NESTEROV ACCELERATION IN LOCAL STEP METHODS

Anonymous authors
Paper under double-blind review

ABSTRACT

Local-step methods reduce communication in distributed training by allowing workers to perform multiple optimizer steps between synchronizations. Prior work has shown that incorporating Nesterov acceleration can improve local-step training, but how it should interact with both periodic communication and updates within each local block remains less understood. We identify four constructions for such interaction, each restricted to communicating one model-size vector between workers per round. We analyze classical, modern, and primal formulations across these constructions, including generalized variants with independent control of smoothing and look-ahead, and establish when they are equivalent. Our language-modeling experiments identify two ways to improve DiLoCo-style training: tuning generalized retention and extrapolation coefficients, and reporting a smoothed iterate. They also identify a construction that performs well with frequent communication, and in what settings Nesterov acceleration does not help.

1 INTRODUCTION

Local-step training reduces communication by allowing K workers to each take H optimizer steps between synchronizations. Often, outer optimizers are used to improve upon parameter averaging by applying an outer optimizer step to the aggregated local displacements (Reddi et al., 2021; Wang et al., 2020). For language-model training, DiLoCo found Nesterov to be the strongest outer optimizer among the alternatives it evaluated (Douillard et al., 2023).

Interestingly, the original DiLoCo paper reported benefits even in the single-worker setting, showing that these benefits are not dependent on inter-worker communication. These benefits were further investigated in subsequent scaling experiments (Charles et al., 2025) and in SNOO (Kallusky et al., 2025). SNOO connects the two-loop structure to the Lookahead optimizer (Zhang et al., 2019), and proposes an averaging-based implicit-regularization interpretation.

Generalized primal averaging (GPA) develops this perspective by replacing DiLoCo's periodic outer updates with per-step primal Nesterov averaging (Defazio et al., 2026). It separates interpolation and smoothing coefficients, interpreting the resulting method as a smoothed alternative to single-worker DiLoCo. This connects Nesterov acceleration to base-optimizer wrapping approaches such as primal averaging and Schedule-Free (Defazio, 2020; Defazio et al., 2024).

From this wrapper perspective, DiLoCo applies Nesterov to an aggregated block of local updates, whereas GPA motivates wrapping individual updates within the block. Extending this idea to multiple workers raises the question of *how per-step Nesterov acceleration should use shared and local history when communication remains periodic*. This is the core investigation of this paper.

We compare four constructions under the constraint of one model-sized communication payload every H local steps. *Step-H outer Nesterov* (SHN) wraps the whole block, and includes DiLoCo. Within blocks, *round-conditioned local Nesterov* (RCN) uses a fixed shared reference, *per-local-step Nesterov* (PLN) evolves local Nesterov state, and *two-timescale Nesterov* (TTN) combines both. We study their available classical, modern, and primal formulations, allow for independent coefficients inspired by GPA, and establish when their local trajectories and round-boundary states are equivalent.

Our language-model experiments compare nine accelerated variants and FedAvg (McMahan et al., 2017) on a 124M-parameter model trained on FineWeb with AdamW. We vary worker count, batch size, and synchronization interval, and transfer selected recipes to a 1B-parameter model. We identify two ways to improve DiLoCo-style training: generalizing momentum retention and extrapolation,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Formulation		Properties	
Classical		Look-ahead point	y_t^{C}
$y_t^{\mathrm{C}} = x_t^{\mathrm{C}} + \mu v_{t-1},$		Reported iterate	x_t^{C}
$v_t = \mu v_{t-1} + U_t^{\mathrm{C}}(y_t^{\mathrm{C}}),$	(3)	Nesterov state	$(x_t^{\mathrm{C}}, v_{t-1})$
$x_{t+1}^{\mathrm{C}} = x_t^{\mathrm{C}} + v_t.$		Initialization	$x_0^{\mathrm{C}} = x_0,\ v_{-1} = 0$
Modern		Look-ahead point	x_t^{M}
$v_t = \mu v_{t-1} + U_t^{\mathrm{M}}(x_t^{\mathrm{M}}),$	(4)	Reported iterate	x_t^{M}
$x_{t+1}^{\mathrm{M}} = x_t^{\mathrm{M}} + U_t^{\mathrm{M}}(x_t^{\mathrm{M}}) + \mu v_t.$		Nesterov state	$(x_t^{\mathrm{M}}, v_{t-1})$
		Initialization	$x_0^{\mathrm{M}} = x_0,\ v_{-1} = 0$
Primal		Look-ahead point	y_t^{P}
$y_t^{\mathrm{P}} = \mu x_t^{\mathrm{P}} + (1-\mu) z_t,$		Reported iterate	x_t^{P}
$z_{t+1} = z_t + U_t^{\mathrm{P}}(y_t^{\mathrm{P}}),$	(5)	Nesterov state	(x_t^{P}, z_t)
$x_{t+1}^{\mathrm{P}} = \mu x_t^{\mathrm{P}} + (1-\mu) z_{t+1}.$		Initialization	$x_0^{\mathrm{P}} = z_0 = x_0$

Table 1: Nesterov formulations with tied coefficient $0 \le \mu < 1$.

and reporting the smoothed iterate. Our RCN-C variant is particularly competitive with frequent communication, and we identify settings where Nesterov acceleration does not improve on FedAvg.

2 Background: Nesterov acceleration

This section reviews established Nesterov formulations and their interpretation as base-optimizer wrappers, introducing the notation used for our local-step constructions and independent coefficients. Appendix A.1 gives broader related work.

2.1 Classical, modern, and primal formulations

We use the following standard Nesterov recurrence as our reference (Nesterov, 1983; Defazio, 2019):

$$y_t = x_t + \mu(x_t - x_{t-1}), \tag{1}$$

$$x_{t+1} = y_t + U_t(y_t). \tag{2}$$

Here $t = 0, 1, \ldots$, $\mu \in [0, 1)$ is constant, and $x_{-1} = x_0$. $U_t(a)$ denotes the parameter displacement supplied by a base optimizer evaluated at a, with its state and stochastic inputs implicit (Defazio et al., 2026), e.g. $U_t(a) = -\gamma_t g(a; \xi_t)$ for SGD. The *look-ahead point* y_t determines where the base optimizer update is evaluated. The *reported iterate* x_t is the parameter vector used for model evaluation. The *Nesterov state* is the minimal set of variables needed to continue the recurrence, here (x_t, x_{t-1}), distinct from the base optimizer's state. A single coefficient μ controls both look-ahead displacement and retention of history in the Nesterov state. We call this a *tied* coefficient choice, Section 4 generalizes it by allowing two independent coefficients.

Table 1 gives equivalent classical (C), modern (M), and primal (P) formulations of this recurrence (Bengio et al., 2013; Defazio, 2019; Defazio et al., 2026). In exact arithmetic, the state and complete-update scale conversions below preserve the recurrences and base-update evaluation points under compatible initialization, matched minibatches, and compatible base-optimizer states and transitions. Formulation superscripts are omitted when unambiguous. Appendices B.2.2 and B.2.2 give detailed source locations and the equivalence proof.

The **classical formulation** rewrites the canonical recurrence using momentum state $v_{t-1} = x_t - x_{t-1}$, with $x_t^{\mathrm{C}} = x_t$ and $U_t^{\mathrm{C}} = U_t$ (Nesterov, 1983; Sutskever et al., 2013). The **modern formulation** stores $x_t^{\mathrm{M}} = y_t^{\mathrm{C}} = x_t^{\mathrm{C}} + \mu v_{t-1}$, retaining the same momentum state and base displacement. Avoiding a separately constructed or temporarily installed look-ahead vector makes this representation convenient for frameworks such as Lasagne and TensorFlow/Keras, hence *modern* (Bengio et al., 2013; Lasagne Developers, n.d.; TensorFlow Developers, n.d.). It natively reports a different sequence, with the classical reported iterate recovered as $x_t^{\mathrm{C}} = x_t^{\mathrm{M}} - \mu v_{t-1}$. The **primal formulation** represents momentum through a fast sequence z_t and its averaged sequence x_t (Lan, 2012; Defazio et al., 2026). It is equivalent to the classical formulation via $x_t^{\mathrm{P}} = x_t^{\mathrm{C}}$, $z_t = x_t^{\mathrm{C}} + \frac{\mu}{1-\mu} v_{t-1}$, $y_t^{\mathrm{P}} = y_t^{\mathrm{C}}$,

Single-worker DiLoCo	**GPA**	**FedAvg**
$\theta_{r,0} = x_r^{\mathrm{M}},$ $\theta_{r,h+1} = \theta_{r,h} + U_{r,h}(\theta_{r,h}),$ $d_r = \theta_{r,H} - x_r^{\mathrm{M}},$ $v_r = \mu v_{r-1} + \alpha d_r,$ $x_{r+1}^{\mathrm{M}} = x_r^{\mathrm{M}} + \alpha d_r + \mu v_r.$ (7)	$y_t^{\mathrm{P}} = \mu_y x_t^{\mathrm{P}} + (1-\mu_y) z_t,$ $z_{t+1} = z_t + U_t^{\mathrm{P}}(y_t^{\mathrm{P}}),$ $x_{t+1}^{\mathrm{P}} = \mu_x x_t^{\mathrm{P}} + (1-\mu_x) z_{t+1}.$ (8)	$\theta_{k,r,0} = x_r,$ $\theta_{k,r,h+1} = \theta_{k,r,h} + U_{k,r,h}(\theta_{k,r,h}),$ $\Delta_{k,r}(x_r) = \theta_{k,r,H} - x_r,$ $\overline{\Delta}_r(x_r) = \frac{1}{K}\sum_{k=1}^{K} \Delta_{k,r}(x_r),$ $x_{r+1} = x_r + \overline{\Delta}_r(x_r).$ (9)

Table 2: **(Left two columns)**: Single-worker DiLoCo applies modern Nesterov once per block of H base updates, while GPA applies generalized primal Nesterov at every base update. **(Right)**: FedAvg applies the mean block displacement without a Nesterov wrapper, as discussed in Section 3.

$U_t^{\mathrm{C}} = U_t^{\mathrm{M}} = (1-\mu)U_t^{\mathrm{P}}$. Averaging attenuates the primal displacement by $1-\mu$, requiring this complete-update scaling. For $\mu > 0$ and nonzero momentum, z_t lies beyond y_t along the momentum direction, so interpolation between x_t and z_t reaches the extrapolation point.

2.2 Nesterov as an optimization wrapper

Each formulation wraps an update-producing map with Nesterov state, allowing the wrapper and the update-producing map it wraps to vary separately.

The Nesterov wrapper admits equivalent history and iterate-averaging interpretations. Its momentum state exponentially weights previous base updates, while the classical/primal reported iterate accumulates their contributions, or equivalently exponentially averages fast iterates:

$$v_t = \sum_{j=0}^{t} \mu^{t-j} U_j^{\mathrm{C}}(y_j), \quad x_t^{\mathrm{C}} = x_0 + \sum_{j=0}^{t-1} \frac{1-\mu^{t-j}}{1-\mu} U_j^{\mathrm{C}}(y_j) = \mu^t x_0 + (1-\mu)\sum_{j=1}^{t} \mu^{t-j} z_j = x_t^{\mathrm{P}}. \quad (6)$$

Like Polyak–Ruppert averaging, primal Nesterov reports an averaged sequence (Ruppert, 1988; Polyak & Juditsky, 1992), but feeds it back into optimization through look-ahead interpolation rather than using it only for output. This feedback also permits other averaging schedules, as in Schedule-Free's combination of interpolation with cumulative averaging (Defazio et al., 2024).

Single-worker DiLoCo and *generalized primal averaging* (GPA), shown in Table 2, illustrate different wrapper and update-producing map choices (Douillard et al., 2023; Defazio et al., 2026). DiLoCo wraps each round of H base updates with modern Nesterov, using $U_r^{\mathrm{M}} = \alpha d_r$, where d_r is the round's net parameter displacement and $\alpha > 0$ is a constant outer scale. GPA instead wraps every base update with primal Nesterov, independently controlling interpolation through $\mu_y \in [0, 1]$ and averaging through $\mu_x \in [0, 1)$. GPA motivates per-step wrapping as a smoothed alternative to single-worker DiLoCo and reports improved language-model training (Defazio et al., 2026). Its comparison changes both wrapper frequency and coefficient coupling rather than isolating either choice.

3 Nesterov acceleration with multi-worker local steps

We extend the single-worker wrappers of Section 2.2 to K workers performing H local updates between synchronizations. Extending single-worker DiLoCo gives DiLoCo, whereas extending GPA raises interesting questions about how per-step Nesterov updates should interact with periodic communication. This motivates four constructions for incorporating Nesterov into this setting.

Workers minimize $F(x) = K^{-1}\sum_{k=1}^{K} F_k(x)$, with rounds indexed by r and local steps by $h = 0, \ldots, H-1$. Each base update $U_{k,r,h}$ uses the worker's minibatch and base-optimizer state, with the same initialization, schedule, and state-persistence policy across constructions.

The right column of Table 2 defines the multi-worker local-step approach without a Nesterov wrapper, which is equivalent to FedAvg (McMahan et al., 2017). DiLoCo wraps this block update with modern Nesterov (Douillard et al., 2023), giving $v_r = \mu v_{r-1} + \alpha\overline{\Delta}_r(x_r)$, $x_{r+1} = x_r + \alpha\overline{\Delta}_r(x_r) + \mu v_r$.

Table 3: Four Nesterov constructions with one model-sized vector averaged per round. *Connections concern local updates, not full-algorithm equivalence under the original synchronization policies (Section 3.3). Dashes indicate no direct identification made here.

Construction	Within each block	At synchronization	Related methods
Step-H outer Nesterov (SHN)	Fixed shared state sets the start; only the base optimizer advances.	Update shared Nesterov state.	DiLoCo (SHN-M).
Round-conditioned local Nesterov (RCN)	A fixed shared reference conditions every local update.	Update the shared reference.	–
Per-local-step Nesterov (PLN)	Worker-local Nesterov state evolves at every step.	Reconstruct state from mean displacement per step.	Nesterov FedAvg*, FedNAG* (PLN-C). Multi-worker GPA* (PLN-P).
Two-timescale Nesterov (TTN)	Combine a fixed shared reference with evolving local momentum.	Update shared reference; reset local momentum.	–

Classical (RCN-C)

$$
\begin{aligned}
\theta_{k,r,0} &= x_r, \\
y_{k,r,h} &= \theta_{k,r,h} + \frac{\mu}{H} v_{r-1}, \\
\theta_{k,r,h+1} &= y_{k,r,h} + U_{k,r,h}(y_{k,r,h}), \\
\Delta_{k,r}^{\text{fresh}} &:= \theta_{k,r,H} - x_r - \mu v_{r-1}, \\
v_r &= \mu v_{r-1} + \frac{\alpha_r}{K} \sum_k \Delta_{k,r}^{\text{fresh}}, \\
x_{r+1} &= x_r + v_r.
\end{aligned}
\tag{10}
$$

Primal (RCN-P)

$$
\begin{aligned}
z_{k,r,0} &= z_r, \\
y_{k,r,h} &= \mu x_r + (1-\mu) z_{k,r,h}, \\
z_{k,r,h+1} &= z_{k,r,h} + U_{k,r,h}(y_{k,r,h}), \\
\overline{\Delta}_r^z &= \frac{1}{K} \sum_k (z_{k,r,H} - z_r), \\
z_{r+1} &= z_r + \alpha_r \overline{\Delta}_r^z, \\
x_{r+1} &= \mu x_r + (1-\mu) z_{r+1}.
\end{aligned}
\tag{11}
$$

Table 4: Tied round-conditioned Nesterov recurrences. RCN-C repeatedly injects fixed incoming momentum, while RCN-P interpolates against a fixed slow iterate. Initialize $v_{-1} = 0$ for RCN-C and $z_0 = x_0$ for RCN-P.

Table 3 distinguishes our four constructions by how Nesterov state influences local updates and changes at synchronization. We constrain all our constructions to communicate one model-sized vector per round, matching FedAvg and DiLoCo. *Round-level* (outer) state is updated at synchronization boundaries, while *worker-local* state can evolve within a block. *Shared* state is identical across workers. The suffixes C, M, and P identify classical, modern, and primal formulations, respectively.

3.1 Step-H outer Nesterov (SHN)

Step-H outer Nesterov (SHN) applies Nesterov once per round to the aggregated local-block displacement. Its modern formulation, SHN-M, is the DiLoCo update introduced above (Douillard et al., 2023) and the SHN formulation used in our experiments. Classical and primal wrappers give SHN-C and SHN-P, equivalent under the state and scale transformations of Section 2 for tied coefficients. Appendix B.3 gives all three recurrences, the equivalence proof, and parameter-averaging implementations.

3.2 Round-conditioned local Nesterov (RCN)

RCN conditions each local update on a shared reference held fixed throughout the round. The classical and primal variants use different references and generally produce different local trajectories. Table 4 gives the tied recurrences.

Classical (RCN-C). RCN-C distributes the incoming shared momentum across the local updates. Before applying outer momentum, it subtracts the total injected displacement from the endpoint displacement to avoid double counting. This subtraction does not undo the injection's effect on intervening base-update evaluations.

PLN-C

$$
\begin{aligned}
x_{k,r,0} &= x_r, \\
u_{k,r,0} &= v_{r-1}, \\
y_h &= x_h + \mu_y u_h, \\
u_{h+1} &= \mu_x u_h + U_h(y_h), \\
x_{h+1} &= x_h + u_{h+1}, \\
x_{r+1} &= \frac{1}{K}\sum_k x_{k,r,H}, \\
v_r &= \frac{x_{r+1} - x_r}{H}.
\end{aligned} \tag{12}
$$

PLN-P

$$
\begin{aligned}
x_{k,r,0} &= x_r, \\
z_{k,r,0} &= z_r, \\
y_h &= \mu_y x_h + (1-\mu_y) z_h, \\
z_{h+1} &= z_h + U_h^{\mathrm{P}}(y_h), \\
x_{h+1} &= \mu_x x_h \\
&\quad + (1-\mu_x) z_{h+1}, \\
x_{r+1} &= \frac{1}{K}\sum_k x_{k,r,H}, \\
z_{r+1}^{\mathrm{1V}} &= x_{r+1} \\
&\quad + \frac{\mu_x}{1-\mu_x}\frac{x_{r+1} - x_r}{H}.
\end{aligned} \tag{13}
$$

TTN-C

$$
\begin{aligned}
\theta_{k,r,0} &= x_r, \\
u_{k,r,0} &= 0, \\
y_h &= \theta_h + c_r + \mu^{\ell} u_h, \\
u_{h+1} &= \mu^{\ell} u_h + U_h(y_h), \\
\theta_{h+1} &= \theta_h + c_r + u_{h+1}, \\
\overline{\Delta}_r^{\mathrm{corr}} &= \frac{1}{K}\sum_k \theta_{k,r,H} \\
&\quad - x_r - \mu^{\mathrm{r}} v_{r-1}, \\
v_r &= \mu^{\mathrm{r}} v_{r-1} + \alpha_r \overline{\Delta}_r^{\mathrm{corr}}, \\
x_{r+1} &= x_r + v_r.
\end{aligned} \tag{14}
$$

Table 5: Local-state Nesterov recurrences with one-vector communication. Within each local block, worker and round indices are suppressed: $x_h = x_{k,r,h}$, and similarly for y, z, u, θ and $U_h = U_{k,r,h}$. PLN uses the independent coefficients defined in Section 4, with $0 \le \mu_x < 1$ for PLN-P reconstruction. TTN-C uses $\mu^{\mathrm{r}}, \mu^{\ell}$ for its round and local timescales, respectively, with tied coefficients within each timescale and $c_r = (\mu^{\mathrm{r}}/H) v_{r-1}$. Initially, the shared states satisfy $v_{-1} = 0$ for PLN-C and TTN-C, and $z_0 = x_0$ for PLN-P.

Primal (RCN-P). RCN-P holds the slow anchor x_r fixed while workers advance their fast iterates. Interpolation changes the look-ahead point without adding the reference displacement to the fast iterate, so no echo subtraction is needed. RCN-P is equivalent to SHN under appropriate state, coefficient, and update-scale transformations, whereas RCN-C generally is not (Section 5). Appendix B.4 gives the local-path comparison, state-recovery identities, and modern-representation caveat. Appendix B.4 gives the independent-coefficient recurrences.

3.3 Per-local-step Nesterov (PLN)

PLN applies a Nesterov wrapper at every local step, evolving worker-local state throughout the block. Its classical local recurrence connects to Nesterov FedAvg and FedNAG (Qu et al., 2023; Yang et al., 2020), while its primal recurrence applies GPA independently on each worker (Defazio et al., 2026). PLN-C/P are equivalent under the appropriate state, scale, and coefficient maps (Sections 2 and 4).

To satisfy the one-vector communication constraint, workers average their terminal model parameters and reconstruct the remaining Nesterov state from the mean block displacement. This boundary preserves the C/P equivalence but generally differs from the cited methods' synchronization policies and from averaging both GPA states. Reconstruction substitutes the mean displacement per step over the block for the mean terminal-step displacement, which can change subsequent updates even at $K = 1$ when $H > 1$. Appendix A.1.1 distinguishes these connections, and Appendix B.5 analyzes the reconstruction error. Table 5 compares the PLN recurrences and their one-vector boundaries with TTN-C, developed in Section 3.4.

In **PLN-C**, each worker initializes its local momentum from the shared state and updates it after every base-optimizer call, unlike RCN-C's fixed round-level reference. In **PLN-P**, each worker updates both the fast and smoothed iterates at every local step. The updated smoothed iterate enters the next interpolation, unlike RCN-P's fixed anchor.

3.4 Two-timescale Nesterov (TTN)

Two-timescale Nesterov (TTN) combines RCN's fixed round-level reference with worker-local Nesterov momentum updated at every step. We use the classical variant, TTN-C, shown in Table 5, which resets local momentum at the start of each round.

Setting $\mu^{\ell} = 0$ recovers tied RCN-C with the same base updates and outer scale. TTN therefore tests whether additional local Nesterov memory improves on round conditioning when the base optimizer already supplies its own memory. Appendix C.2 gives the general template and alternative local-state policies, alongside the independent-coefficient algorithm (Algorithm 10).

4 GENERALIZED NESTEROV WITH INDEPENDENT COEFFICIENTS

GPA separates primal smoothing and interpolation through independent coefficients μ_x and μ_y (Defazio et al., 2026). We apply the same separation to the classical and modern formulations:

$$\textbf{Primal.}\ y_t = \mu_y x_t + (1-\mu_y) z_t, \quad z_{t+1} = z_t + U_t(y_t), \quad x_{t+1} = \mu_x x_t + (1-\mu_x) z_{t+1}. \tag{15}$$

$$\textbf{Classical.}\ y_t = x_t + \mu_y v_{t-1}, \quad v_t = \mu_x v_{t-1} + U_t(y_t), \quad x_{t+1} = x_t + v_t. \tag{16}$$

$$\textbf{Modern.}\ v_t = \mu_x v_{t-1} + U_t(x_t), \qquad x_{t+1} = x_t + U_t(x_t) + \mu_y v_t. \tag{17}$$

Using formulation superscripts where needed, $\mu_x \in [0,1)$ controls primal smoothing or classical/modern momentum retention. Classical extrapolation and modern correction allow $\mu_y^{\mathrm{C}}, \mu_y^{\mathrm{M}} \geq 0$, including values above one, whereas primal interpolation uses $\mu_y^{\mathrm{P}} \in [0,1]$.

Equivalence requires the following coefficient and update-scale maps (derived in Appendix B.2):

$$\mu_x^{\mathrm{C}} = \mu_x^{\mathrm{P}}, \qquad \mu_y^{\mathrm{C}} = \frac{\mu_x^{\mathrm{P}}(1-\mu_y^{\mathrm{P}})}{1-\mu_x^{\mathrm{P}}}, \qquad U_t^{\mathrm{C}} = (1-\mu_x^{\mathrm{P}}) U_t^{\mathrm{P}}, \tag{18}$$

$$\mu_x^{\mathrm{M}} = \mu_x^{\mathrm{P}}, \qquad \mu_y^{\mathrm{M}} = \frac{(1-\mu_x^{\mathrm{P}})\mu_y^{\mathrm{P}}}{1-\mu_y^{\mathrm{P}}},\ (\mu_y^{\mathrm{P}} < 1), \qquad U_t^{\mathrm{M}} = (1-\mu_y^{\mathrm{P}}) U_t^{\mathrm{P}}. \tag{19}$$

For fixed $0 < \mu_x^{\mathrm{P}} < 1$, varying $\mu_y^{\mathrm{P}} \in [0,1)$ covers $0 < \mu_y^{\mathrm{C}} \leq \mu_x^{\mathrm{P}}/(1-\mu_x^{\mathrm{P}})$, while zero retention maps to $\mu_y^{\mathrm{C}} = 0$. Thus primal interpolation covers only part of the classical domain at fixed retention, but can represent strong extrapolation: $(\mu_x^{\mathrm{P}}, \mu_y^{\mathrm{P}}) = (0.99, 0.8)$ maps to $(\mu_x^{\mathrm{C}}, \mu_y^{\mathrm{C}}) = (0.99, 19.8)$, with $U^{\mathrm{C}} = 0.01 U^{\mathrm{P}}$. The modern map covers the full domain $[0,1) \times [0,\infty)$.

5 ANALYSIS

Formulations are equivalent within SHN and PLN. For **SHN**, this follows directly from the single-step equivalences. The required update scaling is absorbed into the outer scale, leaving local base updates unchanged (Appendices B.3 and B.4). For **PLN**, within block equivalence is given from the single-step equivalences, and worker averaging and the one-vector reconstructions can be shown to preserve the same state map at round boundaries (Appendix B.5).

RCN-C versus RCN-P and SHN. In both RCN constructions, the fixed shared Nesterov state determines the local look-ahead points. In RCN-C, it also contributes an explicit displacement between successive look-ahead points. In RCN-P, these increments consist only of scaled base-optimizer updates, so its look-ahead sequence follows an ordinary local block, as in SHN. **RCN-C and RCN-P are therefore generally not equivalent**. For $\mu_y^{\mathrm{P}} < 1$, **RCN-P is equivalent to SHN-P** after jointly adjusting the local-update scale and the outer scale. The formulation maps in Section 4 extend this equivalence to SHN-C/M, but the mapped coefficients need not lie within the same prescribed coefficient ranges, and native reported iterates can differ. Appendices B.4 and B.4 give the local-path comparison, conversions, and proofs.

TTN. TTN reduces to RCN-C at zero local momentum (Section 3.4).

When Nesterov history may help each construction When shared history remains informative throughout a local block, RCN-C may benefit from its repeated use. If it becomes stale, SHN/RCN-P may be preferable, suggesting that larger H can disadvantage RCN-C. PLN and TTN combine shared history with fresh local history, potentially mitigating this staleness, but the local contribution can instead be harmful when noisy or worker-specific. Its value also depends on what the base optimizer already captures: TTN's additive combination of shared and local history may offer little benefit when the base optimizer already makes effective use of local history. If shared history itself is uninformative, plain local steps may suffice. These are intuitions suggested by the update structures, not formal performance guarantees.

Table 6: Language model experiment settings.

SETTING	G	K	H	G/K
S0	128	8	32	16
S1	256	1	32	256
S2	256	4	32	64
S3	256	8	32	32
S4	256	8	128	32
S5	256	8	512	32
S6	512	4	32	128
S7	512	4	128	128
S8	512	8	32	64
S9	512	8	128	64
S10	1024	8	32	128

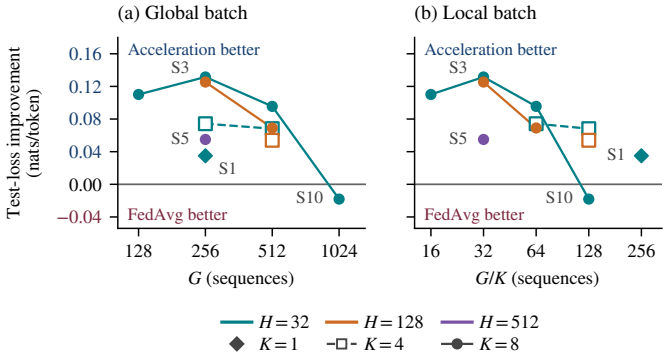


Figure 1: Test-loss improvement over FedAvg from the best accelerated method, using validation-selected recipes.

6 Language-model experiments

Experimental setup. We compare the four Nesterov constructions with FedAvg (McMahan et al., 2017) at 124M parameters, then transfer selected per-method recipes to the 1B scale. Both use decoder implementations inspired by nanoGPT (Karpathy, 2022), with RoPE (Su et al., 2021), RMSNorm (Zhang & Sennrich, 2019), and SwiGLU (Shazeer, 2020). Models are trained on FineWeb (Penedo et al., 2024) with sequence length 1,024 and a single fixed seed. All methods use AdamW (Loshchilov & Hutter, 2019) and a common warmup/cosine schedule shape, with numerical details in Appendix D.

Training scales and methods. The 124M comparison uses approximately 1.074B tokens per trial across eleven settings (Table 6), varying global batch size G (sequences), logical workers K, and local updates per synchronization H. Local batch size is G/K.

The 124M comparison includes nine accelerated variants: tied/generalized SHN-M, tied/generalized RCN-C, generalized RCN-P, tied/generalized PLN-C, generalized PLN-P, and TTN-C tied within each timescale. The 1B experiments extend S9 ($K = 8, G = 512, H = 128$) to 10B and 20B-token horizons, retaining six methods (Table 10). Every method communicates one FP32 model-sized optimization vector per round (Appendix C).

Tuning and evaluation. At 124M, we adaptively search learning rates, Nesterov coefficients, and outer multipliers α, selecting each method–setting recipe by final validation loss. The 1B study inherits each method's selected S9 coefficients and α, adding only a two-rate validation pilot whose selected recipe is used at both horizons without further tuning. Both scales evaluate selected final checkpoints on the same separate 10.49M-token test stream, without reselection. Appendix D gives the search and pilot protocols.

6.1 124M results across training settings

Generalized RCN-P wins seven settings, generalized RCN-C three, and FedAvg one (Table 7). RCN-P also has the lowest mean test loss. Selected coefficients and outer scales are in Table 8, with full recipes in Appendix E.

When is Nesterov useful? FedAvg wins only at the largest tested global batch, $G = 1024$, while Nesterov acceleration improves on FedAvg in all smaller-batch settings (Figure 1). At fixed $G = 256, H = 32$, the best accelerated method's advantage grows with worker count, partially offsetting the deterioration from partitioned training. Conversely, increasing H from 32 to 512 at $G = 256, K = 8$ reduces the acceleration gain from 0.1316 to 0.0550 nats/token.

Generalizing DiLoCo's outer Nesterov. Generalized SHN-M improves on the tuned tied SHN-M (DiLoCo) in nine of eleven settings. Compared with the DiLoCo paper's tied momentum of 0.9 (Douillard et al., 2023), selected recipes combine lower retention, $\mu_x^{\mathrm{M}} \in [0.25, 0.75]$, with strong

Table 7: 124M test loss for validation-selected recipes (nats/token; lower is better). Bold and underline mark the best and second-best losses per column; shading shows the gap to the column minimum using the scale below. Corresponding validation losses are in Table 15.

METHOD	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	AVG
FedAvg	3.8086	3.5286	3.6453	3.8034	3.8946	3.9454	3.7215	3.7602	3.8608	3.9219	**4.0869**	3.8161
SHN-M tied	3.7057	3.5008	3.6073	3.6824	3.9286	3.9221	3.7695	3.7266	3.8949	3.8775	4.1529	3.7971
SHN-M gen.	3.7057	3.5008	3.5830	3.6802	3.7745	3.8922	3.6698	3.7198	3.7941	3.8612	4.1323	3.7558
RCN-C tied	3.7226	3.4998	3.6008	3.7124	3.8779	3.9835	3.6918	3.7773	3.8210	3.9477	4.1884	3.8021
RCN-C gen.	3.7020	3.4969	**3.5711**	3.6737	3.8014	3.9705	**3.6533**	3.7308	**3.7652**	3.8699	4.1277	3.7602
RCN-P gen.	**3.6985**	**3.4936**	3.5785	**3.6719**	**3.7693**	**3.8905**	3.6655	**3.7063**	3.7940	**3.8529**	4.1098	**3.7482**
PLN-C tied	3.8679	3.8152	3.9466	4.1531	4.2944	4.3343	4.3556	4.5154	4.5054	4.7462	5.1865	4.3382
PLN-C gen.	3.8180	3.5075	3.6719	3.8179	3.8835	3.9476	3.7281	3.7682	3.8560	3.9290	4.1050	3.8212
PLN-P gen.	3.7299	3.4972	3.6105	3.7312	3.8840	3.9341	3.7352	3.7603	3.8783	3.9114	4.2010	3.8067
TTN-C	3.7093	3.5465	3.6471	3.6799	3.9377	4.0519	3.7728	3.7260	3.8001	3.8747	4.2642	3.8191

Loss above column best (nats/token): 0 | 0.01 | 0.025 | ≥ 0.05

Table 8: Selected coefficients for Table 7. Entries are $(\alpha; \mu_x, \mu_y)$ for SHN/RCN/PLN and $(\alpha; \mu^{\mathrm{r}}, \mu^{\ell})$ for TTN (round/local, each tied), with $T_n = 0.9^{1/n}$. Bold marks setting winners. Asterisks identify equivalent SHN-M/RCN-P coefficient pairs under Equation (25). Full recipes are in Appendix E.

SETTING	SHN-M gen.	RCN-C gen.	RCN-P gen.	PLN-C gen.	PLN-P gen.	TTN-C
S0	$(0.8; 0.75, 0.75)^*$	$(0.7; 0.75, 1.75)$	$\mathbf{(1; 0.75, 0.75)}^*$	$(1; 0.5, 0.9)$	$(1; T_8, 0.7)$	$(1; 0.75, 0.05)$
S1	$(0.8; 0.5, 0.5)$	$(1; 0.5, 0.9)$	$\mathbf{(0.7; 0.75, 0.75)}$	$(1; 0.5, 0.9)$	$(1; T_8, 0.8)$	$(1; 0.9, 0)$
S2	$(0.8; 0.5, 1)^*$	$\mathbf{(1; 0.75, 0.9)}$	$(0.8; 0.5, 2/3)^*$	$(1; 0.5, 0.9)$	$(1; T_8, 0.8)$	$(1; 0.9, 0)$
S3	$(0.8; 0.75, 1)$	$(1; 0.75, 0.9)$	$\mathbf{(1; 0.75, 0.75)}$	$(1; 0.5, 0.9)$	$(1; T_8, 0.8)$	$(1; 0.75, 0.1)$
S4	$(0.8; 0.5, 1.75)^*$	$(1; 0.5, 0.9)$	$\mathbf{(0.8; 0.5, 7/9)}^*$	$(1; 0.5, 1.5)$	$(1; T_{32}, 0.8)$	$(1; 0.9, 0)$
S5	$(0.8; 0.25, 1)^*$	$(0.7; 0.5, 1)$	$\mathbf{(0.8; 0.25, 4/7)}^*$	$(1; 0.5, 0.9)$	$(1; T_8, 0.5)$	$(1; 0.9, 0)$
S6	$(0.8; 0.5, 0.9)^*$	$\mathbf{(1; 0.5, 0.9)}$	$(0.8; 0.5, 9/14)^*$	$(1; 0.5, 0.9)$	$(1; T_8, 0.7)$	$(1; 0.9, 0)$
S7	$(0.8; 0.25, 0.9)^*$	$(1; 0.5, 0.9)$	$\mathbf{(0.8; 0.25, 6/11)}^*$	$(1; 0.5, 0.9)$	$(1; T_8, 0.5)$	$(1; 0.5, 0)$
S8	$(0.7; 0.5, 1.25)^*$	$\mathbf{(1; 0.5, 0.9)}$	$(0.8; 0.5, 5/7)^*$	$(1; 0.5, 1.5)$	$(1; T_8, 0.7)$	$(1; 0.75, 0)$
S9	$(0.8; 0.5, 0.9)^*$	$(1; 0.5, 0.9)$	$\mathbf{(0.8; 0.5, 9/14)}^*$	$(1; 0.5, 0.9)$	$(1; T_8, 0.6)$	$(1; 0.5, 0)$
S10	$(0.7; 0.75, 0.9)$	$(0.7; 0.5, 0.75)$	$(0.7; 0.75, 0.75)$	$(1; 0.5, 0.9)$	$(1; T_4, 0.5)$	$(1; 0.9, 0)$

correction, $\mu_y^{\mathrm{M}} \geq 0.9$ in nine settings and above one in two, suggesting a trade-off hidden by tied coefficients. Nine selections also use $\alpha = 0.8$ rather than DiLoCo's 0.7, consistent with theoretical motivation for exploring larger outer rates (Khaled et al., 2025).

SHN-M versus RCN-P: reporting and numerical effects. Generalized SHN-M and RCN-P are equivalent under the maps in Section 5, and in eight settings they chose equivalent coefficients, yet RCN-P has better test loss. SHN-M returning look-ahead points rather than the smoothed iterate explains much of this gap: across 21 matched comparisons, evaluating SHN-M's smoothed iterate reduces the setting-averaged validation gap from 0.01928 to 0.00768 (Table 9). We provide evidence that finite-precision errors contribute to the remaining differences by causing the trajectories to drift apart (Appendix D.5.3), but we are unable to establish why residual gaps generally favor RCN-P.

When is round-conditioned classical Nesterov useful? RCN-C is most competitive with low H, winning S2/S6/S8, all at $H = 32$. At $G = 512$, increasing H to 128 shifts the winner to RCN-P, while RCN-C underperforms FedAvg in the $H = 512$ setting. All three winning recipes combine lower retention, $\mu_x \in \{0.5, 0.75\}$, with extrapolation $\mu_y = 0.9$ and outer scale $\alpha = 1$. The results show that the benefit of stronger extrapolation depends on the outer scale, motivating joint tuning (Appendix D.4).

Does evolving additional local Nesterov state help? Neither PLN nor TTN wins a setting, and TTN selects zero additional local momentum in nine settings, reducing it to RCN-C. Thus, the evidence favors tuning shared momentum over adding another local momentum timescale.

6.2 Transfer to a 1B model

Acceleration and coefficient generalization transfer to the 1B model: all five accelerated methods outperform FedAvg at both horizons, and generalized SHN-M improves on its tied counterpart (Table 10). RCN-P performs best among native outputs, while RCN-C overtakes native generalized

Table 9: Smoothed reporting for 21 matched generalized SHN-M/RCN-P pairs. Gaps are SHN-M minus RCN-P smoothed-iterate validation loss (nats/token), using SHN-M's look-ahead (native) or smoothed (remaining) iterate; smoothing gain is their difference. Means average within settings, then equally across seven settings; the range spans individual pairs. Per-setting results: Table 17.

Settings	Pairs	Native gap	SHN smoothing gain	Remaining gap	Remaining-gap range
All 7 settings	21/21	$+0.01928$	$+0.01160$	$+0.00768$	$[-0.05557, +0.15664]$

Table 10: 1B-model validation and test loss at 10B and 20B training tokens (nats/token). Recipe notation follows Table 8. r is the searched rate and η the AdamW rate: $\eta = r/(1-\mu_y)$ for RCN-P and $\eta = r$ otherwise. † marks RCN-P coefficients equivalent to SHN-M's $(0.5, 0.9)$. Indented rows evaluate the smoothed iterate from the same SHN-M checkpoints without retraining; dashes indicate unevaluated validation losses. Bold/underline mark the best/second-best scores, including smoothed outputs; shading uses the scale below.

METHOD	RECIPE	r	η	10B horizon VAL.	10B horizon TEST	20B horizon VAL.	20B horizon TEST
FedAvg	$(1; –, –)$	0.0008	0.0008	3.03024	3.05822	2.89716	2.92481
SHN-M tied	$(0.8; 0.5, 0.5)$	0.0008	0.0008	2.97915	3.00659	2.85849	2.88664
smoothed output	$(0.8; 0.5, 0.5)$	0.0008	0.0008	—	3.00427	—	2.88327
SHN-M gen.	$(0.8; 0.5, 0.9)$	0.0008	0.0008	<u>2.95083</u>	2.97829	2.84341	2.87106
smoothed output	$(0.8; 0.5, 0.9)$	0.0008	0.0008	—	<u>2.97450</u>	—	**2.86568**
RCN-C gen.	$(1; 0.5, 0.9)$	0.0008	0.0008	2.95611	2.98340	<u>2.84263</u>	2.87020
RCN-P gen.	$(0.8; 0.5, 9/14)^{\dagger}$	0.0008	0.00224	**2.94625**	**2.97386**	**2.83778**	<u>2.86594</u>
PLN-P gen.	$(1; T_8, 0.6)$	0.00125	0.00125	2.97293	3.00056	2.85628	2.88395

Loss above column best (nats/token): 0 | 0.01 | 0.02 | ≥ 0.03

SHN-M at 20B. Smoothed reporting improves both SHN-M variants without retraining or reselection, leaving generalized SHN-M slightly behind RCN-P at 10B and slightly ahead at 20B.

7 Conclusion

We presented four ways to combine Nesterov acceleration with multi-worker local updates under a common one-vector communication constraint, and generalized their coefficients to allow independent momentum retention and extrapolation. Our analysis identifies when these constructions are equivalent under coefficient, update-scale, and state mappings, and when their use of shared history or boundary rules produces different update paths.

Our language-model experiments identify two ways to improve DiLoCo-style training: generalizing momentum retention and extrapolation improves on tied coefficients in nine of eleven 124M settings and at both 1B training horizons, while reporting the smoothed iterate improves loss without retraining in most tested comparisons. RCN-C is particularly competitive with frequent synchronization, whereas the tested PLN and TTN variants provide less evidence for adding another local momentum timescale on top of AdamW. RCN-P generally outperforms the equivalent DiLoCo-style wrapper, but matching their reported iterates substantially reduces the gap and slightly reverses it in the 1B-model, 20B-token comparison. Precision experiments further show how floating-point perturbations can separate algebraically equivalent trajectories. Whether the primal recurrence offers a reproducible numerical-stability advantage remains an open question.

Limitations Our experiments cover one architecture family, FineWeb, and AdamW, with one seed per configuration. Small differences and rankings may change across seeds or workloads and do not establish intrinsic formulation superiority. Although the hyperparameter search is extensive, its adaptive and unequal coverage does not establish each method's optimum or rankings under equal tuning budgets. The 1B study tests recipe transfer in one setting rather than independently optimized methods across scales.

8 AI USE STATEMENT

Generative AI tools assisted with drafting and editing portions of this manuscript. We reviewed all AI-assisted text and take full responsibility for the final content.

REFERENCES

Atish Agarwala. High dimensional theory of two-phase optimizers, 2026. URL https://arxiv.org/abs/2603.26954.

Zeyuan Allen-Zhu and Lorenzo Orecchia. Linear coupling: An ultimate unification of gradient and mirror descent. In *Innovations in Theoretical Computer Science Conference*, volume 67, pp. 3:1–3:22, 2017.

Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8624–8628, 2013. doi: 10.1109/ICASSP.2013.6639349. URL https://arxiv.org/abs/1212.0901. Preprint first posted in December 2012.

Zachary Charles, Gabriel Teston, Lucio Dery, Keith Rush, Nova Fallen, Zachary Garrett, Arthur Szlam, and Arthur Douillard. Communication-efficient language model training scales reliably and robustly: Scaling laws for DiLoCo. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://arxiv.org/abs/2503.09799.

Ashok Cutkosky. Anytime online-to-batch, optimism and acceleration. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1446–1454, 2019. URL https://proceedings.mlr.press/v97/cutkosky19a.html.

Aaron Defazio. On the curved geometry of accelerated optimization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Aaron Defazio. Momentum via primal averaging: Theoretical insights and learning rate schedules for non-convex optimization, 2020. URL https://arxiv.org/abs/2010.00406.

Aaron Defazio. ScheduleFree+: Scaling learning-rate-free & schedule-free learning to large language models, 2026. URL https://arxiv.org/abs/2605.19095.

Aaron Defazio, Xingyu Yang, Harsh Mehta, Konstantin Mishchenko, Ahmed Khaled, and Ashok Cutkosky. The road less scheduled, 2024. URL https://arxiv.org/abs/2405.15682.

Aaron Defazio, Konstantin Mishchenko, Parameswaran Raman, Hao-Jun Michael Shi, and Lin Xiao. Smoothing DiLoCo with primal averaging for faster training of LLMs, 2026. URL https://arxiv.org/abs/2512.17131v3. Version 3, February 27, 2026. First posted in December 2025.

Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. DiLoCo: Distributed low-communication training of language models, 2023. URL https://arxiv.org/abs/2311.08105.

Arthur Douillard, Yanislav Donchev, Keith Rush, Satyen Kale, Zachary Charles, Zachary Garrett, Gabriel Teston, Dave Lacey, Ross McIlroy, Jiajun Shen, Alexandre Ramé, Arthur Szlam, Marc'Aurelio Ranzato, and Paul Barham. Streaming DiLoCo with overlapping communication: Towards a distributed free lunch, 2025. URL https://arxiv.org/abs/2501.18512.

Alex Iacob, Andrej Jovanovic, Mher Safaryan, Meghdad Kurmanji, Lorenzo Sani, Samuel Horváth, William F. Shen, Xinchi Qiu, and Nicholas D. Lane. MT-DAO: Multi-timescale distributed adaptive optimizers with local updates, 2025. URL https://arxiv.org/abs/2510.05361.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2018. URL https://arxiv.org/abs/1803.05407.

Sami Jaghouar, Jack Min Ong, and Johannes Hagemann. OpenDiLoCo: An open-source framework for globally distributed low-communication training, 2024. URL https://arxiv.org/abs/2407.07852.

Pooria Joulani, Anant Raj, András György, and Csaba Szepesvári. A simpler approach to accelerated optimization: Iterative averaging meets optimism. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4984–4993, 2020. URL https://proceedings.mlr.press/v119/joulani20a.html.

Dominik Kallusky, Vinay Rao, Vishal Nandavanam, and Hao-Jun Michael Shi. SNOO: Step-k nesterov outer optimizer—the surprising effectiveness of nesterov momentum applied to pseudo-gradients, 2025. URL https://arxiv.org/abs/2510.15830.

Andrej Karpathy. nanoGPT, 2022. URL https://github.com/karpathy/nanoGPT. Software repository. Accessed September 25, 2026.

Ahmed Khaled, Satyen Kale, Arthur Douillard, Chi Jin, Rob Fergus, and Manzil Zaheer. Understanding outer optimizers in local SGD: Learning rates, momentum, and acceleration, 2025. URL https://arxiv.org/abs/2509.10439.

Guanghui Lan. An optimal method for stochastic composite optimization. *Mathematical Programming*, 133:365–397, 2012. doi: 10.1007/s10107-010-0434-y. URL https://doi.org/10.1007/s10107-010-0434-y.

Lasagne Developers. Lasagne 0.1 documentation: lasagne.updates.nesterov_momentum, n.d. URL https://lasagne.readthedocs.io/en/stable/modules/updates.html. Accessed September 17, 2026.

Bo Liu, Rachita Chhaparia, Arthur Douillard, Satyen Kale, Andrei A. Rusu, Jiajun Shen, Arthur Szlam, and Marc'Aurelio Ranzato. Asynchronous local-SGD training for language modeling, 2024. URL https://arxiv.org/abs/2401.09135.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*, pp. 1273–1282, 2017.

Yurii Nesterov. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Doklady Akademii Nauk SSSR*, 269(3):543–547, 1983. URL https://www.mathnet.ru/eng/dan46009.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992. doi: 10.1137/0330046. URL https://doi.org/10.1137/0330046.

PyTorch Contributors. PyTorch torch.optim.SGD: Implementation and documentation, version 2.5.1, 2024. URL https://github.com/pytorch/pytorch/blob/v2.5.1/torch/optim/sgd.py. Accessed September 17, 2026.

Zhaonan Qu, Kaixiang Lin, Zhaojian Li, Jiayu Zhou, and Zhengyuan Zhou. A unified linear speedup analysis of federated averaging and Nesterov FedAvg. *Journal of Artificial Intelligence Research*, 78:1143–1200, 2023. URL https://arxiv.org/abs/2007.05690v4.

Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021.

David Ruppert. Efficient estimators from a slowly convergent Robbins–Monro process. Technical Report 781, School of Operations Research and Industrial Engineering, Cornell University, 1988. URL https://ecommons.cornell.edu/bitstream/handle/1813/8664/TR000781.pdf.

Noam Shazeer. GLU variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Sebastian U. Stich. Local SGD converges fast and communicates little. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1805.09767.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding, 2021. URL https://arxiv.org/abs/2104.09864v1.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pp. 1139–1147, 2013. URL https://proceedings.mlr.press/v28/sutskever13.html.

TensorFlow Developers. TensorFlow api documentation: tf.keras.optimizers.SGD, n.d. URL https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/SGD. Accessed September 17, 2026.

Jianyu Wang and Gauri Joshi. Cooperative SGD: A unified framework for the design and analysis of local-update SGD algorithms. *Journal of Machine Learning Research*, 22(213):1–50, 2021. URL https://jmlr.org/papers/v22/20-147.html.

Jianyu Wang, Vinayak Tantia, Nicolas Ballas, and Michael Rabbat. SlowMo: Improving communication-efficient distributed SGD with slow momentum. In *International Conference on Learning Representations*, 2020.

Zhengjie Yang, Wei Bao, Dong Yuan, Nguyen H. Tran, and Albert Y. Zomaya. Federated learning with nesterov accelerated gradient, 2020. URL https://arxiv.org/abs/2009.08716.

Honglin Yuan and Tengyu Ma. Federated accelerated stochastic gradient descent. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://arxiv.org/abs/2006.08950.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://arxiv.org/abs/1910.07467.

Michael R. Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

A Related work and additional background

A.1 Related work

Local steps and outer optimization. SGD with multiple local updates between parameter synchronizations is commonly called *local SGD*. Related terminology includes *local-update SGD*, *periodic model averaging*, and *parallel restarted SGD* (Stich, 2019; Wang & Joshi, 2021). FedAvg uses local SGD and server-side averaging, while FedOpt separates the client optimizer from an adaptive server optimizer (McMahan et al., 2017; Reddi et al., 2021). SlowMo similarly applies momentum after blocks of a base algorithm (Wang et al., 2020). We study fixed-replica language-model training, where workers can drift apart even when sampling a common distribution.

Outer Nesterov in language-model training. DiLoCo combines local AdamW with outer Nesterov. Subsequent scaling work studies its dependence on model size, replica count, hyperparameters, and token budget (Douillard et al., 2023; Charles et al., 2025). OpenDiLoCo develops a geographically distributed implementation, Streaming DiLoCo overlaps communication with computation, and asynchronous local SGD uses delayed Nesterov to account for stale worker updates (Jaghouar et al., 2024; Douillard et al., 2025; Liu et al., 2024). Our comparison holds the optimization payload fixed while varying how Nesterov acts within each local block.

Single-worker blocks and per-step alternatives. Beyond the single-worker evidence reviewed in the introduction, SNOO connects block-level Nesterov to Lookahead and proposes an implicit-regularization interpretation (Zhang et al., 2019; Kallusky et al., 2025). GPA's per-step averaging with independent coefficients is a smoothed alternative to single-worker DiLoCo, not an exact trajectory-preserving rewrite (Defazio et al., 2026). Complementary two-phase theory studies how multiple workers change the noise seen by an outer phase (Agarwala, 2026).

Iterate averaging and base-optimizer wrappers. SWA averages training iterates without feeding that average into subsequent updates, whereas primal-averaging representations of momentum expose a coupling between an averaged iterate and future gradient locations (Izmailov et al., 2018; Defazio, 2020). Schedule-Free separates the interpolation used for gradient evaluation from the weights used for iterate averaging. GPA uses independently chosen constant interpolation and exponential averaging weights (Defazio et al., 2024; 2026). ScheduleFree+ further studies retaining base-optimizer momentum alongside the averaging wrapper for large-batch language-model training (Defazio, 2026). Online-to-batch analyses provide theoretical foundations for averaging wrappers under specific oracle and objective assumptions (Cutkosky, 2019; Joulani et al., 2020). Appendix A.2 discusses their scope for local-block updates. Section 2 reviews the established Nesterov formulations and their attribution. Section 4 develops independent coefficients for our constructions.

Acceleration within distributed local updates. Nesterov FedAvg has direct convergence analyses, and FedAc combines worker-local accelerated updates with periodic aggregation for convex objectives (Qu et al., 2023; Yuan & Ma, 2020). Recent local-SGD theory analyzes outer learning rates and Nesterov acceleration, including regimes where outer acceleration improves communication-round dependence relative to local acceleration (Khaled et al., 2025). MT-DAO is a related adaptive method that uses multiple timescales to control drift (Iacob et al., 2025). Our focus is how the specified Nesterov constructions, formulations, and state policies interact under one-vector synchronization, and which resulting variants help in language-model training.

A.1.1 Local Nesterov, GPA, and synchronization

The PLN row of Table 3 relates methods by their local recurrences and boundary synchronization policies. We use the matching assumptions in Appendix B.1.

Nesterov FedAvg. In Qu et al. (2023, Section 4.1), $v_{t+1}^k = w_t^k - \alpha_t g_{t,k}$ is the post-gradient iterate, and $w_{t+1}^k = v_{t+1}^k + \beta_t(v_{t+1}^k - v_t^k)$ is the next look-ahead point between synchronizations. For constant $\beta_t = \mu$, these are the canonical local Nesterov updates, with (v_t^k, w_t^k) corresponding to our classical $(x_{k,t}, y_{k,t})$. Their boundary averages the extrapolated w candidates but leaves the auxiliary v sequence worker-local. PLN-C instead averages the reported x endpoints and reconstructs a shared momentum state. Both methods communicate one model-sized vector, but synchronize different states. Their varying-coefficient schedules also lie outside this constant-coefficient identification.

FedNAG. Yang et al. (2020, Algorithm 1 and equations (4)–(7)) use the modern local recurrence and average both model and momentum states, with weights proportional to workers' data sizes. Writing their model and momentum as (w, u), the map $x = w - \mu u$ converts the local recurrence to the tied classical formulation. Because this map is linear, averaging (w, u) is equivalent to averaging (x, u) with the same weights. With uniform weights and matched base updates, FedNAG's training recurrence therefore corresponds to the complete-state classical reference in Equation (79), not projected PLN-C. Its native model is the modern look-ahead iterate w, rather than the classical reported iterate x.

GPA and the C/P connection. GPA applies generalized primal Nesterov at each base-optimizer step in a non-distributed setting (Defazio et al., 2026). A direct multi-worker extension averages both terminal states (x, z). For tied coefficients, the local map $z = x + \mu u/(1-\mu)$ and update conversion $U^{\mathrm{C}} = (1-\mu)U^{\mathrm{P}}$ identify this extension with complete-state classical averaging. For independent coefficients, use Equation (18) instead of equating the numerical coefficient pairs. The same maps also identify projected PLN-C with projected PLN-P, since reconstructing u from the mean block displacement and then reconstructing z preserves the state relation. The classical/primal

correspondence holds separately under each boundary policy. Appendix B.5 quantifies their difference and shows why projection can change even a single-worker trajectory.

A.2 Connections to averaging and acceleration

Geometric and linear-coupling views give complementary interpretations of Nesterov's alternative formulations (Allen-Zhu & Orecchia, 2017; Defazio, 2019). Local-step extensions additionally require a choice of states to use within each block and synchronize at its boundary.

Scope of averaging-wrapper guarantees. Online-to-batch analyses establish accelerated guarantees for suitable averaging wrappers under their oracle and objective assumptions (Cutkosky, 2019; Joulani et al., 2020). Those guarantees do not automatically extend to an arbitrary local-block displacement, which need not be an unbiased gradient at the block start.

B Equivalence proofs and boundary analysis

B.1 Definitions and matching assumptions

An equivalence below means that corresponding base updates are evaluated at identical parameter vectors and that the stated maps between recurrent states are preserved. Equality of native reported parameters is a separate claim: modern Nesterov reports a look-ahead point, whereas classical and primal Nesterov report the corresponding smoothed iterate. All identities assume exact arithmetic, constant Nesterov coefficients, compatible initial Nesterov states, identical stochastic inputs, and matched base-optimizer states and state transitions. Update-scale conversions apply to the complete displacement, including weight decay. Changing the learning rate implements such a conversion only when it scales that complete displacement without changing the required optimizer-state transitions. Workers may retain different private base-optimizer states. Corresponding workers must have matched states across the algorithms being compared. The claims are algebraic identities, not convergence or generalization guarantees. Table 11 summarizes the construction comparisons proved below.

B.2 Single-step formulation equivalence

We derive the generalized maps in Section 4 under the assumptions of Appendix B.1, then recover tied Nesterov as a special case. All explicit Nesterov momentum states use parameter-displacement units, whereas PyTorch SGD stores a gradient-space buffer and applies the learning rate afterward (PyTorch Contributors, 2024). Comparisons with gradient-buffer implementations require matching these conventions, particularly when the learning rate varies. The equivalences below assume constant Nesterov coefficients and do not cover simply substituting time-varying coefficients into the recurrences.

B.2.1 Classical and primal recurrences

The primal reporting recurrence depends only on μ_x. For $0 \le \mu_x < 1$, common initialization $z_0 = x_0$ and the reporting recurrence give (setting $x_{-1} = x_0$ at initialization)

$$x_t^{\mathrm{P}} = \mu_x^t x_0 + (1-\mu_x)\sum_{j=1}^{t} \mu_x^{t-j} z_j, \tag{20}$$

$$z_t = \frac{x_t^{\mathrm{P}} - \mu_x x_{t-1}^{\mathrm{P}}}{1-\mu_x} = x_t^{\mathrm{P}} + \frac{\mu_x}{1-\mu_x}(x_t^{\mathrm{P}} - x_{t-1}^{\mathrm{P}}). \tag{21}$$

Writing $v_{t-1}^{\mathrm{P}} = x_t^{\mathrm{P}} - x_{t-1}^{\mathrm{P}}$ and substituting Equation (21) into the primal interpolation gives

$$y_t^{\mathrm{P}} = x_t^{\mathrm{P}} + \frac{\mu_x^{\mathrm{P}}(1-\mu_y^{\mathrm{P}})}{1-\mu_x^{\mathrm{P}}} v_{t-1}^{\mathrm{P}}. \tag{22}$$

Increasing μ_y^{P} at fixed $0 < \mu_x^{\mathrm{P}} < 1$ reduces effective extrapolation, whereas increasing μ_y^{C} increases it. At fixed retention, primal interpolation covers only the classical look-ahead range stated in Section 4, but its effective look-ahead is unbounded as μ_x^{P} approaches one with $\mu_y^{\mathrm{P}} < 1$ fixed. A large μ_y^{C}

Table 11: Equivalence results under the matching assumptions in Appendix B.1.

COMPARISON	RESULT AND QUALIFICATION
Tied SHN-C/M/P	One block-level recurrence under outer scale and state conversion, with different modern stored and classical reported outputs (Proposition 2).
Generalized SHN-C/M/P	Equivalent on corresponding coefficient domains after state and outer-scale conversion, not generally at equal coefficient pairs (Equations (19) and (50)).
Generalized RCN-P and SHN-P	Identical local evaluation points and both primal boundary states for $\mu_y^{\mathrm{P}} < 1$, at the same coefficient pair after local-update and outer-scale conversion (Equation (63)).
Tied RCN-P and SHN-C	Identical local evaluation points and reported boundaries after local-update scaling, at equal outer scale (Proposition 4).
Generalized RCN-P and SHN-C	A block-wrapper representation remains possible after remapping coefficients and both local and outer scales (Equation (67)).
Generalized SHN-M and RCN-P	Identical local evaluation points after coefficient and local-scale conversion, at equal outer scale. Native reported outputs differ by a momentum correction (Equations (25) and (71)).
RCN-C and RCN-P	Generally different local paths for $H > 1$ because only RCN-C repeatedly injects the reference. Tied, correctly scaled versions agree at $H = 1$ (Appendix B.4).
Tied projected PLN-C/P	Equivalent after local scale conversion and the matched one-vector projection, but not generally equal to unprojected continuation for $H > 1$ (Section 3.3).
Generalized projected PLN-C/P	Equivalent local paths and reported boundaries after coefficient and local-update conversion with compatible projection. Using the same unequal coefficient pair generally gives different methods (Appendix B.5).
Modern storage of RCN-C	A boundary-state representation, not an additional native within-block trajectory (Appendix B.4).
TTN and RCN-C	TTN reduces to RCN-C at zero local momentum. The preceding maps do not establish a general TTN equivalence to SHN or PLN (Section 3.4).

alone establishes neither a larger net update nor instability, since equivalence also requires the state map and update scaling. The state map is $x_t^{\mathrm{C}} = x_t^{\mathrm{P}}$ and $z_t = x_t^{\mathrm{C}} + \mu_x^{\mathrm{P}} v_{t-1}^{\mathrm{C}}/(1-\mu_x^{\mathrm{P}})$. Common initialization with zero momentum satisfies it.

The classical look-ahead coefficient in Equation (16) is μ_y^{C}. If $\mu_x^{\mathrm{C}} = \mu_x^{\mathrm{P}} = \mu_x$ and $\mu_y^{\mathrm{C}} = \mu_y^{\mathrm{P}} = \mu_y$, then under the reported-displacement state map, the classical and primal look-ahead points agree for arbitrary incoming displacement exactly when

$$\frac{\mu_x(1-\mu_y)}{1-\mu_x} = \mu_y \quad \Longleftrightarrow \quad \boxed{\mu_x = \mu_y}. \tag{23}$$

Thus using the same pair of unequal coefficients generally defines different classical and primal trajectories. To derive the coefficient remapping, write the primal reporting identity as

$$v_t^{\mathrm{P}} = \mu_x v_{t-1}^{\mathrm{P}} + (1-\mu_x) U_t^{\mathrm{P}}(y_t^{\mathrm{P}}). \tag{24}$$

Together, Equations (22) and (24) yield the coefficient and displacement map in Equation (18). With compatible initialization and matched base-optimizer state transitions at common stochastic query points, induction preserves the state map and reported iterates. The scaling applies to the complete optimizer displacement, including weight decay.

Single-step modern conversion and full state map. Inverting Equation (19), modern coefficients $0 \le \mu_x^{\mathrm{M}} < 1$, $\mu_y^{\mathrm{M}} \ge 0$ map to

$$\mu_x^{\mathrm{P}} = \mu_x^{\mathrm{M}}, \qquad \mu_y^{\mathrm{P}} = \frac{\mu_y^{\mathrm{M}}}{1+\mu_y^{\mathrm{M}} - \mu_x^{\mathrm{M}}}. \tag{25}$$

The associated single-step states satisfy

$$x_t^{\mathrm{C}} = x_t^{\mathrm{P}}, \qquad z_t = x_t^{\mathrm{C}} + \frac{\mu_x^{\mathrm{P}}}{1-\mu_x^{\mathrm{P}}} v_{t-1}^{\mathrm{C}}, \tag{26}$$

$$v_{t-1}^{\mathrm{M}} = \frac{1-\mu_y^{\mathrm{P}}}{1-\mu_x^{\mathrm{P}}} v_{t-1}^{\mathrm{C}}, \qquad x_t^{\mathrm{M}} = y_t^{\mathrm{P}} = x_t^{\mathrm{C}} + \mu_x^{\mathrm{P}} v_{t-1}^{\mathrm{M}}. \tag{27}$$

For $0 \le \mu_y^{\mathrm{P}} < 1$, scaling Equation (24) gives

$$v_t^{\mathrm{M}} = \mu_x^{\mathrm{P}} v_{t-1}^{\mathrm{M}} + (1-\mu_y^{\mathrm{P}}) U_t^{\mathrm{P}}, \tag{28}$$

$$x_{t+1}^{\mathrm{M}} = x_t^{\mathrm{M}} + (1-\mu_y^{\mathrm{P}}) U_t^{\mathrm{P}} + \frac{(1-\mu_x^{\mathrm{P}})\mu_y^{\mathrm{P}}}{1-\mu_y^{\mathrm{P}}} v_t^{\mathrm{M}}. \tag{29}$$

These are the generalized modern recurrence and the scale map in Equation (19), proving preservation of the state correspondence. The classical/primal map also admits $\mu_y^{\mathrm{P}} = 1$, giving zero classical extrapolation. The modern conversion is singular there and does not apply. Classical and primal report the same x_t, whereas modern reports their look-ahead point unless transformed back. At $\mu_x^{\mathrm{P}} = \mu_y^{\mathrm{P}} = \mu$, all three coefficient pairs equal (μ, μ) and the tied scale conversion is recovered.

Eliminating the fast iterate. For $\mu_y < 1$, eliminating z_t from the primal recurrence gives

$$x_{t+1}^{\mathrm{P}} = \frac{\mu_x - \mu_y}{1-\mu_y} x_t^{\mathrm{P}} + \frac{1-\mu_x}{1-\mu_y} y_t^{\mathrm{P}} + (1-\mu_x) U_t^{\mathrm{P}}(y_t^{\mathrm{P}}) \tag{30}$$

$$= x_t^{\mathrm{P}} + \underbrace{\frac{1-\mu_x}{1-\mu_y}}_{a}(y_t^{\mathrm{P}} - x_t^{\mathrm{P}}) + \underbrace{(1-\mu_x)}_{b} U_t^{\mathrm{P}}(y_t^{\mathrm{P}}). \tag{31}$$

The tied choice $\mu_x = \mu_y = \mu$ sets $a = 1$ and recovers $x_{t+1} = y_t + (1-\mu)U_t(y_t)$. When $\mu_x \neq \mu_y$, a changes how far the reported sequence moves toward the look-ahead point, while b controls how much of the new fast update is reported.

B.2.2 Tied coefficients and canonical Nesterov

We derive the standard equivalence in the displacement-scaled notation of Section 2 (Bengio et al., 2013; Defazio, 2019; Defazio et al., 2026). The classical-to-modern and classical-to-primal state transformations are

$$x_t^{\mathrm{M}} = y_t^{\mathrm{C}} = x_t^{\mathrm{C}} + \mu v_{t-1}, \qquad v_t^{\mathrm{M}} = v_t^{\mathrm{C}}, \tag{32}$$

$$x_t^{\mathrm{P}} = x_t^{\mathrm{C}}, \qquad z_t = x_t^{\mathrm{C}} + \frac{\mu}{1-\mu} v_{t-1}, \qquad y_t^{\mathrm{P}} = y_t^{\mathrm{C}}. \tag{33}$$

At matched evaluation points and compatible base-optimizer states, the complete displacements must satisfy

$$U_t^{\mathrm{C}} = U_t^{\mathrm{M}} = (1-\mu) U_t^{\mathrm{P}}. \tag{34}$$

When the learning rate scales the complete base-optimizer displacement without changing the required state transitions, this conversion is achieved by $\gamma_{\mathrm{C}} = \gamma_{\mathrm{M}} = (1-\mu)\gamma_{\mathrm{P}}$. Scaling only the gradient-dependent component is insufficient when other terms, such as weight decay, contribute to the displacement.

The shared momentum state is not the modern parameter increment:

$$x_{t+1}^{\mathrm{M}} - x_t^{\mathrm{M}} = U_t^{\mathrm{M}}(x_t^{\mathrm{M}}) + \mu v_t = (1+\mu) v_t - \mu v_{t-1}. \tag{35}$$

The classical-output reconstruction therefore uses the stored momentum state, not simply the preceding modern parameter increment.

Proposition 1 (One-step equivalence)**.** *Under the initialization in Table 1 and the scale map in Equation* (34)*, the canonical recurrence and the three recurrences in Table 1 evaluate every stochastic update at the same point when supplied with the same stochastic samples. They satisfy* $x_t = x_t^{\mathrm{P}} = x_t^{\mathrm{C}}$, $y_t = x_t^{\mathrm{M}} = y_t^{\mathrm{C}} = y_t^{\mathrm{P}}$*, and* $v_t^{\mathrm{M}} = v_t^{\mathrm{C}} = x_{t+1} - x_t$ *for every* t*.*

Proof. Setting $\mu_x^{\mathrm{P}} = \mu_y^{\mathrm{P}} = \mu$ in the generalized maps above gives Equations (32) to (34). Zero momentum and $z_0 = x_0$ satisfy these maps initially, so their preservation gives identical query points at every step. The canonical recurrence is the classical recurrence after substituting $v_{t-1} = x_t - x_{t-1}$, and the primal reporting map preserves $x_t^{\mathrm{P}} = x_t^{\mathrm{C}}$. □

Attribution. These single-step formulations and their equivalences are established background. The classical look-ahead recurrence is given by Sutskever et al. (2013, Section 2, Eqs. (3)–(4)). The modern change of variables is derived by Bengio et al. (2013, Section 3.5, Eqs. (6)–(7)) and used in Lasagne and TensorFlow/Keras (Lasagne Developers, n.d.; TensorFlow Developers, n.d.). The primal three-sequence structure appears in Lan (2012, Section 3.1, Eqs. (28)–(29)). Defazio et al. (2026, Section 2.1, Eq. (4), and Appendix D) state the constant-coefficient recurrence and develop formulation equivalences. The two-sequence heavy-ball representation in Defazio (2020) is distinct from the Nesterov recurrence used here.

B.3 SHN FORMULATIONS, EQUIVALENCE, AND ONE-VECTOR IMPLEMENTATION

For comparison, we collect all three tied SHN formulations, including the modern recurrence used in the main text. Each uses the same local-block interface from Equation (9), with $0 \le \mu < 1$ and a common initial parameter x_0. For a common block start a, let $\Delta_{k,r}(a) = \theta_{k,r,H} - a$ be worker k's endpoint displacement when $\theta_{k,r,0} = a$, and write $\overline{\Delta}_r(a) = K^{-1} \sum_k \Delta_{k,r}(a)$. The argument specifies the block's starting parameters, with incoming base-optimizer states, stochastic inputs, and schedules implicit. This notation allows the classical and primal blocks to start at their look-ahead point y_r, rather than the reported iterate x_r.

Classical outer update. Step-H outer Nesterov (SHN) replaces the one-step displacement in the canonical recurrence by the mean displacement of a complete local block,

$$y_r = x_r + \mu(x_r - x_{r-1}), \tag{36}$$

$$x_{r+1} = y_r + \alpha_r \overline{\Delta}_r(y_r). \tag{37}$$

Initialize $x_{-1} = x_0$ in the canonical recurrence. Here y_r is the common worker initialization, and for $\alpha_r = 1$ the next reported iterate is simply the mean local endpoint. Setting $v_{r-1} = x_r - x_{r-1}$ gives

the classical outer recurrence

$$y_r^{\mathrm{C}} = x_r^{\mathrm{C}} + \mu v_{r-1}, \tag{38}$$

$$\Delta_{k,r}^{\mathrm{C}} = \Delta_{k,r}(y_r^{\mathrm{C}}), \qquad \overline{\Delta}_r^{\mathrm{C}} = \frac{1}{K}\sum_k \Delta_{k,r}^{\mathrm{C}}, \tag{39}$$

$$v_r = \mu v_{r-1} + \alpha_r^{\mathrm{C}} \overline{\Delta}_r^{\mathrm{C}}, \tag{40}$$

$$x_{r+1}^{\mathrm{C}} = x_r^{\mathrm{C}} + v_r. \tag{41}$$

The classical Nesterov state is $(x_r^{\mathrm{C}}, v_{r-1})$, initialized by $x_0^{\mathrm{C}} = x_0$ and $v_{-1} = 0$, and x_r^{C} is the reported iterate. The local block begins at the extrapolated point, but the Nesterov recurrence is applied only once per H local steps: its state is not refreshed again until the block has finished.

Modern outer update. The modern form stores the block start:

$$\Delta_{k,r}^{\mathrm{M}} = \Delta_{k,r}(x_r^{\mathrm{M}}), \qquad \overline{\Delta}_r^{\mathrm{M}} = \frac{1}{K}\sum_k \Delta_{k,r}^{\mathrm{M}}, \tag{42}$$

$$v_r = \mu v_{r-1} + \alpha_r^{\mathrm{M}} \overline{\Delta}_r^{\mathrm{M}}, \tag{43}$$

$$x_{r+1}^{\mathrm{M}} = x_r^{\mathrm{M}} + \alpha_r^{\mathrm{M}} \overline{\Delta}_r^{\mathrm{M}} + \mu v_r. \tag{44}$$

The modern Nesterov state is $(x_r^{\mathrm{M}}, v_{r-1})$, initialized by $x_0^{\mathrm{M}} = x_0$ and $v_{-1} = 0$. Its native reported iterate x_r^{M} is the stored look-ahead point. The matched classical output is $x_r^{\mathrm{M}} - \mu v_{r-1}$. DiLoCo uses this structure with local AdamW updates (Douillard et al., 2023).

Primal outer update. The primal form makes the averaging-and-feedback wrapper explicit:

$$y_r^{\mathrm{P}} = \mu x_r^{\mathrm{P}} + (1-\mu) z_r, \tag{45}$$

$$\Delta_{k,r}^{\mathrm{P}} = \Delta_{k,r}(y_r^{\mathrm{P}}), \qquad \overline{\Delta}_r^{\mathrm{P}} = \frac{1}{K}\sum_k \Delta_{k,r}^{\mathrm{P}}, \tag{46}$$

$$z_{r+1} = z_r + \alpha_r^{\mathrm{P}} \overline{\Delta}_r^{\mathrm{P}}, \tag{47}$$

$$x_{r+1}^{\mathrm{P}} = \mu x_r^{\mathrm{P}} + (1-\mu) z_{r+1}. \tag{48}$$

The primal Nesterov state is (x_r^{P}, z_r), initialized by $x_0^{\mathrm{P}} = z_0 = x_0$, and x_r^{P} is the reported iterate. Here the fast state accumulates block displacements, the reported state averages the fast state, and their interpolation determines the next common block start. The base optimizer still performs each local update. The wrapper acts on the aggregated result only once per round.

Block displacements and pseudo-gradients. In the gradient-subtraction convention, the outer optimizer receives $-\overline{\Delta}_r(a)$ as a *pseudo-gradient*: the starting parameters minus the mean local endpoint, as in DiLoCo and, for one worker, SNOO (Douillard et al., 2023; Kallusky et al., 2025). This is an accumulated optimization displacement, not generally a gradient of F at a. Even for SGD with a constant learning rate, normalizing by the accumulated step size gives an average of stochastic gradients evaluated along moving local trajectories, not generally an unbiased estimate of $\nabla F(a)$. With AdamW, the displacement additionally depends on optimizer history, adaptive scaling, and parameter-dependent weight decay. The SlowMo analysis explicitly retains a bias term between the expected averaged update and the full objective gradient (Wang et al., 2020, Theorem 1 and its discussion). Our wrapper equivalences require matched update maps. For the assumptions needed by averaging-wrapper convergence analyses, see Appendix A.2 (Defazio et al., 2024; 2026).

Equivalence for a complete block. The same local-block procedure can be wrapped in all three formulations without changing its internal optimizer or local learning rates. The required scale conversion applies to the *outer* displacement.

Proposition 2 (An arbitrary local block preserves equivalence between outer formulations)**.** *Suppose the three methods use the same worker-local block map whenever its input is the same. Initialize their round-level Nesterov states according to Equations* (32) *and* (33)*, and choose*

$$\alpha_r^{\mathrm{C}} = \alpha_r^{\mathrm{M}} = (1-\mu)\alpha_r^{\mathrm{P}}. \tag{49}$$

Then the boundary iterates satisfy the same state transformations at every round, for every H and every (possibly nonlinear and stochastic) local block.

Proof. At round r, the induction hypothesis gives $y_r^{\mathrm{C}} = x_r^{\mathrm{M}} = y_r^{\mathrm{P}}$. Each worker therefore starts from the same parameter and produces the same endpoint displacement in all three runs. The outer recurrences are then precisely the one-step recurrences of Proposition 1, with $U_r^{\mathrm{C}} = \alpha_r^{\mathrm{C}}\overline{\Delta}_r$ and $U_r^{\mathrm{P}} = \alpha_r^{\mathrm{P}}\overline{\Delta}_r$. The scale relation in Equation (49) completes the induction. □

Equivalence can be lost when a construction changes the local path, leaves recurrent state unspecified, resets state variables inconsistently across formulations, or compares unconverted outer learning rates. Endpoint averaging with state reconstruction preserves equivalence when the reconstruction follows the formulation map, as for tied PLN in Equations (12) and (13).

SHN: scaling the outer update. For SHN, replace the single-step primal displacement by $\alpha_r^{\mathrm{P}}\overline{\Delta}_r$ and implement the displacement conversion through the outer scales, keeping local base updates unchanged:

$$\alpha_r^{\mathrm{C}} = (1-\mu_x^{\mathrm{P}})\alpha_r^{\mathrm{P}}, \qquad \alpha_r^{\mathrm{M}} = (1-\mu_y^{\mathrm{P}})\alpha_r^{\mathrm{P}}. \tag{50}$$

The state maps give a common block-start point, so matched base-optimizer states and stochastic inputs generate identical worker-local blocks. Applying the single-step identities to their aggregated displacement preserves the state maps at the next boundary. Induction establishes equivalence for every K, H. The modern output is q_r^{M}, while the classical/primal output is $q_r^{\mathrm{M}} - \mu_x^{\mathrm{P}} v_{r-1}^{\mathrm{M}}$. Tied coefficients recover the usual conversion $\alpha_r^{\mathrm{C}} = \alpha_r^{\mathrm{M}} = (1-\mu)\alpha_r^{\mathrm{P}}$.

Endpoint averaging and state recovery. The mean pseudo-update can be implemented by averaging complete worker candidates. In the classical formulation define

$$v_{k,r} = \mu v_{r-1} + \alpha_r \Delta_{k,r}(y_r), \qquad x_{k,r+1} = x_r + v_{k,r}. \tag{51}$$

Then linearity gives

$$x_{r+1} = \frac{1}{K}\sum_k x_{k,r+1}, \qquad v_r = x_{r+1} - x_r = \frac{1}{K}\sum_k v_{k,r}. \tag{52}$$

For the modern formulation, averaging the worker candidates still suffices, but the momentum state is reconstructed differently:

$$x_{r+1}^{\mathrm{M}} = \frac{1}{K}\sum_k x_{k,r+1}^{\mathrm{M}}, \qquad v_r = \frac{x_{r+1}^{\mathrm{M}} - x_r^{\mathrm{M}} + \mu v_{r-1}}{1+\mu}. \tag{53}$$

The modern parameter increment is not itself the momentum state.

For the primal formulation, worker k forms

$$z_{k,r+1} = z_r + \alpha_r \Delta_{k,r}(y_r), \qquad x_{k,r+1} = \mu x_r + (1-\mu) z_{k,r+1}. \tag{54}$$

One may average either endpoint:

$$z_{r+1} = \frac{1}{K}\sum_k z_{k,r+1}, \qquad x_{r+1} = \mu x_r + (1-\mu) z_{r+1}, \tag{55}$$

$$x_{r+1} = \frac{1}{K}\sum_k x_{k,r+1}, \qquad z_{r+1} = \frac{x_{r+1} - \mu x_r}{1-\mu}. \tag{56}$$

These reductions are identical because the boundary reporting map is affine. However, averaging x without reconstructing z leaves the next round underspecified. More generally, averaging a visible parameter is equivalent to averaging pseudo-updates only if every hidden recurrent state is common, averaged as well, or recoverable from common history.

An update-difference view of the modern form. The modern recurrence also has a useful two-step representation. Define the round pseudo-update, with $A_{-1} = 0$,

$$A_r := \alpha_r^{\mathrm{M}}\overline{\Delta}_r(x_r^{\mathrm{M}}). \tag{57}$$

Eliminating the momentum state from Equation (44) gives

$$\boxed{x_{r+1}^{\mathrm{M}} = x_r^{\mathrm{M}} + \mu(x_r^{\mathrm{M}} - x_{r-1}^{\mathrm{M}}) + (1+\mu)A_r - \mu A_{r-1}.} \tag{58}$$

Hence modern Nesterov can be read as parameter extrapolation plus a difference correction between the current and previous pseudo-updates. The $-\mu A_{r-1}$ term is precisely what prevents the increment of the stored modern parameter from being identified with the momentum state. This two-step recurrence represents the same boundary update. With $x_{-1}^{\mathrm{M}} = x_0^{\mathrm{M}}$, it requires retaining either v_{r-1} or the previous parameter and pseudo-update.

B.4 RCN EQUIVALENCES AND LOCAL-PATH DIFFERENCES

The following results use the matching assumptions in Appendix B.1. We first express RCN-P as a scaled SHN block, then compare its path with RCN-C.

Generalized RCN boundaries. For classical RCN, the total injected motion becomes $\mu_y v_{r-1}$ and the boundary retains $\mu_x v_{r-1}$:

$$y_{k,r,h} = \theta_{k,r,h} + \frac{\mu_y}{H} v_{r-1}, \quad \Delta_{k,r}^{\mathrm{fresh}} = \theta_{k,r,H} - x_r - \mu_y v_{r-1}, \quad v_r = \mu_x v_{r-1} + \alpha_r \overline{\Delta}_r^{\mathrm{fresh}}. \tag{59}$$

For primal RCN, use

$$y_{k,r,h} = \mu_y x_r + (1-\mu_y) z_{k,r,h}, \tag{60}$$

$$\overline{\Delta}_r^z = \frac{1}{K}\sum_k (z_{k,r,H} - z_r), \tag{61}$$

$$x_{r+1} = \mu_x x_r + (1-\mu_x)(z_r + \alpha_r \overline{\Delta}_r^z). \tag{62}$$

Proposition 3 (Generalized RCN-P–SHN-P equivalence)**.** *Under the assumptions of Appendix B.1, start both primal methods from the same x_r, z_r and coefficients $0 \le \mu_x^{\mathrm{P}} < 1$, $0 \le \mu_y^{\mathrm{P}} < 1$. Choose the local-update and outer scales as*

$$U_{k,r,h}^{\mathrm{SHN-P}} = (1-\mu_y^{\mathrm{P}}) U_{k,r,h}^{\mathrm{RCN-P}}, \qquad \alpha_r^{\mathrm{SHN-P}} = \frac{\alpha_r^{\mathrm{RCN-P}}}{1-\mu_y^{\mathrm{P}}}. \tag{63}$$

Then all local query points and both primal boundary states agree for every K, H.

Proof. Their initial local evaluation point is $\theta_{k,r,0}^{\mathrm{SHN-P}} = y_{k,r,0}^{\mathrm{RCN-P}} = \mu_y^{\mathrm{P}} x_r + (1-\mu_y^{\mathrm{P}}) z_r$. During RCN-P the slow anchor remains fixed, so its look-ahead points obey

$$y_{k,r,h+1}^{\mathrm{RCN-P}} - y_{k,r,h}^{\mathrm{RCN-P}} = (1-\mu_y^{\mathrm{P}}) U_{k,r,h}^{\mathrm{RCN-P}}(y_{k,r,h}^{\mathrm{RCN-P}}). \tag{64}$$

Unlike RCN-C, this path contains no additional reference-injection term. With the local-update scaling in Equation (63), induction over local steps gives $\theta_{k,r,h}^{\mathrm{SHN-P}} = y_{k,r,h}^{\mathrm{RCN-P}}$ and

$$\Delta_{k,r}^{\mathrm{SHN-P}} = (1-\mu_y^{\mathrm{P}})(z_{k,r,H}^{\mathrm{RCN-P}} - z_r). \tag{65}$$

Thus the outer-scale conversion gives identical fast-state updates,

$$z_{r+1}^{\mathrm{SHN-P}} = z_r + \frac{\alpha_r^{\mathrm{RCN-P}}}{1-\mu_y^{\mathrm{P}}} \overline{\Delta}_r^{\mathrm{SHN-P}} = z_r + \alpha_r^{\mathrm{RCN-P}} \overline{\Delta}_r^z = z_{r+1}^{\mathrm{RCN-P}}, \tag{66}$$

and identical reported states $x_{r+1} = \mu_x^{\mathrm{P}} x_r + (1-\mu_x^{\mathrm{P}}) z_{r+1}$. Both boundary states therefore agree, so the next round satisfies the same starting conditions. Induction completes the proof. □

Composing with the SHN formulation maps yields the generalized RCN-P–SHN-C and RCN-P–SHN-M conversions. The restriction $\mu_y^{\mathrm{P}} < 1$ is essential to this conversion: at $\mu_y^{\mathrm{P}} = 1$, the RCN-P evaluation point remains at x_r throughout the block and the required SHN-P outer scale is undefined.

Classical SHN representation. Composing Proposition 3 with the single-step coefficient map and SHN outer-scale conversion gives, for $0 \le \mu_x^{\mathrm{P}}, \mu_y^{\mathrm{P}} < 1$,

$$\begin{aligned} \mu_x^{\mathrm{C}} &= \mu_x^{\mathrm{P}}, & \mu_y^{\mathrm{C}} &= \frac{\mu_x^{\mathrm{P}}(1-\mu_y^{\mathrm{P}})}{1-\mu_x^{\mathrm{P}}}, \\ U_{k,r,h}^{\mathrm{C}} &= (1-\mu_y^{\mathrm{P}})U_{k,r,h}^{\mathrm{P}}, & \alpha_r^{\mathrm{C}} &= \frac{1-\mu_x^{\mathrm{P}}}{1-\mu_y^{\mathrm{P}}}\alpha_r^{\mathrm{P}}. \end{aligned} \tag{67}$$

The shared states obey $x_r^{\mathrm{C}} = x_r^{\mathrm{P}}$ and $z_r^{\mathrm{P}} = x_r^{\mathrm{C}} + \mu_x^{\mathrm{P}} v_{r-1}^{\mathrm{C}}/(1-\mu_x^{\mathrm{P}})$. Unlike the single-step or PLN conversion, the local scaling is $1-\mu_y^{\mathrm{P}}$. The boundary scale compensates for the difference between interpolation and retention.

Proposition 4 (Tied RCN-P–SHN-C equivalence)**.** *Under the assumptions of Appendix B.1, tied RCN-P and SHN-C with $0 \le \mu < 1$, states related by Equation* (33)*, and equal outer scales have identical local query points and reported boundary iterates when*

$$U_{k,r,h}^{\mathrm{C}}(a) = (1-\mu)U_{k,r,h}^{\mathrm{P}}(a). \tag{68}$$

Proof. Set $\mu_x^{\mathrm{P}} = \mu_y^{\mathrm{P}} = \mu$ in Equation (67). The classical pair becomes (μ, μ) and its outer scale equals the RCN-P scale, so the preceding composition proves the claim. □

Modern SHN representation and reported outputs. SHN-M stores the common block start q_r^{M} and updates it using the mean local-block displacement d_r:

$$v_r^{\mathrm{M}} = \mu_x^{\mathrm{M}} v_{r-1}^{\mathrm{M}} + \alpha_r^{\mathrm{M}} d_r, \qquad q_{r+1}^{\mathrm{M}} = q_r^{\mathrm{M}} + \alpha_r^{\mathrm{M}} d_r + \mu_y^{\mathrm{M}} v_r^{\mathrm{M}}. \tag{69}$$

Combining the RCN-P–SHN-P conversion in Equation (63) with Equation (50) gives

$$\alpha_r^{\mathrm{SHN-M}} = \alpha_r^{\mathrm{RCN-P}}, \qquad U_{k,r,h}^{\mathrm{SHN-M}} = (1-\mu_y^{\mathrm{P}})U_{k,r,h}^{\mathrm{RCN-P}}. \tag{70}$$

Thus the coefficient conversion in Equation (25) relates these constructions with equal outer scales and the indicated local-update scaling. The full state correspondence is

$$x_r^{\mathrm{P}} = q_r^{\mathrm{M}} - \mu_x^{\mathrm{M}} v_{r-1}^{\mathrm{M}}, \tag{71a}$$

$$z_r^{\mathrm{P}} = q_r^{\mathrm{M}} + \frac{\mu_x^{\mathrm{M}}\mu_y^{\mathrm{M}}}{1-\mu_x^{\mathrm{M}}} v_{r-1}^{\mathrm{M}}. \tag{71b}$$

Common initialization with zero momentum satisfies these relations. Equivalently, with $t = 1 + \mu_y^{\mathrm{M}} - \mu_x^{\mathrm{M}} > 0$, $x_r^{\mathrm{C}} = q_r^{\mathrm{M}} - \mu_x^{\mathrm{M}} v_{r-1}^{\mathrm{M}}$ and $w_{r-1} = t v_{r-1}^{\mathrm{M}}$, the mapped classical wrapper is

$$q_r^{\mathrm{M}} = x_r^{\mathrm{C}} + \frac{\mu_x^{\mathrm{M}}}{t} w_{r-1}, \qquad w_r = \mu_x^{\mathrm{M}} w_{r-1} + t\alpha_r^{\mathrm{M}} d_r, \qquad x_{r+1}^{\mathrm{C}} = x_r^{\mathrm{C}} + w_r. \tag{72}$$

This is Equation (67) with $\mu_y^{\mathrm{P}} = \mu_y^{\mathrm{M}}/t$, so the state correspondence follows from the preceding composition. Scaling must include the complete optimizer displacement, including weight decay. In our AdamW implementation, the normalization $\eta^{\mathrm{P}} = r/(1-\mu_y^{\mathrm{P}})$ achieves this with the same searched rate r as SHN-M. The full rate normalization is specified alongside Algorithm 1.

Under this map, SHN-M reports q_r^{M} and RCN-P reports x_r^{P}. Consequently a native-output loss difference alone does not establish different optimization trajectories. Every modern pair in $[0,1)^2$ has a primal representative there, but a primal pair has a modern second coefficient below one only when $\mu_y^{\mathrm{P}} < 1/(2-\mu_x^{\mathrm{P}})$. For example, $(\mu_x^{\mathrm{P}}, \mu_y^{\mathrm{P}}) = (0.5, 0.75)$ requires $\mu_y^{\mathrm{M}} = 1.5$. Even where both pairs are admissible, a finite coefficient grid need not contain the translated pair. The selected recipes and their output conventions are compared in Section 6.1.

Interpreting the local-path difference. The tied local look-ahead paths obey

$$\text{RCN-C:} \quad y_{k,r,h+1} - y_{k,r,h} = \frac{\mu}{H} v_{r-1} + U_{k,r,h}(y_{k,r,h}), \tag{73}$$

$$\text{RCN-P:} \quad y_{k,r,h+1} - y_{k,r,h} = (1-\mu)U_{k,r,h}(y_{k,r,h}). \tag{74}$$

Even with matched initial look-ahead points, the subsequent paths generally differ. Writing $c_r = \mu v_{r-1}/H$, Equations (73) and (74) show that the classical path receives a constant displacement from incoming momentum and the full base displacement, whereas the primal query path advances by a damped fraction of the fast displacement. The relative size and direction of their parameter

displacements depend on the incoming momentum contribution c_r, local curvature, and base optimizer. A controlled comparison must account for both the repeated injection and the displacement scaling. Because RCN changes the local block itself, Proposition 2 does not automatically identify its classical and primal constructions.

RCN-C injects $\mu v_{r-1}/H$ at every local step, whereas RCN-P's look-ahead points follow an ordinary scaled block without this injection (Equations (64) and (73)). Their tied, correctly scaled versions agree at $H = 1$ but generally evaluate base updates at different points for $H > 1$. Generalized RCN-P remains equivalent to SHN after remapping, with distinct factors $1 - \mu_y^{\mathrm{P}}$ for its local path and $1 - \mu_x^{\mathrm{P}}$ for its reported boundary (Equation (67)).

For a scalar counterexample, take $F(s) = s^2/2$, $H = 2$, $\mu = 0.5$, $\alpha = 1$, and incoming classical state $(x_r, v_{r-1}) = (1, 0.2)$, mapped to primal $(x_r, z_r) = (1, 1.2)$. Use $U^{\mathrm{C}}(s) = -0.1s$ and the correctly scaled $U^{\mathrm{P}}(s) = -0.2s$. RCN-C queries 1.05 and 0.995 and reports 0.8955, whereas RCN-P queries 1.1 and 0.99 and reports 0.891. Thus the tied state and update-scale conversion does not equate the two RCN constructions for $H > 1$.

Tied boundary identities and momentum subtraction. The following recovery identities specialize to tied coefficients. Every local update assigns the look-ahead to the evolving parameter before adding the base update. The endpoint therefore includes the total injected momentum displacement:

$$\theta_{k,r,H} - x_r = \mu v_{r-1} + \Delta_{k,r}^{\mathrm{fresh}}. \tag{75}$$

Using the uncorrected endpoint displacement in the outer update would count incoming momentum twice: once in that displacement and once in the retention term μv_{r-1}. The definition of $\Delta_{k,r}^{\mathrm{fresh}}$ in Equation (10) subtracts the explicitly injected momentum contribution. It removes the known displacement injected by the algorithm. It does not undo the effect of that displacement on the look-ahead points.

Boundary and state recovery. Equivalently, each worker may first form $z_{k,r+1}^{\mathrm{cand}} = z_r + \alpha_r(z_{k,r,H} - z_r)$ and then $x_{k,r+1} = \mu x_r + (1-\mu) z_{k,r+1}^{\mathrm{cand}}$. The system may average x, reconstructing

$$z_{r+1} = \frac{x_{r+1} - \mu x_r}{1 - \mu}. \tag{76}$$

Although an intermediate candidate $\widehat{x}_{k,r,h} = \mu x_r + (1-\mu) z_{k,r,h}$ would vary with h, it is not fed back as the anchor in the next interpolation. The construction therefore remains RCN rather than per-local-step Nesterov.

Primal RCN requires no subtraction of injected momentum. Interpolation chooses the point at which $U_{k,r,h}$ is evaluated, but the algorithm never assigns $z_{k,r,h} \leftarrow y_{k,r,h}$. Hence $z_{k,r,H} - z_r$ contains only accumulated base-optimizer displacements. Old momentum is encoded in the common x_r–z_r gap and enters the gradient locations without appearing as an additive term in the measured fast endpoint.

Causality and modern state representation. Modern Nesterov uses the newly formed momentum state in its corrector:

$$v_r = \mu v_{r-1} + \alpha_r \overline{\Delta}_r, \qquad x_{r+1}^{\mathrm{M}} = x_r^{\mathrm{M}} + \alpha_r \overline{\Delta}_r + \mu v_r. \tag{77}$$

A literal attempt to distribute that corrector through the same local block would require adding $\mu v_r/H$ before v_r is known. Since v_r depends on the endpoint produced by those very steps, this literal construction is implicit, not a causal explicit recurrence for a single forward pass through the block. Predicting v_r defines a modified method unless the prediction satisfies the original endpoint relation. Solving that relation can require additional passes or a fixed-point procedure.

Classical RCN can nevertheless be *stored* using the modern state representation at round boundaries:

$$x_r^{\mathrm{M}} = x_r^{\mathrm{C}} + \mu v_{r-1}, \qquad x_{r+1}^{\mathrm{M}} = x_{r+1}^{\mathrm{C}} + \mu v_r. \tag{78}$$

This is a causal change of representation: it reconstructs the classical base from the available incoming state, runs Equation (10), and maps the result back after the block. It is an alternative state representation of classical RCN, not a third native within-round trajectory.

B.5 PLN SYNCHRONIZATION, PROJECTION, AND FORMULATION EQUIVALENCE

Complete-state classical averaging uses

$$x_{r+1}^{2\mathrm{V}} = \frac{1}{K}\sum_k x_{k,r,H}, \qquad v_r^{2\mathrm{V}} = \frac{1}{K}\sum_k u_{k,r,H}. \tag{79}$$

The corresponding primal reference, a direct multi-worker GPA extension, averages both $x_{k,r,H}$ and $z_{k,r,H}$. These reference policies preserve literal local Nesterov continuation at $K = 1$ for any H and preserve correctly remapped formulation equivalence because averaging commutes with the linear state map. They require two model-sized reduction inputs in the general multi-worker case, even if packed into a single collective call.

For a block started from the same state with the same stochastic inputs, 1V and 2V have identical local trajectories and identical shared model endpoints in that block. Their next momentum states differ by

$$e_r := v_r^{1\mathrm{V}} - v_r^{2\mathrm{V}} = \frac{1}{H}\sum_{i=1}^{H}(\bar{u}_{r,i} - \bar{u}_{r,H}), \qquad \bar{u}_{r,i} := \frac{1}{K}\sum_k u_{k,r,i}, \tag{80}$$

$$\|e_r\| \le \frac{1}{H}\sum_{i=1}^{H}\|\bar{u}_{r,i} - \bar{u}_{r,H}\|. \tag{81}$$

Thus the boundary projection replaces the mean terminal local-step displacement by the mean displacement over the block. It is exact at $H = 1$ and whenever the averaged terminal displacement equals that block average, including a constant averaged local momentum. Small within-block variation makes the bound small. No general small-error guarantee follows from low worker disagreement alone. The error can be nonzero at $K = 1$, so projection generally changes even a single worker's trajectory after the first boundary when $H > 1$. Subsequent base-update evaluation points and adaptive base states can then diverge, so identical endpoints for one matched-start block do not imply identical full training runs.

Primal reconstruction versus fast-state averaging. The complete-state reference sets $z_{r+1}^{2\mathrm{V}} = K^{-1}\sum_k z_{k,r,H}$, whereas projected PLN-P uses the reconstructed $z_{r+1}^{1\mathrm{V}}$ in Equation (13) as the next shared z_{r+1}. For the same incoming state and local trajectories, the identity $z_{k,r,H} - x_{k,r,H} = \mu_x(x_{k,r,H} - x_{k,r,H-1})/(1-\mu_x)$ gives

$$z_{r+1}^{1\mathrm{V}} - z_{r+1}^{2\mathrm{V}} = \frac{\mu_x}{1-\mu_x}\left[\frac{x_{r+1} - x_r}{H} - \frac{1}{K}\sum_k (x_{k,r,H} - x_{k,r,H-1})\right]. \tag{82}$$

Averaging terminal z retains the workers' mean final-step displacement through the primal inverse relation, whereas reconstruction substitutes their mean displacement per step over the whole block. Both policies give the same x_{r+1} for this matched-start block but generally different z_{r+1}, changing the next look-ahead point and subsequent trajectory. A smoothing coefficient close to one can amplify this gap in the primal auxiliary state. At $K = 1$, the complete-state reference continues ordinary GPA, whereas projection can change its trajectory when $H > 1$.

Proposition 5 (Generalized projected PLN-C/P equivalence)**.** *Under the assumptions of Appendix B.1, take primal coefficients $0 \le \mu_x^{\mathrm{P}} < 1$, $0 \le \mu_y^{\mathrm{P}} \le 1$, classical coefficients and complete-update scaling from Equation* (18)*, and compatible initial states. The one-vector PLN-C/P algorithms in Algorithms 8 and 9 have identical local query points and reported boundary iterates for every K, H.*

Proof. Within a block, the single-step map preserves identical reported states and query points, with $z_{k,h} = x_{k,h}^{\mathrm{C}} + \mu_x^{\mathrm{P}} u_{k,h}^{\mathrm{C}}/(1-\mu_x^{\mathrm{P}})$. Averaging therefore yields the same reported endpoint x_{r+1}. The projected classical momentum is $v_r^{\mathrm{C}} = (x_{r+1} - x_r)/H$, and the primal reconstruction satisfies

$$z_{r+1}^{\mathrm{P}} = x_{r+1} + \frac{\mu_x^{\mathrm{P}}}{1-\mu_x^{\mathrm{P}}}\frac{x_{r+1} - x_r}{H} = x_{r+1}^{\mathrm{C}} + \frac{\mu_x^{\mathrm{P}}}{1-\mu_x^{\mathrm{P}}} v_r^{\mathrm{C}}. \tag{83}$$

The next block begins with the same state relation, completing induction over rounds. □

Direct endpoint averaging. PLN averages model endpoints directly, with no additional outer momentum term to correct.

Local memory versus round memory. Comparing local retention μ_x^ℓ with round retention μ^{r}, choosing $\mu_x^\ell = (\mu^{\mathrm{r}})^{1/H}$ matches only the retention of the incoming momentum in the *terminal local recurrence* on a block with zero base-optimizer displacements. Under the projected boundary rule its contribution to v_r instead has coefficient $H^{-1}\sum_{i=1}^{H}(\mu_x^\ell)^i$, so $(\mu_x^\ell)^H$ must not be interpreted as the retention of the shared projected state.

Numerical verification. An independent float64 audit uses matched nonlinear stochastic gradients with $K \in \{1, 4\}$, $H \in \{1, 4, 16\}$, $\mu \in \{0, 0.4, 0.9\}$, eight rounds, and both boundary policies (36 formulation comparisons). The maximum tied C/P state-map discrepancy is 3.11×10^{-15}. The $H = 1$ policy, projection-error, zero-displacement retention, and independent-coefficient primal $H = 1$ reconstruction checks have maximum discrepancies below 3.34×10^{-15}. A single-worker control also exhibits different trajectories under 1V and 2V when $H > 1$. The reproducible script is `experiments/verify_pln_projection.py`, with its numerical record in `experiments/results/pln-projection-audit.json`.

C Complete algorithms

We give the algorithms underlying the language-model experiments and the SHN formulation references. Each primary method communicates one model-sized optimization vector per round. The recurrences use independent coefficients. Tying each retention and look-ahead pair gives the tied variants. The complete-state PLN reference is specified separately in Appendix B.5.

C.1 Conventions and communication

Each algorithm specifies one round r, with H local updates on each of K workers. All vectors have dimension d. Worker loops run independently, and $\mathrm{Mean}_k(a_k) = K^{-1}\sum_k a_k$ denotes one model-sized reduction. Initialize shared parameters at x_0, momentum states at zero, and auxiliary parameters at $z_0 = q_0 = x_0$ where present. The shared momentum is v_{r-1}. For PLN it is measured in local-step displacement units, while TTN additionally maintains worker-local momentum u. Reported parameters are the shared post-update x, except that SHN-M natively reports its stored look-ahead q. Single-worker controls retain the same round-boundary rules.

Single-timescale constructions use μ_x, μ_y for retention and look-ahead/interpolation, including PLN's local updates. TTN distinguishes round-level coefficients $\mu_x^{\mathrm{r}}, \mu_y^{\mathrm{r}}$ from local coefficients μ_x^ℓ, μ_y^ℓ. Retention coefficients μ_x (including both TTN timescales) lie in $[0, 1)$. Classical/modern look-ahead coefficients are nonnegative, while primal interpolation coefficients lie in $[0, 1]$ (Section 4). The normalized RCN-P learning rate used in the experiments requires $\mu_y < 1$. Tied TTN sets $\mu_x^{\mathrm{r}} = \mu_y^{\mathrm{r}}$ and $\mu_x^\ell = \mu_y^\ell$, without requiring equality between timescales. The outer scale $\alpha_r > 0$ appears only where specified. PLN has no additional outer relaxation.

Each $U_{k,r,h}(a)$ evaluates one accumulated stochastic gradient at a and advances worker k's base-optimizer state once, with absolute update index $s = rH + h$. AdamW moments start at zero, persist privately across rounds, and are never averaged across logical workers. Nesterov state is separate from this base-optimizer state.

A logical worker may comprise several physical ranks: gradients are averaged within that group before clipping and the base update. Group leaders perform the cross-worker optimization reduction and broadcast its result within each group. Its input payload is db bytes per worker for b bytes per scalar. Measured network traffic also depends on the collective protocol. The experiments use FP32 optimization payloads and separately account for initialization, within-worker gradient averaging, evaluation/control traffic, and a 20-scalar FP64 diagnostic reduction per round (160 bytes per worker). This scalar reduction does not synchronize additional optimizer-state vectors.

C.1.1 State, storage, and boundary policies

Algorithmic state and storage. The Nesterov-state definitions in Section 2 describe what is needed to continue the recurrence given the step index and prescribed schedules, not the number of extra buffers. They include model iterates. Derived look-ahead points need not be stored independently, and equivalent implementations can reconstruct some variables from others. Base-optimizer state excludes

these model and Nesterov variables. Plain SGD without momentum has no persistent internal state of this kind. Neither a stochastic gradient step nor a general base-optimizer displacement is assumed to decrease the loss at every iteration.

Boundary policies. Base-optimizer momentum remains part of $U_{k,r,h}$ and is distinct from Nesterov state. PLN reconstructs shared Nesterov state. The tested TTN resets its additional local momentum. Private retention or complete-state averaging define different methods, even when local recurrences agree. The latter generally requires an additional model-sized vector. Appendix B.5 analyzes the PLN boundary alternatives.

C.2 ALGORITHMS

SHN-C/P provide formulation references. The experiments use SHN-M, RCN-C/P, PLN-C/P, and TTN-C with the tied/generalized choices specified in Section 6.

Algorithm 1 Base-optimizer displacement $U_{k,r,h}(a)$: accumulation, worker averaging, and SGD/AdamW

Require: Point a; worker's physical ranks $\mathcal{J}_k$ and microbatches $\xi_{j,\ell}$; rate η_s; decay λ and mask M; clip bound τ; private (m_k, q_k, t_k)

for all physical ranks $j \in \mathcal{J}_k$ in parallel **do**
  $g_j \leftarrow \sum_\ell w_{j,\ell} \nabla f(a; \xi_{j,\ell})$, with $\sum_\ell w_{j,\ell} = 1$ ▷ sample-weighted accumulation
end for
$g \leftarrow |\mathcal{J}_k|^{-1} \sum_{j \in \mathcal{J}_k} g_j$ ▷ within-worker gradient average
if $\tau < \infty$ **then**
  $g \leftarrow g \min\{1, \tau/(\|g\|_2 + 10^{-6})\}$
end if
if SGD **then**
  return $-\eta_s(g + \lambda M \odot a)$
else
  $t_k \leftarrow t_k + 1$
  $m_k \leftarrow b_1 m_k + (1 - b_1) g$; $q_k \leftarrow b_2 q_k + (1 - b_2) g \odot g$
  $\widehat{m}_k \leftarrow m_k/(1 - b_1^{t_k})$; $\widehat{q}_k \leftarrow q_k/(1 - b_2^{t_k})$
  return $-\eta_s\big(\widehat{m}_k/(\sqrt{\widehat{q}_k} + \epsilon) + \lambda M \odot a\big)$
end if

All products, square roots, and divisions involving AdamW moments are elementwise. The experiments use equal effective batch sizes per physical rank, clipping bound $\tau = 1$, $(b_1, b_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, and decay $\lambda = 0.1$. The mask M selects parameter tensors with at least two dimensions, including the tied embedding/output matrix, and excludes normalization scales. Decay is evaluated at the gradient query point a. Physical ranks within a worker maintain identical AdamW states, without cross-worker averaging. SGD has no additional base momentum. For search-coordinate rate $\eta_s^{\text{configured}}$, the native rate is $\eta_s^{\text{configured}}/(1 - \mu_y)$ for RCN-P, $(1 - \mu_x)\eta_s^{\text{configured}}$ for PLN-C, $(1 - \mu_x^\ell)\eta_s^{\text{configured}}$ for TTN-C, and $\eta_s^{\text{configured}}$ otherwise. Formulation equivalence requires scaling the complete displacement, including weight decay, and matching base-optimizer state transitions.

Algorithm 2 FedAvg: plain local steps (one vector)

Require: Shared x_r

for all workers k in parallel **do**
  $\theta_{k,0} \leftarrow x_r$
  for $h = 0, \ldots, H - 1$ **do**
    $\theta_{k,h+1} \leftarrow \theta_{k,h} + U_{k,r,h}(\theta_{k,h})$
  end for
end for
$d_r \leftarrow \text{Mean}_k(\theta_{k,H} - x_r)$
$x_{r+1} \leftarrow x_r + d_r$

Algorithm 3 Generalized classical Step-H outer Nesterov (SHN-C, one vector)

Require: Shared x_r, v_{r-1}; μ_x, μ_y, α_r

$y_r \leftarrow x_r + \mu_y v_{r-1}$
for all workers k in parallel **do**
  $\theta_{k,0} \leftarrow y_r$
  for $h = 0, \dots, H-1$ **do**
    $\theta_{k,h+1} \leftarrow \theta_{k,h} + U_{k,r,h}(\theta_{k,h})$
  end for
end for
$d_r \leftarrow \mathrm{Mean}_k(\theta_{k,H} - y_r)$
$v_r \leftarrow \mu_x v_{r-1} + \alpha_r d_r$; $x_{r+1} \leftarrow x_r + v_r$

Algorithm 4 Modern Step-H outer Nesterov (SHN-M, one vector)

Require: Shared stored look-ahead q_r, momentum v_{r-1}; μ_x, μ_y, α_r

for all workers k in parallel **do**
  $\theta_{k,0} \leftarrow q_r$
  for $h = 0, \dots, H-1$ **do**
    $\theta_{k,h+1} \leftarrow \theta_{k,h} + U_{k,r,h}(\theta_{k,h})$
  end for
end for
$d_r \leftarrow \mathrm{Mean}_k(\theta_{k,H} - q_r)$
$v_r \leftarrow \mu_x v_{r-1} + \alpha_r d_r$
$q_{r+1} \leftarrow q_r + \alpha_r d_r + \mu_y v_r$

SHN-M natively reports q_{r+1}. Its matched smoothed output is $q_{r+1} - \mu_x v_r$ under the state and coefficient transformations in Equations (25) and (71). For tied coefficients $\mu_x = \mu_y = \mu$, this is the classical output with the same local updates and outer scale. Independent coefficients require remapping, rather than comparing classical and modern recurrences at the same numerical pair (Equation (72)).

Algorithm 5 Generalized primal Step-H outer Nesterov (SHN-P, one vector)

Require: Shared x_r, z_r; $\mu_x, \mu_y, \alpha_r^{\mathrm{P}}$

$y_r \leftarrow \mu_y x_r + (1-\mu_y) z_r$
for all workers k in parallel **do**
  $\theta_{k,0} \leftarrow y_r$
  for $h = 0, \dots, H-1$ **do**
    $\theta_{k,h+1} \leftarrow \theta_{k,h} + U_{k,r,h}(\theta_{k,h})$
  end for
end for
$d_r \leftarrow \mathrm{Mean}_k(\theta_{k,H} - y_r)$
$z_{r+1} \leftarrow z_r + \alpha_r^{\mathrm{P}} d_r$
$x_{r+1} \leftarrow \mu_x x_r + (1-\mu_x) z_{r+1}$

For tied coefficients, match the same local block and set $\alpha_r^{\mathrm{C}} = (1-\mu)\alpha_r^{\mathrm{P}}$ for the SHN formulation equivalence. This outer-rate conversion is distinct from converting each local displacement in PLN.

Algorithm 6 Generalized classical round-conditioned Nesterov (RCN-C, one vector)

Require: Shared x_r, v_{r-1}; μ_x, μ_y, α_r

$c_r \leftarrow (\mu_y/H)v_{r-1}$
for all workers k in parallel **do**
  $\theta_{k,0} \leftarrow x_r$
  for $h = 0, \dots, H-1$ **do**
    $y_{k,h} \leftarrow \theta_{k,h} + c_r$
    $\theta_{k,h+1} \leftarrow y_{k,h} + U_{k,r,h}(y_{k,h})$
  end for
end for
$d_r \leftarrow \mathrm{Mean}_k(\theta_{k,H} - x_r)$
$v_r \leftarrow \mu_x v_{r-1} + \alpha_r(d_r - \mu_y v_{r-1})$
$x_{r+1} \leftarrow x_r + v_r$

The subtraction removes the known injected displacement $\mu_y v_{r-1}$ before applying outer momentum, without removing its effect on the local gradient queries.

Algorithm 7 Generalized fixed-anchor primal RCN (RCN-P, one vector)

Require: Shared x_r, z_r; μ_x, μ_y, α_r

for all workers k in parallel **do**
  $z_{k,0} \leftarrow z_r$
  for $h = 0, \dots, H-1$ **do**
    $y_{k,h} \leftarrow \mu_y x_r + (1-\mu_y) z_{k,h}$
    $z_{k,h+1} \leftarrow z_{k,h} + U^{\mathrm{P}}_{k,r,h}(y_{k,h})$
  end for
end for
$d_r^z \leftarrow \mathrm{Mean}_k(z_{k,H} - z_r)$
$z_{r+1} \leftarrow z_r + \alpha_r d_r^z$
$x_{r+1} \leftarrow \mu_x x_r + (1-\mu_x) z_{r+1}$

The anchor x_r stays fixed throughout the local loop. Averaging worker candidates $\mu_x x_r + (1-\mu_x)[z_r + \alpha_r(z_{k,H} - z_r)]$ is the equivalent one-vector implementation. Recover $z_{r+1} = (x_{r+1} - \mu_x x_r)/(1-\mu_x)$.

Algorithm 8 Generalized PLN-C with endpoint projection (one vector)

Require: Shared x_r, v_{r-1}; local coefficients μ_x, μ_y

for all workers k in parallel **do**
  $x_{k,0} \leftarrow x_r$; $u_{k,0} \leftarrow v_{r-1}$
  for $h = 0, \dots, H-1$ **do**
    $y_{k,h} \leftarrow x_{k,h} + \mu_y u_{k,h}$
    $u_{k,h+1} \leftarrow \mu_x u_{k,h} + U^{\mathrm{C}}_{k,r,h}(y_{k,h})$
    $x_{k,h+1} \leftarrow x_{k,h} + u_{k,h+1}$
  end for
end for
$d_r \leftarrow \mathrm{Mean}_k(x_{k,H} - x_r)$; $x_{r+1} \leftarrow x_r + d_r$
$v_r \leftarrow d_r/H$ ▷ reconstruct shared local-step momentum

Algorithm 9 Generalized PLN-P with endpoint projection (one vector)

Require: Shared x_r, z_r; local coefficients μ_x, μ_y

for all workers k in parallel **do**
  $x_{k,0} \leftarrow x_r$; $z_{k,0} \leftarrow z_r$
  for $h = 0, \dots, H-1$ **do**
    $y_{k,h} \leftarrow \mu_y x_{k,h} + (1-\mu_y) z_{k,h}$
    $z_{k,h+1} \leftarrow z_{k,h} + U^{\mathrm{P}}_{k,r,h}(y_{k,h})$
    $x_{k,h+1} \leftarrow \mu_x x_{k,h} + (1-\mu_x) z_{k,h+1}$
  end for
end for
$d_r \leftarrow \mathrm{Mean}_k(x_{k,H} - x_r)$; $x_{r+1} \leftarrow x_r + d_r$
$z_{r+1} \leftarrow x_{r+1} + \dfrac{\mu_x}{1-\mu_x}\dfrac{d_r}{H}$ ▷ reconstruct shared fast state

Algorithm 10 Generalized classical two-timescale Nesterov (TTN-C, one vector, local reset)

Require: Shared x_r, v_{r-1}; $\mu^{\mathrm{r}}_x, \mu^{\mathrm{r}}_y, \mu^{\ell}_x, \mu^{\ell}_y, \alpha_r$

$c_r \leftarrow (\mu^{\mathrm{r}}_y / H) v_{r-1}$
for all workers k in parallel **do**
  $\theta_{k,0} \leftarrow x_r$; $u_{k,0} \leftarrow 0$
  for $h = 0, \dots, H-1$ **do**
    $y_{k,h} \leftarrow \theta_{k,h} + c_r + \mu^{\ell}_y u_{k,h}$
    $u_{k,h+1} \leftarrow \mu^{\ell}_x u_{k,h} + U_{k,r,h}(y_{k,h})$
    $\theta_{k,h+1} \leftarrow \theta_{k,h} + c_r + u_{k,h+1}$
  end for
end for
$d_r \leftarrow \mathrm{Mean}_k(\theta_{k,H} - x_r)$
$v_r \leftarrow \mu^{\mathrm{r}}_x v_{r-1} + \alpha_r(d_r - \mu^{\mathrm{r}}_y v_{r-1})$; $x_{r+1} \leftarrow x_r + v_r$
Discard $u_{k,H}$ and retain each worker's base-optimizer state

Setting both local coefficients to zero recovers RCN-C with the same outer parameters and base updates.

General TTN template and local-state policies. More generally, TTN can distribute the shared reference through weights a_h and specify a local initialization $u^{\mathrm{init}}_{k,r}$:

$$\theta_{k,r,0} = x_r, \qquad u_{k,r,0} = u^{\mathrm{init}}_{k,r}, \tag{84}$$

$$y_{k,r,h} = \theta_{k,r,h} + a_h v_{r-1} + \mu^{\ell}_y u_{k,r,h}, \tag{85}$$

$$u_{k,r,h+1} = \mu^{\ell}_x u_{k,r,h} + U_{k,r,h}(y_{k,r,h}), \tag{86}$$

$$\theta_{k,r,h+1} = \theta_{k,r,h} + a_h v_{r-1} + u_{k,r,h+1}, \qquad \sum_{h=0}^{H-1} a_h = \mu^{\mathrm{r}}_y. \tag{87}$$

The boundary subtracts the known injection before updating shared momentum:

$$\overline{\Delta}^{\mathrm{corr}}_r = \frac{1}{K}\sum_k (\theta_{k,r,H} - x_r) - \mu^{\mathrm{r}}_y v_{r-1}, \tag{88}$$

$$v_r = \mu^{\mathrm{r}}_x v_{r-1} + \alpha_r \overline{\Delta}^{\mathrm{corr}}_r, \qquad x_{r+1} = x_r + v_r. \tag{89}$$

The experiments use uniform injection $a_h = \mu^{\mathrm{r}}_y / H$, reset $u^{\mathrm{init}}_{k,r} = 0$, and tie coefficients within each timescale. Private retention of terminal u would change the method without adding communication. Averaging it would require a second vector. Both alternatives differ from the tested reset policy.

D ADDITIONAL LANGUAGE-MODEL EXPERIMENTAL DETAILS

This appendix gives the workload, search protocol, validation results, and numerical controls supporting Section 6. Unless explicitly identified as test or larger-model results, losses are 124M endpoint validation losses in nats/token. Complete selected recipes are in Appendix E.

D.1 WORKLOADS AND IMPLEMENTATION

D.1.1 124M WORKLOAD AND NUMERICAL PROTOCOL

The model has 123,587,328 parameters: 12 layers, width 768, 12 attention heads, sequence length 1,024, SwiGLU intermediate width 2,048, RoPE, RMSNorm, and tied token embeddings, without biases or dropout. It uses a GPT-2 tokenizer and a 50,304-row embedding with 50,257 active vocabulary entries. FineWeb (Penedo et al., 2024) is packed into fixed-length sequences and shuffled using a fixed order. Attention is causal without document isolation. Exact document matches between training and held-out data are excluded, and checksums record the source and prepared datasets. Every full-budget 124M trial uses 1,073,741,824 training tokens and seed 501. This single-seed design measures hyperparameter and setting dependence, not seed robustness. The comparison covers eleven settings, S0–S10, spanning global batches 128–1024, logical worker counts 1, 4, and 8, and local intervals 32, 128, and 512 (Table 6). It is a selected grid rather than the full Cartesian product. All settings use eight physical GPUs, grouped into K logical workers.

Compared methods. The nine accelerated reporting variants are tied/generalized SHN-M, tied/generalized RCN-C, generalized RCN-P, tied/generalized PLN-C, generalized PLN-P, and TTN-C tied within each timescale. FedAvg with local AdamW is the baseline, retaining AdamW's internal momentum without an additional Nesterov update. SHN-M uses DiLoCo-style outer acceleration, RCN-C injects shared momentum at each local step, and PLN-C and TTN-C also update local Nesterov momentum. The primal variants use the averaging formulation.

We omit tied SHN-C/P, tied RCN-P, and tied PLN-P as separate reporting variants because they duplicate included methods under the state, scale, and output conversions in Section 5. SHN-M reports its modern stored iterate, with the reconstructed smoothed iterate evaluated separately. Generalized C/P recipes use the parameter domains and rate conversions specified in Section 4.

Optimizer and evaluation. Every method sends one FP32 model-sized optimization vector per round. PLN averages model endpoints and reconstructs incoming local-step momentum from displacement divided by H. TTN resets its additional local Nesterov momentum. Each worker retains its Adam states across rounds. Appendix C specifies the recurrences, reporting points, coefficient names, and update-scale conversions. The base optimizer is AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, matrix-only decay 0.1, and gradient-norm clipping at 1 after accumulation and within-worker gradient averaging. S0 uses microbatch 16 to fit its effective local batch of 16. All other settings use microbatch 32, with gradient accumulation and within-worker data parallelism as required. A token-indexed schedule warms up for 5% of training and follows cosine decay to 10% of peak learning rate.

We evaluate the shared model after each eighth of training, using 1,048,576 validation tokens at the seven intermediate checks and 10,485,760 at the endpoint. A separate 10,485,760-token test set is used only after selection. No best intermediate checkpoint is substituted.

D.1.2 1B-MODEL TRANSFER AND EVALUATION PROTOCOL

Larger-model workload and transferred recipes. We retain the model family, tokenizer, sequence length, AdamW hyperparameters, reporting conventions, and one-vector communication rule of the 124M study. The larger model has 1,013,786,624 parameters: 18 layers, width 2,048, 32 attention heads, and SwiGLU intermediate width 5,504, with RoPE, RMSNorm, and tied embeddings. An expanded FineWeb training stream preserves the original validation and test streams and excludes their exact document matches. The setting is $K = 8$, $G = 512$, $H = 128$, and sequence length 1,024, matching S9's worker, batch, and synchronization configuration. Each worker uses one H100, microbatch 4, and 16 accumulation steps for an effective local batch of 64. Seed 501 is unchanged. All six selected runs complete 9,999,220,736 tokens (19,072 updates, 149 rounds). The full-horizon

schedule uses 954 warmup updates followed by cosine decay to 10% of peak rate, including during the pilots.

The six methods are FedAvg, tied and generalized SHN-M, generalized RCN-C, generalized RCN-P, and generalized PLN-P. Each inherits its selected S9 coefficients and outer scale α, with only learning rate tuned at the larger scale. The two candidate rate coordinates are $r \in \{0.0004, 0.0008\}$ for the first five methods and $r \in \{0.000625, 0.00125\}$ for PLN-P. Both candidates train to approximately 1B tokens, then to 2B unless one has a sustained validation lead. We select early if its lead is at least 0.02 nats/token at rounds 13 and 15 without shrinking, otherwise by the lower round-30 loss, using the same 10,485,760-token validation set. All six methods selected the higher candidate rate: RCN-C at 1.007B tokens and the other five at 2.013B. Selected runs resume from their checkpoints without resetting training state or further tuning.

Longer-horizon study: fresh 20B runs. The 20B study retains the 10B model, setting, seed, and six selected recipes from Table 10. Each run starts from fresh initialization and optimizer state, with a new full-horizon schedule: 19,998,441,472 tokens (38,144 updates, 298 rounds), 1,908 warmup updates, then cosine decay to 10% of peak rate. There is no further learning-rate or coefficient search or pruning. The expanded FineWeb corpus excludes exact held-out document matches and uses a new training order with seed 501. Validation and test data are unchanged. These are fresh runs with a longer schedule and expanded corpus, not continuations from the 10B endpoints.

Final validation and test each use the same 10,485,760-token streams at both horizons. Their results are reported in Table 10. The two-rate pilot and single-seed, single-setting transfer do not establish optimal learning rates, seed-robust rankings, or a general scaling law.

Smoothed-output evaluation protocol. For the fixed tied and generalized SHN-M selections at both horizons, we reconstruct $q - 0.5v$ in FP32 from the final shared states without retraining or test-based reselection. Both outputs use the original 10,485,760-token test stream, BF16 forward passes, and microbatch 4 on one L4. Re-evaluating the native outputs changes their original eight-H100 scores by less than 9×10^{-6} nats/token. Table 10 reports the smoothed test scores alongside the unchanged primary scores. Smoothed validation losses were not evaluated.

D.2 124M HYPERPARAMETER SEARCH AND SELECTION

We tuned learning rates, Nesterov coefficients, and outer scales through an adaptive search guided by validation loss. The model, data, seed, training budget, and AdamW hyperparameters were fixed. Tables 12 and 13 summarize the values evaluated across settings. Learning rates use the search coordinate r, with the method-specific conversions to the implemented AdamW rate given in Appendix C.

The search began with learning-rate screening at coefficient choices motivated by prior work. We then varied retention and look-ahead independently, expanded promising coefficient regions, and revisited learning rates and outer scales when coefficient changes improved performance. Exploration concentrated initially on S3, S5, and S7, which cover different worker counts and synchronization intervals. Promising configurations were evaluated in other settings and refined separately for each method–setting pair. Additional comparisons tested tied versus generalized coefficients, look-ahead coefficients above one, and equivalent configurations obtained through the formulation maps.

Each completed run used 1,073,741,824 training tokens and seed 501. For each method–setting pair, we selected the configuration with the lowest final validation loss. During screening, some runs were stopped early after persistent validation deficits relative to comparable completed runs. Only full-budget runs were eligible for final selection, and qualifying stopped runs were resumed before subsequent search or transfer decisions. Test evaluation followed validation selection and did not guide tuning.

The search produced 853 completed runs across the eleven settings and 25 early-stopped runs, with counts by family in Table 14. Generalized methods also include eligible tied configurations in their selection pools. Search effort varied across methods and settings, and the tables report pooled values rather than a complete Cartesian grid. The trial manifest records the combinations evaluated.

Table 12: Realized coefficient and outer-scale coverage in the completed 124M search. Values are pooled across settings. Unreached conditional candidates are excluded. Coverage is unequal and uses one seed. $T_n = 0.9^{1/n}$ parameterizes retention. n is not the synchronization interval H.

Family	Retention / outer coefficient	Look-ahead / interpolation / inner coefficient	α
SHN-M	$\mu_x : \{.25, .5, .75, .9, .95\}$	$\mu_y : \{.25, .5, .75, .9, .95, 1, 1.25, 1.5, 1.75, 2\}$	$.7, .8, 1$
RCN-C	$\mu_x : \{.25, .5, .75, .9\}$	$\mu_y : \{.5, .75, .9, .95, 1, 1.25, 1.5, 1.75, 2\}$	$.5, .7, 1$
RCN-P	$\mu_x : \{.25, .5, .75, .9, .95\}$	$\mu_y : \{.5, .75, .9, .95\}$ plus translated values below	$.5, .7, .8, 1$
PLN-C	$\mu_x : \{.5, .75, .9, .95, .99\}$	$\mu_y : \{.75, .9, .95, .99, 1, 1.5, 2\}$	1
PLN-P	$\mu_x : \{.5, T_2, T_4, T_8, T_{32}\}$	$\mu_y : \{.5, .6, .7, .8, .9, .95\}$	1
TTN-C	$\mu^{\mathrm{r}} : \{.5, .75, .9, .95\}$	$\mu^{\ell} : \{0, .05, .1, .25, .5, .75, .9\}$	$.7, 1$

Table 13: Search-coordinate peak learning rates covered across settings and coefficient recipes in the completed 124M search. Each family covers a subset of the grouped core grid. The exact trial manifest records all combinations.

Families	Core / pooled rates	Additional targeted rates
FedAvg, SHN-M, RCN-C/P	.0001, .0004, .0008, .0016, .0032, .0064	.0002 (SHN-M/RCN-P), .00112, .0024 (RCN-C)
PLN-C, TTN-C	.000025, .0001, .0004, .0008, .0016, .0032	.0064 (PLN-C)
PLN-P	.0005, .001, .00125, .0025, .005, .01, .02	—

The additional RCN-P interpolation values are $\{\frac{6}{11}, \frac{4}{7}, \frac{9}{14}, \frac{2}{3}, \frac{5}{7}, \frac{7}{9}\}$, obtained by mapping SHN-M coefficients through Equation (25).

D.3 Validation and test consistency

Table 15 gives the validation losses used to select the checkpoints evaluated on test in Table 7. The winning method agrees between splits in all eleven settings, and the complete method ordering agrees in ten. The sole reversal is a near tie in S7: TTN trails tied SHN-M by 3.35×10^{-5} on validation but leads by 0.00060 on test. Neither wins the setting. This agreement concerns held-out data for fixed checkpoints, not robustness across training seeds.

D.4 Coefficient, learning-rate, and outer-scale ablations

All losses and gaps in this subsection are final validation quantities. Selected recipes may differ in more than one hyperparameter unless a comparison explicitly holds them fixed.

Table 8 summarizes selected coefficients, and Appendix E gives full recipes and gaps to alternatives.

D.4.1 When independent coefficients improve on tied recipes

Coefficient choices can reverse a method's comparison with plain steps. In S4, tied SHN is 0.0339 worse than plain, whereas generalized SHN is 0.1212 better. Generalization also improves the best tied loss by 0.0242 in S2 and 0.0298 in S5. Generalized pools include tied candidates, so strict improvements by unequal-coefficient recipes are the relevant evidence, subject to unequal search effort.

For RCN-C, tied recipes improve on plain by 0.0301 and 0.0401 in S6/S8, while generalized recipes improve by 0.0688 and 0.0962. All four selected trials use peak LR 0.0016. Both generalized winners use $(\mu_x, \mu_y) = (0.5, 0.9)$ and $\alpha = 1$, whereas their tied comparisons use $\alpha = 0.7$. Their gains of 0.0386 and 0.0561 over tied RCN-C combine coefficient and outer-scale changes.

In the more extensively searched S7 setting, generalizing SHN, RCN-C, and PLN-C improves their best tied losses by 0.00637, 0.04680, and 0.75931 nats/token, respectively. The selected SHN recipes share searched LR and α. RCN-C shares searched LR but increases α from 0.7 to 1. At fixed RCN-C pair $(0.5, 0.9)$ and searched LR 0.0016, that outer-scale control improves S7 from 3.73768 to 3.70215 and reverses its comparison with plain, so the gain cannot be assigned to coefficient decoupling alone.

Table 14: Realized full-budget search effort in the completed search. Synchronous references and short diagnostics are excluded. Generalized/tied reporting reuse does not add a trial. The unequal counts reflect adaptive allocation rather than matched tuning compute.

Implementation family	Completed grid trials	Pruned trials
FedAvg	40	3
SHN-M	164	1
RCN-C	212	0
RCN-P	111	0
PLN-C	106	8
PLN-P	115	7
TTN-C	105	6
Total	853	25

Table 15: Final validation losses used to select the 124M recipes evaluated on test in Table 7. Each cell minimizes final validation loss over that method–setting pool. The 110 cells use 108 distinct checkpoints. Validation and test each contain 10,485,760 tokens. AVG weights the eleven settings equally. Bold marks the lowest unrounded loss per setting and average. Generalized pools include tied trials. All results use seed 501 and unequal adaptive tuning coverage.

METHOD	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	AVG
FedAvg	3.7827	3.4997	3.6167	3.7760	3.8681	3.9188	3.6932	3.7316	3.8336	3.8947	**4.0612**	3.7888
SHN-M tied	3.6789	3.4714	3.5789	3.6542	3.9021	3.8953	3.7423	3.6976	3.8680	3.8500	4.1305	3.7699
SHN-M gen.	3.6789	3.4714	3.5546	3.6524	3.7470	3.8655	3.6406	3.6912	3.7666	3.8334	4.1079	3.7281
RCN-C tied	3.6958	3.4701	3.5722	3.6848	3.8502	3.9568	3.6630	3.7489	3.7935	3.9208	4.1665	3.7748
RCN-C gen.	3.6754	3.4676	**3.5423**	3.6455	3.7737	3.9439	**3.6244**	3.7022	**3.7374**	3.8420	4.1024	3.7324
RCN-P gen.	**3.6718**	**3.4640**	3.5497	**3.6435**	**3.7415**	**3.8636**	3.6368	**3.6774**	3.7663	**3.8254**	4.0847	**3.7204**
PLN-C tied	3.8419	3.7873	3.9194	4.1308	4.2747	4.3144	4.3362	4.4995	4.4905	4.7354	5.1821	4.3193
PLN-C gen.	3.7916	3.4782	3.6438	3.7913	3.8567	3.9210	3.6999	3.7402	3.8290	3.9022	4.0797	3.7940
PLN-P gen.	3.7035	3.4682	3.5815	3.7036	3.8575	3.9076	3.7071	3.7321	3.8512	3.8845	4.1772	3.7795
TTN-C	3.6825	3.5170	3.6182	3.6515	3.9105	4.0256	3.7447	3.6976	3.7723	3.8472	4.2426	3.7918

For PLN-C, both use $\alpha = 1$, but the generalized trial uses searched LR 0.0016 versus 0.0004 for the tied trial, and the coefficient-dependent conversion changes the native AdamW rate as well. The larger PLN-C gain therefore combines coefficient and rate choices.

Learning rates can change method rankings. Keeping SHN-M's S4 pair $(0.5, 1.75)$ and $\alpha = 0.8$, increasing searched LR from 0.0004 to 0.0008 and then 0.0016 lowers loss from 3.8009 to 3.7491 and 3.7470, overtaking RCN-C's 3.7737. The translated RCN-P recipe improves from 3.7955 to 3.7415 at 0.0008, but rises to 3.7518 at 0.0016. RCN-P therefore wins S4 at a lower selected rate than SHN-M. In S6, raising searched LR from 0.0008 to 0.0016 improves both matched SHN-M and RCN-P recipes to 3.6406 and 3.6368, without overtaking RCN-C. The same rate increase worsens the corresponding S8 recipes. Increasing the S6 rate again to 0.0032 worsens SHN-M and RCN-P to 4.2939 and 4.1334, respectively. Generalized PLN-C's S1 loss falls from 3.6442 to 3.5010 and then 3.4782 at the unchanged pair $(0.5, 0.9)$ as searched LR rises to 0.0016 and 0.0032. This beats plain's 3.4997, but increasing the rate again to 0.0064 worsens loss to 5.2823. Plain itself improves at LR 0.0032 in S2 (3.6347 to 3.6167) and S3 (3.7889 to 3.7760), narrowing the acceleration gaps. RCN-C's fixed S3/S8 recipes also favor 0.0016 over both 0.0008 and 0.0032.

D.4.2 When does look-ahead above one help?

Look-ahead above one can improve performance, but its benefit depends on the construction and the outer multiplier α. DiLoCo selected an outer learning rate of 0.7 and Nesterov momentum of 0.9 for its experiments (Douillard et al., 2023). This outer-scale choice motivates our $\alpha = 0.7$ comparisons. Here $\mu_y > 1$ increases classical extrapolation or modern correction independently of the retained momentum $\mu_x < 1$, so it does not mean that the retained state is multiplied by a coefficient above one.

RCN-C benefits at the DiLoCo outer scale, but often prefers a larger scale. At fixed $\alpha = 0.7$, the best tested RCN-C recipe has $\mu_y > 1$ in seven settings: S0/S2/S3/S4/S7/S8/S9. Matched coefficient probes establish that this is not solely a learning-rate effect: at $\mu_x = 0.75$ and searched

LR 0.0016, increasing μ_y from 1 to 1.75 improves S0 from 3.68650 to 3.67542, S2 from 3.56044 to 3.55194, and S3 from 3.67450 to 3.66146. In S4, holding $\mu_x = 0.5$, $\alpha = 0.7$, and the same LR fixed, increasing look-ahead from 1 to 1.5 improves 3.83604 to 3.82253. These fixed-scale gains do not identify the best joint recipe: allowing $\alpha = 1$ and look-ahead 0.9 gives still lower losses in eight settings, including all three RCN-C setting wins. For example, the best $\alpha = 0.7$, above-one recipes reach 3.80407/3.76801 in S4/S8, whereas $(\mu_x, \mu_y) = (0.5, 0.9)$ with $\alpha = 1$ reaches 3.77367/3.73738. Only S0 retains an above-one look-ahead in its final RCN-C selection, and its advantage over the $\alpha = 1$, look-ahead 0.9 alternative is just 0.00050. The preferred look-ahead therefore depends on outer scale.

SHN-M and translated RCN-P retain useful larger look-ahead. In S4 ($G = 256, K = 8, H = 128$), the selected SHN-M look-ahead coefficient remains above one after learning-rate tuning. At fixed SHN-M retention 0.5, $\alpha = 0.8$, and searched LR 0.0004, increasing μ_y from 1 to 1.75 improves loss from 3.84151 to 3.80088. Extending to 2 worsens it to 3.81006. The selected LR retains look-ahead 1.75 and reaches 3.74697, while its translated RCN-P pair $(\mu_x^{\mathrm{P}}, \mu_y^{\mathrm{P}}) = (0.5, \frac{7}{9})$ reaches the setting-best loss 3.74149 at $\alpha = 0.8$ and searched LR 0.0008. In S8 ($G = 512, K = 8, H = 32$), the matched SHN-M comparison at retention 0.5, $\alpha = 0.8$, and LR 0.0008 gives a smaller improvement, 3.77193 to 3.76928, when look-ahead increases from 1 to 1.25. At $\alpha = 0.7$, SHN-M attains its selected loss 3.76661. RCN-P selects the translated pair $(0.5, \frac{5}{7})$ at $\alpha = 0.8$, reaching 3.76632 without overtaking RCN-C. By Equation (25), $\mu_y^{\mathrm{P}} = \mu_y^{\mathrm{M}}/(1 + \mu_y^{\mathrm{M}} - \mu_x^{\mathrm{M}})$, so the useful above-one SHN-M look-ahead corresponds to an RCN-P interpolation coefficient below one, rather than to testing $\mu_y^{\mathrm{P}} > 1$. The matched-output study in Section 6.1 separates this algebraic relation from differences in native reported losses.

Increasing α is not a general substitute for larger look-ahead. A 34-trial matched comparison changes only α to 1 relative to fixed reference recipes. Every SHN-M probe (eleven settings) and every RCN-P probe (nine settings, excluding S0/S3 whose reference recipes already use 1) worsens its reference loss. At S4's reference LR 0.0004, retaining SHN-M look-ahead 1.75 while increasing α from 0.8 to 1 worsens loss from 3.80088 to 4.35786. Its translated RCN-P recipe worsens from 3.79546 to 4.20977. Matched $\alpha = 0.7$ controls improve SHN-M in S8 but do not improve SHN-M in S4 or RCN-P in either setting. TTN instead improves in all eleven settings under its $\alpha = 1$ controls. The three additional RCN-C probes improve S1 but worsen S5/S10. The analogous PLN-C look-ahead $\mu_y > 1$ is also useful at its tested $\alpha = 1$: increasing μ_y from 1 to 1.5 at fixed retention 0.5 and searched LR 0.0016 improves S4 from 3.86542 to 3.85666 and S8 from 3.83347 to 3.82896. Both remain selected PLN-C recipes, but neither wins its setting. PLN-P and TTN have no above-one probes that would support an analogous conclusion. These results favor tuning look-ahead jointly with outer scale and learning rate.

D.4.3 Recurring choices and near ties

Generalized RCN-C selects stronger look-ahead than retention in all eleven settings, and selects look-ahead 0.9 and $\alpha = 1$ in eight settings. Only S0 retains look-ahead above one (1.75). S5 selects 1 and S10 selects 0.75. SHN selects tied pairs in two settings and unequal pairs in nine. Its selected look-ahead is 1 in S2/S3/S5, 1.25 in S8, and 1.75 in S4. RCN-P selects numerically tied pairs in four settings. The unequal pairs in the other seven are deliberate translations of selected SHN recipes. PLN-C selects $(0.5, 0.9)$ in nine settings and $(0.5, 1.5)$ in S4/S8. Increasing PLN-C's searched LR from 0.0004 to 0.0016 at unchanged pair $(0.5, 0.9)$ also improves S5/S9 to 3.9210/3.9022, leaving gaps of only 0.0022/0.0075 above plain. PLN-P selects $\mu_x = T_8$ in nine settings, T_{32} in S4, and T_4 in S10, with μ_y ranging from 0.5 to 0.8 and peak LR from 0.001 to 0.005. Its S7 recipe is $(T_8, 0.5)$ at LR 0.0025, with loss 3.73212. On test loss, PLN-P improves on FedAvg in seven settings, while PLN-C is the closest accelerated method in S10 (Table 7).

TTN selects zero additional inner momentum in nine settings, with 0.05 in S0 and 0.1 in S3, and $\alpha = 1$ throughout. At fixed outer pair $(0.5, 0.5)$ and searched LR 0.0016 in S7, removing inner momentum 0.05 improves loss from 3.69920 to 3.69761. Matched S8/S9 controls also favor zero inner momentum, while S3 gains only 0.00063 from retaining it (3.65149 versus 3.65212). Thus the improvements in S7–S9 do not require additional local momentum, and the evidence for its benefit in S3 is small and limited to one seed.

Frequently selected coefficient pairs sometimes differ little in validation loss from alternatives. In S7, SHN's selected $(0.25, 0.9)$ is only 8.6×10^{-6} better than $(0.25, 0.75)$. SHN's S2 preference for look-ahead 1 over 0.9 is separated by only 3.4×10^{-5}. Several S0/S10 pools contain only one completed pair, while the generalized SHN pools in S6/S8 contain twelve and thirteen candidates, respectively, versus 35 in S7. Their selected unequal pairs improve substantially on the initial tied configurations, illustrating the sensitivity of method rankings to sparse tuning coverage. The alternative-pair gaps in Appendix E describe observed separation, not uncertainty across seeds or untested recipes.

Unequal search coverage. PLN-P can represent classical look-ahead coefficients well beyond the tested PLN-C range (Section 4), while TTN varies two timescales but remains tied within each. These differences limit comparisons between families. Local tuning can also substantially improve transferred recipes: in S10, increasing PLN-C's searched LR from 0.0004 to 0.0016 at fixed $(\mu_x, \mu_y) = (0.5, 0.9)$ and $\alpha = 1$ lowers loss from 4.3878 to 4.0797. Nevertheless, FedAvg remains best in S10 after tuning. Poor performance of an initial or transferred recipe alone is therefore insufficient to rule out a family.

D.5 Equivalent formulations in practice

We compare matched recipes, evaluate corresponding outputs, and test whether finite precision separates theoretically equivalent trajectories. Recipe and output comparisons use full-budget 124M validation losses, while precision controls use short training runs and a smaller validation set. These diagnostics do not affect method selection or primary test scores.

D.5.1 Selected recipes and coefficient translation

The results in Section 6.1 select each method's best completed recipe independently. Appendix B.4 gives the exact SHN-M–RCN-P matching conditions and reported-iterate conversion. For SHN-M coefficients $(\mu_x^{\mathrm{M}}, \mu_y^{\mathrm{M}})$, Equations (25) and (70) requires RCN-P coefficients $(\mu_x^{\mathrm{P}}, \mu_y^{\mathrm{P}}) = (\mu_x^{\mathrm{M}}, \mu_y^{\mathrm{M}}/(1 + \mu_y^{\mathrm{M}} - \mu_x^{\mathrm{M}}))$, the same outer multiplier, and the same searched rate under our native-rate normalization. Table 16 compares this required pair with the actual RCN-P selection.

Matching coefficients, rates, and outer scales. The selected coefficient pairs satisfy the translation in eight settings: S0/S2/S4/S5/S6/S7/S8/S9. Five settings also match outer multiplier and searched rate: S2/S5/S6/S7/S9. S0 differs in outer multiplier (0.8 for SHN-M versus 1 for RCN-P). S4 differs in searched rate (0.0016 versus 0.0008), and S8 differs in outer multiplier (0.7 versus 0.8). Thus the native RCN-P advantage in S4 compares different selected learning rates, while the small native RCN-P advantage in S8 compares different outer scales. S6 retains a full match after both families improve at the same higher searched rate. The matches reflect deliberate coverage of equivalent recipes, rather than independent searches rediscovering the map. Only S1/S3/S10 retain different selected translated coefficient pairs, and unequal coverage remains important when interpreting their gaps.

The inverse translation is $\mu_x^{\mathrm{M}} = \mu_x^{\mathrm{P}}$ and $\mu_y^{\mathrm{M}} = (1-\mu_x^{\mathrm{P}})\mu_y^{\mathrm{P}}/(1-\mu_y^{\mathrm{P}})$. For example, RCN-P's selected $(0.5, 7/9)$ in S4 represents modern look-ahead 1.75, while SHN-M's S10 pair $(0.75, 0.9)$ requires RCN-P $(0.75, 18/23)$, rather than its selected tied pair. The source recipe's outer scale and searched rate must accompany either translation.

D.5.2 Aligning reported outputs

The evaluated pools contain 59 pairs of trials matched in translated coefficients, outer multiplier, and searched rate: 38 tied pairs and twenty-one unequal-coefficient comparisons. Five unequal-coefficient comparisons match both current selections. The other sixteen use alternative recipes, including the S4/S8 reference controls and the lower-rate S6 pair. The state relation in Equation (71) distinguishes RCN-P's native endpoint x^{P} from its mapped SHN-M point $\mu_y^{\mathrm{P}} x^{\mathrm{P}} + (1 - \mu_y^{\mathrm{P}}) z^{\mathrm{P}}$. All twenty-one unequal-coefficient RCN-P comparisons score both points on the same final validation set. The main tables consistently retain the native endpoint score. For the currently matched S2/S6 selections, native RCN-P advantages of 0.0049/0.0039 shrink to 0.0004/0.0002 under the mapped output. In S5, mapping reverses the native advantage of 0.0019, leaving SHN-M ahead by 0.0001. In S7/S9, RCN-P's advantages remain 0.0115/0.0087 after mapping, compared with native gaps of

Table 16: Selected generalized recipes and their translation. The required RCN-P pair is mapped from selected SHN-M by Equation (25). A pair match does not by itself match the outer multiplier, searched rate, or primary output. Rates are in units of 10^{-4}. The validation gap is $\Delta_{\mathrm{val}} = L_{\mathrm{P,val}} - L_{\mathrm{M,val}}$. Positive values favor SHN-M. Full means mapped coefficients, α and searched rate coincide. Primary outputs still differ. Values use the same completed single-seed study as Table 7.

Setting	SHN-M	Required RCN-P	Selected RCN-P	$\alpha_{\mathrm{M}}/\alpha_{\mathrm{P}}$	$r_{\mathrm{M}}/r_{\mathrm{P}}$	Δ_{val}	Match
S0	$(0.75, 0.75)$	$(0.75, 0.75)$	$(0.75, 0.75)$	0.8/1	16/16	-0.0071	Pair
S1	$(0.5, 0.5)$	$(0.5, 0.5)$	$(0.75, 0.75)$	0.8/0.7	16/16	-0.0074	–
S2	$(0.5, 1)$	$(0.5, \frac{2}{3})$	$(0.5, \frac{2}{3})$	0.8/0.8	16/16	-0.0049	Full
S3	$(0.75, 1)$	$(0.75, 0.8)$	$(0.75, 0.75)$	0.8/1	16/16	-0.0088	–
S4	$(0.5, 1.75)$	$(0.5, \frac{7}{9})$	$(0.5, \frac{7}{9})$	0.8/0.8	16/8	-0.0055	Pair
S5	$(0.25, 1)$	$(0.25, \frac{4}{7})$	$(0.25, \frac{4}{7})$	0.8/0.8	16/16	-0.0019	Full
S6	$(0.5, 0.9)$	$(0.5, \frac{9}{14})$	$(0.5, \frac{9}{14})$	0.8/0.8	16/16	-0.0039	Full
S7	$(0.25, 0.9)$	$(0.25, \frac{6}{11})$	$(0.25, \frac{6}{11})$	0.8/0.8	16/16	-0.0138	Full
S8	$(0.5, 1.25)$	$(0.5, \frac{5}{7})$	$(0.5, \frac{5}{7})$	0.7/0.8	8/8	-0.0003	Pair
S9	$(0.5, 0.9)$	$(0.5, \frac{9}{14})$	$(0.5, \frac{9}{14})$	0.8/0.8	16/16	-0.0080	Full
S10	$(0.75, 0.9)$	$(0.75, \frac{18}{23})$	$(0.75, 0.75)$	0.7/0.7	8/8	-0.0232	–

0.0138/0.0080. The S4/S8 reference controls at $\alpha = 0.8$ leave mapped residuals of 0.0016/0.0006, but they are not comparisons of both current validation minima. At the matched S4 rate 0.0008, RCN-P's native advantage is 0.0076 and its mapped advantage is 0.0015. At 0.0016, SHN-M instead leads by 0.0049 natively and 0.0169 after mapping. For the unselected S4 $\alpha = 1$ pair, the native gap shrinks from 0.1481 to 0.0045 after mapping, although both recipes perform worse than their lower-scale alternatives.

Evaluating SHN-M's smoothed output. For all twenty-one unequal-coefficient pairs and a separate tied S7 control, we reconstruct $q - \mu_x^{\mathrm{M}} v$ in FP32 from SHN-M's final checkpoint without retraining. Each pair matches the mapped coefficients and rates above, outer scale, setting, model, data, seed, microbatch, and full 1,073,741,824-token training budget. We evaluate SHN-M's native and smoothed outputs and RCN-P's native output on the original 10,485,760-token validation set, using BF16 forward passes and microbatch 32 on one L4. No held-out test data are used. For the selected S2 pair, smoothing accounts for 92.6% of the native gap, reducing it to 0.00036. In S7, it improves SHN-M by 0.00231 but leaves an RCN-P advantage of 0.01148. Across all 21 generalized pairs, native re-evaluation drift is at most 1.21×10^{-5}. Table 17 reports each setting's mean gaps and individual remaining-gap range. The main summary in Table 9 averages the seven setting means with equal weight and gives the range across all 21 pairs. Before rounding, the native gap equals the smoothing gain plus the remaining gap. The high-rate S6 pair ($r = 0.0032$, $\alpha = 0.8$) retains a 0.15664 RCN-P advantage after smoothing, whereas the selected-rate S6 pair ($r = 0.0016$) leaves only 0.00027. Conversely, S4 at $r = 0.0016$, $\alpha = 0.8$ favors smoothed SHN-M by 0.01643. The ranges show both reversals and large residuals that setting means conceal. They reflect recipe variation, not uncertainty across seeds. Smoothing improves 20 of the 21 generalized pairs. The selected S9 recipe instead worsens by 0.00063. The single tied S7 control is excluded from these generalized summaries: its native/smoothed SHN-M losses are 4.23384/4.24502, versus RCN-P 4.24532, showing that smoothing need not improve tied SHN-M either. Neither the mean nor the individual residuals identify the cause of trajectory divergence.

D.5.3 Finite-precision trajectory separation

Precision comparison. We compare translated SHN-M and RCN-P trajectories for 256 local updates (two communication blocks) in S7, using the 124M model, sequence length 1024, $G = 512$, $K = 4$, $H = 128$, and seed 501 on eight H100 GPUs. The SHN-M coefficients are $(0.25, 0.9)$ and the mapped RCN-P coefficients are $(0.25, 6/11)$. Both use $\alpha = 0.8$, searched rate $r = 0.0016$, and the original 2048-update schedule, without shortening its horizon. All cases use the same data, FP32

Settings	Pairs	Native gap	SHN smoothing gain	Remaining gap	Remaining-gap range
S2	2/2	$+0.01405$	$+0.00927$	$+0.00478$	$[+0.00036, +0.00919]$
S4	5/5	$+0.03155$	$+0.03338$	-0.00183	$[-0.01643, +0.00466]$
S5	2/2	$+0.01903$	$+0.01861$	$+0.00042$	$[-0.00006, +0.00090]$
S6	4/4	$+0.04387$	$+0.00418$	$+0.03969$	$[-0.00135, +0.15664]$
S7	2/2	$+0.02063$	$+0.00541$	$+0.01523$	$[+0.01148, +0.01898]$
S8	4/4	-0.00797	$+0.00917$	-0.01714	$[-0.05557, +0.00067]$
S9	2/2	$+0.01379$	$+0.00119$	$+0.01260$	$[+0.00861, +0.01659]$

Table 17: Completed per-setting smoothed-output validation diagnostic. Pairs gives completed/planned generalized comparisons. Entries are within-setting means and the minimum–maximum remaining gap. Native gap is $L_{\mathrm{SHN,native}} - L_{\mathrm{RCN}}$. Smoothing gain is $L_{\mathrm{SHN,native}} - L_{\mathrm{SHN,smooth}}$. Remaining gap is $L_{\mathrm{SHN,smooth}} - L_{\mathrm{RCN}}$, so positive gaps favor RCN-P. All pairs match transformed coefficients, outer scale, searched rate, setting, seed, and training budget. All outputs use the same 10,485,760 validation tokens after 1,073,741,824 training tokens. Recipe variation is not uncertainty across seeds. The tied control and primary test scores are excluded.

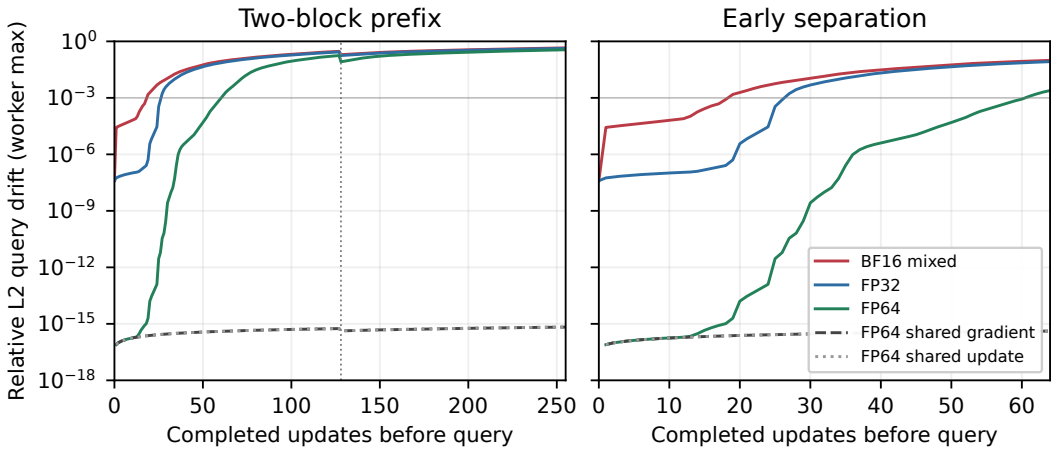


Figure 2: Actual gradient-query drift for translated SHN-M/RCN-P trajectories in S7. Each curve is the maximum relative L2 discrepancy across four logical workers, measured in FP64 before every gradient computation. Exact zeros are omitted from the logarithmic axes. The right panel enlarges the first 64 updates of the same trajectories. The vertical line marks synchronization at update 128. The horizontal line marks 10^{-3} drift. All cases use microbatch 4 and mathematical attention. Higher precision delays separation. Both FP64 controls using shared gradients or shared updates stay near numerical precision.

initialization (promoted exactly for FP64), model constants, and microbatch 4. The controlled BF16, FP32, and FP64 cases use mathematical attention. BF16 uses mixed-precision network operations with FP32 state, whereas FP32 and FP64 use matching precision throughout the model, optimizer state, accumulation, clipping, and reductions. TF32 is disabled, and FP64 is preserved through normalization and cross-entropy. A separate 32-update BF16 control uses the training code's usual attention implementation.

Drift at actual gradient queries. For logical worker k, let $q^{\mathrm{M}}_{t,k}$ and $q^{\mathrm{P}}_{t,k}$ be the actual parameter tensors passed to the network before update t. We measure

$$d_{t,k} = \frac{\|q^{\mathrm{M}}_{t,k} - q^{\mathrm{P}}_{t,k}\|_2}{\frac{1}{2}(\|q^{\mathrm{M}}_{t,k}\|_2 + \|q^{\mathrm{P}}_{t,k}\|_2)}$$

in FP64 before every update. Figure 2 shows $\max_k d_{t,k}$, with t counting completed updates before the next gradient query. Thus $t = 0$ precedes training, and $t = 255$ is the last measured gradient query. We also report the distance after the final update and synchronization at $t = 256$.

Case	First query drift $> 10^{-3}$	Post-sync drift at update 256	Smoothed val. SHN − RCN-P
BF16 mixed	19	0.4362	$+0.01108$
FP32	27	0.4003	-0.00559
FP64	61	0.2866	-0.00171
FP64 shared gradient	—	5.67×10^{-16}	$+0.00000$
FP64 shared update	—	5.79×10^{-16}	$+0.00000$

Table 18: Completed 256-update S7 precision controls, with fixed recipes, data, schedule, and microbatch. The threshold uses actual gradient queries indexed by completed updates. A dash means no crossing during the measured prefix. The final distance is measured after update 256 and synchronization, separately from the last actual query at update 255. Validation gaps compare matched smoothed outputs on 131,072 tokens, evaluated uniformly in FP64. Positive values favor RCN-P. These short, single-seed controls are excluded from full-budget rankings.

Local rounding and accumulated feedback. After one update, query discrepancies are 2.70×10^{-5}, 5.62×10^{-8}, and 7.82×10^{-17} in BF16, FP32, and FP64, respectively. Higher precision delays substantial divergence, but does not prevent it: the last actual gradient-query distances are 0.4505, 0.4248, and 0.3575. To separate fresh rounding error from accumulated drift, we start both formulations from equivalent states at FP64 reference updates 0, 128, and 256. At each checkpoint, we round to the tested precision, use identical Adam states, and measure the difference after one independent update. At those three reference checkpoints, one-step drift ranges from 1.28×10^{-5} to 2.96×10^{-5} in BF16, 5.62×10^{-8} to 6.20×10^{-8} in FP32, and 7.86×10^{-17} to 8.28×10^{-17} in FP64. Higher precision therefore reduces newly introduced discrepancies as well as delaying accumulated drift. Initial query drift is zero in FP64 and approximately 3.98×10^{-8} in the lower-precision cases, reflecting the finite-precision representation of the equivalent initial query.

Shared-gradient and shared-update controls. Repeated gradient computations at identical SHN queries match exactly at the sampled updates 0 and 128 in all three precisions. Supplying the same gradients to both FP64 trajectories, or supplying the same mapped applied updates, keeps final query discrepancies at 5.67×10^{-16} and 5.79×10^{-16}, respectively, with zero measured aligned-validation gap. For the independent cases, we evaluate both smoothed outputs at block boundaries using FP64 mathematical attention on a fixed 131,072-token validation prefix. No held-out test data are used. The final SHN-minus-RCN-P gaps are $+0.01108$, -0.00559, and -0.00171 for BF16, FP32, and FP64: the loss ordering does not consistently favor RCN-P. BF16 with the usual attention implementation also separates, crossing 10^{-3} after 17 updates and reaching a post-update distance of 0.01662 at update 32, compared with the mathematical-attention crossing at update 19. Together, these results support amplification of floating-point perturbations as a mechanism for separating equivalent trajectories, but do not identify which formulation better approximates an exact-arithmetic trajectory or establish an intrinsic stability advantage.

Microbatch-8 and same-method controls. A separate S7 diagnostic uses microbatch 8 for two local blocks, with the original schedule, matched coefficients, and corresponding reported points. Shared-gradient and shared-update replays keep mapped-parameter relative errors near 3.1×10^{-7} and final aligned validation residuals below 10^{-7} when evaluated in FP32, whereas independent BF16 and FP32 trajectories separate substantially. Identical-query gradient repeats match exactly at the sampled checks. With matched microbatch 8, the final mapped relative parameter distances are 0.429 in BF16 and 0.399 in FP32. FP32 delays initial separation but does not prevent its later growth in this prefix. A same-method SHN-M control given only the original initial rounding perturbation produces comparable final distances, 0.437 and 0.395, so a change of formulation is not necessary for this sensitivity. Together, the controls support numerical feedback and output choice as distinct contributors to the observed gaps. The close agreement with shared gradients or updates argues against a large recurrence error in this tested recipe. Determining whether either formulation has a systematic full-budget advantage requires repeated full-budget trials.

E SELECTED COEFFICIENTS AND LEARNING RATES FOR EVERY LM SETTING

The following tables report final validation losses and expand Table 8 to all ten reporting methods and all eleven settings. Every row identifies the same validation-selected trial as the corresponding cell of Table 7, using the same completed search and selection rule. Generalized pools include tied trials, and repeated trials across reporting rows are not independent evidence. The counts describe completed full-budget trials, excluding pruned or unfinished runs. These are descriptive per-setting selections, not the frozen coefficient-transfer choices or held-out test results.

Coefficient pairs are (μ_x, μ_y) for SHN/RCN/PLN and $(\mu^{\mathrm{r}}, \mu^{\ell})$ for TTN's outer and inner timescales, each internally tied. $T_n = 0.9^{1/n}$. The implemented peak AdamW rate is $\eta = r/(1-\mu_y)$ for RCN-P, $\eta = r(1-\mu_x)$ for PLN-C, $\eta = r(1-\mu^{\ell})$ for tied TTN, and $\eta = r$ otherwise. Equal numerical pairs across formulations therefore need not mean equal effective look-ahead or update scale.

For each reporting pool, Δ_{pair} compares the selected pair with the best completed trial using a different coefficient pair, allowing the available learning rate and α to differ. A small gap means the selected pair performs similarly to the best tested alternative. A large gap does not establish robustness to untested coefficients, learning rates, or seeds. "–" means that no alternative coefficient pair was completed, not that the selected pair is insensitive to tuning. The denominator in N/M counts distinct pairs. Differences in r or α increase the trial count N but not the pair count M. Bold marks the lowest validation loss across all methods in each setting. These tables report validation-selected recipes, not test losses.

Translated RCN-P interpolation coefficients μ_y^{P} are displayed as fractions when their decimal expansions do not terminate.

Table 19: Plain: validation-selected recipes by setting.

SETTING	PAIR	α	r	η	VAL. LOSS	Δ_{pair}	N/M
S0	–	1	0.0032	0.0032	3.78268	–	4/0
S1	–	1	0.0016	0.0016	3.49970	–	4/0
S2	–	1	0.0032	0.0032	3.61666	–	5/0
S3	–	1	0.0032	0.0032	3.77605	–	5/0
S4	–	1	0.0016	0.0016	3.86813	–	4/0
S5	–	1	0.0016	0.0016	3.91878	–	4/0
S6	–	1	0.0016	0.0016	3.69318	–	2/0
S7	–	1	0.0016	0.0016	3.73155	–	4/0
S8	–	1	0.0016	0.0016	3.83360	–	3/0
S9	–	1	0.0016	0.0016	3.89474	–	2/0
S10	–	1	0.0016	0.0016	**4.06120**	–	3/0

Table 20: SHN-M tied: validation-selected recipes by setting.

SETTING	PAIR	α	r	η	VAL. LOSS	Δ_{pair}	N/M
S0	$(0.75, 0.75)$	0.8	0.0016	0.0016	3.67889	–	4/1
S1	$(0.5, 0.5)$	0.8	0.0016	0.0016	3.47141	0.02679	6/2
S2	$(0.5, 0.5)$	0.8	0.0016	0.0016	3.57886	0.01000	5/2
S3	$(0.75, 0.75)$	0.8	0.0016	0.0016	3.65424	0.05009	12/5
S4	$(0.9, 0.9)$	0.7	0.0004	0.0004	3.90205	0.01944	4/2
S5	$(0.5, 0.5)$	0.8	0.0016	0.0016	3.89529	0.01973	11/5
S6	$(0.75, 0.75)$	0.7	0.0004	0.0004	3.74230	0.00435	5/3
S7	$(0.5, 0.5)$	0.8	0.0016	0.0016	3.69758	0.02169	12/5
S8	$(0.9, 0.9)$	0.7	0.0004	0.0004	3.86796	0.00260	5/3
S9	$(0.5, 0.5)$	0.8	0.0016	0.0016	3.85000	0.10773	3/2
S10	$(0.75, 0.75)$	0.7	0.0004	0.0004	4.13053	0.08961	4/2

Table 21: SHN-M gen.: validation-selected recipes by setting.

SETTING	PAIR	α	r	η	VAL. LOSS	Δ_{pair}	N/M
S0	$(0.75, 0.75)$	0.8	0.0016	0.0016	3.67889	0.00116	8/5
S1	$(0.5, 0.5)$	0.8	0.0016	0.0016	3.47141	0.00600	9/5
S2	$(0.5, 1)$	0.8	0.0016	0.0016	3.55462	3.4e-05	9/5
S3	$(0.75, 1)$	0.8	0.0016	0.0016	3.65236	0.00188	25/13
S4	$(0.5, 1.75)$	0.8	0.0016	0.0016	3.74697	0.05972	13/7
S5	$(0.25, 1)$	0.8	0.0016	0.0016	3.86552	0.00363	23/12
S6	$(0.5, 0.9)$	0.8	0.0016	0.0016	3.64063	0.01581	12/6
S7	$(0.25, 0.9)$	0.8	0.0016	0.0016	3.69121	8.6e-06	35/17
S8	$(0.5, 1.25)$	0.7	0.0008	0.0008	3.76661	0.00532	13/7
S9	$(0.5, 0.9)$	0.8	0.0016	0.0016	3.83339	0.00509	7/5
S10	$(0.75, 0.9)$	0.7	0.0008	0.0008	4.10786	0.02267	10/6

Table 22: RCN-C tied: validation-selected recipes by setting.

SETTING	PAIR	α	r	η	VAL. LOSS	Δ_{pair}	N/M
S0	$(0.75, 0.75)$	0.7	0.0016	0.0016	3.69583	–	2/1
S1	$(0.75, 0.75)$	0.7	0.0016	0.0016	3.47013	0.00779	6/3
S2	$(0.75, 0.75)$	0.7	0.0016	0.0016	3.57216	0.02951	6/3
S3	$(0.75, 0.75)$	0.7	0.0016	0.0016	3.68479	0.05141	8/3
S4	$(0.75, 0.75)$	0.7	0.0016	0.0016	3.85022	0.10453	6/3
S5	$(0.5, 0.5)$	0.7	0.0016	0.0016	3.95676	0.03998	7/3
S6	$(0.75, 0.75)$	0.7	0.0016	0.0016	3.66304	0.10941	6/3
S7	$(0.5, 0.5)$	0.7	0.0016	0.0016	3.74895	0.02711	8/3
S8	$(0.75, 0.75)$	0.7	0.0016	0.0016	3.79351	0.11057	6/3
S9	$(0.5, 0.5)$	0.7	0.0016	0.0016	3.92077	0.03381	5/3
S10	$(0.75, 0.75)$	0.7	0.0004	0.0004	4.16651	–	2/1

Table 23: RCN-C gen.: validation-selected recipes by setting.

SETTING	PAIR	α	r	η	VAL. LOSS	Δ_{pair}	N/M
S0	$(0.75, 1.75)$	0.7	0.0016	0.0016	3.67542	0.00050	11/7
S1	$(0.5, 0.9)$	1	0.0016	0.0016	3.46762	0.00251	13/7
S2	$(0.75, 0.9)$	1	0.0016	0.0016	**3.54227**	0.00967	23/13
S3	$(0.75, 0.9)$	1	0.0016	0.0016	3.64554	0.01592	27/12
S4	$(0.5, 0.9)$	1	0.0016	0.0016	3.77367	0.03039	23/14
S5	$(0.5, 1)$	0.7	0.0016	0.0016	3.94391	0.00056	18/9
S6	$(0.5, 0.9)$	1	0.0016	0.0016	**3.62440**	0.03066	21/10
S7	$(0.5, 0.9)$	1	0.0016	0.0016	3.70215	0.03030	25/12
S8	$(0.5, 0.9)$	1	0.0016	0.0016	**3.73738**	0.03063	26/14
S9	$(0.5, 0.9)$	1	0.0016	0.0016	3.84202	0.05989	16/10
S10	$(0.5, 0.75)$	0.7	0.0016	0.0016	4.10242	0.00266	9/5

Table 24: RCN-P gen.: validation-selected recipes by setting.

SETTING	PAIR	α	r	η	VAL. LOSS	Δ_{pair}	N/M
S0	$(0.75, 0.75)$	1	0.0016	0.0064	**3.67178**	0.01331	6/2
S1	$(0.75, 0.75)$	0.7	0.0016	0.0064	**3.46403**	0.02902	6/2
S2	$(0.5, \frac{2}{3})$	0.8	0.0016	0.0048	3.54969	0.00718	7/3
S3	$(0.75, 0.75)$	1	0.0016	0.0064	**3.64352**	0.00425	18/6
S4	$(0.5, \frac{7}{9})$	0.8	0.0008	0.0036	**3.74149**	0.06428	10/3
S5	$(0.25, \frac{4}{7})$	0.8	0.0016	0.00373333	**3.86363**	0.07316	10/4
S6	$(0.5, \frac{9}{14})$	0.8	0.0016	0.00448	3.63676	0.08558	12/5
S7	$(0.25, \frac{6}{11})$	0.8	0.0016	0.00352	**3.67742**	0.04874	22/10
S8	$(0.5, \frac{5}{7})$	0.8	0.0008	0.0028	3.76632	0.10008	8/3
S9	$(0.5, \frac{9}{14})$	0.8	0.0016	0.00448	**3.82541**	0.05894	6/3
S10	$(0.75, 0.75)$	0.7	0.0008	0.0032	4.08468	0.09635	6/3

Table 25: PLN-C tied: validation-selected recipes by setting.

SETTING	PAIR	α	r	η	VAL. LOSS	Δ_{pair}	N/M
S0	$(0.75, 0.75)$	1	0.0016	0.0004	3.84193	–	2/1
S1	$(0.75, 0.75)$	1	0.0004	0.0001	3.78734	0.83902	3/2
S2	$(0.75, 0.75)$	1	0.0016	0.0004	3.91941	0.88738	3/2
S3	$(0.75, 0.75)$	1	0.0004	0.0001	4.13077	0.78805	7/3
S4	$(0.75, 0.75)$	1	0.0004	0.0001	4.27465	0.82152	3/2
S5	$(0.75, 0.75)$	1	0.0004	0.0001	4.31439	0.92299	8/3
S6	$(0.75, 0.75)$	1	0.0004	0.0001	4.33616	1.06376	5/2
S7	$(0.75, 0.75)$	1	0.0004	0.0001	4.49948	1.00732	9/4
S8	$(0.75, 0.75)$	1	0.0004	0.0001	4.49050	0.98917	3/2
S9	$(0.75, 0.75)$	1	0.0004	0.0001	4.73544	0.88181	3/2
S10	$(0.75, 0.75)$	1	0.0004	0.0001	5.18214	–	2/1

Table 26: PLN-C gen.: validation-selected recipes by setting.

SETTING	PAIR	α	r	η	VAL. LOSS	Δ_{pair}	N/M
S0	$(0.5, 0.9)$	1	0.0032	0.0016	3.79164	0.05029	5/2
S1	$(0.5, 0.9)$	1	0.0032	0.0016	3.47820	0.30914	8/3
S2	$(0.5, 0.9)$	1	0.0016	0.0008	3.64381	0.27560	4/3
S3	$(0.5, 0.9)$	1	0.0016	0.0008	3.79127	0.29782	15/6
S4	$(0.5, 1.5)$	1	0.0016	0.0008	3.85666	0.00200	10/6
S5	$(0.5, 0.9)$	1	0.0016	0.0008	3.92097	0.32086	15/6
S6	$(0.5, 0.9)$	1	0.0016	0.0008	3.69986	0.63629	8/3
S7	$(0.5, 0.9)$	1	0.0016	0.0008	3.74017	0.58529	21/10
S8	$(0.5, 1.5)$	1	0.0016	0.0008	3.82896	0.00450	10/6
S9	$(0.5, 0.9)$	1	0.0016	0.0008	3.90222	0.83322	5/3
S10	$(0.5, 0.9)$	1	0.0016	0.0008	4.07972	1.10242	5/2

Table 27: PLN-P gen.: validation-selected recipes by setting.

SETTING	PAIR	α	r	η	VAL. LOSS	Δ_{pair}	N/M
S0	$(T_8, 0.7)$	1	0.005	0.005	3.70352	0.00532	4/3
S1	$(T_8, 0.8)$	1	0.005	0.005	3.46816	0.00272	7/3
S2	$(T_8, 0.8)$	1	0.005	0.005	3.58154	0.00424	7/3
S3	$(T_8, 0.8)$	1	0.005	0.005	3.70358	0.00292	19/9
S4	$(T_{32}, 0.8)$	1	0.005	0.005	3.85746	0.00489	9/4
S5	$(T_8, 0.5)$	1	0.001	0.001	3.90762	0.00304	16/7
S6	$(T_8, 0.7)$	1	0.0025	0.0025	3.70709	0.02779	8/4
S7	$(T_8, 0.5)$	1	0.0025	0.0025	3.73212	0.02690	22/10
S8	$(T_8, 0.7)$	1	0.0025	0.0025	3.85117	0.01452	6/3
S9	$(T_8, 0.6)$	1	0.0025	0.0025	3.88452	0.00089	11/5
S10	$(T_4, 0.5)$	1	0.0025	0.0025	4.17719	0.04646	6/4

Table 28: TTN-C: validation-selected recipes by setting.

SETTING	PAIR	α	r	η	VAL. LOSS	Δ_{pair}	N/M
S0	$(0.75, 0.05)$	1	0.0016	0.00152	3.68246	0.01385	4/2
S1	$(0.9, 0)$	1	0.0004	0.0004	3.51696	0.17679	5/3
S2	$(0.9, 0)$	1	0.0004	0.0004	3.61822	0.24825	5/3
S3	$(0.75, 0.1)$	1	0.0016	0.00144	3.65149	0.00063	23/9
S4	$(0.9, 0)$	1	0.0004	0.0004	3.91049	0.37981	5/3
S5	$(0.9, 0)$	1	0.0004	0.0004	4.02565	0.05903	13/5
S6	$(0.9, 0)$	1	0.0004	0.0004	3.74474	0.41359	5/3
S7	$(0.5, 0)$	1	0.0016	0.0016	3.69761	0.00159	27/11
S8	$(0.75, 0)$	1	0.0016	0.0016	3.77233	0.01271	7/5
S9	$(0.5, 0)$	1	0.0016	0.0016	3.84723	0.00215	8/5
S10	$(0.9, 0)$	1	0.0004	0.0004	4.24255	0.07452	3/2