PASO: STEP PARALLEL STOCHASTIC OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper approaches the fundamental challenge of accelerating the inherently autoregressive nature of gradient descent (GD) like SGD and Adam through a dynamic system perspective. Specifically, we introduce a unified framework that recasts the autoregressive GD process as solving a system of triangular nonlinear equations (TNEs), thereby facilitating a paradigm shift toward non-autoregressive GD featuring parallel gradient computation across iteration steps. Within this generic framework, we establish that: (1) the TNE system admits a unique solution corresponding precisely to the autoregressive GD iterative trajectory; (2) solving the TNEs system guarantees convergence to the GD iterative trajectory in equal or far fewer iterations. Building on these insights, we present *PASO*, a step parallel optimizer for accelerating a broad class of autoregressive GD optimizers like SGD and Adam. Extensive experiments (*e.g.*, Llama-3.2-1B) validate that PASO achieves up to **91**$\times$ reduction in GD steps and **7.5**$\times$ speedup in wall-clock time, with no measurable model quality loss. The source code will be released publicly.

1 INTRODUCTION

Stochastic gradient descent (SGD) (Robbins and Monro, 1951) and its variants (Kingma and Ba, 2014; Duchi et al., 2011; Tieleman, 2012; Loshchilov and Hutter, 2017; Liu et al., 2025; Nesterov, 1983; Pagliardini et al., 2024; Hwang, 2024), continue to be fundamental optimization engines for training deep neural networks. These methods underpin breakthroughs across domains (Vaswani et al., 2017; He et al., 2016; Shi et al., 2022; Lu et al., 2024; 2023), exemplified by large language models (LLMs) (Brown et al., 2020; OpenAI, 2023a; Touvron et al., 2023) and computer vision systems (Radford et al., 2021; Rombach et al., 2022; Lu et al., 2025). At its essence, SGD operates through iterative parameter adjustments where each iterative step follows the negative gradient direction of a randomly sampled data batch. To produce high-quality models, however, SGD typically necessitates an enormous number of iterative steps involving repeated forward and backward passes through massive datasets, resulting in prolonged training durations. For instance, training modern LLMs like DeepSeek-V3 (Liu et al., 2024) and GPT-4 (OpenAI, 2023b) often demands hundreds of thousands to millions of iteration steps. As a result, training a large-scale model consumes millions of GPU hours, roughly amounting to 1-3 months or longer (OpenAI, 2023b; Liu et al., 2024). Therefore, the staggering SGD steps caused by the exponential growth of model and dataset sizes have made a fundamental efficiency bottleneck.

Existing efforts to accelerate SGD training follow two main paradigms. The first develops more advanced optimizers(Zhang et al., 2025; Robert et al., 2025; Cheng and Glasgow, 2025; Hwang, 2024; Duchi et al., 2011; Liu et al., 2025; Polyak, 1964) (e.g., Adam (Kingma and Ba, 2014)) to accelerate convergence through refined update rules. However, the reduction in iterative steps is modest at best, as these techniques continue to rely on the sequential step-by-step execution of SGD. The second strategy develops parallel SGD, where workers update the model synchronously or asynchronously. Synchronous methods, typical in distributed learning, require waiting for all nodes to compute gradients before each update, while asynchronous approaches (e.g., DC-ASGD (Zheng et al., 2017) and HOGWILD! (Recht et al., 2011a)) allow workers to update global parameters independently. However, they risk harming model performance because of stale gradients. More critically, their parallelization is confined to intra-step operations, while leaving the inter-step sequential dependency intact, thereby maintaining the total number of training iterations unchanged.

This paper investigates a critical question: *can we drastically reduce gradient descent steps by parallelizing the step execution without sacrificing model performance?* At first glance, the challenge seems insurmountable—GD is inherently sequential, bound by a rigid Markov dependency chain (Zinkevich et al., 2010a). Surprisingly, we demonstrate that it is possible to completely sever this dependency chain to enable fully parallel gradient computation across different GD steps. In particular, we approach this problem through the lens of non-linear equations, treating the points on the GD iteration trajectory as mutually independent unknown variables. This independence thus naturally eradicates the sequential dependencies between iterations. By solving this system of equations, we achieve fully parallel gradient computation across all steps. Empirical results show this approach converges in far fewer iterations compared to standard sequential GD. Furthermore, our framework is inherently orthogonal to existing approaches, allowing seamless integration with both sequential optimizers (e.g., Adam) and parallel GD variants (e.g., model, pipeline, and data parallelism).

In summary, this paper contributes the following theoretical and practical advancements:

- we present PASO, an innovative step-level parallel GD paradigm, through transforming the autoregressive GD process into solving a system of triangular nonlinear equations;
- we establish that PASO exhibits guaranteed convergence to the GD trajectory points with iteration complexity equal to or surpassing that of the autoregressive gradient descent;
- our comprehensive evaluation showcases that PASO reduces iteration steps by up to **91**$\times$ and accelerates wall-clock time by **7.5**$\times$, all without sacrificing quality.

2 Related Work

Model Parallelism. Model parallelism (Jia et al., 2019; Narayanan et al., 2021; Xu et al., 2021; Yuan et al., 2021; Rajbhandari et al., 2020; Ren et al., 2021; Xu et al., 2020; Gao et al., 2025) shards a neural network's parameters across multiple devices to accommodate models too large for a single device's memory. During training, all devices perform partial computations in parallel over its designated subset of parameters. Subsequently, through collective communication methods like NCCL (NVIDIA, 2023), the results from each device are aggregated to compute the global gradient, which is then used to update the parameters. Early works by (Dean et al., 2012; Chilimbi et al., 2014; Xing et al., 2015) introduced model parallelism by partitioning model parameters across machines. Recent advances include Mesh-TensorFlow (Shazeer et al., 2018), PyTorch's fully sharded data parallel (Zhao et al., 2023), and Megatron-LM (Shoeybi et al., 2019) which efficiently parallelizes large models to achieve substantial speedups. Despite enabling the training of enormous models, these frameworks only address model parallelization within one step of gradient descent.

Pipeline Parallelism. Pipeline parallelism(He et al., 2021; Kim et al., 2020; Li et al., 2021; Sun et al., 2025; Zhao et al.; Tang et al.) aims to reduce idle time by partitioning models among workers per the direction of data flow into multi-stages and processing micro-batches in an interleaved manner. GPipe (Huang et al., 2019) introduced gradient accumulation for consistency across pipeline stages, while PipeDream (Narayanan et al., 2019) improved efficiency with "1F1B" scheduling and weight stashing. Despite these innovations, pipeline parallelism maintains sequential dependencies between gradient steps, as each micro-batch must wait for previous steps' gradients to update parameters.

Data Parallelism. Data parallelism partitions the training data across multiple workers (e.g., GPUs or nodes), and each worker computes gradients on its local data subset. The gradients are then aggregated to update the model parameters through two primary mechanisms: synchronous SGD (SSGD) and asynchronous SGD (ASSGD). In SSGD (Zinkevich et al., 2010b; Dekel et al., 2012; 2010; Ye et al., 2022; McMahan et al., 2017), all workers compute gradients in parallel, but the parameter server waits for all workers to finish before applying the aggregated gradients to the model. This ensures consistency but may suffer from stragglers. ASSGD (Baudet, 1978; Bertsekas and Tsitsiklis, 2015; Cohen et al., 2021; Recht et al., 2011b; Feyzmahdavian and Johansson, 2023; Stich et al., 2021; Nguyen et al., 2022; Even et al., 2024; Recht et al., 2011a; Zhang et al., 2015) address this limitation by enabling independent parameter updates without synchronization. A prominent example is HOGWILD! (Recht et al., 2011a), which implements lock-free updates to the shared model parameters in memory. However, ASSGD faces challenges with gradient staleness (Dutta et al., 2018).

Current methods primarily focus on parallelization within individual GD steps, which retains the inherent limitations of traditional autoregressive GD. In contrast, PASO disrupts this autoregressive dependency chain, introducing step parallelism where multiple different GD steps are executed simultaneously. We humbly believe that PASO's step-level parallelization naturally complements existing intra-step parallelization methods, establishing a new avenue for parallel GD. We observe that recent work (Shu et al., 2024) leaves kernelized gradient estimation to enable approximately parallelized iterations; our method is compatible with it, as the kernelized gradient estimation can be used to give more precise initial points for our PASO.

3 Preliminary

3.1 Stochastic Gradient Descent (SGD)

Given a mini-batch ζ of size B, the loss function for the batch is defined as the average loss over the samples in ζ:

$$\mathcal{L}(w,\zeta)=\frac{1}{B}\sum_{x,y\in\zeta}\ell(w;x,y), \tag{1}$$

where $\ell(w;x,y)$ denotes the loss for a single sample (x,y). The model parameters w are updated iteratively using the gradient of the batch loss:

$$w_t=w_{t-1}-\eta_{t-1}\nabla_{w_{t-1}}\mathcal{L}(w_{t-1},\zeta_{t-1}), \tag{2}$$

where η_t is the learning rate at iteration t, ζ_t stands for the mini-batch used at iteration t, and $\nabla_w\mathcal{L}(w_t,\zeta_t)$ is the gradient of the batch loss. Other popular optimizers, such as Adaptive Moment Estimation (Adam), also update parameters iteratively. For brevity, a detailed description of these methods is provided in Appendix E.

4 Proposed Method

4.1 Motivation

Gradient descent (GD) algorithms like SGD and Adam use historical weights to compute the current weight, which is the essence of an autoregressive process. We formally define this process below.

Definition 1 (Autoregressive GD Procedure)**.** *Initiating with a model weight w_0, the GD process like SGD and Adam represents an autoregressive procedure in the specific form of*

$$w_t=w_0-\sum_{\tau=0}^{t-1}\eta_\tau\, g_\tau\left(w_\tau,\dots,w_{\tau-r+1};\zeta_\tau,\dots,\zeta_{\tau-r+1}\right),t\in\{1,\cdots,T\}, \tag{3}$$

where $1\leq r\leq\tau+1$ and the general gradient term g_τ is determined by the specific optimizer. For example, the g_τ for SGD depends only on the most recent weight and mini-batch (i.e., $r=1$):

$$g_{t-1}(w_{t-1};\zeta_{t-1})=\nabla_{w_{t-1}}\mathcal{L}(w_{t-1},\zeta_{t-1}). \tag{4}$$

The explicit g_{t-1} formulations for more complex optimizers like Adam are detailed in Appendix E.

We observe that when all the model weights $w_0,\cdots,w_T$ are considered as unknown variables, the autoregressive GD procedure above transforms into a system of $T+1$ nonlinear equations (NEs). By providing an initial set of guesses for the true weights, this system of NEs can be solved in parallel since there are no dependencies among the $T+1$ NEs. As a result, the model weights $w_0,\cdots,w_T$ can be computed concurrently.

4.2 Recasting Autoregressive GD as Triangular Nonlinear Equation Solving

Inspired by current parallel algorithms (Song et al., 2021; Tang et al., 2024; Lu et al., 2025), such a series of cascaded functions in Definition 1 can be regarded as a system of $T+1$ NEs with a triangular structure. Denote by $\hat{w}_0,\cdots,\hat{w}_T$ the unknown variables corresponding to the iterative tracjectory $w_0,\cdots,w_T$ generated from the autoregressive GD process in Definition 1.

Definition 2 (Triangular NEs)**.** *We define the system of triangular NEs for the autoregressive procedure in Definition 1 as*

$$\mathcal{F}(\hat{w}_0,\cdots,\hat{w}_T)=\begin{cases}\hat{w}_0-w_0=0,\\ \hat{w}_t-F_{t-1}(\hat{w}_0,\cdots,\hat{w}_{t-1};\zeta_0,\dots,\zeta_{t-1})=0,t\in\{1,\cdots,T\},\end{cases} \tag{5}$$

where F_{t-1} is defined as:

$$F_{t-1}(\hat{w}_0, \cdots, \hat{w}_{t-1}; \zeta_0, \ldots, \zeta_{t-1}) = \hat{w}_0 - \sum_{\tau=0}^{t-1} \eta_\tau g_\tau(\hat{w}_\tau, \ldots, \hat{w}_{\tau-r+1}; \zeta_\tau, \ldots, \zeta_{\tau-r+1}), \quad (6)$$

where g_τ depends on the choice of the specific GD algorithms. In Appendix E, we include the explicit form of g_τ for various GD algorithms like AdamW.

This formulation offers several advantages. First, it decouples the dependencies among w_t, enabling synchronous calculation for all gradients $\nabla_{w_t}\mathcal{L}(w_t, \zeta_t), t \in \{0, \cdots, T-1\}$. This parallelism makes the approach especially suitable for modern parallel computing infrastructures, such as distributed systems. Second, the triangular NEs have been extensively studied in mathematics, providing access to a variety of well-established methods for solving such systems efficiently.

While we can now calculate the gradients across steps in parallel by solving the triangular NEs, an important question remains: do the solutions found via equation solving yield model weights comparable to those generated by the autoregressive GD process? Specifically, can we assert that $\hat{w}_t = w_t$ holds for all $t \in [0, T]$?

Proposition 1 (Unbiased Estimation (see *App.* F for proof))**.** *The TNEs system in Eq.* (5) *possesses a unique solution that unbiasedly estimates the GD trajectory* $\{w_\tau\}_{\tau=0}^{T}$ *in Definition 1.*

This finding demonstrates that by solving for the TNEs, a model of comparable quality to that derived from the traditional autoregressive GD process can be obtained.

4.3 Solving the System of TNEs

The field of optimization provides various methods for solving a system of NEs. Since our primary goal is to study a fundamental step-parallel GD optimizer, we implement only the classical fixed-point iteration (FPI) method (Banach, 1922) and postpone more advanced alternatives for future exploration. Applying FPI to find the solution of an equation system involves reformulating the equation system into an iterative form. It is easy to know the iterative form of Eq. (5) corresponds to a system with T iterative components in Eq. (7). Therefore, given an initial set of guesses $\hat{w}_0^{(0)}, \cdots, \hat{w}_T^{(0)}$, and randomly sampled T mini-batches $\zeta_0, \ldots, \zeta_{T-1}$, the system of FPI for the TNEs is as follows:

$$\hat{w}_t^{(k)} = F_{t-1}(\hat{w}_0^{(k-1)}, \cdots, \hat{w}_{t-1}^{(k-1)}; \zeta_0, \ldots, \zeta_{t-1}), t \in \{0, \cdots, T-1\}, \quad (7)$$

where $\hat{w}_0^{(k)} = w_0, \forall k \in \{0, \cdots, K\}$; K is the number of parallel iterations. For a more intuitive FPI system, see Definition 3 in Appendix.

Proposition 2 (Convergence Analysis (see *App.* G for proof))**.** *From any initial guess* $\{\hat{w}_t^{(0)}\}_{t=0}^{T}$*, the fixed-point iteration in Eq.* (7) *converges exactly to the autoregressive GD trajectory* $\{w_t\}_{t=0}^{T}$ *defined in Definition 1. This convergence is achieved in at most* T *steps.*

In practice, the number of parallel iterations K required for convergence is significantly smaller than T, resulting in substantial empirical speedups. Besides, we also provide empirical validation that the PASO trajectory is functionally equivalent to the original, with the near-zero average L2 norm and variance between them confirming a high-fidelity reproduction (*App. B.2.1*).

4.4 Computation-efficient Subequations Solving

Solving the above triangular NEs necessitates computing T gradients $\{\nabla_{\hat{w}_t}\mathcal{L}(\hat{w}_t, \zeta_t)\}_{t=0}^{T-1}$ in parallel across the entire time horizon. For large values of T, this becomes computationally prohibitive when restricted to a limited number of computing nodes. To tackle this, our core idea is to perform the fixed-point iteration only on $p \leq T$ subequations per iteration via a sliding window technique in (Shih et al., 2024). Specifically, we perform parallel equation solving only on a subset of $T+1$ NEs, within a sliding window of size p:

$$\hat{w}_{t+i}^{(k)} = F_{t-1+i}(\hat{w}_0^{(k-1)}, \cdots, \hat{w}_{t-1+i}^{(k-1)}; \zeta_0, \ldots, \zeta_{t-1+i}), i \in \{0, \cdots, \min\{p-1, T-1\}\}, \quad (8)$$

This window size can be tuned to match the number of available computing nodes. Additionally, the window slides forward dynamically, with the sliding distance determined by the number of equations for which solutions have been found in the current window.

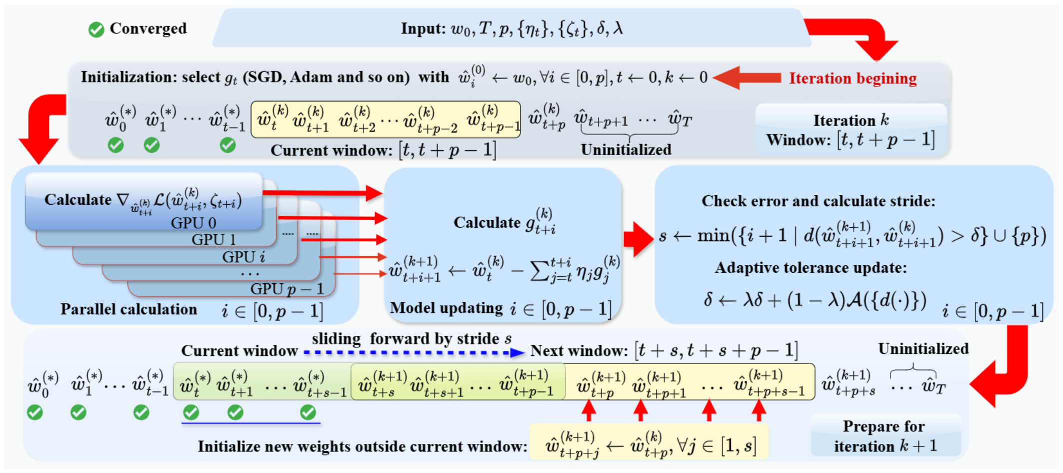

Converged
Input: $w_0, T, p, \{\eta_t\}, \{\zeta_t\}, \delta, \lambda$
Initialization: select g_t (SGD, Adam and so on) with $\hat{w}_i^{(0)} \leftarrow w_0, \forall i \in [0,p], t \leftarrow 0, k \leftarrow 0$
Iteration begining
$\hat{w}_0^{(*)}$ $\hat{w}_1^{(*)} \cdots \hat{w}_{t-1}^{(*)}$ $\hat{w}_t^{(k)}$ $\hat{w}_{t+1}^{(k)}$ $\hat{w}_{t+2}^{(k)} \cdots \hat{w}_{t+p-2}^{(k)}$ $\hat{w}_{t+p-1}^{(k)}$ $\hat{w}_{t+p}^{(k)}$ $\hat{w}_{t+p+1} \cdots \hat{w}_T$
Current window: $[t, t+p-1]$
Uninitialized
Iteration k
Window: $[t, t+p-1]$
Calculate $\nabla_{\hat{w}_{t+i}^{(k)}} \mathcal{L}(\hat{w}_{t+i}^{(k)}, \zeta_{t+i})$
GPU 0
GPU 1
GPU i
GPU $p-1$
Parallel calculation $i \in [0, p-1]$
Calculate $g_{t+i}^{(k)}$
$\hat{w}_{t+i+1}^{(k+1)} \leftarrow \hat{w}_t^{(k)} - \sum_{j=t}^{t+i} \eta_j g_j^{(k)}$
Model updating $i \in [0, p-1]$
Check error and calculate stride:
$s \leftarrow \min(\{i+1 \mid d(\hat{w}_{t+i+1}^{(k+1)}, \hat{w}_{t+i+1}^{(k)}) > \delta\} \cup \{p\})$
Adaptive tolerance update:
$\delta \leftarrow \lambda\delta + (1-\lambda)\mathcal{A}(\{d(\cdot)\})$ $i \in [0, p-1]$
Current window
sliding forward by stride s
Next window: $[t+s, t+s+p-1]$
Uninitialized
$\hat{w}_0^{(*)}$ $\hat{w}_1^{(*)} \cdots \hat{w}_{t-1}^{(*)}$ $\hat{w}_t^{(*)}$ $\hat{w}_{t+1}^{(*)} \cdots \hat{w}_{t+s-1}^{(*)}$ $\hat{w}_{t+s}^{(k+1)}$ $\hat{w}_{t+s+1}^{(k+1)} \cdots \hat{w}_{t+p-1}^{(k+1)}$ $\hat{w}_{t+p}^{(k+1)}$ $\hat{w}_{t+p+1}^{(k+1)} \cdots \hat{w}_{t+p+s-1}^{(k+1)}$ $\hat{w}_{t+p+s}^{(k+1)} \cdots \hat{w}_T$
Initialize new weights outside current window: $\hat{w}_{t+p+j}^{(k+1)} \leftarrow \hat{w}_{t+p}^{(k)}, \forall j \in [1, s]$
Prepare for iteration $k+1$

Figure 1: Illustration of Step-parallel Training Paradigm PASO. During iteration k, PASO performs simultaneous weight updates across steps within a p-size sliding window through parallel gradient computations. The process consists of: (1) computing update terms $g^{(k)}$ based on current weights $\hat{w}^{(k)}$ after calculating their graidients in parallel; followed by (2) determining new weights $\hat{w}^{(k+1)}$.

4.5 Stopping Criterion

To ensure that the parallel optimizer achieves performance on par with that of the autoregressive GD process, it's essential to establish a suitable stopping criterion to assess whether the solution values are no longer changing each parallel iteration. Let δ represent the convergence tolerance threshold, governing the allowable variation in solution values between successive iterations. In accordance with (Zhou et al., 2024), we define the stopping criterion as:

$$d(\hat{w}_t^{(k)}, \hat{w}_t^{(k-1)}) := \frac{1}{n} \left\| \hat{w}_t^{(k)} - \hat{w}_t^{(k-1)} \right\|^2 \leq \delta, \tag{9}$$

where $\|\cdot\|$ denotes the Frobenius norm; n is model dimension. δ is updated adaptively via exponential moving average (EMA):

$$\delta = \lambda\delta + (1-\lambda) \cdot \mathcal{A}\Big(\{d(\hat{w}_{t+i}^{(k)}, \hat{w}_{t+i}^{(k-1)}) | i = 1, \cdots, p\}\Big), \tag{10}$$

where λ is the EMA decay rate; $\mathcal{A}$ is a mean or median function.

4.6 Initialization

The parallel iteration in Eq. (8) begins with a set of initial weights $\{\hat{w}_t^{(0)}\}_{t=0}^{p}$. We initialize all the model parameters within the p-sized sliding window using the default initial weight w_0: $\hat{w}_t^{(0)} = w_0, \forall t \in \{0, \cdots, p\}$. When the sliding window moves forward, we initialize all the newly introduced model parameters using the rightmost weight from the previous window. We emphasize that initialization also stands as a crucial component for improving parallel efficiency. By starting iterations with weights that are close approximations of the target solutions $\{w_t\}_{t=0}^{p}$, more rapid convergence can be achieved. We defer this critical aspect to future research.

4.7 Complete PASO Algorithm

Algorithm 1 details the complete process of the proposed PASO over a sliding window. After obtaining the autoregressive GD update rule (Line 1) and preparing an array of initial weights $\{\hat{w}_t^{(0)}\}_{t=0}^{p-1}$ via the default model weight (Line 2), PASO initiates the parallel optimization loop at Line 3 in which a batch of weights $\{\hat{w}_t^{(k)}\}_{t=0}^{p-1}$ within a sliding window undergo synchronous updating. Line 5 compute the gradients, which are the basic computational units of parallelism. The updates are computed in Line 6, and Line 7 updates the current model weights, preparing for the next

Algorithm 1: PASO: Step Parallel Stochastic Optimization within A Sliding Window

```
Input  : Default model initial weight $w_0$, gradient descent steps $T$, learning rate $\{\eta_t\}_{t=0}^{T-1}$, random
         mini-batches $\{\zeta_t\}_{t=0}^{T-1}$, tolerance $\delta$, window size $p$, model dimension $n$, EMA decay rate $\lambda$.
Output : $\hat{w}_T^K$.
1  Obtain update rule $g_t(\hat{w}_t, \cdots, \hat{w}_{t-r+1}; \zeta_t, \cdots, \zeta_{t-r+1})$ by a GD algorithm.   // E.g., Eq. (4) or Eq. (15)
2  Initialize $\{\hat{w}_t^{(0)} = w_0, t = 0, \cdots, p\}$   // Initialize $p$ model weights within the sliding window.
3  $t, k \leftarrow 0, 0; k \in [0, K]$
4  while $t < T$ do
5      $\nabla_{\hat{w}_{t+i}^{(k)}} \mathcal{L}(\hat{w}_{t+i}^{(k)}, \zeta_{t+i}), \forall i \in \{0, \cdots, p-1\}$   // Compute each gradient concurrently.
6      $g_{t+i}^{(k)}, \forall i \in \{0, \cdots, p-1\}$   // Calculate updates in parallel (e.g., via Eq. (4)).
7      $w_{t+i+1}^{(k+1)} \leftarrow \hat{w}_t^{(k)} - \sum_{j=t}^{t+i} \eta_j g_j^{(k)}, \forall i \in \{0, \cdots, p-1\}$   // Update weights at iteration $k$ via Eq. (7).
8      $s \leftarrow \min\left(\{i+1; \hat{w}_{t+i+1}^{(k+1)}$ unsatisfying Eq. (9), $\forall i \in \{0, \cdots, p-1\}\} \cup \{p\}\right)$   // The sliding stride.
9      $\hat{w}_{t+p+j}^{(k+1)} \leftarrow \hat{w}_{t+p}^{(k)}, \forall j \in \{1, \cdots, s\}$   // Initialize new model weights.
10     $\delta \leftarrow$ Eq. (10)   // Update tolerance via exponential moving average.
11     $t \leftarrow t + s, \quad k \leftarrow k + 1, \quad p \leftarrow \min(p, T - t)$
Return: $\hat{w}_T^{(K)}$
```

parallel iteration $k+1$. Line 8 checks the variation between new weights and the current weights and then determines the stride to which the window can slide forward. Line 9 initializes new model parameters outside the current window using the rightmost weight in the current window according to the sliding stride s. Fig. 1 shows the pipeline of PASO.

5 Computational Cost, Memory, and Speedup Ratio Analysis

PASO introduces a novel *step-parallel* approach that is orthogonal to traditional parallelization paradigms. This naturally raises a key question: *under comparable computational and memory constraints, how does the speedup efficiency of PASO compare to that of conventional methods?* To this end, Table 1 provides a comprehensive comparison against existing methods, indicating three main conclusions:

- **Acceptable Overhead:** The total computational cost of PASO (mT) is comparable to that of model and pipeline parallelism (T) and significantly lower than data parallelism (NT). This minimal overhead is a worthwhile trade-off for the performance gains.
- **Superior Speedup:** The speedup ratio of PASO is $\frac{N}{m(1+\alpha N/p)}$. Since $m \approx 1$ and $p > 1$, PASO's speedup is strictly greater than the $\frac{N}{1+\alpha N}$ achieved by other methods. This indicates that PASO can be approximately up to **p** times faster than existing parallel approaches.
- **Better Scalability:** When the window size equals the number of GPUs ($p = N$), PASO's speedup ratio simplifies to $\frac{N}{m(1+\alpha)}$. As N increases, the denominator in PASO's speedup formula grows much more slowly than in other methods (where it is dominated by the αN term), demonstrating PASO's superior scalability, especially in communication-bound scenarios.

Table 1: Comparison of computational cost, storage, and speedup ratio across parallel training methods. The analysis shows PASO's superior speedup potential and scalability. Denote by N the number of GPUs, $\alpha \triangleq t_{\text{comm}}/t_{\text{comp}}$ the communication-to-computation time ratio, and $m \triangleq T/pK \approx 1$ empirically (see Fig. 7 in *Appendix*). Detailed derivations are available in *Appendix* C.2.

Method	**Computational Gradient Count**	**Storage per Device**	**Speedup Ratio** (S)	**Scalability Limit** ($\lim_{N\to\infty} S$)
Sequential	T	1 model + 1 optimizer	1	1
Data Parallel	NT	1 model + 1 optimizer	$\frac{N}{1+\alpha N}$	$1/\alpha$
Model Parallel	T	$\sim \frac{1}{N}$ model + 1 optimizer	$\frac{N}{1+\alpha N}$	$1/\alpha$
Pipeline Parallel	T	$\sim \frac{1}{N}$ model + 1 optimizer	$\frac{N}{1+\alpha N}$	$1/\alpha$
Step Parallel (PASO)	$\mathbf{pK = mT}$	**1 model + 1 optimizer**	$\mathbf{\frac{N}{m(1+\alpha N/p)}}$	$\mathbf{p/(m\alpha)}$

In summary, under similar costs, PASO achieves a higher theoretical speedup and exhibits better scalability than existing parallel methods. We believe our step-parallel approach holds significant potential for fully leveraging modern parallel hardware for deep learning training.

6 Experiment

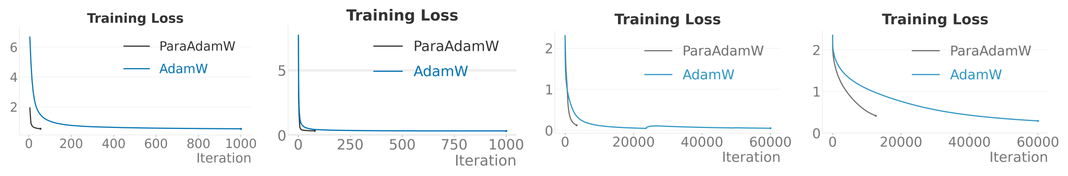
6.1 Experiment Settings

Dataset and Model. We investigate our PASO on both an image classification task and a language modeling task. For the image task, we use the CIFAR-10 dataset (Krizhevsky et al., 2009) over a compact convolutional neural network (CNN) and Tiny-ImageNet dataset (Le and Yang, 2015) on a more harder ViT model. For the language modeling task, we train a GPT-2 model and 1B large model Llama-3.2-1B on the WikiText dataset. Further experimental details are provided in the appendix.

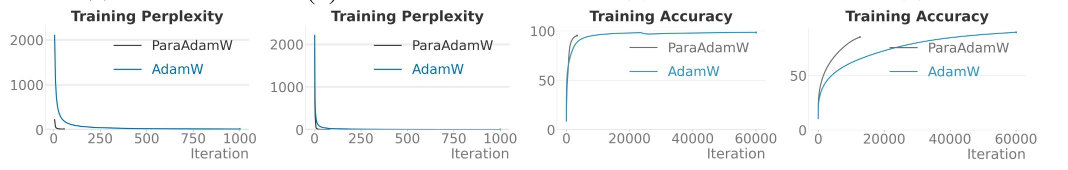
Evaluation Metrics. For the CIFAR-10 dataset, we evaluate the performance using six standard metrics: testing accuracy, testing precision, testing recall, testing F1-score, iterations, and wall-clock time. For the WikiText dataset, we evaluate the performance using four standard metrics: testing accuracy, testing perplexity, iterations, and wall-clock time.

Hyperparameter Settings. For the GPT2, Llama-3.2-1B, and ViT model, we use 8 NVIDIA A100 GPUs. For CNN, we use 8 NVIDIA 3090 GPUs. We employ the sweep function in Wandb (WandB, 2023) to investigate the influence of hyperparameters on the model's performance metrics, configuring a hyperparameter sweep with the following search ranges: tolerance threshold δ sampled uniformly in $[10^{-6}, 10^{-4}]$, EMA decay rate λ sampled uniformly in $[0.8, 0.9999]$, and adaptivity scheme $\mathcal{A}$ chosen between *mean* and *median* operators. We found setting $\delta = 10^{-5}$ and $\lambda \in [0.9, 0.9999]$ easily yields comparable performance across models and datasets. Please refer to *Appendix* for the sweep results in Fig. 6.

6.2 Experiment Results



(a) GPT2 (b) Llama-3.2-1B (c) CNN (d) ViT



(a) GPT2 (b) Llama-3.2-1B (c) CNN (d) ViT

Figure 2: The comparison of loss, perplexity, and accuracy curves.

Language Modeling Task. Table 2 demonstrate that PASO accelerates the convergence of these optimizers without sacrificing model performance. Notably, PASO reduces the required iteration steps for sequential methods by a factor of 12.6 $\sim$ 19.2, resulting in a up to 7.5$\times$ improvement in wall-clock time. This speedup implies that an LLM originally requiring 100 days of training can now be trained in just 13 days. Note that larger batch sizes could yield higher runtime speedups for Llama-3.2-1B, but our present implementation supports a maximum batch size of only 30.

Image Classification Task. Table 3 compares SGD, Adam, and AdamW with their PASO-enhanced versions, showing that PASO consistently accelerates convergence while preserving model performance. For instance, in CNN model, Adam+PASO achieves a 31.2$\times$ step reduction (to 1919) with a 2.7$\times$ runtime speedup. The accuracy, precision, recall, and F1-score, confirm PASO's efficiency without compromising model quality. Note that in CV tasks, the smaller runtime speedup than the LLM tasks is due to the higher communication-to-computation ratio, as CV models are smaller and compute faster per step, making communication overhead more significant.

Table 2: Quantitative comparisons of different methods on WiKiText. The best results are highlighted in **bold**. "↑" (resp. "↓") means the larger (resp. smaller), the better.

Method	WiKiText, GPT-2 Model, $B = 130$, $\eta = 6e - 5$, $T = 1000$				
	Iters ↓	Accuracy ↑	Perplexity ↓	Time (s) ↓	Speedup ↑
SGD	1000	86.8	1.8	713	1.0×
ParaSGD (SGD + PASO)	**52** (19.2×)	86.8	1.8	**95**	**7.5**×
Adam	1000	86.9	1.6	716	1.0×
ParaAdam (Adam + PASO)	**57** (17.5×)	86.9	1.6	**106**	**6.8**×
AdamW	1000	86.9	1.6	715	1.0×
ParaAdamW (AdamW + PASO)	**57** (17.5×)	86.9	1.6	**107**	**6.7**×
Method	WiKiText, Llama-3.2-1B Model, $B = 30$, $\eta = 6e - 5$, $T = 1000$				
	Iters ↓	Accuracy ↑	Perplexity ↓	Time (s) ↓	Speedup ↑
SGD	1000	86.3	1.6	827	1.0×
ParaSGD (SGD + PASO)	**69** (14.5×)	86.4	1.6	**266**	**3.1**×
Adam	1000	86.3	1.4	838	1.0×
ParaAdam (Adam + PASO)	**79** (12.6×)	86.3	1.4	**279**	**3.0**×
AdamW	1000	86.3	1.4	854	1.0×
ParaAdamW (AdamW + PASO)	**78** (12.8×)	86.3	1.4	**281**	**3.0**×

Impact of the Window Size p. Tab. 4 illustrates the influence of the window size p on the speedup of AdamW with 1000 steps over the GPT-2 model with batch size 130. As p increases, the number of iterations needed for convergence significantly drops, from 184 to 11, yielding a step reduction ranging from 4.0× to **91**×. This suggests that we can achieve up to 91× walk-clock time acceleration without loss of model quality using 201 GPUs. However, due to our current constrained GPU resources, larger p values introduce higher computational overhead per GPU, leading to increased wall-clock time. We believe that with more computing cores, the time speedup of PASO could be substantially unleashed.

Table 4: Impact of p on accuracy (ACC), perplexity (PPL), and speedup.

	Iters	PPL	ACC	Time (s)	Speedup
AdamW	1000	1.6	86.9	1063	1×
$p = 7$	184	1.6	86.9	182	5.8×
$p = 35$	39	1.6	86.9	173	**6.1**×
$p = 77$	21	1.6	86.8	209	5.1×
$p = 117$	16	1.6	86.8	235	4.5×
$p = 159$	12	1.6	86.8	240	4.4×
$p = 201$	11	1.6	86.7	267	4.0×

Impact of Batch Size. As shown in Tab. 5, the speedup of PASO becomes more significant as the batch size increases, while maintaining performance comparable to the baseline. This is because a larger batch size allows for better utilization of the GPU's computing capabilities for large-scale matrix operations, thereby improving parallel efficiency. More ablation studies on EMA decay rate and tolerance are shown in *Appendix* B.2.4.

Table 5: Impact of batch size (B) on perplexity (PPL) and Accuracy (ACC). Top to bottom: $B = 10, 50, 90, 130$.

Method	Iters	ACC	PPL	Time (s)	Speedup
AdamW	1000	86.9	1.6	146	1×
PASO	64	86.9	1.6	61	2.4×
AdamW	1000	87.0	1.6	301	1×
PASO	57	87.0	1.6	90	3.3×
AdamW	1000	86.9	1.6	822	1×
PASO	56	86.9	1.6	121	6.8×
AdamW	1000	86.9	1.6	1063	1×
PASO	56	86.9	1.6	155	**6.9**×

Impact of Learning Rate. Figure 3 illustrates that PASO performs robustly across a practical range of learning rates from 4×10^{-4} to 1×10^{-2}. These rates are comparable to those in standard optimizers (e.g., Adam's default of 1×10^{-3}), demonstrating that the model can converge effectively without additional limitations on learning rates.

7 LIMITATIONS AND FUTURE DIRECTIONS

Table 3: Quantitative comparisons of different methods on CIFAR-10.

Method	CIFAR-10, CNN Model, $B = 4096$, $\eta = 1e-3$						
	Iters↓	Accuracy ↑	Precision↑	Recall↑	F1-score↑	Time (s) ↓	Speedup↑
SGD	60000	66.0	66.0	66.0	66.0	4277	1.0×
ParaSGD (SGD + PASO)	**1723** (34.8×)	66.1	66.2	66.1	66.1	**1339**	**3.2**×
Adam	60000	59.7	59.7	60.0	59.7	4223	1.0×
ParaAdam (Adam + PASO)	**1919** (31.2×)	60.0	60.0	60.0	60.0	**1574**	**2.7**×
AdamW	60000	60.4	60.4	60.4	60.3	4267	1.0×
ParaAdamW (AdamW + PASO)	**1924** (31.2×)	60.2	60.2	60.2	60.2	**1573**	**2.7**×
Method	CIFAR-10, ViT Model, $B = 2048$, $\eta = 1e-5$						
	Iters↓	Accuracy ↑	Precision↑	Recall↑	F1-score↑	Time (s) ↓	Speedup↑
SGD	60000	37.2	39.1	37.2	37.0	36305	1.0×
ParaSGD (SGD + PASO)	**3975** (15.1×)	37.6	39.1	37.6	37.3	**10481**	**3.5**×
Adam	60000	71.6	71.7	71.6	71.5	36282	1.0×
ParaAdam (Adam + PASO)	**4219** (14.2×)	71.6	71.7	71.6	71.5	**11185**	**3.2**×
AdamW	60000	72.0	72.0	72.0	72.0	36310	1.0×
ParaAdamW (AdamW + PASO)	**4231** (14.2×)	71.9	72.0	71.9	72.0	**11208**	**3.2**×

While PASO achieves significant runtime acceleration (2×–7.5×), this remains substantially below its step-level speedup (up to 91×). Two primary factors limit performance: (1) our constrained GPU resources that inherently restrict maximum acceleration, and (2) our current implementation exists inefficient inter-GPU communication requiring model transfers to pass through a CPU intermediary. These limitations are further exacerbated by unoptimized handling of gradient synchronization, load imbalance, and kernel launch overheads, and so on.

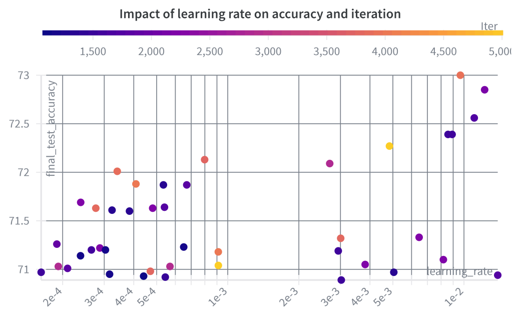


Figure 3: Impact of learning rate on accuracy and iterations. The total steps are 10000. Darker points indicate faster convergence.

Looking ahead, PASO's efficiency can be substantially improved through: (1) algorithmic refinements like gradient compression to reduce overhead; (2) system-level enhancements such as collective operations (e.g., NCCL all-reduce) to alleviate bottlenecks. These advancements could position PASO as a promising paradigm for more efficient parallel training, with broader implications for large-scale deep learning.

8 Conclusion

This paper introduces PASO, a novel framework that accelerates stochastic optimization by reformulating its autoregressive process as a system of triangular nonlinear equations (TNEs), enabling step parallel gradient computation. Theoretically, we prove that the TNE system has a unique solution matching the stochastic optimization's iteration trajectory, and solving it converges as efficiently as or faster than sequential method. Empirically, PASO achieves up to 91× speedup in steps without quality degradation.

Ethics & Reproducibility Statements

Ethics Statement. This work presents a fundamental methodology for accelerating stochastic optimization algorithms. To the best of our knowledge, it does not raise any immediate ethical concerns. The research is theoretical and empirical in nature, based on mathematical analysis and standard benchmark tasks. We do not employ any private or sensitive data. However, we acknowledge that any optimization technology has the potential for dual use. We encourage the community to utilize this work responsibly.

Reproducibility Statement. We are committed to fostering reproducible research. To this end:

- The theoretical claims in this paper, including the uniqueness of the solution to the triangular nonlinear equations and the convergence guarantees, are supported by formal proofs provided in appendix.
- The empirical results are obtained using standard datasets and benchmarks. To ensure reproducibility, we will open-source the complete implementation of the PASO framework, including scripts for all experiments.
- The code package will include detailed documentation, instructions for setting up the computational environment, and scripts to replicate the reported speedup and performance comparisons against sequential baselines.
- All hyperparameters and experimental settings are explicitly documented in the paper's experimental section.

We believe these measures will enable other researchers to verify our findings and build upon this work.

References

Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta mathematicae*, 3(1):133–181, 1922.

Gerard M Baudet. Asynchronous iterative methods for multiprocessors. *Journal of the ACM (JACM)*, 25(2):226–244, 1978.

Dimitri Bertsekas and John Tsitsiklis. *Parallel and distributed computation: numerical methods*. Athena Scientific, 2015.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020.

Ziheng Cheng and Margalit Glasgow. Convergence of distributed adaptive optimization with local updates. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=VNg7srnvD9.

Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX symposium on operating systems design and implementation (OSDI 14)*, pages 571–582, 2014.

Alon Cohen, Amit Daniely, Yoel Drori, Tomer Koren, and Mariano Schain. Asynchronous stochastic optimization robust to arbitrary delays. *Advances in Neural Information Processing Systems*, 34: 9024–9035, 2021.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Robust distributed online prediction. *arXiv preprint arXiv:1012.1370*, 2010.

Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *The Journal of Machine Learning Research*, 13(1):165–202, 2012.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd. In *International conference on artificial intelligence and statistics*, pages 803–812. PMLR, 2018.

Mathieu Even, Anastasia Koloskova, and Laurent Massoulié. Asynchronous sgd on graphs: a unified framework for asynchronous decentralized and federated optimization. In *International Conference on Artificial Intelligence and Statistics*, pages 64–72. PMLR, 2024.

Hamid Reza Feyzmahdavian and Mikael Johansson. Asynchronous iterations in optimization: New sequence results and sharper algorithmic guarantees. *Journal of Machine Learning Research*, 24 (158):1–75, 2023.

Yunqi Gao, Zechao Zhang, Bing Hu, Mahdi Boloursaz Mashhadi, A-Long Jin, and Pei Xiao. Net-placer+: Model parallelism based on load balance in distributed deep learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2025.

Chaoyang He, Shen Li, Mahdi Soltanolkotabi, and Salman Avestimehr. Pipetransformer: Automated elastic pipelining for distributed training of transformers. *arXiv preprint arXiv:2102.03161*, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

Dongseong Hwang. Fadam: Adam is a natural gradient optimizer using diagonal empirical fisher information. *arXiv preprint arXiv:2405.12807*, 2024.

Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *Proceedings of Machine Learning and Systems*, 1:1–13, 2019.

Chiheon Kim, Heungsub Lee, Myungryong Jeong, Woonhyuk Baek, Boogeon Yoon, Ildoo Kim, Sungbin Lim, and Sungwoong Kim. torchgpipe: On-the-fly pipeline parallelism for training giant models. *arXiv preprint arXiv:2004.09910*, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Yann Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

Zhuohan Li, Siyuan Zhuang, Shiyuan Guo, Danyang Zhuo, Hao Zhang, Dawn Song, and Ion Stoica. Terapipe: Token-level pipeline parallelism for training large-scale language models. In *International Conference on Machine Learning*, pages 6543–6552. PMLR, 2021.

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.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Jianrong Lu, Lulu Xue, Wei Wan, Minghui Li, Leo Yu Zhang, and Shengqing Hu. Preserving privacy of input features across all stages of collaborative learning. In *2023 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 191–198, 2023. doi: 10.1109/ISPA-BDCloud-SocialCom-SustainCom59178.2023.00058.

Jianrong Lu, Shengshan Hu, Wei Wan, Minghui Li, Leo Yu Zhang, Lulu Xue, and Hai Jin. Depriving the survival space of adversaries against poisoned gradients in federated learning. *IEEE Transactions on Information Forensics and Security*, 19:5405–5418, 2024. doi: 10.1109/TIFS.2024.3360869.

Jianrong Lu, Zhiyu Zhu, and Junhui Hou. Parasolver: A hierarchical parallel integral solver for diffusion models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=2JihLwirxO.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

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*, pages 1–15, 2019.

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*, pages 1–15, 2021.

Yurii Nesterov. A method for solving the convex programming problem with convergence rate o (1/k2). In *Dokl akad nauk Sssr*, volume 269, page 543, 1983.

John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. In *International conference on artificial intelligence and statistics*, pages 3581–3607. PMLR, 2022.

NVIDIA. Nvidia collective communications library (nccl). https://developer.nvidia.com/nccl, 2023. Accessed: 2023-10-01.

OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023a.

OpenAI. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023b. URL https://arxiv.org/abs/2303.08774. Accessed: 2023-03-15.

Matteo Pagliardini, Pierre Ablin, and David Grangier. The ademamix optimizer: Better, faster, older. *arXiv preprint arXiv:2409.03137*, 2024.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

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, 2011a.

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, 2011b.

Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. {Zero-offload}: Democratizing {billion-scale} model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 551–564, 2021.

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

Thomas Robert, Mher Safaryan, Ionut-Vlad Modoranu, and Dan Alistarh. LDAdam: Adaptive optimization from low-dimensional gradient statistics. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Zkp1GuHerF.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems*, 31, 2018.

Junyu Shi, Wei Wan, Shengshan Hu, Jianrong Lu, and Leo Yu Zhang. Challenges and approaches for mitigating byzantine attacks in federated learning. In *2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 139–146, 2022. doi: 10.1109/TrustCom56396.2022.00030.

Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Yao Shu, Jiongfeng Fang, Ying Tiffany He, and Fei Richard Yu. Optex: Expediting first-order optimization with approximately parallelized iterations. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=MzNjnbgcPN.

Yang Song, Chenlin Meng, Renjie Liao, and Stefano Ermon. Accelerating feedforward computation via parallel nonlinear equation solving. In *International Conference on Machine Learning*, pages 9791–9800. PMLR, 2021.

Sebastian Stich, Amirkeivan Mohtashami, and Martin Jaggi. Critical parameters for scalable distributed learning with large batches and asynchronous updates. In *International Conference on Artificial Intelligence and Statistics*, pages 4042–4050. PMLR, 2021.

Zhenbo Sun, Shengqi Chen, Yuanwei Wang, Jian Sha, Guanyu Feng, and Wenguang Chen. Mepipe: Democratizing llm training with memory-efficient slice-level pipeline scheduling on cost-effective accelerators. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1263–1278, 2025.

Yu Tang, Lujia Yin, Qiao Li, Hongyu Zhu, Hengjie Li, Xingcheng Zhang, Linbo Qiao, Dongsheng Li, and Jiaxin Li. Koala: Efficient pipeline training through automated schedule searching on domain-specific language. *ACM Transactions on Architecture and Code Optimization*.

Zhiwei Tang, Jiasheng Tang, Hao Luo, Fan Wang, and Tsung-Hui Chang. Accelerating parallel sampling of diffusion models. In *Forty-first International Conference on Machine Learning*, 2024.

Tijmen Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26, 2012.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Lukasz Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

WandB. Weights & biases. https://wandb.ai, 2023. Version 0.16.0.

Eric P Xing, Qirong Ho, Wei Dai, Jin-Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: A new platform for distributed machine learning on big data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1335–1344, 2015.

Yuanzhong Xu, HyoukJoong Lee, Dehao Chen, Hongjun Choi, Blake Hechtman, and Shibo Wang. Automatic cross-replica sharding of weight update in data-parallel training. *arXiv preprint arXiv:2004.13336*, 2020.

Yuanzhong Xu, HyoukJoong Lee, Dehao Chen, Blake Hechtman, Yanping Huang, Rahul Joshi, Maxim Krikun, Dmitry Lepikhin, Andy Ly, Marcello Maggioni, et al. Gspmd: general and scalable parallelization for ml computation graphs. *arXiv preprint arXiv:2105.04663*, 2021.

Qing Ye, Yuhao Zhou, Mingjia Shi, and Jiancheng Lv. Flsgd: free local sgd with parallel synchronization. *The Journal of Supercomputing*, 78(10):12410–12433, 2022.

Jinhui Yuan, Xinqi Li, Cheng Cheng, Juncheng Liu, Ran Guo, Shenghang Cai, Chi Yao, Fei Yang, Xiaodong Yi, Chuan Wu, et al. Oneflow: Redesign the distributed deep learning framework from scratch. *arXiv preprint arXiv:2110.15032*, 2021.

Sixin Zhang, Anna E Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd. *Advances in neural information processing systems*, 28, 2015.

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Diederik P Kingma, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=iBExhaU3Lc.

Hairui Zhao, Hongliang Li, Qi Tian, Jie Wu, Meng Zhang, Xiang Li, and Haixiao Xu. Arraypipe: Introducing job-array pipeline parallelism for high throughput model exploration.

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

Shuxin Zheng, Qi Meng, Taifeng Wang, Wei Chen, Nenghai Yu, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic gradient descent with delay compensation. In *International conference on machine learning*, pages 4120–4129. PMLR, 2017.

Linqi Zhou, Andy Shih, Chenlin Meng, and Stefano Ermon. Dreampropeller: Supercharge text-to-3d generation with parallel sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4610–4619, 2024.

Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. Parallelized stochastic gradient descent. *Advances in neural information processing systems*, 23, 2010a.

Martin Zinkevich, Markus Weimer, Lihong Li, and Alex Smola. Parallelized stochastic gradient descent. *Advances in neural information processing systems*, 23, 2010b.

CONTENTS

A USE OF LLM

During the preparation of this work, we used Large Language Models (LLMs) to assist with the writing process. The primary uses included polishing and improving the fluency of the text, generating preliminary drafts of proofs, and assisting in the creation and formatting of tables. After using these tools, the author(s) reviewed and edited the content extensively. We take full responsibility for the entire content of this publication, including the ideas, proofs, and presentations ultimately contained in the final manuscript.

B MORE EXPERIMENTAL DETAILS

We evaluate PASO over two popular model training tasks, including image classification and text generation model. The results of these experiments demonstrate that PASO enhances the efficiency of autoregressive GD methods by approximately 1.5 times, all while maintaining consistent model quality as measured by metrics like accuracy or perplexity.

B.1 EXPERIMENT SETTINGS

Dataset and Model. We investigate our PASO on both an image classification task and a language modeling task. For the image task, we use the CIFAR-10 dataset (Krizhevsky et al., 2009), a widely recognized benchmark in computer vision. The dataset comprises 60,000 32 $\times$ 32 RGB images spanning 10 distinct classes, divided into 50,000 training and 10,000 test samples. The small image size and real-world noise make CIFAR-10 a challenging yet efficient testbed for lightweight models. We evaluate our approach by training a compact Convolutional Neural Network (CNN), following a standard architecture with convolutional and pooling layers, as shown in Table 6. Besides, we train a Vision Transformer (ViT) model [1] on CIFAR-10. For the language task, we train GPT-2 model[2] and Llama-3.2-1B[3] on the WikiText dataset[4], a large-scale corpus of Wikipedia articles preprocessed for language modeling. The dataset is publicly available and commonly used for training and benchmarking autoregressive models like GPT-2. We adopt the standard GPT-2 architecture, leveraging Hugging Face's `transformers` library for tokenization and training loops. We assess the performance of all methods on 8 NVIDI A100 GPUs.

Table 6: CNN Architecture

Layer Type	**Parameter Configuration**
Conv2D	Input channels 3, output channels 32, kernel size 3×3, padding 1
ReLU	Activation function
MaxPool2D	Pooling kernel 2×2, stride 2
Conv2D	Input channels 32, output channels 64, kernel size 3×3, padding 1
ReLU	Activation function
MaxPool2D	Pooling kernel 2×2, stride 2
Flatten	Flatten to 64×8×8 vector
Linear	Input dimension 4096 (64×8×8), output dimension 128
ReLU	Activation function
Linear	Input dimension 128, output dimension 10

Evaluation Metrics. For the CIFAR-10 dataset, we evaluate the model performance using six standard metrics: accuracy, precision, recall, F1-score, iterations, and wall-clock time. For the WikiText dataset, we evaluate the performance using four standard metrics: accuracy, perplexity, iterations, and wall-clock time. These metrics collectively assess both the effectiveness and efficiency of our PASO. For evaluation of the wall-clock time, we use the *torch.cuda.Event*[5] method provided by Pytorch (Paszke et al., 2019).

Algorithms. We accelerate three widely used optimizers: SGD, Adam, and AdamW. We refer their parallel variants as ParaSGD, ParaAdam, and ParaAdamW, respectively.

B.2 Experiment Results

B.2.1 Empirical Validation of Optimization Trajectory Equivalence

The objective of this study is to verify that the PASO algorithm faithfully reproduces the optimization path of various standard sequential optimizers. The experimental design is as follows:

- **Model and Task:** We train a CNN on the CIFAR-10 dataset. The total iterations is 10000 GD steps.
- **Optimizer Comparison:** For each base optimizer (SGD, Adam, AdamW), we conduct two parallel training procedures: one using the standard sequential optimizer and another using its PASO-enhanced version. Critically, all training runs commence from an identical set of randomly initialized weights, learning rate, and batch size to ensure a fair comparison.
- **Evaluation Metric:** At each training step t, we compute the squared L2 norm of the difference between the model weight vectors produced by the two methods, defined as:

$$d^t = ||w^t_{\text{PASO}} - w^t_{\text{Sequential}}||^2$$

[1]https://www.modelscope.cn/models/iic/multi-modal_clip-vit-large-patch14_336_zh
[2]https://github.com/openai/gpt-2
[3]https://huggingface.co/meta-llama/Llama-3.2-1B
[4]https://www.salesforce.com/blog/the-wikitext-long-term-dependency-language-modeling-dataset
[5]https://pytorch.org/docs/stable/generated/torch.cuda.Event.html

where w^t_{PASO} and $w^t_{\text{Sequential}}$ represent the model weights obtained by the PASO variant and its standard sequential counterpart at step t, respectively. This metric, d^t, quantifies the instantaneous deviation between the two optimization trajectories in the parameter space. To provide a comprehensive assessment, we report the mean and variance of d^t across the entire training process for each optimizer.

Results and Analysis. The statistical summary of the trajectory divergence d^t for all optimizers over the complete training process is presented in Table 7.

Table 7: Statistical summary of trajectory divergence (d^t) between PASO and sequential optimizers.

Optimizer	**Mean of d^t**	**Variance of d^t**
SGD and SGD + PASO	3.14×10^{-6}	6.71×10^{-12}
Adam and Adam + PASO	3.56×10^{-3}	5.75×10^{-6}
AdamW and AdamW + PASO	3.38×10^{-3}	4.70×10^{-6}

The results demonstrate that for all three optimizers, the mean and variance of the divergence d^t remain exceptionally small throughout the training process. The consistently minimal values across all optimizers empirically confirm that PASO faithfully reproduces the optimization trajectory of the standard sequential optimizer, regardless of the specific optimization algorithm employed. This high-fidelity replication ensures that the convergence properties and final solution quality of the original optimizer are preserved. Note that while the average L2 norm for Adam and AdamW appear larger than SGD's, they remain highly insignificant when considered in context. For a model with millions of parameters, an average squared L2 norm difference on the order of 10^{-3} corresponds to an extremely small per-parameter discrepancy. For example, for a model with $n \approx 5 \times 10^6$ parameters, this corresponds to a *root mean squared error (RMSE)* per parameter of approximately $\sqrt{3.56 \times 10^{-3}/5 \times 10^6} \approx 2.7 \times 10^{-5}$. Consequently, the trajectories of PASO and sequential optimizers are functionally equivalent.

B.2.2 LANGUAGE MODELING TASK

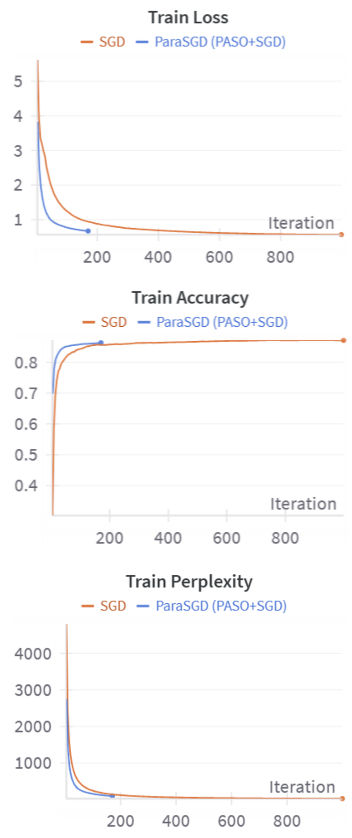
Complete Training Process. Figure 4 presents the loss, accuracy, and perplexity curves of ParaSGD, ParaAdam, and ParaAdamW, respectively. We can see that ParaSGD, ParaAdam, and ParaAdamW achieve faster convergence across iterations. For example, for our ParaSGD method (Figure 2a, from top to bottom), both the training loss and perplexity decrease rapidly after approximately 200 iterations, while the loss and perplexity of SGD drop slowly and stabilizes around 1000. The training accuracy of ParaSGD increases quickly, reaching close to 80% by the $200th$ iteration. This suggests that our PASO is effective in improving the training efficiency.

B.2.3 IMAGE CLASSIFICATION TASK

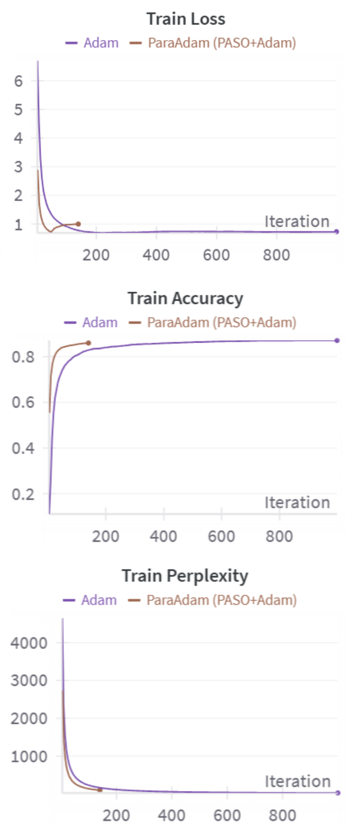
Complete Training Process. Figure 5 shows the training loss and accuracy trajectories for ParaSGD, ParaAdam, and ParaAdamW. All three methods demonstrate accelerated convergence compared to their baseline counterparts. Particularly notable is ParaAdamW (Figure 3c, top to bottom), where the training loss exhibits a sharp decline after approximately 3.5k iterations - in contrast to AdamW's gradual reduction that only stabilizes around 30k iterations. Furthermore, ParaAdamW achieves a training accuracy of nearly 82% by the 3.5k-th iteration, significantly outperforming AdamW's 69% accuracy at 30k iterations. These results demonstrate that our PASO framework effectively enhances both training efficiency and model performance metrics.

B.2.4 THE IMPACT OF HYPERPARAMETERS

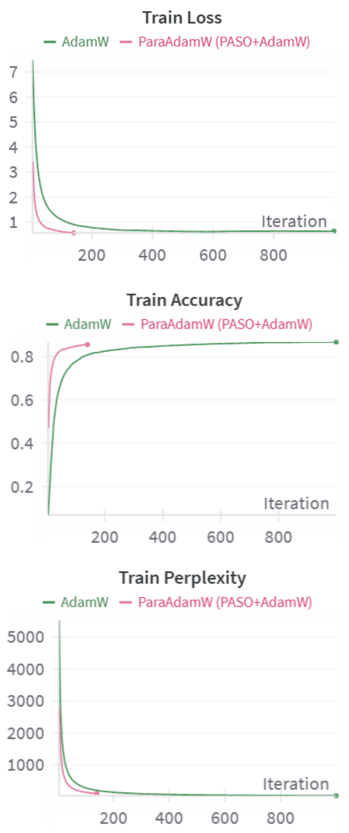
Impact of Tolerance δ and EMA Decay Rate λ. The Fig. 6 illustrates the impact of different tolerance (δ) and EMA decay rate (λ) on model performance. The results show that different combinations of δ and λ achieve a speedup of 4.61$\times$ (13000 v.s. 60000) to 4.81$\times$ (12450 v.s. 60000) while maintaining the similar model quality as Adam. In addition, the interplay between δ and λ highlights a trade-off: aggressive smoothing ($\lambda \uparrow$) with loose tolerance ($\delta \uparrow$) may reduce computational effort, while finer tolerance ($\delta \downarrow$) with moderate λ could enhance model quality at the expense of convergence speed.



(a) ParaSGD

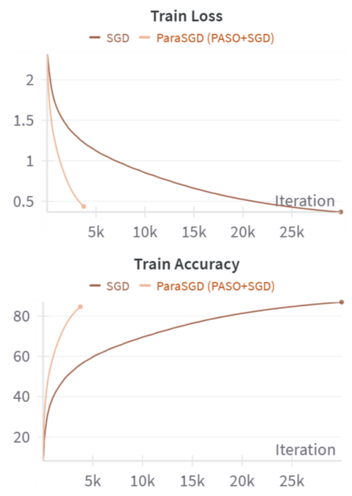


(b) ParaAdam

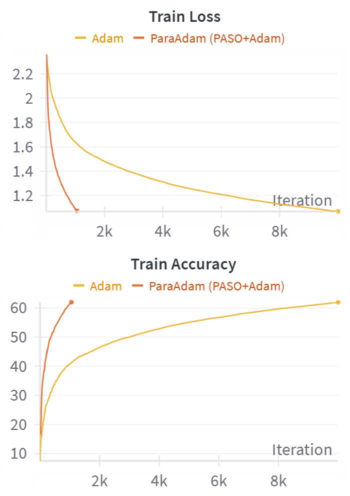


(c) ParaAdamW

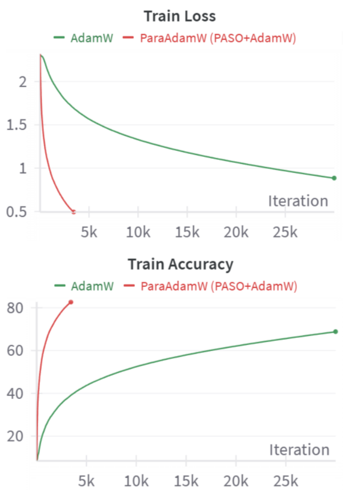
Figure 4: The loss, accuracy, and perplexity curve of WikiText Task



(a) ParaSGD



(b) ParaAdam



(c) ParaAdamW

Figure 5: The loss and accuracy curve of CIFAR10 Task

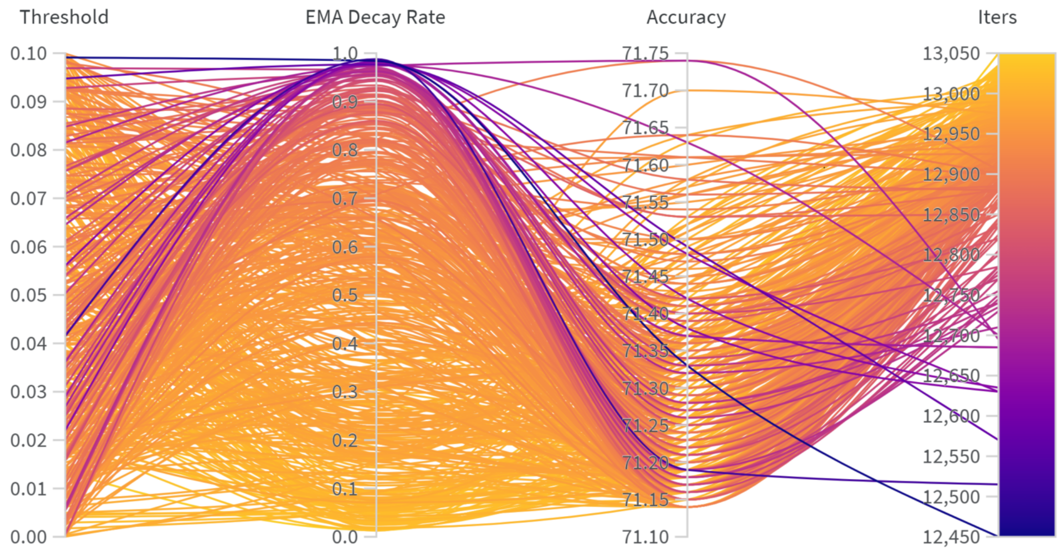


Figure 6: The impact of δ and λ over CIFAR-10 by running 1200 experiments. We use PASO with $p = 7$ to accelerate Adam with 60000 steps. Darker lines indicate runs with fewer iterations.

In summary, these new parameters do not require extensive tuning and are quite intuitive in selection:

- **Tolerance** (δ): This parameter controls the convergence precision of the fixed-point iteration. Manually setting this could be tedious. For this reason, we employ an adaptive tolerance schedule. The tolerance starts loose and automatically tightens as training progresses. This makes the method robust and largely removes δ from the list of parameters requiring manual tuning.
- **EMA Decay Rate** (λ): This is used within our adaptive tolerance schedule. Like most EMA parameters in deep learning (e.g., in batch normalization or Adam), it is not highly sensitive.
- **Window Size** (p): This is less of a hyperparameter and more of a hardware configuration parameter. For good efficiency, p can be simply set as the number of available processors.

C Detailed Comparative Analysis

In this section, we provide the detailed derivations and analyses of computational cost, memory footprint, and speedup ratios for sequential SGD and various parallel training methods, as summarized in Table 1.

C.1 Computational Cost and Memory Footprint Analysis

To quantify the overhead of PASO, let T be the total number of training steps for a standard sequential method. PASO converges in K iterations, with each iteration performing p parallel gradient computations (where p is the window size). The total maximum number of gradient computations is therefore $p \times K$. Since the sliding window size p will gradually decrease at the end of the convergence, the practical number for gradients computations (we denote it as G) is less than pK. We define the *computational cost ratio* m as the ratio of PASO's total gradient computations to that of the sequential method:

$$m = \frac{pK}{T}$$

Empirically, as shown in Figure 7, our experiments for $T = 10000$ demonstrate that m remains close to 1 and does not exceed 1.5 across various window sizes. This indicates that PASO introduces minimal computational overhead.

In terms of memory, PASO requires storing only one model and one optimizer state per device. This is identical to the requirements of sequential, model, and pipeline parallelism. It is also significantly more memory-efficient than data parallelism, where the storage for optimizer states typically scales with the number of devices N.

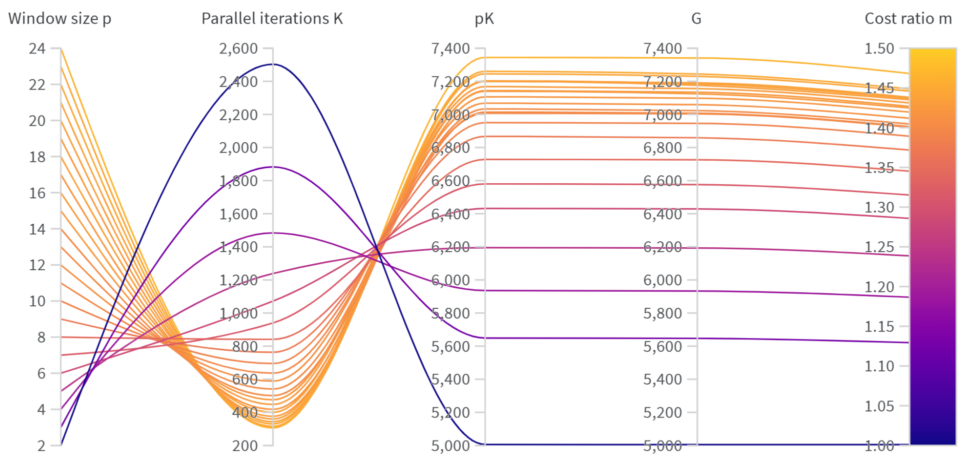


Figure 7: Empirical evaluation of the computational cost ratio $m = pK/T$ for $T = 10000$ across different window sizes p. Since the sliding window size p will gradually decrease at the end of the convergence, the actual total number of gradient computations (we denote it as G) is marginally less than pK. The ratio remains close to 1, indicating minimal computational overhead.

C.2 Speedup Ratio Analysis

In this section, we provide a detailed derivation of the speedup ratios for sequential SGD and various parallel training methods, as summarized in Table 1.

C.2.1 Definitions and Assumptions

For a clear and consistent analysis, we define the following notations:

- N: The number of GPUs, assumed to have identical compute capabilities.
- T: The total number of iterations (steps) required for a model to converge using sequential SGD.
- t_{comp}: The time required for the computation within one SGD step on a single GPU. For simplicity, we normalize this to t_{step} in some contexts.
- t_{comm}: The time required for necessary communication (e.g., synchronization) per parallel step.
- $\alpha \triangleq t_{\text{comm}}/t_{\text{comp}}$: The communication-to-computation ratio, a critical factor in parallel efficiency.

The **speedup ratio** S for any parallel method is defined as the ratio of the total time taken by sequential SGD to the time taken by the parallel method:

$$S = \frac{T_{\text{sequential}}}{T_{\text{parallel}}}$$

C.2.2 Baseline: Sequential SGD

The total time for sequential SGD is the product of the number of iterations and the time per iteration.

$$T_{\text{sequential}} = T \times t_{\text{comp}}$$

By definition, its speedup ratio is $S_{\text{sequential}} = 1$.

C.2.3 Data-Parallel Training

In synchronous data parallelism, the computation for each step is divided across N GPUs, but a communication step (e.g., AllReduce) is required to synchronize gradients. The time for one parallel step is $(\frac{t_{\text{comp}}}{N} + t_{\text{comm}})$. The total time over T iterations is:

$$T_{\text{data}} = T \times \left(\frac{t_{\text{comp}}}{N} + t_{\text{comm}} \right)$$

The speedup ratio is therefore:

$$S_{\text{data}} = \frac{T \cdot t_{\text{comp}}}{T \left(\frac{t_{\text{comp}}}{N} + t_{\text{comm}} \right)} = \frac{t_{\text{comp}}}{\frac{t_{\text{comp}}}{N} + \alpha \cdot t_{\text{comp}}} = \frac{1}{\frac{1}{N} + \alpha} = \frac{N}{1 + \alpha N}$$

In the communication-bound limit ($N \to \infty$), the speedup is capped at $S_{\text{data}} \to 1/\alpha$.

C.2.4 Model-Parallel and Pipeline-Parallel Training

For both model and pipeline parallelism, assuming perfect load balancing and ignoring initial pipeline-filling latency for large T, the computation is similarly distributed. The model is partitioned across N devices, and each device computes its part in parallel, followed by communication of activations or gradients between devices. The total time can be approximated as:

$$T_{\text{model/pipeline}} \approx T \times \left(\frac{t_{\text{comp}}}{N} + t_{\text{comm}} \right)$$

This yields the same speedup ratio form as data parallelism:

$$S_{\text{model/pipeline}} = \frac{N}{1 + \alpha N}$$

Practical limitations such as load imbalance or pipeline bubble latency often result in a lower effective speedup.

C.2.5 Step-Parallel Training (PASO)

PASO operates differently by parallelizing across training steps. We introduce three key parameters for its analysis:

- p: The window size, representing the number of gradient steps computed in parallel.
- K: The total number of PASO iterations required for convergence.
- m: The computational cost ratio, $m = \frac{pK}{T}$, where pK is the total number of gradient computations performed by PASO. Our empirical results show $m \approx 1$.

In each of the K iterations, p gradients are computed in parallel across N devices. The computation time per iteration is $\frac{p \cdot t_{\text{comp}}}{N}$, followed by a single communication phase t_{comm}. The total time for PASO is:

$$T_{\text{PASO}} = K \times \left(\frac{p \cdot t_{\text{comp}}}{N} + t_{\text{comm}} \right)$$

To compare this with sequential training over T steps, we substitute $K = \frac{mT}{p}$:

$$T_{\text{PASO}} = \frac{mT}{p} \left(\frac{p \cdot t_{\text{comp}}}{N} + t_{\text{comm}} \right) = mT \left(\frac{t_{\text{comp}}}{N} + \frac{t_{\text{comm}}}{p} \right)$$

The speedup ratio for PASO is then:

$$S_{\text{PASO}} = \frac{T \cdot t_{\text{comp}}}{mT \left(\frac{t_{\text{comp}}}{N} + \frac{t_{\text{comm}}}{p} \right)} = \frac{t_{\text{comp}}}{m \left(\frac{t_{\text{comp}}}{N} + \frac{\alpha \cdot t_{\text{comp}}}{p} \right)} = \frac{1}{m \left(\frac{1}{N} + \frac{\alpha}{p} \right)} = \frac{N}{m(1 + \alpha N/p)}$$

Since our experiments show $m \approx 1$ (see Figure 7), the speedup is approximately:

$$S_{\text{PASO}} \approx \frac{N}{1 + \alpha N/p}$$

As $p > 1$, it follows that $1 + \alpha N/p < 1 + \alpha N$, which confirms that $S_{\text{PASO}} > S_{\text{data/model/pipeline}}$. In the communication-bound limit ($N \to \infty$), the speedup is capped at $S_{\text{PASO}} \to p/(m\alpha)$, which is p times higher than other methods.

D Notation Summary

Table 8: Summary of Notations

Notation	Description
T	Total number of gradient descent steps
t	Current step index, $t \in \{0, 1, \ldots, T-1\}$
w_t	Model parameters at step t
η_t	Learning rate at step t
ζ_t	Mini-batch of data used at step t
$\mathcal{L}(w_t, \zeta_t)$	Loss function evaluated at parameters w_t with data ζ_t
$\nabla_{w_t}\mathcal{L}(w_t, \zeta_t)$	Gradient of loss with respect to w_t
$g_t(\cdot)$	Update function specific to optimizer (SGD, Adam, etc.)
r	The number of history weights used for existing autoregressive optimizers
$F_t(\cdot)$	Nonlinear equation function at step t
$\hat{w}_t^{(k)}$	Estimated parameters at step t, iteration k
K	Number of parallel iterations
p	Sliding window size for parallel computation
δ	Convergence tolerance threshold
λ	Exponential moving average decay rate
n	Dimension of model parameters
L	Lipschitz constant for gradients
M	Bound on gradient norm
ϵ	Small constant for numerical stability
β_1, β_2	Exponential decay rates for optimizer with momentum

E Update Rules for Various Optimizers in Definition 2

Adam Optimizer. At each iteration t, Adam computes the gradient of the loss function $\mathcal{L}(w_t, \zeta_t)$ over the mini-batch ζ_t. It then updates two key quantities: the first moment m_t, which captures the momentum of the gradients, and the second moment v_t, which estimates the variability of the gradients. These updates are governed by exponential moving averages:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_{w_{t-1}}\mathcal{L}(w_{t-1}, \zeta_{t-1}), v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_{w_{t-1}}\mathcal{L}(w_{t-1}, \zeta_{t-1}))^2, \tag{11}$$

where β_1 and β_2 are hyperparameters controlling the decay rates of the moving averages. Adam applies bias correction to the moments:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \hat{v}_t = \frac{v_t}{1-\beta_2^t}. \tag{12}$$

The model parameters w are then updated using the following rule:

$$w_t = w_{t-1} - \eta_{t-1}\frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}, \tag{13}$$

where the division is defined as the Hadamard division, and ϵ is a small constant.

For Adam , reformulating Eq. (11) produces the formulas of their general terms:

$$m_t = (1-\beta_1)\sum_{\tau=0}^{t-1}\beta_1^{t-1-\tau}\nabla_{w_\tau}\mathcal{L}(w_\tau, \zeta_\tau), v_t = (1-\beta_2)\sum_{\tau=0}^{t-1}\beta_2^{t-1-\tau}\left(\nabla_{w_\tau}\mathcal{L}(w_\tau, \zeta_\tau)\right)^2. \tag{14}$$

Through the combination of Eq. (12), Eq. (13), and Eq. (14), we derive g_{t-1} with $r = t$ for Adam as follows:

$$g_{t-1}(w_{t-1}, \cdots, w_0; \zeta_{t-1}, \cdots, \zeta_0) = \frac{\frac{1-\beta_1}{1-\beta_1^t}\sum_{\tau=0}^{t-1}\beta_1^{t-1-\tau}\nabla_{w_\tau}\mathcal{L}(w_\tau, \zeta_\tau)}{\sqrt{\frac{(1-\beta_2)\sum_{\tau=0}^{t-1}\beta_2^{t-1-\tau}(\nabla_{w_\tau}\mathcal{L}(w_\tau,\zeta_\tau))^2}{1-\beta_2^t} + \epsilon}}. \tag{15}$$

AdamW Optimizer. The explicit form of g_τ for AdamW is derived by decoupling weight decay from the Adam update rule. Let $r=\tau$, then:

$$g_\tau(w_\tau,\ldots,w_0;\zeta_\tau,\ldots,\zeta_0)=\frac{\frac{1-\beta_1}{1-\beta_1^{\tau+1}}\sum_{k=0}^{\tau}\beta_1^{\tau-k}\nabla_{w_k}\mathcal{L}(w_k,\zeta_k)}{\sqrt{\frac{(1-\beta_2)}{1-\beta_2^{\tau+1}}\sum_{k=0}^{\tau}\beta_2^{\tau-k}\left(\nabla_{w_k}\mathcal{L}(w_k,\zeta_k)\right)^2+\epsilon}}+\lambda w_\tau,$$

where λ is the weight decay coefficient. The term λw_τ is explicitly added to the original Adam update, independent of gradient history.

Adagrad Optimizer. For Adagrad, the update function g_τ is defined using the explicit sum of squared gradients up to iteration τ:

$$g_\tau(w_\tau,\ldots,w_0;\zeta_\tau,\ldots,\zeta_0)=\frac{\nabla_{w_\tau}\mathcal{L}(w_\tau,\zeta_\tau)}{\sqrt{\sum_{k=0}^{\tau}\left(\nabla_{w_k}\mathcal{L}(w_k,\zeta_k)\right)^2+\epsilon}}.$$

Here, the denominator is the square root of the *non-decaying cumulative sum* of all historical squared gradients. ϵ is a small constant added for numerical stability.

SAM Optimizer. The explicit form of g_τ for SAM (Sharpness-Aware Minimization) involves computing the gradient at a perturbed point. Here in SAM $r=1$, then:

$$g_\tau(w_\tau;\zeta_\tau)=\nabla_{w_\tau+\varepsilon_\tau}\mathcal{L}(w_\tau+\varepsilon_\tau,\zeta_\tau),$$

where the perturbation ε_τ is defined as:

$$\varepsilon_\tau=\rho\cdot\frac{\nabla_{w_\tau}\mathcal{L}(w_\tau,\zeta_\tau)}{\|\nabla_{w_\tau}\mathcal{L}(w_\tau,\zeta_\tau)\|_2+\delta}.$$

Substituting the expression for ε_τ into the gradient formula, we get:

$$g_\tau(w_\tau;\zeta_\tau)=\nabla_{w_\tau}\mathcal{L}\left(w_\tau+\rho\cdot\frac{\nabla_{w_\tau}\mathcal{L}(w_\tau,\zeta_\tau)}{\|\nabla_{w_\tau}\mathcal{L}(w_\tau,\zeta_\tau)\|_2+\epsilon},\zeta_\tau\right).$$

This formulation explicitly shows SAM computes the gradient at a point that is perturbed in the direction of steepest ascent within a neighborhood of radius ρ, seeking parameters that are robust to adversarial perturbations. Here, ρ is the perturbation radius that controls the magnitude of the perturbation, and ϵ is a small constant added for numerical stability.

F PROOF OF PROPOSITION 1

We begin by showing that the function $F_{t-1}(\hat{w}_0,\cdots,\hat{w}_{t-1};\zeta_0,\ldots,\zeta_{t-1})$ has a unique set of solutions. Assume there exist two distinct solutions, $A_0,\cdots,A_T$ and $B_0,\cdots,B_T$. For all $t\in\{0,T\}$, these solutions must satisfy:

$$\begin{cases}A_t=F_{t-1}(A_0,\cdots,A_{t-1};\zeta_0,\ldots,\zeta_{t-1})\\ B_t=F_{t-1}(B_0,\cdots,B_{t-1};\zeta_0,\ldots,\zeta_{t-1}).\end{cases}\tag{16}$$

By mathematical induction, suppose $A_\tau=B_\tau$ for $0\le\tau\le t$. Then,

$$A_{t+1}=F_t(A_0,\cdots,A_t;\zeta_0,\ldots,\zeta_t)=F_t(B_0,\cdots,B_t;\zeta_0,\ldots,\zeta_t)=B_{t+1},\tag{17}$$

which implies $A_0,\cdots,A_T$ and $B_0,\cdots,B_T$ are identical. Thus, the solution is unique.

Next, we show that the solution of the triangular nonlinear equation system is an unbiased estimator for the autoregressive gradient descent process. From Eq. (3), the expectation of the autoregressive GD process is:

$$\begin{aligned}E[w_t]&=E[F_{t-1}(w_0,\cdots,w_{t-1};\zeta_0,\cdots,\zeta_{t-1})]\\&=E[w_0]-\sum_{\tau=0}^{t-1}E\left[\eta_\tau g_\tau(w_\tau,\ldots,w_{\tau-r+1};\zeta_\tau,\ldots,\zeta_{\tau-r+1})\right].\end{aligned}\tag{18}$$

For the triangular NE system, we have:

$$\begin{aligned} E[\hat{w}_t] &= E[F_{t-1}(\hat{w}_0, \cdots, \hat{w}_{t-1}; \zeta_0, \ldots, \zeta_{t-1})] \\ &= E[\hat{w}_0] - E\left[\sum_{\tau=0}^{t-1} \eta_\tau g_\tau(\hat{w}_\tau, \ldots, \hat{w}_{\tau-r+1}; \zeta_\tau, \ldots, \zeta_{\tau-r+1})\right]. \end{aligned} \tag{19}$$

Since w_0 and $\hat{w}_0$ follow the same distribution, and η_t and the mini-batches ζ_t are identical across all time steps, it follows that:

$$E[\hat{w}_t] = E[w_t], \quad \forall\, 0 \le t \le T.$$

G Proof of Convergence for Fixed-Point Iteration in Proposition 2

G.1 Assumptions and Lemmas

To give the proof, we first state the underlying assumptions and lemmas used:

Assumption 1. *The gradient $\nabla_{w_\tau}\mathcal{L}(w_\tau, \zeta_\tau)$ is L-Lipschitz continuous:*

$$\|\nabla_{w_\tau}\mathcal{L}(w_\tau, \zeta_\tau) - \nabla_{w_\tau}\mathcal{L}(x_\tau, \zeta_\tau)\| \le L\|w_\tau - x_\tau\|. \tag{20}$$

Assumption 2. *The gradient norm is bounded:*

$$\|\nabla_{w_\tau}\mathcal{L}(w_\tau, \zeta_\tau)\| \le M. \tag{21}$$

This implies bounded model weights w_τ. For example, consider the simply quadratic loss $\mathcal{L}(w, \zeta) = w^2$ (with $w \in \mathbb{R}$ independent of ζ). Here:

$$\nabla_w\mathcal{L} = 2w, \quad \textit{so} \quad |\nabla_w\mathcal{L}| = |2w|.$$

The bounded gradient condition $|2w| \le M$ directly implies $|w| \le M/2$, proving w is constrained to a compact set.

Lemma 1. *If $U, V \in \mathbb{R}^{n\times t}$ satisfy $U_{ij}, V_{ij} \ge \mu$ for all i, j, then*

$$\|\sqrt{U} - \sqrt{V}\|_F \;\le\; \frac{1}{2\sqrt{\mu}}\,\|U - V\|_F,$$

where the square-root is taken element-wise.

Proof. For any scalars $a, b \ge \mu > 0$,

$$\left|\sqrt{a} - \sqrt{b}\right| = \frac{|a-b|}{\sqrt{a}+\sqrt{b}} \;\le\; \frac{|a-b|}{2\sqrt{\mu}},$$

because $\sqrt{a} + \sqrt{b} \ge 2\sqrt{\mu}$.

Applying this entrywise with $a = U_{ij}$ and $b = V_{ij}$ yields

$$\left|\sqrt{U_{ij}} - \sqrt{V_{ij}}\right| \;\le\; \frac{1}{2\sqrt{\mu}}\,|U_{ij} - V_{ij}| \qquad (\forall\, i, j).$$

Squaring and summing over (i, j),

$$\sum_{i,j}\left(\sqrt{U_{ij}} - \sqrt{V_{ij}}\right)^2 \;\le\; \frac{1}{4\mu}\sum_{i,j}(U_{ij} - V_{ij})^2.$$

The left–hand side equals $\|\sqrt{U} - \sqrt{V}\|_F^2$ and the right–hand side equals $\frac{1}{4\mu}\|U - V\|_F^2$.

Taking square roots gives

$$\|\sqrt{U} - \sqrt{V}\|_F \ \le \ \frac{1}{2\sqrt{\mu}} \|U - V\|_F,$$

□

G.2 Problem Restatement

Notation. We denote the collection of weights up to time τ as $W_\tau = [\hat{w}_0, \dots, \hat{w}_\tau]$ and note $W_{T-1} = [\hat{w}_0, \dots, \hat{w}_{T-1}]$ as W. The norm $\|\cdot\|$ is the Frobenius norm. For model weights $w \in \mathbb{R}^n$ with $n > 1$, multiplication and division are element-wise (Hadamard product and division).

Definition 3 (Iterative Mapping). *Let the iterative mapping $\mathcal{H} : \mathbb{R}^{n \times T} \to \mathbb{R}^{n \times T}$ (T components) be defined as follows for a sequence of model weights $W = [\hat{w}_0, \hat{w}_1, \dots, \hat{w}_{T-1}]$:*

$$\mathcal{H}(\hat{w}_0, \cdots, \hat{w}_{T-1}) = \begin{cases} \hat{w}_0 = w_0^{seq}, \\ F_0(\hat{w}_0; \zeta_0), \\ F_1(\hat{w}_0, \hat{w}_1; \zeta_0, \zeta_1), \\ \cdots, \\ F_{T-1}(\hat{w}_0, \cdots, \hat{w}_{T-1}; \zeta_0, \dots, \zeta_{T-1}), \end{cases} \tag{22}$$

where w_0^{seq} denotes the initialized model for the sequential gradient descent and each sub-mapping F_{t-1} is of the form:

$$F_{t-1}(\hat{w}_0, \cdots, \hat{w}_{t-1}) = \hat{w}_0 - \sum_{\tau=0}^{t-1} \eta_\tau g_\tau(\hat{w}_\tau, \dots, \hat{w}_0). \tag{23}$$

The fixed-point iteration is thus defined by the sequence $W^k = \mathcal{H}(W^{k-1})$.

Definition 4 (Autoregressive Gradient Descent Trajectory). *The target fixed point, denoted by $W^{seq} = [w_0^{seq}, w_1^{seq}, \dots, w_{T-1}^{seq}]$, is the trajectory generated by autoregressive gradient descent:*

$$w_0^{seq} = \textit{initial model weight} \tag{24}$$

$$w_t^{seq} = w_0^{seq} - \sum_{\tau=0}^{t-1} \eta_\tau g_\tau(w_\tau^{seq}, \dots, w_0^{seq}) \quad \textit{for } t \ge 1 \tag{25}$$

It is straightforward to see that W^{seq} is a fixed point of $\mathcal{H}$, since $\mathcal{H}(W^{seq}) = W^{seq}$.

G.3 Objectives

We aim to prove two key properties of this iterative process:

1. **Convergence:** The fixed-point iteration $W^k = \mathcal{H}(W^{k-1})$ converges to the unique fixed point W^{seq}, which corresponds to the trajectory of autoregressive gradient descent.
2. **Finite Convergence Steps:** In the worst-case scenario, the number of iterations K required for convergence ($W^K = W^{seq}$) is at most T.

G.4 Proof of Convergence (Objective 1)

We will prove by mathematical induction on the time step t that for each $t \in \{0, \dots, T-1\}$, the sequence of iterates $\{\hat{w}_t^k\}_{k=1}^\infty$ converges to w_t^{seq}.

Proof. Let $W^k = [\hat{w}_0^k, \dots, \hat{w}_{T-1}^k]$ be the iterates at step k. From the definition of $\mathcal{H}$, we have:

$$\hat{w}_0^k = w_0^{seq} \tag{26}$$

$$\hat{w}_t^k = \hat{w}_0^{k-1} - \sum_{\tau=0}^{t-1} \eta_\tau g_\tau(\hat{w}_\tau^{k-1}, \dots, \hat{w}_0^{k-1}) \quad \text{for } t \ge 1 \tag{27}$$

Base Case ($t = 0$): From the definition of $\mathcal{H}$, $\hat{w}_0^k = w_0^{seq}$ for all $k \geq 1$. Thus,

$$\lim_{k\to\infty} \left\| \hat{w}_0^k - w_0^{seq} \right\|_F = 0$$

The base case holds trivially.

Inductive Hypothesis: Assume for a given $t \geq 0$ that for all $\tau \in \{0, \ldots, t\}$, we have:

$$\lim_{k\to\infty} \left\| \hat{w}_\tau^k - w_\tau^{seq} \right\|_F = 0$$

Inductive Step: We must show that the statement holds for $t+1$, i.e., $\lim_{k\to\infty} \left\| \hat{w}_{t+1}^k - w_{t+1}^{seq} \right\|_F = 0$.

The iterate $\hat{w}_{t+1}^k$ and the target w_{t+1}^{seq} are given by:

$$\hat{w}_{t+1}^k = \hat{w}_0^{k-1} - \sum_{\tau=0}^{t} \eta_\tau g_\tau(W_\tau^{k-1})$$

$$w_{t+1}^{seq} = w_0^{seq} - \sum_{\tau=0}^{t} \eta_\tau g_\tau(W_\tau^{seq})$$

Since $\hat{w}_0^{k-1} = w_0^{seq}$ for $k - 1 \geq 1$, the difference is:

$$\hat{w}_{t+1}^k - w_{t+1}^{seq} = \sum_{\tau=0}^{t} \eta_\tau \left(g_\tau(W_\tau^{seq}) - g_\tau(W_\tau^{k-1}) \right)$$

Taking the norm and applying the triangle inequality:

$$\left\| \hat{w}_{t+1}^k - w_{t+1}^{seq} \right\|_F \leq \sum_{\tau=0}^{t} \eta_\tau \left\| g_\tau(W_\tau^{k-1}) - g_\tau(W_\tau^{seq}) \right\|_F$$

From Appendix G.6, G.7, and G.8. we can know that the gradient function g_τ for various optimizers is upper bounded with respect to its arguments. Denote uniformly by these boundaries C, we have:

$$\left\| \hat{w}_{t+1}^k - w_{t+1}^{seq} \right\|_F \leq \sum_{\tau=0}^{t} \eta_\tau C \left\| W_\tau^{k-1} - W_\tau^{seq} \right\|_F$$

By the inductive hypothesis, for each $\tau \in \{0, \ldots, t\}$, every component of W_τ^{k-1} converges to the corresponding component of W_τ^{seq} as $k \to \infty$. This implies that:

$$\lim_{k\to\infty} \left\| W_\tau^{k-1} - W_\tau^{seq} \right\|_F = \lim_{k\to\infty} \left(\sum_{j=0}^{\tau} \left\| \hat{w}_j^{k-1} - w_j^{seq} \right\|_F^2 \right)^{1/2} = 0$$

Since the sum on the right-hand side is a finite sum of terms each converging to zero, the entire expression converges to zero:

$$\lim_{k\to\infty} \left\| \hat{w}_{t+1}^k - w_{t+1}^{seq} \right\|_F \leq \sum_{\tau=0}^{t} \eta_\tau C \cdot 0 = 0$$

As the norm is non-negative, we conclude $\lim_{k\to\infty} \left\| \hat{w}_{t+1}^k - w_{t+1}^{seq} \right\|_F = 0$. This completes the inductive step.

By the principle of mathematical induction, $\hat{w}_t^k \to w_t^{seq}$ for all $t \in \{0, \ldots, T-1\}$. Therefore, the iteration $W^k = \mathcal{H}(W^{k-1})$ converges to W^{seq}. $\square$

G.5 Proof of Convergence Steps (Objective 2)

We now prove a stronger result: in worst-case scenario, the fixed-point iteration converges to the exact fixed point W^{seq} in at most T iterations.

Worst-Case Scenario Analysis. The structure of the mapping $\mathcal{H}$ imposes a causal dependency: the calculation of $\hat{w}_t^k$ depends only on the components $\hat{w}_0^{k-1}, \dots, \hat{w}_{t-1}^{k-1}$ from the previous iteration. The initial models for the fixed-point iteration and the autoregressive gradient descent are identical at $t = 0$ ($\hat{w}_0^k = w_0^{seq}$). Consequently, convergence cannot occur "out of order". The component $\hat{w}_1$ can only converge after $\hat{w}_0$ has, $\hat{w}_2$ can only converge after $\hat{w}_0$ and $\hat{w}_1$ have, and so on.

The worst-case scenario occurs when each iteration k can only ensure the convergence of one component, leading to the convergence proceeding sequentially, one component at a time. This sequential "locking-in" of the correct values is equivalent in its step-by-step nature to the autoregressive gradient descent. We will formalize this intuition below.

Proof. We will prove by induction on the component index t the statement $P(t)$:

$$P(t): \quad \hat{w}_t^k = w_t^{seq} \quad \text{for all } k \geq t+1.$$

Base Case ($t = 0$): We must prove $P(0)$: $\hat{w}_0^k = w_0^{seq}$ for all $k \geq 1$. By the definition of $\mathcal{H}$ in Eq. (22), $\hat{w}_0^k$ is set to w_0^{seq} for every iteration $k \geq 1$. The base case holds.

Inductive Hypothesis: Assume for some $t \geq 1$ that $P(\tau)$ holds for all $\tau \in \{0, 1, \dots, t-1\}$. This means for each such τ:

$$\hat{w}_\tau^k = w_\tau^{seq} \quad \text{for all } k \geq \tau + 1.$$

Inductive Step: We must prove that $P(t)$ holds: $\hat{w}_t^k = w_t^{seq}$ for all $k \geq t+1$.

Consider an arbitrary iteration k such that $k \geq t+1$. This implies $k - 1 \geq t$. The iterate $\hat{w}_t^k$ is defined as:

$$\hat{w}_t^k = \hat{w}_0^{k-1} - \sum_{\tau=0}^{t-1} \eta_\tau g_\tau(\hat{w}_\tau^{k-1}, \dots, \hat{w}_0^{k-1}).$$

The arguments to the functions g_τ are the components of W^{k-1}. Let's examine an arbitrary component $\hat{w}_\tau^{k-1}$ in this expression, where $\tau \in \{0, 1, \dots, t-1\}$. From our condition on k, we have $k - 1 \geq t > \tau$, which implies $k - 1 \geq \tau + 1$.

According to our inductive hypothesis, since $k - 1 \geq \tau + 1$, each of these components has already converged to its final value:

$$\hat{w}_\tau^{k-1} = w_\tau^{seq} \quad \text{for each } \tau \in \{0, 1, \dots, t-1\}.$$

This demonstrates that for any iteration $k \geq t+1$, all the inputs required to compute $\hat{w}_t^k$ have already stabilized to their fixed-point values at the preceding step, $k - 1$.

Substituting these converged values back into the expression for $\hat{w}_t^k$:

$$\hat{w}_t^k = w_0^{seq} - \sum_{\tau=0}^{t-1} \eta_\tau g_\tau(w_\tau^{seq}, \dots, w_0^{seq}).$$

The right-hand side of this equation is precisely the definition of the target sequential weight w_t^{seq}. Therefore,

$$\hat{w}_t^k = w_t^{seq}.$$

Since our choice of $k \geq t+1$ was arbitrary, this equality holds for all such k. This proves $P(t)$ and completes the inductive step.

Conclusion on Iteration Count. By induction, we have shown that $\hat{w}_t^k = w_t^{seq}$ for all $k \geq t+1$. For the entire vector $W^k = [\hat{w}_0^k, \dots, \hat{w}_{T-1}^k]$ to converge, every component must have converged. The last component to converge is $\hat{w}_{T-1}^k$. Applying our result for $t = T - 1$:

$$\hat{w}_{T-1}^k = w_{T-1}^{seq} \quad \text{for all } k \geq (T-1) + 1 = T.$$

At iteration $k = T$, we have $T \geq t+1$ for all $t \in \{0, \dots, T-1\}$. This implies that every component $\hat{w}_t^T$ has converged to w_t^{seq}. Thus, the entire vector has converged:

$$W^T = W^{seq}.$$

Therefore, the fixed-point iteration requires exactly $K = T$ iterations to converge to the fixed point in the worst case, and it remains there for all subsequent iterations. The number of iterations K required does not exceed the number of autoregressive steps T. □

G.6 Upper Bound for the Difference of g_t in SGD

For SGD, the update function g_t takes the form:

$$g_t(w_t;\zeta_t) = \nabla_{w_t}\mathcal{L}(w_t,\zeta_t) \tag{28}$$

We aim to find an upper bound for $\|g_t(w_t)-g_t(x_t)\|$. By directly applying Assumption 1 (L-Lipschitz continuity), we get:

$$\|g_t(w_t)-g_t(x_t)\| = \|\nabla_{w_t}\mathcal{L}(w_t,\zeta_t) - \nabla_{x_t}\mathcal{L}(x_t,\zeta_t)\| \leq L\|w_t - x_t\| \tag{29}$$

Therefore, for SGD, the Lipschitz constant of the update function g_t is L.

G.7 Upper Bound for the Difference of g_t in Adam

Notation. We denote the collection of weights up to time t as $W_\tau = [w_0,\ldots,w_t]$ and note $W_{T-1} = [w_0,\ldots,w_{T-1}]$ as W. Analogously, $X_t = [x_0,\ldots,x_t]$ and $X = [x_0,\ldots,x_{T-1}]$.

Our objective is to derive an upper bound for the difference $\|g_{t-1}(W_{t-1}) - g_{t-1}(X_{t-1})\|_F$ for any W_{t-1} and X_{t-1}.

The function g_{t-1} is defined as:

$$g_{t-1}(W_{t-1}) = \frac{A(W_{t-1})}{\sqrt{B(W_{t-1})+\epsilon}} \tag{30}$$

where the division and square root are element-wise operations. The numerator $A(W_{t-1})$ and denominator component $B(W_{t-1})$ are defined as the bias-corrected first and second moment estimates:

$$A(W_{t-1}) = \frac{1-\beta_1}{1-\beta_1^t}\sum_{\tau=0}^{t-1}\beta_1^{t-1-\tau}\nabla_{w_\tau}\mathcal{L}(w_\tau,\zeta_\tau) \tag{31}$$

$$B(W_{t-1}) = \frac{1-\beta_2}{1-\beta_2^t}\sum_{\tau=0}^{t-1}\beta_2^{t-1-\tau}\left(\nabla_{w_\tau}\mathcal{L}(w_\tau,\zeta_\tau)\right)^2 \tag{32}$$

This proof relies on two standard assumptions:

1. L-Lipschitz Gradient: The gradient of the loss function is L-Lipschitz continuous, i.e., $\|\nabla\mathcal{L}(w) - \nabla\mathcal{L}(x)\|_F \leq L\|w-x\|_F$.

2. Bounded Gradient Norm: The Frobenius norm of the stochastic gradients is uniformly bounded by a constant M, i.e., $\|\nabla\mathcal{L}(w,\zeta)\|_F \leq M$.

For clarity, we will temporarily omit the subscript $t-1$ from W and X within the derivation and re-introduce it in the final result. We begin by decomposing the difference $g(W)-g(X)$ by adding and subtracting an intermediate term:

$$g(W)-g(X) = \left(\frac{A(W)-A(X)}{\sqrt{B(W)+\epsilon}}\right) + \left(\frac{A(X)}{\sqrt{B(W)+\epsilon}} - \frac{A(X)}{\sqrt{B(X)+\epsilon}}\right) \tag{33}$$

This can be expressed using the element-wise Hadamard product ($\odot$) as:

$$g(W)-g(X) = (A(W)-A(X))\odot\frac{1}{\sqrt{B(W)+\epsilon}} + A(X)\odot\left(\frac{1}{\sqrt{B(W)+\epsilon}} - \frac{1}{\sqrt{B(X)+\epsilon}}\right) \tag{34}$$

By applying the triangle inequality to the Frobenius norm, we get:

$$\|g(W)-g(X)\|_F \leq \left\|(A(W)-A(X))\odot\frac{1}{\sqrt{B(W)+\epsilon}}\right\|_F + \left\|A(X)\odot\left(\frac{1}{\sqrt{B(W)+\epsilon}} - \frac{1}{\sqrt{B(X)+\epsilon}}\right)\right\|_F \tag{35}$$

Next, we use the property of the Hadamard product, $\|U \odot V\|_F \leq \|U\|_{\max}\|V\|_F$, where $\|U\|_{\max}$ is the maximum absolute value of any element in U. This yields our main inequality:

$$\|g(W)-g(X)\|_F \leq \left\|\frac{1}{\sqrt{B(W)+\epsilon}}\right\|_{\max} \|A(W)-A(X)\|_F + \|A(X)\|_{\max} \left\|\frac{1}{\sqrt{B(W)+\epsilon}} - \frac{1}{\sqrt{B(X)+\epsilon}}\right\|_F \tag{36}$$

We now bound the four terms in Eq. (36).

1. Bound for $\|A(W_{t-1}) - A(X_{t-1})\|_F$

From the definition in Eq. (31), we have:

$$A(W) - A(X) = \frac{1-\beta_1}{1-\beta_1^t} \sum_{\tau=0}^{t-1} \beta_1^{t-1-\tau} \left(\nabla_{w_\tau}\mathcal{L}(w_\tau, \zeta_\tau) - \nabla_{x_\tau}\mathcal{L}(x_\tau, \zeta_\tau)\right) \tag{37}$$

Taking the Frobenius norm and applying the triangle inequality, then using the L-Lipschitz assumption and the fact that $\|w_\tau - x_\tau\|_F \leq \|W - X\|_F$:

$$\begin{aligned}\|A(W) - A(X)\|_F &\leq \frac{1-\beta_1}{1-\beta_1^t} \sum_{\tau=0}^{t-1} \beta_1^{t-1-\tau} \|\nabla_{w_\tau}\mathcal{L}(w_\tau, \zeta_\tau) - \nabla_{x_\tau}\mathcal{L}(x_\tau, \zeta_\tau)\|_F \\ &\leq \frac{1-\beta_1}{1-\beta_1^t} \sum_{\tau=0}^{t-1} \beta_1^{t-1-\tau} L\|w_\tau - x_\tau\|_F \\ &\leq L\|W - X\|_F \left(\frac{1-\beta_1}{1-\beta_1^t} \sum_{\tau=0}^{t-1} \beta_1^{t-1-\tau}\right)\end{aligned} \tag{38}$$

The sum of the bias-correction weights is equal to one. Thus, we have:

$$\|A(W_{t-1}) - A(X_{t-1})\|_F \leq L\|W_{t-1} - X_{t-1}\|_F \tag{39}$$

2. Bound for $\left\|\frac{1}{\sqrt{B(W)+\epsilon}}\right\|_{\max}$

Since each entry of $B(W)$ is a weighted average of squared gradients, $B_{ij}(W) \geq 0$ for all i, j. It follows that $\sqrt{B_{ij}(W)+\epsilon} \geq \sqrt{\epsilon}$. Taking the reciprocal gives the bound:

$$\left\|\frac{1}{\sqrt{B(W)+\epsilon}}\right\|_{\max} = \max_{i,j} \frac{1}{\sqrt{B_{ij}(W)+\epsilon}} \leq \frac{1}{\sqrt{\epsilon}} \tag{40}$$

3. Bound for $\|A(X)\|_{\max}$

Given the bounded gradient assumption $\|\nabla\mathcal{L}\|_F \leq M$, and since $\|\cdot\|_{\max} \leq \|\cdot\|_F$, we have $\|\nabla\mathcal{L}\|_{\max} \leq M$.

$$\begin{aligned}\|A(X)\|_{\max} &\leq \left\|\frac{1-\beta_1}{1-\beta_1^t} \sum_{\tau=0}^{t-1} \beta_1^{t-1-\tau} \nabla_{x_\tau}\mathcal{L}(x_\tau, \zeta_\tau)\right\|_{\max} \\ &\leq \frac{1-\beta_1}{1-\beta_1^t} \sum_{\tau=0}^{t-1} \beta_1^{t-1-\tau} \|\nabla_{x_\tau}\mathcal{L}\|_{\max} \leq M\end{aligned} \tag{41}$$

4. Bound for $\left\|\frac{1}{\sqrt{B(W)+\epsilon}} - \frac{1}{\sqrt{B(X)+\epsilon}}\right\|_F$

Let $u = B(W) + \epsilon$ and $v = B(X) + \epsilon$. We have:

$$\left\|\frac{1}{\sqrt{u}} - \frac{1}{\sqrt{v}}\right\|_F = \left\|\frac{\sqrt{v} - \sqrt{u}}{\sqrt{u}\sqrt{v}}\right\|_F \leq \left\|\frac{1}{\sqrt{uv}}\right\|_{\max} \|\sqrt{v} - \sqrt{u}\|_F \leq \frac{1}{\epsilon}\|\sqrt{v} - \sqrt{u}\|_F \tag{42}$$

The function $f(x) = \sqrt{x}$ is $\frac{1}{2\sqrt{\epsilon}}$-Lipschitz on $[\epsilon, \infty)$, which implies $\|\sqrt{v} - \sqrt{u}\|_F \leq \frac{1}{2\sqrt{\epsilon}}\|v - u\|_F$ (see Lemma 1). Therefore:

$$\left\| \frac{1}{\sqrt{B(W) + \epsilon}} - \frac{1}{\sqrt{B(X) + \epsilon}} \right\|_F \leq \frac{1}{2\epsilon^{3/2}} \|B(W) - B(X)\|_F \tag{43}$$

To complete this bound, we must bound $\|B(W) - B(X)\|_F$. From Eq. (32), we analyze the difference of squares term $(\nabla_{w_\tau}\mathcal{L})^2 - (\nabla_{x_\tau}\mathcal{L})^2 = (\nabla_{w_\tau}\mathcal{L} - \nabla_{x_\tau}\mathcal{L}) \odot (\nabla_{w_\tau}\mathcal{L} + \nabla_{x_\tau}\mathcal{L})$. Taking the norm:

$$\begin{aligned} \|(\nabla_{w_\tau}\mathcal{L})^2 - (\nabla_{x_\tau}\mathcal{L})^2\|_F &\leq \|\nabla_{w_\tau}\mathcal{L} - \nabla_{x_\tau}\mathcal{L}\|_F \cdot \|\nabla_{w_\tau}\mathcal{L} + \nabla_{x_\tau}\mathcal{L}\|_{\max} \\ &\leq (L\|w_\tau - x_\tau\|_F) \cdot (\|\nabla_{w_\tau}\mathcal{L}\|_{\max} + \|\nabla_{x_\tau}\mathcal{L}\|_{\max}) \\ &\leq (L\|w_\tau - x_\tau\|_F) \cdot (M + M) = 2LM\|w_\tau - x_\tau\|_F \end{aligned} \tag{44}$$

Summing over τ with the bias-corrected weights gives $\|B(W) - B(X)\|_F \leq 2LM\|W - X\|_F$. Substituting this into Eq. (43):

$$\left\| \frac{1}{\sqrt{B(W) + \epsilon}} - \frac{1}{\sqrt{B(X) + \epsilon}} \right\|_F \leq \frac{2LM}{2\epsilon^{3/2}} \|W - X\|_F = \frac{LM}{\epsilon^{3/2}} \|W - X\|_F \tag{45}$$

Final Result. We now substitute the bounds from Eq. (39), Eq. (40), Eq. (41), and Eq. (45) into our main inequality Eq. (36).

$$\begin{aligned} \|g_{t-1}(W) - g_{t-1}(X)\|_F &\leq \left(\frac{1}{\sqrt{\epsilon}}\right) \cdot (L\|W - X\|_F) + (M) \cdot \left(\frac{LM}{\epsilon^{3/2}}\|W - X\|_F\right) \\ &= \left(\frac{L}{\sqrt{\epsilon}} + \frac{M^2 L}{\epsilon^{3/2}}\right) \|W_{t-1} - X_{t-1}\|_F \end{aligned} \tag{46}$$

This final result provides an upper bound for the difference in the Adam update step that depends only on the problem constants L, M, ϵ.

G.8 Upper Bound for the Difference of g_t in AdamW

The AdamW update function g_t can be decomposed into the Adam update term and a decoupled weight decay term:

$$g_t(W_t) = g_t^{\text{Adam}}(W_t) + \lambda_t w_t \tag{47}$$

where λ_t is the weight decay coefficient. We analyze the norm of its difference using the triangle inequality:

$$\|g_t(W) - g_t(X)\|_F = \|(g_t^{\text{Adam}}(W) - g_t^{\text{Adam}}(X)) + \lambda_t(w_t - x_t)\|_F \tag{48}$$

$$\leq \|g_t^{\text{Adam}}(W) - g_t^{\text{Adam}}(X)\|_F + \lambda_t\|w_t - x_t\|_F \tag{49}$$

We now substitute the final bound derived for the Adam component in Appendix G.7:

$$\|g_t^{\text{Adam}}(W) - g_t^{\text{Adam}}(X)\|_F \leq \left(\frac{L}{\sqrt{\epsilon}} + \frac{M^2 L}{\epsilon^{3/2}}\right) \|W_t - X_t\|_F \tag{50}$$

Assuming an upper bound for the weight decay coefficient, $\lambda_t \leq \lambda_{\max}$, and noting that $\|w_t - x_t\|_F \leq \|W_t - X_t\|_F$, we have:

$$\|g_t(W) - g_t(X)\|_F \leq \left(\frac{L}{\sqrt{\epsilon}} + \frac{M^2 L}{\epsilon^{3/2}}\right) \|W_t - X_t\|_F + \lambda_{\max}\|W_t - X_t\|_F \tag{51}$$

$$= \left(\lambda_{\max} + \frac{L}{\sqrt{\epsilon}} + \frac{M^2 L}{\epsilon^{3/2}}\right) \|W_t - X_t\|_F \tag{52}$$

This provides a rigorous upper bound for the difference in the AdamW update step.