Google AI

# From Honey Bee to Mouse Brain

## Scaling neural networks beyond 80 billion parameters


Orhan Firat, PhD
Research Scientist
Mountain View - CA
orhanf@google.com


Oct 28, 2019

Stats 285 - Stanford

# Who am I?

- Research Scientist at Google Research
    - Natural Language Understanding/Machine Translation

- PhD Thesis
    - Connectionist Multi-Sequence Modelling
    Supervisors: Dr. Fatos Yarman Vural - Middle East Technical University
    Dr. Kyunghyun Cho - New York University

- Research Interests
    - Sequence to sequence models: NMT
    - Multilingual Models for NLP
    - Multi-task, Continual learning, Meta-learning
    - Trainability of Neural Networks
    - Used to do some computational neuroscience

Google AI

Google

how many neurons in the brain

All | Images | Videos | News | Maps | More | Settings | Tools

About 39,600,000 results (0.58 seconds)

# 100 billion neurons

Neurons in the Human Brain

According to many estimates, the human brain contains around **100 billion neurons** (give or take a few billion). Jun 11, 2019

How Many Neurons Are in the Brain? - Verywell

https://www.verywellmind.com/how-many-neurons-are-in-the-b

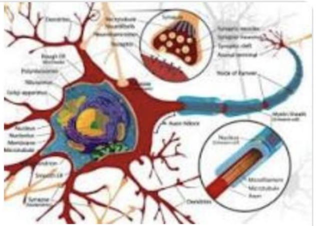
Google

how many connections in the brain

All | Images | Videos | News | Shopping | More | Settings | Tools

About 160,000,000 results (0.54 seconds)

Each individual neuron can form thousands of links with other neurons in this way, giving a typical **brain** well over 100 trillion synapses (up to 1,000 trillion, by some estimates). Functionally related neurons connect to each other to form neural networks (also known as neural nets or assemblies).



Neurons & Synapses - Memory & the Brain - The Human Memory

www.human-memory.net/brain_neurons.html

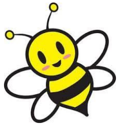
# Number of Synapses

#synapses [wiki]

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

**Fruit fly**

**Honey bee**

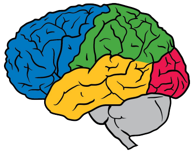


**Mouse**

**Cat**

**Macaque**

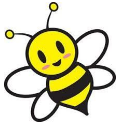
**Human**

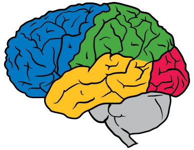


# Number of Synapses

**NMT with Attention**
**Resnet50**
**[25-50M]**

| $10^6>$ | $10^9$ | $10^{12}$ | $10^{13}$ | $10^{14}$ | $10^{15}$ | #synapses [wiki] |
|---|---|---|---|---|---|---|
| **Fruit fly** | **Honey bee** | **Mouse** | **Cat** | **Macaque** | **Human** | |

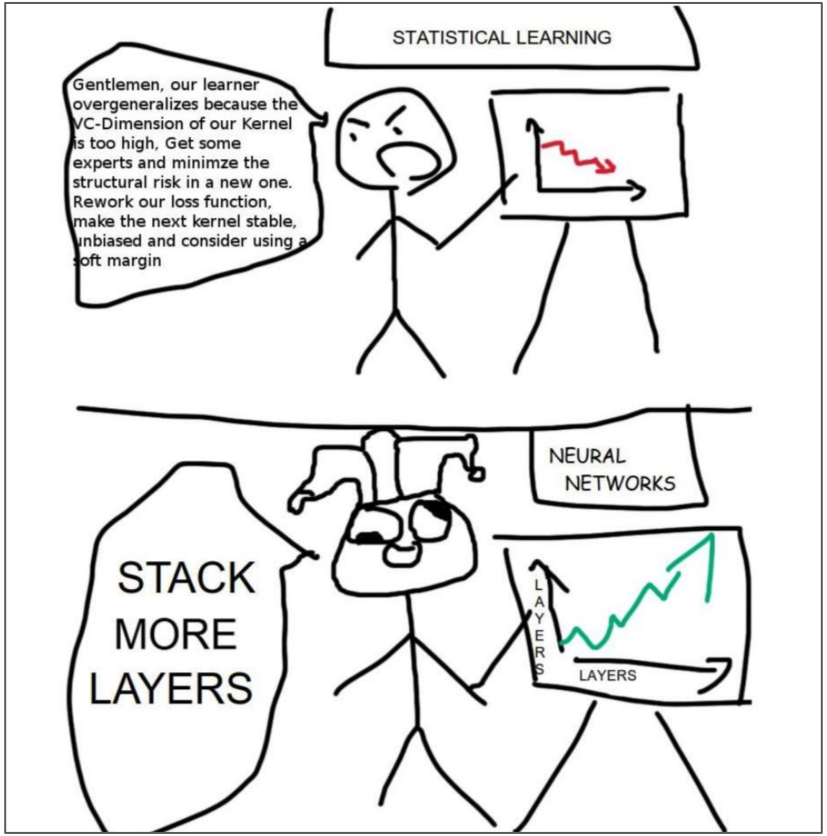




# Unstoppable Force

# Unstoppable Force vs Immovable Object

# Scaling Up Neural Networks

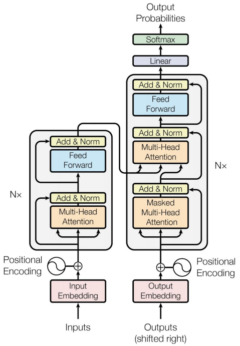





[Machine Learning Memes for Convolutional Teens]

Google AI

# Number of Synapses



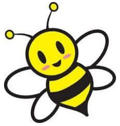

**Transformer**
**[400M]**

#synapses [wiki]

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

**Fruit fly**

**Honey bee**

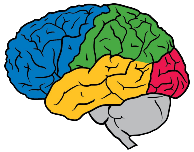


**Mouse**

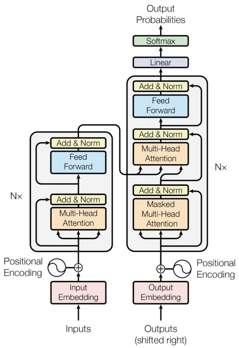
**Cat**

**Macaque**

**Human**



# Number of Synapses




**Transformer**
**[400M]**

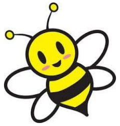
**Facebook ResNeXt101**
**Open AI GPT2**
**MSR ZeRO**
**NVidia Megatron-LM**
**[1-8B]**

#synapses
[wiki]

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

**Fruit fly**

**Honey bee**

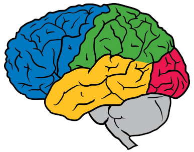


**Mouse**

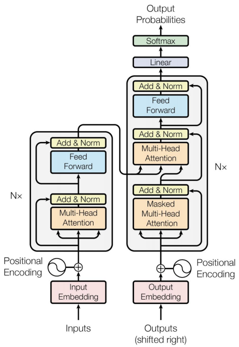
**Cat**

**Macaque**

**Human**



# Number of Synapses




Transformer
[400M]

Facebook ResNeXt101
Open AI GPT2
MSR ZeRO
NVidia Megatron-LM
[1-8B]

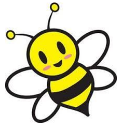
M4:
Massively Multilingual Massive Machine Translation
[80B]

#synapses
[wiki]

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

Fruit fly

Honey bee

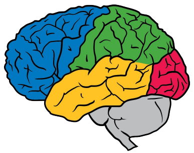


Mouse

Cat

Macaque

Human



Google AI

# Number of Synapses

Google AI Blog

The latest news from Google AI

## Exploring Massively Multilingual, Massive Neural Machine Translation

Friday, October 11, 2019

Posted by Ankur Bapna, Software Engineer and Orhan Firat, Research Scientist, Google Research

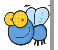
*"... perhaps the way [of translation] is to descend, from each language, down to the common base of human communication — the real but as yet undiscovered universal language — and then re-emerge by whatever particular route is convenient."* — Warren Weaver, 1949

Massive

ion

$10^6$>

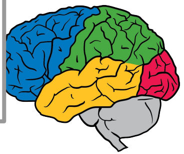
Fruit



#synapses [wiki]

$10^{15}$

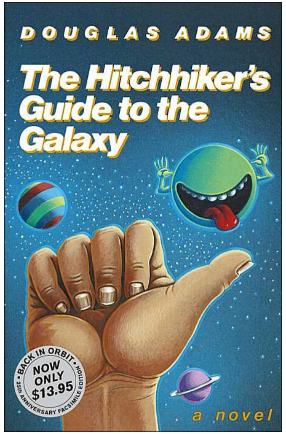
Human

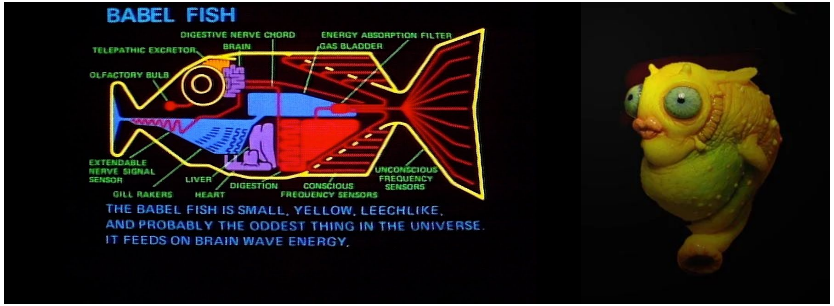


# Towards Universal Translation

- **Number of Languages: 103 languages**
  → Single Neural Network Capable of Translating between any two.
  → Massively Multilingual, Multi-task Setup
  → Open domain dataset, crawled from Web **25 billion examples**

- **Number of Neurons: Model Size**
  → Orders of magnitude larger than traditional translation models (Transformer 400M)
  → Massive in terms of number of trainable parameters (**6B - 90B parameters**)
  → Very deep and wide (**1024 layers deep, 16k wide**)



# Outline

- **Prototyping** - *Engineering perspective*
- **Debugging** - *ML perspective*
- **Hyper-parameter Search** - ML perspective
- **Reproducibility** - *Engineering perspective*
- Bonus: Coordination

Google AI

# Machine Translation

in a nutshell

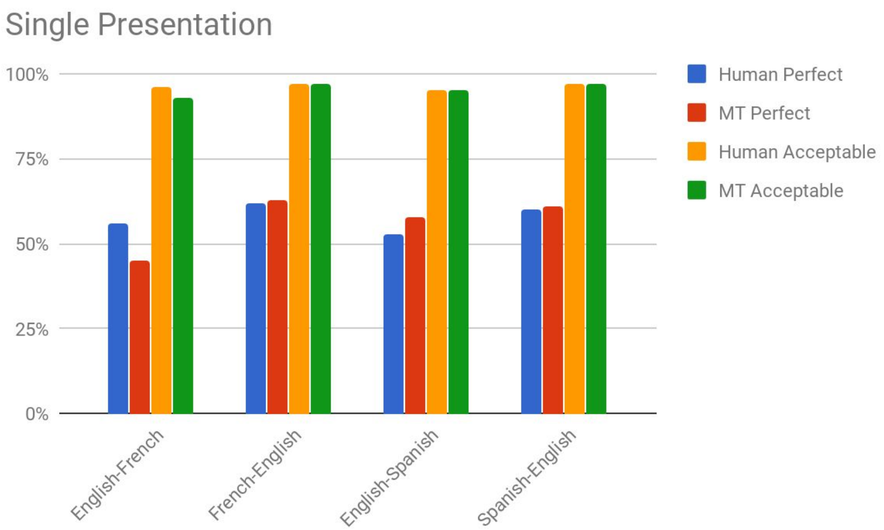
Google Translate

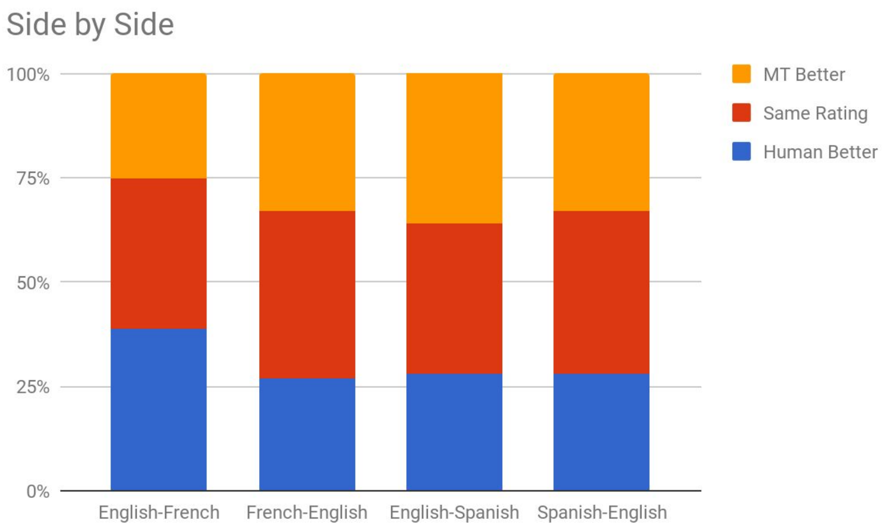
# 30 trillion

sentences translated per year

Google AI

# Sentence-level **quality** is improving



# What is behind?

## The Paradigm:
Sequence to Sequence Learning

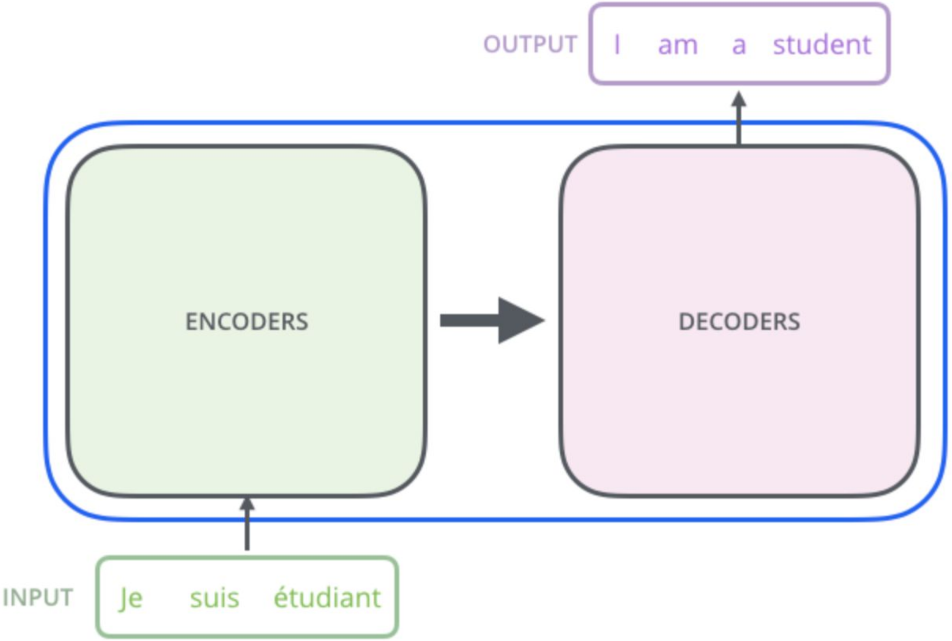
## The Task:
Neural Machine Translation

# Sequence to Sequence Refresher

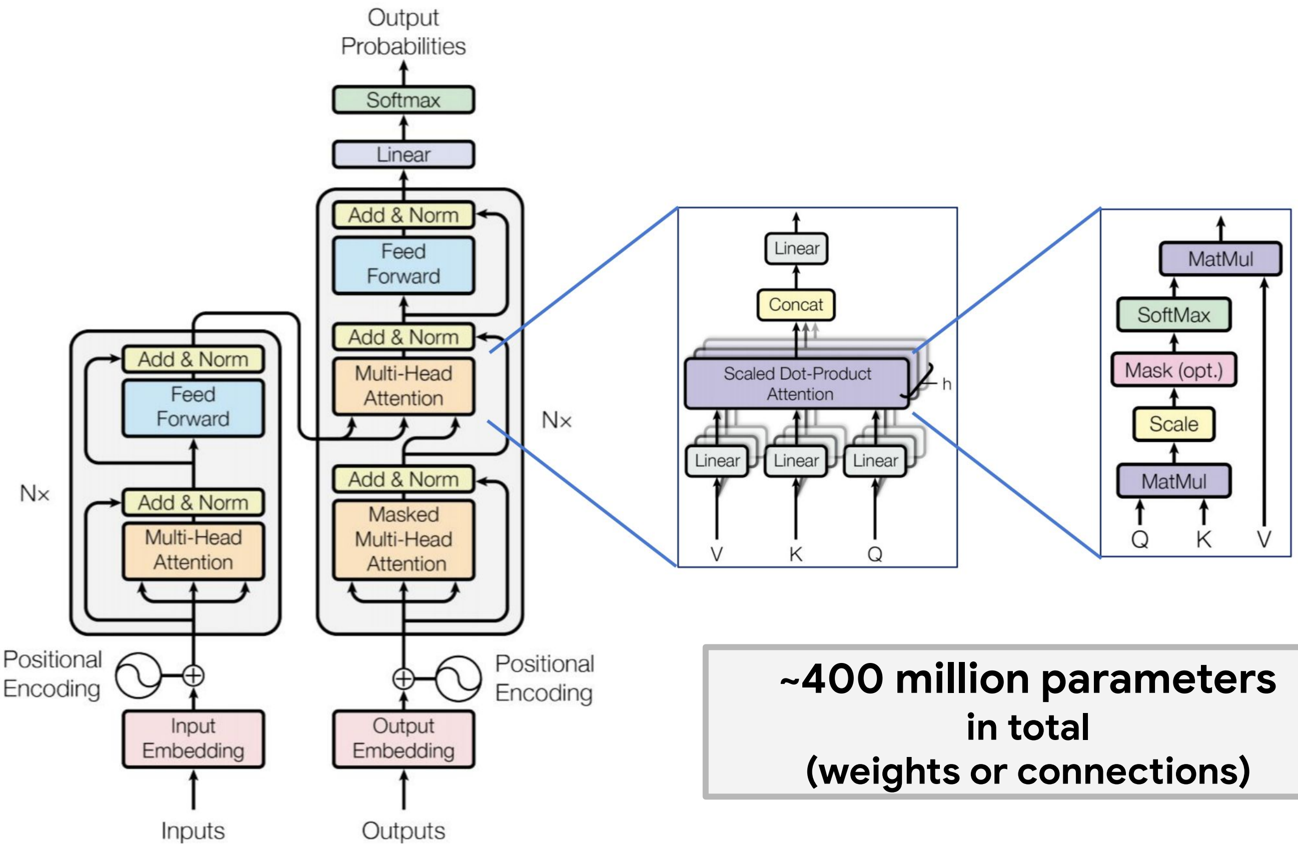

Figure Credit: Jan Chorowski

Google AI

# Neural Machine Translation

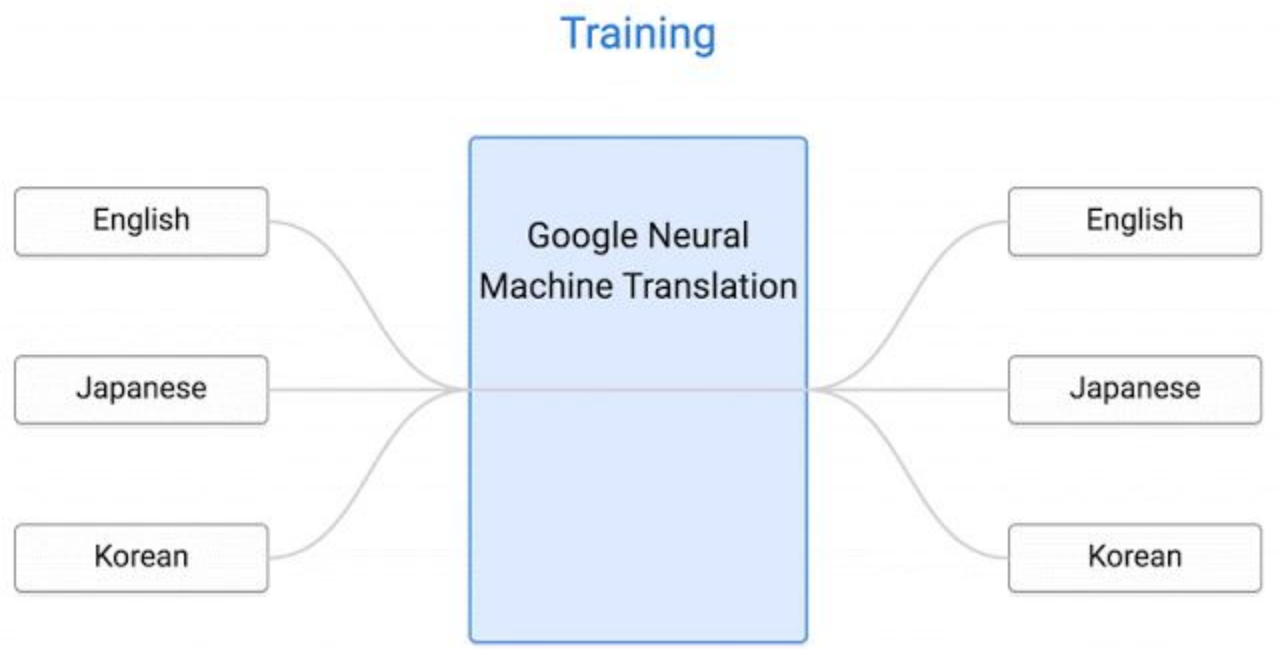


# Transformer: Attention is All You Need - Vaswani et al. 2017




**~400 million parameters**
**in total**
**(weights or connections)**

# Multilingual Neural Machine Translation



# Our goal

**Develop a universal translation model (i.e,. a single model across 1000+ languages)**

*"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."*

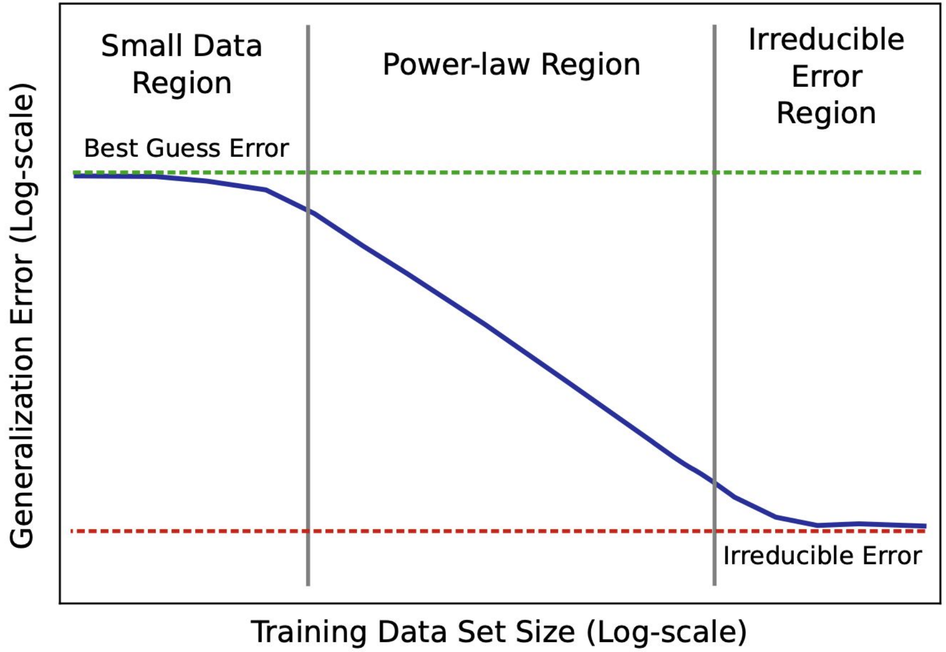
Warren Weaver (1949)

# 1. Massively Multilingual MT

## A/ Data

Massively Multilingual Neural Machine Translation, Aharoni et al. NAACL 2019
Massively Multilingual NMT in the Wild: Findings and Challenges, Arivazhagan et al. 2019

# A/Data




**Sketch of Power Law Learning Curves** [Hestness et al. 2017]

# A/Data

- Any Sequence
- Arbitrary length

- Sequence-to-Sequence
  - **Machine Translation**
  - **Sentiment Analysis**
  - **Speech Recognition**
  - **Image Captioning**

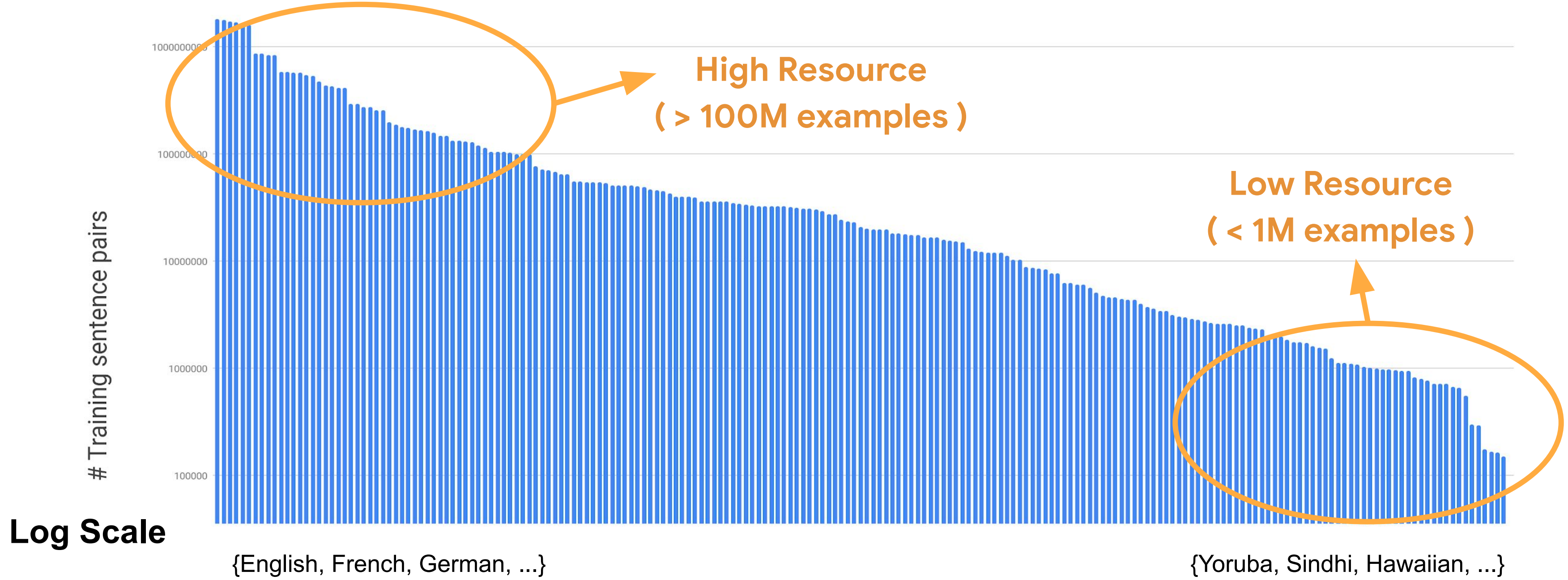
## Now at Massive Amounts

- BERT (Devlin et al. 2018): Wikipedia
- GPT-2 (Radford et al. 2019) : Reddit
- M4 (Arivazhagan et al. 2019): Entire Internet

## Convert Compute into Data

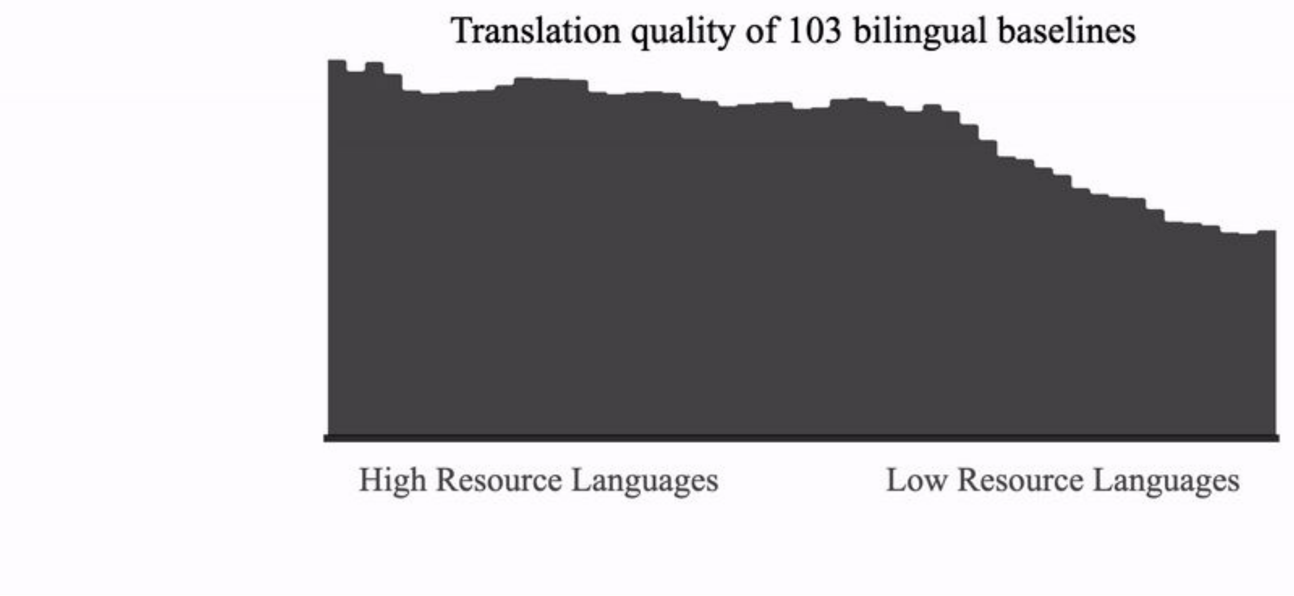
- AlphaZero, OpenAI Five: Self-Play
- AlphaStar: Multi-agent

# A/Data


1000000000
100000000
10000000
1000000
100000
# Training sentence pairs
High Resource
( > 100M examples )
Low Resource
( < 1M examples )
Log Scale
{English, French, German, ...}
{Yoruba, Sindhi, Hawaiian, ...}

# Massively Multilingual MT - Arivazhagan et al. 2019

## Large Gains on Low-Resource Languages with Multilinguality



# 2. Massive Neural Networks

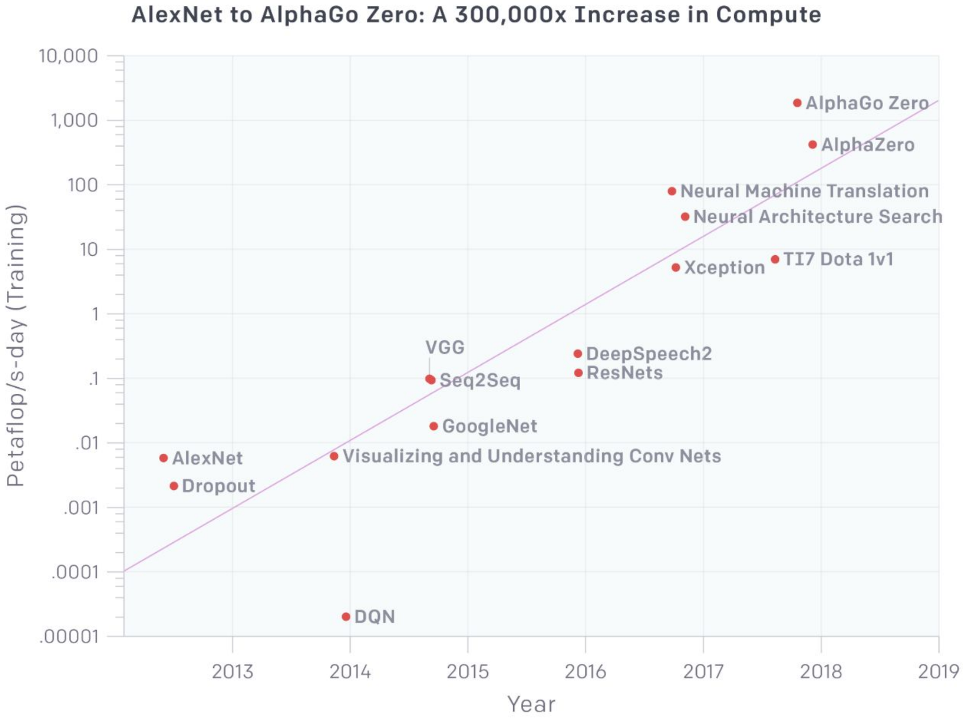
## A/ Compute

## B/ Models

Training Deeper Neural Machine Translation Models with Transparent Attention, Bapna et al. EMNLP 2018
GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism, Huang et al. NeurIPS 2019
Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer, Shazeer et al. ICLR 2017

Google AI

# A/Compute

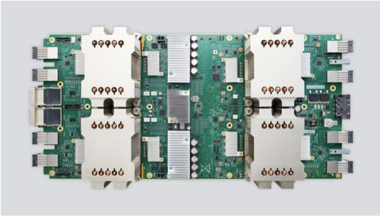



**AI and Compute** - Damodei and Hernandez 2018

# A/Compute

- Training on 1024 TPU-v3 chips
- Bfloat16 (Brain Floating Point)
- GPipe: Micro-Batch Pipeline Parallelism (Huang et al., 2019)
  - Rematerialization (gradient checkpointing)
  - Large batches (4M examples)

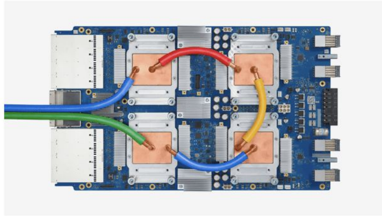
# A/Compute - Tensor Processing Units



**Cloud TPU v2**

180 teraflops

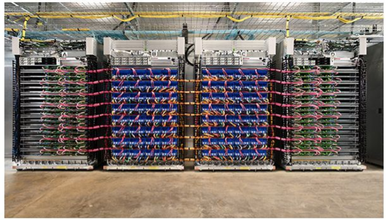
64 GB High Bandwidth Memory (HBM)



**Cloud TPU v3**

420 teraflops

128 GB HBM



**Cloud TPU v2 Pod (beta)**

11.5 petaflops

4 TB HBM

2-D toroidal mesh network

**Cloud TPU v3 Pod (beta)**

100+ petaflops

32 TB HBM

2-D toroidal mesh network

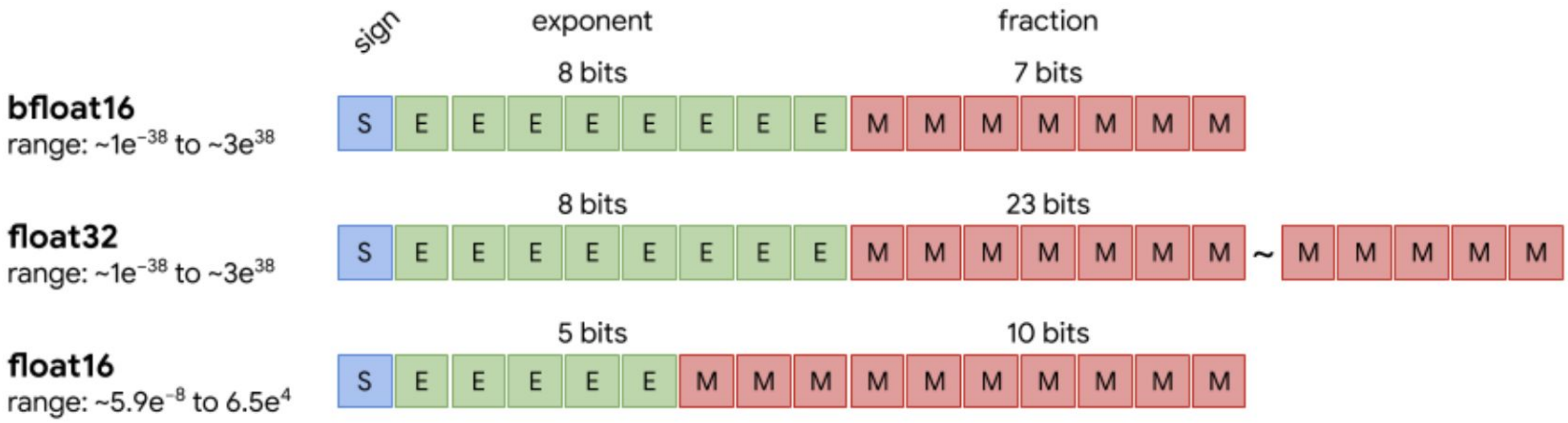


https://cloud.google.com/tpu/

# A/Compute - BFloat16

Google AI

- fp32 - IEEE single-precision floating-point
- fp16 - IEEE half-precision floating point
- bfloat16 - 16-bit *brain floating point*




The dynamic range of bfloat16 is greater than that of fp16.

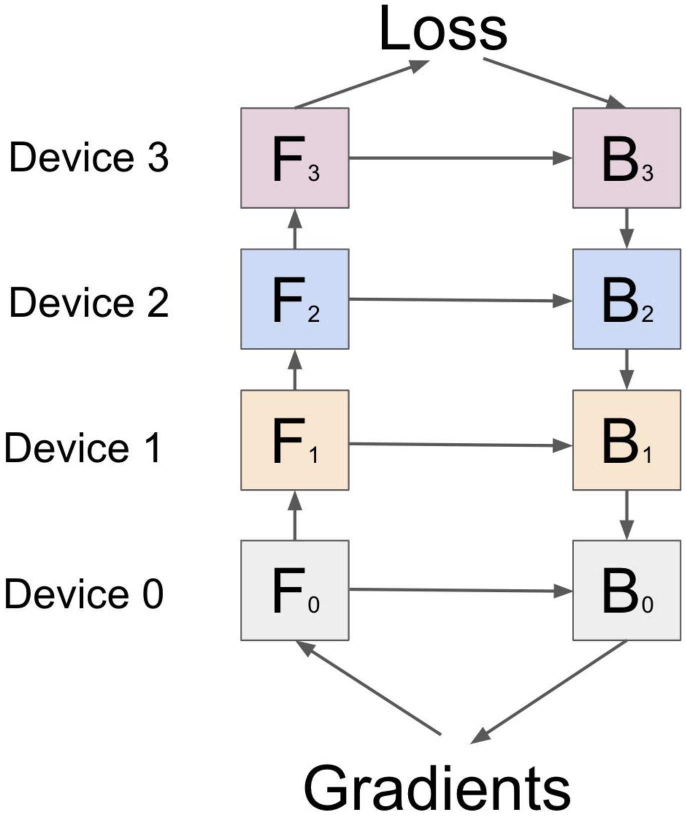


https://cloud.google.com/tpu/docs/bfloat16

Google AI

# A/Compute - Data and Model Parallelism

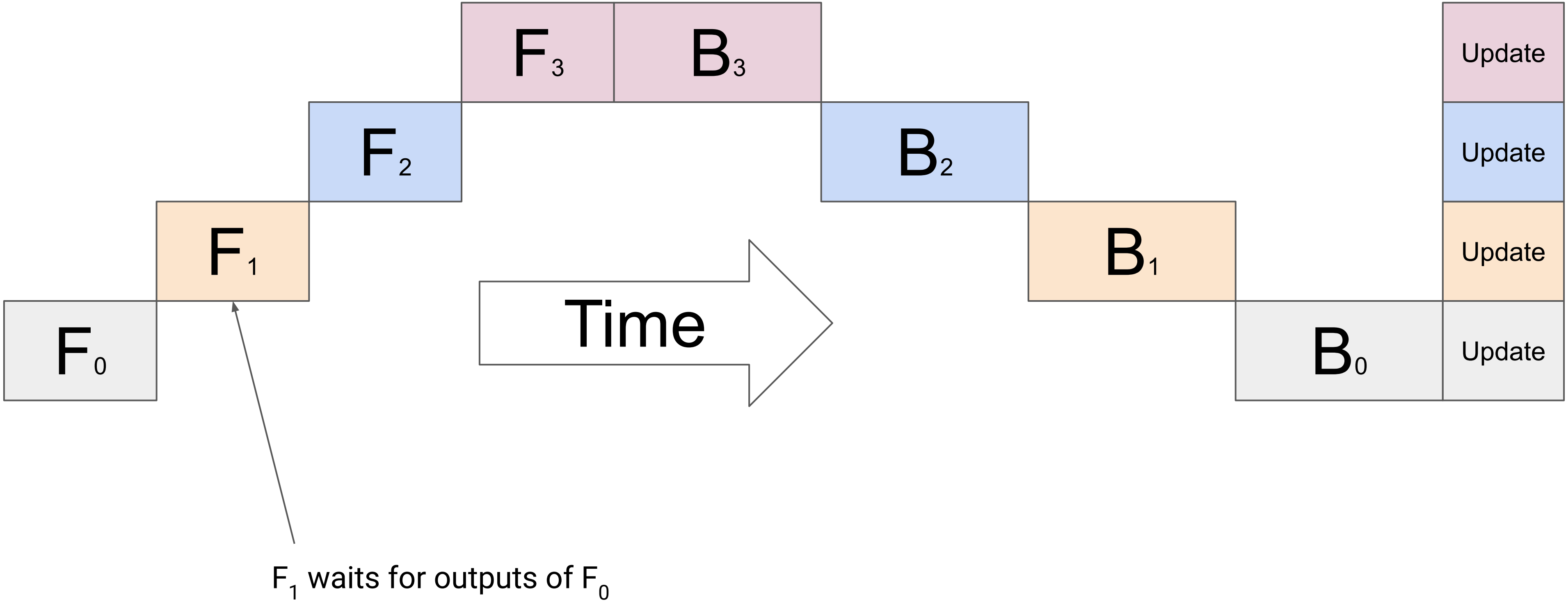
- Training on 1024 TPU-v3 chips
- Bfloat16 (Brain Floating Point)
- GPipe: Micro-Batch Pipeline Parallelism (Huang et al., 2019)
  - Rematerialization (gradient checkpointing)
  - Large batches (4M examples)





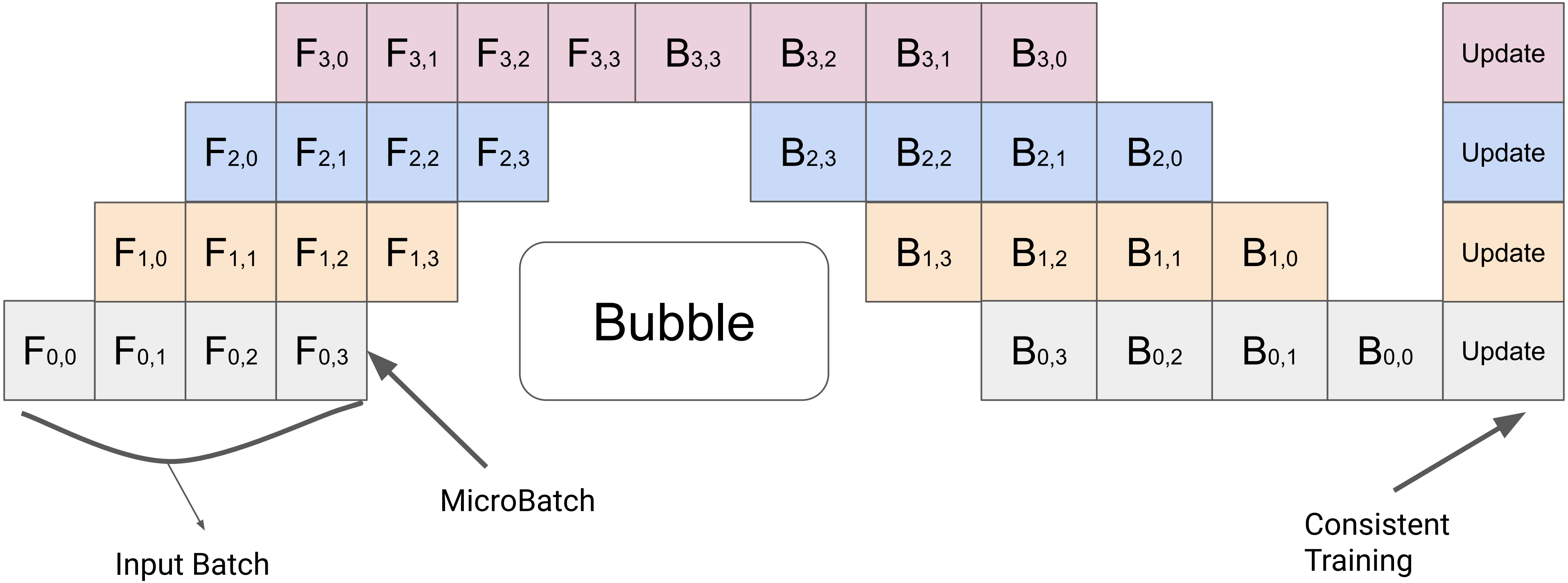

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism, Huang et al. NeurIPS 2019

Only one accelerator is active when the model is distributed across the accelerators






[GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism, Huang et al. NeurIPS 2019]






[GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism, Huang et al. NeurIPS 2019]

# More resources to ease handling large networks, painlessly (?)

- Fitting larger networks into memory
- Reducing the Memory Usage
- Training Neural Nets on Larger Batches: Practical Tips for 1-GPU, Multi-GPU & Distributed setups

# B/Models

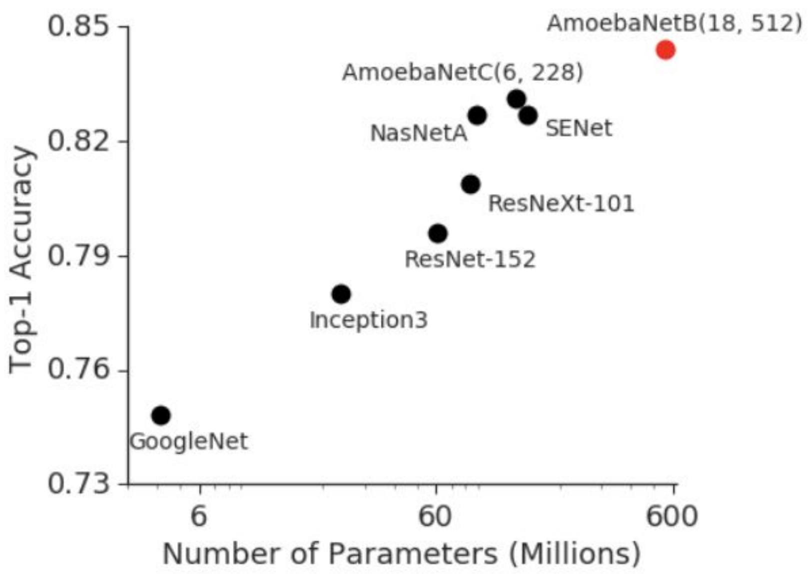
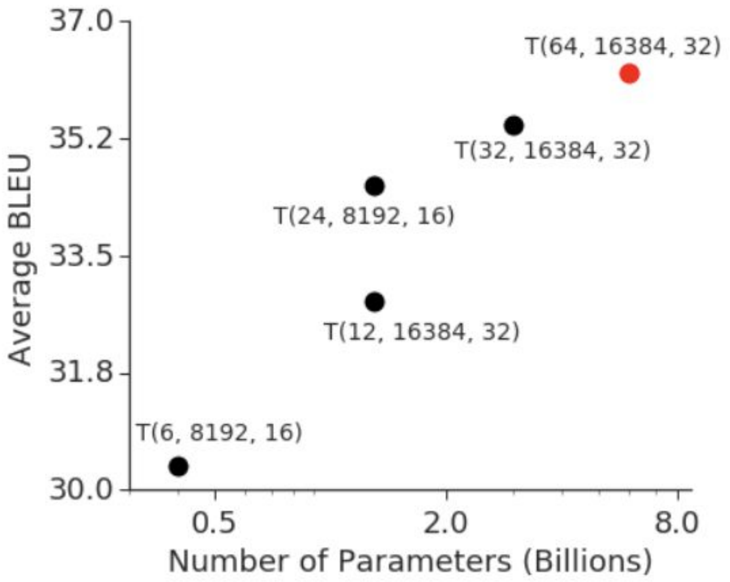


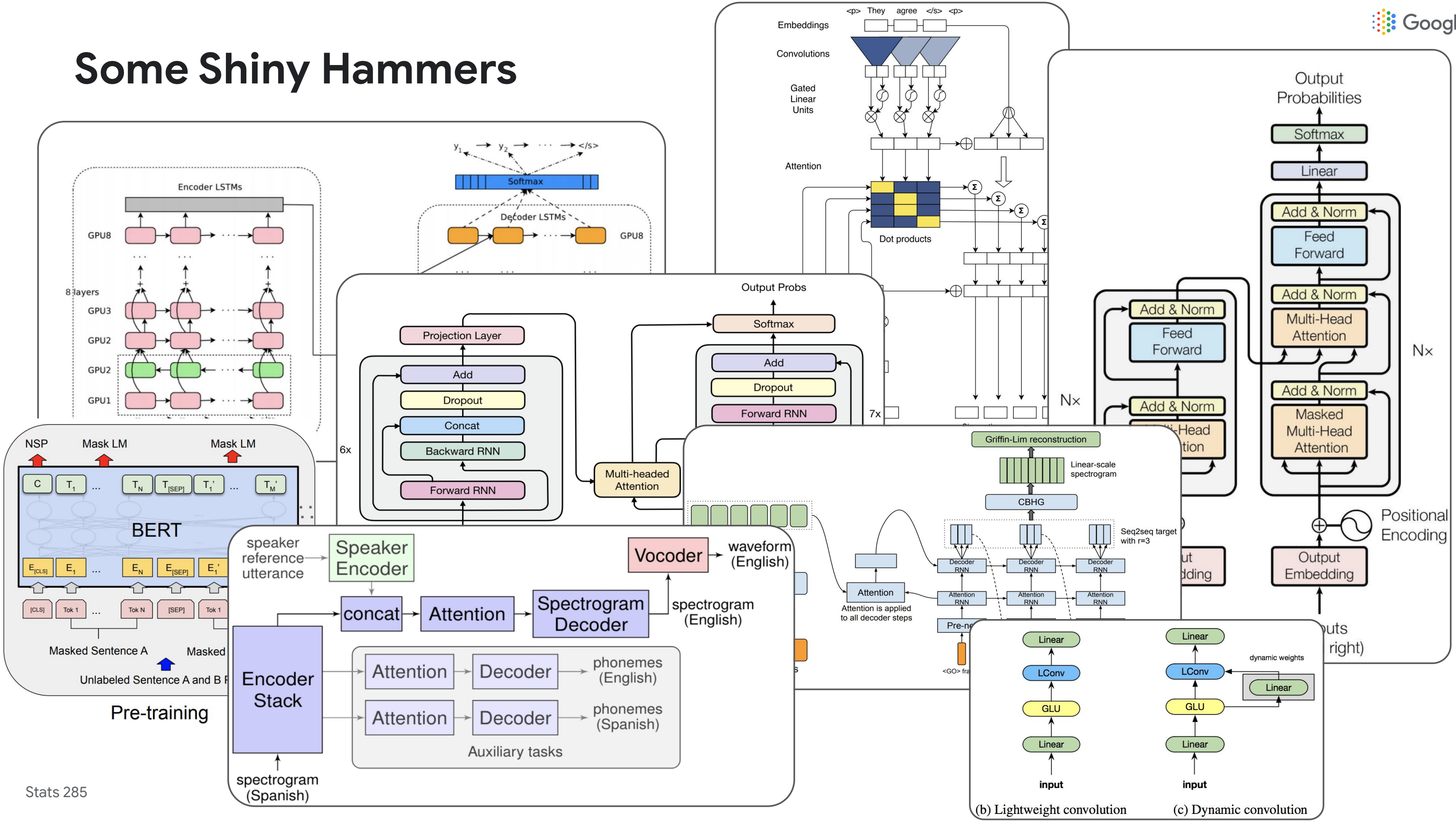

**Image-Net**




**Machine Translation**

**GPipe: Easy Scaling with Pipeline Parallelism** - Huang et al., 2019

# Some Shiny Hammers



## Aiding the Model

Model enhancements to ease the training:

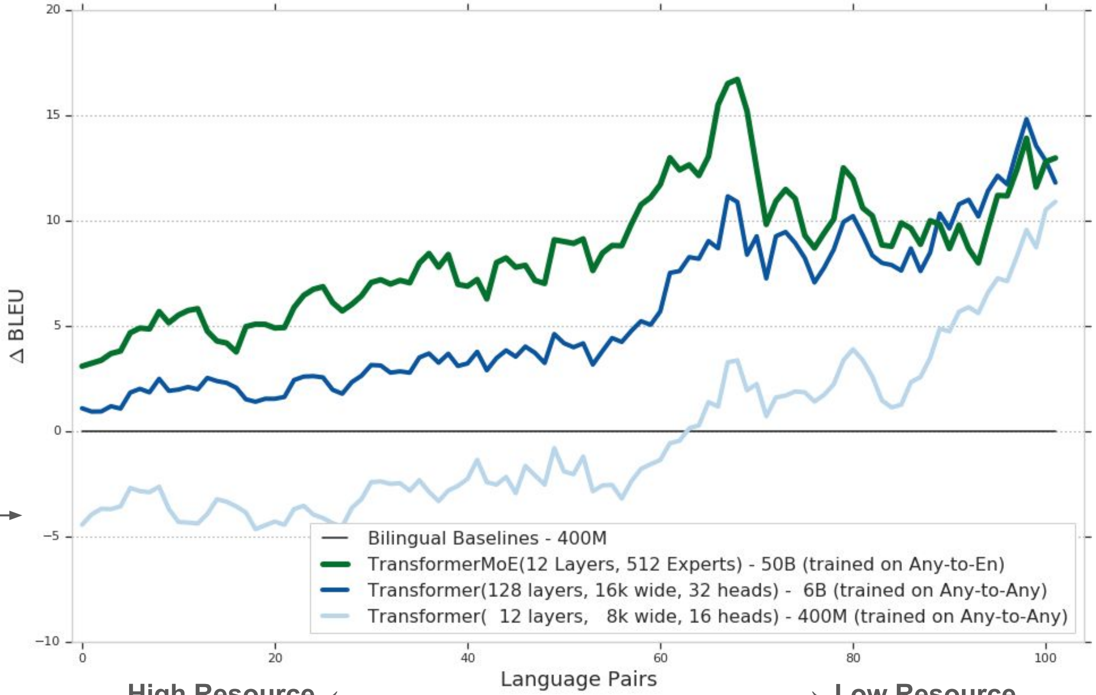
- Residuals
- Normalizations (layer, batch, spectral)
- Transparent Attention (Bapna et al. 2018)
- Parameter Sharing
  (Press and Wolf 2016, Jean et al. 2018, Dehghani et al. 2018)
- Sparsely Gated Mixture of Experts
  (Shazeer et al. 2017)

## Aiding the Optimizer

Step rule enhancements to ease the training:

- Sync-training
- Grad-norm tracker (Chen et al. 2018)
- Large batches
  (Goyal et al. 2017, Ott et al. 2018)
- Learning Rate Schedules (Bengio 2012)
- New step rules
  (Shazeer and Stern 2018, Gupta et al. 2018)
- Logit clipping
- Smart Initializations, Fixup
  (Zhang et al. 2019)

# Putting it all together: M4

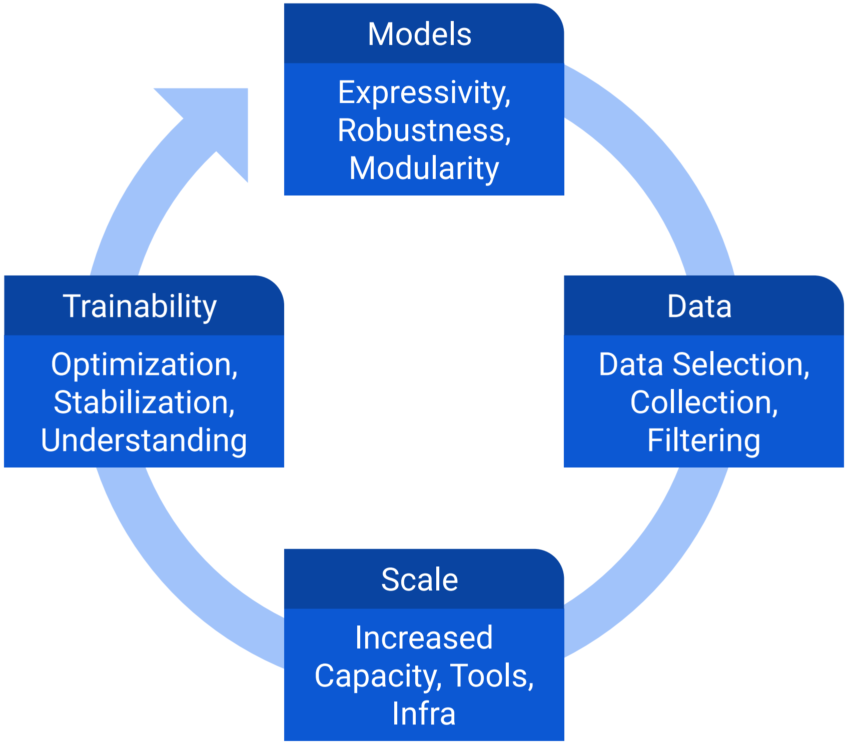



Bilingual baselines →

**High Resource ←** **→ Low Resource**

# Workflow

# Workflow of AI Researchers working at Scale



# Outline

- **Prototyping**
- Debugging
- Hyper-parameter Search
- Reproducibility
- Bonus: Coordination

# Components of an NMT System

**Training**

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

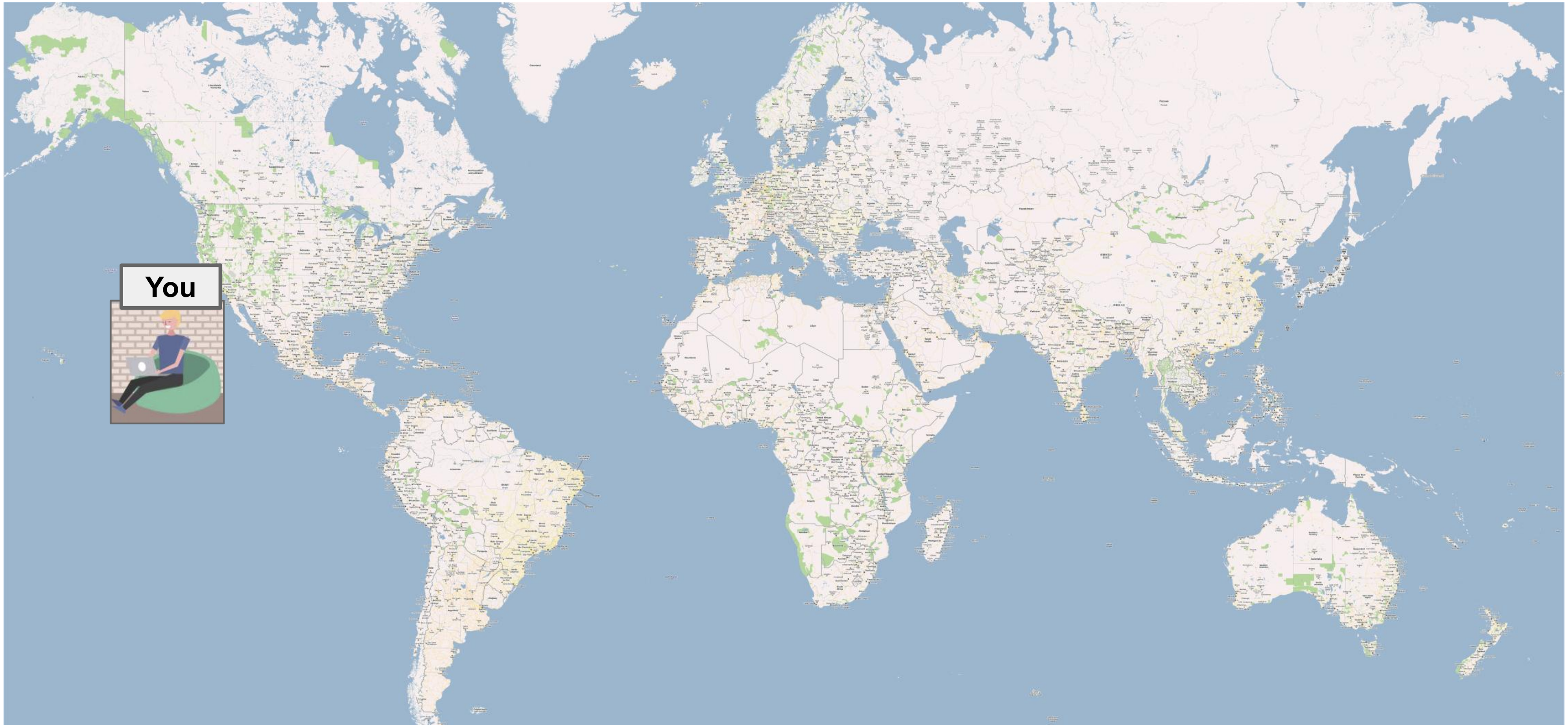
**Inference**

- 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

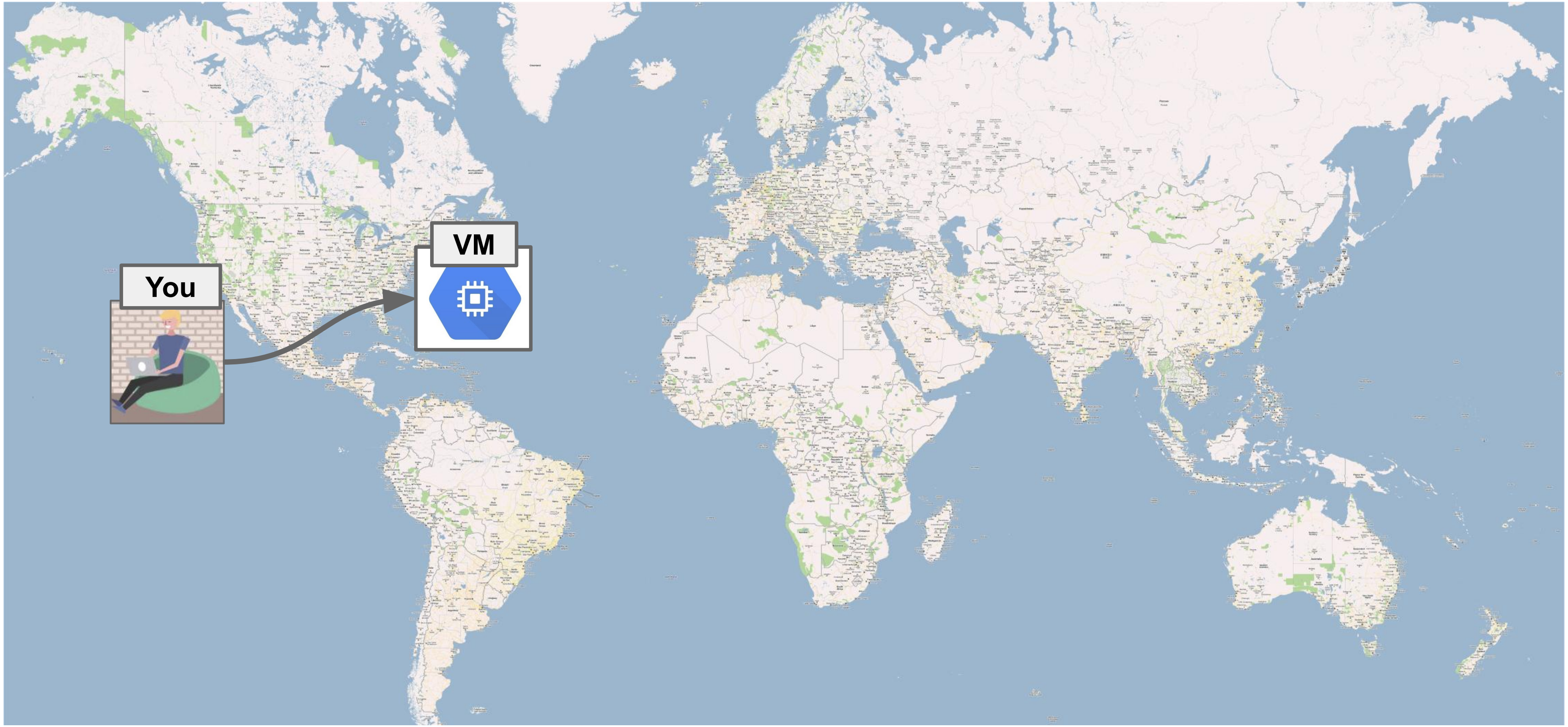





*This is just a sketch, exact locations are inaccurate.

# Under the hood

Google AI

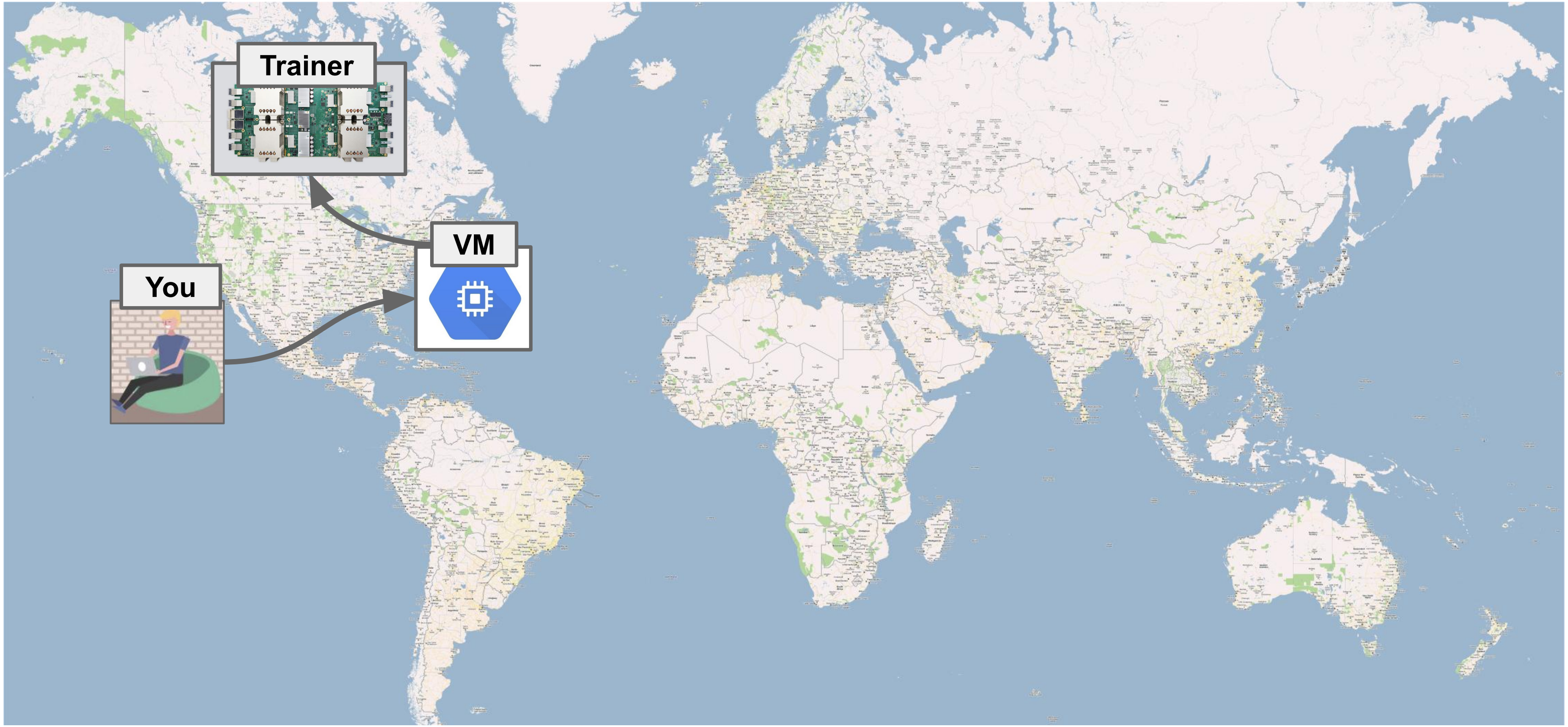





*This is just a sketch, exact locations are inaccurate.

# Under the hood

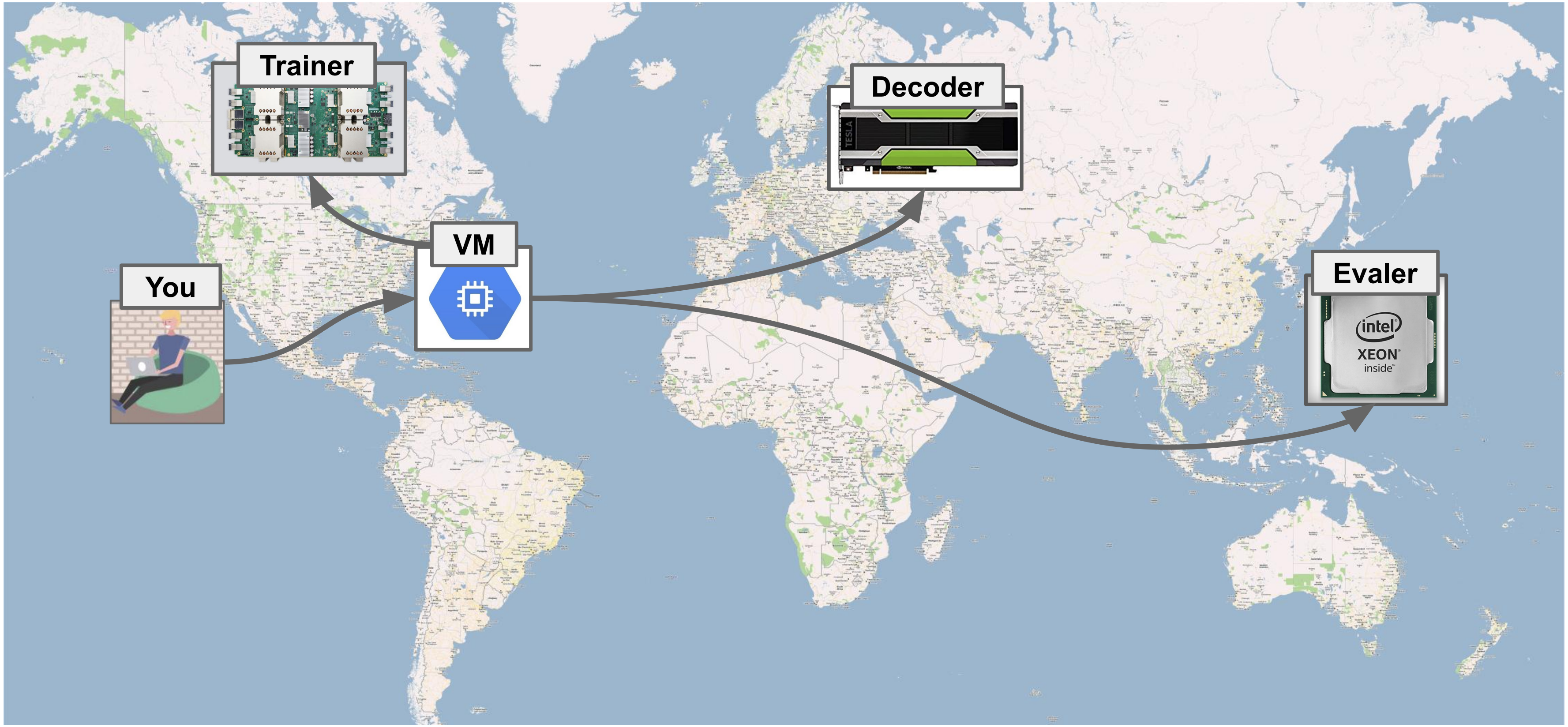
Google AI




Stats 285

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

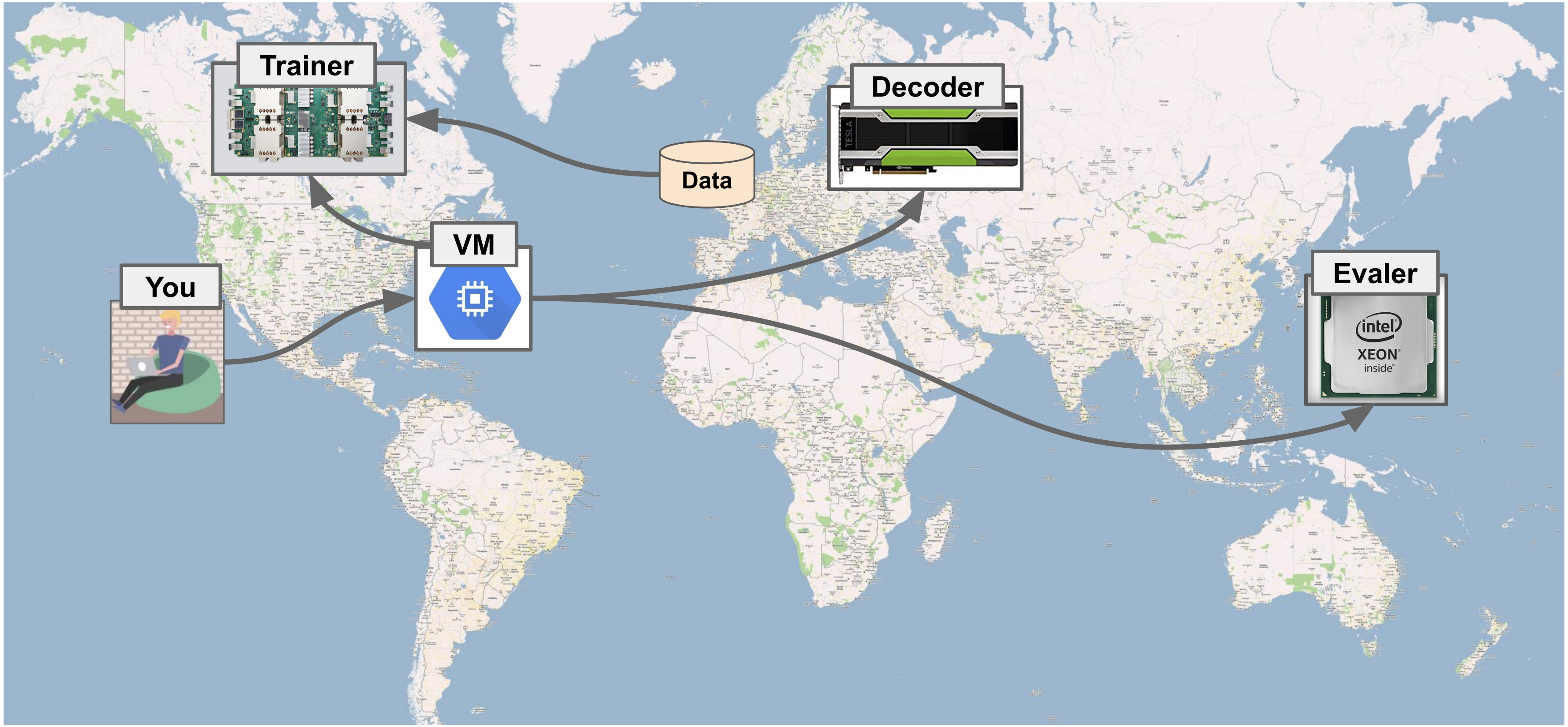
# Under the hood






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

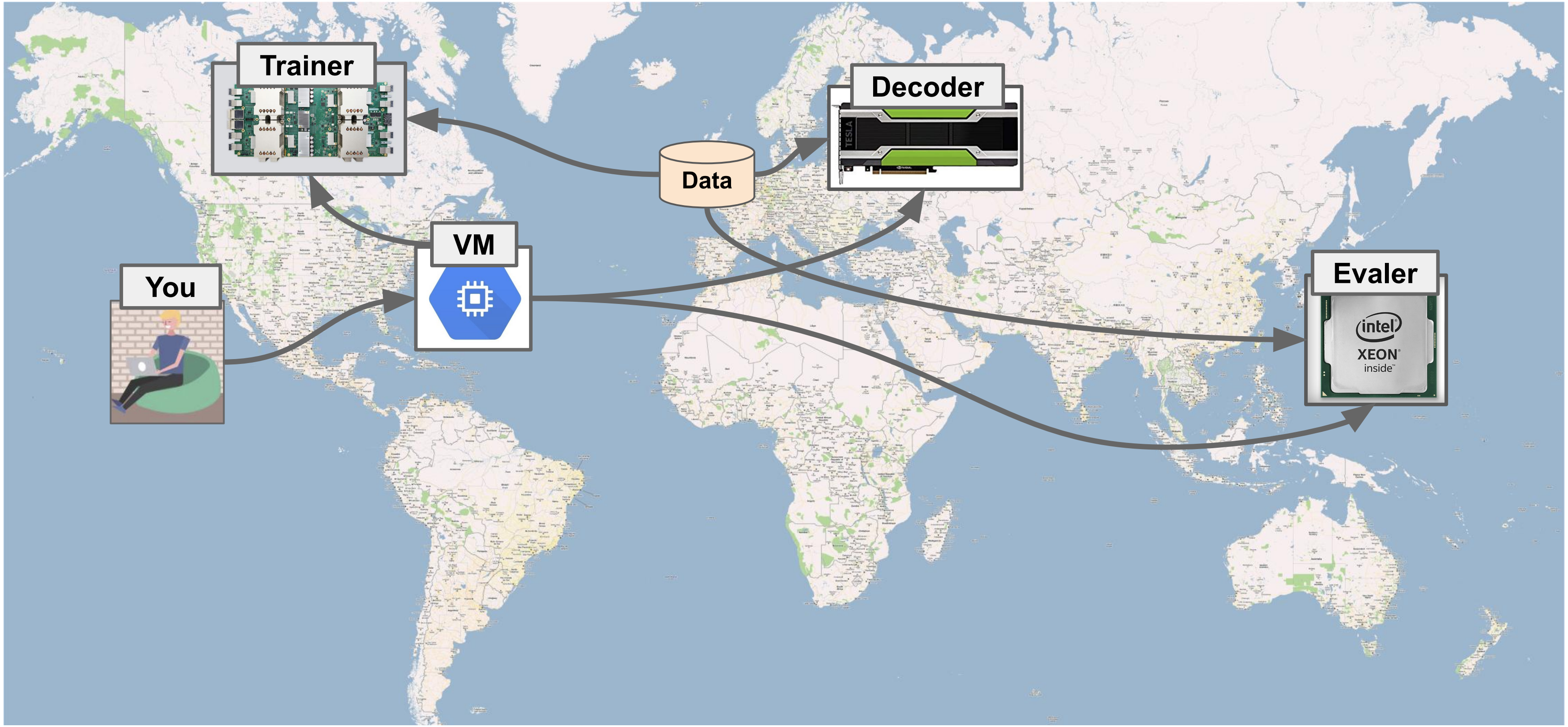
# Under the hood






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

# Under the hood

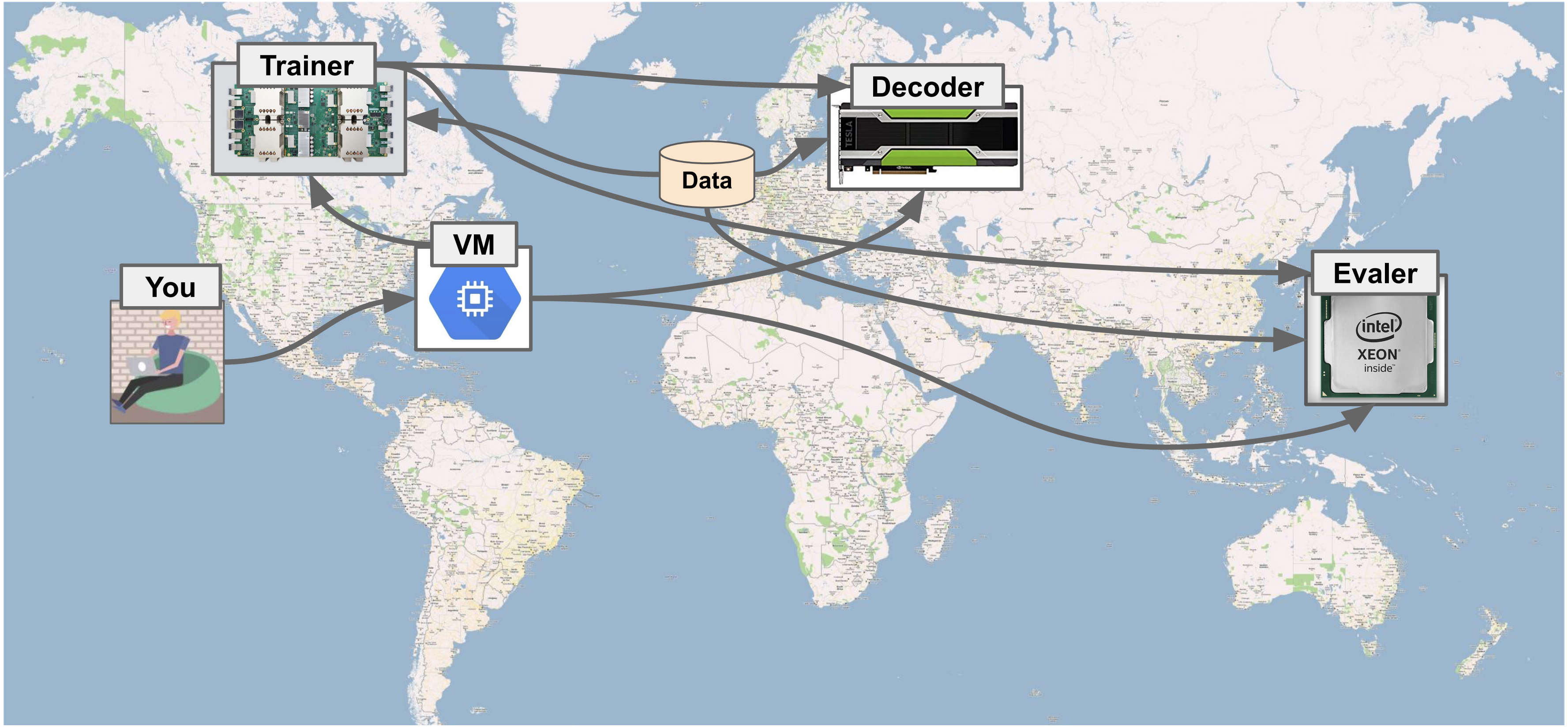





*This is just a sketch, exact locations are inaccurate.

# Under the hood

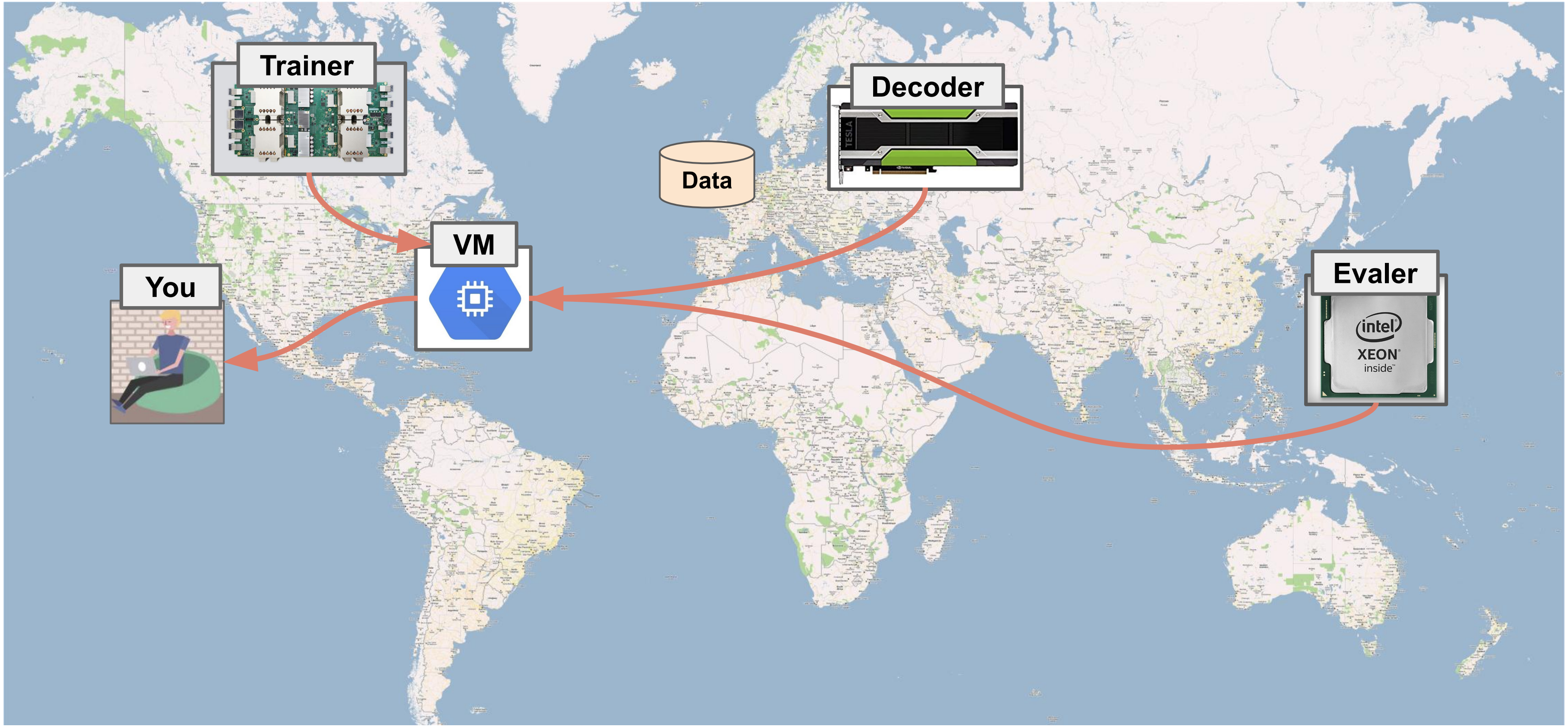
Google AI






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

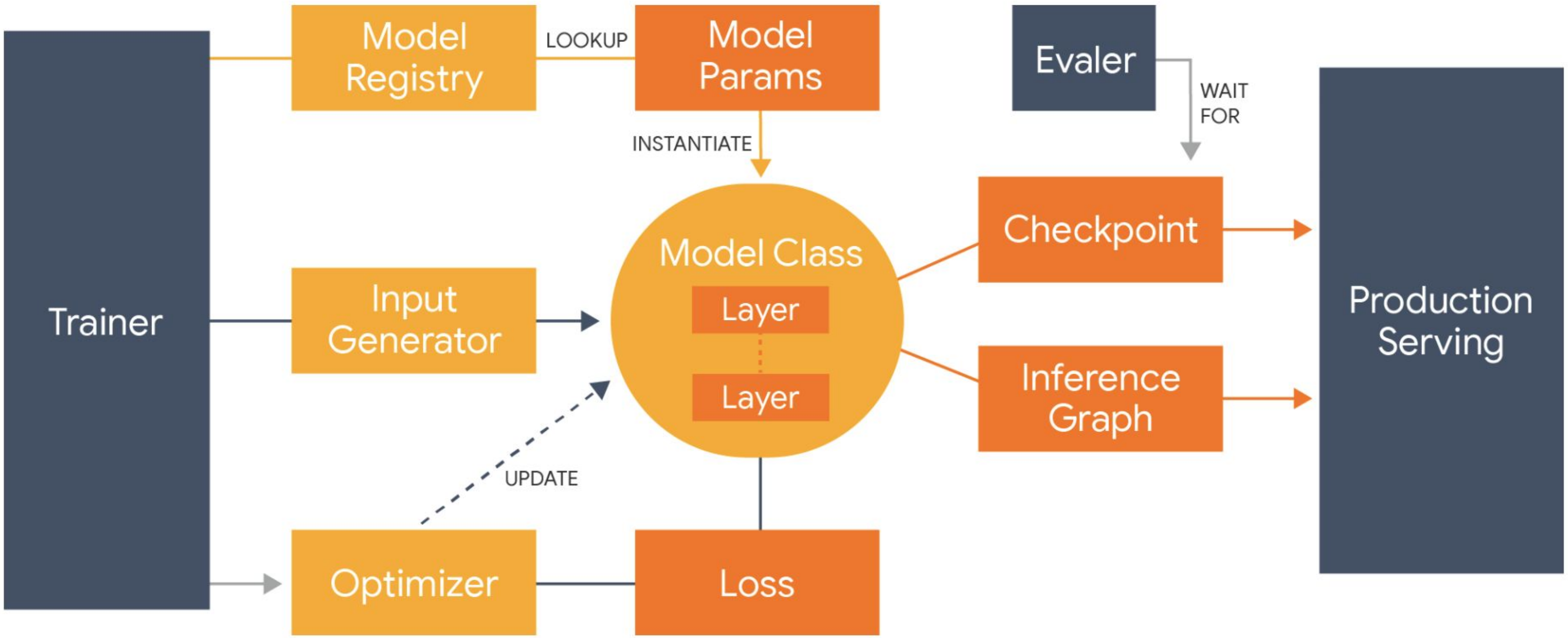
# Under the hood






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

# **Tensorflow Lingvo:** github.com/tensorflow/lingvo



# Trainer

- Construct TF training graph for a model. (cs)
  - Place variables
  - Place forward/backward computations
  - Summary

- Repeatedly calls: (TF 1.x)
  - TF session.run(train_op)

```python
class TrainerTpu(base_runner.BaseRunner):
  """Trainer on TPU."""

  def __init__(self, *args, **kwargs):
    super(TrainerTpu, self).__init__(*args, **kwargs)

    # Multiple TPU trainer tasks not tested/implemented.
    assert self._cluster.num_replicas == 1
    data_parallelism = self._cluster.num_splits_per_client
    assert data_parallelism
    num_devices_per_split = self._cluster.num_devices_per_split
    tf.logging.info('data_parallelism: %d, num_devices_per_split: %d',
                    data_parallelism, num_devices_per_split)

    self._steps_per_loop = min(self.params.train.tpu_steps_per_loop,
                               self.params.train.max_steps)

    self._cluster_def = self._cluster.worker_cluster_def

    self._initialized = threading.Event()

    tf.logging.info(
        'Creating TrainerTpu using data parallelism %s '
        'and %s steps_per_loop', data_parallelism, self._steps_per_loop)
```

https://github.com/tensorflow/lingvo/blob/master/lingvo/trainer.py

# Prototyping Workflow - I

0. Read bunch of papers, ask a research question, hypothesize a solution
1. Play first in a colab (quick & dirty)
   a. Isolated component, not even a layer (method) but a subroutine
   b. Validate that the math works

# Prototyping Workflow - II

0. Read bunch of papers, ask a research question, hypothesize a solution
1. Play first in a colab (quick & dirty)
   a. Isolated component, not even a layer (method) but a subroutine
   b. Validate that the math works


2. Write a unit test (quick & NOT dirty)
   a. Start with some smoke tests
   b. Then validity tests

# Writing Tests

Inherit from test_utils.TestCase.

In a test method:

- Construct the layer graph.

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

# Writing Tests

Inherit from test_utils.TestCase.

In a test method:

- Construct the layer graph.
- Verify:
  - The graph is constructible

```
def testEncoderConstruction(self):
  p = self._EncoderParams()
  _ = encoder.MTEncoderV1(p)
```

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

# Writing Tests

Inherit from test_utils.TestCase.

In a test method:

- Construct the layer graph.
- Verify:
  - The graph is constructible
  - # variables

```
def testEncoderVars(self):
  p = self._EncoderParams()
  mt_enc = encoder.TransformerEncoder(p)
  enc_vars = mt_enc.vars
  flatten_vars = enc_vars.Flatten()
  self.assertEqual(len(flatten_vars), 91)
```

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

# Writing Tests

Inherit from test_utils.TestCase.

In a test method:

- Construct the layer graph.
- Verify:
  - The graph is constructible
  - # variables
- Create inputs (tf.constant, feed_dict).
- Run the graph.

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

Google AI

# Writing Tests

Inherit from test_utils.TestCase.

In a test method:

- Construct the layer graph.
- Verify:
  - The graph is constructible
  - # variables
- Create inputs (tf.constant, feed_dict).
- Run the graph.
- Verify:
  - Shapes

```
a, m = frnn.FPropDefaultTheta(src_encs, src_paddings, inputs, paddings)
frnn_out = tf.concat([a, m], 2)

# Initialize all the variables, and then run one step.
tf.global_variables_initializer().run()
ys, = sess.run([frnn_out])
self.assertEqual(ys.shape, (7, 6, 8))
```

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

# Writing Tests

Inherit from test_utils.TestCase.

In a test method:

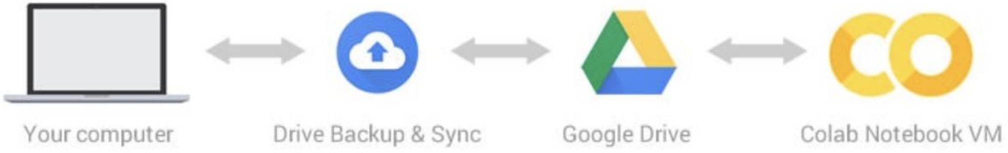
- Construct the layer graph.
- Verify:
  - The graph is constructible
  - # variables
- Create inputs (tf.constant, feed_dict).
- Run the graph.
- Verify:
  - Shapes
  - Output values

```
with self.session(use_gpu=True) as sess:
  tf.global_variables_initializer().run()
  actual_decode = sess.run(decode)

expected_topk_ids = [[2, 0, 0, 0, 0], [11, 2, 0, 0, 0], [2, 0, 0, 0, 0],
                     [6, 2, 0, 0, 0]]

expected_topk_lens = [1, 2, 1, 2]
expected_topk_scores = [[-3.78467, -5.771077], [-3.334115, -5.597376]]

self.assertAllEqual(expected_topk_ids, actual_decode.topk_ids)
self.assertAllEqual(expected_topk_lens, actual_decode.topk_lens)
self.assertAllClose(expected_topk_scores, actual_decode.topk_scores)
```

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

# Prototyping Workflow - III

0. Read bunch of papers, ask a research question, hypothesize a solution
1. Play first in a colab (quick & dirty)
   a. Isolated component, not even a layer (method) but a subroutine
   b. Validate that the math works




2. Write a unit test (quick & NOT dirty)
   a. Start with some smoke tests
   b. Then validity tests

3. Actual Runs
   a. Test it locally (within a simulated environment), if passes
   b. Run it on Data Centers with single machine, if passes
   c. Fully fledged run using multiple machines.

# Prototyping

Google AI

## do's

- Minimal code for the research question
- Does only one thing
- Math validated in a colab
- Written smoke tests
- Written detailed functionality tests

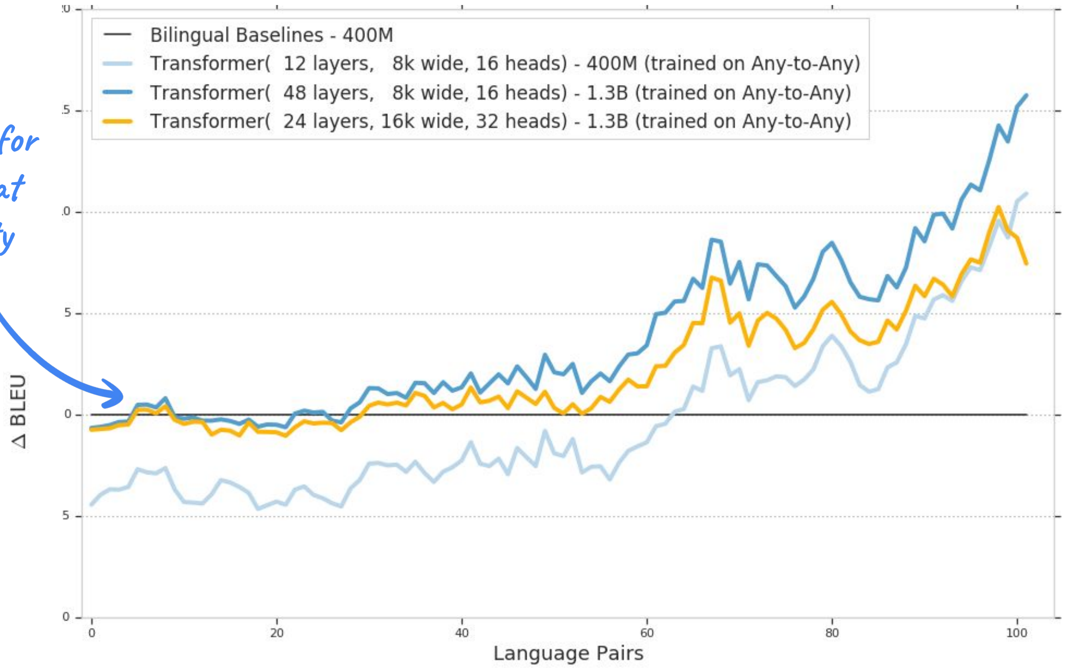
## don'ts

- Start with the final framework
- Add multiple functionalities
  - Too much branching in the code
  - Multiple options in the signature
- Missing tests
  - No tests for varying precision
  - No functionality tests (loss decreasing)

# **Prototyping Massive Models:** Width vs Depth (1.3B wide vs 1.3B deep)

Any⟶En translation performance with model size

Higher quality for deep vs. wide at same capacity



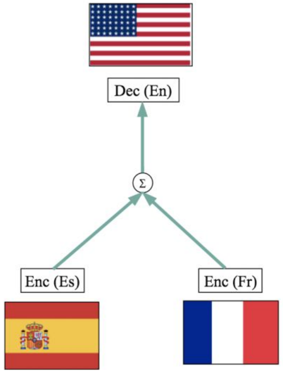

Bilingual Baselines

{Spanish, French, German, ...}

{Yoruba, Sindhi, Hawaiian, ...}

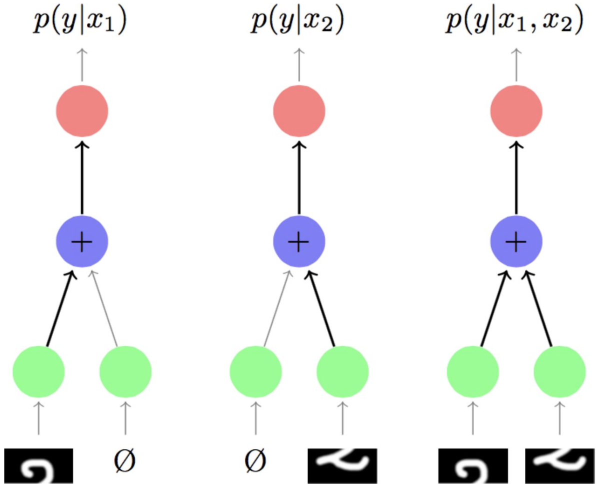
# **Prototyping Massive Models:** Reducing the Problem

- Multi-source Neural Machine Translation (Zoph and Knight, 2016)
  - Need to device a merger operation (i.e. sum, avg, gate)
  - Models take too long to train
  - Too many options to try




Train the model for $p(y|x_1, x_2)$

- Reduce the problem to a chewable size
  - Perhaps down to MNIST level



# Outline

- Prototyping
- **Debugging**
- Hyper-parameter Search
- Reproducibility
- Bonus: Coordination

# Debugging Large Scale Models

Sources of “bugs” in large scale machine learning:

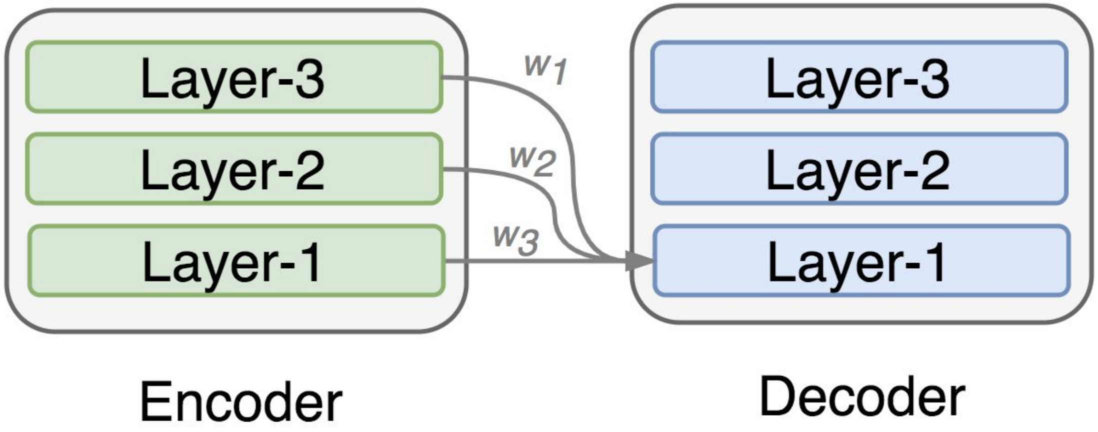
1. **Regular bugs:** introduced by ML practitioner
    a. Soln. go grab a coffee

2. **Device/Infra bugs:** hideous bugs
    a. Soln. change device, data, data center

3. **Theoretical bugs:** well... this should’ve never happened in the first place
    a. Soln. brush up your ML.
    b. Look at the right thing, norm of the gradient vs norm of the weights.
    c. Isolate initialization, optimization and malicious data.

# Transparent Attention or Encoder -I

## (Bapna et al. 2018- Training Deeper NMT Models with Transparent Attention)



# Transparent Attention or Encoder -II

## (Bapna et al. 2018- Training Deeper NMT Models with Transparent Attention)

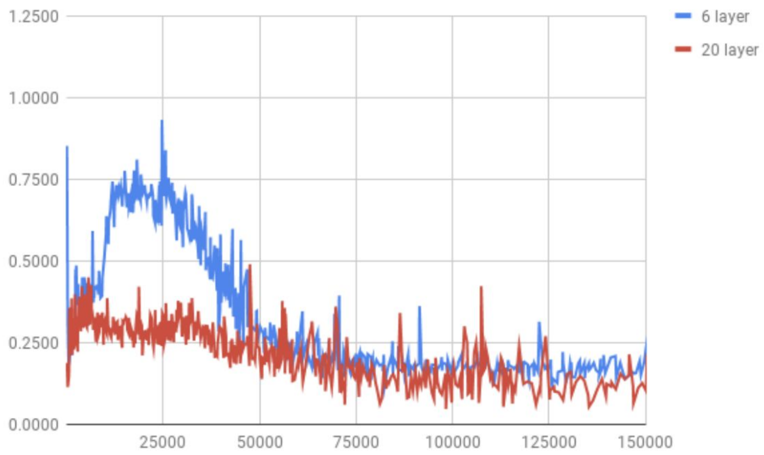



Figure 1: Grad-norm ratio ($r_t$) vs training step ($t$) comparison for a 6 layer (blue) and 20 layer (red) Transformer trained on WMT 14 En→De.

$$r_t = \left( \|\nabla_{h_1} L^{(t)}\| / \|\nabla_{h_N} L^{(t)}\| \right)$$

Indicator of a healthy training (Raghu et al. 2017)

- Lower layers converge quickly
- Topmost layers take longer

Expect large grad-norm ratio at the early stages of the training, then flatten.

# Transparent Attention or Encoder -III

## (Bapna et al. 2018- Training Deeper NMT Models with Transparent Attention)

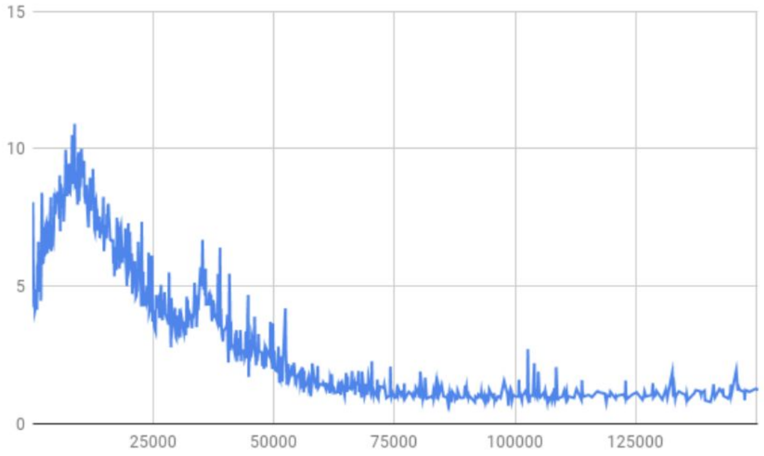



Figure 3: Grad-norm ratio ($r_t$) vs training step for 20 layer Transformer with transparent attention.

$$r_t = \left( \|\nabla_{h_1} L^{(t)}\| / \|\nabla_{h_N} L^{(t)}\| \right)$$

Indicator of a healthy training (Raghu et al. 2017)

- Lower layers converge quickly
- Topmost layers take longer

Expect large grad-norm ratio at the early stages of the training, then flatten.

# Transparent Attention or Encoder -IV

## (Bapna et al. 2018- Training Deeper NMT Models with Transparent Attention)

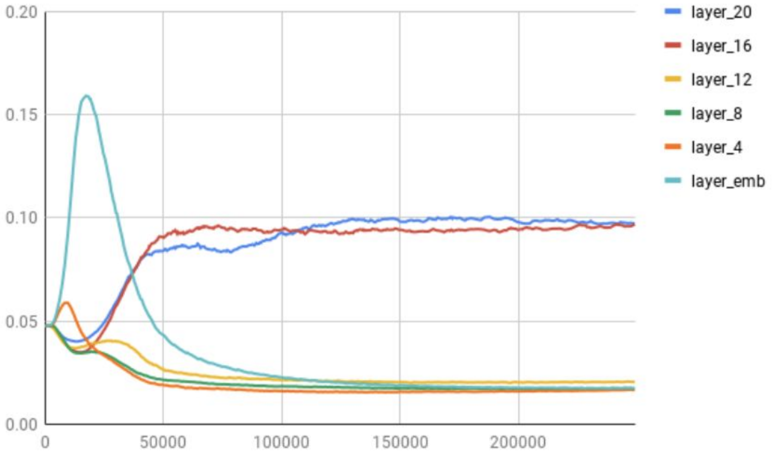



Figure 4: Plot illustrating the variations in the learned attention weights $s_{i,6}$ for the 20 layer Transformer encoder over the training process.

| En→De WMT 14 | Transformer (Base) | | | | (Big) |
|---|---|---|---|---|---|
| Encoder layers | 6 | 12 | 16 | 20 | 6 |
| Num. Parameters | 94M | 120M | 137M | 154M | 375M |
| Baseline | 27.26 | * | * | * | 27.94 |
| Baseline - residuals | * | 6.00 | * | * | N/A |
| Transparent | 27.52 | 27.79 | **28.04** | 27.96 | N/A |

Training dynamics:

- Raghu et al. 2017

Caveats:

- Residuals & Skip-connections → Shallowness

# Outline

- Prototyping
- Debugging
- **Hyper-parameter Search**
- Reproducibility
- Bonus: Coordination

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

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

**Data**

- Any Sequence
- Arbitrary length

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

## Data

- Any Sequence
- Arbitrary length

## Model

- Architectures
- Neural wiring
- How to parameterize

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

### Data

- Any Sequence
- Arbitrary length

### Model

- Architectures
- Neural wiring
- How to parameterize

### Objective & HParams

- Loss functions
- Optimizers
- All the other governing hyper-parameters

# Hyper-parameter Search

First: No one has infinite resources ⟶ the more compute we get, the larger we scale.

Some rule-of-thumbs

- All variables are interconnected: if you are changing one, expect the others to be changed
- Always start with the learning rate, then the batch-size
- Hill-climbing is as good as random search

Some tools to automate

- Vizier for Cloud
- Tune for Pytorch

# The Learning Rate Schedules

**“Often the single most important hyper-parameter”**

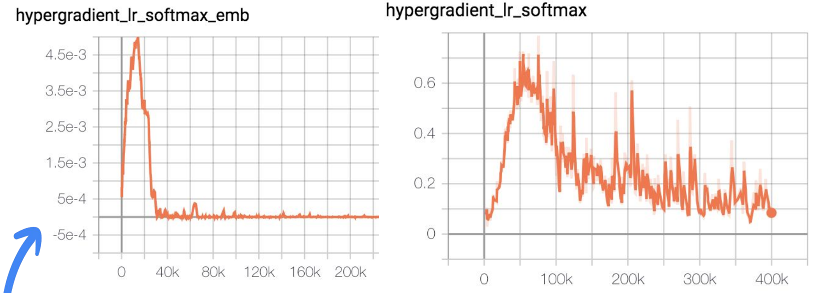
Practical recommendations for gradient-based training of deep architectures, Bengio 2012

**Should always be tuned.**

# Automate via Meta-Learning

Learning the learning rate: “Online Learning Rate Adaptation with Hypergradient Descent” Baydin et al. 2017

- Apply gradient descent on the learning rate (+underlying optimizer)
- Comparison
  - Single pair (wmt’19 en-de): HG ~ Baseline
  - Multi-task (wmt en-{de,fr}): HG > Baseline
  - BERT: HG ~ Baseline




*Learnt learning rate schedules (per-layer)*

# Outline

- Prototyping
- Debugging
- Hyper-parameter Search
- **Reproducibility**
- Bonus: Coordination

# 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

```
@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

# Bonus

Platform independent frameworks

TF -> CPU/GPU/TPU/Mobile/Browser

Don’t get lost in the nuances,
Ask yourself, which research question I’m trying to answer all the time
There is no end in optimization

Working with larger teams
    Async approaches

We need something post-silicone

# “Essentially, all models are wrong, but some are useful”

George E. P. Box

Google AI

# Thank You

orhanf@google.com