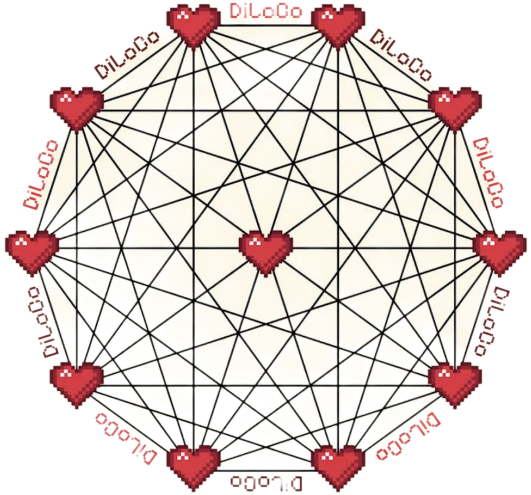
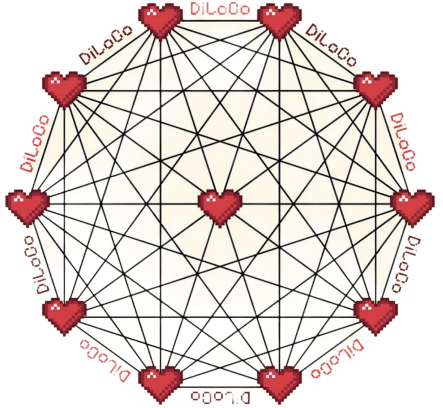
AIMS - 2025

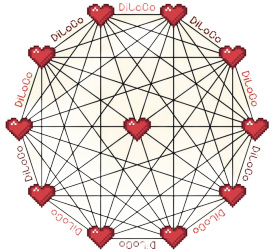
Diloco

Andrei A. Rusu

11/04/2025







Goal

At the end of this lecture you will know ...

how to quickly train an LLM using data-centers across the map

Outline

Motivation

Basic Diloco

Scaling Laws

Async Local-SGD

Streaming DiLoCo

Open challenges

Motivation

The scale and costs of cutting-edge AI data centers are hard to believe.

Motivation

xAI are planning to expand Colossus data centre to 400,000 *collocated GPUs*,

requiring 1.2 gigawats of power.

Alternatives? 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” together adds up to a lot of compute.

Outline

Motivation

Basic Diloco

Scaling Laws

Async Local-SGD

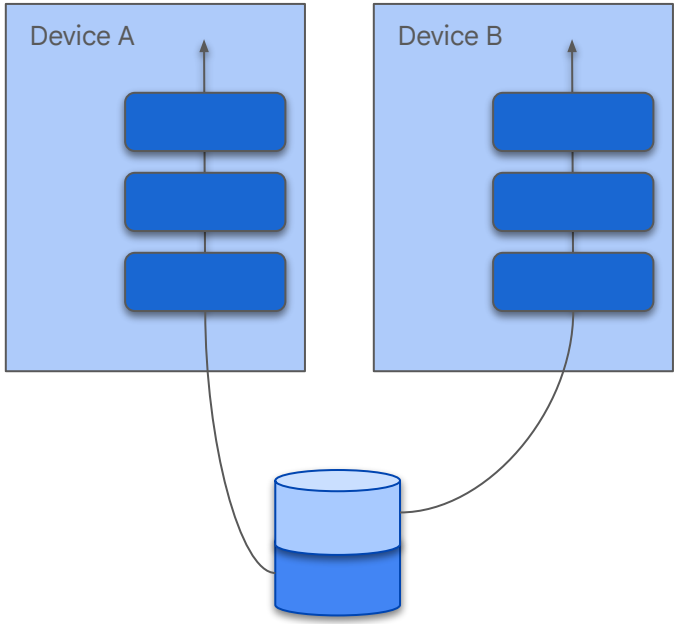
Streaming DiLoCo

Open challenges

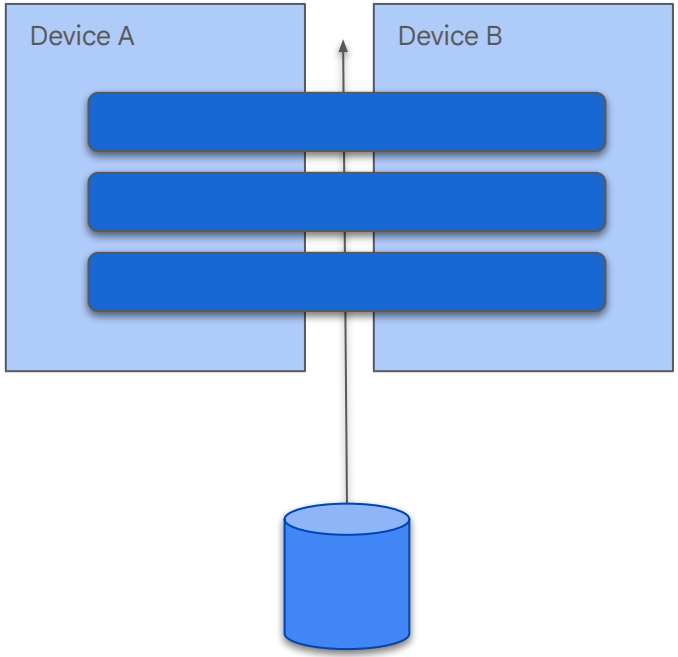
Recap

Parallelism Flavors

Data Parallelism

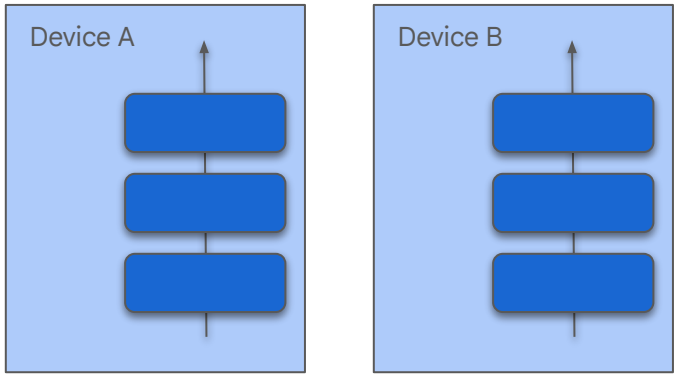


Tensor Parallelism

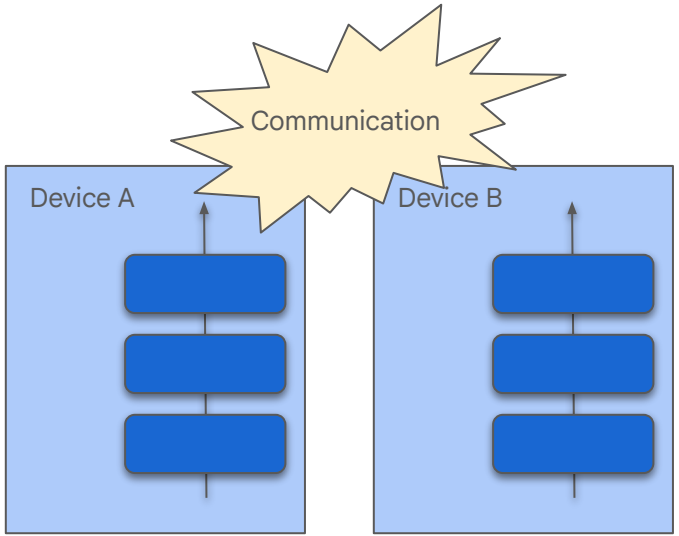


Parallelism Flavors

Data Parallelism



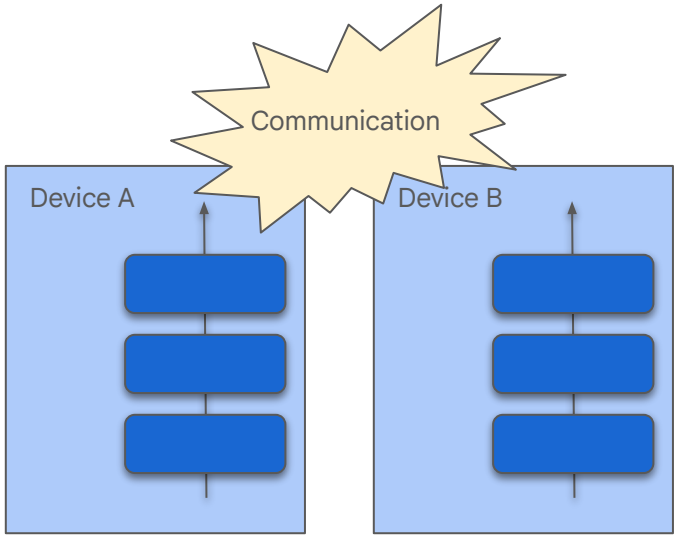
Parallelism Flavors



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



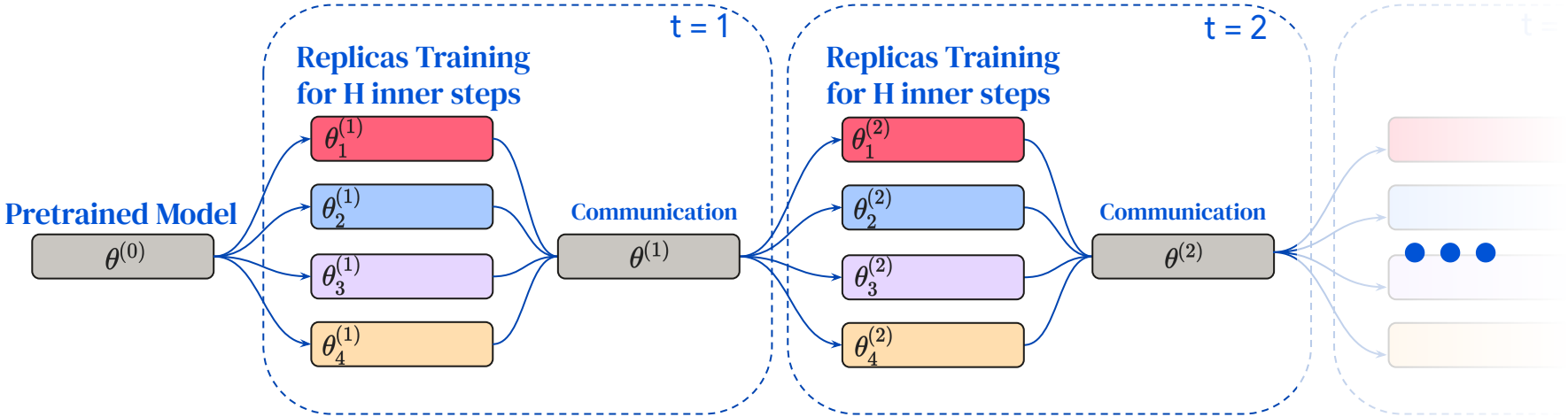
⚠️ **Problem!**

Communication at every training step

Hard to scale to non co-located devices

Communicating less ⇒ Independent Optimization

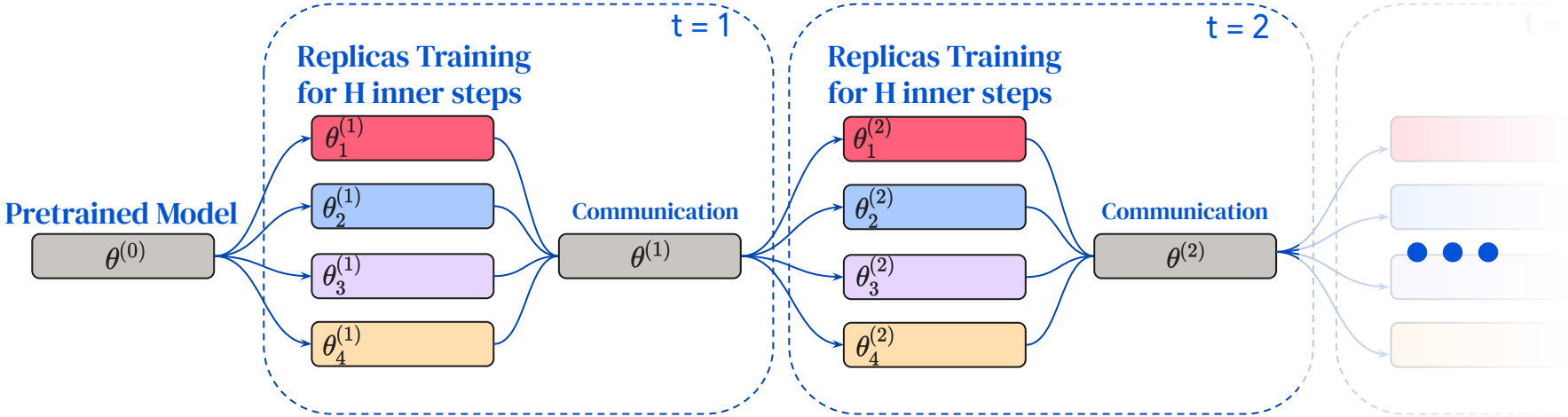
The lessons of Federated Learning:
World-wide distributed training should communicate less often.
Less communication requires some form of *independent optimization*.



Diloco Training Paradigm

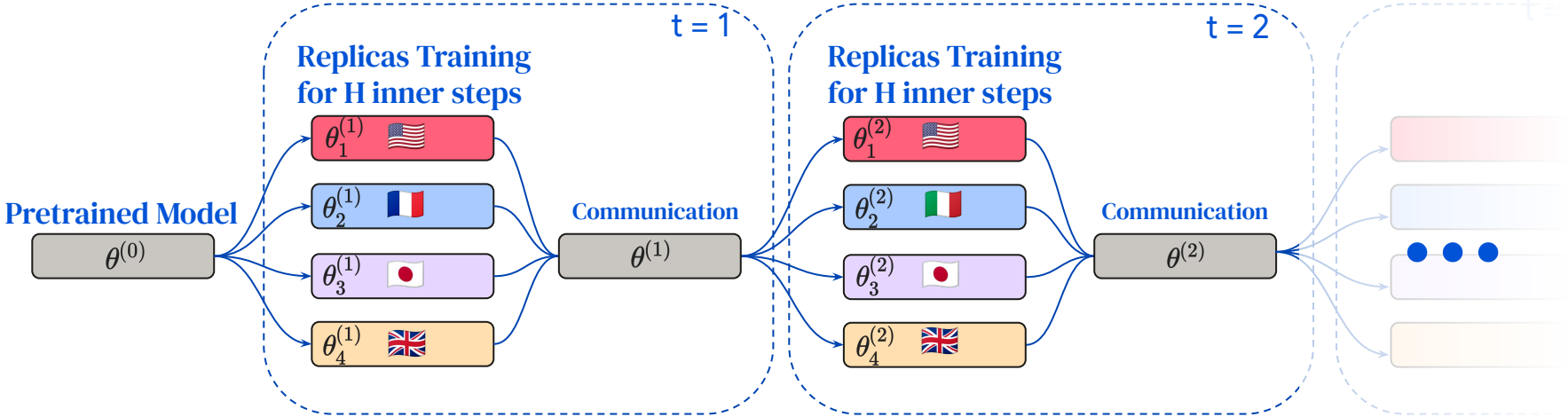
Our Assumptions

1. Each node can fit the entire model.
2. All nodes train a copy of the latest global model for H steps.
3. All nodes synchronize quickly to compute a new global model.
4. Overtraining regime, large-batch setting relative to model size.



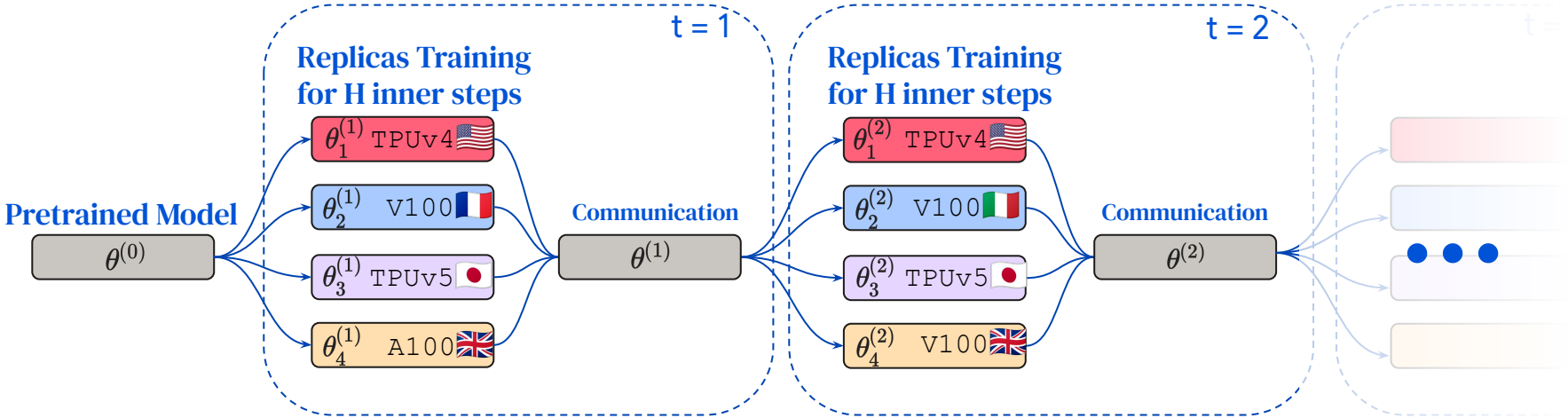
Our findings (TL;DR):

Independent *local* optimization with infrequent *global* communication effectively trains LLMs cross-country.



Our findings (TL;DR):

Fun fact:
We can even make use of 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 $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$
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

The recipe 🍜

For every round of training-communication:

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`

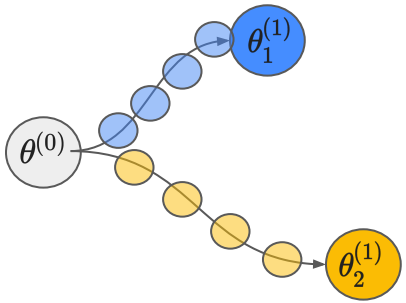
for `outer step` $t = 1 \ldots T$ **do**
  for `worker` $i = 1 \ldots k$ **do**
    $\theta_i^{(t)} \leftarrow \theta^{(t-1)}$
    for `inner step` $h = 1 \ldots H$ **do**
      $x \sim \mathcal{D}_i$
      $\mathcal{L} \leftarrow f(x, \theta_i^{(t)})$
      ▷ Inner optimization:
      $\theta_i^{(t)}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

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`

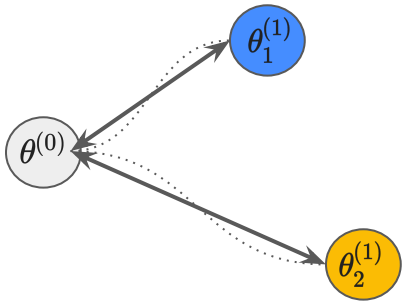
for `outer step` $t = 1 \ldots T$ **do**
  for `worker` $i = 1 \ldots k$ **do**
    $\theta_i^{(t)} \leftarrow \theta^{(t-1)}$
    for `inner step` $h = 1 \ldots H$ **do**
      $x \sim \mathcal{D}_i$
      $\mathcal{L} \leftarrow f(x, \theta_i^{(t)})$
      ▷ Inner optimization:
      $\theta_i^{(t)}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

The recipe 🍜

Afterwards, compute “outer gradients” per node.

While usually a gradient is an infinitesimal difference,

“outer gradients” are deltas in parameter space,
across hundreds or thousands of *inner* training steps.



Algorithm 1 DiLoCo Algorithm

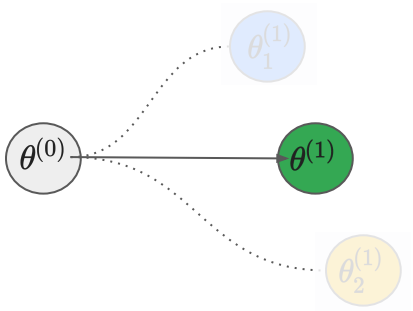
Require: Initial model $\theta^{(0)}$
Require: k workers
Require: Data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$
Require: Optimizers `InnerOpt` and `OuterOpt`

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:
      $\theta_i^{(t)} \leftarrow \texttt{InnerOpt}(\theta_i^{(t)}, \nabla_{\mathcal{L}})$
    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

The recipe 🍜

What if we averaged "outer gradients" across nodes to compute a *global* pseudo-gradient and plugged it into an "**outer optimizer**"?

If the outer optimizer is `SGD(lr=1.0)`, we recover classic **model averaging (FedAvg).**



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`

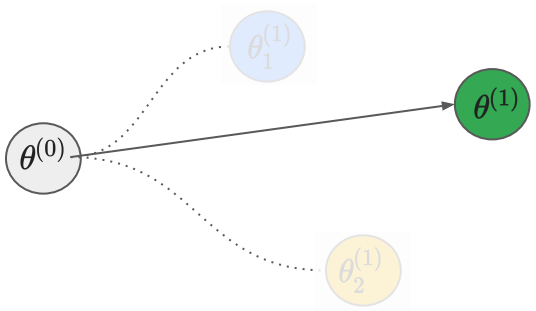
for `outer step` $t = 1 \ldots T$ **do**
  for `worker` $i = 1 \ldots k$ **do**
    $\theta_i^{(t)} \leftarrow \theta^{(t-1)}$
    for `inner step` $h = 1 \ldots H$ **do**
      $x \sim \mathcal{D}_i$
      $\mathcal{L} \leftarrow f(x, \theta_i^{(t)})$
      ▷ Inner optimization:
      $\theta_i^{(t)}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

The recipe 🍜

What if we averaged "outer gradients" across nodes to compute a *global* pseudo-gradient and plugged it into an "**outer optimizer**"?

However, we have more options here ...

Indeed, we find that `SGD(lr=1.0, momentum=0.9, nesterov=True)`, can significantly **improve generalisation**.



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

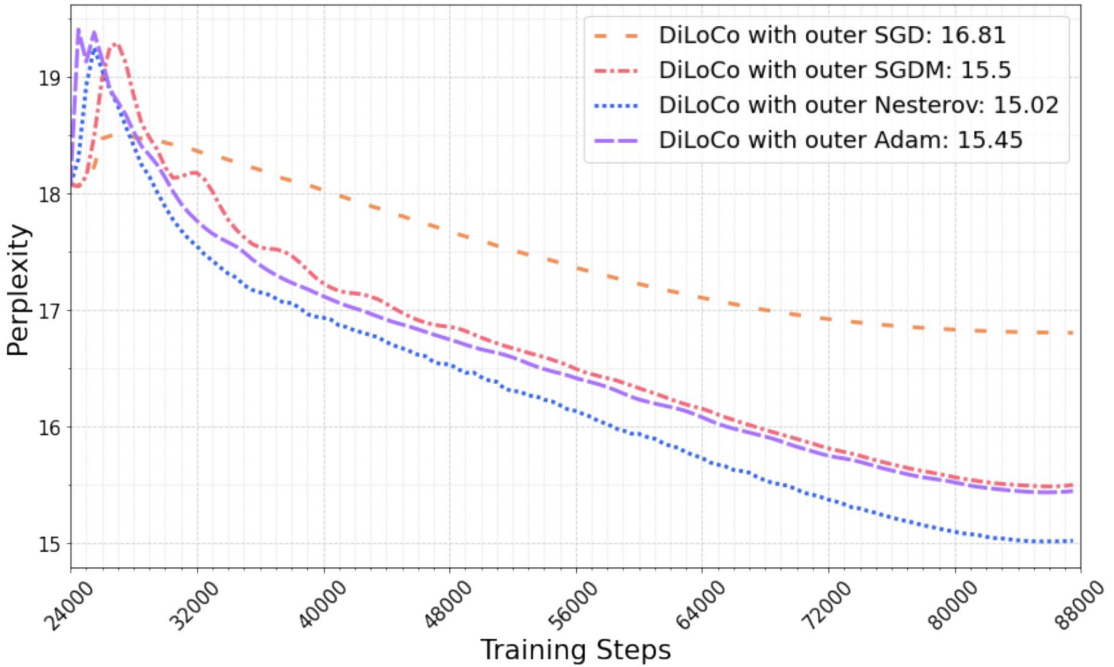
DiLoCo's outer optimizers

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

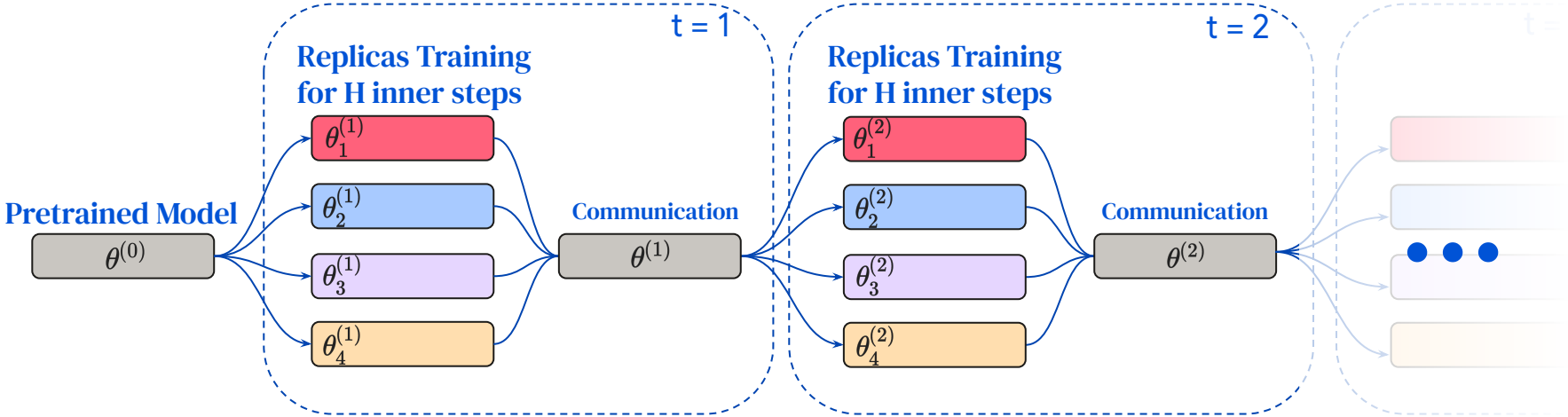
We found empirically that Nesterov:

- Leads to better generalization
- Is more stable across HPs
- Scales to O(100) inner steps, while previous literature uses O(10).



Our Assumptions

1. Each node can fit the entire model.
2. All nodes train a copy of the latest global model for H steps.
3. All ? nodes synchronize quickly to compute a new global model.
4. Overtraining regime, large-batch setting for model size.

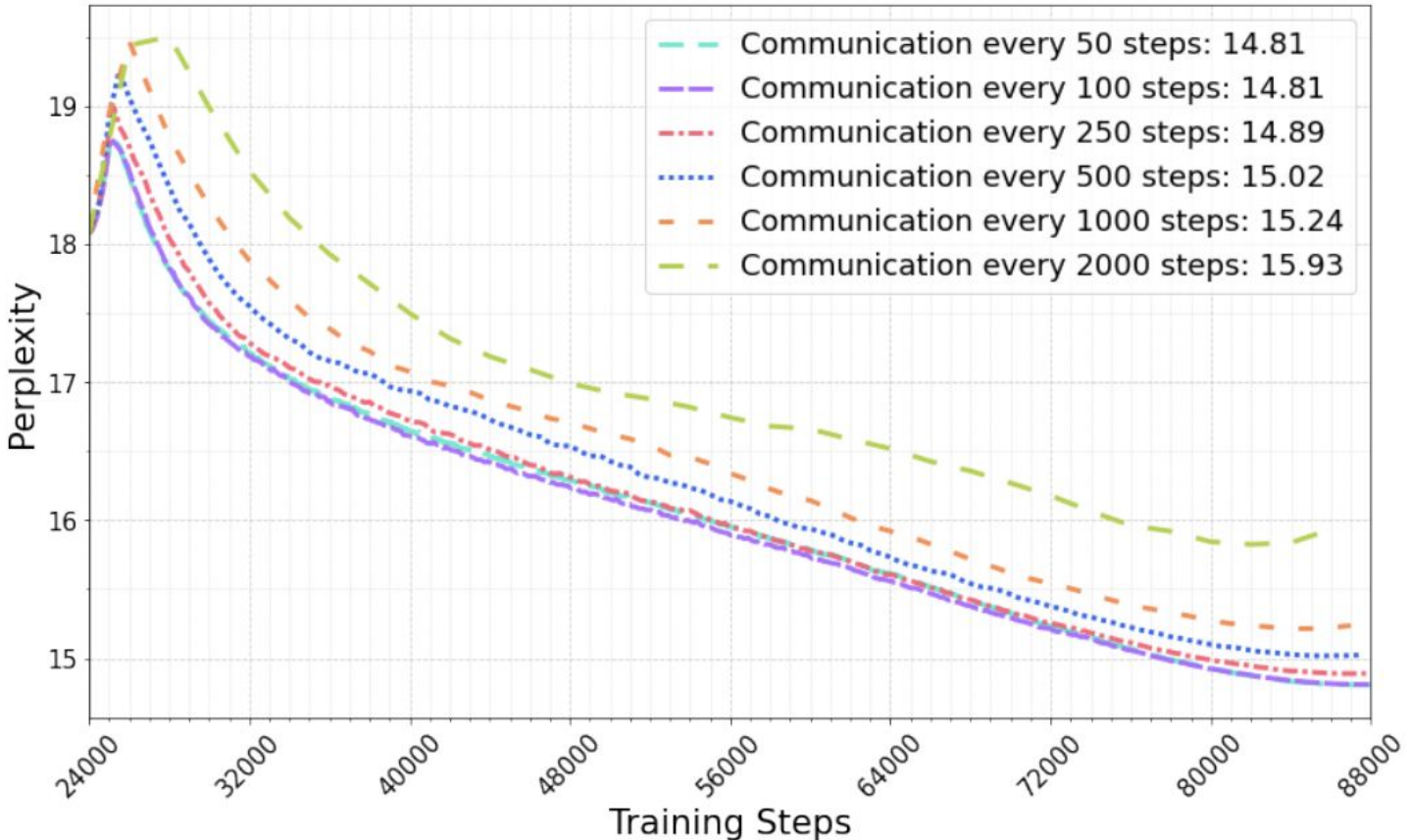


Resiliency to communication frequency

If bandwidth is small, reduce communication.

Amortize time spent in synchronization!

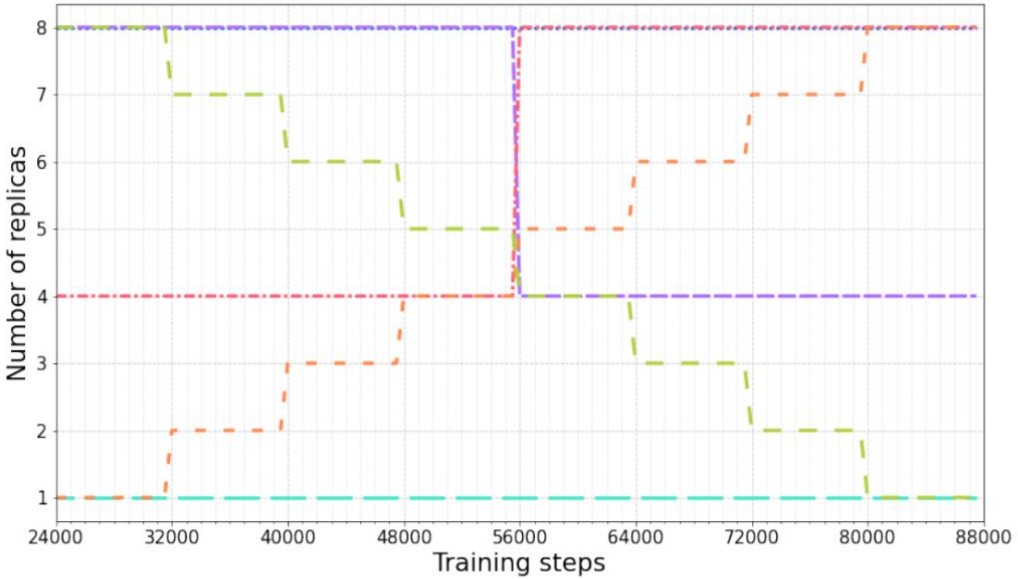
Typical Federated Learning set up: O(10) inner steps. We work with one or two orders of magnitude more.



Resiliency to compute availability

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

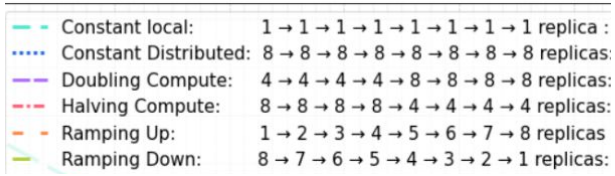
We tried varying the number of workers across time...



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...



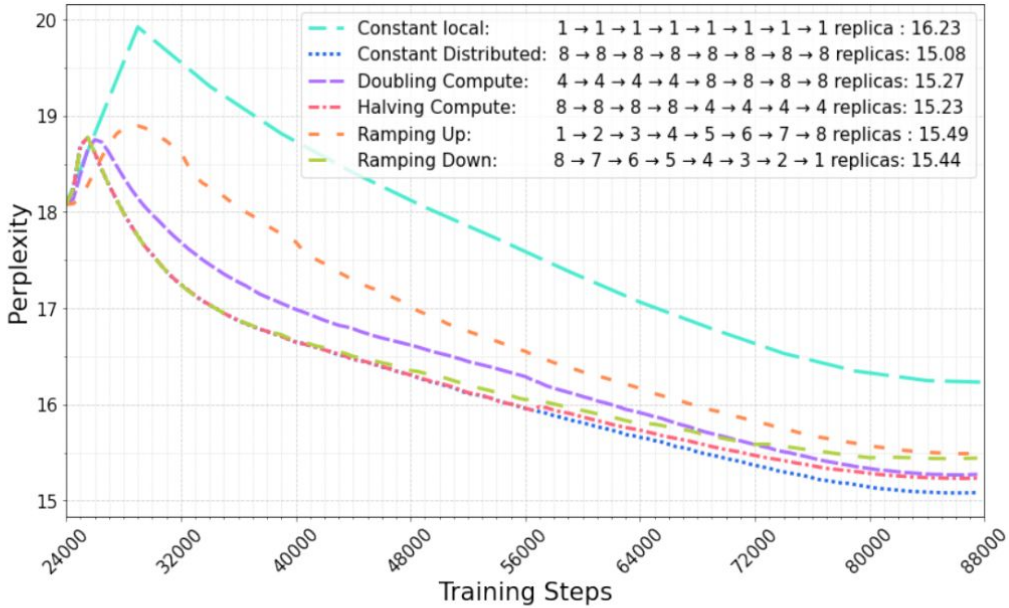
Number of replica per phase.

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

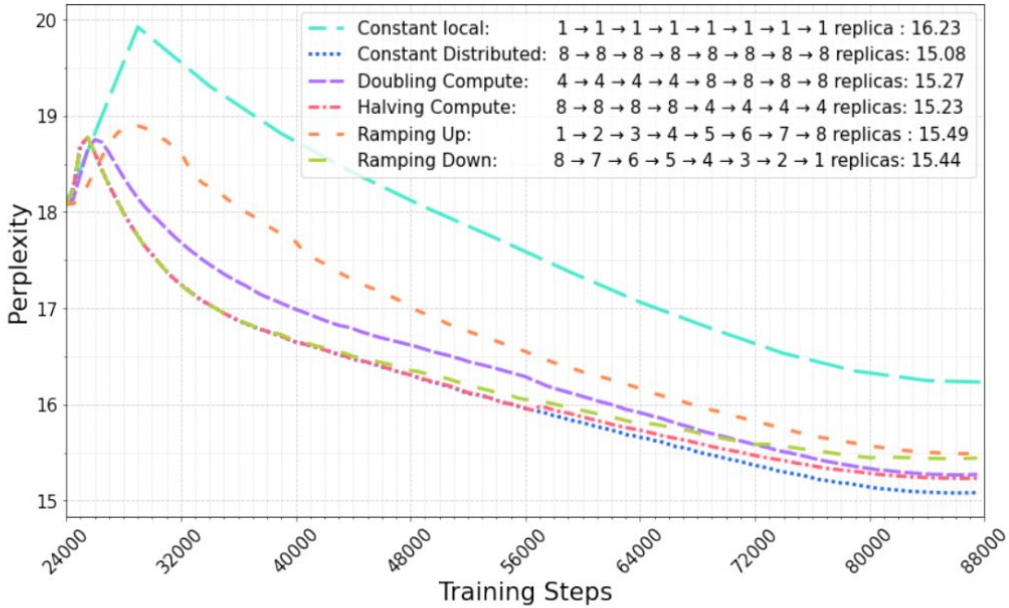


Resiliency to compute availability

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.

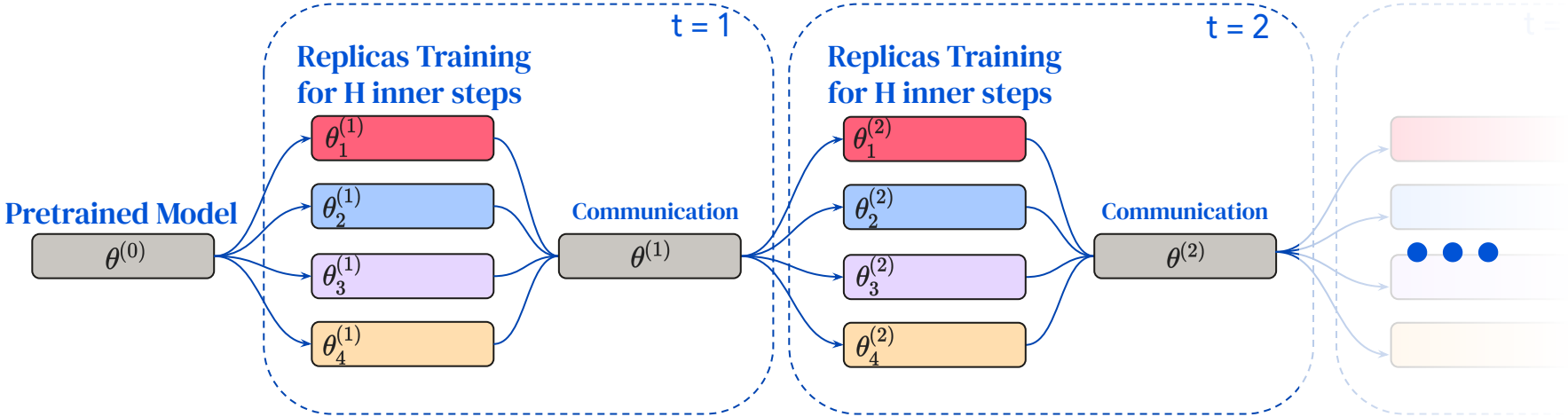
However, there is still, infrequently, a communication cost that can be problematic.

→ we experiment **compressing** *outer gradients* by pruning 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%

Our Assumptions

1. Each node can fit the entire model.
2. All nodes train a copy of the latest global model for H steps.
3. All nodes synchronize quickly to compute a new global model.
4. Overtraining regime, large-batch setting for model size?



Model Sizes

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

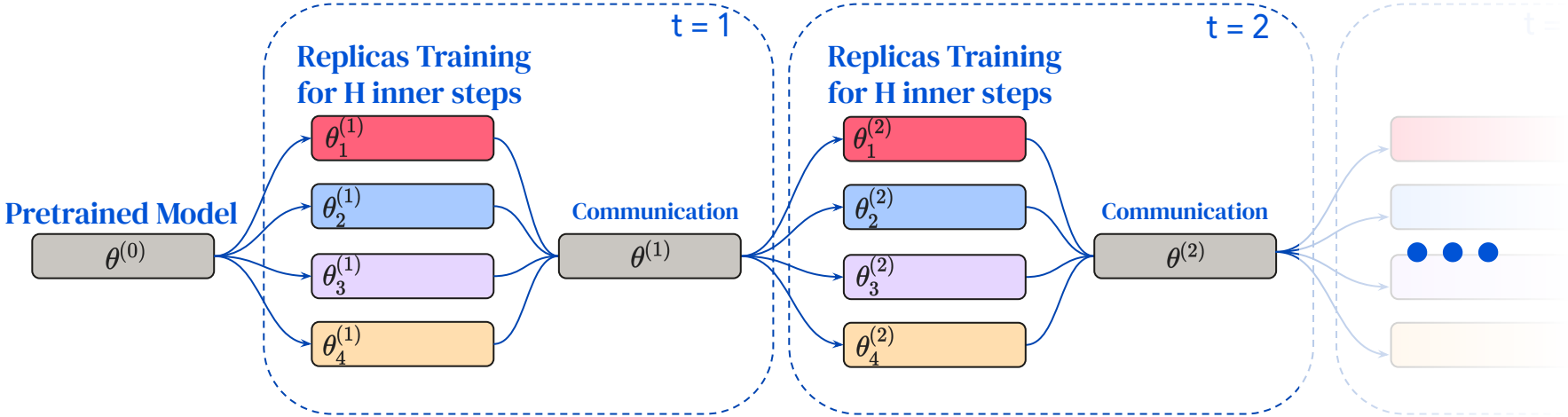
Preliminary experiments up to 400M scale indicates that it may be true for DiLoCo too ⇒ We may be good here!

Table 4. Varying the model size: For each model size, we report the relative and absolute perplexity (PPL) improvements of DiLoCo over the baseline (using a single worker). DiLoCo uses 8 workers and non-i.i.d. shards.

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

Our Assumptions

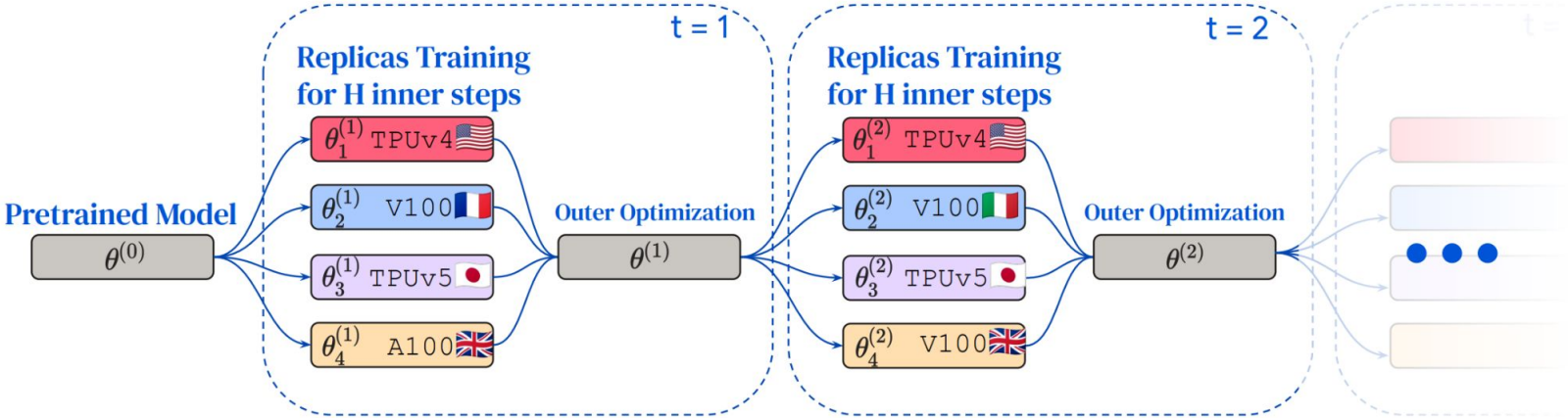
1. Each node can fit the entire model.
2. All nodes train a copy of the latest global model for H steps.
3. All nodes synchronize quickly ? to compute a new global model.
4. Overtraining regime, large-batch setting for model size.



But we are waiting for laggers... 🐢 vs 🐇

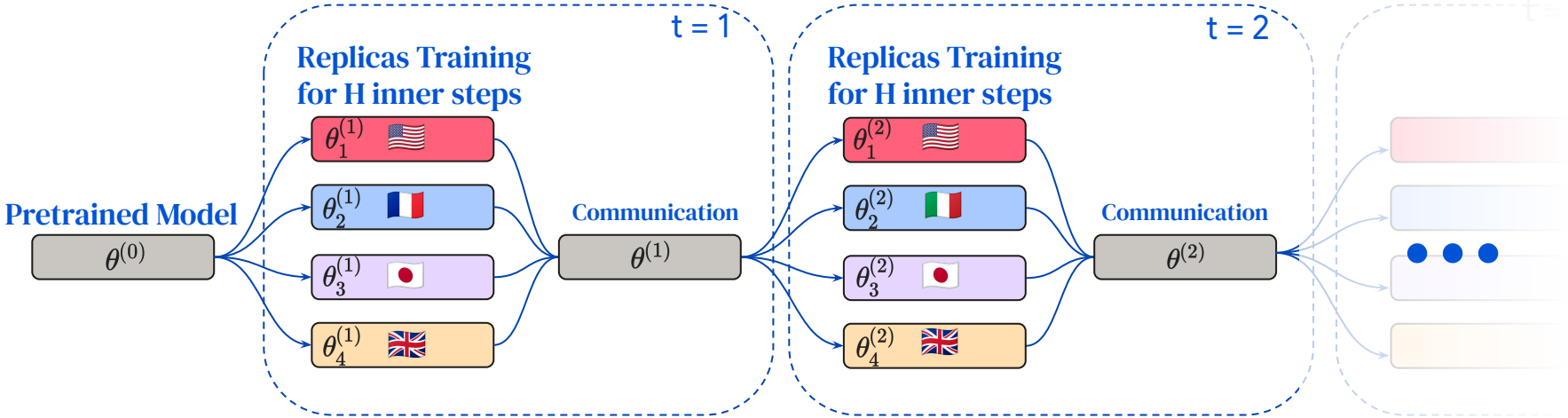
A100 is twice faster than V100.

Being distributed is cool, but if we have to wait for laggers that's a bit sad ⇒ More work needed!



Conclusions

Independent ***local*** optimization with infrequent ***global*** communication effectively (over-)trains (smallish) LLMs cross-country.



Outline

Motivation

Basic Diloco

Scaling Laws

Async Local-SGD

Streaming DiLoCo

Open challenges

Outline

What is DiLoCo?

Methodology

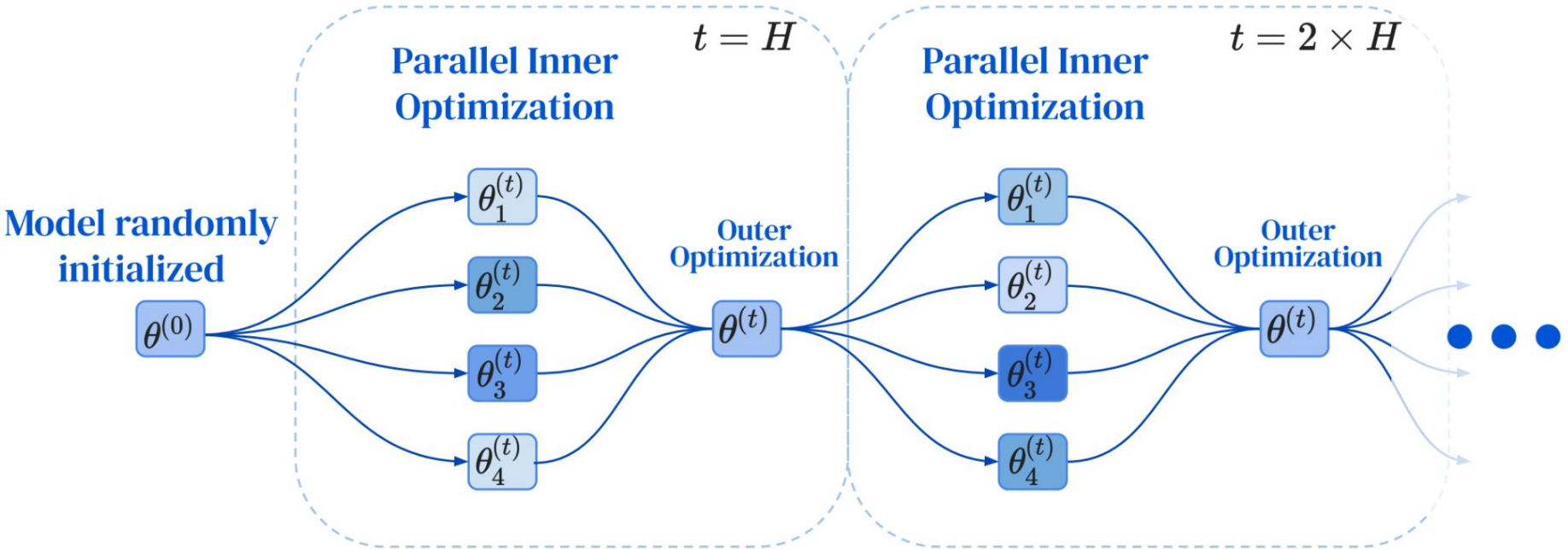
Findings

Ablations

Scaling Laws

Google Research

DiLoCo



Google Research

DiLoCo

Algorithm 1 DiLoCo

Require: Loss function $f(\theta, x)$, number of replicas M, synchronization cadence H
Require: Initial model weights $\theta^{(0)}$, data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_M\}$
Require: Optimizers `InnerOpt` and `OuterOpt`

$\forall m, \theta_m^{(0)} \leftarrow \theta^{(0)}$
for step $t = 1 \ldots T$ **do**
  parallel for replica $m = 1 \ldots M$ **do**
    $x_m^{(t)} \sim \mathcal{D}_m$
    $g_m^{(t)} \leftarrow \nabla_\theta f(\theta_m^{(t-1)}, x_m^{(t)})$
    $\theta_m^{(t)} \leftarrow \texttt{InnerOpt}(\theta_m^{(t-1)}, g_m^{(t)})$
  end parallel for

  if $t \bmod H = 0$ **then**
    $\Delta_m^{(t)} \leftarrow \theta^{(t-H)} - \theta_m^{(t)}$
    $\Delta^{(t)} \leftarrow \frac{1}{M} \sum_{m=1}^{M} \Delta_m^{(t)}$
    $\theta^{(t)} \leftarrow \texttt{OuterOpt}(\theta_m^{(t-H)}, \Delta^{(t)})$
    $\forall m, \theta_m^{(t)} \leftarrow \theta^{(t)}$

Each of the M model replicas does a step of data-parallel training via **InnerOpt**

Every H steps, compute “outer gradients” in parameter space, and do a global model update via **OuterOpt**

Google Research

Historical Note

FedOpt - for communication-efficient federated learning [Reddi et al., 2021]

Algorithm 1 FEDOPT

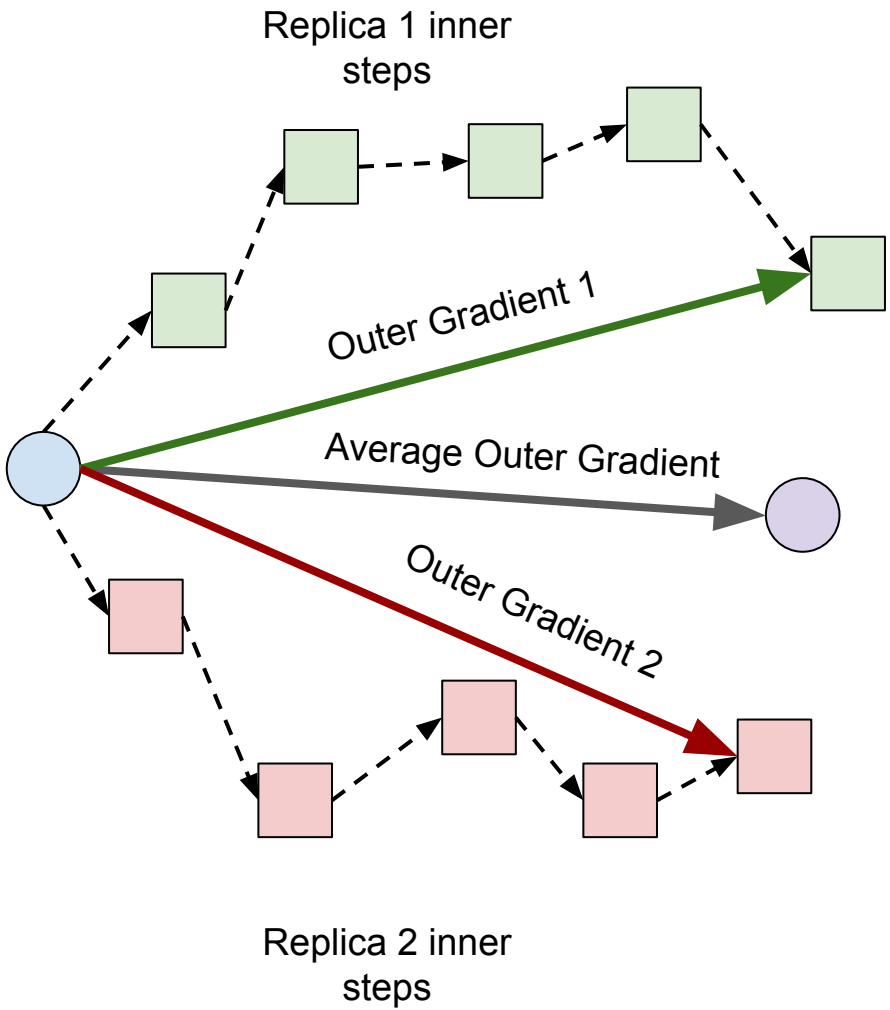
Input: x_0, CLIENTOPT, SERVEROPT
for $t = 0, \cdots, T-1$ **do**
  Sample a subset $\mathcal{S}$ of clients
  $x_{i,0}^t = x_t$
  for each client $i \in \mathcal{S}$ **in parallel do**
    for $k = 0, \cdots, K-1$ **do**
      Compute an unbiased estimate $g_{i,k}^t$ of $\nabla F_i(x_{i,k}^t)$
      $x_{i,k+1}^t = \text{CLIENTOPT}(x_{i,k}^t, g_{i,k}^t, \eta_l, t)$
  $\Delta_i^t = x_{i,K}^t - x_t$
  $\Delta_t = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \Delta_i^t$
  $x_{t+1} = \text{SERVEROPT}(x_t, -\Delta_t, \eta, t)$

Key difference: DiLoCo assumes fixed replicas and carries over optimizer state

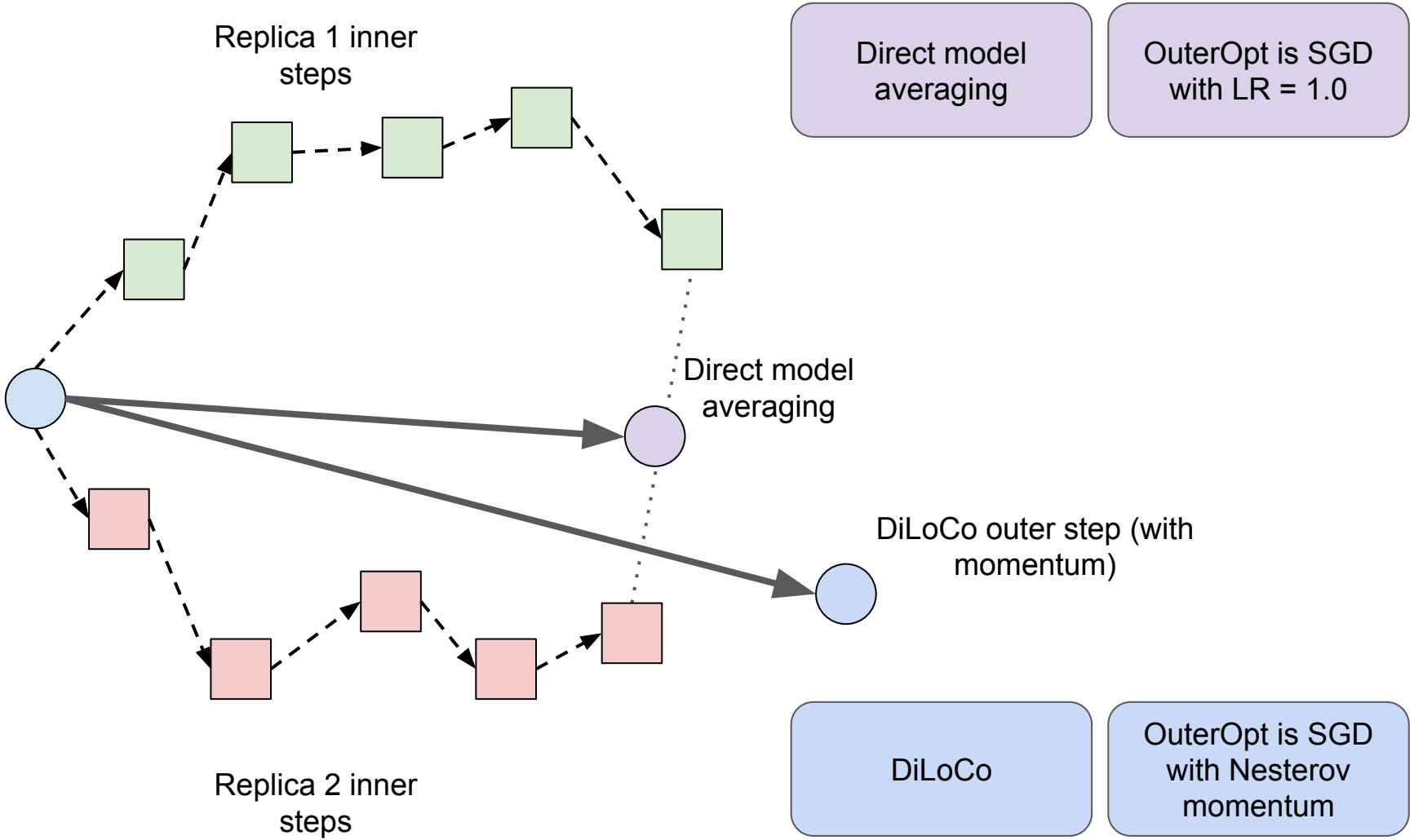
DiLoCo - for communication-efficient datacenter training [Douillard et al., 2023]

Algorithm 1 DiLoCo Algorithm

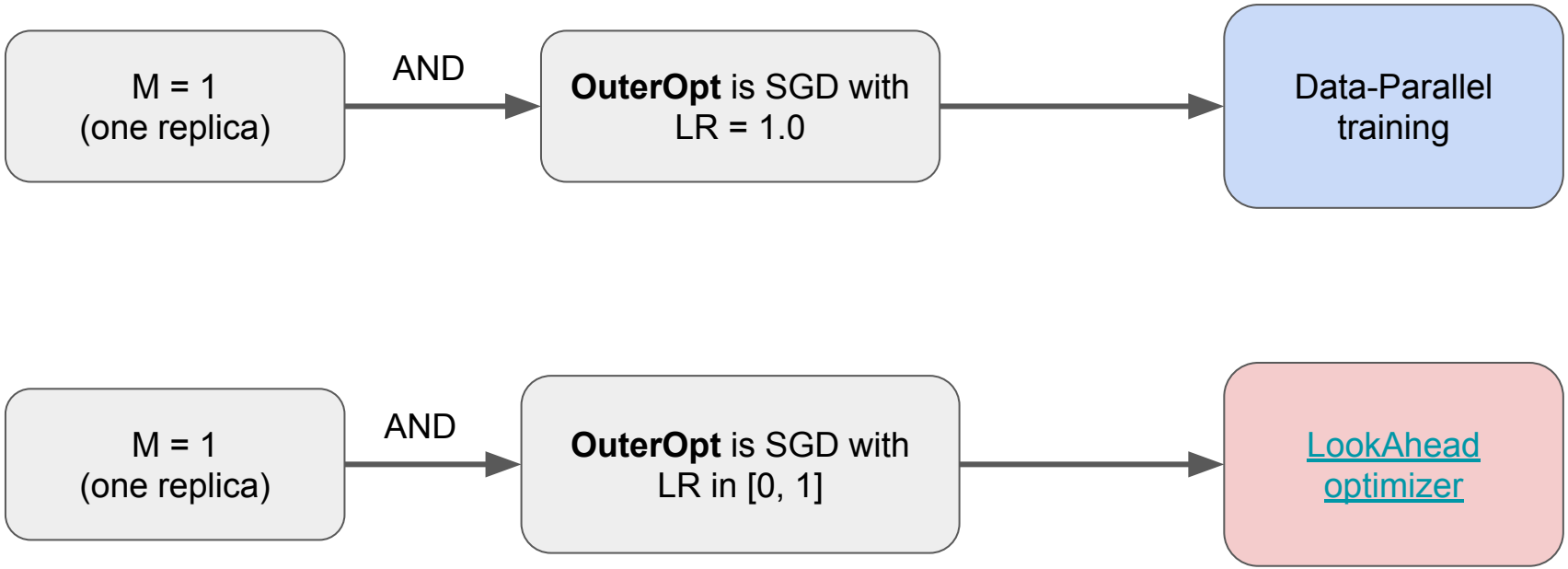
Require: Initial model $\theta^{(0)}$
Require: k workers
Require: Data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_k\}$
Require: Optimizers `InnerOpt` and `OuterOpt`
for `outer step` $t = 1 \ldots T$ **do**
  for `worker` $i = 1 \ldots k$ **do**
    $\theta_i^{(t)} \leftarrow \theta^{(t-1)}$
    for `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



Google Research

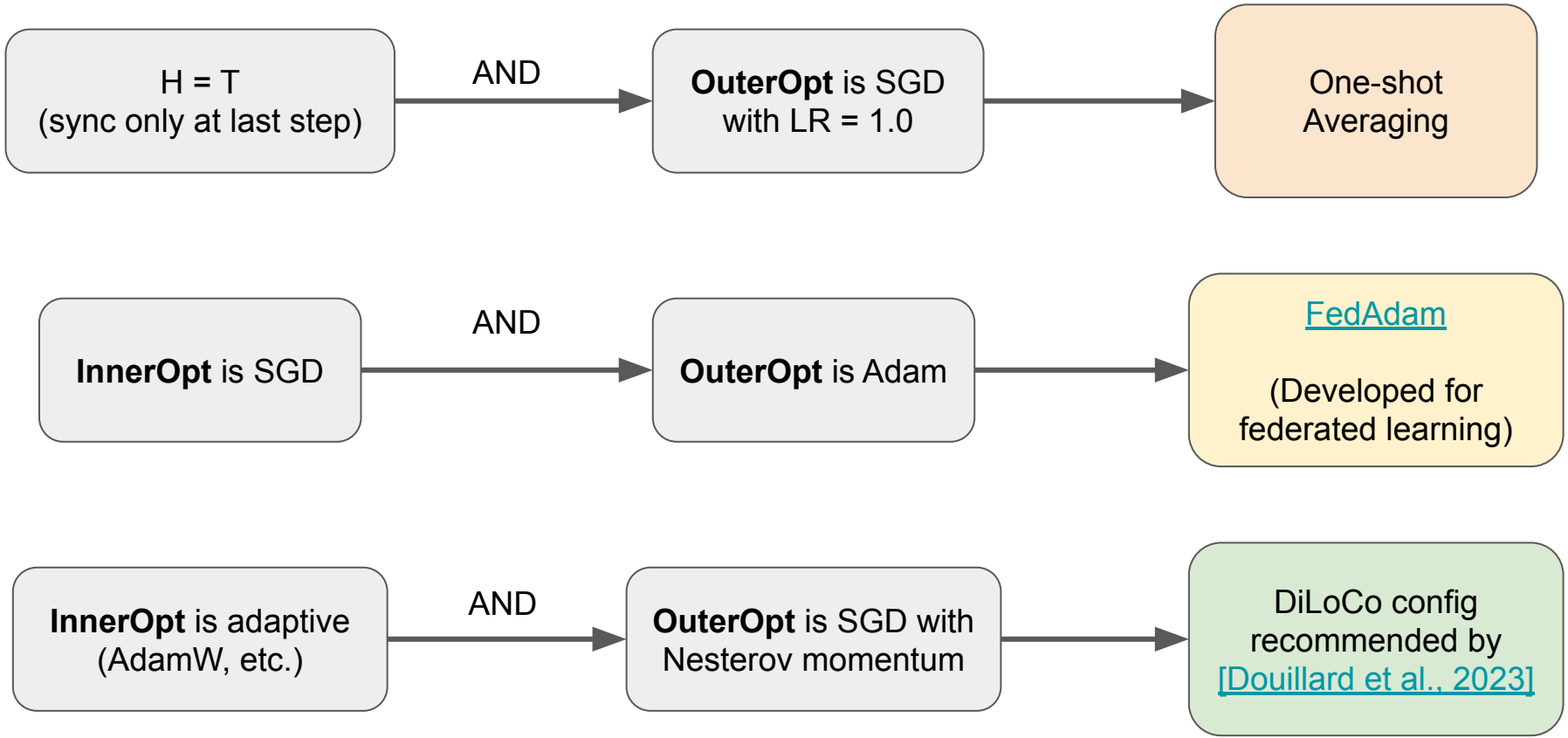

Replica 1 inner steps
Direct model averaging
OuterOpt is SGD with LR = 1.0
Direct model averaging
DiLoCo outer step (with momentum)
Replica 2 inner steps
DiLoCo
OuterOpt is SGD with Nesterov momentum

DiLoCo, special cases



Google Research

DiLoCo, special cases



Motivating questions

1. How do Data-Parallel and DiLoCo compare for varying model size N and number of replicas M?
   Key comparison: Data-Parallel versus DiLoCo with M = 1

2. How should shared hyperparameters (inner learning rate, batch size) change between Data-Parallel & DiLoCo?

3. DiLoCo has extra hyperparameters (e.g. outer learning rate). How do we make them easier to tune?

4. How are the above altered by the **synchronization cadence** H or **overtraining**?

5. When fitting scaling laws for loss L, should we fit L(N) for each value of M (**independent fit**), or jointly fit L(N, M) (**joint fit**)?

Methodology

General approach: Train models of different sizes with Data-Parallel & DiLoCo, using Chinchilla-optimal numbers of tokens.

- For DiLoCo, vary the number of model replicas M
- Do extensive tuning of learning rates & batch sizes on models up to 2.4B scales
- Extrapolate to 4B and 10B models
- QK-layer norm and z-loss for stability

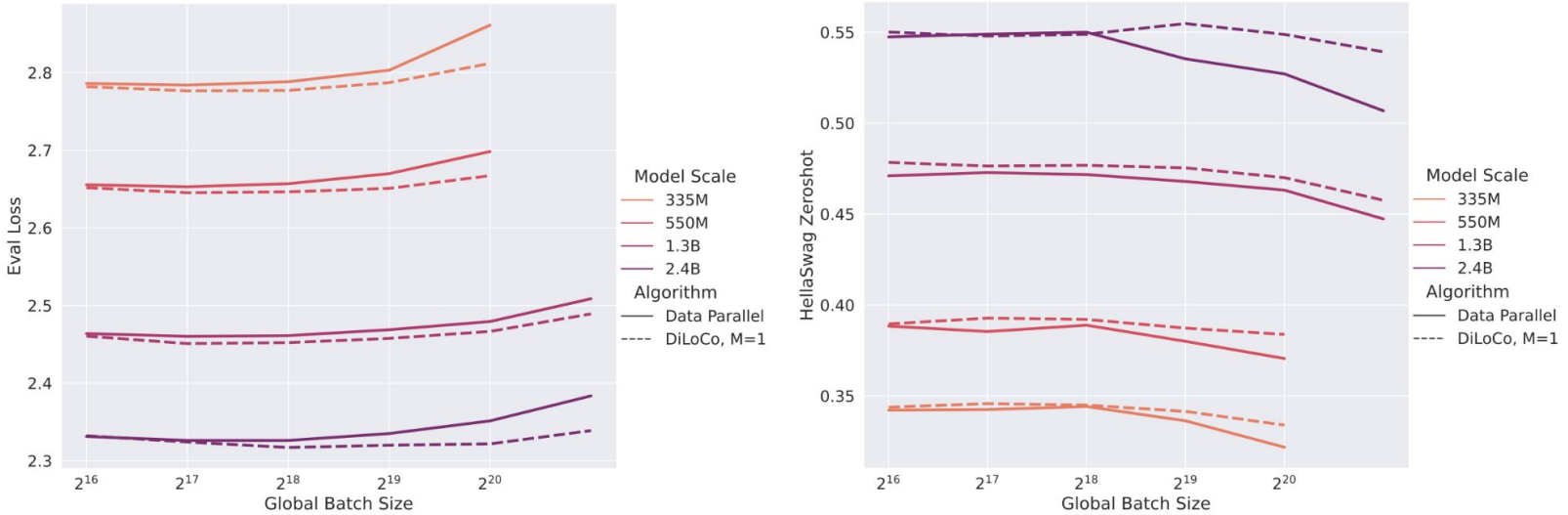
Model Scale	Transformer Layers	Attention Heads	QKV Dimension	Hidden Dimension	Token Budget	Hyperparameter Sweep
35M	6	8	512	2,048	70M	✓
90M	9	12	768	3,072	1.8B	✓
180M	12	16	1,024	4,096	3.6B	✓
330M	15	20	1,280	5,120	6.6B	✓
550M	18	24	1,536	6,144	11B	✓
1.3B	24	32	2,048	8,192	26B	✓
2.4B	30	40	2,560	10,240	48B	✓
4B	36	48	3,072	12,288	80B	✗
10B	48	64	4,096	16,384	200B	✗

Methodology

Details:

- Pre-train from scratch on C4 dataset (overtraining ablations on Dolma)
- Evaluate on C4 held-out and do zero-shot evals on HellaSwag, PIQA, and Arc-Easy
- Inner optimizer:
  - AdamW, set weight decay = 1/(number of training steps)
  - Warmup for 1000 steps, cosine decay to 5% of peak LR
  - Clip inner gradients to L2 norm of 1.0
- Outer optimizer:
  - SGD with Nesterov momentum, momentum of 0.9
  - Constant learning rate
- Typically, set H = 30
- Vary global batch size, inner LR, and outer LR until we find a clear local minima

Finding #1: Single Replica DiLoCo



(a) Evaluation loss.

(b) Zero-shot accuracy on HellaSwag.

Figure 2: **DiLoCo with $M = 1$ outperforms Data-Parallel**. We present the evaluation loss and downstream accuracy of Data-Parallel and DiLoCo with $M = 1$ for varying model and global batch sizes. In all settings, DiLoCo with $M = 1$ attains better performance, and that the gap increases for larger batch sizes.

Finding #2: Critical Batch Size

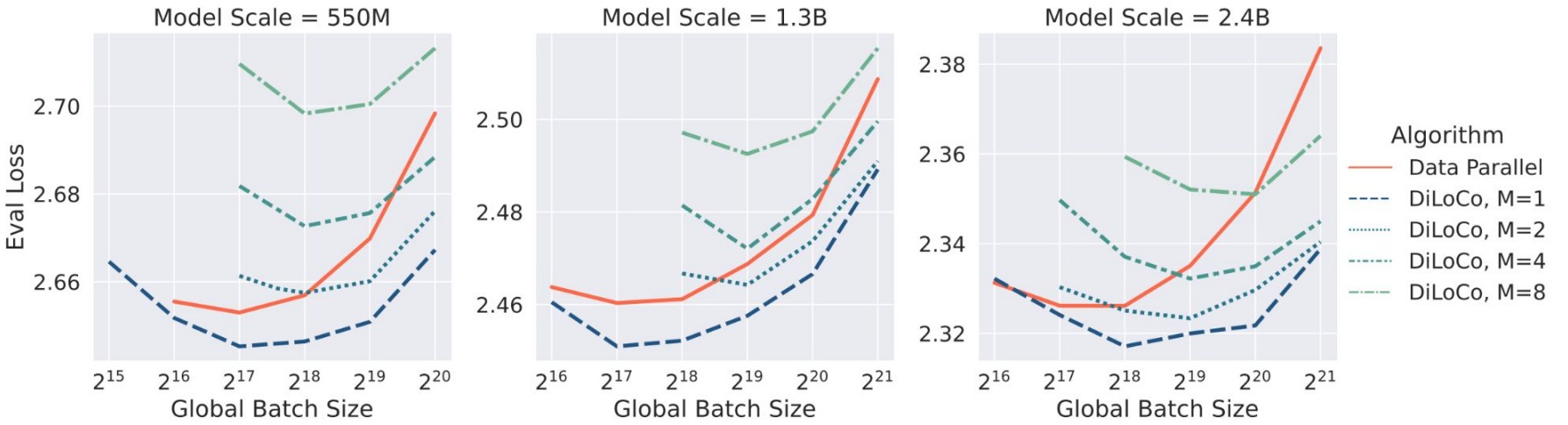


Figure 3: **DiLoCo increases critical batch size, part 1.** We present the evaluation loss of Data-Parallel and DiLoCo as a function of global batch size. For all values of M, DiLoCo exhibits significantly increased critical batch size than Data-Parallel. Moreover, the critical batch size increases as a function of M.

Finding #2: Critical Batch Size

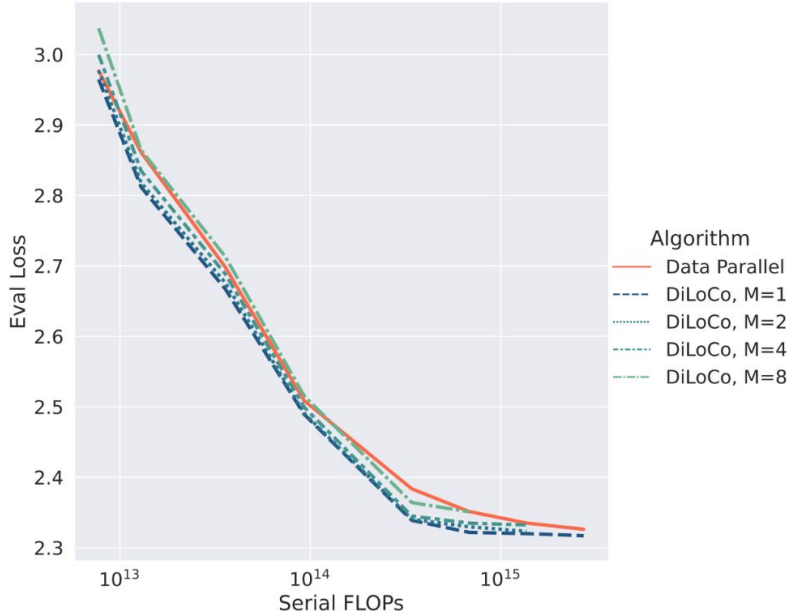
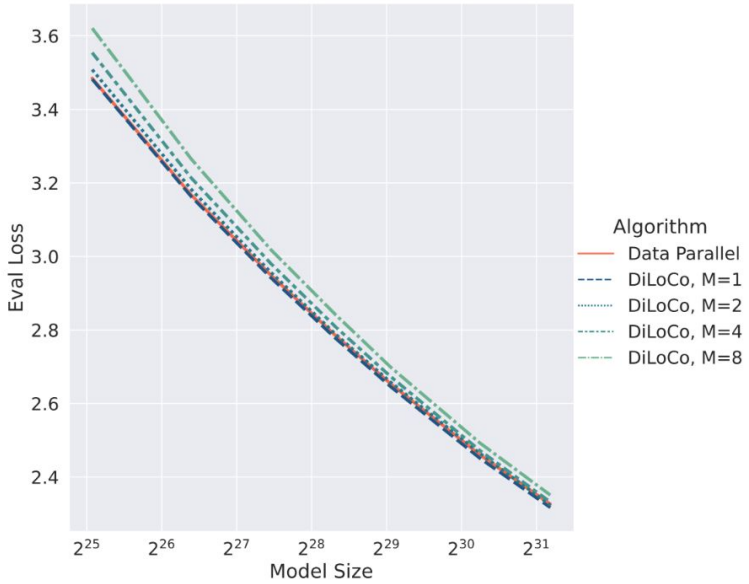


Figure 5: **DiLoCo speeds up training by leveraging horizontal scalability.** We plot loss with respect to serial FLOPs (C/B). For each algorithm, we plot those points across all model sizes and batch sizes, within 10% of the lower convex envelope. We see that DiLoCo can achieve comparable loss to Data-Parallel with many fewer serial FLOPs.

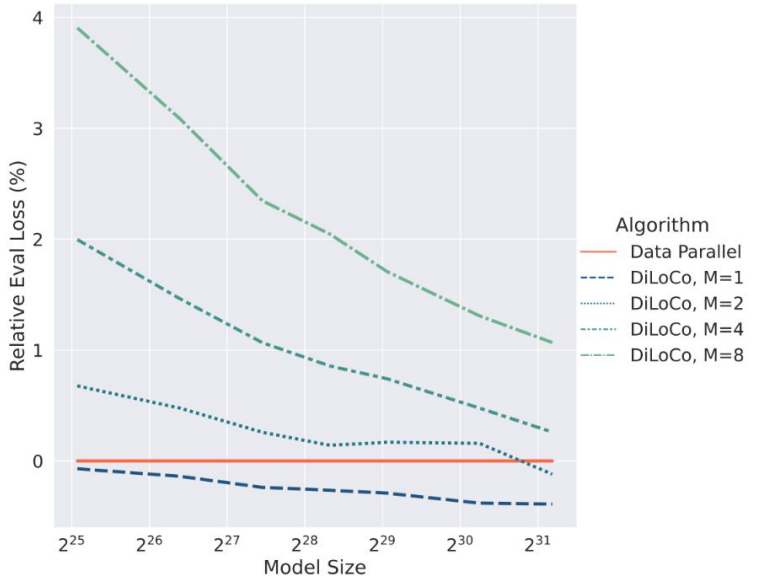
Finding #3: Outer Learning Rate

Figure 6: **Optimal outer learning rate is independent of model size.** Here we present the best outer learning rate for DiLoCo for varying M and model sizes. We select the best outer learning rate over $\{0.2, 0.4, 0.6, 0.8, 1.0\}$, optimizing over all other hyperparameters. For sufficiently large models, the best outer learning rate is clearly constant.

Finding #4: Scale



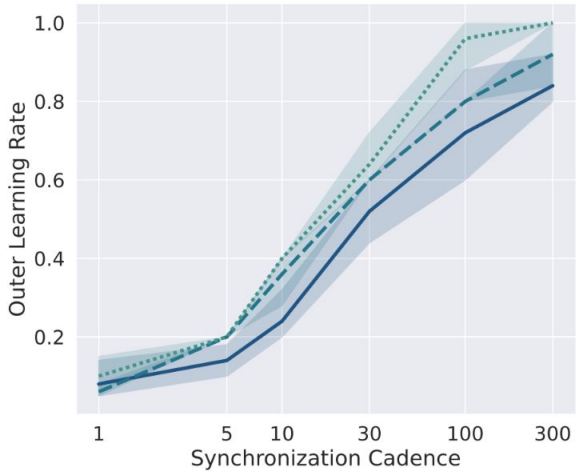
(a) Evaluation loss for various algorithms, as a function of N.



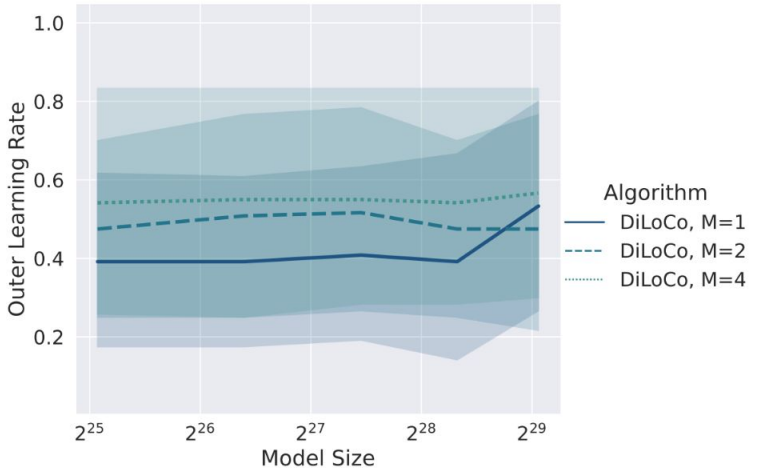
(b) Percentage difference in evaluation loss, relative to Data-Parallel.

Figure 7: **DiLoCo does better with scale.** We compare Data-Parallel to DiLoCo for varying model sizes N. For all M, DiLoCo improves monotonically wrt Data-Parallel as N increases.

Ablation #1 - Synchronization cadence



(a) Optimal outer learning rate for each synchronization cadence. Shaded regions represent the variance across model sizes.



(b) Optimal outer learning rate for each model size. Shaded regions represent the variance across synchronization cadences.

Figure 8: **Outer learning rate scales with M and H, not N.** The optimal outer learning rate η is a monotonically increasing function of synchronization cadence H and M (left), but essentially independent of model size N (right). As in Figure 6, this means that we can tune outer learning rate at smaller model scales.

Ablation #1 - Synchronization cadence

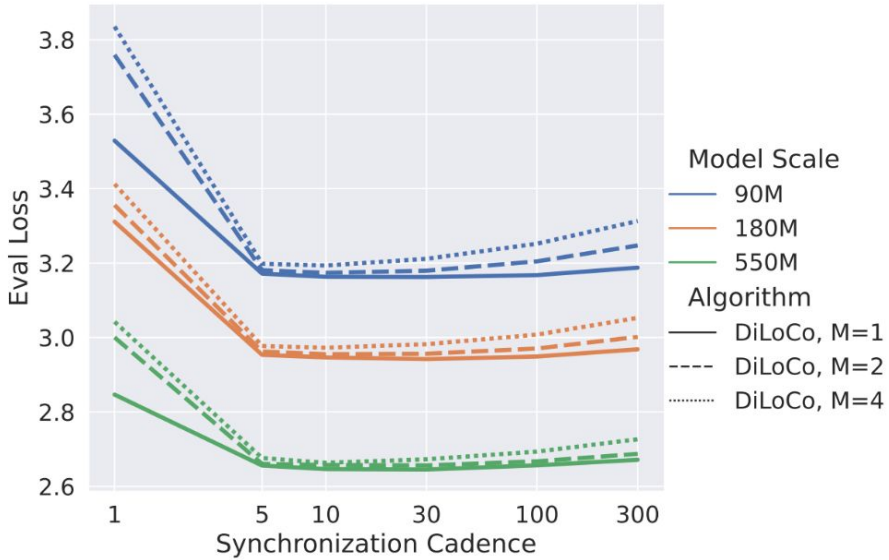
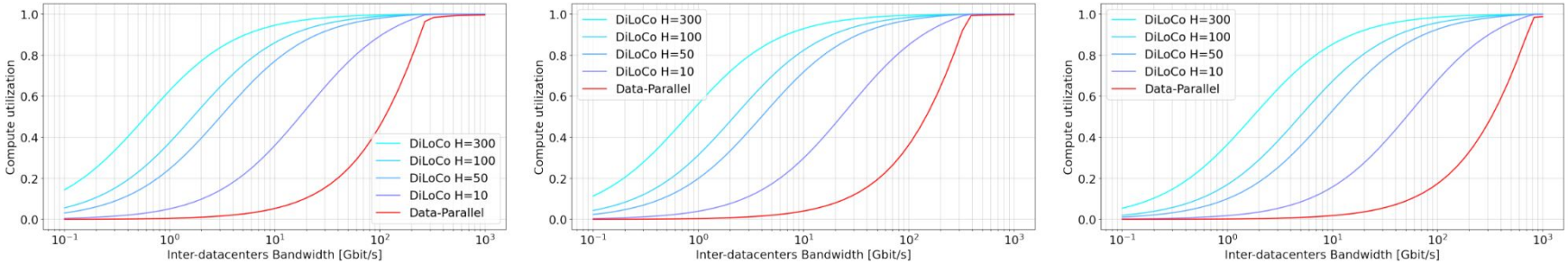


Figure 9: **Infrequent synchronization works better for larger models.** Outside of $H = 1$, which performs the worst, evaluation loss increases with H. However, the rate of increase is less pronounced for DiLoCo with $M = 1$ and for larger models, suggesting that large models can be synchronized quite infrequently.

Ablation #1 - Synchronization cadence



(a) 10B Chinchilla (b) 405B Llama3 (c) 671B DeepSeek-V3

Figure 10: **DiLoCo greatly increases compute utilization**. Here we present simulated compute utilization for DiLoCo and Data-Parallel across a range of bandwidth and synchronization cadences H. A compute utilization of 0.8 means 80% of the time is spent in computation, and 20% in communication. A higher value is better.

Ablation #2 - Overtraining

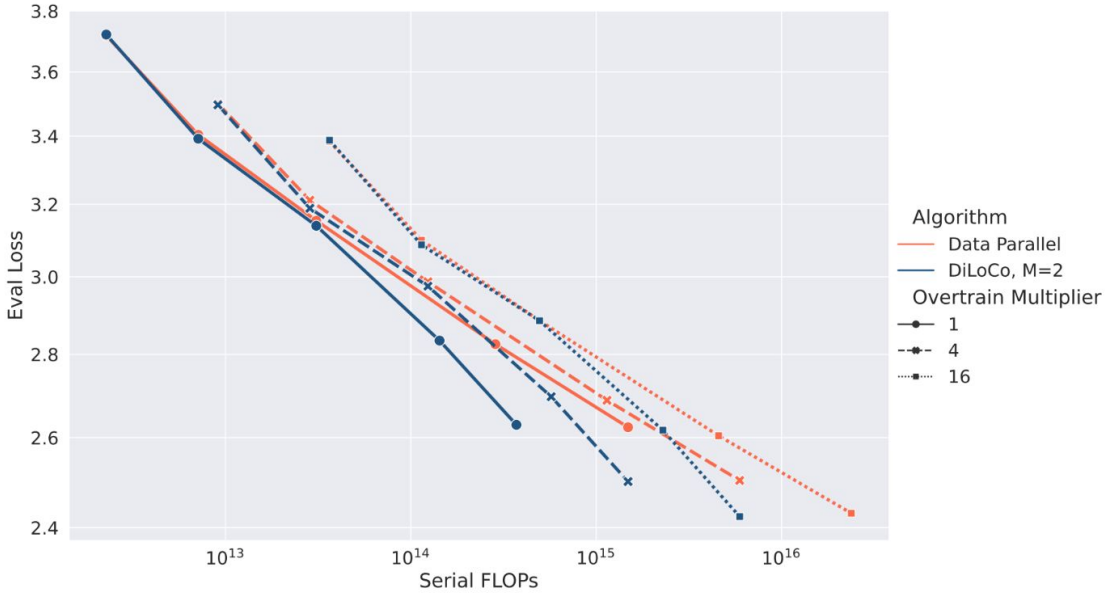


Figure 12: **DiLoCo speeds up overtraining by leveraging horizontal scalability.** For Data-Parallel and DiLoCo, $M = 2$, we plot serial FLOPs (C/B) and evaluation loss, for multiple overtraining amounts. DiLoCo can greatly reduce the amount of serial FLOPs needed to achieve a given loss. We see similar results for other M, but present $M = 2$ for clarity.

Scaling Laws

Independent fit scaling laws:

- For Data-Parallel, and DiLoCo with a fixed number of model replicas M:
  - Fit L(N)
  - Fit γ(N)
  - Fit B(N)
- Power law

Joint fit scaling laws:

- For DiLoCo:
  - Fit L(N, M)
  - Fit γ(N, M)
  - Fit B(N, M)
- Joint power law

Notes:

- Chinchilla-optimal compute budget, so fitting f(N) is analogous to fitting f(C)
- L = evaluation loss
- γ = (inner) learning rate
  - LR for InnerOpt when using DiLoCo
- B = batch size
  - Sum of per-replica batch sizes for DiLoCo
- M = number of DiLoCo replicas

For DiLoCo, which one works better?

Google Research

Scaling Laws

General findings:

- Quantitatively similar power laws between Data-Parallel & DiLoCo
- DiLoCo M = 1 is predicted to be always better than data-parallel
- DiLoCo M = 2 is predicted to be eventually better than Data-Parallel

Leave-one-out evaluation:

- Fit power laws with data up to 1.3B, evaluate on held-out 2.4B data
- Compute residual |log actual - log predicted|

Table 7: Power law approximations for loss $L(N) \approx AN^{\alpha}$.

	A	α
Data-Parallel	18.129	−0.0953
DiLoCo, $M = 1$	18.363	−0.0961
DiLoCo, $M = 2$	18.768	−0.0969
DiLoCo, $M = 4$	19.762	−0.0992
DiLoCo, $M = 8$	21.051	−0.1018

	Fit	L	γ_{DLC}	B_{DLC}
$M = 1$	Independent	0.011	0.35	0.00088
	Joint	0.019	0.14	0.19
$M = 2$	Independent	0.0099	0.18	0.44
	Joint	0.013	0.29	0.28
$M = 4$	Independent	0.012	0.051	0.25
	Joint	0.0082	0.086	0.11
$M = 8$	Independent	0.014	0.62	0.076
	Joint	0.0076	0.23	0.19
Average over M	Independent	**0.012**	0.30	**0.19**
	Joint	**0.012**	**0.19**	**0.19**

Joint fit is equivalent or better! →

Scaling Laws

Extrapolating to larger models:

- Use predicted optimal hyperparameters at 4B and 10B scales, and measure evaluation loss

Table 12: **Joint fit hyperparameters extrapolate well to larger models.** Here we show the evaluation results on 4B and 10B models, using hyperparameters predicted by individual and joint scaling laws. We highlight DiLoCo evaluation results that were better (ie. lower) than Data-Parallel. We see that while DiLoCo with $M = 2$ does better than Data-Parallel with either independent or joint fit rules, DiLoCo $M = 1$ only does better when using joint fit.

Algorithm	Fit Method	Loss	
		4B	10B
Data-Parallel	Independent	2.224	2.090
DiLoCo, $M = 1$	Independent	2.229	2.103
	Joint	**2.219**	**2.086**
DiLoCo, $M = 2$	Independent	**2.218**	**2.083**
	Joint	**2.220**	**2.086**
DiLoCo, $M = 4$	Independent	2.232	2.098
	Joint	2.230	2.096

Scaling Laws

Parametric function fitting:

- Unclear how loss should vary as a function of N and M.
  - Chinchilla paper used classic risk decomposition to do decomposition
  - No analog exists (to our knowledge) for DiLoCo
- Data-driven approach: Try multiple parametric forms, and use leave-one-out validation.

Table 13: **Parametric function fitting improves joint scaling laws.** We showcase various parametric approximations to the empirical loss function $L(N, M)$, along with their validation error on held-out loss data at the $N = 2.4B$ scale. We see that joint power laws (the first) and classical loss decompositions (the last) are significantly worse at predicting loss on the held-out data.

Parametric form	Average Residual
$L_{\text{DLC}}(N, M) \approx AN^{\alpha}M^{\beta}$	0.0044
$L_{\text{DLC}}(N, M) \approx AN^{\alpha}M^{\beta} + C$	0.0035
$L_{\text{DLC}}(N, M) \approx AN^{\alpha+\beta M} + C$	**0.0025**
$L_{\text{DLC}}(N, M) \approx AN^{\alpha} + BM^{\beta} + C$	0.0043

Outline

Bo Liu
Lead author
(internship)

Motivation

Basic Diloco

Scaling Laws

Async Local-SGD

Streaming DiLoCo

Open challenges

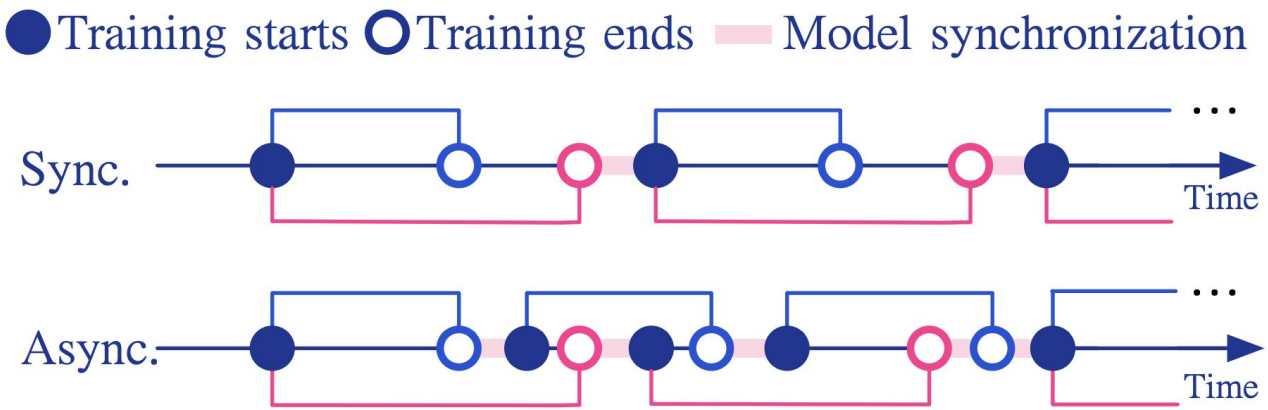
Let's make it Async

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.

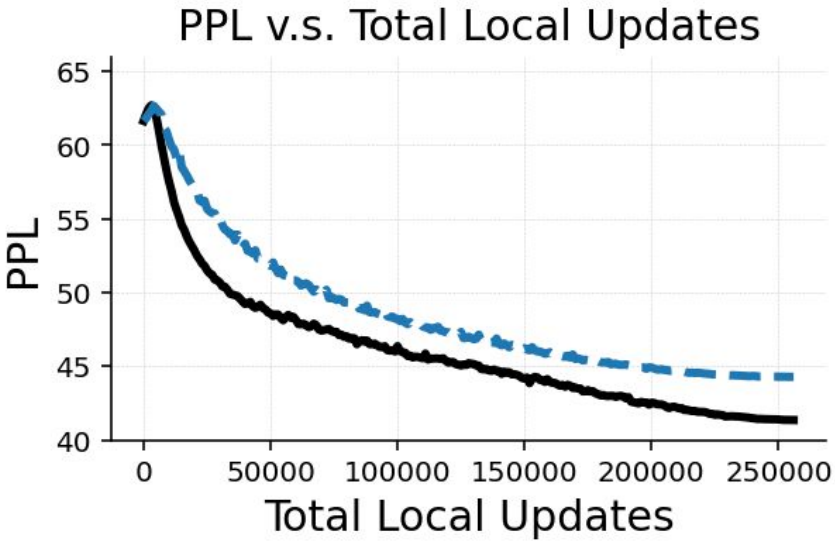
Example with two replicas:

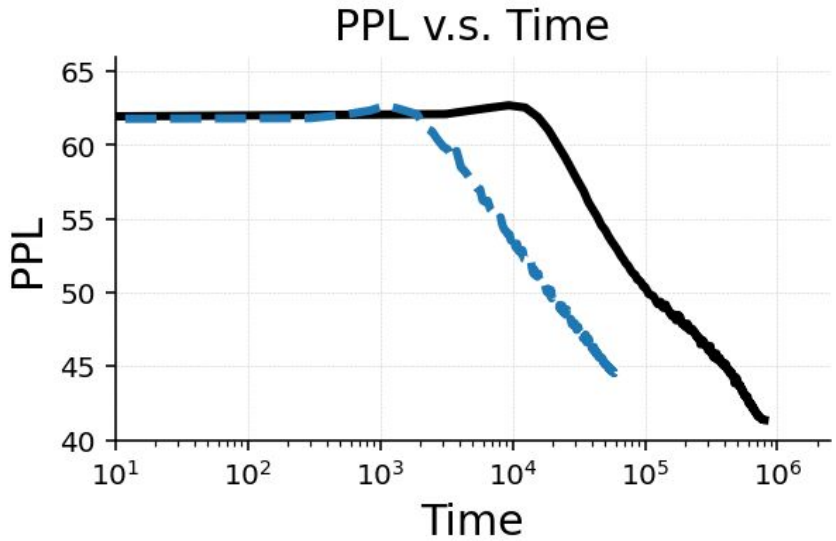
Fast devices

Slow devices



Does async works out-of-the-box?





When dealing with heterogeneous devices, it's faster!

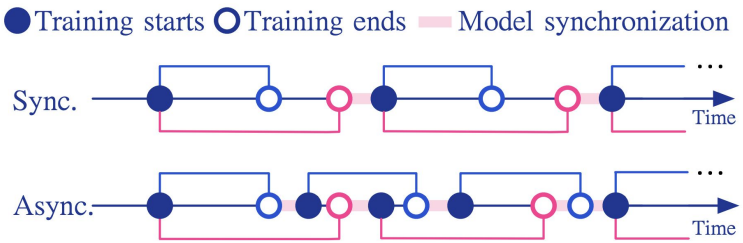
But with respect to the number of updates, that's quite worse despite using as much data and compute. ← Can we do better?

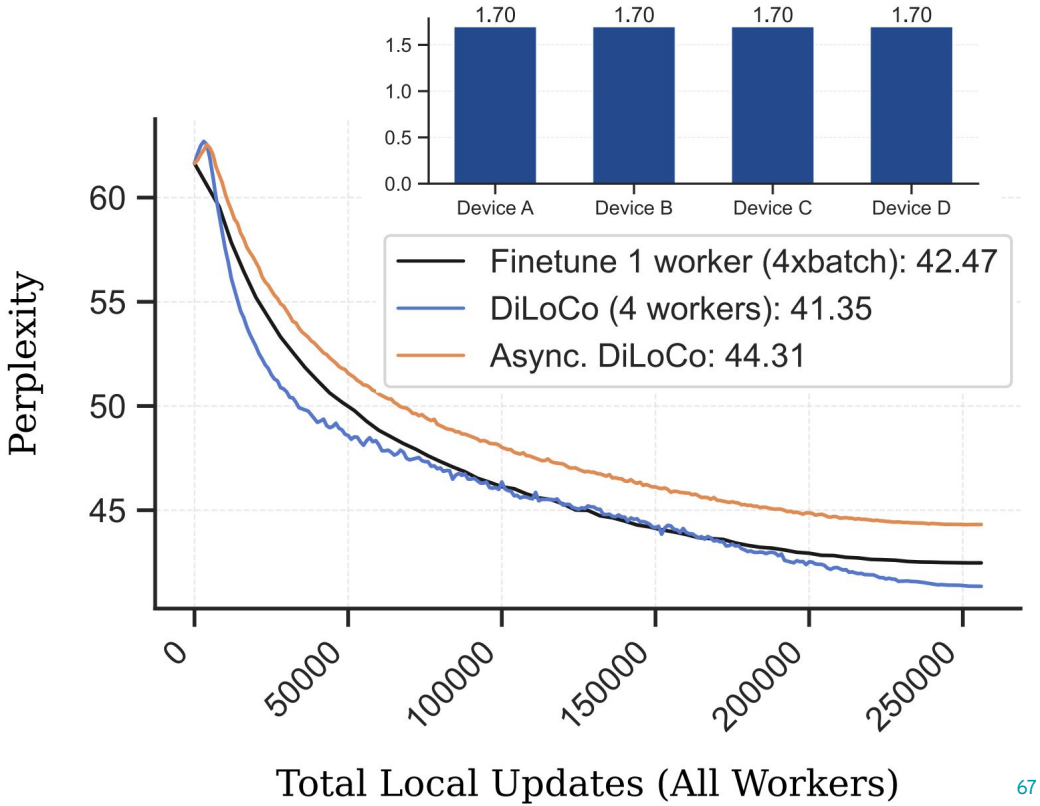
Let's try async with the same speed/replica

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

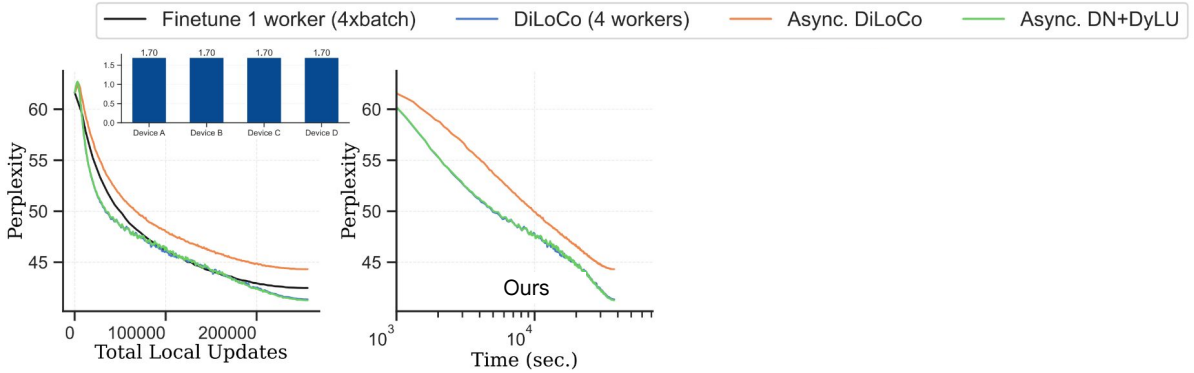
Algorithm 3 Delayed Nesterov Update.

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

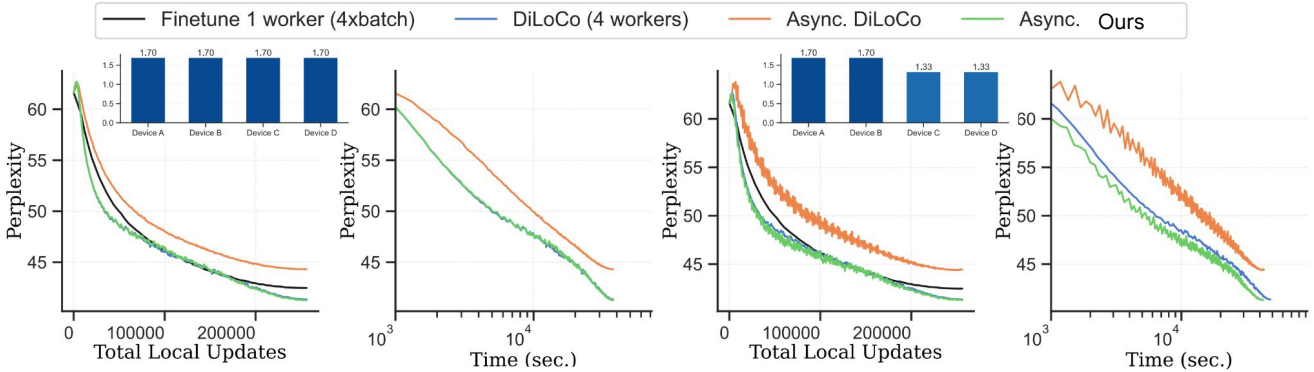
What's the results?

Given little staleness, Our method is as good as DiLoCo



What's the results?

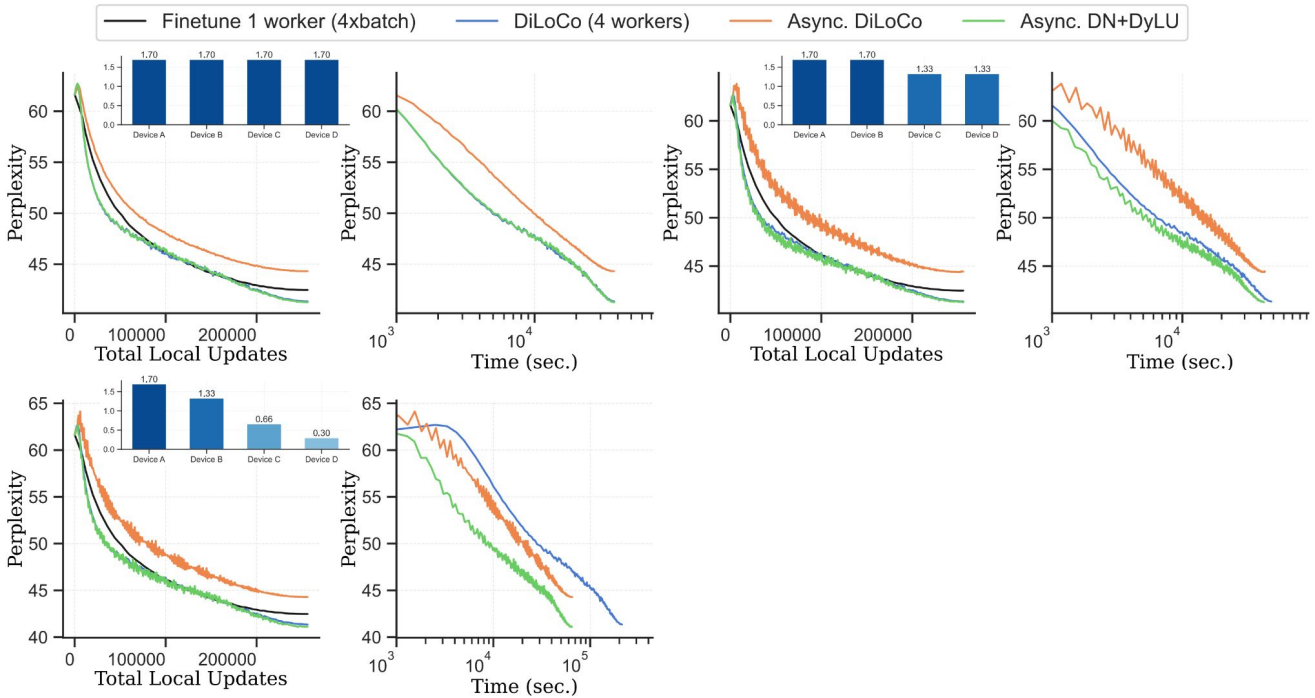
But add more staleness by using different device types per replicas...



What's the results?

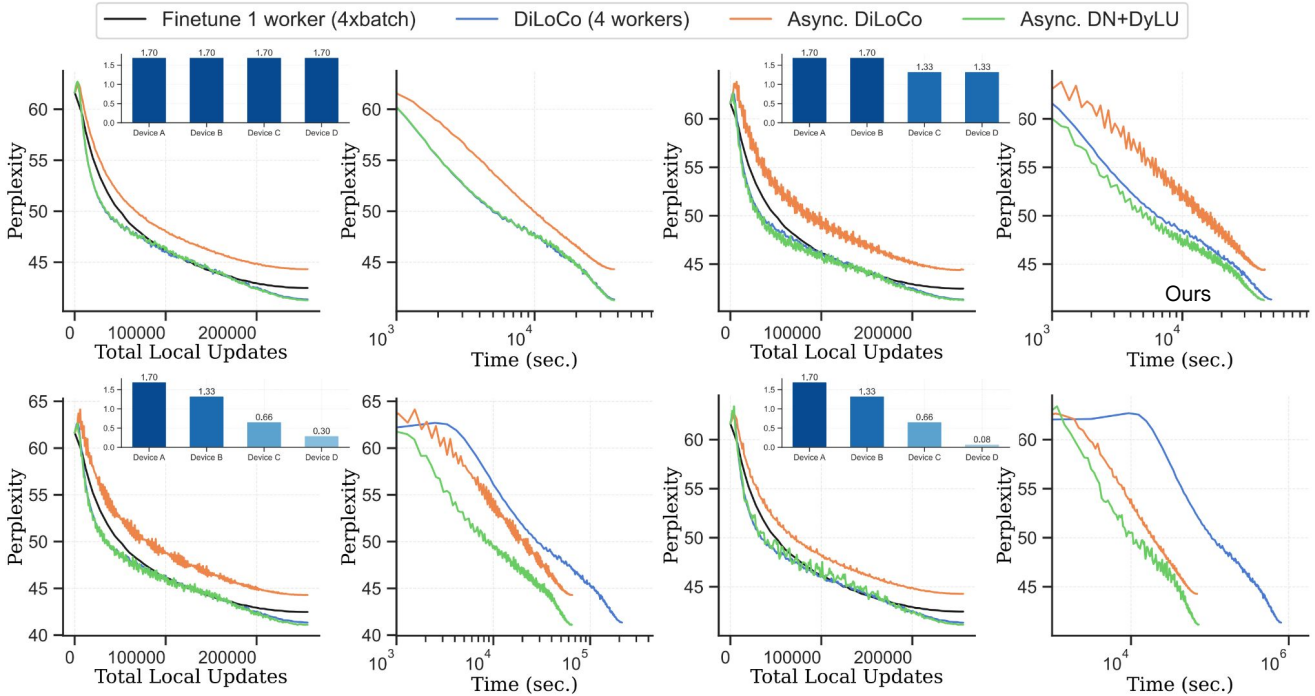
But add more staleness by using different device types per replicas...

Ours



What's the results?

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

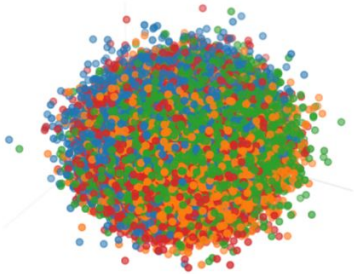


Open-source toy setting

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!

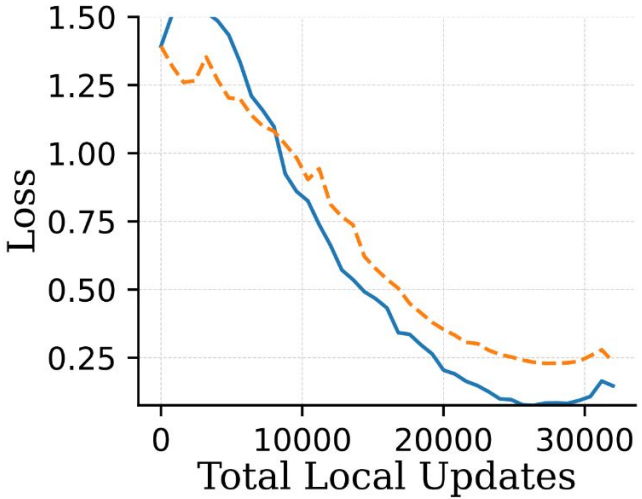
The Dataset



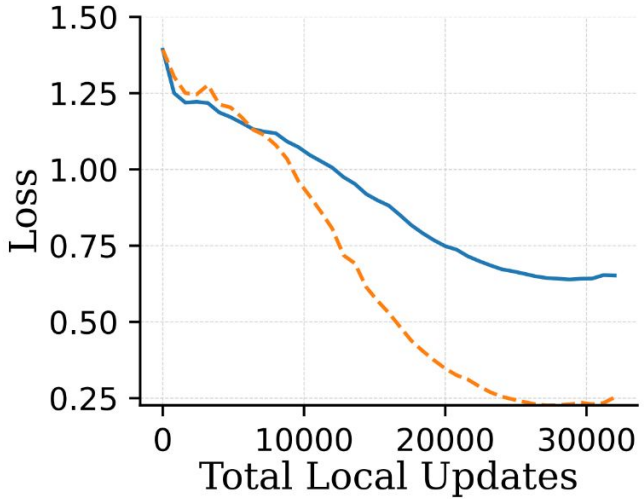
Async.

Sync.

AdamW + Nesterov



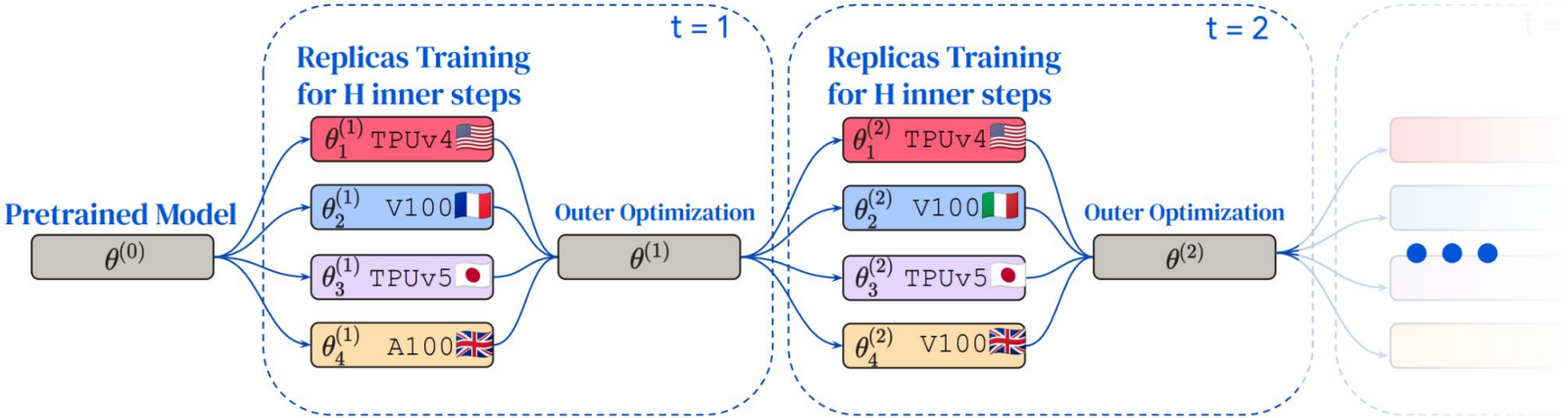
AdamW + SGD



🤔 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



Outline

Motivation

Basic Diloco

Scaling Laws

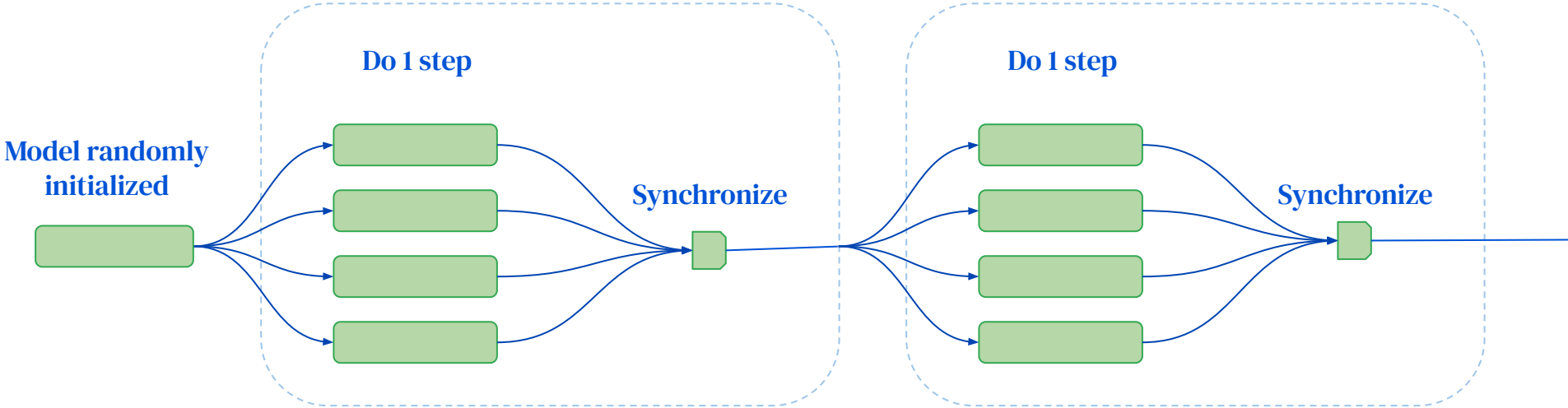
Async Local-SGD

Streaming DiLoCo

Open challenges

Data-Parallel

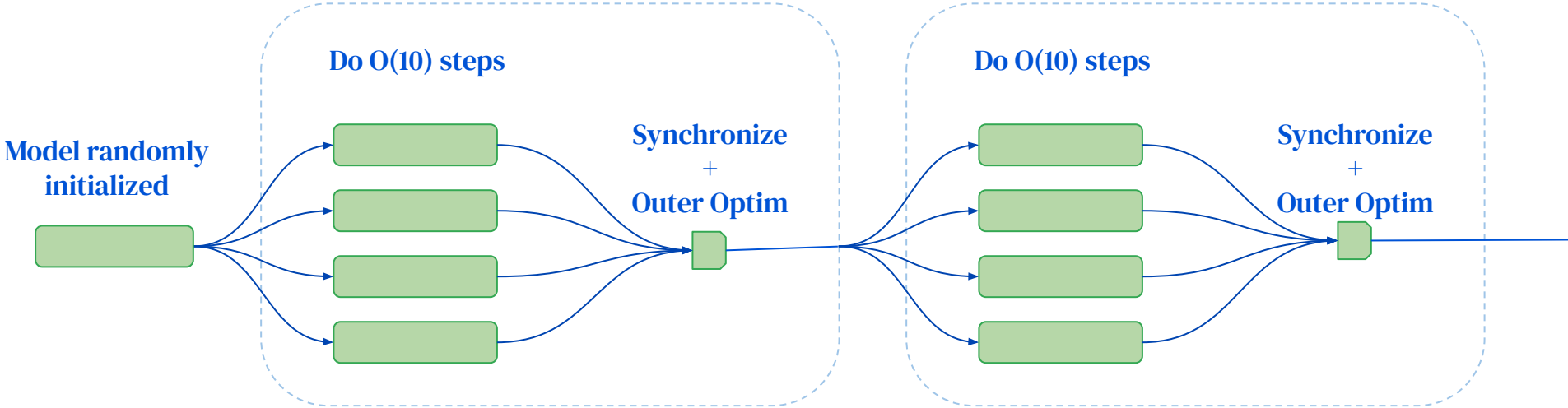
Synchronizing at every steps



DiLoCo

Bi-level optimization framework

Synchronize replicas every O(10) steps



How? Local training + Outer optimization!

Bi-level optimization:
Inner optim on the gradients, outer optim on the trajectories/outer gradients

Inspired from Local-SGD / FedOpt, DiLoCo works in two alternating phases:

Inner Optimization:

- Uses data-, sequence-, and model-parallelism
- Each DiLoCo replica does multiple forward/backward/optim independently

Outer Optimization:

- Compute each replica's *trajectory*, e.g. the delta between the original parameters and the new ones
- We use it as an "*outer gradient*" that is all-reduced among replicas
- Optimize with Nesterov momentum

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

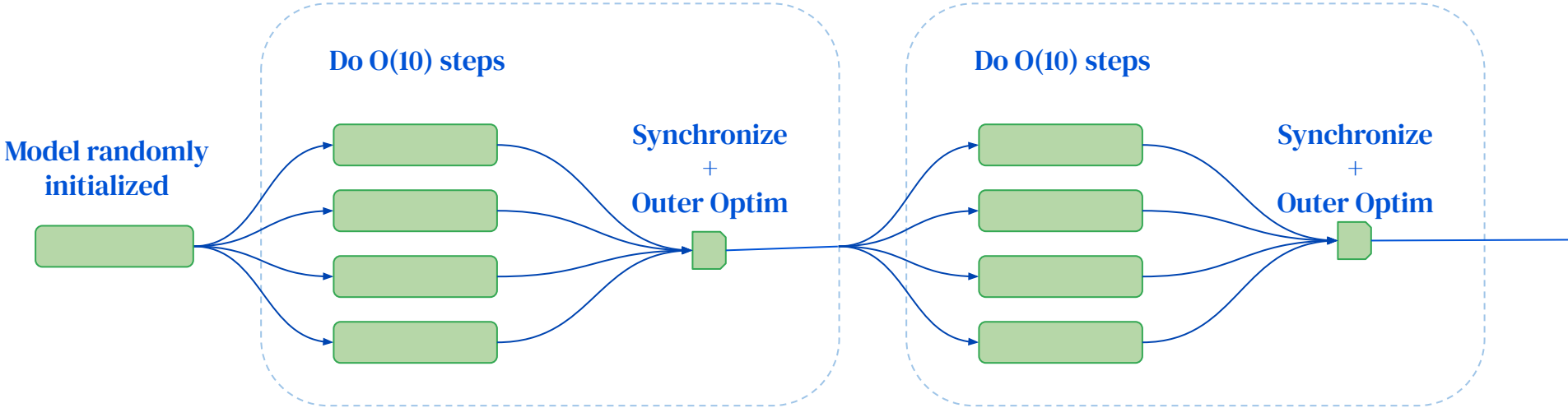
for `outer step` $t = 1 \ldots T$ **do**
  for `worker` $i = 1 \ldots k$ **do**
    $\theta_i^{(t)} \leftarrow \theta^{(t-1)}$
    for `inner step` $h = 1 \ldots H$ **do**
      $x \sim \mathcal{D}_i$
      $\mathcal{L} \leftarrow f(x, \theta_i^{(t)})$
      ▷ Inner optimization:
      $\theta_i^{(t)}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

Streaming DiLoCo

DiLoCo

Bi-level optimization framework

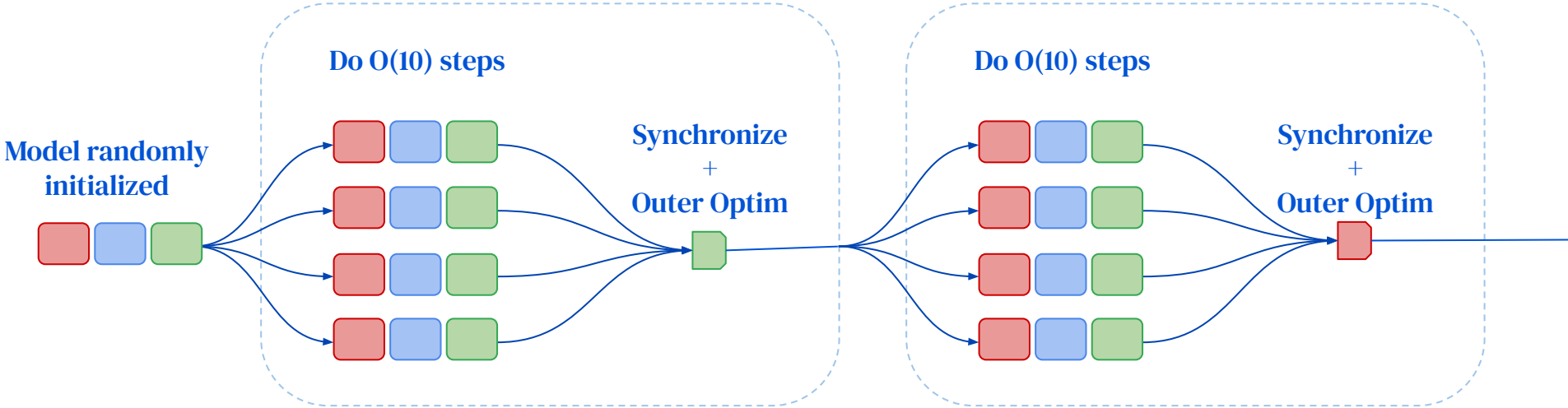
#1 Synchronize replicas every O(10) steps



Streaming DiLoCo

Bi-level optimization framework

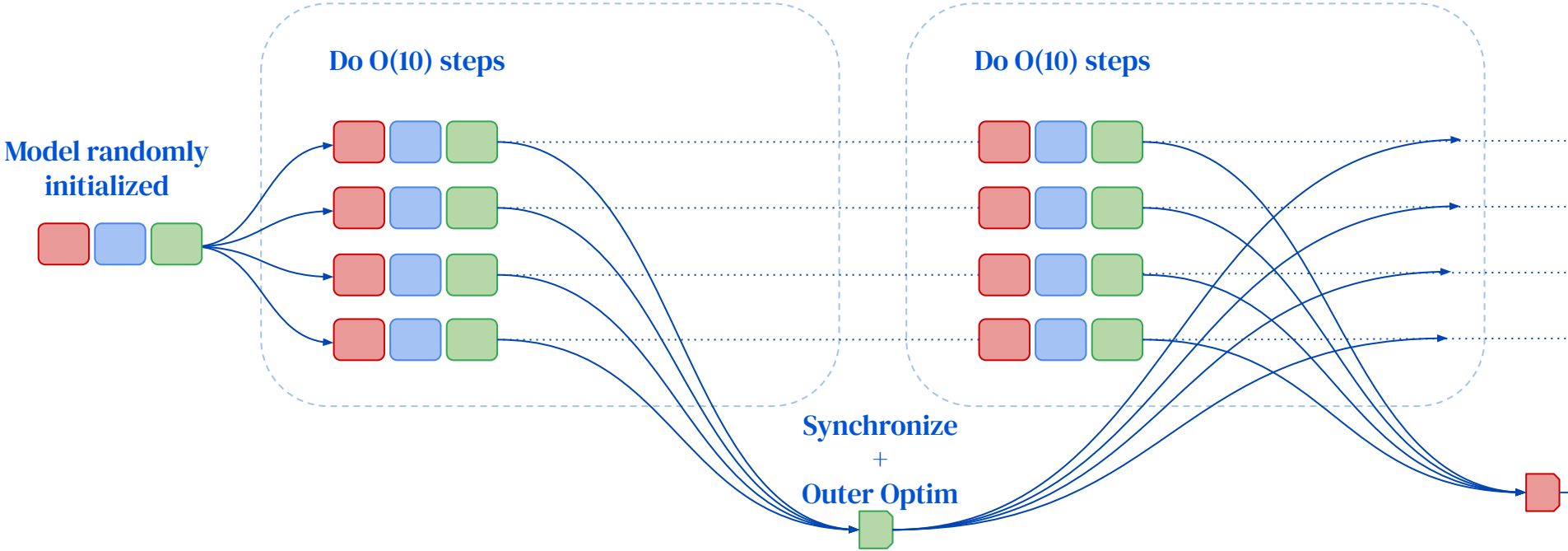
#2 Partial synchronization of parameters



Streaming DiLoCo

Bi-level optimization framework

#3 Synchronization overlapped with computation



Streaming DiLoCo

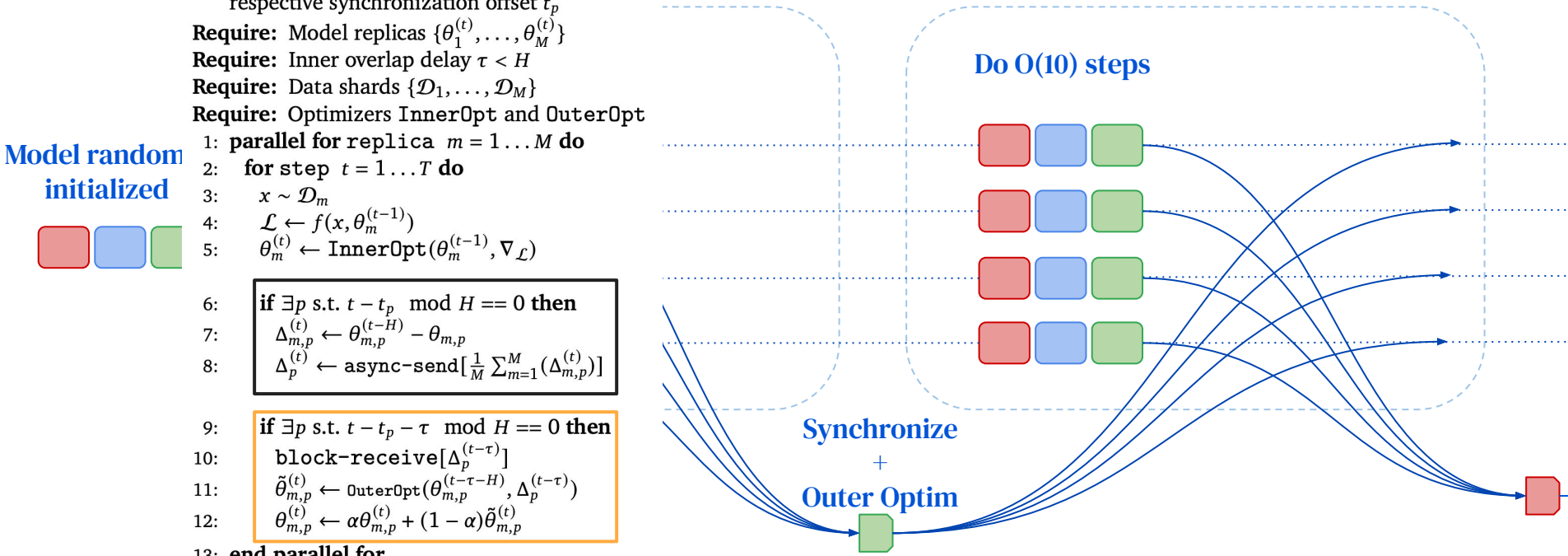
Bi-level optimization framework

Algorithm 2 Streaming DiloCo

Require: M replicas
Require: Number of inner steps H
Require: Fragments $p \in \{1, \dots, P\}$ with their respective synchronization offset t_p
Require: Model replicas $\{\theta_1^{(t)}, \dots, \theta_M^{(t)}\}$
Require: Inner overlap delay $\tau < H$
Require: Data shards $\{\mathcal{D}_1, \dots, \mathcal{D}_M\}$
Require: Optimizers `InnerOpt` and `OuterOpt`

1: **parallel for** `replica` $m = 1 \dots M$ **do**
2: **for** `step` $t = 1 \dots T$ **do**
3: $x \sim \mathcal{D}_m$
4: $\mathcal{L} \leftarrow f(x, \theta_m^{(t-1)})$
5: $\theta_m^{(t)} \leftarrow \texttt{InnerOpt}(\theta_m^{(t-1)}, \nabla_{\mathcal{L}})$
6: **if** $\exists p$ s.t. $t - t_p \mod H == 0$ **then**
7: $\Delta_{m,p}^{(t)} \leftarrow \theta_{m,p}^{(t-H)} - \theta_{m,p}$
8: $\Delta_p^{(t)} \leftarrow \texttt{async-send}[\frac{1}{M} \sum_{m=1}^{M} (\Delta_{m,p}^{(t)})]$
9: **if** $\exists p$ s.t. $t - t_p - \tau \mod H == 0$ **then**
10: `block-receive`$[\Delta_p^{(t-\tau)}]$
11: $\tilde{\theta}_{m,p}^{(t)} \leftarrow \texttt{OuterOpt}(\theta_{m,p}^{(t-\tau-H)}, \Delta_p^{(t-\tau)})$
12: $\theta_{m,p}^{(t)} \leftarrow \alpha \theta_{m,p}^{(t)} + (1 - \alpha)\tilde{\theta}_{m,p}^{(t)}$
13: **end parallel for**

#3 Synchronization overlapped with computation



Streaming DiLoCo

Bi-level optimization framework

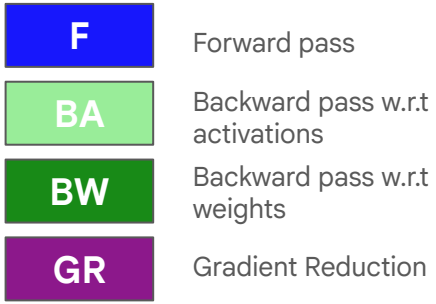
#1 Synchronize replicas every O(10) steps

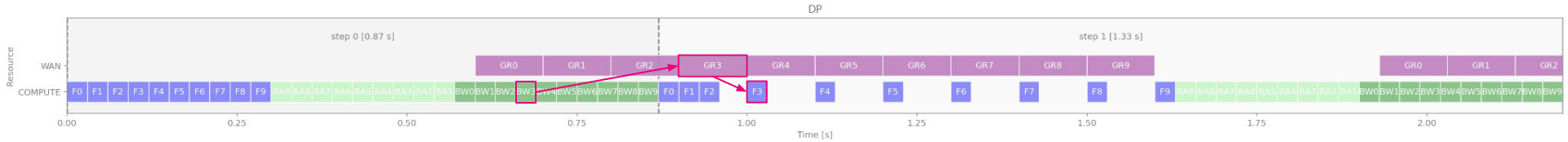
#2 Partial synchronization of parameters

#3 Synchronization overlapped with computation

Communication pattern

Overlapping of the gradient reduction with the next forward pass



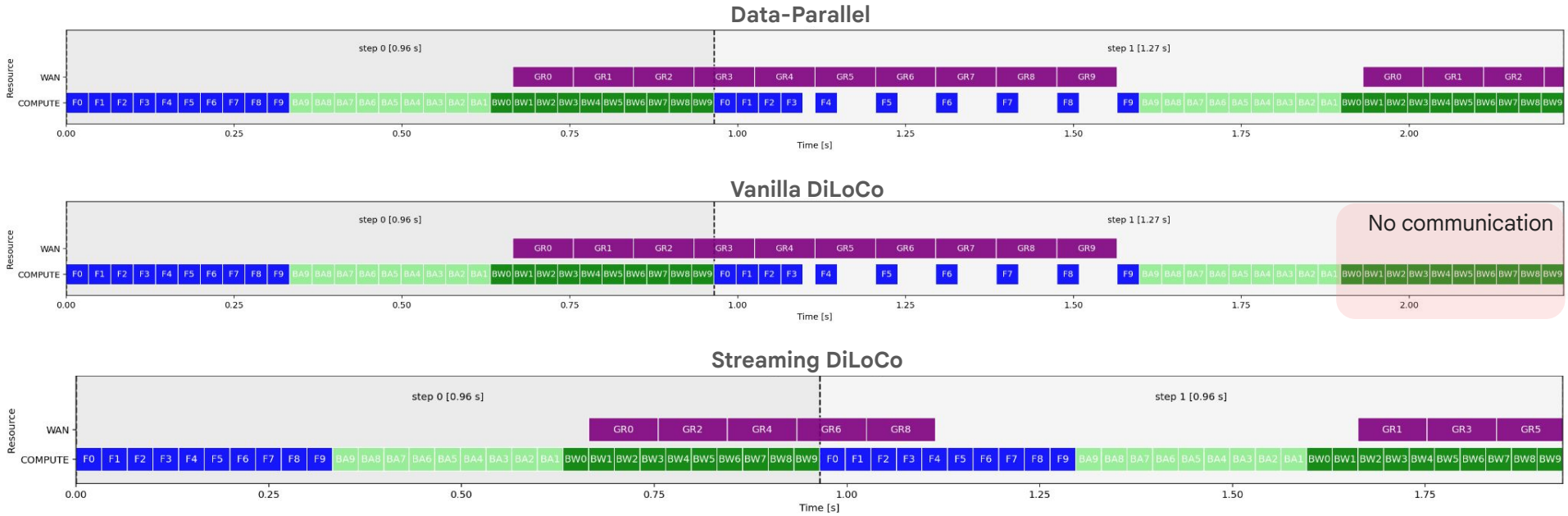


The **forward for layer 3 in step 1** F3 needs to wait for the **layer 3 gradients** generated by BW3 that are **transmitted** by GR3.

Google DeepMind

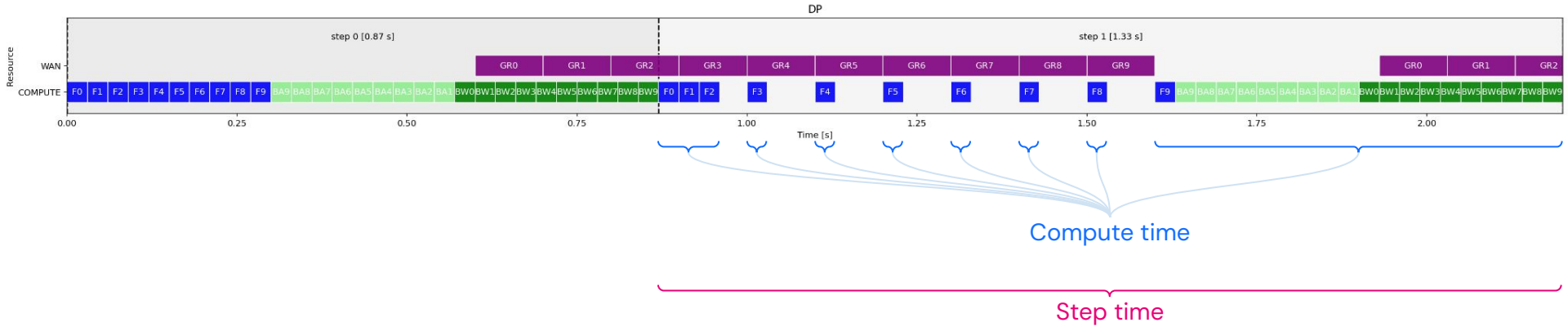
Communication pattern

We can fully overlap the gradient reduction with the next forward pass



Compute utilization

How much time is spent in compute vs total time

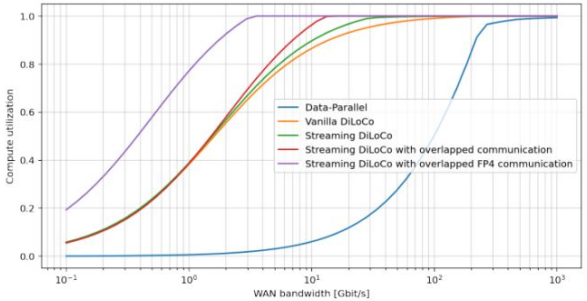


$$\text{Compute utilization} = \frac{\text{Compute time}}{\text{Step time}}$$

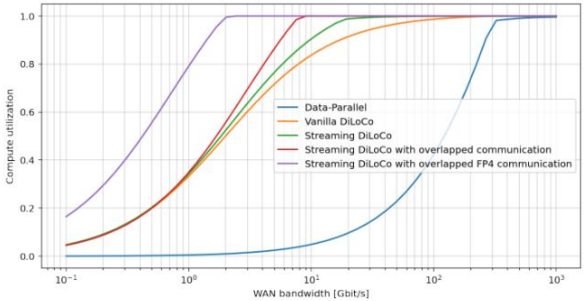
Google DeepMind

Requiring less bandwidth as the model scales!

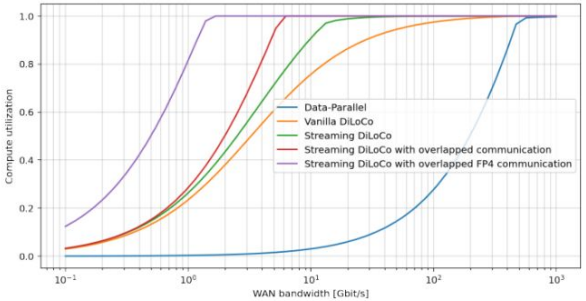
Larger models have longer step time, and thus it counteracts the fact that we have to transfer more bits!



(a) 1B parameters model.



(b) 10B parameters model



(c) 100B parameters model

Bandwidth required for 95% compute utilization:

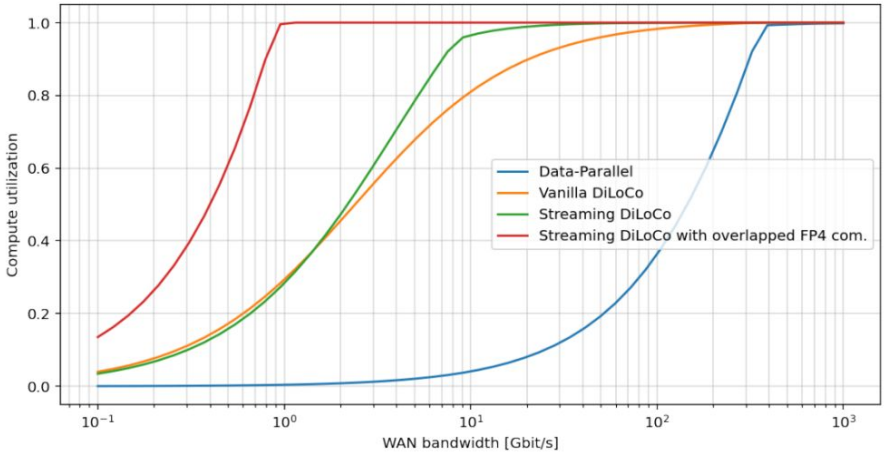
DP 222.3 Gbit/s
DiLoCo 2.0 Gbit/s

DP 268.3 Gbit/s
DiLoCo 1.4 Gbit/s

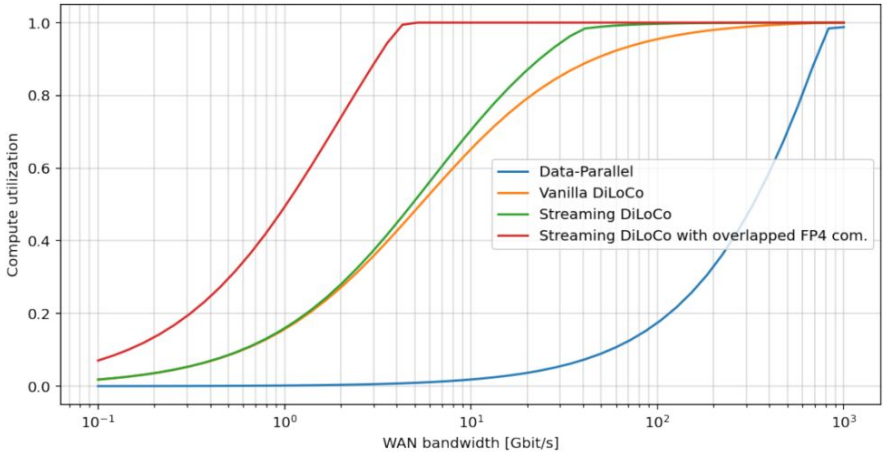
DP 390.7 Gbit/s
DiLoCo 1.1 Gbit/s

Requiring less bandwidth as the model scales!

Some famous models



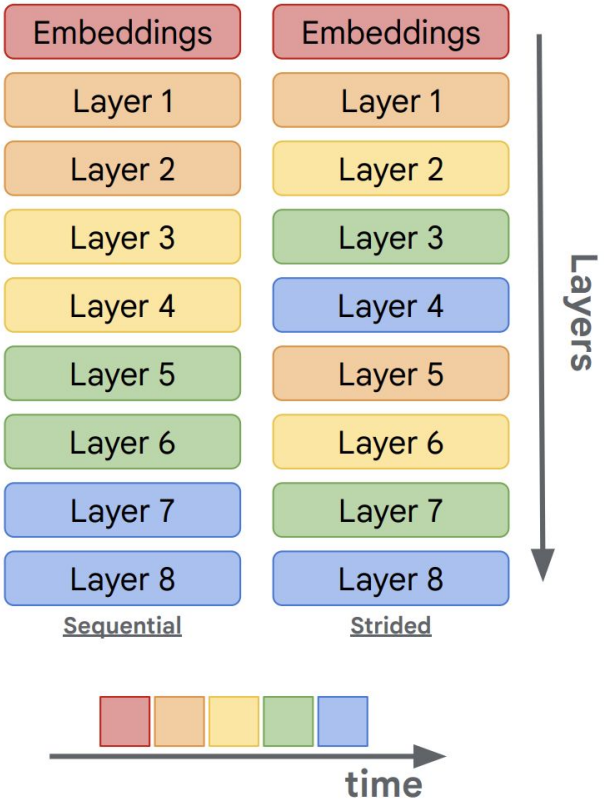
(a) Llama405B.

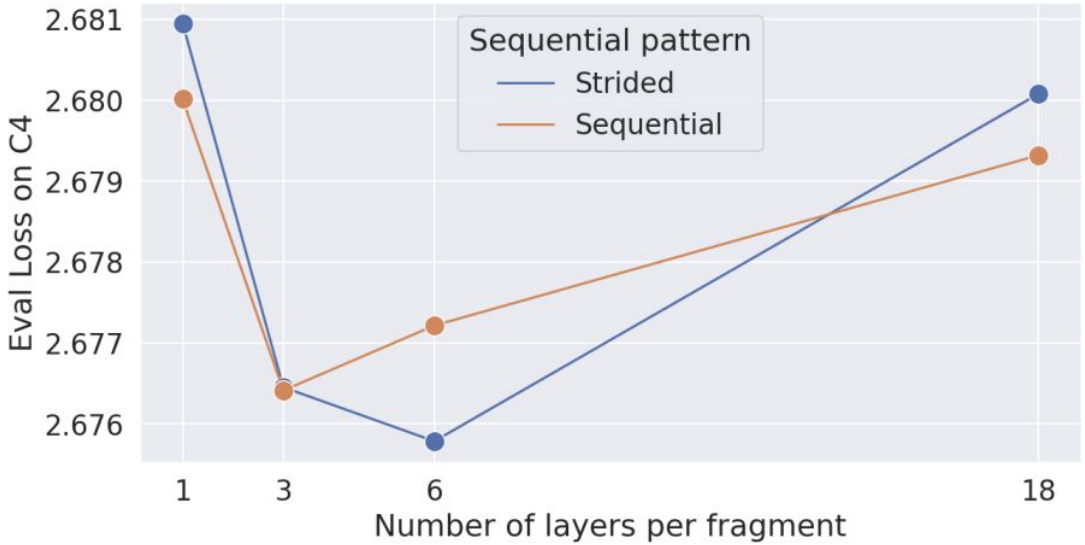


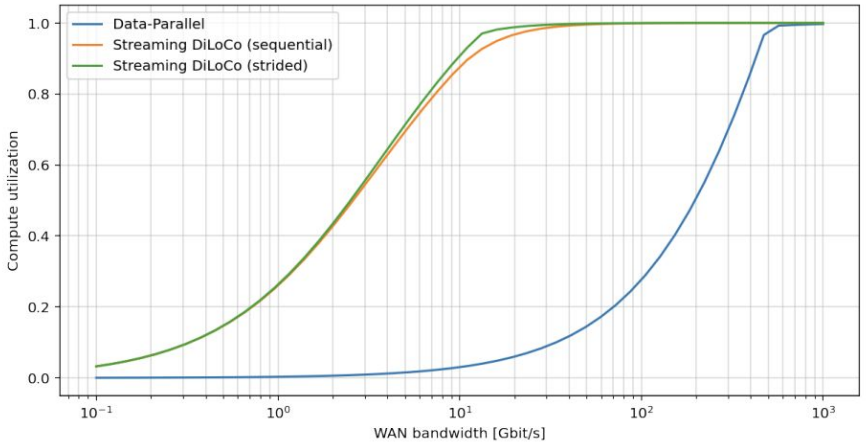
(b) DeepSeek-V3 (671B total, 35B activated).

ML performance?

Streaming Structure





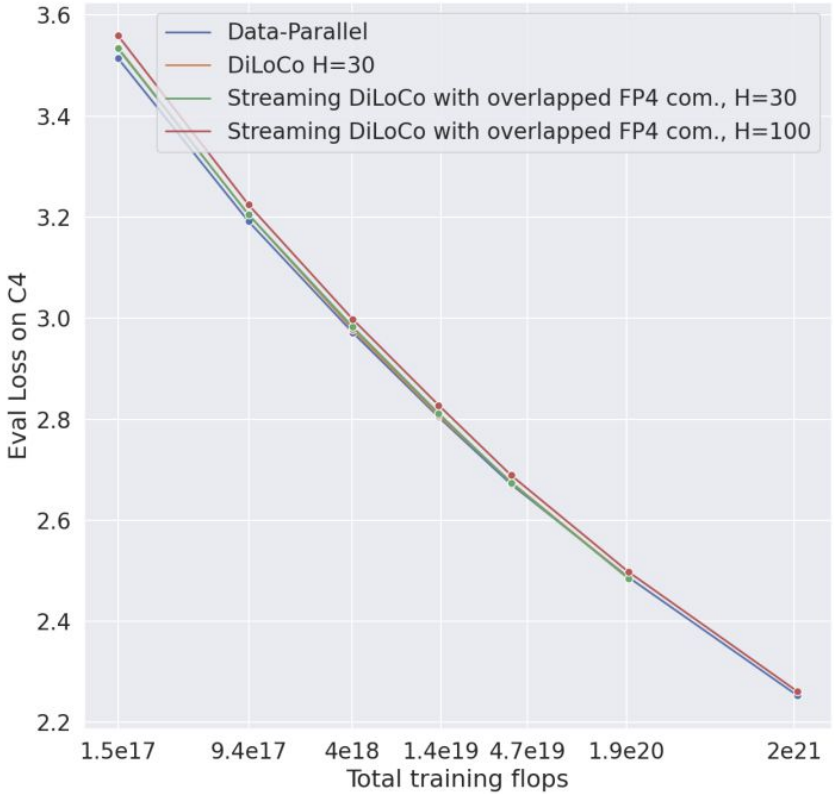


Scaling laws up to 4B dense chinchilla

Slightly worse at small scale (35M).

As good as Data-Parallel at 4B scale.

Note that DiLoCo and Streaming DiLoCo are sensibly the same in term of ML perf!



(a) Evaluation loss on C4

Overtraining

Overtraining by 1xOT, 4xOT, and 10xOT

DiLoCo is better as we overtrain!

Method	Token Budget	Hours spent w/ +∞ Gbits/s	Hours spent w/ 1 Gbits/s	Terabytes exchanged	Eval Loss ↓	HellaSwag ↑	Piqa ↑	Arc Easy ↑
Data-Parallel	25B	0.67	109	441	2.67	**42.09**	67.35	**40.42**
	100B	2.7	438	1,767	2.52	49.78	69.15	**44.03**
	250B	6.75	1097	4,418	**2.45**	53.86	70.45	**44.21**
Streaming DiLoCo with overlapped FP4 communication	25B	0.67	0.88	1.10	**2.66**	42.08	**67.46**	38.42
	100B	2.7	3.5	4.42	**2.51**	**49.98**	**69.96**	**44.03**
	250B	6.75	8.75	11.05	**2.45**	**54.24**	**71.38**	41.92

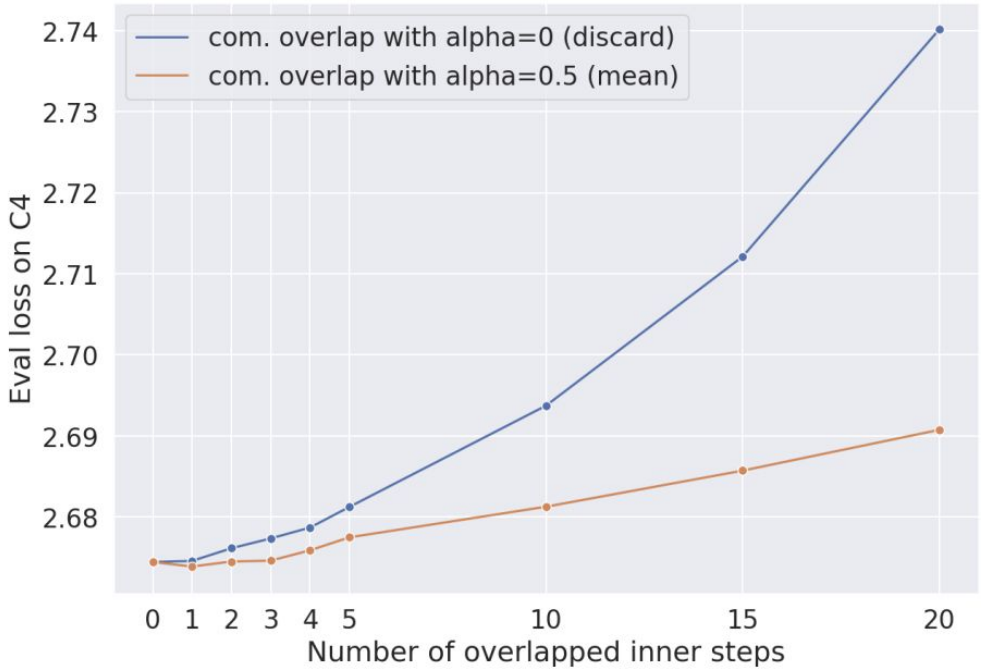
Table 1 | **Overtraining** Data-Parallel and our method on Dolma with a 1 billion parameters model. The latter performs slightly better despite exchanging in total 400× fewer bits, reducing the peak bandwidth by 8×, and with a significantly relaxed training communication latency constraint: allow communication to be as long as a full compute step.

Overlapping compute

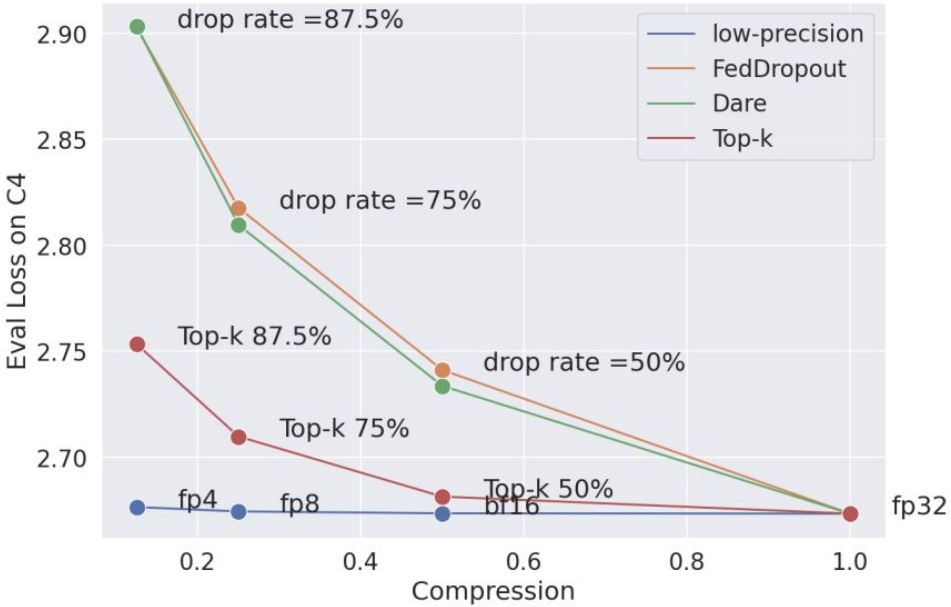
Algorithm 2 Streaming DiLoCo

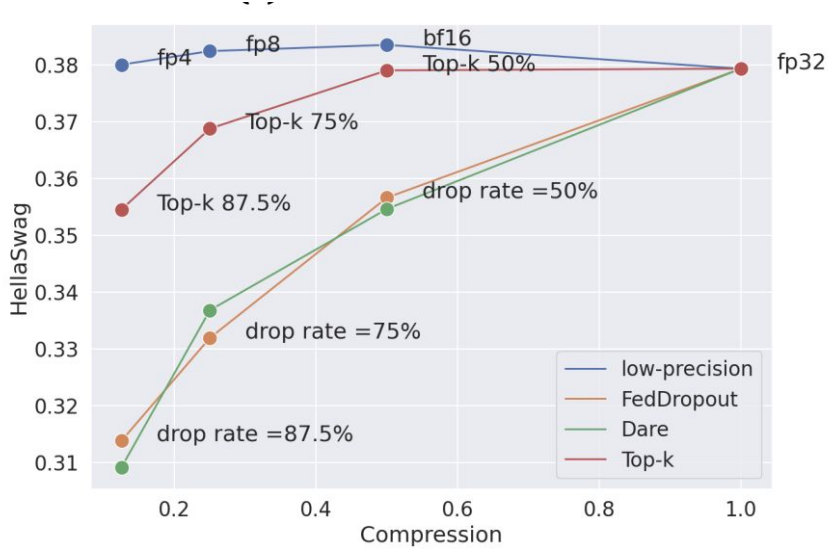
Require: M replicas
Require: Number of inner steps H
Require: Fragments $p \in \{1, \ldots, P\}$ with their respective synchronization offset t_p
Require: Model replicas $\{\theta_1^{(t)}, \ldots, \theta_M^{(t)}\}$
Require: Inner overlap delay $\tau < H$
Require: Data shards $\{\mathcal{D}_1, \ldots, \mathcal{D}_M\}$
Require: Optimizers `InnerOpt` and `OuterOpt`

1: **parallel for** `replica` $m = 1 \ldots M$ **do**
2: **for** `step` $t = 1 \ldots T$ **do**
3: $x \sim \mathcal{D}_m$
4: $\mathcal{L} \leftarrow f(x, \theta_m^{(t-1)})$
5: $\theta_m^{(t)} \leftarrow \texttt{InnerOpt}(\theta_m^{(t-1)}, \nabla_{\mathcal{L}})$
6: **if** $\exists p$ s.t. $t - t_p \mod H == 0$ **then**
7: $\Delta_{m,p}^{(t)} \leftarrow \theta_{m,p}^{(t-H)} - \theta_{m,p}$
8: $\Delta_p^{(t)} \leftarrow \texttt{async-send}[\frac{1}{M} \sum_{m=1}^{M} (\Delta_{m,p}^{(t)})]$
9: **if** $\exists p$ s.t. $t - t_p - \tau \mod H == 0$ **then**
10: `block-receive`$[\Delta_p^{(t-\tau)}]$
11: $\tilde{\theta}_{m,p}^{(t)} \leftarrow \texttt{OuterOpt}(\theta_{m,p}^{(t-\tau-H)}, \Delta_p^{(t-\tau)})$
12: $\theta_{m,p}^{(t)} \leftarrow \alpha \theta_{m,p}^{(t)} + (1 - \alpha) \tilde{\theta}_{m,p}^{(t)}$
13: **end parallel for**



Quantizing communication



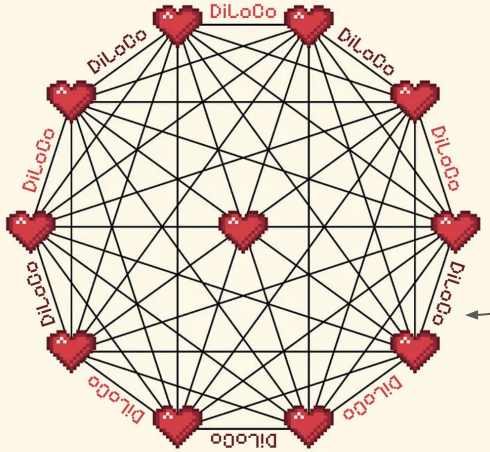


(b) HellaSwag accuracy

Google

DiLoCo

Distributed training across the world



As seen in
The Economist !

Outline

Motivation

Basic Diloco

Scaling Laws

Async Local-SGD

Streaming DiLoCo

Open challenges

Open challenges

Remove/relax even more assumptions:

- Nodes still need to fit the entire model.
- Batch size per node still needs to be quite large.
- Use even more heterogeneous hardware (see BLOOM).
- Good scaling up to 2-4 nodes ⇒ scale better >> 4 nodes.

Recap

how to quickly train an LLM using data-centers across the map

use Streaming Diloco

Sources and References

Main Papers

[1] DiLoCo: Distributed Low-Communication Training of Language Models

[2] Communication-Efficient Language Model Training Scales Reliably and Robustly: Scaling Laws for DiLoCo

[3] Asynchronous Local-SGD Training for Language Modeling

[4] Streaming DiLoCo with overlapping communication: Towards a Distributed Free Lunch

Media

Training AI models might not need enormous data centres