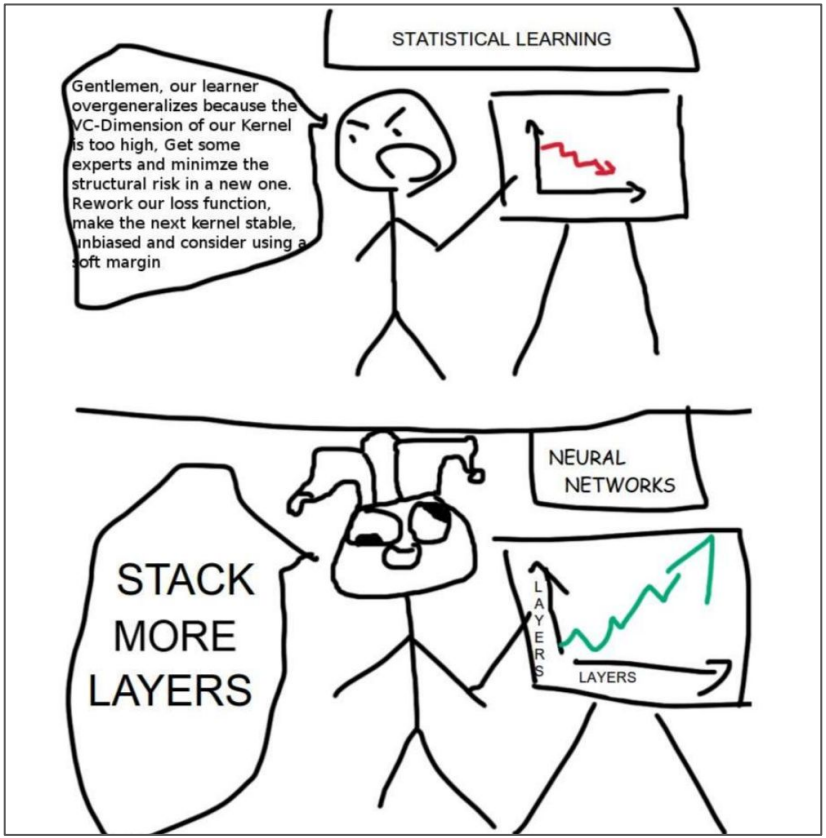
GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding

Aditya Siddhant, Ali Dabirmoghaddam, Alison Lui, Ankur Bapna, Ankur Parikh, Biao Zhang, Colin Cherry, Dmitry Lepikhin, George Foster, Isaac Caswell, James Kuczmarski, Kun Zhang, Macduff Hughes, Mahdis Mahdieh, Markus Freitag, Maxim Krikun, Melvin Johnson, Mia Chen, Naveen Arivazhagan, Orhan Firat, Roee Aharoni, Sébastien Jean, Sneha Kudugunta, Tao (Alex) Yu, Thang Luong, Wei Wang, Wolfgang Macherey, Yanping Huang, Xavier Garcia, Yonghui Wu, Yuan Cao, Zhifeng Chen, Zirui Wang

ICLR 2021

Google AI

Recent trend in Machine Learning

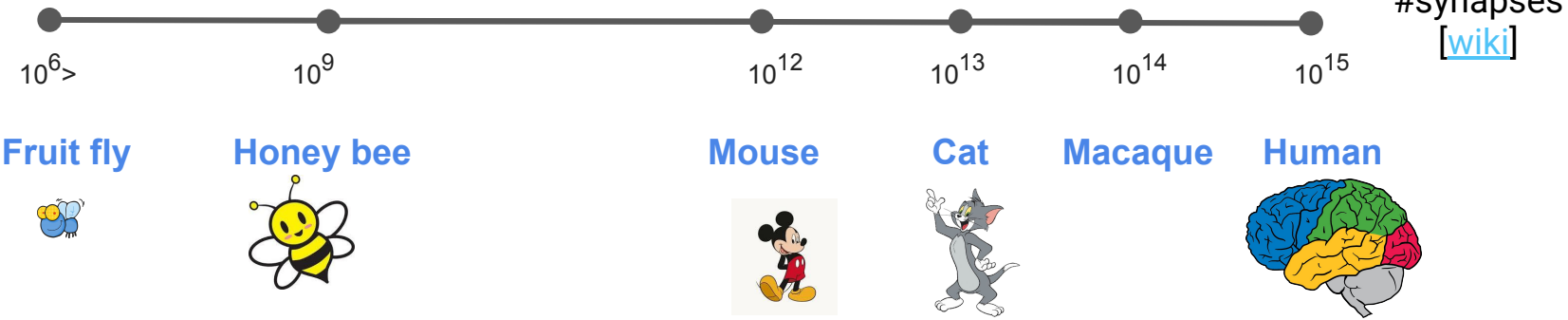


Machine Learning Memes for Convolutional Teens

Recent trend in Machine Learning

Number of Synapses in Biological & Artificial Systems

Google AI



Number of Synapses

NMT with Attention
Resnet50
[25-50M]

Fruit fly

Honey bee

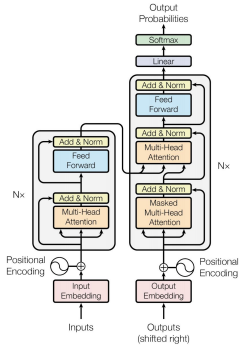
Mouse

Cat

Macaque

Human

Number of Synapses



Transformer [400M]

10^{6}> 10^{9} 10^{12} 10^{13} 10^{14} 10^{15}

#synapses [wiki]

Fruit fly

Honey bee

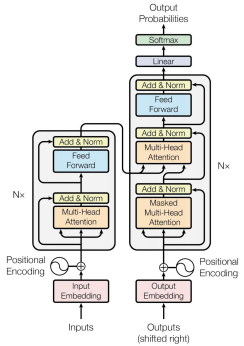
Mouse

Cat

Macaque

Human

Number of Synapses



Google GPipe
Facebook ResNeXt101
Open AI GPT2
MSR ZeRO
NVidia Megatron-LM
Google-T5
[1-10B]

Transformer
[400M]

#synapses
[wiki]

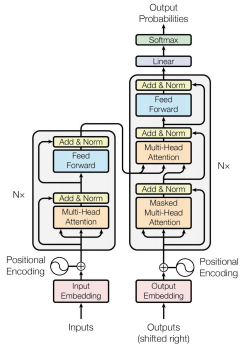
10^{6}> 10^{9} 10^{12} 10^{13} 10^{14} 10^{15}

Fruit fly | Honey bee | Mouse | Cat | Macaque | Human





Number of Synapses



Google GPipe
Facebook ResNeXt101
Open AI GPT2
MSR ZeRO
NVidia Megatron-LM
Google-T5
[1-10B]

GPT3
M4:
[80-600B]

Transformer
[400M]

10^{6}> 10^{9} 10^{12} 10^{13} 10^{14} 10^{15}

#synapses
[wiki]

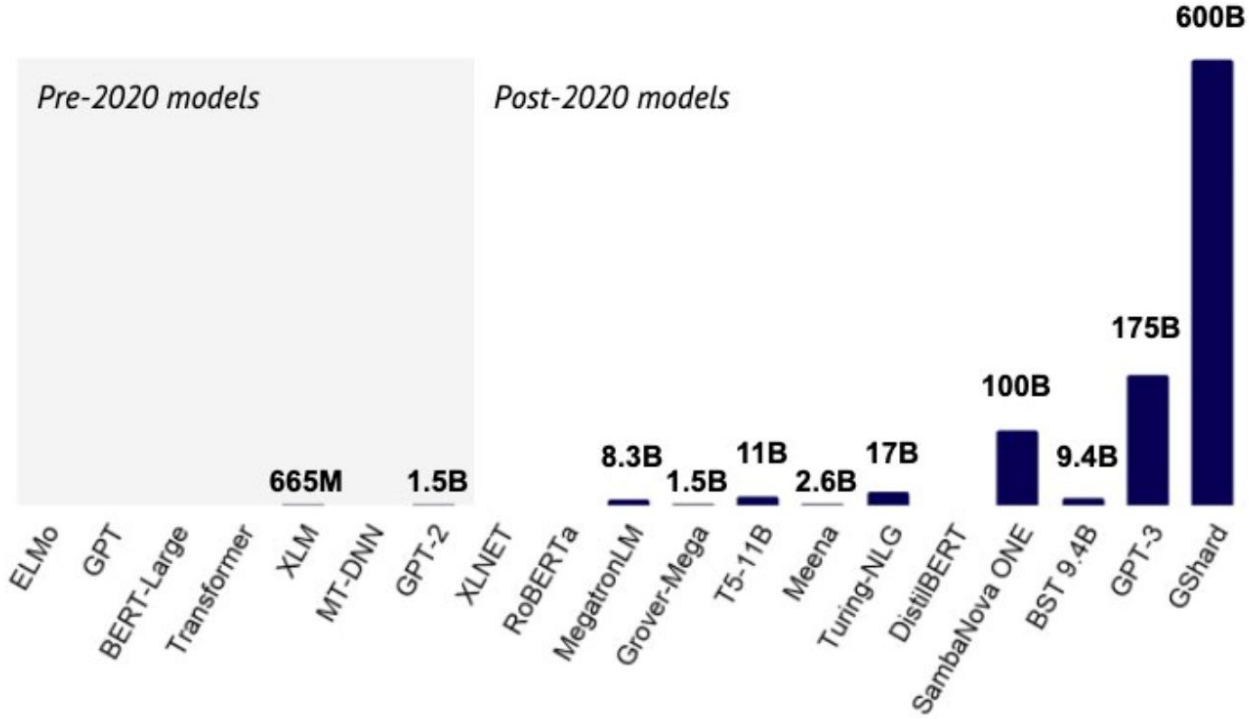
Fruit fly | Honey bee | Mouse | Cat | Macaque | Human





Language models: Welcome to the Billion Parameter club

Charting major NLP model size by publication date, February 2018 (left) to June 2020 (right)



Our motivation

Develop a universal machine translation model (i.e. one model for all the people in the world)

*"Perhaps the way [of translation] is to descend, from each language, down to the common base of human communication -- the real but as yet **undiscovered universal language** -- and then re-emerge by whatever particular route is convenient."*

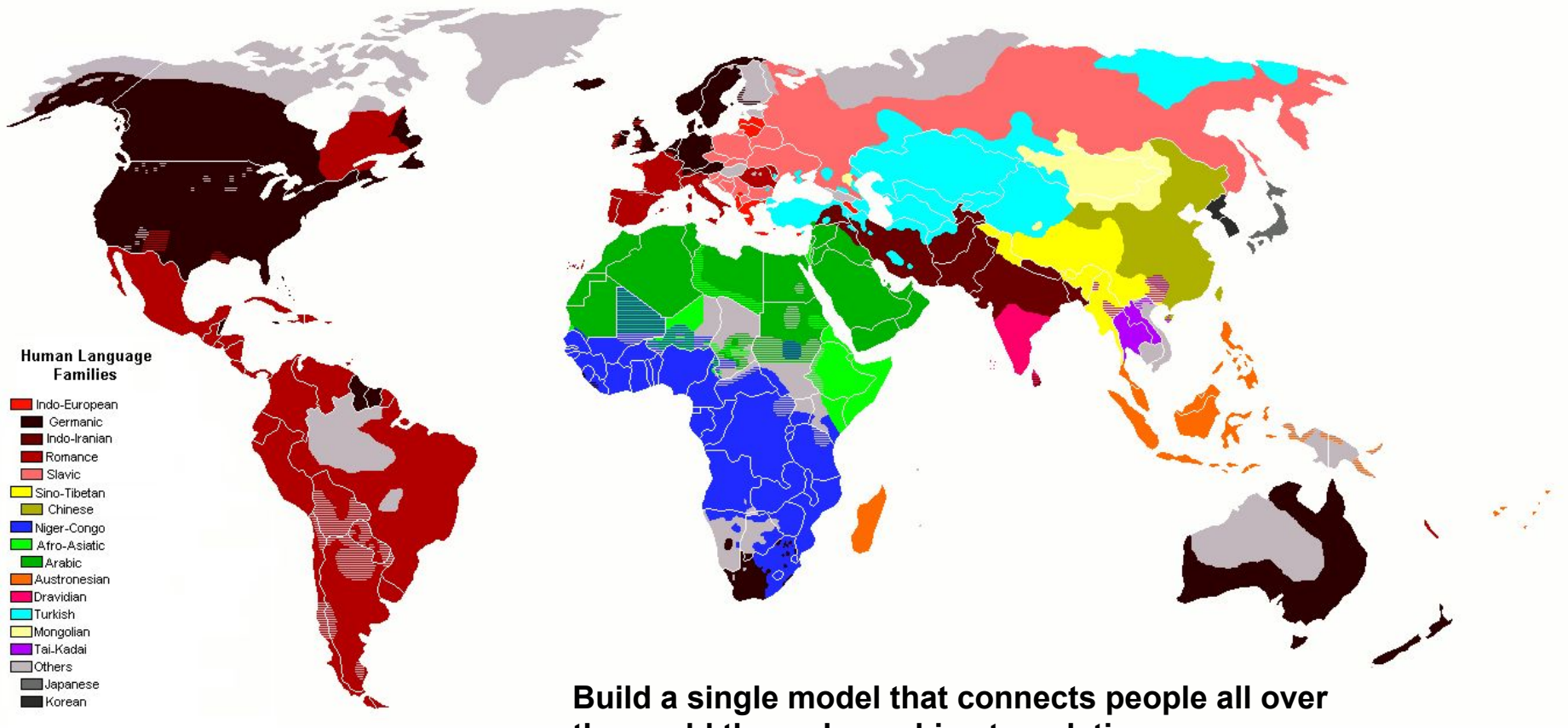
Warren Weaver (1949)

Google AI Blog

The latest news from Google AI

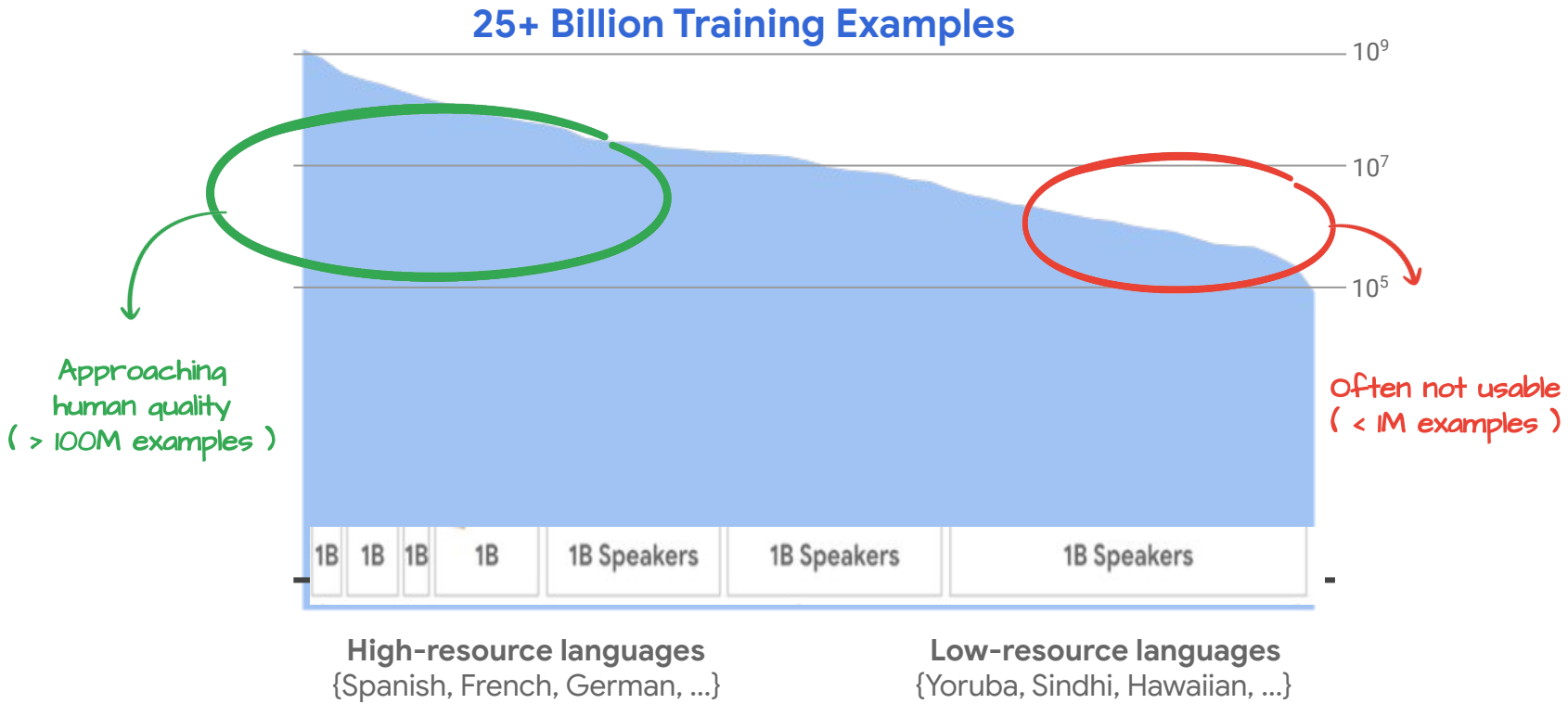
Exploring Massively Multilingual, Massive Neural Machine Translation

List of languages by number of speakers

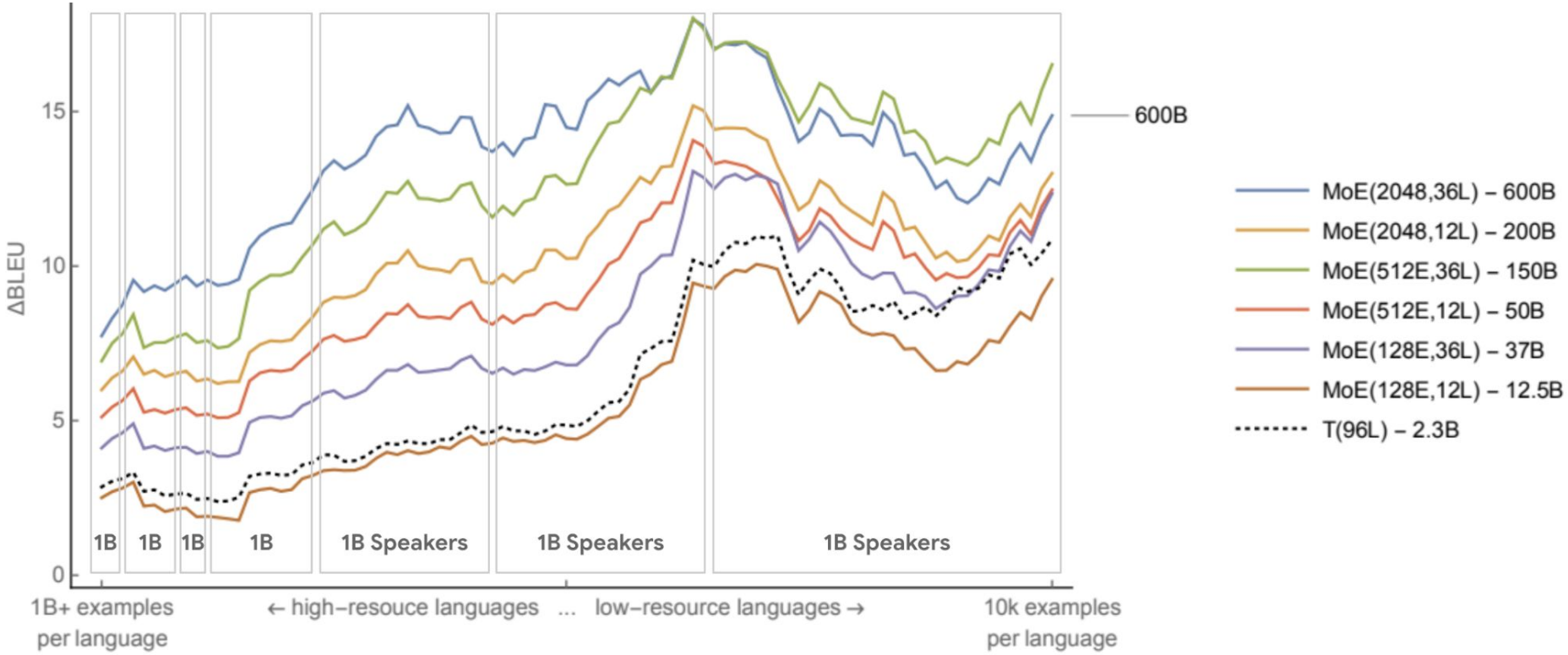


Build a single model that connects people all over the world through machine translation.

Motivation:
Improve translation quality for all language pairs



GShard: Building a single model that improves translation quality across 100+ languages.

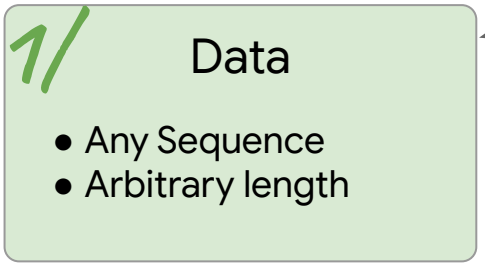


Enumerate all relevant (research) challenges

$$\text{quality} = f(\text{X}, \theta, \mu)$$

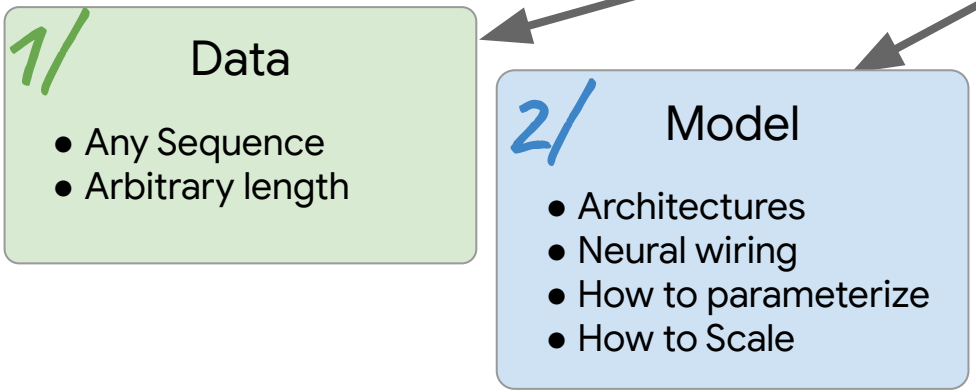
Enumerate all relevant (research) challenges

$$\text{quality} = f(\mathrm{X}, \theta, \mu)$$



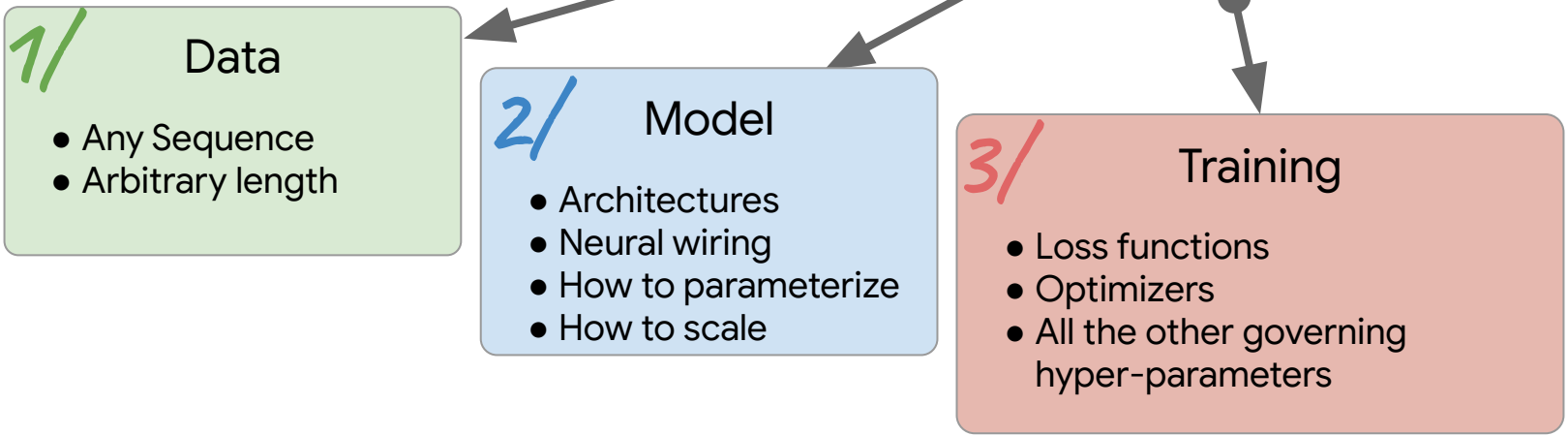
Enumerate all relevant (research) challenges

$$\text{quality} = f(X, \theta, \mu)$$



Enumerate all relevant (research) challenges

$$\text{quality} = f(X, \theta, \mu)$$



Enumerate all relevant (research) challenges

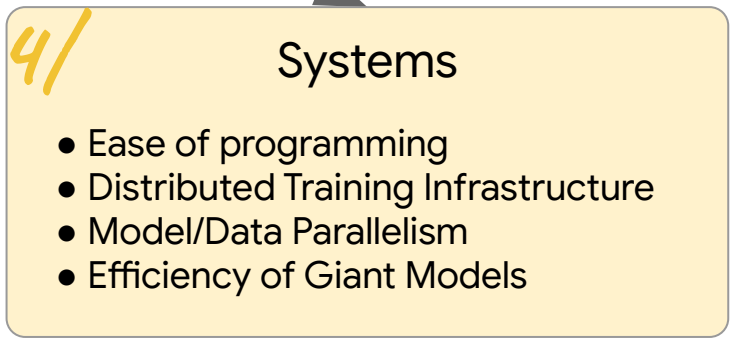
$$\text{quality} = f(\mathrm{X}, \theta, \mu)$$

One more crucial component!

Often overlooked :-(

Enumerate all relevant (research) challenges

$$\text{quality} = f(\text{X}, \theta, \mu)$$



Overview

1/ The Shades of Training Data

2/ Modeling of the Massively Multi-Task Neural Networks

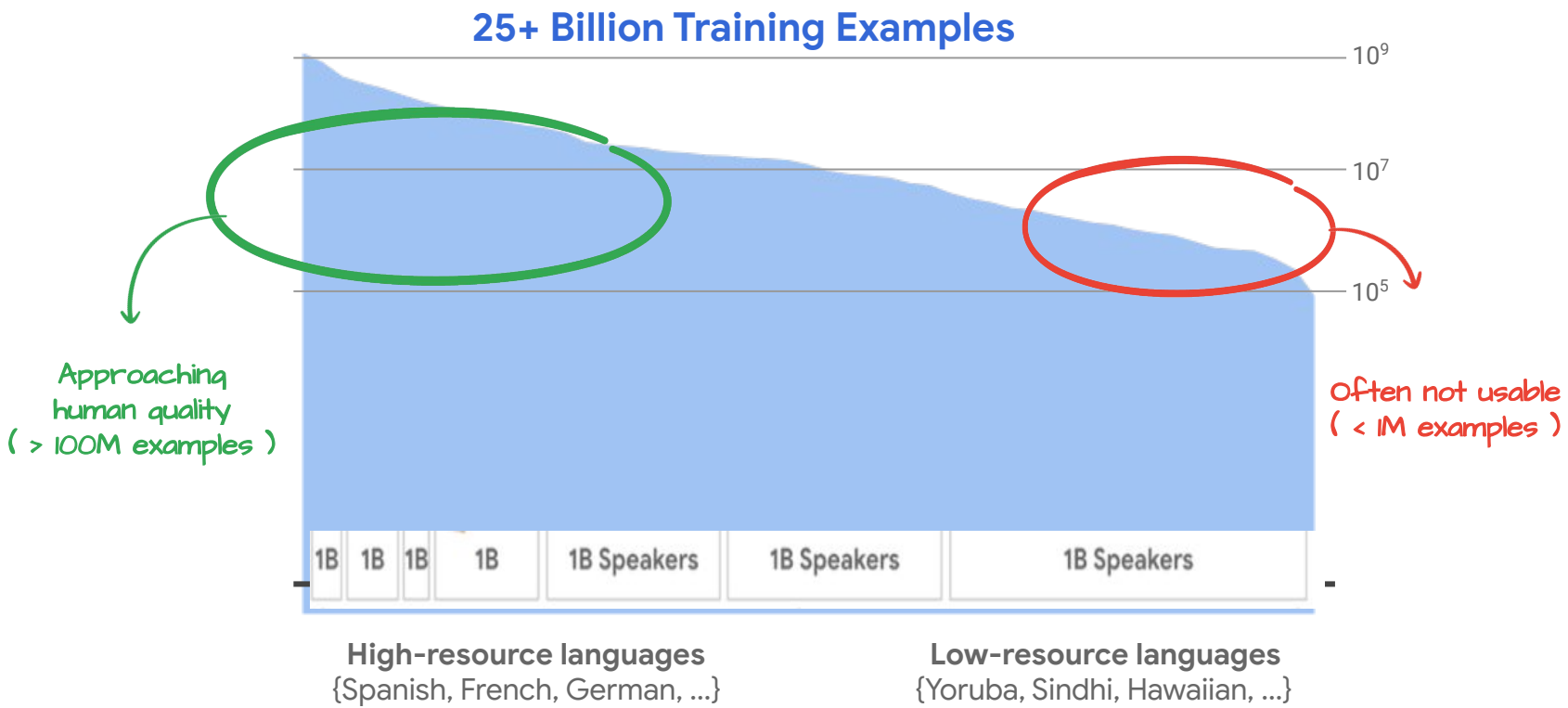
3/ Training Regimes and Optimization at Scale

4/ Systems at the Age of Giant Models

1/ The Shades of Training Data

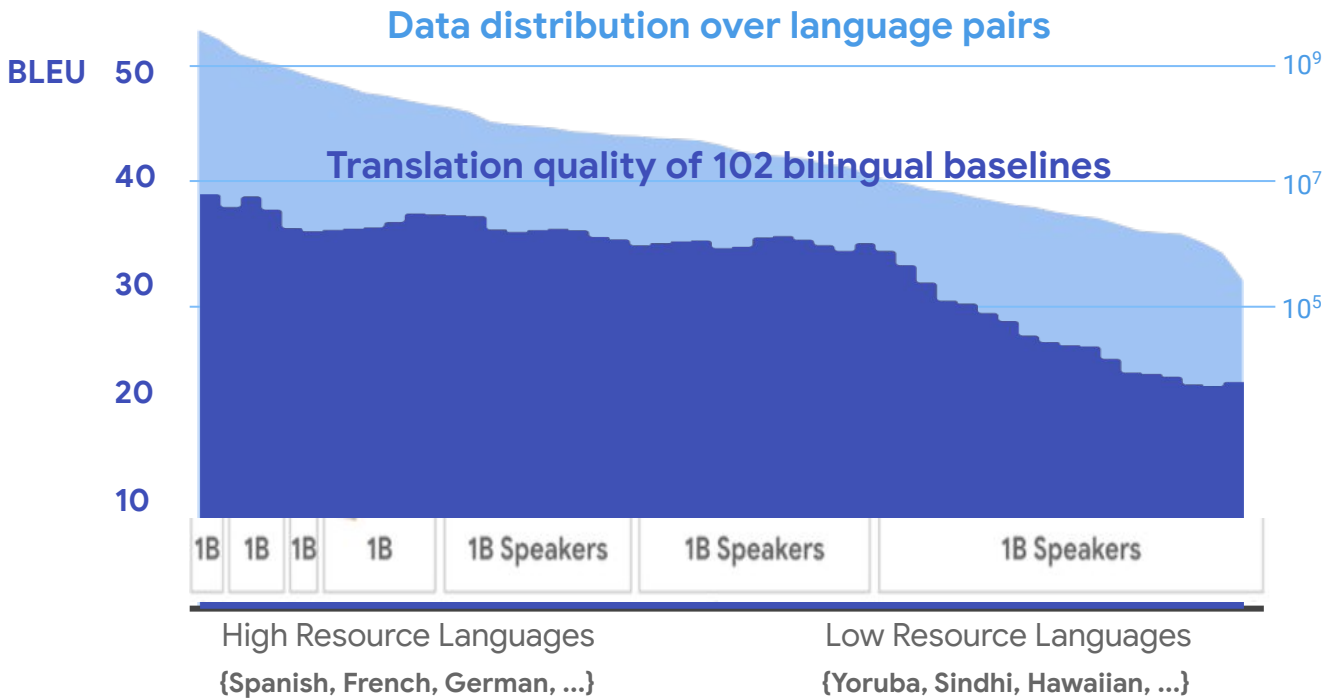
We trained and evaluated new bilingual models as controls

Data distribution over language pairs



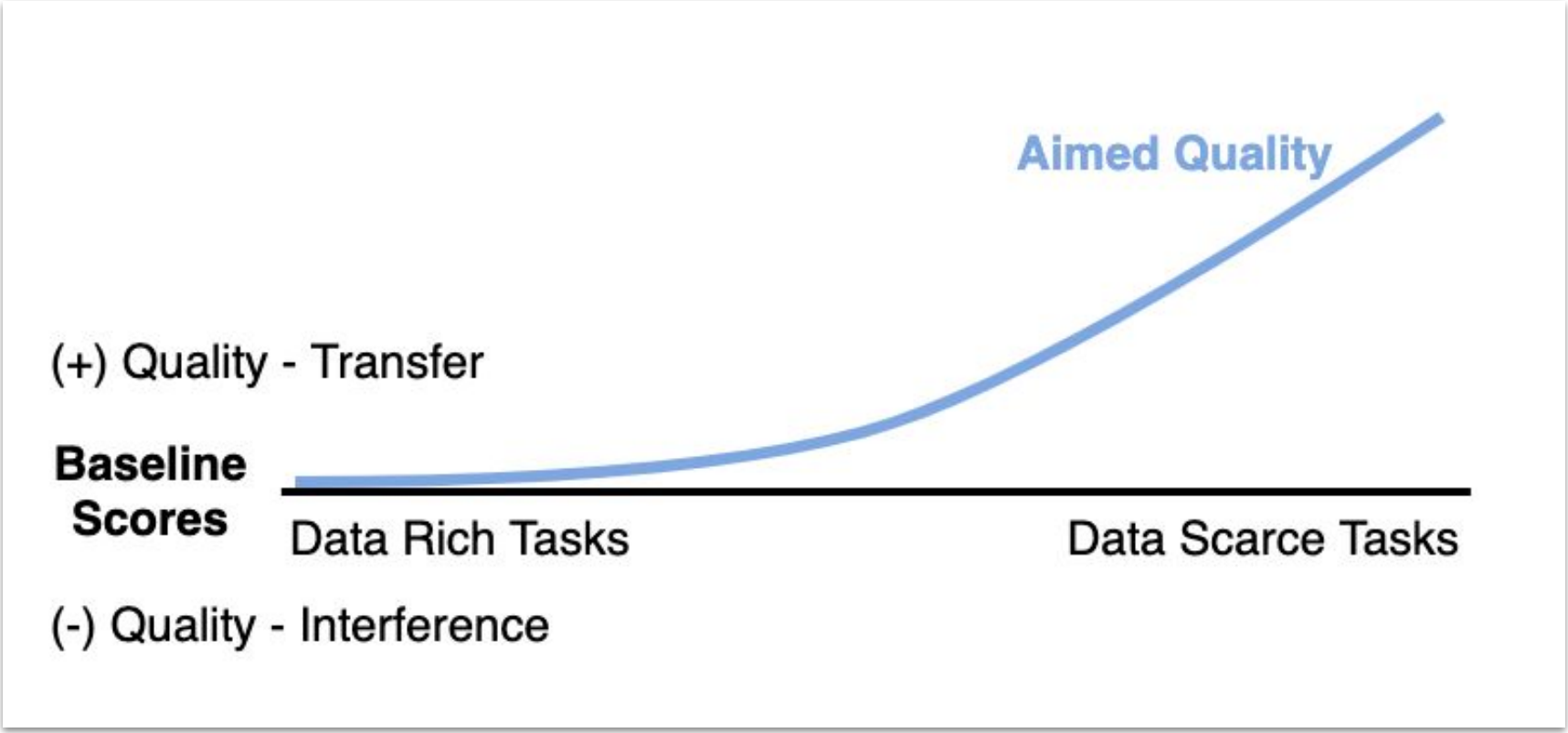
1/ The Shades of Training Data

We trained and evaluated new bilingual models as controls



1/ The Shades of Training Data

Ideal Results that we Aim

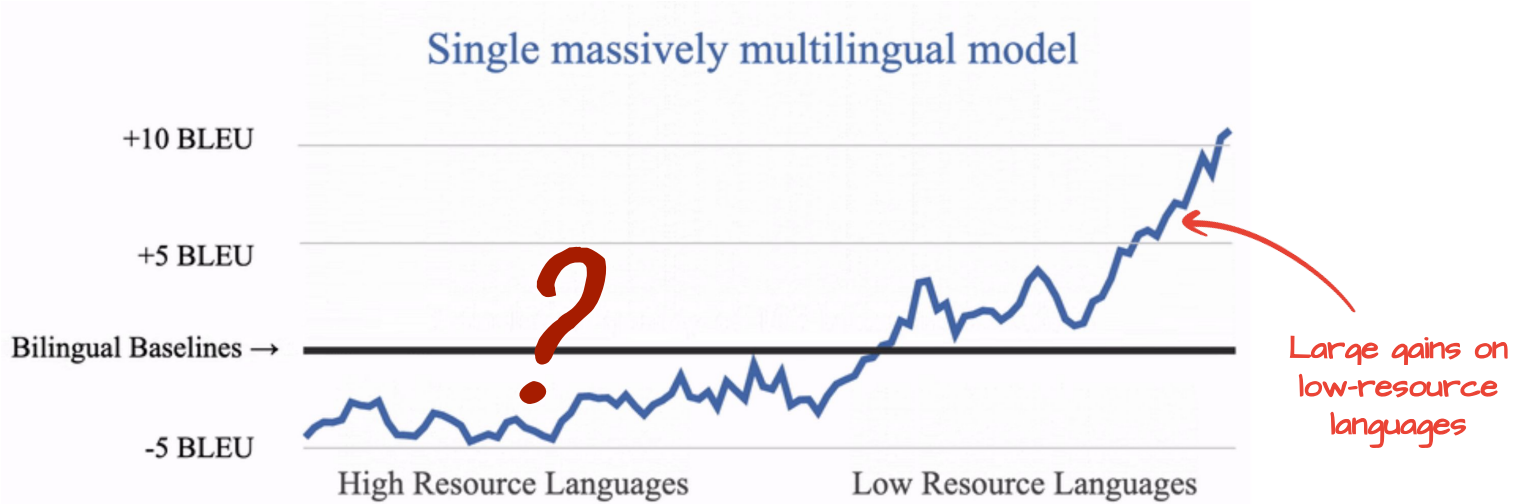


Results we got instead

Recipe:

⟶ Train Transformer-Big sizes (400M) massively multilingual model (M3)

⟶ Use monolithic (share everything) models



Overview

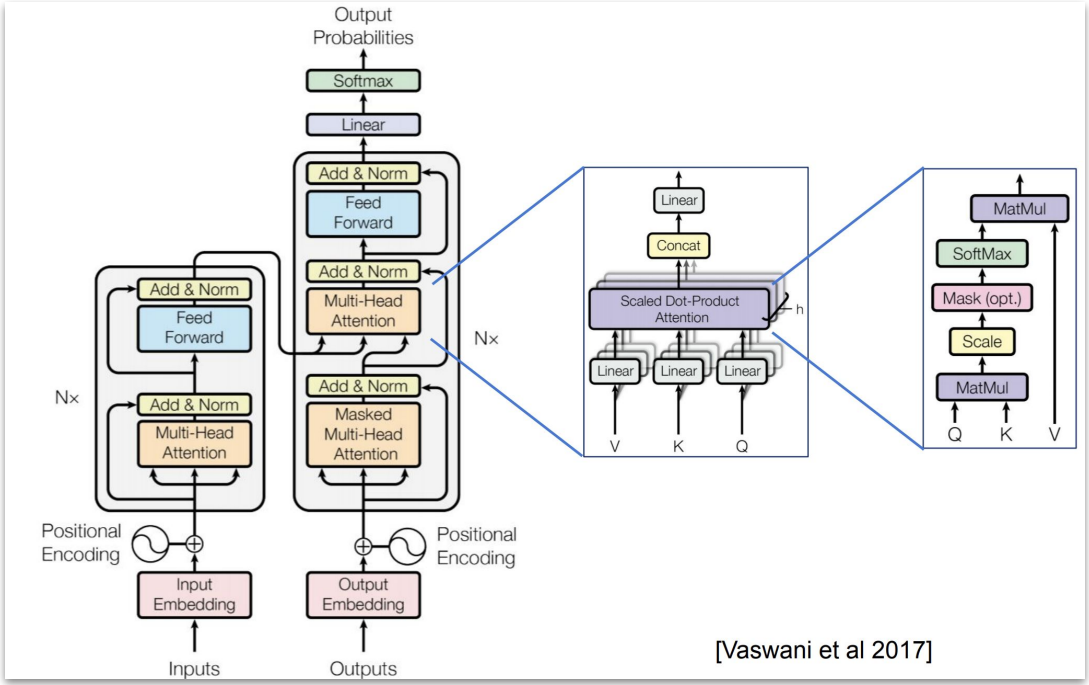
1/ The Shades of Training Data

2/ **Modeling of the Massively Multi-Task Neural Networks**

3/ Training Regimes and Optimization at Scale

4/ Systems at the Age of Giant Models

Models
Network: Transformer as explained in [1][2] and [3]



[Vaswani et al 2017]

Key Model Parameters:

- Depth: the number of layers
- Width: The embedding dimension.

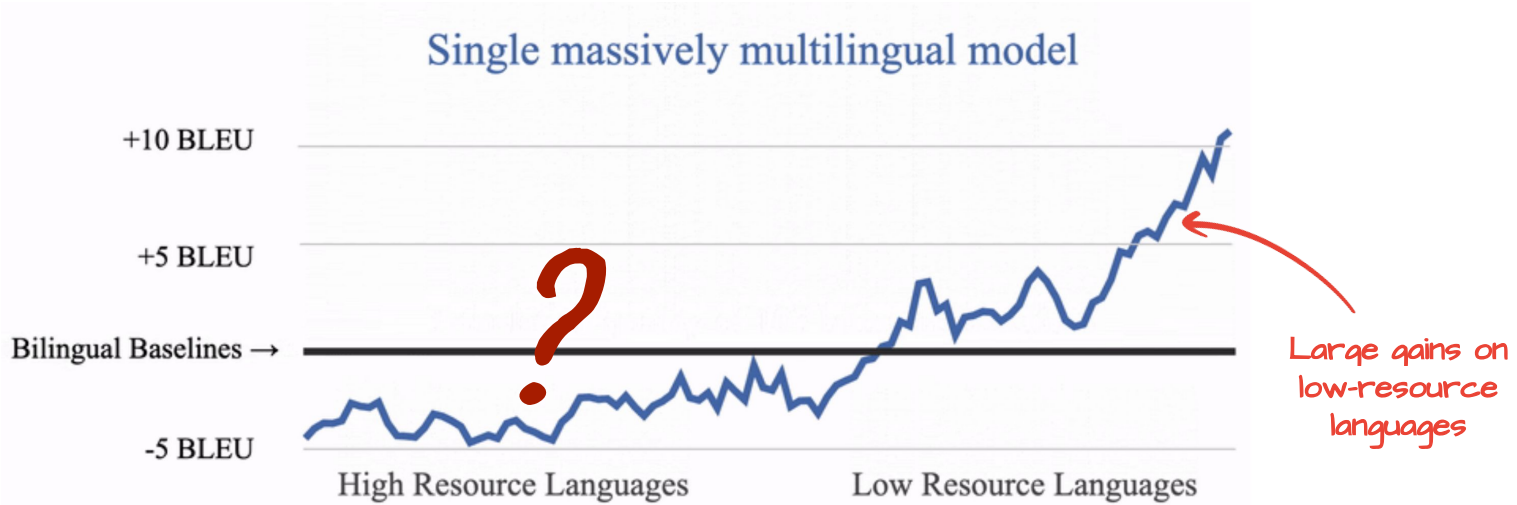
[1] Vaswani et al. 2017, “Attention is All You Need”
[2] Chen et al. 2018, “The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation”
[3] Bapna et al. 2018, “Training Deeper Neural Machine Translation Models with Transparent Attention”

2/ Modeling of the Massively Multi-Task Neural Networks

Recipe:

⟶ Train Transformer-Big sizes (400M) massively multilingual model (M3)

⟶ Use monolithic (share everything) models



2/ Modeling of the Massively Multi-Task Neural Networks

To reduce quality losses, we needed greater model capacity

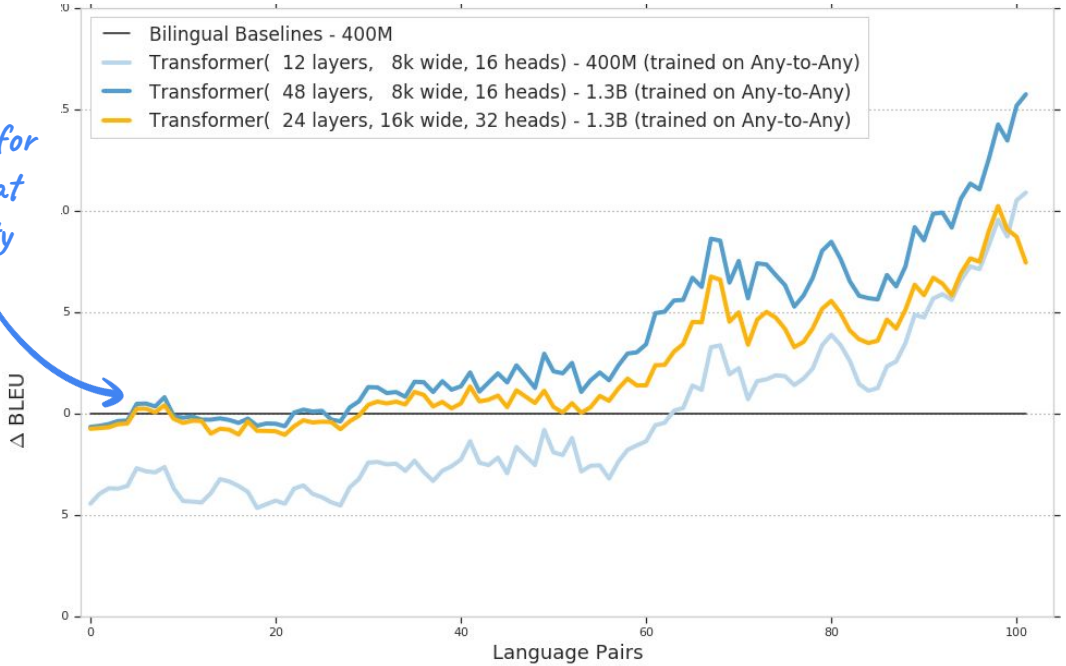
Relevant research questions:

- Do deep or wide networks drive greater quality gains?
- How deep or wide should we go?
- Efficiency, generalization of Sparse vs Dense scaling?

2/ Modeling of the Massively Multi-Task Neural Networks

Do deep or wide networks drive greater quality gains?

Any→En translation performance with model size



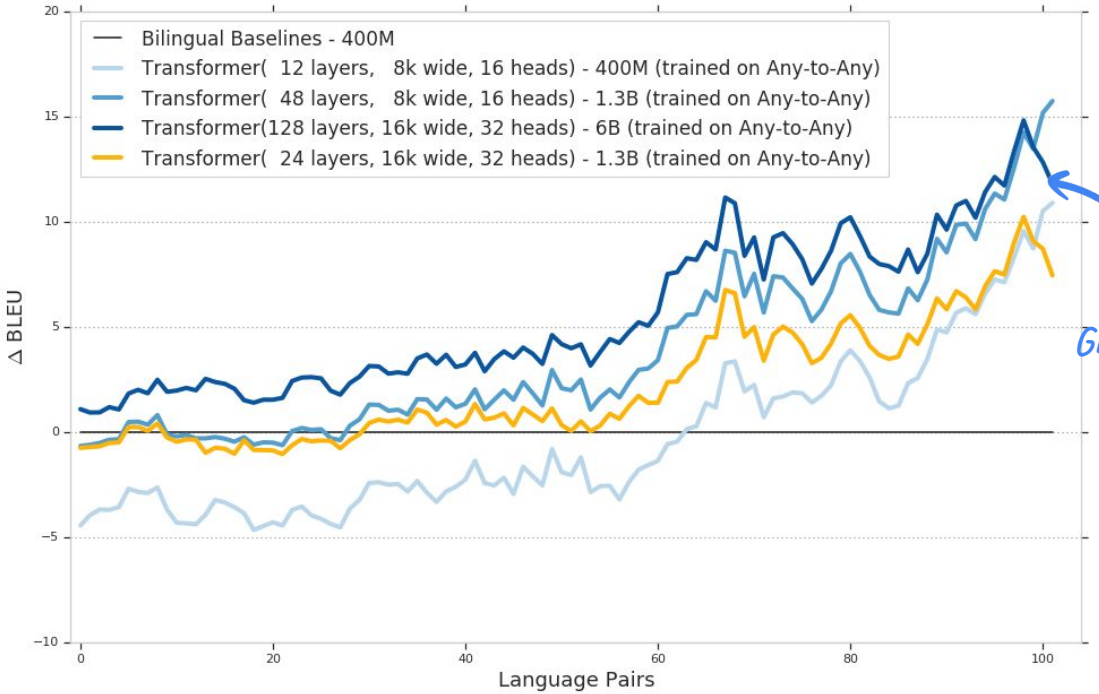
Higher quality for deep vs. wide at same capacity

← Bilingual Baselines

3/ Increase model capacity:

How deep or wide should we go?

All-to-English Translation Quality for M4 Models.



Greatest quality with 128-layer GPipe

Expensive Dense Scaling

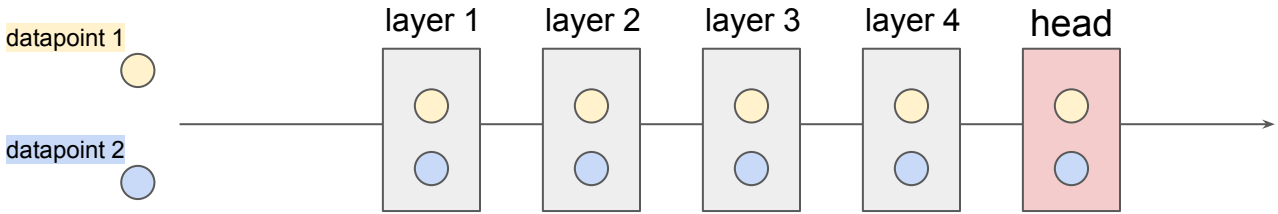
- Increase Depth or width.
- Every input touches the entire network.
- GPT-3, T5, GPipe M4.
- Bottlenecks:
  - Optimization
  - Long training times. 5 weeks!

Relatively Cheap Sparse Scaling

- GShard MoE
- Conditional computation, where only part of the network is active for a given example became crucial after certain point for training and inference
- 4 days training

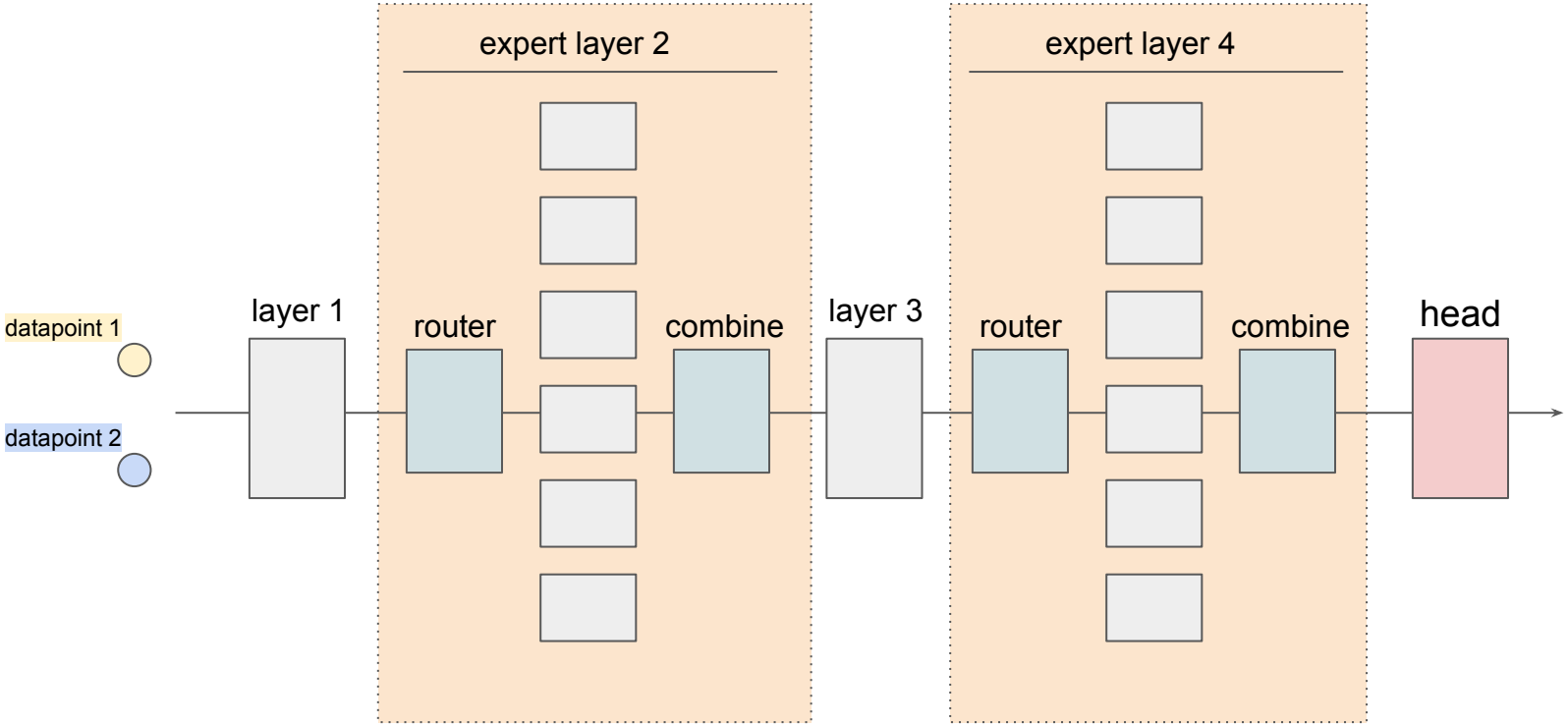
A day in life of a datapoint

dense model



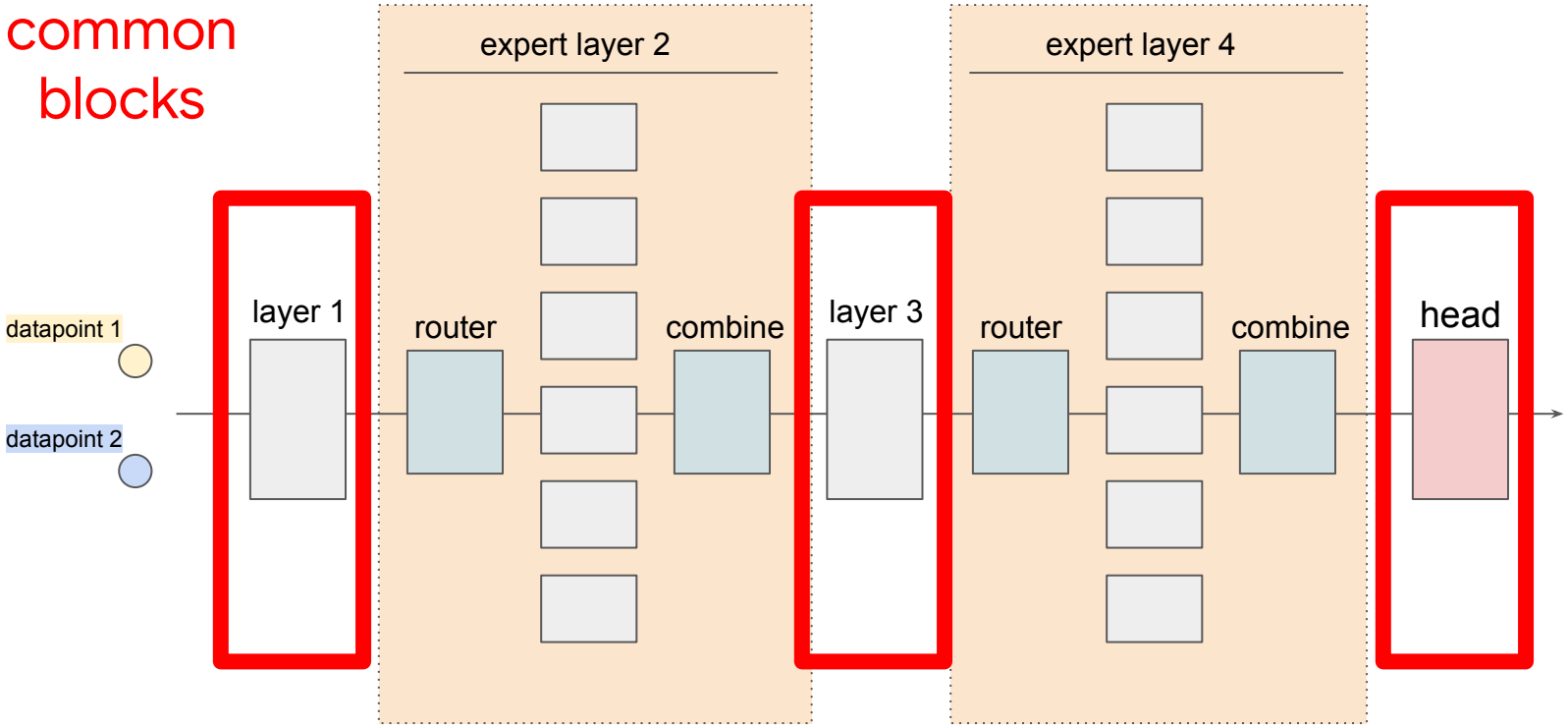
A day in life of a datapoint

Sparsely activated model



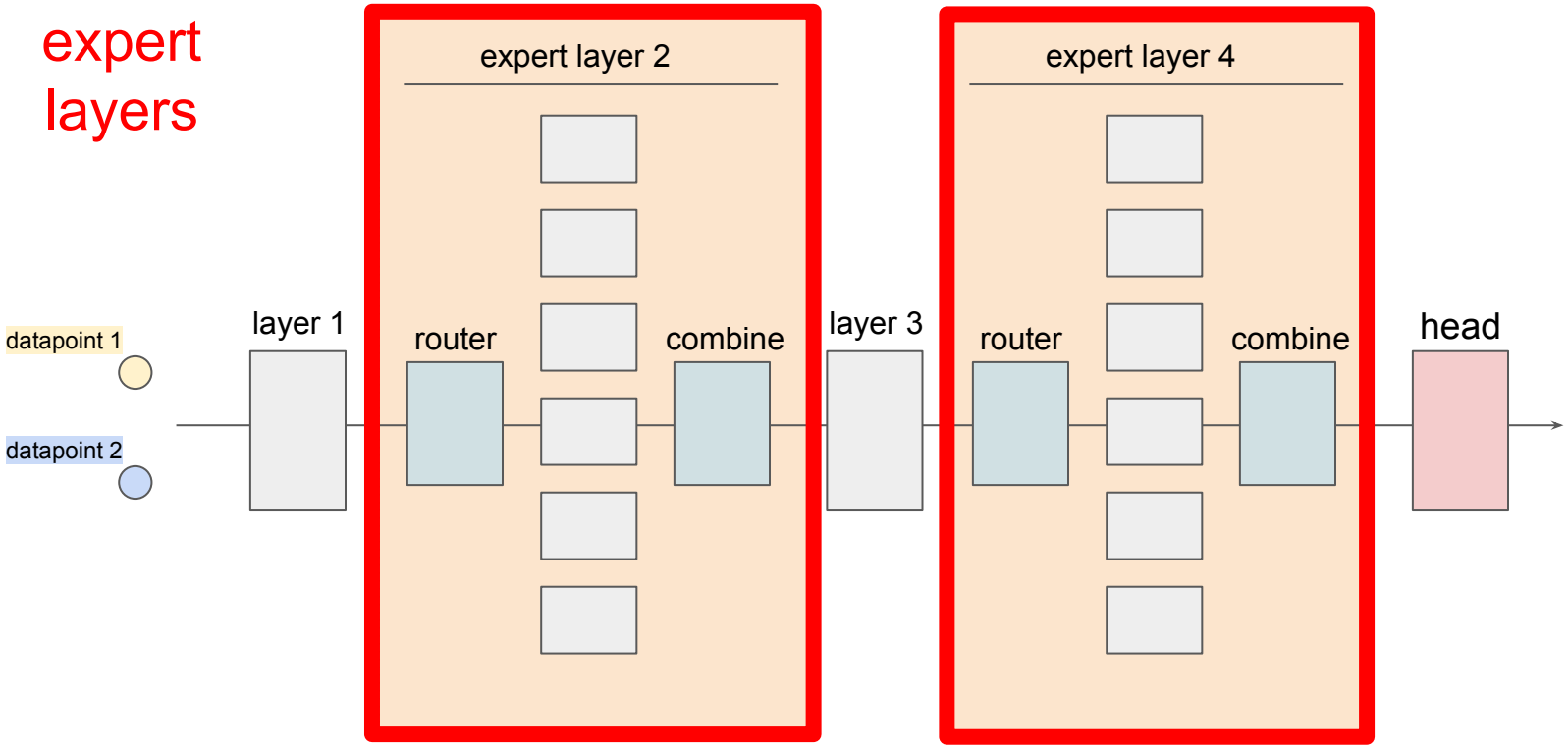
A day in life of a datapoint

Sparsely activated model



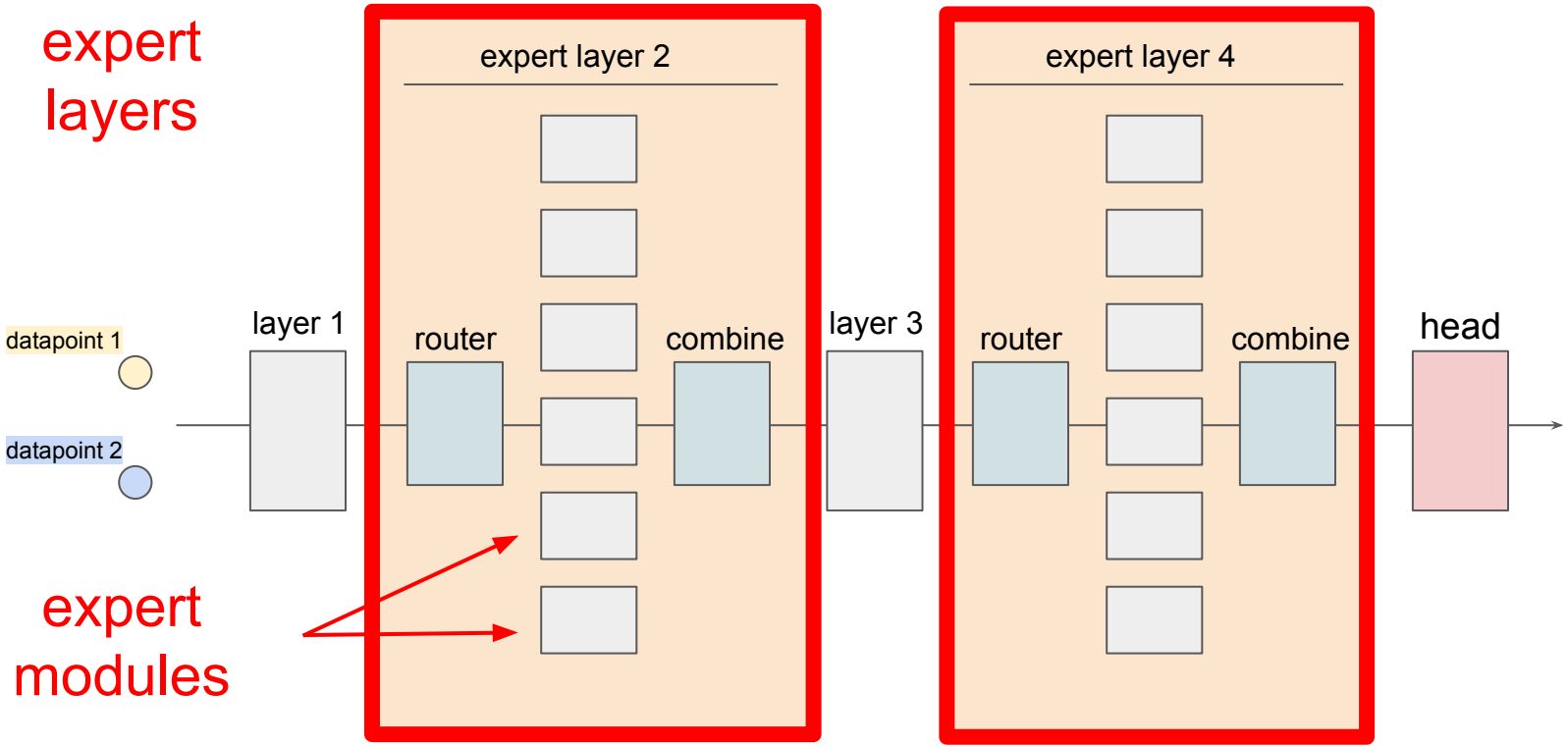
A day in life of a datapoint

Sparsely activated model



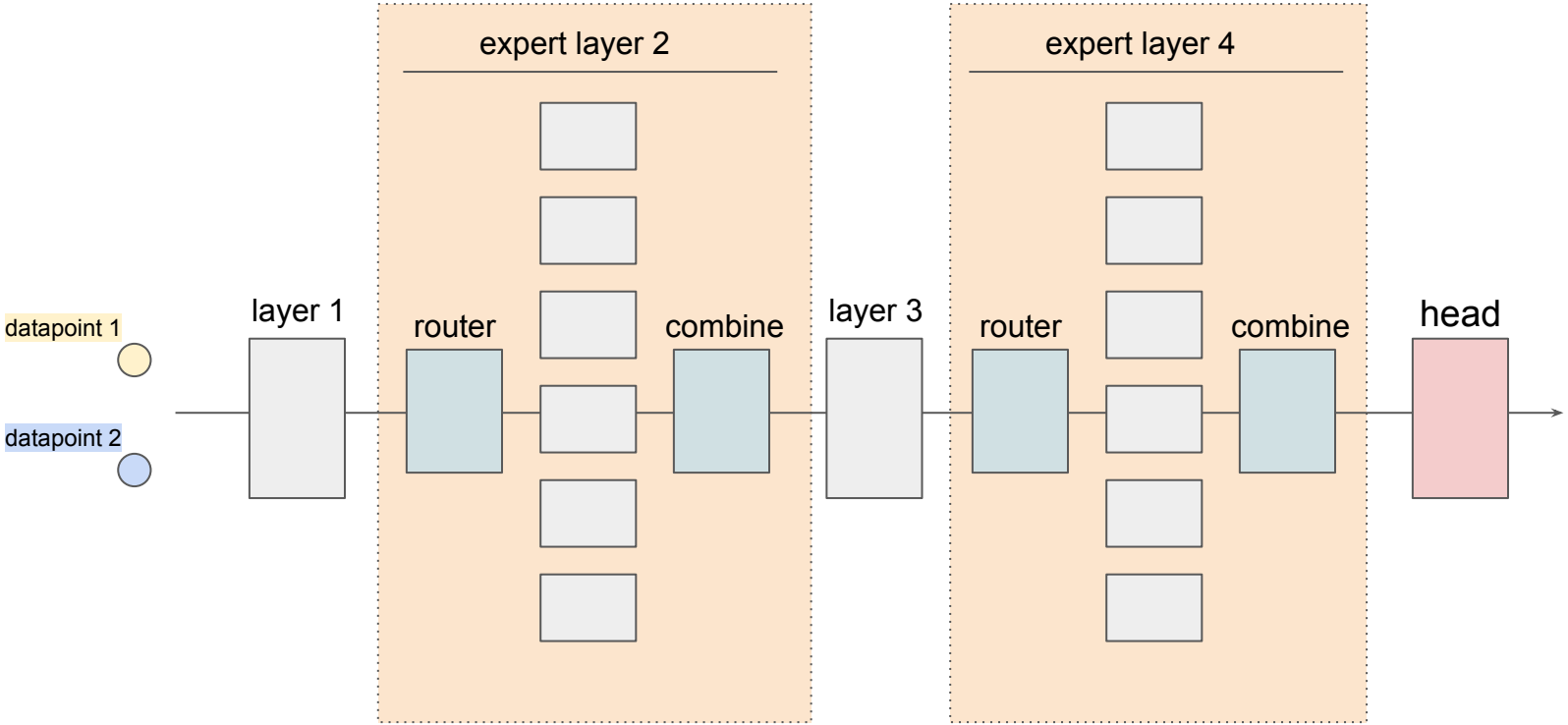
A day in life of a datapoint

Sparsely activated model



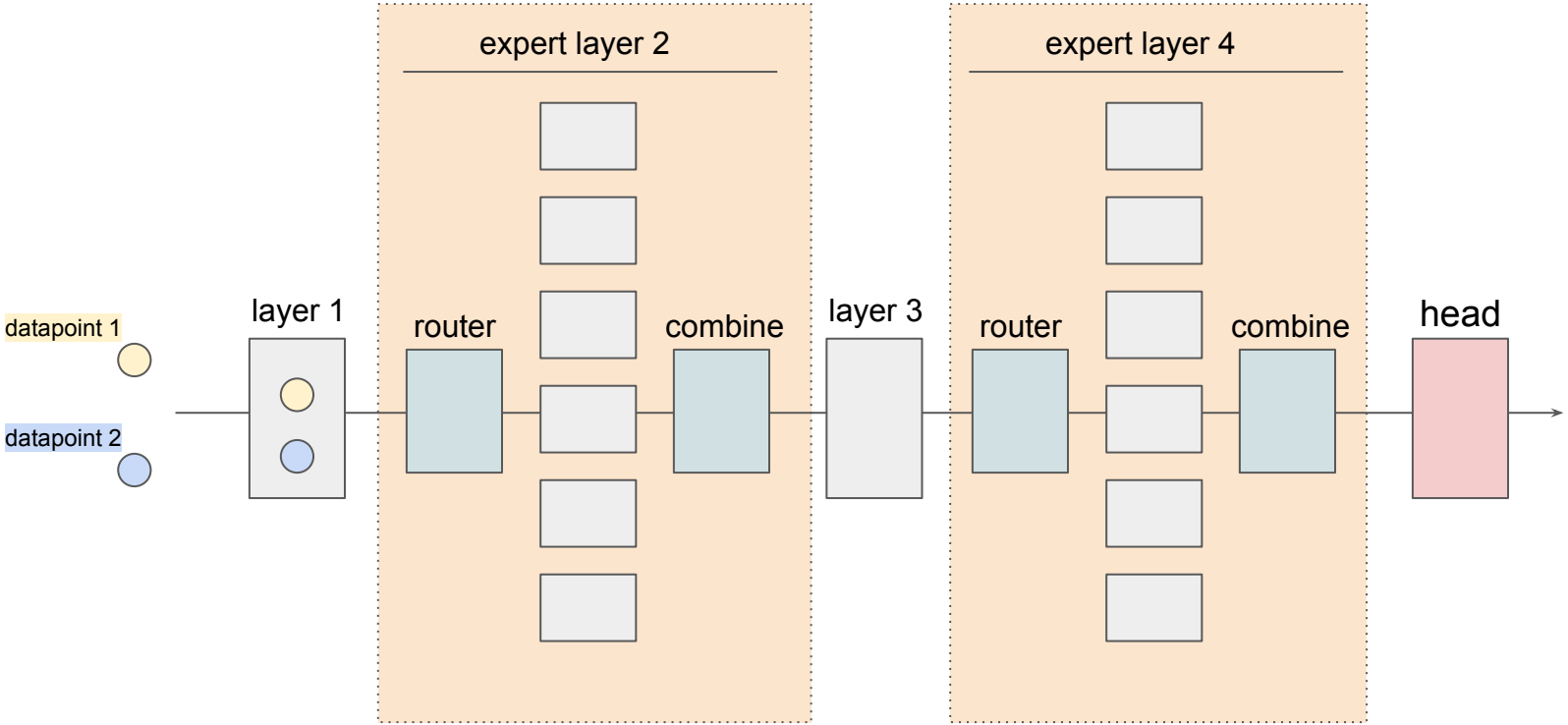
A day in life of a datapoint

Sparsely activated model



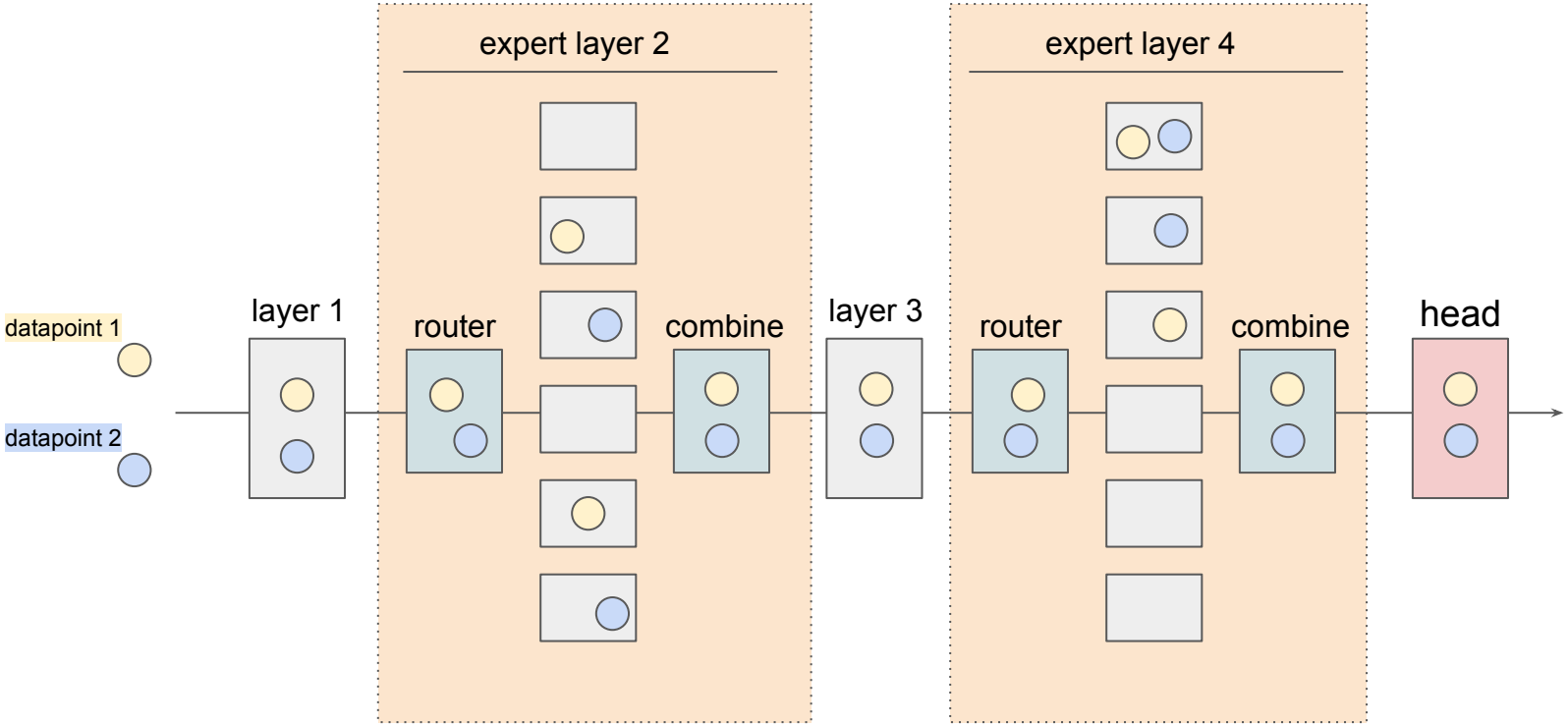
A day in life of a datapoint

Sparsely activated model

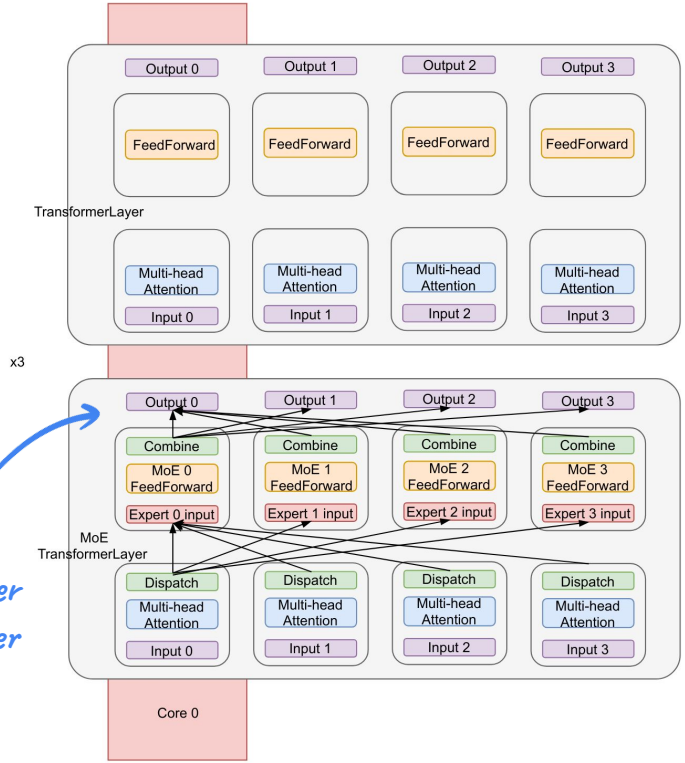


A day in life of a datapoint

Sparsely activated model



How can we scale to 1k chips and beyond?
Conditional computation and Mixture-of-Experts with 50B+ weights



Replaces every other Transformer feed-forward layer with MoE

As we increase number of experts, training and inference scales sublinearly:

- FLOPS $\propto$ batch_size * avg. gated subnetwork size

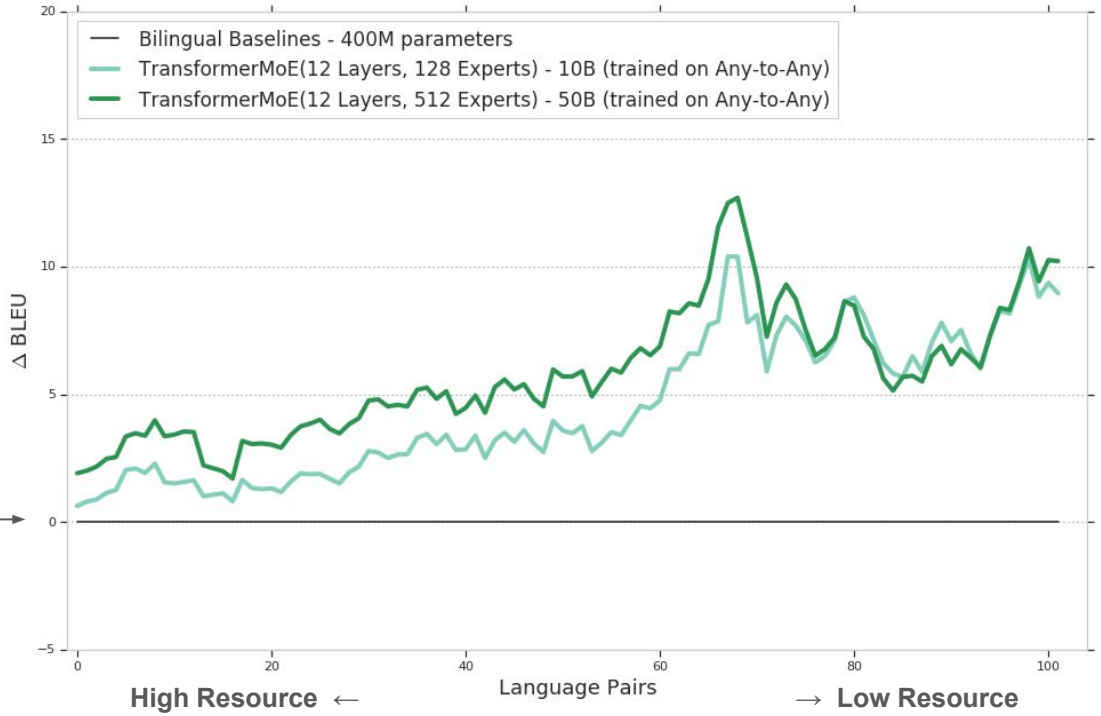
Tradeoffs of 512 token-level experts & 1 expert/core:

- Don't need MoE gradient aggregation
- MoE weight update broadcast is not required
- However, an extra network is required to dispatch activations

2/ Increase model capacity:

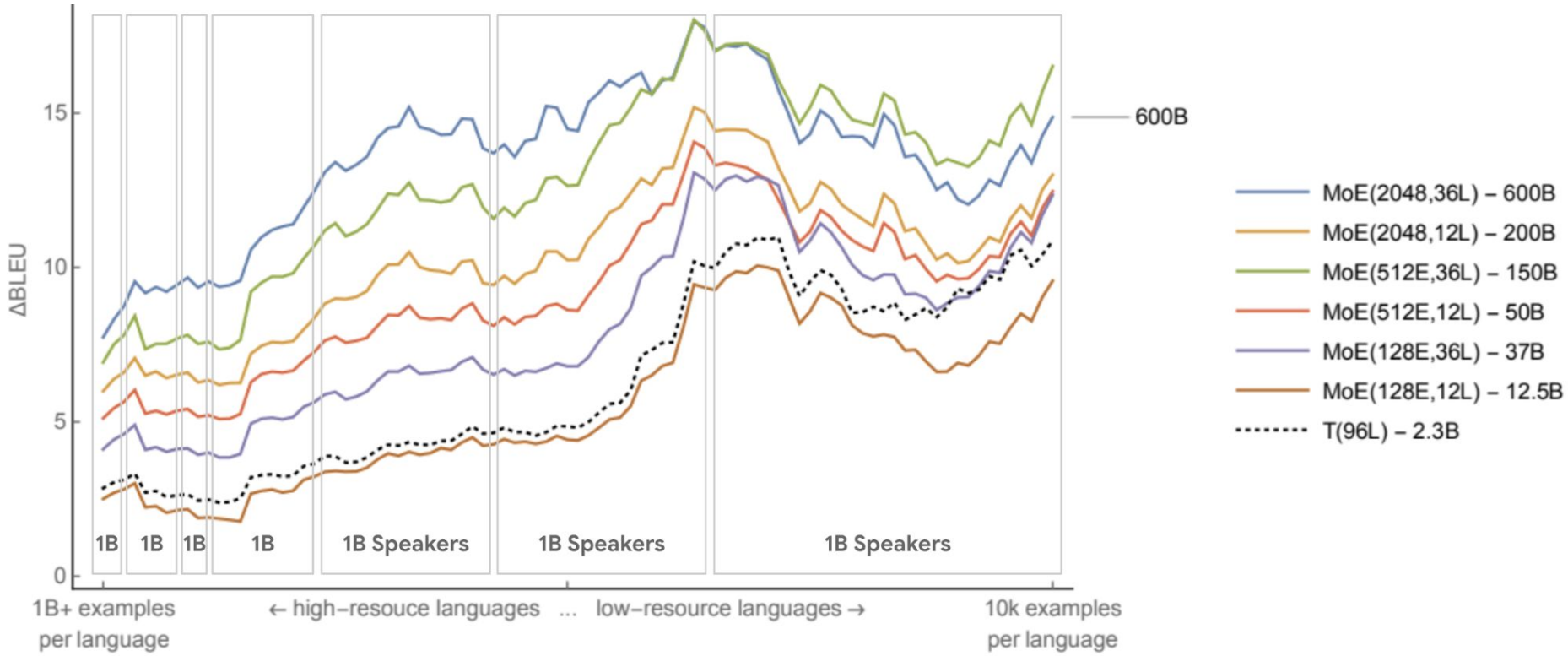
With MoE, bigger is better: 50B 512-MoE > 10B 128-MoE

Bilingual baselines →



2/ Increase model capacity:

Limits of Sparsely Activated Scaling and the Transfer Dynamics

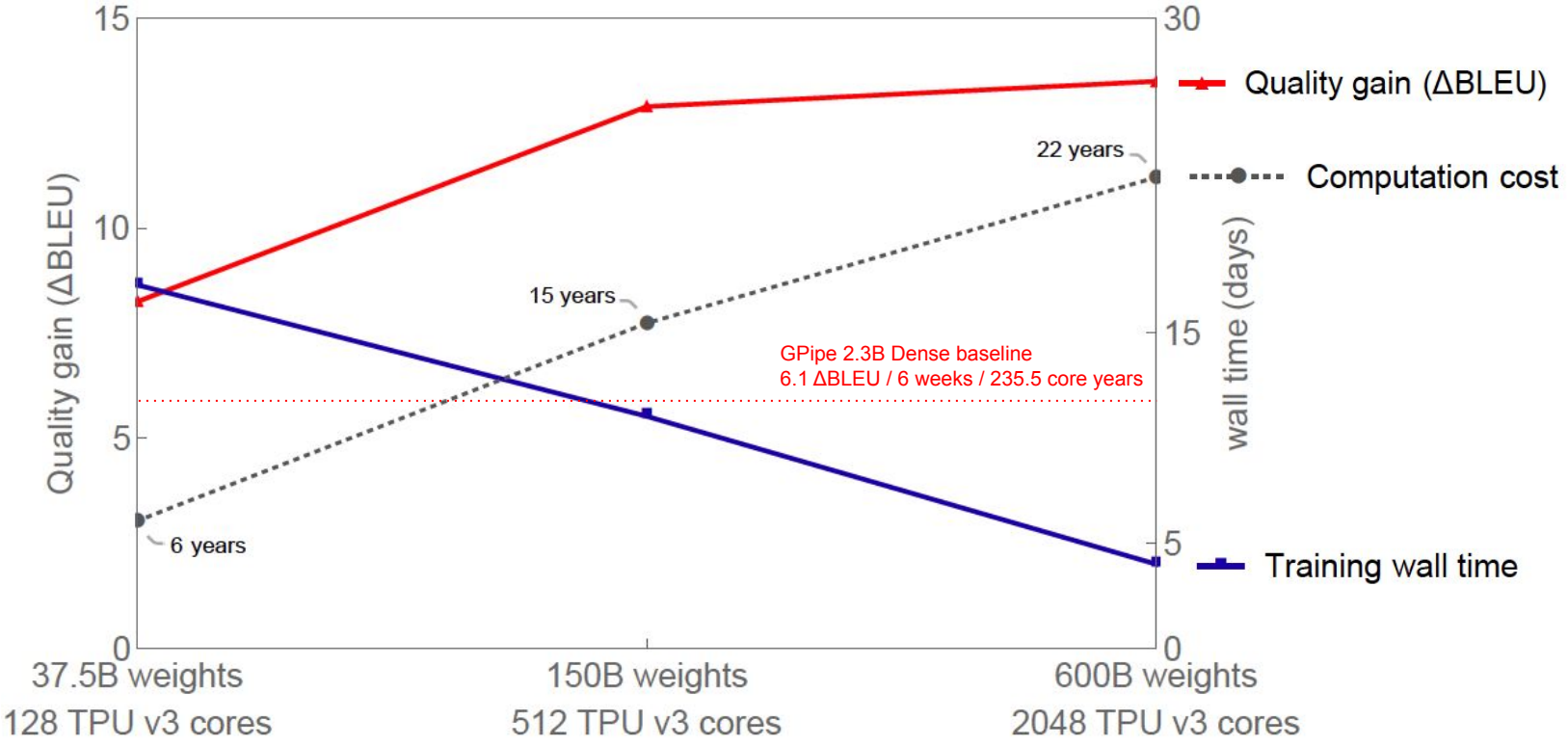


- 600B MoE brings additional 7.4 BLEU gains and is 10x more efficient compared to the GPipe Dense baseline
  - 2.3B => 600B
    ΔBLEU 6.1 => 13.5
    walltime 6 weeks => 4 days
    cost 235.5 => 22 TPU core years
    scaling dense (pipelined) => sparse (sharded)

- Scaling MoE from 37.5B to 600B (16x), results in sublinear training cost increase from 6 to 22 years (3.6x)

2/ Modeling of the Massively Multi-Task Neural Networks

- Deeper models are more sample efficient, converge faster with fewer examples.
- Relaxing the capacity bottleneck grants pronounced quality gains.
- Model with 600B parameters trained under 4 days achieved the best quality.
- Deeper models are better at positive transfer towards low-resource.



- 600B MoE brings additional 7.4 BLEU gains and is 10x more efficient compared to the best Dense baseline.
- Scaling MoE from 37.5B to 600B (16x), results in sublinear training cost increase from 6 to 22 years (3.6x).

Overview

1/ The Shades of Training Data

2/ Modeling of the Massively Multi-Task Neural Networks

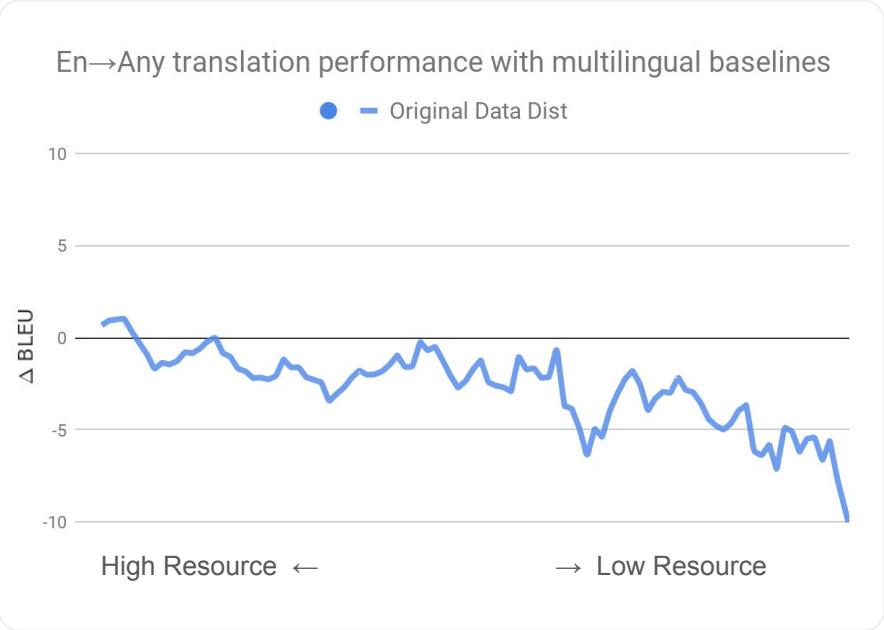
3/ **Training Regimes and Optimization at Scale**

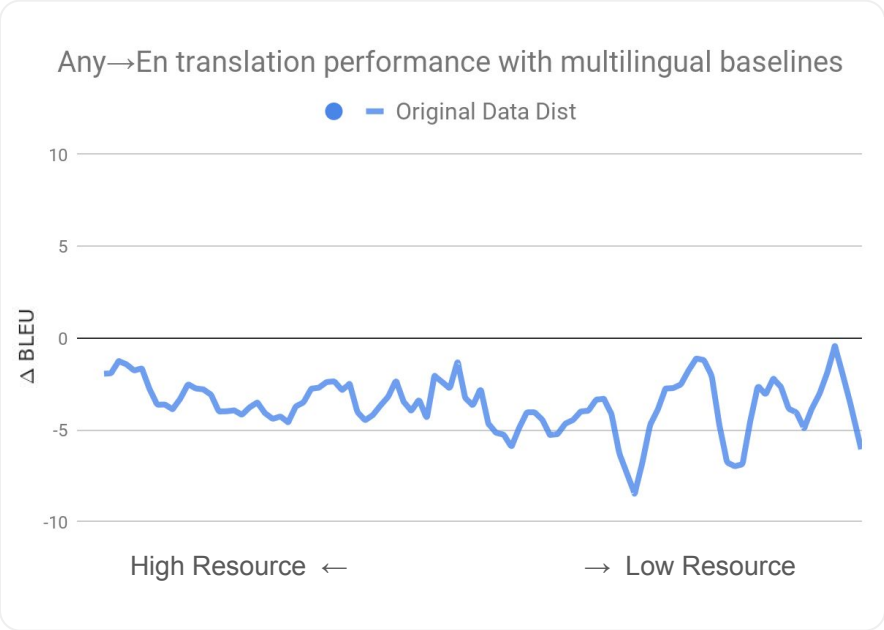
4/ Systems at the Age of Giant Models

3/ Training Regimes and Optimization at Scale

Learn, given data imbalance:

Initial baseline on Any→Any model





3/ Training Regimes and Optimization at Scale

How to sample, form a batch and update under imbalanced scenarios?

1

Look at the data:
Sampling Heuristics
Learn to sample

2

Look at the model:
Alter Learning Rates
Alter Gradients

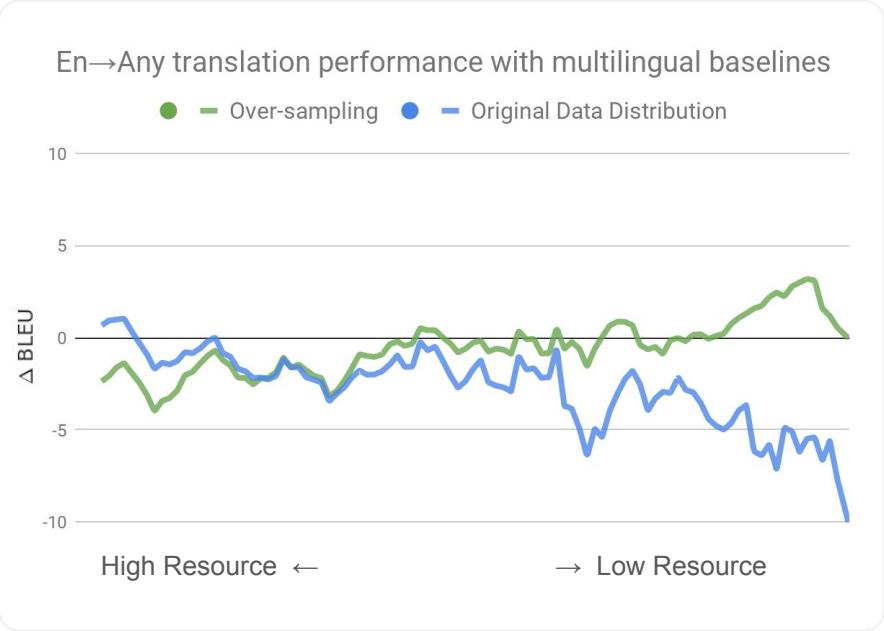
3

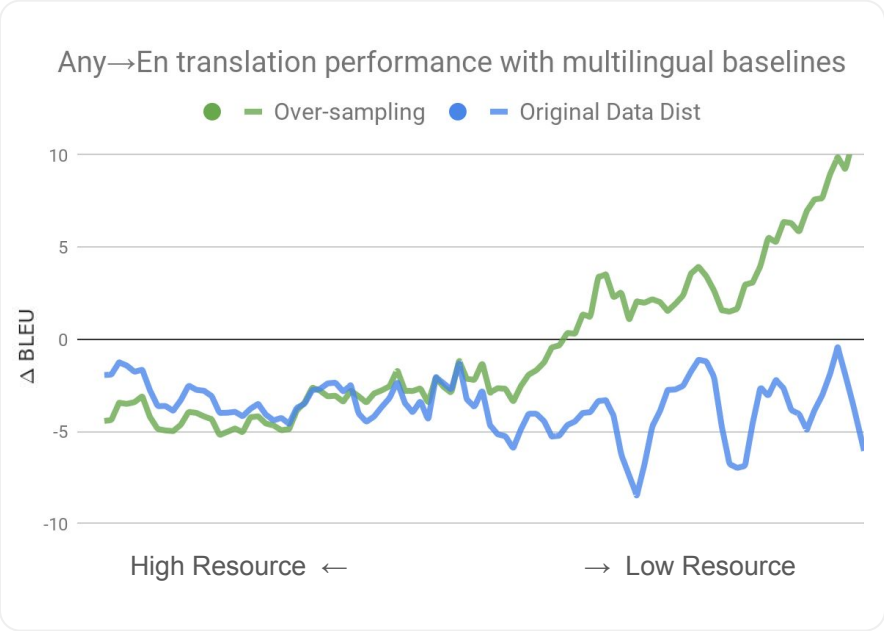
Look at both:
?

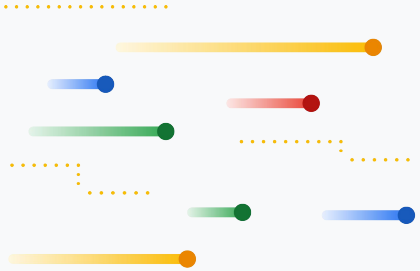
3/ Training Regimes and Optimization at Scale

Learn, given data imbalance:

Re-balance data with temperature







Google AI

The Learning Rate Schedules

“Often the single most important hyper-parameter”

Practical recommendations for gradient-based training of deep architectures, Bengio 2012

Should always be tuned.

Importance of Configs

For large scale experiments:

- Reproducibility is more important than code reuse, cosmetics and other conventions
- Maintaining sufficient checkpoints
- Having experimental results attached to the configs

DO NOT INHERIT CONFIG CLASSES

```
@model_registry.RegisterSingleTaskModel
class WmtEnDeTransformerBase(base_model_params.SingleTaskModelParams):
  """Params for WMT'14 En->De."""

  DATADIR = '/usr/local/google/wmt14/wpm/'
  VOCAB_SIZE = 32000

  @classmethod
  def Train(cls):
    p = input_generator.NmtInput.Params()
    p.file_pattern = 'tfrecord:'+os.path.join(cls.DATADIR, 'train.tfrecords-*')
    p.tokenizer.token_vocab_filepath = os.path.join(cls.DATADIR,'wpm-ende.voc')
    p.bucket_batch_limit = ([128, 102, 85, 73, 64, 51, 42])
    return p

  @classmethod
  def Dev(cls):
    p = input_generator.NmtInput.Params()
    p.file_pattern = 'tfrecord:' + os.path.join(cls.DATADIR, 'dev.tfrecords')
    p.tokenizer.token_vocab_filepath = os.path.join(cls.DATADIR, 'wpm-ende.voc')
    p.bucket_batch_limit = [16] * 8 + [4] * 2
    return p

  @classmethod
  def Test(cls):
    p = input_generator.NmtInput.Params()
    p.file_pattern = 'tfrecord:' + os.path.join(cls.DATADIR, 'test.tfrecords')
    p.tokenizer.token_vocab_filepath = os.path.join(cls.DATADIR, 'wpm-ende.voc')
    p.bucket_batch_limit = [16] * 8 + [4] * 2
    return p

  @classmethod
  def Task(cls):
    p = base_config.SetupTransformerParams(
        model.TransformerModel.Params(),
        name='wmt14_en_de_transformer_base',
        vocab_size=cls.VOCAB_SIZE,
        model_dim=512,
        hidden_dim=2048,
        num_heads=8,
        num_layers=6,
        residual_dropout_prob=0.1,
        input_dropout_prob=0.1,
        learning_rate=3.0,
        warmup_steps=40000)
    p.eval.samples_per_summary = 7500
    return p
```

https://github.com/tensorflow/lingvo/blob/master/lingvo/tasks/mt/params/wmt14_en_de.py

3/ Training Regimes and Optimization at Scale

- Sampling heuristics can become sup-optimal at scale
- Model internals, chosen carefully, can be handy
- Simple gradient manipulation can go long way, and it scales

Overview

1/ The Shades of Training Data

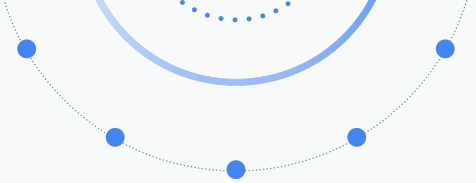
2/ Modeling of the Massively Multi-Task Neural Networks

3/ Training Regimes and Optimization at Scale

4/ Systems at the Age of Giant Models

Google AI

How to Scale using GShard



Google AI

Components of a ML System

Training

- Reads the data, computes loss and gradients, applies parameter update.
- The most compute intensive job, runs on TPU

Inference

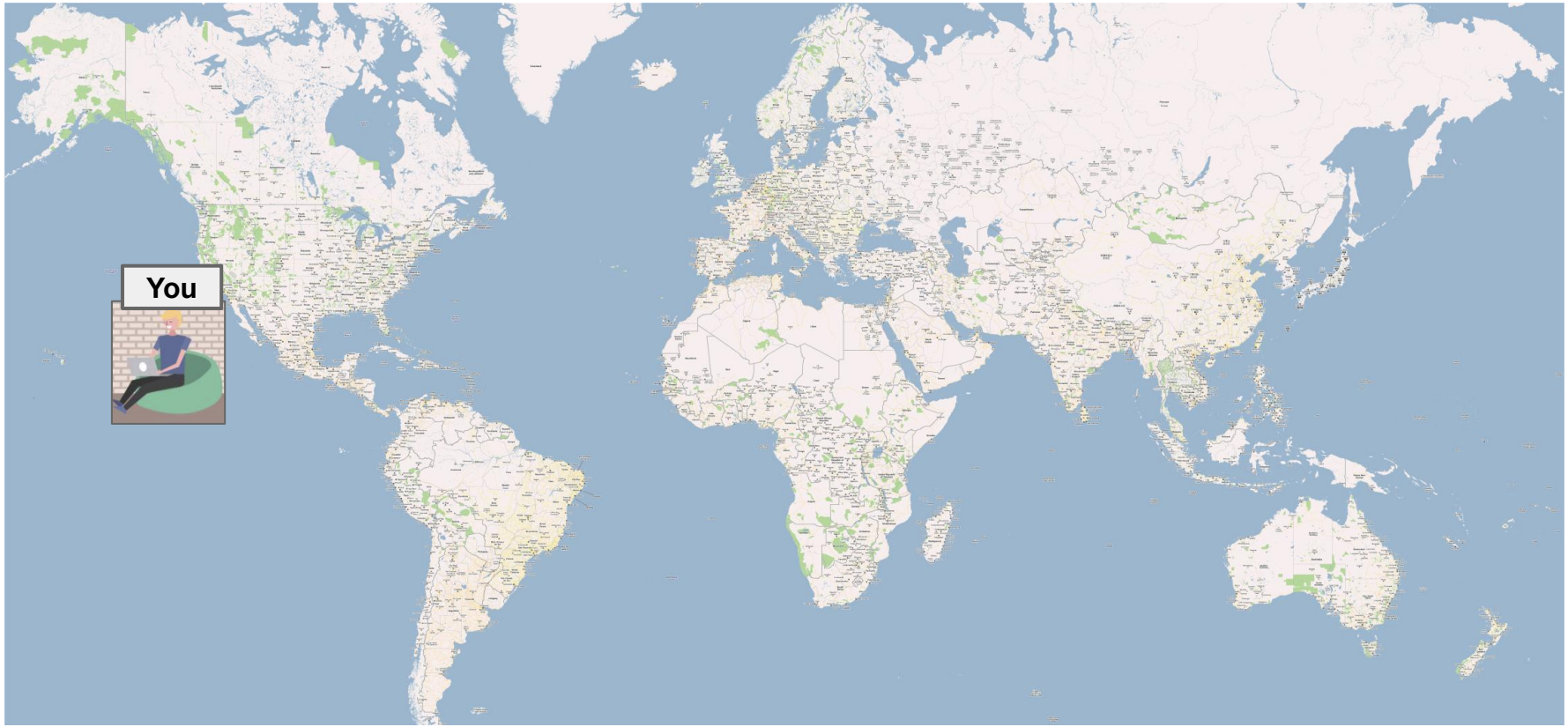
- Reads a checkpoint of the trained model, runs inference (beam-search)
- Generating output sequences, usually runs on GPU

Evaluation

- Reads a checkpoint of the trained model, computes loss on dev set.
- Used for monitoring the progress, usually runs on GPU or CPU

Under the hood

Google AI



*This is just a sketch, exact locations are inaccurate.

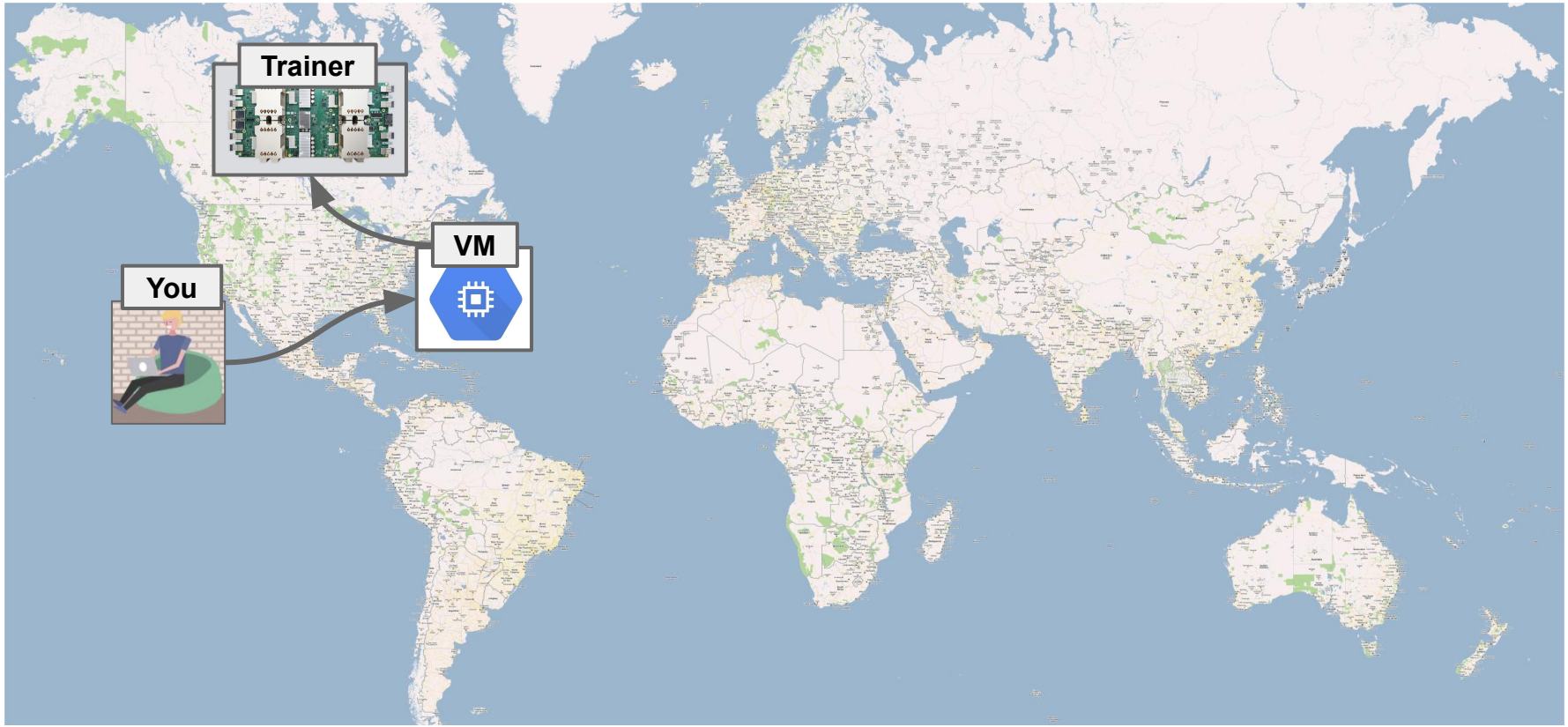
Under the hood

Google AI

*This is just a sketch, exact locations are inaccurate.

Under the hood

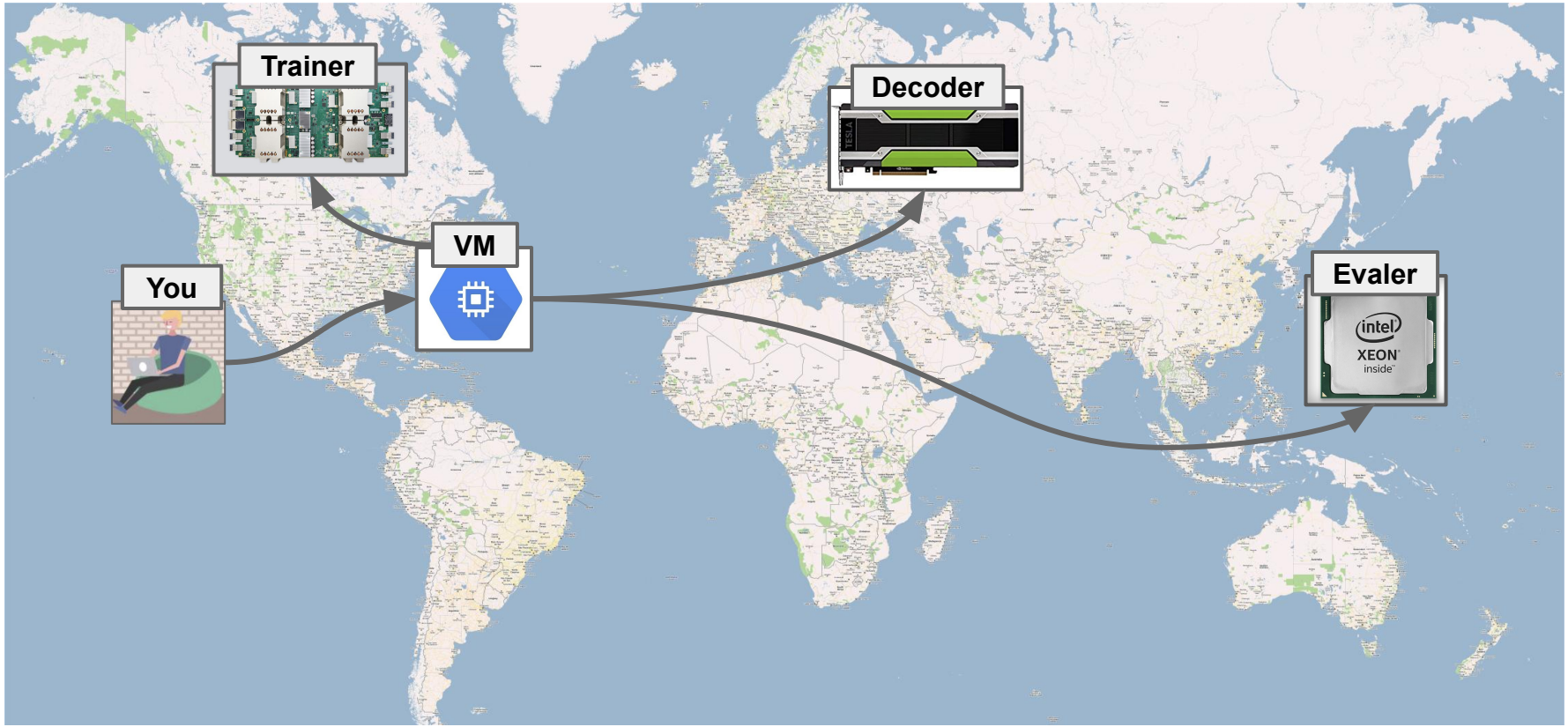
Google AI



*This is just a sketch, exact locations are inaccurate.

Under the hood

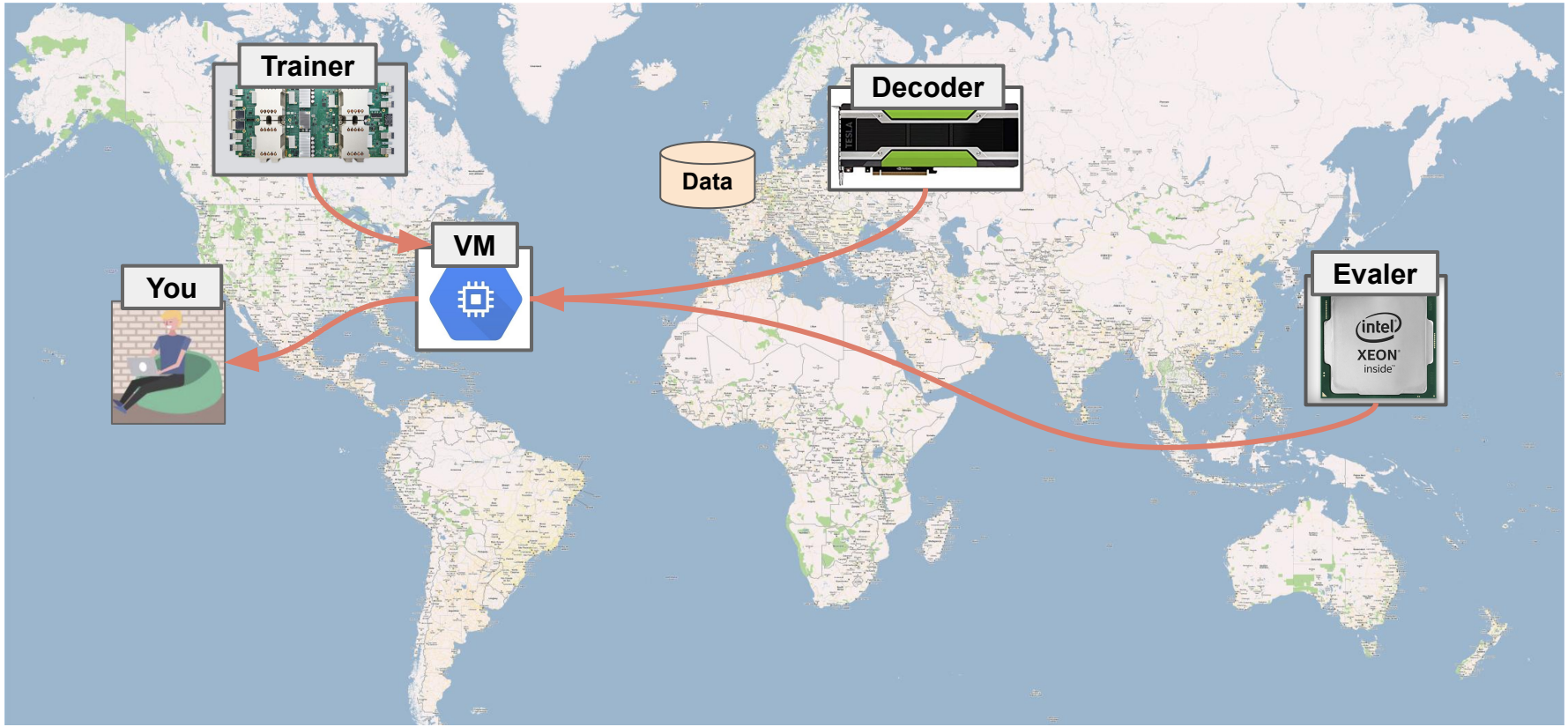
Google AI



*This is just a sketch, exact locations are inaccurate.

Under the hood

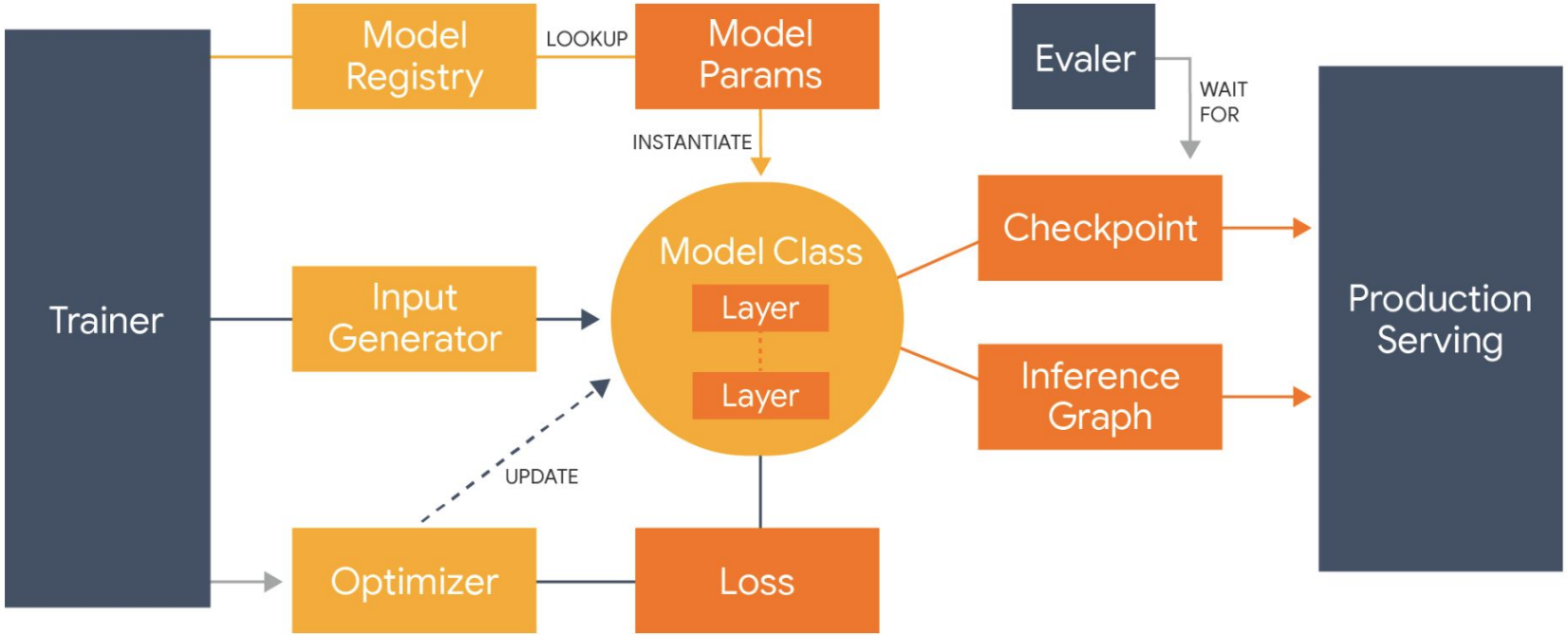
Google AI



*This is just a sketch, exact locations are inaccurate.

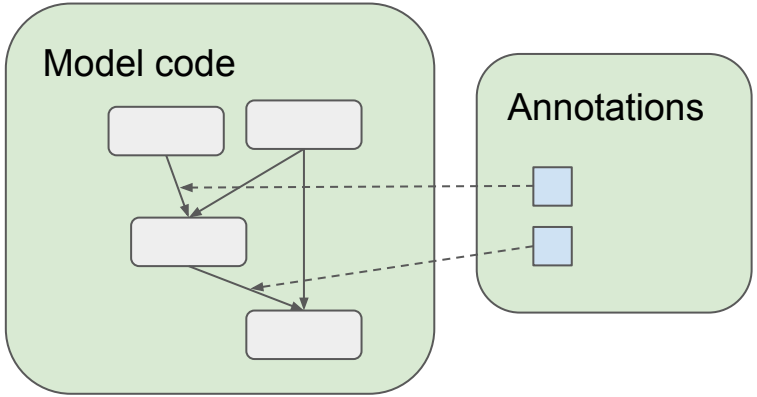
Google AI

Tensorflow Lingvo: github.com/tensorflow/lingvo

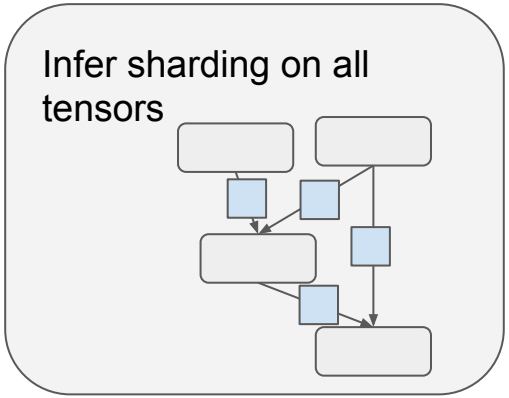


Decoupling model building and sharding

Write model code as if there were a single, large device

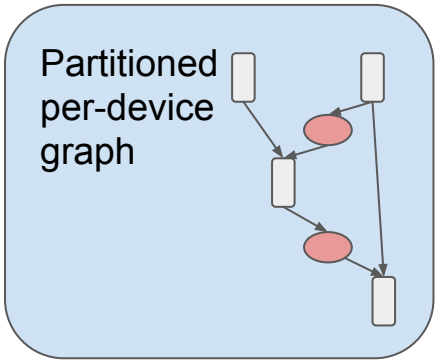


Annotate some key tensors for sharding



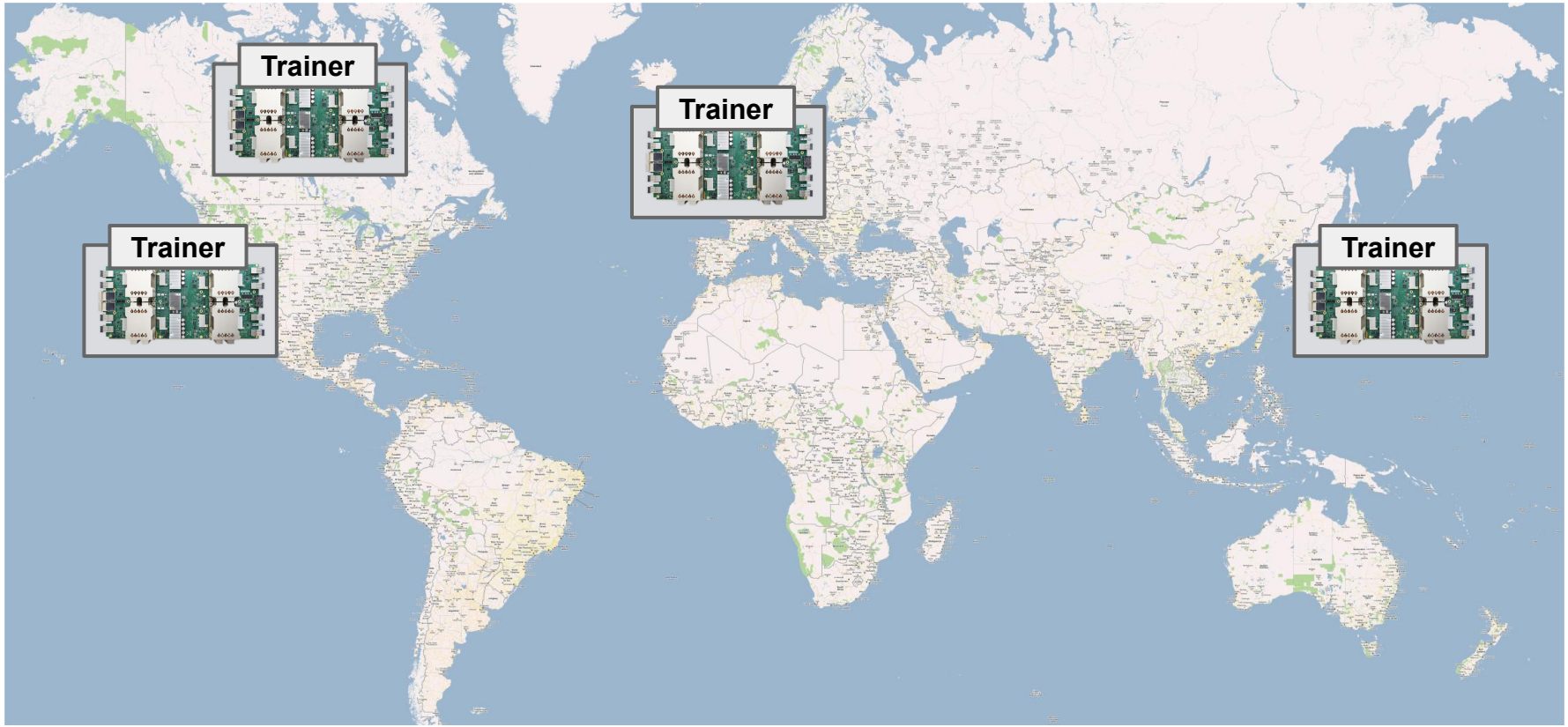
XLA SPMD partitioner

Partitioned graph with collective cross-core communication ops

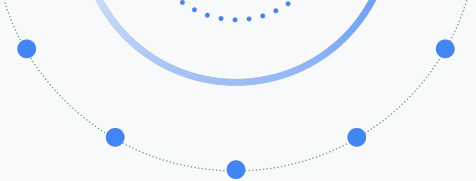


Scaling the Model Being Trained

Google AI



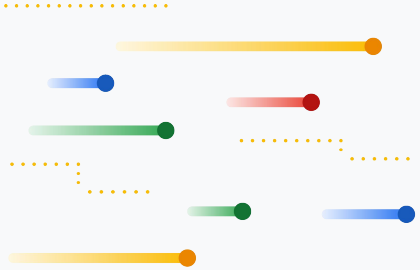
*This is just a sketch, exact locations are inaccurate.



Google AI

THANKS!

huangyp@google.com



Google AI

Appendix

Massively parallel implementation using GShard

GShard

A module composed of a set of lightweight annotation APIs and an extension to the XLA compiler.

↦ Decoupling model building and sharding
↦ Single Program, Multiple Data (SPMD) Partitioning
↦ Generic sharding abstraction: per-tensor annotation
↦ Python annotation APIs

SPMD partitioner internals

↦ Sharding propagation
↦ Per-operator partitioning

Case studies

↦ Sparse Language model (MoE)

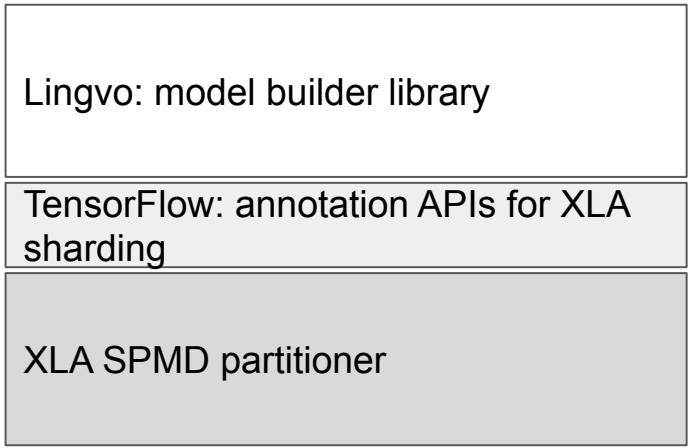
↦ Dense Language model

↦ Image spatial partitioning

GShard Infrastructure Overview

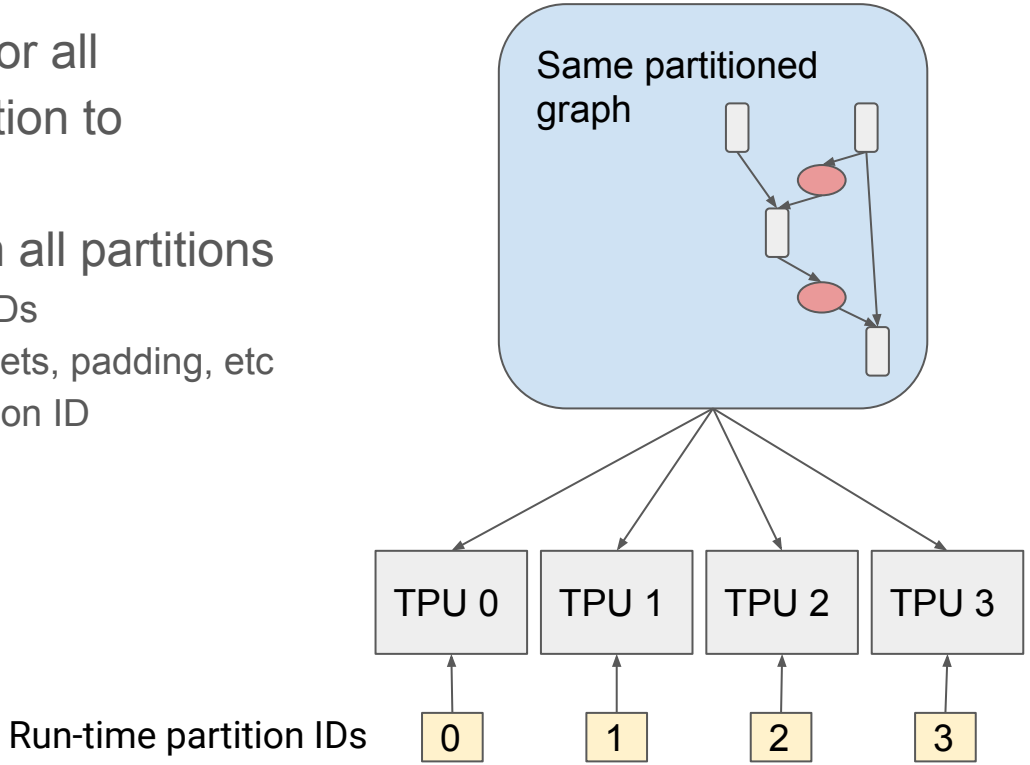
GShard consists of

- High-level sharding utilities and libraries
- TensorFlow wrapper APIs for XLA sharding
- XLA transformations:
  - Sharding propagation
  - SPMD partitioner

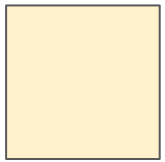


Single Program, Multiple Data (SPMD) Partitioning

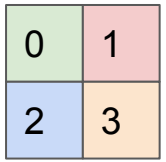
- One-time compilation for all partitions, fast compilation to thousands of partitions
- Same program runs on all partitions
  - Runtime sets partition IDs
  - Program calculates offsets, padding, etc based on runtime partition ID



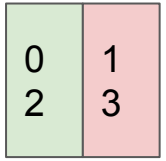
Generic sharding abstraction: per-tensor annotation



1. Replicated:
   - Every partition has the full data



2. Tiled:
   - Every partition has one ¼ data
   - Device order can be specified



3. Partially Tiled:
   - Replicated in subgroups
   - Each subgroup has a different subset of data

Per-tensor specification allows model to switch between different parallelism modes in different parts of the model.

Python annotation APIs

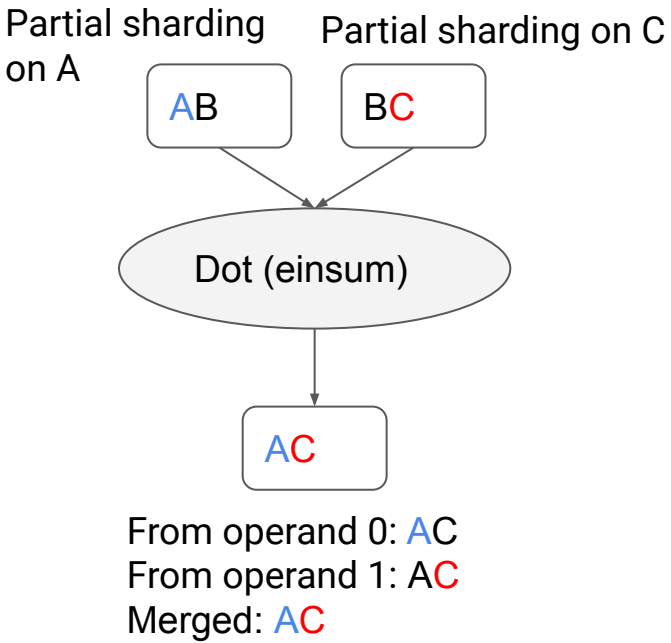
TensorFlow APIs that are semantically `tf.identity` to the user, and only convey sharding annotations for XLA

- `replicate(x)`
- `tile(x, tile_assignment)` # tile_assignment specifies ND array of devices
- `partial_tile(x, tile_assignment)` # last dim of tile_assignment is replication

- `split(x, split_dimension, num_devices)` # wrapper of tile for simple 1D sharding
- `mesh_split(x, device_mesh, split_dims_mapping)` # MeshTF-like API for ND sharding, wrapper of tile/partial_tile

Sharding propagation: auto-completion of sharding specifications

- Pass sharding on each dimension through the HLO ops
- Merge different shardings from different operands
- Different priorities of each propagation rule
- Forward + backward propagation, repeat until fix point

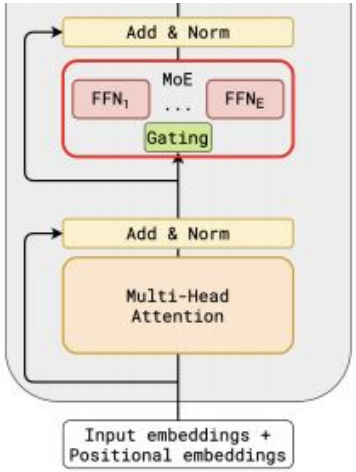


SPMD partitioner

- Per-op transformation to graph with shard sized tensors
- Collective communication
  - AllReduce
  - AllGather
  - ReduceScatter
  - AllToAll
  - CollectivePermute
- Single program for all partitions, yet supports uneven partitioning on static XLA shapes and configurations
  - Padding and masking
- Examples in following case studies

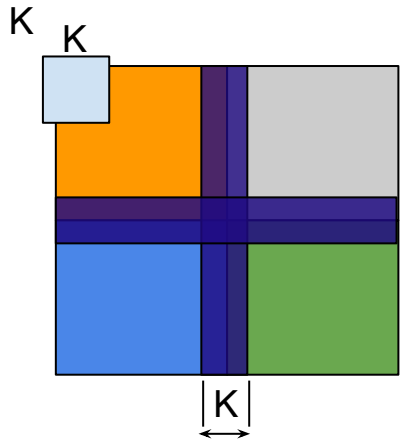
Flexibility: 3 case studies

Sparse Transformer (MoE)



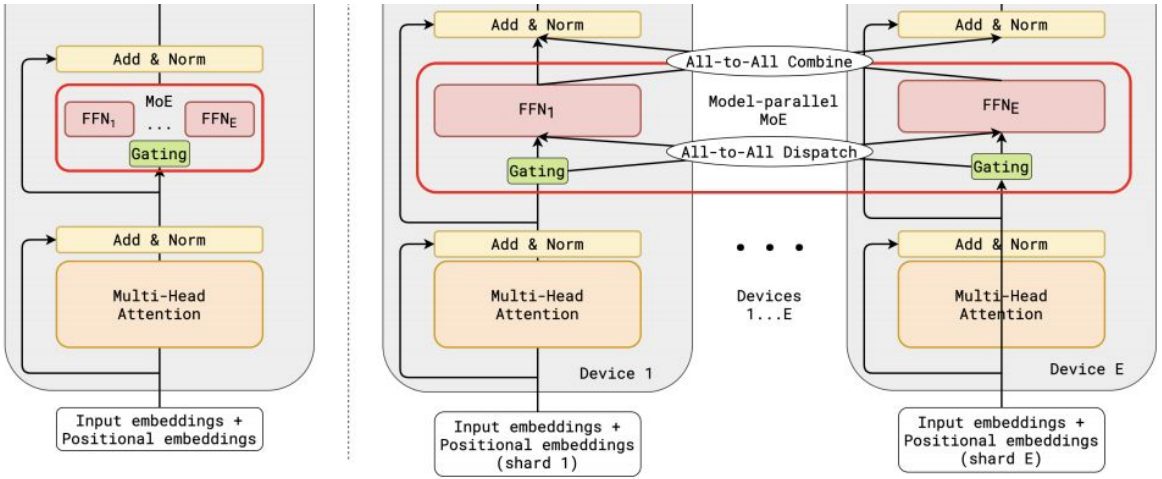
Dense Transformer

Image spatial partitioning



Case Study 1: MoE Transformer

MoE feed-forward layer weights have an expert dimension. Gating network selects experts for each token.



MoE user model

Partitioned program

Partition non-MoE on batch dim (data parallelism)

```
input = split(input, batch_dim)
atten_out = attention(input)
```

Partition MoE feed-forward layer on expert dim

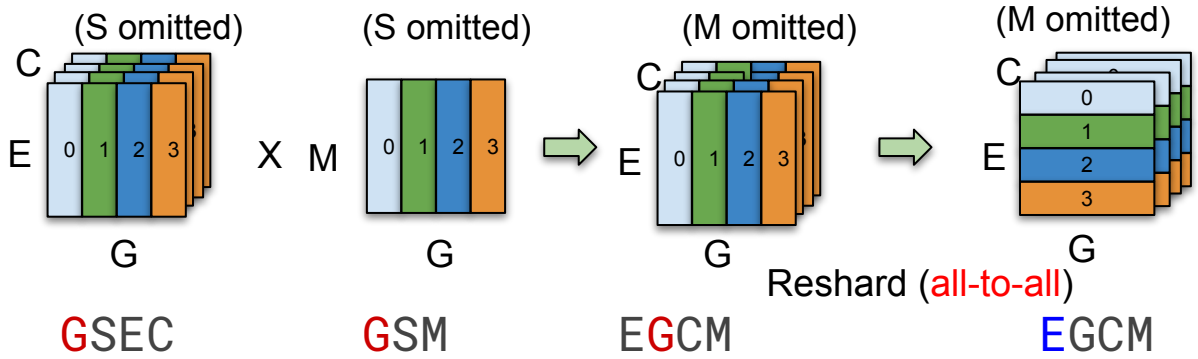
```
in = split(input, expert_dim)
w = split(w, expert_dim)
h = einsum("EAM,EMH->EAH", in, w)
h = relu(h)
out = einsum("EAH,EHM->EAM", h)
```

MoE gating/expert dispatching switches the sharding dim

```
  # Partition inputs along group (G) dim.
+ inputs = split(inputs, 0, D)
  # Replicate the gating weights
+ wg = replicate(wg)
  gates = softmax(einsum("GSM,ME->GSE", inputs, wg))
  combine_weights, dispatch_mask = Top2Gating(gating_logits)
  dispatched_expert_inputs = einsum(
    "GSEC,GSM->EGCM", dispatch_mask, reshaped_inputs)
  # Partition dispatched inputs along expert (E) dim.
+ dispatched_expert_inputs = split(dispatched_expert_inputs, 0, D)
  h = einsum("EGCM,EMH->EGCH", dispatched_expert_inputs, wi)
  ...
```

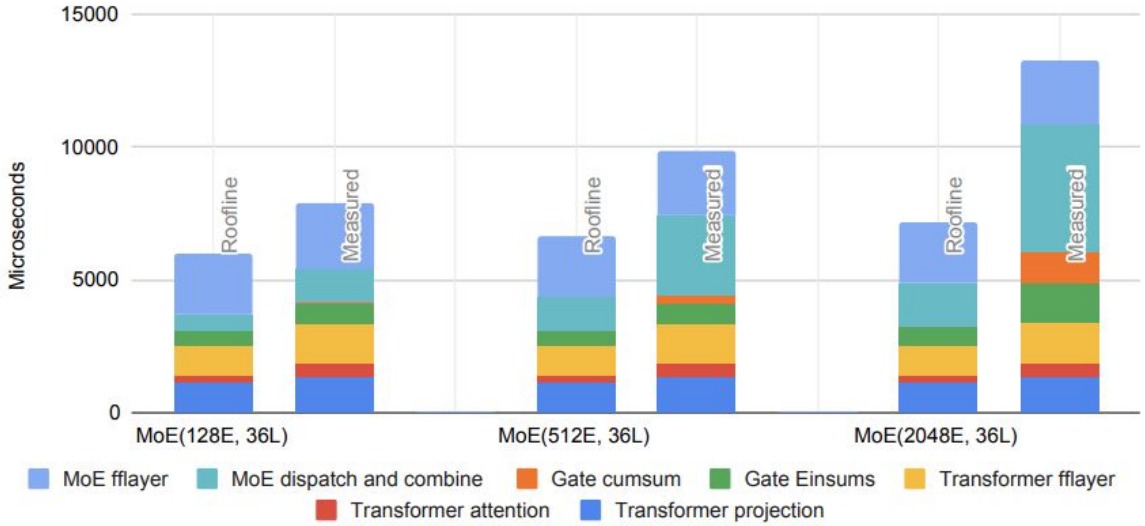
Input split on group (batch) dim

Output split on expert dim



SPMD partitioner will use AllToAll to reshard the tensor

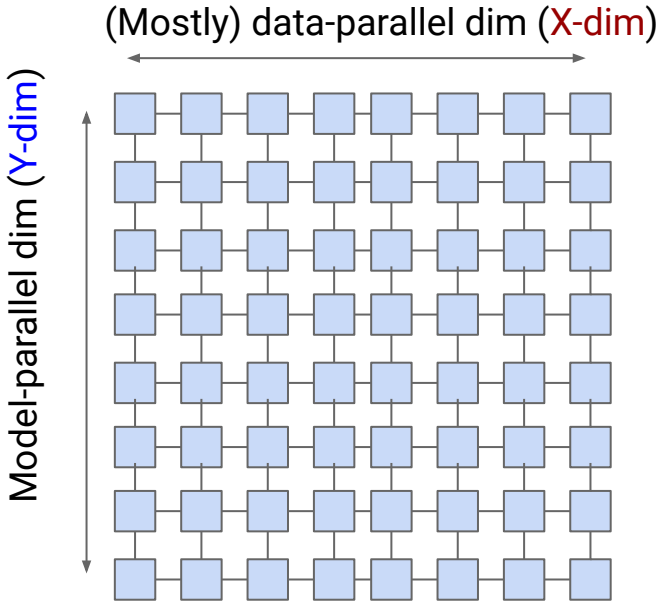
MoE Transformer: performance scaling



- Non-MoE layers are same as small model (pure data parallelism)
- MoE has dispatching cost, AllToAll cost is O(sqrt(num_devices))
- TPUv3-128 FLOPS utilization 52%, TPUv3-2048 FLOPS utilization 32%

Case Study 2: Dense Transformer

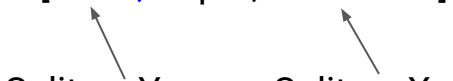
- Wide layers: 2B parameters per layer
- Deep model: > 100 layers
- 2D device mesh
  - Prevents dimensions being sharded too small (which affects TPU efficiency)



2D weight sharding

Attention weight

[Head, Depth, ModelDim]




```
mesh_split(w, mesh, [1, -1, 0])
```

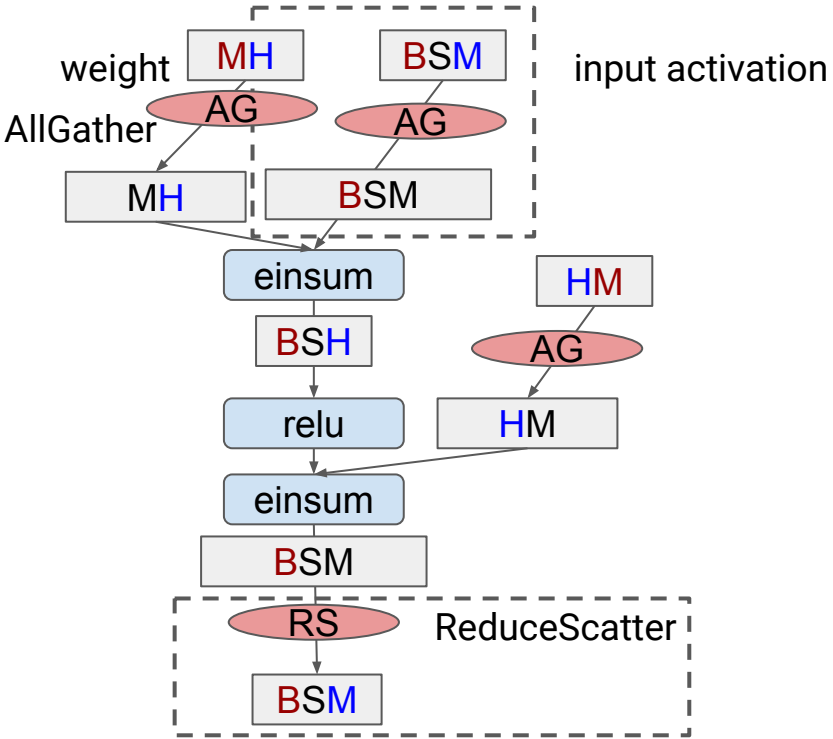
Feed-forward weight

[ModelDim, HiddenDim]


```
mesh_split(w, mesh, [0, 1])
```

- Dims split across Y-dim on mesh
  - Primary model-parallel partitioning. Activations will be split in the same way.
- Dims split across X-dim on mesh
  - Activations already split across batch
  - SPMD Partitioner will AllGather (concat across partitions) weight before use
- Weight is fully partitioned across all devices, zero duplication

Dense Transformer: 2D partitioned graph (feed-forward layer)



Graph produced by SPMD partitioner graph

- Scales to 1T+ parameters on TPUv3-2048
- Communication ops are added by XLA automatically
- Combine the following techniques:
  - Data parallelism
  - Megatron-style model parallelism
  - Weight update sharding (arxiv.org/abs/2004.13336) / optimizer state sharding (arxiv.org/abs/1910.02054)
  - Fully sharded activations

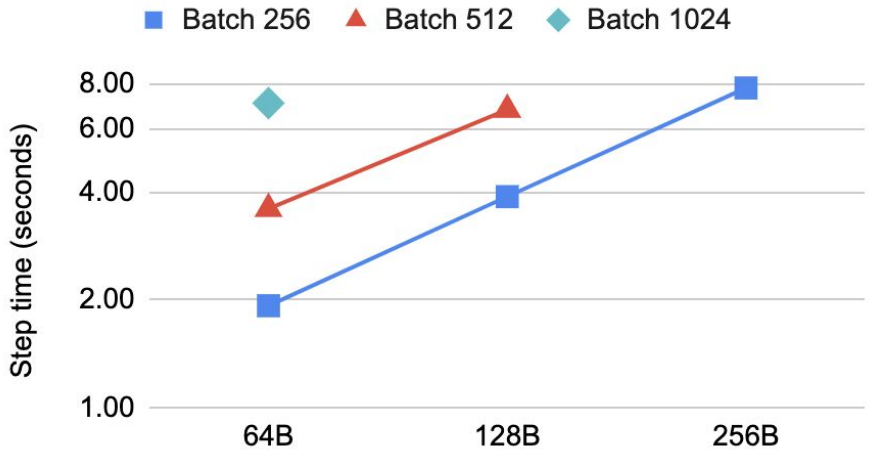
Dense Transformer: nearly perfect memory scaling

- Peak memory is linear to number of layers, and batch size
- No rematerialization needed
- Open source: https://github.com/tensorflow/lingvo/tree/master/lingvo/tasks/lm

Parameter count	# of layers	Max batch size with BFloat16 activations	
		TPUv3-1024	TPUv3-2048
64B	32	512	1024
128B	64	256	512
256B	128	128	256

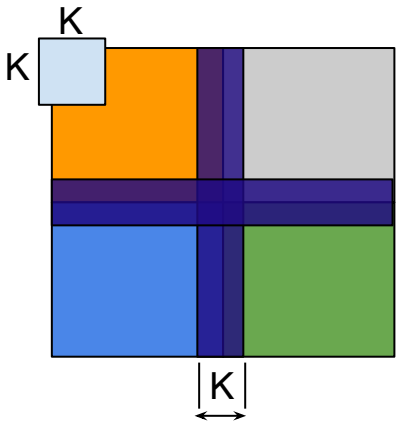
Good, predictable performance scaling

- Step time roughly linear to batch size and number of layers
- > 50% FLOPS utilization on TPUv3 pod



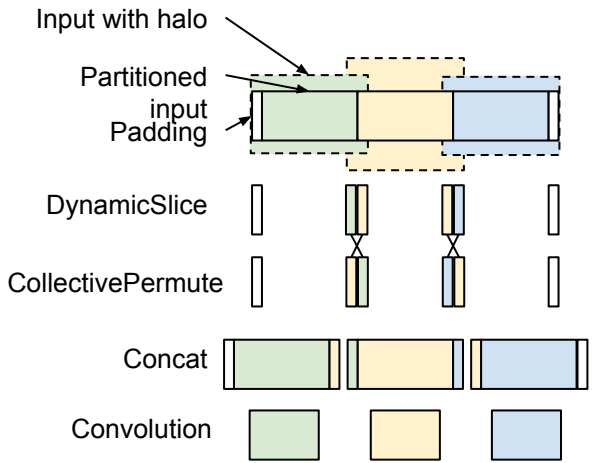
Case Study 3: Image spatial partitioning

- Object detection/segmentation models
  - High resolution images
  - Very large inputs and activations
- Spatial partitioning
  - Partitions input on spatial dimensions
- User ***only needs to annotate input***
  - GShard will propagate spatial sharding to all layers



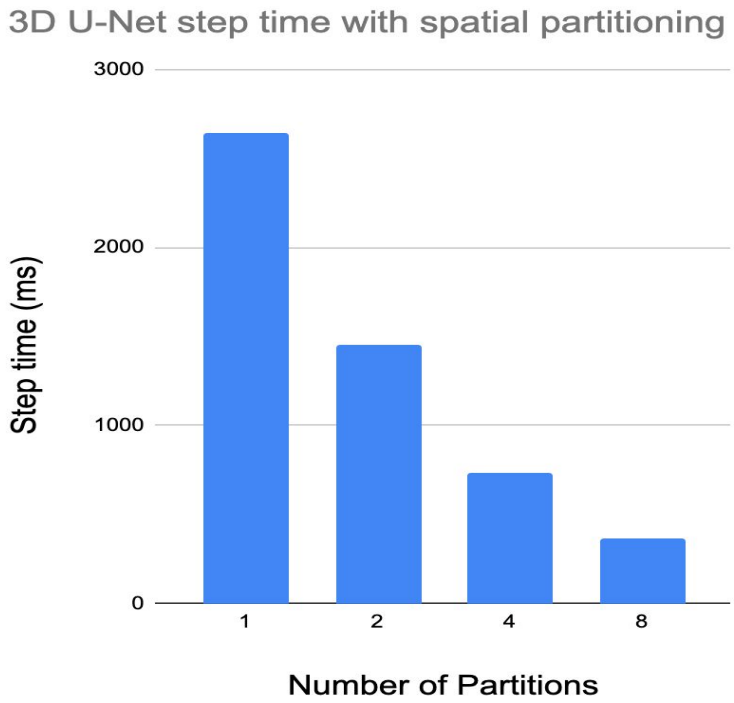
Convolution partitioning

- Halo exchange for overlapping windows
  - Data exchange with other partitions
  - Need extra padding and slicing to deal with dynamism:
    - Convolution's paddings can depend on PartitionID
    - Halo size could depend on PartitionID



3D U-Net benchmarks

- 3D inputs
- Scales well with GShard spatial partitioning:
  - 7.2x speedup on 8 partitions



GShard summary

- GShard decouples model definition and partitioning
- GShard conveniently supports different parallelism patterns
- GShard is flexible and supports various image and language models
- We used it to train a 600B multilingual translation model (for 100 languages) achieving massive 13.5 BLEU gain

Dense:

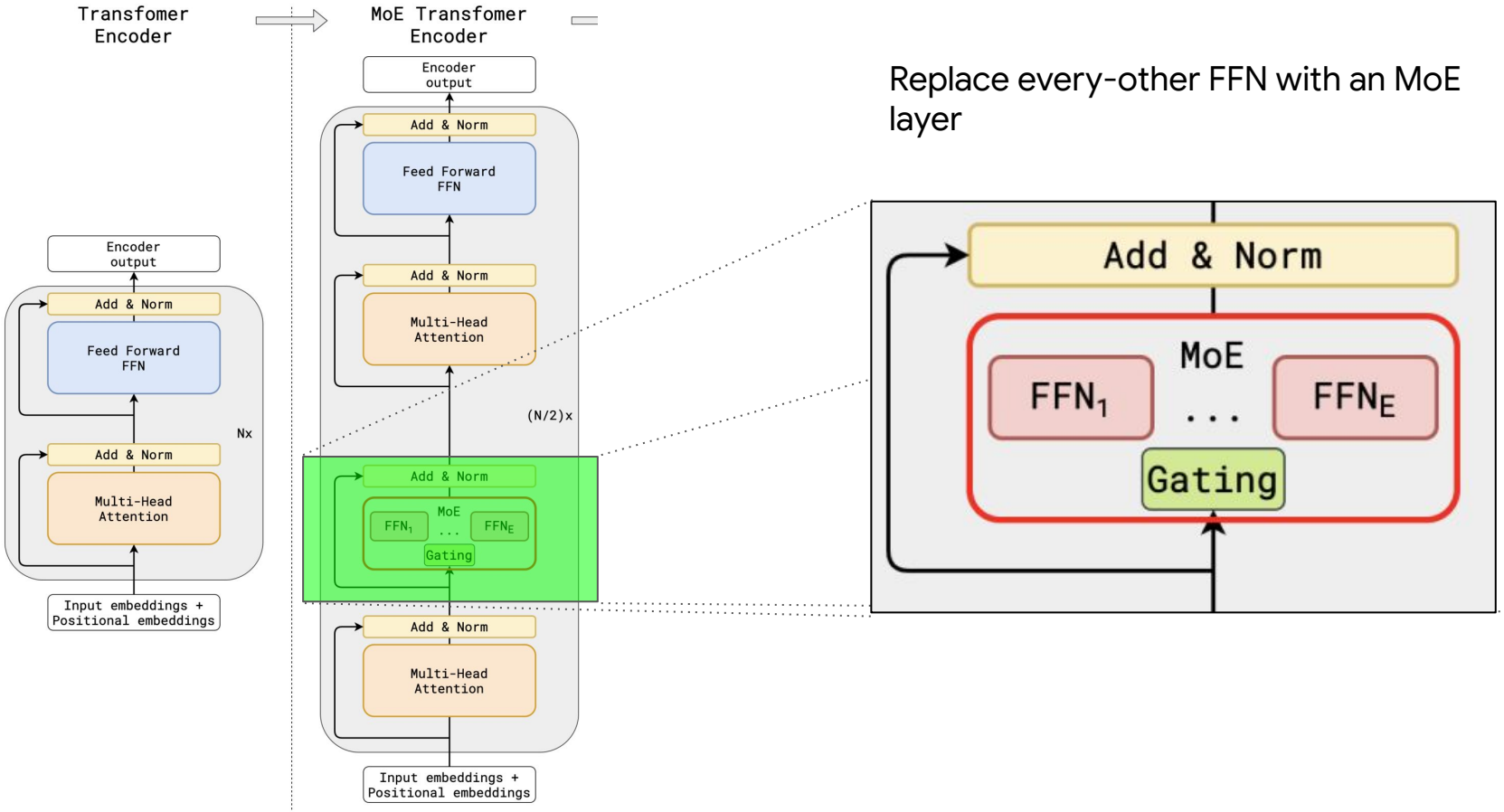
Super-linear scaling of computation cost vs model size Straightforward scaling of the mode size by increasing the depth or width [6, 15] generally results in at least linear increase of training step

Sparse (Conditional computation and MoE):

Sub-linear Scaling First, model architecture should be designed to keep the computation and communication requirements sublinear in the model capacity. Conditional computation [25, 16, 26, 27] enables us to satisfy training and inference efficiency by having a sub-network activated on the per-input basis. Scaling capacity of RNN-based machine translation and language models by adding Position-wise Sparsely Gated Mixture-of-Experts (MoE) layers [16] allowed to achieve state-of-the-art results with sublinear computation cost. We therefore present our approach to extend Transformer architecture with MoE layers in Section 2.

Each training example consists of a pair of sequences of subword tokens. Each token activates a sub-network of the MoE Transformer during both training and inference. The size of the sub-network is roughly independent of the number of experts per MoE Layer, allowing sublinear scaling of the computation cost as described in the previous section. Computation complexity is further analyzed in Section 3.1 and training performance in Section 5.

Mixture-of-Experts (MoE) Transformer

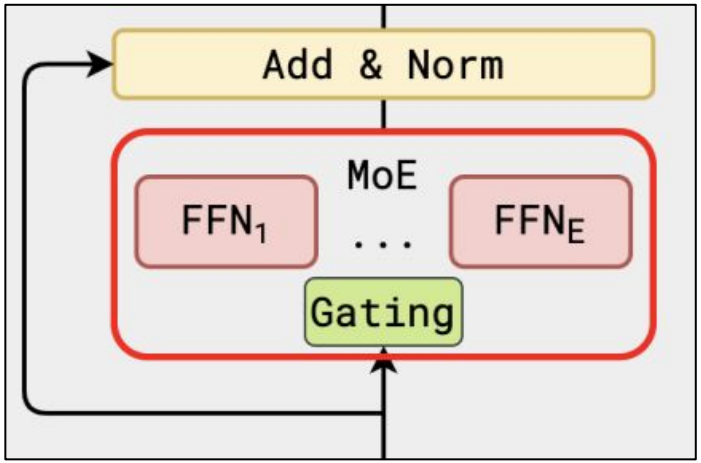


Position-wise Mixture-of-Experts Layer

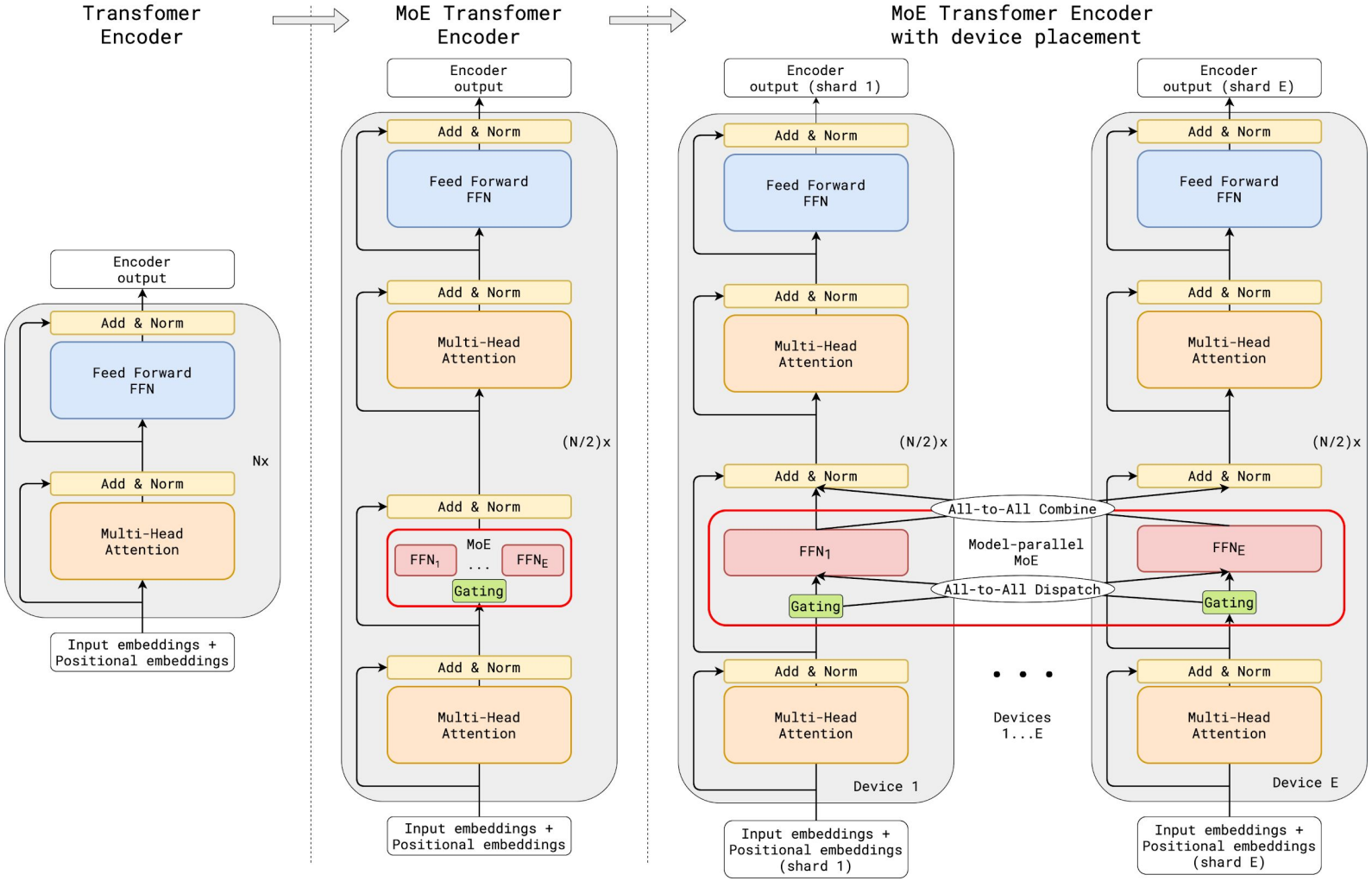
x_s is the input token

$$\mathcal{G}_{s,E} = \text{GATE}(x_s)$$
$$\text{FFN}_e(x_s) = wo_e \cdot \text{ReLU}(wi_e \cdot x_s)$$
$$y_s = \sum_{e=1}^{E} \mathcal{G}_{s,e} \cdot \text{FFN}_e(x_s)$$

E feed-forward networks $\text{FFN}_1 \ldots \text{FFN}_E$



$\mathcal{G}_{s,E}$ is computed by a gating network.
y_s, is the weighted average


Transfomer Encoder
MoE Transfomer Encoder
MoE Transfomer Encoder with device placement
Encoder output
Add & Norm
Feed Forward FFN
Multi-Head Attention
Nx
Input embeddings + Positional embeddings
(N/2)x
MoE
FFN1
FFNE
Gating
Encoder output (shard 1)
Encoder output (shard E)
All-to-All Combine
All-to-All Dispatch
Model-parallel MoE
Devices 1...E
Device 1
Device E
Input embeddings + Positional embeddings (shard 1)
Input embeddings + Positional embeddings (shard E)

Critical Components of the MoE Transformer

Expert Capacity

↦ Ensure the load is balanced
↦ Keep #tokens per-expert low

DETAILS

- Total #tokens: *N*
- Each token dispatched to max 2 experts
- Expert capacity *O(N/E)*
- Keep a counter c_e and overflow if capacity is reached

⟶ overflown token skips layer

Local Group Dispatch

↦ Ensure even batch partitioning
↦ Groups processed in parallel

DETAILS

- GATE partitions into *G* groups
- Each group

⟶ contains *S=N/G* tokens
⟶ uses fractional capacity of each expert, *2N/(GxE)*

↦ Ensures the load is balanced

Auxiliary Loss

↦ Ensure expert selection diversity
↦ No winner takes all & overflow

DETAILS

- $\mathcal{L} = \ell_{nll} + k * \ell_{aux}$
- where

$$\ell_{aux} = \frac{1}{E} \sum_{e=1}^{E} \frac{c_e}{S} \cdot m_e$$

- m_e mean of the gates per expert

↦ Differentiable approx. of top-2 op.