PIPETEMPO: STAGEWISE ACCUMULATION FOR ASYNCHRONOUS PIPELINE-PARALLEL TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

In GPU-rich language-model pretraining, large synchronous batches can improve hardware utilization but leave fewer optimizer updates within a fixed token budget. We show that more frequent asynchronous updates can improve model quality in this regime, despite using stale gradients. We introduce PipeTempo, which sets each pipeline stage's accumulation interval from its local delay. Later stages use smaller batches and update more often, enabling more frequent updates than PipeDream-2BW while limiting local staleness to one update and retaining at most two weight versions per stage. We generalize the convergence analysis of stochastic conditional-gradient updates with momentum to coupled pipeline stages with arbitrary fixed accumulation intervals. Under an expected μ-Kurdyka–Łojasiewicz condition, we prove convergence for both PipeTempo and PipeDream, preserving the synchronous method's leading $\widetilde{O}(\varepsilon^{-3})$ stochastic complexity for expected loss gap ε. Language-model pretraining experiments with Muon on FineWeb-Edu show lower validation loss than synchronous training, PipeDream, and PipeDream-2BW at matched token budgets in the large-batch regime.

1 INTRODUCTION

In this work, we show that *asynchronous optimization can improve large-scale training in GPU-rich settings*: when synchronous batches are too large, the benefit of more frequent updates can outweigh the cost of stale gradients. We consider a setting where a large GPU cluster is available and the goal is to pretrain a language model of a given architecture and size on a *fixed token budget* as quickly as possible, while preserving model quality. Spreading a fixed amount of work across more GPUs leaves less computation per GPU. Using each GPU efficiently as the cluster grows can therefore require more work in parallel, which in turn means processing more tokens at the same time. Synchronous training combines these tokens into one optimizer step, increasing the batch size.

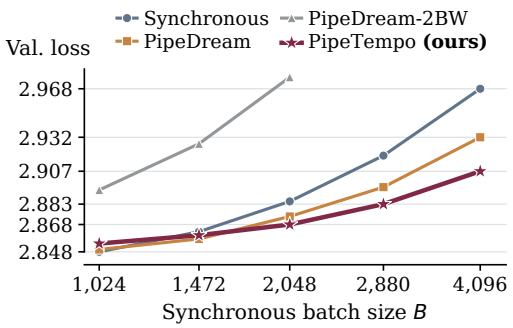

Figure 1: Final validation loss of SmolLM2-135M after 2.71B tokens, for pipeline depth $S = 8$. B counts sequences per synchronous optimizer update; PipeDream and PipeTempo use microbatches of $B/8$ sequences. Asynchronous methods use weight prediction (Chen et al., 2022). PipeDream-2BW results at larger batch sizes are omitted for readability ($B = 4096$, loss $= 3.155$).

Larger batches, however, do not remain useful indefinitely. Beyond a *critical batch size*, increasing the batch gives diminishing reductions in the number of optimizer steps needed to reach a target loss (McCandlish et al., 2018; Zhang et al., 2025; Merrill et al., 2025). Under a fixed token budget, excessively large batches can lead to worse final loss even after tuning the learning rate (Bergsma et al., 2025; Li et al., 2025). This happens because larger batches leave fewer optimizer steps within the same budget, and the reduction in gradient noise may no longer compensate for the loss of updates. Despite considerable effort to

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.

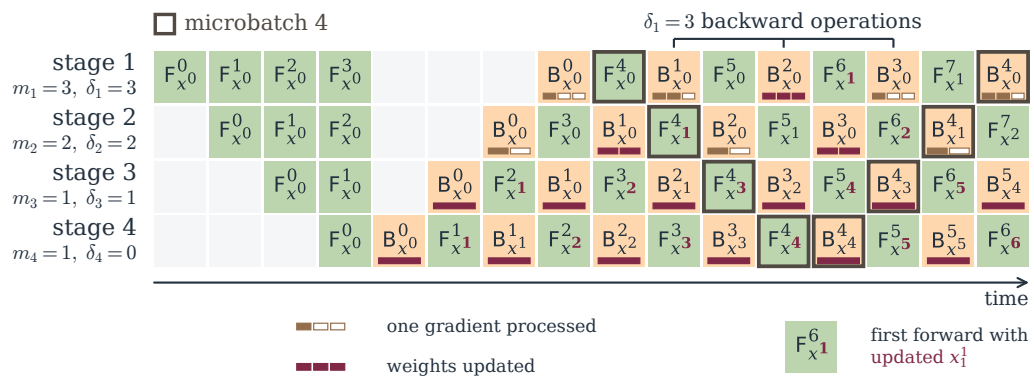

Figure 2: PipeTempo with accumulation intervals $(3, 2, 1, 1)$. F^t and B^t denote the forward and backward of microbatch t; x^k is a stage-local weight version. Counting the backwards between the outlined F^4 and B^4 gives $\delta_s = S - s$; the bracket marks B^1, B^2, B^3 at stage 1. Segmented indicators count gradients since the last update; full indicators mark updates. Setting all $m_s = 1$ recovers PipeDream with the same operation order, but an update after every backward and correspondingly different version labels.

develop optimizers that tolerate larger batches (You et al., 2017; 2020), this remains a challenge, including for Muon (Jordan et al., 2024), which has recently gained popularity for language-model pretraining (Essential AI et al., 2025; Kimi Team et al., 2026). This limitation also appears in theory: for stochastic conditional-gradient methods closely aligned with Muon, Islamov et al. (2026) derive a convergence bound that stops improving and eventually worsens as batch size grows under a fixed token budget.

The batch needed to use a large GPU cluster efficiently can exceed the range that is useful for optimization. Within a given synchronous setup, keeping the batch smaller can instead mean lower GPU utilization, limiting how much the available hardware can accelerate training. Recent reports from practitioners suggest that this tradeoff is already shaping large-scale training decisions. For example, the Falcon-H1 report describes limiting batch-size growth to balance throughput against convergence (Zuo et al., 2025). Similar considerations appear in other recent training reports (Anthony et al., 2025; Ajroldi et al., 2026; Cheon et al., 2026).

This tension raises a practical question: can we keep the GPUs well utilized without making each update batch too large? Asynchronous optimization offers a way to do so (Recht et al., 2011). Instead of waiting to combine all concurrent work into one batch, it applies completed gradients while the remaining computations continue. This allows more frequent updates with smaller batches, at the cost of some gradients becoming stale before they are applied.

We study this approach in pipeline-parallel training. PipeDream, a pioneering asynchronous pipeline-parallel method (Narayanan et al., 2019), uses a one-forward-one-backward (1F1B) schedule: in steady state, each stage alternates forward and backward passes and updates after every microbatch's backward pass. The forward–backward operation order is illustrated in Figure 2. Other microbatches continue through the pipeline while that stage updates, so training does not repeatedly stop to drain and refill the pipeline. This allows each stage to update using a single microbatch rather than accumulating a large batch before every step.

However, this aggressive reduction in batch size comes at the cost of increasing gradient staleness. For S stages numbered from input to output, stage s performs $\delta_s = S - s$ other backward passes between a microbatch's forward and backward in steady state, as illustrated in Figure 2. Since every backward triggers an update, the gradient is applied to weights that are already δ_s local updates newer. This staleness grows with pipeline depth S. Frequent updates also come with a memory cost: preserving forward–backward consistency requires storing up to $\delta_s + 1$ weight versions per stage.

PipeDream-2BW (Narayanan et al., 2021a) addresses these staleness and memory costs while retaining the same 1F1B operation order. It accumulates $m \geq S$ microbatches before each update, using the same window at every stage. Together with coordinated weight-version selection, this limits

gradient staleness to one update and requires only two weight versions per stage, independently of pipeline depth. Zmushko et al. (2026) show that Muon can closely approach synchronous training quality under a one-step delay, but their batch-size sweeps also show that the gap grows at larger batch sizes. However, accumulating at least S microbatches at every stage gives up the small update batches that motivated our turn to asynchrony, leaving the original batch-size problem unresolved. In our fixed-token-budget comparison, PipeDream-2BW's final validation loss increases with batch size, and its gap to PipeDream widens (Figure 1).

Our goal is to move closer to PipeDream's update frequency while retaining PipeDream-2BW's bounds on staleness and weight storage. With the microbatch size fixed, this requires accumulating fewer microbatches. The stages' different positions suggest how to do this. The first stage must wait for a microbatch to travel through the rest of the network and back. By contrast, the last stage can run its backward immediately after its forward, so $\delta_S = 0$. Yet PipeDream-2BW makes both stages wait for the same number of gradients, unnecessarily increasing the last stage's batch per update. This leads to our design question: *How small can each stage's update batch be while limiting its local staleness to one update?*

We introduce PipeTempo, where each stage *updates at its own tempo*: stage s uses the accumulation interval $m_s = \max\{1, \delta_s\}$. Under ordered pipeline execution with weight stashing, this is the shortest fixed interval that limits local staleness to one update. Each stage averages its accumulated gradients and then calls the optimizer once; later stages use smaller batches and update more often. The last stage updates after every microbatch, as in PipeDream, while earlier stages accumulate enough to control their staleness and weight-storage cost. Each backward uses weights at most one local update old. Figure 2 shows this schedule for four stages. PipeTempo reduces the update batch wherever the pipeline permits it.

Contributions. **(i) Smaller update batches with controlled staleness.** We introduce PipeTempo (Algorithm 1), which sets each stage's accumulation interval from its local delay so that later stages use smaller batches and update more often (Figure 2). It enables more frequent updates than PipeDream-2BW (Narayanan et al., 2021a) while limiting local staleness to one update and retaining at most two weight versions per stage (Proposition 1). The accumulation rule can be used with any stage-local optimizer.

(ii) A general convergence theory for stagewise accumulation. We generalize the convergence analysis of synchronous stochastic conditional-gradient (SCG) updates with momentum (Pethick et al., 2025; Islamov et al., 2026) to coupled pipeline stages with different delays and arbitrary fixed accumulation intervals. Under smoothness, unbiased gradients with bounded variance, and an expected μ-Kurdyka–Łojasiewicz (μ-KL) condition at the gradient reads, Theorem 1 guarantees convergence. Its PipeTempo and ordered, weight-stashed PipeDream specializations have the same convergence order up to constants and match synchronous SCG's leading $\widetilde{O}(\varepsilon^{-3})$ stochastic complexity for expected loss gap ε.

(iii) Smaller update batches can outweigh the cost of staleness. In language-model pretraining with Muon, PipeTempo outperforms synchronous training and PipeDream+EF at the largest tested batches, with matched token budgets and sequences in flight. With selective weight prediction, it recovers roughly half the loss gap between large-batch synchronous training and synchronous training at its optimal batch size. We introduce a zero-delay reference that keeps PipeTempo's stagewise batches and update frequencies but removes gradient delay, allowing us to measure its effect on training quality. We also ablate accumulation weights, update schedules, and weight prediction.

2 Related Work

Our analysis builds on stochastic conditional-gradient (SCG) methods with momentum (Pethick et al., 2025). Under a μ-KL condition, Islamov et al. (2026) derive bounds that predict how the benefits of larger batches saturate and eventually reverse under a fixed token budget. Their language-model experiments support these predicted regimes and stepsize scalings. This empirical agreement motivates extending the SCG framework to asynchronous pipeline training.

Gradient accumulation already plays a role in controlling pipeline staleness. ADL uses uniform accumulation and provides convergence guarantees for SGD (Zhuang et al., 2021). AMDP combines

accumulation with synchronized replicas across multiple pipeline directions. Its analysis covers SGD, momentum SGD, and Adam-type updates under a one-step delayed-gradient model, but not LMO-based methods (Chen et al., 2026). PACI uses accumulation and local flow control to retain one weight copy per stage; its theory bounds forward–backward weight inconsistency but does not establish optimization convergence (Elam et al., 2026). PipeTempo sets each stage's accumulation interval from its own delay, enabling more frequent updates at later stages while preserving matching forward and backward weights.

Other pipeline convergence analyses cover Adam-type updates (Jung et al., 2026) and randomized stale-block SGD (Ilin & Richtárik, 2026). These guarantees do not cover SCG updates with unequal accumulation intervals under ordered 1F1B execution. We jointly analyze such updates at mixed-version reads under an expected μ-KL condition, retaining synchronous SCG's leading stochastic convergence order. Appendix A discusses further related methods.

3 PipeTempo

We consider the stochastic optimization problem

$$\min_{x\in\mathcal{X}} F(x), \qquad F(x)=\mathbb{E}_\xi[f(x;\xi)], \qquad \mathcal{X}=\prod_{s=1}^{S}\mathcal{X}_s,$$

where $f(x;\xi)$ is the loss on microbatch ξ. The model parameters $x=(x_1,\dots,x_S)$ are partitioned into S blocks, with $x_s\in\mathcal{X}_s$ assigned to pipeline stage s on a separate GPU.

Pipeline execution and weight versions. We use the ordered 1F1B schedule in Figure 2. Independent microbatches ξ_t, indexed by $t=0,1,\dots$, pass through all stages; each stage processes forwards and backwards in microbatch order. The number of other backwards between microbatch t's forward and backward at stage s is $\min\{t,\delta_s\}$, where

$$\delta_s = S-s. \tag{1}$$

Accumulation determines how many of those backwards cause weight updates.

Every forward uses the latest weights, which are retained for its matching backward as in PipeDream (Narayanan et al., 2019). Microbatches using the same version share its buffer. Suppose stage s updates after every $m_s\geq 1$ gradients, and let x_s^k denote its weights after k local updates. Microbatch t reads the weights

$$y_s^t = x_s^{v_s(t)}, \qquad v_s(t)=\left\lfloor \tfrac{\max\{t-\delta_s,0\}}{m_s}\right\rfloor, \qquad g_s^t=\nabla_s f(y^t;\xi_t), \tag{2}$$

where $y^t=(y_1^t,\dots,y_S^t)$. For example, the outlined microbatch in Figure 2 reads $y^4=(x_1^0,x_2^1,x_3^3,x_4^4)$. Each backward reuses its forward's weights, so together they compute $\nabla f(y^4;\xi_4)$, with its blocks becoming available at different times.

Choosing the accumulation interval. Local staleness counts a stage's updates between a forward and its matching backward. We count buffers beyond the current weights, excluding activations, optimizer state, and workspace.

Proposition 1 (Local staleness and weight storage)**.** *For any fixed integer $m_s\geq 1$, $\lceil\delta_s/m_s\rceil$ bounds both local staleness and the number of additional weight buffers at stage s. Both bounds are tight.*

Proof. Before backward t, local staleness is $\lfloor t/m_s\rfloor - v_s(t)\leq\lceil\delta_s/m_s\rceil$, with equality at $t=\delta_s+m_s-1$. Replaced versions remain needed for at most δ_s backwards. Updates occur every m_s backwards; freeing buffers before coincident updates leaves at most $\lceil\delta_s/m_s\rceil$ old versions. In a continuing run, each version's last forward immediately precedes replacement, giving the full δ_s-backward lifetime and attaining the storage bound after $\lceil\delta_s/m_s\rceil$ updates. □

Both bounds are at most one exactly when $m_s\geq\delta_s$. PipeTempo chooses the shortest interval:

$$\boxed{m_s=\max\{1,\delta_s\}=\max\{1,S-s\}.} \tag{3}$$

Unlike PipeDream-2BW (Narayanan et al., 2021a), however, PipeTempo permits mixed-version reads across stages.

Accumulation and optimizer updates. Stage s averages each window of gradients,

$$\bar{g}_s^k = \frac{1}{m_s} \sum_{j=km_s}^{(k+1)m_s-1} g_s^j, \tag{4}$$

then passes $\bar{g}_s^k$ to its optimizer for one step. This allows a stage-local optimizer to be used without modifying its internal updates. Changing m_s changes the number of microbatches combined in an update; the microbatch size and 1F1B operation order stay the same.

Algorithm 1 PipeTempo with SCG at stage s

Initially $d_s = \bar{g}_s = 0$.
Forward t**:** use $y_s^t = x_s$.
Backward t**:**
  Compute g_s^t using y_s^t.
  $\bar{g}_s \leftarrow \bar{g}_s + g_s^t/m_s$.
  If m_s divides $t+1$:
    $d_s \leftarrow (1-\alpha_s)d_s + \alpha_s \bar{g}_s$.
    $x_s \leftarrow (1-\gamma_s)x_s + \gamma_s \eta \,\mathrm{lmo}_s(d_s)$.
    $\bar{g}_s \leftarrow 0$.

For the convergence analysis, we use stochastic conditional gradient (SCG) (Pethick et al., 2025). Starting from $d_s^0 = 0$, each window updates momentum and weights as follows, for $k \geq 0$:

$$d_s^{k+1} = (1-\alpha_s)d_s^k + \alpha_s \bar{g}_s^k, \tag{5}$$

$$x_s^{k+1} = (1-\gamma_s)x_s^k + \gamma_s \eta \,\mathrm{lmo}_s(d_s^{k+1}). \tag{6}$$

With the layerwise norm choices of Pethick et al. (2025), these updates recover the constrained SCION form, applied here to each stage's accumulated gradient. The SCG framework is closely aligned with Muon (Islamov et al., 2026), which we use in the experiments.

Here $\mathrm{lmo}_s(d) \in \arg\min_{\|u\|_s \leq 1} \langle d, u \rangle$, for a chosen norm $\|\cdot\|_s$ on $\mathcal{X}_s$. The EMA coefficient is $0 < \alpha_s \leq 1$, and $\gamma_s > 0$ is the stage stepsize. The radius $\eta > 0$ controls the norm of the weights, as in Islamov et al. (2026). Equivalently, (6) uses weight-decay coefficient $1/\eta$.

4 Theory

Our main theoretical contribution is a convergence guarantee for the coupled pipeline updates with different accumulation intervals. A stage's gradient depends on weights throughout the network: changes to other stages can alter $\nabla_s F(y^t)$ even while y_s^t stays fixed. Treating each stage as a separate fixed objective therefore does not capture the updates made by the pipeline.

The pipeline fixes $\delta_s = S - s$; our analysis allows any fixed integer accumulations $m_s \geq 1$. Define

$$M = \max_s m_s, \qquad H = \max_s (\delta_s + m_s - 1).$$

Here M is the largest window, and H bounds the combined delay from pipeline execution and accumulation: $t - m_s v_s(t) \leq H$. We consider horizons $T \geq M$.

For the analysis, we linearly interpolate consecutive stage weights:

$$z_s^{km_s+r} = \left(1 - \tfrac{r}{m_s}\right) x_s^k + \tfrac{r}{m_s} x_s^{k+1}, \qquad r = 0, \ldots, m_s, \tag{7}$$

for each complete window $0 \leq k < \lfloor T/m_s \rfloor$. The full sequence $z^t = (z_s^t)_s$ is defined for $0 \leq t \leq K_T := \min_s m_s \lfloor T/m_s \rfloor$. This is an auxiliary sequence used only in the proof, as with virtual iterates in asynchronous optimization (Mania et al., 2017; Stich & Karimireddy, 2019; Mishchenko et al., 2022); the algorithm never computes it.

Assumptions. Each $\mathcal{X}_s$ is a finite-dimensional real inner-product space. We use the product norm $\|x\| = \max_s \|x_s\|_s$ and its dual $\|h\|_* = \sum_s \|h_s\|_{s,*}$. Let $\rho > 0$ be a norm-equivalence constant satisfying $\|h\|_* \leq \rho \|h\|_2$, where $\|\cdot\|_2$ is induced by the product inner product.

Assumption 1 (Smoothness)**.** F is differentiable, and there exists $L > 0$ such that

$$\|\nabla F(x) - \nabla F(y)\|_* \leq L \|x - y\| \qquad \text{for all } x, y \in \mathcal{X}.$$

Assumption 2 (Expected μ-KL)**.** Let $F_* := \inf_{x \in \mathcal{X}} F(x) > -\infty$. There exists $\mu > 0$ such that

$$\mathbb{E}[\|\nabla F(y^t)\|_*] \geq \mu \mathbb{E}[F(y^t) - F_*], \qquad 0 \leq t < K_T.$$

This is weaker than the global μ-KL condition of Islamov et al. (2026): it is required only in expectation at the gradient-evaluation points.

Assumption 3 (Unbiasedness and bounded variance)**.** The stochastic gradient is unbiased and has bounded variance:

$$\mathbb{E}_\xi[\nabla f(x;\xi)] = \nabla F(x), \qquad \mathbb{E}_\xi[\|\nabla f(x;\xi) - \nabla F(x)\|_2^2] \le \sigma^2$$

for all $x \in \mathcal{X}$ and some $\sigma \ge 0$.

Theorem 1 (Iteration complexity; Proof in Appendix B)**.** *Fix x^0 and $0 < \varepsilon < F(x^0) - F_*$. Set $\gamma_s = m_s\gamma$ and $\alpha_s = 1-(1-\alpha)^{m_s}$, with the base parameters in (15) from Appendix B. Take η equal to the logarithmic term in (14); assume this term is at least $2\left\|x^0\right\|$. If Assumptions 1–3 hold for the resulting run, the final stage weights satisfy $\mathbb{E}[F((x_s^{\lfloor T/m_s \rfloor})_{s=1}^S) - F_*] \le \varepsilon$ after a sufficient number of microbatches*

$$\boxed{T = O\left(\max\left\{\frac{L(H+M)}{\mu^2\varepsilon}, \quad \frac{L\rho^2\sigma^2}{\mu^4\varepsilon^3}\right\}\log\frac{4(F(x^0)-F_*+4\left\|\nabla F(x^0)\right\|_*/\mu)}{\varepsilon}\right).} \tag{8}$$

The same μ must satisfy Assumption 2 across the accuracy-dependent runs. The appendix gives the explicit budget for every radius satisfying (14). Stage s makes $\lfloor T/m_s \rfloor$ weight updates; T includes incomplete final windows.

Proof sketch. We adapt the containment and momentum-tracking arguments of Islamov et al. (2026) to the coupled stage updates. The interpolation places all stages on a common microbatch clock. We expand each stage's momentum into its constituent gradients, separating the error from zero initialization, gradient drift, and sampling noise. Since interpolated points can depend on later samples, we prove descent pathwise and center the noise only at the actual gradient reads. Bounds on weight movement connect these reads to the interpolation, so the expected μ-KL condition yields loss contraction up to the tracking error. We retain the decay of the initialization error when choosing the parameters. Finally, a bound on the movement between each stage's last complete window and the common interpolation endpoint transfers the guarantee to the final weights. □

Setting $m_s = 1$ and $\delta_s = S - s$ recovers ordered, weight-stashed PipeDream with SCG and $\alpha_s = \alpha$. For $S \ge 2$, PipeDream has $(M, H) = (1, S-1)$ and PipeTempo has $(M, H) = (S-1, 2S-3)$. Thus their schedule terms $H + M$ are S and $3S-4$, respectively, and their stochastic terms coincide. Under common valid problem constants and initial objective gap, both have the same convergence rate up to constants. For $S = 1$, both have $m_1 = M = 1$ and $H = 0$. Its leading stochastic term also matches the $\widetilde{O}(\varepsilon^{-3})$ rate of synchronous SCG (Islamov et al., 2026): for fixed problem constants, delay and accumulation enter only the lower-order ε^{-1} term.

5 Experiments

5.1 Setup and baselines

We pretrain SmolLM2 models (Allal et al., 2025) with 135M and 360M parameters on FineWeb-Edu (Penedo et al., 2024) at a context length of 1024. While our theory analyzes SCG, our experiments follow Zmushko et al. (2026) and use Muon with Nesterov momentum (Jordan et al., 2024), which they found robust to 1-step gradient delay. For PipeTempo, we use stage learning rates $\gamma_s = \gamma\sqrt{m_s}$ and a momentum coefficient shared across stages within each run. The main setting trains the 135M model at the Chinchilla budget of 20 tokens per parameter (Hoffmann et al., 2022); the 360M model at the same ratio and the 135M model at $5\times$ the budget separate the effects of model size and token budget. For each setting, we locate the optimal synchronous batch $B^\star$ and compare methods on a grid from $2B^\star$ to $8B^\star$. The global batch B counts sequences in flight: synchronous training uses B per update, while asynchronous schedules use microbatches of B/S sequences. We use $S \in \{8, 16\}$ stages of contiguous layers; $S = 16$ matches the pipeline depth used by DeepSeek-V3 (Liu et al., 2024). For every method, batch and depth, we sweep the base learning rate on a $\sqrt{2}$-spaced ladder and report the best final evaluation loss. Appendix C gives the full recipe and per-method momentum.

Baselines. We compare against synchronous training and the 2 asynchronous schedules that bracket PipeTempo: PipeDream-2BW (Narayanan et al., 2021a), which accumulates a common window of S microbatches at every stage, and PipeDream (Narayanan et al., 2019), which updates after every

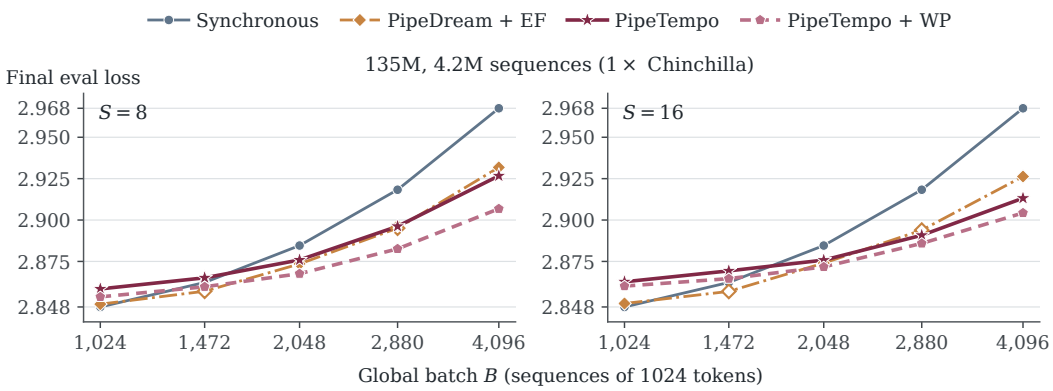

Figure 3: Final evaluation loss against the global batch B for the 135M model at the Chinchilla budget, $S = 8$ (left) and $S = 16$ (right). Every point is the best over a learning-rate ladder. Filled markers are interior optima. The same comparison for the 2 other settings is in Figure 6 (Appendix C).

microbatch and keeps $\delta_s + 1$ weight versions at stage s. Both are run with the update-level Error Feedback correction of Zmushko et al. (2026), which improved both schedules in that work; we write PipeDream-2BW+EF and PipeDream+EF. Both Ajanthan et al. (2025) and Zmushko et al. (2026) identify the momentum coefficient as the hyperparameter that matters most under gradient delay, so we tune it on a grid for every baseline (Appendix C). Ajanthan et al. (2025) run PipeDream with NAdam (Dozat, 2016) and a retuned β_1; Muon already uses Nesterov momentum and a tuned μ, and Error Feedback further improves PipeDream with Muon (Zmushko et al., 2026), so PipeDream+EF covers this configuration. PipeMare (Yang et al., 2019) avoids weight stashing to save memory. SpecTrain, XPipe, and PipeOptim (Chen et al., 2019; Guan et al., 2020; 2025) use weight prediction, closely related to Error Feedback, but do not guarantee exact forward–backward consistency. We therefore use PipeDream+EF, combining delay correction with weight stashing to ensure matching forward and backward weights.

5.2 PipeTempo at large batch

Weight prediction. Besides plain PipeTempo we evaluate it with the weight-prediction heuristic of SAPipe (Chen et al., 2022). Let G_s^k be the gradient that stage s accumulates over its window k and u_s its optimizer step, with optimizer state omitted from the notation. Consider a stage with 1-step delay: at weights x_s^k, the latest available gradient is G_s^{k-1}, while G_s^k has not yet arrived. The stage first takes the real update $x_s^{k+1} = x_s^k - \gamma_s\, u_s(G_s^{k-1})$. The idea is to predict where the weights would be without the 1-step lag. Since G_s^k is still unavailable, we reuse G_s^{k-1} for 1 additional, temporary optimizer step, $\tilde{x}_s^{k+2} = x_s^{k+1} - \gamma_s\, u_s(G_s^{k-1})$. We compute the next gradient at these predicted weights, then discard $\tilde{x}_s^{k+2}$ and continue the actual trajectory from x_s^{k+1}. The optimizer state is also restored to its value after the real update, so the prediction does not add a persistent optimizer step. For a common 1-step delay this is closely related to the Error Feedback correction of Zmushko et al. (2026), which applies the same idea at the level of updates. In PipeTempo we apply weight prediction only to later stages, which have shorter accumulation intervals. Applying it to early stages can instead make training worse; Section 5.3 examines this choice through stage-subset ablations.

Results. Figure 3 shows the comparison for the 135M model, and Figure 6 for all 3 settings. Plain PipeTempo reaches lower loss than synchronous training from about $4B^\star$ upward in every setting and at both depths, with the gap growing as the batch increases. PipeDream+EF, the strongest asynchronous baseline, is level with plain PipeTempo up to $4B^\star$ and slightly ahead on the 360M model, but falls behind from $5.6B^\star$, with a clear gap at $8B^\star$. Weight prediction further lowers PipeTempo's loss at the largest batches in every setting; it helps throughout the grid at $S = 8$, while its effect is smaller at $S = 16$ and limited to the largest batches on the 360M model. Increasing depth from $S = 8$ to $S = 16$ leaves PipeTempo's loss at $8B^\star$ essentially unchanged, with an improvement for the 135M model at the Chinchilla budget.

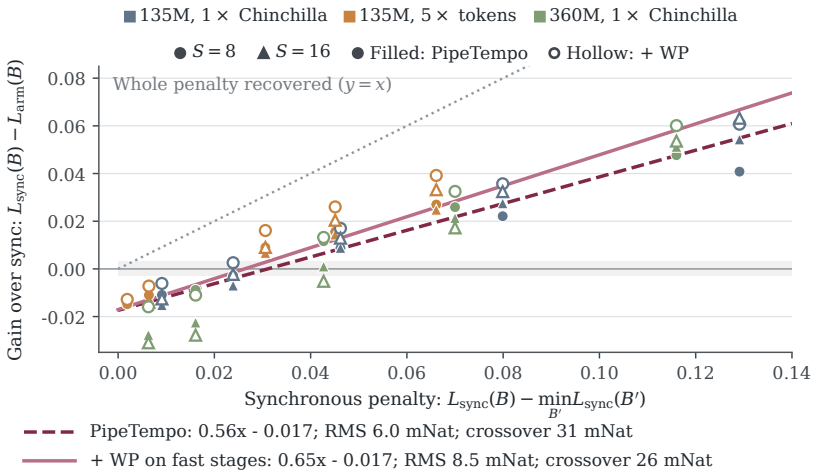

Figure 4: Gain over synchronous training at the same batch, $L_{\text{sync}}(B) - L(B)$, against the penalty synchronous training pays for that batch, $L_{\text{sync}}(B) - \min_{B'} L_{\text{sync}}(B')$. 1 point per cell of Figure 6: all 3 settings and both depths; filled markers are PipeTempo, hollow markers PipeTempo with weight prediction. The diagonal would mean recovering the whole penalty. The lines are least-squares fits.

Note that at $2B^\star$ both variants are still behind synchronous training, slightly at $S = 8$ and more clearly at $S = 16$. This is expected: PipeTempo was never meant to compete there. Near $B^\star$ the synchronous baseline has nothing to lose from its batch, while the asynchronous schedule still pays for staleness; moreover, its per-stage batches are up to S times smaller than B, so at $S = 16$ and the smallest batch of the grid the last stage updates on microbatches $8\times$ smaller than the synchronous optimum, which hurts in its own right.

Figure 4 summarizes the grid in one plot. For each cell we plot the gain over synchronous training at the same batch against the penalty that synchronous training pays for that batch relative to its own best batch. The points of all 3 settings and both depths lie close to 1 line: the gain of plain PipeTempo is $\approx 0.56\times$ the synchronous penalty minus 0.017, and that of PipeTempo+WP is $\approx 0.65\times$ the penalty minus 0.017. The intercept is the fixed cost of the asynchronous schedule near $B^\star$; the slope is the fraction of every additional nat of synchronous penalty that the schedule gives back. The 2 lines cross 0 at a penalty of 0.026–0.031, which corresponds to about $3B^\star$ in every setting. At $8B^\star$ PipeTempo recovers 41–44% of the synchronous penalty and PipeTempo+WP 49–59%; equivalently, PipeTempo+WP at $8B^\star$ reaches the loss synchronous training reaches at a 1.6–$2.2\times$ smaller batch. These fractions do not decrease with the length of training (the $5\times$ setting has $2.5\times$ more optimizer steps at the same $B/B^\star$), so we state the result in fractions rather than in absolute loss.

5.3 Zero-delay reference

The schedule works, but how much of the gap to an ideal asynchronous method is still open? To measure this we construct a reference that keeps everything about PipeTempo except the delay. At each stage, PipeTempo is PipeDream-2BW with a window of m_s microbatches: the stage accumulates 1 window, then applies it while the next window is already being computed, so each update is 1 window old. The synchronous counterpart of PipeDream-2BW is ordinary synchronous training, which applies each window as soon as it is complete. The *zero-delay reference* does the same at every stage of PipeTempo at once: stage s still accumulates m_s microbatches and updates $(S-1)/m_s$ times as often as stage 1, but at the end of window k it applies the gradient of window k itself rather than that of window $k-1$. For example, at stage 1 in Figure 2, it is as if the forward $F^3_{x^0}$ were computed after the backward $B^2_{x^0}$ and the resulting weight update, using x^1—the same weights that PipeTempo first uses in $F^6_{x^1}$. The reference cannot be executed by a pipeline, since a window's gradient is not available until the window has finished; it is the loss PipeTempo would reach if the delay were free.

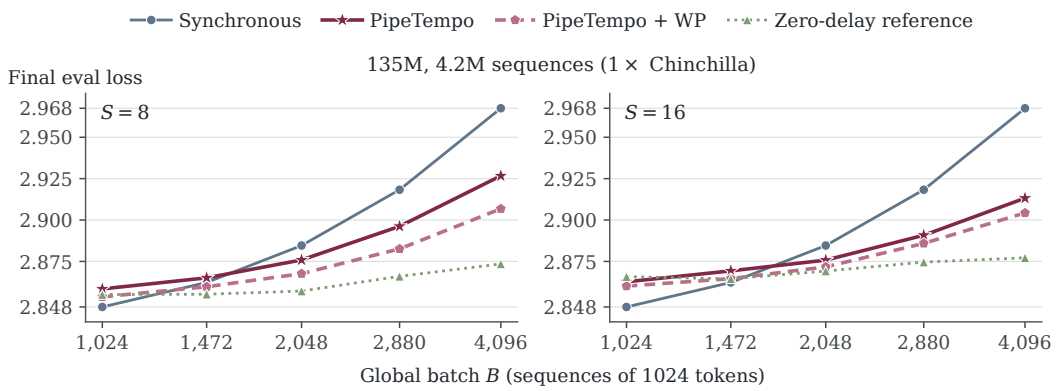

Figure 5: PipeTempo, PipeTempo with weight prediction, and the zero-delay reference against synchronous training, 135M model at the Chinchilla budget. The zero-delay reference keeps PipeTempo's accumulation intervals and per-stage batches but applies each window without delay; it is not executable in a pipeline. The setting with a $5\times$ token budget is in Figure 7 (Appendix C).

Figure 5 places the reference next to PipeTempo and PipeTempo+WP for the 135M model. Near $B^\star$ the reference and PipeTempo coincide within noise: there the delay costs nothing. Above that the reference is below both, and the gap between it and plain PipeTempo, the price of the 1-window delay, grows with the batch. At $8B^\star$ and $S = 8$ weight prediction on the fast stages closes about $1/3$ of the gap between PipeTempo and the reference, and the remaining $2/3$ are larger than everything weight prediction gained. This leaves room to reduce staleness costs, but raises a question: why restrict weight prediction to the later stages?

Table 1: Weight prediction on selected stages, 135M model, $S = 8$. Best final evaluation loss over the learning-rate ladder; Δ is relative to no WP. Stage 8 has no delay.

	$B = 2048$		$B = 4096$	
Stages with WP	Loss	Δ	Loss	Δ
None	2.876	0.000	2.927	0.000
1 only	2.889	+0.013	2.952	+0.024
1–4	2.891	+0.015	2.958	+0.031
4–7	2.868	−0.009	2.907	−0.021
6–7	2.872	−0.005	2.912	−0.015
1–7	2.885	+0.009	2.940	+0.013

Where weight prediction helps. We initially applied weight prediction to all delayed stages, but this made training worse: at $S = 8$ and $B = 4096$, the final loss increased by 0.013. To understand why, we tested prediction on different stage subsets (Table 1). Predicting on stages 1–4 increases the loss by 0.031, with stage 1 alone accounting for most of the harm. In contrast, predicting on stages 4–7 reduces it by 0.021. The same pattern holds at $B = 2048$, with smaller changes. The $S = 16$ sweep shows the same broad pattern (Table 2 in Appendix C). We therefore recommend using WP selectively on later stages; Appendix C.2 provides further empirical analysis.

6 Conclusion

PipeTempo provides an optimizer-independent schedule for asynchronous pipeline training in GPU-rich settings. By adapting accumulation to each stage's delay, it enables more frequent updates while limiting local staleness to one update and retaining at most two weight versions per stage. The schedule can be paired with any stage-local optimizer.

We establish convergence within the SCG framework with momentum, preserving synchronous SCG's leading stochastic rate under the stated assumptions. Experiments with Muon demonstrate the benefits of this schedule in language-model pretraining at large batches and matched token budgets. With selective weight prediction, PipeTempo recovers roughly half the synchronous large-batch loss penalty. Extending these guarantees to Muon and other practical optimizers is a natural direction for future work.

AI USE STATEMENT

We used generative AI tools under the authors' direction to revise and rewrite text based on author-provided drafts and instructions, locate additional relevant references, assist with experimental code, and prepare scientific figures and plots of experimental results. For the theoretical analysis, the authors provided proof strategies and detailed guidance, while AI tools assisted with developing, writing, and checking parts of the proofs. The authors reviewed the AI-assisted text, references, code, figures, and mathematical arguments and take full responsibility for the final manuscript and results.

REFERENCES

Thalaiyasingam Ajanthan, Sameera Ramasinghe, Yan Zuo, Gil Avraham, and Alexander Long. Nesterov method for asynchronous pipeline parallel optimization. In *International Conference on Machine Learning*, 2025.

Niccolò Ajroldi, Diana Alexandra Onutu, Haider Al-Tahan, Jörg Franke, Sampo Pyysalo, Jenia Jitsev, and Aaron Klein. Deriving scaling laws for OpenEuroLLM models: Learning rate, batch size and loss. *arXiv preprint arXiv:2608.28308*, 2026.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, et al. SmolLM2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.

Quentin Anthony, Yury Tokpanov, Skyler Szot, Srivatsan Rajagopal, Praneeth Medepalli, Anna Golubeva, Vasu Shyam, Robert Washbourne, Rishi Iyer, Ansh Chaurasia, Tomas Figliolia, Xiao Yang, Abhinav Sarje, Drew Thorstensen, Amartey Pearson, Zack Grossbart, Jason van Patten, Emad Barsoum, Zhenyu Gu, Yao Fu, and Beren Millidge. Training foundation models on a full-stack AMD platform: Compute, networking, and system design. *arXiv preprint arXiv:2511.17127*, 2025.

Shane Bergsma, Nolan Simran Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Power lines: Scaling laws for weight decay and batch size in LLM pre-training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Chi-Chung Chen, Chia-Lin Yang, and Hsiang-Yun Cheng. Efficient and robust parallel DNN training through model parallelism on multi-GPU platform. *arXiv preprint arXiv:1809.02839*, 2019.

Ling Chen, Houming Wu, and Wenjie Yu. AMDP: Asynchronous multi-directional pipeline parallelism for large-scale models training. In *Forty-third International Conference on Machine Learning*, 2026.

Yangrui Chen, Cong Xie, Meng Ma, Juncheng Gu, Yanghua Peng, Haibin Lin, Chuan Wu, and Yibo Zhu. SAPipe: Staleness-aware pipeline for data parallel DNN training. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 17981–17993. Curran Associates, Inc., 2022. doi: 10.52202/068431-1307.

Sung Jun Cheon, Jaekyung Cho, Seongho Choi, Hyunjun Eun, Seokhwan Jo, Jaehyun Jun, Minsoo Kang, Jin Kim, Jiwon Kim, Minsang Kim, et al. A.X K1 technical report. *arXiv preprint arXiv:2601.09200*, 2026.

Timothy Dozat. Incorporating Nesterov momentum into Adam. *Proceedings of the 4th International Conference on Learning Representations, Workshop Track*, 2016.

Itay Elam, Eliron Rahimi, Avi Mendelson, and Chaim Baskin. Breaking the bubble: Asynchronous pipeline parallel training with bounded weight inconsistency. *arXiv preprint arXiv:2606.07881*, 2026.

Essential AI, Ishaan Shah, Anthony M. Polloreno, Karl Stratos, Philip Monk, Adarsh Chaluvaraju, Andrew Hojel, Andrew Ma, Anil Thomas, Ashish Tanwer, Darsh J Shah, Khoi Nguyen, Kurt Smith, Michael Callahan, Michael Pust, Mohit Parmar, Peter Rushton, Platon Mazarakis, Ritvik Kapila, Saurabh Srivastava, Somanshu Singla, Tim Romanski, Yash Vanjani, and Ashish Vaswani. Practical efficiency of Muon for pretraining. *arXiv preprint arXiv:2505.02222*, 2025.

Louis Fournier and Edouard Oyallon. Cyclic data parallelism for efficient parallelism of deep neural networks. *arXiv preprint arXiv:2403.08837*, 2024.

Lei Guan, Wotao Yin, Dongsheng Li, and Xicheng Lu. XPipe: Efficient pipeline model parallelism for multi-GPU DNN training. *arXiv preprint arXiv:1911.04610*, 2020.

Lei Guan, Dongsheng Li, Yongle Chen, Jiye Liang, Wenjian Wang, and Xicheng Lu. PipeOptim: Ensuring effective 1F1B schedule with optimizer-dependent weight prediction. *IEEE Transactions on Knowledge and Data Engineering*, 37(5):2831–2845, 2025. doi: 10.1109/TKDE.2025.3543225.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Ivan Ilin and Peter Richtárik. Demystifying pipeline parallelism: First theory for PipeDream. *arXiv preprint arXiv:2606.03498*, 2026.

Rustem Islamov, Roman Machacek, Aurelien Lucchi, Tony Silveti-Falls, Eduard Gorbunov, and Volkan Cevher. On the role of batch size in stochastic conditional gradient methods. In *Forty-third International Conference on Machine Learning*, 2026.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Hyunji Jung, Sungbin Shin, and Namhoon Lee. Mitigating staleness in asynchronous pipeline parallelism via basis rotation. In *Forty-third International Conference on Machine Learning*, 2026.

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2026.

Anastasiia Koloskova, Sebastian U Stich, and Martin Jaggi. Sharper convergence guarantees for asynchronous SGD for distributed and federated learning. *Advances in Neural Information Processing Systems*, 35:17202–17215, 2022.

Houyi Li, Wenzhen Zheng, Qiufeng Wang, Hanshan Zhang, Zili Wang, Shijie Xuyang, Yuantao Fan, Zhenyu Ding, Haoying Wang, Ning Ding, Shuigeng Zhou, Xiangyu Zhang, and Daxin Jiang. Predictable scale: Part I, step law – optimal hyperparameter scaling law in large language model pretraining. *arXiv preprint arXiv:2503.04715*, 2025.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *SIAM Journal on Optimization*, 27(4):2202–2229, 2017.

Artavazd Maranjyan, Alexander Tyurin, and Peter Richtárik. Ringmaster ASGD: The first asynchronous SGD with optimal time complexity. In *International Conference on Machine Learning*, 2025.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

William Merrill, Shane Arora, Dirk Groeneveld, and Hannaneh Hajishirzi. Critical batch size revisited: A simple empirical approach to large-batch language model training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Konstantin Mishchenko, Francis Bach, Mathieu Even, and Blake E Woodworth. Asynchronous SGD beats minibatch SGD under arbitrary delays. *Advances in Neural Information Processing Systems*, 35:420–433, 2022.

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R Devanur, Gregory R Ganger, Phillip B Gibbons, and Matei Zaharia. PipeDream: generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pp. 1–15, 2019.

Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. Memory-efficient pipeline-parallel DNN training. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7937–7947. PMLR, 18–24 Jul 2021a.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on GPU clusters using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–15, 2021b.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The FineWeb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. Training deep learning models with norm-constrained LMOs. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 49069–49104. PMLR, 13–19 Jul 2025.

Penghui Qi, Xinyi Wan, Guangxing Huang, and Min Lin. Zero bubble (almost) pipeline parallelism. In *The Twelfth International Conference on Learning Representations*, 2024.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. HOGWILD!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in Neural Information Processing Systems*, 24, 2011.

Abdurakhmon Sadiev, Artavazd Maranjyan, Ivan Ilin, and Peter Richtárik. Ringmaster LMO: Asynchronous linear minimization oracle momentum method. *arXiv preprint arXiv:2605.18174*, 2026.

Sebastian Stich and Sai Praneeth Karimireddy. The error-feedback framework: Better rates for SGD with delayed gradients and compressed communication. *arXiv preprint arXiv:1909.05350*, 2019.

Nouamane Tazi, Ferdinand Mom, Haojun Zhao, Phuc Nguyen, Mohamed Mekkouri, Leandro Werra, and Thomas Wolf. The ultra-scale playbook: Training LLMs on GPU clusters, 2025. URL https://huggingface.co/spaces/nanotron/ultrascale-playbook.

Bowen Yang, Jian Zhang, Jonathan Li, Christopher Ré, Christopher R Aberger, and Christopher De Sa. PipeMare: Asynchronous pipeline parallel DNN training. *arXiv preprint arXiv:1910.05124*, 2019.

PengCheng Yang, Xiaoming Zhang, Wenpeng Zhang, Ming Yang, and Hong Wei. Group-based interleaved pipeline parallelism for large-scale DNN training. In *International Conference on Learning Representations*, 2022.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *International Conference on Learning Representations*, 2020.

Hanlin Zhang, Depen Morwani, Nikhil Vyas, Jingfeng Wu, Difan Zou, Udaya Ghai, Dean Foster, and Sham M. Kakade. How does critical batch size scale in pre-training? In *The Thirteenth International Conference on Learning Representations*, 2025.

Huiping Zhuang, Zhenyu Weng, Fulin Luo, Toh Kar-Ann, Haizhou Li, and Zhiping Lin. Accumulated decoupled learning with gradient staleness mitigation for convolutional neural networks. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 12935–12944. PMLR, 18–24 Jul 2021.

Philip Zmushko, Egor Petrov, Nursultan Abdullaev, Khrushchev Mikhail, and Samuel Horváth. One-step gradient delay is not a barrier for large-scale asynchronous pipeline parallel LLM pretraining. In *Forty-third International Conference on Machine Learning*, 2026.

Jingwei Zuo, Maksim Velikanov, Ilyas Chahed, Younes Belkada, Dhia Eddine Rhayem, Guillaume Kunsch, Hakim Hacid, Hamza Yous, Brahim Farhat, Ibrahim Khadraoui, Mugariya Farooq, Giulia Campesan, Ruxandra Cojocaru, Yasser Djilali, Shi Hu, Iheb Chaabane, Puneesh Khanna, Mohamed El Amine Seddik, Ngoc Dung Huynh, Phuc Le Khac, Leen AlQadi, Billel Mokeddem, Mohamed Chami, Abdalgader Abubaker, Mikhail Lubinets, Kacper Piskorski, and Slim Frikha. Falcon-H1: A family of hybrid-head language models redefining efficiency and performance. *arXiv preprint arXiv:2507.22448*, 2025.

A Further related work

We expand the comparison in Section 2 and discuss further asynchronous data-parallel methods and pipeline schedules.

Scope of pipeline convergence analyses. PipeMare studies stability in scalar quadratic models (Yang et al., 2019), while Ajanthan et al. (2025) analyze fixed-delay Nesterov updates for a fixed convex objective. These analyses do not jointly track the changing weights of coupled pipeline stages. ADL provides full-network convergence guarantees for SGD with uniform accumulation (Zhuang et al., 2021). AMDP analyzes a one-step delayed-gradient model and extends its SGD analysis to momentum and adaptive preconditioning (Chen et al., 2026). PACI's formal guarantee bounds forward/backward weight inconsistency rather than optimization error (Elam et al., 2026). Jung et al. (2026) give joint stationarity bounds for Adam-type updates, including stage-dependent delays and a separate momentum extension for uniform delay; gradients are evaluated at past full-model iterates. Ilin & Richtárik (2026) allow mixed-version reads but fix the read before uniformly sampling the stage and minibatch. Their convergence theorem covers this randomized update model, rather than deterministic 1F1B. Our analysis follows ordered pipeline execution with SCG and arbitrary fixed accumulation intervals, which can differ across stages.

Asynchronous data-parallel optimization. Workers in asynchronous data-parallel training compute gradients while a shared model continues to change. HOGWILD! removes locking between shared-memory updates (Recht et al., 2011); later work studies convergence and time complexity under gradient delays (Koloskova et al., 2022; Mishchenko et al., 2022; Maranjyan et al., 2025). Ringmaster LMO studies asynchronous LMO-based optimization with momentum (Sadiev et al., 2026). These methods concern workers computing full-model gradients. Our pipeline analysis instead follows coupled parameter blocks with mixed-version reads and different update frequencies, and measures progress in processed microbatches.

Pipeline scheduling and asynchronous updates. PipeDream combines updates during pipeline execution with weight stashing (Narayanan et al., 2019); PipeDream-2BW controls delay and weight storage through accumulation, while PipeDream-Flush is synchronous (Narayanan et al., 2021a). WPipe changes the partition and execution order to combine fresh and one-step-delayed updates (Yang et al., 2022). Other schedules improve utilization by interleaving model chunks (Narayanan et al., 2021b), separating input- and parameter-gradient computations (Qi et al., 2024), or overlapping bidirectional pipelines and communication (Liu et al., 2024). The Ultra-Scale Playbook surveys their practical tradeoffs (Tazi et al., 2025). Cyclic Data Parallelism staggers microbatch computations to smooth activation-memory usage and communication; its CDP-v1 update recovers the PipeDream-2BW recursion in the pipeline setting (Fournier & Oyallon, 2024). PipeTempo chooses stagewise update frequencies within ordered 1F1B execution. Its storage guarantee concerns weight versions; activation memory and optimizer state remain separate costs.

B Convergence proofs

In this appendix, write $x^{\mathrm{out}} = (x_s^{\lfloor T/m_s \rfloor})_s$ for the final weights after all T microbatches finish. An incomplete final accumulation window is discarded without an optimizer step.

The proof of Theorem 1 follows the whole model as its stages update at different times. Bounds on weight movement and momentum error give a descent recurrence along the interpolation z^t. We then transfer the resulting loss bound to the final weights x^{out} returned by the algorithm.

The interpolation is a proof device. An interior point z_s^t uses the next weight update at stage s, which can depend on samples with indices greater than t. We therefore prove descent pathwise before taking expectation, and center the noise only at the causal reads y^j. All endpoints needed for z^{K_T} occur within the run, although faster stages may overwrite them before training ends. The transfer to the returned model means that the implementation need not construct or retain z^{K_T}.

Conventions. The initial model x^0 is fixed. Each LMO uses an exact, fixed measurable choice of minimizer, with $\mathrm{lmo}_s(0) = 0$, and the stochastic-gradient map $(x, \xi) \mapsto \nabla f(x; \xi)$ is jointly

measurable. Matrix inner products are Frobenius inner products. The physical 1F1B schedule has $\delta_s = S - s$; the finite-horizon analysis also allows arbitrary fixed nonnegative integer delays δ_s in the causal read model (2). A physical execution must realize that read order.

We first establish a finite-horizon bound for arbitrary stage stepsizes and stage EMA weights $\alpha_s = 1 - (1-\alpha)^{m_s}$, with $0 < \alpha \le 1/(4M)$. This bound guides the relative stage stepsizes and the parameter choices in the main theorem. The expected KL condition concerns this algorithm's own causal gradient reads. Define the minimum and maximum effective stepsizes per microbatch by

$$\underline{\gamma} := \min_s \frac{\gamma_s}{m_s}, \qquad \overline{\gamma} := \max_s \frac{\gamma_s}{m_s}. \tag{9}$$

Recall $M = \max_s m_s$.

Theorem 1 (detailed statement). ***(i) General bound and choice of stage stepsizes.*** *Under Assumptions 1–3, run Algorithm 1 with*

$$0 < \alpha \le \frac{1}{4M}, \qquad \alpha_s = 1 - (1-\alpha)^{m_s}, \qquad \gamma_s > 0, \qquad T \ge M, \qquad \left\|x^0\right\| \le \eta/2.$$

Recall

$$H = \max_s(\delta_s + m_s - 1), \qquad K_T = \min_s m_s \lfloor T/m_s \rfloor.$$

Suppose the stage stepsizes satisfy

$$T\overline{\gamma} \le \frac{1}{2}, \qquad 0 < \mu\eta\underline{\gamma} \le 4. \tag{10}$$

Define the deterministic quantity

$$\begin{aligned} \mathcal{B}_z := & \left(1 - \frac{\mu\eta\underline{\gamma}}{4(1+2\mu\eta\overline{\gamma}H)}\right)^{K_T} \left(F(x^0) - F_*\right) \\ & + \frac{4L\eta\overline{\gamma}^2(1+2\mu\eta\overline{\gamma}H)}{\mu\underline{\gamma}} \left(2 + \frac{25}{2}H + \frac{12}{\alpha}\right) \\ & + \frac{8\sqrt{2}\rho\sigma\overline{\gamma}(1+2\mu\eta\overline{\gamma}H)}{\mu\underline{\gamma}}\sqrt{\alpha} \\ & + \frac{2\eta\overline{\gamma}\left\|\nabla F(x^0)\right\|_*}{\alpha}. \end{aligned} \tag{11}$$

The returned model $x^{\mathrm{out}} = (x_s^{\lfloor T/m_s \rfloor})_s$ satisfies

$$\mathbb{E}[F(x^{\mathrm{out}}) - F_*] \le \left(\sqrt{\mathcal{B}_z} + \sqrt{L/2}\, 2\eta\overline{\gamma}(M-1)\right)^2. \tag{12}$$

The stage stepsizes need not be proportional to m_s. The four terms in $\mathcal{B}_z$ represent contraction of the initial loss, accumulated gradient drift, sampling noise, and the initial momentum error, respectively.

To choose the stage stepsizes, fix $\underline{\gamma}$, α, η, T, the schedule, and the initial model, with the same problem constants valid for the compared runs. Both $\mathcal{B}_z$ and the additional term in (12) are minimized at $\overline{\gamma} = \underline{\gamma}$. Thus choose $\gamma_s = \gamma m_s$, where $\gamma = \underline{\gamma}$. Then

$$\begin{aligned} \mathcal{B}_z = & \left(1 - \frac{\mu\eta\gamma}{4(1+2\mu\eta\gamma H)}\right)^{K_T} \left(F(x^0) - F_*\right) \\ & + \frac{4L\eta\gamma(1+2\mu\eta\gamma H)}{\mu} \left(2 + \frac{25}{2}H + \frac{12}{\alpha}\right) \\ & + \frac{8\sqrt{2}\rho\sigma(1+2\mu\eta\gamma H)}{\mu}\sqrt{\alpha} + \frac{2\eta\gamma\left\|\nabla F(x^0)\right\|_*}{\alpha}. \end{aligned} \tag{13}$$

The proof retains the decay of the initial momentum error when deriving part (ii).

(ii) Selected parameters and iteration complexity. *Fix $0 < \varepsilon < F(x^0) - F_*$ and choose*

$$\eta \geq \max\left\{2\left\|x^0\right\|, \frac{16}{\mu}\log\frac{4(F(x^0) - F_* + 4\left\|\nabla F(x^0)\right\|_* /\mu)}{\varepsilon}\right\}. \tag{14}$$

The initial gradient norm may be replaced by its upper bound $\sqrt{2L(F(x^0) - F_*)}$, established in (41); the prescription therefore does not require an additional gradient query. Use the single prescription

$$\boxed{\begin{aligned} \gamma &= \frac{1}{2T}, \qquad \gamma_s = \gamma m_s, \\ \alpha &= \min\left\{\frac{1}{4M}, \frac{\mu^2\varepsilon^2}{2048\rho^2\sigma^2}\right\}, \qquad \alpha_s = 1 - (1-\alpha)^{m_s}. \end{aligned}} \tag{15}$$

The noise-dependent entry is $+\infty$ when $\sigma = 0$. The assumptions, including the expected inequality at the gradient reads with the stated μ, must hold for this run. If

$$T \geq \left\lceil \max\left\{2M, \frac{8L\eta}{\mu\varepsilon}\left(2 + 12H + \frac{12}{\alpha}\right)\right\}\right\rceil, \tag{16}$$

then

$$\boxed{\mathbb{E}[F(x^{\mathrm{out}}) - F_*] < \left(\sqrt{4/5} + \frac{1}{384}\right)^2 \varepsilon < \varepsilon.} \tag{17}$$

This guarantee concerns the final published weights; no interpolated checkpoint or additional gradients are required. If the logarithmic entry in (14) is at least $2\left\|x^0\right\|$, choose η equal to that entry. Substituting this radius and (15) into (16) gives the sufficient count below, with the same read-based KL constant over the accuracy-dependent runs:

$$\boxed{T = O\left(\max\left\{\frac{L(H+M)}{\mu^2\varepsilon}, \frac{L\rho^2\sigma^2}{\mu^4\varepsilon^3}\right\}\log\frac{4(F(x^0) - F_* + 4\left\|\nabla F(x^0)\right\|_* /\mu)}{\varepsilon}\right).} \tag{8}$$

The logarithm is explicit; for a larger radius, retain η in (16). Stage s makes $\lfloor T/m_s \rfloor$ weight updates. The budget counts all admitted microbatches, including incomplete final windows.

Proof of part (i): the finite-horizon loss bound. We establish containment and momentum tracking before proving descent. At this point, the stage stepsizes need not be proportional to the accumulation intervals; the EMA weights satisfy $\alpha_s = 1 - (1-\alpha)^{m_s}$ with $0 < \alpha \leq 1/(4M)$.

The product norm and its dual. The blockwise dual-norm inequality gives $\langle h, u\rangle \leq \sum_s \|h_s\|_{s,*}$ whenever $\max_s \|u_s\|_s \leq 1$. Conversely, choose a maximizing unit-ball vector in each block, which exists by finite-dimensional compactness. Their tuple attains this upper bound, proving

$$\|h\|_* = \sum_s \|h_s\|_{s,*}, \qquad \|h\|_2^2 = \sum_s \langle h_s, h_s\rangle.$$

The second identity comes from the fixed product inner product and will be used for temporal noise cancellation. For Euclidean or Frobenius block norms, Cauchy–Schwarz gives $\|h\|_* \leq \sqrt{S}\,\|h\|_2$; for other fixed block norms, finite-dimensional norm equivalence supplies ρ. Only $\|h\|_* \leq \rho\,\|h\|_2$ is used below.

Containment and movement. The descent argument needs a margin between the iterates and the boundary of the norm ball, so that the oracle direction can overcome weight decay. The following lemma establishes this margin and bounds how far the model can move while a gradient is delayed.

Lemma 1 (Containment and movement)**.** *Suppose $T \geq M$, $\gamma_s > 0$, $T\overline{\gamma} \leq 1/2$, and $\left\|x^0\right\| \leq \eta/2$. Every available stage publication satisfies*

$$\eta - \left\|x_s^k\right\|_s \geq \frac{\eta}{4}, \qquad km_s \leq T.$$

For $0 \le t < K_T$ and $k_s = \lfloor t/m_s \rfloor$,

$$z_s^{t+1} - z_s^t = \frac{\gamma_s}{m_s}\big(\eta \,\mathrm{lmo}_s(d_s^{k_s+1}) - x_s^{k_s}\big). \tag{18}$$

Consequently, for every admitted sample index $0 \le j < T$,

$$\big\|z^{t+1} - z^t\big\| \le 2\eta\overline{\gamma}, \qquad \big\|y^j - z^t\big\| \le 2\eta\overline{\gamma}(H + |j-t|). \tag{19}$$

All stashed and interpolated models have norm at most $3\eta/4$.

Proof. All statements hold pathwise. The horizon condition gives $0 < \gamma_s \le m_s/(2T) \le 1/2$, so each weight update is a convex combination of its predecessor and a point of norm at most η. Consequently,

$$\eta - \big\|x_s^k\big\|_s \ge (1-\gamma_s)^k(\eta - \big\|x_s^0\big\|_s) \ge \frac{\eta}{2}(1 - k\gamma_s) \ge \frac{\eta}{4}.$$

Here the second inequality is Bernoulli's inequality, and the last uses $k\gamma_s \le T\overline{\gamma} \le 1/2$. Stashed blocks are stored weight versions, while interpolated blocks are convex combinations of consecutive versions; hence both have norm at most $3\eta/4$.

By (7), the two indices $t, t+1$ lie in the same closed stage window $[k_s m_s, (k_s+1)m_s]$, including when $t+1$ is its right endpoint. Both endpoints exist because $t < K_T$. Subtracting the two interpolation values gives $(z_s^{t+1} - z_s^t) = (x_s^{k_s+1} - x_s^{k_s})/m_s$; the weight update then gives (18). This calculation holds for every integer increment along each stage's interpolation. Each block increment has norm at most $2\eta\gamma_s/m_s$, proving the first bound in (19).

For an arbitrary read j, its version $v_s(j) = \lfloor \max\{j - \delta_s, 0\}/m_s \rfloor$ satisfies

$$y_s^j = z_s^{m_s v_s(j)}, \qquad 0 \le j - m_s v_s(j) \le \delta_s + m_s - 1 \le H.$$

The bound follows directly from the floor when $j \ge \delta_s$, and from $v_s(j) = 0$ otherwise. Both $m_s v_s(j)$ and t belong to the available block interval $[0, m_s\lfloor T/m_s \rfloor]$. Telescoping the block increments across all intervening windows yields

$$\big\|y_s^j - z_s^t\big\|_s \le \frac{2\eta\gamma_s}{m_s}|m_s v_s(j) - t| \le \frac{2\eta\gamma_s}{m_s}(H + |j-t|).$$

Taking the maximum over stages proves the second bound in (19). This uses each block's available path; it does not require a full vector z^j when $j > K_T$. □

From the proof trajectory to the returned model. The stages finish their last complete windows at different microbatch counts. Bounding the loss difference between these final weights and the common interpolation endpoint gives a guarantee for the model actually returned.

Lemma 2 (Final-model transfer)**.** *Under Assumption 1 and the hypotheses of Lemma 1, suppose $F_* := \inf_{x\in\mathcal{X}} F(x) > -\infty$. Define*

$$q_s = m_s\lfloor T/m_s \rfloor, \qquad x_s^{\mathrm{out}} = x_s^{\lfloor T/m_s \rfloor} = z_s^{q_s}, \qquad D = 2\eta\overline{\gamma}(M-1).$$

Then $\big\|x^{\mathrm{out}} - z^{K_T}\big\| \le D$ pathwise. For every deterministic $\mathcal{B} \ge 0$ such that $\mathbb{E}[F(z^{K_T}) - F_*] \le \mathcal{B}$,

$$\mathbb{E}[F(x^{\mathrm{out}}) - F_*] \le \Big(\sqrt{\mathcal{B}} + \sqrt{L/2}\, D\Big)^2. \tag{20}$$

No KL assumption at either endpoint is needed.

Proof. Each block interpolation is defined through its own q_s, even when $q_s > K_T$. The proof of Lemma 1 bounds every increment on that individual block path by $2\eta\gamma_s/m_s$. Since $T - M + 1 \le K_T \le q_s \le T$, telescoping gives, pathwise,

$$\big\|x_s^{\mathrm{out}} - z_s^{K_T}\big\|_s \le \frac{2\eta\gamma_s}{m_s}(q_s - K_T), \qquad \big\|x^{\mathrm{out}} - z^{K_T}\big\| \le D.$$

Write $G(x) = F(x) - F_* \ge 0$. Global smoothness and the finite global infimum imply

$$\|\nabla F(x)\|_*^2 \le 2LG(x) \qquad \text{for every } x.$$

Indeed, choose a unit vector attaining the dual pairing and apply the smoothness upper bound in its negative direction with step $\|\nabla F(x)\|_* / L$; comparison with F_* gives the inequality. Applying smoothness again, now from z^{K_T} to x^{out}, yields

$$G(x^{\mathrm{out}}) \le G(z^{K_T}) + \sqrt{2LG(z^{K_T})}\, D + \frac{L}{2} D^2.$$

Take expectations and use Jensen's inequality for the square root to obtain (20). □

Momentum tracking after gradient accumulation. At interpolation index t, each stage contributes the momentum used in its next update. That momentum averages gradients within each completed window, then weights the window averages geometrically. We expand it back into individual gradients so that all stages can still be compared on the same microbatch clock.

Lemma 3 (Momentum tracking)**.** *Under Assumptions 1 and 3 and the hypotheses of Lemma 1, suppose* $0 < \alpha \le 1/(4M)$ *and* $\alpha_s = 1 - (1-\alpha)^{m_s}$. *For every* $0 \le t < K_T$ *and* $k_s = \lfloor t/m_s \rfloor$,

$$\begin{aligned} &\mathbb{E} \sum_s \frac{\gamma_s}{m_s} \left\| d_s^{k_s+1} - \nabla_s F(z^t) \right\|_{s,*} \\ &\qquad \le \overline{\gamma} \left[(1-\alpha)^{t+1} \left\| \nabla F(x^0) \right\|_* + 6L\eta\overline{\gamma} \left(H + \frac{1}{\alpha} \right) + \sqrt{2}\rho\sigma\sqrt{\alpha} \right]. \end{aligned} \tag{21}$$

Proof. Fix $0 \le t < K_T$, write $k_s = \lfloor t/m_s \rfloor$, and let

$$n_s(t) = m_s(k_s+1), \qquad t \le n_s(t) - 1 \le t + m_s - 1, \qquad n_s(t) \le T.$$

Thus the momentum $d_s^{k_s+1}$ uses exactly the gradients with indices $0, \dots, n_s(t)-1$. Some of these indices can exceed t, but every included gradient is computed before the corresponding weight update.

The noise is centered at the actual reads. Let $\mathcal{F}_j = \sigma(\xi_0, \dots, \xi_{j-1})$. The weight version $x_s^{v_s(j)}$ used by the read depends on gradients only through $m_s v_s(j) - 1 \le j - 1$; recursively, y^j is $\mathcal{F}_j$-measurable. Joint measurability permits substitution of this random read: conditional integration first bounds the nonnegative squared error, ensuring integrability for the signed mean identity. Independence of the fresh sample and Assumption 3 give

$$\epsilon^j := \nabla f(y^j; \xi_j) - \nabla F(y^j), \qquad \mathbb{E}[\epsilon^j \mid \mathcal{F}_j] = 0, \qquad \mathbb{E}[\left\|\epsilon^j\right\|_2^2 \mid \mathcal{F}_j] \le \sigma^2. \tag{22}$$

The noise is $\mathcal{F}_{j+1}$-measurable and square-integrable. These conditional identities hold at y^j. They are not applied to z^t, which can depend on later samples (Section 4).

The weight of each gradient. For this proof only, write $q = 1 - \alpha$. Unrolling the EMA from $d_s^0 = 0$ gives

$$\begin{aligned} d_s^{k_s+1} &= \alpha_s \sum_{\ell=0}^{k_s} q^{m_s(k_s-\ell)} \bar{g}_s^\ell = \sum_{j=0}^{T-1} c_{s,j} g_s^j, \\ c_{s,j} &= \begin{cases} (\alpha_s/m_s) q^{m_s(k_s - \lfloor j/m_s \rfloor)}, & 0 \le j < n_s(t), \\ 0, & n_s(t) \le j < T. \end{cases} \end{aligned} \tag{23}$$

The coefficients depend on the fixed index t, not on the samples. They are nonnegative, and the geometric sum over windows gives

$$\sum_{j=0}^{T-1} c_{s,j} = \alpha_s \sum_{\ell=0}^{k_s} (1-\alpha_s)^{k_s-\ell} = 1 - q^{n_s(t)}.$$

The missing mass $q^{n_s(t)}$ is the contribution of the zero initial momentum; we account for it explicitly below.

We next bound these coefficients uniformly over stages. Bernoulli's inequality and $\alpha M \le 1/4$ give $q^M \ge 3/4$. The geometric identity gives $\alpha_s/m_s = \alpha(1 + q + \cdots + q^{m_s-1})/m_s \le \alpha$. For $j \le t$, writing t and j as multiples of m_s plus their remainders shows

$$m_s(k_s - \lfloor j/m_s \rfloor) \ge t - j - (m_s - 1).$$

Combining this with $q^{-(m_s-1)} \le 4/3$ gives the first bound below. For $j > t$ with positive weight, j is in the same window as t, so $\lfloor j/m_s \rfloor = k_s$ and $j \le t + M - 1$. Thus

$$\max_s c_{s,j} \le \begin{cases} \dfrac{4\alpha}{3} q^{t-j}, & 0 \le j \le t, \\ \alpha, & t < j \le t + M - 1, \\ 0, & j > t + M - 1. \end{cases} \tag{24}$$

Numerical sums may be extended beyond $T-1$ using these upper bounds; this introduces no additional samples.

The geometric series for total mass, age, and squared coefficients now give

$$\begin{aligned} \sum_{j=0}^{T-1} \max_s c_{s,j} &\le \frac{4}{3} + \alpha(M-1) < 3, \\ \sum_{j=0}^{T-1} |j-t| \max_s c_{s,j} &\le \frac{4q}{3\alpha} + \frac{\alpha M(M-1)}{2} < \frac{2}{\alpha}, \\ \sum_{j=0}^{T-1} (\max_s c_{s,j})^2 &\le \frac{16\alpha}{9(2-\alpha)} + \alpha^2(M-1) < 2\alpha. \end{aligned} \tag{25}$$

Here we used $\sum_{r\ge 0} q^r = 1/\alpha$, $\sum_{r\ge 0} r q^r = q/\alpha^2$, and $\sum_{r\ge 0} q^{2r} = 1/[\alpha(2-\alpha)]$. The future contributions in the three lines are at most $1/4$, $1/(32\alpha)$, and $\alpha/4$, respectively; also $2 - \alpha \ge 7/4$. These facts justify the final numerical bounds.

Initialization bias and gradient drift. Substituting $g_s^j = \nabla_s F(y^j) + \epsilon_s^j$ into (23) and using the coefficient sum above yields

$$\begin{aligned} d_s^{k_s+1} - \nabla_s F(z^t) = & -q^{n_s(t)} \nabla_s F(x^0) \\ & + q^{n_s(t)} \big(\nabla_s F(x^0) - \nabla_s F(z^t) \big) \\ & + \sum_{j=0}^{T-1} c_{s,j} \big(\nabla_s F(y^j) - \nabla_s F(z^t) \big) + \sum_{j=0}^{T-1} c_{s,j} \epsilon_s^j. \end{aligned} \tag{26}$$

Since $n_s(t) \ge t+1$, summing block norms bounds the first line by $q^{t+1} \left\| \nabla F(x^0) \right\|_*$.

We bound the next two terms pathwise after summing over stages. For each fixed j, its largest coefficient bounds the weighted sum of block-gradient differences by the full dual norm. Smoothness then applies once to this full gradient difference. This order avoids an additional factor S. By Lemma 1, $\left\| z^t - x^0 \right\| \le 2\eta\overline{\gamma} t$. Hence the second line of (26) contributes at most

$$q^{t+1} \left\| \nabla F(x^0) - \nabla F(z^t) \right\|_* \le 2L\eta\overline{\gamma} t q^{t+1} \le \frac{L\eta\overline{\gamma}}{\alpha}. \tag{27}$$

For the last inequality, $tq^{t+1} \le 1/(2\alpha)$: for $t \ge 1$, pairing q^i and q^{2t-1-i} gives $2tq^t \le \sum_{i=0}^{2t-1} q^i \le 1/\alpha$; for $t = 0$ the inequality is immediate. For the remaining gradient drift, (19) and the first two bounds in (25) give

$$2L\eta\overline{\gamma} \sum_{j=0}^{T-1} (H + |j-t|) \max_s c_{s,j} \le 6L\eta\overline{\gamma} H + \frac{4L\eta\overline{\gamma}}{\alpha}.$$

Together with (27), the total gradient drift is bounded by

$$6L\eta\overline{\gamma} \left(H + \frac{1}{\alpha} \right). \tag{28}$$

Noise and temporal cancellation. For $i < j$, ϵ_s^i is $\mathcal{F}_j$-measurable, so (22) implies

$$\mathbb{E}\langle \epsilon_s^i, \epsilon_s^j \rangle = \mathbb{E}\left\langle \epsilon_s^i, \mathbb{E}[\epsilon_s^j \mid \mathcal{F}_j] \right\rangle = 0. \tag{29}$$

Square-integrability justifies the multiplication identity by truncation and Cauchy–Schwarz; bounded noise is not required. All $c_{s,j}$ are deterministic. Expanding the squared product Hilbert norm and applying (29) gives

$$\begin{aligned}\mathbb{E}\left\|\left(\sum_{j=0}^{T-1} c_{s,j}\epsilon_s^j\right)_{s=1}^{S}\right\|_2^2 &= \sum_{j=0}^{T-1}\sum_{s} c_{s,j}^2\mathbb{E}\left\|\epsilon_s^j\right\|_2^2 \\ &\le \sigma^2\sum_{j=0}^{T-1}(\max_s c_{s,j})^2 \le 2\sigma^2\alpha.\end{aligned} \tag{30}$$

The squared product norm has no cross-stage inner products, so stages within one microbatch need not have independent noise. Norm comparison and Cauchy–Schwarz therefore give

$$\mathbb{E}\left\|\left(\sum_{j=0}^{T-1} c_{s,j}\epsilon_s^j\right)_{s=1}^{S}\right\|_* \le \rho\sqrt{\mathbb{E}\left\|\left(\sum_{j=0}^{T-1} c_{s,j}\epsilon_s^j\right)_{s=1}^{S}\right\|_2^2} \le \sqrt{2}\rho\sigma\sqrt{\alpha}. \tag{31}$$

All cancellation used the causal reads; no conditioning on z^t or independence between window averages was used.

Finally, take block norms in (26), sum over stages, and take expectations. Insert the initialization bound, (28), and (31). Multiplying the resulting unweighted sum by $\overline{\gamma}$ bounds the sum weighted by γ_s/m_s, proving (21). □

Proof of Theorem 1, part (i). We first prove descent pathwise, then take expectation and solve the resulting recurrence. The weight-decay terms use the actual stage weights, and momentum is controlled by the tracking estimate above. Expected KL is applied only after taking unconditional expectation.

Pathwise descent. Optimality of the block LMO and the dual-norm inequality give, for any block covectors h_s, d,

$$\begin{aligned}\langle h_s, \mathrm{lmo}_s(d)\rangle &\le \langle d, \mathrm{lmo}_s(h_s)\rangle + \|h_s - d\|_{s,*} \\ &\le -\|h_s\|_{s,*} + 2\|d - h_s\|_{s,*}.\end{aligned} \tag{32}$$

The identity $\langle h_s, \mathrm{lmo}_s(h_s)\rangle = -\|h_s\|_{s,*}$ uses symmetry of the unit ball; the inequality also covers zero inputs. With $h_s = \nabla_s F(z^t)$ and $k_s = \lfloor t/m_s \rfloor$, (32) and the interior margin in Lemma 1 imply

$$\begin{aligned}\langle h_s, \eta\,\mathrm{lmo}_s(d_s^{k_s+1}) - x_s^{k_s}\rangle &\le -(\eta - \left\|x_s^{k_s}\right\|_s)\|h_s\|_{s,*} + 2\eta\left\|d_s^{k_s+1} - h_s\right\|_{s,*} \\ &\le -\frac{\eta}{4}\|h_s\|_{s,*} + 2\eta\left\|d_s^{k_s+1} - h_s\right\|_{s,*}.\end{aligned}$$

The weight-decay term uses the actual weights $x_s^{k_s}$ appearing in (18). Multiplying by γ_s/m_s, summing over stages, and using the product dual norm bounds the linear descent term. Assumption 1 also gives the usual upper model

$$F(x+v) \le F(x) + \langle \nabla F(x), v\rangle + \frac{L}{2}\|v\|^2, \tag{33}$$

by integrating $\langle \nabla F(x+uv) - \nabla F(x), v\rangle$ over $0 \le u \le 1$. Applying (33) with $v = z^{t+1} - z^t$ and using (19) yields

$$\begin{aligned}F(z^{t+1}) - F_* \le F(z^t) - F_* &- \frac{\eta}{4}\underline{\gamma}\left\|\nabla F(z^t)\right\|_* \\ &+ 2\eta\sum_s \frac{\gamma_s}{m_s}\left\|d_s^{k_s+1} - \nabla_s F(z^t)\right\|_{s,*} + 2L\eta^2\overline{\gamma}^2.\end{aligned} \tag{34}$$

This inequality is pathwise, so dependence between z^t and its endpoint momenta causes no difficulty.

Transferring the KL inequality from the gradient reads. Assumption 2 is imposed at y^t, where the stochastic gradient is evaluated. The descent inequality instead involves z^t. We relate these points using their deterministic distance bound

$$D = 2\eta\overline{\gamma}H, \qquad \left\|y^t - z^t\right\| \le D.$$

Smoothness, the triangle inequality, and the read-based KL condition give

$$\mathbb{E}\left\|\nabla F(z^t)\right\|_* \ge \mathbb{E}\left\|\nabla F(y^t)\right\|_* - LD \ge \mu\mathbb{E}[F(y^t) - F_*] - LD.$$

The lower smoothness bound from z^t to y^t also gives

$$\mathbb{E}[F(y^t) - F_*] \ge \mathbb{E}[F(z^t) - F_*] - D\mathbb{E}\left\|\nabla F(z^t)\right\|_* - \frac{L}{2}D^2.$$

Substituting and rearranging yields

$$\begin{aligned}\mathbb{E}\left\|\nabla F(z^t)\right\|_* &\ge \frac{\mu}{1+\mu D}\mathbb{E}[F(z^t) - F_*] - LD\frac{1+\mu D/2}{1+\mu D} \\ &\ge \frac{\mu}{1+2\mu\eta\overline{\gamma}H}\mathbb{E}[F(z^t) - F_*] - 2L\eta\overline{\gamma}H.\end{aligned} \tag{35}$$

All expectations are unconditional, and D is deterministic. No independence between the read and the interpolation is needed.

Expectation and recurrence. Containment and continuity bound F and its gradient on a deterministic compact ball; (22) ensures integrable finite EMA sums. For the remaining general-bound proof, write the contraction amount as

$$c = \frac{\mu\eta\underline{\gamma}}{4(1+2\mu\eta\overline{\gamma}H)}. \tag{36}$$

Taking unconditional expectation in (34), then applying (35) and (21), gives

$$\begin{aligned}\mathbb{E}[F(z^{t+1}) - F_*] \le{}& (1-c)\mathbb{E}[F(z^t) - F_*] \\ &+ L\eta^2\overline{\gamma}^2\left(2 + \frac{25}{2}H + \frac{12}{\alpha}\right) + 2\sqrt{2}\eta\overline{\gamma}\rho\sigma\sqrt{\alpha} \\ &+ 2\eta\overline{\gamma}\left\|\nabla F(x^0)\right\|_*(1-\alpha)^{t+1}.\end{aligned} \tag{37}$$

The transfer adds $L\eta^2\underline{\gamma}\overline{\gamma}H/2 \le L\eta^2\overline{\gamma}^2H/2$ to the drift term, explaining the coefficient $25/2$. KL is used only at y^t for $0 \le t < K_T$, not at an interpolated point or the returned model. Conditional noise centering was used only at the causal reads in Lemma 3.

Positivity and (10) imply $0 < c \le 1$. Thus (37) can be iterated without reversing inequalities. Starting from $z^0 = x^0$, each constant forcing term is multiplied by

$$\sum_{t=0}^{K_T-1}(1-c)^{K_T-1-t} = \frac{1-(1-c)^{K_T}}{c}.$$

Keeping the initialization contribution inside its finite sum gives

$$\begin{aligned}\mathbb{E}[F(z^{K_T}) - F_*] \le{}& (1-c)^{K_T}\left(F(x^0) - F_*\right) \\ &+ \frac{L\eta^2\overline{\gamma}^2}{c}\left(2 + \frac{25}{2}H + \frac{12}{\alpha}\right)[1-(1-c)^{K_T}] \\ &+ \frac{2\sqrt{2}\eta\overline{\gamma}\rho\sigma}{c}\sqrt{\alpha}[1-(1-c)^{K_T}] \\ &+ 2\eta\overline{\gamma}\left\|\nabla F(x^0)\right\|_*\sum_{t=0}^{K_T-1}(1-c)^{K_T-1-t}(1-\alpha)^{t+1}.\end{aligned} \tag{38}$$

We use $0^0 = 1$ in finite sums, including when $1 - c = 0$. To obtain the simpler theorem statement, apply

$$1-(1-c)^{K_T} \le 1, \qquad \sum_{t=0}^{K_T-1}(1-c)^{K_T-1-t}(1-\alpha)^{t+1} \le \sum_{t=0}^{K_T-1}(1-\alpha)^{t+1} \le \frac{1}{\alpha}.$$

These inequalities give

$$\mathbb{E}[F(z^{K_T}) - F_*] \le \mathcal{B}_z, \tag{39}$$

where substituting (36) gives exactly $\mathcal{B}_z$ in (11). Lemma 2 now gives (12) for x^{out}.

Why the stage stepsizes scale linearly. Fix $\underline{\gamma}$, α, the schedule, radius, horizon, and initial model, with common valid problem constants (including μ) for the compared runs. Replacing γ_s by $m_s\underline{\gamma}$ preserves the minimum effective step and decreases its maximum to $\overline{\gamma} = \underline{\gamma}$. This does not decrease c, so the contracting term cannot increase in (11); each remaining term is nondecreasing in $\overline{\gamma}$. The transfer term $2\eta\overline{\gamma}(M-1)$ is also nondecreasing in $\overline{\gamma}$. Thus the full output bound is minimized by the same choice, which preserves (10). Writing $\gamma = \underline{\gamma}$ gives (13). The same comparison holds for the sharper bound (38): in its underlying recurrence (37), decreasing $\overline{\gamma}$ decreases both the nonnegative multiplier $1-c$ and every nonnegative forcing term. Induction from the common initial gap therefore decreases the scalar upper bound at every index. This comparison concerns the bound, rather than the actual loss, and does not optimize other stage EMA profiles. We retain (38) to prove part (ii). □

Proof of part (ii): parameter selection and iteration complexity. The general bound treats the initial momentum error conservatively. For the selected parameters, retaining its decay together with the loss contraction gives a sharper estimate. We therefore return to the unrolled recurrence instead of substituting directly into the simpler bound from part (i). Set $\underline{\gamma} = \overline{\gamma} = \gamma$ in (36) and (38), so that

$$c = \frac{\mu\eta\gamma}{4(1+2\mu\eta\gamma H)}.$$

The initialization term contains the finite geometric convolution

$$\sum_{t=0}^{K_T-1}(1-c)^{K_T-1-t}(1-\alpha)^{t+1} = \begin{cases}(1-\alpha)\dfrac{(1-c)^{K_T}-(1-\alpha)^{K_T}}{\alpha-c}, & \alpha \neq c,\\ (1-\alpha)K_T(1-c)^{K_T-1}, & \alpha = c.\end{cases} \tag{40}$$

The first identity follows by telescoping after multiplication by its denominator; in the second, every summand is equal. The convention $0^0 = 1$ covers a zero contraction factor and $K_T = 1$.

Proof of Theorem 1, part (ii). We first verify the hypotheses of part (i), then apply (40) without discarding the loss contraction. The expected KL inequality at the reads remains an assumption for the prescribed run.

Feasibility. Since $y^0 = x^0$ is deterministic, Assumption 2 gives $\|\nabla F(x^0)\|_* \geq \mu\big(F(x^0) - F_*\big)$. Take a unit vector u attaining $\langle \nabla F(x^0), u\rangle = \|\nabla F(x^0)\|_*$. Applying the smoothness model (33) from x^0 to $x^0 - \|\nabla F(x^0)\|_* u/L$ and using the global lower bound F_* yields

$$\|\nabla F(x^0)\|_*^2 \leq 2L\big(F(x^0) - F_*\big), \qquad \mu^2\varepsilon < \mu^2\big(F(x^0) - F_*\big) \leq 2L. \tag{41}$$

The selected parameters satisfy $0 < \alpha \leq 1/(4M)$ and $0 < \alpha_s = 1-(1-\alpha)^{m_s} < 1$, together with $\|x^0\| \leq \eta/2$ and $T\overline{\gamma} = T\underline{\gamma} = 1/2$. Moreover, (16) and (41) give

$$\frac{\mu\eta\gamma}{\alpha} = \frac{\mu\eta}{2\alpha T} \leq \frac{\mu^2\varepsilon}{16L\alpha(2+12H+12/\alpha)} \leq \frac{\mu^2\varepsilon}{192L} < \frac{1}{96}. \tag{42}$$

Thus $0 < \mu\eta\gamma \leq \alpha \leq 1/4$, verifying the remaining condition in (10); also $c < \alpha/384$.

We next bound the loss of contraction from transferring KL to z^t. Using $\gamma = 1/(2T)$, (16), and (41) gives

$$2\mu\eta\gamma H = \frac{\mu\eta H}{T} \leq \frac{\mu^2\varepsilon H}{8L(2+12H+12/\alpha)} \leq \frac{1}{48}. \tag{43}$$

For $H = 0$ the left side is zero. The same budget and the cap $\alpha \leq 1/(4M)$ imply

$$T \geq \frac{384L\eta M}{\mu\varepsilon} > 192\mu\eta M.$$

By (14), $\mu\eta \geq 16\log 4$, so in particular $T > 4M$. Since $T - K_T \leq M-1$, this gives $K_T \geq 3T/4$. Consequently

$$cK_T \geq \frac{\mu\eta}{8T(49/48)}\frac{3T}{4} = \frac{9}{98}\mu\eta > \frac{\mu\eta}{16}, \qquad (1-c)^{K_T} \leq e^{-cK_T} \leq e^{-\mu\eta/16}. \tag{44}$$

This establishes parameter feasibility before using the loss bound.

Retaining the initial momentum contraction. The coupling above implies $\alpha - c \geq \alpha/2 > 0$. Hence (40) gives

$$2\eta\gamma \left\|\nabla F(x^0)\right\|_* \sum_{t=0}^{K_T-1} (1-c)^{K_T-1-t}(1-\alpha)^{t+1}$$
$$\leq \frac{4\eta\gamma \left\|\nabla F(x^0)\right\|_*}{\alpha}(1-c)^{K_T} \leq \frac{4\left\|\nabla F(x^0)\right\|_*}{\mu}(1-c)^{K_T}.$$

The last inequality uses $\mu\eta\gamma \leq \alpha$. Substitute this estimate into (38), use (44), $\gamma = 1/(2T)$, and $1-(1-c)^{K_T} \leq 1$. We obtain

$$\begin{aligned}\mathbb{E}[F(z^{K_T}) - F_*] \leq & \left(F(x^0) - F_* + \frac{4\left\|\nabla F(x^0)\right\|_*}{\mu}\right) e^{-\mu\eta/16} \\ & + \frac{2L\eta}{\mu T}\left(1 + \frac{\mu\eta H}{T}\right)\left(2 + \frac{25}{2}H + \frac{12}{\alpha}\right) \\ & + \frac{8\sqrt{2}\rho\sigma}{\mu}\left(1 + \frac{\mu\eta H}{T}\right)\sqrt{\alpha}.\end{aligned} \tag{45}$$

The first term is at most $\varepsilon/4$ by (14). For the second and third terms, (43) gives $1 + \mu\eta H/T \leq 49/48$, and

$$2 + \frac{25}{2}H + \frac{12}{\alpha} \leq \frac{25}{24}\left(2 + 12H + \frac{12}{\alpha}\right).$$

Using (16) and (15) therefore yields

$$\mathbb{E}[F(z^{K_T}) - F_*] \leq \frac{\varepsilon}{4} + \frac{49}{48}\frac{25}{24}\frac{\varepsilon}{4} + \frac{49}{48}\frac{\varepsilon}{4} = \frac{3553}{4608}\varepsilon < \frac{4\varepsilon}{5}. \tag{46}$$

When $\sigma = 0$, the third term in (45) is zero.

Accuracy of the returned weights. For the selected parameters, the distance in Lemma 2 is $D = \eta(M-1)/T$. The bound $T \geq 384L\eta M/(\mu\varepsilon)$ established above and $\mu^2\varepsilon < 2L$ from (41) imply

$$\sqrt{L/2}\, D \leq \frac{\mu\varepsilon}{384\sqrt{2L}} < \frac{\sqrt{\varepsilon}}{384}.$$

Using $\mathcal{B} = 4\varepsilon/5$ from (46) in Lemma 2, we conclude

$$\mathbb{E}[F(x^{\text{out}}) - F_*] < \left(\sqrt{4/5} + \frac{1}{384}\right)^2 \varepsilon < \varepsilon.$$

This proves (17) for the returned model with the single budget (16).

Expanding the iteration complexity. The prescription (15) gives, including when $\sigma = 0$,

$$\frac{1}{\alpha} = \max\left\{4M, \frac{2048\rho^2\sigma^2}{\mu^2\varepsilon^2}\right\}.$$

Choose T equal to the ceiling in (16). Substitution yields

$$T = O\left(\max\left\{M, \quad \frac{L\eta(H+M)}{\mu\varepsilon}, \quad \frac{L\eta\rho^2\sigma^2}{\mu^3\varepsilon^3}\right\}\right). \tag{47}$$

Now choose equality in (14). If its logarithmic entry dominates initialization, then specifically

$$\eta = \frac{16}{\mu}\log\frac{4(F(x^0) - F_* + 4\left\|\nabla F(x^0)\right\|_*/\mu)}{\varepsilon}.$$

Inserting this expression in (47) gives (8), with this logarithm multiplying both terms. The baseline is absorbed because (41) gives $L(H+M)/(\mu^2\varepsilon) > M/2$. If $2\left\|x^0\right\|$ dominates or a larger radius is chosen, use (47) with that η instead. A common read-based KL constant μ is required across any accuracy-dependent runs being compared.

For the stationary-noise allowance $(49/48)\varepsilon/4$ used above, the two caps in (15) are the constraints on α. The sufficient budget (16) decreases with α, so choosing the largest permitted α minimizes it under these caps. This does not assert global optimality of the allocation or the exact finite bound. □

C Additional experiments

C.1 Experimental setup

Settings and budgets. The three settings are: the 135M model at approximately 20 non-padding tokens per parameter (4.2M sequences), the 360M model at the same ratio (11.3M sequences), and the 135M model at five times the budget (21M sequences). The synchronous optimum $B^\star$ is about 512 sequences for the 135M model at the Chinchilla budget and 1024 sequences for the two other settings; the batch grids are $B \in \{1024, 1472, 2048, 2880, 4096\}$ and $B \in \{2048, 2880, 4096, 5760, 8192\}$ sequences respectively.

Padding and token counts. As in Zmushko et al. (2026), sequences are padded to a context length of 1024, while their average non-padding length is approximately 700 tokens. Thus, multiplying a sequence count by 1024 counts padded token positions rather than only training tokens. The Chinchilla multipliers above refer to approximate non-padding token budgets; the sequence counts specify the training horizons used in our comparisons. Within each setting, all methods process the same number of sequences.

Recipe and momentum. All methods use Muon with Nesterov momentum for matrix parameters and AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$) for embeddings and normalization parameters, weight decay 0.1, gradient clipping at 1.0, a cosine learning-rate schedule decaying to 0.1 of the peak, and a warmup of 10% of the token budget. The learning rate is swept on a $\sqrt{2}$-spaced ladder for every cell; all reported optima are interior points of their ladders unless marked otherwise. Each cell is a single run. The observed run-to-run variation in final evaluation loss is approximately 0.001–0.002. Synchronous training uses $\mu = 0.95$, and 0.9 at the largest batches where it is better. For PipeDream the optimal momentum is set by the microbatch: we sweep it per cell and find $\mu = 0.95$ at a microbatch of 1024 sequences rising to 0.995 at 128, with a steep penalty for overshooting; Error Feedback does not move the optimum, so PipeDream+EF uses the momentum tuned without it. PipeTempo uses a common momentum coefficient across stages within each run. In the main comparisons, WP uses stages 4–7 for $S = 8$ and 8–15 for $S = 16$, except for the 135M Chinchilla runs at $B \in \{1024, 2048, 4096\}$, which use stages 9–15.

C.2 Weight prediction by stage

Weight-prediction subsets at $S = 16$. Table 2 reports all tested suffixes of delayed stages in the 135M ablation at $B = 4096$. Predicting on all stages increases loss, while excluding early stages recovers the benefit. The lowest measured loss is obtained with WP on stages 9–15, though several nearby subsets are within the run-to-run variation. The comparison between stages 1–15 and 3–15 measures the effect of adding WP to stages 1–2 when the later stages already use it; it is not a separate run with WP only on stages 1–2.

Table 2: Stage-subset ablation at $S = 16$, $B = 4096$, 135M model, same budget and optimizer as Table 1. Each row reports the best final loss over the learning-rate ladder; Δ is relative to no WP. Stage 16 has no delay.

Stages with WP	Loss	Δ
None	2.9132	0
1–15 (all delayed)	2.9291	$+0.0159$
3–15	2.9116	-0.0016
5–15	2.9059	-0.0073
7–15	2.9046	-0.0087
9–15	2.9043	-0.0090
11–15	2.9062	-0.0070
13–15	2.9104	-0.0028
15 only	2.9137	$+0.0005$

Weight and gradient errors by stage. We compare stale and predicted weights with a local zero-delay reference along a plain PipeTempo trajectory. Using the notation of Section 3, let $\mathcal{W}$ be a sampled window of m_s microbatches during which stage s reads x_s^k. Its exact next weights are x_s^{k+1}, and WP predicts $\tilde{x}_s^{k+1}$. For each microbatch ξ_t in this window, we replace only stage s's weights with a candidate z and recompute

$$\bar{g}_{s,\mathcal{W}}(z) = \frac{1}{m_s} \sum_{t \in \mathcal{W}} \nabla_s f(y_1^t, \ldots, y_{s-1}^t, z, y_{s+1}^t, \ldots, y_S^t; \xi_t).$$

Every other stage keeps its original read y_j^t from the same PipeTempo trajectory, including when $z = x_s^{k+1}$. These reads may vary across microbatches, but are identical across the stale, predicted, and reference evaluations of a given microbatch. We do not advance the other stages to their next weights or rerun training along a new trajectory.

We measure the relative gradient error and the remaining weight error:

$$E_g(z) = \frac{\|\bar{g}_{s,\mathcal{W}}(z) - \bar{g}_{s,\mathcal{W}}(x_s^{k+1})\|_2}{\|\bar{g}_{s,\mathcal{W}}(x_s^{k+1})\|_2}, \qquad R_x = \frac{\|\tilde{x}_s^{k+1} - x_s^{k+1}\|_2}{\|x_s^k - x_s^{k+1}\|_2}.$$

Here $E_{\text{plain}} = E_g(x_s^k)$ and $E_{\text{WP}} = E_g(\tilde{x}_s^{k+1})$; $R_x < 1$ means that the prediction is closer to the next weights than the stale point is.

Table 3 shows median $R_x < 1$ at every stage. Yet at stages 1–2, the remaining weight errors are 0.922 and 0.863 of the stale distances, while WP increases gradient error by 18–21%. Closer weights and worse gradients occur together in 14 of 18 sampled windows at stage 1 and 15 of 18 at stage 2. Thus, predicting a closer point does not necessarily produce a more accurate gradient. The stale gradient error itself falls from 0.467 at stage 1 to 0.046 at stage 7, a roughly $10\times$ difference. WP reduces gradient error by 47–66% at stages 5–7, where it is already smaller, while leaving the larger errors at early stages unresolved.

Table 3: Weight and gradient errors against the local reference, 135M model, $S = 8$, $B = 4096$, base learning rate 5.66×10^{-3}, $\mu = 0.95$. Entries are medians over 17–18 sampled windows per stage after the first 25% of training. The last column is the median of per-window ratios. Stage 8 has no delay.

Stage s	Window m_s	R_x	E_{plain}	E_{WP}	$E_{\text{WP}}/E_{\text{plain}}$
1	7	0.922	0.467	0.593	1.18
2	6	0.863	0.264	0.328	1.21
3	5	0.832	0.221	0.210	0.83
4	4	0.808	0.181	0.148	0.85
5	3	0.797	0.128	0.065	0.53
6	2	0.784	0.087	0.030	0.38
7	1	0.768	0.046	0.015	0.34

Mirrored accumulation schedule. In the standard PipeTempo schedule, early stages have long accumulation windows and update less often, while later stages have short windows and update more often. The WP subset ablation therefore leaves 2 properties linked: a stage's position in the network and its update frequency. To examine their roles, we construct a mirrored schedule that assigns short windows to early stages and long windows to later stages. The model's layers remain in their original order and keep the same stage assignments.

Specifically, we replace the decoder stages' delay assignment $\delta_s = S - s$ with $\delta_s = s - 1$ and set $m_s = \max(\delta_s, 1)$. For $S = 8$, the original accumulation windows $(7, 6, 5, 4, 3, 2, 1, 1)$ become $(1, 1, 2, 3, 4, 5, 6, 7)$. A stage with window m_s updates once every m_s microbatches, using the gradient accumulated over its previous window, so its local delay remains 1 update. The exception is stage 1, which now has no delay and uses the current microbatch's gradient. The final normalization and the shared embedding/output weights also remain without delay, as in the original setup. We retain the learning-rate rule $\gamma\sqrt{m_s}$, using the mirrored windows. This is a diagnostic assignment of delays and update intervals; it does not correspond to the standard 1F1B execution order.

We compare the mirrored schedule without WP to 2 variants: WP on early stages 2–4, whose windows contain 1–3 microbatches, and WP on later stages 5–8, whose windows contain 4–7 microbatches. Here, "slow" and "fast" refer to the frequency of optimizer updates. The prediction rule is unchanged, and stage 1 needs no prediction because it has no delay. Table 4 compares these variants at each of 2 common base learning rates.

In both the original and mirrored schedules, applying WP to slow-updating stages with long accumulation windows worsens training, while applying it to fast-updating stages with short windows improves training. In the mirrored schedule, WP on the slow stages increases loss by 0.016–0.018, while WP on the fast stages reduces it by 0.013–0.014. These results suggest that slow-updating stages may need a different approach to handling staleness.

Table 4: Weight prediction in the mirrored schedule, 135M model, $S = 8$, $B = 4096$, Chinchilla budget, $\mu = 0.95$. Each entry is a single run; Δ is relative to no WP at the same base learning rate.

	$\gamma = 5.66 \times 10^{-3}$		$\gamma = 8 \times 10^{-3}$	
Stages with WP	Loss	Δ	Loss	Δ
None	2.881	0.000	2.882	0.000
2–4 (fast, $m_s = 1$–3)	**2.867**	-0.014	**2.868**	-0.013
5–8 (slow, $m_s = 4$–7)	2.896	$+0.016$	2.900	$+0.018$

C.3 Weighted gradient accumulation

Later microbatches in an accumulation window are computed using more recent weights at other stages, suggesting that their gradients might be more important and should receive greater weight. We test this idea by replacing uniform averaging with weighted accumulation. For a window of $m_s > 1$ microbatches, linear weights increase from oldest to newest as $w_i \propto r + (1 - r)(i - 1)/(m_s - 1)$, where r is the ratio of the oldest to the newest weight and $\sum_i w_i = 1$. Thus, $r = 1$ gives uniform averaging, while smaller r puts more weight on recent gradients. We also test geometric weights $w_i \propto \rho^{m_s - i}$ with $\rho = 0.5$, and the linear $r = 0$ weights in reverse order, favoring older gradients. Windows of 1 microbatch are unchanged.

Table 5 shows no improvement over uniform averaging among the tested configurations. Mild linear reweighting has little effect, while stronger reweighting increases the final loss at both batch sizes. Reversing the strongest linear weighting is less harmful, but still worse than uniform averaging. Several strongly weighted variants reach their best loss at the lowest tested learning rate, so their optima are not fully resolved. One possible explanation is that unequal weights reduce the effective batch size and increase gradient noise. We did not separately test this explanation.

Table 5: Weighted accumulation in PipeTempo, 135M model, $S = 8$, Chinchilla budget, $\mu = 0.95$, without weight prediction. Each loss is the best over 4 learning rates, with the stage learning-rate scaling $\gamma\sqrt{m_s}$ unchanged; Δ is relative to uniform averaging.

	$B = 2048$		$B = 4096$	
Weighting	Loss	Δ	Loss	Δ
Uniform	**2.876**	0.000	**2.927**	0.000
Linear, $r = 0.75$	2.878	$+0.002$	2.929	$+0.002$
Linear, $r = 0.5$	2.880	$+0.005$	2.932	$+0.006$
Linear, $r = 0.25$	2.890	$+0.014$	2.947	$+0.021$
Linear, $r = 0$	2.982	$+0.107$	3.097	$+0.170$
Geometric, $\rho = 0.5$	3.178	$+0.302$	3.134	$+0.207$
Reversed linear, $r = 0$	2.916	$+0.040$	2.967	$+0.041$

C.4 Full results

Figures 6 and 7 provide the full comparisons across training settings and pipeline depths.

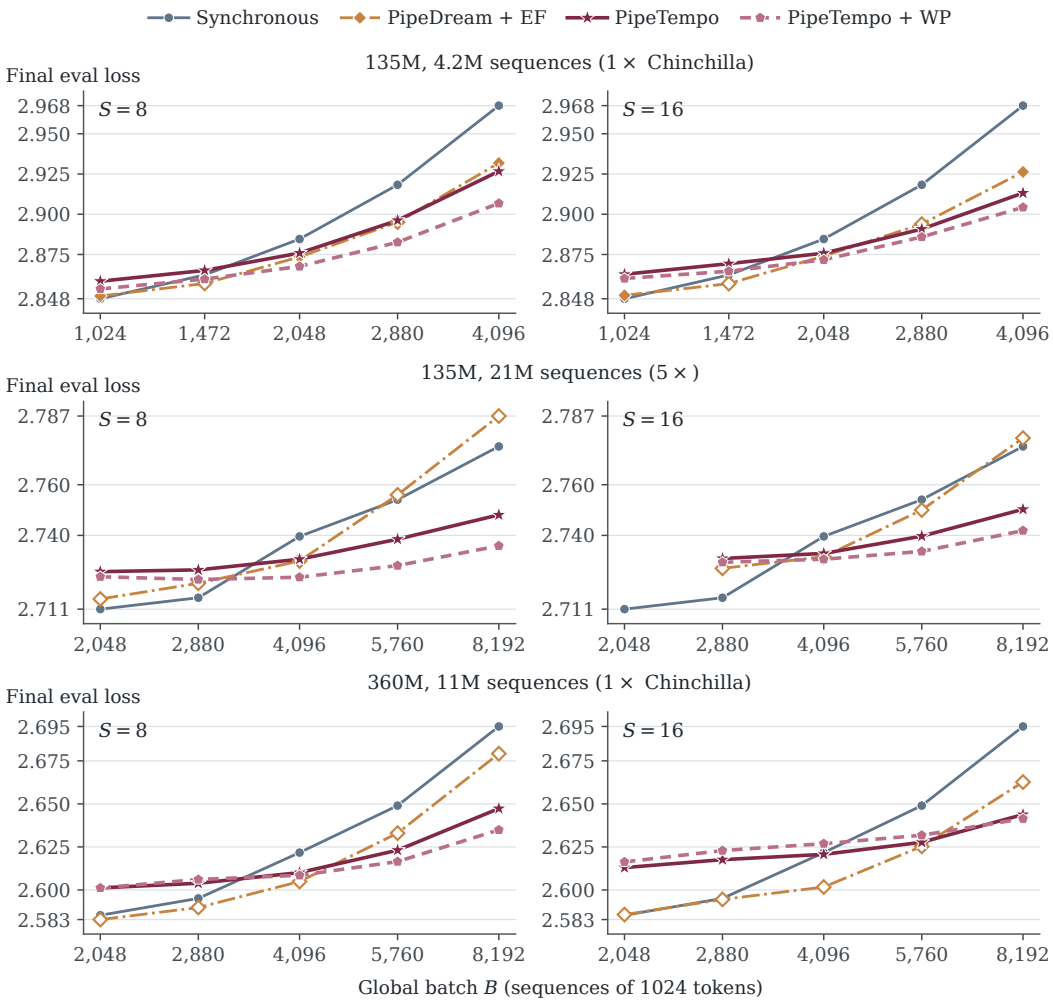

Figure 6: Final evaluation loss against the global batch B for all three settings (rows) and both depths (columns): synchronous training, PipeDream+EF, PipeTempo, and PipeTempo with weight prediction. Every point is the best over a learning-rate ladder; hollow markers are PipeDream+EF cells run at a single learning rate (the PipeDream optimum of the same cell). Missing points are cells that diverged or were not run.

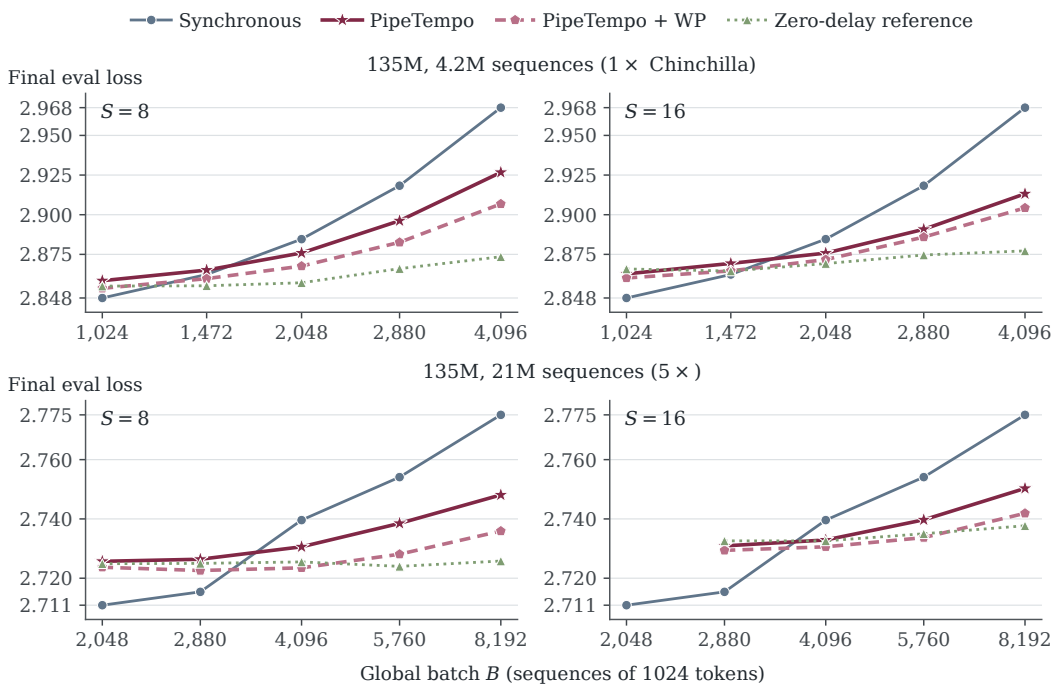

Figure 7: PipeTempo, PipeTempo with weight prediction, and the zero-delay reference against synchronous training for the 135M model at the Chinchilla budget (top) and at five times the budget (bottom).

C.5 Momentum tuning

Following Ajanthan et al. (2025); Zmushko et al. (2026), we carefully tune momentum for each method, sweeping the learning rate for each momentum value. Figure 8 shows the sensitivity of synchronous training, PipeDream, and PipeTempo to μ at different batch sizes. We use the momentum selected for PipeDream in its Error Feedback variant, as described in Appendix C.1.

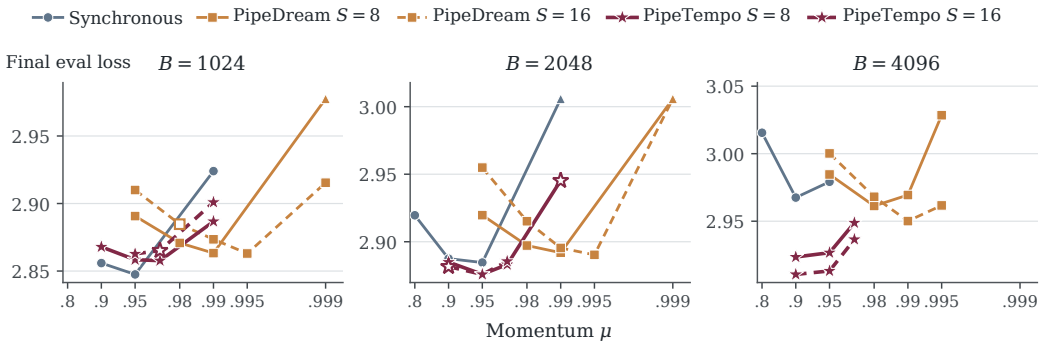

Figure 8: Momentum tuning for the 135M model at the Chinchilla budget: synchronous training, PipeDream without Error Feedback, and PipeTempo at $S = 8, 16$. PipeTempo uses constant μ and stage learning rates $\gamma\sqrt{m_s}$. Each point is the best final loss over a learning-rate grid. Hollow markers indicate optima at the edge of that grid; triangles mark losses above the plotted range.