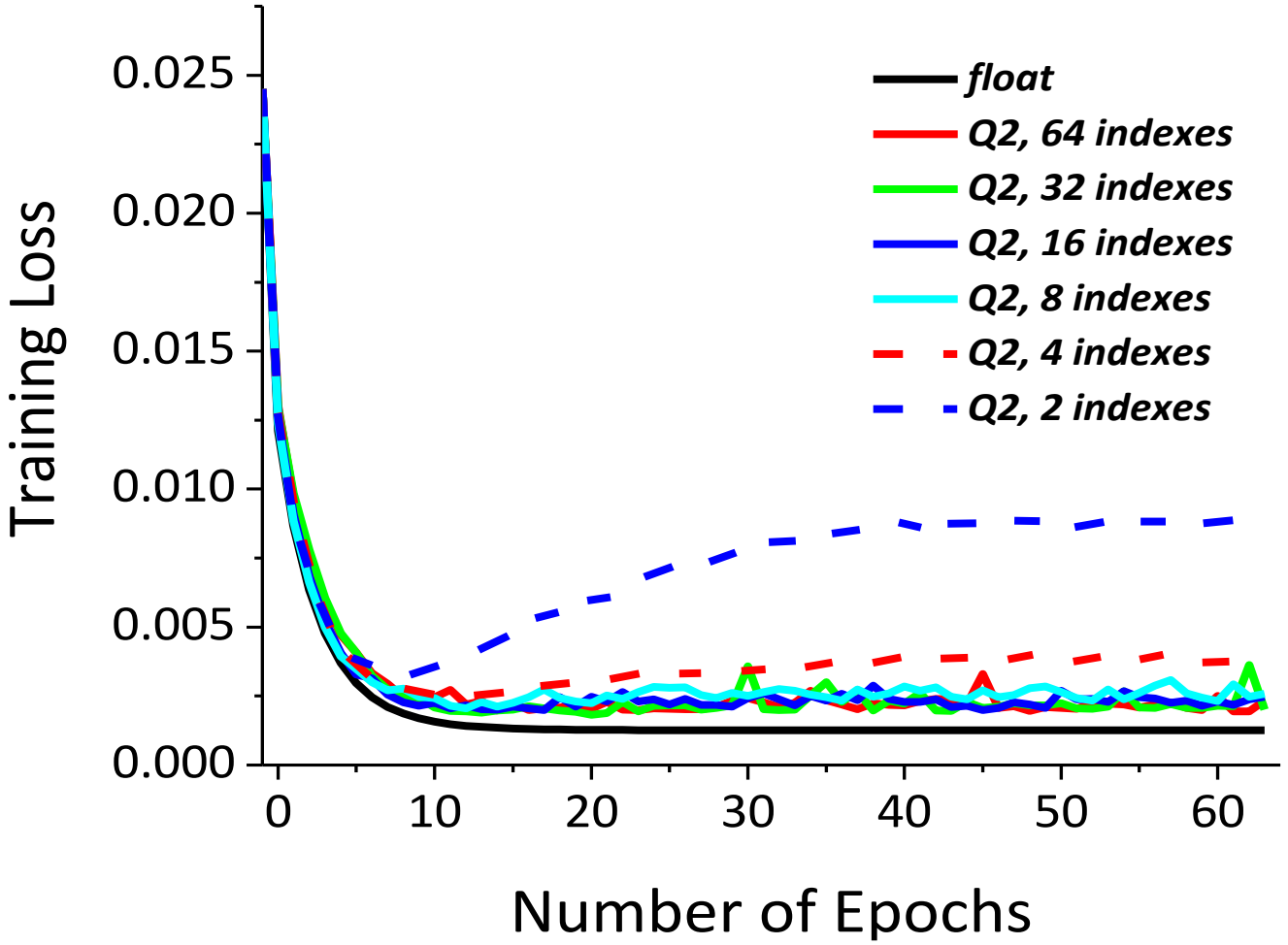


Why is training speed important?

- Often a human-in-the-loop process.
- Configuring parameters, selecting features, data dependency etc.



Effect of Reusing Indexes



In practice 8 indexes are enough!