ASCOMP 2026

Distributed Optimization Beyond the Datacenter

Communication, Compression, and Asynchrony

Thalaiyasingam Ajanthan

Pluralis Research

Casablanca · Tuesday 3 November 2026



PR

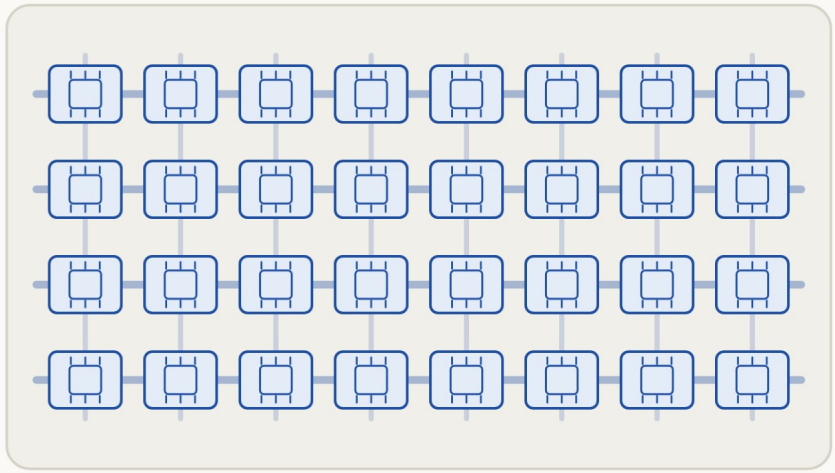
THE QUESTION

Can we train large models across the internet?

Large models are trained in one or a few datacenters, on GPUs joined by very fast links. Many more GPUs sit elsewhere, with different owners, on ordinary internet links.

One datacenter

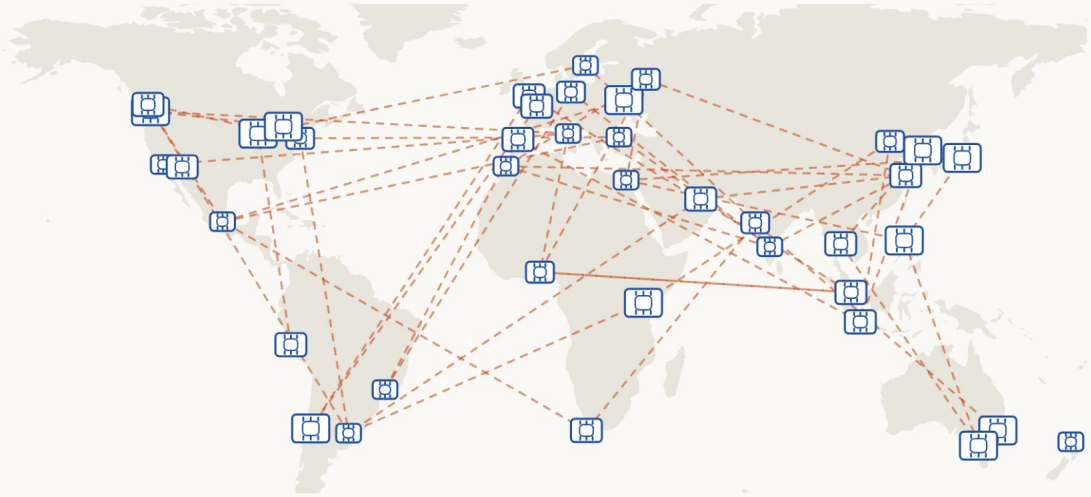
co-located GPUs, one operator



Links: **50–450 GB/s**

Devices across the internet

many owners, different hardware, they come and go



Links: **0.0125–0.125 GB/s**

What must change in the model, optimizer, and infra when links are slow and machines are unequal?

THE QUESTION

Motivation



100k GPUs

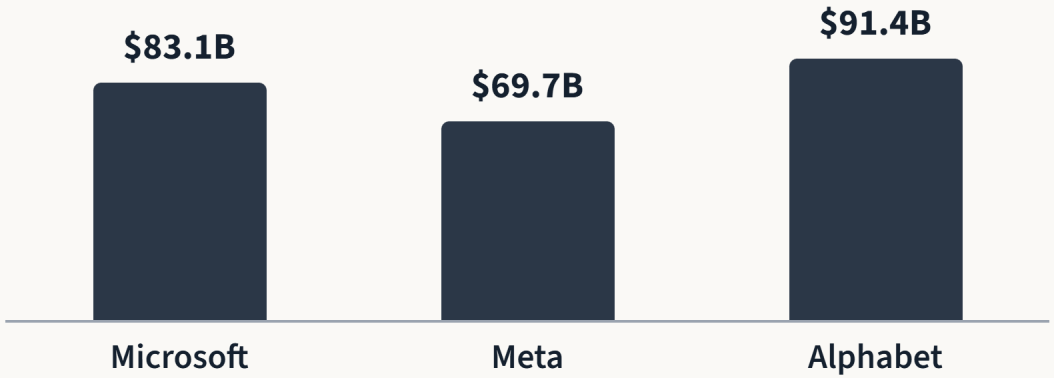
“AI infrastructure today is thinking too small”

A closed system · a single organization · no resource sharing · cannot scale · no pathway to significant data or compute

Nic Lane, Protocol Learning workshop, 26 April 2026

Training is concentrated

Cash capital expenditure in 2025, company-wide



The hardware is everywhere

100M+

RTX GPUs installed (NVIDIA, Jan 2024)

9.2M (Q1 2025) to 12.5M (Q2 2026) desktop graphics boards shipped per quarter

Installed ≠ available ≠ useful for training. These counts are not a measure of training capacity.

Pooling existing hardware could widen who can train a large model — if the algorithms can use it.

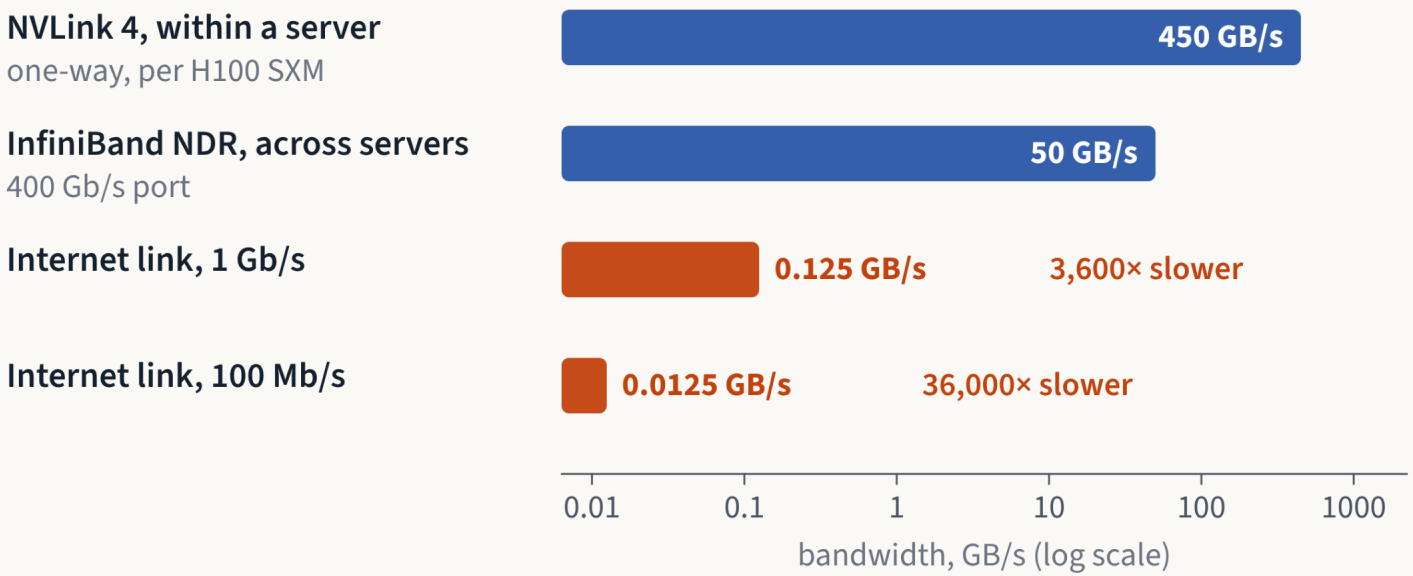
N. Lane, Protocol Learning workshop, Apr 2026. Capex: 2025 cash purchases of property and equipment. NVIDIA, Jan 2024; JPR.

PR

THE QUESTION

What changes outside the datacenter

1 · Links are 3,600–36,000× slower than NVLink



2 · Devices are unequal

Memory from 24 GB to 96 GB per GPU; throughput differs several-fold between workers.

3 · Membership changes

Workers join, leave and fail during the run. Nobody can be waited for indefinitely.

Nominal one-way rates: NVLink 4 (18 × 25 GB/s per H100 SXM), ConnectX-7 NDR 400 Gb/s; internet links at full line rate.

PR

THE PLAN

A cost model for wall-clock time

Wall-clock time to reach accuracy ε has three parts. Each method we study targets one of them.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

steps to ε × compute per step | rounds × time per round | waiting: barriers, stragglers

Local steps	↑ drift	↓ ÷H rounds	
Compression	↑ compression error	↓ fewer bytes per round	
Gossip	↑ slower agreement	same bytes per worker	↓ no global barrier
Asynchrony, overlap	↑ staleness	hidden behind compute	↓ no waiting
Split the model	↑ from its fixes (Part II)	activations every microbatch	pipeline bubbles

Each method reduces one term and often increases K(ε).

K: steps to ε · R: rounds · teal row: Part II · with overlap, compute and communication run concurrently and + becomes max.

PR

PART I

Every worker holds the whole model

Four ways to reduce communication time. Each reduces one term of the cost model and often increases K(ε).

1 · Fewer rounds

every step



every H steps



↓ rounds R(ε)

↑ drift between replicas

Local SGD, DiLoCo

2 · Smaller messages

dense message



compressed message



↓ time per round

↑ compression error

PowerSGD, error feedback

3 · Fewer peers



everyone



neighbors

↓ global barrier

↑ slower agreement

Gossip, D-PSGD

4 · No waiting

wait for the message



keep computing



↓ idle time

↑ staleness

Asynchronous SGD, overlap

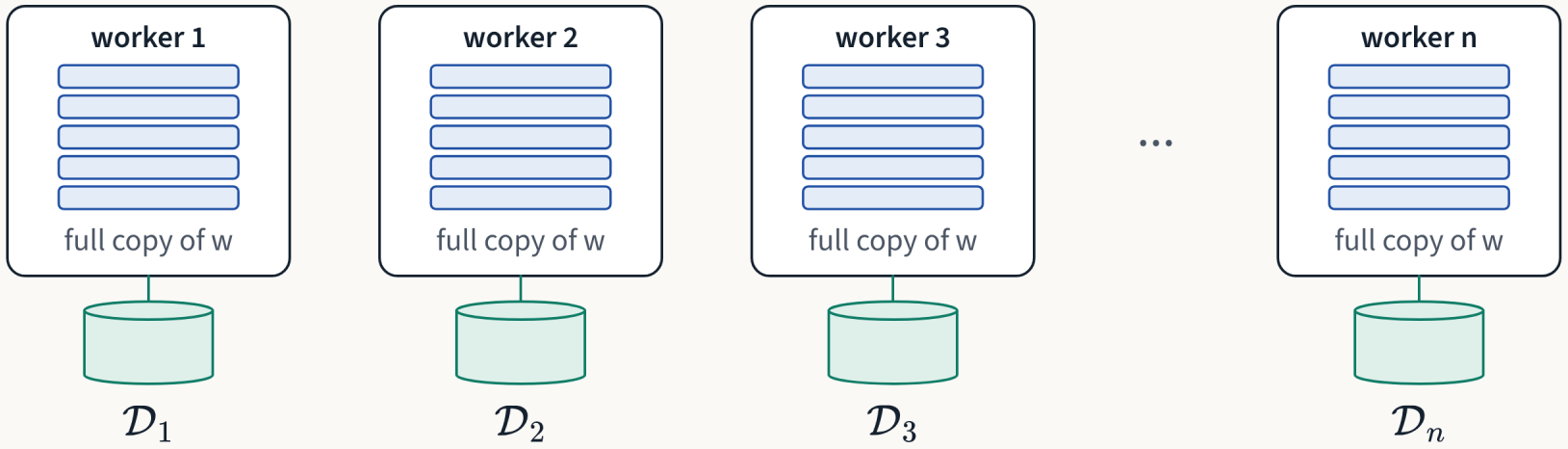
$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

PART I · SETUP

One objective, many replicas

Each of n workers holds a full copy (a replica) of the parameters w and samples from its own data. Together they minimize the average loss.

$$\min_{w \in \mathbb{R}^d} \; f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w), \qquad f_i(w) = \mathbb{E}_{\xi \sim \mathcal{D}_i}\left[\ell(w; \xi)\right]$$



Same data distribution

$\mathcal{D}_i = \mathcal{D}$ (IID)

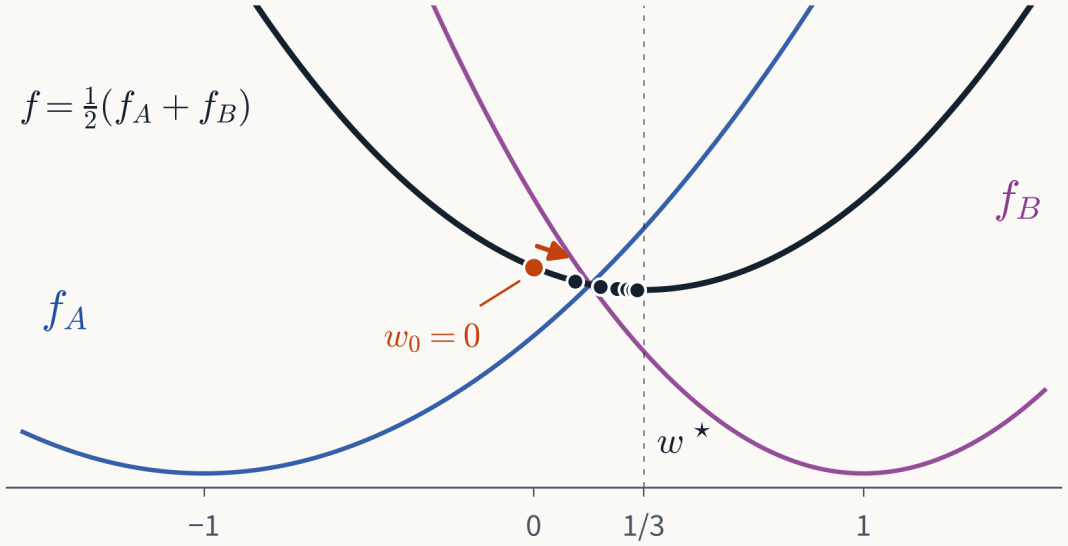
Different data

$\frac{1}{n}\sum_i \|\nabla f_i - \nabla f\|^2 \le \zeta^2$

Identical data ($\zeta = 0$) is the easy case. Different data ($\zeta > 0$) adds drift and bias to several methods below.

Notation used throughout: n workers · w parameters · η step size · σ^2 gradient-noise variance · ζ^2 heterogeneity · L smoothness · $F_0 = f(w_0) - f^*$.

Synchronous data parallelism is minibatch SGD



Toy: A's loss ½(w+1)², B's loss (w−1)², start at w = 0, η = ¼. Gradients 1 and −2 average to −½: both workers move to 1/8, then on to w* = 1/3.

Average the n gradients, apply the same step everywhere:

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{i=1}^{n} g_i(w_t)$$

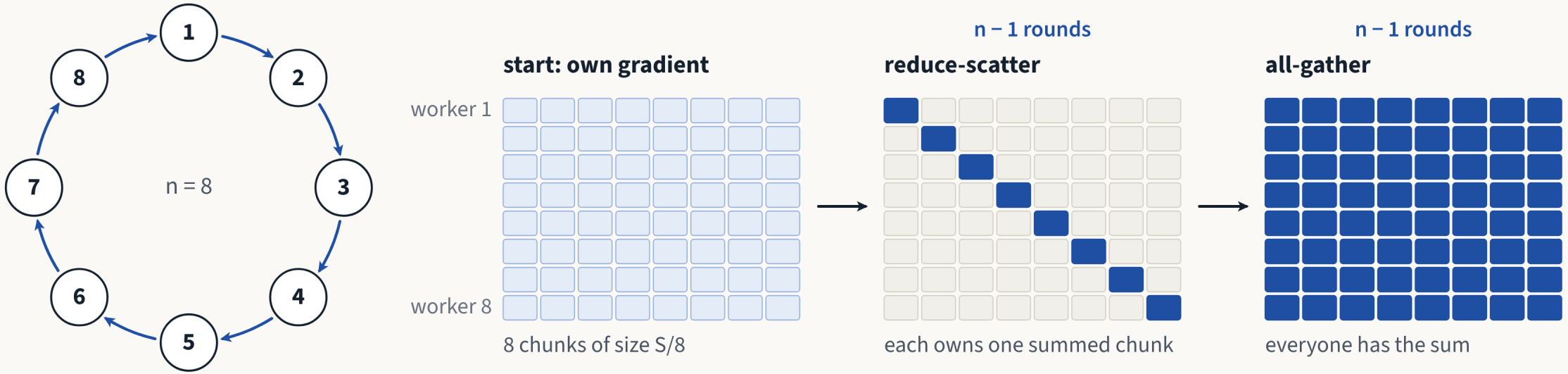
$$K(\varepsilon) = \mathcal{O}\left(\frac{\sigma^2}{n\,\varepsilon^2} + \frac{1}{\varepsilon} \right) L F_0$$

The noise term shrinks with n: up to n× fewer steps, until the 1/ε term dominates.

L-smooth nonconvex f; unbiased gradients, variance σ². K(ε): steps until the expected squared gradient norm ≤ ε.

Time per round: ring all-reduce

Split the gradient into n chunks. Pass partial sums around the ring, then pass the finished sums around again.



$$T_{\text{ring}} \approx 2(n-1)\,\alpha + \frac{2(n-1)}{n}\,\frac{S}{B}$$

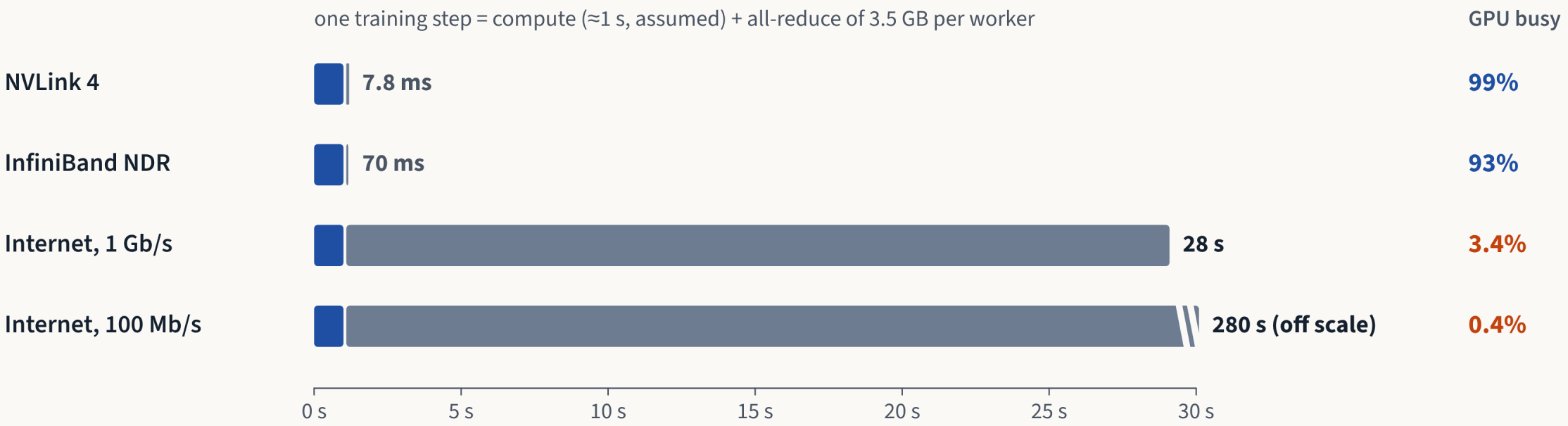
α: latency per round · S: gradient bytes · B: link bandwidth.
Each worker sends 2(n−1)/n · S ≈ 2S bytes.

Bandwidth-optimal: in any all-reduce some worker sends at least 2(n−1)S/n bytes, and the ring meets this at every worker. **Not latency-optimal:** 2(n−1) steps in sequence, against O(log n) for trees.

Patarasuk & Yuan 2009: reduction at the endpoints, full-duplex links, no contention. In-switch reduction (e.g. SHARP) sends ≈S.

A 1B-parameter model on 8 workers

A bf16 gradient is S = 2 GB, so each step's ring all-reduce moves 3.5 GB per worker.

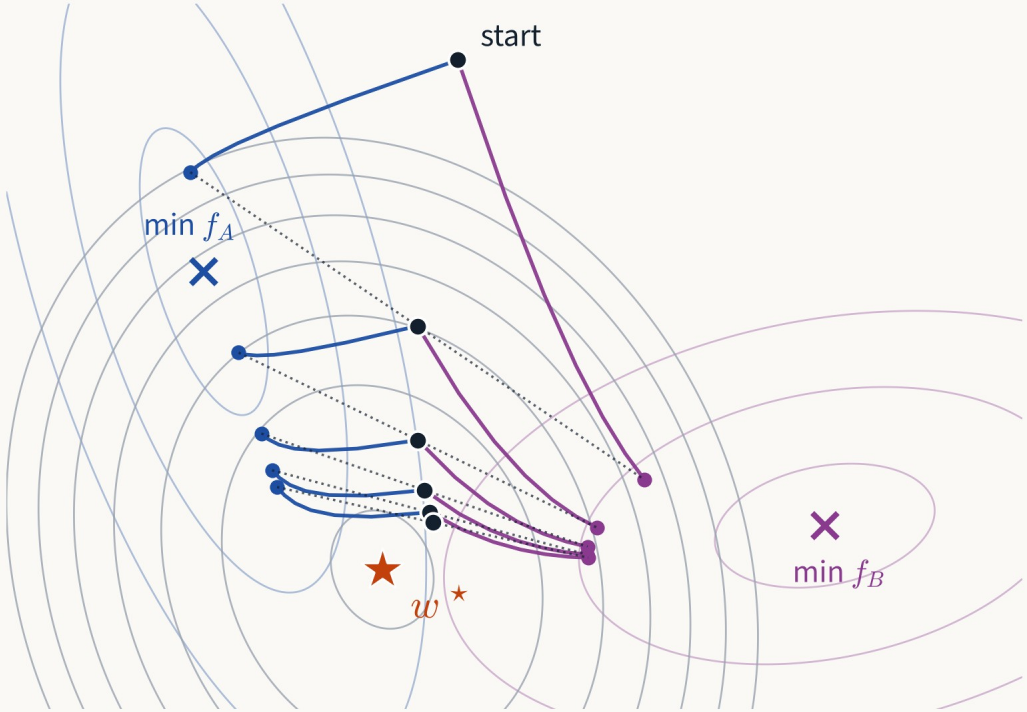


Over the internet, synchronous training leaves the GPU idle more than 96% of the time.

Best case: full line rate, latency ignored, no overlap; ≈1 s of compute per step assumed.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Local SGD



each worker i : $\quad w_i \leftarrow w, \qquad w_i \leftarrow w_i - \eta\, g_i(w_i) \quad (H \text{ times})$

then average: $\quad w \leftarrow \dfrac{1}{n} \sum_i w_i$

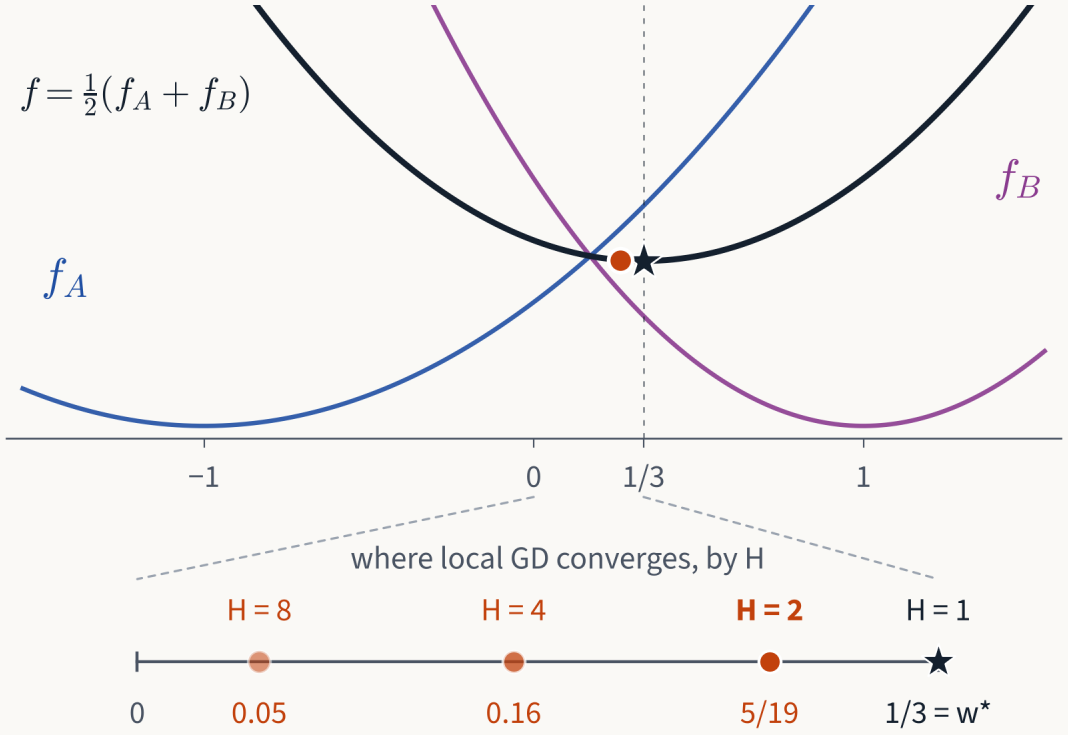
Rounds drop from K to K/H.

Toy: two local steps then one average reach 5/32; two synchronous steps reach 13/64. Same gradient count, a different algorithm.

McMahan et al., AISTATS 2017 (FedAvg) · Stich, ICLR 2019 · Figure: two quadratics, H = 6, five rounds.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

With different data, drift becomes bias



On the toy, local gradient descent with H = 2 and η = ¼ converges to **5/19 ≈ 0.26**, not to w* = 1/3. With H = 8 it converges to 0.05.

Within a round each worker heads for its own minimum; the average of the endpoints is not where the average gradient points.

Remedies: shrink η as training proceeds, or correct each worker's gradient with a control variate (SCAFFOLD, ProxSkip).

Heterogeneous data turns local steps from noisy into biased.

Karimireddy et al., ICML 2020 (SCAFFOLD) · Mishchenko et al., ICML 2022 (ProxSkip) · deterministic local GD on the toy.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

What theory says about local steps

One bound covers local SGD and, later, gossip (Koloskova et al., ICML 2020). Steps K(ε) until the expected squared gradient norm is at most ε:

$$K(\varepsilon) = \mathcal{O}\left(\underbrace{\frac{\sigma^2}{n\,\varepsilon^2}}_{\text{same as minibatch SGD}} + \underbrace{\frac{\zeta H + \sigma\sqrt{\rho H}}{\rho\,\varepsilon^{3/2}}}_{\text{drift and disagreement}} + \underbrace{\frac{H}{\rho\,\varepsilon}}_{\text{fewer, sparser averages}} \right) \cdot L F_0$$

same as minibatch SGD
speedup ∝ n

drift and disagreement
grows with H, ζ, 1/ρ

fewer, sparser averages
grows with H/ρ

Local SGD: H local steps per average, exact averaging (ρ = 1).

Gossip (later in Part I) reuses this bound with H = 1.

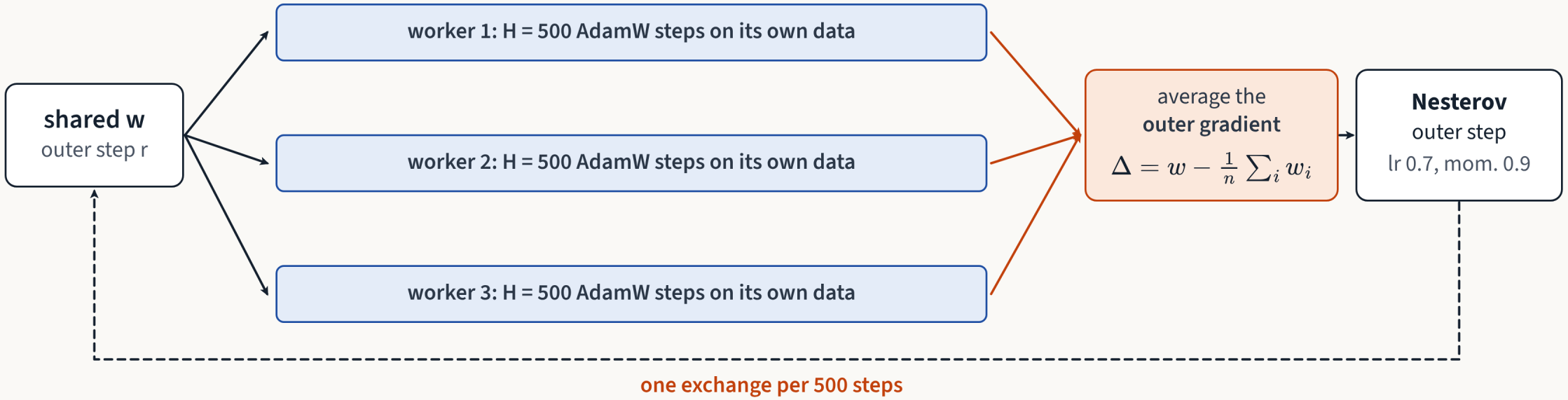
In the 2020 convex analysis, minibatch SGD with the same rounds and gradients matched or beat earlier local-SGD bounds; an improved one is better only for small ζ.

Local objectives L-smooth, nonconvex; variance σ^2, dissimilarity ζ^2; paper's M = 0, P = 1, τ = H, p = ρ. Woodworth et al., ICML and NeurIPS 2020.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

DiLoCo: AdamW inside, Nesterov outside

As in FedOpt, the change after H local steps is an “outer gradient” for an outer optimizer. FedAvg is the special case of outer SGD with step size 1, which is plain averaging; DiLoCo uses Nesterov momentum.



On C4, 8 workers matched fully synchronous training while communicating 500× less.

DiLoCo: Douillard et al., WANT@ICML 2024 · FedAvg: AISTATS 2017 · FedOpt: ICLR 2021. Running example: 1 s + 28 s/500 ≈ 1.06 s per step.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

DiLoCo has many variants, and a single-worker view

Overlap and asynchrony

Async Local-SGD: delayed Nesterov · Streaming DiLoCo: overlapped fragments · Decoupled DiLoCo: learners never wait for each other · HALoS: hierarchy across regions

Compressed outer gradients

Streaming DiLoCo: outer gradients in 4 bits · SparseLoCo: top-k, 2-bit values and error feedback

Outer optimizer and scale

outer learning rate > 1 can help · scaling laws: with 1–2 replicas, DiLoCo slightly beats data-parallel at 4B and 10B · SNOO: an outer Nesterov step helps even on one worker

One worker, every step: GPA

$$y \leftarrow \mu_y\, x + (1 - \mu_y)\, z$$

where the gradient is taken

$$z \leftarrow \mathrm{BaseOpt}\big(z,\ g(y)\big)$$

e.g. AdamW

$$x \leftarrow \mu_x\, x + (1 - \mu_x)\, z$$

smoothed iterate (evaluated)

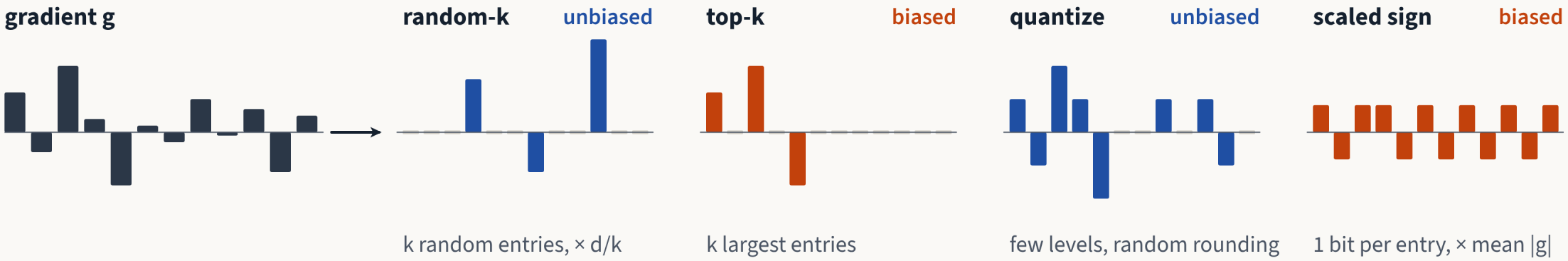
“A smoothed version of single-worker DiLoCo”: x averages the fast iterates z at every step, not every H steps. 8.7–10.1% fewer steps than AdamW at 160M–8B, with more FLOPs per step.

Variants: references. GPA: Defazio, Mishchenko, Raman, Shi, Xiao, arXiv 2512.17131 (2025), Eq. 8.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Compressing the message

Send C(g) instead of g. Two families of compressors behave very differently in the analysis.



Unbiased

$$\mathbb{E}\, C(g) = g, \qquad \mathbb{E}\|C(g) - g\|^2 \le \omega \|g\|^2$$

Biased (contractive)

$$\|C(g) - g\|^2 \le (1 - \delta)\, \|g\|^2, \quad 0 < \delta \le 1$$

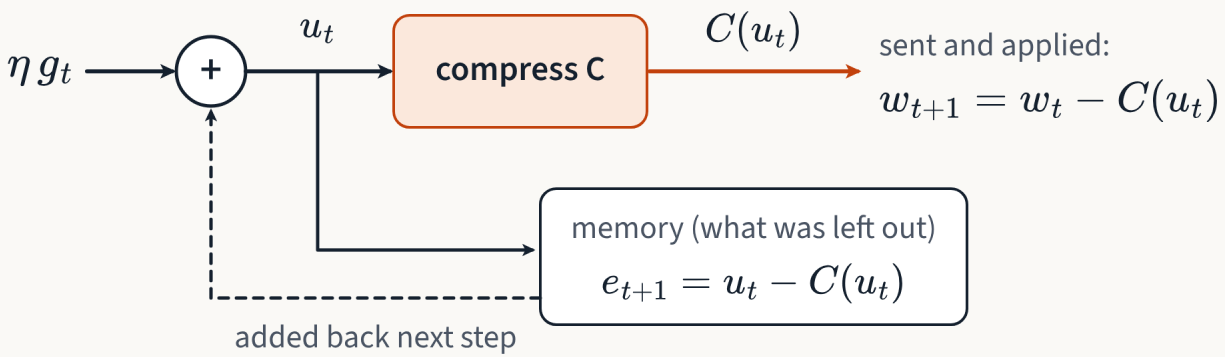
Unbiased compressors add variance. Biased ones are often better in practice, but usually need error feedback.

With local steps, workers send weights or weight changes instead of gradients, so compressors, delays and failures can behave differently. A random subset of the weights, for example, can be averaged without rescaling or error feedback, and a missed round only delays agreement.

QSGD: Alistarh et al. 2017 · signSGD: Bernstein et al. 2018 · scaled sign: Karimireddy et al. 2019 · top-k with memory: Stich et al. 2018.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Error feedback: remember what you did not send



Top-1 compressor, two steps

step	new	+ memory	sent
1	(3, 1)	(3, 1)	(3, 0)
2	(0, 1)	(0, 2)	(0, 2)

Total sent (3, 2) equals total proposed. Nothing is lost, only delayed.

$$\min_{t \le K} \mathbb{E}\|\nabla f(w_t)\|^2 \le \mathcal{O}\left(\frac{F_0 + LG^2}{\sqrt{K}}\right) + \underbrace{\mathcal{O}\left(\frac{L^2 G^2 (1-\delta)}{\delta^2 K}\right)}$$

same as plain SGD — **added by compression**

Error feedback delays information instead of losing it: compression adds only a term that decays like 1/K.

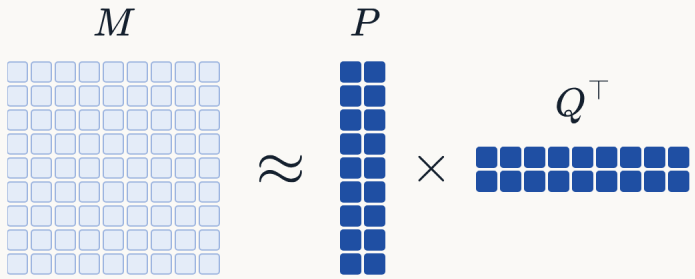
EF-SGD (Karimireddy et al., ICML 2019, Thm II): one worker, η = 1/√(K+1); G^2 bounds the gradient's second moment. Many workers: EF21.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

PowerSGD: low-rank messages that still average

PowerSGD, rank r (each worker, every step)

1. **for each weight matrix:** $M \leftarrow G + E$ ▷ *gradient plus error memory*
2. $P \leftarrow M\,Q$ ▷ *Q kept from the previous step*
3. $P \leftarrow \text{AllReduceMean}(P), \quad P \leftarrow \text{orthogonalize}(P)$
4. $Q \leftarrow M^{\top} P$
5. $E \leftarrow M - P\,Q^{\top}$ ▷ *local residual: error feedback*
6. $Q \leftarrow \text{AllReduceMean}(Q)$
7. apply the averaged low-rank update $P\,Q^{\top}$



values sent per matrix $m_1 m_2 \;\rightarrow\; r\,(m_1 + m_2)$

1024 × 1024 at r = 4: 128× fewer

Both all-reduces are linear in M: the factors of the average are the averages of the factors.

It works: at rank 2–4, the paper's 16-GPU runs reach the same accuracy as SGD while sending 120–136× less.

Vogels, Karimireddy, Jaggi, NeurIPS 2019 (Algorithms 1–2, Table 3). Running example at ~136×: 28 s of communication becomes ≈0.2 s per step.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

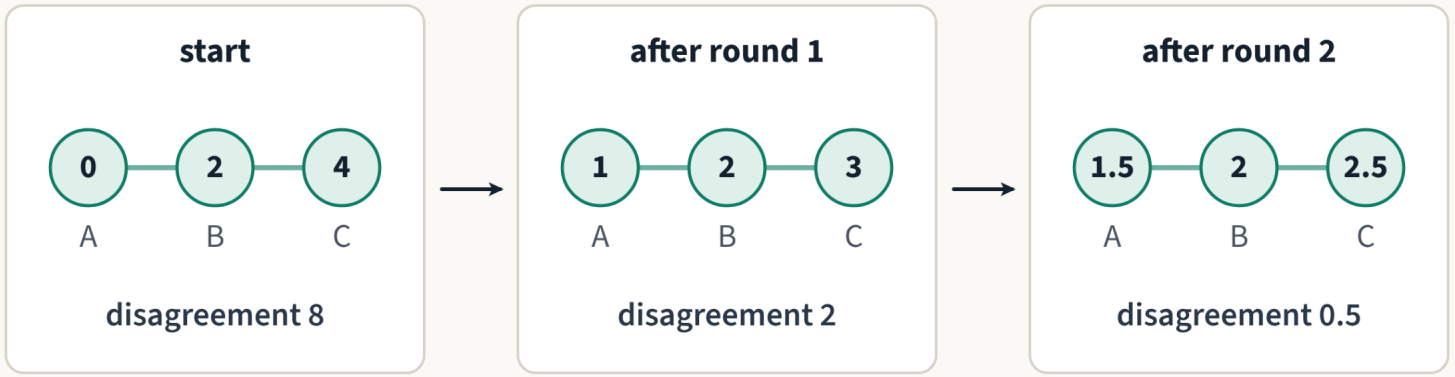
Gossip: average with your neighbors

Each worker replaces its parameters by a weighted average of its neighbors' parameters: w ← Ww, one row of W per worker.

mixing matrix

$$W = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

rows and columns sum to 1, so the mean is preserved



disagreement = $\sum_i (w_i - \bar{w})^2$ shrinks 4× per round; the mean stays at 2

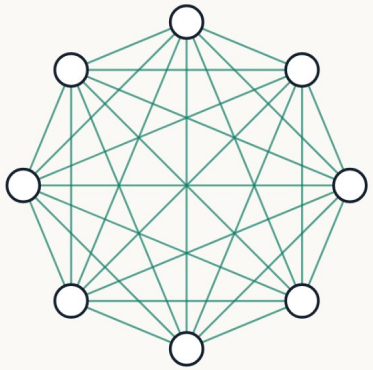
The mean never moves. Disagreement shrinks geometrically, at a rate set by the graph.

No global collective: a worker that fails only affects its neighbors, which carry on with the rest, as long as the graph stays connected.

Xiao, Boyd 2004; Boyd et al. 2006: averaging time ≈ log(1/ε)/(1 − λ); λ: second-largest |eigenvalue| of W; 1 − λ: the spectral gap.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

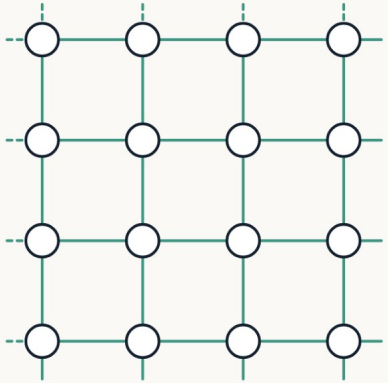
The graph sets the speed of agreement



complete graph

each node talks to n − 1

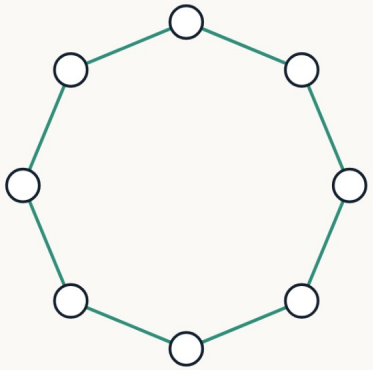
spectral gap **1**



2-D torus

each node talks to 4

spectral gap **Θ(1/n)**



ring

each node talks to 2

spectral gap **Θ(1/n²)**

$$w_i \leftarrow \sum_j W_{ij}\left(w_j - \eta\, g_j(w_j)\right)$$

Decentralized SGD (Koloskova et al. 2020 form): a local step, then one gossip round. Same bound with H = 1 and ρ = 1 − λ² ∈ [gap, 2·gap]: the extra terms grow at most like 1/ρ.

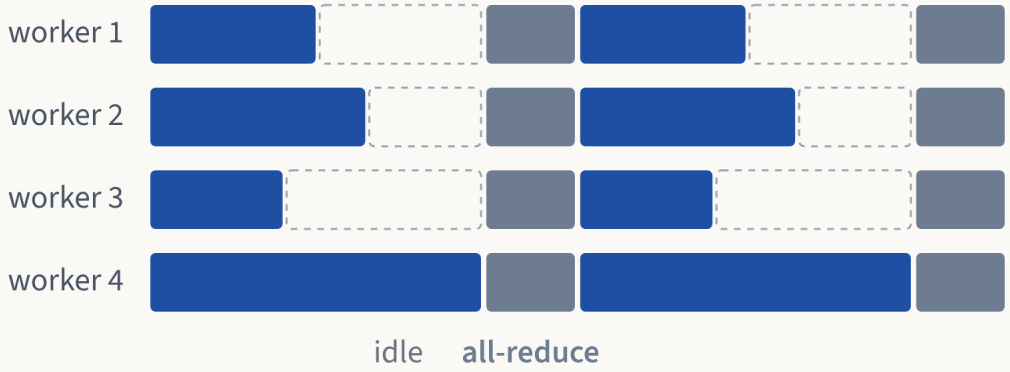
Gossip replaces the global all-reduce with neighbor exchanges, but not bytes: a ring all-reduce already sends ≈2S per worker (S: gradient size in bytes).

Lian et al., NeurIPS 2017 · Koloskova et al., ICML 2019 and 2020. Ring with weights ⅓: gap = (4/3)·sin²(π/n) ≈ 4π²/3n².

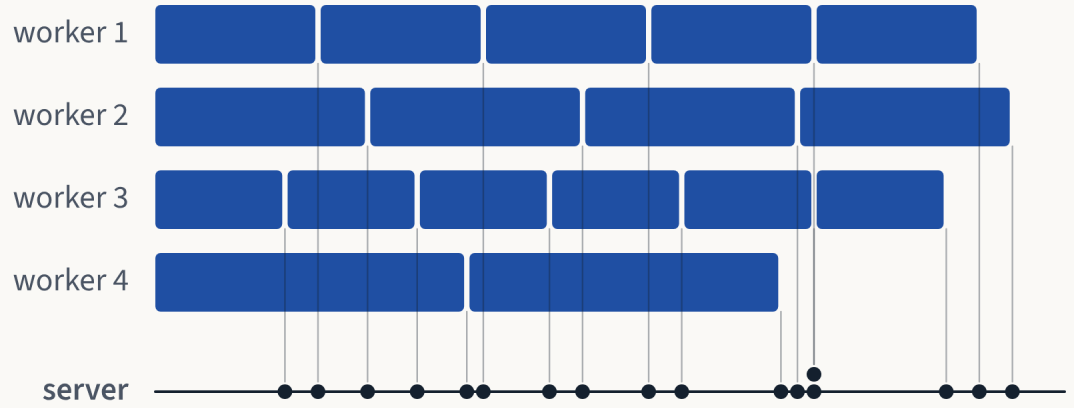
$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Stragglers, and asynchronous SGD

Synchronous: everyone waits for the slowest



Asynchronous: apply each gradient on arrival



worker 4 started at version 0; 4 updates happened meanwhile → delay τ = 4

$$w_{k+1} = w_k - \eta \nabla f\left(w_{k-\tau_k};\, \xi_k\right)$$

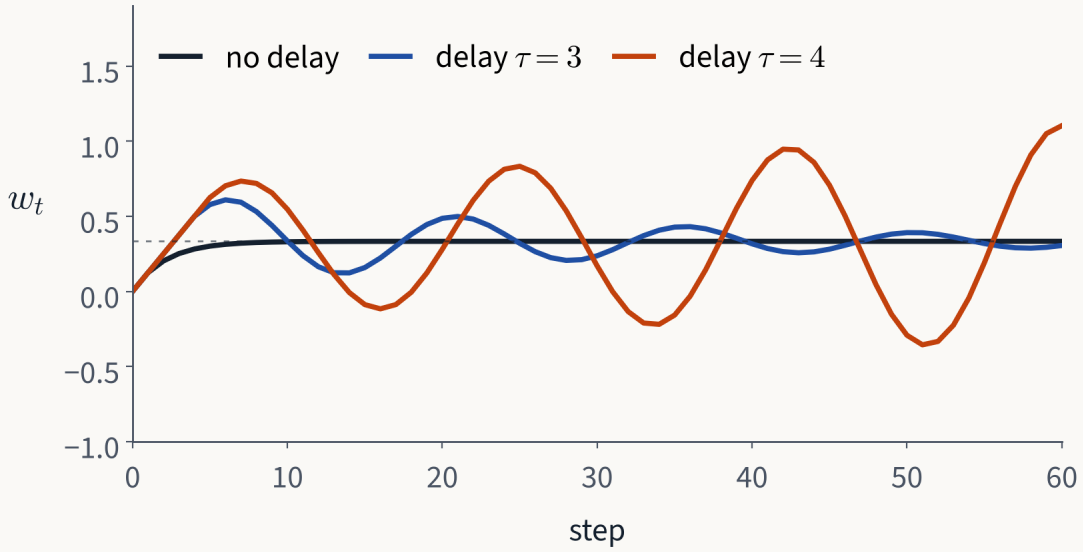
Each gradient is applied to the current parameters, but was computed several updates earlier; the delay τ varies from step to step.

Agarwal & Duchi, NeurIPS 2011 · Lian et al., NeurIPS 2015.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

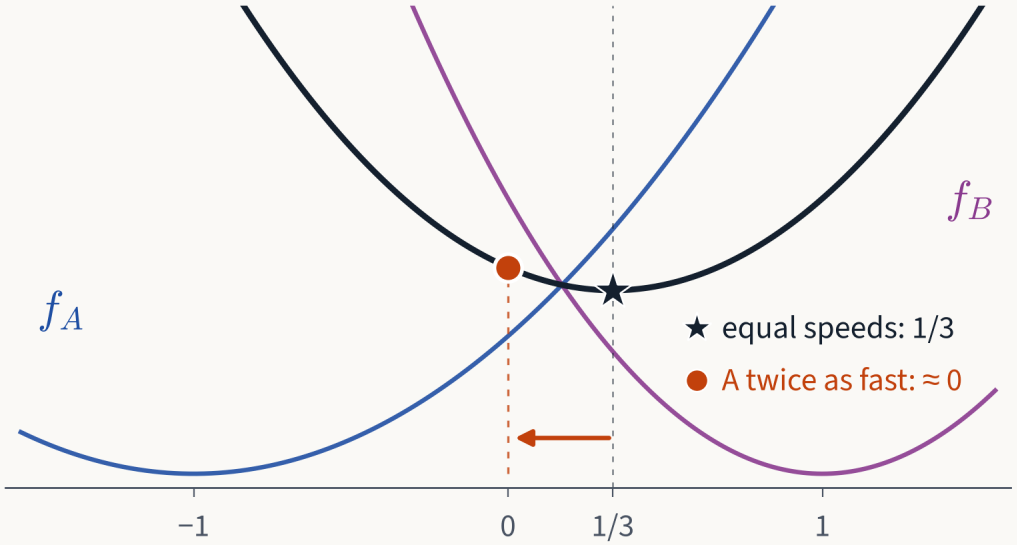
Two ways staleness hurts

1 · Delay shrinks the range of stable step sizes



Full gradients, fixed delay τ, η = ¼ (ηL = 3/8). Stable if and only if ηL < 2 sin(π/(2(2τ+1))): true at τ = 3, false at τ = 4.

2 · Speed reweights the objective



If A sends gradients twice as often, SGD descends twice A's loss plus B's on average: the iterates hover around w = 0, not 1/3.

Delays force smaller steps. With different data, unequal speeds change which objective is minimized.

Toy: f is the average of A's and B's quadratics, L = 3/2. The stability condition is a classical result for delayed linear recursions (backup slide).

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

What theory says about delays

From delay-dependent iteration bounds to wall-clock-optimal methods. L-smooth nonconvex f, gradient variance σ^2.

iterations to ε: how the delay enters

worst delay

$\tau_{\max}$

the maximum delay sets the rate

classical analyses

→

average delay

τ_{avg}

delay-adaptive step sizes

Koloskova et al. 2022

→

number of workers

n

no delay term; bound never above minibatch

Mishchenko et al. 2022

wall-clock time to ε: seconds per gradient c

→

fastest m workers

$c_1 \le \cdots \le c_m$

optimal time may skip the slowest (Rennala)

Tyurin, Richtárik 2023

→

fully asynchronous

$\tau < R \approx \sigma^2/\varepsilon$

same optimal time; drops older gradients

Ringmaster ASGD 2025

→

different data

c_n

every worker counts: slowest and mean c

Ringleader ASGD 2026

Same data: the time-optimal method may ignore the slowest workers. Different data: it must count every worker.

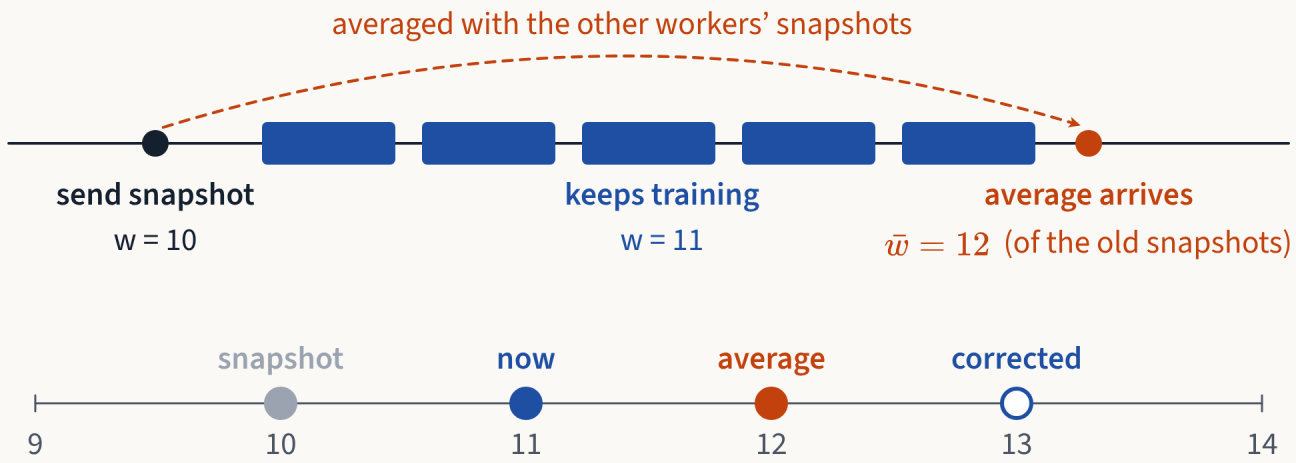
Boxes 1–5: identical data. Ringmaster: first fully asynchronous SGD with optimal time; Ringleader: the same when data differ.

PR

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Overlap: keep training while the average travels

one coordinate of one worker



overwrite

$$w \leftarrow \bar{w} = 12$$

loses the local +1

keep your own progress

$$w \leftarrow w + (\bar{w} - w_{\text{snap}}) = 11 + (12 - 10) = 13$$

replicas still differ by their local progress

Overlapped averaging with the delta rule (each worker)

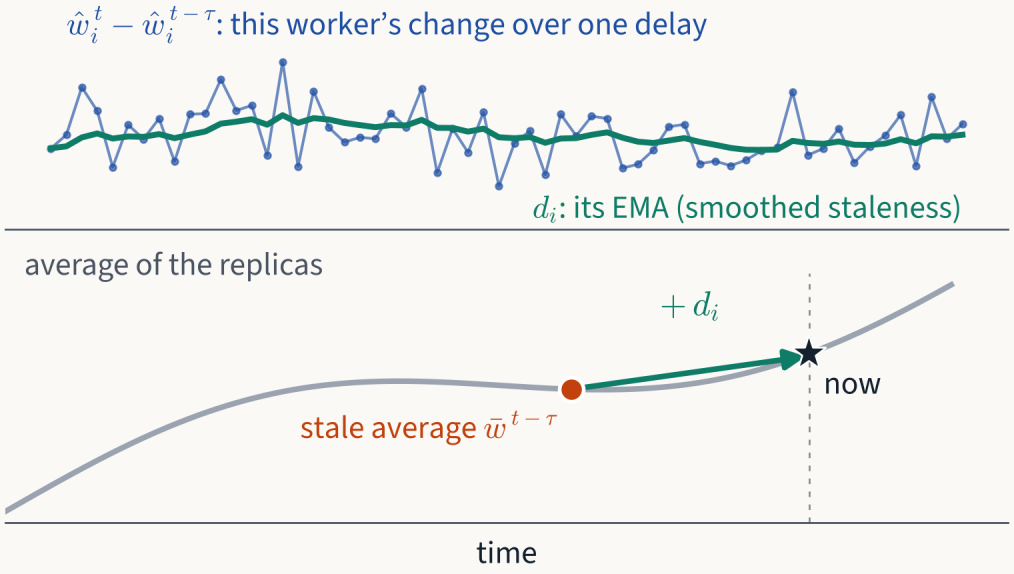
1. **every R steps:** $w^{\text{snap}} \leftarrow w[S]$; start averaging it with the other workers
2. **every step:** local optimizer step on w (training never stops)
3. **when the average $\bar{w}$ arrives:** $w[S] \leftarrow w[S] + \left(\bar{w} - w^{\text{snap}}\right)$

Agora's AsyncSPARTA uses this rule: the λ = 1 case of AsyncMesh (next slide). Also LOSCAR-SGD (NeurIPS 2026); Streaming DiLoCo blends.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

AsyncMesh: correct with smoothed staleness

The late average describes step t − τ. Each worker adds d, an exponential moving average of its own staleness: how far its weights moved over one delay window.



AsyncMesh, data-parallel side (worker i)

1 **every step:** $\hat{w}_i \leftarrow$ local optimizer step

2 **at each averaging interval:** start averaging a new random 5% subset

3 **when** the average $\bar{w}^{t-\tau}[S]$ **arrives:**

4 $\quad d_i[S] \leftarrow (1-\lambda_t)\, d_i[S] + \lambda_t \big(\hat{w}_i^t - \hat{w}_i^{t-\tau}\big)[S]$ ▷ *smoothed staleness*

5 $\quad w_i[S] \leftarrow \bar{w}^{t-\tau}[S] + d_i[S]$ ▷ *λ = 1: the delta rule*

With the same initialization, hyperparameters and i.i.d. data, every replica's expected staleness is the same, so the stale average plus d estimates the current mean.

Fault tolerant. If an averaging round is missed, or only part of a tensor is averaged, workers keep their local values for the rest and carry on, with little effect on training. Only frequent failures slow it (Part III).

One proof idea behind these rates

Find a virtual sequence that runs plain SGD, then bound how far from it the gradients are evaluated.

$$\frac{1}{K}\sum_{t<K}\mathbb{E}\|\nabla f(\tilde{w}_t)\|^2 \lesssim \frac{F_0}{\eta K} + \eta L\sigma_{\text{eff}}^2 + \underbrace{L^2\frac{1}{K}\sum_{t<K}\mathbb{E}\|\tilde{w}_t - x_t\|^2}_{\text{deviation: how far the gradients are taken from the virtual SGD sequence}}$$

Local SGD, gossip

virtual sequence

$\tilde{w}_t = \bar{w}_t = \frac{1}{n}\sum_i w_{i,t}$

gradients taken at

$x_{i,t} = w_{i,t}$, each i

disagreement between replicas

Error feedback

virtual sequence

$\tilde{w}_t = w_t - e_t$

gradients taken at

$x_t = w_t$

size of the stored residual

Asynchronous SGD

virtual sequence

$\tilde{w}_t = w_t$

gradients taken at

$x_t = w_{t-\tau_t}$

how far w moved during the delay

Drift, disagreement, residuals and staleness all enter as the same deviation term.

Mania et al. 2017 · Stich & Karimireddy 2020 · Koloskova et al. 2020. η ≤ 1/L; middle term: noise of the update, e.g. σ^2/n.

What each family delivers, and runs that combine them

Fewer rounds

The workhorse: DiLoCo matched synchronous training with 500× less communication.

Smaller messages

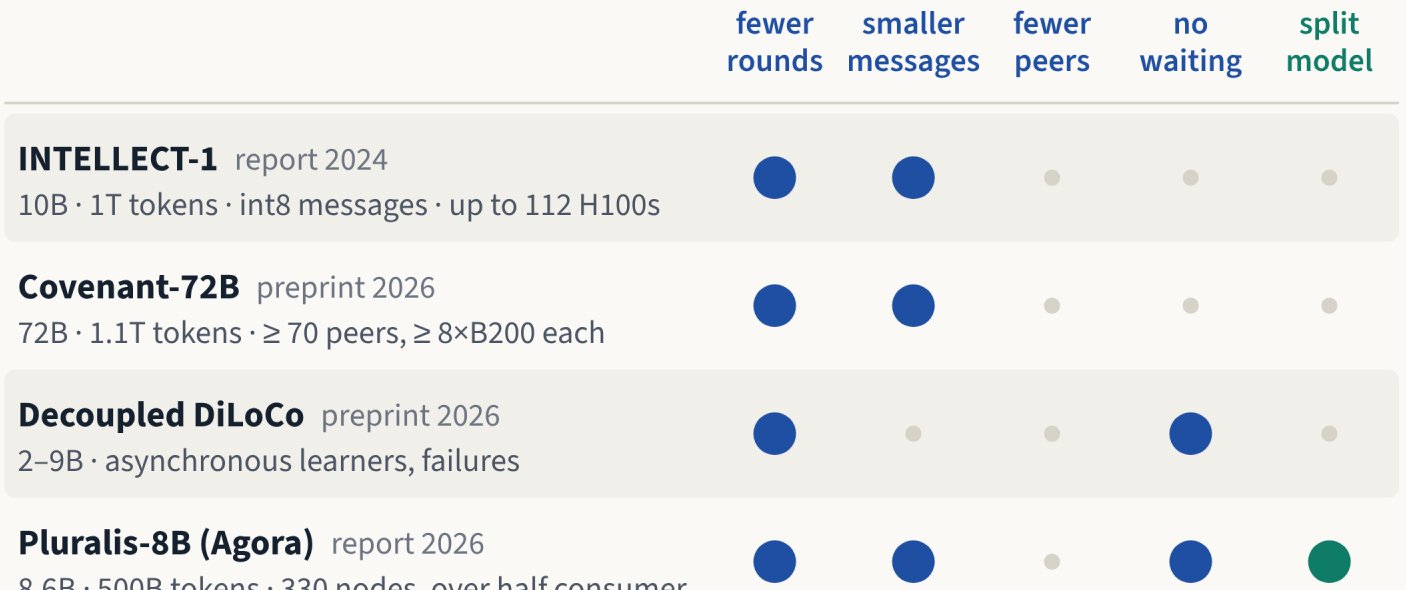
Help on slow links. In datacenter setups, compression sped up only 6 of 200+ cases.

Fewer peers

SGP beat all-reduce on 10 Gb/s Ethernet: 1.7 h vs 5.1 h on 256 GPUs, at 75.0% vs 76.2% accuracy. None of the runs here uses it.

No waiting

Overlap is now standard; asynchrony also rides out failures (88% vs 58% goodput, simulated).



These runs combine two or more families, most often local steps with compression.

Agarwal et al., MLSys 2022 · SGP: Assran et al., ICML 2019 · goodput: Decoupled DiLoCo (2026).

Part I in one table

Per-step time only. The “Introduces” column is what increases K(ε).

Method	Reduces	Introduces	Assumes	Seconds per step at 1 Gb/s, before any change in K
Synchronous all-reduce	—	—	—	≈ 29 s
Local SGD, DiLoCo (H = 500)	rounds	drift (H, ζ)	small ζ, moderate H (theory)	≈ 1.06 s
PowerSGD + error feedback	bytes per round	compression error	a good low-rank fit	≈ 1.2 s
Gossip, D-PSGD (ring)	barrier, latency	slower agreement (1/ρ)	connected graph	≈ 33 s
Asynchronous SGD	idle time	staleness (τ)	same data, or equal weighting (Ringleader)	hides waiting
Overlapped sparse averaging	bytes and idle time	stale averages	a merge rule	≈ 1 s

Running example: 1B, 8 workers, ≈1 s compute per step, full line rate. Sparse averaging: 5% every 20 steps, overlapped (≈0.07 s of traffic per step).

BREAK · 10 MINUTES

Back in ten minutes

After the break: what if the model does not fit on one GPU?

Training an 8B-parameter model needs ≈128 GB for weights, gradients and Adam states — before activations. A high-end consumer GPU has 24–32 GB.

PR

PART II

The model is split across workers

Activations now cross the network inside every training step. The same two ideas apply: smaller messages across each boundary, and no waiting for them.

1 · Smaller boundary messages

dense message



compressed message



↓ boundary bytes

↑ a constrained architecture

Subspace networks

2 · No waiting

wait for the message



keep computing



↓ pipeline bubbles

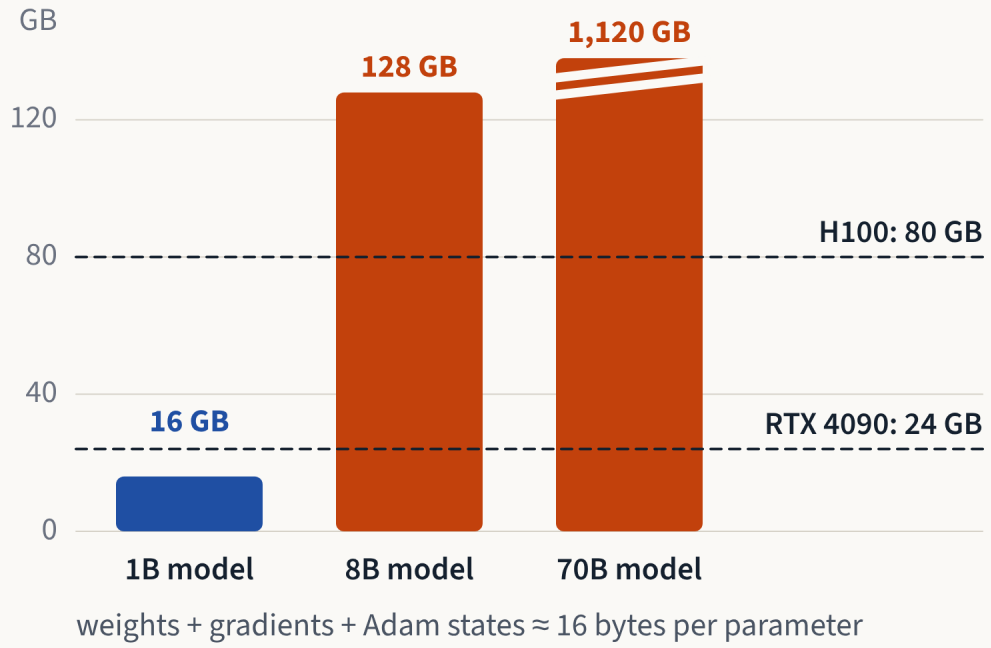
↑ stale gradients

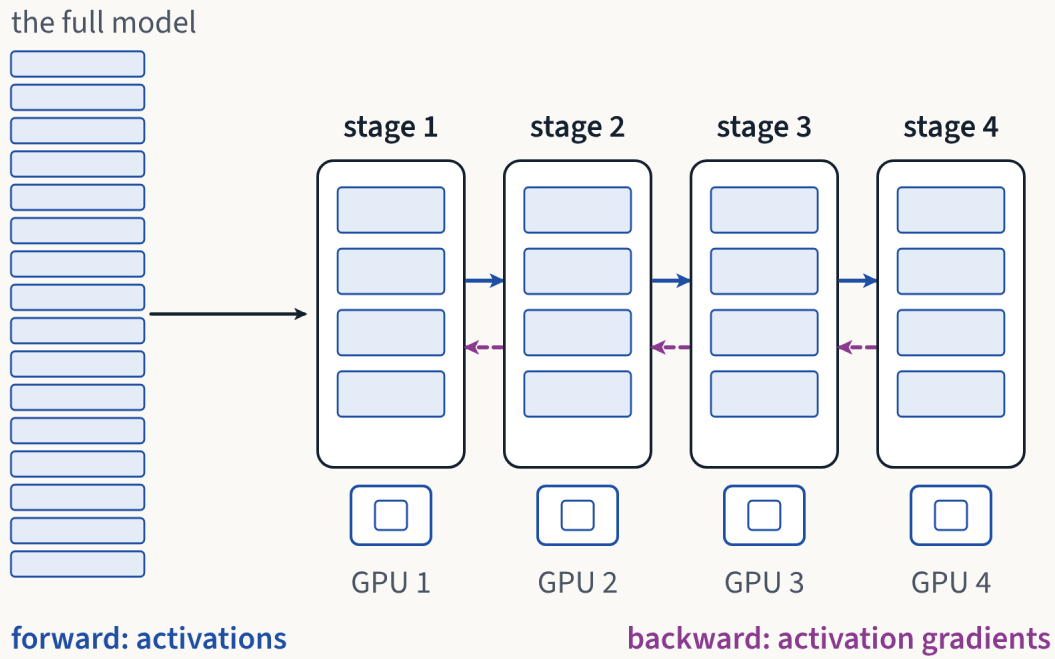
Asynchronous pipelines, Nesterov look-ahead

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

If the model does not fit, split it into stages

Training state costs ≈16 bytes per parameter before activations, so above ≈1.5B parameters it no longer fits on a 24 GB GPU.



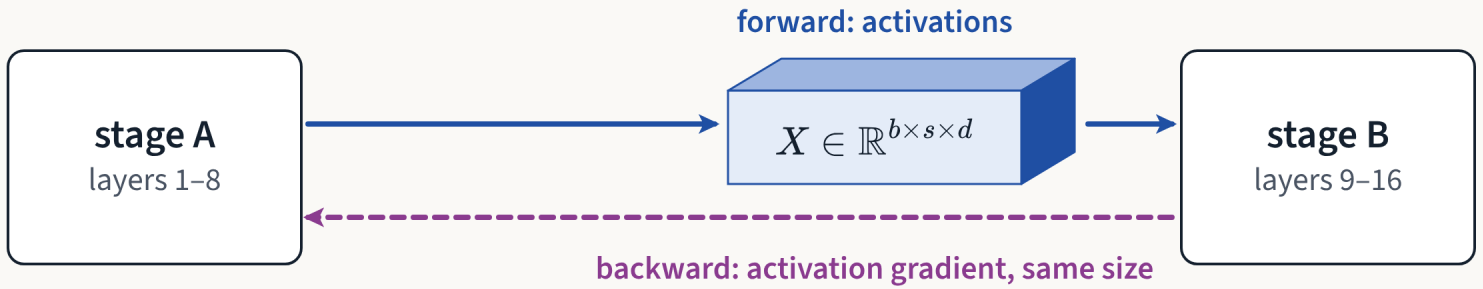


With P stages, each GPU holds 1/P of the model — but activations now cross every stage boundary.

16 B/param: bf16 weights, gradients (2 + 2); fp32 master weights, Adam moments (4 + 4 + 4), as in ZeRO.

What crosses a stage boundary

Every microbatch sends its activations forward and receives an activation gradient of the same size back — at every boundary, in every step.



one sequence (b = 1), s = 2048 tokens, width d = 4096, bf16:

16 MiB per message

0.13 s at 1 Gb/s

1.3 s at 100 Mb/s

per direction, per boundary, per microbatch

Unlike gradient averaging, these messages are on the critical path: the next stage cannot start without them.

Nominal line rate; latency, framing and congestion ignored. Inside a datacenter (400 Gb/s or NVLink) the same message takes under a millisecond.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Why not compress activations like gradients?

A lossy compressor at each boundary adds an error to the next stage's input, and later stages can amplify it.



$$\tilde{x}_B = x_A + e_A$$

$$\tilde{x}_C = f_B(x_A + e_A) + e_B$$

later stages can amplify earlier errors

Compressing a gradient
perturbs one update; error feedback adds it back later

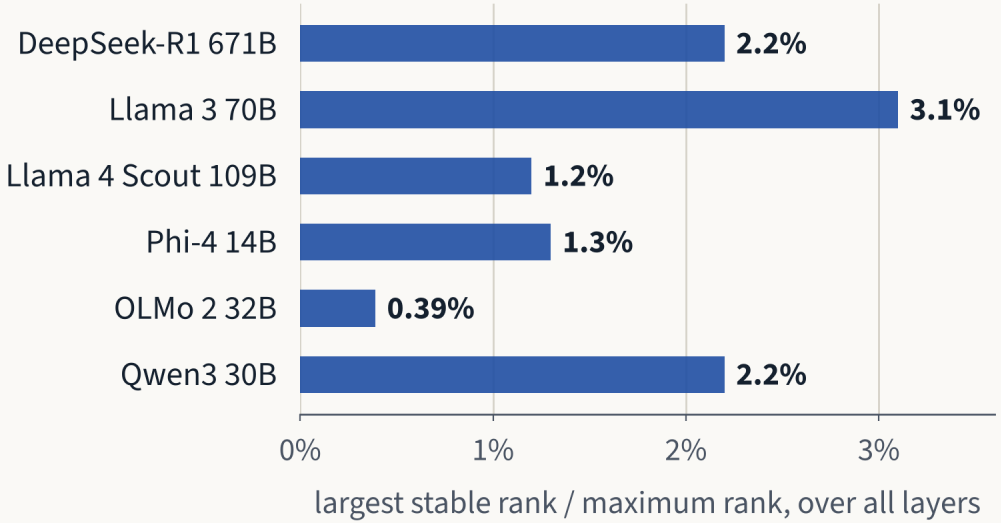
Compressing an activation
perturbs the function itself, at every stage, every step

Instead of approximating arbitrary activations, change the model so that a short message is exact.

Bian et al., MLSys 2024 (activation-compression benchmark) · AQ-SGD, NeurIPS 2022 · error propagation: Ramasinghe et al., NeurIPS 2025, App. B.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Output projections have low stable rank



Stable rank — squared Frobenius norm over squared spectral norm — is a soft count of the directions that carry most of the energy.

In six released models, attention and MLP output projections stay at ≈3% of their maximum possible rank or less, in every layer.

$$X^{(\ell)} - \underbrace{E_{\text{tok}}[\text{token}] + E_{\text{pos}}}_{\text{token and position embeddings}} = \underbrace{\sum_{j \le \ell} \left(\text{attn}_j + \text{mlp}_j\right)}_{\text{low stable rank}}$$

token and position embeddings
high rank, but set by the token IDs

low stable rank
every term leaves through an output projection

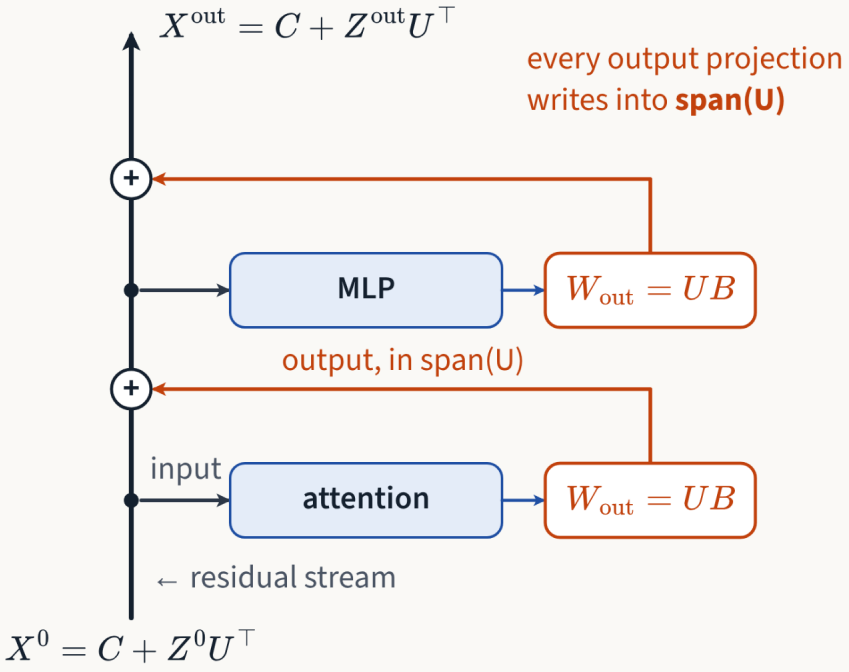
Trained weights seem to converge to a low-rank solution. Subspace networks use it to compress activations.

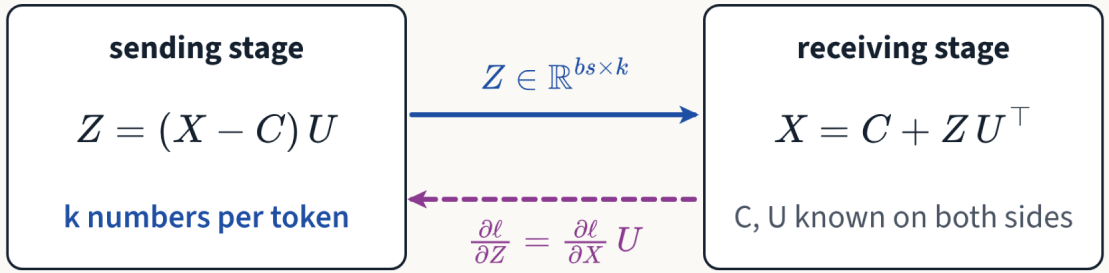
Ramasinghe et al., NeurIPS 2025; arXiv 2506.01260v3, §4.1–4.2, App. G, Fig. 16. E_pos only where positions are added to the stream.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Subspace networks: send k numbers per token, not d

Constrain every attention and MLP output projection to the form UB, with one shared U of k orthonormal columns. Then the residual stream stays $X = C + ZU^{\top}$, and only the coordinates Z travel.





The constraint

Both directions carry k instead of d numbers per token, exactly.
But everything the layers add to the residual stream
lies in the same k directions.

Exact for the constrained model: d/k ≈ 100× fewer values in both directions (d = 4096, k = 40).

Ramasinghe et al., NeurIPS 2025, §4.2–4.4. C: fixed token and position embeddings, known to every stage.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Keeping a subspace network exact during training

C is the fixed part of the input; three rules keep the rest in span(U) while U itself slowly moves.

$$X^{(\ell)} = \underbrace{T_{\text{fixed}}[\text{token}] + E_{\text{pos}}}_{\text{fixed, high rank: subtract it}} + \underbrace{T_S[\text{token}] + \sum_{j \le \ell} \left(\text{attn}_j + \text{mlp}_j\right)}_{\text{trainable, low rank: inside the shared subspace}}$$

fixed, high rank: subtract it
stored on every node, rebuilt from the token IDs

trainable, low rank: inside the shared subspace
under SSN, exactly k numbers per token cross the link

1 attention output projection

$$W_O \leftarrow U U^\top W_O$$

Its gradient leaves span(U), so it is projected back after every step: ≈ 0.05 s, about 1% of a forward pass.

2 MLP output projection

$$w_j \leftarrow w_j - \eta\, m_j / \sqrt{\bar{v}_j + \epsilon}$$

AdamW with one step size per hidden unit, from the mean of its second moments: the update stays in span(U).

3 the shared subspace U

$$U \leftarrow \text{qr}\left(U - \eta\,(I - UU^\top)\nabla_U \mathcal{L}\right)$$

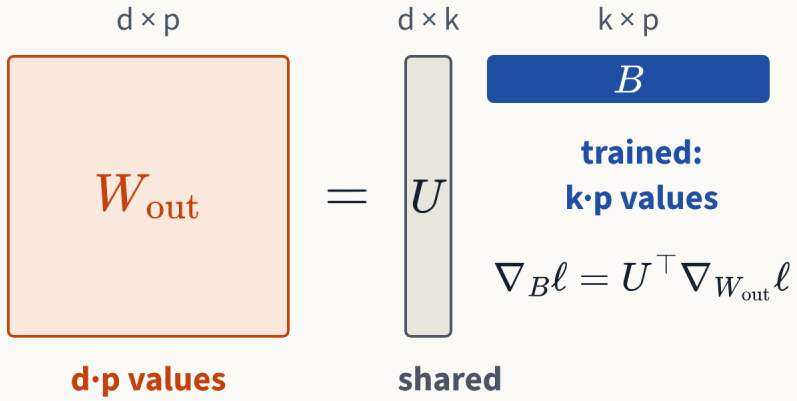
Every 500 steps: one Riemannian step on $\mathcal{L}$, the gradient energy outside span(U), then QR; the new U is sent to every node.

Ramasinghe et al., NeurIPS 2025: embeddings §4.3.1 · projection and AdamW §5, App. A · Grassmann step §4.5, §6.

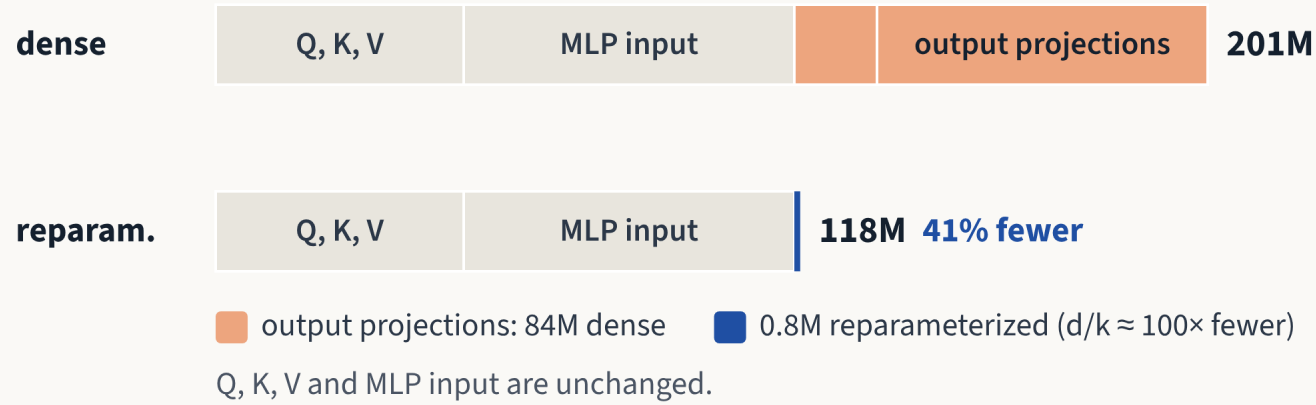
$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Reparameterized SSN: train k·p values, not d·p

Write each output projection as UB and train B directly. The constraint then holds by construction: no projection step, and AdamW unchanged.



Trainable weights in one block (d = 4096, k = 40, 4d MLP)

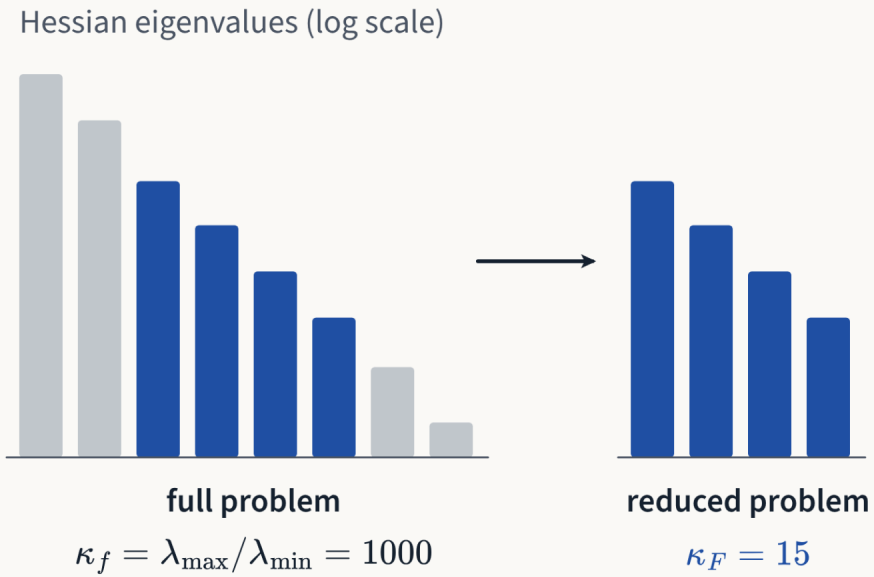


The trainable token table is factored the same way. Agora trained its 8B model in this form: convergence unaffected, gradients more stable.

Avraham et al., Agora, arXiv 2607.13332 (2026), §4.1, Eq. 3. p: input width, d for attention and 4d for the MLP; biases and norms omitted.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Why a constraint need not hurt optimization



A reduction mapping optimizes over a manifold that contains good solutions: F(u) = f(u, Ψ(u)).

If the removed directions carry the extreme curvature, smoothness falls and the Morse–Bott (local PŁ) constant rises, in the metric the mapping induces:

$$\kappa_F < \kappa_f \;\Rightarrow\; \text{preconditioned GD: } \mathcal{O}\big(e^{-t/\kappa_f}\big) \;\to\; \mathcal{O}\big(e^{-t/\kappa_F}\big)$$

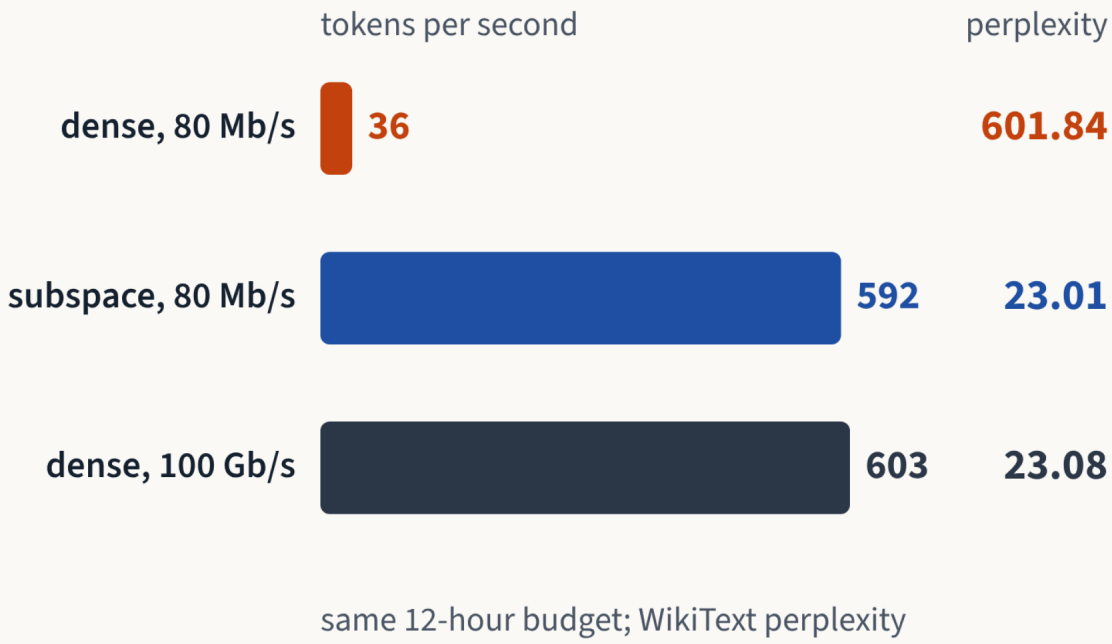
Local results near a solution manifold, in the mapping's metric, for GD rather than SGD. In the Euclidean metric smoothness can even get worse.

If solutions lie in a reduced space, such as low-rank weights, optimizing within it may improve conditioning.

Markou, Ajanthan, Gould, NeurIPS 2025, Thms 1–3, Cor. 3.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

At 80 Mb/s, subspace networks keep datacenter throughput



≈2B parameters · 8 stages · 12-hour budget · simulated 80 Mb/s between stages, against a 100 Gb/s reference

Dense messages at 80 Mb/s: ≈17× fewer tokens per second, so after 12 hours the model has barely trained.

Subspace messages: 592 vs 603 tokens/s, and perplexity comparable to the datacenter run (23.01 vs 23.08).

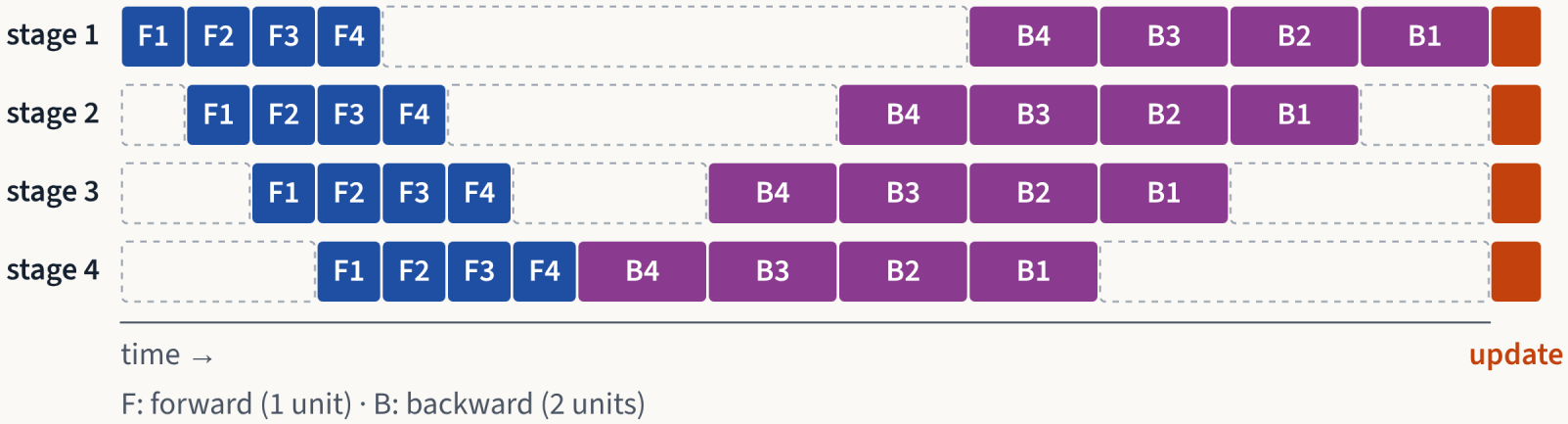
There are extreme cases where this low-rank constraint restricts model capacity.

Ramasinghe et al., NeurIPS 2025, Table 1 (WikiText perplexity; d = 4096, k = 40).

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Microbatches keep stages busy — until the step ends

A 164 kB subspace message takes ≈16 ms to serialize at 80 Mb/s; latency and pipeline flushes can then dominate the wait. GPipe splits each minibatch into M microbatches, then flushes before one synchronous update.



idle fraction (bubble)

$$\frac{P-1}{M+P-1}$$

P stages, M microbatches

P = 4, M = 4: 43% idle
each stage: 9 of 21 units idle

P = 4, M = 16: 16% idle

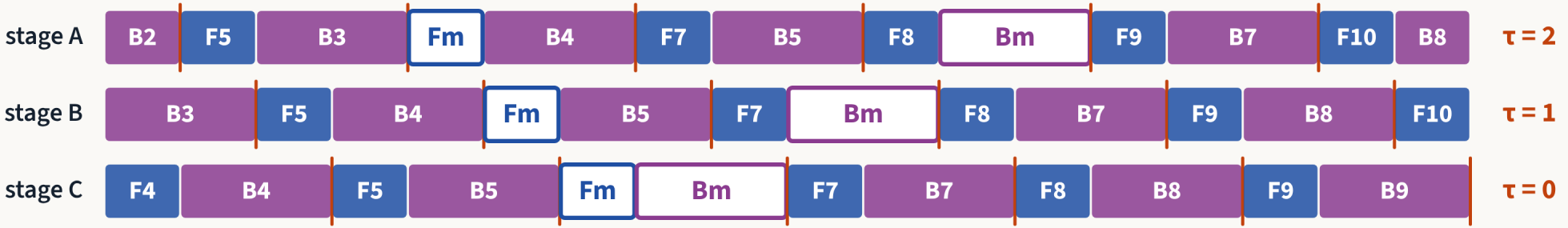
The bubble is the idle term of the cost model. More microbatches shrink it, but each microbatch's activations wait in memory until its backward pass.

GPipe (Huang et al., NeurIPS 2019). Backward = 2 × forward; transfer time ignored. The 1F1B schedule reduces memory, not the bubble.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Asynchronous pipelines: no flush, but stale gradients

Let every stage update its weights as soon as its backward pass finishes. In a balanced steady state no stage waits for a flush, but microbatch m's gradient lands after other updates.



orange ticks: weight updates · outlined: the microbatch m we follow

F: forward (1 unit) · B: backward (2 units)

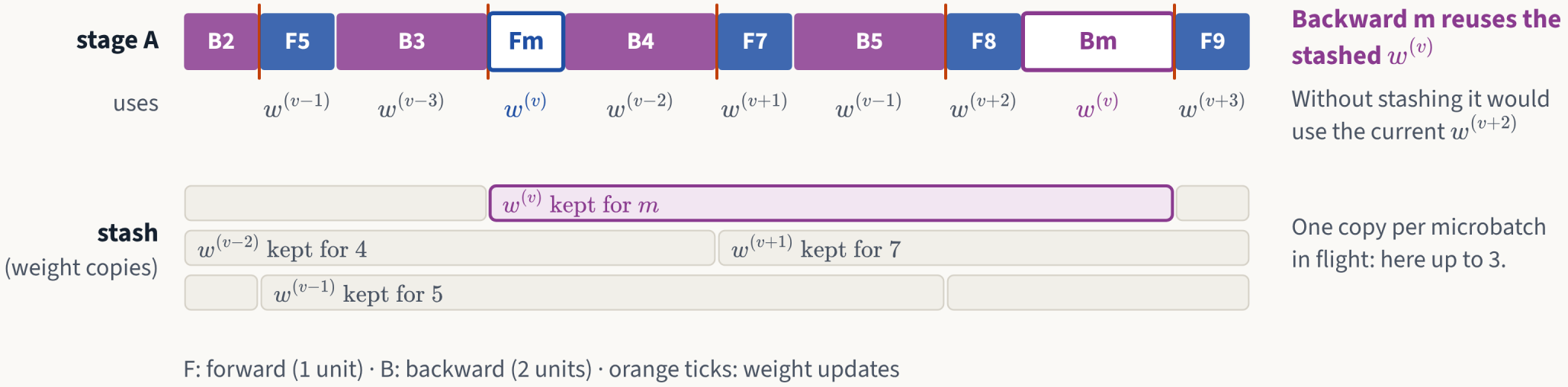
Idle time becomes staleness: earlier stages apply older gradients (here τ = 2, 1, 0).

PipeDream (SOSP 2019); Ajanthan et al., ICML 2025, §2. Equal stages, backward = 2 × forward; warm-up and transfer time ignored.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Weight stashing fixes consistency, not delay

Between microbatch m's forward and backward, stage A has updated its weights twice. Which weights should the backward pass use?



Backward m reuses the stashed $w^{(v)}$

Without stashing it would use the current $w^{(v+2)}$

One copy per microbatch in flight: here up to 3.

F: forward (1 unit) · B: backward (2 units) · orange ticks: weight updates

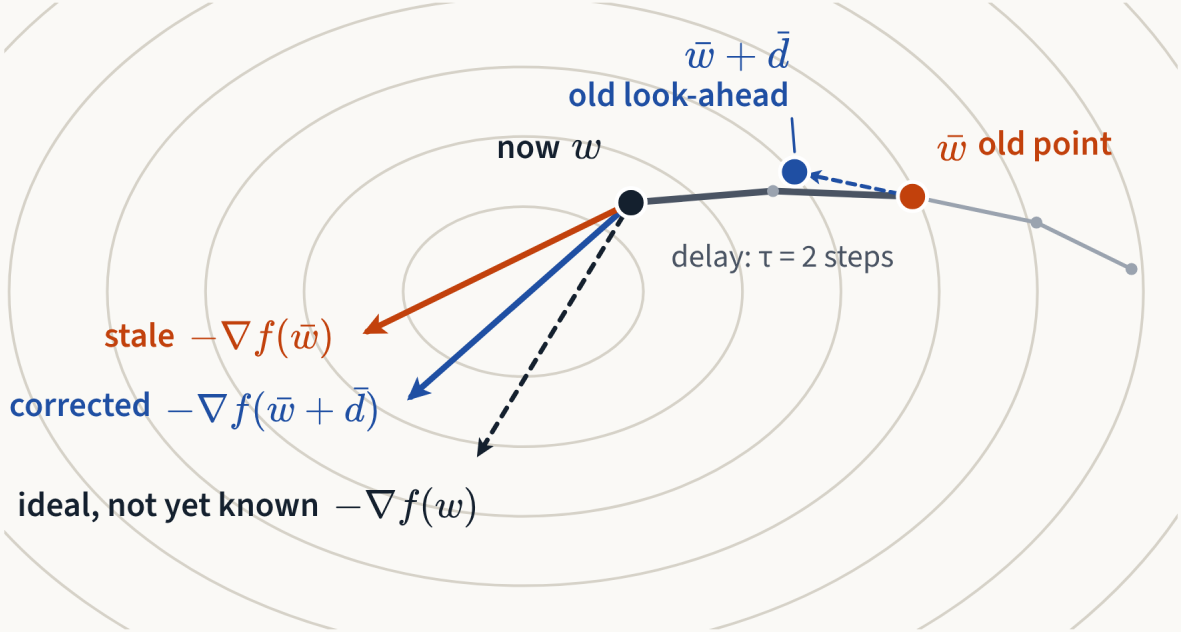
Stashing makes each gradient exact for the weights each stage used. It is still τ updates late.

PipeDream §3.3 (weight stashing); Ajanthan et al., ICML 2025, §§2.2, 3.1. Stashing costs memory: one weight copy per microbatch in flight.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Correct the delay: look ahead along the motion

Nesterov evaluates the gradient at a look-ahead point. With a delay, the old look-ahead point lies roughly along the motion that happened during the delay.



Nesterov momentum

$$d_t = \gamma_t \left(w_t - w_{t-1}\right)$$

$$w_{t+1} = w_t + d_t - \eta \nabla f(w_t + d_t)$$

with a delay τ (stage-wise)

$$w_{t+1} = w_t + d_t - \eta \left(1 - \gamma_t\right) \nabla f(\bar{w}_t + \bar{d}_t)$$

$$\bar{w}_t = w_{t-\tau}, \qquad \bar{d}_t = d_{t-\tau}$$

Theorem 1 (convex, L-smooth, bounded gradients, fixed τ, η = 1/L, momentum (t−2)/t after a warm-up): converges in function value

Compute the delayed gradient at the old look-ahead point: momentum absorbs the delay.

Ajanthan et al., Nesterov Method for Asynchronous Pipeline Parallel Optimization, ICML 2025: Eq. 10, Thm 1. Transformers: NAdam.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Corrected asynchrony outperforms waiting, per iteration

Validation perplexity after 50k iterations; a ≈134M-parameter decoder split into 8 stages. Lower is better.

Method	Correction for delay	WikiText	BookCorpus	OpenWebText	Memory
GPipe (synchronous)	none needed: it waits	30.63	42.39	65.17	O(N)
PipeDream	weight stashing only	99.48	52.98	224.30	O(PN)
PipeMare	weight prediction, LR schedule	71.38	76.93	239.13	O(N)
Nesterov look-ahead	look-ahead, with stashing	27.72	39.85	62.86	O(PN)
Nesterov, no stashing	look-ahead + LR discounting	29.90	42.61	108.20	O(N)

Teal: lower perplexity than GPipe. Orange: more than 5% higher. Memory for weights: N parameters, P stages.

Another correction: basis rotation. Rotate the optimizer's coordinates toward the Hessian eigenbasis, so that delayed updates stay close to the ones they replace. Jung, Shin, Lee, ICML 2026: 81.7% fewer iterations to the same training loss, in a partial run of a 3B, 32-stage model.

Uncorrected staleness degrades perplexity; corrected, the asynchronous pipeline outperforms GPipe.

Ajanthan et al., ICML 2025, Table 1. Same data: async methods update every microbatch, GPipe every 4. Compared per iteration, not wall-clock.

PART II · OPEN COMPETITION

Protocol NanoGPT: an open speedrun over slow links



Task 8-stage, 203M-parameter GPT to 3.276 validation loss on FineWeb

Links one 8×H100 server; links simulated at 200 Mb/s and 50 ms

Score T = steps to the target × steady-state step time

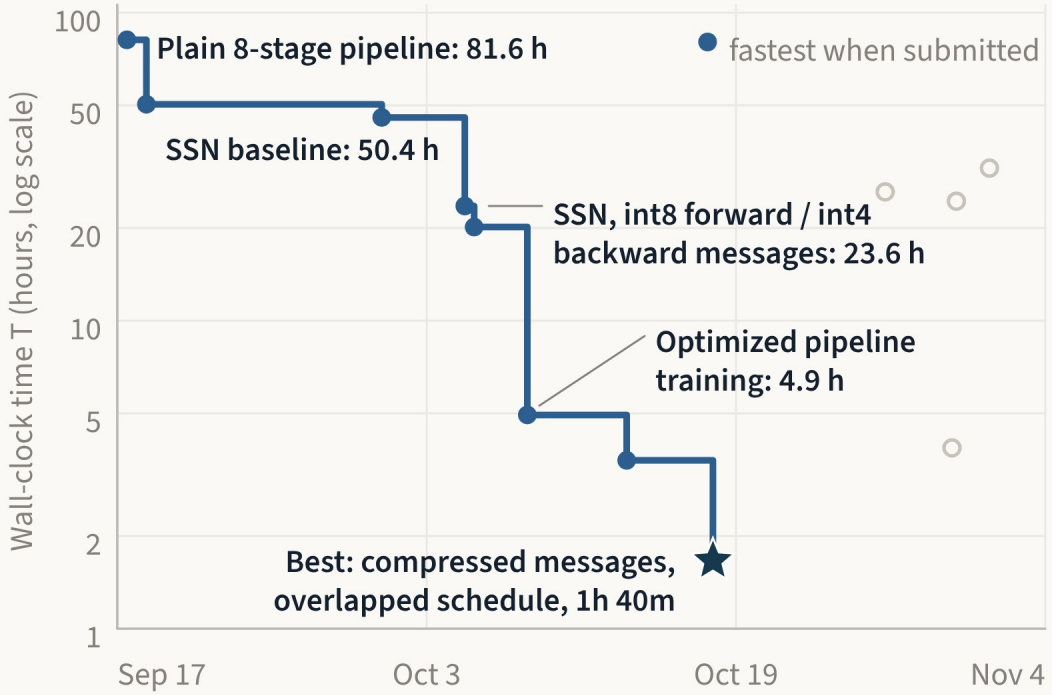
Open model, optimizer, schedule, compression

Take part: open until 1 December

pluralis.ai/protocol-nanogpt

$18k in prizes · winners at NeurIPS 2026 (CODEC-FM workshop)

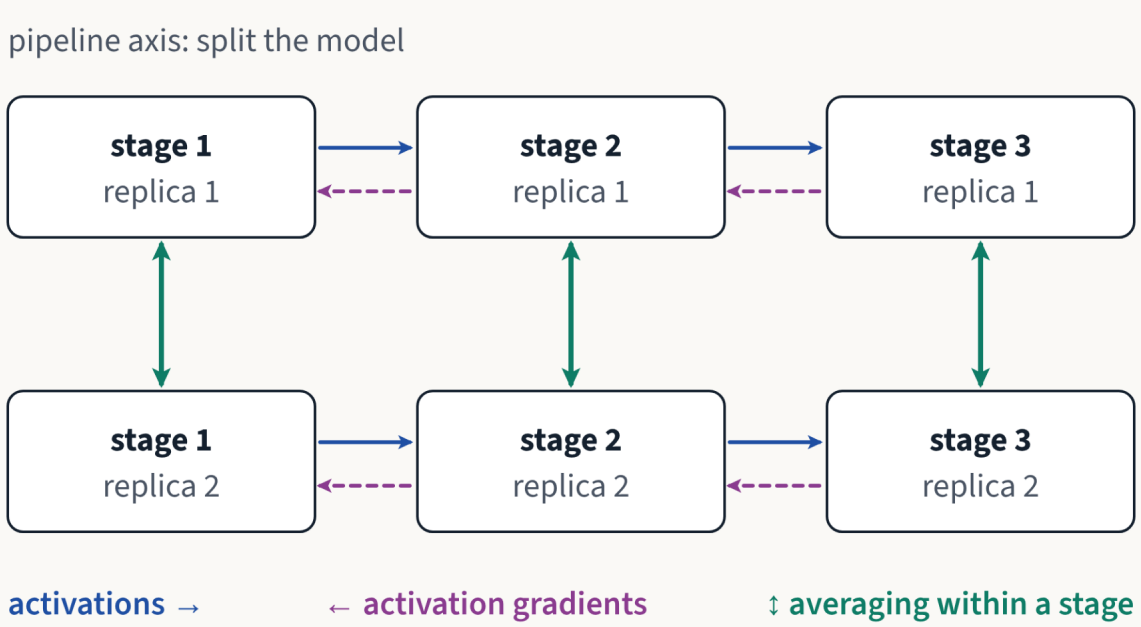
Leaderboard: the best entry is 49× faster than the plain pipeline

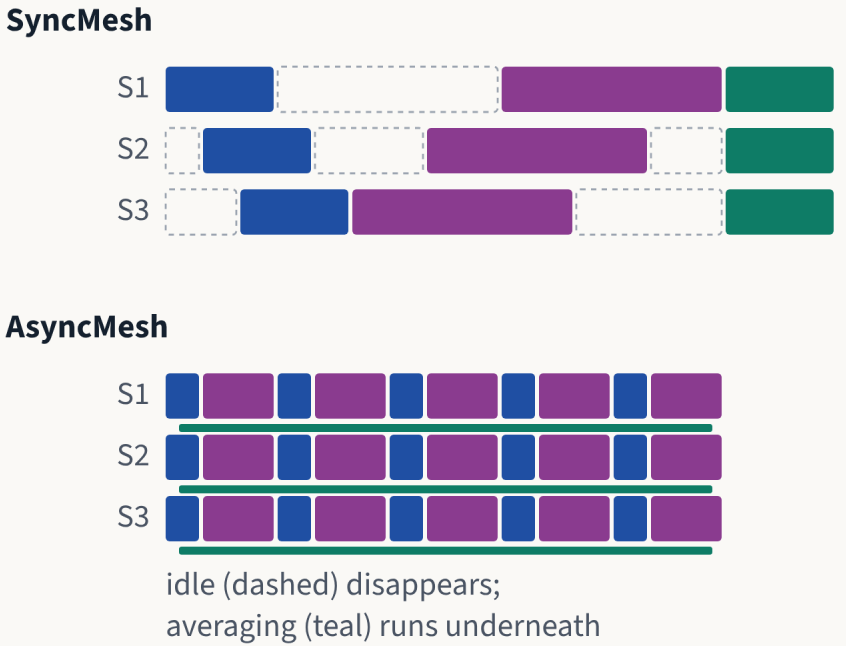


$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Both axes at once: the mesh

Real runs split the model into stages and replicate every stage. A synchronous mesh waits on both axes: pipeline flushes and averaging barriers.





Part II in one line: small exact messages across stages, and delay-corrected asynchrony on both axes.

Ajanthan et al., AsyncMesh, NeurIPS 2026: matches synchronous training up to 1B with far less communication.

PART III

The tools at work

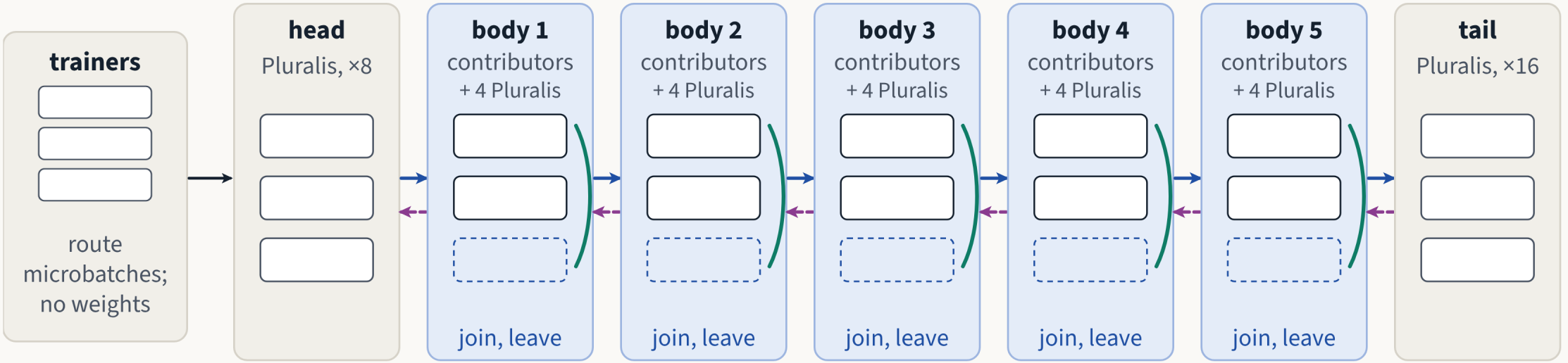
One public training run, a summary of the methods, and the questions that are still open.

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

$$T(\varepsilon) \approx K(\varepsilon)\, t_{\text{comp}} + R(\varepsilon)\, t_{\text{comm}} + t_{\text{idle}}$$

Agora: an 8B model trained across the internet

Seven pipeline stages. Contributors, admitted by a managed service, held replicas of the five body stages next to four Pluralis workers in each.



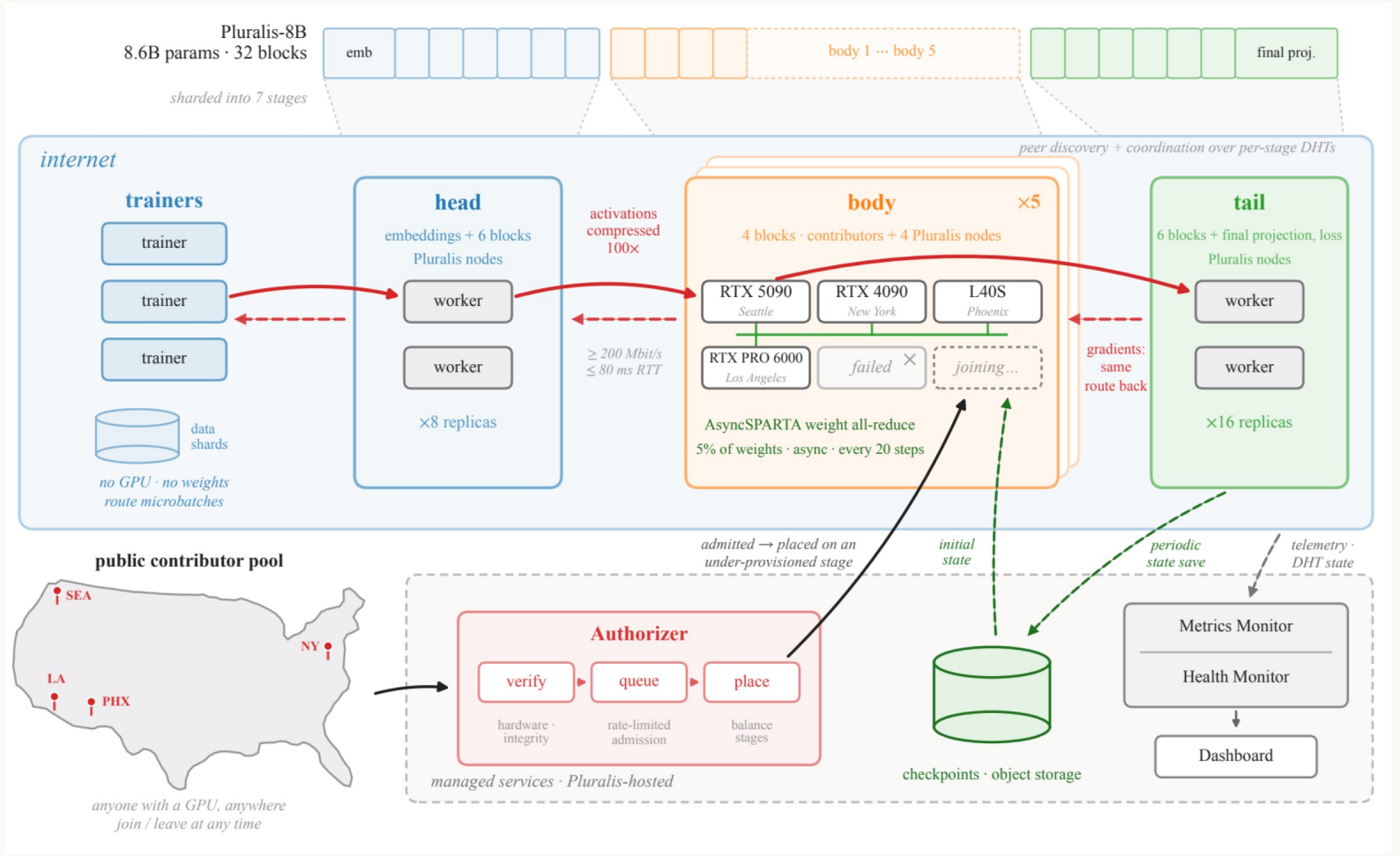
- **between stages: subspace-network activations, ≈100× smaller; admission: ≥ 200 Mb/s, RTT ≤ 80 ms**
- **within a stage: AsyncSPARTA averages 5% of the weights every 20 steps, asynchronously**

admission, checkpoints and monitoring run as managed services

≈100× smaller activations: one 4096-token sequence (d = 5120, fp32) is ≈7 Mb, not 671 Mb: ≈34 ms to serialize at 200 Mb/s, not 3.4 s.

Avraham et al., Agora, arXiv 2607.13332 (2026), Fig. 1, §3. Times: serialization only.

The Agora system

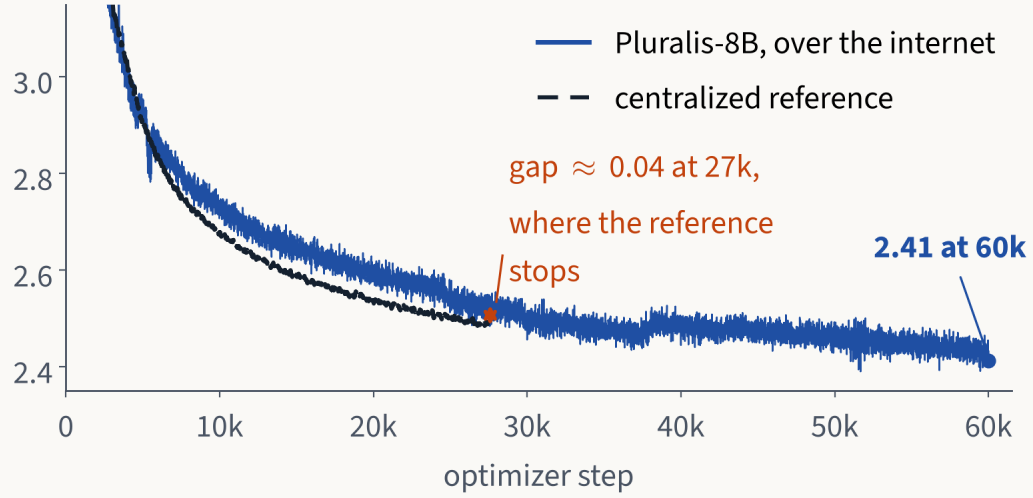


Avraham et al., Agora, arXiv 2607.13332 (2026), Figure 1.

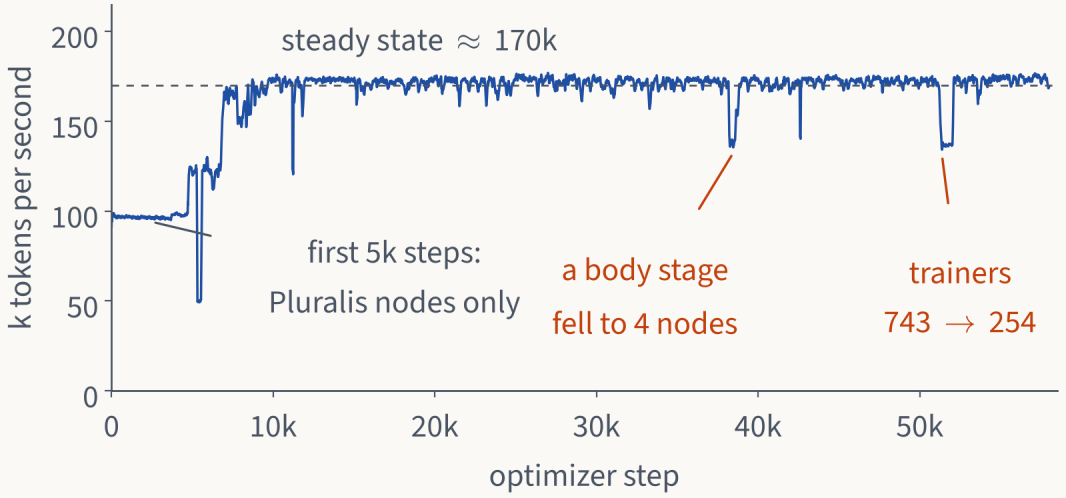
Pluralis-8B: the public run

8.6B	500B	40 days	330	669 / 607	≈170k
parameters, 7 stages	FineWeb-Edu tokens	mid-2026, in public	contributor nodes over the run	joins / departures	tokens per second at steady state

Training loss against a centralized reference



Throughput while nodes came and went

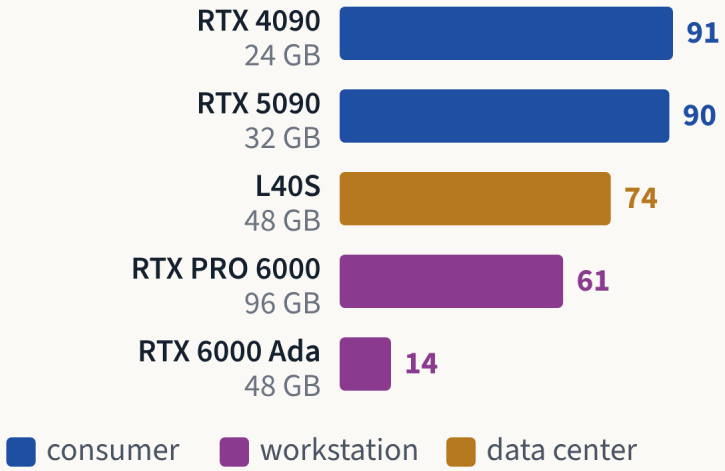


4.2 tokens per pooled spec-sheet TFLOP (dense BF16): 63% of a centralized Llama 3.1 8B H100 baseline.

Avraham et al., Agora, arXiv 2607.13332 (2026), Figs. 17a, 19b. Reference: same model and averaging on a homogeneous cluster, to step 27.5k.

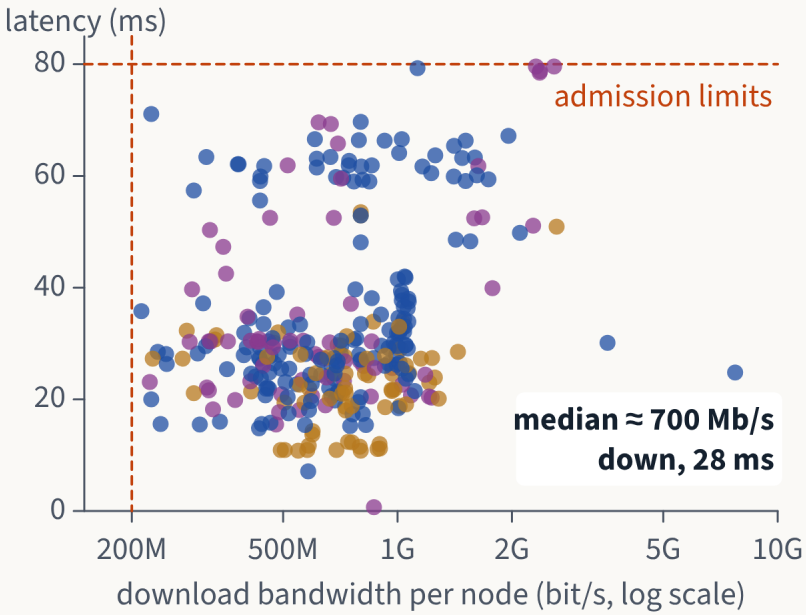
What Pluralis-8B ran on: unequal GPUs, internet links

330 contributor GPUs, five models

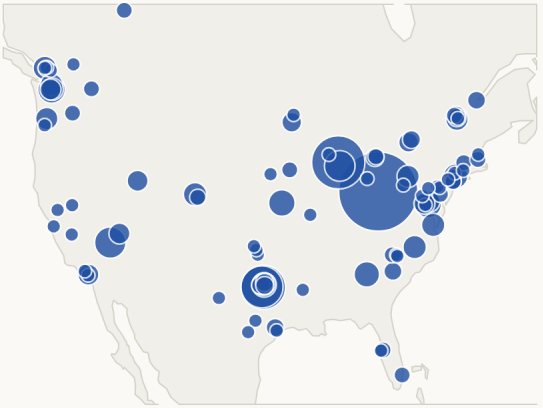


Admission: ≥ 24 GB VRAM, ≥ 200 Mb/s, ≤ 80 ms latency.

Links: internet grade, and uneven



All nodes: US and Canada



96 cities; circle area ∝ nodes.
Most contributors rented GPUs on marketplaces (Vast.ai, RunPod).

RAM 80 GB–2 TB (median 504 GB) · 60 contributor slots, ≈61 nodes online at a time · 3.3M join requests from 612 clients; 208 admitted, after a median wait of 5.8 h.

Agora report §5.3 (GPU mix, RAM, links, admission) · public dashboard, contributors who joined by 23 June 2026.

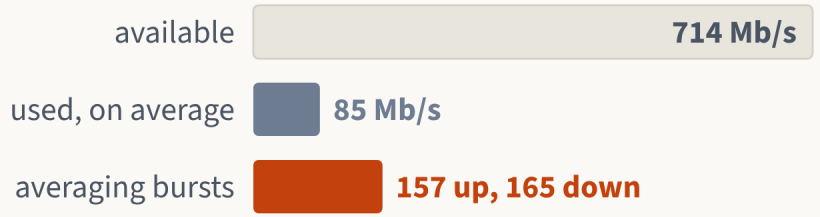
Where the time went: compute and latency, not bandwidth

With activations 100× smaller and averaging in the background, a microbatch was mostly compute. What separated fast nodes from slow ones was latency.

One microbatch, body stages (median)

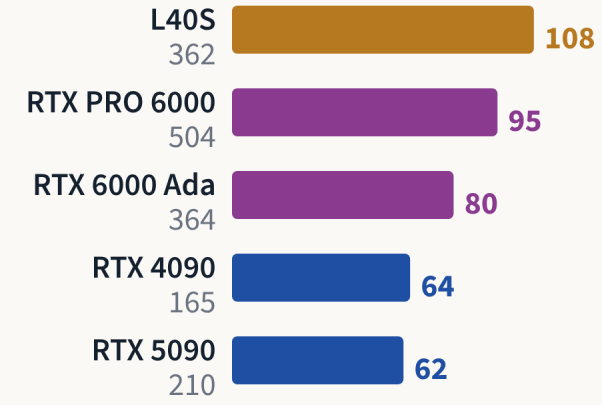


Link use, body nodes (mean)

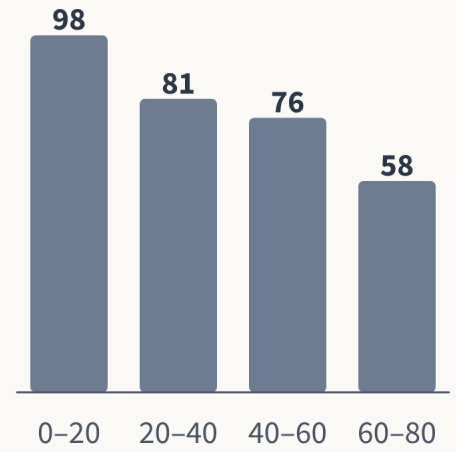


Work per node, batches per minute

by GPU (peak TFLOP/s)



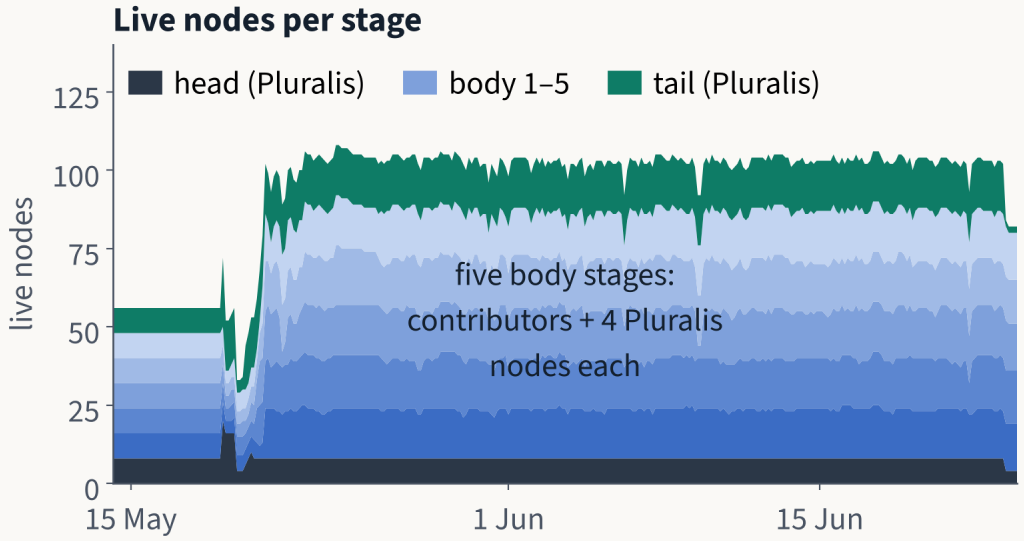
by latency (ms)

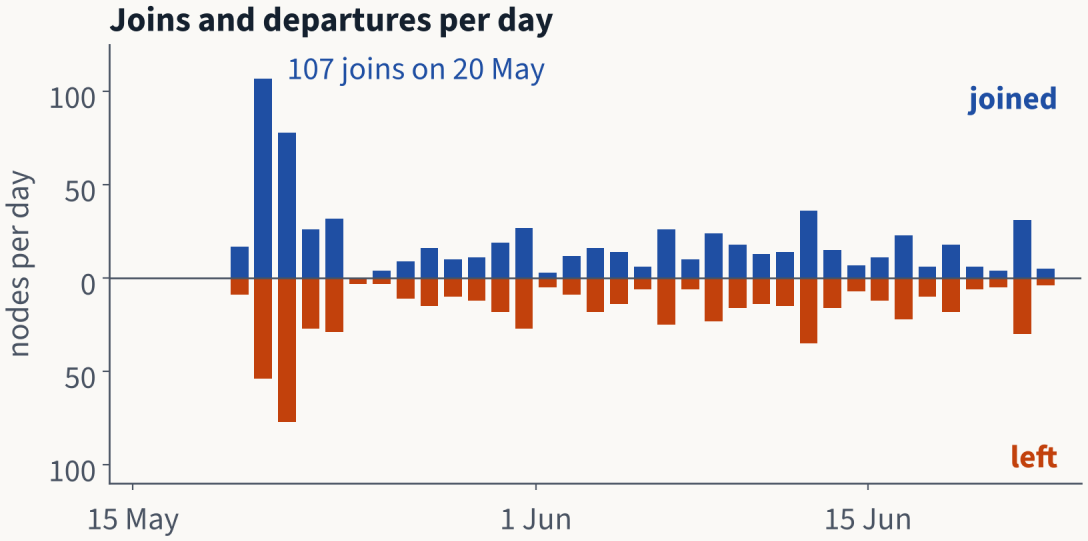


Bandwidth above the 200 Mb/s floor made no difference: about 80 microbatches per minute in every band up to 1 Gb/s.

Agora report: Table 4 and Fig. 18 (steps 20k–30k), Fig. 30 (body nodes), Fig. 36. Latency and GPU type are confounded (Fig. 37).

Churn, and how the run absorbed it





Joining

a newcomer averages with weight 0 for 400 steps, then warms its optimizer for 100 more; only then do its samples count

Leaving

nothing to hand over: other replicas hold the stage; a node that fails 2 of its last 3 averaging rounds removes itself

Recovering

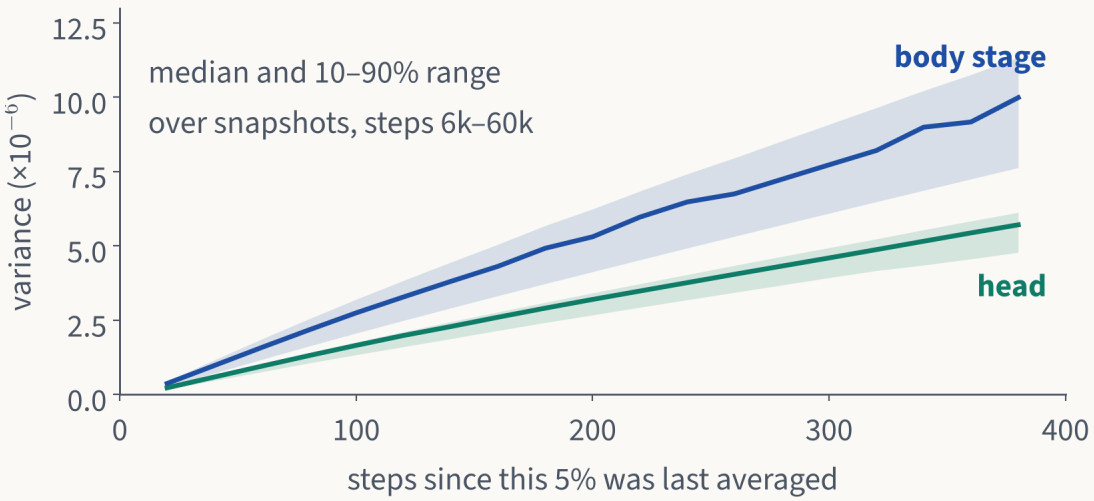
after a mass exit near step 40k, the loss recovered within a few thousand steps, without intervention

Agora report: Fig. 15a, §3.3.1 and Algorithm 1, Fig. 22, §5.5 · public dashboard, to 24 June 2026.

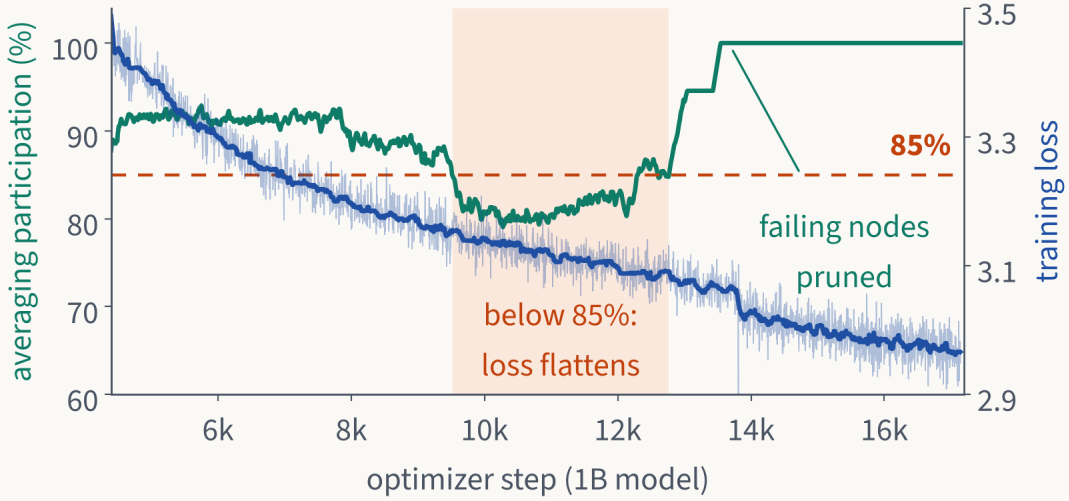
Averaging under churn: drift, and failed rounds

Each 5% slice of a stage is averaged once every 400 steps. In between, replicas drift apart; when rounds fail, they drift further.

Replicas drift apart between averages



When averaging rounds fail



Only Pluralis nodes reduce; contributors only send. A slice that does not come back keeps its local values, so every round finishes.

Agora report: §5.7, Fig. 25 (snapshots of Pluralis replicas: body 3, head); §6.4, Fig. 45 (1B, equal speeds, injected failures).

Summary of the methods

Method	Reduces	Introduces	Used in Agora?
Local steps (Local SGD, DiLoCo)	rounds R	drift between replicas	✓ 5% every 20 steps
Low-rank compression (PowerSGD)	bytes per round	compression error	✗
Gossip (D-PSGD)	global all-reduce	slower agreement, 1/ρ	✗
Sparse, overlapped averaging	bytes and idle time	stale averages	✓ AsyncSPARTA
Pipeline stages	memory per GPU	boundary traffic, bubbles	✓ 7 stages
Subspace networks	boundary bytes, ≈100×	a constraint on the architecture	✓
Asynchronous pipelines (Nesterov)	bubbles	stale gradients	✗

Each method reduces one term of T(ε) and often increases K(ε). Good methods keep that increase small.

White rows: every worker holds the whole model (Part I). Tinted rows: the model is split (Part II). ρ: consensus rate of the gossip matrix.

Open problems, as research questions

1 · Placing the model

Devices and links differ by orders of magnitude. How should a model be split into stages and replicated across them? Today: heuristics.

2 · Wall-clock theory for pipelines

Time-optimal rates, as for asynchronous SGD, when the model is split and every stage has its own delay.

3 · Compression with guarantees

Lossy activations across stages; compressed updates or local steps across replicas. Which schemes provably converge, and why does DiLoCo work with H = 500?

4 · Scaling laws

How does quality scale with model size, data and bandwidth when communication is constrained, as with subspace networks, compression or local steps?

5 · Churn

Optimization that stays stable as workers join, leave and fail mid-step — with provable recovery.

6 · Trust

Checking that a contributor did the work, at lower cost than recomputing it, and staying robust when one did not.

Measure wall-clock time and total resources to the same model quality — not steps alone.

THANK YOU

Questions

One cost model

Time = steps × compute + rounds × communication + idle. Each method reduces one term and usually needs more steps.

Whole model per worker

Fewer rounds, smaller messages, fewer peers, no waiting.

Split model

Make boundary messages small and exact; correct stale gradients instead of waiting.

Thalaiyasingam Ajanthan

Pluralis Research · pluralis.ai

PR

Ring all-reduce, counted

$$\underbrace{(n-1)\frac{S}{n}}_{\text{reduce-scatter}} + \underbrace{(n-1)\frac{S}{n}}_{\text{all-gather}} = \frac{2(n-1)}{n}S \ \text{ bytes sent per worker}$$

Chunks. Each worker splits its S-byte vector into n chunks of S/n bytes.

Reduce-scatter. n − 1 steps: pass one chunk to the right and add. Each worker then owns one fully summed chunk.

All-gather. n − 1 more steps: pass the summed chunks around until everyone has all of them.

Time. 2(n − 1)(α + S/(nB)): the bandwidth term barely depends on n; the latency term grows with n.

Running example: n = 8, S = 2 GB ⟶ 2 · 7/8 · 2 GB = 3.5 GB sent per worker, per step.

Bandwidth-optimal: in any all-reduce, some worker sends ≥ 2(n − 1)S/n.

Patarasuk & Yuan, J. Parallel Distrib. Comput. 2009. α: per-message latency; B: link bandwidth; full-duplex ring without contention.

When does a delay destabilize gradient descent?

$$w_{t+1} = w_t - \eta \nabla f(w_{t-\tau}), \quad f \text{ quadratic, curvature } L: \quad \text{stable} \iff \eta L < 2 \sin\left(\frac{\pi}{2(2\tau+1)}\right)$$

Reduce to a scalar recursion. Along an eigenvector with curvature L, the error obeys a delayed linear recursion with gain a = ηL (below).

The bound shrinks with τ. τ = 0: a < 2; τ = 1: a < 1; large τ: a ≲ π/(2τ + 1), so η must scale like 1/τ.

Characteristic roots. Stability means every root of the characteristic polynomial (below) lies inside the unit disc.

The toy. a = ηL = 3/8. τ = 3: bound 0.445, stable. τ = 4: bound 0.347, unstable.

$$x_{t+1} = x_t - a\, x_{t-\tau}, \quad a = \eta L \qquad \Longrightarrow \qquad z^{\tau+1} - z^{\tau} + a = 0$$

With delay τ, stability forces ηL < 2 sin(π/(2(2τ + 1))): η must shrink like 1/τ.

Levin & May, Theor. Popul. Biol. 1976. Delay-adaptive steps for asynchronous SGD: Koloskova, Stich, Jaggi, NeurIPS 2022.

Part I: local steps and compression

McMahan et al. *Communication-efficient learning of deep networks from decentralized data.* AISTATS 2017.

Reddi et al. *Adaptive federated optimization.* ICLR 2021.

Stich. *Local SGD converges fast and communicates little.* ICLR 2019.

Woodworth et al. *Is local SGD better than minibatch SGD?* ICML 2020.

Woodworth, Patel, Srebro. *Minibatch vs local SGD for heterogeneous distributed learning.* NeurIPS 2020.

Koloskova et al. *A unified theory of decentralized SGD with changing topology and local updates.* ICML 2020.

Karimireddy et al. *SCAFFOLD: stochastic controlled averaging for federated learning.* ICML 2020.

Mishchenko et al. *ProxSkip: yes! Local gradient steps provably lead to communication acceleration! Finally!* ICML 2022.

Douillard et al. *DiLoCo: distributed low-communication training of language models.* ICML 2024 workshop (WANT).

Alistarh et al. *QSGD: communication-efficient SGD via gradient quantization and encoding.* NeurIPS 2017.

Bernstein et al. *signSGD: compressed optimisation for non-convex problems.* ICML 2018.

Stich, Cordonnier, Jaggi. *Sparsified SGD with memory.* NeurIPS 2018.

Vogels, Karimireddy, Jaggi. *PowerSGD: practical low-rank gradient compression for distributed optimization.* NeurIPS 2019.

Karimireddy et al. *Error feedback fixes SignSGD and other gradient compression schemes.* ICML 2019.

Richtárik, Sokolov, Fatkhullin. *EF21: a new, simpler, theoretically better, and practically faster error feedback.* NeurIPS 2021.

Beton et al. *Improving the efficiency of distributed training using sparse parameter averaging.* ICLR 2025 workshop.

Part I: gossip and asynchrony

Boyd, Ghosh, Prabhakar, Shah. *Randomized gossip algorithms.* IEEE Trans. Inf. Theory 2006.

Lian et al. *Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent.* NeurIPS 2017.

Xiao, Boyd. *Fast linear iterations for distributed averaging.* Systems & Control Letters 2004.

Koloskova, Stich, Jaggi. *Decentralized stochastic optimization and gossip algorithms with compressed communication.* ICML 2019.

Mania et al. *Perturbed iterate analysis for asynchronous stochastic optimization.* SIAM J. Optim. 2017.

Stich, Karimireddy. *The error-feedback framework: SGD with delayed gradients.* JMLR 2020.

Agarwal, Duchi. *Distributed delayed stochastic optimization.* NeurIPS 2011.

Lian, Huang, Li, Liu. *Asynchronous parallel stochastic gradient for nonconvex optimization.* NeurIPS 2015.

Koloskova, Stich, Jaggi. *Sharper convergence guarantees for asynchronous SGD for distributed and federated learning.* NeurIPS 2022.

Mishchenko, Bach, Even, Woodworth. *Asynchronous SGD beats minibatch SGD under arbitrary delays.* NeurIPS 2022.

Tyurin, Richtárik. *Optimal time complexities of parallel stochastic optimization methods under a fixed computation model.* NeurIPS 2023.

Tyurin, Richtárik. *On the optimal time complexities in decentralized stochastic asynchronous optimization.* NeurIPS 2024.

Maranjyan, Tyurin, Richtárik. *Ringmaster ASGD: the first asynchronous SGD with optimal time complexity.* ICML 2025.

Maranjyan, Richtárik. *Ringleader ASGD: the first asynchronous SGD with optimal time complexity under data heterogeneity.* ICLR 2026.

Douillard et al. *Streaming DiLoCo with overlapping communication: towards a distributed free lunch.* arXiv 2501.18512, 2025.

Maziane et al. *LOSCAR-SGD: local SGD with communication-computation overlap and delay-corrected sparse model averaging.* NeurIPS 2026.

Ajanthan et al. *AsyncMesh: fully asynchronous optimization for data and pipeline parallelism.* NeurIPS 2026.

Parts II and III: split models and the Agora run

Rajbhandari et al. *ZeRO: memory optimizations toward training trillion parameter models.* SC 2020.

Huang et al. *GPipe: efficient training of giant neural networks using pipeline parallelism.* NeurIPS 2019.

Narayanan et al. *PipeDream: generalized pipeline parallelism for DNN training.* SOSP 2019.

Yang et al. *PipeMare: asynchronous pipeline parallel DNN training.* MLSys 2021.

Ryabinin et al. *SWARM parallelism: training large models can be surprisingly communication-efficient.* ICML 2023.

Yuan et al. *Decentralized training of foundation models in heterogeneous environments.* NeurIPS 2022.

Bian et al. *Does compressing activations help model parallel training?* MLSys 2024.

Wang et al. *Fine-tuning language models over slow networks using activation quantization with guarantees.* NeurIPS 2022.

Ramasinghe et al. *Subspace networks: scaling decentralized training with communication-efficient model parallelism.* NeurIPS 2025.

Markou, Ajanthan, Gould. *Sharper convergence rates for nonconvex optimisation via reduction mappings.* NeurIPS 2025.

Ajanthan et al. *Nesterov method for asynchronous pipeline parallel optimization.* ICML 2025.

Jung, Shin, Lee. *Mitigating staleness in asynchronous pipeline parallelism via basis rotation.* ICML 2026.

Avraham et al. *Agora: collective and permissionless internet-scale pretraining of large language models.* arXiv 2607.13332, 2026.

Pluralis Research. *Protocol NanoGPT: speedrunning pipeline parallel training over low-bandwidth links.* github.com/PluralisResearch/ProtocolNanoGPT, 2026.

Patarasuk, Yuan. *Bandwidth optimal all-reduce algorithms for clusters of workstations.* J. Parallel Distrib. Comput. 2009.

Levin, May. *A note on difference-delay equations.* Theor. Popul. Biol. 1976.

Opening: N. Lane, talk at the Protocol Learning workshop, 26 April 2026; Microsoft quarterly reports, Meta and Alphabet 2025 annual reports; NVIDIA press release, 8 Jan 2024; Jon Peddie Research; maps from Natural Earth.

Part I in practice: DiLoCo variants and large runs

Liu et al. *Asynchronous local-SGD training for language modeling.* ICML 2024 workshop.

Decoupled DiLoCo team (Douillard et al.). *Decoupled DiLoCo for resilient distributed pre-training.* arXiv 2604.21428, 2026.

Kim et al. *HALoS: hierarchical asynchronous local SGD over slow networks for geo-distributed LLM training.* ICML 2025.

Jaghouar et al. *INTELLECT-1 technical report.* arXiv 2412.01152, 2024.

Sarfi, Thérien, Lidin, Belilovsky. *Overcoming the communication-performance tradeoff in LLM pretraining (SparseLoCo).* NeurIPS 2025 workshop (OPT).

Khaled, Kale, Douillard, et al. *Understanding outer optimizers in local SGD: learning rates, momentum, and acceleration.* NeurIPS 2025.

Charles et al. *Communication-efficient language model training scales reliably and robustly: scaling laws for DiLoCo.* NeurIPS 2025.

Kallusky et al. *SNOO: step-K Nesterov outer optimizer.* arXiv 2510.15830, 2025.

Defazio, Mishchenko, Raman, Shi, Xiao. *Smoothing DiLoCo with primal averaging for faster training of LLMs.* arXiv 2512.17131, 2025.

Lidin et al. *Covenant-72B: pre-training a 72B LLM with trustless peers over-the-internet.* arXiv 2603.08163, 2026.

Agarwal et al. *On the utility of gradient compression in distributed training systems.* MLSys 2022.

Assran et al. *Stochastic gradient push for distributed deep learning.* ICML 2019.