EIPO: Efficient Inter-Step Parallel Optimization

Jianrong Lu[1], **Zhuoya Gu**[2], **Zhiyu Zhu**[3,1], **Hui Liu**[4], **Junhui Hou**[1,*]
[1]Department of Computer Science, City University of Hong Kong, Hong Kong SAR, China
[2]Zhejiang University, Hangzhou, China
[3]City University of Hong Kong (Dongguan), Dongguan, China
[4]School of Computing and Information Sciences, Saint Francis University, Hong Kong SAR, China
jrong.alvin@gmail.com; zy_gu@zju.edu.cn; zhiyuzhu2@my.cityu.edu.hk
h2liu@sfu.edu.hk;jh.hou@cityu.edu.hk
*Corresponding author

Abstract

The wall-clock cost of training large neural networks is dominated by the long horizon of sequential optimizer iterations. We recast the gradient-descent (GD) trajectory of modern optimizer (e.g., SGD and Adam) as the unique root of a nonlinear operator, enabling a parallel root-finding approach that updates multiple GD steps simultaneously. We then propose Efficient Inter-step Parallel Optimization (EIPO) for accelerating a wide range of optimizers. EIPO contributes four pieces: (i) a Preconditioned Nonlinear-Equations (PNE) formulation that recovers the exact sequential GD trajectory; (ii) Memory-Efficient Adaptive Anderson Acceleration, which extracts the multisecant geometry directly from the optimizer's intrinsic momentum buffers and therefore avoids the memory overhead of classical Anderson Acceleration. (iii) We prove that EIPO converges to the optimizer's GD trajectory within fewer iterations than its sequential counterpart. (iv) Across language modeling on WikiText-2 (GPT-2, Llama-3.2-1B, Qwen-1.5-4B/2.5-3B, Gemma-2B, GPT-J-**6B**), image classification on CIFAR-10 (ResNet50, ViT, CNN), and diffusion training on LSUN Church (UNet/DDPM/DDIM), EIPO reduces optimizer iterations by up to **21**$\times$ and wall-clock time by up to **4.6**$\times$ while matching baseline perplexity, accuracy, and FID, all under comparable GPU memory and token consumption. Source code is available at: https://github.com/Jianrong-Lu/EIPO.git.

1 Introduction

Modern deep learning increasingly resembles a numerical-linear-algebra workload: training a large neural network w involves iteratively applying a fixed GD-based update operator $F_{t-1}(\cdot)$ with $w_t = F_{t-1}(w_{t-1}, \cdots, w_0; \zeta_{t-1}, \ldots, \zeta_0)$ to a long sequence of weights $w_0, w_1, \ldots, w_T$, given a sequence of data mini-batches $\{\zeta_\tau\}_{t=0}^{t-1}$. Because each application of F_{t-1} depends on $w_0, w_1, \ldots, w_{t-1}$, the GD trajectory is computed strictly left-to-right (i.e., autoregressive dependence). This structural rigidity is the proximate cause of the *million-GPU-hour* budgets reported for state-of-the-art LLM training [54, 39, 79, 9], and it is essentially orthogonal to the operator-design question that drives the optimizer (e.g., Adam) literature [34, 16, 77, 42, 41, 51, 55, 93, 62, 10, 30]. We approach the problem from a different vantage point: *numerical analysis of nonlinear systems*. Fix any modern optimizer (SGD, Adam, AdamW, ...). The trajectory $\{w_t\}_{t=0}^{T}$ that the optimizer would emit is the unique zero of a piecewise operator

$$\mathcal{H}(\hat{w}_0, \ldots, \hat{w}_T) \;=\; \begin{cases} \hat{w}_0 - w_0, \\ \hat{w}_t - F_{t-1}(\hat{w}_{t-1}, \ldots, \hat{w}_0; \zeta_{t-1}, \ldots, \zeta_0), \quad t \in [1, T]. \end{cases}$$

where $\{\hat{w}_t\}_{t=0}^{T}$ declare every entry of the sequential GD trajectory $\{w_t\}_{t=0}^{T}$ as an independent unknown variable $\hat{w}_t$.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

This is a standard system of nonlinear equations (NEs) [35, 32]: it has a unique root, and the root is precisely the sequence $\{w_t\}_{t=0}^{T}$ the GD-based optimizers would produce. The autoregressive "for-loop" view of the GD optimizers is therefore only a particular numerical scheme for this system—specifically, forward substitution [24]. From this perspective, the GD trajectory $\{w_t\}_{t=0}^{T}$ can be obtained concurrently via finding the roots of $\mathcal{H} = 0$. As a result, the classical autoregressive question is not whether parallel solvers *exist*, but which one is best suited to the structure of $\mathcal{H}$ in the deep-learning regime.

Why a Quasi-Newton viewpoint matters. For nonlinear systems with strong sequential coupling, decades of numerical-analysis research show that Newton-type methods dominate fixed-point iteration whenever a reasonable inverse-Jacobian approximation is available [2, 80, 56, 78, 20]. Anderson Acceleration (AA) [80], in particular, builds an explicit multisecant approximation of the inverse Jacobian of $\mathcal{H}$ from a buffer of q historical residuals. AA enjoys local superlinear convergence on $\mathcal{H} = 0$, but classical AA stores $\mathcal{O}(m \cdot d)$ extra parameters, fatal for $d \sim 10^{10}$ as in modern LLMs. We argue that the missing ingredient is not parallelism per se but a *memory-tractable Quasi-Newton preconditioner* for the trajectory operator.

Our framework. EIPO follows this thesis. We introduce a Preconditioned Nonlinear-Equations (PNE) form, $\mathcal{Z} \cdot \mathcal{H} = 0$, with $\mathcal{Z}$ a non-singular preconditioner that is allowed to depend on the parallel iteration index k. Two preconditioners are studied. The trivial choice $\mathcal{Z} = -I$ recovers a Picard-style fixed-point iteration [6, 67, 71, 97, 43] and is included as a Quasi-Newton baseline. Our central contribution is Memory-Efficient Adaptive Anderson Acceleration: instead of storing the q-deep residual buffer required by standard AA, we extract the secant geometry directly from the *momentum states already maintained inside the optimizer*. We prove that the resulting preconditioner satisfies the inverse multisecant condition with the smallest Frobenius perturbation from $-I$, and we apply a spectral safeguard [78, 91] to obtain global contraction.

Why this differs from existing "parallel GD". Existing parallel paradigms—data parallelism [98, 59, 96, 48], model parallelism [68, 58, 95], pipeline parallelism [29, 49]—all focus on partitioning data, parameters, or layers within a single GD update step. These methods provide only intra-step parallel speedups. In contrast, EIPO enables the concurrent execution of multiple successive GD steps, achieving inter-step parallelism, which is algorithmically distinct. Consequently, EIPO is fully orthogonal to and can be seamlessly combined with all existing intra-step parallel frameworks.

Our contributions are as follows.

1. We recast modern optimizer trajectories as a Preconditioned Nonlinear-Equations system and prove that its unique root recovers the exact optimizer trajectory.
2. We design Memory-Efficient Adaptive Anderson Acceleration, a Quasi-Newton preconditioner that uses optimizer momentum as a virtual spatial secant, eliminating the $\mathcal{O}(m \cdot d)$ memory overhead of classical AA.
3. We propose EIPO, an inter-step parallel training method designed to accelerate a wide range of GD-based optimizers (e.g., Adam). We prove that EIPO converges to the optimizer's trajectory within fewer iterations than its serial counterpart.
4. Across LLMs up to 6B parameters (GPT-2, Llama-3.2-1B [47], Qwen-1.5-4B/2.5-3B [5, 76], Gemma-2B [75], GPT-J-6B [81]), CIFAR-10 classification (ResNet50 [26], ViT [82]), and LSUN-Church [88] diffusion training (UNet [63], DDPM [28], DDIM [70], FID [27]), EIPO reduces iteration counts by up to $21\times$ and wall-clock time by up to $4.6\times$ at comparable model quality, per-GPU memory, and total token consumption.

2 Related Work

Anderson Acceleration and Quasi-Newton Solvers. EIPO inherits its algorithmic backbone from the Quasi-Newton family. Anderson Acceleration [2, 80, 56] constructs a multisecant approximation of the inverse Jacobian from a depth-q buffer of past residual variations. Convergence and stabilization analyses [78, 91, 20] show that AA achieves local superlinear behavior whenever the secant matrix is well-conditioned, but the explicit history buffer makes naive AA prohibitive for billion-parameter models. EIPO provides the first Quasi-Newton preconditioner that obtains the inverse multisecant geometry from *intrinsic* optimizer state rather than from an explicit buffer.

Time-Parallel Numerical Integration. The idea that long sequential time-evolutions can be solved by global root-finding has a rich history in scientific computing. Parareal [38], ParaExp [22] and multilevel time-parallel schemes [19] expose concurrency along the time axis of differential equations using coarse-fine corrector pairs. **EIPO** is conceptually related to these methods, but operates on ML training with discrete optimization map F_{t-1} rather than an ODE flow solving, and its corrector is an Memory-efficient Anderson preconditioner rather than a coarse-grid propagator. Besides, this class of coarse-fine corrector is not practical for ML training as their single iteration requires multiple sequential gradient evaluations for the coarse solver, followed by an parallel gradient computation for the fine solver. This makes its per-iteration cost several times higher than the baseline.

Intra-Step Parallelism. Modern parallel training paradigms primarily optimize the computation within individual GD steps by partitioning workloads across various dimensions. These techniques can be broadly categorized based on the target of parallelization: (i) **Data parallelism** distributes training samples across multiple workers to compute local gradients concurrently [48, 98, 15, 14, 87, 45, 59, 92, 21, 7, 8, 12, 60, 72, 52, 18, 96, 17]; (ii) **Model parallelism** shards network parameters across devices to overcome memory constraints and facilitate large-scale model training [31, 50, 68, 58, 86, 89, 61, 85, 23, 13, 11, 84, 66, 95]; and (iii) **Pipeline parallelism** decomposes the model into sequential stages and interleaves micro-batches to reduce worker idle time [29, 49, 25, 33, 37, 73, 94, 74, 4, 3, 40, 1]. While these methods effectively leverage collective communication [53] and partitioned tensors to improve per-step throughput, they strictly adhere to the autoregressive Markov chain of optimization, where each global parameter update is constrained by the completion of prior gradient computations. In contrast, **EIPO** is orthogonal: it disrupts this dependency chain by introducing *Inter-step parallelism*, enabling the simultaneous execution of multiple distinct GD steps.

Inter-Step Parallelism. To date, research on inter-step parallelism in the field of model training remains much sparse. The most closely related work is PASO [44], which recasts sequential GD as a system of triangular nonlinear equations (TNEs) and solves them via standard fixed-point iteration. PASO is a special case of our PNE formulation, corresponding to identity preconditioning; **EIPO** replaces this identity step with a memory-efficient Anderson multisecant preconditioner built from the optimizer's intrinsic momentum states, consistently outperforming the identity-preconditioned variant in both iteration count and wall-clock time (Tabs. 3, 4). OptEx [69] explores GD-trajectory speculation using kernelized future-gradient predictors; however, its predictive accuracy degrades sharply in high-dimensional spaces, severely hindering its scalability to modern large-scale models. Furthermore, OptEx relies on kernel estimation methods that necessitate storing extensive gradient histories, creating significant memory bottlenecks that **EIPO**'s memory-efficient design explicitly avoids. App. A provides theoretical and experimental evidence that **EIPO** significantly outperforms OptEx.

3 Preliminary

Sequential Autoregressive GD Procedure. Standard neural-network training applies an optimizer-specific update rule F_{t-1} left-to-right starting from w_0. The trajectory $\{w_t\}_{t=1}^{T}$ obeys

$$w_t = F_{t-1}(w_{t-1}, \dots, w_0; \zeta_{t-1}, \dots, \zeta_0) \triangleq w_0 - \sum_{\tau=0}^{t-1} \eta_\tau \, g_\tau\big(w_\tau, \dots, w_{\tau-r+1}; \zeta_\tau, \dots, \zeta_{\tau-r+1}\big), \quad (1)$$

with learning rate η_τ, mini-batch ζ_τ, and an optimizer-dependent gradient term g_τ whose memory depth r varies with the optimizer. For SGD ($r = 1$) and Adam ($r = \tau + 1$):

$$g_\tau^{\text{SGD}}(w_\tau, \zeta_\tau) = \nabla_{w_\tau} \mathcal{L}(w_\tau, \zeta_\tau), \quad g_\tau^{\text{Adam}} = \frac{\hat{m}_\tau}{\sqrt{\hat{v}_\tau} + \epsilon}, \quad (2)$$

where $\hat{m}_\tau, \hat{v}_\tau$ are the bias-corrected first and second moments. We treat Eq. (1) as the *specification* of the target trajectory, not as the algorithm we run.

4 Proposed Method

4.1 The PNE Reformulation: GD as a Preconditioned Root-finding Problem

The bottleneck of Eq. (1) is structural rather than computational: each w_t is presented as a function of all earlier w_τ. We adopt the natural numerical-analysis perspective: declare every entry of the

trajectory as an independent unknown $\hat{w}_t$ and treat their joint specification as a system of nonlinear equations. Following classical Quasi-Newton theory [35, 32, 2], we reformulate the GD process into a system of **Preconditioned Nonlinear Equations (PNEs)** $\mathcal{Z} \cdot \mathcal{H}(\hat{w}_0, \ldots, \hat{w}_T) = 0$:

$$\mathcal{H}(\hat{w}_0, \ldots, \hat{w}_T) = \begin{cases} \hat{w}_0 - w_0 = 0, \\ \hat{w}_t - F_{t-1}(\hat{w}_{t-1}, \ldots, \hat{w}_0; \zeta_{t-1}, \ldots, \zeta_0) = 0, \quad t \in [1, T]. \end{cases} \tag{3}$$

where $F_{t-1}(\cdot)$ is the mapping defined in Eq. (1), and $\mathcal{Z}$ are non-singular (invertible) preconditioning matrix. The matrix $\mathcal{Z}$ should ideally approximate the inverse Jacobian of $\mathcal{H}$, in which case Eq. (3) is a single Newton step on the trajectory operator and converges at a Newton-like rate [35, 32].

Theorem 4.1 (Unique-root recovery; proof in *App. G.0.1*)**.** *For any non-singular $\mathcal{Z}$, fixed initial weights w_0, learning-rate schedule $\{\eta_t\}_{t=0}^{T}$, and data sequence $\{\zeta_t\}_{t=0}^{T}$, the system in Eq. (3) admits a unique root, which equals the sequential trajectory $\{w_t\}_{t=0}^{T}$ defined by Eq. (1).*

Remark 4.2. *The theorem shows that finding any root of the preconditioned system reproduces the optimizer's exact behavior. It also liberates the algorithm designer from the constraint of computing $\hat{w}_t$ in any particular order: any root-finding scheme is mathematically equivalent to running the optimizer.*

4.2 Solving the PNE System via Block-Krylov Iteration

Notation. For $0 \le t_1 \le t_2 \le T$ and any sequence of vectors or matrices $v_0, \ldots, v_T$, write $v_{t_1:t_2} = [v_{t_1}^\top, \ldots, v_{t_2}^\top]^\top$. Let $R_t^{(k)} = F_{t-1}(\hat{w}_0^{(k)}, \ldots, \hat{w}_{t-1}^{(k)}) - \hat{w}_t^{(k)}$ be the local residual at trajectory index t and iteration k.

We solve Eq. (3) via a preconditioned block iteration on the concatenated trajectory vector $\hat{w}_{0:T} = [\hat{w}_0^\top, \ldots, \hat{w}_T^\top]^\top$. Using the global residual $R(\hat{w}_{0:T}^{(k)}) = R_{0:T}^{(k)}$, the canonical Quasi-Newton step reads

$$\hat{w}_{0:T}^{(k+1)} = \hat{w}_{0:T}^{(k)} - \mathcal{Z}^{(k)} \cdot R_{0:T}^{(k)}, \tag{4}$$

with $\mathcal{Z}^{(k)}$ chosen as a Quasi-Newton approximation of the inverse Jacobian of $\mathcal{H}$.

Sliding-window block restart. For long horizons T, materializing the global Jacobian or evaluating the residual on the full trajectory exceeds GPU memory. We instead use a Krylov-style *block restart* [67, 43]: iterate Eq. (4) on a local block of unknowns $\hat{w}_{t_1:t_2}^{(k)}$ with $t_2 - t_1 + 1 \le p$, advance the block forward once the leading variables have stabilized, and re-warmstart the next block from the most recent estimate. This is structurally analogous to GMRES restart [32] applied to the trajectory operator and is denoted hereafter as the **EIPO**'s p-size sliding window.

Stopping criterion. Block stability is checked through the per-step normalized squared change. Following existing practices [67, 97, 43], we define the stopping criterion as:

$$d(\hat{w}_t^{(k+1)}, \hat{w}_t^{(k)}) := \tfrac{1}{n} \left\| \hat{w}_t^{(k+1)} - \hat{w}_t^{(k)} \right\|^2 \le \delta, \tag{5}$$

where $\|\cdot\|$ is the Frobenius norm and n the model dimension. The tolerance δ is tracked by an exponential moving average over the within-window changes:

$$\delta = \lambda\delta + (1-\lambda) \cdot \mathcal{A}\Big(\{d(\hat{w}_{t+i}^{(k)}, \hat{w}_{t+i}^{(k-1)}) | i = 1, \cdots, p\}\Big), \tag{6}$$

with EMA decay λ and aggregator $\mathcal{A} \in \{\text{mean}, \text{median}\}$. The complete **EIPO** is summarized in Alg. 1. Fig. 1 shows the core idea of **EIPO**.

We discuss two preconditioners for Alg. 1. **(1) Identity Preconditioner ($\mathcal{Z}^{(k)} = -I$).** Setting the preconditioner to the negative identity reduces Eq. (4) to $\hat{w}_{t_1:t_2}^{(k+1)} = \hat{w}_{t_1:t_2}^{(k)} - R(\hat{w}_{t_1:t_2}^{(k)})$. This is Picard iteration [6, 44] on $\mathcal{H}$. It ignores the secant geometry of the operator, and its rate degrades sharply with the depth of the autoregressive chain. **(2) Quasi-Newton Preconditioner.** Based on the standard Anderson Acceleration, we further construct a Quasi-Newton preconditioner $\mathcal{Z}^{(k)}$ as a memory-tractable adaptive multisecant approximation of the inverse Jacobian. Section 4.2.1 derives this preconditioner from optimizer momentum.

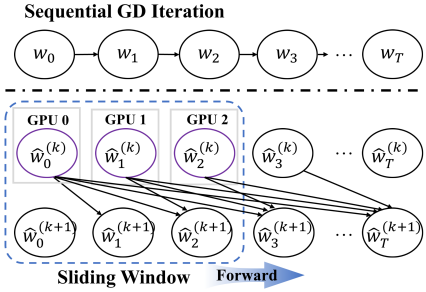


Figure 1: Illustration of **EIPO** that breaks sequential dependencies across GD steps, allowing multiple model weights to be updated in parallel.

Algorithm 1 EIPO: Inter-step Parallel Training with Sliding-window Restart

Require: Optimizer (SGD/Adam/AdamW), initial weight w_0, total gradient steps T, window size p, learning rate $\{\eta_t\}_{t=0}^{T-1}$, mini-batches $\{\zeta_t\}_{t=0}^{T-1}$, tol. δ, EMA λ, damping factor γ, max iters K.

Initialize $\hat{w}_t^{(0)} = w_0$ for all $t \in \{0, \dots, p\}$ ▷ Warm-start the first block
$t_1 \leftarrow 0, \quad t_2 \leftarrow p, \quad k \leftarrow 0$
for $t_1 < T$ **do**
  Compute residuals $R_t^{(k)} = F_{t-1}(\hat{w}_0^{(k)}, \dots, \hat{w}_{t-1}^{(k)}) - \hat{w}_t^{(k)}$ for $t \in [t_1, t_2]$ in parallel.
  Preconditioned step: $\hat{w}_{t_1:t_2}^{(k+1)} \leftarrow \hat{w}_{t_1:t_2}^{(k)} - \mathcal{Z} R_{t_1:t_2}^{(k)}$ with $\mathcal{Z} = -I$ *or* $\mathcal{Z} = \bar{J}_{t_1:t_2}^{(k)}$ from Eq. (16).
  Compute convergence distance $d_t = \frac{1}{n}\|\hat{w}_t^{(k+1)} - \hat{w}_t^{(k)}\|_F^2, \forall t \in \{t_1, \dots, t_2\}$.
  $t_1^{new} \leftarrow \min\Big\{\{t+1 \mid d_t > \delta, t_1 \le t \le t_2\} \cup \{t_2+1\}\Big\}$.
  $s \leftarrow t_1^{new} - t_1$ ▷ Determine sliding stride
  if $s > 0$ **then** initialize $\hat{w}_{t_2+1:t_2+s}^{(k+1)} \leftarrow \hat{w}_{t_2}^{(k+1)}$.
  Update $k \leftarrow k+1$, $t_2 \leftarrow \min(t_2 + s, T-1)$, and $t_1 \leftarrow t_1^{new}$.
  Refresh δ via Eq. (6).
end for
return $\hat{w}_T^{(K)}$

4.2.1 Memory-Efficient Adaptive Anderson Acceleration

We start from a depth-q Anderson preconditioner [2, 80] for the block iteration in Eq. (4). Given an initial trajectory estimate $\hat{w}_{0:T}^{(0)}$ and any $k \ge 1$, classical AA constructs a multisecant inverse-Jacobian model from buffers of q previous variations.

Notation. Let $q_k = \min\{q, k\}$, $\Delta\hat{w}_t^{(k)} = \hat{w}_t^{(k+1)} - \hat{w}_t^{(k)}$, $\mathcal{X}_t^{(k)} = \Big[\Delta\hat{w}_t^{(k-q_k)}, \dots, \Delta\hat{w}_t^{(k-1)}\Big]$, $\Delta R_t^{(k)} = R_t^{(k+1)} - R_t^{(k)}$, and $\mathcal{F}_t^{(k)} = \Big[\Delta R_t^{(k-q_k)}, \dots, \Delta R_t^{(k-1)}\Big]$.

Standard Anderson Acceleration. For the active block $t = t_1, \dots, t_2$, assuming $\mathcal{F}_{t_1:t_2}^{(k)\top}\mathcal{F}_{t_1:t_2}^{(k)}$ has full rank, classical AA assigns the block of the preconditioner $\mathcal{Z}^{(k)}$ as $G^{(k)}$, with update rule

$$\hat{w}_{t_1:t_2}^{(k+1)} = \hat{w}_{t_1:t_2}^{(k)} - G^{(k)} R_{t_1:t_2}^{(k)}, \tag{7}$$

where the closed-form expression of $G^{(k)}$ is

$$G^{(k)} = -I + \Big(\mathcal{X}_{t_1:t_2}^{(k)} + \mathcal{F}_{t_1:t_2}^{(k)}\Big)\Big(\mathcal{F}_{t_1:t_2}^{(k)\top}\mathcal{F}_{t_1:t_2}^{(k)}\Big)^{-1}\mathcal{F}_{t_1:t_2}^{(k)\top}. \tag{8}$$

The matrix $G^{(k)}$ is justified by the inverse multisecant condition [20]

$$G^{(k)}\mathcal{F}_{t_1:t_2}^{(k)} = \mathcal{X}_{t_1:t_2}^{(k)}, \tag{9}$$

together with the property that the Frobenius norm $\|G^{(k)} + I\|$ is the smallest among all matrices satisfying the condition [80].

The memory bottleneck. Evaluating $G^{(k)}$ requires the full history buffers $\mathcal{X}_{t_1:t_2}^{(k)}, \mathcal{F}_{t_1:t_2}^{(k)}$, costing $\mathcal{O}(m \cdot d)$ extra memory. For LLMs with $d \sim 10^{10}$, even $m = 4$ doubles the optimizer state and is therefore intractable.

Optimizer momentum as a virtual spatial secant. Our key observation is that the momentum buffer of every modern optimizer is, by construction, an exponential moving average of past gradients—a weighted sum of historical residual information. The information that classical AA stores in $\mathcal{F}_{t_1:t_2}^{(k)}$ is therefore *already encoded* in the optimizer's first-moment buffer, up to the chosen weighting. Exploiting this fact, we replace the explicit history buffer by a single *virtual spatial secant* anchored at the previous iterate:

$$\mathcal{F}_{q=1,t}^{(k)} \triangleq R_t^{(k)} - \mathbf{D}_t \in \mathbb{R}^{d\times 1}, \tag{10}$$

where $\mathbf{D}_t$ is an optimizer-specific historical-momentum proxy and $\mathbf{M}_t$ is the metric the optimizer effectively uses to measure displacement. The depth-1 buffer collapses the auxiliary memory to a single tensor (already maintained by the optimizer), eliminating the AA memory overhead entirely.

Optimizer-specific instantiations. We provide the proxies for the three most widely-used optimizers:

- **SGD with Momentum $\bar{m}_t$.** Velocity tracks the smoothed gradient; the underlying space is Euclidean.

$$\mathbf{D}_t = -\eta \bar{m}_t, \quad \mathbf{M}_t = I. \tag{11}$$

- **Adam.** The optimizer maintains EMAs $\hat{m}_t$ and $\hat{v}_t$, and effectively uses a second-moment metric.

$$\mathbf{D}_t = -\eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}, \quad \mathbf{M}_t = \mathrm{diag}(\sqrt{\hat{v}_t} + \epsilon). \tag{12}$$

- **AdamW.** Decoupled weight decay λ contributes a deterministic displacement, which the proxy is.

$$\mathbf{D}_t = -\eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \eta\lambda \hat{w}_t^{(k)}, \quad \mathbf{M}_t = \mathrm{diag}(\sqrt{\hat{v}_t} + \epsilon). \tag{13}$$

Theorem 4.3 (Optimal multisecant preconditioner; proof in *App. G.1*)**.** *Assume that* $\mathcal{F}_{q=1,t_1:t_2}^{(k)\top} \mathbf{M}_{t_1:t_2}^{-1} \mathcal{F}_{q=1,t_1:t_2}^{(k)}$ *has full rank, and define*

$$Q_{t_1:t_2}^{(k)} = (\mathcal{X}_{q=1,t_1:t_2}^{(k)} + \mathcal{F}_{q=1,t_1:t_2}^{(k)}) \left(\mathcal{F}_{q=1,t_1:t_2}^{(k)\top} \mathbf{M}_{t_1:t_2}^{-1} \mathcal{F}_{q=1,t_1:t_2}^{(k)} \right)^{-1} \mathcal{F}_{q=1,t_1:t_2}^{(k)\top} \mathbf{M}_{t_1:t_2}^{-1}. \tag{14}$$

Then $J_{t_1:t_2}^{(k)} = -I + Q_{t_1:t_2}^{(k)}$ *satisfies the inverse multisecant condition Eq. (9), and* $\|J_{t_1:t_2}^{(k)} + I\|_{\mathbf{M}_{t_1:t_2}^{-1}}$ *is minimal among all matrices satisfying that condition.*

Remark 4.4. *The optimal preconditioner exactly cancels local error along the secant direction. As the iterate approaches the trajectory's true root, the secant approximation tightens and the residual along the secant direction is suppressed superlinearly. This local Newton-like behavior is structurally inaccessible to identity preconditioning.*

Spectral safeguard. Following standard global-stabilization techniques for AA [78, 91], we apply a damping parameter $\gamma \in (0, 1)$ and a spectral clip:

$$\bar{Q}_{t_1:t_2}^{(k)} = \gamma \cdot Q_{t_1:t_2}^{(k)} / \max\left(1, \ \|Q_{t_1:t_2}^{(k)}\|_2\right). \tag{15}$$

By construction $\|\bar{Q}_{t_1:t_2}^{(k)}\|_2 \leq \gamma < 1$: when $\|Q\|_2 < 1$ the clip is inactive and $\bar{Q} = \gamma Q$; otherwise we shrink the step length while preserving the Quasi-Newton direction. The safeguarded preconditioner reads

$$\bar{J}_{t_1:t_2}^{(k)} = -I + \bar{Q}_{t_1:t_2}^{(k)}. \tag{16}$$

Smaller γ trades acceleration for stability; $\gamma \to 0$ continuously recovers identity preconditioning, which yields a single, unified analysis below.

4.3 Convergence Analysis

We provide convergence guarantees for Algorithm 1 under $\mathcal{Z}^{(k)} = -I$ and $\mathcal{Z}^{(k)} = \bar{J}_{t_1:t_2}^{(k)}$. The two cases are unified by the damping parameter $\gamma \in [0, 1)$; setting $\gamma = 0$ recovers identity preconditioning.

Notation. The p-size sliding window partitions the T-step trajectory into m consecutive blocks $\mathcal{B}_1, \ldots, \mathcal{B}_m$ with $|\mathcal{B}_i| \leq p$. Let $W_{\mathcal{B}_i}^{(j)}$ be the block state at iteration j, $W_{\mathcal{B}_i}^{\mathrm{seq}}$ the corresponding sequential GD target, and H_{i-1} the converged-history prefix supplying inputs to block i.

Assumption 4.5 (Lipschitz Gradient)**.** *There exists* $C > 0$ *such that* $\|g_\tau(X) - g_\tau(Y)\|_F \leq C\|X - Y\|_F$ *for all* τ *and any inputs* X, Y.

Assumption 4.6 (Bounded Initial Drift)**.** $\max_i \|W_{\mathcal{B}_i}^{(0)} - W_{\mathcal{B}_i}^{\mathrm{seq}}\|_F \leq D$ *for some constant* $D > 0$.

Theorem 4.7 (Unified convergence; proof in App. G.2)**.** *Algorithm 1 converges* exactly *within $K \le T$ iterations. For $K < T$, under Assumptions 4.5–4.6, define $\rho_1 = (1+\gamma)\sqrt{p} \cdot \max_i \sum_{\tau \in \mathcal{B}_{i-1}} \eta_\tau C + \gamma$ and $\rho_2 = (1+\gamma)\sqrt{p} \cdot \max_i \sum_{\tau \in H_{i-1}} \eta_\tau C$. If the schedule satisfies $\rho_1 < 1$ (i.e., $\max_i \sum_{\tau \in \mathcal{B}_{i-1}} \eta_\tau < \frac{1-\gamma}{(1+\gamma)\sqrt{p}C}$), then after K total parallel iterations (K/m updates per block), the convergence error admits an upper bound exhibiting exponential decay w.r.t K/m:*

$$\|W_{\mathcal{B}_m}^{(K/m)} - W_{\mathcal{B}_m}^{\text{seq}}\|_F \le D\,\rho_1^{K/m}\left(1 + \frac{\rho_2(1-\rho_1^{K/m})}{1-\rho_1}\right)^{m-1}. \tag{17}$$

The method achieves ϵ-precision with $K < T$ whenever $\rho_1^{T/m}\left(1 + \frac{\rho_2(1-\rho_1^{T/m})}{1-\rho_1}\right)^{m-1} < \epsilon/D$.

Corollary 4.8 (Uniform Contraction)**.** *If $\rho_1 \le \rho$ and $\rho_2 \le \rho$ for some $\rho \in (0,1)$, the bound simplifies to $\|W_{\mathcal{B}_m}^{(K/m)} - W_{\mathcal{B}_m}^{\text{seq}}\|_F \le D\,\rho^{K/m}\left(\frac{1-\rho^{K/m+1}}{1-\rho}\right)^{m-1}$.*

Remark 4.9. *Modern training recipes routinely satisfy the rate condition: (i) decaying schedules give per-step learning rates of 0 to 10^{-3}, so $\sum \eta_\tau$ remains tiny; (ii) adaptive methods like Adam normalize updates to near-unit scale, keeping C moderate; (iii) The factor $\sqrt{p}$ arises from a worst-case analysis; in practice, it is typically tighter than the theoretical bound suggests. We empirically observe robust convergence across all experiments using standard learning rate configurations ($[10^{-5}, 10^{-3}]$)*

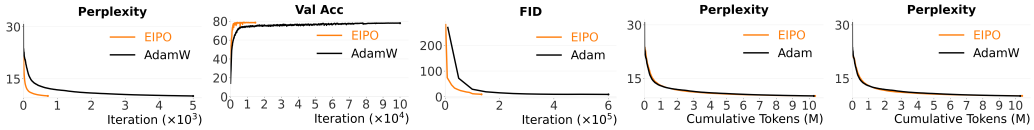
5 Experiments

5.1 Experiment Settings

Datasets and Models. For **language modeling** (App. B), we train GPT-2 [57], Llama3.2-1B [47], Qwen1.5-4B [5], Qwen2.5-3B [76], Gemma-2B [75], and GPT-J-6B [81] on the WikiText-2 dataset [46]. For **image classification** (App. C), we evaluate a CNN (App. C.2.1), ResNet50 [26], and Vision Transformer (ViT) [82] on CIFAR-10 [36]. For the **diffusion task** (App. D), following DDPM [28], we train a UNet model at 128×128 resolution on the LSUN-Church dataset [88].

Metrics and Hyperparameters. Language modeling uses testing perplexity (PPL); image classification uses top-1 accuracy, precision, recall and F1; diffusion uses FID [27]. *Detailed settings for language modeling, image classification, and diffusion training are summarized in Tabs. 8, 11, and 12.* We use 8 A100 GPUs unless otherwise stated.

5.2 Experimental Results



(a) LLM task, Llama3.2-1B (b) Classification task, ViT (c) Diffusion task, UNet (d) Perplexity w.r.t. tokens, Adam (e) Perplexity w.r.t. tokens, AdamW

Figure 2: Comparison of perplexity, FID, and accuracy curves. Panels (d) and (e) plot perplexity against cumulative token consumption for Llama-3.2-1B on WikiText-2.

Language Modeling. Table 3 reports results at near-identical token budgets. For both models, EIPO compresses the optimizer iteration budget by $5.8 \sim 7\times$ across all optimizers, yielding up to $4.6\times$ wall-clock time speedups. The Quasi-Newton variant ($\mathcal{Z} = \bar{J}$) consistently outperforms the identity variant ($\mathcal{Z} = -I$), confirming the value of the multisecant preconditioner. Figure 2 shows the corresponding perplexity curves against both iterations and tokens.

Table 1: Comparison on diffusion training: UNet trained from scratch on LSUN-Church, FID computed over 50k generated images. Listing 10 and 11 provide reproduction scripts; Tab. 12 lists settings.

Method	Iters	Peak Mem.	FID	Time (days)	Speedup
Adam	600k	78.6G	9.18	7.39	1×
Adam+EIPO ($\mathcal{Z} = -I$)	136k	78.6G	9.54	2.46	3.0×
Adam+EIPO ($\mathcal{Z} = \bar{J}$)	**133k**	78.6G	9.58	**2.34**	**3.2**×

Table 3: Quantitative comparison on WikiText-2 with pre-trained initial weights. Tokens denote total tokens processed by the model and Mem. denotes peak per-GPU memory. Means and standard deviations are reported over 5 runs. Listings 1 and 2 provide reproduction scripts; Tab. 8 lists hyperparameters.

Method	GPT-2 ($B = 112, \eta = 6e-5$)						Llama-3.2-1B ($B = 30, \eta = 1e-5$)					
	Iters ↓	Mem. ↓	Tokens ↓	PPL ↓	Time ↓	Speedup ↑	Iters ↓	Mem. ↓	Tokens ↓	PPL ↓	Time ↓	Speedup ↑
SGD	5000	53.0GB	36.73M	$36.7_{\pm 0.14}$	3067s	1.0×	5000	62.9GB	10.28M	$13.6_{\pm 0.16}$	4061s	1.0×
SGD+EIPO ($\mathcal{Z} = -I$)	721 (6.9×)	52.2GB	37.07M	$36.9_{\pm 0.12}$	748s	4.1×	715 (7.0×)	65.3GB	10.28M	$13.6_{\pm 0.14}$	1392s	2.9×
SGD+EIPO ($\mathcal{Z} = \bar{J}$)	**715** (7.0×)	52.2GB	36.71M	$37.0_{\pm 0.20}$	**662s**	**4.6×**	**715** (7.0×)	65.3GB	10.26M	$13.7_{\pm 0.15}$	**1375s**	**3.0×**
Adam	5000	53.3GB	36.73M	$19.8_{\pm 0.21}$	3081s	1.0×	5000	65.3GB	10.28M	$10.6_{\pm 0.24}$	4162s	1.0×
Adam+EIPO ($\mathcal{Z} = -I$)	865 (5.8×)	52.2GB	44.44M	$19.7_{\pm 0.24}$	844s	3.7×	723 (6.9×)	65.3GB	10.39M	$10.7_{\pm 0.13}$	1611s	2.6×
Adam+EIPO ($\mathcal{Z} = \bar{J}$)	**778** (6.4×)	52.2GB	39.95M	$19.8_{\pm 0.19}$	**760s**	**4.1×**	**715** (7.0×)	65.3GB	10.29M	$10.8_{\pm 0.14}$	**1509s**	**2.8×**
AdamW	5000	53.3GB	36.73M	$19.9_{\pm 0.15}$	3074s	1.0×	5000	65.3GB	10.28M	$10.6_{\pm 0.22}$	4160s	1.0×
AdamW+EIPO ($\mathcal{Z} = -I$)	863 (5.8×)	52.2GB	44.44M	$19.7_{\pm 0.19}$	845s	3.6×	722 (6.9×)	65.3GB	10.43M	$10.6_{\pm 0.16}$	1654s	2.5×
AdamW+EIPO ($\mathcal{Z} = \bar{J}$)	**775** (6.5×)	52.2GB	39.84M	$19.8_{\pm 0.12}$	**768s**	**4.0×**	**715** (7.0×)	65.3GB	10.23M	$10.7_{\pm 0.13}$	**1608s**	**2.6×**

Table 4: Quantitative comparison on CIFAR-10. Mean and standard deviation reported over 5 runs with different random seeds. Models trained from scratch. Listings 8 and 9 provide reproduction scripts; Tab. 11 lists hyperparameters.

Method	ResNet50, $B = 2048$, $\eta = 5e-5$, cosine decay, warmup steps 10k, $p = 7$							
	Iters↓	Accuracy↑	F1-score↑	Precision↑	Recall↑	Peak memory ↓	Time (s) ↓	Speedup↑
SGD	240000	$83.8_{\pm 0.24}$	$83.8_{\pm 0.22}$	$83.9_{\pm 0.12}$	$83.8_{\pm 0.15}$	50550MB	152918.5	1.0×
SGD + EIPO ($\mathcal{Z} = -I$)	38398 (6.3×)	$83.8_{\pm 0.17}$	$83.8_{\pm 0.20}$	$83.8_{\pm 0.19}$	$83.8_{\pm 0.21}$	50918MB	45784.5	3.3×
SGD + EIPO ($\mathcal{Z} = \bar{J}$)	**36513** (6.6×)	$83.8_{\pm 0.29}$	$83.8_{\pm 0.18}$	$83.8_{\pm 0.21}$	$83.8_{\pm 0.25}$	51010MB	**44306.0**	**3.5×**
Adam	100000	$90.1_{\pm 0.11}$	$90.1_{\pm 0.15}$	$90.1_{\pm 0.12}$	$90.1_{\pm 0.13}$	50641MB	62316.0	1.0×
Adam + EIPO ($\mathcal{Z} = -I$)	14914 (6.7×)	$89.5_{\pm 0.25}$	$89.5_{\pm 0.22}$	$89.5_{\pm 0.24}$	$89.5_{\pm 0.23}$	50641MB	17771.6	3.5×
Adam + EIPO ($\mathcal{Z} = \bar{J}$)	**14730** (6.8×)	$89.6_{\pm 0.24}$	$89.6_{\pm 0.21}$	$89.6_{\pm 0.23}$	$89.6_{\pm 0.26}$	50918MB	**17324.2**	**3.6×**
AdamW	100000	$90.3_{\pm 0.17}$	$90.3_{\pm 0.16}$	$90.3_{\pm 0.18}$	$90.3_{\pm 0.17}$	50641MB	62201.0	1.0×
AdamW + EIPO ($\mathcal{Z} = -I$)	14921 (6.7×)	$89.8_{\pm 0.19}$	$89.8_{\pm 0.21}$	$89.8_{\pm 0.18}$	$89.8_{\pm 0.20}$	50641MB	17875.9	3.5×
AdamW + EIPO ($\mathcal{Z} = \bar{J}$)	**14717** (6.8×)	$89.6_{\pm 0.20}$	$89.5_{\pm 0.18}$	$89.5_{\pm 0.19}$	$89.6_{\pm 0.22}$	50737MB	**17521.4**	**3.6×**
Method	ViT, $B = 2048$, $\eta = 1.5e-4$, cosine decay, warmup steps 10k, $p = 7$							
	Iters↓	Accuracy↑	F1-score↑	Precision↑	Recall↑	Peak memory↓	Time (s) ↓	Speedup↑
SGD	240000	$71.8_{\pm 0.21}$	$71.8_{\pm 0.16}$	$71.8_{\pm 0.23}$	$71.8_{\pm 0.20}$	17962MB	145486.7	1.0×
SGD + EIPO ($\mathcal{Z} = -I$)	43111 (5.6×)	$71.4_{\pm 0.19}$	$71.3_{\pm 0.20}$	$71.2_{\pm 0.23}$	$71.4_{\pm 0.22}$	18144MB	40908.6	3.6×
SGD + EIPO ($\mathcal{Z} = \bar{J}$)	**40892** (5.9×)	$71.4_{\pm 0.24}$	$71.4_{\pm 0.21}$	$71.4_{\pm 0.19}$	$71.4_{\pm 0.20}$	18108MB	**38725.9**	**3.8×**
Adam	100000	$77.7_{\pm 0.15}$	$77.5_{\pm 0.12}$	$77.5_{\pm 0.14}$	$77.7_{\pm 0.13}$	17999MB	59709.2	1.0×
Adam + EIPO ($\mathcal{Z} = -I$)	14999 (6.7×)	$77.9_{\pm 0.20}$	$77.8_{\pm 0.18}$	$77.8_{\pm 0.19}$	$77.9_{\pm 0.21}$	17999MB	13958.1	4.3×
Adam + EIPO ($\mathcal{Z} = \bar{J}$)	**14781** (6.8×)	$77.3_{\pm 0.19}$	$77.2_{\pm 0.22}$	$77.2_{\pm 0.20}$	$77.3_{\pm 0.18}$	18144MB	**13563.6**	**4.4×**
AdamW	100000	$78.0_{\pm 0.10}$	$77.9_{\pm 0.11}$	$77.9_{\pm 0.10}$	$78.0_{\pm 0.10}$	17999MB	59876.7	1.0×
AdamW + EIPO ($\mathcal{Z} = -I$)	15019 (6.7×)	$78.4_{\pm 0.14}$	$78.3_{\pm 0.13}$	$78.2_{\pm 0.15}$	$78.4_{\pm 0.14}$	17999MB	13830.9	4.2×
AdamW + EIPO ($\mathcal{Z} = \bar{J}$)	**14694** (6.8×)	$78.6_{\pm 0.13}$	$78.5_{\pm 0.16}$	$78.4_{\pm 0.12}$	$78.6_{\pm 0.15}$	18144MB	**13753.3**	**4.4×**

Image Classification. Table 4 compares ResNet50 and ViT trained from scratch on CIFAR-10. Iteration counts are cut by 6.7–6.9× and wall-clock by 3.5–4.5× across the three optimizers, while peak per-GPU memory remains essentially unchanged. The Quasi-Newton variant routinely matches or exceeds the identity variant on accuracy and time. Figure 2 shows the accuracy curves.

Image Diffusion Training. Table 1 reports diffusion training on LSUN-Church. The Quasi-Newton EIPO variant reaches comparable FID 9.5 in only 133k iterations (2.34 days), versus 600k iterations (7.39 days) for the Adam baseline at FID 9.1. Peak memory is identical at 78.6 GB, confirming that the Quasi-Newton corrector imposes no additional GPU footprint. Figs. 2 and 3 display the FID curve and visual comparisons respectively.

Table 2: Inter-step vs. data parallelism (DP) on GPT-2 / WikiText-2. Hyperparameters in Tab. 13 (App. E); execution scripts in Listing 12.

Method	**Iters**	**Tokens**	**PPL**	**Time (s)**	**Speedup**
Adam	1k	7344k	20.4	575	1.0×
Ours (Academic Prototype) [a]	**48** (21×)	7342k	20.7	**126**	**4.6×**
DP (Academic Prototype) [a]	1k (1×)	7349k	20.5	361	1.6×
DP (PyTorch DDP) [b]	1k (1×)	7349k	20.6	133	4.3×
DP (DeepSpeed) [b]	1k (1×)	7349k	19.1	131	4.4×

[a] Identical academic communication kernels; [b] Industry-grade kernels.

Inter-step Parallelism vs. Intra-step Data Parallelism. Table 2 compares EIPO against PyTorch `DistributedDataParallel` (DDP) and DeepSpeed-ZeRO-3, plus an academic DP prototype that uses the same communication kernels as our research code for an apples-to-apples comparison. EIPO delivers a 2.9× wall-clock advantage over the academic DP prototype while consuming the fewest tokens, and even the unoptimized prototype

Table 5: Comparison on large-scale LLMs (WikiText-2, pre-trained init, 8× H20 GPUs). Baseline uses AdamW for Gemma and Qwen; for GPT-J-6B memory limits force SGD with $p = 2$ on 2 GPUs, capping the speedup. Listing 3 provide reproduction scripts; Tab. 9 lists hyperparameters.

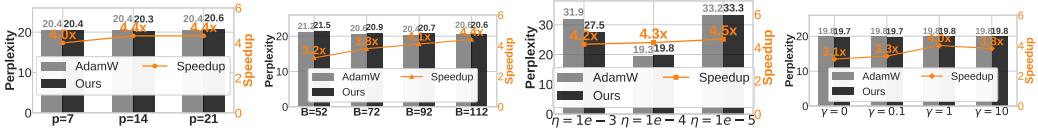
Method	Gemma-2B [75]				Qwen1.5-4B [5]				Qwen2.5-3B [76]				GPT-J-6B [81]			
	Tokens↓	PPL↓	Time↓	Speedup↑	Tokens↓	PPL↓	Time↓	Speedup↑	Tokens↓	PPL↓	Time↓	Speedup↑	Tokens↓	PPL↓	Time↓	Speedup↑
Baseline	1587k	$19.9_{\pm 0.12}$	1696s	1.0×	1363k	$10.5_{\pm 0.19}$	2252s	1.0×	1653k	$10.2_{\pm 0.16}$	2182s	1.0×	1558k	$23.5_{\pm 0.15}$	3794s	1.0×
Ours	1597k	$19.5_{\pm 0.18}$	**375s**	**4.5×**	1393k	$10.6_{\pm 0.17}$	**620s**	**3.6×**	1660k	$10.3_{\pm 0.15}$	**541s**	**4.0×**	1579k	$23.5_{\pm 0.11}$	**2146s**	**1.8×**

outperforms heavily-engineered industrial DP frameworks. Critically, **EIPO**'s 21× reduction in iteration count is a quantity that intra-step parallelism cannot in principle reach: DP merely speeds up the computation of a single GD step. **EIPO** is therefore complementary to intra-step parallelism.

Large-scale Models. Table 5 extends **EIPO** to 2B–6B model regimes. Across Gemma-2B, Qwen-1.5-4B, Qwen-2.5-3B, and GPT-J-6B, **EIPO** yields wall-clock speedups of 1.8×–4.5× at matched or improved perplexity. The 1.8× speedup on GPT-J-6B is a hardware artifact: our limited GPU VRAM forced us to use a window size of 2 on SGD only, where the maximum theoretical speedup is inherently capped at 2.



Figure 3: Visual comparison for diffusion models. See Figs. 6 and 7 in App. for more comparisons.



(a) (a) Impact of window size p *(b)* (b) Impact of batch size B *(c)* (c) Impact of learning rate η *(d)* (d) Impact of damping factor γ

Figure 4: Effect of window size p, batch size B, and learning rate η. Listings 4, 5. 6, and 7 provides reproduction scripts, respectively; Tab. 8 lists hyperparameters.

Impact of the Window Size p, Batch Size B, Learning Rate η, and Damping Factor γ. As illustrated in Fig. 4 (a), the speedup improves as p increases from 7 to 14, but plateaus at $p = 21$. This stagnation stems from the increased FLOPs required by larger window sizes, which, given our 8-GPU limit, diminishes the marginal benefits of parallelization and results in a lower speedup. Fig. 4 (b) illustrates that our method achieves greater speedup as the batch size increases, without compromising performance relative to the baseline. This is attributed to the enhanced utilization of GPU resources for large-scale matrix computations, which leads to superior parallel efficiency. Figure 4 (c) illustrates that our method performs robustly across a practical range of learning rates from 1×10^{-5} to 1×10^{-3}. Finally, Fig. 4 (d) demonstrates the influence of γ. We observe that the speedup consistently improves as γ increases. Even when γ exceeds the theoretical requirement of $\gamma < 1$ (e.g., reaching $\gamma = 10$), the model quality continues to match the baseline performance. This suggests that our approach is insensitive to the specific choice of γ and exhibits strong robustness in practical applications.

Sensitivity to Tolerance δ and EMA Decay λ. Fig. 5 highlights two findings: (i) **EIPO** is robust across $\delta \in [10^{-7}, 0.1]$ and $\lambda \in [0.9, 0.9999]$, with substantial iteration savings retained throughout the

grid; and (ii) the gain becomes more pronounced as $\lambda \to 1$, consistent with the secant approximation tightening when the EMA observes more local geometry.

5.3 Limitations and Discussion

EIPO closes a $3\times$–$4.6\times$ wall-clock gap against GD but still trails its 6–$21\times$ *algorithmic* speedup in iteration count. The discrepancy is dominated by the engineering artifacts of our academic prototype: the inter-GPU communication is funneled through a Python multiprocessing queue with a CPU intermediary. None of these artifacts are intrinsic to the algorithm. We expect concrete additional gains from (i) higher-depth Anderson buffers fused with optimizer state [2, 80, 56], (ii) production-grade comm. kernels (NCCL collectives, ZeRO-style sharding [58]), and (iii) composition with intra-step parallel method. Together these would push **EIPO** toward the speedup upper bound.

6 Conclusion

We formulate the GD trajectory as the unique zero of a piecewise operator, recast it as a PNE system, and solve it with a Quasi-Newton corrector built from the optimizer's own momentum buffers. The resulting Memory-Efficient Adaptive Anderson Acceleration is, to our knowledge, the first practical inverse-multisecant preconditioner that requires no auxiliary history beyond what optimizers already maintain. Theory establishes fewer-step exact recovery and exponential contraction; experiments across LLMs, vision models, and diffusion models confirm large iteration- and wall-clock-savings under matched per-GPU memory and total token consumption.

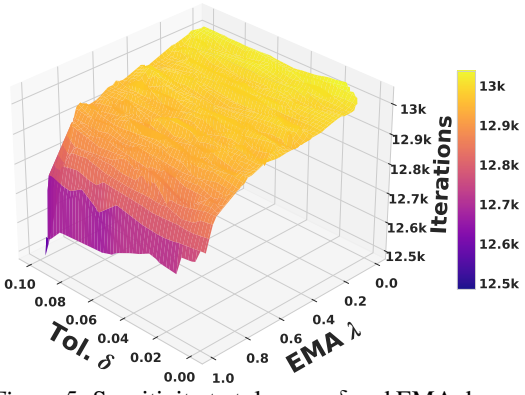


Figure 5: Sensitivity to tolerance δ and EMA decay λ on CNN: iterations needed by **EIPO** to reach the baseline accuracy at 60k steps, 1500 runs.

Acknowledgments

This project was supported in part by the NSFC Excellent Young Scientists Fund 62422118, in part by the Hong Kong RGC under Grants RFS2627-1S02, R1018-25F, N_CityU1114/25, and 11219324, and in part by the Hong Kong UGC under Grant UGC/FDS11/E03/24 and Grant UGC/FDS11/E03/25.

References

[1] Thalaiyasingam Ajanthan, Sameera Ramasinghe, Yan Zuo, Gil Avraham, and Alexander Long. Nesterov method for asynchronous pipeline parallel optimization. In *Forty-second International Conference on Machine Learning*.

[2] Donald G Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4):547–560, 1965.

[3] Anonymous. AMDP: Asynchronous multi-directional pipeline parallelism for large-scale models training. In *Submitted to The Fourteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=cfZNOO6ZHf. under review.

[4] Sun Ao, Weilin Zhao, Xu Han, Cheng Yang, Xinrong Zhang, Zhiyuan Liu, Chuan Shi, and Maosong Sun. Seq1f1b: Efficient sequence-level pipeline parallelism for large language model training. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8998–9008, 2025.

[5] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren,

Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

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

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

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

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

[17] 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.

[18] 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.

[19] Robert D Falgout, Stephanie Friedhoff, Tz V Kolev, Scott P MacLachlan, and Jacob B Schroder. Parallel time integration with multigrid. *SIAM Journal on Scientific Computing*, 36(6):C635–C661, 2014.

[20] Haw-ren Fang and Yousef Saad. Two classes of multisecant methods for nonlinear acceleration. *Numerical linear algebra with applications*, 16(3):197–221, 2009.

[21] 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.

[22] Martin J Gander and Stefan Guttel. Paraexp: A parallel integrator for linear initial-value problems. *SIAM Journal on Scientific Computing*, 35(2):C123–C142, 2013.

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

[24] Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

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

[26] 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.

[27] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[28] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[29] 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.

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

[31] 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.

[32] Carl T Kelley. *Iterative methods for linear and nonlinear equations*. SIAM, 1995.

[33] 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.

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

[35] Dana A Knoll and David E Keyes. Jacobian-free newton–krylov methods: a survey of approaches and applications. *Journal of Computational Physics*, 193(2):357–397, 2004.

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

[37] 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.

[38] Jacques-Louis Lions, Yvon Maday, and Gabriel Turinici. Résolution d'edp par un schéma en temps «pararéel». *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, 332(7): 661–668, 2001.

[39] 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.

[40] Bin Liu, Zhonghao Zhang, Banghao Zhai, Xing Wei, Gaobin Tian, and Zeyu Ji. Tlpipe: An efficient two-level optimization pipeline model parallelism for large model training. In *2025 6th International Conference on Computer Engineering and Application (ICCEA)*, pages 01–05. IEEE, 2025.

[41] 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.

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

[43] 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=2JihLwirx0.

[44] Jianrong Lu, Zhuoya Gu, Haobo Li, Zhiyu Zhu, Yechao Zhang, Jianhai Chen, Minghui Yang, Junwei Liu, Jian Wang, Qinming He, Hui LIU, and Junhui Hou. PASO: Step parallel stochastic optimization. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=pqG8AQFqhG.

[45] 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.

[46] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[47] Meta. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models. https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/, September 2024. Accessed: 2026-01-15.

[48] C Nagaraju, Yenda Ramesh, and C Krishna Mohan. A data parallel approach for distributed neural networks to achieve faster convergence. In *Sixteenth International Conference on Machine Vision (ICMV 2023)*, volume 13072, pages 380–389. SPIE, 2024.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

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

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

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

[56] Péter Pulay. Convergence acceleration of iterative sequences. the case of scf iteration. *Chemical physics letters*, 73(2):393–398, 1980.

[57] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[58] 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.

[59] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in neural information processing systems*, 24, 2011.

[60] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in neural information processing systems*, 24, 2011.

[61] 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.

[62] 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.

[63] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[64] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016.

[65] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[66] 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.

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

[68] 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.

[69] 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.

[70] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[71] 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.

[72] 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.

[73] 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.

[74] 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*.

[75] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[76] Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

[77] 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.

[78] Alex Toth and Carl T Kelley. Convergence analysis for anderson acceleration. *SIAM Journal on Numerical Analysis*, 53(2):805–819, 2015.

[79] 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.

[80] Homer F Walker and Peng Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011.

[81] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

[82] Phil Wang. vit-pytorch: Implementation of vision transformer in pytorch. https://github.com/lucidrains/vit-pytorch, 2020.

[83] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

[88] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[89] 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.

[90] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[91] Junzi Zhang, Brendan O'Donoghue, and Stephen Boyd. Globally convergent type-i anderson acceleration for nonsmooth fixed-point iterations. *SIAM Journal on Optimization*, 30(4):3170–3197, 2020.

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

[93] 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.

[94] 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.

[95] 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.

[96] 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.

[97] 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.

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

Appendix Contents

A Empirical and Theoretical Comparison Against OptEx **19**

A.1 Performance Comparison Against OptEx 19

A.2 Experimental Configuration 19

A.3 Evaluation Metrics and Results 19

A.4 Theoretical Complexity Analysis: EIPO vs. OptEx 20

B Experiment Details for LLM Tasks **21**

B.1 Dataset and Tokenization 21

B.2 Metrics and Evaluation 24

B.3 Hyperparameter Settings for Language Modeling tasks. 24

C Experiment Details for Image Classification Tasks **24**

C.1 Dataset and Data Augmentation 24

C.2 Model Architectures 25

C.3 Hyperparameter Settings 26

D Experiment Details for Diffusion Training Tasks **27**

D.1 Dataset and Data Augmentation 27

D.2 Model Architecture and Diffusion Settings 28

D.3 Training Process 28

D.4 Sampling Process 28

D.5 Metrics and Evaluation Details 28

D.6 Hyperparameter Settings for Diffusion Training 29

D.7 Visual Comparison for Diffusion Model 29

E Experiment Details: Inter-step parallelism v.s. Data-parallelism in Table 2 **29**

F Execution Commends **30**

F.1 Execution Commands of Quantitative Comparison in Table 3 30

F.2 Execution Commands of Large-scale Models in Tab. 5 31

F.3 Execution Commands for Impact of Window Size p in Fig. 4 32

F.4 Execution Commands of Impact of Batch Size B in Fig. 4 32

F.5 Execution Commands of Impact of Learning Rate η in Fig. 4 33

F.6 Execution Commands of Impact of Damping Factor γ in Fig. 4 33

F.7 Execution Commands of Quantitative Comparison in Table 4 34

F.8 Execution Commands for Reproducing Table 1 35

F.9 Execution Commands for Data-parallelism Comparision Commands in Table 2 . . 36

G Proof Appendix **36**

G.1 Proof of Optimality for Quasi-Newton Preconditioning in Theorem 4.3 37

G.2 Proof of Convergence for **EIPO** with Sliding Window 38

A Empirical and Theoretical Comparison Against OptEx

A.1 Performance Comparison Against OptEx

In this section, we present a direct empirical evaluation comparing our approach with **OptEx**. To ensure a fair assessment, we adopt the evaluation protocol from OptEx and test the algorithms on the identical set of synthetic benchmark functions provided in their official code repository[1]. These include the Sphere, Ackley, and Rosenbrock functions.

Specifically, for a d-dimensional input vector $\theta = [\theta_i]_{i=1}^d$, these target optimization functions are defined as follows:

- **Sphere Function**:

$$F(\theta) = \sqrt{\frac{1}{d}\sum_{i=1}^{d}\theta_i^2}$$

  The Sphere function possesses a global minimum of $F(\theta) = 0$, which is located at $\theta^* = 0$.

- **Ackley Function**:

$$F(\theta) = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d}\theta_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi\theta_i)\right) + 20 + \exp(1)$$

  Similar to the Sphere function, the Ackley function reaches its global minimum of $F(\theta) = 0$ at $\theta^* = 0$.

- **Rosenbrock Function**:

$$F(\theta) = \frac{1}{d}\sum_{i=1}^{d-1}\left[100(\theta_{i+1} - \theta_i)^2 + (1 - \theta_i)^2\right]$$

  In contrast to the previous two, the Rosenbrock function achieves its minimum of $F(\theta) = 0$ at $\theta^* = 1$.

A.2 Experimental Configuration

Our experimental environment closely mirrors the setup established in the OptEx repository. Because the publicly available OptEx codebase only includes fully reproducible implementations for these three specific synthetic problems, we restrict our comparative analysis to these tasks. This decision avoids potential misconfigurations related to their complex kernel function estimation and guarantees an equitable baseline comparison.

Throughout our tests, the Adam optimizer acts as the foundational solver, with its standard sequential execution representing our **Vanilla** baseline. To guarantee identical testing conditions—including random weight initializations, evaluation harnesses, and function querying—we seamlessly embedded our **EIPO** algorithm into the existing *optex_cmp.py* script from the OptEx codebase.

A.3 Evaluation Metrics and Results

Our primary criterion for assessment is the **Optimality Gap**. We define this as the terminal objective score, $f(x_T)$, reached by the Vanilla Adam baseline after completing $T = 1000$ iterations. Given that the global minimum f^* is exactly 0 for all three test functions, the final loss value $f(x_T)$ perfectly corresponds to this optimality gap.

We subsequently record the total number of sequential **Iterations** (representing communication overhead) that both OptEx and **EIPO** need to attain this target optimality gap. The overall **Speedup** is defined by dividing the baseline iterations (1000) by the number of iterations consumed by the respective parallelized approach.

Table 6 summarizes these quantitative findings, strongly illustrating the enhanced operational efficiency of the **EIPO** framework.

[1] https://github.com/youyve/OptEx

Table 6: Quantitative performance comparison across different optimization schemes. Top-performing metrics are marked in **bold**. Arrows indicate whether higher ("↑") or lower ("↓") values are preferable. Results denote the mean and standard deviation of the optimality gap computed across 5 distinct experimental trials.

Method	Sphere function, $d = 1e4$, $\eta = 1e-3$, $T = 1000$		
	Optimality gap ↓	Iters ↓	Speedup ↑
Adam	1.95 ± 0.12	1000	$1.0\times$
OptEx	1.95 ± 0.08	504	$1.98\times$
EIPO ($\mathcal{Z} = -I$)	1.95 ± 0.15	214	$4.67\times$
EIPO ($\mathcal{Z} = \bar{J}$)	1.95 ± 0.13	**102**	**9.80**$\times$
Method	Ackley function, $d = 1e4$, $\eta = 1e-3$, $T = 1000$		
	Optimality gap ↓	Iters ↓	Speedup ↑
Adam	0.40 ± 0.18	1000	$1.0\times$
OptEx	0.40 ± 0.06	162	$6.17\times$
EIPO ($\mathcal{Z} = -I$)	0.40 ± 0.11	86	$11.63\times$
EIPO ($\mathcal{Z} = \bar{J}$)	0.40 ± 0.14	**44**	**22.73**$\times$
Method	Rosenbrock function, $d = 1e4$, $\eta = 1e-3$, $T = 1000$		
	Optimality gap ↓	Iters ↓	Speedup ↑
Adam	5.50 ± 0.09	1000	$1.0\times$
OptEx	5.50 ± 0.17	479	$2.08\times$
EIPO ($\mathcal{Z} = -I$)	5.50 ± 0.04	210	$4.76\times$
EIPO ($\mathcal{Z} = \bar{J}$)	5.50 ± 0.06	**95**	**10.53**$\times$

- **Regarding the Sphere function**, every evaluated method successfully converges to a 1.95 optimality gap. Yet, EIPO achieves this in a mere **214** iterations, translating to a **4.67**$\times$ acceleration. This notably outpaces OptEx (504 iterations, $1.98\times$ speedup) by a factor of **2.36**$\times$.
- **For the Ackley function**, EIPO's advantage becomes even more stark. It hits the 0.4 target gap using only **86** iterations, yielding a remarkable **11.63**$\times$ speedup, whereas OptEx consumes 162 iterations to secure a $6.17\times$ speedup.
- **Evaluating the complex Rosenbrock function**, EIPO sustains its substantial lead. To hit the 5.5 optimality threshold, EIPO requires just 210 iterations (**4.76**$\times$ speedup), easily surpassing the $2.08\times$ speedup (479 iterations) delivered by OptEx.

Ultimately, across the suite of standard benchmarks, EIPO demonstrates consistent and decisive superiority over OptEx by demanding drastically fewer communication cycles to secure identical optimization quality. These outcomes underscore the robustness of EIPO's adaptive, error-driven termination strategy, proving it facilitates a vastly more efficient parallel execution than the kernel approximation mechanisms employed by OptEx.

A.4 Theoretical Complexity Analysis: EIPO vs. OptEx

To further understand the operational advantages of our approach, we provide a comparative complexity analysis focusing on the computational, space, and communication overheads.

A.4.1 Conceptual Framework

We define a "main iteration" as the unit of comparison:

- For EIPO, this is one parallel iteration k of our method solver.

- For **OptEx**, this is a single sequential iteration t that dispatches N parallel gradient computations.

A.4.2 Efficiency Breakdown

1. Computational Complexity: OptEx's computational cost is heavily influenced by its dependence on historical data. Building its kernelized estimation model μ_t involves operations on a $T_0 \times T_0$ matrix, resulting in a $\mathcal{O}(T_0^3)$ cost. Additionally, its serial multi-step proxy updates scale as $\mathcal{O}(NT_0d)$. In contrast, **EIPO**'s overhead stems from serial post-processing of parallel gradients. For stateful optimizers like Adam, this involves a dependency chain of length p, with weight updates requiring $\mathcal{O}(p^2d)$ operations. For large-scale models where d is substantial, the comparison effectively becomes $\mathcal{O}(NT_0d)$ for OptEx versus $\mathcal{O}(p^2d)$ for **EIPO**.

2. Space Complexity (Memory Overhead): **EIPO** demonstrates a significant advantage in memory efficiency. Its footprint is strictly linear with the parallelism degree, $\mathcal{O}(pd)$, storing only current window weights and optimizer states. Conversely, OptEx must buffer T_0 historical gradients and the kernel matrix, leading to a much larger $\mathcal{O}((T_0 + N)d)$ requirement.

3. Communication and Scaling: While both methods share similar communication costs—scaling with N or p to gather gradients—**EIPO** provides a superior speedup scaling law. **EIPO** is theoretically capable of achieving near-linear speedup ($\mathcal{O}(N)$), whereas OptEx is bounded by a sub-linear rate of $\mathcal{O}(\sqrt{N})$.

A.4.3 Summary of Analysis

Table 7: Comparative analysis of per-iteration complexity, assuming parallelism degree $p = N = G$.

Metric	**OptEx**	**EIPO**
Parallelism Degree	N	$p = N$
Total Computation	$\mathcal{O}(T_0^3 + NT_0(d + T_0))$	$\mathcal{O}(N^2d)$
Total Space (Overhead)	$\mathcal{O}(T_0d + T_0^2 + Nd)$	$\mathcal{O}(Nd)$
Total Communication	$\mathcal{O}(t_{comm}(N, d))$	$\mathcal{O}(t_{comm}(N, d))$
Speedup rate	$\mathcal{O}(\sqrt{N})$	$\mathcal{O}(N)$

This analysis suggests that while computational costs are comparable, OptEx's reliance on historical gradient records results in a memory overhead approximately $\frac{T_0+N}{N}$ times higher than **EIPO**. This makes OptEx increasingly impractical for large-scale models where VRAM is a critical bottleneck. Furthermore, **EIPO**'s linear scaling ensures it can more effectively leverage massive GPU clusters compared to the sub-linear scaling of OptEx.

B Experiment Details for LLM Tasks

This section provides a technical specification of the experimental environment, dataset configurations, and the exact execution parameters used for the Large Language Model (LLM) tasks.

B.1 Dataset and Tokenization

The **WikiText-2** dataset, a widely recognized benchmark in the field of natural language processing, is employed for all language modeling evaluations in this study. Derived from a curated collection of verified "Good" and "Featured" articles on Wikipedia, it comprises approximately million tokens with a vocabulary size of unique words, ensuring a high level of lexical diversity and linguistic complexity compared to earlier datasets like Penn Treebank. The dataset is utilized under a causal language modeling (CLM) objective, where the primary task is to predict the probability of a succeeding token given its preceding context. By preserving the original punctuation, casing, and multi-paragraph structure of the source long-form articles, WikiText-2 serves as a rigorous testbed for evaluating a model's ability to capture long-range dependencies and maintain semantic coherence across extended sequences.



Figure 6: Randomly generated 25 images on the LSUN Church dataset using the **EIPO** method.

B.1.1 Tokenizer and Preprocessing

- **GPT-2 Configuration:** We use the pre-trained GPT-2 tokenizer. To facilitate batch training, the `eos_token` is utilized as the `pad_token`.
- **Llama-3.2-1B Configuration:** We utilize the tiktoken-based tokenizer associated with the Llama-3 architecture.
- **Sequence Formatting:** All text is tokenized and truncated to a maximum sequence length of 512 tokens. During training, labels are generated by shifting the input sequences. Padding tokens are assigned a label value of -100, ensuring they are bypassed by the Cross-Entropy loss function.

B.1.2 Data Loading

For optimal throughput, the `DataLoader` is configured with `pin_memory=True` and `persistent_workers=True`, utilizing 8 worker threads per process. In the parallel paradigm of **EIPO**, updates are computed across a sliding window of size p. If `pin_memory` and `persistent_workers` are disabled, the system incurs a significant initialization overhead at every iteration due to the repeated creation of worker threads. Given the frequent synchronization re-



Figure 7: Large-Scale Visual Comparison of Generated Samples: Assessing the Impact of **EIPO** on Adam-based Diffusion Generation. The figure presents a comprehensive set of 50 image pairs divided into five structural blocks. Each block pairs the outputs of the baseline Adam (top rows) and the proposed **EIPO** +Adam (bottom rows) using matched random seeds. This extensive visualization demonstrates the robustness of the **EIPO** method, verifying that the algorithmic modifications do not compromise, and in some cases enhance, the visual diversity and texture details of the generated images.

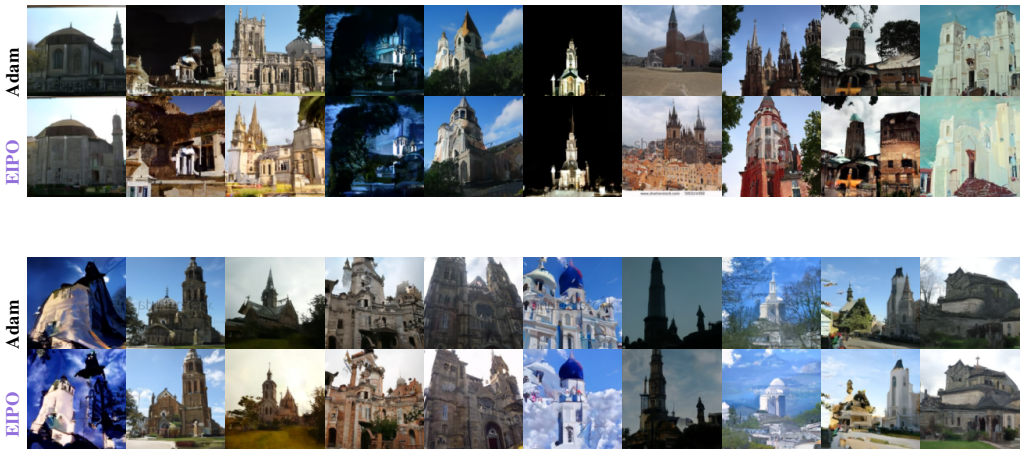


Figure 8: Additional visual comparison of generated samples with 20 previously unused random seeds (seeds 200–219), continuing Fig. 7. For each seed, Adam (top rows) and **EIPO** +Adam (bottom rows) share the identical initial noise, yielding a strict one-to-one correspondence. Across all pairs, **EIPO** preserves the sample composition and structure of the Adam baseline while maintaining comparable visual quality, further corroborating that step-parallel training does not disturb the optimization trajectory of sequential Adam.

quired in parallel step computation, these latencies would exceed the computational gains, effectively nullifying the efficiency provided by the parallel window.

B.2 Metrics and Evaluation

- **Perplexity (PPL):** The standard evaluation metric for language models, computed as $\exp(\mathcal{L})$, where $\mathcal{L}$ is the average negative log-likelihood.
- **Token Throughput:** We define total tokens as the cumulative count of non-padding tokens processed during training.
- **Peak Memory:** Reported as the maximum GPU memory allocated during the training window.

B.3 Hyperparameter Settings for Language Modeling tasks.

Table 8 provides a unified summary of the hyperparameters applied across all Large Language Model experiments.

C Experiment Details for Image Classification Tasks

In this section, we provide the detailed configurations for the image classification experiments conducted on the CIFAR-10 dataset. Our experiments evaluate the performance and efficiency of different optimization strategies across two representative architectures: Vision Transformers (ViT) and ResNet50.

C.1 Dataset and Data Augmentation

We utilize the **CIFAR-10** dataset, a foundational benchmark in computer vision consisting of color images distributed uniformly across mutually exclusive classes, including categories such as airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is partitioned into a training set of images and a test set of images, providing a balanced environment for evaluating multiclass image classification algorithms. To improve model robustness and prevent overfitting, we apply the following data augmentation and preprocessing pipeline during the training phase:

Table 8: Hyperparameter Settings for Language Modeling tasks.

Hyperparameter	**GPT-2**	**Llama-3.2-1B**
Max Steps (T)	5,000	5,000
Sequence Length	512	512
Learning Rate (η)	6×10^{-5}	1×10^{-5}
LR Scheduler	Constant	Cosine Annealing
SGD Momentum	$\beta = 0.9$	$\beta = 0.9$
Adam Momentum	$\beta_1 = 0.9, \beta_2 = 0.999$	$\beta_1 = 0.9, \beta_2 = 0.999$
AdamW Momentum	$\beta_1 = 0.9, \beta_2 = 0.999$	$\beta_1 = 0.9, \beta_2 = 0.999$
SGD Weight Decay	0	0
Adam Weight Decay	0	0
AdamW Weight Decay	0.01	0.01
Warmup Steps	100	100
Data Precision	bfloat16	bfloat16
Batch Size (B)	112	30
DataLoader Workers	8	8
Pin Memory	True	True
DataLoader Workers Persistent	True	True
EIPO Damping factor γ for $\mathcal{Z} = \bar{J}$	**Values**	
Damping factor γ on SGD	0.99	0.2
Damping factor γ on Adam	0.99	0.2
Damping factor γ on AdamW	0.99	0.2
EIPO's Other Hyperparameters	**Values**	
Window Size (p)	7	7
Threshold (δ)	1×10^{-5}	1×10^{-4}
EMA Decay (λ)	0.999	0.999
Adaptivity Method	Mean	Mean

- **Random Cropping**: Images are padded by 4 pixels on each side and then randomly cropped to .
- **Random Horizontal Flip**: Images are flipped horizontally with a probability of 0.5.
- **Normalization**: All input images are normalized using the CIFAR-10 dataset-specific mean and standard deviation .

C.2 Model Architectures

We employ two distinct types of architectures to validate the generalizability of our training approach:

C.2.1 CNN Model

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 10.

C.2.2 Vision Transformer (ViT)

The ViT model is configured for input resolution. To adapt the architecture for small-scale image data, we use a patch size of . The specific structural parameters are:

- **Embedding Dimension**: 512
- **Depth (Number of Layers)**: 6
- **Attention Heads**: 8
- **MLP Dimension**: 512

Table 9: Hyperparameter Settings for Large-scale Model tasks.

Hyperparameter	**Gemma-2B**	**Qwen1.5-4B**	**Qwen2.5-3B**	**GPT-J-6B**
Max Steps (T)	1,000	1,000	1,000	1,000
Sequence Length	512	512	512	512
Learning Rate (η)	1×10^{-5}	1×10^{-5}	1×10^{-5}	1×10^{-5}
LR Scheduler	Cosine Annealing	Cosine Annealing	Cosine Annealing	Cosine Annealing
SGD Momentum	$\beta = 0.9$	$\beta = 0.9$	$\beta = 0.9$	$\beta = 0.9$
Adam Momentum	$\beta_1 = 0.9, \beta_2 = 0.999$	$\beta_1 = 0.9, \beta_2 = 0.999$	$\beta_1 = 0.9, \beta_2 = 0.999$	-
AdamW Momentum	$\beta_1 = 0.9, \beta_2 = 0.999$	$\beta_1 = 0.9, \beta_2 = 0.999$	$\beta_1 = 0.9, \beta_2 = 0.999$	-
SGD Weight Decay	0	0	0	0
Adam Weight Decay	0	0	0	-
AdamW Weight Decay	0.01	0.01	0.01	-
Warmup Steps	100	100	100	100
Data Precision	bfloat16	bfloat16	bfloat16	bfloat16
Batch Size (B)	24	22	24	20
DataLoader Workers	8	8	8	8
Pin Memory	True	True	True	True
DataLoader Workers Persistent	True	True	True	True
EIPO Damping factor γ for $\mathcal{Z} = \bar{J}$	**Values**			
Damping factor γ on SGD	0.2	0.2	0.2	0.2
Damping factor γ on Adam	0.2	0.2	0.2	-
Damping factor γ on AdamW	0.2	0.2	0.2	-
EIPO's Other Hyperparameters	**Values**			
Window Size (p)	7	7	7	7
Threshold (δ)	1×10^{-1}	1×10^{-1}	1×10^{-1}	1×10^{-4}
EMA Decay (λ)	0.999	0.999	0.999	0.999
Adaptivity Method	Mean	Mean	Mean	Mean

Table 10: 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

C.2.3 ResNet50 with Group Normalization (ResNet50-GN)

For the convolutional neural network baseline, we use the ResNet50 architecture. However, we modify the standard architecture by replacing all Batch Normalization (BN) layers with Group Normalization (GN) using 32 groups. This modification is motivated by the fact that BN relies on batch-wise statistics, which can introduce instability and performance degradation in certain parallel training regimes or large-batch settings. GN provides a more stable alternative by normalizing across groups of channels, making the training process more robust to batch size variations and synchronization noise.

C.3 Hyperparameter Settings

All models are trained for a maximum of 100,000 steps using a large batch size of 2048 to facilitate efficient hardware utilization. We use a Cosine Annealing learning rate scheduler with a warmup phase of 10,000 steps.

The detailed hyperparameter settings for both ViT and ResNet50 are summarized in Table 11.

Table 11: Hyperparameter settings for Image Classification tasks on CIFAR-10. All experiments are implemented using PyTorch. Our publicly available code provides all details.

Hyperparameter	**CNN**	**Vision Transformer (ViT)**	**ResNet50**
Max Steps (T)	600,000	0.1M (Adam/AdamW), 0.24M (SGD)	0.1M (Adam/AdamW), 0.24M (SGD)
Batch Size (B)	4,096	2,048	2,048
Learning Rate (η)	1.0×10^{-3}	1.5×10^{-4}	5.0×10^{-5}
Warmup Steps	0	10,000	10,000
LR Scheduler	Constant	Cosine Annealing	Cosine Annealing
Normalization	BatchNorm	LayerNorm	GroupNorm ($G = 32$)
SGD Momentum	$\beta = 0.9$	$\beta = 0.9$	$\beta = 0.9$
Adam Momentum	$\beta_1 = 0.9, \beta_2 = 0.999$	$\beta_1 = 0.9, \beta_2 = 0.999$	$\beta_1 = 0.9, \beta_2 = 0.999$
AdamW Momentum	$\beta_1 = 0.9, \beta_2 = 0.999$	$\beta_1 = 0.9, \beta_2 = 0.999$	$\beta_1 = 0.9, \beta_2 = 0.999$
SGD Weight Decay	0	0	0
Adam Weight Decay	0	0	0
AdamW Weight Decay	0.01	0.01	0.01
Loss Type	Cross Entropy	Cross Entropy	Cross Entropy
Validation Interval	100 Steps	100 Steps	100 Steps
EIPO Settings for SGD	**Value (CNN)**	**Value (ViT)**	**Value (ResNet50)**
Window Size (p)	7	7	7
Tolerance Threshold (δ)	0.01	0.1	0.9
EMA Decay (λ)	0.9	0.999	0.999
δ update method	median	median	median
Damping factor (γ)	0.1	0.03	0.05
EIPO Settings for Adam	**Value (CNN)**	**Value (ViT)**	**Value (ResNet50)**
Window Size (p)	7	7	7
Tolerance Threshold (δ)	0.01	0.1	0.1
EMA Decay (λ)	0.999	0.999	0.999
δ update method	median	median	median
Damping factor (γ)	0.99	0.4	0.01
EIPO Settings for AdamW	**Value (CNN)**	**Value (ViT)**	**Value (ResNet50)**
Window Size (p)	7	7	7
Tolerance Threshold (δ)	0.2	0.1	0.1
EMA Decay (λ)	0.999	0.999	0.999
δ update method	median	median	median
Damping factor (γ)	0.99	0.2	0.1

D Experiment Details for Diffusion Training Tasks

This section details the experimental configuration, dataset processing, and execution parameters for the Diffusion model training tasks, specifically comparing standard sequential training with the EIPO parallel acceleration framework.

D.1 Dataset and Data Augmentation

We utilize the **LSUN Church** (Church Outdoor) dataset for all image generation experiments. The dataset is processed to ensure high-fidelity image synthesis at a resolution of 128×128.

D.1.1 Preprocessing and "Tokenization" equivalent

While diffusion models operating in pixel space do not utilize discrete tokenizers, the image processing pipeline serves as the input representation stage:

- **Image Rescaling:** All input images are resized to 128×128 pixels.
- **Normalization:** Pixel values are mapped from the range $[0, 255]$ to $[0, 1]$ and subsequently auto-normalized within the `GaussianDiffusion` module to facilitate stable gradient flow.
- **Augmentation:** We apply `RandomHorizontalFlip` and `CenterCrop` during training to enhance the diversity of the training distribution and prevent over-fitting.

D.1.2 High-Throughput Data Loading

The `DataLoader` is configured to maximize GPU utilization and minimize synchronization overhead between the parallel workers in the EIPO window:

- **Worker Configuration:** We assign 12 worker threads per process (`num_workers=12`).
- **Memory Management:** To accelerate host-to-device transfers, `pin_memory=True` is enabled.

- **Persistence:** We set `persistent_workers=True`. In the EIPO framework, since multiple steps are computed in a sliding window P, the persistent workers prevent the overhead of re-initializing the data pipeline at the start of every parallel rollout, which is essential for maintaining the reported $3.0\times$ speedup.
- **Shuffling:** Stochasticity is ensured by setting `shuffle=True` for the training set.

D.2 Model Architecture and Diffusion Settings

We employ a **U-net** backbone within a **Gaussian Diffusion** framework in the source code [2] from DDPM [28] to train the diffusion model and compute FID. Specifically, following DDPM [28], our neural network architecture follows the backbone of PixelCNN++ [65], which is a U-Net [63] based on a Wide ResNet [90]. We replaced weight normalization [64] with group normalization [83] to make the implementation simpler. All models have two convolutional residual blocks per resolution level and self-attention blocks at the 16×16 resolution between the convolutional blocks. Diffusion time t is specified by adding the Transformer sinusoidal position embedding into each residual block.

D.2.1 Unet Hyperparameters

The UNet architecture follows a hierarchical structure with the following specifications:

- **Base Dimension:** 64.
- **Dimension Multipliers:** $(1, 2, 4, 4, 8, 8)$.
- **Attention Layers:** Full self-attention is integrated at the 5^{th} and 6^{th} resolution levels, while linear attention is used in intermediate blocks to balance computational cost and modeling capacity.

D.2.2 Diffusion Probabilistic Process

- **Noise Schedule:** A linear beta schedule is used with $1,000$ diffusion timesteps for training.
- **Objective:** The model is trained under the `pred_noise` (epsilon-prediction) objective.
- **EMA:** An Exponential Moving Average (EMA) of model weights is maintained with a decay rate of 0.9999 and updated every 10 iterations to ensure the stability of the generated samples.

D.3 Training Process

During the training process, following DDPM [28], we applied Exponential Moving Average (EMA) to the model parameters with a decay factor of 0.9999. We configured the system to save the EMA-smoothed model every 10,000 steps; this model is subsequently used for FID calculation during the sampling phase.

D.4 Sampling Process

During the sampling process, we generate 50,000 images and utilize the source code from DDPM [28] to calculate the FID score. Specifically, we employ a 100-step DDIM [70] solver to sample the 50,000 images.

D.5 Metrics and Evaluation Details

- **FID (Fréchet Inception Distance):** Quantitative evaluation is performed by calculating the FID between $50,000$ generated samples and the ground-truth LSUN Church dataset [3]. Feature extraction is performed using a pre-trained InceptionV3 network.
- **Training Efficiency:** Measured by total wall-clock time (days), iteration counts, and peak GPU memory (VRAM).

[2] https://github.com/hojonathanho/diffusion
[3] https://huggingface.co/datasets/tglcourse/lsun_church_train

D.6 Hyperparameter Settings for Diffusion Training

Table 12 summarizes the hyperparameter configurations for both the diffusion process and the EIPO acceleration framework.

Table 12: Hyperparameter settings for Diffusion Training tasks in Table 1.

Hyperparameter	**Value**
Max Steps (T)	600,000
Image Size	128×128
Batch Size (B)	160
Learning Rate (η)	2×10^{-5}
LR Scheduler	Cosine Annealing
Warmup Steps	10,000
Optimizer	Adam
Adam Betas	$\beta_1 = 0.9, \beta_2 = 0.99$
Gradient Clipping	1.0
EMA Model Data Augmentation (Horizontal Flip)	True
DataLoader Workers	12
Pin Memory	True
Persistent Workers	True
UNet Architecture	Hierarchical Structure
UNet Base Dimension	64
UNet Dimension Multipliers	(1, 2, 4, 4, 8, 8)
UNet Full Attention Layers	the 5^{th} and 6^{th} resolution levels.
UNet Linear Attention Layers	Intermediate Blocks
Noise Schedule	Liner beta schedule is used with 1, 000 diffusion timesteps
Training Objective	Noise Prediction
EMA Model Decay Rate	0.9999
EMA Model Updating Interval	10 steps
Sampling Solver	DDIM
Sampling Steps	100
EIPO Parameters	**Values**
Window Size (P)	7
Threshold (δ)	1×10^{-6}
EMA Decay (λ)	0.99
Adaptivity Type	Median
Damping factor γ	0.01

D.7 Visual Comparison for Diffusion Model

To ensure a fair assessment, we generated samples using a fixed set of 50 random seeds, creating a strict one-to-one correspondence between the outputs of Adam and Adam plus EIPO.

The comprehensive visual results are presented in Figure 7. From these visualizations, two primary conclusions can be drawn:

- **Consistency in Convergence Path:** The images generated by EIPO are visually nearly identical to those produced by Adam under the same random seeds. This strongly suggests that EIPO introduces negligible deviation to the optimization trajectory, adhering to the convergence path of the serial Adam optimizer.
- **Preservation of Generative Quality:** The perceptual quality of the generated samples in terms of structural coherence, texture details, and overall fidelity is comparable between the two methods. This indicates that the step parallelization strategy of EIPO does not compromise the generative capabilities or degrade the quality of the model.

E Experiment Details: Inter-step parallelism v.s. Data-parallelism in Table 2

This section provides the comprehensive experimental configuration used to generate the results in Table 2. All experiments were performed using 8 NVIDIA A100 GPUs (80GB) interconnected via NVLink.

Common Configuration To ensure consistency across all methods (Serial Adam, EIPO, DP, DDP, and DeepSpeed), we utilized the GPT-2 (124M parameters) architecture. The training was conducted on the Wikitext-2 dataset with a fixed total batch size of 112 and a maximum sequence length of 512. The optimizer was set to Adam with a constant learning rate of 6×10^{-5}.

EIPO and Naive DP Settings Both EIPO and *DP w/o Optimization* are implemented as academic prototypes. They rely on Python `multiprocessing.Queue` for communication between the master process and worker processes. In EIPO, the master GPU manages the parameter updates and synchronization for a parallel window size of $P = 21$, using an error threshold $\epsilon = 0.1$ and an EMA decay of 0.999 for step adaptivity. *DP w/o Optimization* utilizes the same queue-based architecture to aggregate gradients from 8 GPUs before performing a centralized update.

Optimized Baseline Settings The *DDP* implementation uses the standard PyTorch `DistributedDataParallel` wrapper with the NCCL backend, which employs bucketed All-Reduce for efficient gradient synchronization. The *DeepSpeed* implementation utilizes the ZeRO-3 optimization stage, which partitions gradients and optimizer states across GPUs, effectively reducing memory redundancy and overlapping communication with computation kernels.

Table 13: Detailed Experimental Parameters for Comparison Methods in Table 2

Category	EIPO (Unoptimized)	DP (Unoptimized)	DP (DDP)	DP (DeepSpeed)
Model Architecture	GPT-2			
Dataset	Wikitext-2			
Hardware	$8 \times$ NVIDIA A100 (80GB); All GPU pairs use NVLink 3.0; PCIe is bypassed for inter-GPU data transfer			
Total Batch Size	112	112	112	112
Max Sequence Length	512			
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)			
Learning Rate	6×10^{-5} (Constant Scheduler)			
Precision	BFloat16			
Communication Backend	MP Queue (CPU)	MP Queue (CPU)	NCCL (GPU)	NCCL (GPU)
Comm. Frequency	Every Iteration	Every Iteration	Every Iteration	Every Iteration
Optimization Level	Naive Prototype	Naive Prototype	Bucketed All-Reduce	ZeRO-Stage 3
Window Size (P)	21	N/A	N/A	N/A
Adaptivity Threshold	0.1	N/A	N/A	N/A
Adaptivity EMA Decay	0.999	N/A	N/A	N/A
Damping factor γ	0.0	N/A	N/A	N/A

F Execution Commends

F.1 Execution Commands of Quantitative Comparison in Table 3

The primary results in Table 3 are derived from the following execution commands.

F.1.1 GPT-2 Experiments

The commands below initialize the GPT-2 model with pre-trained weights for 1,000 training steps.

Listing 1: GPT-2 Serial and Parallel (EIPO) Execution Commands for Reproducing GPT-2 Results in Table 3

```
# Serial Adam/AdamW Baseline
python llm.py --local_model_dir ./gpt2 --modelscope_name "gpt2" --max_steps 1000
    ↪ --batch_size 112 --learning_rate 6e-5 --training_mode serial
    ↪ --optimizer_type adam/adamw --lr_scheduler_type constant
# Parallel Adam/AdamW (\eipo)
python llm.py --modelscope_name "gpt2" --max_steps 5000 --batch_size 112
    ↪ --learning_rate 6e-5 --threshold 1e-4 --ema_decay 0.999 --training_mode
    ↪ parallel --gamma 0.99 --P 7 --optimizer_type adam/adamw
    ↪ --lr_scheduler_type constant

# Serial SGD Baseline
python llm.py --modelscope_name "gpt2" --max_steps 1000 --batch_size 112
    ↪ --learning_rate 6e-5 --training_mode parallel --optimizer_type sgd
    ↪ --lr_scheduler_type constant
```

```
# Parallel Adam (\eipo)
python llm.py --modelscope_name "gpt2" --max_steps 5000 --batch_size 112
    ↪ --learning_rate 6e-5 --threshold 1e-4 --ema_decay 0.999 --training_mode
    ↪ parallel --gamma 0.99 --P 7 --optimizer_type sgd --lr_scheduler_type
    ↪ constant
```

F.1.2 Llama-3.2-1B Experiments

For the 1B-parameter Llama model, the commands are as follows.

Listing 2: Llama-3.2-1B Execution Commands for Reproducing Llama-3.2-1B Results in Table 3

```
# Parallel AdamW/Adam/SGD (\eipo)
python llm.py --local_model_dir "./Llama-3.2-1B" --modelscope_name "Llama-3.2-1B"
    ↪ --max_steps 5000 --batch_size 30 --learning_rate 1e-5 --threshold 1e-5
    ↪ --ema_decay 0.999 --training_mode parallel --P 7 --gamma 0.2
    ↪ --optimizer_type adamw --lr_scheduler_type cosine
python llm.py --local_model_dir "./Llama-3.2-1B" --modelscope_name "Llama-3.2-1B"
    ↪ --max_steps 5000 --batch_size 30 --learning_rate 1e-5 --threshold 1e-5
    ↪ --ema_decay 0.999 --training_mode parallel --P 7 --gamma 0.2
    ↪ --optimizer_type adam --lr_scheduler_type cosine
python llm.py --local_model_dir "./Llama-3.2-1B" --modelscope_name "Llama-3.2-1B"
    ↪ --max_steps 5000 --batch_size 30 --learning_rate 1e-5 --threshold 1e-5
    ↪ --ema_decay 0.999 --training_mode parallel --P 7 --gamma 0.2
    ↪ --optimizer_type sgd --lr_scheduler_type cosine


# Serial AdamW/Adam/SGD Baseline
python llm.py --local_model_dir "./Llama-3.2-1B" --modelscope_name "Llama-3.2-1B"
    ↪ --max_steps 5000 --batch_size 30 --learning_rate 1e-5 --training_mode
    ↪ serial --optimizer_type adamw --lr_scheduler_type cosine
python llm.py --local_model_dir "./Llama-3.2-1B" --modelscope_name "Llama-3.2-1B"
    ↪ --max_steps 5000 --batch_size 30 --learning_rate 1e-5 --training_mode
    ↪ serial --optimizer_type adam --lr_scheduler_type cosine
python llm.py --local_model_dir "./Llama-3.2-1B" --modelscope_name "Llama-3.2-1B"
    ↪ --max_steps 5000 --batch_size 30 --learning_rate 1e-5 --training_mode
    ↪ serial --optimizer_type sgd --lr_scheduler_type cosine
```

F.2 Execution Commands of Large-scale Models in Tab. 5

For the large-scale model, the commands are as follows.

Listing 3: Large-scale model Execution Commands for Reproducing Results in Tab. 5

```
#\eipo
python llm.py --local_model_dir "./Gemma-2B" --modelscope_name "Gemma-2B"
    ↪ --max_steps 1000 --batch_size 24 --learning_rate 1e-5 --threshold 1e-1
    ↪ --ema_decay 0.999 --training_mode parallel --P 7 --gamma 0.2
    ↪ --optimizer_type adamw --lr_scheduler_type cosine
python llm.py --local_model_dir "./Qwen1.5-4B" --modelscope_name "Qwen1.5-4B"
    ↪ --max_steps 1000 --batch_size 22 --learning_rate 1e-5 --threshold 1e-1
    ↪ --ema_decay 0.999 --training_mode parallel --P 7 --gamma 0.2
    ↪ --optimizer_type adamw --lr_scheduler_type cosine
python llm.py --local_model_dir "./Qwen2.5-3B" --modelscope_name "Qwen2.5-3B"
    ↪ --max_steps 1000 --batch_size 20 --learning_rate 1e-5 --threshold 1e-1
    ↪ --ema_decay 0.999 --training_mode parallel --P 7 --gamma 0.2
    ↪ --optimizer_type adamw --lr_scheduler_type cosine
python llm.py --local_model_dir "./GPT-J-6B" --modelscope_name "GPT-J-6B"
    ↪ --max_steps 1000 --batch_size 24 --learning_rate 1e-5 --threshold 1e-1
    ↪ --ema_decay 0.999 --training_mode parallel --P 7 --gamma 0.2
    ↪ --optimizer_type sgd --lr_scheduler_type cosine


# Serial AdamW/Adam/SGD Baseline
```

```
python llm.py --local_model_dir "./Gemma-2B" --modelscope_name "Gemma-2B"
    ↪ --max_steps 1000 --batch_size 24 --learning_rate 1e-5 --threshold 1e-1
    ↪ --ema_decay 0.999 --training_mode serial --P 7 --gamma 0.2
    ↪ --optimizer_type adamw --lr_scheduler_type cosine
python llm.py --local_model_dir "./Qwen1.5-4B" --modelscope_name "Qwen1.5-4B"
    ↪ --max_steps 1000 --batch_size 22 --learning_rate 1e-5 --threshold 1e-1
    ↪ --ema_decay 0.999 --training_mode serial --P 7 --gamma 0.2
    ↪ --optimizer_type adamw --lr_scheduler_type cosine
python llm.py --local_model_dir "./Qwen2.5-3B" --modelscope_name "Qwen2.5-3B"
    ↪ --max_steps 1000 --batch_size 20 --learning_rate 1e-5 --threshold 1e-1
    ↪ --ema_decay 0.999 --training_mode serial --P 7 --gamma 0.2
    ↪ --optimizer_type adamw --lr_scheduler_type cosine
python llm.py --local_model_dir "./GPT-J-6B" --modelscope_name "GPT-J-6B"
    ↪ --max_steps 1000 --batch_size 24 --learning_rate 1e-5 --threshold 1e-1
    ↪ --ema_decay 0.999 --training_mode serial --P 7 --gamma 0.2
    ↪ --optimizer_type sgd --lr_scheduler_type cosine
```

F.3 Execution Commands for Impact of Window Size p in Fig. 4

To reproduce the results in Fig. 4, we vary the parallel window size p while keeping other parameters constant.

Listing 4: Command for Window Size (p) Sensitivity Analysis for Reproducing Results in Fig. 4

```
# Parallel AdamW (\eipo)
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 6e-5 --threshold 0.1 --ema_decay
    ↪ 0.999 --training_mode parallel --P 21 --optimizer_type adamw
    ↪ --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 6e-5 --threshold 0.1 --ema_decay
    ↪ 0.999 --training_mode parallel --P 14 --optimizer_type adamw
    ↪ --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 6e-5 --threshold 0.1 --ema_decay
    ↪ 0.999 --training_mode parallel --P 7 --optimizer_type adamw
    ↪ --lr_scheduler_type constant
# Serial AdamW Baseline
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 6e-5 --training_mode serial
    ↪ --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 6e-5 --training_mode serial
    ↪ --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 6e-5 --training_mode serial
    ↪ --optimizer_type adamw --lr_scheduler_type constant
```

F.4 Execution Commands of Impact of Batch Size B in Fig. 4

The batch size scalability results in Fig. 4 can be reproduced by the following scripts:

Listing 5: Command for Batch Size (B) Scalability Analysis for Reproducing Results in Fig. 4

```
# Parallel AdamW (\eipo)
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 6e-5 --training_mode parallel --P
    ↪ 21 --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 92 --learning_rate 6e-5 --training_mode parallel --P 21
    ↪ --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 72 --learning_rate 6e-5 --training_mode parallel --P 21
    ↪ --optimizer_type adamw --lr_scheduler_type constant
```

```
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 52 --learning_rate 6e-5 --training_mode parallel --P 21
    ↪ --optimizer_type adamw --lr_scheduler_type constant
# Serial AdamW Baseline
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 6e-5 --training_mode serial
    ↪ --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 92 --learning_rate 6e-5 --training_mode serial
    ↪ --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 72 --learning_rate 6e-5 --training_mode serial
    ↪ --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 52 --learning_rate 6e-5 --training_mode serial
    ↪ --optimizer_type adamw --lr_scheduler_type constant
```

F.5 Execution Commands of Impact of Learning Rate η in Fig. 4

The learning rate scalability results in Fig. 4 can be reproduced by the following scripts:

Listing 6: Command for Learning Rate η Scalability Analysis for Reproducing Results in Fig. 4

```
# Parallel AdamW (\eipo)
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 1e-3 --training_mode parallel --P
    ↪ 21 --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 1e-4 --training_mode parallel --P
    ↪ 21 --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 1e-5 --training_mode parallel --P
    ↪ 21 --optimizer_type adamw --lr_scheduler_type constant

# Serial AdamW Baseline
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 1e-3 --training_mode serial --P 21
    ↪ --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 1e-4 --training_mode serial --P 21
    ↪ --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 1e-5 --training_mode serial --P 21
    ↪ --optimizer_type adamw --lr_scheduler_type constant
```

F.6 Execution Commands of Impact of Damping Factor γ in Fig. 4

The learning rate scalability results in Fig. 4 can be reproduced by the following scripts:

Listing 7: Command for Damping Factor γ Scalability Analysis for Reproducing Results in Fig. 4

```
# Parallel AdamW (\eipo)
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 5000 --batch_size 112 --learning_rate 6e-5 --training_mode parallel --P 7
    ↪ --gamma 0.0 --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 5000 --batch_size 112 --learning_rate 6e-5 --training_mode parallel --P 7
    ↪ --gamma 0.1 --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 5000 --batch_size 112 --learning_rate 6e-5 --training_mode parallel --P 7
    ↪ --gamma 1 --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 5000 --batch_size 112 --learning_rate 6e-5 --training_mode parallel --P 7
    ↪ --gamma 10 --optimizer_type adamw --lr_scheduler_type constant
```

```
# Serial AdamW Baseline
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 5000 --batch_size 112 --learning_rate 6e-5 --training_mode serial --P 7
    ↪ --gamma 0.0 --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 5000 --batch_size 112 --learning_rate 6e-5 --training_mode serial --P 7
    ↪ --gamma 0.1 --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 5000 --batch_size 112 --learning_rate 6e-5 --training_mode serial --P 7
    ↪ --gamma 1 --optimizer_type adamw --lr_scheduler_type constant
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 5000 --batch_size 112 --learning_rate 6e-5 --training_mode serial --P 7
    ↪ --gamma 10 --optimizer_type adamw --lr_scheduler_type constant
```

F.7 Execution Commands of Quantitative Comparison in Table 4

The primary results in Table 4 are derived from the following execution commands.

F.7.1 ResNet-50 Experiments

The commands below train the ResNet-50 model from scratch.

Listing 8: ResNet-50 Serial and Parallel (EIPO) Execution Commands for Reproducing ResNet-50 Results in Table 4

```
# Serial Adam Baseline
python cv.py --dataset cifar10 --model_name resnet50 --max_steps 100000
    ↪ --batch_size 2048 --learning_rate 0.00005 --training_mode serial
    ↪ --optimizer_type adam --lr_scheduler_type cosine --warmup_steps 10000
# Serial AdamW Baseline
python cv.py --dataset cifar10 --model_name resnet50 --max_steps 100000
    ↪ --batch_size 2048 --learning_rate 0.00005 --training_mode serial
    ↪ --optimizer_type adamw --lr_scheduler_type cosine --warmup_steps 10000
# Serial SGD Baseline
python cv.py --dataset cifar10 --model_name resnet50 --max_steps 240000
    ↪ --batch_size 2048 --learning_rate 0.00005 --training_mode serial
    ↪ --optimizer_type sgd --lr_scheduler_type cosine --warmup_steps 10000

# Parallel Adam (\eipo)
python cv.py --dataset cifar10 --model_name resnet50 --max_steps 100000
    ↪ --batch_size 2048 --learning_rate 0.00005 --threshold 0.1 --ema_decay
    ↪ 0.999 --training_mode parallel --gamma 0.01 --P 7 --optimizer_type adam
    ↪ --lr_scheduler_type cosine --warmup_steps 10000
# Parallel AdamW (\eipo)
python cv.py --dataset cifar10 --model_name resnet50 --max_steps 100000
    ↪ --batch_size 2048 --learning_rate 0.00005 --threshold 0.1 --ema_decay
    ↪ 0.999 --training_mode parallel --gamma 0.1 --P 7 --optimizer_type adamw
    ↪ --lr_scheduler_type cosine --warmup_steps 10000
# Parallel SGD (\eipo)
python cv.py --dataset cifar10 --model_name resnet50 --max_steps 240000
    ↪ --batch_size 2048 --learning_rate 0.00005 --threshold 0.1 --ema_decay
    ↪ 0.999 --training_mode parallel --gamma 0.05 --P 7 --optimizer_type sgd
    ↪ --lr_scheduler_type cosine --warmup_steps 10000
```

F.7.2 ViT Experiments

For the ViT model, which is trained from scratch, the commands are as follows.

Listing 9: ViT Serial and Parallel (EIPO) Execution Commands for Reproducing ViT Results in Table 4

```
# Serial Adam Baseline
```

```
python cv.py --dataset cifar10 --model_name resnet50 --max_steps 100000
    ↪ --batch_size 2048 --learning_rate 0.00005 --training_mode serial
    ↪ --optimizer_type adam --lr_scheduler_type cosine --warmup_steps 10000
# Serial AdamW Baseline
python cv.py --dataset cifar10 --model_name resnet50 --max_steps 100000
    ↪ --batch_size 2048 --learning_rate 0.00005 --training_mode serial
    ↪ --optimizer_type adamw --lr_scheduler_type cosine --warmup_steps 10000
# Serial SGD Baseline
python cv.py --dataset cifar10 --model_name resnet50 --max_steps 240000
    ↪ --batch_size 2048 --learning_rate 0.00005 --training_mode serial
    ↪ --optimizer_type sgd --lr_scheduler_type cosine --warmup_steps 10000

# Parallel Adam (\eipo)
python cv.py --dataset cifar10 --model_name resnet50 --max_steps 100000
    ↪ --batch_size 2048 --learning_rate 0.00005 --threshold 0.1 --ema_decay
    ↪ 0.999 --training_mode parallel --P 7 --gamma 0.4--optimizer_type adam
    ↪ --lr_scheduler_type cosine --warmup_steps 10000
# Parallel AdamW (\eipo)
python cv.py --dataset cifar10 --model_name resnet50 --max_steps 100000
    ↪ --batch_size 2048 --learning_rate 0.00005 --threshold 0.1 --ema_decay
    ↪ 0.999 --training_mode parallel --P 7 --gamma 0.2 --optimizer_type adamw
    ↪ --lr_scheduler_type cosine --warmup_steps 10000
# Parallel SGD (\eipo)
python cv.py --dataset cifar10 --model_name resnet50 --max_steps 240000
    ↪ --batch_size 2048 --learning_rate 0.00005 --threshold 0.1 --ema_decay
    ↪ 0.999 --training_mode parallel --gamma 0.03 --P 7 --gamma 0.05
    ↪ --optimizer_type sgd --lr_scheduler_type cosine --warmup_steps 10000
```

F.8 Execution Commands for Reproducing Table 1

The following commands were used to reproduce the quantitative results in Table 1. The experimental environment was synchronized using a fixed seed of 42.

F.8.1 Training Execution

The training scripts for both sequential (Adam) and parallel (**EIPO**) modes are provided below.

Listing 10: Execution scripts for Diffusion Training Tasks in Table 1

```
# Serial Training (Adam Baseline)
python diffusion.py --results_folder diffusion_v2_serial --dataset lsun_church
    ↪ --model_name unet --training_mode serial --lr_scheduler_type cosine
    ↪ --warmup_steps 10000 --save_and_sample_every 10000

# Parallel Training (Adam + \eipo)
python diffusion.py --results_folder diffusion_v2_parallel --dataset lsun_church
    ↪ --model_name unet --threshold 0.000001 --EIPO_ema_decay 0.99
    ↪ --training_mode parallel --P 7 --gamma 0.01 --lr_scheduler_type cosine
    ↪ --warmup_steps 10000 --save_and_sample_every 10000
```

F.8.2 FID Evaluation Execution

After training checkpoints were saved at the specified milestones, FID was calculated using the following distributed sampling script:

Listing 11: FID Evaluation Commands in Table 1

```
# Parallel Mode FID Calculation
python sweep_fid.py --results_folder ./EIPO/diffusion_v2_parallel --data_folder
    ↪ ./dataset/church_outdoor_images --milestones 10000 50000 100000 150000
    ↪ 200000 250000 300000 350000 400000 450000 500000 550000 600000 --csv_name
    ↪ parallel_fid.csv --batch_size 512 --num_samples 50000
```

```
# Serial Mode FID Calculation
python sweep_fid.py --results_folder ./EIPO/diffusion_v2_serial --data_folder
    ↪ ./dataset/church_outdoor_images --milestones 10000 50000 100000 150000
    ↪ 200000 250000 300000 350000 400000 450000 500000 550000 600000 --csv_name
    ↪ serial_fid.csv --batch_size 512 --num_samples 50000
```

F.9 Execution Commands for Data-parallelism Comparision Commands in Table 2

Below we give the detailed running scripts for reproducing the results in Table 2. In this scripts, we include execution commands for comparision between data-parallelism and our step-parallelism.

Listing 12: Step-parallellism v.s. Data-parallelism Comparision Commands in Table 2

```
# Adam Baseline
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 6e-5 --training_mode serial
    ↪ --optimizer_type adam --lr_scheduler_type constant
# Step Parallelism w/o Optimization (\eipo)
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 6e-5 --threshold 0.1 --ema_decay
    ↪ 0.999 --training_mode parallel --P 21 --optimizer_type adam
    ↪ --lr_scheduler_type constant
# Data Parallelism w/o Optimization
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 6e-5 --training_mode naive_dp
    ↪ --optimizer_type adam --lr_scheduler_type constant
# Data Parallelism w/ Full Optimization (Pytorch's DDP)
python llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2" --max_steps
    ↪ 1000 --batch_size 112 --learning_rate 6e-5 --training_mode ddp
    ↪ --optimizer_type adam --lr_scheduler_type constant
# Data Parallelism w/ Full Optimization (DeepSpeed)
deepspeed --num_gpus=8 llm.py --local_model_dir "./gpt2" --modelscope_name "gpt2"
    ↪ --max_steps 1000 --batch_size 112 --learning_rate 6e-5 --training_mode
    ↪ deepspeed --optimizer_type adam --lr_scheduler_type constant --stage 3
```

G Proof Appendix

G.0.1 Proof of Root Uniqueness in Theorem 4.1

Proof. To prove Theorem 4.1, we must demonstrate that the system of Preconditioned Nonlinear Equations (PNEs) defined in Eq. (3) admits exactly one solution trajectory $\{\hat{w}_t\}_{t=0}^{T}$, and that this solution is strictly identical to the sequential gradient descent (GD) trajectory $\{w_t\}_{t=0}^{T}$ generated by Eq. (1).

By definition, the global preconditioning matrix $\mathcal{Z}$ is non-singular (invertible). Therefore, multiplying the PNE system $\mathcal{Z} \cdot \mathcal{H}(\hat{w}_0, \ldots, \hat{w}_T) = 0$ by $\mathcal{Z}^{-1}$ from the left yields:

$$\mathcal{H}(\hat{w}_0, \ldots, \hat{w}_T) = 0. \tag{18}$$

This ensures the preconditioning process neither introduces extraneous roots nor eliminates existing ones, meaning that finding the roots of the PNE system is strictly equivalent to solving for the roots of the piecewise operator $\mathcal{H}$. We proceed by mathematical induction over the time step t.

Base Case ($t = 0$): According to the first condition of the piecewise formulation of $\mathcal{H}$ in Eq. (3), we directly have:

$$\hat{w}_0 - w_0 = 0 \implies \hat{w}_0 = w_0. \tag{19}$$

Thus, for $t = 0$, the root is uniquely determined and perfectly matches the initial weights of the exact GD trajectory.

Inductive Step: Assume that for an arbitrary step $s \in [0, T-1]$, the system admits a unique solution for all steps up to s, and these solutions match the exact GD trajectory. Formally, our inductive hypothesis states:

$$\hat{w}_\tau = w_\tau, \quad \forall \tau \in \{0, 1, \ldots, s\}. \tag{20}$$

We now consider the equation for the subsequent step, $t = s + 1$. From the second condition of $\mathcal{H}$ in Eq. (3), we have:

$$\hat{w}_{s+1} - F_s(\hat{w}_s, \ldots, \hat{w}_0; \zeta_s, \ldots, \zeta_0) = 0. \tag{21}$$

By substituting the inductive hypothesis ($\hat{w}_\tau = w_\tau$ for $\tau \leq s$) into the mapping operator F_s, we obtain:

$$\hat{w}_{s+1} = F_s(w_s, \ldots, w_0; \zeta_s, \ldots, \zeta_0). \tag{22}$$

By the definition of the sequential autoregressive GD procedure in Eq. (1), the right-hand side of Eq. (22) is exactly the formulation for computing w_{s+1}. Therefore, it directly follows that:

$$\hat{w}_{s+1} = w_{s+1}. \tag{23}$$

Because F_s is a deterministic mapping given fixed historical weights $\{w_\tau\}_{\tau=0}^{s}$ and a fixed data batch sequence $\{\zeta_\tau\}_{\tau=0}^{s}$, the value for $\hat{w}_{s+1}$ is uniquely determined.

By the principle of mathematical induction, $\hat{w}_t = w_t$ is the unique solution for all time steps $t \in [0, T]$. Consequently, the root of the PNE system defined in Eq. (3) exists, is entirely unique, and corresponds identically to the exact sequential GD trajectory formulated in Eq. (1). This concludes the proof. □

G.1 Proof of Optimality for Quasi-Newton Preconditioning in Theorem 4.3

Proof. To simplify the notation for this proof, we omit the iteration superscript (k) and the step subscripts $t_1 : t_2$. Let $\Delta\mathcal{X} = \mathcal{X}^{(k)}_{q=1,t_1:t_2}$, $\Delta\mathcal{F} = \mathcal{F}^{(k)}_{q=1,t_1:t_2}$, and $\mathbf{M} = \mathbf{M}_{t_1:t_2}$. We denote the transformation matrix as Q, and the proposed preconditioner as $J = -I + Q$.

The theorem asserts two fundamental properties:

1. J satisfies the generalized inverse multisecant condition: $J\Delta\mathcal{F} = \Delta\mathcal{X}$.
2. The matrix J minimizes the distance to the negative identity matrix in the preconditioned space, meaning $\|J + I\|_{\mathbf{M}^{-1}}$ is minimal among all valid matrices.

Part 1: Proof of the Inverse Multisecant Condition

By the definition provided in Theorem 4.3, we have:

$$Q = (\Delta\mathcal{X} + \Delta\mathcal{F}) \left(\Delta\mathcal{F}^\top \mathbf{M}^{-1} \Delta\mathcal{F}\right)^{-1} \Delta\mathcal{F}^\top \mathbf{M}^{-1} \tag{24}$$

We apply the transformation matrix Q to the residual variation $\Delta\mathcal{F}$:

$$\begin{aligned} Q\Delta\mathcal{F} &= (\Delta\mathcal{X} + \Delta\mathcal{F}) \left(\Delta\mathcal{F}^\top \mathbf{M}^{-1} \Delta\mathcal{F}\right)^{-1} \Delta\mathcal{F}^\top \mathbf{M}^{-1} \Delta\mathcal{F} \\ &= (\Delta\mathcal{X} + \Delta\mathcal{F})\mathbf{I} \\ &= \Delta\mathcal{X} + \Delta\mathcal{F} \end{aligned} \tag{25}$$

Now, substituting Q into the definition of the preconditioner J:

$$J\Delta\mathcal{F} = (-I + Q)\Delta\mathcal{F} = -\Delta\mathcal{F} + Q\Delta\mathcal{F} = -\Delta\mathcal{F} + (\Delta\mathcal{X} + \Delta\mathcal{F}) = \Delta\mathcal{X} \tag{26}$$

Thus, the inverse multisecant condition is strictly satisfied.

Part 2: Proof of Minimality via Constrained Optimization

Here, we prove that the proposed formulation computes the exact minimum-norm correction satisfying the preconditioned inverse secant condition. Note that minimizing $\|J + I\|_{\mathbf{M}^{-1}}$ is equivalent to minimizing the norm of the transformation matrix $\|Q\|_{\mathbf{M}^{-1}}$, since $J + I = Q$.

Step 1. Formulating the Constrained Optimization Problem

Substituting the base fixed-point operator into the secant condition yields $Q\Delta\mathcal{F} = \Delta\mathcal{X} + \Delta\mathcal{F}$. To minimize the perturbation in the curvature-aware space defined by the generalized metric $\mathbf{M}$ (where $\|Q\|^2_{\mathbf{M}^{-1}} = \mathrm{Tr}(Q\mathbf{M}Q^\top)$), we formulate the following constrained optimization problem:

$$\min_Q \frac{1}{2}\mathrm{Tr}\left(Q\mathbf{M}Q^\top\right) \quad \text{s.t.} \quad Q\Delta\mathcal{F} = \Delta\mathcal{X} + \Delta\mathcal{F} \tag{27}$$

Step 2. Solving with Lagrange Multipliers

We define the Lagrangian with a multiplier matrix Λ:

$$\mathcal{L}(Q, \Lambda) = \frac{1}{2}\text{Tr}\left(Q\mathbf{M}Q^\top\right) - \text{Tr}\left(\Lambda^\top(Q\Delta\mathcal{F} - (\Delta\mathcal{X} + \Delta\mathcal{F}))\right) \tag{28}$$

Taking the derivative with respect to Q and equating it to zero yields:

$$\frac{\partial\mathcal{L}}{\partial Q} = Q\mathbf{M} - \Lambda\Delta\mathcal{F}^\top = \mathbf{0} \implies Q = \Lambda\Delta\mathcal{F}^\top\mathbf{M}^{-1} \tag{29}$$

Substituting this expression for Q back into the equality constraint gives:

$$\left(\Lambda\Delta\mathcal{F}^\top\mathbf{M}^{-1}\right)\Delta\mathcal{F} = \Delta\mathcal{X} + \Delta\mathcal{F} \tag{30}$$

$$\Lambda(\Delta\mathcal{F}^\top\mathbf{M}^{-1}\Delta\mathcal{F}) = \Delta\mathcal{X} + \Delta\mathcal{F} \implies \Lambda = (\Delta\mathcal{X} + \Delta\mathcal{F})(\Delta\mathcal{F}^\top\mathbf{M}^{-1}\Delta\mathcal{F})^{-1} \tag{31}$$

Finally, substituting the solved multiplier Λ back into the equation for Q, the analytical solution for the optimal correction matrix Q_* is obtained via the weighted Moore-Penrose pseudoinverse:

$$Q_* = (\Delta\mathcal{X} + \Delta\mathcal{F})(\Delta\mathcal{F}^\top\mathbf{M}^{-1}\Delta\mathcal{F})^{-1}\Delta\mathcal{F}^\top\mathbf{M}^{-1} \tag{32}$$

Since Q_* exactly matches our derived matrix $Q^{(k)}_{t_1:t_2}$, it is proven that the proposed preconditioner $J^{(k)}_{t_1:t_2} = -I + Q^{(k)}_{t_1:t_2}$ strictly minimizes the change from the identity mapping in the preconditioned space while simultaneously fulfilling the generalized inverse multisecant condition. This completes the proof. □

G.2 Proof of Convergence for EIPO with Sliding Window

G.2.1 Notation and Problem Restatement

Basic Notation. We denote the full sequence of weights up to time $T-1$ as $W = [\hat{w}_0, \ldots, \hat{w}_{T-1}]$. The norm $\|\cdot\|_F$ denotes the Frobenius norm. For model weights $w \in \mathbb{R}^n$ with $n > 1$, operations are assumed to be element-wise. The target fixed point $W^{\text{seq}} = [w_0^{\text{seq}}, w_1^{\text{seq}}, \ldots, w_{T-1}^{\text{seq}}]$ is the exact trajectory generated by standard autoregressive gradient descent.

The base update function for a single model state at index t is defined as:

$$F_{t-1}(\hat{w}_0, \cdots, \hat{w}_{t-1}) = \hat{w}_0 - \sum_{\tau=0}^{t-1}\eta_\tau g_\tau(\hat{w}_\tau, \ldots, \hat{w}_0) \tag{33}$$

We denote the prefix sequence up to time τ as $W_{\leq\tau} = [\hat{w}_0, \ldots, \hat{w}_\tau]$.

Block Decomposition & Core Variables. We systematically partition the total sequence into m consecutive functional blocks, $\mathcal{B}_1, \ldots, \mathcal{B}_m$, where $m \in [\lceil T/p \rceil, T]$.

- **Block Indices:** $\mathcal{B}_i = \{t_i, \ldots, t_{i+|\mathcal{B}_i|-1}\}$. The parallel capacity is strictly bounded by the maximum window size: $|\mathcal{B}_i| \leq p$.
- **Block State:** $W_{\mathcal{B}_i} = [\hat{w}_{t_i}, \ldots, \hat{w}_{t_{i+|\mathcal{B}_i|-1}}]$. The exact sequential target for this block is $W^{\text{seq}}_{\mathcal{B}_i}$.
- **Historical Context:** $H_{i-1} = [\hat{w}_0, \ldots, \hat{w}_{t_i-1}]$ is the actual historical input received by block i from previously processed blocks. The ideal sequential history is H^{seq}_{i-1}.
- **Block Operator:** For a fixed history H_{i-1}, the parallel evaluation mapping over the block at iteration j is:

$$F_{\mathcal{B}_i}(W^{(j-1)}_{\mathcal{B}_i} \mid H_{i-1}) = \left[F_{t_i}(\hat{w}_{t_i} \mid H_{i-1}), \ldots, F_{t_{i+|\mathcal{B}_i|-1}}(\hat{w}_{t_i}, \ldots, \hat{w}_{t_{i+|\mathcal{B}_i|-1}} \mid H_{i-1})\right]$$

- **Block Error:** $E^{(j)}_i = \|W^{(j)}_{\mathcal{B}_i} - W^{\text{seq}}_{\mathcal{B}_i}\|_F$.

Preconditioned Update Rule. As described in Algorithm 1, the general update takes the form:

$$\hat{w}^{(k+1)}_{t_1:t_2} = \hat{w}^{(k)}_{t_1:t_2} - \mathcal{Z}^{(k)}R^{(k)}_{t_1:t_2} \tag{34}$$

where $R^{(k)}_{t_1:t_2} = F_{\mathcal{B}_i}(W^{(k)}_{\mathcal{B}_i} \mid H_{i-1}) - W^{(k)}_{\mathcal{B}_i}$ is the residual. The preconditioner $\mathcal{Z}^{(k)}$ admits two instantiations unified by $\gamma \in [0, 1)$:

1. **Identity Preconditioning ($\gamma = 0$):** $\mathcal{Z}^{(k)} = -I$, yielding $\hat{w}_{t_1:t_2}^{(k+1)} = F_{\mathcal{B}_i}(W_{\mathcal{B}_i}^{(k)} \mid H_{i-1})$.
2. **Safeguarded EIPO Preconditioning ($\gamma \in (0,1)$):** $\mathcal{Z}^{(k)} = \bar{J}_{t_1:t_2}^{(k)} = -I + \bar{Q}_{t_1:t_2}^{(k)}$, where $Q_{t_1:t_2}^{(k)}$ is the optimal transformation matrix from Theorem 4.3, constructed from the virtual spatial secant:
$$\mathcal{X}_{q=1,t}^{(k)} = \hat{w}_t^{(k)} - \hat{w}_t^{(k-1)}, \quad \mathcal{F}_{q=1,t}^{(k)} = R_t^{(k)} - \mathbf{D}_t \tag{35}$$
via the closed-form:
$$Q_{t_1:t_2}^{(k)} = (\mathcal{X}_{q=1,t_1:t_2}^{(k)} + \mathcal{F}_{q=1,t_1:t_2}^{(k)}) \left(\mathcal{F}_{q=1,t_1:t_2}^{(k)\top} \mathbf{M}_{t_1:t_2}^{-1} \mathcal{F}_{q=1,t_1:t_2}^{(k)} \right)^{-1} \mathcal{F}_{q=1,t_1:t_2}^{(k)\top} \mathbf{M}_{t_1:t_2}^{-1} \tag{36}$$
with spectral safeguard $\bar{Q}_{t_1:t_2}^{(k)} = \gamma Q_{t_1:t_2}^{(k)} / \max(1, \|Q_{t_1:t_2}^{(k)}\|_2)$ for damping parameter γ, ensuring $\|\bar{Q}_{t_1:t_2}^{(k)}\|_2 \leq \gamma < 1$. The preconditioner is thus $\bar{J}_{t_1:t_2}^{(k)} = -I + \bar{Q}_{t_1:t_2}^{(k)}$, and the update reads:
$$\hat{w}_{t_1:t_2}^{(k+1)} = \hat{w}_{t_1:t_2}^{(k)} + (I - \bar{Q}_{t_1:t_2}^{(k)}) R_{t_1:t_2}^{(k)} \tag{37}$$

To unify the analysis, we define the *effective iteration operator* for block $\mathcal{B}_i$:
$$\Phi_{\mathcal{B}_i}^{(k)}(W_{\mathcal{B}_i}^{(k)} \mid H_{i-1}) \triangleq W_{\mathcal{B}_i}^{(k)} - \mathcal{Z}^{(k)} \left(F_{\mathcal{B}_i}(W_{\mathcal{B}_i}^{(k)} \mid H_{i-1}) - W_{\mathcal{B}_i}^{(k)} \right) \tag{38}$$

G.2.2 Phase 1: Finite Bounds on Iteration Count

Lemma G.1 (Finite Bounds of Termination)**.** *Alg. 1 exactly converges to the target GD trajectory at $K \leq T$ steps.*

Worst-Case Scenario Analysis. The structure of the preconditioned update imposes a causal dependency: the calculation of the residual for $\hat{w}_t^{(k+1)}$ depends only on the components $\hat{w}_0^{(k)}, \ldots, \hat{w}_{t-1}^{(k)}$ from the previous iteration. Consequently, convergence cannot occur "out of order". The component $\hat{w}_1$ can only securely converge after $\hat{w}_0$ has, $\hat{w}_2$ after $\hat{w}_0$ and $\hat{w}_1$, and so on. The worst-case scenario occurs when the sliding window only advances by the minimum possible stride ($s_k = 1$) at each step, sequentially "locking-in" the correct values one component at a time, strictly mimicking the autoregressive gradient descent.

Proof. We proceed by induction on the iteration index k to prove the proposition $P(k)$, which asserts that the first k model weights strictly converge to their sequential targets after k iterations, and consequently, the starting index of the active window for the next iteration satisfies $t_1^{(k+1)} \geq k$. Formally, the convergence state is defined as:
$$P(k): \quad \hat{w}_\tau^{(k)} = w_\tau^{\text{seq}}, \quad \forall \tau \in \{0, \ldots, k-1\}.$$

Base Case ($k = 1$): We must prove $P(1)$: $\hat{w}_0^{(1)} = w_0^{\text{seq}}$ and $t_1^{(2)} \geq 1$. At iteration $k = 1$, the algorithm initializes the first window starting at $t_1^{(1)} = 0$. The update for the initial weight $\hat{w}_0$ requires no prior history. By the definition of the base operator, F_{-1} yields exactly the initial condition w_0^{seq}. Thus, applying the update rule yields exactly w_0^{seq}. Since the convergence distance d_0 drops to $0 \leq \delta$, the component $\hat{w}_0$ is strictly converged. This guarantees the window will slide forward by at least one position, yielding $t_1^{(2)} \geq 1$. The base case holds.

Inductive Hypothesis: Assume for some $k \geq 1$ that $P(k)$ holds. This means that after k iterations, the prefix up to index $k-1$ has converged:
$$\hat{w}_\tau^{(k)} = w_\tau^{\text{seq}}, \quad \forall \tau \in \{0, \ldots, k-1\},$$
and the sliding window for the next iteration starts at $t^* \triangleq t_1^{(k+1)} \geq k$.

Inductive Step: We must prove that $P(k+1)$ holds: after $k+1$ iterations, the prefix up to index k has converged, meaning $\hat{w}_\tau^{(k+1)} = w_\tau^{\text{seq}}$ for all $0 \leq \tau \leq k$, and $t_1^{(k+2)} \geq k+1$.

At iteration $k+1$, the active window begins at index $t^* \geq k$. The evaluation of the first component in this window, $\hat{w}_{t^*}^{(k+1)}$, relies on computing the residual $R_{t^*}^{(k+1)}$, which requires the historical prefix

$\hat{w}_0^{(k)}, \ldots, \hat{w}_{t^*-1}^{(k)}$. According to our inductive hypothesis, since $t^* \geq k$, all components in this requisite prefix have already stabilized to their exact sequential targets:

$$\hat{w}_\tau^{(k)} = w_\tau^{\text{seq}}, \quad \forall \tau \in \{0, \ldots, t^* - 1\}.$$

This demonstrates that all inputs required to compute the operator for $\hat{w}_{t^*}$ have already reached their exact fixed-point values. Substituting these converged values back into the operator gives:

$$F_{t^*-1}(\hat{w}_0^{(k)}, \ldots, \hat{w}_{t^*-1}^{(k)}) = F_{t^*-1}(w_0^{\text{seq}}, \ldots, w_{t^*-1}^{\text{seq}}) = w_{t^*}^{\text{seq}}.$$

Applying the update rule using this exact base operator output forces $\hat{w}_{t^*}^{(k+1)}$ to exactly equal $w_{t^*}^{\text{seq}}$. Its convergence distance d_{t^*} drops to exactly 0. Since the tolerance condition $0 \leq \delta$ is satisfied, this component is confirmed converged. Because the first component of the active window has securely converged, the algorithm computes a new sliding stride $s_{k+1} \geq 1$, sliding the window forward past t^*. The new starting index becomes:

$$t_1^{(k+2)} \geq t^* + 1.$$

Since $t^* \geq k$ from the inductive hypothesis, we obtain:

$$t_1^{(k+2)} \geq k + 1.$$

This proves that the prefix up to at least index k has exactly converged, establishing $P(k+1)$ and completing the inductive step.

Conclusion on Iteration Count. By induction, we have shown that $P(k)$ holds for all $k \geq 1$. This means that after k iterations, the first k model weights $\hat{w}_{0:k-1}^{(k)}$ must strictly equal the sequential ones $w_{0:k-1}^{\text{seq}}$. For the entire sequence of length T to converge, we evaluate our result at $k = T$:

$$\hat{w}_\tau^{(T)} = w_\tau^{\text{seq}}, \quad \forall \tau \in \{0, \ldots, T - 1\}.$$

At iteration $k = T$, we have $t_1^{(T+1)} \geq T$, meaning the active window has slid past the entire sequence length T. Thus, the entire trajectory vector has converged:

$$W^{(T)} = W^{\text{seq}}.$$

Therefore, it is proven that even in the worst-case scenario where the window advances by only 1 step per iteration, exact convergence happens in $K \leq T$ iterations. □

G.2.3 Phase 2: Block-wise Contraction Mapping

Lemma G.2 (Block-wise Contraction under General Preconditioning). *For any functional block $\mathcal{B}_i$ and fixed historical input H_{i-1}, the effective iteration operator $\Phi_{\mathcal{B}_i}^{(k)}(\cdot \mid H_{i-1})$ is a contraction mapping. The global contraction rate ρ_1 unified by $\gamma \in [0, 1)$ satisfies $\rho_1 = (1+\gamma)\sqrt{p}\max_i \sum_{\tau \in \mathcal{B}_i} \eta_\tau C + \gamma$. Convergence requires $\rho_1 < 1$.*
If the learning rate satisfies $\rho_1 < 1$, then for any two distinct block states W and V:

$$\|\Phi_{\mathcal{B}_i}^{(k)}(W \mid H_{i-1}) - \Phi_{\mathcal{B}_i}^{(k)}(V \mid H_{i-1})\|_F \leq \rho_1 \|W_{\mathcal{B}_i} - V_{\mathcal{B}_i}\|_F \tag{39}$$

Proof. **Case 1: Identity Preconditioning ($\mathcal{Z}^{(k)} = -I, \gamma = 0$).**

When $\mathcal{Z}^{(k)} = -I$, the effective operator reduces to $\Phi_{\mathcal{B}_i}^{(k)} = F_{\mathcal{B}_i}$. Let W and V be two distinct sequence states for block $\mathcal{B}_i$. Since history H_{i-1} is identical, gradient updates prior to index t_i perfectly cancel out. For any specific index $t \in \mathcal{B}_i$:

$$F_t(W \mid H_{i-1}) - F_t(V \mid H_{i-1}) = -\sum_{\tau=t_i}^{t} \eta_\tau \left(g_\tau(W) - g_\tau(V)\right) \tag{40}$$

By Assumption 4.5, since W and V share the identical history prefix, the gradient difference is strictly bounded by their distance across the current block:

$$\|g_\tau(W) - g_\tau(V)\|_F \leq C\|W_{\leq \tau} - V_{\leq \tau}\|_F \leq C\|W_{\mathcal{B}_i} - V_{\mathcal{B}_i}\|_F \tag{41}$$

Applying the triangle inequality and substituting:

$$\|F_t(W \mid H_{i-1}) - F_t(V \mid H_{i-1})\|_F \le \left(\sum_{\tau=t_i}^{t} \eta_\tau C\right) \|W_{\mathcal{B}_i} - V_{\mathcal{B}_i}\|_F \tag{42}$$

Aggregating over the block (bounded by capacity p) via the Frobenius norm:

$$\begin{aligned}\|F_{\mathcal{B}_i}(W \mid H_{i-1}) - F_{\mathcal{B}_i}(V \mid H_{i-1})\|_F &= \sqrt{\sum_{t\in\mathcal{B}_i} \|F_t(W \mid H_{i-1}) - F_t(V \mid H_{i-1})\|_F^2} \\ &\le \left(\sqrt{p}\sum_{\tau\in\mathcal{B}_i} \eta_\tau C\right) \|W_{\mathcal{B}_i} - V_{\mathcal{B}_i}\|_F \end{aligned} \tag{43}$$

Setting $\gamma = 0$ directly establishes the rate $\rho_1 = \sqrt{p}\max_i \sum_{\tau\in\mathcal{B}_i} \eta_\tau C$.

Case 2: Safeguarded EIPO Preconditioning ($\mathcal{Z}^{(k)} = -I + \bar{Q}^{(k)}_{t_1:t_2}$, $\gamma \in (0,1)$).

The effective operator reads:

$$\Phi^{(k)}_{\mathcal{B}_i}(W \mid H_{i-1}) = W_{\mathcal{B}_i} - (-I + \bar{Q}^{(k)}_{t_1:t_2})\,(F_{\mathcal{B}_i}(W \mid H_{i-1}) - W_{\mathcal{B}_i}) = F_{\mathcal{B}_i}(W \mid H_{i-1}) - \bar{Q}^{(k)}_{t_1:t_2} R^{(k)}_{\mathcal{B}_i} \tag{44}$$

where $R^{(k)}_{\mathcal{B}_i} = F_{\mathcal{B}_i}(W \mid H_{i-1}) - W_{\mathcal{B}_i}$. For two block states W and V with identical fixed history H_{i-1}, denote $\Delta W = W_{\mathcal{B}_i} - V_{\mathcal{B}_i}$, $\Delta F = F_{\mathcal{B}_i}(W \mid H_{i-1}) - F_{\mathcal{B}_i}(V \mid H_{i-1})$, and $\Delta R = \Delta F - \Delta W$. Then:

$$\Phi^{(k)}_{\mathcal{B}_i}(W \mid H_{i-1}) - \Phi^{(k)}_{\mathcal{B}_i}(V \mid H_{i-1}) = \Delta F - \bar{Q}^{(k)}_{t_1:t_2}\Delta R = \Delta F - \bar{Q}^{(k)}_{t_1:t_2}(\Delta F - \Delta W) \tag{45}$$

The spectral safeguard ensures $\|\bar{Q}^{(k)}_{t_1:t_2}\|_2 \le \gamma < 1$.

Using $\|\Delta F\|_F \le (\sqrt{p}\max_i \sum_{\tau\in\mathcal{B}_i} \eta_\tau C)\|\Delta W\|_F$ from Case 1, and noting that $\|\Delta R\|_F = \|\Delta F - \Delta W\|_F \le \|\Delta F\|_F + \|\Delta W\|_F \le (1 + \sqrt{p}\max_i \sum \eta_\tau C)\|\Delta W\|_F$:

$$\begin{aligned}\|\Phi^{(k)}_{\mathcal{B}_i}(W \mid H_{i-1}) - \Phi^{(k)}_{\mathcal{B}_i}(V \mid H_{i-1})\|_F &= \|\Delta F - \bar{Q}^{(k)}_{t_1:t_2}\Delta R\|_F \\ &\le \|\Delta F\|_F + \|\bar{Q}^{(k)}_{t_1:t_2}\|_2 \cdot \|\Delta R\|_F \\ &\le \left(\sqrt{p}\sum \eta_\tau C\right)\|\Delta W\|_F + \gamma\,\left(1 + \sqrt{p}\sum \eta_\tau C\right)\|\Delta W\|_F \\ &= \left((1+\gamma)\sqrt{p}\sum \eta_\tau C + \gamma\right)\|\Delta W\|_F \end{aligned} \tag{46}$$

Taking the maximum over all blocks yields the unified contraction rate $\rho_1 = (1+\gamma)\sqrt{p}\max_i \sum_{\tau\in\mathcal{B}_i} \eta_\tau C + \gamma$.

Local superlinear behavior. In a neighborhood of W^{seq} where $F_{\mathcal{B}_i}(W \mid H_{i-1}) \approx W^{\mathrm{seq}} + A_F(W - W^{\mathrm{seq}})$ with $A_F = \nabla_W F_{\mathcal{B}_i}(W^{\mathrm{seq}} \mid H_{i-1})$, the update $\Phi(W) = F(W) - \bar{Q}R$ yields the linearized error propagation:

$$W^{(k+1)}_{\mathcal{B}_i} - W^{\mathrm{seq}}_{\mathcal{B}_i} \approx \left(A_F + \bar{Q}^{(k)}_{t_1:t_2}(I - A_F)\right)(W^{(k)}_{\mathcal{B}_i} - W^{\mathrm{seq}}_{\mathcal{B}_i}) \tag{47}$$

By the inverse multisecant condition, $J = -I + Q$ satisfies $J(A_F - I) \approx I$ along the secant direction, which gives $Q(A_F - I) \approx A_F$, i.e., $Q(I - A_F) \approx -A_F$. Therefore, $A_F + \bar{Q}(I - A_F) \to 0$ as the secant approximation improves, yielding locally superlinear convergence. This robust behavior is fundamentally unattainable by identity preconditioning. □

G.2.4 Phase 3: Global Convergence Error Bound

Proof of Theorem 4.7. The proof proceeds in four steps, unified for $\gamma \in [0, 1)$.

Step 1: Single Iteration Recurrence & Historical Expansion. Consider block i at iteration $j \ge 1$. The preconditioned update gives:

$$W^{(j)}_{\mathcal{B}_i} = \Phi^{(j-1)}_{\mathcal{B}_i}(W^{(j-1)}_{\mathcal{B}_i} \mid H_{i-1}) \tag{48}$$

Bounding the error via the triangle inequality with a cross-term:

$$\begin{aligned} E_i^{(j)} &= \|W_{\mathcal{B}_i}^{(j)} - W_{\mathcal{B}_i}^{\text{seq}}\|_F \\ &\leq \underbrace{\|\Phi_{\mathcal{B}_i}^{(j-1)}(W_{\mathcal{B}_i}^{(j-1)} \mid H_{i-1}) - \Phi_{\mathcal{B}_i}^{(j-1)}(W_{\mathcal{B}_i}^{\text{seq}} \mid H_{i-1})\|_F}_{\text{Term (A): Internal Contraction}} \\ &\quad + \underbrace{\|\Phi_{\mathcal{B}_i}^{(j-1)}(W_{\mathcal{B}_i}^{\text{seq}} \mid H_{i-1}) - W_{\mathcal{B}_i}^{\text{seq}}\|_F}_{\text{Term (B): Historical Error Propagation}} \end{aligned} \tag{49}$$

Bounding Term (A). By Lemma G.2:

$$\text{Term (A)} \leq \rho_1 \cdot E_i^{(j-1)} \tag{50}$$

Bounding Term (B). Since $W_{\mathcal{B}_i}^{\text{seq}}$ is the exact sequential solution, $F_{\mathcal{B}_i}(W_{\mathcal{B}_i}^{\text{seq}} \mid H_{i-1}^{\text{seq}}) = W_{\mathcal{B}_i}^{\text{seq}}$. Thus:

$$\Phi_{\mathcal{B}_i}^{(j-1)}(W_{\mathcal{B}_i}^{\text{seq}} \mid H_{i-1}) = W_{\mathcal{B}_i}^{\text{seq}} - \mathcal{Z}^{(j-1)}\left(F_{\mathcal{B}_i}(W_{\mathcal{B}_i}^{\text{seq}} \mid H_{i-1}) - W_{\mathcal{B}_i}^{\text{seq}}\right) \tag{51}$$

so:

$$\begin{aligned} \text{Term (B)} &= \|\mathcal{Z}^{(j-1)}\left(F_{\mathcal{B}_i}(W_{\mathcal{B}_i}^{\text{seq}} \mid H_{i-1}) - F_{\mathcal{B}_i}(W_{\mathcal{B}_i}^{\text{seq}} \mid H_{i-1}^{\text{seq}})\right)\|_F \\ &\leq \|\mathcal{Z}^{(j-1)}\|_2 \cdot \|F_{\mathcal{B}_i}(W_{\mathcal{B}_i}^{\text{seq}} \mid H_{i-1}) - F_{\mathcal{B}_i}(W_{\mathcal{B}_i}^{\text{seq}} \mid H_{i-1}^{\text{seq}})\|_F \end{aligned} \tag{52}$$

For the base operator difference, internal gradient contributions ($\tau \geq t_i$) cancel since the block states $W_{\mathcal{B}_i}^{\text{seq}}$ are identical. The residual difference is driven purely by history $\tau \in H_{i-1}$:

$$\|F_t(\cdots \mid H_{i-1}) - F_t(\cdots \mid H_{i-1}^{\text{seq}})\|_F \leq \left(\sum_{\tau \in H_{i-1}} \eta_\tau C\right) \|H_{i-1} - H_{i-1}^{\text{seq}}\|_F \tag{53}$$

Aggregating over the block ($|\mathcal{B}_i| \leq p$):

$$\|F_{\mathcal{B}_i}(W_{\mathcal{B}_i}^{\text{seq}} \mid H_{i-1}) - F_{\mathcal{B}_i}(W_{\mathcal{B}_i}^{\text{seq}} \mid H_{i-1}^{\text{seq}})\|_F \leq \sqrt{p}\left(\sum_{\tau \in H_{i-1}} \eta_\tau C\right) \|H_{i-1} - H_{i-1}^{\text{seq}}\|_F \tag{54}$$

With $\|\mathcal{Z}^{(j-1)}\|_2 \leq 1 + \gamma$, we define the unified historical propagation rate $\rho_2 = (1 + \gamma)\sqrt{p} \max_i \sum_{\tau \in H_{i-1}} \eta_\tau C$, and $\delta_{hist} = \rho_2 \|H_{i-1} - H_{i-1}^{\text{seq}}\|_F$. The single iteration recurrence is:

$$E_i^{(j)} \leq \rho_1 \cdot E_i^{(j-1)} + \delta_{hist} \tag{55}$$

Step 2: Unrolling Iterations within the Block. For block i processing over K/m iterations, unrolling the linear recurrence yields:

$$E_i \leq \rho_1^{\frac{K}{m}} E_i^{(0)} + \frac{1 - \rho_1^{\frac{K}{m}}}{1 - \rho_1} \delta_{hist} \tag{56}$$

The initial error is bounded by D (Assumption 4.6). The historical delta satisfies $\delta_{hist} \leq \rho_2 \sum_{k=1}^{i-1} E_k$, establishing:

$$E_i \leq \rho_1^{\frac{K}{m}} D + \frac{\rho_2(1 - \rho_1^{\frac{K}{m}})}{1 - \rho_1} \sum_{k=1}^{i-1} E_k \tag{57}$$

Step 3: Solving the Global Propagation to the Terminal Block. Let $A = \rho_1^{\frac{K}{m}} D$ and $B = \frac{\rho_2(1 - \rho_1^{\frac{K}{m}})}{1 - \rho_1}$, so $E_i \leq A + B \sum_{k=1}^{i-1} E_k$.

We prove by strong induction that $E_i \leq A(1 + B)^{i-1}$.

Base Case ($i = 1$): $E_1 \leq A = A(1 + B)^0$.

Inductive Step: Assume $E_k \leq A(1+B)^{k-1}$ for all $k \in [1, i-1]$. Then:

$$E_i \leq A + B\sum_{k=1}^{i-1} A(1+B)^{k-1} = A + BA \cdot \frac{(1+B)^{i-1}-1}{B} = A(1+B)^{i-1} \tag{58}$$

Evaluating at the terminal block $i = m$ and substituting back:

$$\|W_{\mathcal{B}_m}^{(K/m)} - W_{\mathcal{B}_m}^{\text{seq}}\|_F \leq D \cdot \rho_1^{\frac{K}{m}} \cdot \left(1 + \frac{\rho_2(1-\rho_1^{\frac{K}{m}})}{1-\rho_1}\right)^{m-1} \tag{59}$$

Step 4: Conditions for Parallel Speedup ($K < T$). Since $\rho_1 \in (0,1)$, the exponential decay $\rho_1^{K/m}$ dominates the history aggregation as K grows. Achieving ϵ-precision in $K < T$ iterations requires:

$$D \cdot \rho_1^{\frac{T}{m}} \cdot \left(1 + \frac{\rho_2(1-\rho_1^{\frac{T}{m}})}{1-\rho_1}\right)^{m-1} < \epsilon \tag{60}$$

□

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly outline the claims (the EIPO method, mathematical formulation, memory efficiency, and empirical speedups), which are well-supported by the theoretical proofs in Section 4 and empirical results in Section 5.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The authors explicitly discuss the limitations of their current academic prototype, such as communication overhead and the lack of fused kernels, in Section 5.3 (Limitations and Discussion).

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: All theoretical claims (Theorems 1, 2, and 3) explicitly state their assumptions (e.g., Assumptions 1 and 2 in Section 4.3), and the complete mathematical proofs are provided in Appendix A.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper provides comprehensive details, including hyperparameters, model configurations, and complete execution command scripts to reproduce the experiments in the Appendix (Sections A, B, C, D, and E).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The source code is provided via an anonymous link in the abstract, and the datasets used (WikiText-2, CIFAR-10, LSUN-Church) are publicly available standard benchmarks.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Extensive experimental settings, including batch sizes, learning rates, schedules, and window sizes, are documented in Section 5.1 and fully expanded in Appendices A, B, C, and D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The quantitative results in Table 1 and Table 2 report the means and standard deviations calculated over 5 runs with different random seeds.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The compute resources (e.g., 8 NVIDIA A100 80GB GPUs) and the exact execution times and peak memory are explicitly stated in Section 5.1, Tables 1-4, and Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The paper develops a general mathematical optimization algorithm and safely conforms to the standard NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [No]

Justification: The paper focuses strictly on the foundational algorithmic improvement of optimization trajectories (a solver mechanism) and does not present a direct, unique path to negative societal applications beyond general deep learning capabilities.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper proposes a generic mathematical optimization algorithm and does not release high-risk models or scraped datasets that require specific safeguards.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper utilizes standard open-source datasets (WikiText-2, CIFAR-10, LSUN Church) and models (GPT-2, Llama, Qwen, etc.), citing the original benchmark papers and creators appropriately.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The authors provide the source code for the proposed EIPO algorithm via an anonymous repository link in the abstract.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The research develops an optimization algorithm and does not involve crowdsourcing or human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The research does not involve human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs were not used as a core, original, or non-standard component of the algorithmic method development in this research.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.