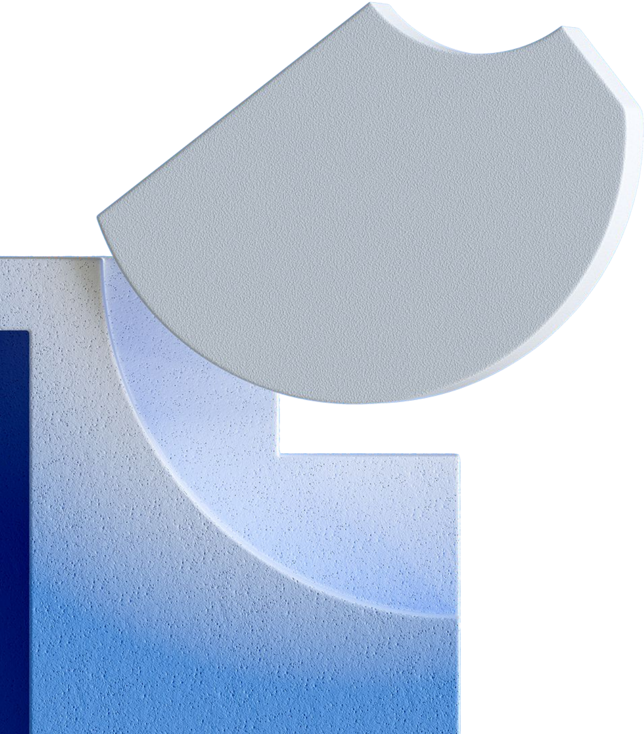
Google DeepMind



DiLoCo

Distributed Low-Communication for training Large Language Models

Feb 21, 2024, At Cohere For AI.

Arthur Douillard
Senior Research Scientist @ Google DeepMind
@ar_douillard

Team Work

Arthur Douillard

Qixuan Feng

Andrei Rusu

Rachita Chhaparia

Yani Donchev

Adhi Kuncoro

Bo Liu
work done as an intern

Satyen Kale

Marc'aurelio Ranzato

Arthur Szlam

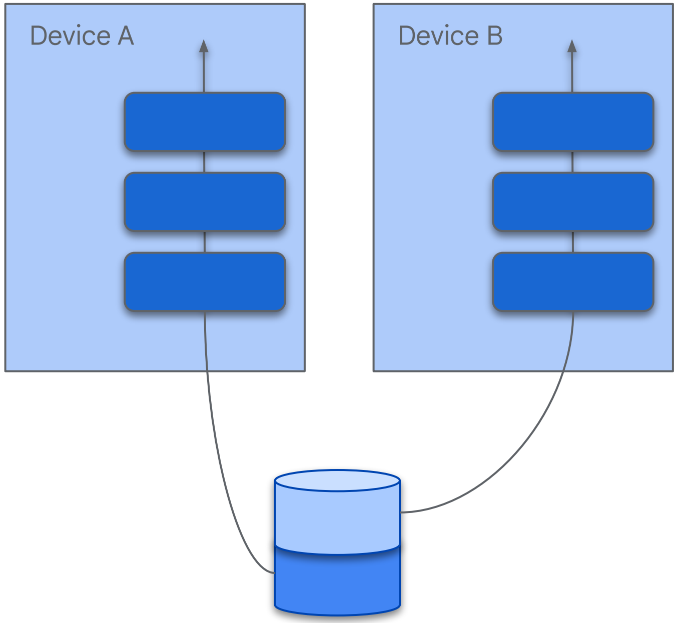
Jiajun Shen

Goals

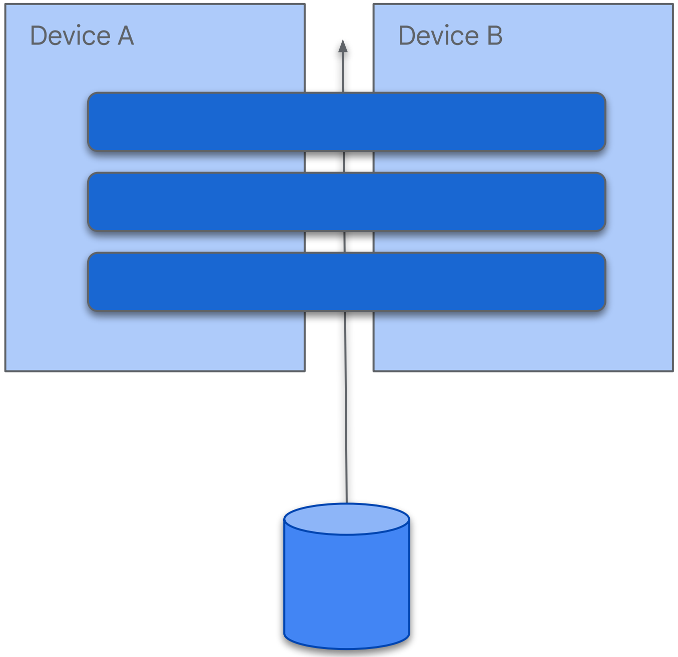
1

Parallelism Flavors

Data Parallelism

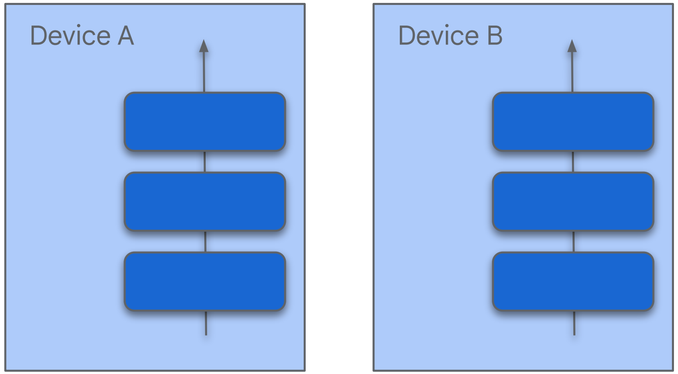


Tensor Parallelism

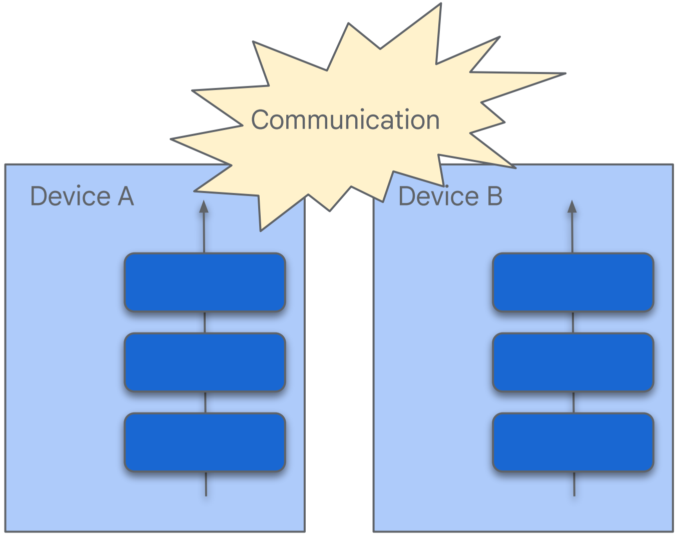


Parallelism Flavors

Data Parallelism



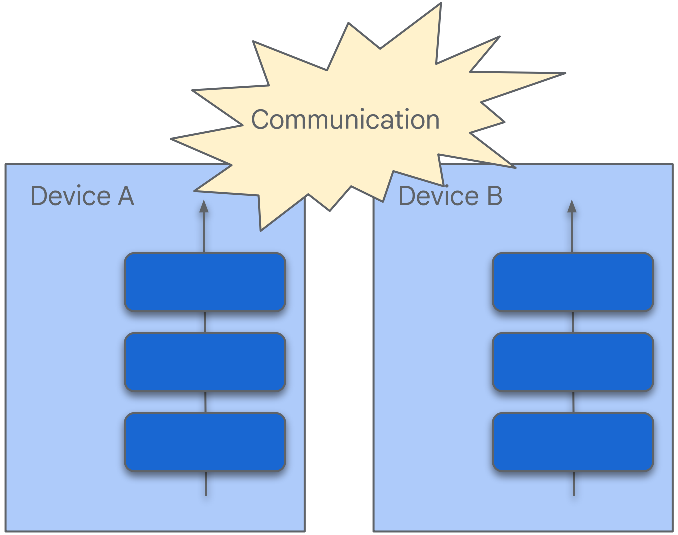
Parallelism Flavors



Data Parallelism

1. Compute per-batch loss on each device
2. Compute gradients on each device
3. All-reduce gradients & apply optimizer
4. Start anew from replicated parameters

Parallelism Flavors



⚠️ **Problem!**

Communication at every training step

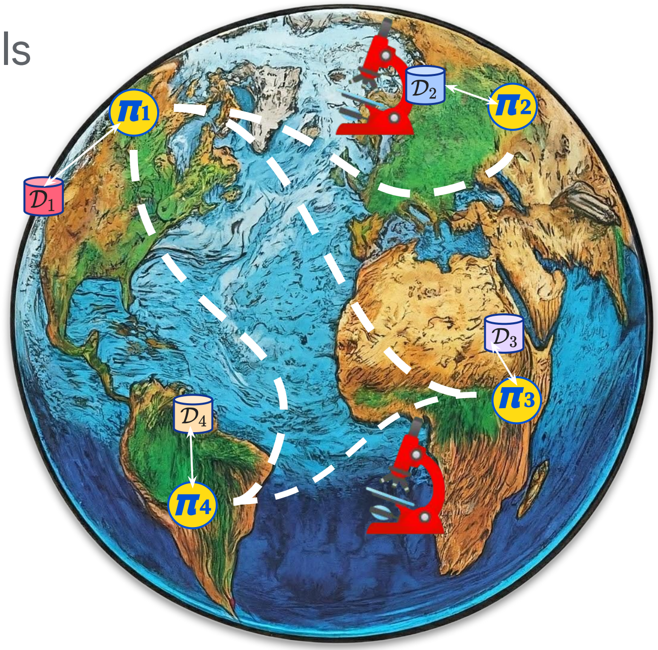
Hard to scale to non co-located devices

Our vision

- 🤝 World-wide collaborative training of large-scale models

- 🎓 Universities could pool their resources together to train larger models

- 🥖 Even if each worker has little compute, pooling all those bread crumbs is a lot of compute



Model

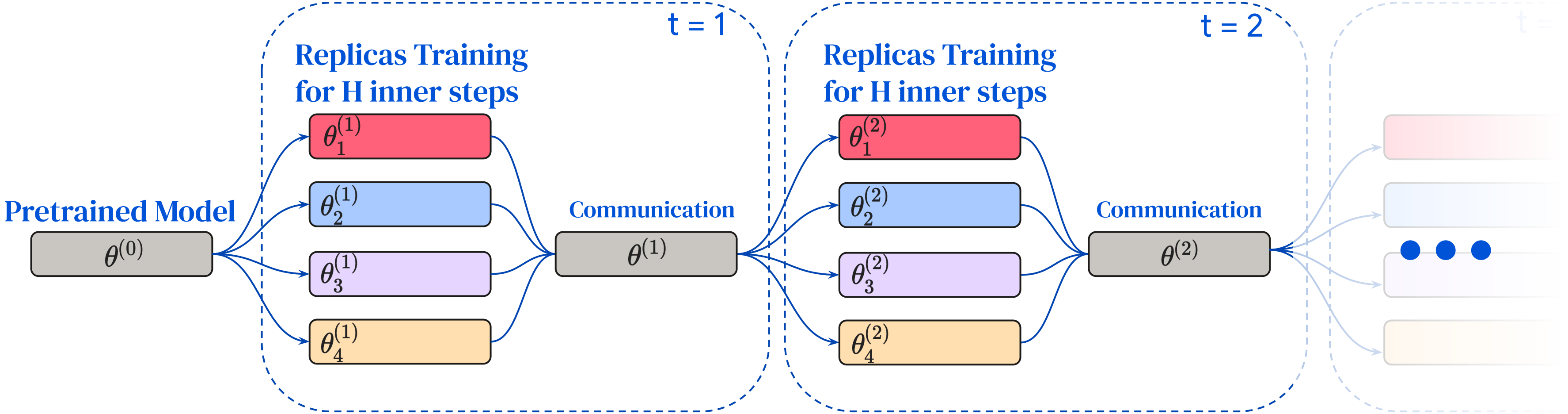
2

Communicating less = Independent Optimization

A world-wide distributed training should communicates less often

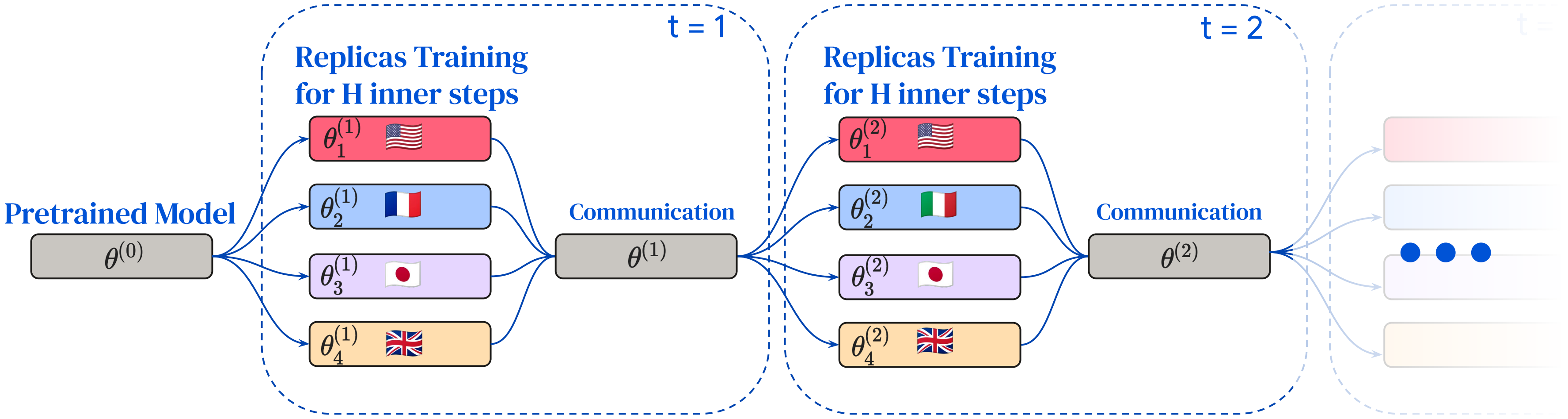
Less communication requires independent optimization

That’s the whole gist of Federated Learning, that we push to an extreme.



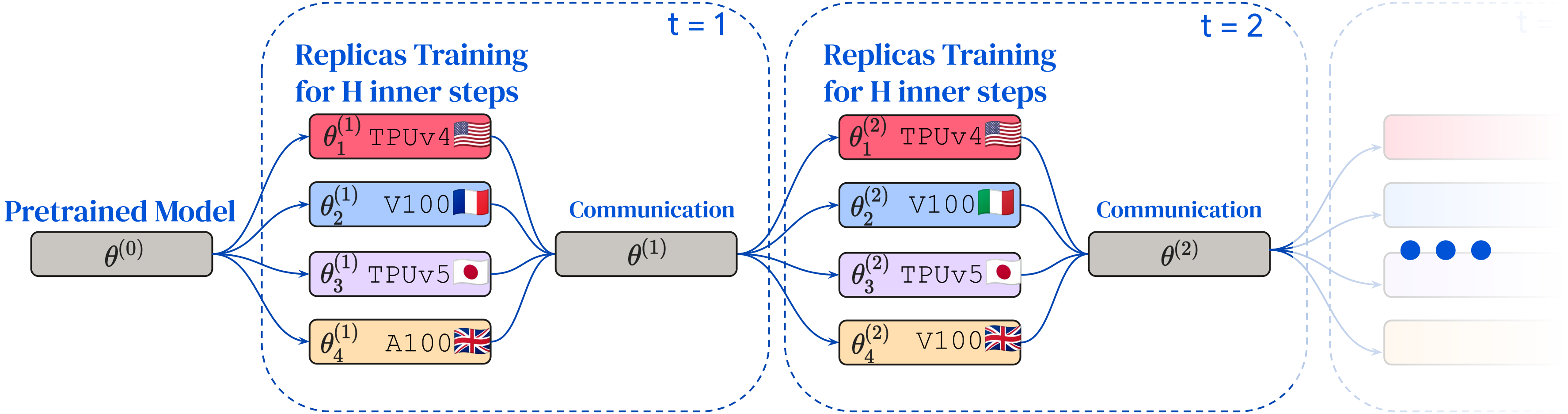
Communicating less = Independent Optimization

Independent training with infrequent communication enables cross-country distributed training



Communicating less = Independent Optimization

And even using different hardware types!



The recipe 🍜

Start from an existing model (optional).

Have k workers, across the world.

Assign a data shard to each worker, iid or not.

And use two optimizers!

Algorithm 1 DiLoCo Algorithm

Require: Initial model $\theta^{(0)}$
Require: k workers
Require: Data shards $\{\mathcal{D}_1, \dots, \mathcal{D}_k\}$
Require: Optimizers `InnerOpt` and `OuterOpt`

```
for outer step t = 1 ... T do
    for worker i = 1 ... k do
        θ_i^(t) ← θ^(t−1)
        for inner step h = 1 ... H do
            x ∼ D_i
            L ← f(x, θ_i^(t))
                                    ▷ Inner optimization:
            θ_i^(t) ← InnerOpt(θ_i^(t), ∇_L)
        end for
    end for
                                    ▷ Averaging outer gradients:
    Δ^(t) ← (1/k) Σ_{i=1}^{k} (θ^(t−1) − θ_i^(t))
                                    ▷ Outer optimization:
    θ^(t) ← OuterOpt(θ^(t−1), Δ^(t))
end for
```

The recipe 🍜

For every round of training-communication:

$\theta^{(0)}$

Algorithm 1 DiLoCo Algorithm

Require: Initial model $\theta^{(0)}$
Require: k workers
Require: Data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$
Require: Optimizers `InnerOpt` and `OuterOpt`

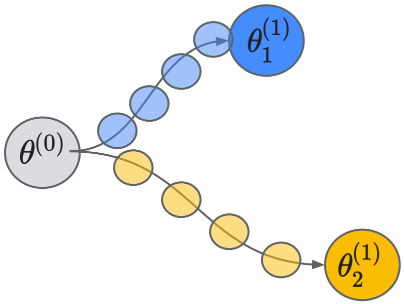
```
for outer step t = 1 ... T do
    for worker i = 1 ... k do
        θ_i^(t) ← θ^(t−1)
        for inner step h = 1 ... H do
            x ~ D_i
            L ← f(x, θ_i^(t))
                                ▷ Inner optimization:
            θ_i^(t) ← InnerOpt(θ_i^(t), ∇_L)
        end for
    end for
                                ▷ Averaging outer gradients:
    Δ^(t) ← (1/k) Σ_{i=1}^{k} (θ^(t−1) − θ_i^(t))
                                ▷ Outer optimization:
    θ^(t) ← OuterOpt(θ^(t−1), Δ^(t))
end for
```

The recipe 🍜

For every round of training-communication:

Train each worker in parallel,

For H training steps



Algorithm 1 DiLoCo Algorithm

Require: Initial model $\theta^{(0)}$
Require: k workers
Require: Data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$
Require: Optimizers `InnerOpt` and `OuterOpt`

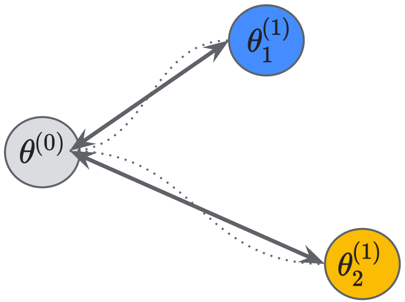
for `outer step` $t = 1 \ldots T$ **do**
  for `worker` $i = 1 \ldots k$ **do**
    $\theta_i^{(t)} \leftarrow \theta^{(t-1)}$
    for `inner step` $h = 1 \ldots H$ **do**
      $x \sim \mathcal{D}_i$
      $\mathcal{L} \leftarrow f(x, \theta_i^{(t)})$
      ▷ Inner optimization:
      $\theta_i^{(t)} \leftarrow$ `InnerOpt`$(\theta_i^{(t)}, \nabla_{\mathcal{L}})$
    end for
  end for
  ▷ Averaging outer gradients:
  $\Delta^{(t)} \leftarrow \frac{1}{k} \sum_{i=1}^{k} (\theta^{(t-1)} - \theta_i^{(t)})$
  ▷ Outer optimization:
  $\theta^{(t)} \leftarrow$ `OuterOpt`$(\theta^{(t-1)}, \Delta^{(t)})$
end for

The recipe 🍜

Afterwards, compute an outer gradient.

While usually a gradient a gradient is an infinitesimal difference,

this outer gradient is a delta in parameter space across hundreds or thousands of training steps!



Algorithm 1 DiLoCo Algorithm

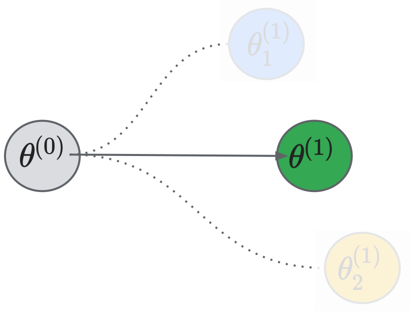
Require: Initial model $\theta^{(0)}$
Require: k workers
Require: Data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$
Require: Optimizers `InnerOpt` and `OuterOpt`

for `outer step` $t = 1 \ldots T$ **do**
  for `worker` $i = 1 \ldots k$ **do**
    $\theta_i^{(t)} \leftarrow \theta^{(t-1)}$
    for `inner step` $h = 1 \ldots H$ **do**
      $x \sim \mathcal{D}_i$
      $\mathcal{L} \leftarrow f(x, \theta_i^{(t)})$
      ▷ Inner optimization:
      $\theta_i^{(t)} \leftarrow$ `InnerOpt`$(\theta_i^{(t)}, \nabla_{\mathcal{L}})$
    end for
  end for
  ▷ Averaging outer gradients:
  $\Delta^{(t)} \leftarrow \frac{1}{k} \sum_{i=1}^{k} (\theta^{(t-1)} - \theta_i^{(t)})$
  ▷ Outer optimization:
  $\theta^{(t)} \leftarrow$ `OuterOpt`$(\theta^{(t-1)}, \Delta^{(t)})$
end for

The recipe 🍜

We can apply an optimization on this outer "gradient"!

If we use `SGD(lr=1.0)`, this is **model averaging (FedAvg)**



Algorithm 1 DiLoCo Algorithm

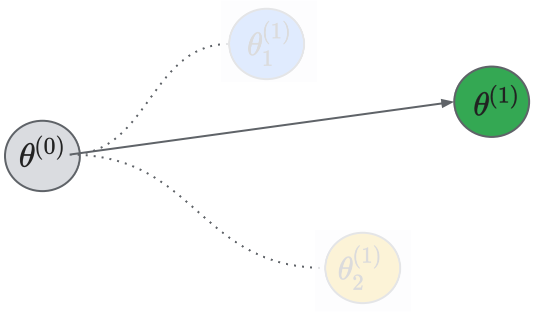
Require: Initial model $\theta^{(0)}$
Require: k workers
Require: Data shards $\{\mathcal{D}_1, \dots, \mathcal{D}_k\}$
Require: Optimizers `InnerOpt` and `OuterOpt`

for `outer step` $t = 1 \dots T$ **do**
  for `worker` $i = 1 \dots k$ **do**
    $\theta_i^{(t)} \leftarrow \theta^{(t-1)}$
    for `inner step` $h = 1 \dots H$ **do**
      $x \sim \mathcal{D}_i$
      $\mathcal{L} \leftarrow f(x, \theta_i^{(t)})$
      ▷ Inner optimization:
      $\theta_i^{(t)} \leftarrow$ `InnerOpt`$(\theta_i^{(t)}, \nabla_{\mathcal{L}})$
    end for
  end for
  ▷ Averaging outer gradients:
  $\Delta^{(t)} \leftarrow \frac{1}{k} \sum_{i=1}^{k} (\theta^{(t-1)} - \theta_i^{(t)})$
  ▷ Outer optimization:
  $\theta^{(t)} \leftarrow$ `OuterOpt`$(\theta^{(t-1)}, \Delta^{(t)})$
end for

The recipe 🍜

We can apply an optimization on this outer gradient!

With `SGD(lr=1.0, momentum=0.9, nesterov=True)`, we can **speed up** training.



Algorithm 1 DiLoCo Algorithm

Require: Initial model $\theta^{(0)}$
Require: k workers
Require: Data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$
Require: Optimizers `InnerOpt` and `OuterOpt`

for `outer step` $t = 1 \ldots T$ **do**
  for `worker` $i = 1 \ldots k$ **do**
    $\theta_i^{(t)} \leftarrow \theta^{(t-1)}$
    for `inner step` $h = 1 \ldots H$ **do**
      $x \sim \mathcal{D}_i$
      $\mathcal{L} \leftarrow f(x, \theta_i^{(t)})$
      ▷ Inner optimization:
      $\theta_i^{(t)} \leftarrow \texttt{InnerOpt}(\theta_i^{(t)}, \nabla_{\mathcal{L}})$
    end for
  end for
  ▷ Averaging outer gradients:
  $\Delta^{(t)} \leftarrow \frac{1}{k} \sum_{i=1}^{k} (\theta^{(t-1)} - \theta_i^{(t)})$
  ▷ Outer optimization:
  $\theta^{(t)} \leftarrow \texttt{OuterOpt}(\theta^{(t-1)}, \Delta^{(t)})$
end for

Results

We compare 3 baselines, with different amount of communication, compute/data, and time spent.

Model	Communication	Time	Compute & Data	Perplexity
Baseline	0	1×	1×	16.23
Baseline, 8× batch size with data parallelism	$8 \times N$	1×	8×	15.30
Baseline, 8× batch size with microbatching	0	8×	8×	15.30
Baseline, 8× updates	0	8×	8×	14.72

Results

We compare 3 baselines, with different amount of communication, compute/data, and time spent.

DiLoCo strikes the best tradeoff between time, communication cost and generalization performance.

Given 8 replicas, N the number of steps, and H the communication frequency in steps

Model	Communication	Time	Compute & Data	Perplexity
Baseline	0	1×	1×	16.23
Baseline, 8× batch size with data parallelism	$8 \times N$	1×	8×	15.30
Baseline, 8× batch size with microbatching	0	8×	8×	15.30
Baseline, 8× updates	0	8×	8×	14.72
DiLoCo	$8 \times N/H$	1×	8×	15.02

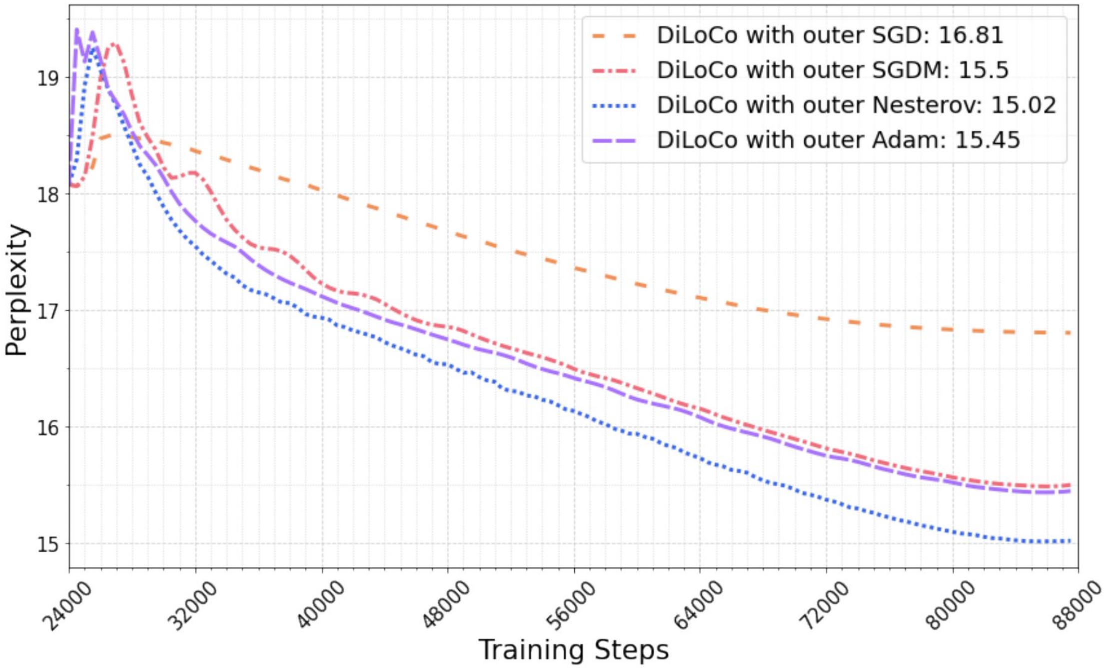
DiLoCo’s outer optimizers

Outer SGD = FedAvg
⟶ McMahan et al. 2016,
Communication-Efficient Learning of Deep Networks from Decentralized Data

Outer Adam = FedOpt
⟶ Reddi et al. 2020,
Adaptive Federated Optimization

We found empirically that Nesterov is:

- Better
- More stable across HPs
- And allowed us to scale to O(100) inner steps while previous literature usually is O(10).

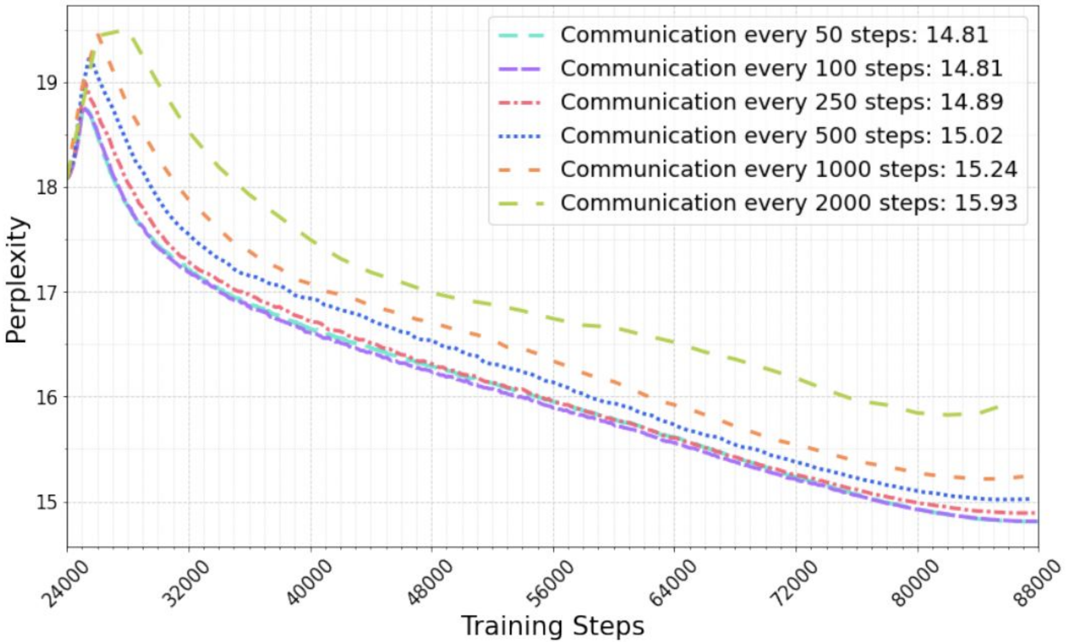


Resiliency to communication frequency

If bandwidth is small, reduce communication.

Amortize time spent in synchronization!

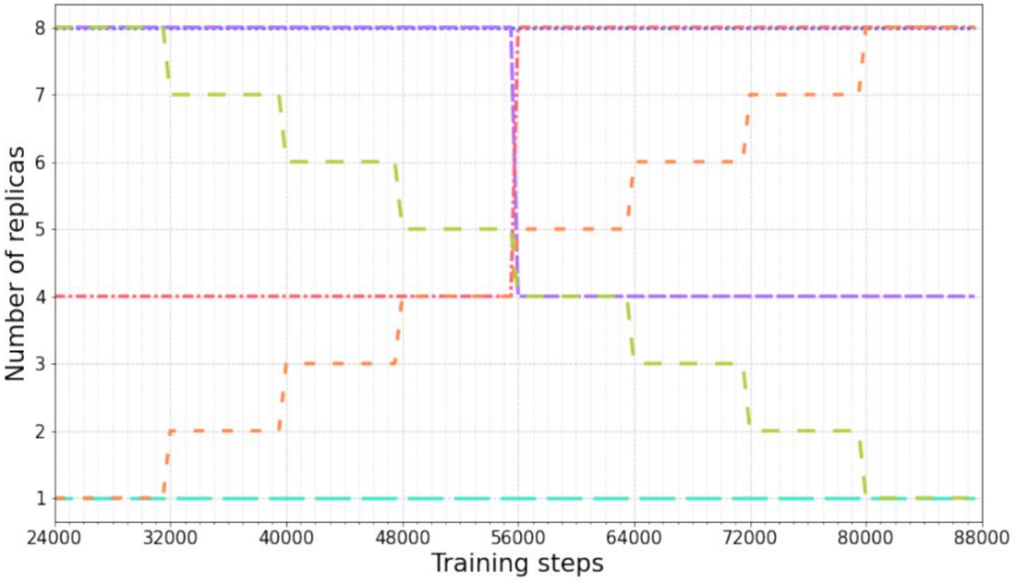
Typical Federated Learning set up: O(10) inner steps.
We work with one or two orders of magnitude more.



Resiliency to compute availability

How about if some workers go down, or become available at a later time?

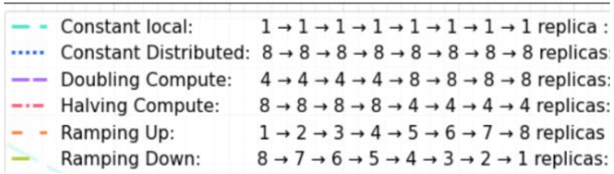
We tried varying the number of workers across time...



Resiliency to compute availability

How about if some workers go down, or become available at a later time?

We tried varying the number of workers across time…



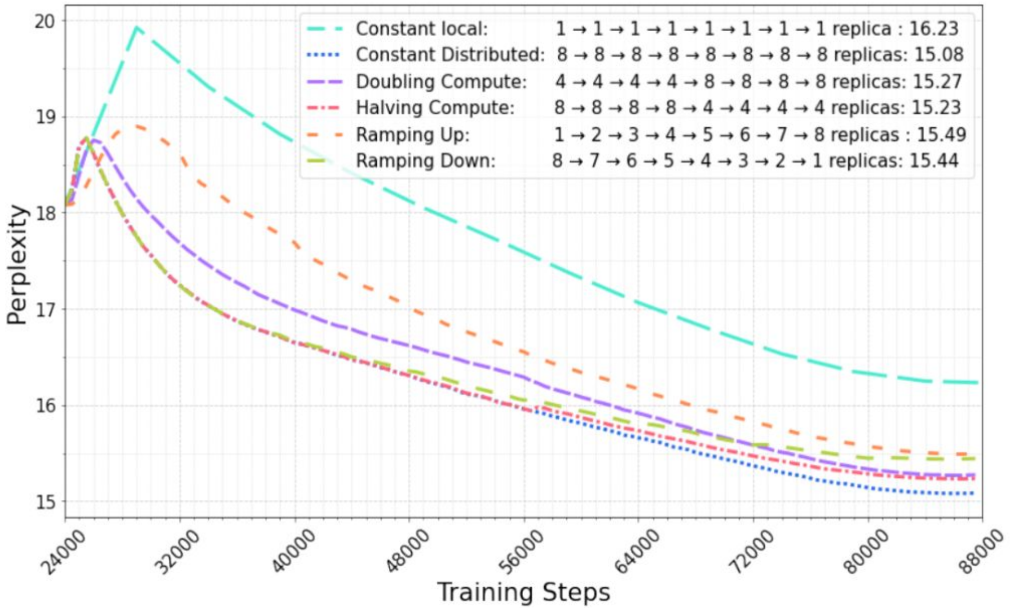
Number of replica per phase.

Resiliency to compute availability

How about if some workers go down, or become available at a later time?

All variations are close to the "ideal" `Constant Distributed`, where the number of workers is always 8.

PS: changing the outer learning rate should probably improve results

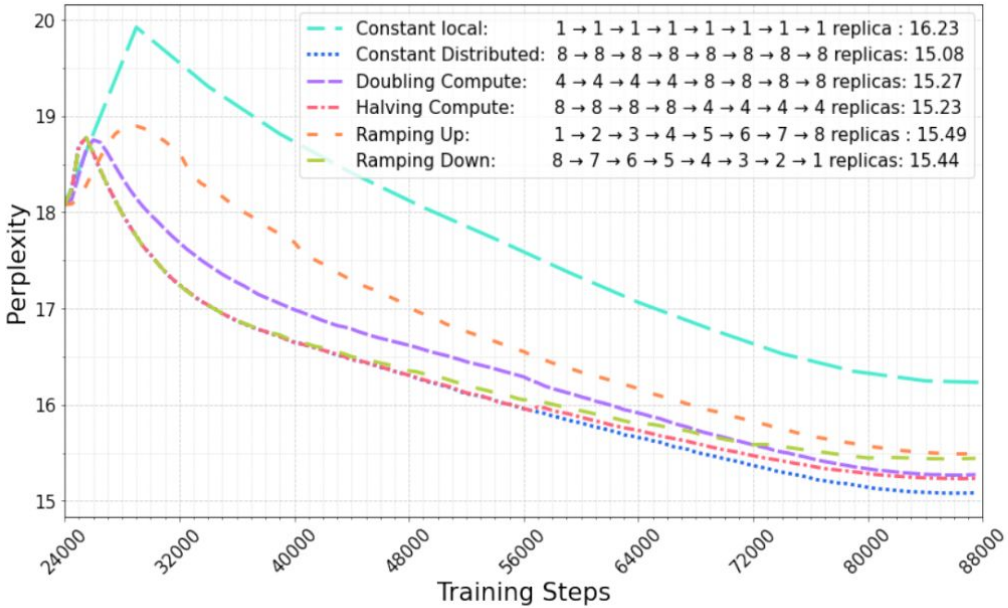


Resiliency to compute availability

How about if some workers go down, or become available at a later time?

Convergence is sensitive to the total amount of compute, not to how this is spread over time.

Name	Total Compute	PPL
Constant	64	15.08
Doubling	48	15.27
Halving	48	15.23
Ramp Up	28	15.49
Ramp Down	28	15.44



Further reducing communication cost

DiLoCo amortizes communication cost only communicating every 500 steps.

However, there is still, infrequently, a communication cost that can be problematic.

→ we experiment **compressing** our *outer gradient* by pruning it values. It seems to be fairly robust!

% of pruned values	Perplexity	Relative change
0%	15.02	0%
25%	15.01	-0.06%
50%	15.08	+0.39%
75%	15.27	+1.66%

Model Sizes

Model merging literature indicates that larger models are easier to merge/soup.

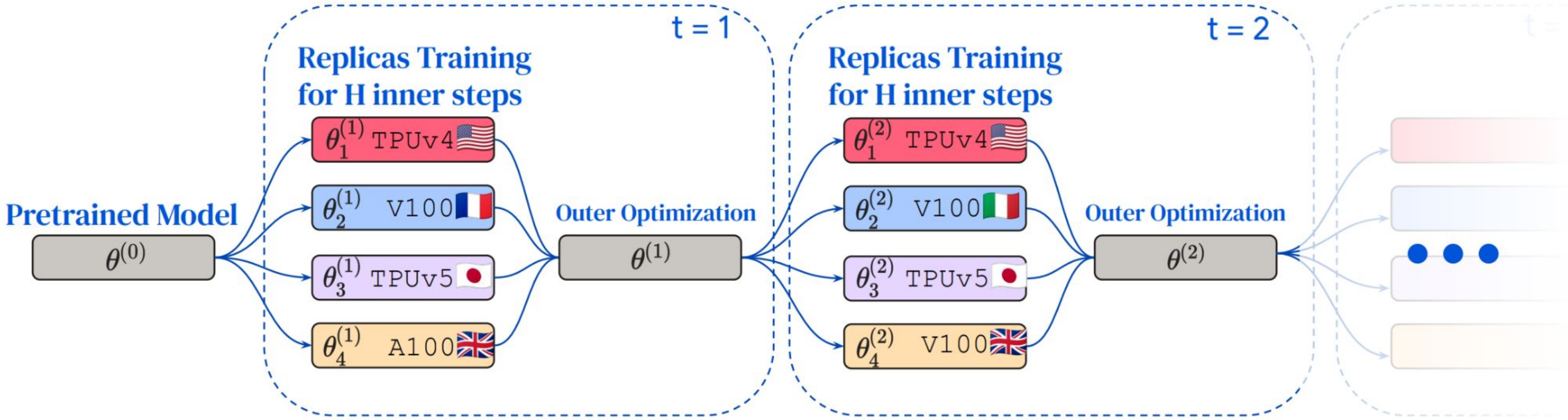
Preliminary experiments up to 400M scale indicates that it may be true for DiLoCo too.

Model Size	Relative (%)	Absolute (PPL)
60M	4.33%	1.01
150M	7.45%	1.21
400M	7.49%	1.01

But wait we are waiting for laggers? 🐢 vs 🐇

A100 is twice faster than V100.

Being distributed is cool, but if we have to wait for laggers that's a bit sad.



Async

3

Bo Liu
Lead author of the async
during his internship

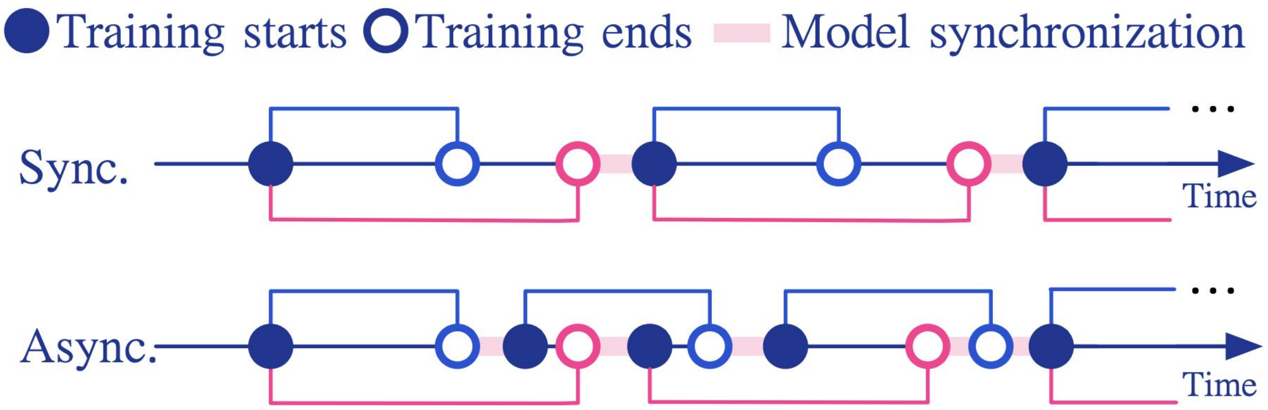
Let’s make it async then easy peasy

Instead of waiting for all replicas to finish before synchronization, let each worker update the global parameter as soon as it has completed its task.

Example with two replicas:

Fast devices

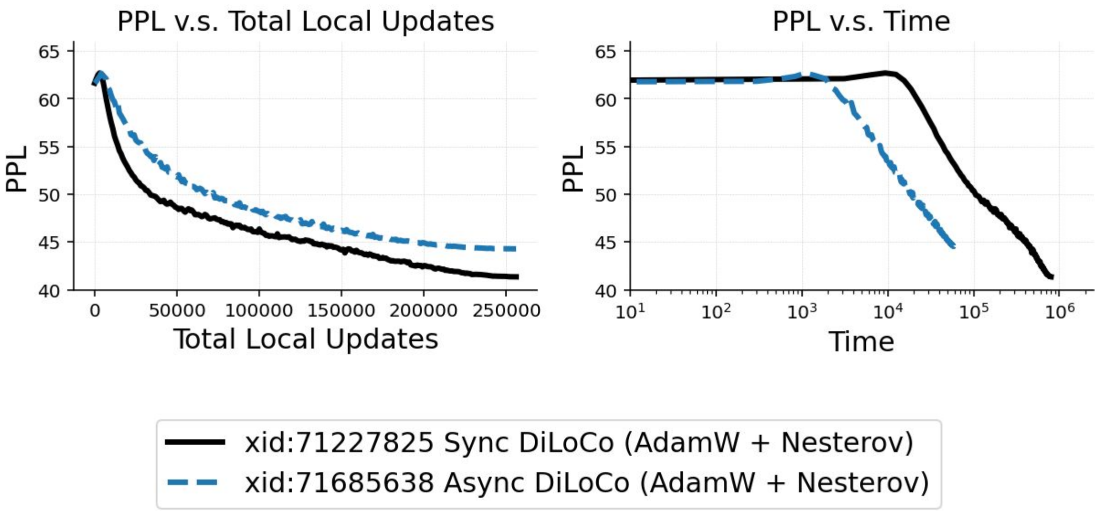
Slow devices



Does async works out-of-the-box?

When dealing with heterogeneous devices, it's faster!

But with respect to the number of updates, that's quite worse despite using as much data and compute.

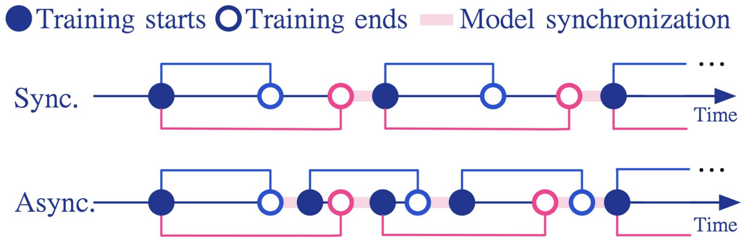


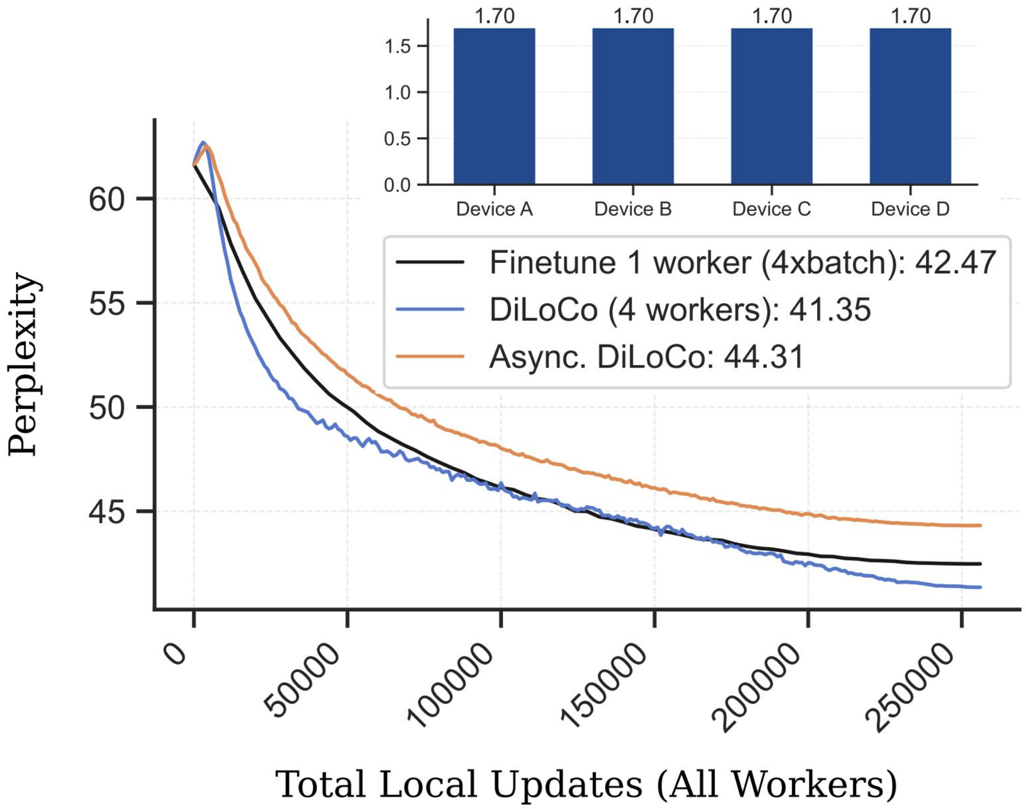
Let's try async with the same speed/replica then

When all devices have the same speed, we synchronize the global model one after the other,

but there isn't significant staleness between updates…

→ even slight staleness in the outer gradient is harmful!





Delayed Momentum Update

The culprit comes from the momentum of the outer optimizer.

Async works fine when using SGD w/o momentum as outer optimizer.

Delayed Momentum Update

The culprit comes from the momentum of the outer optimizer.

Async works fine when using SGD w/o momentum as outer optimizer.

Our solution is to update the outer momentum only once in a while, once the buffer of outer gradient is filled.

→ leading to more accurate outer momentum

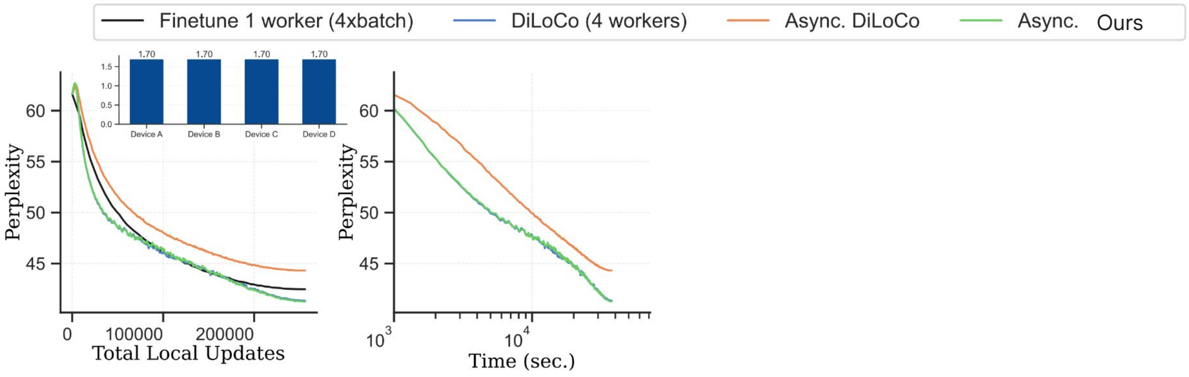
Algorithm 3 Delayed Nesterov Update.

Require: Initial model parameter θ_0
Require: Momentum decay $\beta \in (0, 1)$
Require: Momentum activation $c \in [0, 1/N]$ ▹ default to $c = 0$
Require: Buffer size N
$t = 0$
$m_0 = 0$ ▹ momentum
$\Delta = 0$ ▹ aggregated gradient
while not finished **do**
Receive the pseudo-gradient g_t ▹ sync. step in Alg. 2.
$\Delta \leftarrow \Delta + g_t$
if $(t+1)$ % $N == 0$ **then**
$m_{t+1} \leftarrow \beta m_t + \Delta/N$
$\theta_{t+1} \leftarrow \theta_t - \epsilon\big((1 - cN + c)\beta m_{t+1} + g_t/N\big)$
$\Delta = 0$
else
$m_{t+1} \leftarrow m_t$ ▹ delay momentum update
$\theta_{t+1} \leftarrow \theta_t - \epsilon\big(c\beta m_{t+1} + g_t/N\big)$
end if
$t \leftarrow t + 1$
end while

c=0 for best results

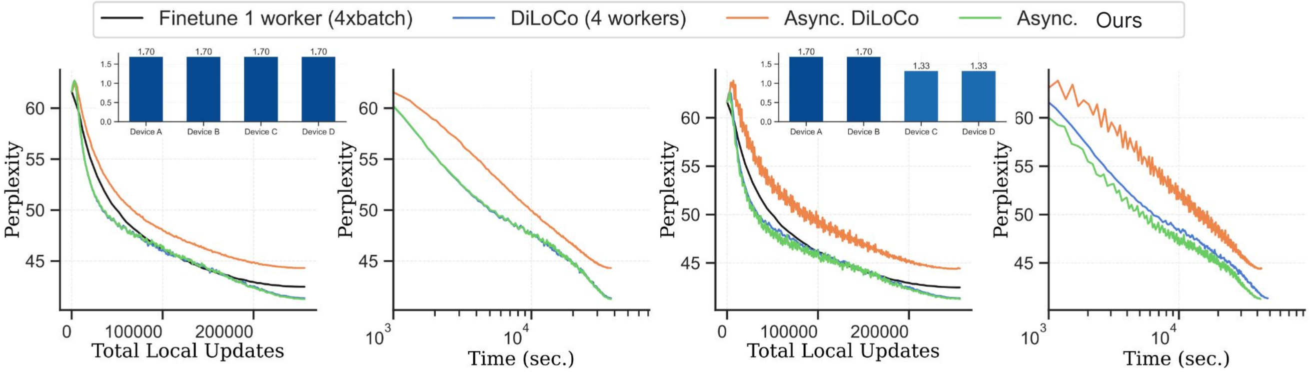
What's the results?

Given little staleness, Our **method** is as good as **DiLoCo**



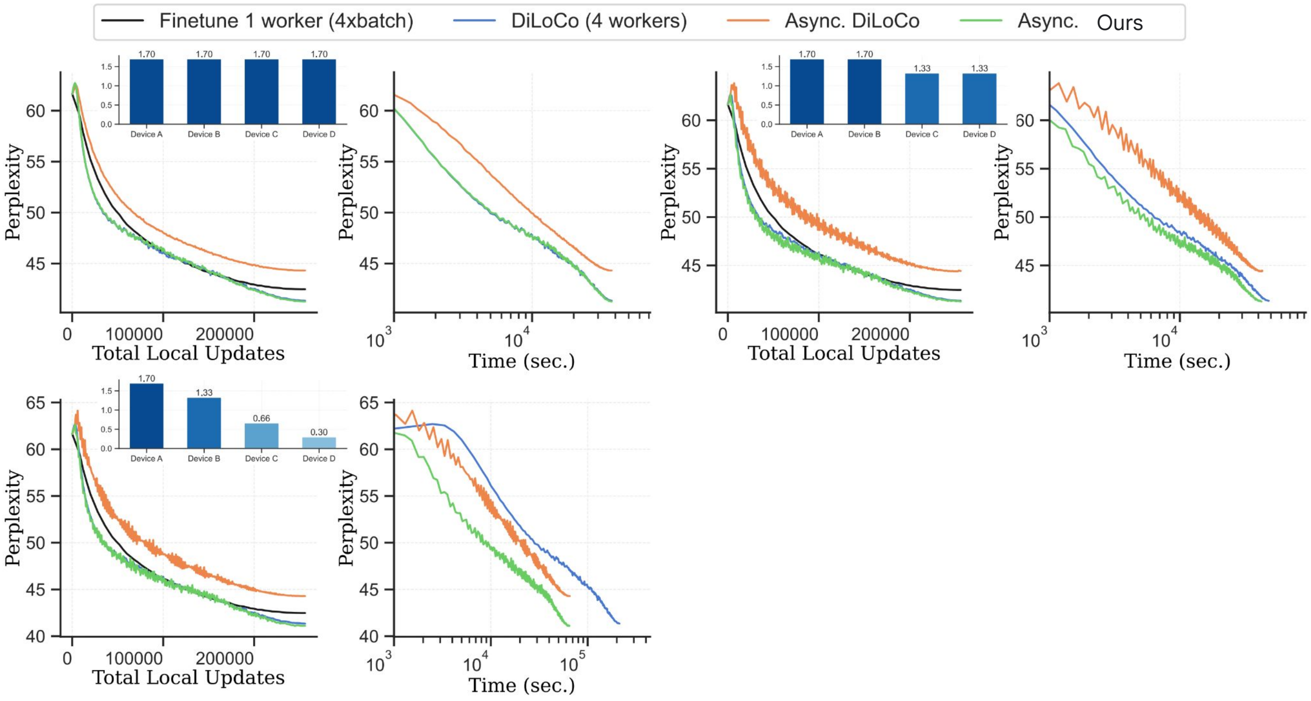
What’s the results?

But add more staleness by using different device types per replicas…



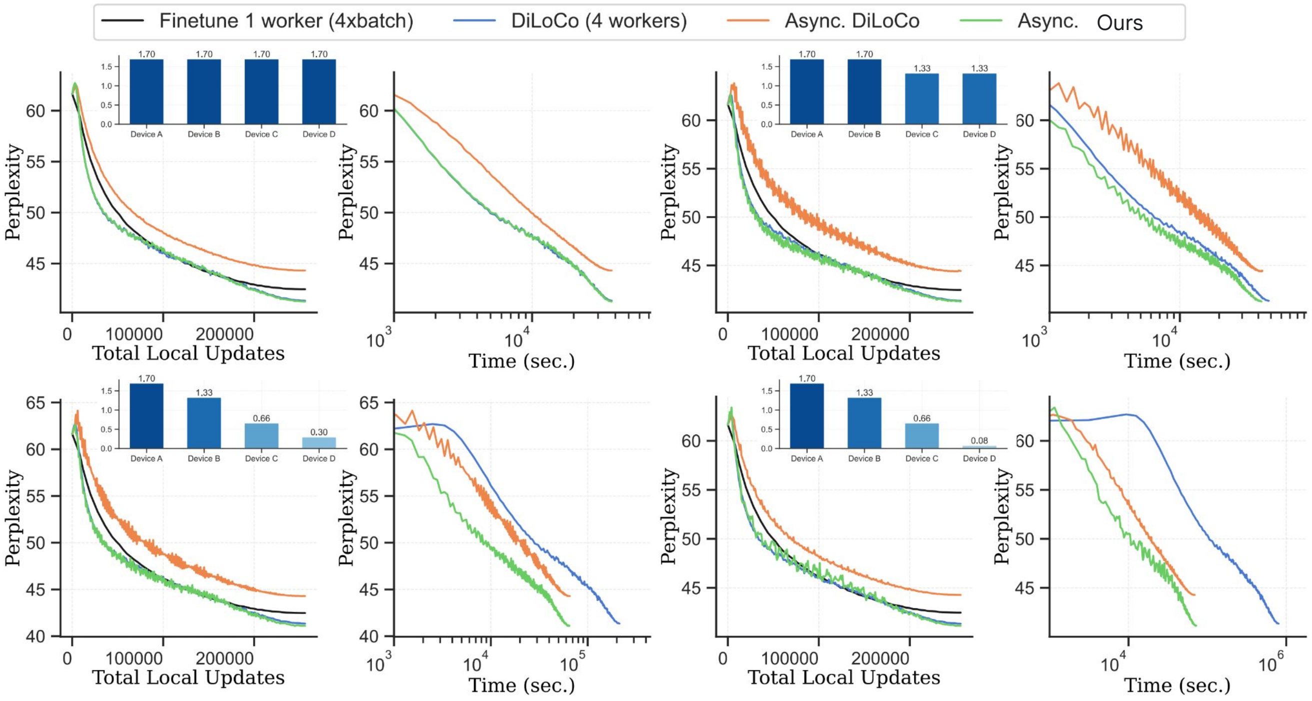
What's the results?

But add more staleness by using different device types per replicas…



What's the results?

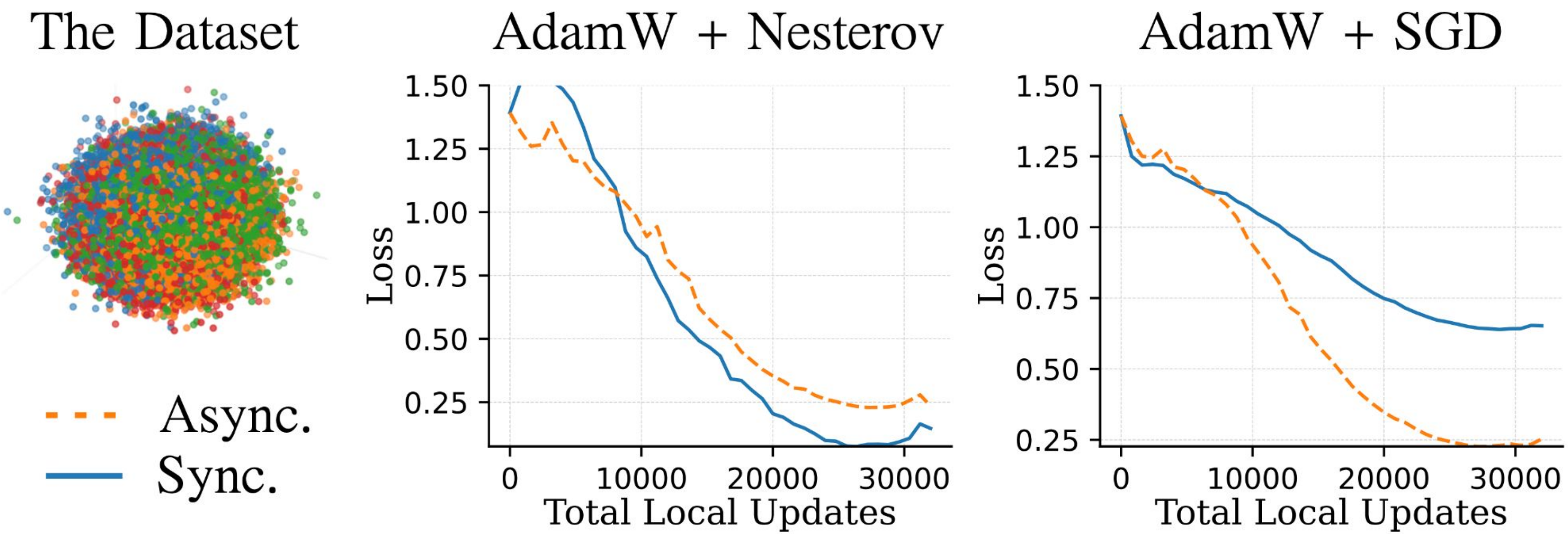
... and our **method** is as good as **DiLoCo** per-step, but much faster w.r.t time!



Open-source toy setting

We run our model distributed across the world thanks to Google infra.

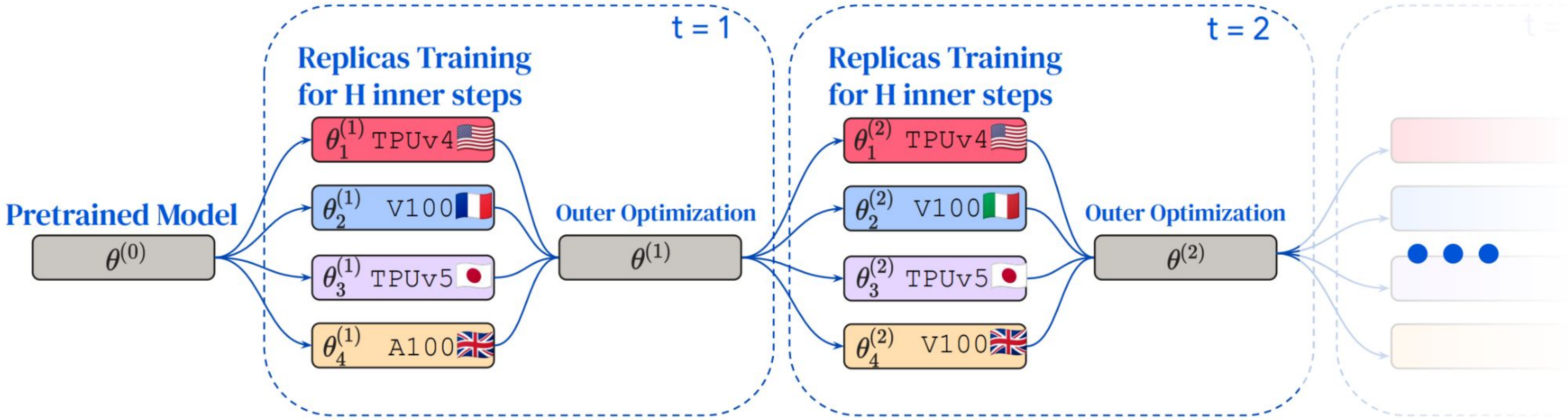
To facilitate reproduction by the community we release a toy setting that can be run in a colab, and that is faithful to the larger scale results!



🤔 TL;DR

We propose a a communication efficient distributed training algorithm:

- With extremely infrequent synchronization (x500 less!)
- Experiments on language models up to 400M
- With actual real experiments done across the world, and not only on a toy setting
- And an asynchronous extension to handle heterogeneous devices



Questions?

DiLoCo: arxiv.org/abs/2311.08105

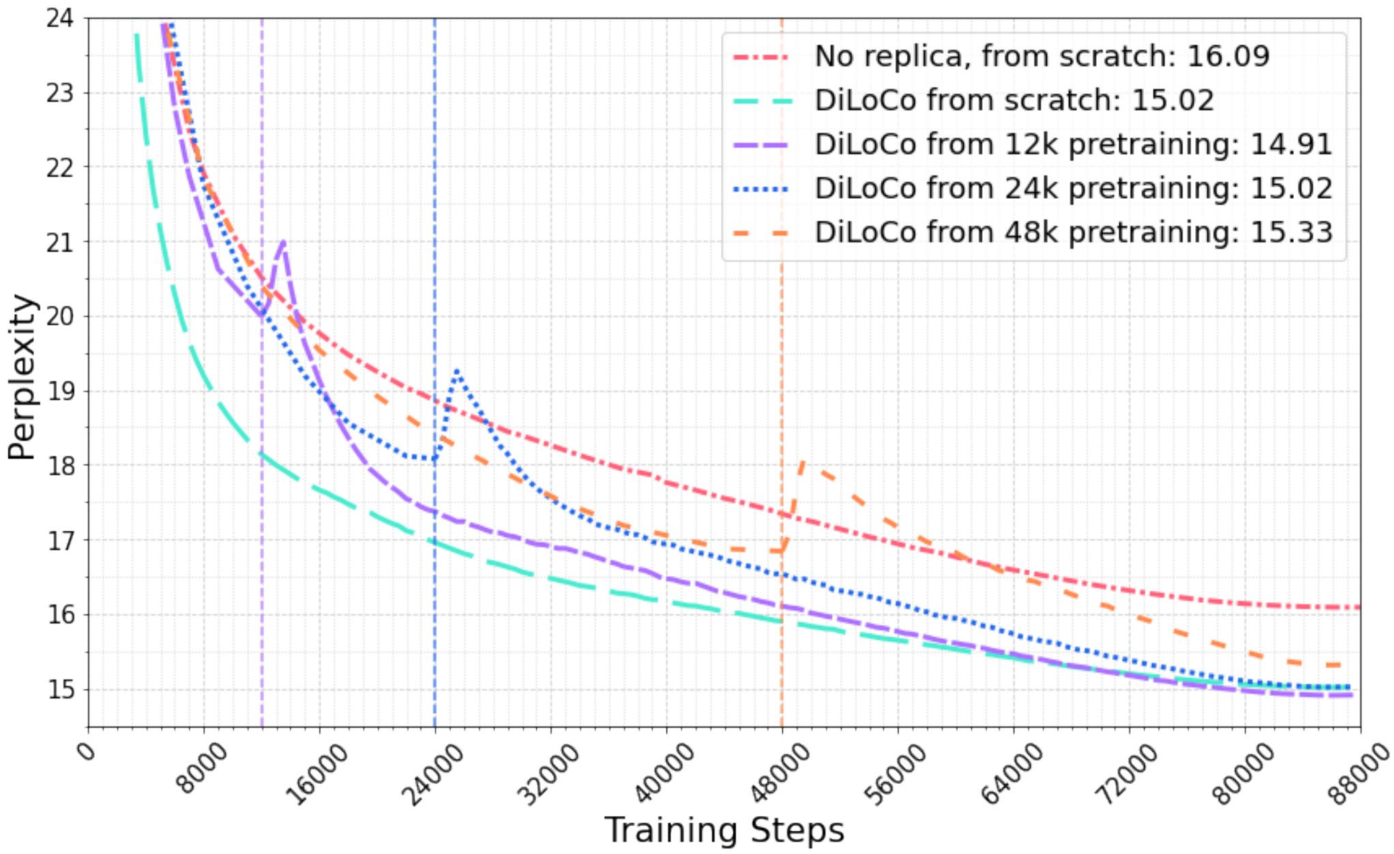
Async-DiLoCo: arxiv.org/abs/2401.09135

Arthur Douillard
@ar_douillard

Google DeepMind

Appendix

DiLoCo's pretraining size



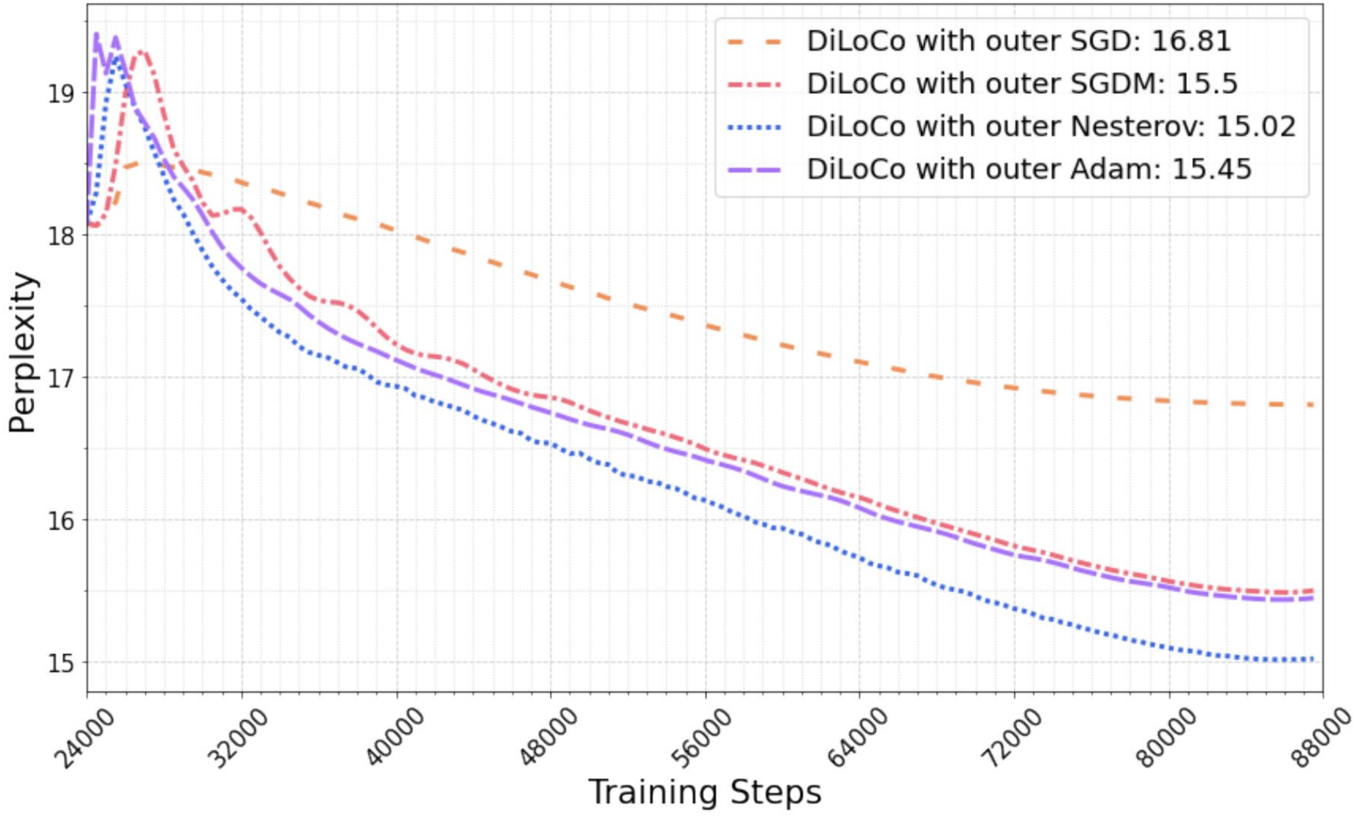
DiLoCo's number of replicas

Number of replicas	*i.i.d*	non-*i.i.d*
1	16.23	
4	15.23	15.18
8	15.08	15.02
16	15.02	14.91
64	14.95	14.96

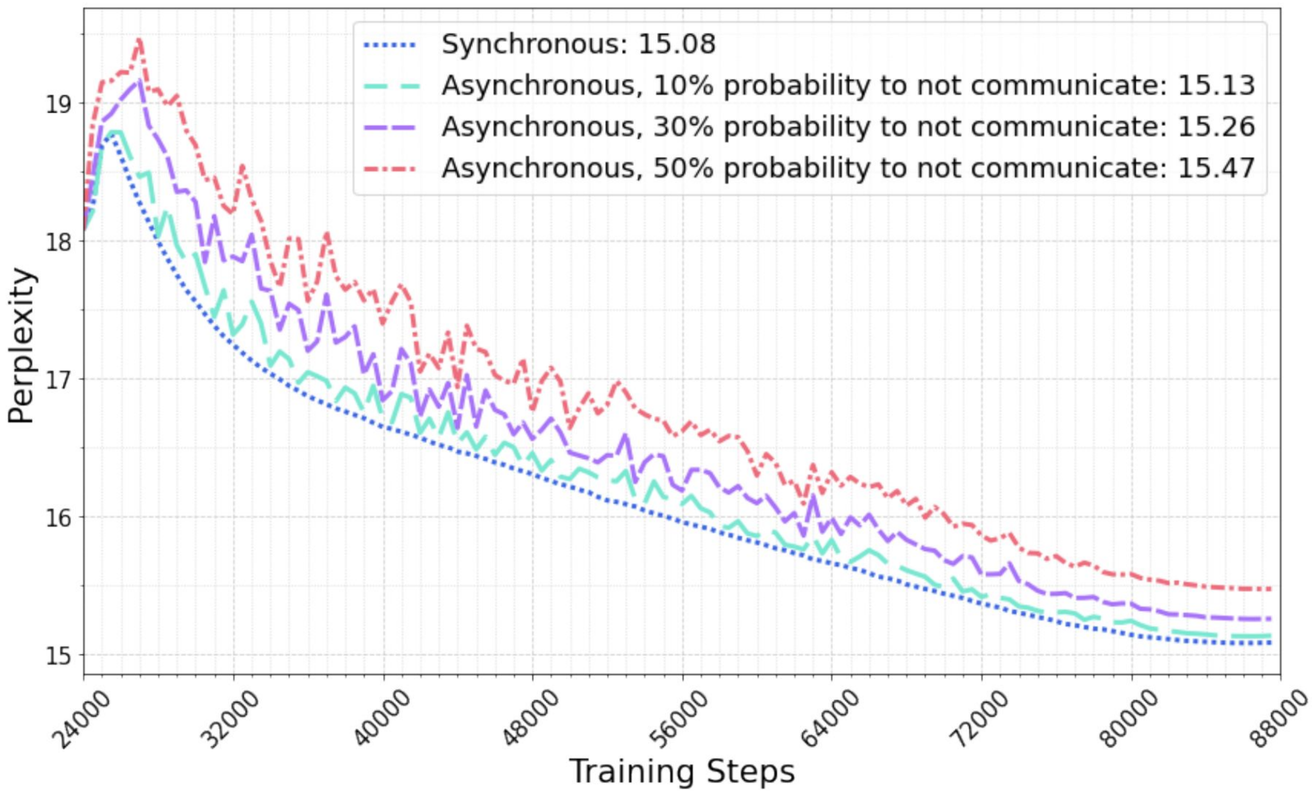
DiLoCo's model sizes

Model Size	Relative (%)	Absolute (PPL)
60M	4.33%	1.01
150M	7.45%	1.21
400M	7.49%	1.01

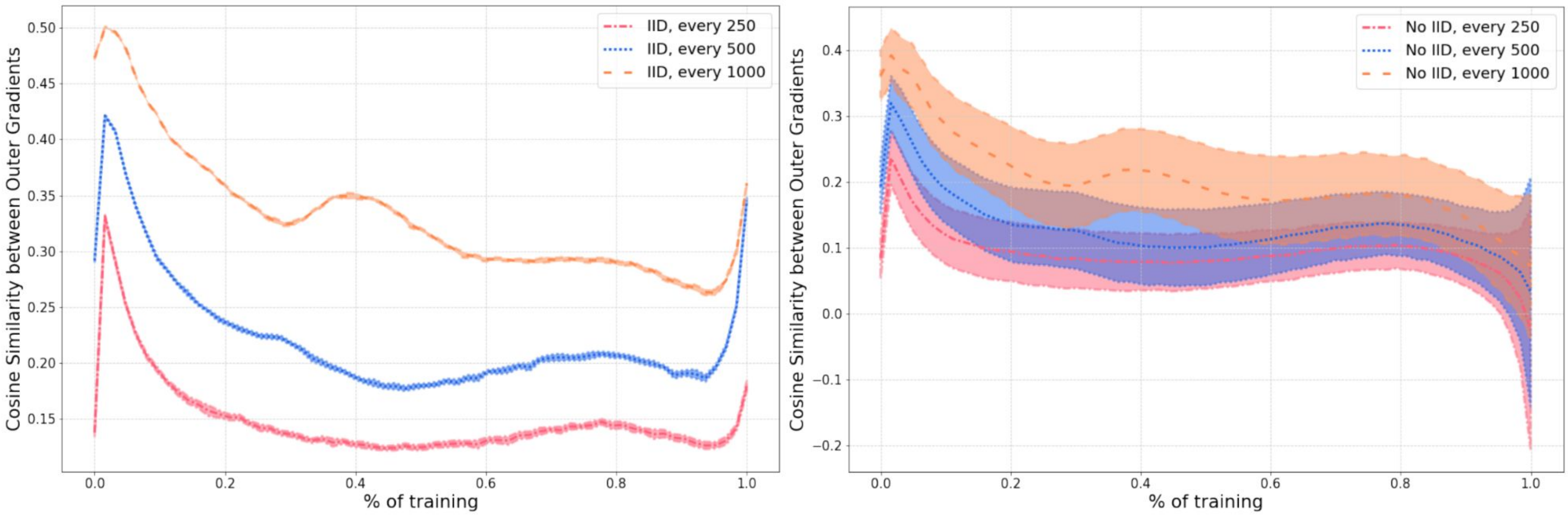
DiLoCo's outer optimizers



DiLoCo’s dropping communication



DiLoCo's outer gradient cosine similarity



(a) **i.i.d.** data regime.

(b) **non-i.i.d.** data regime.

DiLoCo's sparsification of outer gradients

% of pruned values	Perplexity	Relative change
0%	15.02	0%
25%	15.01	-0.06%
50%	15.08	+0.39%
75%	15.27	+1.66%