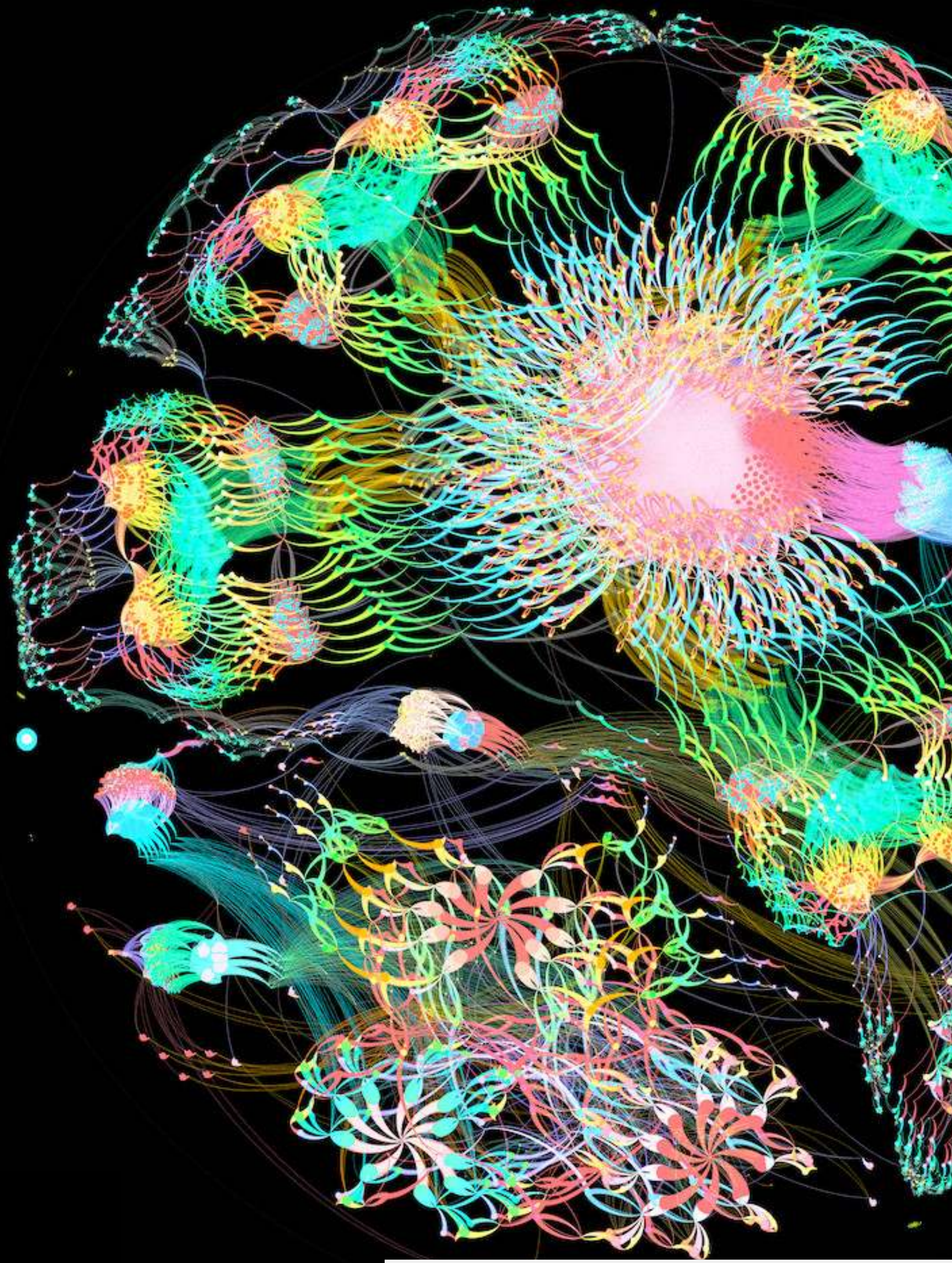
GRAPHCORE

SW/HW CO-OPTIMIZATION ON THE IPU: AN MLPERF™ CASE STUDY

Dr. Mario Michael Krell



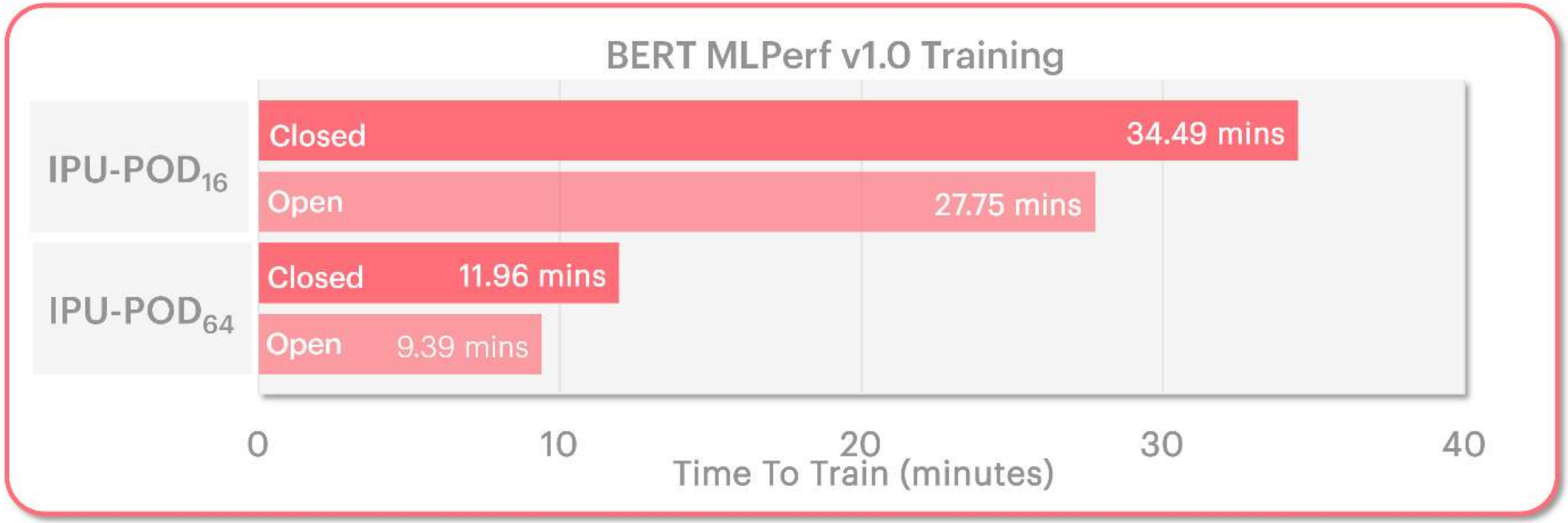
Autoencoder network visualization from POPLAR®

OUTLINE

- Introduction
- Packed BERT
- Parallelism and recomputation
- Other features
- Conclusion

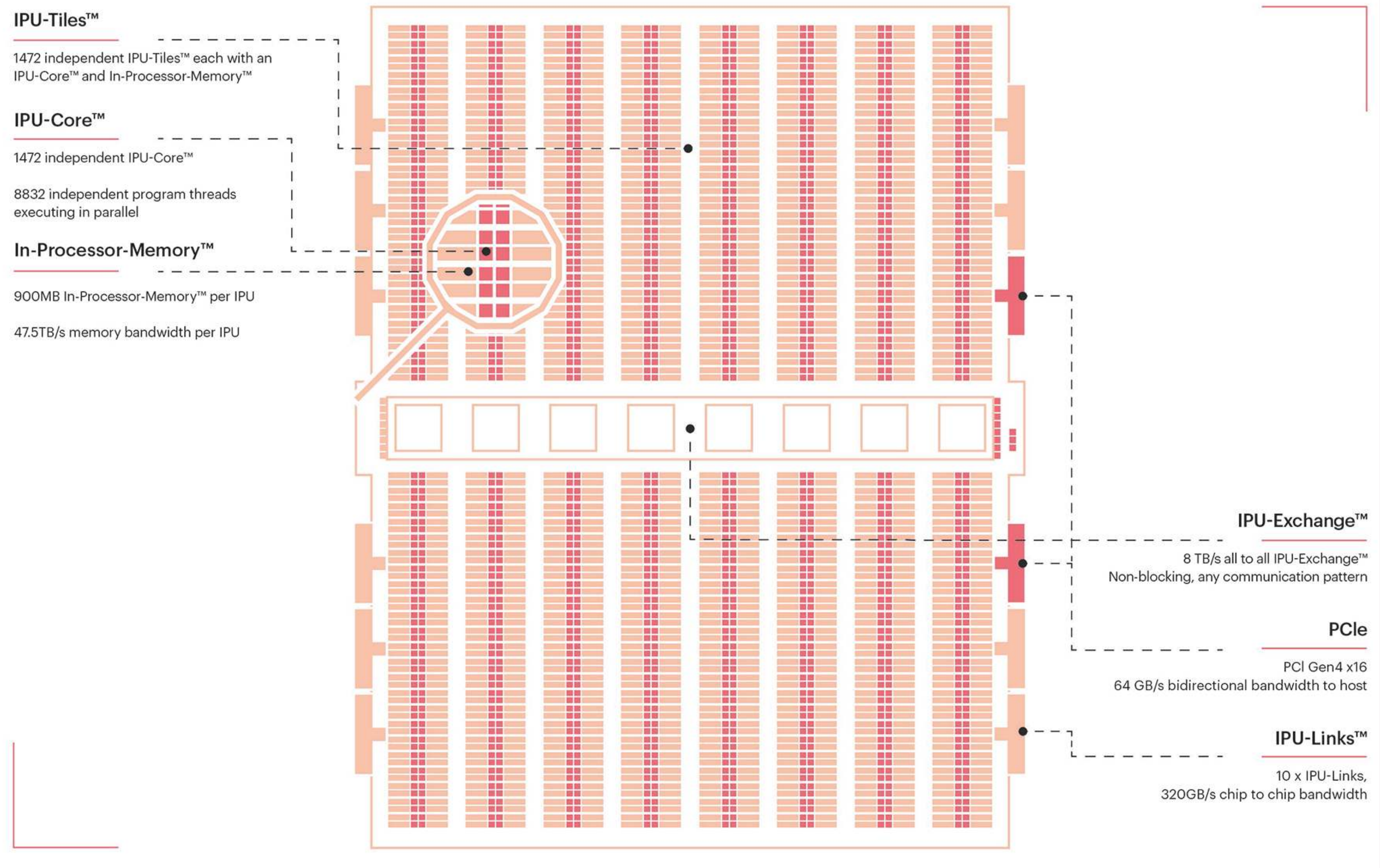
INTRODUCTION

RAISING THE BAR: GRAPHCORE'S FIRST MLPERF™ RESULTS



MLPerf v1.0 Training Results | MLPerf ID: 1.0-1025, 1.0-1026, 1.0-1027, 1.0-1028, 1.0-1098, 1.0-1099

MK2 GC200 IPU



- Massively parallel MIMD
- Supports
  - FP32,
  - FP16, and
  - FP16 with stochastic rounding

Image: https://www.graphcore.ai/products/ipu

IPU-Machine: M2000

4 x Colossus™ GC200 IPU
1 petaFLOPS AI compute
Up to 526GB Exchange Memory™
2.8Tbps IPU-Fabric™

Each Colossus™ GC200 IPU

59.4Bn transistors, TSMC 7nm @ 823mm2
250 teraFLOPS AI compute
1472 independent processor cores
8832 separate parallel threads

IPU-Gateway SoC

Arm Cortex-A quad-core SoC
Super low latency IPU-Fabric™ interconnec

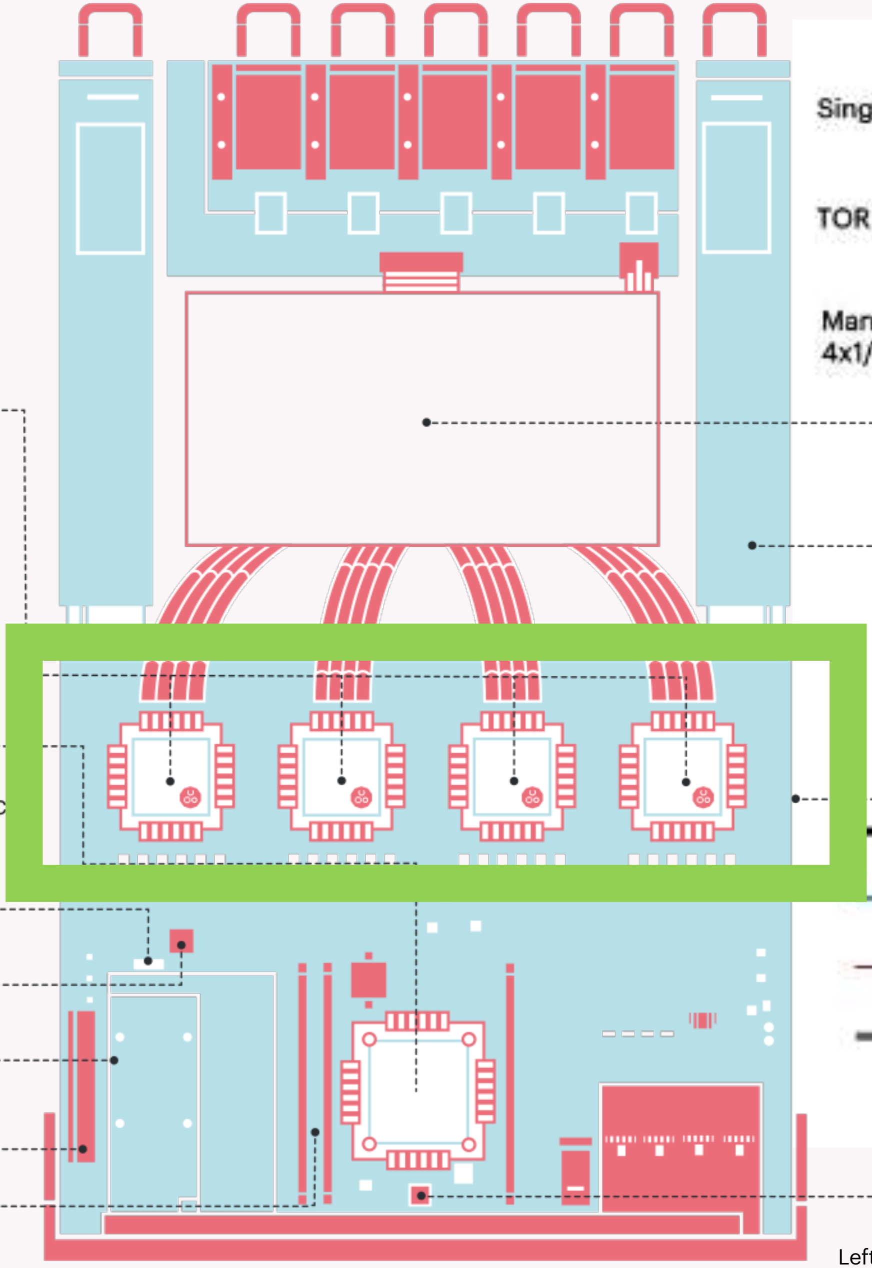
M.2 Connector

Board Management Controller

M.2 Slot

PCIe FH3/4L G4x8 Slot (RNIC/SmartNIC)

DDR4 DIMM DRAM x 2
For model states streaming



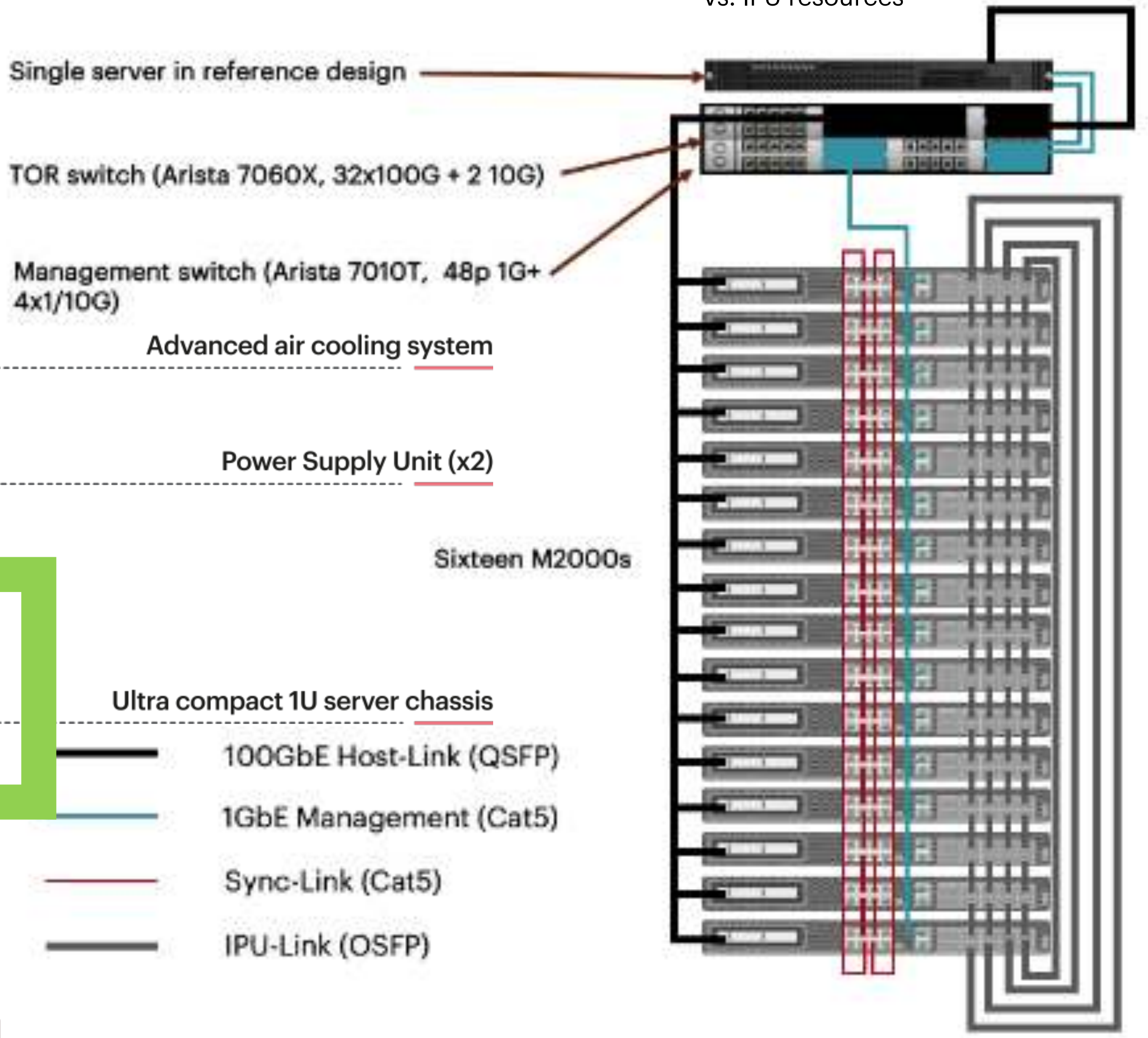
Advanced air cooling system

Power Supply Unit (x2)

Ultra compact 1U server chassis

eMMC 32G Flash device

Network-based disaggregated architecture for host vs. IPU resources



POD 64

Left: https://www.graphcore.ai/hubfs/IPU-M%20SVG_2.svg
Right: https://docs.graphcore.ai/projects/ipu-pod64-datasheet/en/latest/_static/GC-000477-DS-6-IPU-POD64-datasheet.pdf

MORE DETAILS ON HARDWARE?

Visit presentation by Graphcore CTO Simon Knowles
on Tuesday or visit our webpage

OUR MLPERF™ IMPLEMENTATION: FULL STACK SOFTWARE/HARDWARE CO-OPTIMIZATION

- Algorithm: packed BERT
- Software: Parallelism + Recompute
- Hardware: FP16 + stochastic rounding, low batch size

ALGORITHMIC IMPROVEMENT: TOWARDS PACKED BERT

MOTIVATION (MODEL)

- BERT
  - Transformer based model
  - Bidirectional encoder with self-attention
- Efficient use of ALU on the HW requires
  - Batching
  - Padding to same length
- Trade-off: wasted compute for speed-up from the parallelism
- **How much compute are we actually wasting?**

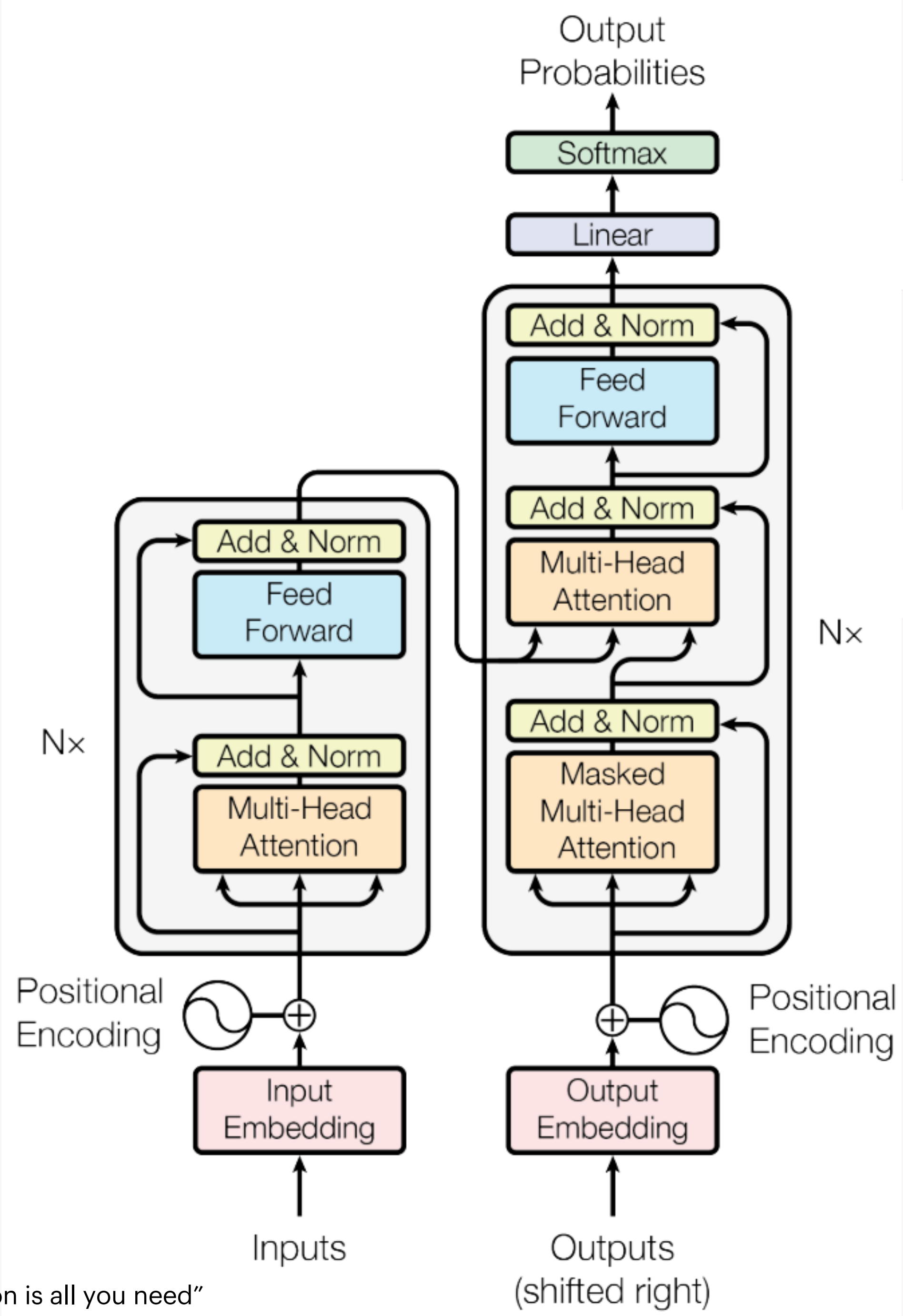
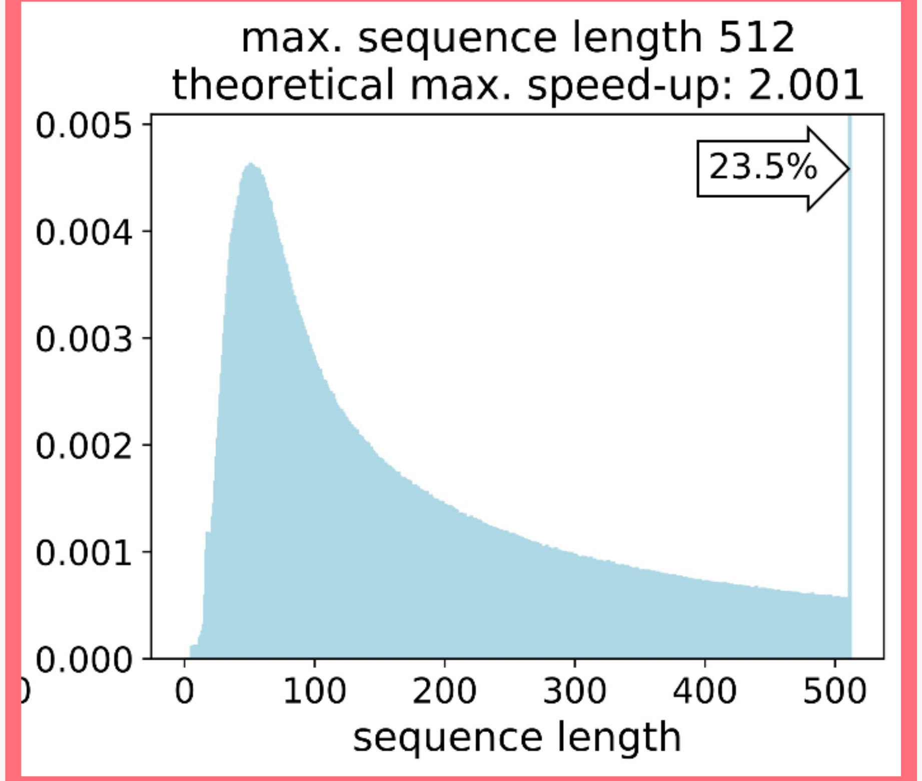


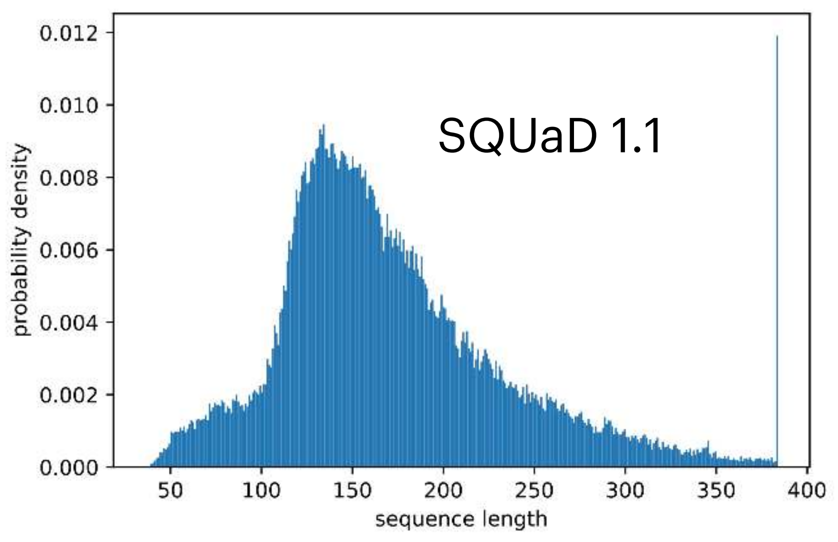
Image: http://arxiv.org/abs/1706.03762 "Attention is all you need"

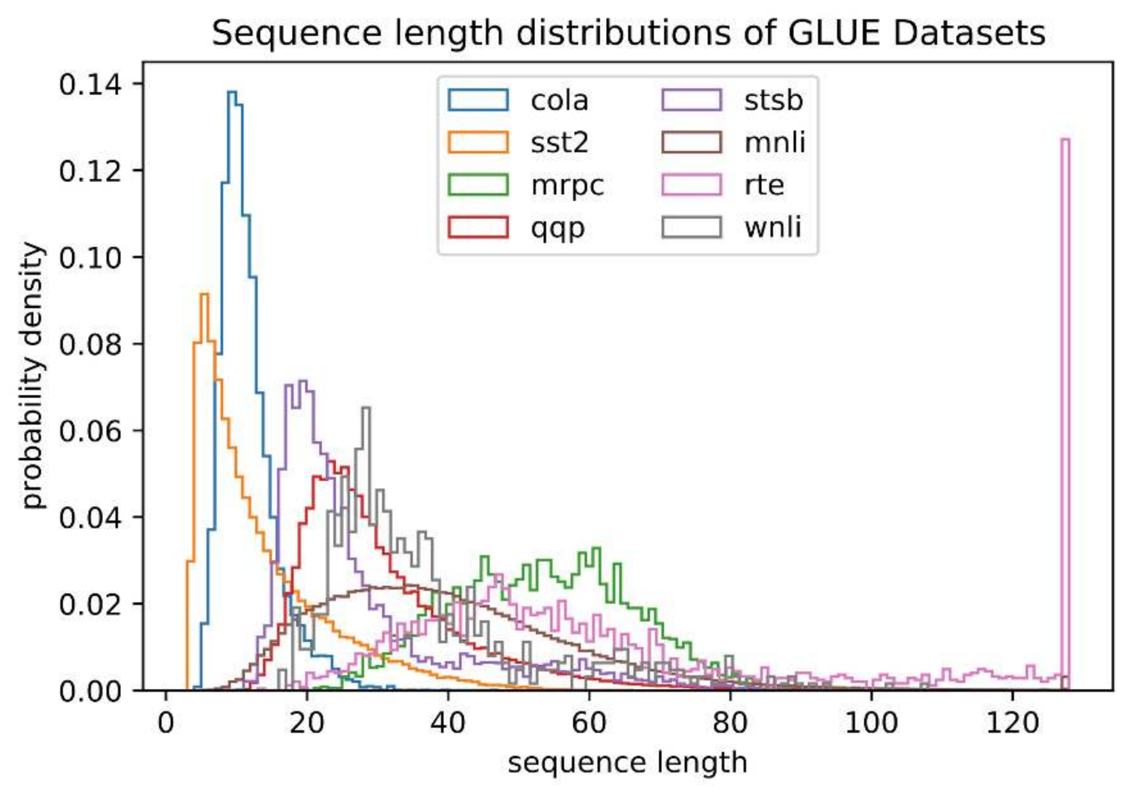
MOTIVATION (DATA)

- Wikipedia
- SQUaD 1.1
- GLUE



Wikipedia





SOLUTION

Packing in three core steps:

1. Pack the data
2. Adjust the model
3. Adjust the hyperparameters

PACKING

SPFHP video from: https://towardsdatascience.com/introducing-packed-bert-for-2x-faster-training-in-natural-language-processing-eadb749962b1

- Packing N sequences is an NP hard problem even when combining 3 sequences
- Impossible on raw sequences -> use histogram of sequences

- Online shortest-pack-first histogram-packing (SPFHP)
- Offline Non-negative least-squares histogram-packing (NNLSHP)

ADJUSTING THE MODEL

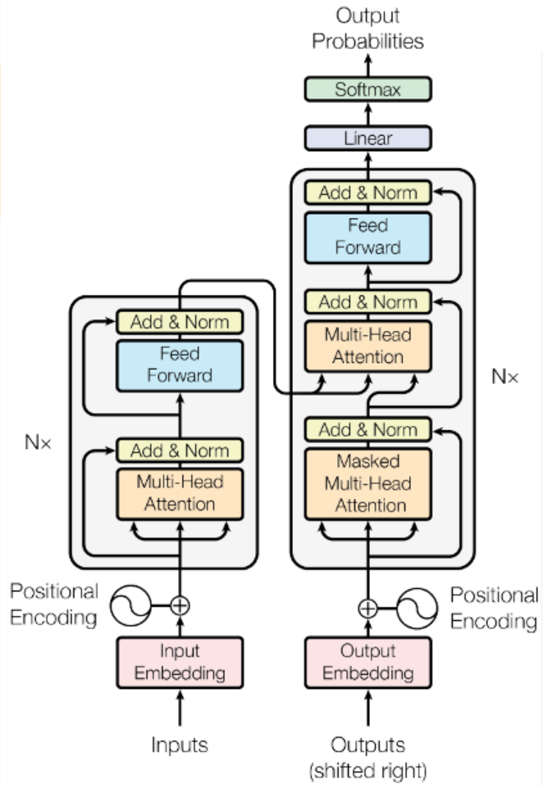
Mandatory: Masking attention

```
mask = np.array([[1, 1, 1, 2, 2]])  # input
zero_one_mask = tf.equal(mask, mask.T)  # 0, 1 mask
# for use with softmax:
softmax_mask = tf.where(zero_one_mask, 0, -1000)
```

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Mandatory: Adjust positional encoding

Optional: Vectorized unpacking for sequence based evaluation like NSP



ADJUSTING HYPERPARAMETERS

LAMB

- Standard optimizer used in BERT
- Calculates weighted moments m and v

$$\text{Compute } g_t = \frac{1}{|\mathcal{S}_t|} \sum_{s_t \in \mathcal{S}_t} \nabla \ell(x_t, s_t).$$
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

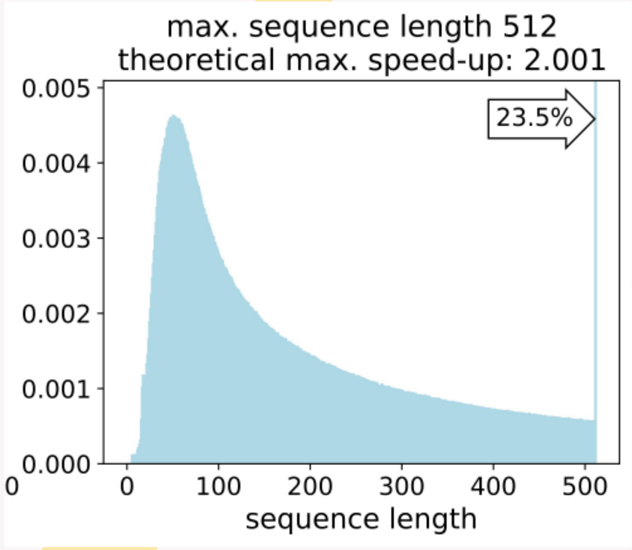
- weights for moments
  power-dependent on packing factor p

$$\beta_1 := \beta_1^p,\ \beta_2 := \beta_2^p$$

Image: https://krishansubudhi.github.io/deeplearning/2019/09/21/LambPaperDisected.html

PACKED BERT EXPERIMENTS

EXPERIMENTS: PACKING WIKIPEDIA

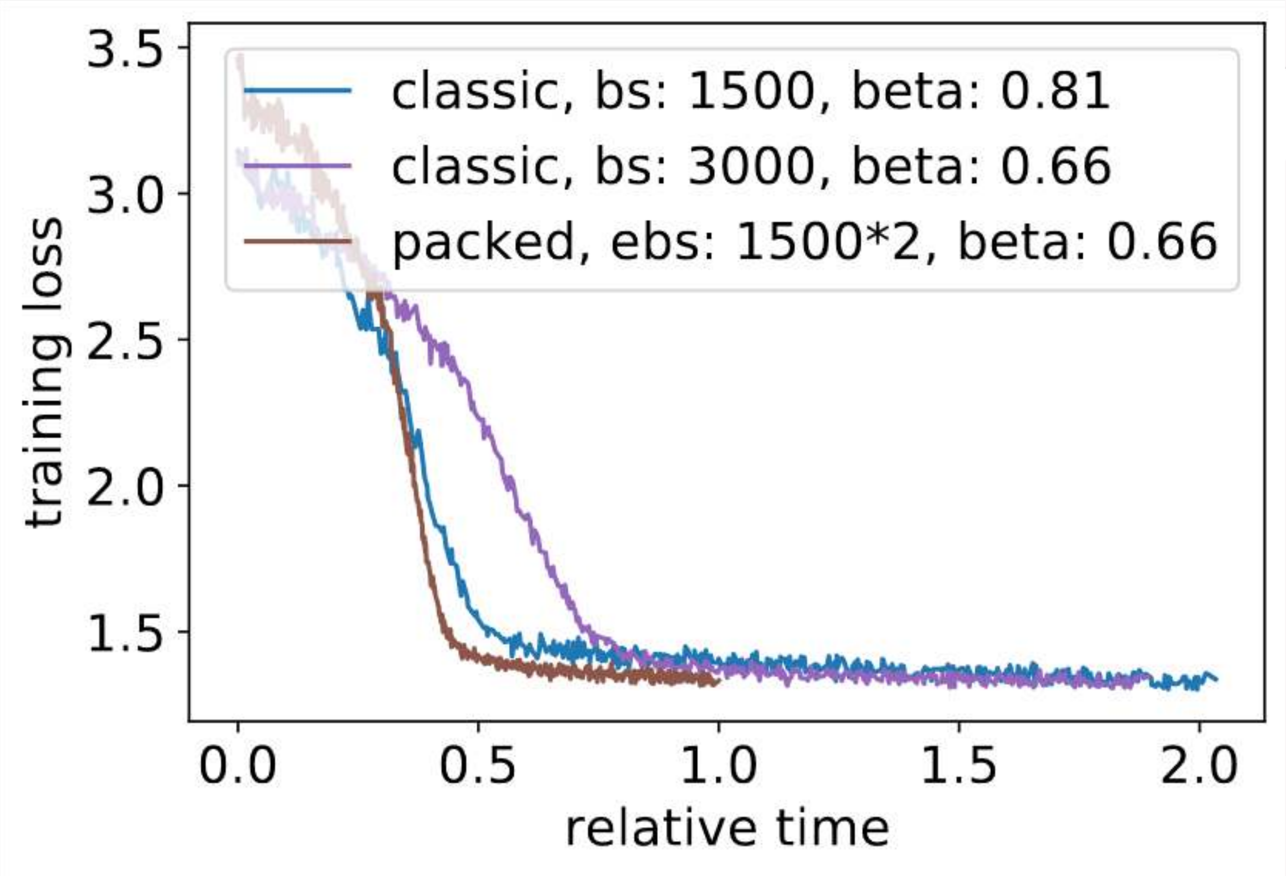


pack. depth	pack. algo.	# packs [M]	packing efficiency (%)	pack. factor	overhead (%)	realized speed-up
1	none	16.280	49.97	1.000	0.000	1.000
2	SPFHP	10.102	80.52	1.612	4.283	1.544
3	SPFHP	9.095	89.44	1.790	4.287	1.716
3	NNLSHP	8.155	99.75	1.996	4.287	**1.913**
4	SPFHP	8.659	93.94	1.880	4.294	1.803
8	SPFHP	8.225	98.90	1.979	4.481	1.895
16/max	SPFHP	8.168	99.60	1.993	4.477	1.905

SO, DO WE GET THE DESIRED 2X SPEED-UP?

Yes!

- FLOPS only reduced by a factor of 1.913,
- Less overhead of IO
- This leads to an additional speed-up which pushes us >2x

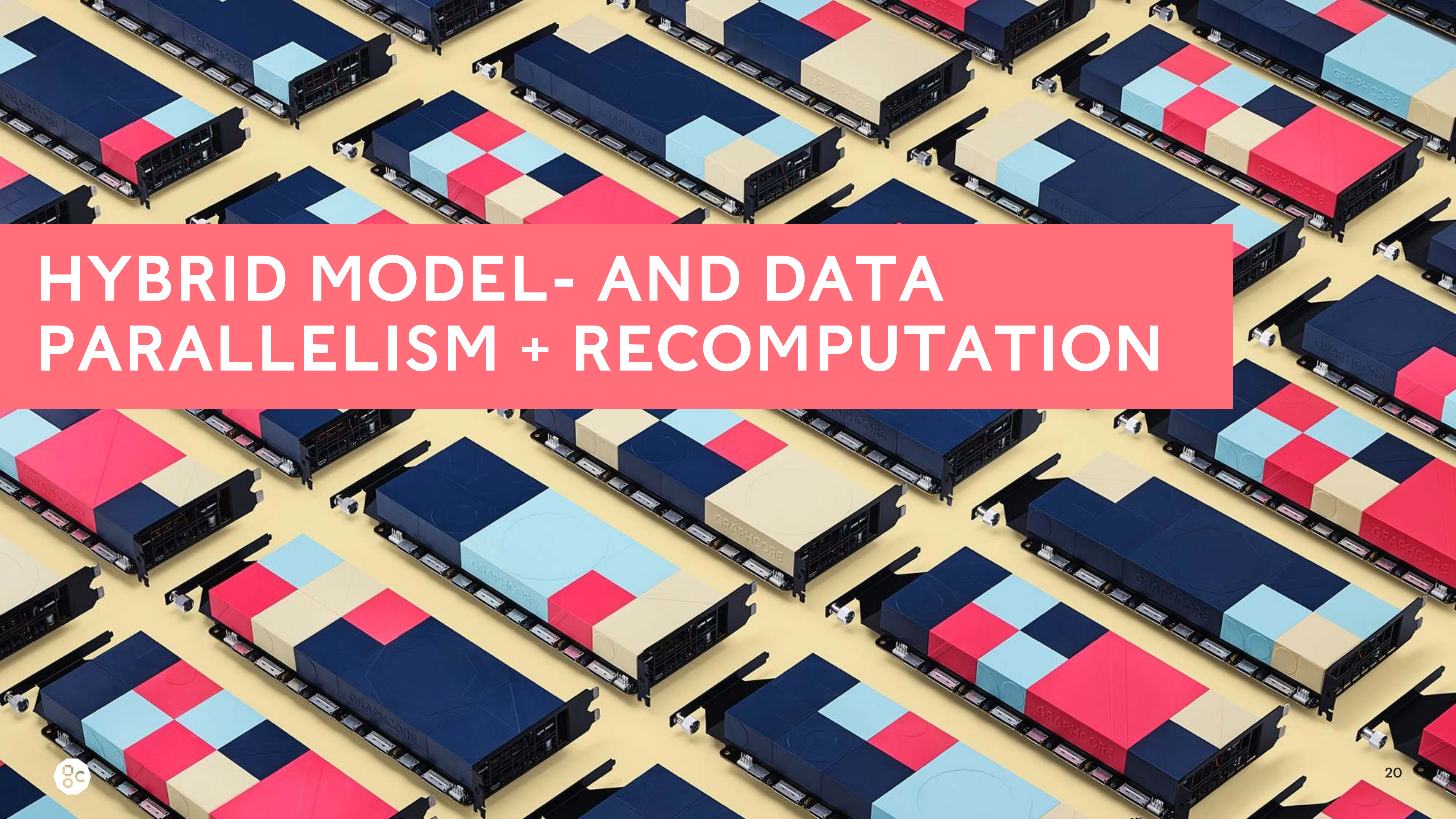


Plot shows runtime compared to packedBERT
Original is around 2x slower

ANY QUESTIONS ON PACKING?

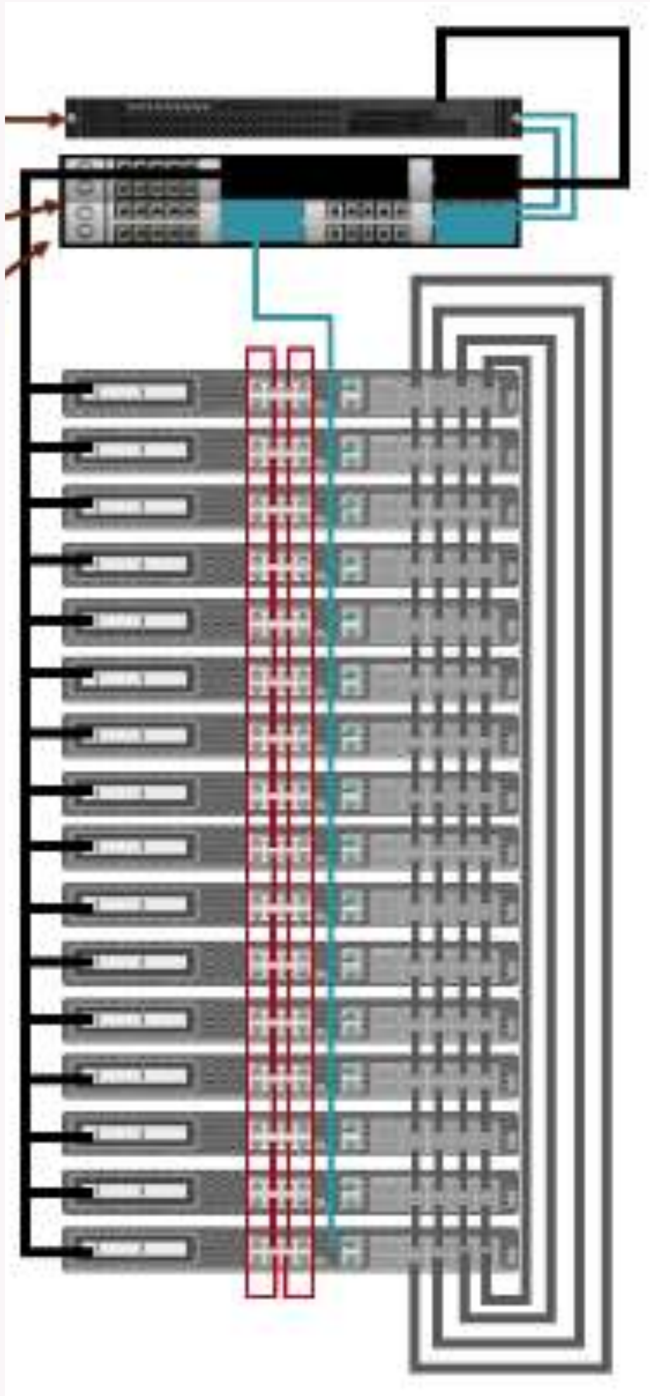
- Read the blog and the paper
  - https://towardsdatascience.com/introducing-packed-bert-for-2x-faster-training-in-natural-language-processing-eadb749962b1
  - https://arxiv.org/pdf/2107.02027.pdf

HYBRID MODEL- AND DATA PARALLELISM + RECOMPUTATION

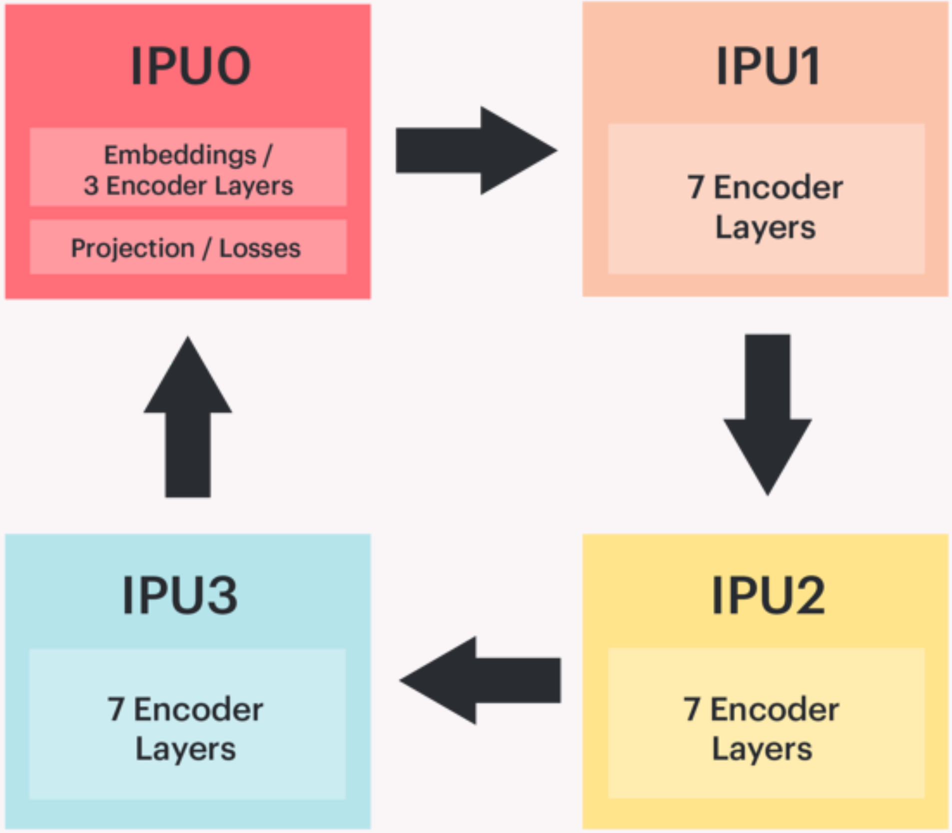


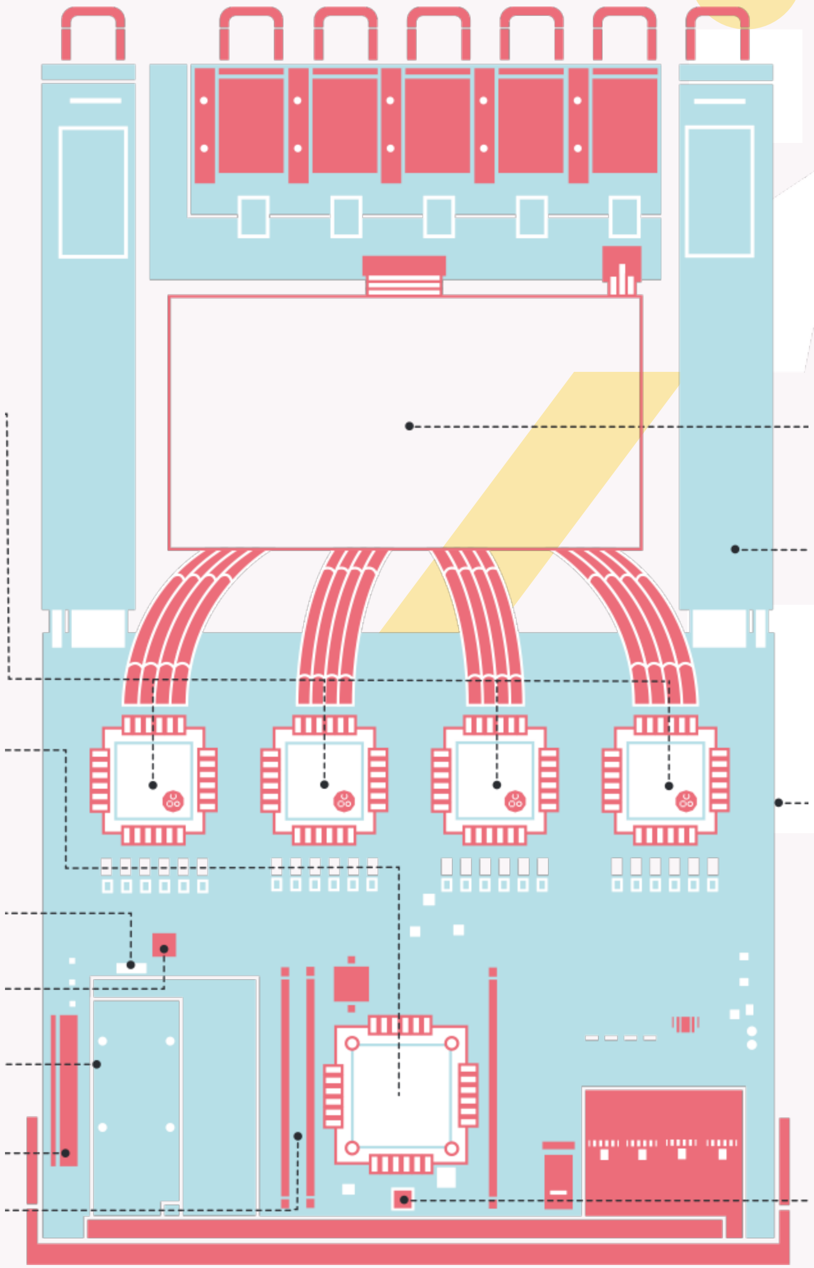
HOST DISAGGREGATION

- BERT is not host bound
  - 1 host even for 64 IPUs
  - Power and cost savings
  - Combines model and data parallelism for job distribution
  - 1 M2000 for POD 16 setup



PIPELINE MODEL PARALLELISM FOR BERT (FORWARD PASS)





Left: https://www.graphcore.ai/posts/bert-large-training-on-the-ipu-explained

PIPELINING

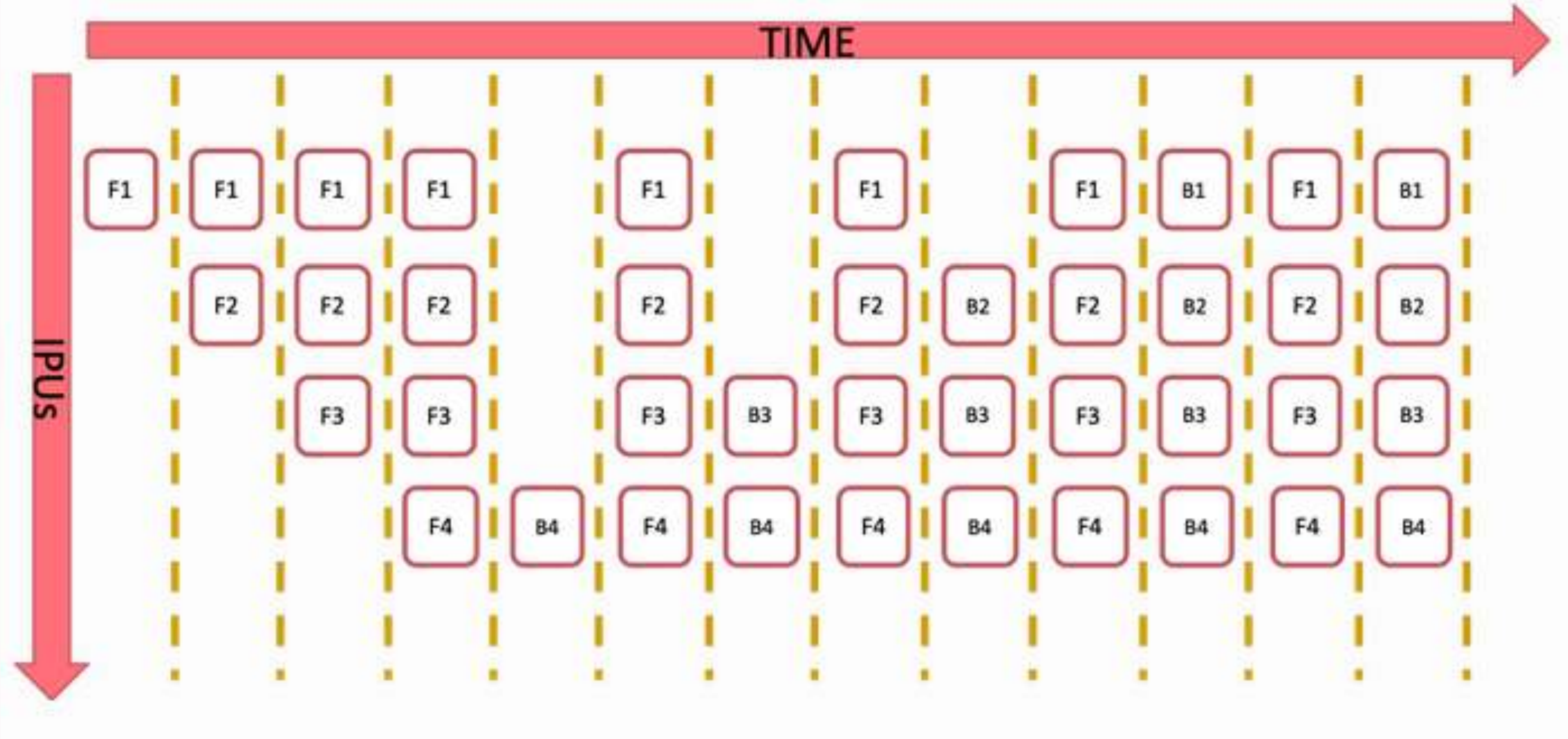
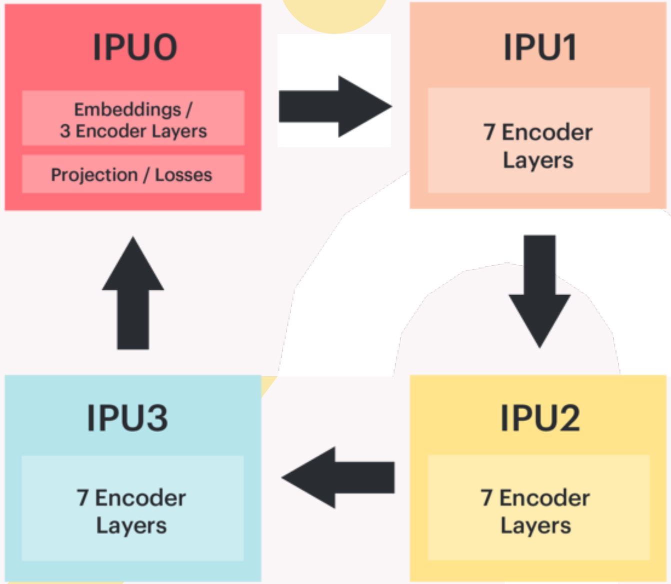


Fig. 3.3 Grouped schedule

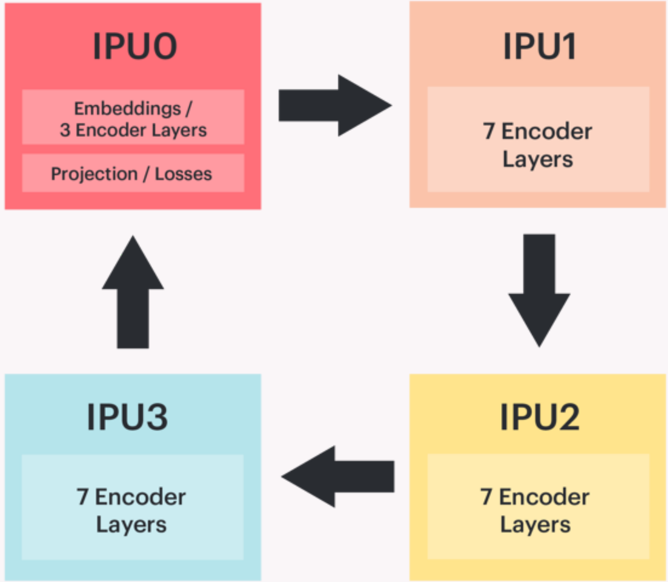


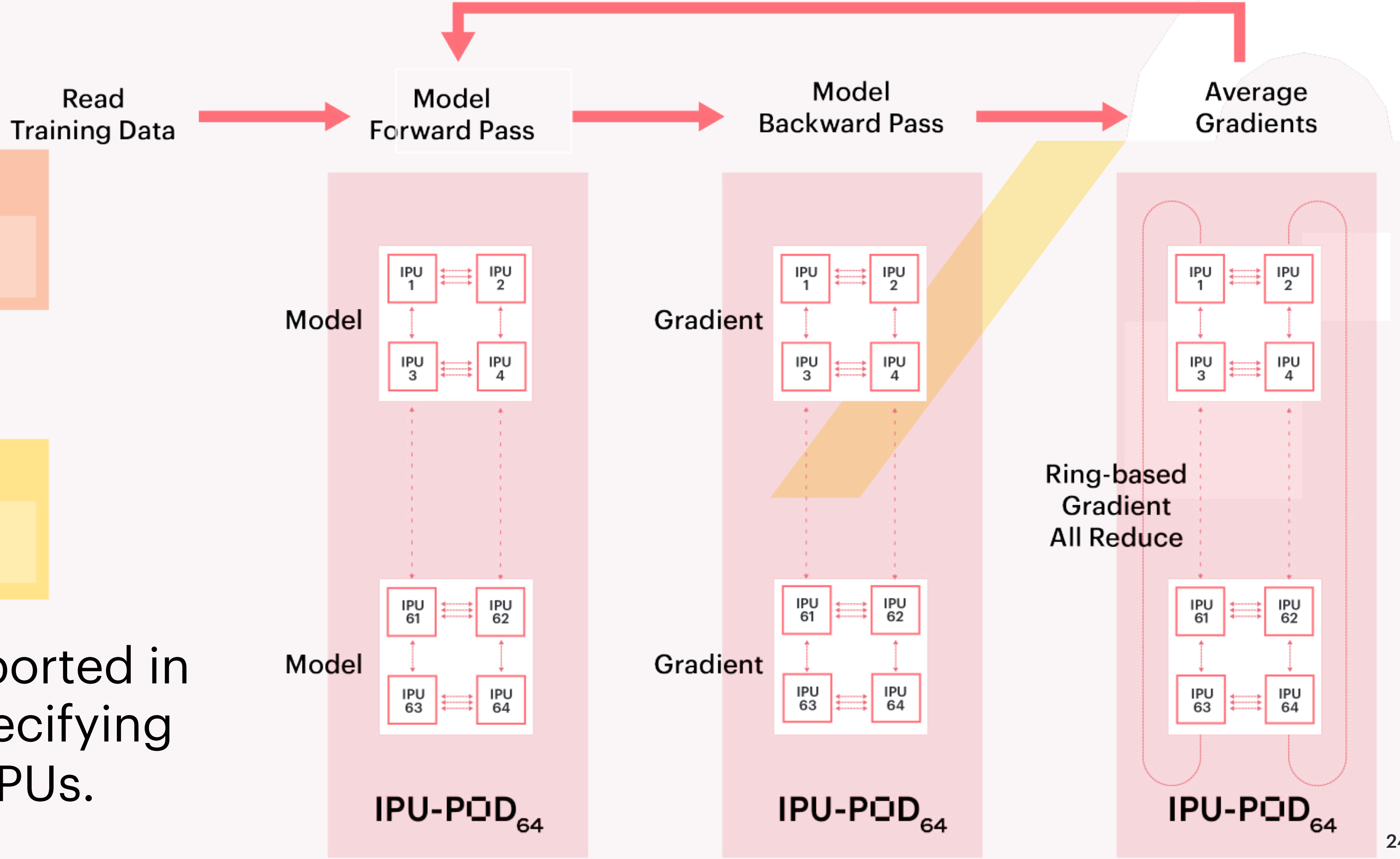
Model parallelism supported over pipeline API for TensorFlow, PyTorch, and Poplar Advanced Runtime (PopART)

Left: https://docs.graphcore.ai/projects/tf-model-parallelism/en/latest/pipelining.html#id3

Note: To save memory, we recompute the local forward passes F1, F2, F3 as part of the backward passes B1, B2, B3.

HYBRID DATA-MODEL PARALLELISM WITH BERT



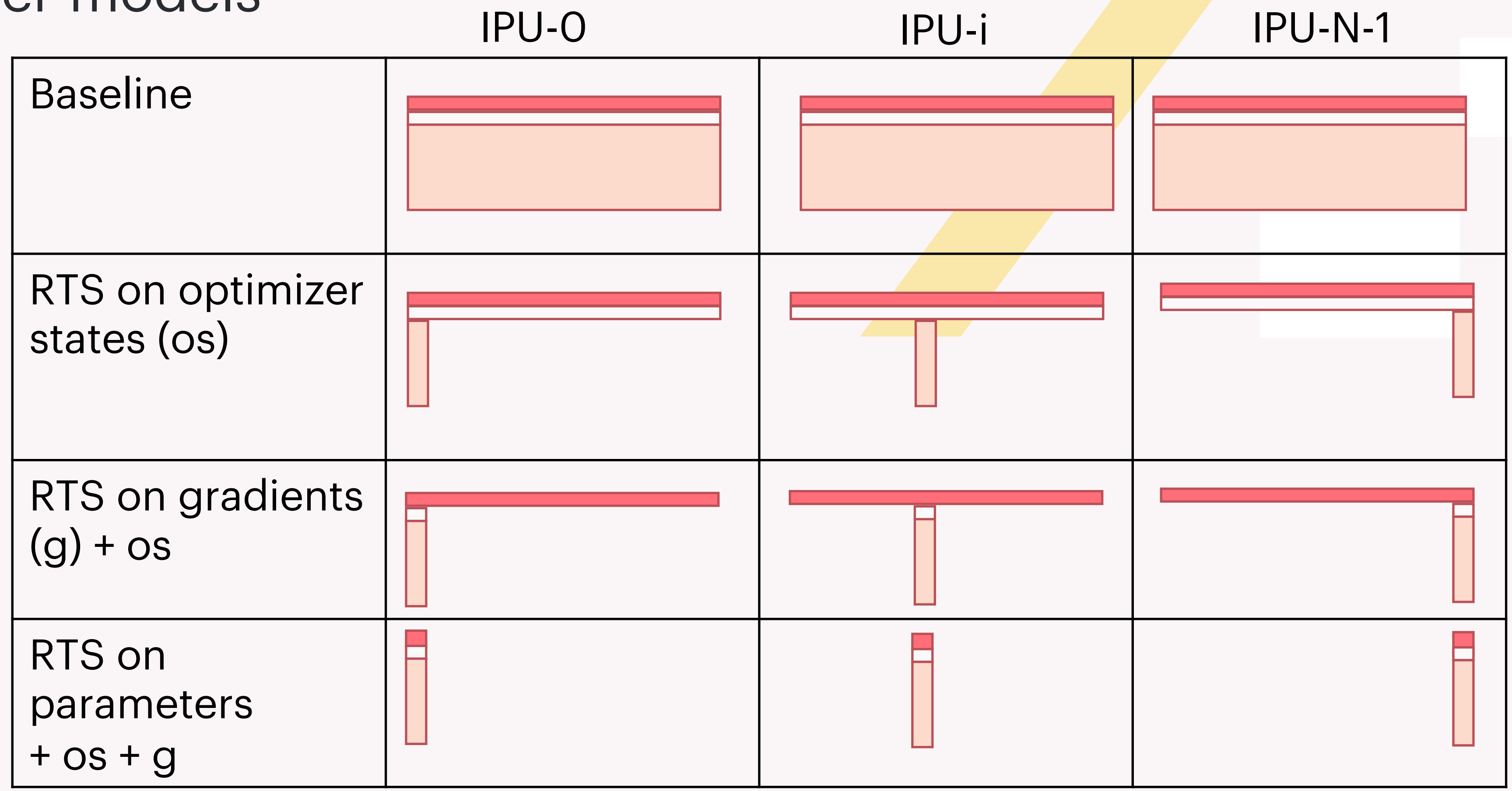


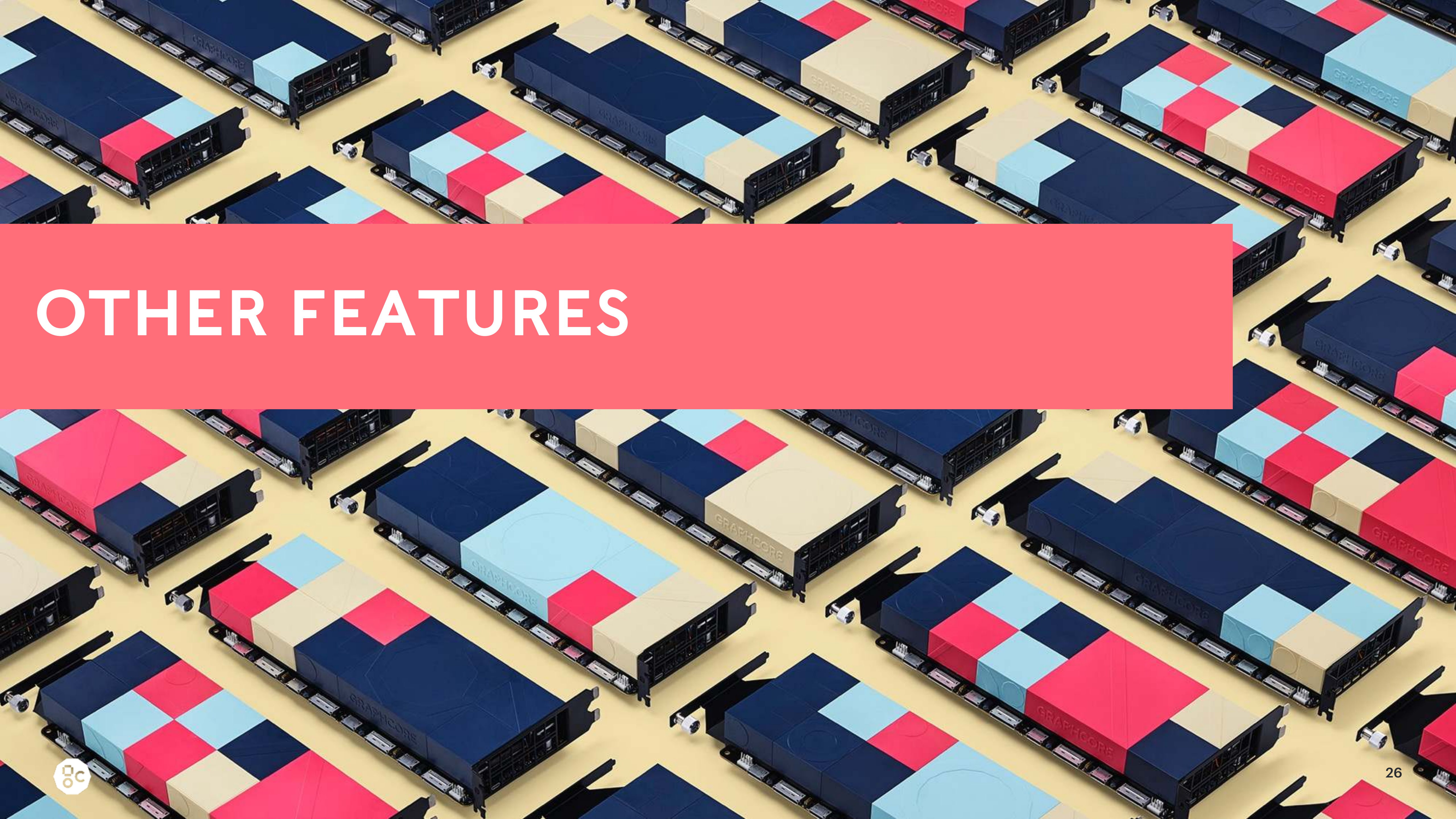
Data parallelism supported in all frameworks by specifying the number of used IPUs.

Images: https://www.graphcore.ai/posts/bert-large-training-on-the-ipu-explained

REPLICATED TENSOR SHARDING (RTS)

- Reduce overall memory footprint
- Can be combined with variable offloading to streaming memory
- Path for even larger models





OTHER FEATURES

LOW BATCH SIZE TRAINING

- Large local batch size requires storing plenty of unused activations
- LBS 2 for BERT
- Gradient accumulation for larger global batch size to save update costs

LOW PRECISION TRAINING

- FP16
- Stochastic rounding
- Weight aggregation in FP32
- Master weights in FP16
- LAMB moments in FP16 (streamed from DRAM on demand)

RESNET 50 SETUP

BATCH SIZE & DISTRIBUTION

- Single IPU data parallel setup with 1-4 hosts
- Introduced convenient recomputation API in our TensorFlow 1.15 framework to boost BS to 20

DISTRIBUTED BATCH NORM

- Batch norm requires statistics of at least 32 samples
- We combine data from 2 IPUs for a sufficient batch of 40 samples
- Simple configuration variable assignment

CONCLUSION

- Several SW strategies applied for amazing performance
- Strategies not benchmark specific: Generalize to variety of models/datasets
- Intuitive support of hybrid data- and model-parallel training including recomputation in TensorFlow, PyTorch, and our own Poplar Advanced Run Time (PopART)
- Try PackedBERT!

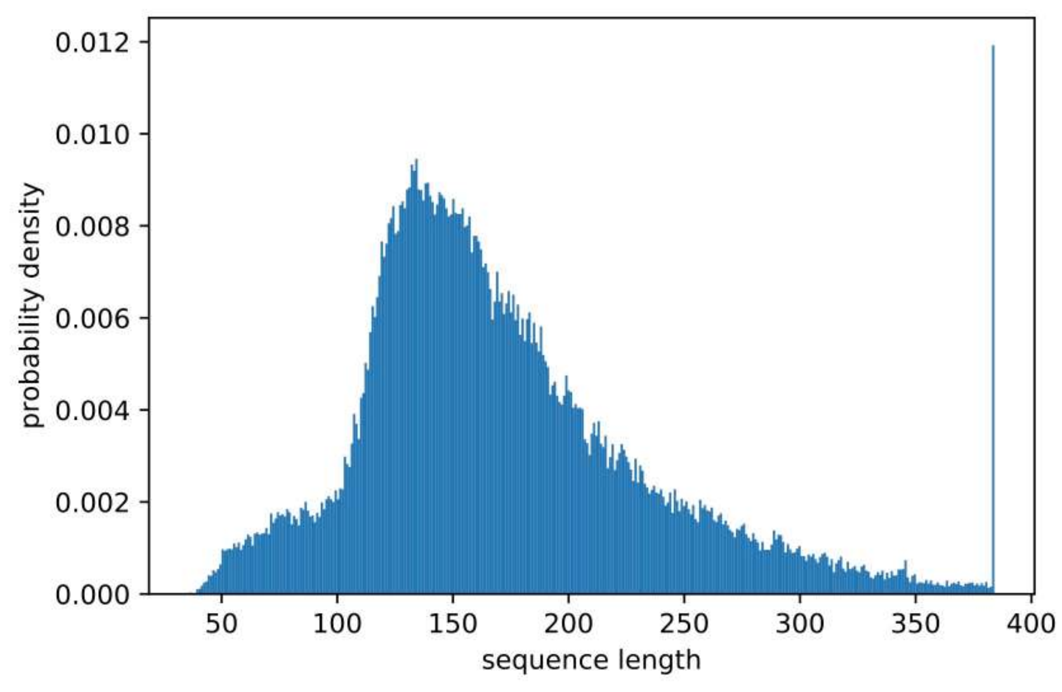
MLPerf v1.0 Training Results | MLPerf ID: 1.0-1025, 1.0-1026, 1.0-1027, 1.0-1028, 1.0-1098, 1.0-1099
The MLPerf name and logo are trademarks. See www.mlperf.org for more information

THANK YOU

Looking for new applications!
We are hiring!

Dr. Mario Michael Krell

EXPERIMENTS: PACKING SQUAD 1.1



pack. depth	pack. algo.	# strat. used	# packs	# tokens	# padding tokens	efficiency (%)	pack. factor
1	none	348	88641	34038144	18788665	44.801	1.000
2	SPFHP	348	45335	17408640	2159161	87.597	1.955
3	NNLSHP	398	40808	15670272	420793	97.310	2.172
3/max	SPFHP	344	40711	15633024	383545	97.547	2.177