SMOOTHING DILOCO WITH PRIMAL AVERAGING FOR FASTER TRAINING OF LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose Generalized Primal Averaging (GPA), an extension of Nesterov's method in its primal averaging formulation, that addresses key limitations of recent averaging-based optimizers such as DiLoCo and Schedule-Free (SF). These two recent algorithmic approaches improve the performance of base optimizers such as AdamW through distinct averaging strategies. Schedule-Free explicitly averages iterates at every step, while DiLoCo performs implicit averaging by periodically aggregating trajectories, called pseudo-gradients, to update the model parameters. This periodic averaging introduces a two-loop structure, increasing its memory requirements and number of hyperparameters to tune. To address these limitations, GPA smoothens DiLoCo in the non-distributed setting by averaging iterates at every iteration using two interpolation constants. When applied to language model pre-training, GPA consistently outperforms DiLoCo while removing the two-loop structure, simplifying hyperparameter tuning and reducing memory overhead to just a single additional buffer. Furthermore, we prove that for any base optimizer with regret bounded by $\mathcal{O}(\sqrt{T})$, where T is the number of iterations, GPA can match or exceed the convergence guarantee of the original optimizer, depending on the choice of the interpolation constants.

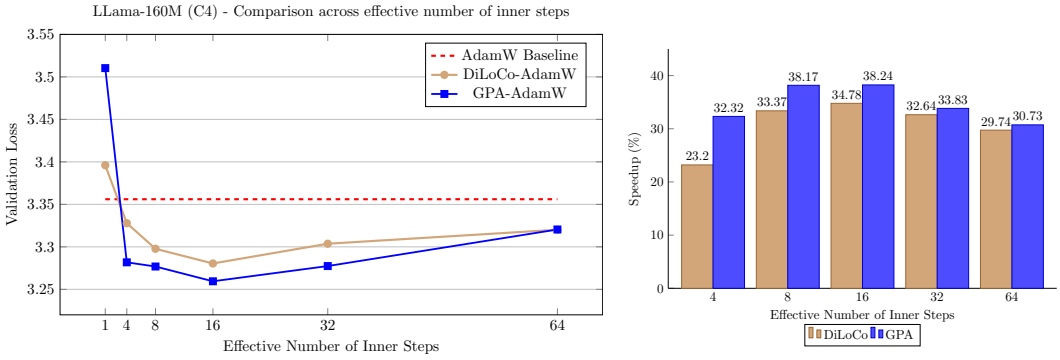
1 INTRODUCTION

As large language models (LLMs) demonstrate increasingly remarkable capabilities at scale (Achiam et al., 2023; Llama Team, 2024; Liu et al., 2024a), the pre-training phase has become one of the most expensive stages in the language model training pipeline, often costing hundreds of millions of dollars per run. This significant investment has driven the development of training algorithms and optimizers that enhance the efficiency, scalability, and robustness of language model pre-training.

One significant area of research is on the design of training algorithms for scalable distributed learning. Among these, the DiLoCo algorithm has emerged as the leading practical approach (Douillard et al., 2023; Liu et al., 2024b; Douillard et al., 2025; Charles et al., 2025). Notably, DiLoCo is capable of outperforming AdamW even in a non-distributed setup.

DiLoCo's consistent improvements over AdamW stem from its novel combination of the Nesterov optimizer with the Lookahead method (Zhang et al., 2019). By periodically combining accumulated weight updates into *pseudo-gradients* and applying Nesterov momentum, DiLoCo achieves substantial efficiency gains; for instance, when applied to AdamW on a 160 million parameter language model, this approach delivers speedups up to 38%; see Figure 1b.

A particularly intriguing behavior of DiLoCo is that its performance improves as the number of inner steps increases. With each base optimizer step, DiLoCo's outer weights drift farther from the inner weights, similar to meta learning optimizers such as Reptile (Nichol & Schulman, 2018) and First-Order MAML (Finn et al., 2017). In DiLoCo, updates to the outer weights occur only at periodic intervals, causing information from the data to be integrated in a discontinuous, choppy manner rather than smoothly at every iteration. This restriction on information flow to the outer weights appears unnecessary from an optimization perspective, yet counterintuitively improves its performance; see Figure 1a.



(a) Both GPA and DiLoCo using AdamW as their base optimizer significantly outperform a strong AdamW baseline for training a 160M parameter Llama model. Notably, increasing the number of inner steps (up to 16) improves the performance of DiLoCo. Unlike DiLoCo, GPA updates the parameters at every step, but uses a heuristic to choose its interpolation constants to match the number of inner steps for DiLoCo.

(b) Speedup achieved by DiLoCo and GPA in reducing the number of steps to reach AdamW's final validation loss, across different effective numbers of inner steps. GPA and DiLoCo attain the highest speedup of 38.24% and 34.78% respectively for the same interval of 16.

Figure 1: Comparison of validation loss and speedup for AdamW, DiLoCo, and GPA.

Concurrently, another line of optimizer research focuses on the design of weight- or iterate-averaging-based algorithms. One such optimizer, Schedule-Free, recently won the AlgoPerf Algorithmic Efficiency challenge self-tuning track (Dahl et al., 2023; Defazio et al., 2024). Its core novelty lies in computing gradients at a point that interpolates between the uniform average of past weights and the current weights. This interpolation step allows Schedule-Free to propagate information into the average weights at every iteration. Empirically, Schedule-Free matches the performance obtained by using learning rate schedules without using any schedule explicitly, while providing stronger theoretical last-iterate convergence guarantees similar to Polyak-Ruppert averaging (Ruppert, 1988; Polyak, 1990; Polyak & Juditsky, 1992).

In this paper, we argue that these two lines of work – DiLoCo and Schedule-Free – are closely related, and can be generalized and improved through a unified framework of *primal averaging*. Specifically, our contributions are as follows:

- We propose a generalization of Nesterov's method in its primal averaging formulation that we call *Generalized Primal Averaging* (GPA), which smooths DiLoCo by averaging at every step.
- In contrast to DiLoCo, GPA eliminates the two-loop structure, and thereby only requires a single additional buffer, has less hyperparameters to tune, and demonstrates more stable training behavior.
- We also provide theoretical justification for GPA through convergence guarantees that demonstrate improved convergence over the base optimizer under certain circumstances with particular choices of the interpolation constants;
- Our preliminary experiments show that GPA consistently outperforms non-distributed DiLoCo and AdamW on dense 160 million parameter language models. GPA demonstrates a convergence speedup of up to 38% in terms of steps taken to achieve the target validation loss of AdamW baseline.

2 Background

We frame language model pre-training as the expected risk minimization problem

$$\min_{x \in \mathbb{R}^n} F(x) = \mathbb{E}_{\xi \sim \mathcal{D}} \left[f(x; \xi) \right], \tag{1}$$

where $\xi \sim \mathcal{D}$ is drawn from an underlying stationary data distribution $\mathcal{D}$. We assume that each optimizer step has access to the stochastic minibatch gradient $g(x^{(t)}; \xi^{(t)}) \in \partial f(x^{(t)}; \xi^{(t)})$ evaluated at each iteration t on a minibatch of data $\xi^{(t)}$, over a total of T steps.[1]

We also assume that the base optimizer is of the form $x^{(t+1)} = x^{(t)} + \gamma^{(t)} d^{(t)}$ with learning rate $\gamma^{(t)} > 0$ and search direction $d^{(t)} \in \mathbb{R}^n$. The search direction is most commonly defined as $d^{(t)} = -H^{(t)} m^{(t)}$, where $m^{(t)} \in \mathbb{R}^n$ is a gradient estimator, and $H^{(t)} \in \mathbb{R}^{n \times n}$ is a symmetric positive definite preconditioner matrix. This includes popular methods such as SGD, Adam, Shampoo, SOAP, AdEMAMix, or Muon for different choices of $m^{(t)}$ and $H^{(t)}$ (Robbins & Monro, 1951; Gupta et al., 2018; Anil et al., 2020; Shi et al., 2023; Vyas et al., 2024; Jordan et al., 2024; Pagliardini et al., 2025; Eschenhagen et al., 2025).

2.1 Different Formulations of Nesterov

Despite Nesterov's importance in optimization for deep learning, there is substantial confusion in the literature regarding its formulation, as it can be written in at least seven different ways (Defazio, 2019). These formulations are equivalent in the sense that a direct mapping exists between them, but they may not return the same iterate.

For instance, Nesterov's method was popularized for deep learning in *Sutskever's formulation* (Sutskever et al., 2013), which presents the algorithm as:

$$\begin{aligned} b^{(t)} &= \mu b^{(t-1)} - \gamma^{(t)} g(x^{(t)} + \mu b^{(t-1)}; \xi^{(t)}), \\ x^{(t+1)} &= x^{(t)} + b^{(t)}, \end{aligned} \tag{2}$$

where $\mu > 0$ is the momentum hyperparameter and $b^{(t)} \in \mathbb{R}^n$ is the momentum buffer initialized at $b^{(0)} = 0$. An alternative formulation, which we call the *modern formulation*, is used by software libraries such as `PyTorch`[2] and `JAX`[3] due to its ease of use:

$$\begin{aligned} b^{(t)} &= \mu b^{(t-1)} + g(x^{(t)}; \xi^{(t)}), \\ x^{(t+1)} &= x^{(t)} - \gamma^{(t)} [\mu b^{(t)} + g(x^{(t)}; \xi^{(t)})]. \end{aligned} \tag{3}$$

In both formulations, we maintain a momentum buffer that averages gradients seen throughout the training process. Unlike Sutskever's formulation (equation 2), the modern formulation (equation 3) uses the iterate $x^{(t)}$ directly for the gradient computation, rather than the ancillary point $x^{(t)} + \mu b^{(t-1)}$, simplifying its practical implementation. If we run both formulations side-by-side with the same seed, they will evaluate gradients at exactly the same point, but the validation loss at the iterate $x^{(t)}$ of each method will differ due to its different definition.

Our approach instead builds upon a third form, which we call the *primal averaging formulation* (Lan, 2012):

$$\begin{aligned} y^{(t)} &= \mu x^{(t)} + (1-\mu) z^{(t)}, \\ z^{(t+1)} &= z^{(t)} - \gamma^{(t)} g(y^{(t)}; \xi^{(t)}), \\ x^{(t+1)} &= \mu x^{(t)} + (1-\mu)\, z^{(t+1)}, \end{aligned} \tag{4}$$

with $\mu \in [0, 1)$. Unlike the Sutskever and modern formulations framed in equations 2 and 3, the primal averaging formulation in equation 4 explicitly names two iterate sequences: a sequence where the gradients (or more generally, the search directions) are computed at, i.e., the *gradient computation sequence* $\{y^{(t)}\}_{t=1}^T$, as well as another sequence used for model evaluation that accumulates a running average of updated iterates $\{z^{(t)}\}_{t=1}^T$, i.e., the *model evaluation sequence* $\{x^{(t)}\}_{t=1}^T$. Since $y^{(t)}$ interpolates the smoothed sequence $x^{(t)}$ and unsmoothed sequence $z^{(t)}$, it increases the contribution of the gradient update to $y^{(t)}$ compared to $x^{(t)}$. This explicit formulation is convenient for implementation and theoretical analysis, and naturally leads to a view of acceleration as built

[1] We assume that f is convex for the convergence analysis, but we verify its performance on non-convex, possibly non-smooth functions.

[2] https://docs.pytorch.org/docs/2.8/generated/torch.optim.SGD.html

[3] https://optax.readthedocs.io/en/latest/api/optimizers.html#optax.sgd

upon *iterate averaging*, rather than from the physics-inspired intuition of *gradient averaging* behind momentum that is more commonly introduced.

We summarize the relationship between the modern and primal averaging formulations in Proposition 1 below.

Proposition 1. *Given fixed learning rates $\gamma_{\text{primal}}, \gamma_{\text{modern}} > 0$, Nesterov's primal averaging formulation (equation 4) is equivalent to Nesterov's modern formulation (equation 3) in the sense that*

$$y^{(t)}_{\text{primal}} = x^{(t)}_{\text{modern}} \quad \textit{and} \quad b^{(t)}_{\text{modern}} = \frac{1}{(1-\mu)\,\gamma_{\text{primal}}}\left(x^{(t)}_{\text{primal}} - x^{(t+1)}_{\text{primal}}\right), \tag{5}$$

when $\mu_{\text{primal}} = \mu_{\text{modern}} = \mu$ and $(1-\mu)\,\gamma_{\text{primal}} = \gamma_{\text{modern}}$.

The proof of this simple statement is rather technical, so we defer it to Appendix C.

It is important to note that the equivalence between the primal averaging and modern formulations of Nesterov acceleration holds only when the learning rates are fixed. When learning rate schedules are introduced, achieving this equivalence would require the momentum parameter to vary with each iteration. Furthermore, the restriction on the choice of μ differs between the modern and primal averaging formulations. These distinctions highlight that formulations based on gradient averaging versus iterate averaging produce different perspectives for hyperparameter tuning, which can have a significant impact on the algorithm's practical performance.

2.2 Non-Distributed DiLoCo and its Weaknesses

DiLoCo was originally introduced as a distributed algorithm for cross-datacenter training (Douillard et al., 2023). In the non-distributed setup, it computes multiple inner steps of the base optimizer on the model weights, then applies Nesterov (equation 3) on the *pseudo-gradient*, which is defined as the difference between the current and updated model parameters. This notably requires storing two additional optimizer states of the same shape as the model parameters: the momentum buffer $b^{(t)}$ as well as the current model parameters $x^{(t)}$ (also known as the *outer weights*) as it is being updated by the base optimizer (applied to the *inner weights*). DiLoCo's handling of the *fast* inner weights and *slow* outer weights can be interpreted as a modified Lookahead method that applies Nesterov acceleration to the outer weight updates (Zhang et al., 2019). The method was recently analyzed in Khaled et al. (2025).

A simplified version of non-distributed DiLoCo with H inner steps of the base optimizer can be described as:

$$\begin{aligned} p^{(t)} &= x^{(t)} - \texttt{BaseOptIteration}(x^{(t)}; \{\gamma^{(j)}\}_{j=1}^{H}, H) \\ b^{(t)} &= \mu b^{(t-1)} + p^{(t)} \\ x^{(t+1)} &= x^{(t)} - \tilde{\gamma}[\mu b^{(t)} + p^{(t)}], \end{aligned} \tag{6}$$

where $\tilde{\gamma} > 0$ is the outer learning rate and `BaseOptIteration` applies H iterations of the base optimizer to the iterate $x^{(t)}$ with inner learning rates $\{\gamma^{(j)}\}_{j=1}^{H}$. While DiLoCo originally introduced AdamW as the base optimizer, DiLoCo generalizes to other optimizers such as Muon (Thérien et al., 2025). A complete description of the algorithm is provided in Appendix B.

This specific application of Nesterov with multiple base optimizer steps is capable of surpassing the performance of the baseline optimizer, which also explains DiLoCo's ability to match the synchronous baseline, such as AdamW, in the distributed setting. This finding is surprising because applying Nesterov solely to the search direction at each iteration (equivalent to applying solely a single base optimizer step) cannot achieve the same performance gains.

However, this two-level structure is undesirable. From an *algorithmic perspective*, one would prefer to average iterates on-the-fly, as opposed to averaging trajectories that implicitly contain multiple iterations of the base optimizer. From the *users' perspective*, the two-level structure introduces an additional copy of the model weights required to compute the pseudo-gradient, and introduces additional hyperparameters to tune, e.g., the inner and outer learning rates, momentum, and number of inner steps. Lastly, from the *distributed training perspective*, DiLoCo couples the number of inner steps as a hyperparameter for both local SGD as well as for the modified Nesterov algorithm,

causing the algorithm's performance to counterintuitively improve as the number of base optimizer steps increases. We would expect that communicating more often should always be beneficial.

These challenges motivate the development of a new algorithm that removes the two-level structure while offering a separate hyperparameter that can smoothly average the observed iterates at every iteration.

2.3 Schedule-Free Learning

In parallel, Schedule-Free learning (SF) (Defazio et al., 2024) is a recently proposed method that wraps any base optimizer using a variant of the primal averaging formulation of Nesterov's method (equation 4) for hyperparameter-free learning:

$$\begin{aligned} y^{(t)} &= \mu x^{(t)} + (1-\mu) z^{(t)} \\ z^{(t+1)} &= z^{(t)} - \gamma g(y^{(t)}; \xi^{(t)}) \\ x^{(t+1)} &= \frac{t}{t+1} x^{(t)} + \Big(1 - \frac{t}{t+1}\Big) z^{(t+1)}. \end{aligned} \tag{7}$$

Originally designed to eliminate the need for manually specified learning rate schedules, Schedule-Free has demonstrated the surprising ability to not only match, but even surpass the practical performance of the original base optimizer. This is done by *decoupling* the momentum hyperparameter used in the $x^{(t)}$ and $y^{(t)}$ sequences, unlike the standard primal averaging formulation of Nesterov (equation 4). Through the choice of μ, the method interpolates between uniform Polyak-Ruppert averaging and stochastic primal averaging (Ruppert, 1988; Polyak, 1990; Tao et al., 2018).

Ignoring the hyperparameter-free learning problem, one could alternatively replace uniform averaging with exponential moving averaging of the iterates (Morales-Brotons et al., 2024). This suggests a different generalization that can recover and extend Nesterov acceleration while offering the potential flexibility necessary to remove the two-level structure in DiLoCo.

3 Generalized Primal Averaging (GPA)

By decoupling the constants for the model evaluation and gradient computation sequences in Nesterov's primal averaging formulation (equation 4) and leveraging the observation of using exponential moving averaging in place of uniform averaging in Schedule-Free (equation 7), we introduce the *Generalized Primal Averaging* (GPA) framework:

$$\begin{aligned} y^{(t)} &= \mu_y x^{(t)} + (1-\mu_y) z^{(t)} \\ z^{(t+1)} &= z^{(t)} - \gamma^{(t)} g(y^{(t)}; \xi^{(t)}) \\ x^{(t+1)} &= \mu_x x^{(t)} + (1-\mu_x)\, z^{(t+1)}. \end{aligned} \tag{8}$$

Here, $\mu_x \in [0, 1)$ and $\mu_y \in [0, 1]$ are independent hyperparameters that separately control the degree of interpolation used in maintaining the model evaluation sequence $x^{(t)}$ and gradient computation sequence $y^{(t)}$. The additional parameter μ_x serves as a smoothening or exponential moving average parameter that replaces Polyak-Ruppert averaging in Schedule-Free and plays a similar role to the number of inner steps in DiLoCo. By replacing Polyak-Ruppert averaging with exponential moving averaging, GPA is not inherently schedule-free and requires the use of a learning rate schedule. The complete pseudocode for a general base optimizer is provided in Algorithm 1.

The choice of μ_x and μ_y enables GPA to interpolate between stochastic primal averaging, exponential moving averages of the iterates, and no iterate averaging. Specifically, when $\mu_y = 1$, $x^{(t)} = y^{(t)}$ and we recover stochastic primal averaging; for $\mu_y = 0$, $x^{(t)}$ and $z^{(t)} = y^{(t)}$ become decoupled and we recover exponential moving averaging of the iterates. When $\mu_x = 0$, $x^{(t)} = y^{(t)} = z^{(t)}$ for any choice of μ_y, and GPA reverts back to the base optimizer.

Unlike DiLoCo, which is built around (pseudo-)gradient averaging (see equation 6), GPA is defined based on the primal or iterate averaging framework, and removes the inner-outer loop by averaging iterates at every step. We argue that this provides a more precise characterization of the method.

Algorithm 1 Generalized Primal Averaging (GPA)

Input: Initial iterate $x^{(1)}$, learning rate schedule $\gamma^{(t)} > 0$, weight decay $\lambda \geq 0$, interpolation parameters $\mu_x, \mu_y \in [0, 1)$, base optimizer `BaseOpt`.
$z^{(1)} = x^{(1)}$
for $t = 1, ..., T$ **do**
  $y^{(t)} = \mu_y x^{(t)} + (1 - \mu_y) z^{(t)}$ ▷ Update gradient computation point $y^{(t)}$.
  $g^{(t)} \in \partial f(y^{(t)}; \xi^{(t)})$ ▷ Gradient is evaluated at $y^{(t)}$.
  $d^{(t)} = \texttt{BaseOpt}(g^{(t)})$ ▷ Compute base optimizer's search direction.
  $z^{(t+1)} = (1 - \gamma^{(t)}\lambda) z^{(t)} + \gamma^{(t)} d^{(t)}$ ▷ Update $z^{(t)}$ iterate.
  $x^{(t+1)} = \mu_x x^{(t)} + (1 - \mu_x)\, z^{(t+1)}$ ▷ Update weighted iterate average $x^{(t)}$.
end for
Return $x^{(T)}$

For example, the primal averaging interpretation motivates its extension to other search directions by replacing $-g(y^{(t)}; \xi^{(t)})$ with the search direction $d^{(t)}$ evaluated at $y^{(t)}$. This extension is not intuitive from the gradient averaging perspective, as it would translate to averaging search directions (with potentially different, evolving preconditioners) in the momentum buffer.

GPA also retains several desirable properties of the base optimizer for deep learning. Because $\mu_x, \mu_y \in [0, 1]$, GPA preserves modular norm bounds of the model parameters. Additionally, GPA requires only one extra copy of the model weights for implementation – specifically, by storing $y^{(t)}$ and reconstructing $x^{(t)}$ from $y^{(t)}$ and $z^{(t)}$ during evaluation – unlike DiLoCo, which demands more memory overhead. Further details on these properties are provided in Appendix B.

3.1 Generalized Primal Averaging as Smoothened DiLoCo

As seen in Figure 1a, increasing the number of inner steps leads to improved performance for DiLoCo in the non-distributed setup. However, the underlying reasons for this behavior are not understood. By examining DiLoCo from the lens of GPA in equation 8, and comparing it to the more restrictive Nesterov formulation in equation 4, we can develop a deeper intuition for DiLoCo's inner workings.

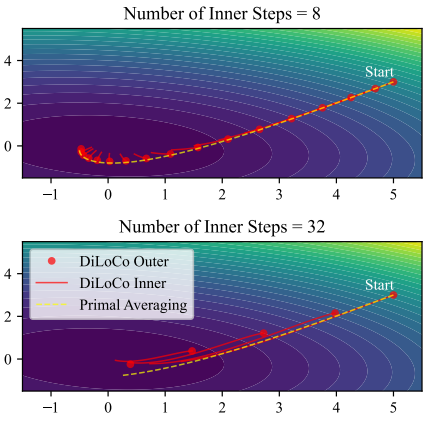


Figure 2: Comparison of DiLoCo and GPA's trajectories on a deterministic quadratic problem. The outer iterates of DiLoCo are shown as red points, and the inner iterates as thin red lines.

Suppose that we increase the number of inner steps in DiLoCo, and we want to maintain the same level of smoothing on the average iterate $x^{(t)}$. One may attempt to increase μ in Nesterov (equation 4) to decrease the weight on the current iterate $z^{(t+1)}$. However, since μ controls both the amount of smoothing in $x^{(t)}$ *and* the amount of interpolation used to update $y^{(t)}$, strictly increasing μ would *decrease the recency of information in* $y^{(t)}$ by a factor of μ^2, resulting in significantly different algorithmic behavior. Numerically, we validate that tuning μ in the primal averaging formulation of Nesterov is not sufficient to reach the performance of DiLoCo; see Appendix D.

GPA addresses this limitation by decoupling the two roles of μ into separate hyperparameters: μ_x for the model evaluation sequence and μ_y for the gradient computation sequence. By controlling these two interpolation constants independently, we can smooth $x^{(t)}$ similarly without changing the amount of information introduced into $y^{(t)}$.

This intuition provides us practical guidelines for hyperparameter tuning. For example, given an optimal number of inner steps H and momentum parameter μ in DiLoCo, we observe for GPA that

$$x^{(t+H)} = \mu_x x^{(t+H-1)} + (1-\mu_x) z^{(t+H)} = \cdots = \mu_x^H x^{(t)} + (1-\mu_x) \sum_{k=0}^{H-1} \mu_x^k z^{(t+H-k)}.$$

Therefore, to match the coefficient in front of $x^{(t)}$ with DiLoCo, one can set $\mu_x = \mu^{1/H}$ while keeping $\mu_y \approx \mu$. With commonly used values $\mu = 0.9$ and $H = 32$, we obtain $\mu_x \approx 0.9967$ and $\mu_y \approx 0.9$. We leverage this heuristic to determine an effective number of inner steps used in Figure 1.

Visually, Figure 2 illustrates how the iterates of GPA follow a smoothed trajectory of the DiLoCo iterates on a simple deterministic quadratic problem. For a small number of inner steps, the methods closely align, but for a larger number of inner steps, their behavior diverges.

GPA not only outperforms DiLoCo, but does so with fewer hyperparameters and lower memory requirements. While DiLoCo requires four hyperparameters, e.g., the inner and outer learning rate, momentum hyperparameter, and number of inner steps, GPA reduces this to just three: the learning rate and two momentum parameters. This simplification is possible because DiLoCo's practical performance is governed by an effective learning rate that couples the effect of the inner and outer learning rates ($\gamma^{(t)}$ and $\tilde{\gamma}$).

4 Convergence Theory

By utilizing the theoretical developments underpinning Schedule-Free learning, we can derive a convergence bound for Generalized Primal Averaging given any base optimizer that has a regret bound, using the framework of online-to-batch conversion (Cesa-Bianchi et al., 2004). We will use the Bregman divergence of F defined as $B_F(a,b) = F(a) - F(b) - \langle \nabla F(b), a-b \rangle$ for $a, b \in \mathbb{R}^n$.

Theorem 1. *Let F be a convex function, and assume that there exists a minimizer x_* that minimizes F. Let $\xi^{(1)}, \ldots, \xi^{(T)}$ be a sequence of i.i.d. random variables. Suppose that we are given arbitrary updates $z^{(1)}, \ldots, z^{(T)}$ from a base optimizer within the Generalized Primal Averaging framework (Equation 8). Then for $\mu_x, \mu_y \in [0,1)$ and average iterate $\bar{x}^{(T)} = \frac{1}{T}\sum_{t=1}^T x^{(t)}$, we have the bound*

$$\begin{aligned}
\mathbb{E}[F(\bar{x}^{(T)}) - F(x_*)] \leq \frac{1}{T}\sum_{t=1}^T \mathbb{E}[\langle \nabla F(y^{(t)}), z^{(t)} - x_* \rangle] \\
+ \frac{\mu_x}{1-\mu_x}\frac{1}{T}\mathbb{E}\left[F(x^{(1)}) - F(x_*)\right] \\
- \frac{1}{1-\mu_y}\frac{1}{T}\sum_{t=1}^T \mathbb{E}[B_F(y^{(t)}, x^{(t)})] - \frac{\mu_y}{1-\mu_y}\frac{1}{T}\sum_{t=1}^T \mathbb{E}[B_F(x^{(t)}, y^{(t)})] \\
- \frac{\mu_x}{1-\mu_x}\frac{1}{T}\sum_{t=1}^T \mathbb{E}[B_F(x^{(t-1)}, x^{(t)})].
\end{aligned}$$

Corollary 1. *Assume that the base optimizer has regret guarantees $\sum_{t=1}^T \mathbb{E}[\langle \nabla F(y^{(t)}), z^{(t)} - x_* \rangle] = \mathcal{O}(\sqrt{T})$. Then:*

$$\mathbb{E}[F(\bar{x}^{(T)}) - F(x_*)] = \mathcal{O}\left(\frac{1}{\sqrt{T}}\right).$$

We give some remarks on Theorem 1:

- The first row on the right-hand side of the regret bound is the average regret of the base optimizer. This term captures the convergence rate from the base optimizer.
- The second row has a positive term, which decays at a $1/T$ rate, which is typically faster than the decay of the term in the first row.

Table 1: Final validation loss versus effective number of inner steps for different optimizers on Llama 160M. For DiLoCo and GPA, the optimal (lowest) validation loss is shown in bold.

Method	# Inner Steps = 8	# Inner Steps = 16	# Inner Steps = 32
AdamW	3.3561	3.3561	3.3561
DiLoCo-AdamW	3.2977	**3.2804**	3.3037
GPA-AdamW	3.2769	**3.2595**	3.2774

- All remaining Bregman divergence terms are negative and so are potentially beneficial. Therefore, if μ_x and μ_y are chosen such that the negative terms are larger than the positive term introduced in the second row, then GPA will converge faster than the base optimizer. The same terms appears in the convergence guarantees for Schedule-Free methods, and can explain when they may work better. Moreover, for strongly convex problems, such Bregman divergences were previously used to get $\mathcal{O}(1/T)$ convergence.
- Note that unlike the guarantees for Schedule-Free, our convergence bound is for the average iterate. For the best performance, a learning rate schedule should be used and the last iterate returned (Defazio et al., 2023).

From a high level, the convergence bound indicates that GPA will be faster than the base optimizer when the objective function varies nonlinearly between consecutive iterates and between $x^{(t)}$ and $y^{(t)}$.

5 Experiments

In this section, we assess the effectiveness of GPA for language model pre-training by comparing its performance against AdamW and DiLoCo. We use AdamW as the base optimizer for both DiLoCo (DiLoCo-AdamW) and GPA (GPA-AdamW).

Setup and hyperparameter tuning. We evaluate AdamW, DiLoCo-AdamW, and GPA-AdamW by pre-training a 160 million parameter Llama 3 model on the C4 dataset from scratch (Raffel et al., 2019). We follow the Chinchilla-optimal token budget of roughly 3.2 billion tokens (Hoffmann et al., 2022). All of our experiments are conducted on a single machine equipped with eight H100 GPUs (97GB memory). We use a batch size of 128 sequences with a sequence length of 2048 tokens, resulting in a total batch size of about 262,000 tokens.

To tune the hyperparameters, we use the following process:

- For AdamW, we fix $(\beta_1, \beta_2) = (0.9, 0.999)$ and $\epsilon = 10^{-8}$, and sweep the learning rate from $5 \cdot 10^{-5}$ through $3 \cdot 10^{-3}$.
- For DiLoCo-AdamW, we fix the inner optimizer's hyperparameters to AdamW's optimal hyperparameters, and sweep the outer learning rate from $[0.25, 1.0]$ and the outer momentum from $[0.7, 0.99]$. We also sweep through the number of inner steps from $[1, 128]$ with powers of 2.
- For GPA-AdamW, we use the optimal AdamW hyperparameters, and sweep μ_x based on the number of inner steps in DiLoCo (see Section 3.1). We sweep μ_y over a fine granular range from $[0.8, 0.999]$. We also increased the learning rate when possible.

All runs use a learning rate schedule that applies linear warmup through the initial 10% of training, then cosine decay through the rest of training to 1% of the specified learning rate. By default, we apply gradient clipping, with a clipping factor of 1.0; weight decay is also fixed to 0.1. A summary of the hyperparameter sweeps are provided in Table 2 in Appendix D.

Performance across number of inner steps. In Figure 1a, as we vary the (effective) number of inner steps, we observe that DiLoCo and GPA-AdamW both outperform AdamW in terms of validation loss, with GPA-AdamW superseding DiLoCo, except when the number of inner steps is 1. Both DiLoCo and GPA-AdamW display U-shaped behavior, improving as the number of steps increases up to a point, then degrading as it becomes too large. In Table 1, we provide the final

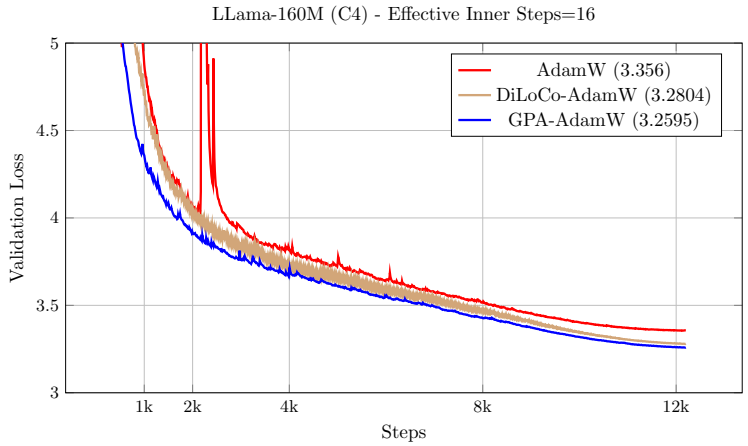


Figure 3: Validation loss versus steps for AdamW, DiLoCo, and GPA. For AdamW, we report the run with the best final validation loss, even though it is less stable than runs with lower learning rates.

validation loss values for each method at the key communication intervals of 8, 16, and 32. Both DiLoCo and GPA share the same optimal effective number of inner steps, validating our hypothesis on the choice of μ_x.

Speedup. In Figure 1b, we find that both DiLoCo and GPA significantly reduce the number of steps required to reach AdamW's final validation loss. DiLoCo achieves a maximum speedup of 34.78% at an interval of 16, while GPA attains an even higher maximum speedup of 38.24% with the same effective interval.

Convergence behavior. Figure 3 shows the validation loss curves for AdamW, DiLoCo-AdamW, and GPA-AdamW for the case where the number of inner steps is 16. In this case, μ_x has been tuned to match the number of inner steps; see Table 3 in Appendix D for details. GPA-AdamW converges faster than both DiLoCo and AdamW throughout the entire training run. The training curves for GPA-AdamW are also noticeably smoother and more stable compared to the other methods. Our hyperparameter sweeps reveal that GPA-AdamW can handle higher learning rates compare to DiLoCo and AdamW, e.g., $5 \cdot 10^{-3}$.

6 Conclusion

Generalized Primal Averaging (GPA) introduces independent interpolation constants for gradient computation and model evaluation that yields a flexible optimization framework. On small-scale dense models, this flexibility allows GPA to outperform DiLoCo, while removing the complexity of its two-loop structure. Consequently, GPA simplifies hyperparameter tuning and reduces memory requirements compared to DiLoCo in standard non-distributed settings.

Future work should validate GPA at scale across diverse model architectures and modalities, and explore its compatibility with other base optimizers (e.g., Shampoo, SOAP, Muon) and hyperparameter transfer techniques such as μP (Yang & Hu, 2021; Yang et al., 2022). Additionally, while our convergence bound partially explains the empirical results, it is limited to the convex setting and does not fully characterize when GPA can outperform the base optimizer.

Finally, GPA's decoupling of parameters also enables new avenues for distributed training. In DiLoCo, the number of inner steps serves as a coupled hyperparameter for both Lookahead with Nesterov and local SGD, leading to the undesirable finding that increasing the number of inner steps can improve convergence – contrary to standard local SGD intuition. By introducing a tunable, continuous smoothing parameter that is independent of the number of local SGD steps, GPA establishes a new foundation for rethinking DiLoCo and related averaging-based methods, especially in their integration with communication-efficient techniques like local SGD.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable second order optimization for deep learning. *arXiv preprint arXiv:2002.09018*, 2020.

Nicolo Cesa-Bianchi, Alex Conconi, and Claudio Gentile. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004.

Zachary Charles, Gabriel Teston, Lucio Dery, Keith Rush, Nova Fallen, Zachary Garrett, Arthur Szlam, and Arthur Douillard. Communication-efficient language model training scales reliably and robustly: Scaling laws for diloco. *arXiv preprint arXiv:2503.09799*, 2025.

George E. Dahl, Frank Schneider, Zachary Nado, Naman Agarwal, Chandramouli Shama Sastry, Philipp Hennig, Sourabh Medapati, Runa Eschenhagen, Priya Kasimbeg, Daniel Suo, Juhan Bae, Justin Gilmer, Abel L. Peirson, Bilal Khan, Rohan Anil, Mike Rabbat, Shankar Krishnan, Daniel Snider, Ehsan Amid, Kongtao Chen, Chris J. Maddison, Rakshith Vasudev, Michal Badura, Ankush Garg, and Peter Mattson. Benchmarking Neural Network Training Algorithms, 2023.

Aaron Defazio. On the curved geometry of accelerated optimization. *Advances in Neural Information Processing Systems 33 (NIPS 2019)*, 2019.

Aaron Defazio, Ashok Cutkosky, Harsh Mehta, and Konstantin Mishchenko. Optimal linear decay learning rate schedules and further refinements. *arXiv preprint arXiv:2310.07831*, 2023.

Aaron Defazio, Xingyu Yang, Harsh Mehta, Konstantin Mishchenko, Ahmed Khaled, and Ashok Cutkosky. The road less scheduled. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 9974–10007. Curran Associates, Inc., 2024.

Arthur Douillard, Qixuan Feng, Andrei A Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. DiLoCo: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023.

Arthur Douillard, Yanislav Donchev, Keith Rush, Satyen Kale, Zachary Charles, Zachary Garrett, Gabriel Teston, Dave Lacey, Ross McIlroy, Jiajun Shen, et al. Streaming DiLoCo with overlapping communication: Towards a distributed free lunch. *arXiv preprint arXiv:2501.18512*, 2025.

Runa Eschenhagen, Aaron Defazio, Tsung-Hsien Lee, Richard E Turner, and Hao-Jun Michael Shi. Purifying shampoo: Investigating shampoo's heuristics by decomposing its preconditioner. *arXiv preprint arXiv:2506.03595*, 2025.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1842–1850. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/gupta18a.html.

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. *arXiv preprint arXiv:2203.15556*, 2022.

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

Ahmed Khaled, Satyen Kale, Arthur Douillard, Chi Jin, Rob Fergus, and Manzil Zaheer. Understanding outer optimizers in local sgd: Learning rates, momentum, and acceleration. *arXiv preprint arXiv:2509.10439*, 2025.

Guanghui Lan. An optimal method for stochastic composite optimization. *Mathematical Programming*, 133(1):365–397, 2012.

Tim Large, Yang Liu, Minyoung Huh, Hyojin Bahng, Phillip Isola, and Jeremy Bernstein. Scalable optimization in the modular norm. *Advances in Neural Information Processing Systems*, 37: 73501–73548, 2024.

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*, 2024a.

Bo Liu, Rachita Chhaparia, Arthur Douillard, Satyen Kale, Andrei A. Rusu, Jiajun Shen, Arthur Szlam, and Marc'Aurelio Ranzato. Asynchronous Local-SGD training for language modeling. *arXiv preprint arXiv:2401.09135*, 2024b.

AI @ Meta Llama Team. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daniel Morales-Brotons, Thijs Vogels, and Hadrien Hendrikx. Exponential moving average of weights in deep learning: Dynamics and benefits. *arXiv preprint arXiv:2411.18704*, 2024.

Alex Nichol and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2(3):4, 2018.

Matteo Pagliardini, Pierre Ablin, and David Grangier. The AdEMAMix optimizer: Better, faster, older. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=jj7b3p5kLY.

Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. Training deep learning models with norm-constrained LMOs. *arXiv preprint arXiv:2502.07529*, 2025.

Boris Polyak. New stochastic approximation type procedures. *Avtomatica i Telemekhanika*, 7:98–107, 01 1990.

Boris T. Polyak and Anatoli B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

David Ruppert. Efficient estimations from a slowly convergent Robbins-Monro process. *Technical Report, Cornell University*, 02 1988.

Hao-Jun Michael Shi, Tsung-Hsien Lee, Shintaro Iwasaki, Jose Gallego-Posada, Zhijing Li, Kaushik Rangadurai, Dheevatsa Mudigere, and Michael Rabbat. A distributed data-parallel pytorch implementation of the distributed shampoo optimizer for training neural networks at-scale. *arXiv preprint arXiv:2309.06497*, 2023.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*. PMLR, 2013.

Wei Tao, Zhisong Pan, Gaowei Wu, and Qing Tao. Primal averaging: A new gradient evaluation step to attain the optimal individual convergence. *IEEE Transactions on Cybernetics*, PP:1–11, 10 2018. doi: 10.1109/TCYB.2018.2874332.

Benjamin Thérien, Xiaolong Huang, Irina Rish, and Eugene Belilovsky. MuLoCo: Muon is a practical inner optimizer for DiLoCo. *arXiv preprint arXiv:2505.23725*, 2025.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Mujin Kwun, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. SOAP: Improving and stabilizing Shampoo using Adam. *arXiv preprint arXiv:2409.11321*, 2024.

Greg Yang and Edward J. Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pp. 11727–11737. PMLR, 2021.

Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E. Hinton. Lookahead optimizer: k steps forward, 1 step back. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

A LLM USAGE

We used an internal AI assistant for revising the grammar and wording in the paper, and used Gemini Pro 2.5 to verify our proofs.

B ALGORITHMIC DETAILS

B.1 PSEUDOCODE FOR NON-DISTRIBUTED DILOCO / LOOKAHEAD WITH NESTEROV

We provide a complete description of non-distributed DiLoCo in Algorithm 2.

Algorithm 2 Non-Distributed DiLoCo / Lookahead with Nesterov

Input: Initial iterate $x^{(1)}$, inner learning rate schedule $\gamma^{(t)} > 0$, constant outer learning rate $\tilde{\gamma} > 0$, weight decay $\lambda \geq 0$, momentum parameter $\mu \in [0, 1)$, base optimizer `BaseOpt`.
$\tilde{x}^{(1)} = x^{(1)}$ ▷ Initialize slow model weights.
$b^{(0)} = 0 \in \mathbb{R}^n$ ▷ Initialize momentum buffer.
for step $t = 1, ..., T$ **do**
  Sample mini-batch $\xi^{(t)}$
  $g^{(t)} \in \partial f(x^{(t)}; \xi^{(t)})$
  $d^{(t)} = \texttt{BaseOpt}(g^{(t)})$ ▷ Computes base optimizer's search direction.
  $x^{(t+1)} = (1 - \gamma^{(t)}\lambda)x^{(t)} + \gamma^{(t)} d^{(t)}$ ▷ Updates inner model weights (with weight decay).
  if $t \bmod H = 0$ **then**
    $\tilde{g}^{(t)} = \tilde{x}^{(t)} - x^{(t+1)}$ ▷ Pseudo-gradient computation.
    $b^{(t+1)} = \mu b^{(t)} + p^{(t)}$ ▷ Accumulates outer momentum.
    $\tilde{x}^{(t+1)} = \tilde{x}^{(t)} - \tilde{\gamma}\left[\mu b^{(t)} + \tilde{g}^{(t)}\right]$ ▷ Nesterov-style parameter update.
    $x^{(t+1)} = \tilde{x}^{(t+1)}$ ▷ Re-initialize inner model weights.
  else
    $\tilde{x}^{(t+1)} = \tilde{x}^{(t)}$
  end if
end for
Returns: $\tilde{x}^{(T)}$

B.2 MEMORY-EFFICIENT FORMULATION OF GENERALIZED PRIMAL AVERAGING

The implementation of the original formulation of GPA in equation 8 requires storing two additional copies of the model's parameters during the optimizer step. This is because the gradient computation occurs on the $y^{(t)}$ sequence, which is computed from the two other sequences $x^{(t)}$ and $z^{(t)}$. To avoid this additional model copy, we can store $y^{(t)}$ instead, and recover $x^{(t)}$ from $y^{(t)}$ and $z^{(t)}$ during evaluation time.

To see how this can be done, we define the *memory-efficient formulation* of GPA as:

$$\begin{aligned} x^{(t)} &= \frac{1}{\mu_y} y^{(t)} + \left(1 - \frac{1}{\mu_y}\right) z^{(t)}, \\ y^{(t)} &= \mu_x y^{(t)} + (1 - \mu_x) z^{(t)} - (1 - \mu_x \mu_y)\gamma^{(t)} g(y^{(t)}; \xi^{(t)}), \\ z^{(t+1)} &= z^{(t)} - \gamma^{(t)} g(y^{(t)}; \xi^{(t)}). \end{aligned} \tag{9}$$

This reformulation is valid only when $\mu_y > 0$. In the $y^{(t)}$ update, the first term can be interpreted as interpolating $y^{(t)}$ towards $z^{(t)}$. The second term is a correction term that applies a dampened update on $y^{(t)}$.

Note that this formulation does not require the computation of $x^{(t)}$ except when necessary. Therefore, our implementation enables a training and evaluation mode similar to neural network modules like batch normalization that enables us to compute $x^{(t)}$ from $y^{(t)}$ and vice-versa. Specifically, when

switching from training to evaluation mode, we can compute $x^{(t)}$ from $y^{(t)}$ and $z^{(t)}$ by:

$$x^{(t)} = \frac{1}{\mu_y} y^{(t)} + \left(1 - \frac{1}{\mu_y}\right) z^{(t)}.$$

Similarly, when switching from evaluation to training mode, we can recover $y^{(t)}$ from $x^{(t)}$ and $z^{(t)}$ by:

$$y^{(t)} = \mu_y x^{(t)} + (1 - \mu_y) z^{(t)}.$$

A proof of the equivalence of these two formulations is provided in Appendix C. The complete pseudocode for arbitrary base optimizers are provided in Algorithm 3.

Algorithm 3 Memory-Efficient Generalized Primal Averaging (GPA)

Input: Initial iterate $y^{(1)}$, learning rate schedule $\gamma^{(t)} > 0$, weight decay $\lambda \geq 0$, interpolation parameters $\mu_x, \mu_y \in [0, 1)$, base optimizer `BaseOpt`.
$z^{(1)} = y^{(1)}$
for $t = 1, ..., T$ **do**
  $g^{(t)} \in \partial f(y^{(t)}; \xi^{(t)})$
  $d^{(t)} = \texttt{BaseOpt}(g^{(t)})$
  $y^{(t)} = \mu_x y^{(t)} + (1 - \mu_x) z^{(t)} + \gamma^{(t)}(1 - \mu_x \mu_y)(d^{(t)} + \lambda z^{(t)})$
  $z^{(t+1)} = (1 - \gamma^{(t)}\lambda) z^{(t)} - \gamma^{(t)} d^{(t)}$
end for
Returns: $x^{(T)} = \frac{1}{\mu_y} y^{(T)} + \left(1 - \frac{1}{\mu_y}\right) z^{(T)}$.

B.3 Compatibility with Modular Norm Theory

Recent work on Muon and similar methods have built on modular norm theory, which suggests that the design of optimization methods for deep learning should constrain the modular norm of the model parameters in order to enable hyperparameter transferability and bounded Lipschitz constants (Large et al., 2024; Jordan et al., 2024; Pethick et al., 2025). Here, we argue that GPA, by definition, preserves these norm constraints.

To see this, assume that $d^{(t)}$ is the search direction for a single parameter that it is constrained with respect to some norm, i.e., $\|d^{(t)}\| \leq M$ for some constant $M \geq 0$. (Typically, we assume it is the RMS-to-RMS norm or similar.) We can preserve these norm constraints on the iterates produced by GPA since:

$$\begin{aligned}
\|y^{(t)}\| &\leq \mu_y \|x^{(t)}\| + (1 - \mu_y)\|z^{(t)}\| \\
\|z^{(t+1)}\| &\leq (1 - \lambda\gamma^{(t)})\|z^{(t)}\| + \gamma^{(t)}\|d^{(t)}\| \\
\|x^{(t+1)}\| &\leq \mu_x \|x^{(t)}\| + (1 - \mu_x)\,\|z^{(t+1)}\|.
\end{aligned}$$

Since $\mu_x, \mu_y \in [0, 1]$, we can see that if $\max\left\{\|x^{(t)}\|, \|y^{(t)}\|, \|z^{(t)}\|\right\} \leq M'$ for $M' \geq 0$, then $\max\left\{\|x^{(t+1)}\|, \|y^{(t+1)}\|, \|z^{(t+1)}\|\right\} \leq (1 - \lambda\gamma^{(t)})M' + \gamma^{(t)} M$, which is the same bound we would obtain for the base optimizer.

C Proofs

C.1 Equivalence Between Nesterov's Formulations

Proposition 2. *Given fixed learning rates $\gamma_{\text{primal}}, \gamma_{\text{modern}} > 0$, Nesterov's primal averaging formulation (equation 4) is equivalent to Nesterov's modern formulation (equation 3) in the sense that*

$$y^{(t)}_{\text{primal}} = x^{(t)}_{\text{modern}} \quad \textit{and} \quad b^{(t)}_{\text{modern}} = \frac{1}{(1 - \mu)\,\gamma_{\text{primal}}} \left(x^{(t)}_{\text{primal}} - x^{(t+1)}_{\text{primal}}\right), \tag{10}$$

when $\mu_{\text{primal}} = \mu_{\text{modern}} = \mu$ and $(1 - \mu)\,\gamma_{\text{primal}} = \gamma_{\text{modern}}$.

Proof. We can prove this by induction. For simplicity of notation, we will use $x_m = x_{\text{modern}}$ and $x_p = x_{\text{primal}}$ and similar for all variables.

For the base case, note that the initializations $z_p^{(1)} = x_p^{(1)} = x_m^{(1)}$ are equal. Therefore,

$$y_p^{(1)} = \mu x_p^{(1)} + (1-\mu) z_p^{(1)} = x_m^{(1)}, \tag{11}$$

as desired. In addition, since $b_m^{(1)} = \mu b_m^{(0)} + g(x_m^{(1)}; \xi^{(1)}) = g(x_m^{(1)})$, we can see that:

$$\begin{aligned} x_p^{(1)} - x_p^{(2)} &= (1-\mu) x_p^{(1)} - (1-\mu) z_p^{(1)} \\ &= (1-\mu)(x_p^{(1)} - z_p^{(2)}) \\ &= (1-\mu)(x_p^{(1)} - z_p^{(1)} + \gamma_p g(y_p^{(1)}; \xi^{(1)})) \\ &= (1-\mu)\gamma_p g(y_p^{(1)}; \xi^{(1)}). \end{aligned}$$

The base case for the momentum buffer $b_m^{(1)}$ follows from rearranging the equation with equation 11 and observing that $b_m^{(1)} = \mu b_m^{(0)} + g(x_m^{(1)}; \xi^{(1)}) = g(x_m^{(1)}; \xi^{(1)})$.

For the inductive step, assume that equation 10 holds for t. Then from the inductive hypothesis, we can show that:

$$\begin{aligned} x_m^{(t+1)} &= x_m^{(t)} - \gamma_m [\mu b_m^{(t)} + g(x_m^{(t)}; \xi^{(t)})] \\ &= y_p^{(t)} - (1-\mu)\gamma_p \left[\mu \left(\frac{1}{(1-\mu)\gamma_p} (x_p^{(t)} - x_p^{(t+1)}) \right) + g(y_p^{(t)}; \xi^{(t)}) \right] \\ &= y_p^{(t)} - \mu(x_p^{(t)} - x_p^{(t+1)}) - (1-\mu)\gamma g(y_p^{(t)}; \xi^{(t)}). \end{aligned} \tag{12}$$

From the primal averaging form in equation 4, we can derive that:

$$\begin{aligned} x_p^{(t+1)} &= \mu x_p^{(t)} + (1-\mu) z_p^{(t+1)} \\ &= \mu x_p^{(t)} + (1-\mu)(z_p^{(t)} - \gamma_p g(y_p^{(t)}; \xi^{(t)}) \\ &= y_p^{(t)} - (1-\mu)\gamma_p g(y_p^{(t)}; \xi^{(t)}). \end{aligned} \tag{13}$$

Rearranging equation 13, we get that:

$$y_p^{(t)} - x_p^{(t+1)} = (1-\mu)\gamma_p g(y_p^{(t)}; \xi^{(t)}). \tag{14}$$

Plugging in equation 14 into equation 12, we obtain:

$$x_m^{(t+1)} = y_p^{(t)} - \mu(x_p^{(t)} - x_p^{(t+1)}) - (y_p^{(t)} - x_p^{(t+1)}) = (1+\mu) x_p^{(t+1)} - \mu x_p^{(t)}. \tag{15}$$

Finally, since $x_p^{(t+1)} = \mu x_p^{(t)} + (1-\mu) z_p^{(t)}$, $(1-\mu) z_p^{(t+1)} = x_p^{(t+1)} - \mu x_p^{(t)}$. Therefore, to see $x_m^{(t+1)}$'s equivalence to $y_p^{(t+1)}$,

$$\begin{aligned} y_p^{(t+1)} &= \mu x_p^{(t+1)} + (1-\mu) z_p^{(t+1)} \\ &= \mu x_p^{(t+1)} + x_p^{(t+1)} - \mu x_p^{(t)} \\ &= (1+\mu) x_p^{(t+1)} - \mu x_p^{(t)}. \end{aligned} \tag{16}$$

Combining equations 15 and 16 gives the result.

To prove that $b_m^{(t+1)} = \frac{1}{(1-\mu)\gamma_p}(x_p^{(t+1)} - x_p^{(t+2)})$, note that:

$$b_m^{(t+1)} = \mu b_m^{(t)} + g(x_m^{(t+1)}; \xi^{(t+1)}) = \frac{\mu}{(1-\mu)\gamma_p}(x_p^{(t)} - x_p^{(t+1)}) + g(y_p^{(t+1)}; \xi^{(t+1)}). \tag{17}$$

To get an expression for $x_p^{(t+1)} - x_p^{(t+2)}$, note that:

$$\begin{aligned} x_p^{(t+2)} &= \mu x_p^{(t+1)} + (1-\mu)(z_p^{(t+1)} - \gamma_p g(y_p^{(t+1)}; \xi^{(t+1)})) \\ &= (\mu x_p^{(t+1)} + (1-\mu) z_p^{(t+1)}) - (1-\mu)\gamma_p g(y_p^{(t+1)}; \xi^{(t+1)}) \\ &= y_p^{(t+1)} - (1-\mu)\gamma_p g(y_p^{(t+1)}; \xi^{(t+1)}) \\ &= ((1+\mu) x_p^{(t+1)} - \mu x_p^{(t)}) - (1-\mu)\gamma_p g(y_p^{(t+1)}; \xi^{(t+1)}), \end{aligned} \tag{18}$$

where equation 18 follows from equation 16. Therefore, plugging-in equation 18 into $x_p^{(t+1)} - x_p^{(t+2)}$ gives:

$$x_p^{(t+1)} - x_p^{(t+2)} = -\mu(x_p^{(t+1)} - x_p^{(t)}) + (1-\mu)\gamma_p g(y_p^{(t+1)}; \xi^{(t+1)}). \tag{19}$$

The result follows from expanding equation 17 as:

$$\begin{aligned} b_m^{(t+1)} &= \frac{1}{(1-\mu)\gamma_p}\left[-\mu(x_p^{(t+1)} - x_p^{(t)}) + (1-\mu)\gamma_p g(y_p^{(t+1)}; \xi^{(t+1)})\right] \\ &= \frac{1}{(1-\mu)\gamma_p}(x_p^{(t+1)} - x_p^{(t+2)}). \end{aligned}$$

□

C.2 Equivalence Between Generalized Primal Averaging Formulations

Proposition 3. *Let $\mu_y > 0$. Then GPA (equation 8) is equivalent to the memory-efficient formulation (equation 9).*

Proof. Note that it is sufficient to show that:

$$x^{(t)} = \frac{1}{\mu_y} y^{(t)} + \left(1 - \frac{1}{\mu_y}\right) z^{(t)}, \tag{20}$$

$$y^{(t+1)} = \mu_x y^{(t)} + (1-\mu_x) z^{(t)} - (1-\mu_x\mu_y)\gamma^{(t)} g(y^{(t)}; \xi^{(t)}). \tag{21}$$

To prove equation 20, note that we can re-write $x^{(t)}$ as a function of $y^{(t)}$ and $z^{(t)}$, i.e., since

$$y^{(t)} = \mu_y x^{(t)} + (1-\mu_y) z^{(t)}$$

and $\mu_y > 0$, we have that

$$x^{(t)} = \frac{1}{\mu_y} y^{(t)} + \frac{1}{\mu_y}(\mu_y - 1) z^{(t)} = \frac{1}{\mu_y} y^{(t)} + \left(1 - \frac{1}{\mu_y}\right) z^{(t)}.$$

To prove equation 20, we can re-write equation 20 as

$$\mu_y x^{(t+1)} = \mu_y z^{(t+1)} + (y^{(t+1)} - z^{(t+1)}) = y^{(t+1)} - (1-\mu_y) z^{(t+1)}. \tag{22}$$

Similarly, by plugging in the original $x^{(t+1)}$ update, i.e., $x^{(t+1)} = \mu_x x^{(t)} + (1-\mu_x) z^{(t)}$, we also have:

$$\mu_y x^{(t+1)} = \mu_y(\mu_x x^{(t)} + (1-\mu_x) z^{(t)}) = \mu_x\mu_y x^{(t)} + (1-\mu_x)\mu_y z^{(t+1)}. \tag{23}$$

Combining these two equalities in equations 22 and 23 and rearranging, we get:

$$y^{(t+1)} = \mu_x\mu_y x^{(t)} + (1-\mu_x\mu_y) z^{(t+1)}. \tag{24}$$

Plugging-in equation 20 and the update $z^{(t+1)} = z^{(t)} - \gamma^{(t)} g(y^{(t)}; \xi^{(t)})$ from equation 8 into equation 24, we obtain:

$$\begin{aligned} y^{(t+1)} &= \mu_x\mu_y\left(\frac{1}{\mu_y} y^{(t)} + \left(1 - \frac{1}{\mu_y}\right) z^{(t)}\right) + (1-\mu_x\mu_y)(z^{(t)} - \gamma^{(t)} g(y^{(t)}; \xi^{(t)})) \\ &= \mu_x y^{(t)} + (1-\mu_x) z^{(t)} - (1-\mu_x\mu_y)\gamma^{(t)} g(y^{(t)}; \xi^{(t)}), \end{aligned}$$

as desired. □

C.3 Convergence Bounds Based On Online-to-Batch Theory

Our proofs similarly rely on the online-to-batch conversion theory used in Defazio et al. (2024).

Lemma 1. *Suppose we define $w^{(t)}$ as the weighting:*

$$w^{(t)} = \begin{cases} 1 & \text{if } t = 1, \\ (1-\mu_x)\,\mu_x^{-t+1} & \text{if } t > 1. \end{cases}$$

Then the model evaluation sequence $x^{(t)}$ is equivalent to the weighted average:

$$x^{(t+1)} = \frac{\sum_{i=1}^{t} w^{(i)}}{\sum_{i=1}^{t+1} w^{(i)}} x^{(t)} + \frac{w^{(t+1)}}{\sum_{i=1}^{(t+1)} w^{(i)}} z^{(t+1)} = \frac{w^{(1:t)}}{w^{(1:t+1)}} x^{(t)} + \frac{w^{(t+1)}}{w^{(1:t+1)}} z^{(t+1)},$$

with

$$w^{(1:t)} = \sum_{s=1}^{t} w^{(s)} = \mu_x^{-t+1}.$$

Furthermore, $x^{(t)}$ can be expressed as the closed form expression:

$$x^{(t)} = \mu_x^{t-1} \sum_{s=1}^{t} w^{(s)} z^{(s)}.$$

Theorem 2. *Let F be a convex function, and assume that there exists a minimizer x_* that minimizes F. Let $\xi^{(1)}, \ldots, \xi^{(T)}$ be a sequence of i.i.d. random variables. Suppose that we are given arbitrary updates $z^{(1)}, \ldots, z^{(T)}$ from a base optimizer within the Generalized Primal Averaging framework (Equation 8). Then for $\mu_x, \mu_y \in [0, 1)$ and average iterate $\bar{x}^{(T)} = \frac{1}{T} \sum_{t=1}^{T} x^{(t)}$, we have the bound*

$$\begin{aligned}
\mathbb{E}[F(\bar{x}^{(T)}) - F(x_*)] \leq \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[\langle \nabla F(y^{(t)}), z^{(t)} - x_* \rangle] \\
+ \frac{\mu_x}{1-\mu_x} \frac{1}{T} \mathbb{E}\left[F(x^{(1)}) - F(x_*)\right] \\
- \frac{1}{1-\mu_y} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[B_F(y^{(t)}, x^{(t)})] - \frac{\mu_y}{1-\mu_y} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[B_F(x^{(t)}, y^{(t)})] \\
- \frac{\mu_x}{1-\mu_x} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[B_F(x^{(t-1)}, x^{(t)})].
\end{aligned}$$

Proof. We start with the same analysis as in the Schedule-Free work (Defazio et al., 2024). Notice that by definition of $x^{(t)}$, it holds $w^{(1:t-1)}(x^{(t)} - x^{(t-1)}) = w^{(t)}(z^{(t)} - x^{(t)})$. Therefore,

$$\begin{aligned}
w^{(1:t)} F(x^{(t)}) - w^{(1:t-1)} F(x^{(t-1)}) - w^{(t)} F(x_*) \\
&= w^{(1:t-1)}(F(x^{(t)}) - F(x^{(t-1)})) + w^{(t)}(F(x^{(t)}) - F(x_*)) \\
&= w^{(1:t-1)}(\langle \nabla F(x^{(t)}), x^{(t)} - x^{(t-1)} \rangle - B_F(x^{(t-1)}, x^{(t)})) + w^{(t)}(F(x^{(t)}) - F(x_*)) \\
&= w^{(t)} \langle \nabla F(x^{(t)}), z^{(t)} - x^{(t)} \rangle - w^{(1:t-1)} B_F(x^{(t-1)}, x^{(t)}) + w^{(t)}(F(x^{(t)}) - F(x_*)).
\end{aligned}$$

Next, we observe that by definition of $y^{(t)}$, it holds $z^{(t)} - y^{(t)} = \frac{\mu_y}{1-\mu_y}(y^{(t)} - x^{(t)})$, and, thus,

$$\begin{aligned}
&\langle \nabla F(x^{(t)}), z^{(t)} - x^{(t)} \rangle \\
&= \langle \nabla F(x^{(t)}) - \nabla F(y^{(t)}), z^{(t)} - y^{(t)} \rangle + \langle \nabla F(y^{(t)}), z^{(t)} - y^{(t)} \rangle \\
&\quad + \langle \nabla F(x^{(t)}), y^{(t)} - x^{(t)} \rangle \\
&= \frac{\mu_y}{1-\mu_y} \langle \nabla F(x^{(t)}) - \nabla F(y^{(t)}), y^{(t)} - x^{(t)} \rangle + F(x_*) - F(y^{(t)}) - B_F(x_*, y^{(t)}) + \langle \nabla F(y^{(t)}), z^{(t)} - x_* \rangle \\
&\quad + F(y^{(t)}) - F(x^{(t)}) - B_F(y^{(t)}, x^{(t)}) \\
&\leq -\frac{\mu_y}{1-\mu_y}(B_F(x^{(t)}, y^{(t)}) + B_F(y^{(t)}, x^{(t)})) + F(x_*) - F(x^{(t)}) - B_F(y^{(t)}, x^{(t)}) + \langle \nabla F(y^{(t)}), z^{(t)} - x_* \rangle \\
&= -\frac{\mu_y}{1-\mu_y} B_F(x^{(t)}, y^{(t)}) - \frac{1}{1-\mu_y} B_F(y^{(t)}, x^{(t)}) + F(x_*) - F(x^{(t)}) + \langle \nabla F(y^{(t)}), z^{(t)} - x_* \rangle,
\end{aligned}$$

where the inequality step used $-B_F(x_*, y^{(t)}) \leq 0$, which follows from convexity of F. Plugging this back, we obtain

$$\begin{aligned}
&w^{(1:t)}F(x^{(t)}) - w^{(1:t-1)}F(x^{(t-1)}) - w^{(t)}F(x_*) \\
&\qquad \leq -w^{(t)}\frac{\mu_y}{1-\mu_y}B_F(x^{(t)}, y^{(t)}) - \frac{w^{(t)}}{1-\mu_y}B_F(y^{(t)}, x^{(t)}) + w^{(t)}(F(x_*) - F(x^{(t)})) \\
&\qquad\quad + w^{(t)}\langle \nabla F(y^{(t)}), z^{(t)} - x_*\rangle - w^{(1:t-1)}B_F(x^{(t-1)}, x^{(t)}) + w^{(t)}(F(x^{(t)}) - F(x_*)) \\
&\qquad = w^{(t)}\langle \nabla F(y^{(t)}), z^{(t)} - x_*\rangle - \frac{w^{(t)}}{1-\mu_y}B_F(y^{(t)}, x^{(t)}) \\
&\qquad\quad - \frac{w^{(t)}\mu_y}{1-\mu_y}B_F(x^{(t)}, y^{(t)}) - w^{(1:t-1)}B_F(x^{(t-1)}, x^{(t)}).
\end{aligned} \tag{25}$$

We may adapt this bound to our setting by using an exponentially increasing weighting sequence, given by Lemma 1. Using those weights, we have simplified expressions for the following quantities:

$$\begin{aligned}
\frac{w^{(1:t)}}{w^{(t)}} &= \frac{\mu_x^{-t+1}}{(1-\mu_x)\,\mu_x^{-t+1}} = \frac{1}{1-\mu_x}, \\
\frac{w^{(1:t-1)}}{w^{(t)}} &= \frac{\mu_x^{-(t-1)+1}}{(1-\mu_x)\,\mu_x^{-t+1}} = \frac{\mu_x}{1-\mu_x},
\end{aligned}$$

with a special case for the first iterate $\frac{w^{(1:1)}}{w^{(1)}} = 1$ and $\frac{w^{(1:t-1)}}{w^{(1)}} = 0$.

To obtain an average regret bound, we divide Equation 25 by $w^{(t)}$, take expectation, and sum from 1 to T. The left-hand side is a telescoping sum, which we can simplify as follows:

$$\begin{aligned}
&\sum_{t=1}^{T}\left[\frac{w^{(1:t)}}{w^{(t)}}\mathbb{E}[F(x^{(t)})] - \frac{w^{(1:t-1)}}{w^{(t)}}\mathbb{E}[F(x^{(t-1)})]\right] - TF(x_*) \\
&= F(x^{(1)}) - \frac{w^{(1:1)}}{w^{(2)}}F(x^{(1)}) + \frac{1}{1-\mu_x}\sum_{t=2}^{T}\mathbb{E}[F(x^{(t)})] - \frac{\mu_x}{1-\mu_x}\sum_{t=2}^{T-1}\mathbb{E}[F(x^{(t)})] - TF(x_*) \\
&= F(x^{(1)}) - \frac{1}{(1-\mu_x)\,\mu_x^{-1}}F(x^{(1)}) + \frac{1}{1-\mu_x}\mathbb{E}[F(x^{(T)})] + \sum_{t=2}^{T-1}\left(\frac{1}{1-\mu_x} - \frac{\mu_x}{1-\mu_x}\right)\mathbb{E}[F(x^{(t)})] - TF(x_*) \\
&= F(x^{(1)}) - \frac{\mu_x}{1-\mu_x}F(x^{(1)}) + \frac{1}{1-\mu_x}\mathbb{E}[F(x^{(T)})] + \sum_{t=2}^{T-1}\left(\frac{1}{1-\mu_x} - \frac{\mu_x}{1-\mu_x}\right)\mathbb{E}[F(x^{(t)})] - TF(x_*) \\
&= -\frac{\mu_x}{1-\mu_x}F(x^{(1)}) + \frac{\mu_x}{1-\mu_x}\mathbb{E}[F(x^{(T)})] + \sum_{t=1}^{T}\mathbb{E}[F(x^{(t)})] - TF(x_*).
\end{aligned}$$

Plugging-in this simplified expression, moving the extra $F(x^{(1)}) - F(x^{(t)})$ term to the right-hand side, and simplifying gives:

$$\begin{aligned}
\sum_{t=1}^{T}\mathbb{E}\left[F(x^{(t)}) - F(x_*)\right] &\leq \sum_{t=1}^{T}\mathbb{E}[\langle \nabla F(y^{(t)}), z^{(t)} - x_*\rangle] + \frac{\mu_x}{1-\mu_x}\mathbb{E}\left[F(x^{(1)}) - F(x^{(T)})\right] \\
&\quad - \frac{1}{1-\mu_y}\sum_{t=1}^{T}\mathbb{E}[B_F(y^{(t)}, x^{(t)})] - \frac{\mu_y}{1-\mu_y}\sum_{t=1}^{T}\mathbb{E}[B_F(x^{(t)}, y^{(t)})] \\
&\quad - \frac{\mu_x}{1-\mu_x}\sum_{t=1}^{T}\mathbb{E}[B_F(x_{t-1}, x^{(t)})].
\end{aligned}$$

We get a bound on the average iterate $\bar{x}_T = \sum_{t=1}^{T} x^{(t)}$ by dividing by T and applying Jensen's inequality:

$$\mathbb{E}[F(\bar{x}_T) - F(x_*)] \leq \frac{1}{T}\mathbb{E}\sum_{t=1}^{T}\langle \nabla F(y^{(t)}), z^{(t)} - x_* \rangle + \frac{\mu_x}{1-\mu_x}\frac{1}{T}\mathbb{E}\left[F(x^{(1)}) - F(x^{(T)})\right]$$
$$- \frac{1}{1-\mu_y}\frac{1}{T}\mathbb{E}\sum_{t=1}^{T} B_F(y^{(t)}, x^{(t)}) - \frac{\mu_y}{1-\mu_y}\frac{1}{T}\mathbb{E}\sum_{t=1}^{T} B_F(x^{(t)}, y^{(t)})$$
$$- \frac{\mu_x}{1-\mu_x}\frac{1}{T}\mathbb{E}\sum_{t=1}^{T} B_F(x_{t-1}, x^{(t)}).$$

Finally, we use $F(x_*) \leq F(x^{(T)})$ to get the claimed bound. □

Corollary 2. *Assume that the base optimizer has regret guarantees* $\sum_{t=1}^{T}\mathbb{E}[\langle \nabla F(y^{(t)}), z^{(t)} - x_* \rangle] = \mathcal{O}(\sqrt{T})$. *Then:*

$$\mathbb{E}[F(\bar{x}^{(T)}) - F(x_*)] = \mathcal{O}\left(\frac{1}{\sqrt{T}}\right).$$

Proof. Note that we can upper bound the inequality in Theorem 1 by ignoring the negative Bregman divergence terms, i.e.,

$$\mathbb{E}[F(\bar{x}^{(T)}) - F(x_*)] \leq \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[\langle \nabla F(y^{(t)}), z^{(t)} - x_* \rangle] + \frac{\mu_x}{1-\mu_x}\frac{1}{T}\mathbb{E}\left[F(x^{(1)}) - F(x_*)\right].$$

The result follows from noting that the first term is $\mathcal{O}(1/\sqrt{T})$ and the second term is $\mathcal{O}(1/T)$. □

D Experimental Details

D.1 Comparison Between GPA and Nesterov

In order to validate that DiLoCo's performance can only be matched or improved upon with decoupled interpolation constants in GPA, we test the case where $\mu_x = \mu_y$, which corresponds to Nesterov's primal averaging formulation in equation 4. Here, we apply the same heuristic for $\mu_x = \mu^{1/H}$ also to μ_y and tune the learning rate.

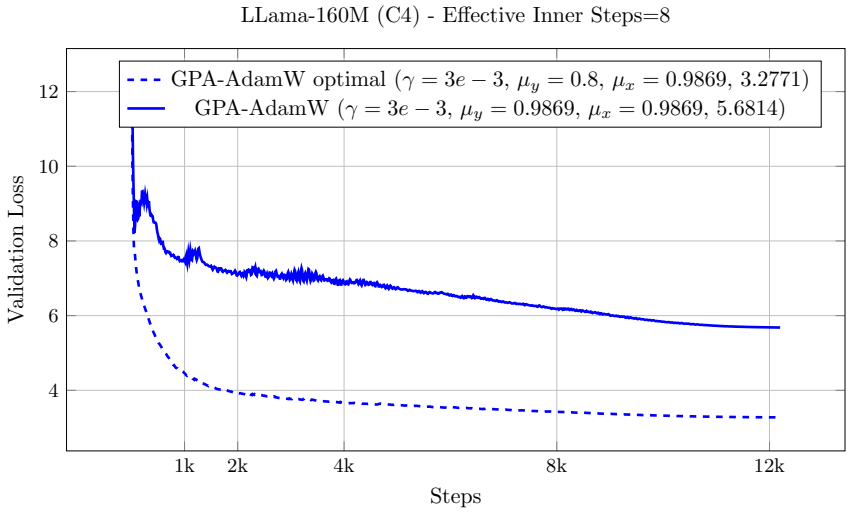


Figure 4: Comparison between Nesterov's primal averaging formulation with coupled constants $\mu_x = \mu_y$ and GPA with decoupled constants.

In Figure 4, we observe that coupling the interpolation constants is sub-optimal, and decoupling these coefficients is indeed necessary for optimal performance from GPA.

D.2 Additional Validation Loss Curves for Different Effective Number of Inner Steps

In Figures 5 and 6, we provide additional validation loss curves for the cases where the effective number of inner steps equals 8 or 32, respectively. The results are generally consistent with the case where the number of inner steps is equal to 16 in Figure 3. When the effective number of inner steps is 32, we observe that AdamW outperforms DiLoCo for approximately the first 2,000 steps.

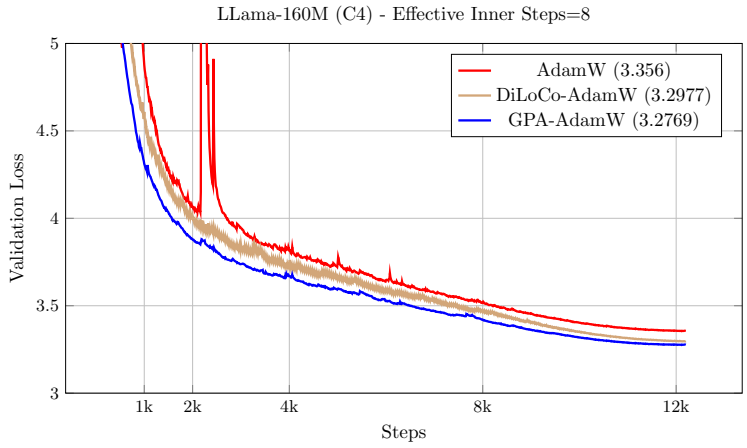


Figure 5: Validation loss versus steps for GPA, DiLoCo and AdamW when the effective number of inner steps equals 8.

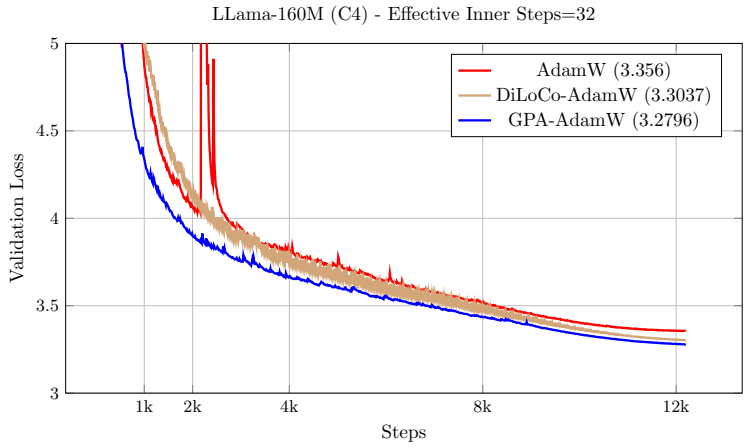


Figure 6: Validation loss versus steps for GPA, DiLoCo and AdamW when the effective number of inner steps equals 32.

D.3 Hyperparameter Sweeps

In this section, we summarize the hyperparameter sweeps used in our experiments in Table 2. In Table 3, we provide a table of conversions from optimal choices of μ and H in DiLoCo to GPA's choice of μ_x.

Table 2: Summary of hyperparameter sweeps used in the experiments.

Hyperparameter	**AdamW**	**DiLoCo-AdamW**	**GPA-AdamW**
Batch size	262K tokens	262K tokens	262K tokens
Sequence length	2048	2048	2048
Weight decay	0.1	0.1	0.1
Total training tokens	3.2B	3.2B	3.2B
Total training steps	12208	12208	12208
Inner optimizer	AdamW	AdamW	GPA-AdamW
Inner optimizer lr	5e-5, 1e-4, 2e-4, 3e-4, 5e-4, 7e-4, 1e-3, 3e-3	5e-4, 7e-4, 1e-3, 3e-3, 5e-3, 8e-3, 1e-2, 3e-2	5e-4, 7e-4, 1e-3, 3e-3, 5e-3, 8e-3, 1e-2, 3e-2
Inner Adam β_1	0.9	0.9	0.5, 0.7, 0.9
Inner Adam β_2	0.999	0.999	0.999
Inner Adam ϵ	10^{-8}	10^{-8}	10^{-8}
Warmup fraction	10%	10%	10%
Learning rate schedule	cosine	cosine	cosine
Learning rate min fraction %	0.01	0.01	0.01
GPA coeff μ_y	-	-	0.8, 0.9, 0.95, 0.9740, 0.9869, 0.99, 0.9913, 0.9934, 0.9956,0.9967, 0.9978, 0.9984, 0.9989, 0.9992
GPA coeff μ_x	-	-	0.9, 0.9740, 0.9869, 0.9934, 0.9967, 0.9984, 0.9992
Outer optimizer	-	Nesterov	-
Outer lr	-	0.25, 0.5, 0.75, 1.0	-
Outer momentum	-	0.7, 0.9, 0.95, 0.9913, 0.9967, 0.9984, 0.9989, 0.9992	-
Communication frequency H	-	1, 8, 16, 32, 64, 128	-

Table 3: Correspondence between the number of inner steps H and momentum coefficient μ_{diloco} in DiLoCo and the momentum coefficient μ_x in GPA. The values of μ_x were computed using the expression $\mu_x = \mu_{\text{diloco}}^{1/H}$, with $\mu_{\text{diloco}} = 0.9$ and H as the number of inner steps.

Number of inner steps (DiLoCo)	μ_x **(GPA)**
1	0.9000
4	0.9740
8	0.9869
16	0.9934
32	0.9967
64	0.9984
128	0.9992