Boosting Machine Learning Innovation: Computing Systems that Learn and Adapt

Ameer Abdelhadi, Jorge Albericio, Omar Awad, Ciaran Bannon, Alberto Delmas Lascorz, Isak Edo, Ali Hadi Zadeh, Patrick Judd, Mostafa Mahmoud, Milos Nikolic, Zissis Poulos, Eugene Sha, Sayeh Sharify, Kevin Siu, Dylan Stuart, Enrique Torres, Jiahui Wang

Tayler Hetherington, Natalie Enright Jerger,Tor Aamodt, Gennady Pekhimenko

Andreas Moshovos

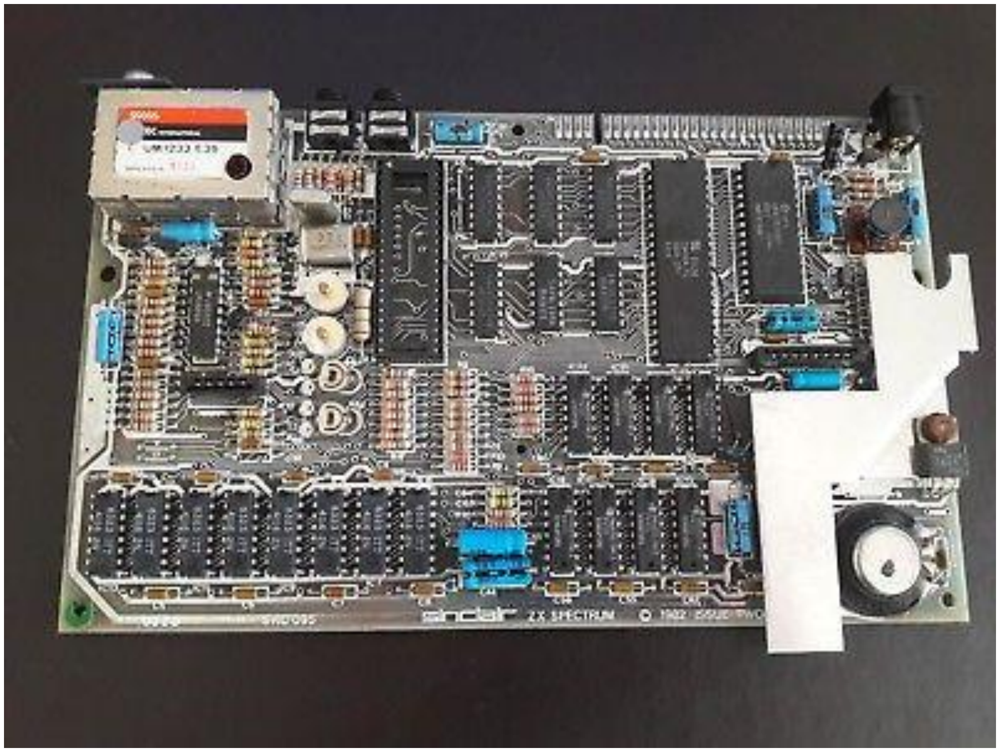
The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

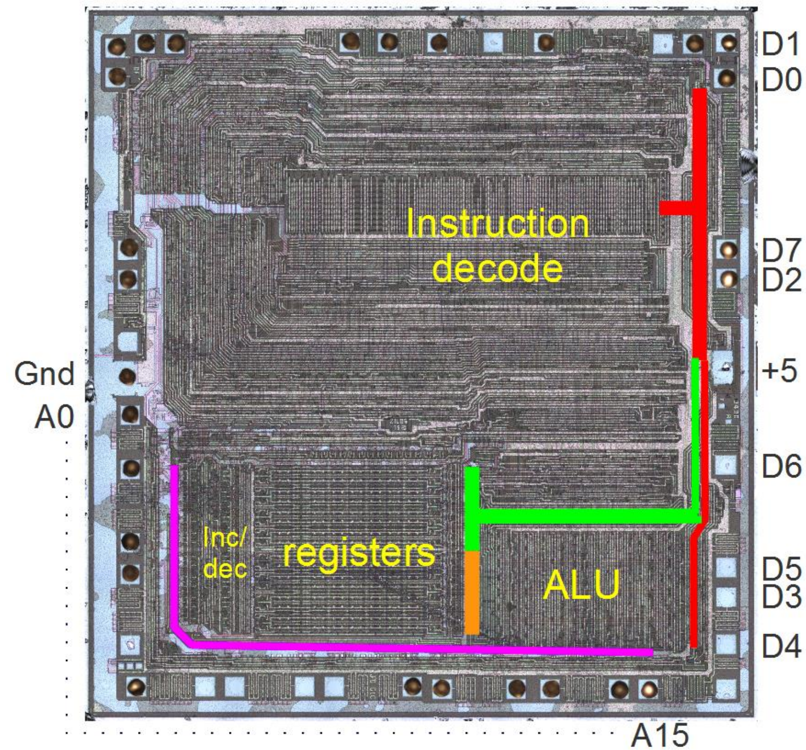
A bit about my background


sinclair
ZX Spectrum
BLUE EDIT 1 DEF FN
RED CAPS LOCK 2 FN
MAGENTA TRUE VIDEO 3 LINE
GREEN INV. VIDEO 4 OPEN #
CYAN 5 CLOSE #
YELLOW 6 MOVE
WHITE 7 ERASE
8 POINT
GRAPHICS 9 CAT
BLACK DELETE 0 FORMAT
SIN Q PLOT ASN
COS W DRAW ACS
TAN E REM ATN
INT R RUN VERIFY
RND T RAND MERGE
STR $ Y AND RETURN
CHR $ U OR IF
CODE I AT INPUT IN
PEEK O POKE OUT
TAB P PRINT
READ A STOP NEW
RESTORE S NOT SAVE
DATA D STEP DIM
SGN F TO FOR
ABS G THEN GOTO
SQR H GOSUB CIRCLE
VAL J LOAD VAL $
LEN K LIST SCREEN $
USR L LET ATTR
ENTER
CAPS SHIFT
LN Z COPY BEEP
EXP X CLEAR INK
L PRINT C CONT PAPER
L LIST V CLS FLASH
BIN B BORDER BRIGHT
IN KEY $ N NEXT OVER
PI M PAUSE INVERSE
SYMBOL SHIFT
BREAK SPACE

```
10>LET a=10
20 PRINT a
```

K




D1
D0
Instruction
decode
D7
D2
Gnd
+5
A0
D6
Inc/
dec
registers
ALU
D5
D3
D4
A15


12
10

Computing Hardware

We build tools

Used by “everyone” for “everything”

Science, medicine, commerce, …

Our Current Goal

- **Enabling Further Innovation in Machine Learning**
  - **Reduce compute, memory footprint and communication**
  - **Edge, Server, IoT**

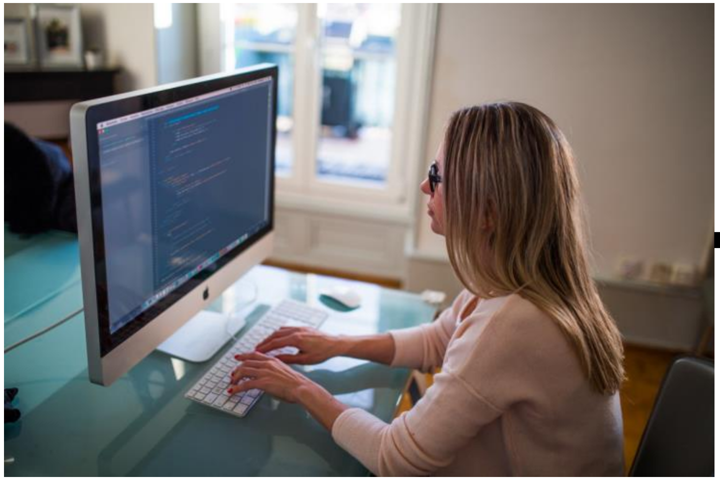
- **Two Guiding Principles…**

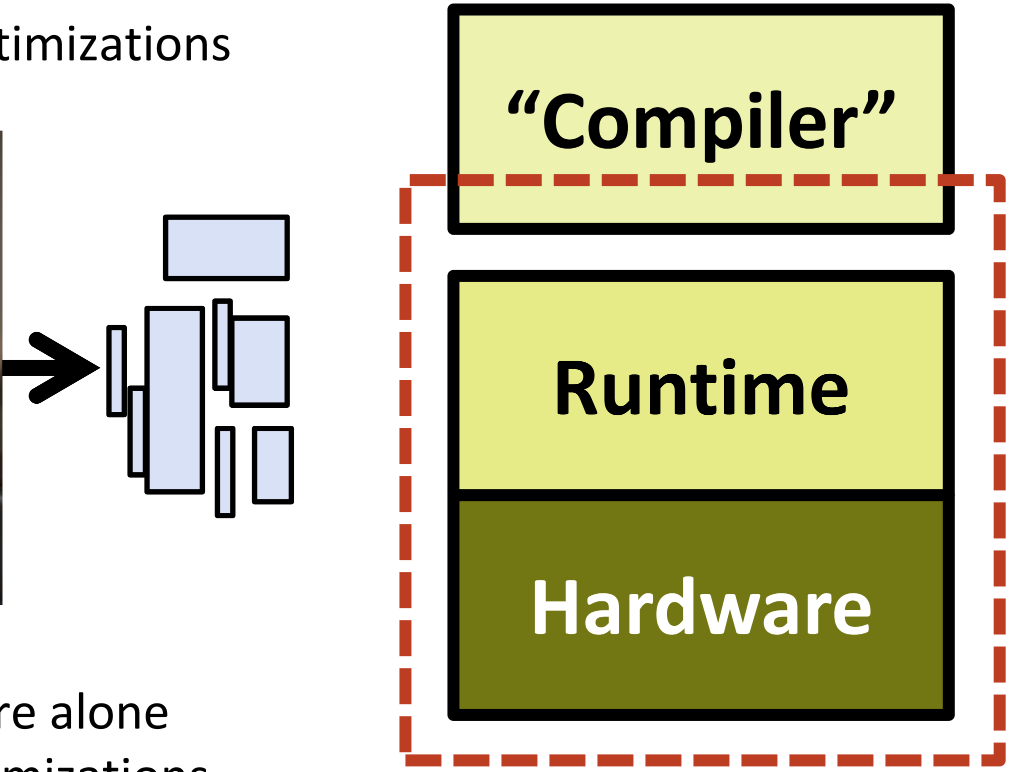
Principle #1

The advantage of natural occurring properties in Deep Learning Models

Do not require any changes
In the ML network/software
Developing software is hard…

But, …Reward model optimizations

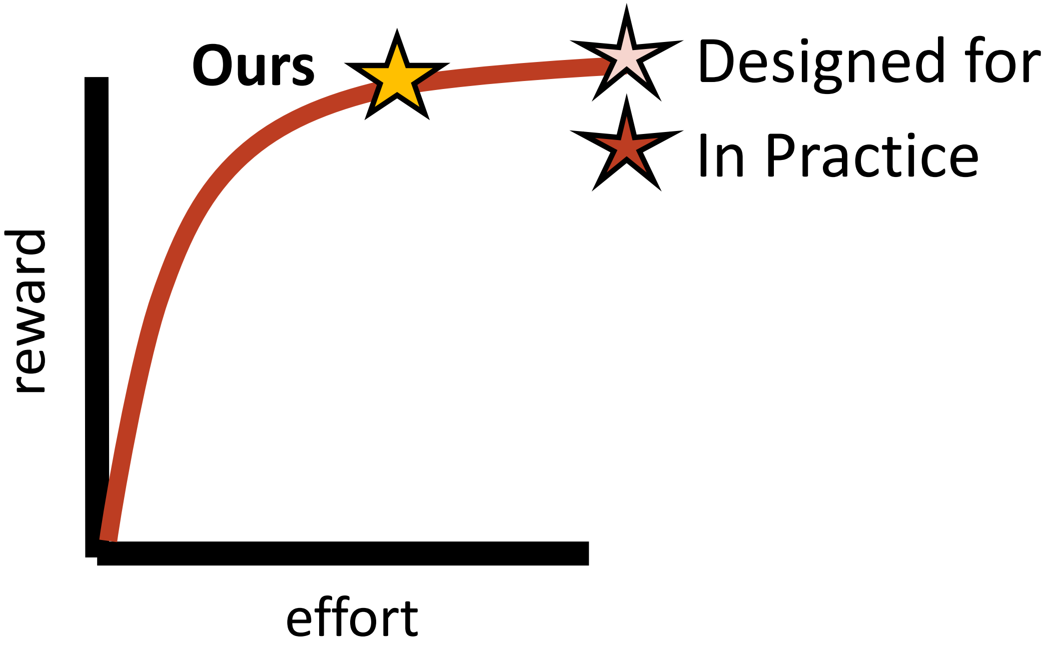




Improvements come from hardware alone
or low-level runtime/compiler optimizations

Principle #2

Balance hardware (area/energy) cost vs. reward (compute/memory amplification)



Behaviour-based approach to ML acceleration

7+ years of research

Family of techniques:

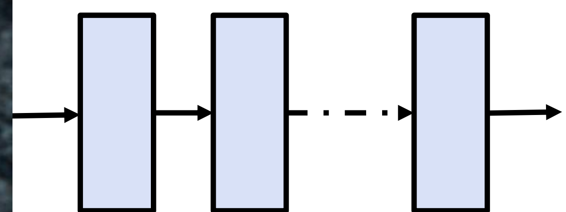
- Zero/near zero activation skipping
- Bit-serial designs → static + dynamic precision
- Memory compression (data width + delta) / on-chip /off-chip
- Bit-skipping designs
  - Computational Imaging
- Sparsity
- Inference + Training
- Software Tools:
  - Training Algorithm → bitwidth selection
  - Profiling

- **Apack**
  - Lossless compression for fixed-point inference
- **Mokey**
  - Quantization for Transformers

- **Schrödinger's FP**
  - Dynamic Adaptation of Floating-Point Containters

Example: Convolutional Neural Networks

Layers



Poutine
maybe

Tons of **Out += A x W**

For other types of networks too

Neural Nets do...

Out_0 += A_0 x W_0	Out_1 += A_0 x W_0	Out_0 += A_0 x W_0	Out_1 += A_0 x W_0	Out_0 += A_0 x W_0	Out_1 += A_0 x W_0	Out_0 += A_0 x W_0	Out_1 += A_0 x W_0
Out_0 += A_1 x W_1	Out_1 += A_1 x W_1	Out_0 += A_1 x W_1	Out_1 += A_1 x W_1	Out_0 += A_1 x W_1	Out_1 += A_1 x W_1	Out_0 += A_1 x W_1	Out_1 += A_1 x W_1
Out_0 += A_2 x W_2	Out_1 += A_2 x W_2	Out_0 += A_2 x W_2	Out_1 += A_2 x W_2	Out_0 += A_2 x W_2	Out_1 += A_2 x W_2	Out_0 += A_2 x W_2	Out_1 += A_2 x W_2
Out							W_3
Out							W_4
Out_0 += A_0 x W_0	Out_1 += A_0 x W_0	Out_0 += A_0 x W_0	Out_1 += A_0 x W_0	Out_0 += A_0 x W_0	Out_1 += A_0 x W_0	Out_0 += A_0 x W_0	Out_1 += A_0 x W_0
Out_0 += A_1 x W_1	Out_1 += A_1 x W_1	Out_0 += A_1 x W_1	Out_1 += A_1 x W_1	Out_0 += A_1 x W_1	Out_1 += A_1 x W_1	Out_0 += A_1 x W_1	Out_1 += A_1 x W_1
Out_0 += A_2 x W_2	Out_1 += A_2 x W_2	Out_0 += A_2 x W_2	Out_1 += A_2 x W_2	Out_0 += A_2 x W_2	Out_1 += A_2 x W_2	Out_0 += A_2 x W_2	Out_1 += A_2 x W_2
Out_0 += A_3 x W_3	Out_1 += A_3 x W_3	Out_0 += A_3 x W_3	Out_1 += A_3 x W_3	Out_0 += A_3 x W_3	Out_1 += A_3 x W_3	Out_0 += A_3 x W_3	Out_1 += A_3 x W_3
Out							W_4
Out							W_0
Out_0 += A_1 x W_1	Out_1 += A_1 x W_1	Out_0 += A_1 x W_1	Out_1 += A_1 x W_1	Out_0 += A_1 x W_1	Out_1 += A_1 x W_1	Out_0 += A_1 x W_1	Out_1 += A_1 x W_1
Out_0 += A_2 x W_2	Out_1 += A_2 x W_2	Out_0 += A_2 x W_2	Out_1 += A_2 x W_2	Out_0 += A_2 x W_2	Out_1 += A_2 x W_2	Out_0 += A_2 x W_2	Out_1 += A_2 x W_2
Out_0 += A_3 x W_3	Out_1 += A_3 x W_3	Out_0 += A_3 x W_3	Out_1 += A_3 x W_3	Out_0 += A_3 x W_3	Out_1 += A_3 x W_3	Out_0 += A_3 x W_3	Out_1 += A_3 x W_3
Out_0 += A_4 x W_4	Out_1 += A_4 x W_4	Out_0 += A_4 x W_4	Out_1 += A_4 x W_4	Out_0 += A_4 x W_4	Out_1 += A_4 x W_4	Out_0 += A_4 x W_4	Out_1 += A_4 x W_4

Many MACs

Lots of **data to transfer**

When we started we assumed: Everyone in industry will target parallelism and data blocking first.

We wanted to be ready with the next technologies once these two are “perfected”.

We targeted “behavior” based optimizations: what ML does at runtime that we can take advantage of.
The programmer specifies a way to compute a result, as long as we produce the same result we can play tricks at the hardware level to improve efficiency. Lots of experience from CPUs: caches, branch prediction, etc.

Do as you are told?

Out_0 += A_3 x W_3 Out_1 += A_3 x W_3 Out_0 += A_3 x W_3 Out_1 += A_3 x W_3 Out_0 += A_3 x W_3 Out_1 += A_3 x W_3 Out_0 += A_3 x W_3 Out_1 += A_3 x

Out_0 += A_4 x W_4 Out_1 += A_4 x W_4 Out_0 += A_4 x W_4 Out_1 += A_4 x W_4 Out_0 += A_4 x W_4 Out_1 += A_4 x W_4 Out_0 += A_4 x W_4 Out_1 += A_4 x

Instead calculate the *same* output but … do less work

Out_0 += A_2 x W_2 Out_1 += A_2 x W_2 Out_0 += A_2 x W_2 Out_1 += A_2 x W_2 Out_0 += A_2 x W_2 Out_1 += A_2 x W_2 Out_0 += A_2 x W_2 Out_1 += A_2 x

Out_0 += A_3 x W_3 Out_1 += A_3 x W_3 Out_0 += A_3 x W_3 Out_1 += A_3 x W_3 Out_0 += A_3 x W_3 Out_1 += A_3 x W_3 Out_0 += A_3 x W_3 Out_1 += A_3 x

Out_0 += A_4 x W_4 Out_1 += A_4 x W_4 Out_0 += A_4 x W_4 Out_1 += A_4 x W_4 Out_0 += A_4 x W_4 Out_1 += A_4 x W_4 Out_0 += A_4 x W_4 Out_1 += A_4 x

$Out_0 \mathrel{+}= A_0 \times W_0$

$Out_0 \mathrel{+}= A_4 \times W_4$

$Out_0 \mathrel{+}= A_2 \times W_2$

$Out_0 \mathrel{+}= A_3 \times W_3$

$Out_0 \mathrel{+}= A_4 \times W_4$

$Out_0 \mathrel{+}= A_4 \times W_4$

$Out_1 \mathrel{+}= A_0 \times W_0$

$Out_1 \mathrel{+}= A_3 \times W_3$

$Out_1 \mathrel{+}= A_0 \times W_0$

$Out_1 \mathrel{+}= A_1 \times W_1$

$Out_1 \mathrel{+}= A_0 \times W_0$

$Out_0 \mathrel{+}= A_0 \times W_0$

$Out_0 \mathrel{+}= A_0 \times W_0$

$Out_0 \mathrel{+}= A_1 \times W_1$

$Out_0 \mathrel{+}= A_0 \times W_0$

$Out_0 \mathrel{+}= A_1 \times W_1$

$Out_1 \mathrel{+}= A_0 \times W_0$

$Out_1 \mathrel{+}= A_1 \times W_1$

$Out_1 \mathrel{+}= A_4 \times W_4$

2

$Out_1 \mathrel{+}= A_3 \times W_3$

$Out_1 \mathrel{+}= A_4 \times W_4$

$Out_1 \mathrel{+}= A_0 \times W_0$

$Out_1 \mathrel{+}= A_4 \times W_4$

$Out_0 \mathrel{+}= A_0 \times W_0$

$Out_0 \mathrel{+}= A_0 \times W_0$

$Out_0 \mathrel{+}= A_1 \times W_1$

$Out_1 \mathrel{+}= A_0 \times W_0$

$Out_1 \mathrel{+}= A_1 \times W_1$

$Out_1 \mathrel{+}= A_0 \times W_0$

$Out_1 \mathrel{+}= A_4 \times W_4$

$Out_0 \mathrel{+}= A_4 \times W_4$

$Out_0 \mathrel{+}= A_1 \times W_1$

$Out_0 \mathrel{+}= A_0 \times W_0$

$Out_1 \mathrel{+}= A_0 \times$

$Out_1 \mathrel{+}= A_4 \times$

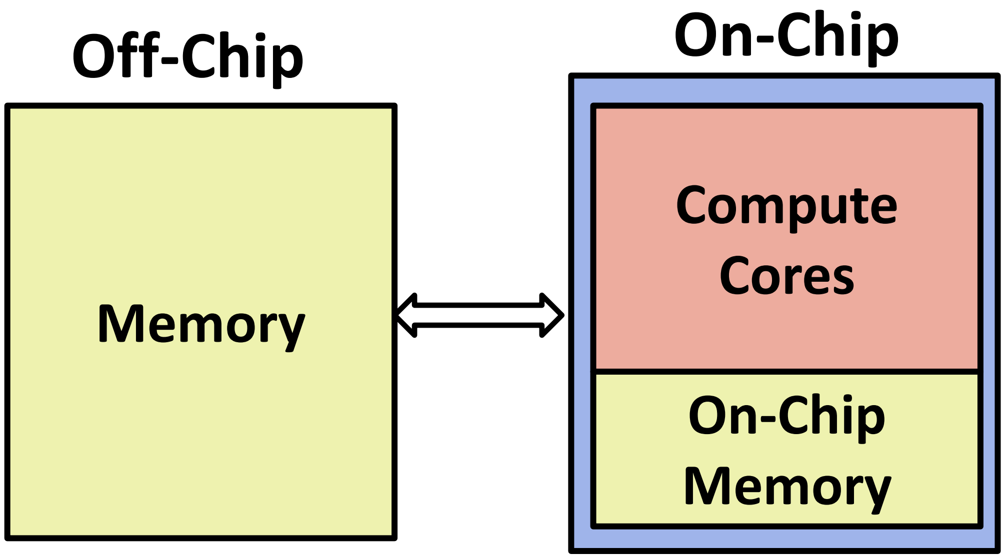
$Out_1 \mathrel{+}= A_0 \times$

$Out_1 \mathrel{+}= A_1 \times$

$Out_1 \mathrel{+}= A_0 \times$

$Out_1 \mathrel{+}= A_4 \times$

Technology #1: Memory Transfers: Shapeshifter



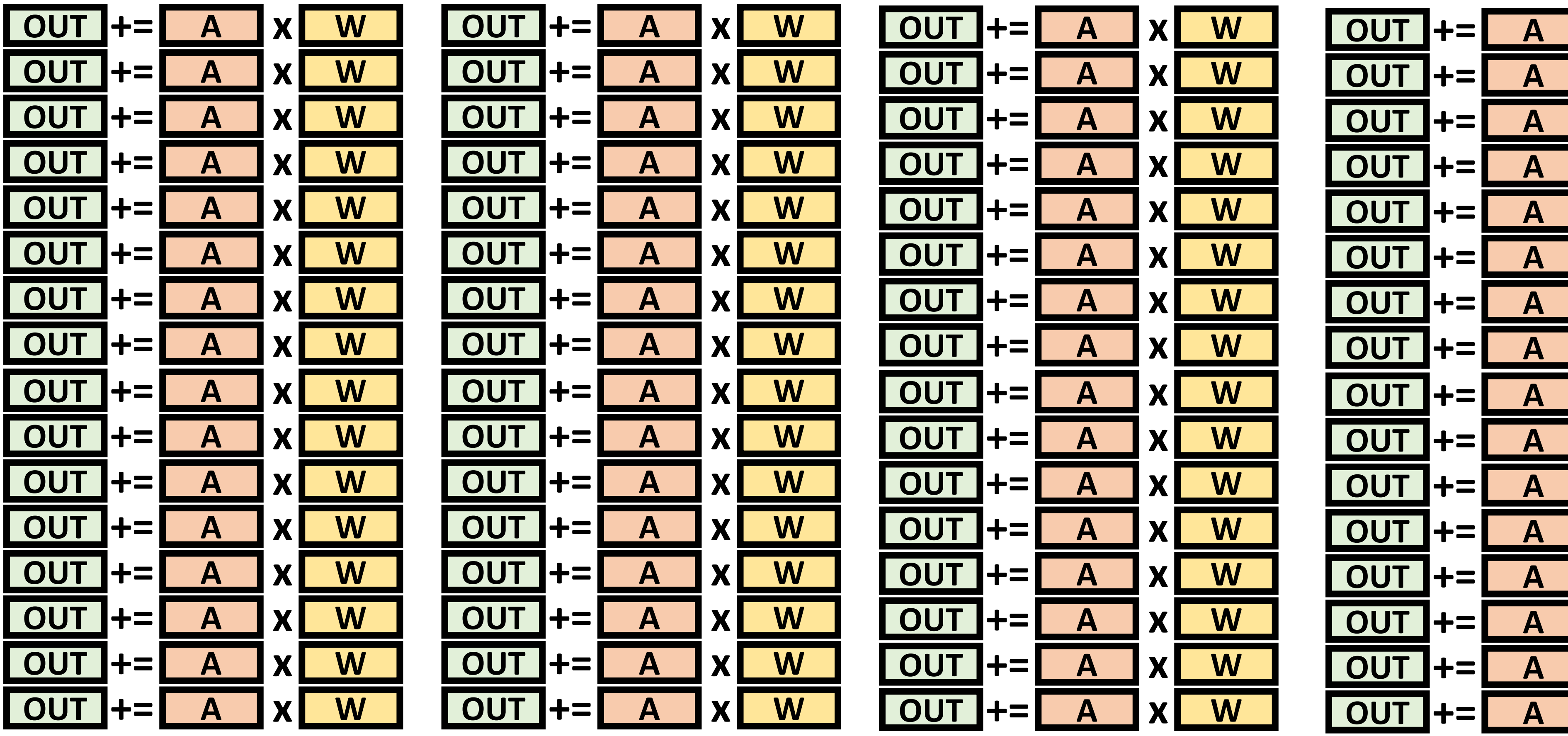
On- vs. Off-Chip

Energy: ~100x

Latency: ~50x

Compute/Watt is the primary design constraint

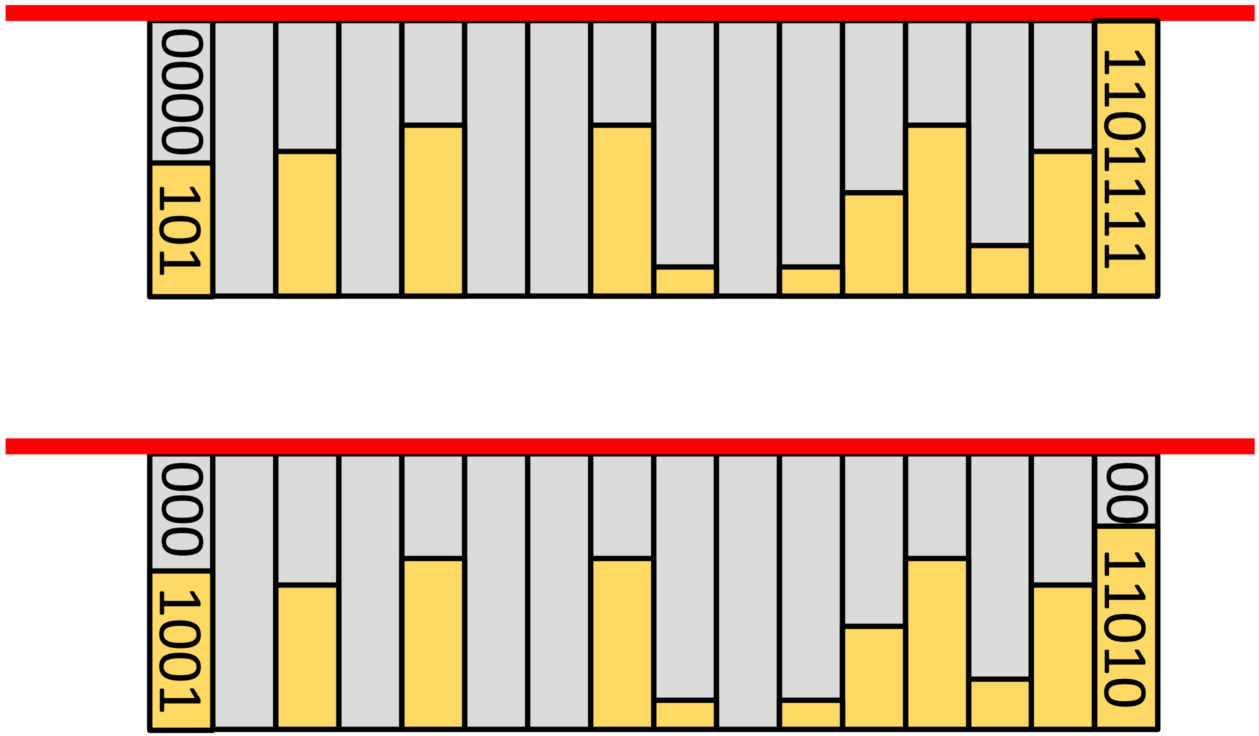
Conventional Approach:
One Datawidth to Rule them All



Pick a datatype that fits the range of *all* values… this proves excessive for ML workloads…

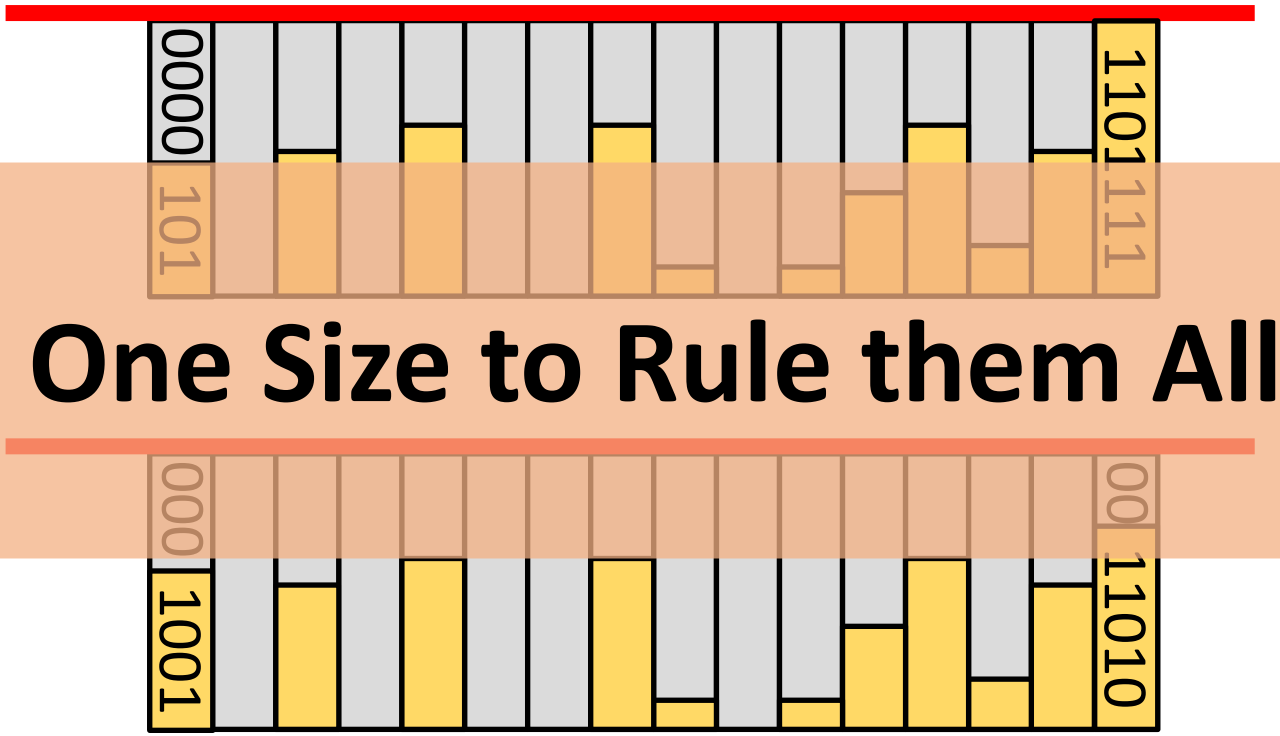
Conventional Data Transfers: Fixed Size Container Per Value

e.g., transfer 16 values at a time all using 8b each



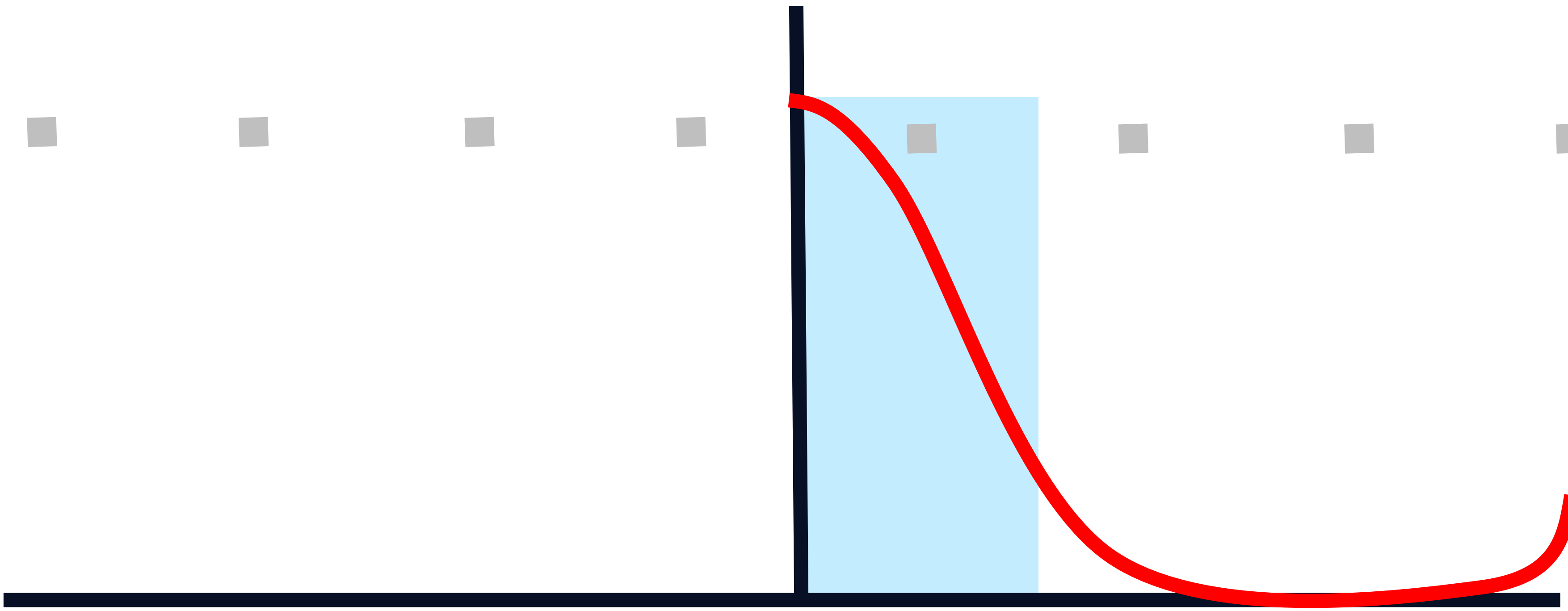
Conventional Data Transfers: Fixed Size Container Per Value

e.g., transfer 16 values at a time all using 8b each



Most ML values can fit in much narrow containers

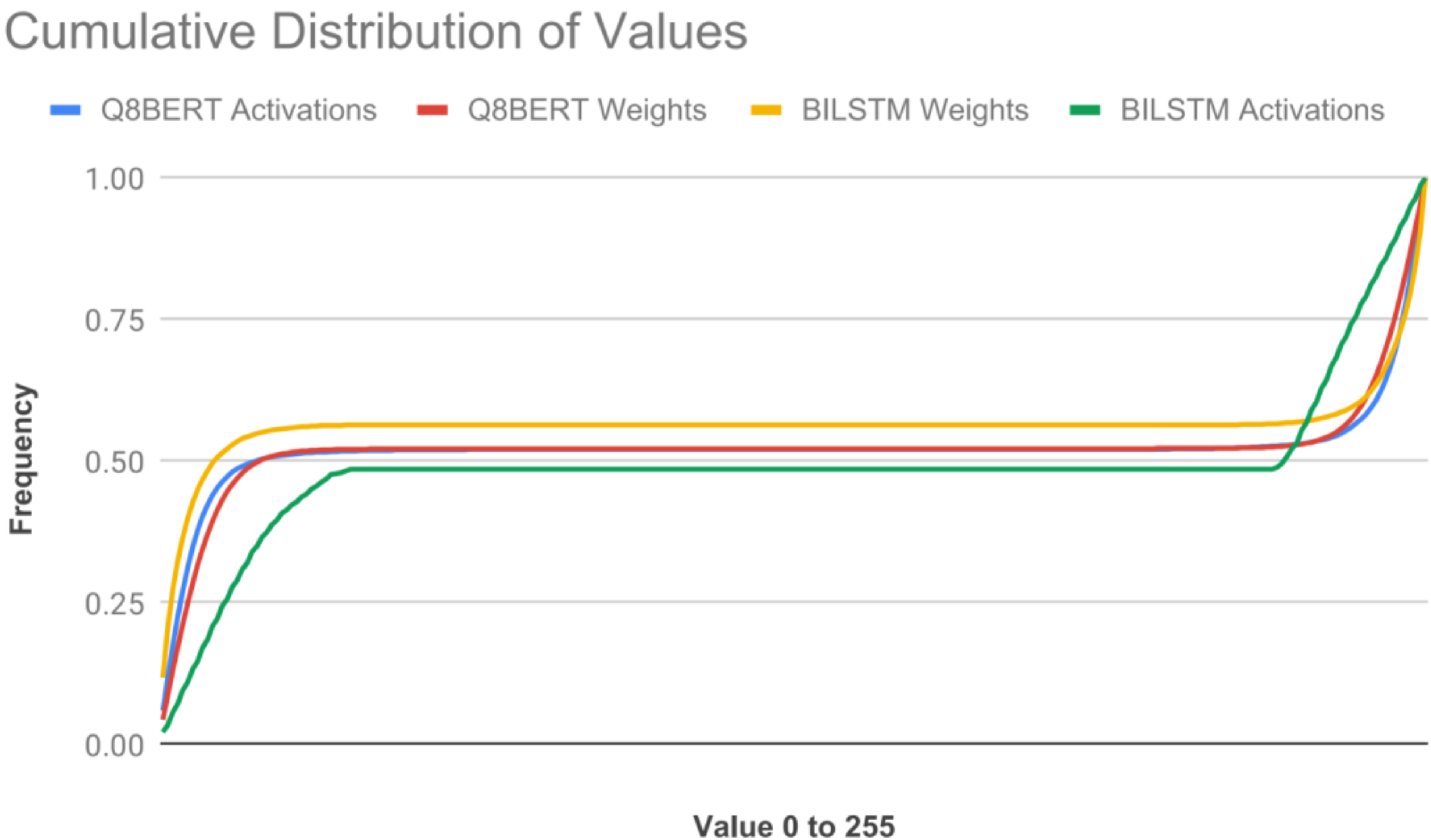
Out_0 += A_0 x W_0 Out_1 += A_0 x W_0 Out_0 += A_0 x W_0 Out_1 += A_0 x W_0 Out_0 += A_0 x W_0 Out_1 += A_0 x W_0 Out_0 += A_0 x W_0 Out_1 += A_0 x W



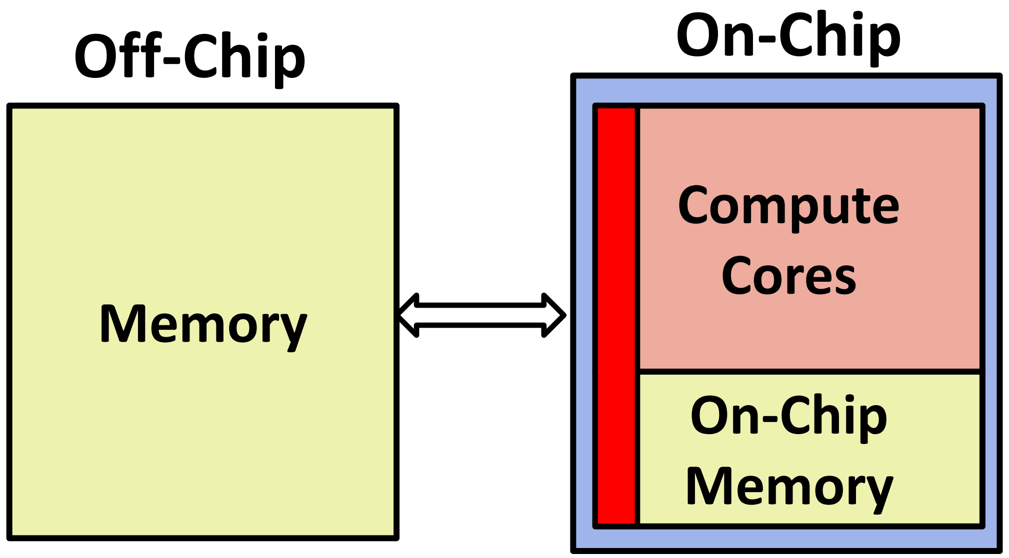
DPRed: Making Typical Activation and Weight Values Matter In Deep Learning Computing, Delmas et al.,
https://arxiv.org/abs/1804.06732

Value Distribution During Inference

Far from Uniform: Few Values -> Most Frequent



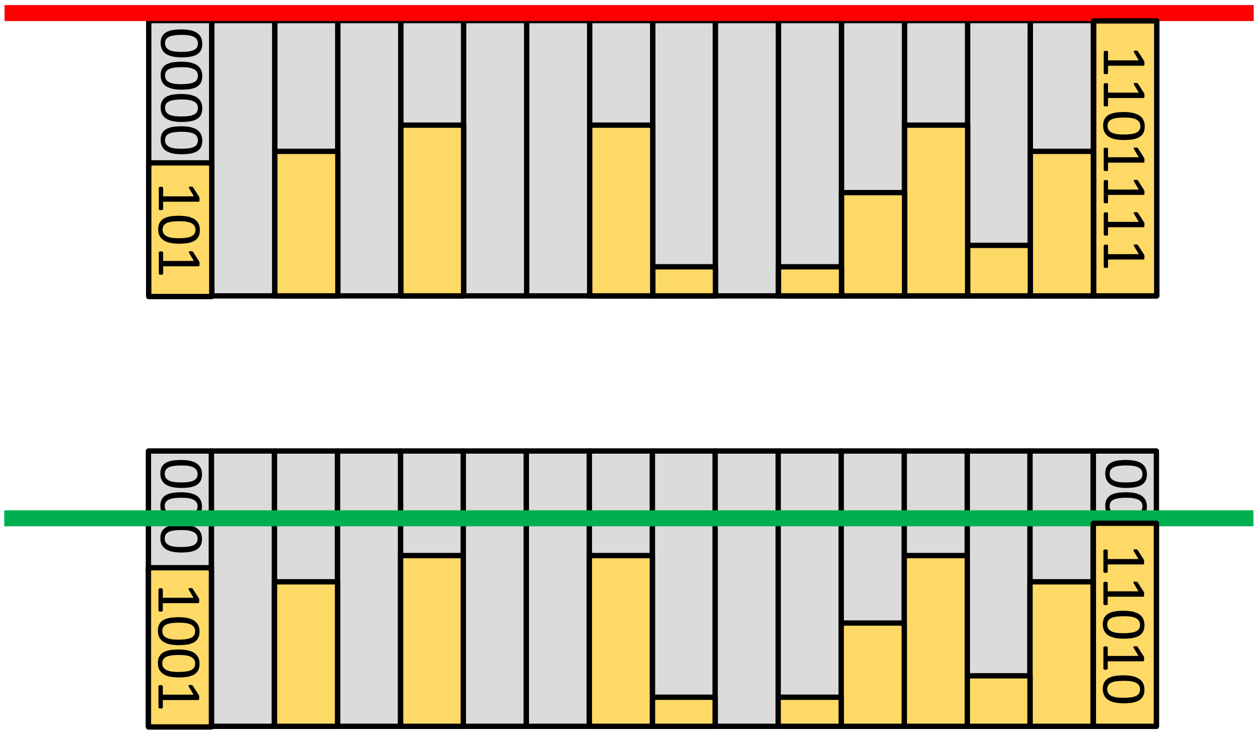
Technology #1: Memory Transfers: Shapeshifter



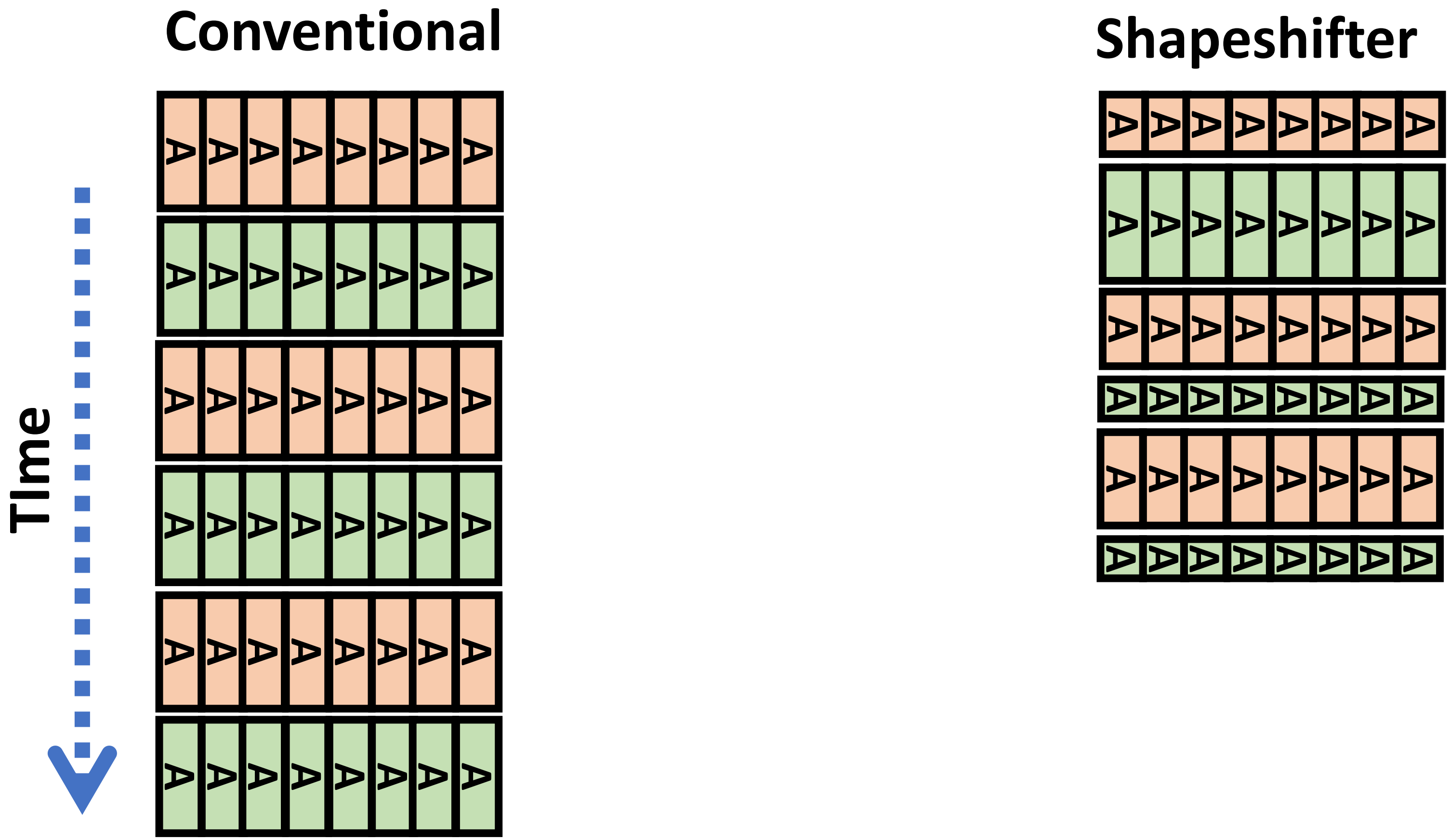
Encode/Decode Value to/from Memory

Shapeshifter: Make Typical Values Matter

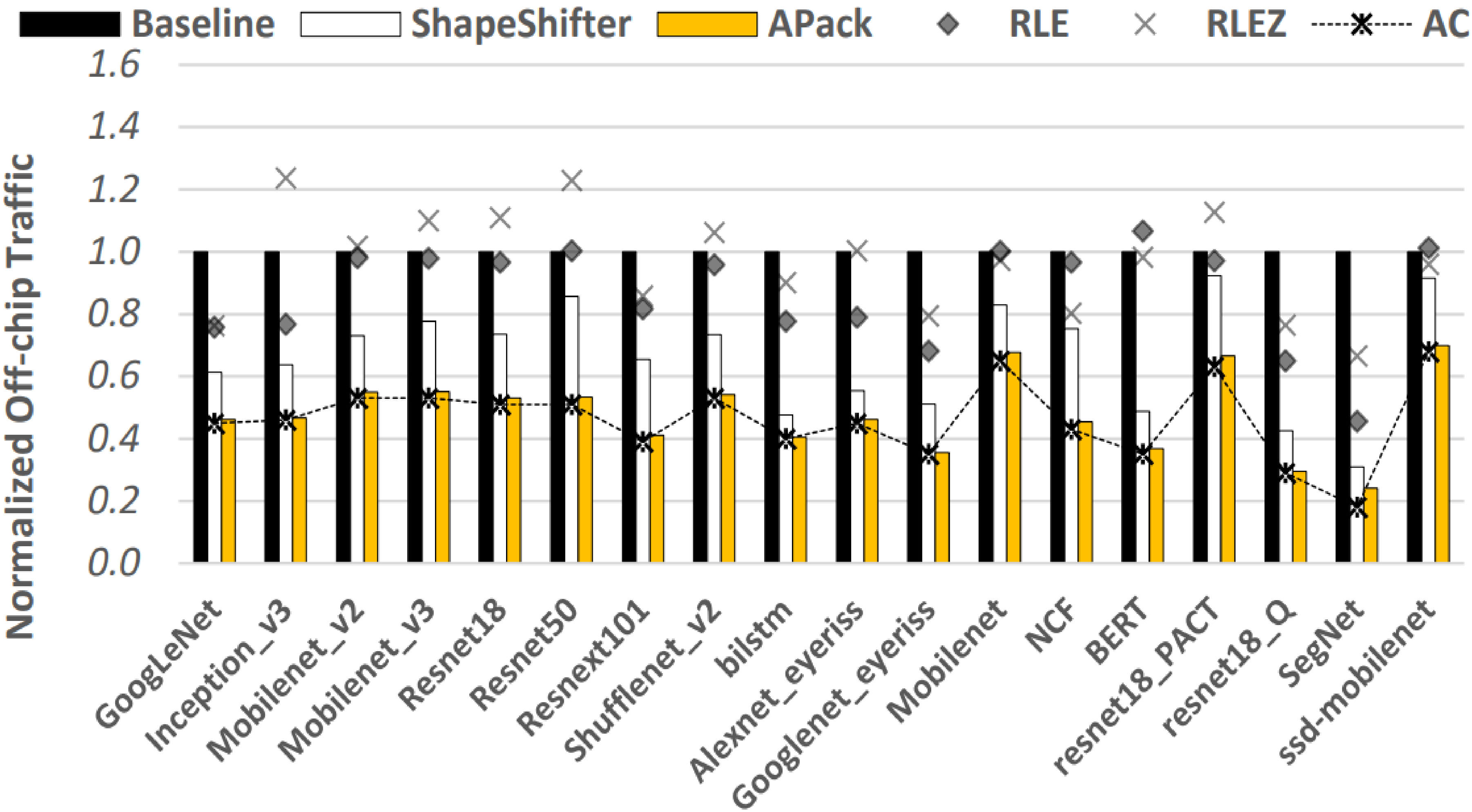
Container adapts to value content. Weights and activations.



Memory Transfers

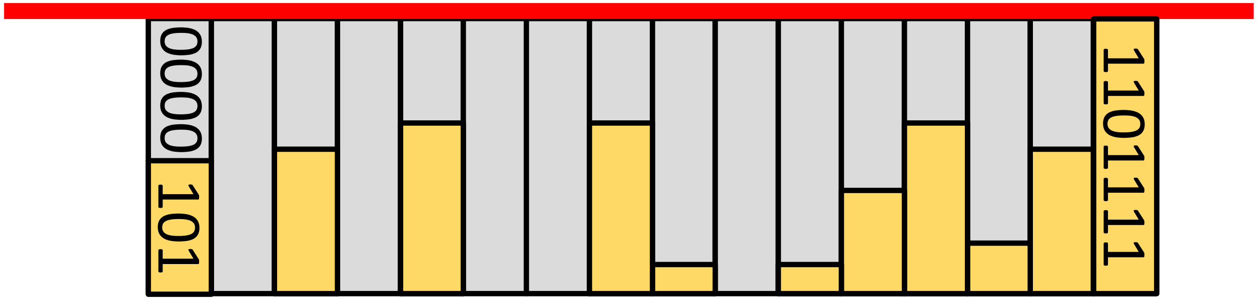


Shapeshifter Effectiveness



Shapeshifter: Life is not always fair

This may happen often depending on the network



APACK

12 0 23 45 67 127 18 22 88 103 234 22 1 0 2 3 5 8 19 9 0 9 8 20 28 220 20 20 244 223 2 1 1 0 1 0
19 9 0 9 8 20 28 220 20 20 244 223 2 1 1 0 1 0 12 0 23 45 67 127 18 22 88 103 234 22 1 0 2 3 5 8
234 22 1 0 2 3 5 8 9 8 20 28 220 20 20 244 223 2 1 1 0 1 0 12 0 23 45 19 9 0 67 127 18 22 88 103

….

28 220 20 20 244 223 2 1 1 0 1 0 234 22 1 0 2 3 19 9 0 67 127 18 22 88 103 5 8 9 8 20 12 0 23 45

0.10238464892028374628298383933….333292

12 0 23 45 67 127 18 22 88 103 234 22 1 0 2 3 5 8 19 9 0 9 8 20 28 220 20 20 244 223 2 1 1 0 1 0
19 9 0 9 8 20 28 220 20 20 244 223 2 1 1 0 1 0 12 0 23 45 67 127 18 22 88 103 234 22 1 0 2 3 5 8
234 22 1 0 2 3 5 8 9 8 20 28 220 20 20 244 223 2 1 1 0 1 0 12 0 23 45 19 9 0 67 127 18 22 88 103

….

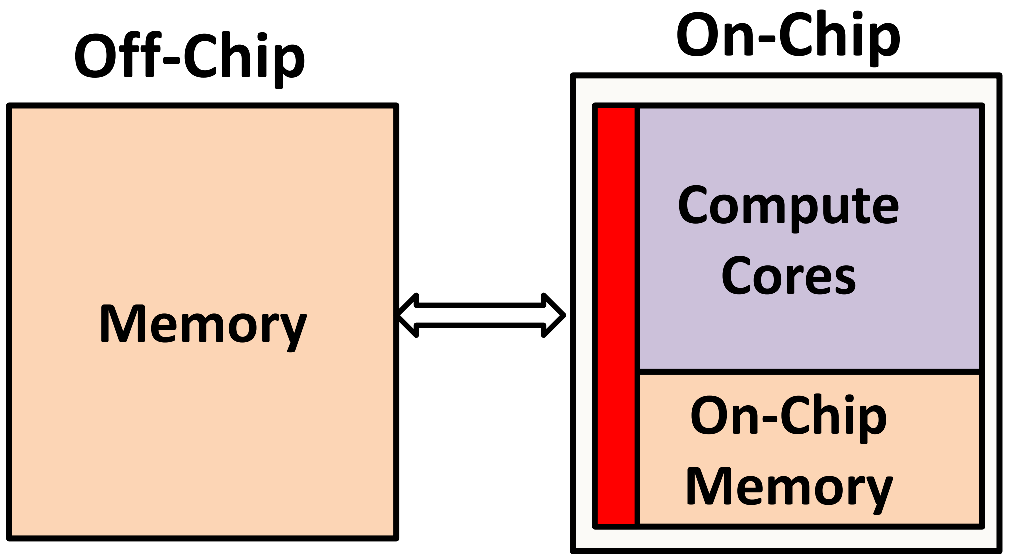
28 220 20 20 244 223 2 1 1 0 1 0 234 22 1 0 2 3 19 9 0 67 127 18 22 88 103 5 8 9 8 20 12 0 23 45

0.1023846489202837462829838393….333292

$0.1101010101010101010111101 01…111001_{(2)}$

Frequent values → less than ONE BIT

Technology #1: Memory Transfers: Shapeshifter



Encode/Decode Value to/from Memory

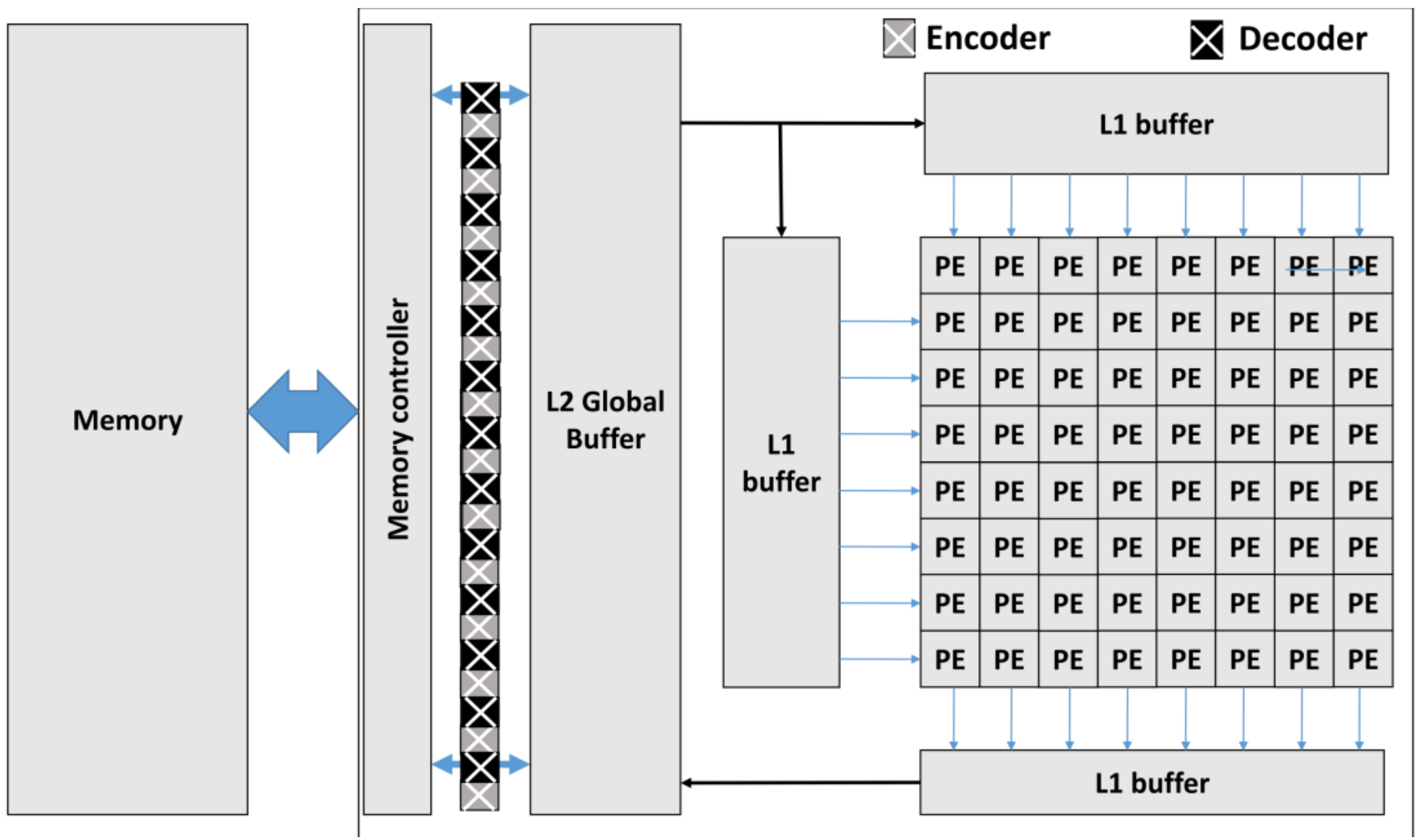
APACK: Lossless Compression for fixed-point

- Based on Arithmetic Coding
  - Encode a TENSOR with unique REAL number
- Precision needed:
  - Sequence Length
  - Frequency of values

- Outline:
  - Classical Arithmetic Coding
    - Too expensive – too slow
  - Apack

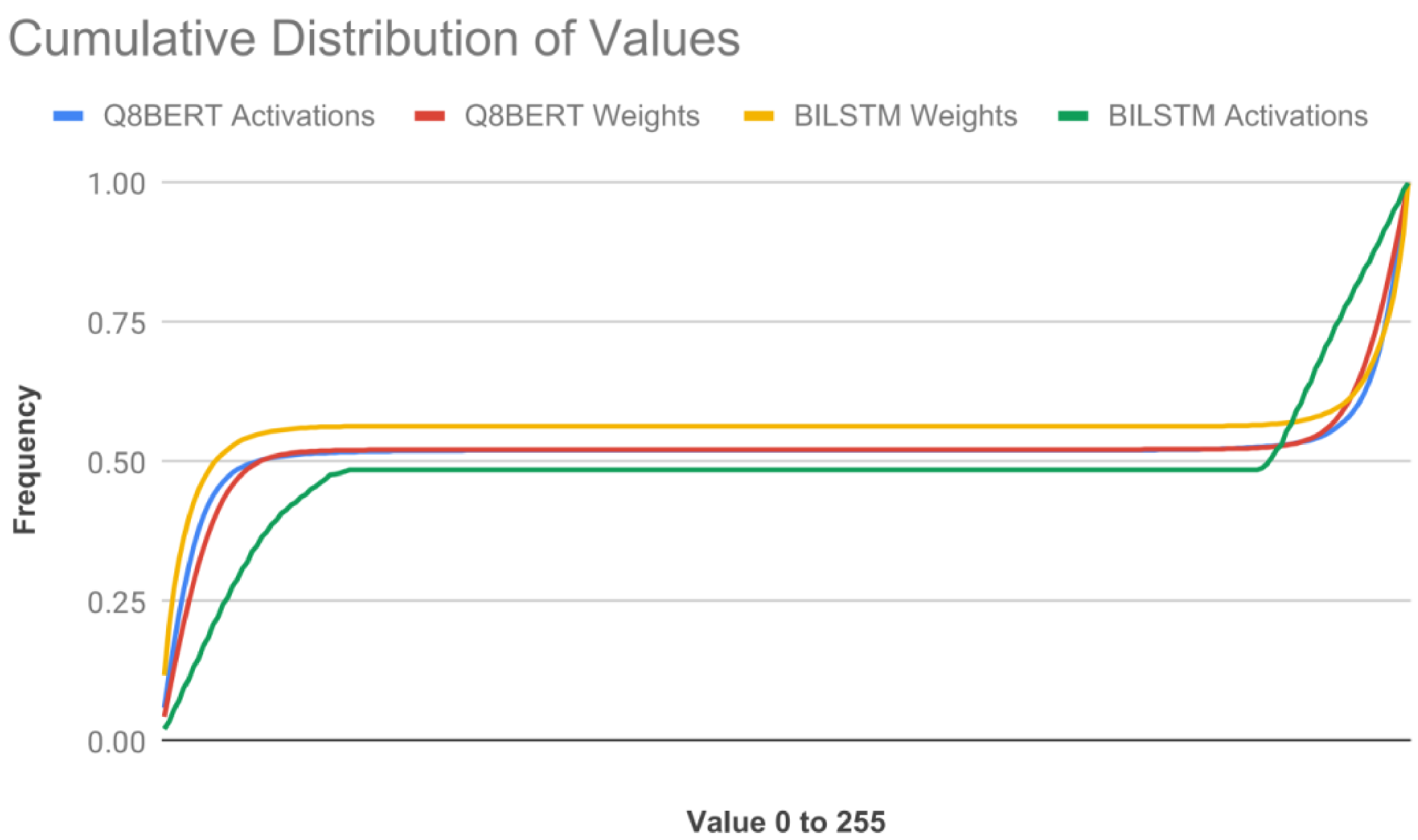
Key Idea: Encode Values According to Frequency

- Transparently encode/decode
- Lossless
- Weights in advance / Activations Profiling



Value Distribution During Inference

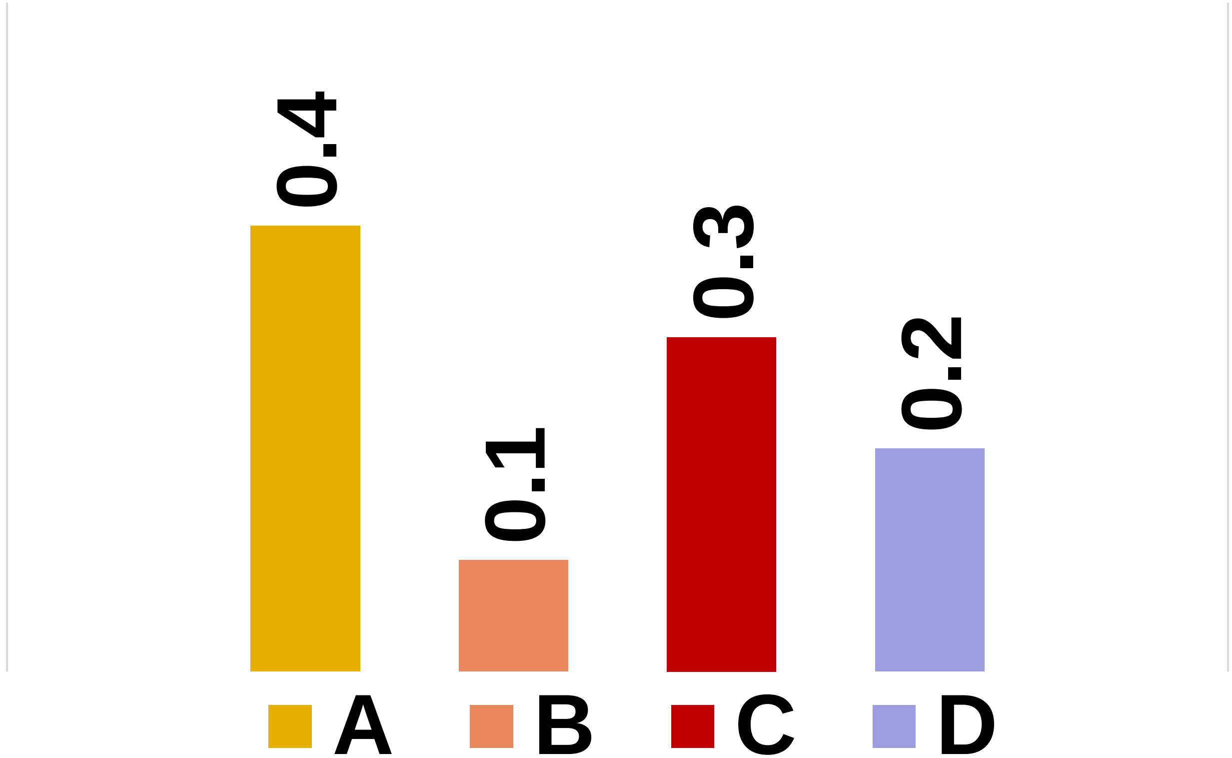
Far from Uniform: Few Values -> Most Frequent



Values change with input → Distributin not so much

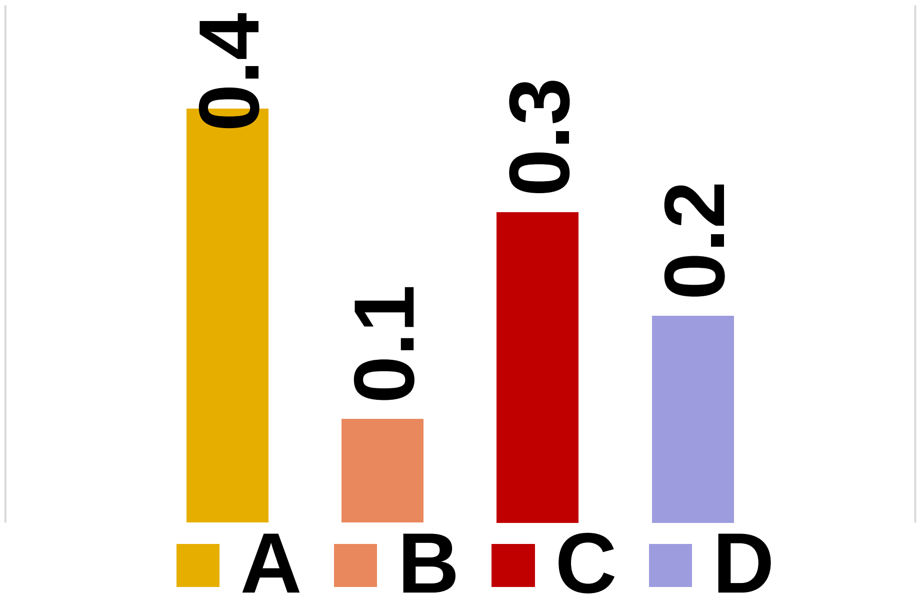
Classical Arithmetic Coding

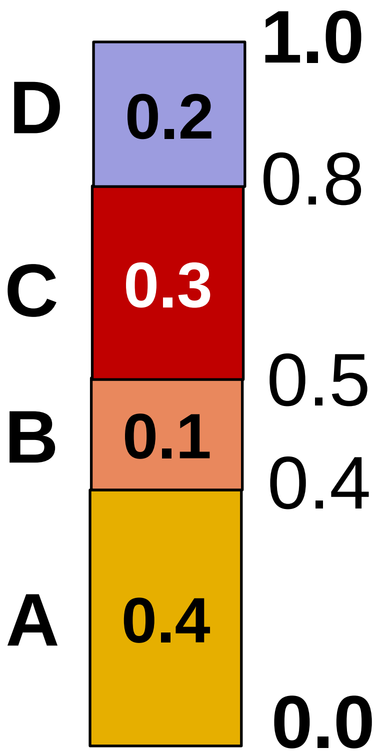
- Symbols w/ Frequencies



Classical Arithmetic Coding

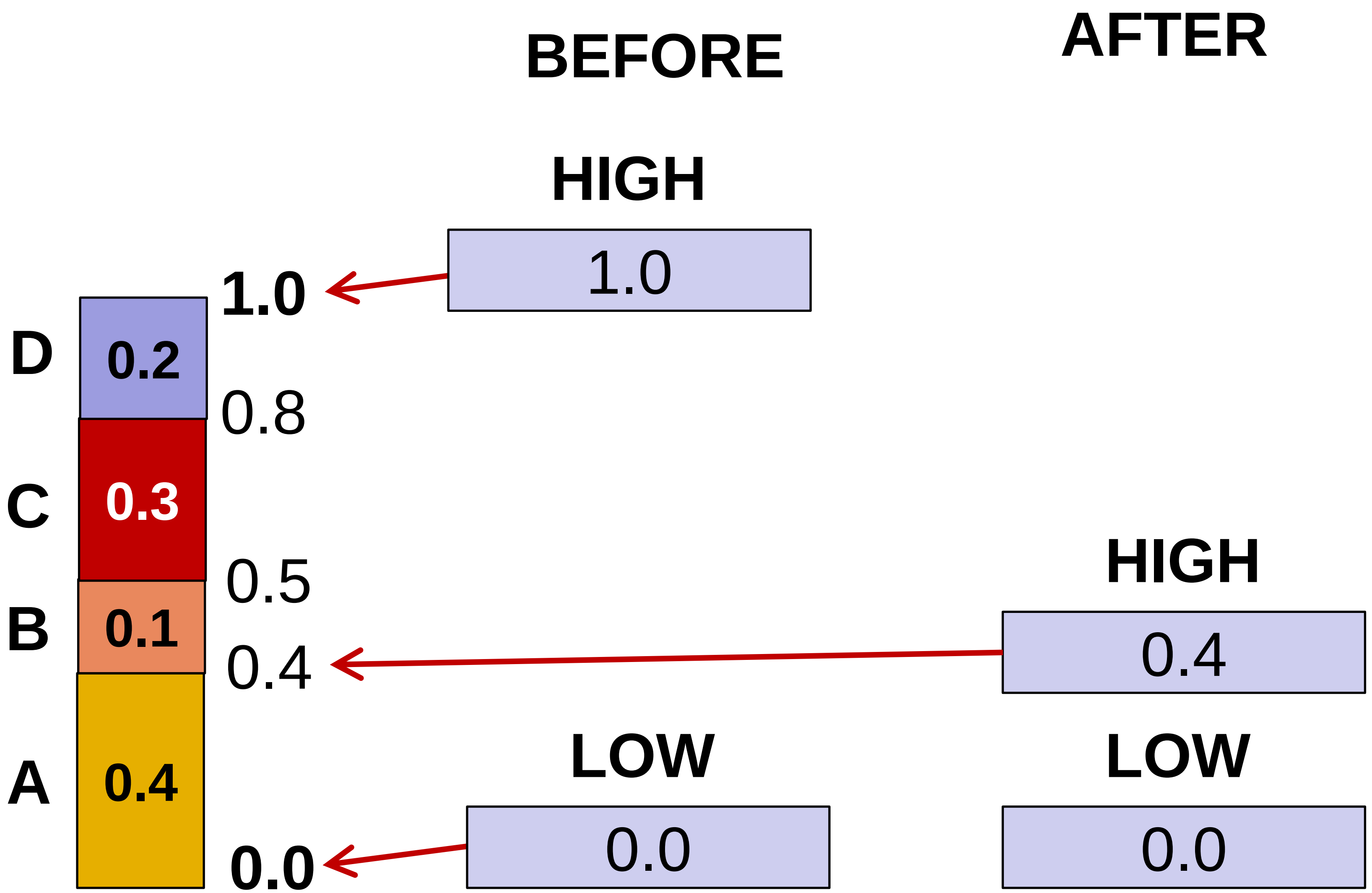
- #1 Range/Probability Assignment
  - All the info needed to encode decode





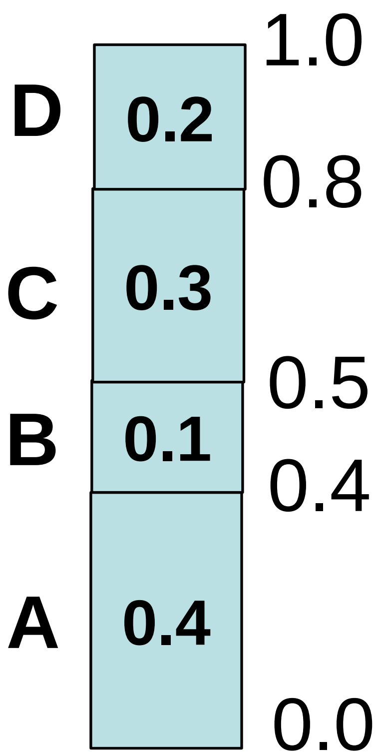
Encoding ABA

- Incoming Symbol: A



Encoding the first value: A

- A



BEFORE

HIGH

1.0

LOW

0.0

AFTER

HIGH

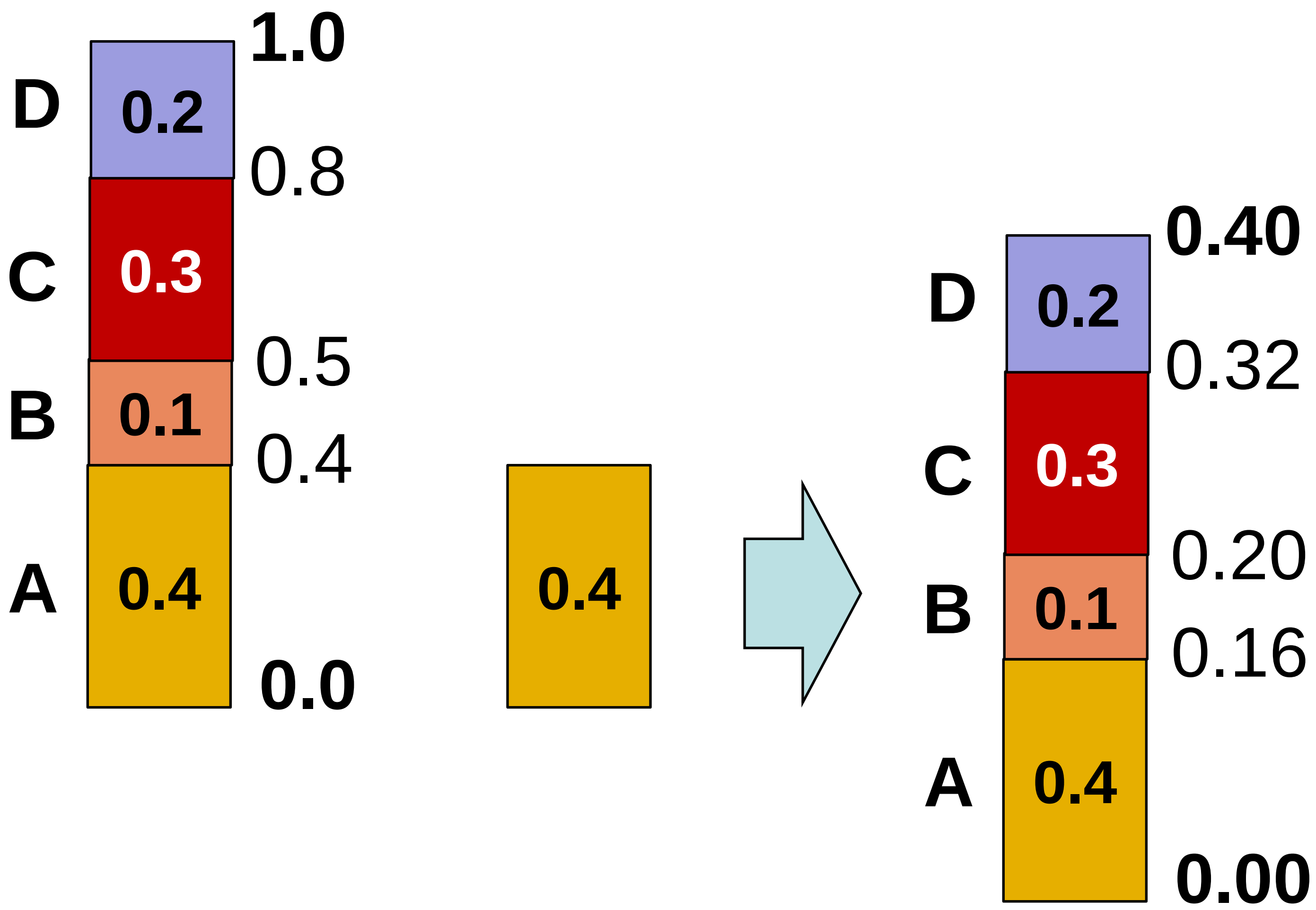
0.4

LOW

0.0

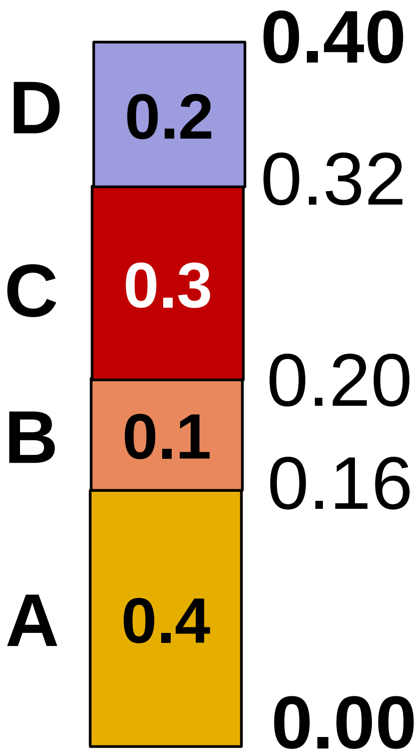
After Encoding A

- How our range table looks



Encoding the second value: B

- B

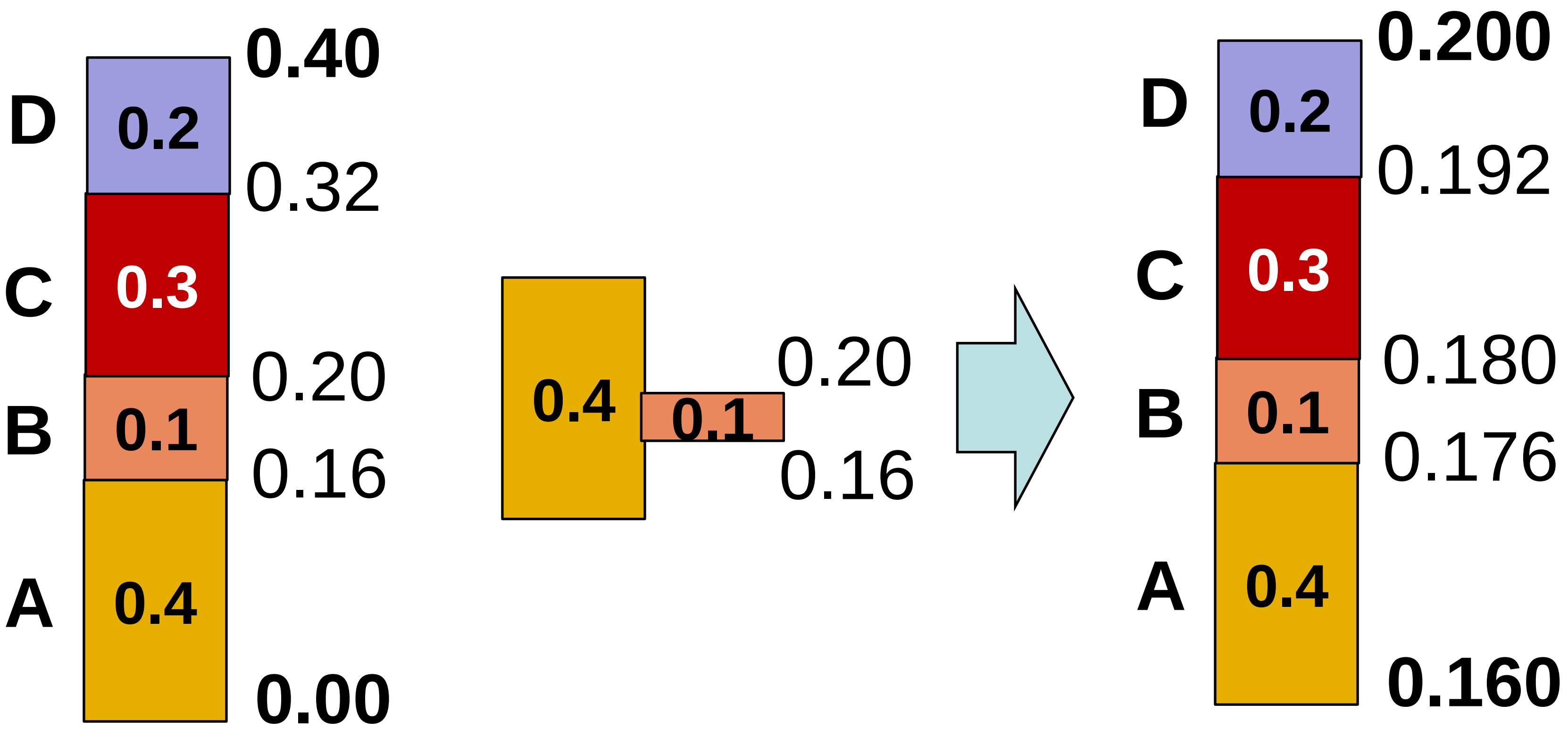


	BEFORE	AFTER
HIGH	0.4	0.20
LOW	0.0	0.16

Initial range: HIGH – LOW = 0.4
B's prob = 0.1
B's offset = 0.4
New offset = LOW x B's offset = 0.16
New range = initial range x B's prob = 0.04
LOW = 0.16
HIGH = 0.16 + 0.04

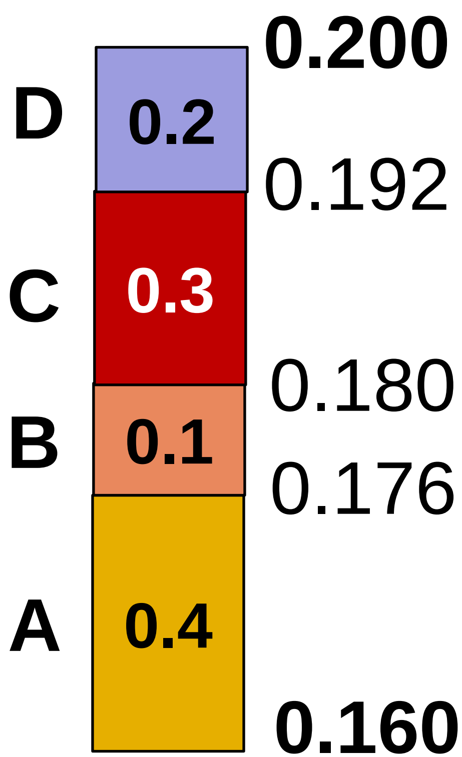
Enconding the B

- Incoming Symbols: **A****B**



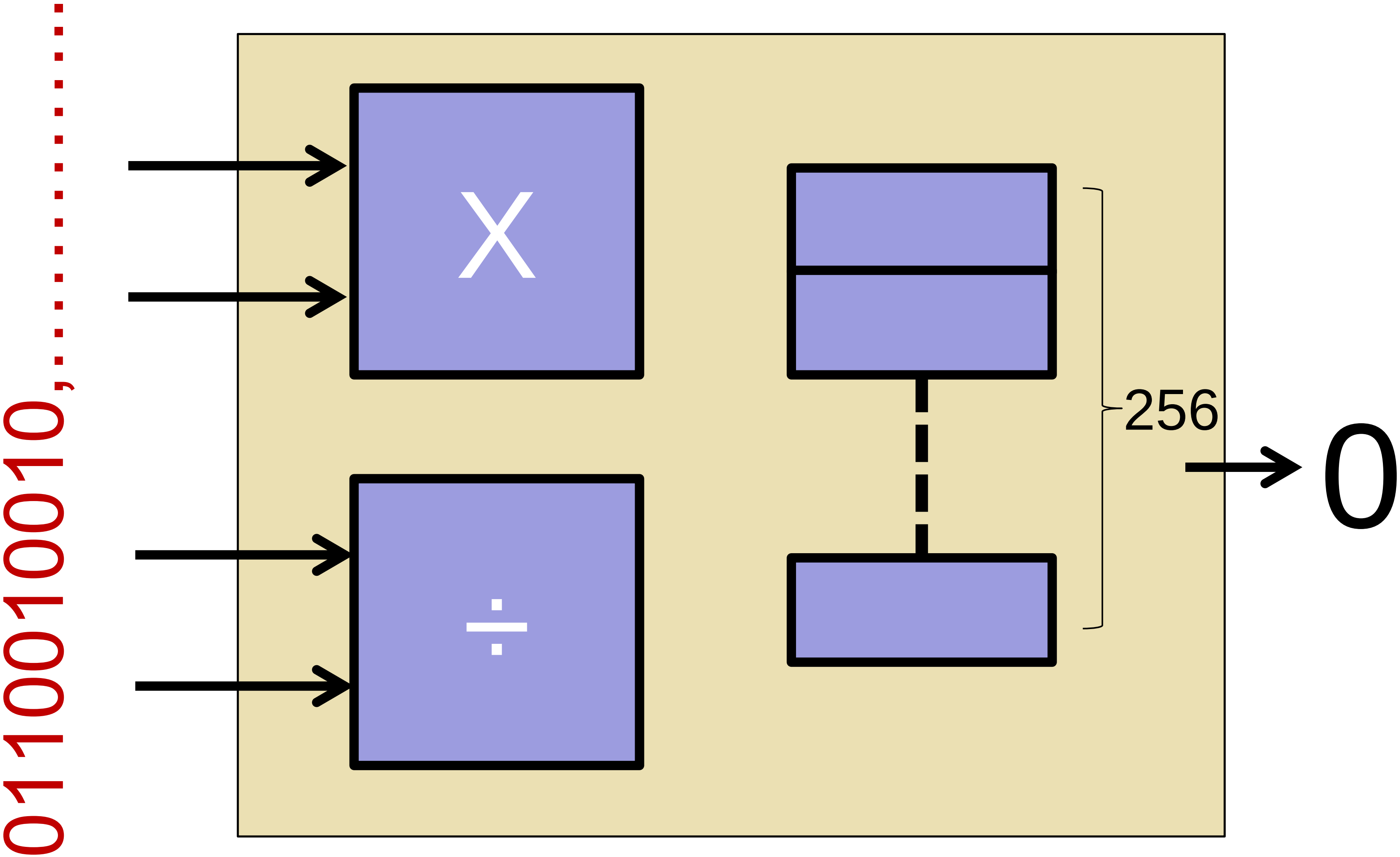
Encoding the third value: A

- A



	BEFORE	AFTER
HIGH	0.20	0.176
LOW	0.16	0.160

Initial range: HIGH – LOW = 0.04
A's prob = 0.4
B's offset = 0.0
New offset = LOW x A's offset = 0.16 + 0.0
New range = initial range x A's prob = 0.016
LOW = 0.16
HIGH = 0.16 + 0.016 = 0.176


0110010010,
X
÷
256
0

Challenges with Arithmetic Coding

- Arbitrary Precision Arithmetic
  - Multiplications and Divisions
- Expensive Range Table
  - 256 entries for 8b fixed-point
- Low Bandwidth
  - 1 bit per invocation

APACK Key Idea

V = PREFIX + OFFSET

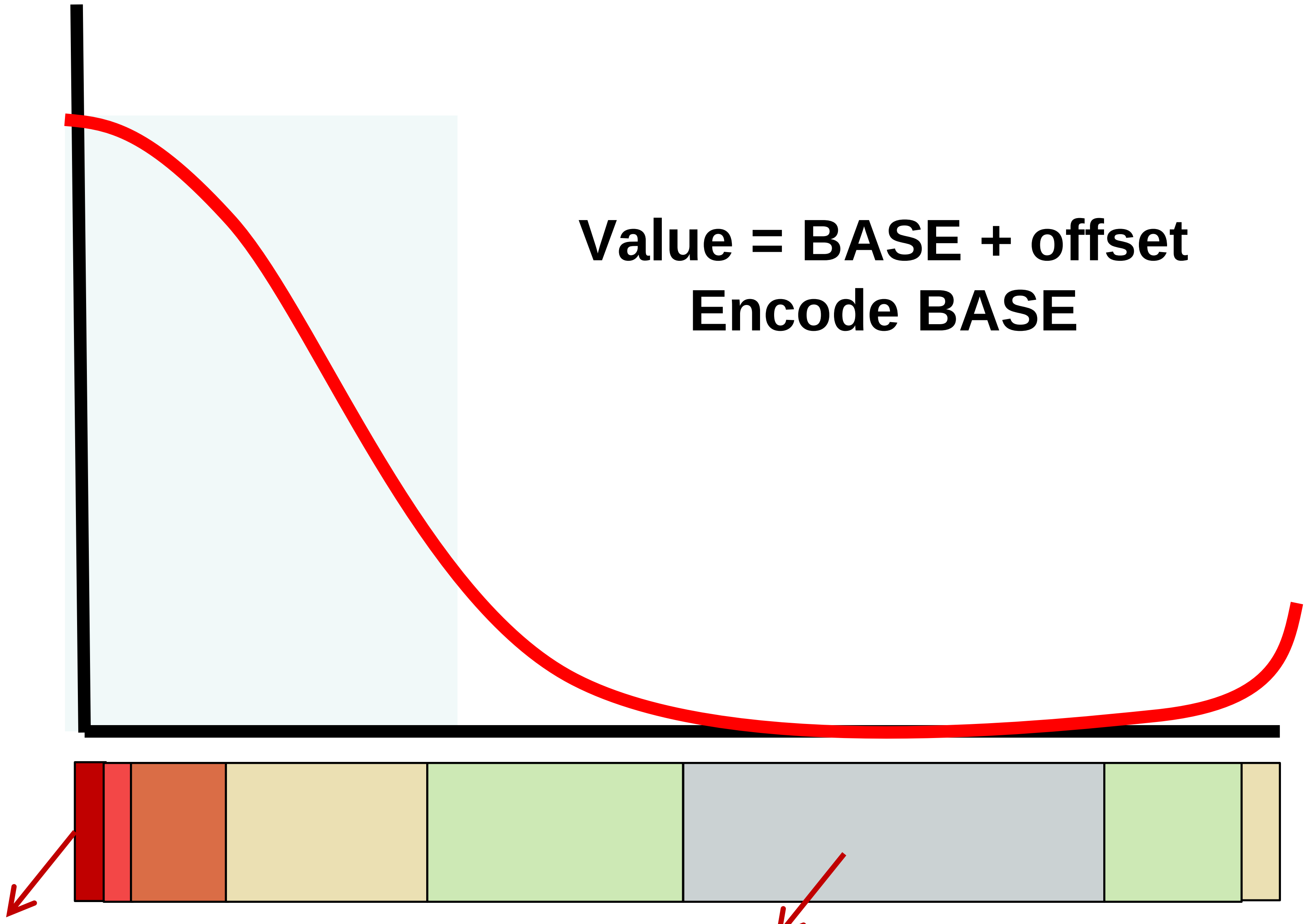
APACK Key Idea

0000 0000 = 0

0101 0111 =

01000 0000+1 0111

Apack Key Idea



Value = BASE + offset
Encode BASE

0x00 → 0x00,0b
0x01 → 0x00,1b

0x8F → 0x80, 001111b
0x91 → 0x80, 010001b

Table Generation: Done in Advance

IDX	v_min	v_max	OL	tlow	thigh	p
0	0x00	0x03	2	0x000	0x1EB	0.4795
1	0x04	0x07	2	0x1EB	0x229	0.0605
2	0x08	0x0F	3	0x229	0x238	0.0146
3	0x10	0x3F	6	0x238	0x23A	0.0020
4	0x40	0x4F	4	0x23A	0x23A	0.0000
5	0x50	0x5F	4	0x23A	0x23A	0.0000
6	0x60	0x6F	4	0x23A	0x23A	0.0000
7	0x70	0x7F	4	0x23A	0x23A	0.0000
8	0x80	0x8F	4	0x23A	0x23A	0.0000
9	0x90	0x9F	4	0x23A	0x23A	0.0000
10	0xA0	0xAF	4	0x23A	0x23A	0.0000
11	0xB0	0xBF	4	0x23A	0x23A	0.0000
12	0xC0	0xCF	4	0x23A	0x23A	0.0000
13	0xD0	0xF3	6	0x23A	0x23C	0.0020
14	0xF4	0xFB	3	0x23C	0x276	0.0566
15	0xFC	0xFF	2	0x276	0x3FF	0.3838

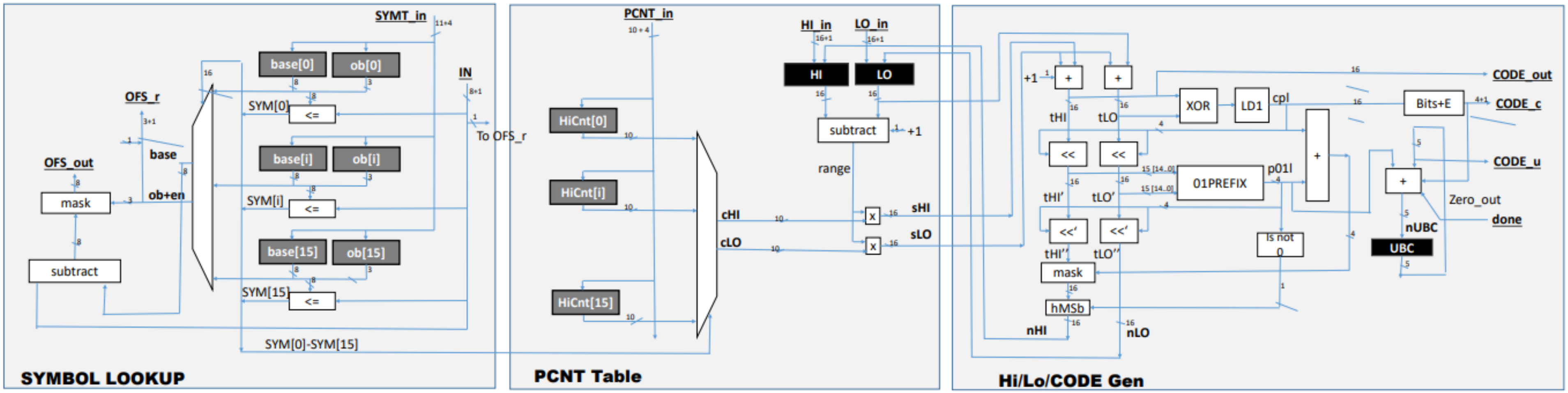
Table Generation: Done in Advance

IDX	v_min	v_max	OL	tlow	thigh	p
0	0x00	0x03	2	0x000	0x1EB	0.4795
1	0x04	0x07	2	0x1EB	0x229	0.0605
2	0x08	0x0F	3	0x229	0x238	0.0146
3	0x10	0x3F	6	0x238	0x23A	0.0020
4	0x40	0x4F	4	0x23A	0x23A	0.0000
5	0x50	0x5F	4	0x23A	0x23A	0.0000
6	0x60	0x6F	4	0x23A	0x23A	0.0000
7	0x70	0x7F	4	0x23A	0x23A	0.0000
8	0x80	0x8F	4	0x23A	0x23A	0.0000
9	0x90	0x9F	4	0x23A	0x23A	0.0000
10	0xA0	0xAF	4	0x23A	0x23A	0.0000
11	0xB0	0xBF	4	0x23A	0x23A	0.0000
12	0xC0	0xCF	4	0x23A	0x23A	0.0000
13	0xD0	0xF3	6	0x23A	0x23C	0.0020
14	0xF4	0xFB	3	0x23C	0x276	0.0566
15	0xFC	0xFF	2	0x276	0x3FF	0.3838
	8b		3b		10b	

$0.11010101010101010\ldots1011110101_{(2)}$

$0.11010101010101010\ldots101111010111001_{(2)}$

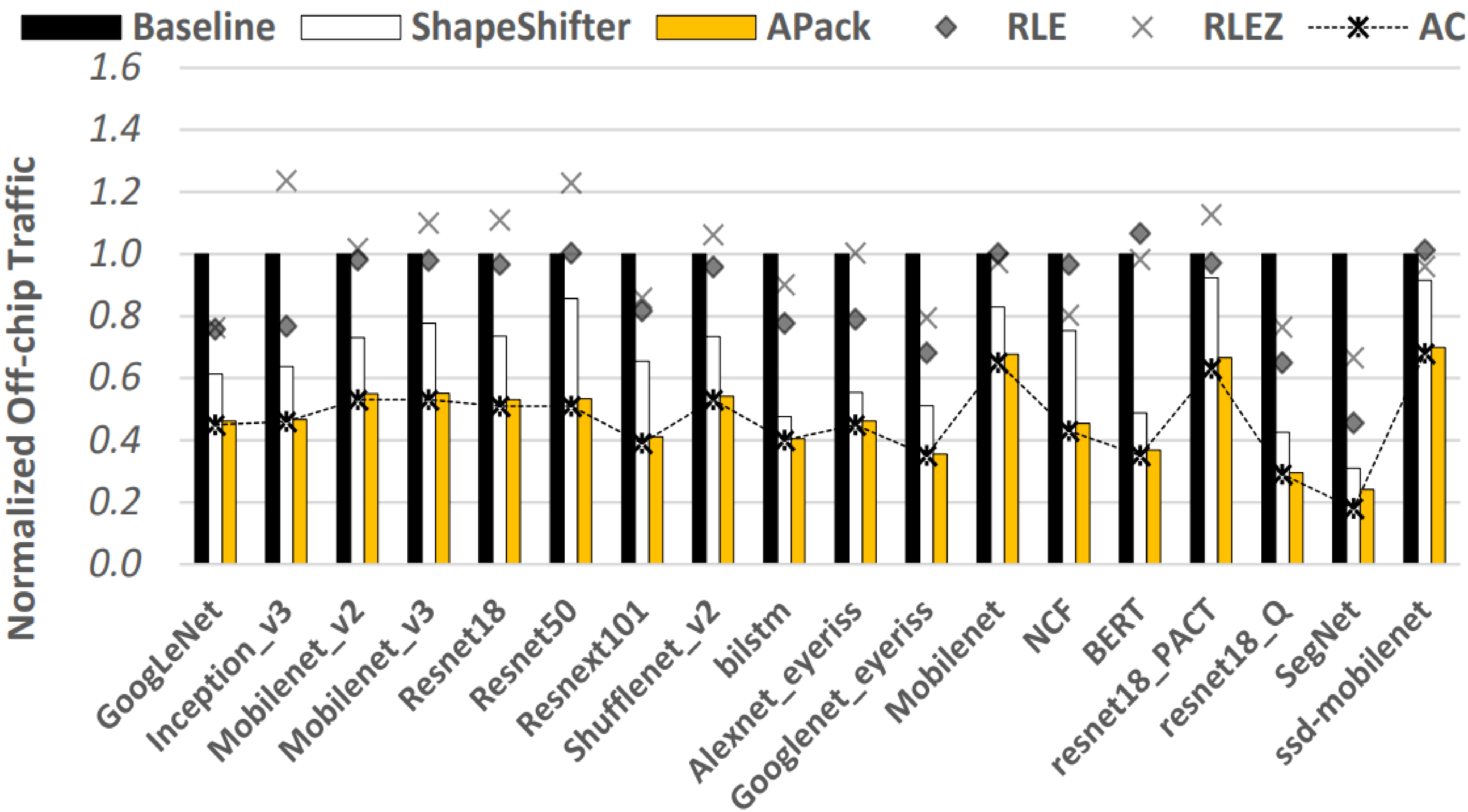
APACK Encoder



Hardware

- Fixed-Point
- 10b x 16b Multiplications and 16b comparisons
- A few leading 1
- One value per “cycle”
- Use multiple to sustain BW needed
- Externally: Sequential Streams

APACK Activations



Mokey

Enabling Narrow Fixed-Point Inference

for Out-of-the-Box Floating-Point Transformer Models

Challenges

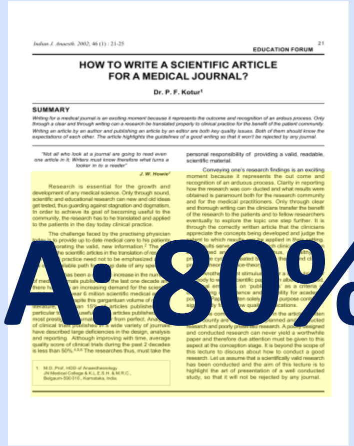
Weights

2018
1.2GB

2021
2TB

Activations

A: 5%



A: 80%

Challenges

Weights

2021

2TB

Activations

A:80%

Memory:

Performance & Energy Bottleneck

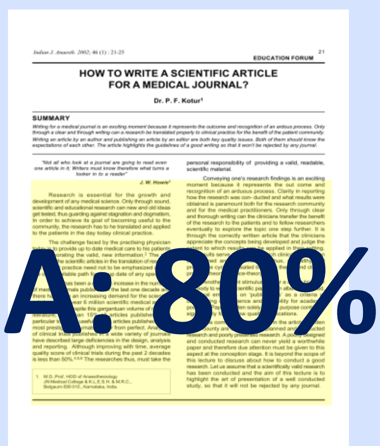
Challenges

Weights

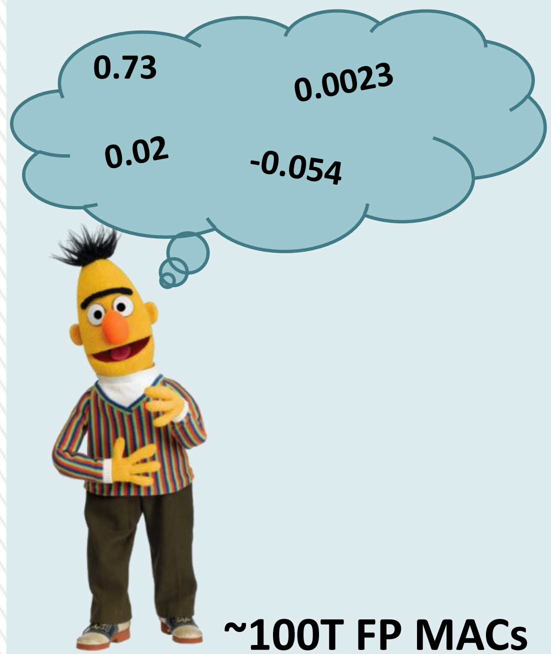
2018
1.2GB

2021
2TB

Activations

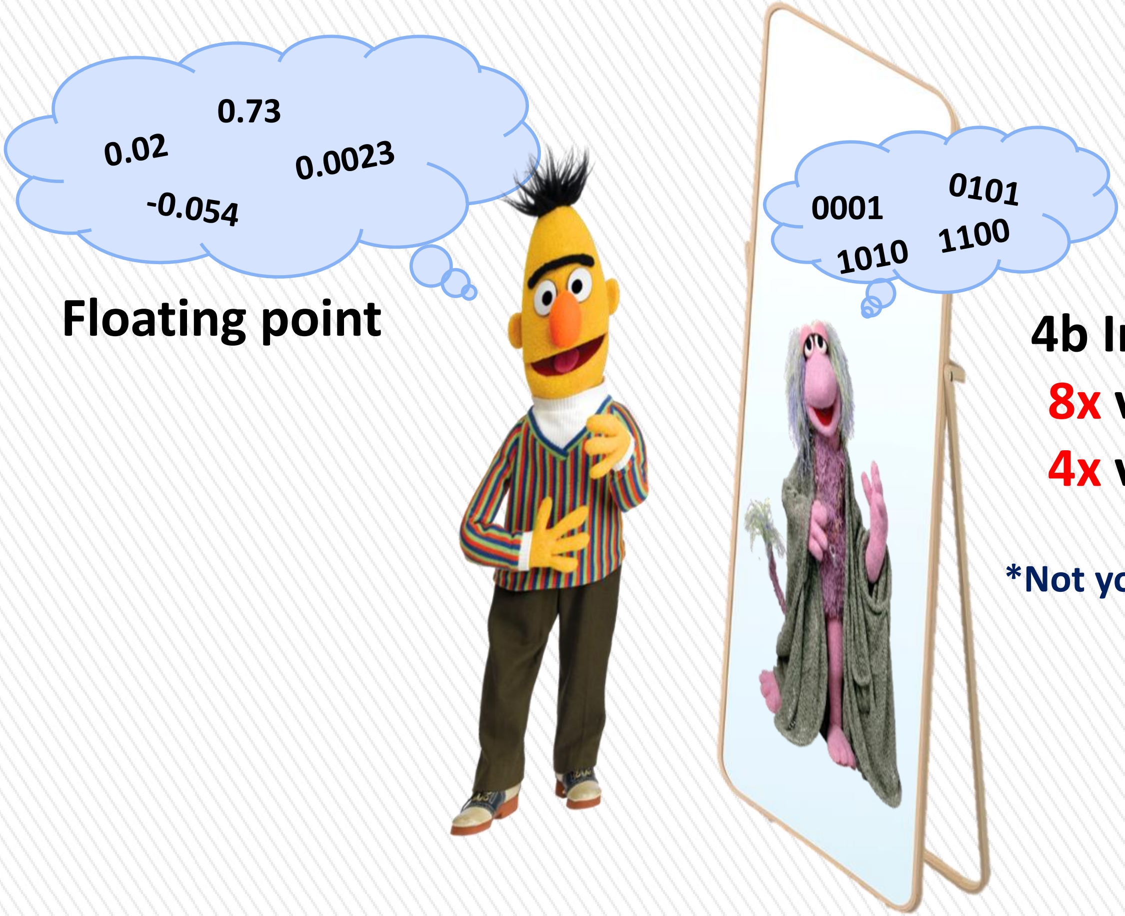


FP Compute



~100T FP MACs

Mokey: BERT's Better Self

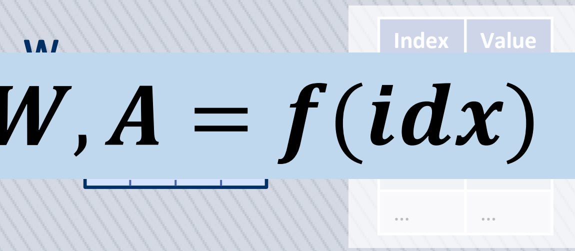


4b Int index
8x vs FP32
4x vs FP16

***Not your typical 4b quantization ;)**

Mokey

4-bit Quantization: W+A



$$W, A = f(idx)$$

Post-training

+= A x W. ➔ Count idx

Fixed-point compute

Mokey HW Accelerator

Vs. Tensor Cores: ***15x*** Faster + ***100x*** Energy Efficient

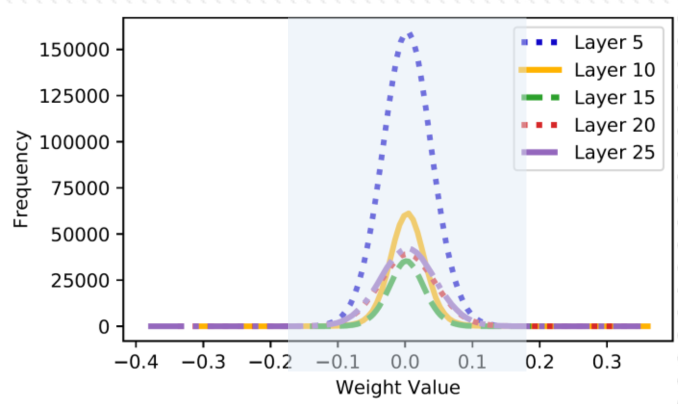
Mokey Memory Compression

For Tensor Cores:

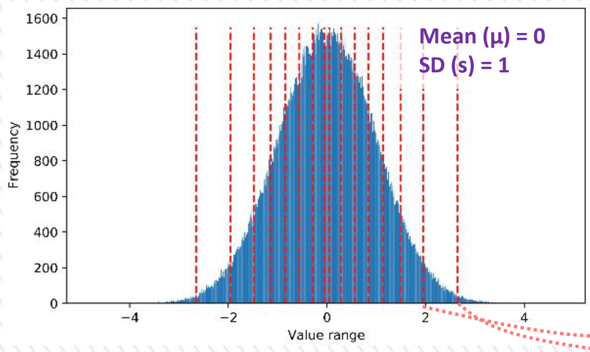
Off-chip Only: ***4x*** Faster + ***8x*** Energy Efficient

Off- and on-chip*: ***10x*** Faster + ***50x*** Energy Efficient

A Dictionary for All Layers



Reference Distribution



Golden Dict. (GD)

Index	Value
I	-2.7
II	-1.98
…	…
VI	1.98
XVI	2.7

Scale and **Shift** is All You Need

Inference Computation

Original

A = 0.2 W = 0.7

A×W += 0.2 × 0.7 = 0.14

Dictionary Quant.

A = I W= II

Index	Value
I	0.2
II	0.7
III	1.1
IV	1.4
…	…

A × W += I × II

A × W += 0.2 × 0.7 = 0.14

Mokey Quant.

A = I W= II

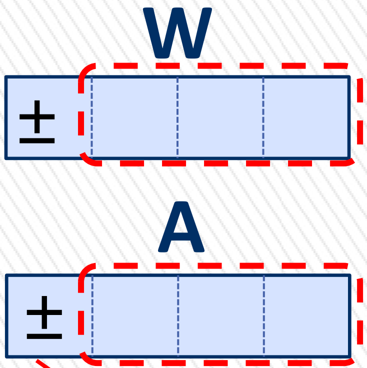
How?!

A × W += I × II = 0.14

Values Format

Index	Value
I	0.05
II	0.35
…	…
VI	1.97
VII	2.6

Golden Dict. (GD)



$$\text{Values} = \pm(a^{i} + b)$$

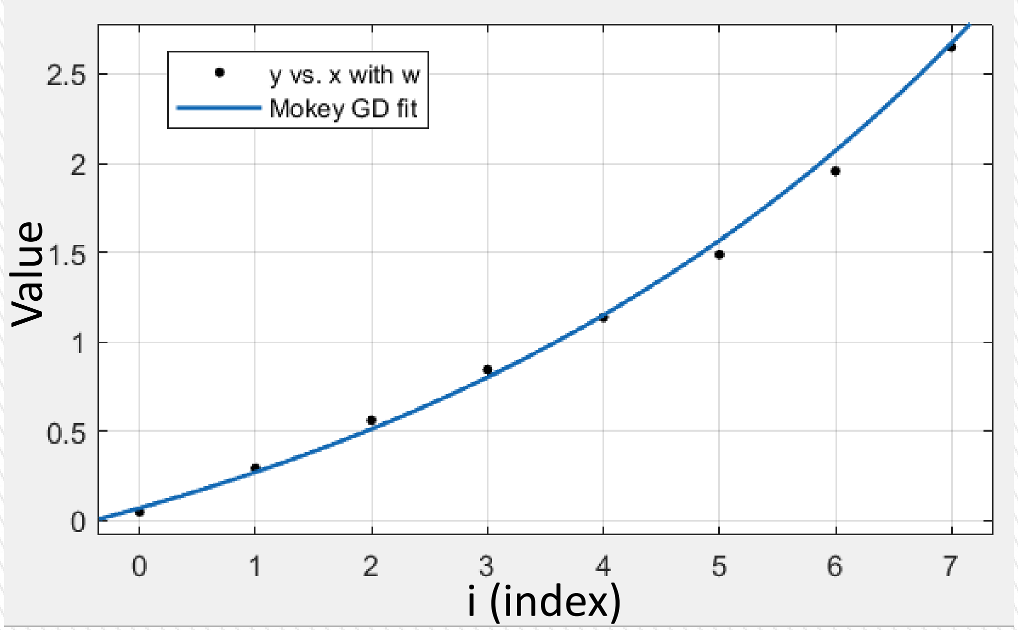
per value

Fixed

Exponential Function

Index	Value
I	0.05
II	0.35
…	…
VI	1.97
VII	2.6

Golden Dict. (GD)



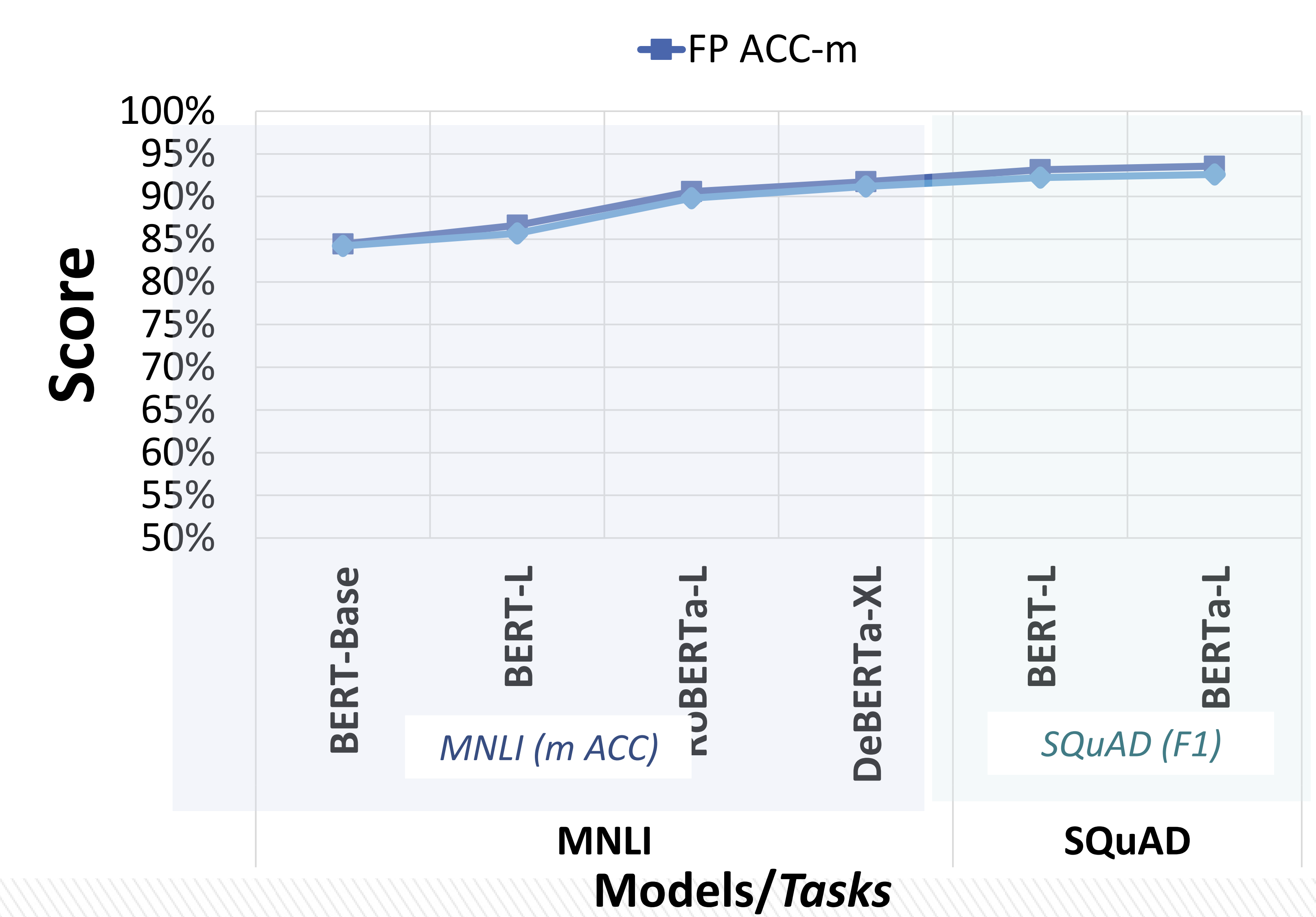
$GD = a^i + b$

Evaluation

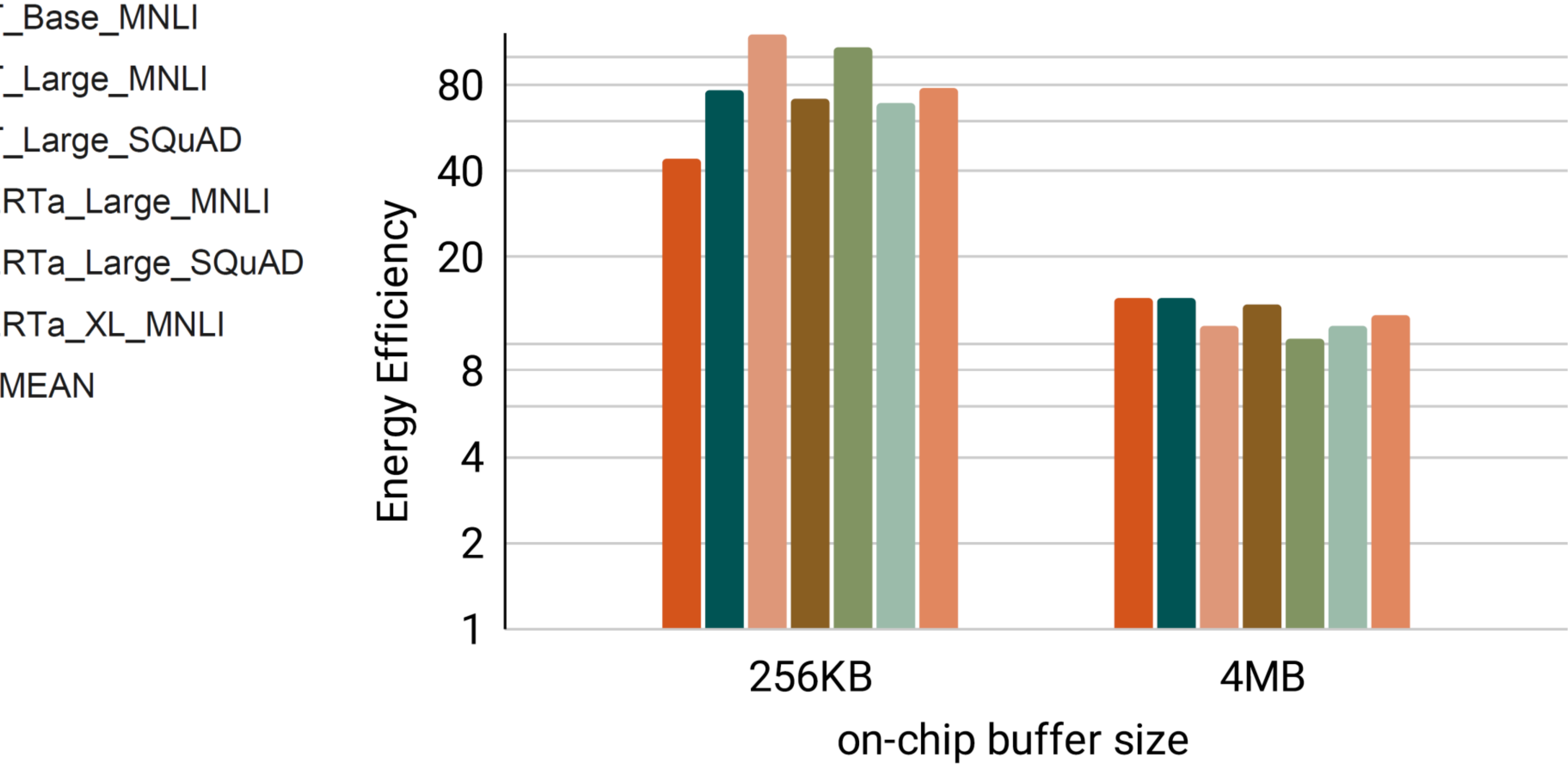
- FP16 Tensor Cores baseline
- Wide range of on-chip buffers
- 110M - 750M parameter models

- Custom cycle accurate simulator.
  - DRAMsim3: Dual Channel DDR4-3200
- On-chip Memory: CACTI
- Synthesis: Synopsis Design Compiler
  - 65nm TSMC – 1Ghz
- Layout: Cadence Innovus
- Signal Activity: Modelsim
- Power Estimation: Cadence Innovus

Quantization Accuracy



Accelerator Energy Efficiency



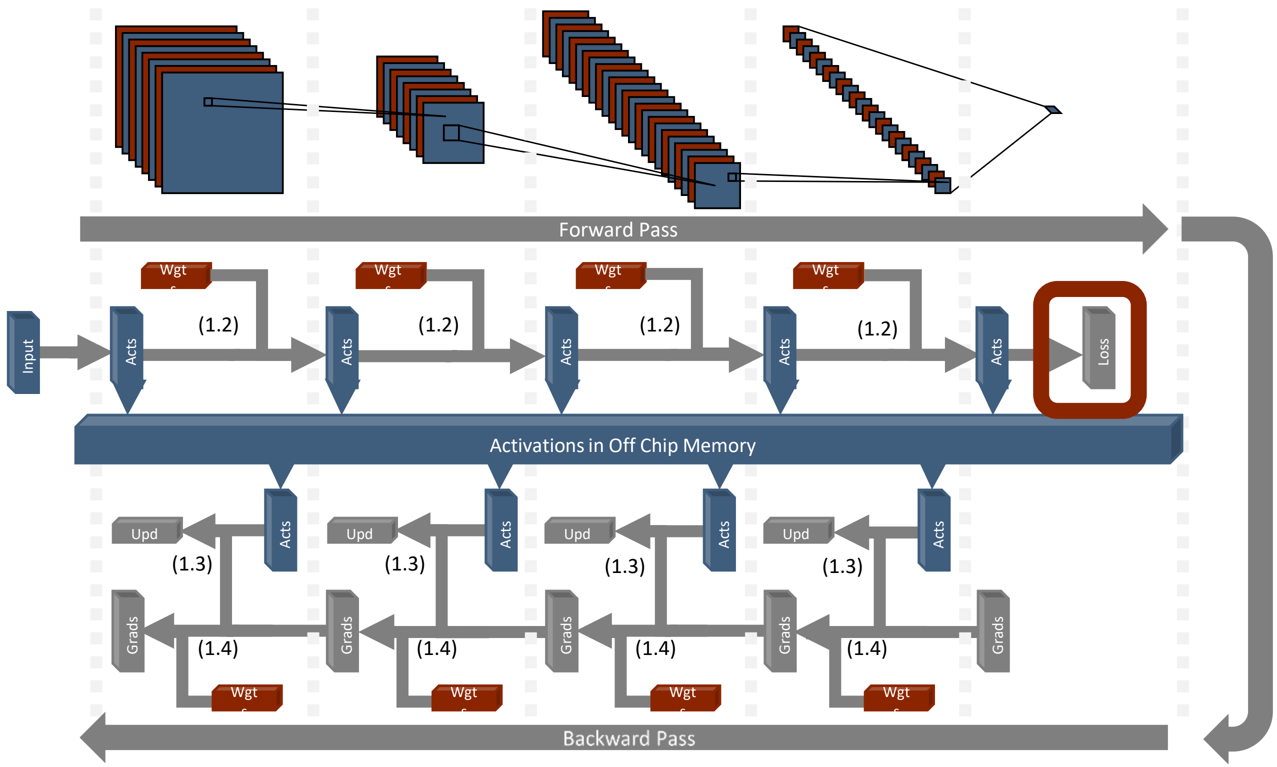
Memory Compression and more in paper ☺

Schrödinger's FP
Dynamic Adaptation of Floating-Point Containers During Training

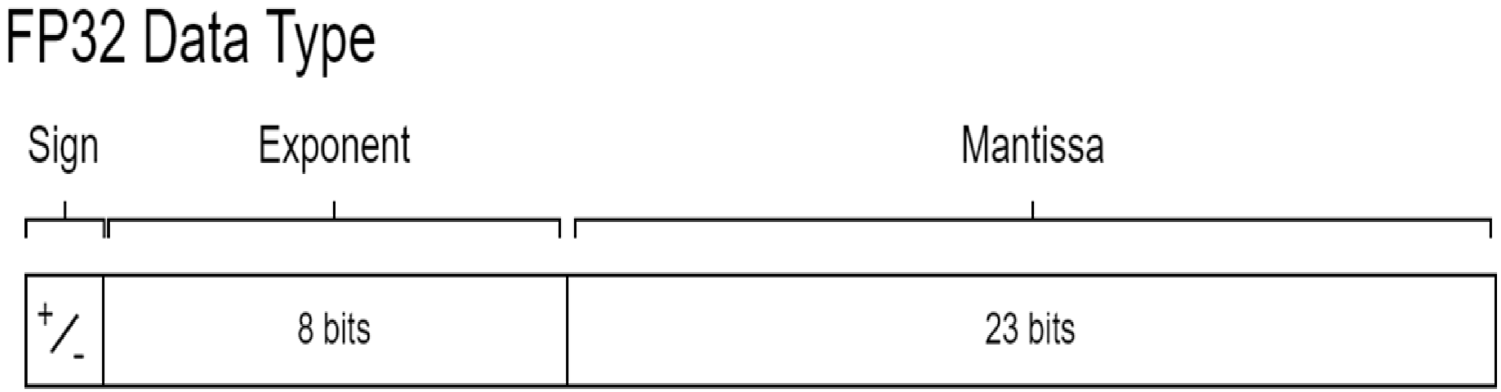
Gradient Descent – Overview

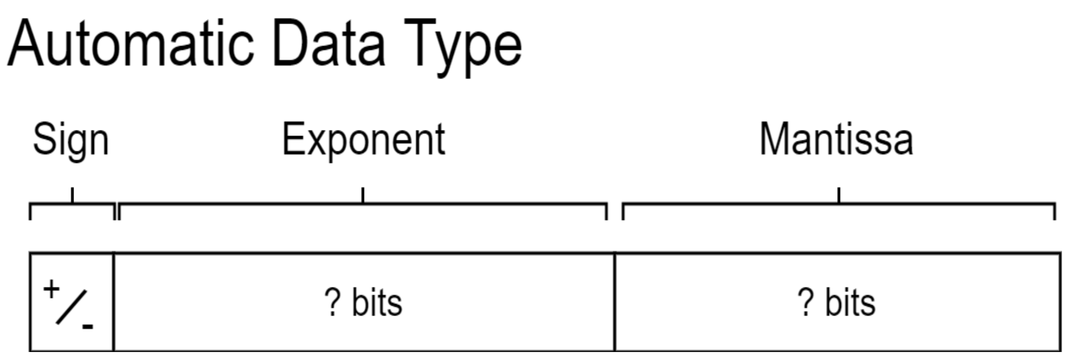
- Loss function

$$w_i^l = w_i^l - LR \times \frac{\partial L}{\partial w_i^l}$$

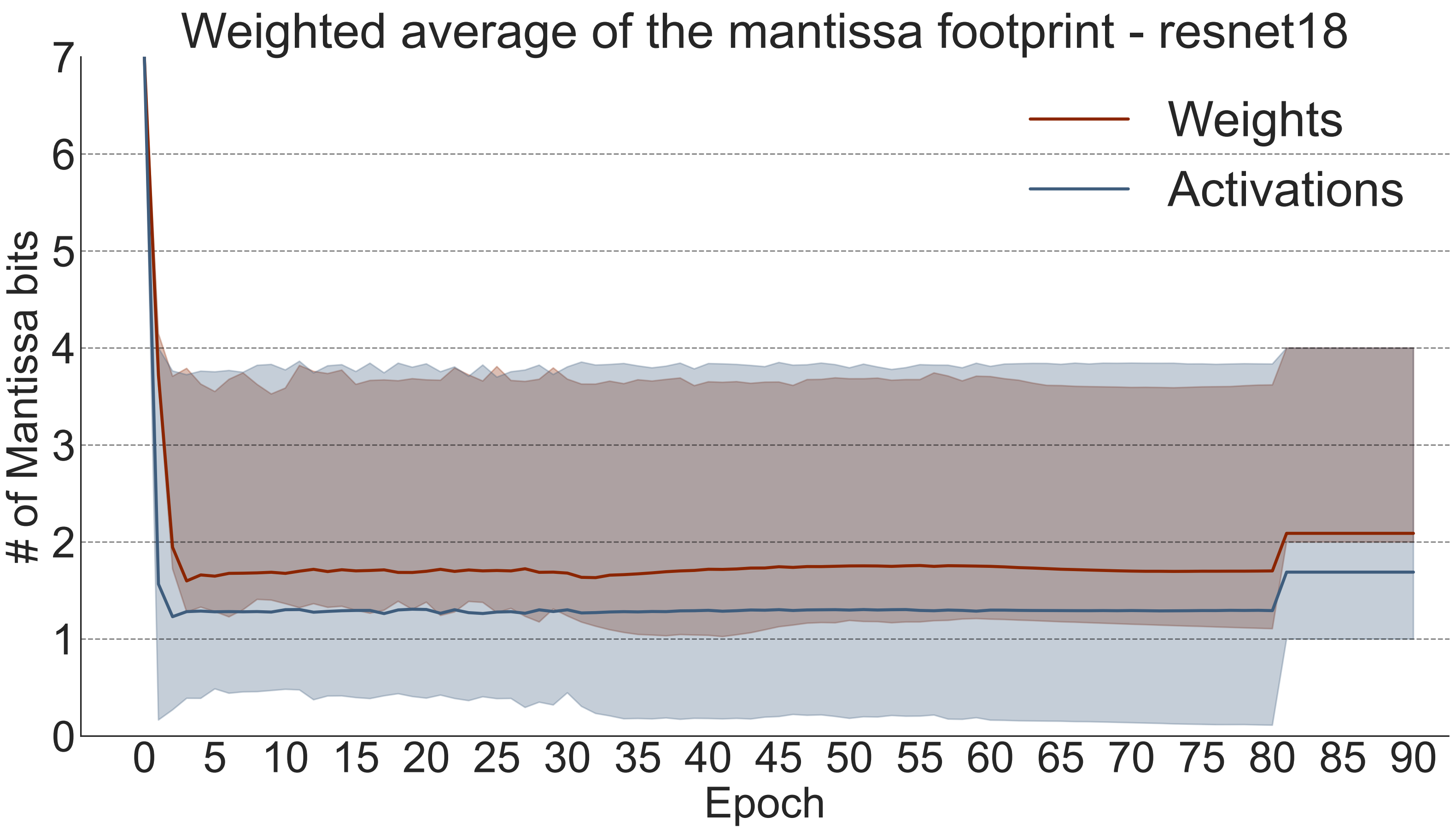


The Precision Problem

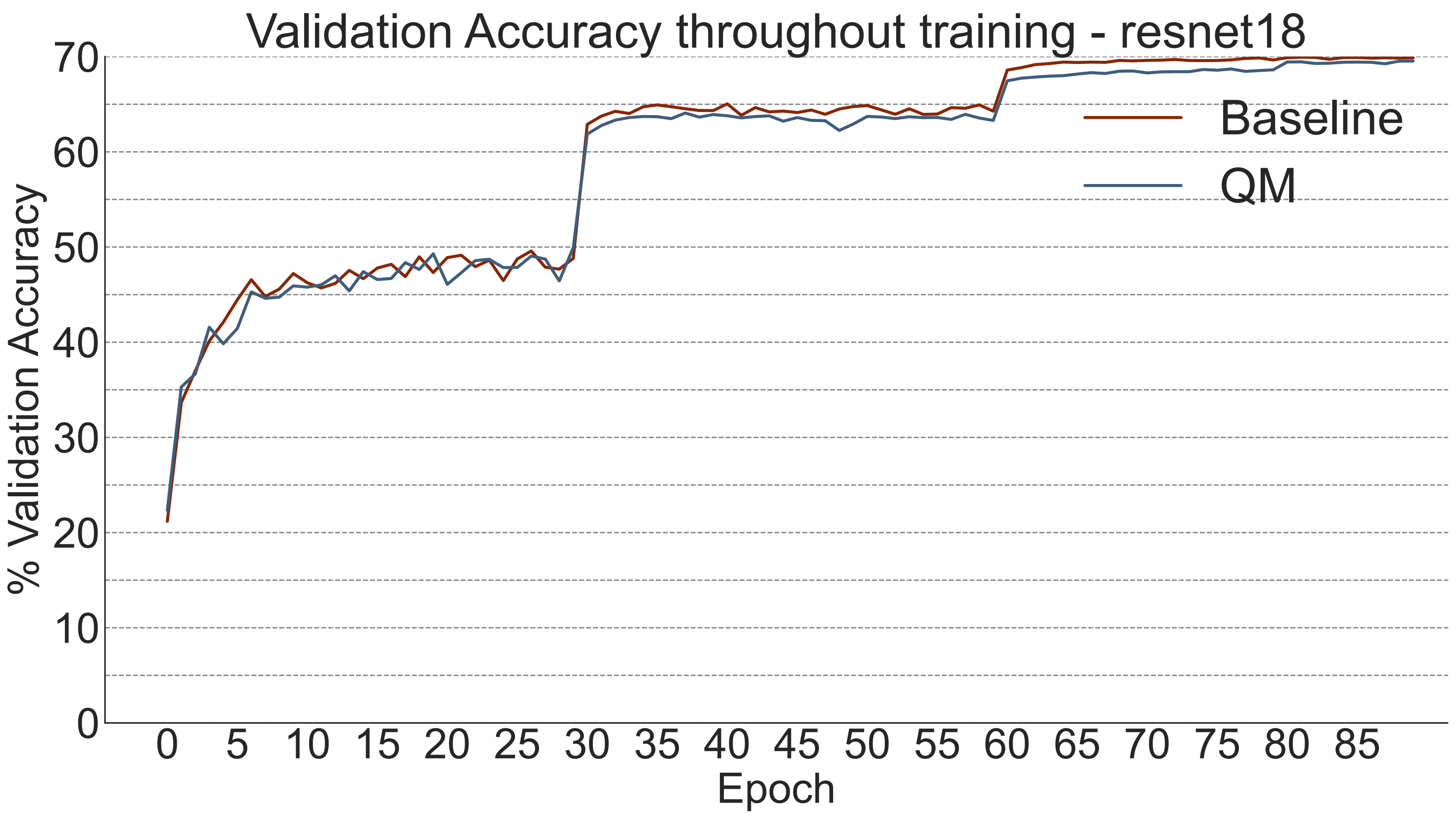




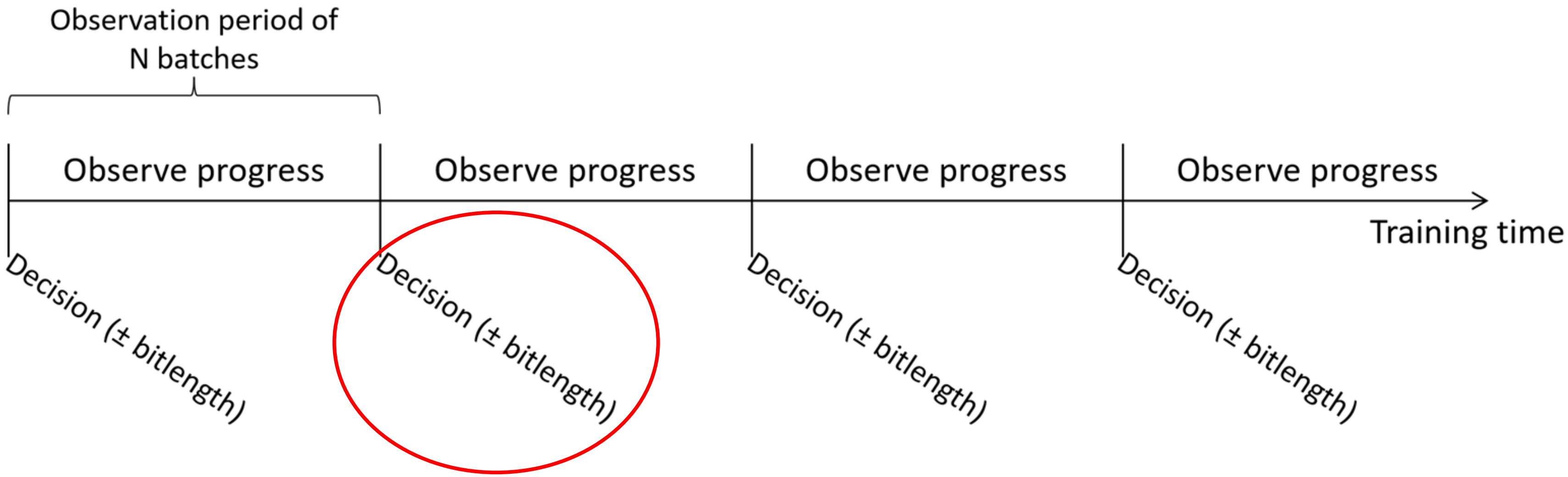
Datatype – Does it work?



Datatype – Does it work?



BitChop


Observation period of
N batches
Observe progress
Observe progress
Observe progress
Observe progress
Training time
Decision (± bitlength)
Decision (± bitlength)
Decision (± bitlength)
Decision (± bitlength)

BitChop - Moving Average Policy

- Exponential decay factor and dynamic threshold:

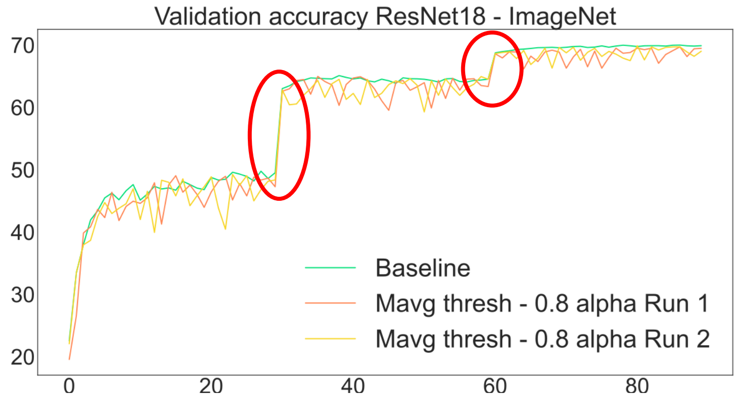
$$Mavg_{i+1} = Mavg_i + \alpha \times (L_i - Mavg_i)$$

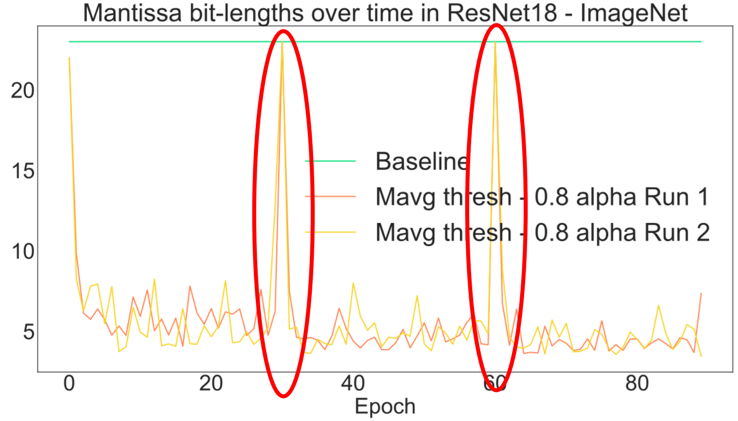
- Full precision on learning rate change
- 4 bits mantissa on average
  - Slight volatility in accuracy
- 75% mantissa footprint reduction on average

$$ErrAvg_i = \frac{\sum_{n=i-N}^{i-1} \frac{|Mavg_n - L_n|}{L_n}}{N}$$

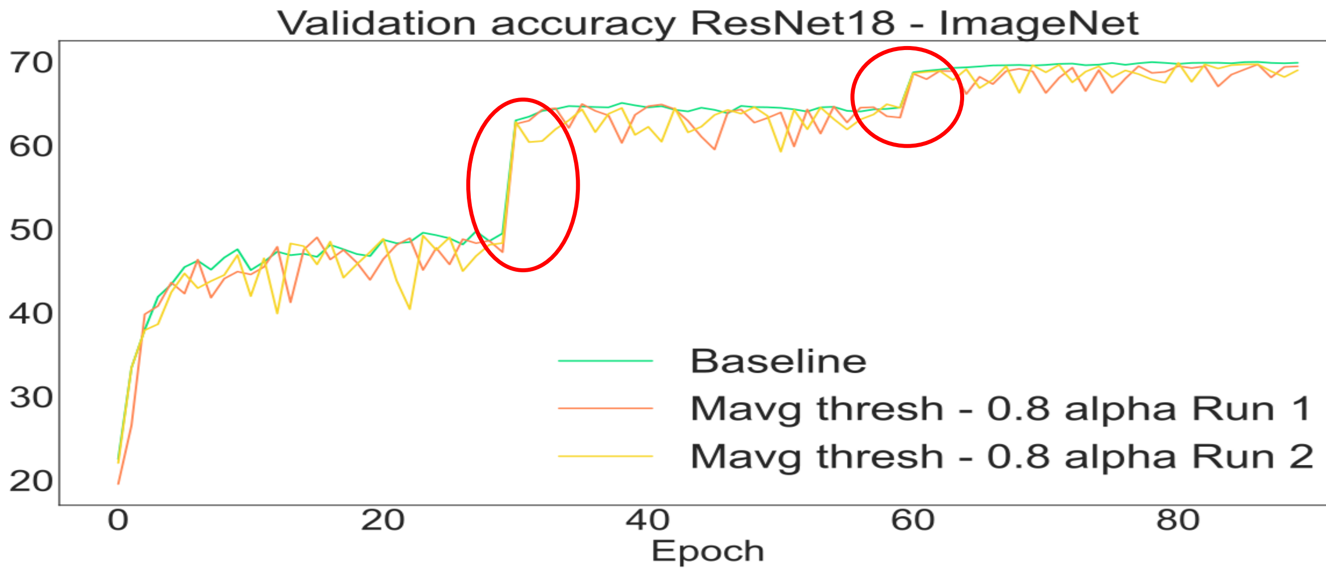
$$\epsilon_i = Mavg_i \times ErrAvg_i$$

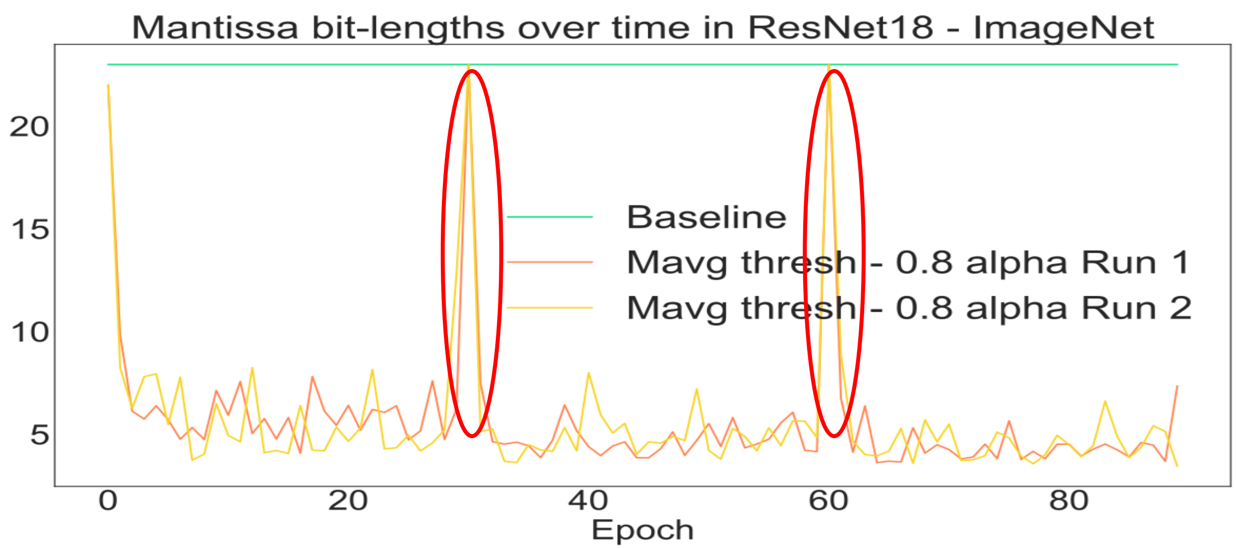
$$bitlength_{i+1} = \begin{cases} bitlength_i - 1, & \text{when } Mavg_i > L_i + \epsilon \\ bitlength_i, & \text{when } L_i - \epsilon_i \le Mavg_i \le L_i + \epsilon_i \\ bitlength_i + 1, & \text{when } Mavg_i < L_i - \epsilon \end{cases}$$





BitChop - Moving Average Policy





Datatype – Does it work?

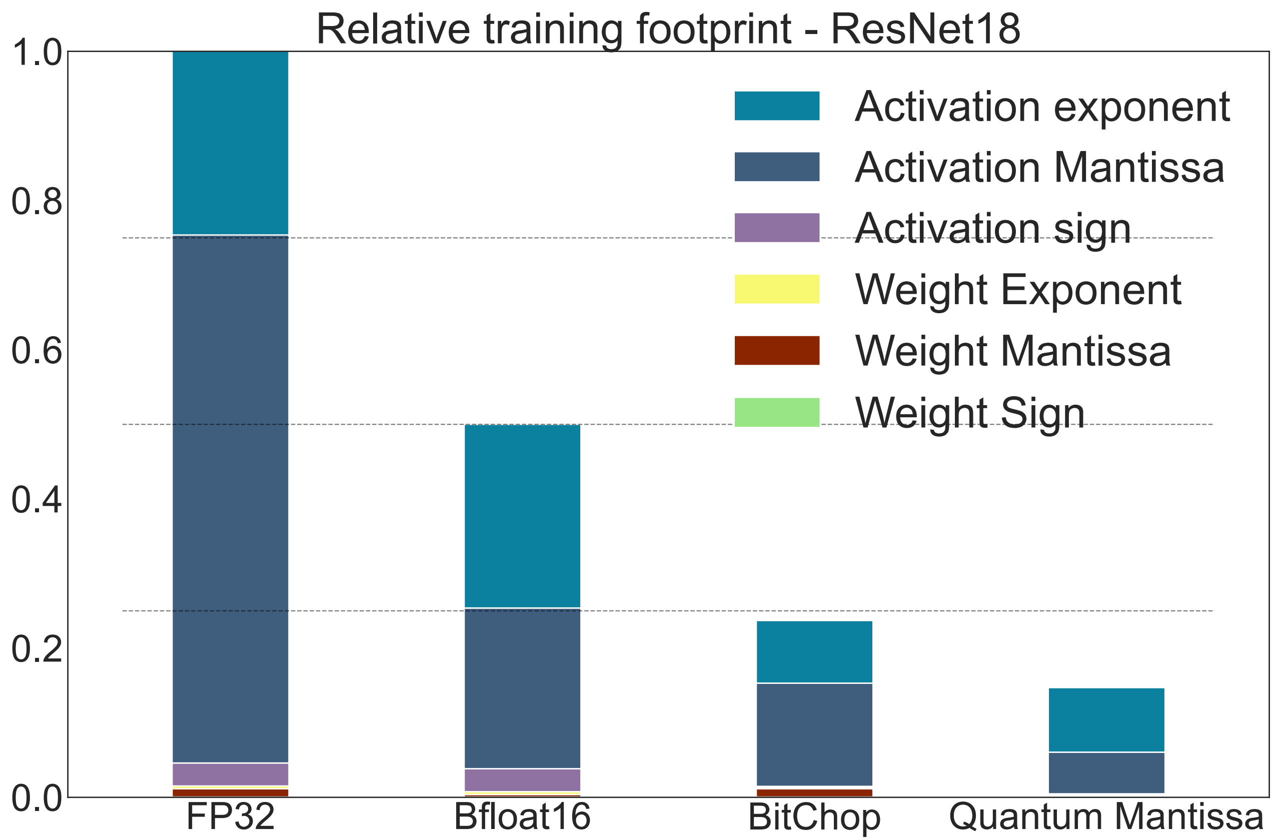


Table 2: Performance and Energy Efficiency gains in comparison w/ FP32

Network	Performance			Energy Efficiency		
	Bfloat 16	SFP_{QM}	SFP_{BC}	**Bfloat 16**	SFP_{QM}	SFP_{BC}
ResNet18	1.53×	2.30×	2.09×	2.00×	6.12×	4.22×
MobileNet V3 Small	1.72×	2.37×	2.14×	2.00×	3.95×	3.60×

Summary

- HW and SW that improves performance and energy efficiency
- w/o requiring any changes to the models
- Rewards further optimizations
- Apack
- Mokey
- Schrödinger's FP