ICOMP 2026

Protocol Learning

Collective Training of Foundation Models over the Internet

Thalaiyasingam Ajanthan

Pluralis Research

Casablanca · Friday 6 November 2026



PR

JOINT WORK

Technical Team



Gil Avraham



Violetta Shevchenko



Hadi Mohaghegh Dolatabadi



Karol Pajak



James Snewin



Harry Xi



Rodney O'Donnell



Sameera Ramasinghe



Chamin Hewa Koneputugodage



Shamane Siriwardhana



Evan Markou



Thalaiyasingam Ajanthan



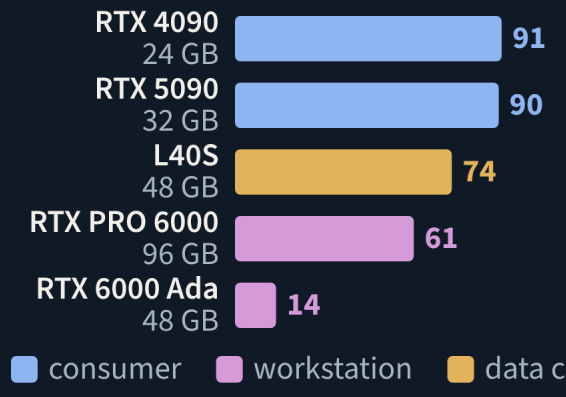
Alexander Long

OUR AIM

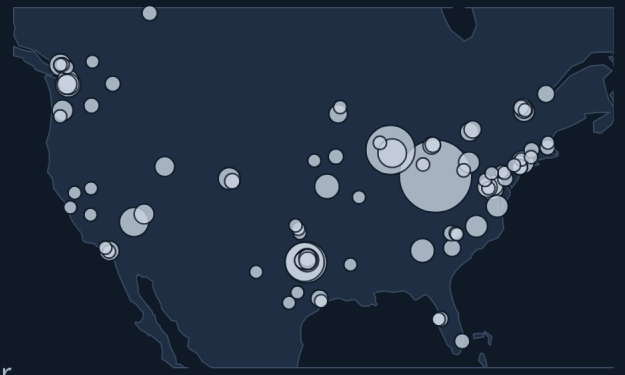
Train foundation models collectively, over the internet

Many participants, each with a GPU and an internet connection. None needs the whole model, or a datacenter.

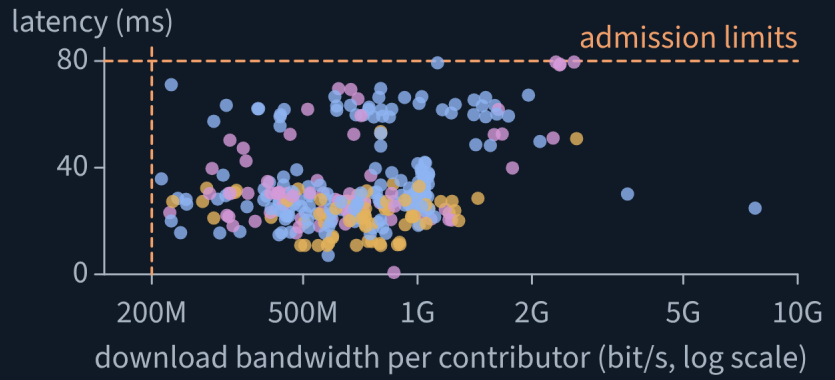
330 contributor GPUs, five models



Where: 96 cities, US and Canada



Their links: internet, not NVLink



median contributor
0.7 Gb/s down, 28 ms

NVLink in an H100 server
3,600 Gb/s, ≈5,000× more

Pluralis-8B (2026): 40 days, 500B tokens, 669 joins and 607 departures. GPU mix: Agora report; links and locations: public dashboard.

MOTIVATION

Open models are close to the frontier, and widely used

LAG BEHIND THE CLOSED FRONTIER

≈3–5 months

Open-weight models trail the best closed ones by ≈3 months on the Artificial Analysis index and ≈5 on the software-engineering subset of Epoch's Capabilities Index.

NUMBER OF ML PAPERS MENTIONING

Qwen ≈ GPT

Qwen, an open model family, is now mentioned in almost as many arXiv ML papers as OpenAI's GPT models.

LABS WITHIN A YEAR OF THE FRONTIER

10+ labs

governments and large companies; such a model costs hundreds of millions of dollars, approaching billions.

Why it matters

Science
models anyone can inspect, experiment with and teach with

Ownership
adapt a model you control, on your own data

Less concentration
no single group decides how the technology is used

Safety research
study risks on models the community can access

Open models are a large part of what the research community studies and builds on — but who can afford to train them?

N. Lambert, Open Language Models, Harvard, 22 Oct 2026 (Artificial Analysis; Epoch AI; scan of ≈370k arXiv ML papers).

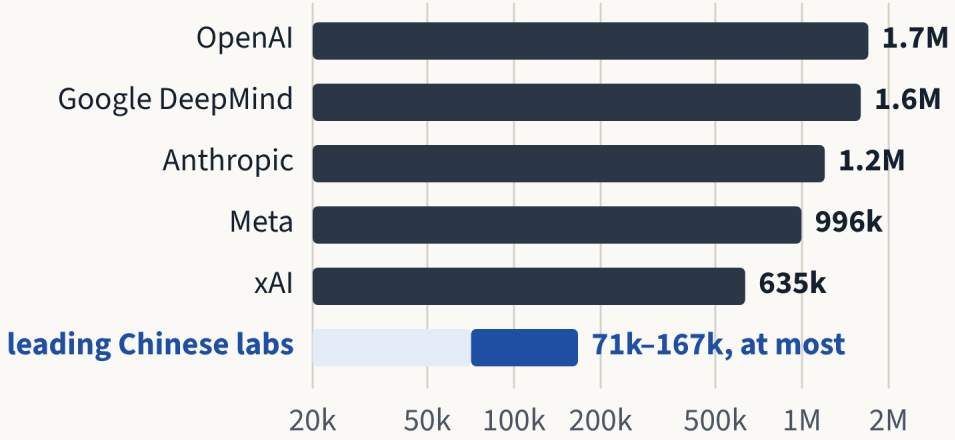
PR

MOTIVATION

But few can afford to build them

Frontier compute sits with a few labs

AI compute per lab, H100-equivalents (log scale)



blue: the labs releasing the strongest open-weight models

"And I think there won't be 50 of those. There may be like a handful."

Jeff Dean, on frontier models, Sequoia AI Ascent, 2 May 2025

Open weights share a model, not the means to build the next one

Releasing weights is a gift from whoever paid for training. As costs grow, fewer can afford to keep giving.

"...it may be that the capital costs, which are so immense, fundamentally changes how software is built."

Eric Schmidt, Stanford ECON 295/CS 323, April 2024

"AI infrastructure today is thinking too small."

Nic Lane, Protocol Learning workshop, 26 April 2026

Open model development needs a way to pool resources — and to reward those who pool them.

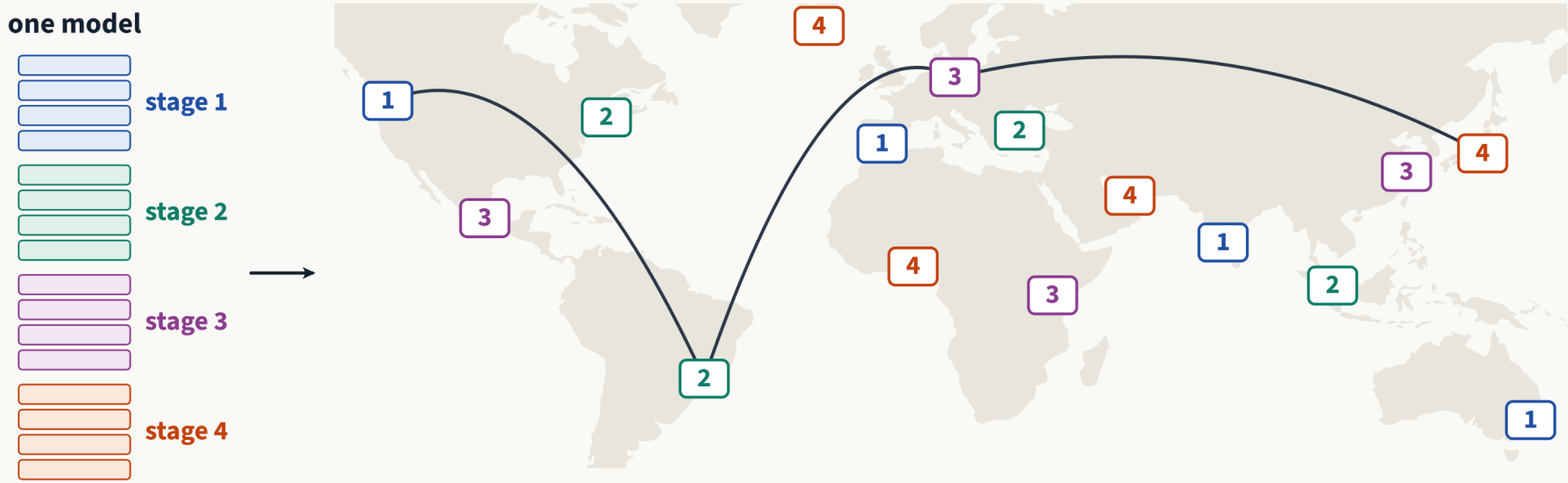
Compute: Epoch AI median estimates for US labs, end-2025, incl. inference; Chinese labs: power-based estimate (D. Patel, Aug 2026).

PR

MOTIVATION

Protocol Learning

Split one model into stages and spread them across contributors on the internet. Each contributor holds and trains one stage.



each contributor holds one stage; each microbatch visits one holder of every stage

No co-location
model-parallel training over low-bandwidth links

No one holds the whole model
participants hold a part — a possible basis for sharing value

Open participation
anyone with a suitable GPU can join; code and protocol open

Protecting a part from extraction is a research question (Long et al., NeurIPS 2025).

PR

MOTIVATION

Research challenges

1 **Communication-efficient model-parallel training**
SSN (NeurIPS 2025) · SSN++ (under review)
today · A

2 **Heterogeneous, unreliable devices**
workers at different speeds, joining and leaving · AsyncMesh (NeurIPS 2026) · PACE (under review)
today · B

3 **Fault tolerance at scale**
Node0 · SWARM · TorchFT · Agora

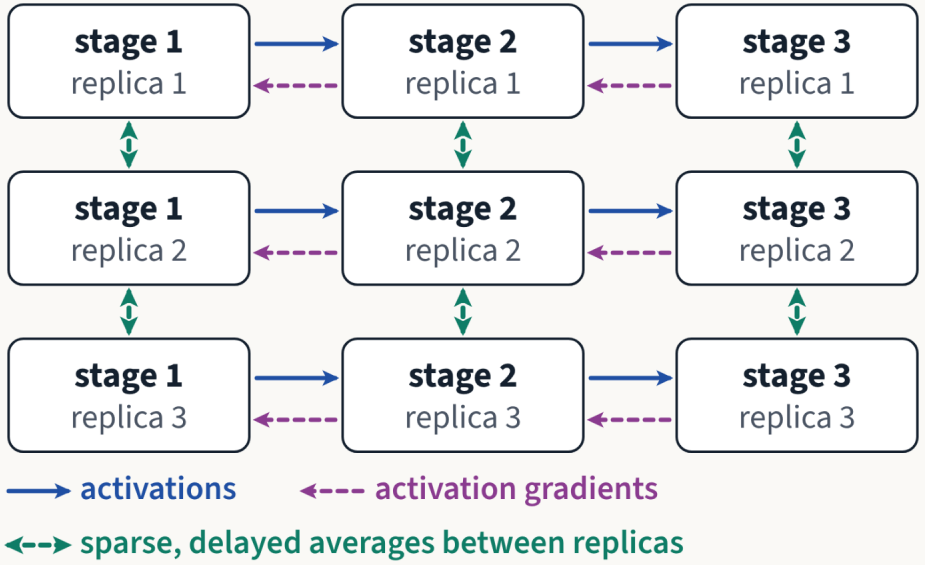
4 **Security and robustness to malicious participants**
verifying pipeline work (SENTINEL, Dolatabadi et al., 2026) · Byzantine robustness

5 **Protection against model extraction**
Unextractable Protocol Models (Long et al., NeurIPS 2025)

THE SETTING

The optimization problem: one model, two couplings

$$\min_{\{w_{s,r}\}} \mathbb{E}_{x,\pi}\, \ell\Big(f_S\big(\cdots f_1(x;\, w_{1,\pi_1})\cdots;\, w_{S,\pi_S}\big)\Big) \qquad \text{s.t.} \quad w_{s,r} = w_{s,r'} \ \text{ for all } s, r, r'$$



Q1 · Across stages

Messages sit inside the composition. Can a narrow message carry enough for the model to learn well?

Q2 · Within a stage

Replicas are coupled by consensus. How should they use averages that arrive late, from workers at different speeds?

$$T(\varepsilon) \;\approx\; \underbrace{K(\varepsilon)\, t_{\text{comp}}}_{\text{steps} \times \text{compute per step}} + \underbrace{R(\varepsilon)\, t_{\text{comm}}}_{\text{rounds} \times \text{time per round}} + \underbrace{t_{\text{idle}}}_{\text{waiting for other workers}}$$

Part A: narrow messages across stages — **Part B: no waiting within a stage**

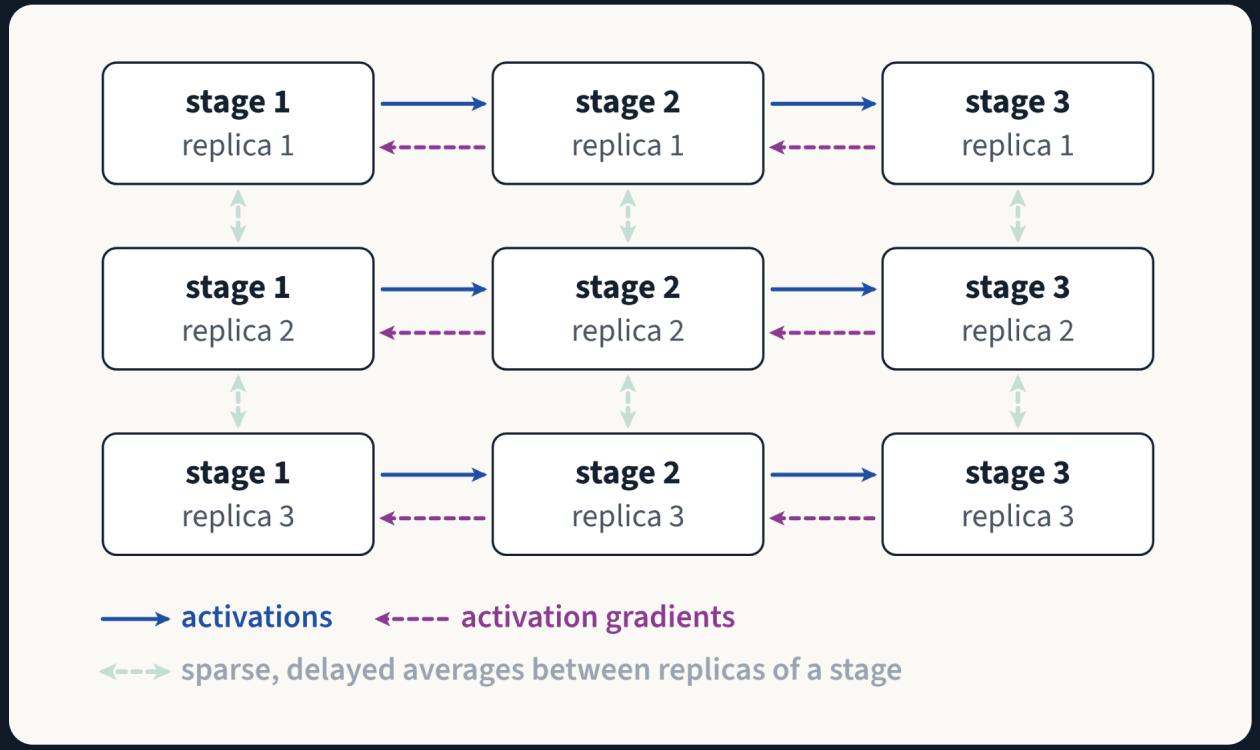
π: the route of a microbatch, one replica of each stage (routes may cross rows). The constraint is enforced only through sparse, delayed averages.

PR

PART A

Narrow messages, rich computation

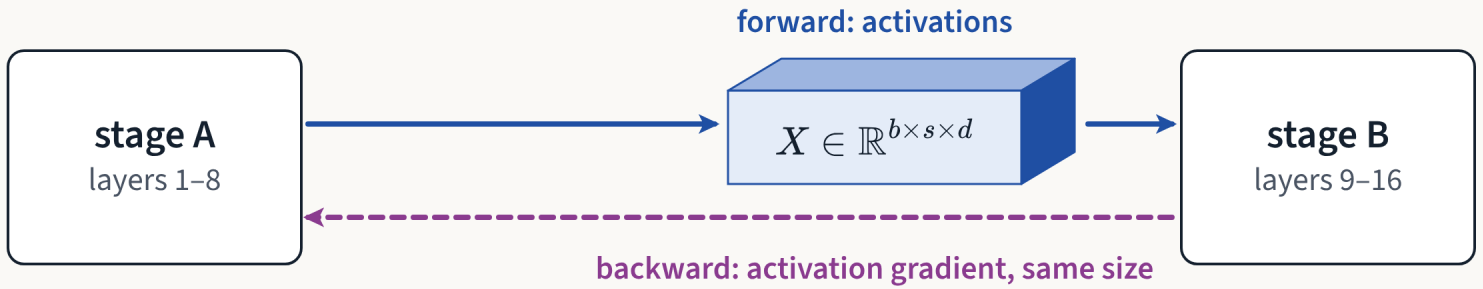
Q1 · Across stages: can a narrow message carry enough for the model to learn well?



A · NARROW MESSAGES

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

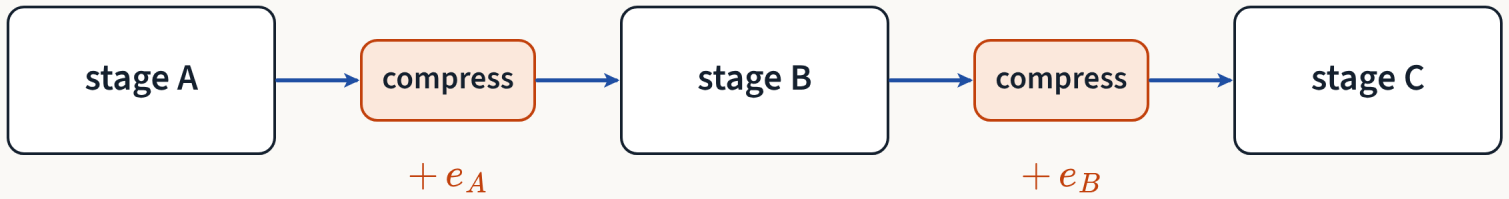
Per token and boundary, forward + backward in bf16: 8,192 B at d = 2048, 16,384 B at d = 4096. Line rate; latency ignored.

PR

A · NARROW MESSAGES

Why not compress activations like gradients?

Lossy compression adds an error at every boundary, 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

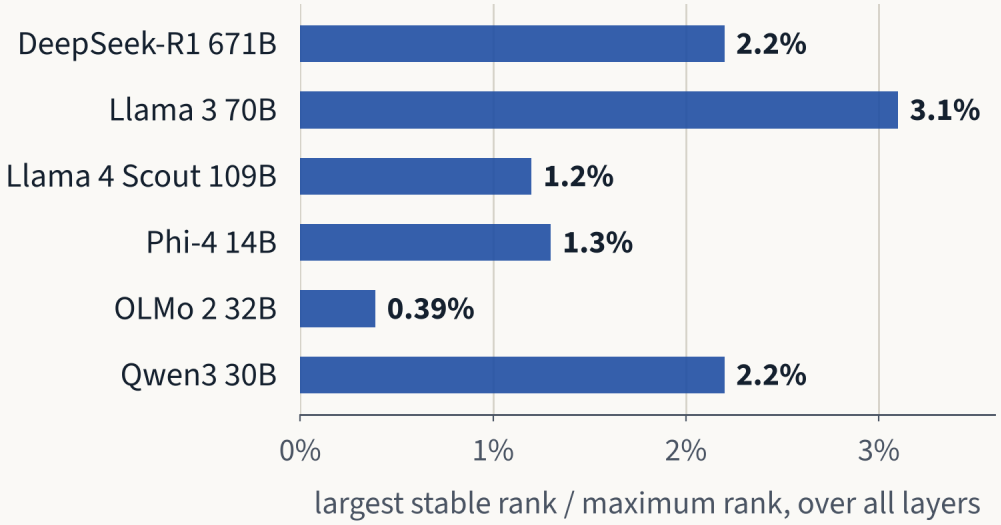
Compressing an activation
perturbs the function itself, at every stage, every step

Goal: about 50–100× fewer bytes per token, in a form the model is trained with, not an error added afterwards.

Error propagation: Ramasinghe et al., NeurIPS 2025, App. B. Activation compression: Bian et al., MLSys 2024.

PR

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

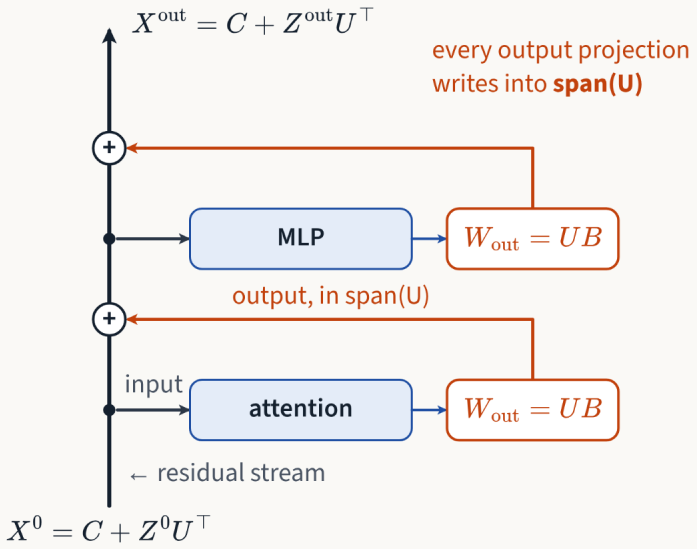
low stable rank
every term leaves through an output projection

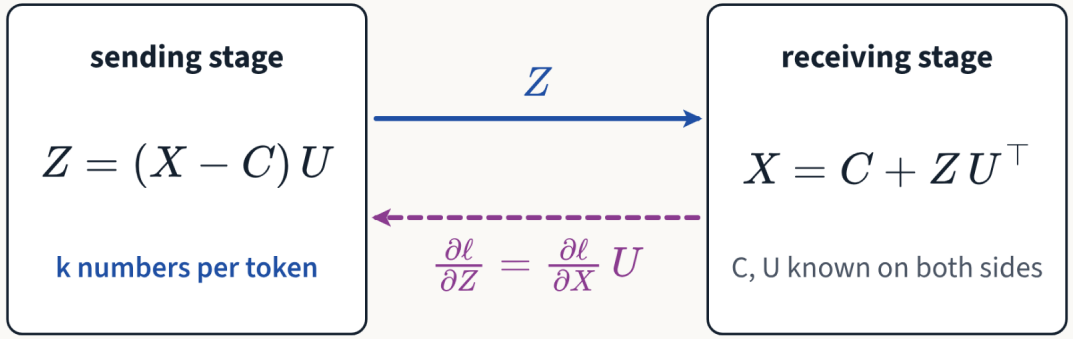
Trained weights seem to converge to a low-rank solution. Subspace networks use it to compress activations.

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.

Subspace networks: k numbers per token, exactly

Output projections are UB with one shared U (d × k), so the stream stays X = C + ZU$^{\top}$: only Z crosses.





Exact for the constrained model: d/k ≈ 100× fewer values in both directions at d = 4096, k = 40, and 51× at d = 2048.

The constraint: everything the layers add to the stream lies in the same k directions.

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

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

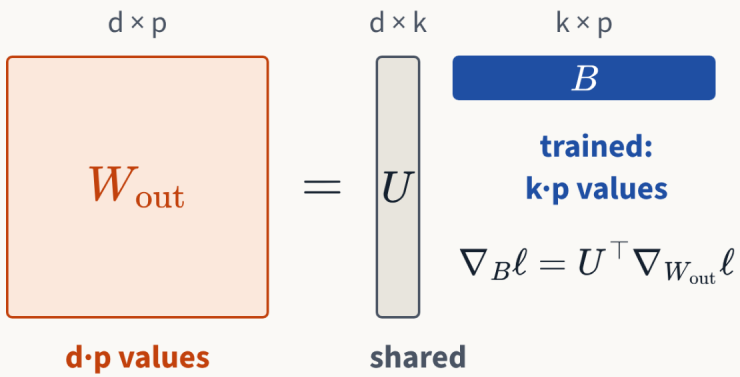
trainable, low rank: inside the shared subspace
under SSN, exactly k numbers per token cross the link

Ramasinghe et al., NeurIPS 2025, §4.3.1 (d = 4096, k = 40). C: fixed token table and position embedding, known to every stage.

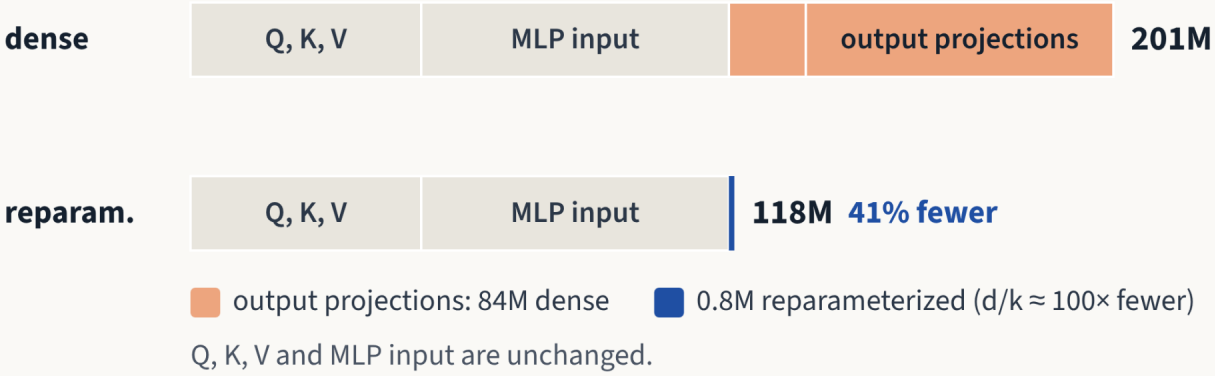
A · NARROW MESSAGES

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

Train B directly in W = UB: the constraint holds by construction, with no projection and plain AdamW.



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



How can we make the model more expressive while keeping the rank constraint between stages?

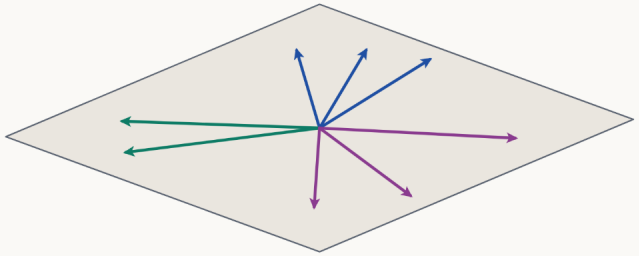
Avraham et al., Agora, arXiv 2607.13332 (2026), §4.1, Eq. 3; SSN++ uses this form. Counts: one block with a 4d MLP; p: input width.

PR

One shared subspace limits the model

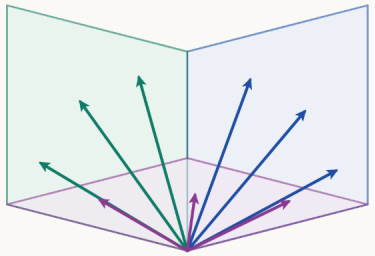
But restrictive: when every layer boundary is compressed, every layer must write into the same k directions.

SSN: one shared subspace



every layer writes into the same k directions

SSN++: a subspace per group



each group is low-rank; together they span more inside the stage, while the link still carries k numbers

best error: $\mathrm{tail}_k\big(\sum_j A_j\big)$

best error: $\sum_j \mathrm{tail}_k(A_j)$

$$\Gamma_k \;=\; \mathrm{tail}_k\big(\textstyle\sum_j A_j\big) \;-\; \textstyle\sum_j \mathrm{tail}_k(A_j) \;\geq\; 0$$ **the error that per-group subspaces remove**

A_j: second moment of group j's target updates · $\mathrm{tail}_k(A)$: the energy outside A's best k directions

Γ = 0 only if all groups prefer the same k directions; the more their directions differ, the more SSN++ can gain.

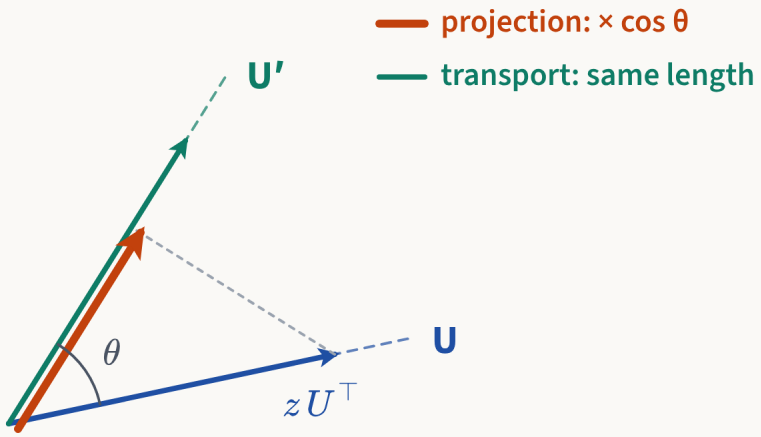
SSN++ (under review), §3.1, Thm C.3. Errors of the best subspaces; with SSN++'s fixed random subspaces the gain is measured (Prop. C.4).

A · NARROW MESSAGES

SSN++, step 1: a basis per layer, joined by transport

Give each layer its own fixed basis. To pass the k coordinates on, use the orthogonal factor of the cross-projection: it keeps their norm.

a · one direction, two subspaces



b · norm kept, k = 40 coordinates

between random subspaces of $\mathbb{R}^{2048}$

projection $U^\top U'$

≈ 14% (typical)

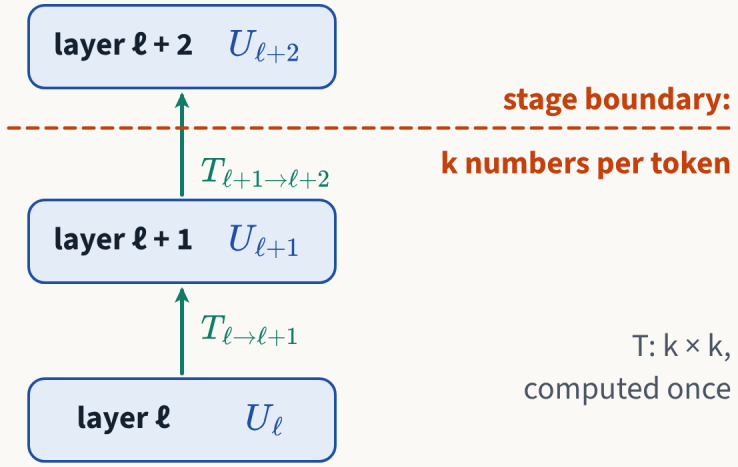
polar transport $PQ^\top$

100%

$$U^\top U' = P\Sigma Q^\top$$

Σ: cosines of the principal angles, ≤ 0.29 here

c · from layer to layer

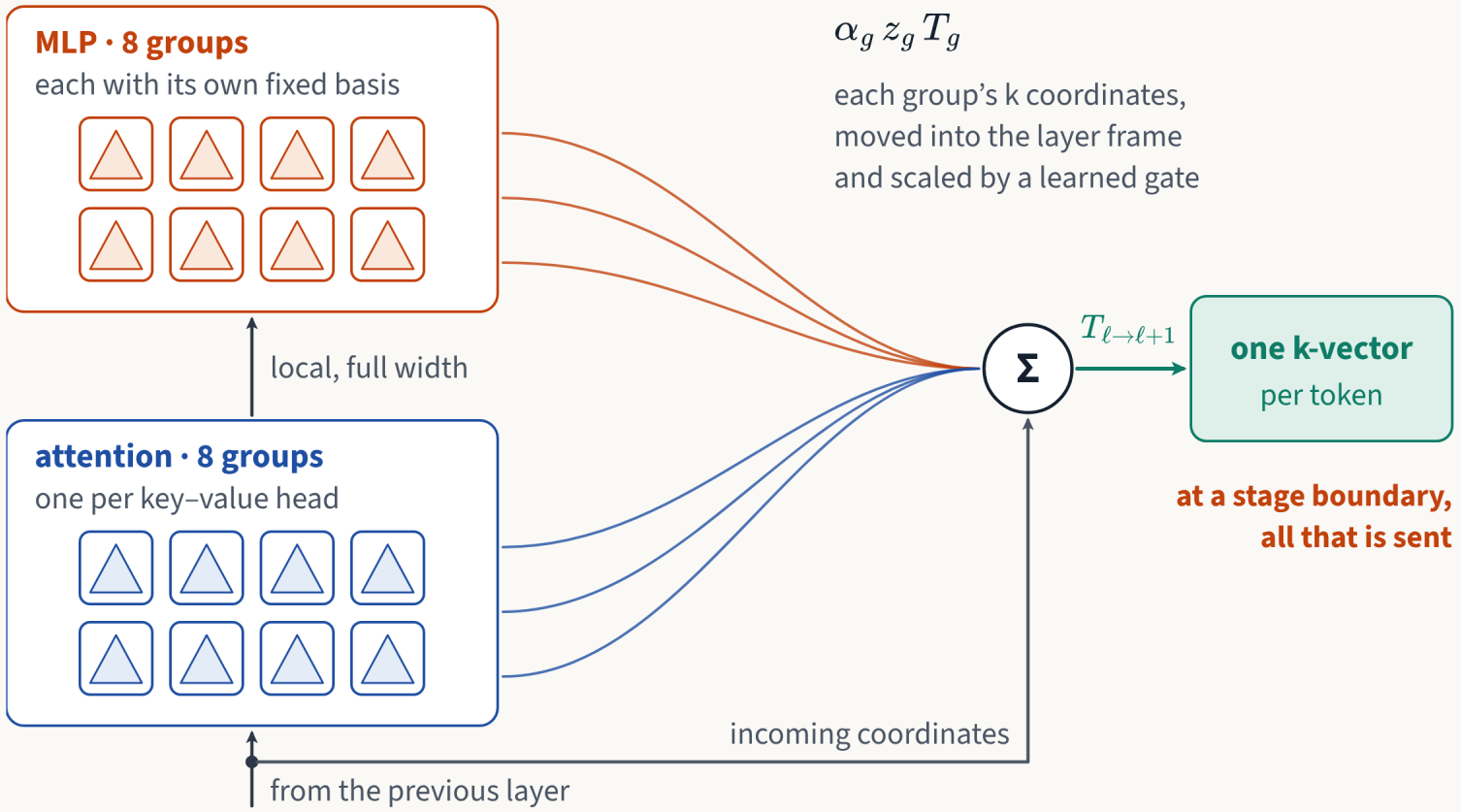


Transport preserves the k coordinates, not the ambient vector. On its own, a basis per layer stays within 0.01 nats of SSN: it is what enables step 2.

SSN++ (under review), Eq. 2, §3.1, App. B.2. Bases: fixed random draws; backward uses $T^\top$.

PR

SSN++, step 2: attention and MLP write through groups

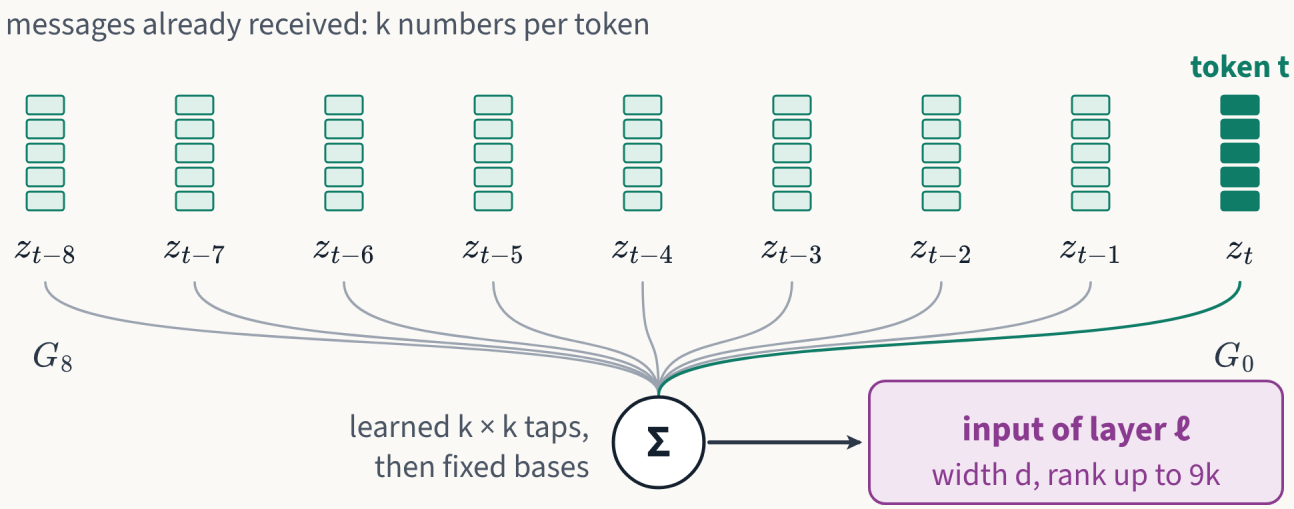


Same trainable entries as one shared factor; the groups' sum can use many more than k directions.

Learned gates decide how much of each group is passed on: still one k-vector per token.

SSN++ (under review), Eqs. 3–5, App. B.3–B.4. Gates start at one, so aggregation begins as a plain sum.

SSN++, step 3: a causal filter over earlier tokens



$$\hat{y}_t \;=\; z_t\, U_\ell^\top \;+\; \sum_{j=0}^{8} z_{t-j}\, G_j^\top V_j^\top$$

No extra traffic. The k numbers of the 8 preceding tokens have already crossed the boundary.

A richer input. The decoded input can reach rank (m + 1)k = 360, not k = 40.

Three filters per layer: one into the residual stream, one into the attention input, one into the MLP input.

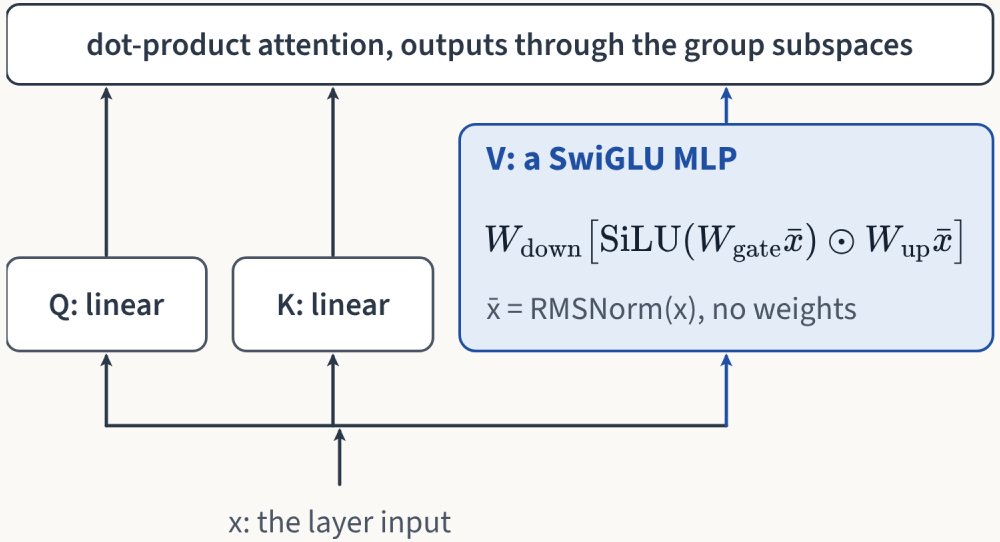
The taps start at zero, so training begins as SSN.

SSN++ (under review), Eq. 6, Prop. D.1, App. B.5. m = 8; G learned (k × k), V fixed (d × k).

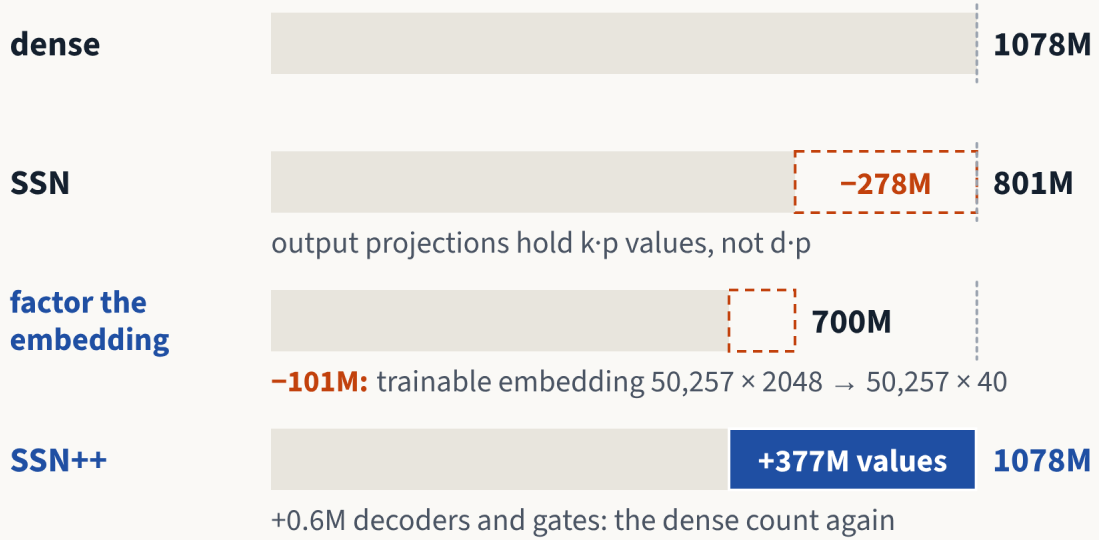
SSN++, step 4: spend the saved parameters on values

Confining output projections to k directions frees about a quarter of the parameters, and factoring the embedding 101M more. SSN++ spends them on the value maps of attention.

Attention in a compressed layer



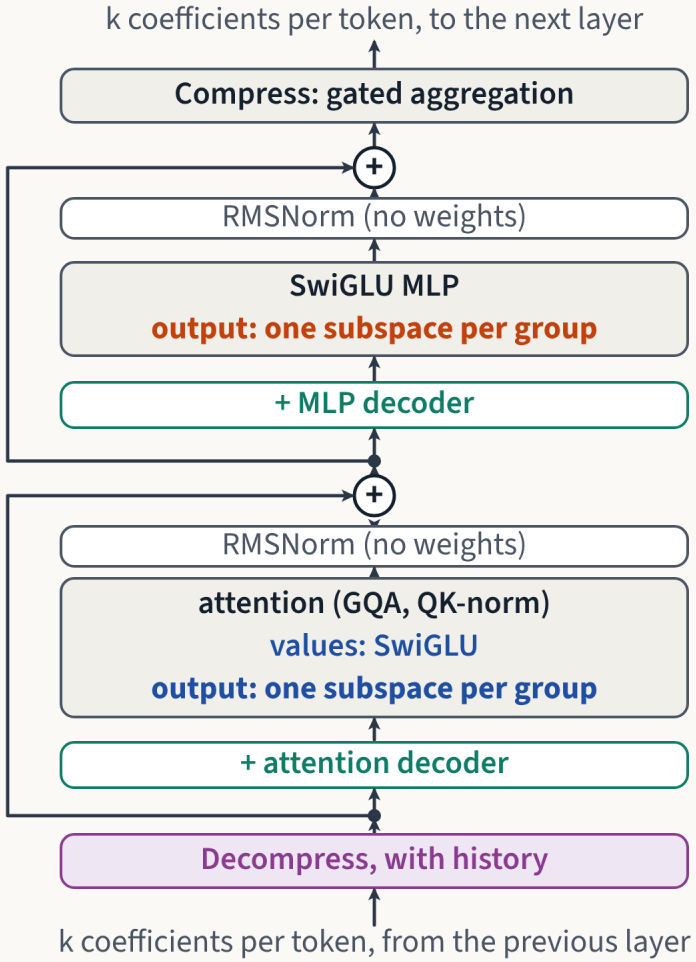
Trainable parameters at 1B



SSN++ has the dense model's parameter count and SSN's message size.

SSN++ (under review), Eq. 8, App. B.6, Table 5, App. G.3. Value width 5688 at 1B; values' stable rank 7.6 ⟶ 16.9.

SSN++: the full layer



Compress: gated aggregation

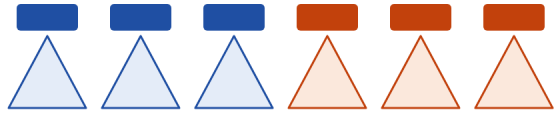
attention groups, each × its gate
MLP groups, each × its gate
incoming residual, embedding removed

moved into one frame and summed:
one k-vector, int8 or int4 on the wire

$$z_{\text{out}} = \left(r + \sum_g \alpha_g \, z_g \, T_g\right) T_{\ell \to \ell+1}$$

One subspace per group

Each group computes k coefficients and writes them through its own fixed random basis.



attention: 8 groups, MLP: 8 groups

Decoders on the receiving side



this + 8 earlier tokens

Causal filters over messages the receiver already holds, added to the attention and MLP inputs, not to the residual stream.

no extra traffic

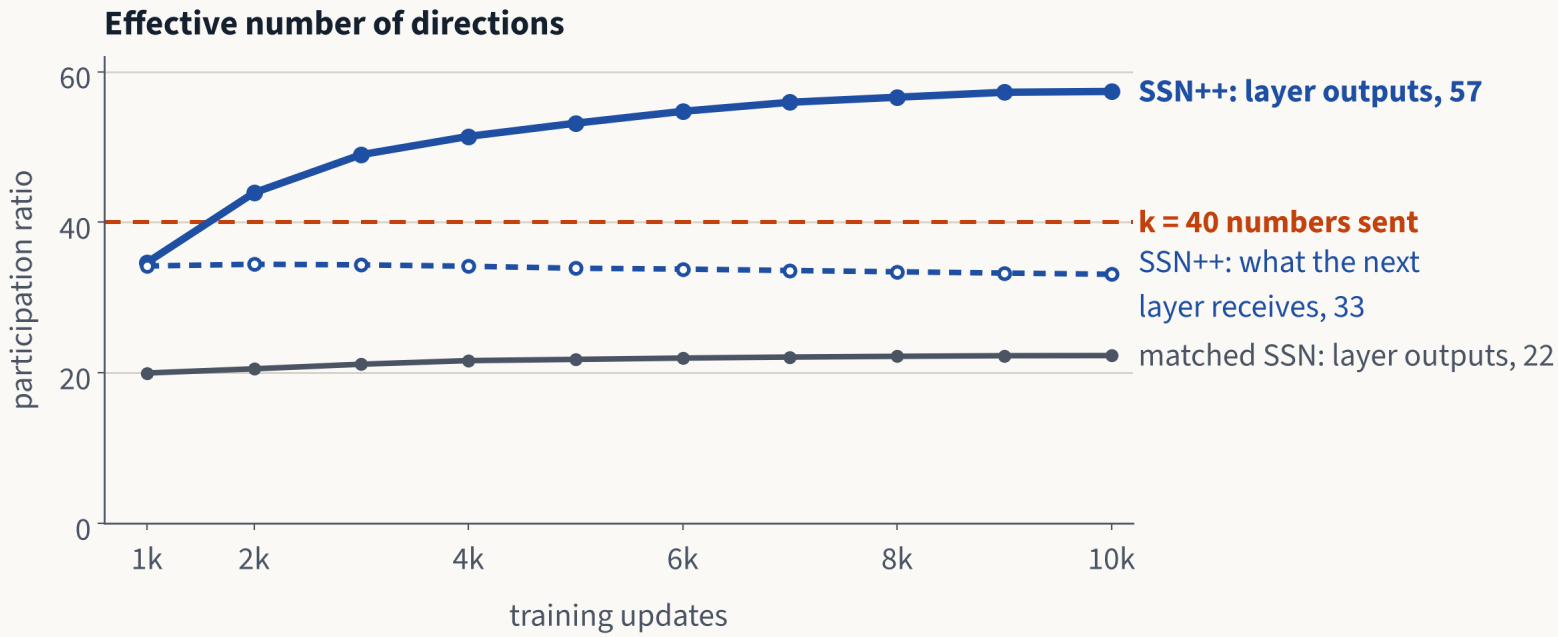
Decompress with history

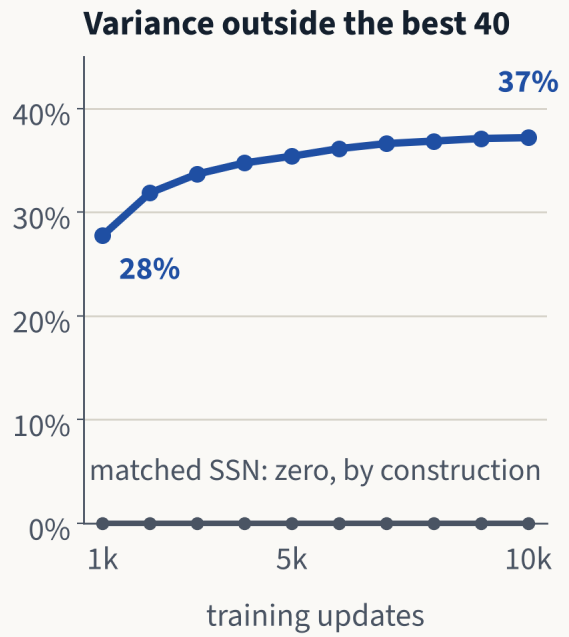
$$\hat{x}_t = c_t + z_t \, U_\ell^\top + \sum_j z_{t-j} \, G_j^\top V_j^\top$$

Embedding c rebuilt from token IDs, the current message in this layer's basis, and a causal filter over earlier messages.

SSN++ (under review, 2026), Fig. 1, Eqs. 3–6. Bases: fixed random draws. Decoder: G learned (k × k), V fixed (d × k).

Inside a stage, SSN++ uses more directions than it sends





Effective number of directions: participation ratio $\mathrm{tr}(C)^2/\mathrm{tr}(C^2)$ of the output covariance C.

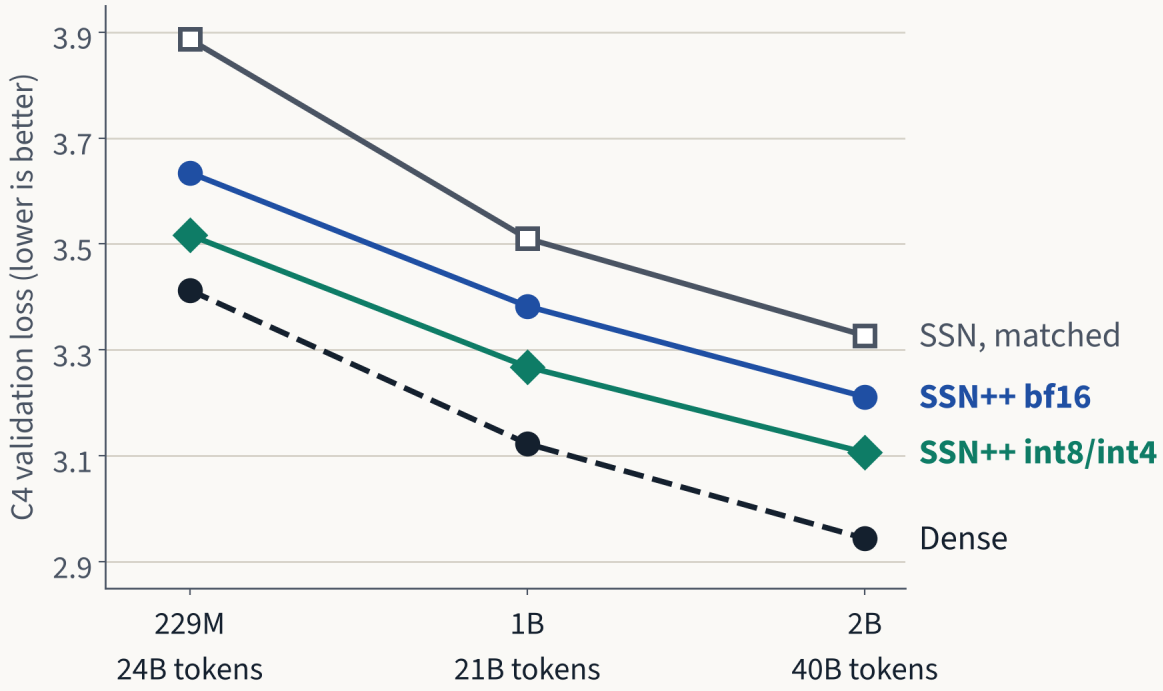
The boundary carries 40 numbers per token; the stage computes in more directions than that.

SSN++ (under review), §4.5, Fig. 3a–b, Fig. 11a, App. J.2–J.3: 1B ablation, k = 40.

A · NARROW MESSAGES

Quality at nearly matched boundary budgets

Same data and steps at each size. SSN++ sends about as many bytes as the matched SSN: 80, 160 and ≈200 bytes per token.



Rank k and share of SSN's gap to dense closed

	229M	1B	2B
bytes per token	80	160	≈200
bf16: rank k	20	40	50
bf16: gap closed	53%	33%	30%
int8/int4: rank k	52	104	130
int8/int4: gap closed	78%	63%	58%

Time per update at 1B, one 8-GPU node: +20% (bf16) and +27% (int8/int4) over SSN.

More quality at nearly the same bytes, for more local compute; still short of dense.

SSN++ (under review), Tables 1, 3; data-parallel runs. Matched SSN: 80/160/204 B. int8/int4: quantized messages used in training.

PR

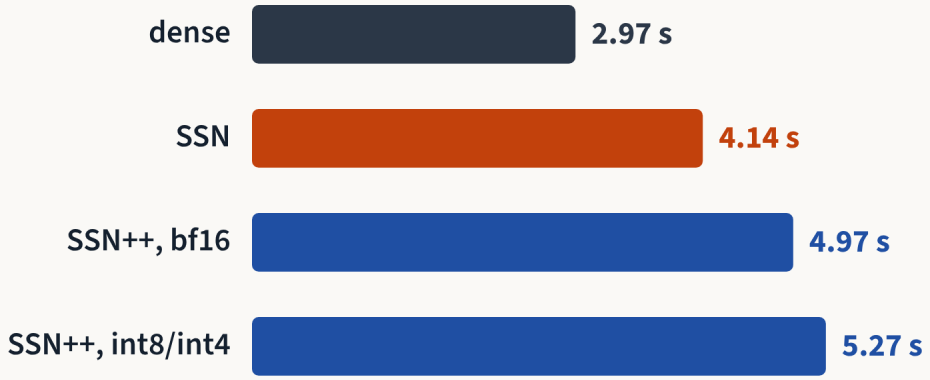
A · NARROW MESSAGES

Over slow links, the boundary dominates

Same 1B models, now as a 15-stage pipeline with emulated 200 Mb/s and 50 ms at each of the 14 boundaries.

one node, 8 × H100: seconds per update

compute only: the extra local work is visible



15-stage pipeline, 200 Mb/s + 50 ms per boundary

software-emulated links: the boundary dominates



Over slow links: SSN++ 19.6 s per update, SSN 19.1 s (+3%), dense 486 s.

SSN++ (under review, 2026), Table 3, App. F.5: 15 H100s, 32 microbatches, 1F1B, emulated links; 44 measured updates (8 for dense).

PR

A · 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

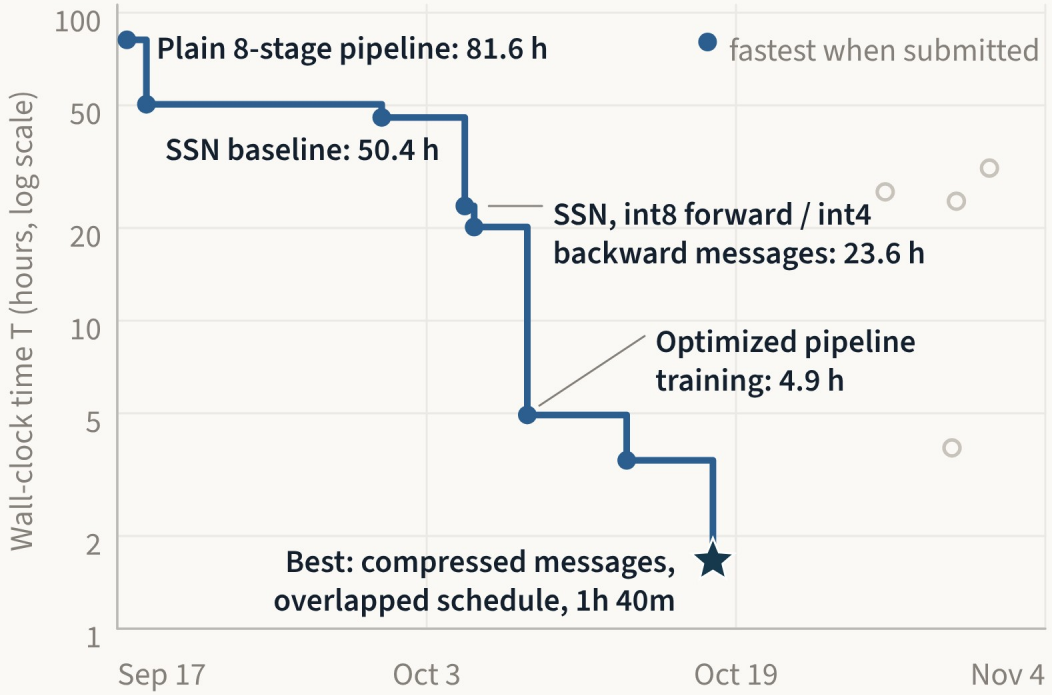
Open model, optimizer, schedule, compression

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



FROM A TO B

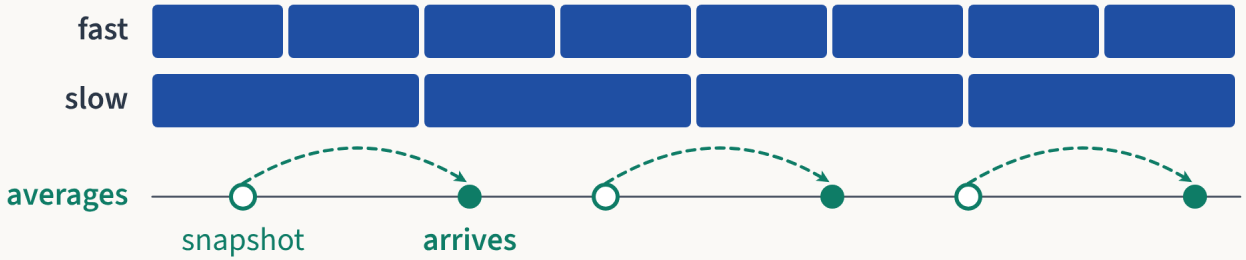
The other axis: replicas must agree

Wait for the slowest replica



the fast replica idles (dashed) before every average

Do not wait



nobody idles, but each average describes the past

Each stage is replicated across contributors that see different samples and run at different speeds.

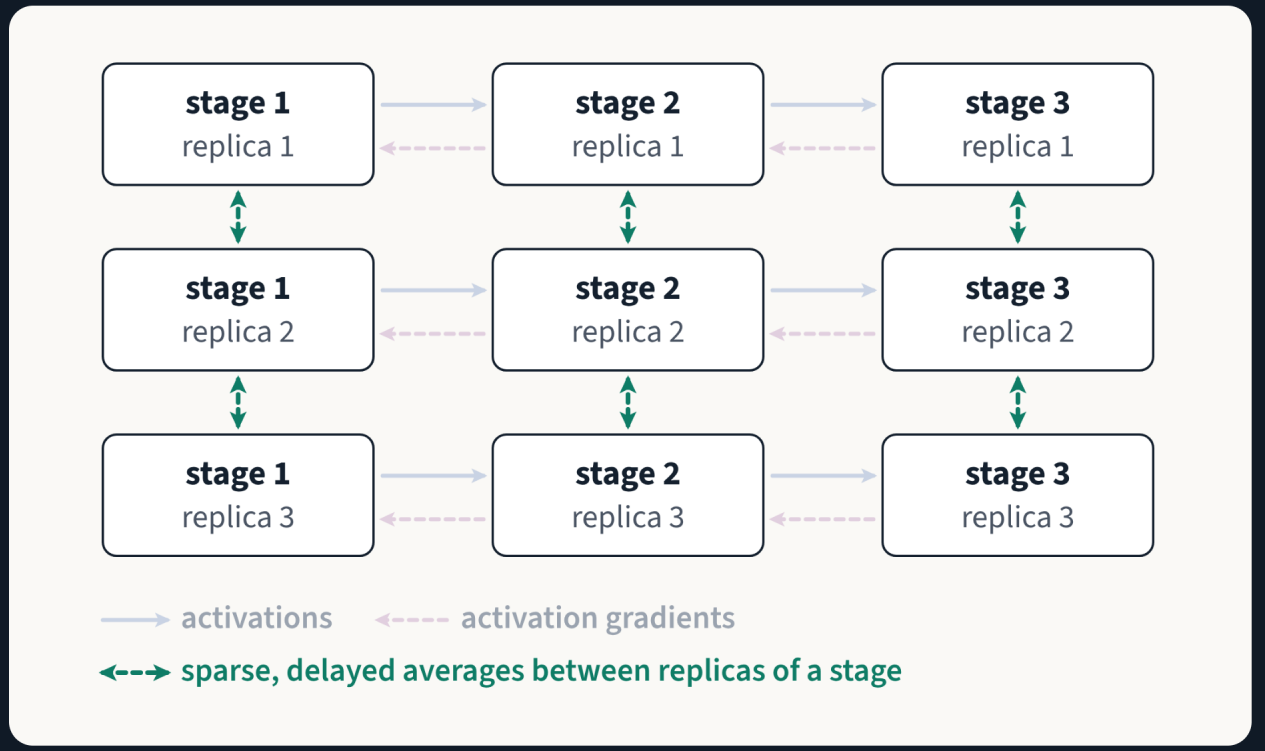
Wait for the slowest replica, and the fast ones idle.

Do not wait, and every average arrives late — describing where the group was, not where it is.

PART B

Asynchronous replicas

Q2 · Within a stage: how should replicas use averages that arrive late, from workers at different speeds?



B · ASYNCHRONOUS REPLICAS

Local steps, sparse averages, no waiting

Replicas of a stage must stay close, but averaging the whole stage every step is too slow over the internet.

Average every step all-reduce, as inside a datacenter		**the whole stage, every step** everyone waits for the slowest
Local steps average every H steps (local SGD, DiLoCo)		**H× fewer rounds** replicas drift in between
Sparse averages each round averages a random 5%		**20× fewer bytes per round** each round fixes only 5%
No waiting keep training while the average travels		**nobody idles** averages arrive late

one training step averaging

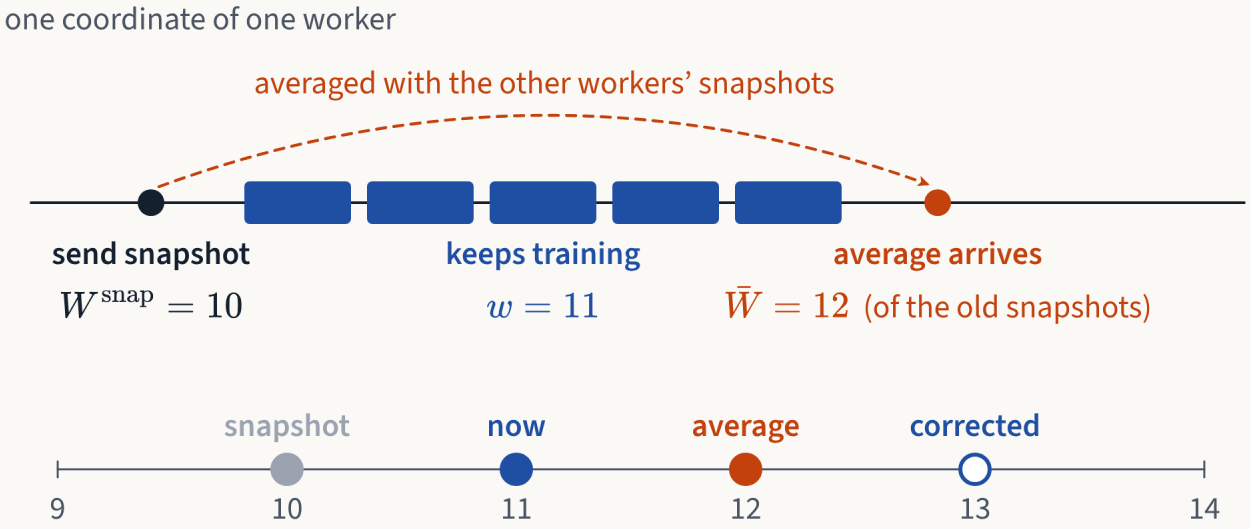
Agora used all three: a rotating 5% every 20 steps, in the background — 1/400 of the traffic, and nobody waits.

Local SGD: Stich, ICLR 2019 · DiLoCo: Douillard et al., WANT@ICML 2024 · AsyncSPARTA: Agora (Avraham et al., 2026, §4.2.2).

PR

Sparse averages arrive late

In the PACE experiments, every R = 5 ticks (1 tick = one step of the fastest worker), workers start averaging a shared random 5% of coordinates; the result returns D = 5 ticks later. Agora averages a rotating 5% every 20 steps.



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

$$\bar{W}[i] = \frac{\sum_{r \in P} \rho_r \, W_r^{\text{snap}}[i]}{\sum_{r \in P} \rho_r}, \qquad w_r[i] \leftarrow w_r[i] + \bar{W}[i] - W_r^{\text{snap}}[i]$$

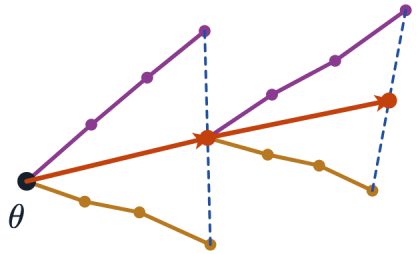
The correction keeps local progress, but the average still describes the past.

Delta rule: AsyncSPARTA in Agora (Avraham et al., 2026, §4.2.2, Alg. 3), LOSCAR-SGD (NeurIPS 2026). ρ: throughput weights; P: participants.

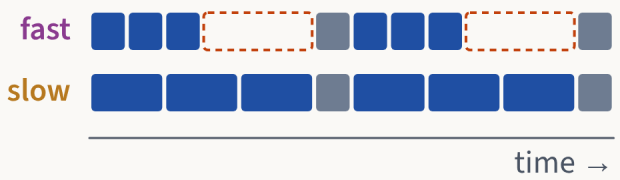
Keeping replicas together

DiLoCo drifts, then waits; AsyncSPARTA pulls with late averages; PACE also uses the group's velocity M.

DiLoCo (synchronous)

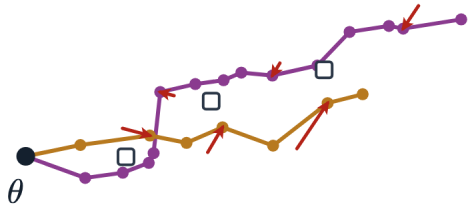


dashed: exchange — orange: outer update

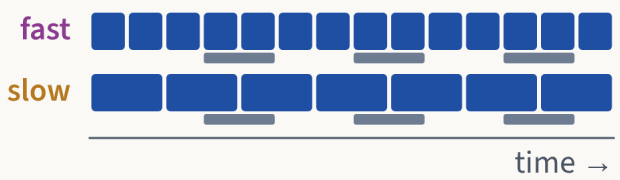


Replicas drift apart for H steps;
all wait for the slowest, then average.

AsyncSPARTA

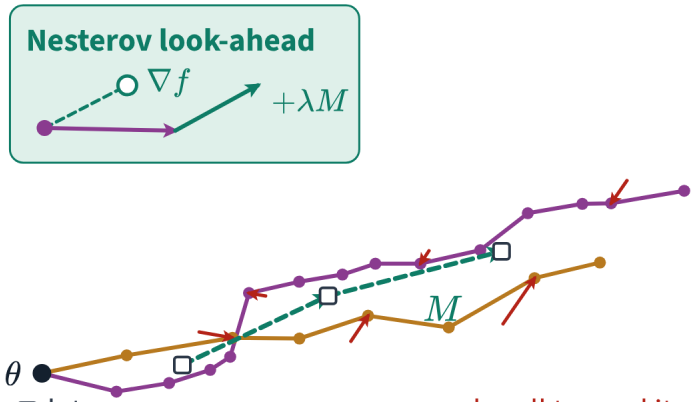


☐ late average — red: pull toward it

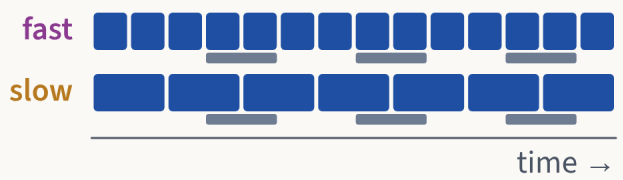


No waiting: each late average pulls
replicas together, but the gap reopens.

PACE



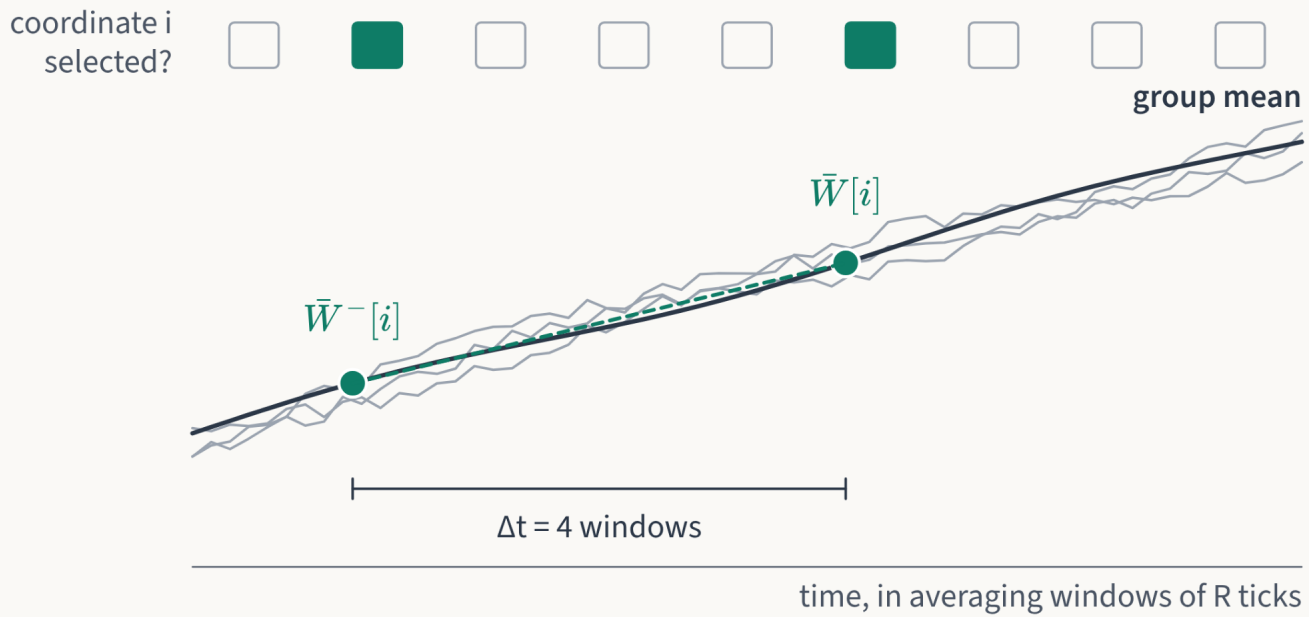
☐ late average — red: pull toward it



M, read from successive averages,
is used twice in every step (inset).

DiLoCo: Douillard et al., WANT@ICML 2024 · AsyncSPARTA: Agora (Avraham et al., 2026) · PACE: under review.

PACE: read the group's velocity



$$v_i^{\text{raw}} = \frac{\bar{W}[i] - \bar{W}^{-}[i]}{\Delta t_i}$$

Per coordinate, each with its own interval Δt.

Built from averages that already arrived: no extra payload.

	AsyncMesh (NeurIPS 2026)	PACE
smooths	its own staleness: how far its weights moved in one delay	the velocity of the group's averages, minus its own echo
uses it	once per arrival, to correct the late average	every step: a look-ahead (inner Nesterov) and a drift

PACE: Parameter Averaging with Consensus Extrapolation for Distributed Training (under review, 2026), §4.2, Eq. 4.

The PACE step

Between observations, each worker uses its velocity estimate M twice: to look ahead, and to drift with the group.

1 · look ahead
gradient at $w + \lambda_t M$

→

2 · local step
AdamW on that gradient

→

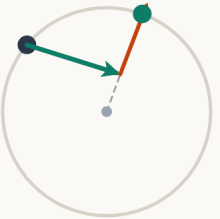
3 · assist
move along $\lambda_t M$
keeping each norm

→

4 · on arrival
snapshot correction, 5%

repeat every step; averages arrive every R ticks

assist on a weight matrix X:
shift by δ, then rescale
back to ‖X‖



reported settings: μ = 0.5 · q = 5% of coordinates · R = D = 5 ticks
coupling decays from 4 to 1 during warm-up

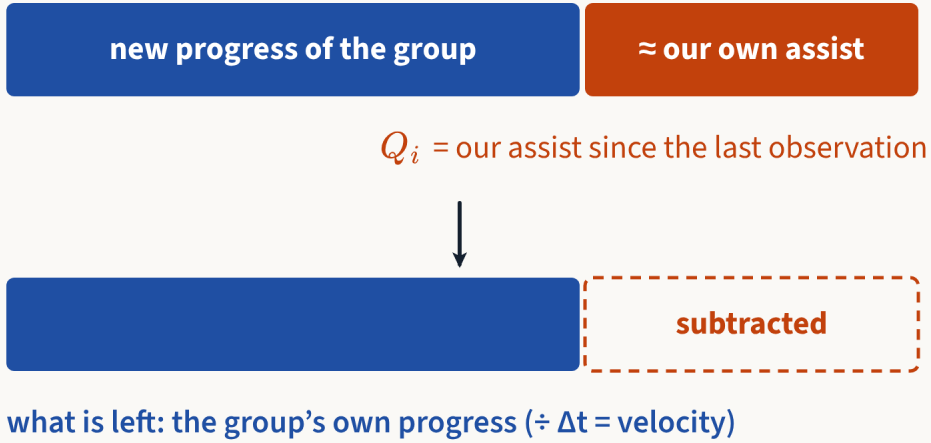
$$\mathcal{A}_\delta(X) = (X + \delta)\,\frac{\|X\|_F}{\|X + \delta\|_F}, \qquad \delta = \lambda_t M_X$$

PACE (under review, 2026), Eqs. 1 and 6, Alg. 1. Embeddings, output head and vectors use an additive assist.

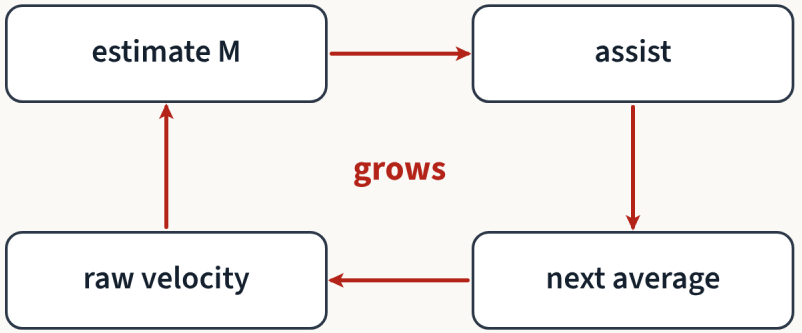
Removing the echo

The next average already contains the assist we applied. Counted as group progress, the estimate would feed on itself.

observed change of the mean between two observations



without subtraction: a feedback loop



ablation without subtraction: training diverges

$$\hat{v}_i = v_i^{\text{raw}} - \frac{Q_i}{\Delta t_i}, \qquad M_i \leftarrow \mu M_i + (1-\mu)\,\hat{v}_i$$

Subtract your own assist before estimating the group's velocity.

PACE (under review), Eqs. 3–4; Table 3 (226M, mixed speeds). Q includes the norm rescaling; exact if all workers apply the same assist.

AsyncSPARTA and PACE, side by side

AsyncSPARTA (each worker)

every step:

$g \leftarrow \nabla f(w)$

$w \leftarrow \mathrm{AdamW}(w, g)$

every R ticks:

pick a shared random 5% S ; $W^{\mathrm{snap}} \leftarrow w[S]$; start averaging

when $\bar{W}[S]$ arrives (D ticks later):

$w[S] \leftarrow w[S] + \bar{W}[S] - W^{\mathrm{snap}}[S]$ ▷ *delta rule*

PACE (each worker)

every step:

$g \leftarrow \nabla f(w + \lambda M)$ ▷ *look ahead: inner Nesterov*

$w \leftarrow \mathrm{AdamW}(w, g)$ ▷ *slow workers: B samples at fixed λ, M*

$w \leftarrow w + \lambda M$ ▷ *drift with the group (norm kept)*

every R ticks:

pick a shared random 5% S ; $W^{\mathrm{snap}} \leftarrow w[S]$; start averaging

when $\bar{W}[S]$ arrives (D ticks later):

$w[S] \leftarrow w[S] + \bar{W}[S] - W^{\mathrm{snap}}[S]$

$M[S] \leftarrow \mathrm{EMA}_{\mu}\big((\bar{W} - \bar{W}^{-} - Q)/\Delta t\big)[S]$

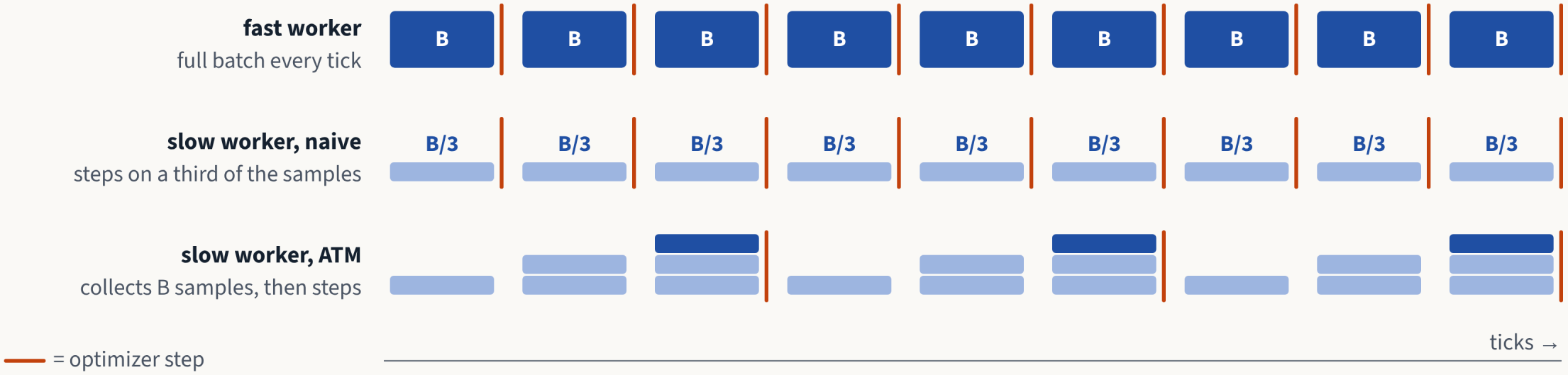
▷ *velocity of the averages, minus the echo Q of its own drift*

Same messages as AsyncSPARTA. In our practical implementation, PACE adds only one additional optimizer state.

PACE (under review), Alg. 1, Eqs. 2–6. Teal bars: what PACE adds or changes. Drift: norm-preserving per matrix; Q: the shift actually applied.

Equal samples per step for slow workers

If a one-third-speed worker steps every tick, each step sees B/3 samples. Accumulate-to-Match (ATM) waits until it has B.



$$m_r = \max\left(1,\ \mathrm{round}(1/\rho_r)\right)$$ ticks per step for a worker at relative speed ρ, each step on ≈B samples

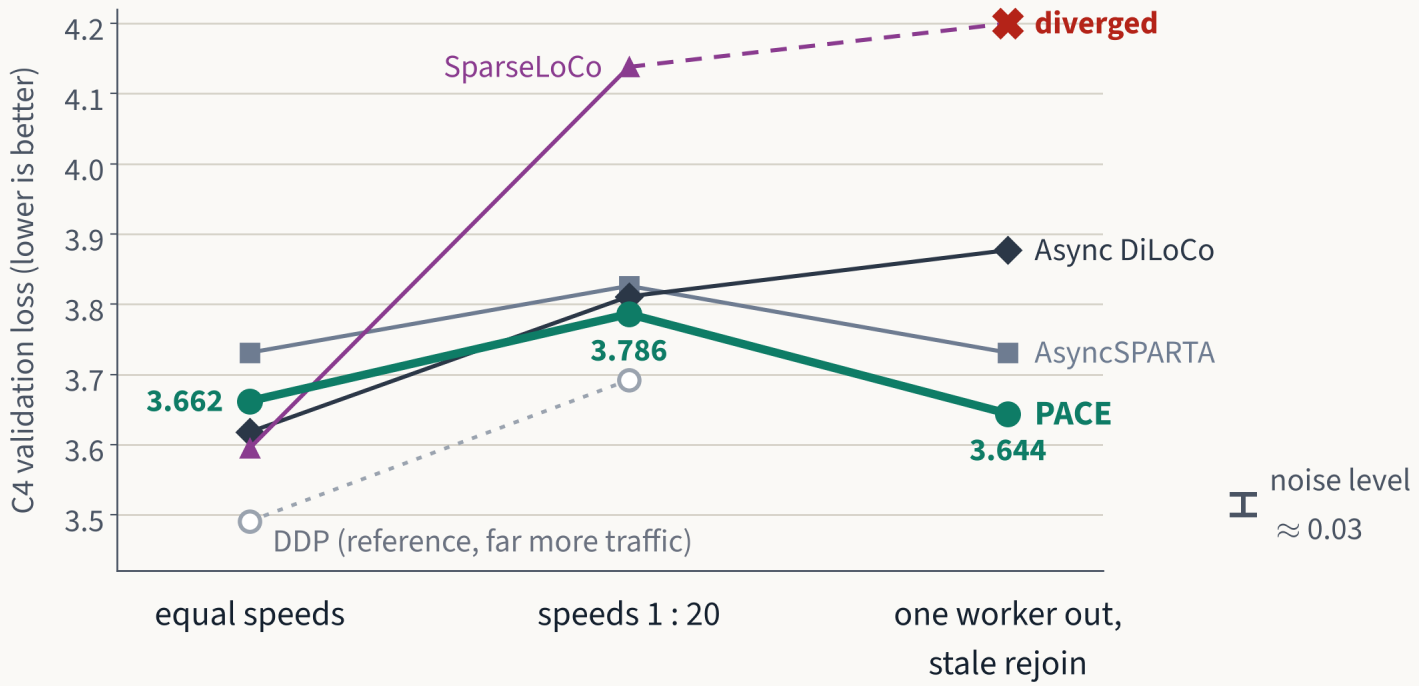
500M, mixed speeds: naive stepping instead of ATM worsens C4 loss by 0.130 (±0.004 over paired evaluations). The look-ahead pair λ, M is held fixed while a step accumulates.

PACE (under review, 2026), §4.3–4.4, Table 15 (paired statistics). Throughput weights alone do not restore unprocessed samples.

B · ASYNCHRONOUS REPLICAS

Results across worker regimes

500M model, 8 whole-model replicas, 30k steps; efficient methods get about one dense model of traffic per 100 steps. Outage: one worker absent from step 10k to 15k.



Equal speeds

SparseLoCo and Async DiLoCo are better.

Speeds 1 : 20

PACE and Async DiLoCo tie, within noise.

One worker out, stale rejoin

PACE unchanged; Async DiLoCo degrades; SparseLoCo diverges.

PACE has the best worst case across regimes; with equal speeds, SparseLoCo and Async DiLoCo do better.

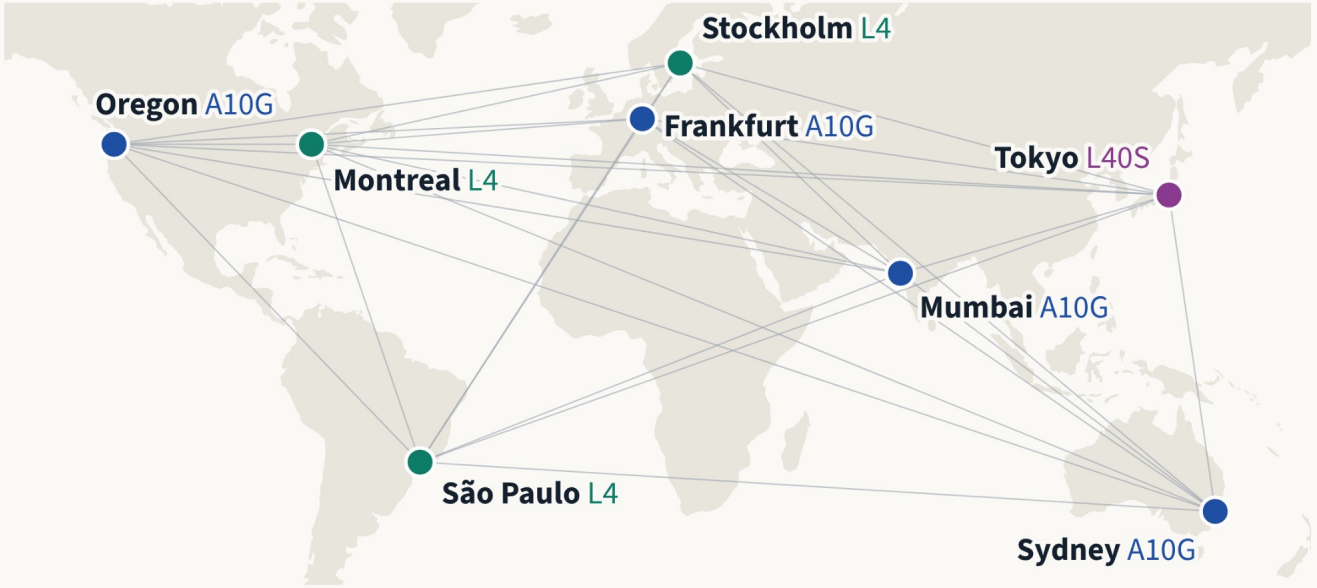
PACE (under review), Table 2; IID shards; differences below ≈0.03 are within noise. SparseLoCo diverged without a fresh checkpoint.

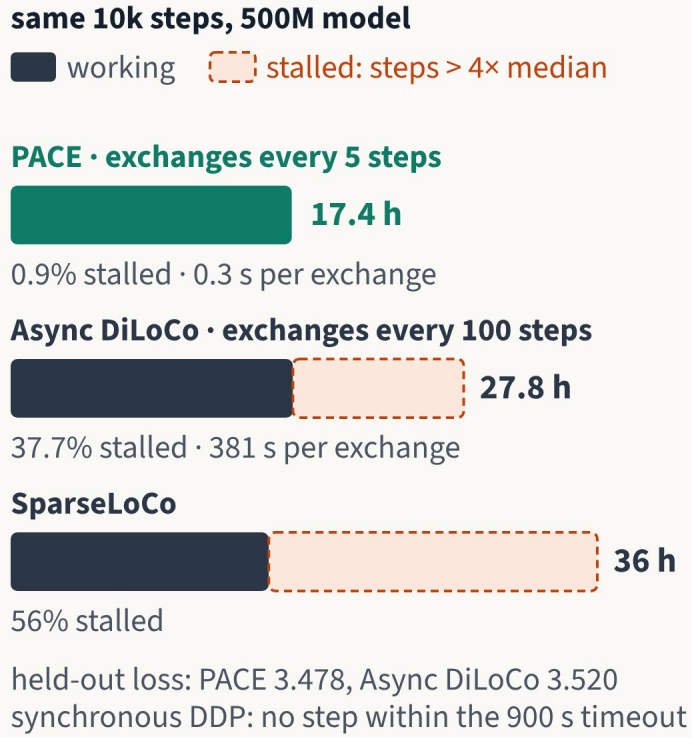
PR

B · ASYNCHRONOUS REPLICAS

A real wide-area run

Eight whole-model replicas in cloud regions on five continents: A10G, L4 and L40S GPUs; 3.06:1 speed spread; RTT 22–308 ms.





Async DiLoCo is asynchronous across workers, not within one: every 100 steps a worker sends its update and waits for the server's model before computing again, 381 s on these links. PACE's 5% exchanges overlap with training.

PACE (under review), Tables 1 and 8, App. A.4. Stalled: step intervals over 4× the median step, not all communication.

DiLoCo and RCN-C differ only in where the push enters

Written in classical form, both carry the same state and the same end-of-round update.

DiLoCo (outer Nesterov)
classical form; K workers

start of round r:

$\theta \leftarrow x_r + \mu v_{r-1}$ ▷ *jump to the look-ahead*

every local step:

$\theta \leftarrow \theta + U(\theta)$ ▷ *plain step*

end of round r:

$\bar{\Delta}_r \leftarrow \bar{\theta}_H - x_r - \mu v_{r-1}$ ▷ *from the start point*

$v_r \leftarrow \mu v_{r-1} + \alpha \bar{\Delta}_r$

$x_{r+1} \leftarrow x_r + v_r$

the same if $H = 1$

← all of μv_{r-1} · none yet →

← none · $\frac{\mu}{H} v_{r-1}$ per step →

Round-conditioned Nesterov (RCN-C)
K workers

start of round r:

$\theta \leftarrow x_r$ ▷ *start at x*

every local step:

$y \leftarrow \theta + \frac{\mu}{H} v_{r-1}$ ▷ *look ahead*

$\theta \leftarrow y + U(y)$ ▷ *the push stays*

end of round r:

$\bar{\Delta}_r \leftarrow \bar{\theta}_H - x_r - \mu v_{r-1}$ ▷ *remove the push*

$v_r \leftarrow \mu v_{r-1} + \alpha \bar{\Delta}_r$

$x_{r+1} \leftarrow x_r + v_r$

DiLoCo's usual form stores the look-ahead point. Nesterov acceleration in local-step methods (under review), Eqs. 10, 38–41.

Simplified PACE is round-conditioned Nesterov

Round-conditioned Nesterov applies the momentum at every inner step; DiLoCo applies it once per round, at the outer step.

Round-conditioned Nesterov (RCN-C)
K workers, outer scale α = 1

every local step:

$y \leftarrow \theta + \frac{\mu}{H} v_{r-1}$ ▷ *look ahead*

$\theta \leftarrow y + U(y)$ ▷ *the push stays*

end of round r:

$\bar{\Delta}_r \leftarrow \bar{\theta}_H - x_r - \mu v_{r-1}$ ▷ *remove the push*

$v_r \leftarrow \mu v_{r-1} + \bar{\Delta}_r$

$x_{r+1} \leftarrow x_r + v_r = \bar{\theta}_H$

the same if $\lambda H(1-\mu) = \mu$

$\frac{\mu}{H} v_{r-1} = \lambda M_{r-1}$

$\mu v_{r-1} = Q_r$

$H\lambda M_r = \mu(\mu v_{r-1} + \bar{\Delta}_r) = \mu v_r$

PACE, simplified
additive assist, dense averages, no delay

every local step:

$y \leftarrow \theta + \lambda M_{r-1}$ ▷ *look ahead*

$\theta \leftarrow y + U(y)$ ▷ *the push stays*

end of round r:

$\bar{\Delta}_r \leftarrow \bar{\theta}_H - x_r - Q_r$ ▷ *remove the echo*

$M_r \leftarrow \mu M_{r-1} + (1-\mu)\,\bar{\Delta}_r$

$x_{r+1} \leftarrow \bar{\theta}_H$ ▷ *delta rule*

θ: a worker's weights (bar: averaged over K workers); x: the average at the start of the round; U(y): an AdamW step at y; M, λ, Q: PACE's velocity, coupling and echo. Nesterov acceleration in local-step methods (under review), Table 4. PACE (under review), Eqs. 1, 4, 6, 7, App. L.3.

PR

Primal RCN: GPA, smoothed once per round

Round-conditioned Nesterov, primal (RCN-P)
K workers, H local steps per round

every local step:

$y \leftarrow \mu_y\, x_r + (1 - \mu_y)\, z$ ▷ *anchor fixed*

$z \leftarrow z + U(y)$ ▷ *no push, no echo*

end of round r:

$z_{r+1} \leftarrow z_r + \alpha\, \overline{(z_{k,H} - z_r)}$ ▷ *average*

$x_{r+1} \leftarrow \mu_x\, x_r + (1 - \mu_x)\, z_{r+1}$ ▷ *once per round*

the same if $H = 1,\ K = 1,\ \alpha = 1$

x_r: GPA's x, held for the round

$K = 1,\ \alpha = 1$: $z_{r+1} = z_{1,H}$

$H = 1$: once per round = every step

GPA
one worker, every step

every step:

$y \leftarrow \mu_y\, x + (1 - \mu_y)\, z$ ▷ *look-ahead point*

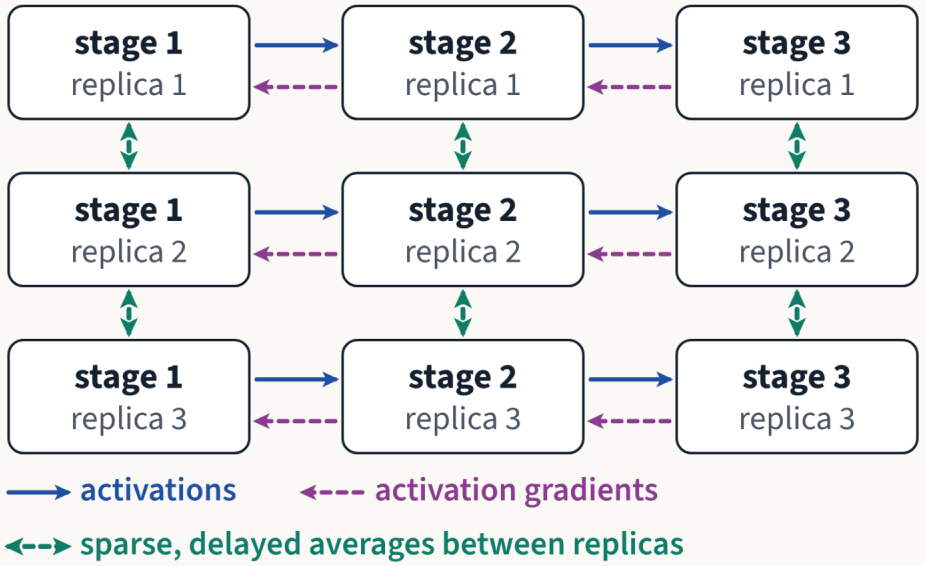
$z \leftarrow z + U(y)$ ▷ *fast iterate*

$x \leftarrow \mu_x\, x + (1 - \mu_x)\, z$ ▷ *smooth every step*

x: slow iterate; z: fast iterate. Nesterov acceleration in local-step methods (under review), Eqs. 15, 60–62. GPA: arXiv 2512.17131.

PUTTING IT TOGETHER

Recap: one mesh, two methods



SSN++ · pipeline-parallel communication

Along each row: at 1B, 160 bytes per token at every stage boundary, 51× less than dense. Richer stages close 33–63% of SSN's gap to dense at 1B (30–78% across sizes), for 20–27% more time per update.

PACE · data-parallel communication, unequal workers

Down each column: sparse exchanges overlap local work (0.9% long-stall time across eight regions). Best worst case across the three regimes tested; WAN: 17.4 h vs 27.8 h.

Narrow messages along the pipeline; sparse averaging overlaps work across replicas.

SSN++ and PACE: under review. So far tested separately: SSN++ in data-parallel runs and an emulated pipeline; PACE with whole-model replicas.

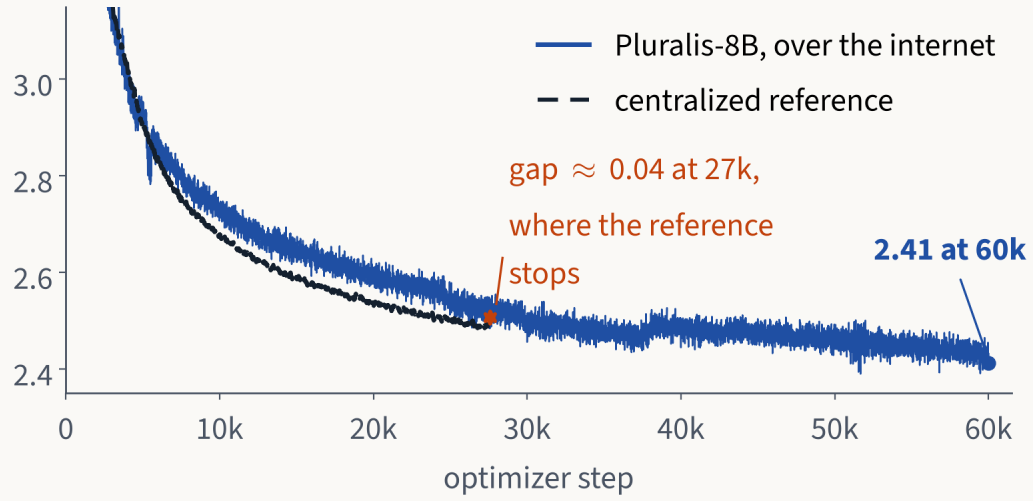
PR

PUTTING IT TOGETHER

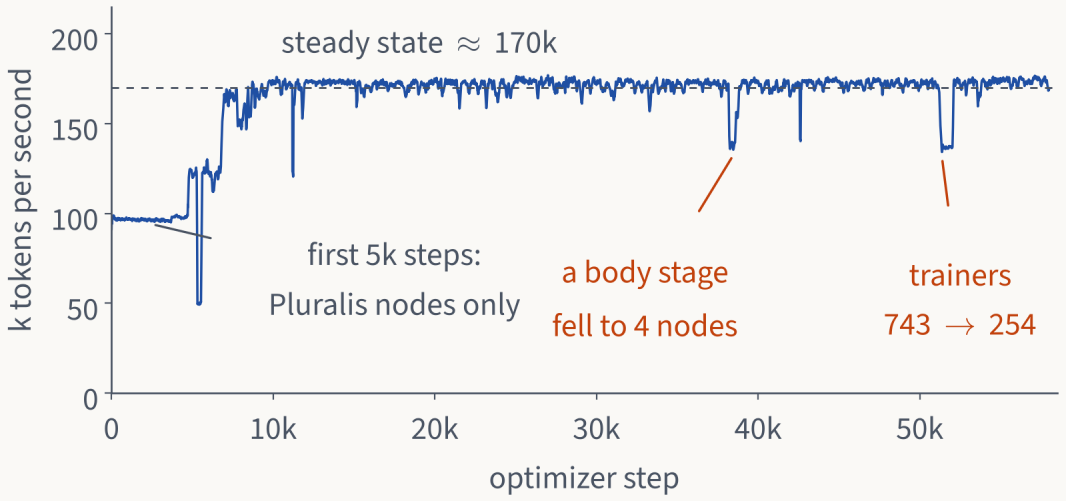
Pluralis-8B on Agora: the run

8.6B	500B	330	669 / 607	≈170k	63%
parameters, 7 stages	tokens in 40 days	contributor GPUs	joins / departures	tokens per second	of a centralized H100 baseline's efficiency

Training loss against a centralized reference



Throughput while nodes came and went



Within ≈0.04 of the centralized reference where it stopped, at ≈170k tokens per second while nodes came and went.

Avraham et al., Agora, arXiv 2607.13332 (2026), Figs. 17a, 19b. Efficiency: per pooled dense-BF16 TFLOP.

PR

PUTTING IT TOGETHER

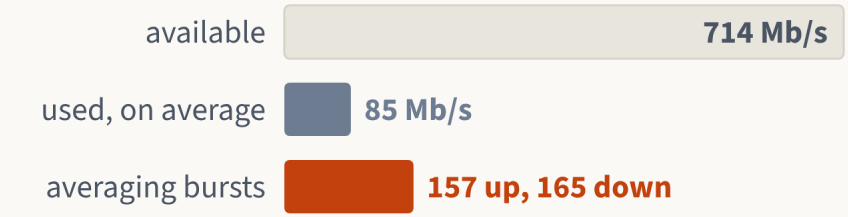
Where the time went: compute and latency, not bandwidth

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)

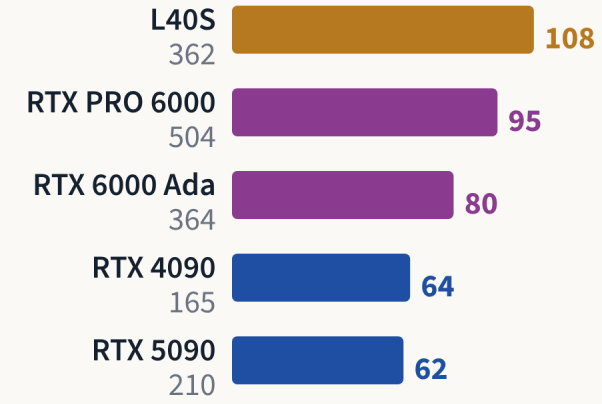


Link use, body nodes (mean)

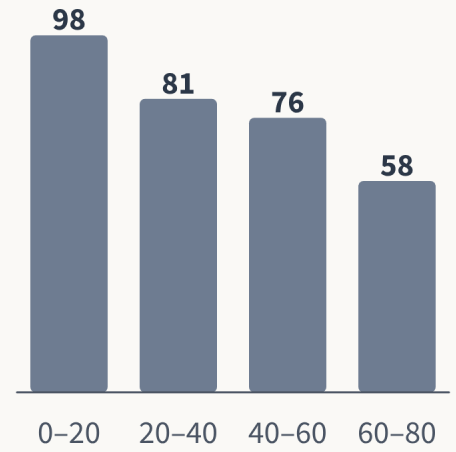


Work per node, batches per minute

by GPU (peak TFLOP/s)



by latency (ms)



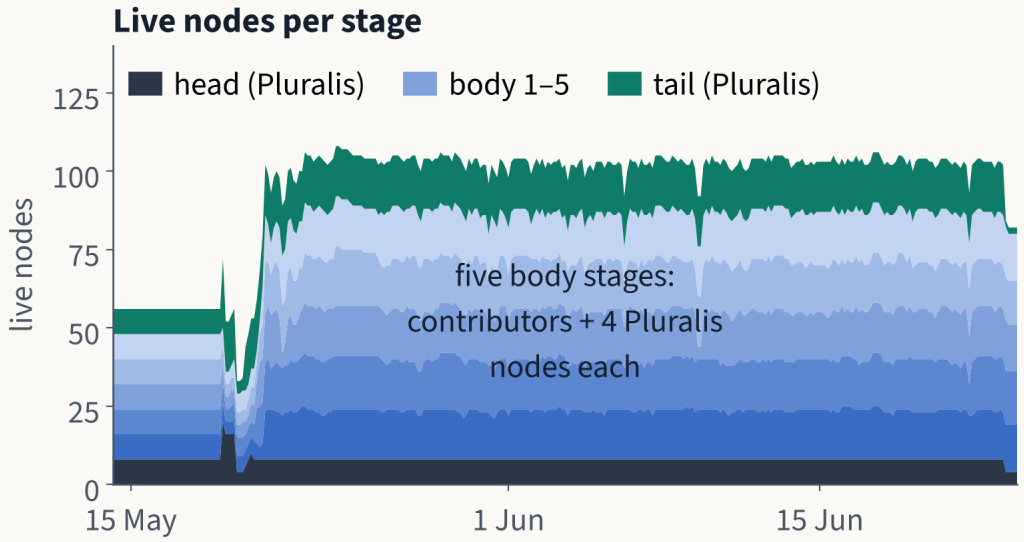
Bandwidth above the 200 Mb/s floor made no difference: about 80 microbatches per minute in every band up to 1 Gb/s.

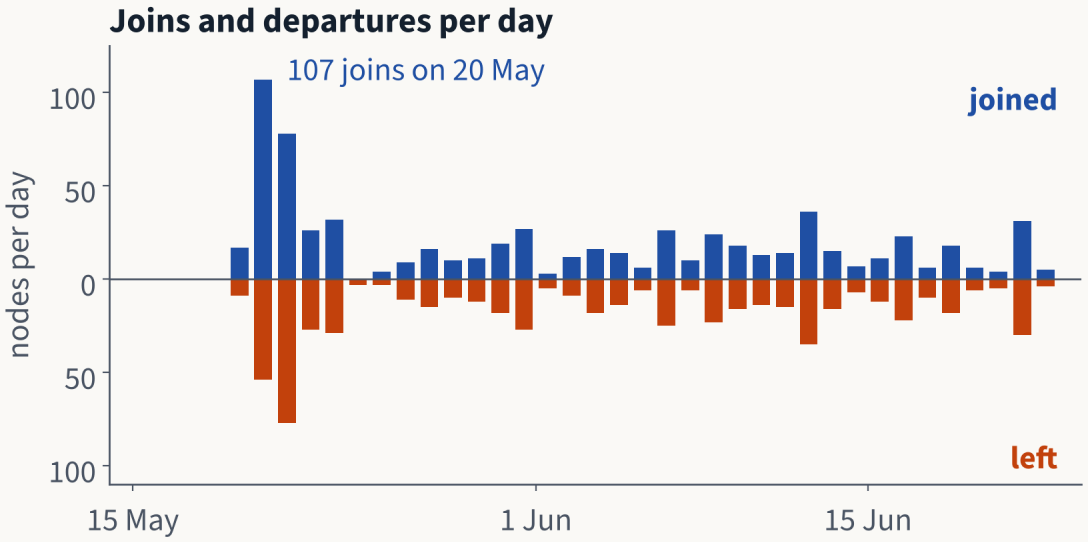
Agora report: Table 4 and Fig. 18 (steps 20k–30k), Fig. 30 (body nodes), Fig. 36. Latency and GPU type are confounded (Fig. 37).

PR

PUTTING IT TOGETHER

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

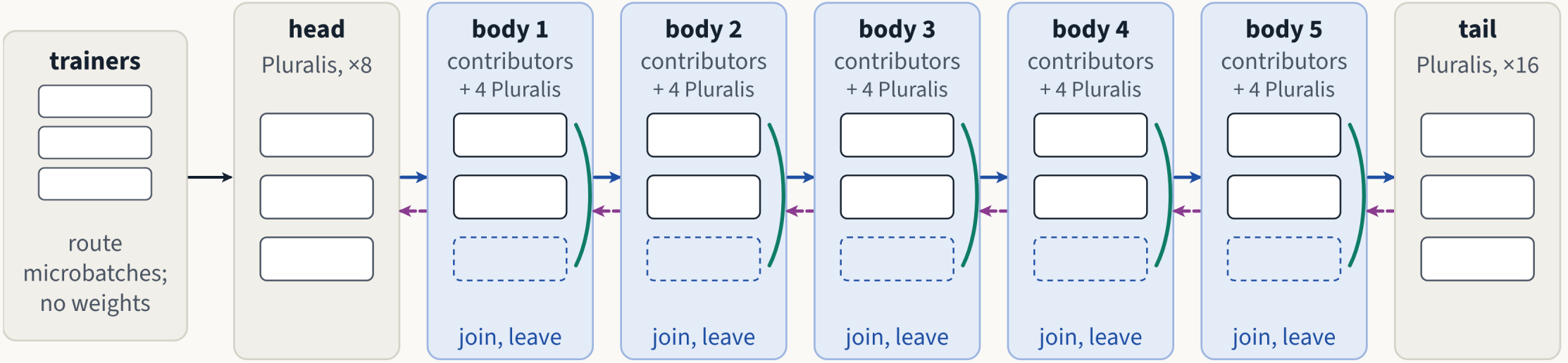
after a mass exit near step 40k, the loss recovered within a few thousand steps, without intervention

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

PR

PUTTING IT TOGETHER

Where SSN++ and PACE would fit



SSN++ for SSN: each body stage computes in many subspaces; the boundary keeps its size.

PACE for AsyncSPARTA: aimed at this run's unequal GPUs and stale rejoins; same messages per exchange.

SSN++, PACE (under review): tested separately, ≤ 2B. Per-step traffic depends on cadence: 5% every 5 ticks (PACE) vs every 20 steps (Agora).

THANK YOU

Questions

Protocol Learning

Split one model across contributors on the internet; no one needs the whole model.

A · Across stages

Communicate narrow, compute wide: SSN++ (under review) closes 30–78% of SSN's gap to dense, at nearly the same bytes per token.

B · Within a stage

Late averages carry motion: PACE (under review) reads the group's velocity from late averages, after removing its own echo.

Thalaiyasingam Ajanthan

Pluralis Research · pluralis.ai

PR

BACKUP · A

From SSN to SSN++: what changes

	SSN (NeurIPS 2025)	SSN++ (under review)
Where outputs live	one shared k-dimensional subspace, for every layer	a fixed basis per group and per layer; their sum spans more than k directions
What is sent	the k coefficients of the residual: exact	one gated k-vector: many groups, one message
How it is read	$x = c + zU^T$	the same, plus a causal decoder over this and the 8 earlier token messages
Bytes per token	160 B: bf16, k = 40	160 B: bf16 with k = 40, or int8/int4 with k = 104
C4 loss, 1B model	3.51 (dense: 3.12)	3.38 bf16 · 3.27 int8/int4: 33% and 63% of the gap
Compute per update	4.14 s (dense: 2.97 s)	4.97 s bf16 · 5.27 s int8/int4

Same bytes on the link; richer computation on both sides, at more compute; still short of dense.

SSN++ (under review), Tables 1 and 3: 1B, 21B FineWeb-Edu tokens, one 8 × H100 node; SSN = parameter-matched baseline (same SwiGLU).

BACKUP · B

Which parts of PACE matter

226M model, mixed worker speeds; one component removed at a time. C4 loss, lower is better.

Configuration	C4 loss	Change
PACE	4.460	—
no echo subtraction	diverged	—
no decay of the coupling (λ = 4 throughout)	6.490	+2.030
no consensus velocity (no mean-motion assist)	4.599	+0.140
no norm preservation	4.481	+0.021

Echo subtraction and a decaying coupling are essential at this operating point; removing the mean-motion assist raises the loss by 0.14.

PACE (under review, 2026), Table 3. One operating point; other sparse fractions and delays have their own failure boundaries (Apps. D–E).

PR

The convergence result, in more detail

Algorithm analyzed

PACE-SGD: SGD local steps, an additive assist clipped by a guard, the snapshot correction on random masks with delay D.

Assumptions

L-smooth, possibly nonconvex f; unbiased gradients with bounded variance; bounded updates; IID shards; diminishing step size.

Terms of the bound

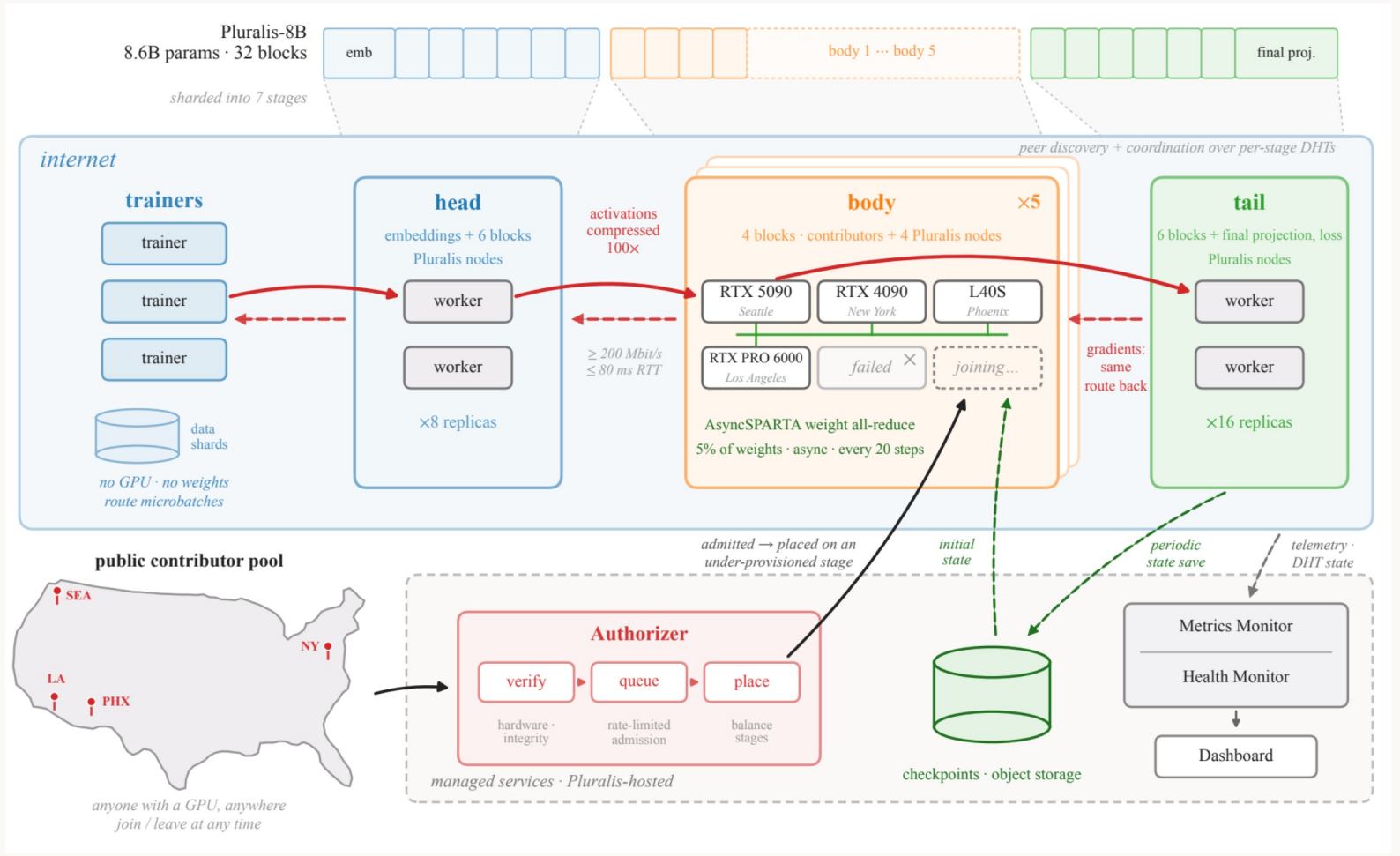
$O(1/\sqrt{(NBT)})$, the minibatch-SGD rate, while the step-size caps and consensus term do not dominate; consensus of order 1/T, growing with the refresh horizon R/q and with D^2; a floor proportional to the average squared guard.

Reading it

The bound is at the uniform mean of the replicas; the consensus term covers their distance from it. A suitably decaying guard removes the floor; nothing shows the assist accelerates.

PACE (under review, 2026), Appendix L, Remark 6. Experiments use the norm-preserving AdamW variant, which the theorem does not cover.

The Agora system



Avraham et al., Agora: Collective and Permissionless Internet-Scale Pretraining of Large Language Models, arXiv 2607.13332 (2026), Figure 1.

References: methods

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

SSN++: layerwise subspace networks for low-bandwidth pipeline training. Under review, 2026.

PACE: parameter averaging with consensus extrapolation for distributed training. Under review, 2026.

Nesterov acceleration in local-step methods. Under review, 2026.

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

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

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

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

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

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

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

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

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

References: systems and context

Avraham et al. *Agora: collective and permissionless internet-scale pretraining of large language models* (§4.2.2 and Alg. 3: delta-rule sparse averaging). arXiv 2607.13332, 2026.

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

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

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

Avraham et al. *Node0: model parallel training over the internet with protocol models.* GitHub, Pluralis Research.

torchft: fault tolerance for PyTorch. GitHub, meta-pytorch/torchft.

Dolatabadi et al. *SENTINEL: stagewise integrity verification for pipeline parallel decentralized training.* arXiv 2603.03592, 2026.

Long et al. *Unextractable protocol models: collaborative training and inference without weight materialization.* NeurIPS 2025.

Jeff Dean, in conversation at Sequoia Capital AI Ascent, 2 May 2025 (published transcript).

Eric Schmidt, guest lecture, Stanford ECON 295/CS 323, 9 April 2024 (unofficial transcript; video withdrawn by Stanford).

Nic Lane, talk at the Protocol Learning workshop, 26 April 2026.

Lambert. *Open language models.* Guest lecture, Harvard CS 2881R, 22 Oct 2026; compute per lab from Epoch AI (US labs, end of 2025) and D. Patel (Chinese labs, Aug 2026).