Milo: A Magnitude-Invariant Learning Optimizer with Group-Wise Gradient Normalization

Anonymous Author(s)
Affiliation
Address
email

Abstract

We introduce *Milo* (Magnitude-Invariant Learning Optimizer), a stochastic optimizer that partitions gradients into network-wise or layer-wise groups and normalizes each group to decouple update magnitudes from raw gradient norms. By preventing overly aggressive steps in high-gradient regions and amplifying small gradients, Milo produces smoother loss trajectories, more stable training, and better generalization, while dramatically reducing sensitivity to the learning rate. An optional scale-aware blending step preserves useful raw-gradient information alongside invariance. We prove sublinear convergence in both smooth convex and non-convex settings. Empirically, on synthetic loss landscapes, logistic regression, multilayer perceptrons, ResNet-style CNNs, reinforcement-learning benchmarks, and large-language-model pertaining, Milo matches or exceeds modern optimizers in convergence speed and stability, cuts learning-rate tuning effort by up to 100 %, and incurs only minimal runtime overhead. Our code is available at <URL>.

1 Introduction

Optimization is central to any decision-making problem, particularly modern-day machine learning (ML) algorithms. Stochastic gradient descent (SGD) with momentum [Polyak, 1964, Sutskever et al., 2013] and adaptive methods like AdamW [Loshchilov and Hutter, 2019, Kingma and Ba, 2015] have become the most popular optimizers, with SGD performing exceedingly well in computer vision [Goodfellow et al., 2016], and AdamW leading breakthroughs in Natural Language Processing (NLP), large-scale transformers, and reinforcement learning [Vaswani et al., 2017, Haarnoja et al., 2018]. Yet despite their widespread prevalence, both optimizers exhibit weaknesses.

SGD's effectiveness is highly contingent on initial learning rates and weight initializations, undermining its robustness across diverse tasks and architectures. Likewise, AdamW's use of exponential moving averages of squared gradients can produce unstable second-moment estimates, either vanishing or exploding, which, while accelerating early convergence, often degrade ultimate generalization performance. Due to these issues, in practical deep-learning workflows, substantial disparities in gradient magnitudes across layers necessitate extensive per-layer hyperparameter searches, bespoke (custom-made) learning-rate schedules, or reliance on gradient clipping and rescaling heuristics (practical techniques).

These shortcomings lengthen hyperparameter tuning cycles, weaken training pipelines, and impede reproducibility. For example, when training ResNet-50 on ImageNet, gradient norms in early convolutional layers may be an order of magnitude smaller than those in the final classifier, causing negligible updates in the former while the latter shifts erratically unless learning rates are calibrated at each layer. In multi-task architectures, excessively large gradients from one branch can dominate shared backbone updates, stalling progress on other tasks entirely.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

Contributions. We aim to develop an optimizer that seamlessly handles gradient-scale heterogeneity (when gradient sizes vary, often by orders of magnitude, across different layers or parameters), combining the stability and generalization of SGD with the adaptability of AdamW, yet requiring minimal tuning.

To achieve this, we propose *Milo*, a magnitude-invariant learning optimizer that partitions gradients into network-wise (Milo) or layer-wise (Milo$_{LW}$) groups and normalizes each group before any momentum or adaptive scaling, in contrast to methods like NovoGrad [Ginsburg et al., 2019] which normalize moment estimates only after accumulation. We further introduce optional scale-aware blending to preserve useful raw-gradient information alongside invariance. Theoretically, we establish rigorous convergence guarantees in convex settings, with an $O(1/\sqrt{T})$ regret bound, and in non-convex settings, guarantees $\min_{t<T} \|\nabla F(w_t)\|^2 = O\big(T^{-1/2}\big)$, meaning sublinear convergence to a first-order stationary point. Empirically, we validate Milo on six diverse tasks (synthetic landscapes, logistic regression, multilayer perceptrons (MLPs), Residual Neural Network with 18 layers (ResNet-18), reinforcement learning (RL), and large language model (LLM) pre-training), each averaged over five seeds with ablations on grouping size and blending. To ensure reproducibility, we provide the repository containing our testing, self-contained pseudocode, a comprehensive nomenclature, and details for hyperparameters, data-splits, and hardware. Across a wide range of benchmarks, Milo matches or exceeds the performance of modern optimizers, reduces learning-rate tuning effort by up to 100%, and adding minimal runtime overhead, making it a practical plug-and-play optimizer for modern deep-learning workflows.

Table 1: Averaged Train Loss, Test F1 Score, and Runtime by Optimizer and Model

Optimizer	**Logistic**			**MLP**			**ResNet-18**		
	Loss	F1	Time (s)	Loss	F1	Time (s)	Loss	F1	Time (s)
Milo	0.271	0.913	32.5	**0.004**	0.973	34.4	0.637	**0.495**	148.5
Milo$_{LW}$	**0.226**	**0.919**	32.7	0.020	0.969	36.1	1.419	0.417	182.8
SGD	0.277	0.915	**29.7**	0.073	0.969	**30.3**	0.572	0.448	**130.6**
AdamW	0.248	0.916	30.0	0.015	**0.975**	30.9	**0.343**	0.416	131.1
AdaGrad	0.468	0.879	29.8	0.196	0.942	30.7	0.468	0.433	130.7
NovoGrad	0.281	0.914	32.2	0.042	0.973	34.2	0.989	0.487	172.3

2 Related Work

SGD and Momentum-Based Methods. SGD [Robbins and Monro, 1951] iteratively updates parameters using noisy mini-batch gradients. While computationally efficient, it is sensitive to learning rates, can stall in saddle points, and may get trapped in local minima, although it often generalizes well [Ruder, 2016, Keskar et al., 2017]. In real-world training, this can manifest as extremely slow progress on deep models—e.g., early convolutional layers in ResNets barely update unless one carefully hand-tunes per-layer learning rates. To counter its slow convergence, momentum methods were introduced. These incorporate an exponential moving average of past gradients to smooth updates and accelerate convergence. Nesterov's Accelerated Gradient (NAG) improves this further by computing the gradient at a look-ahead position [Nesterov, 1983]. Techniques like gradient normalization and centralization also help improve stability and convergence by recentring or scaling gradients across layers, but they still require careful hyperparameter choices to avoid under- or over-shooting [Yong et al., 2020, Xu et al., 2019].

Adaptive Optimizers. Adaptive optimizers adjust the learning rate on a per-parameter basis, enhancing training stability and convergence [Duchi et al., 2011]. Early methods such as AdaGrad and RMSProp assign individual learning rates using past gradients [Duchi et al., 2011, Tieleman and Hinton, 2012], though they can overly reduce the effective rate in later stages of training, leading to vanishing updates on rarely-seen features [Wilson et al., 2017]. Adam [Kingma and Ba, 2015] combines momentum with adaptive scaling (tracking both first and second moments), making it less sensitive to hyperparameter tuning and widely popular in NLP and RL. However, Adam's independent per-coordinate normalization can cause certain parameters to "overshoot" or overfit, especially in convolutional or recurrent architectures where gradients exhibit strong correlations [Heusel et al., 2017]. AdamW [Loshchilov and Hutter, 2019] improves on this by decoupling weight decay, enhanc-

ing generalization [Hettinger et al., 2022], but still inherits Adam's sensitivity to extreme gradient magnitudes. NovoGrad [Ginsburg et al., 2019] builds on these ideas by applying layer-wise gradient normalization to the moment estimates, yielding stable, balanced updates even in high-variance or large-batch settings. Yet in practice it can still require manual tuning of subgroup sizes to avoid under-attenuation of noisy gradients.

3 Algorithm

Algorithm 1 Milo with median/MAD fallback and per-layer LR multipliers

Require: $\{p_i\}, \{g_i\}, \text{layer}(\cdot), \eta, \{\mu_\ell\}, M, S, \alpha, \varepsilon$

for each layer ℓ **do** ▷ process one layer at a time
  $v \leftarrow [\text{vec}(g_i) \mid \text{layer}(p_i) = \ell]$ ▷ collect & flatten layer gradients
  $m \leftarrow \max\big(1, \lceil |v|/M \rceil\big),\ s \leftarrow \lceil |v|/m \rceil$ ▷ compute #subgroups m and size s
  for $j = 0 \dots m-1$ **do** ▷ normalize subgroup j
    $v_j \leftarrow v[j\,s : \min((j+1)\,s, |v|)]$ ▷ extract segment
    if $\max|v_j| > 10^4$ or $\min|v_j| < 10^{-6}$ **then** ▷ detect outliers
      $\mu_j \leftarrow \text{median}(v_j)$ ▷ robust center
      $d_j \leftarrow \text{median}(|v_j - \mu_j|)$ ▷ MAD measure
      $\sigma_j \leftarrow 1.4826\, d_j + \varepsilon$ ▷ scale s̃td
    else
      $\mu_j \leftarrow \text{mean}(v_j),\ \sigma_j \leftarrow \sqrt{\text{mean}((v_j - \mu_j)^2)} + \varepsilon$ ▷ standard mean/std
    end if
    $v_j \leftarrow (1 - S)\,\frac{v_j - \mu_j}{\sigma_j} \ + \ S\, v_j$ ▷ blend raw & normalized
  end for
  write back normalized v into $\{g_i\}$ ▷ update gradients in place
end for
for each parameter p_i **do** ▷ apply per-layer-scaled update
  $\ell \leftarrow \text{layer}(p_i)$
  $\gamma_i \leftarrow \eta \times (\mu_\ell \text{ if set else } 1)$ ▷ lookup layer LR multiplier
  $p_i \leftarrow p_i - \gamma_i\, g_i$ ▷ SGD-style update
end for

3.1 Defining Milo

The key novelty of Milo is to *group gradients before normalization* and then apply per-layer learning-rate multipliers. Instead of normalizing the entire gradient or each coordinate individually, Milo partitions gradients into meaningful groups, normalizes each group independently (with an automatic median/median absolute deviation(MAD) fallback when needed), and finally scales each layer's update by a user-specified multiplier. This yields balanced, outlier-resilient updates with minimal tuning.

Optionally, gradients may first be clipped:

$$g \leftarrow \frac{g}{\max\big(1, \|g\|_2/\text{clip\_norm}\big)} \quad \text{(gradient clipping)}. \tag{3.1}$$

Given a parameter tensor p with raw gradient $g = \nabla f(p)$:

1. **Grouping.**
   - *Layer-Wise (if enabled):*

$$\mathcal{G}_\ell = \{\, g_i \mid \text{layer}(p_i) = \ell \} \quad \text{(grouping by layer)}. \tag{3.2}$$

   - *Network-Wise:*

$$G = \text{reshape}(g, (-1, s)), \quad s = \max\big(1, \lfloor \sqrt{N} \rfloor\big) \quad \text{(flatten and chunk globally)}. \tag{3.3}$$

2. **Group Normalization with Robust Fallback.** For each group G_k, compute mean and variance:

$$\mu_k = \frac{1}{|G_k|} \sum_{i \in G_k} g_i, \quad \sigma_k = \sqrt{\frac{1}{|G_k|} \sum_{i \in G_k} (g_i - \mu_k)^2 + \epsilon}. \tag{3.4}$$

If $\max_i |g_i| > 10^4$ or $\sigma_k < 10^{-6}$, use MAD:

$$\mu_k \leftarrow \text{median}(G_k), \quad \sigma_k \leftarrow 1.4826 \times \text{median}(|G_k - \mu_k|) + \epsilon. \tag{3.5}$$

Then normalize:

$$\tilde{g}_i = \frac{g_i - \mu_k}{\sigma_k} \quad \forall i \in G_k. \tag{3.6}$$

3. **Scale-Aware Blending (optional).**

$$\tilde{g}_i \leftarrow (1 - \alpha)\, \tilde{g}_i + \alpha\, g_i, \quad \alpha \in [0, 1] \quad \text{(preserve raw gradient structure)}. \tag{3.7}$$

4. **Weight Decay.**

$$\tilde{g}_i \leftarrow \tilde{g}_i + \lambda\, p_i \quad \text{(L2 regularization)}. \tag{3.8}$$

5. **Momentum & Adaptive Scaling.**

$$m_t = \beta\, m_{t-1} + \tilde{g} \quad \text{(momentum accumulation)}, \tag{3.9}$$

$$\hat{m}_t = \begin{cases} \dfrac{m_t}{\sqrt{\sum_{\tau=0}^{t} m_\tau^2} + \text{adaptive\_eps}}, & \text{if adaptive = True,} \\ m_t, & \text{otherwise} \end{cases} \quad \text{(adaptive RMS scaling)}. \tag{3.10}$$

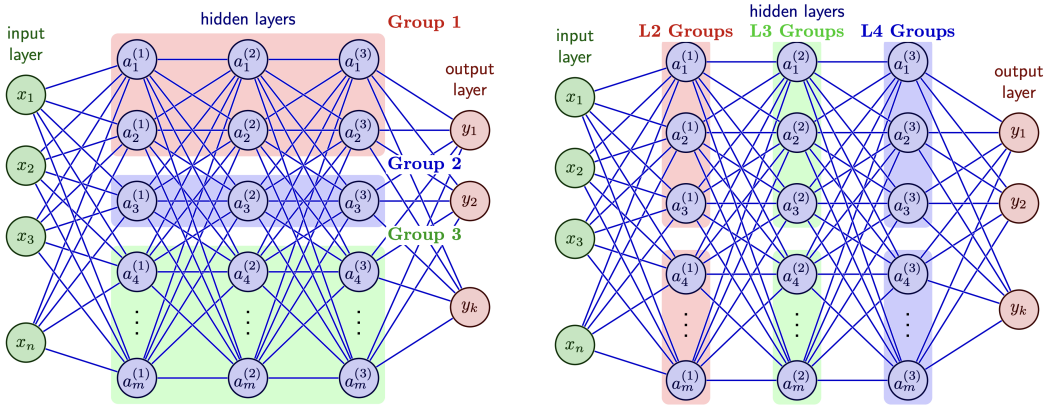
6. **Per-Layer LR Multipliers & Update.** Let γ be the base LR and μ_ℓ the per-layer multiplier:

$$\gamma_i = \gamma \times (\mu_\ell \text{ if defined, else } 1) \quad \text{(scale per layer)}. \tag{3.11}$$

Then update parameters:

$$p \leftarrow p - \gamma_i\, \hat{m}_t \quad \text{(SGD-style update)}. \tag{3.12}$$

3.2 Group Normalization Methods

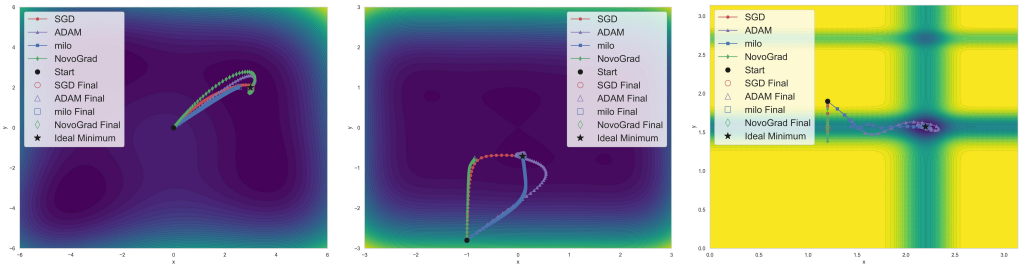


(a) Network-Wise Grouping (s per group) (b) Layer-Wise Grouping (per layer)

Figure 1: Comparison of Grouping Strategies

We propose two strategies to partition gradients for normalization:

Network-Wise Grouping (Figure 1a). Partition gradients into consecutive chunks of size s. Then apply mean/std normalization. Network-wise grouping, while reducing gradient variance through averaging, may disrupt intra-layer coordination when a single layer's parameters are spread across multiple groups, resulting in behavior closer to that of SGD.



(a) Himmelblau Loss (b) Camel6 Loss (c) Michalewicz Loss

Figure 2: 2D contour loss maps for three benchmark functions

Layer-Wise Grouping (Figure 1b). Group all parameters within each layer, then normalize as above. Layer-wise grouping fixes the effective gradient norm (scaled by η) per layer, thereby reducing noise due to variance in gradient magnitudes. This normalization helps prevent any single layer from dominating the update dynamics, leading to smoother loss curves and promoting broader minima that may enhance generalization.

Properties and Limitations Milo's per-group normalization yields the following properties. Variance reduction results from the balanced updates by scaling down large-magnitude gradients and amplifying smaller ones. Magnitude invariance is an additional component from decoupling step size from raw gradient scale, in turn requiring less hyperparameter tuning. Scale-aware blending preserves raw gradient information based on the desired. Additionally, there is an architectural agnosticism which works across heterogeneous layer types with minimal sensitivity. The optimizer is also more robust, automatically switching to a median/MAD-based normalization when encountering extreme gradient values. Lastly, Milo has demonstrated smoother and more direct convergence when compared to other optimizers. Some negative properties of Milo include dense-update sensitivity and a consistency versus adaptivity trade-off. There is an inherent grouping strategy dependency, with performance critically depending on how gradients are partitioned. Additionally, the extra grouping normalization steps can add runtime or memory overhead.

Algorithm Intuition To gain intuition, we compare Milo with SGD, Adam, and NovoGrad on three-dimensional loss landscapes. On Himmelblau [Himmelblau, 1972] (Figure 2a), all methods reach the global minimum, but Milo and SGD follow smooth, direct paths while Adam and NovoGrad oscillate. In the Camel6 landscape [Molga, 2005] (Figure 2b), Milo's uniform step size produces steady descent, avoiding the initial overshooting seen in the others. Finally, on Michalewicz [Michalewicz, 1996] (Figure 2c), only Milo and Adam escape the shallow basin to reach the deeper minimum; SGD and NovoGrad remain trapped. These observations highlight that Milo's fixed step size and group-wise normalization reduce variance and oscillations, striking a balance between stable progress and adaptive exploration. Overall, Milo's group-wise normalization was shown to blend SGD's stable, direct convergence with Adam's adaptability. It yields low-variance trajectories that escape shallow valleys and avoid the oscillations typical of fully adaptive methods.[1]

Efficient Implementation A large focus during the development of Milo was to have an efficient implementation due to the inherent cost of partitioning and normalizing gradients. Our implementation greatly minimizes the overhead that comes with this approach, implementing the following. Firstly, we minimize extra memory allocations by copying gradients into a pre-allocated buffer (stored in `_cached_buffers`). We also perform vectorized and batched computations of group mean and standard deviations to ensure a minimal memory footprint. As layer-wise is more computationally expensive, we use cached layer mapping to avoid repeated computations and overhead, generating a mapping of parameters to layers via the `_organize_layer_groups` function.

Difference from NovoGrad Milo partitions raw gradients into network or layer-wise groups and normalizes each group before any momentum or adaptive scaling—decoupling update size from gradient magnitude and offering optional scale-aware blending α. NovoGrad instead normalizes

[1] See Appendix D.5 for precise equations, more in-depth analysis, and three-dimensional plots.

per-layer moment estimates after accumulation, so its step sizes remain tied to those norms and lack explicit blending. Consequently, Milo is plug-and-play with minimal tuning, while NovoGrad requires careful adjustment of learning rate, decay rates, and subgroup sizes.

4 Convergence Guarantees for Milo

In this section, we establish theoretical convergence guarantees for Milo under a non-convex setting. We show that Milo benefits from variance reduction via layer-wise grouping. This variance reduction not only improves the effective gradient norm, yielding lower regret constants, but also has important implications for generalization across a variety of tasks. Due to space constraints, the detailed convex convergence proofs are deferred to Appendix B. Here, we focus on the non-convex setting.[2]

4.1 Convergence in Non-Convex Settings

Under the same normalization and L-smoothness assumptions, one shows:

Proof Sketch of Theorem C. Under Assumptions C.1, C.2, and the variance contraction/moment conditions (Assumptions B.3, B.4), we outline the main steps.

1. Descent Lemma. By L–smoothness of F, for $w_{t+1} = w_t - \eta\tilde{g}_t$:

$$F(w_{t+1}) \leq F(w_t) - \eta\nabla F(w_t), \tilde{g}_t \ + \frac{L\eta^2}{2}\|\tilde{g}_t\|^2. \tag{4.1}$$

2. Alignment Bound. Decompose $\tilde{g}_t = \nabla F(w_t) + \delta_t$. From Lemma B.2,

$$\nabla F(w_t), \tilde{g}_t \ \geq (1-\varepsilon)\|\nabla F(w_t)\|^2.$$

3. Norm Control. Using variance contraction (Lemma B.4),

$$\|\tilde{g}_t\|^2 \leq \rho\, n.$$

4. One-Step Inequality. Combine the above into the descent:

$$F(w_{t+1}) \leq F(w_t) - \eta(1-\varepsilon)\|\nabla F(w_t)\|^2 + \tfrac{L\eta^2}{2}\rho\, n. \tag{4.2}$$

Rearrange:

$$\eta(1-\varepsilon)\|\nabla F(w_t)\|^2 \leq F(w_t) - F(w_{t+1}) + \tfrac{L\eta^2}{2}\rho\, n.$$

5. Telescoping Sum. Summing for $t = 0 \ldots T-1$ and using $F(w_T) \geq F^*$:

$$\sum_{t=0}^{T-1}\|\nabla F(w_t)\|^2 \leq \frac{F(w_0) - F^*}{\eta(1-\varepsilon)} + \frac{L\eta T}{2(1-\varepsilon)}\rho\, n. \tag{4.3}$$

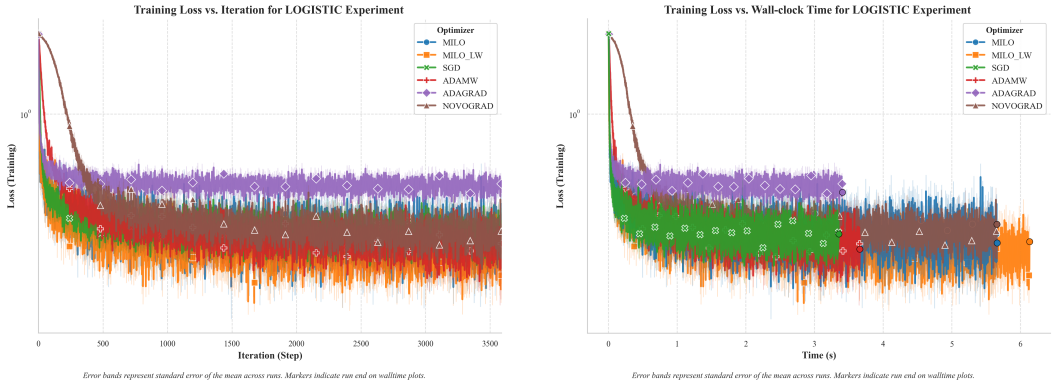
6. Rate via Step-Size. Choose $\eta = \min\{1/L,\, c/\sqrt{T}\}$ for constant $c > 0$. Then both terms scale as $O(T^{-1/2})$, yielding

$$\min_{t<T}\|\nabla F(w_t)\|^2 = O(T^{-1/2}). \tag{4.4}$$

□

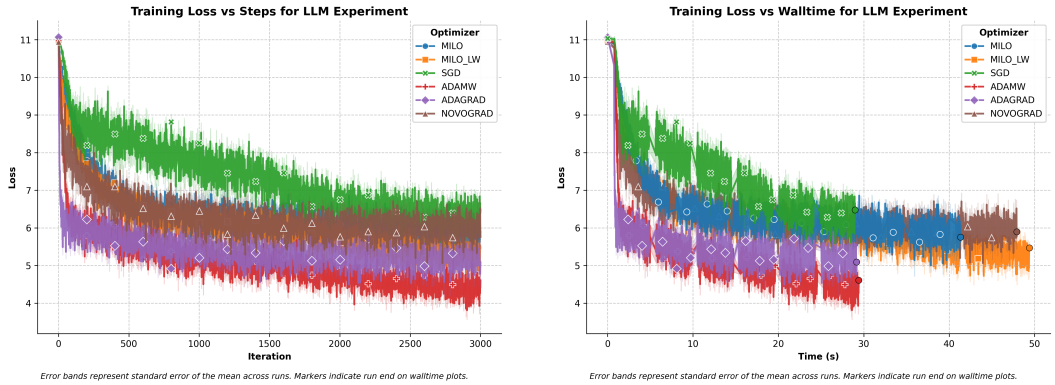
By normalizing groups of gradient components, Milo achieves variance contraction, which in turn improves convergence rates compared to standard SGD in both convex and non-convex settings. Additionally, the variance contraction also stabilizes the optimization trajectory. This stability can serve as an implicit regularizer, biasing the algorithm toward flatter minima, a trait empirically linked to improved generalization performance [Gower et al., 2020]. Additionally, a discussion of key technical aspects in these theorems is given in **??**.

[2]For full convergence proofs, see Appendix B for convex and Appendix C for non-convex. Here we have precise equations, review key convexity properties, derive regret bounds for standard SGD, and then show that Milo benefits from variance reduction via layer-wise grouping.



(a) Train Loss (b) Walltime Validation Cost

Figure 3: Comparison of Logistic Regression training loss by iterations and wall time.



(a) Train Loss (b) Walltime Loss

Figure 4: Comparison of LLM performance by training loss and wall time.

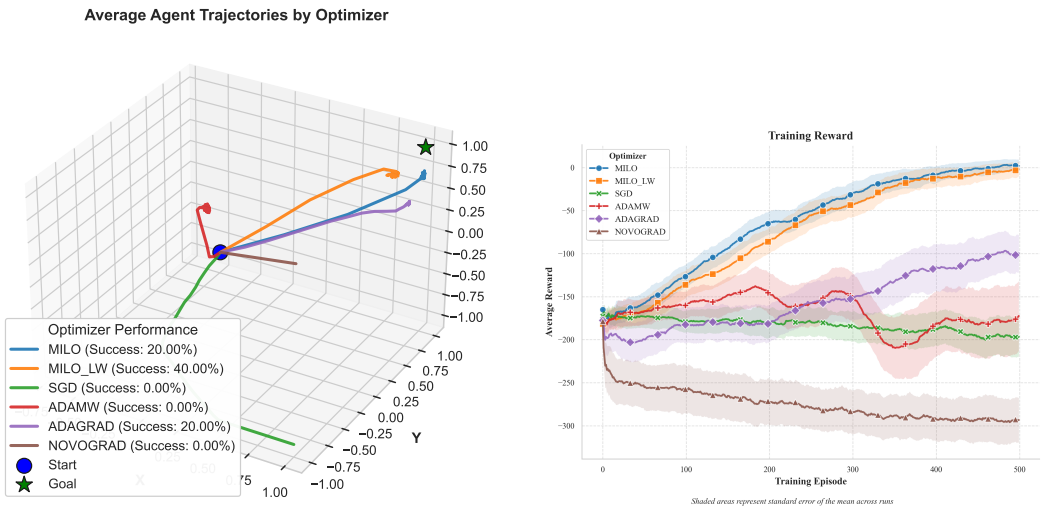
5 Experiments

To evaluate Milo empirically, we compare it with SGD with momentum, AdamW, AdaGrad, and NovoGrad across six experiments to cover common machine learning models. These experiments include finding the global minima for loss maps, logistic regression, MLPs, convolutional neural networks (CNNs) [LeCun et al., 1998] via ResNet-18 [He et al., 2016], RL in a three-dimensional space, and LLM pre-training. The models were trained in 5 separate instances for each empirical experiment, and the results of these trainings were averaged. The same parameter initialization is used when comparing the optimizers, and an extensive hyperparameter search and tuning were performed individually for each optimizer, as shown in Appendix D.6.[3] All Adam variants use default parameters $\beta 1 = 0.9$, $\beta 2 = 0.999$, $\epsilon = 10^{-8}$. This experimentation demonstrates that Milo can solve deep-learning problems with a high loss reduction and convergence rate.

5.1 Experimentation Analysis

Our experimentation found that across diverse tasks, Milo and its layer-wise variant (Milo_{LW}) consistently achieved improved or on-par convergence and enhanced stability over traditional optimizers. It can be observed that the normalized gradient updates reduce the variance and noise, while also ensuring invariance to raw gradient scales. These overall contribute to its consistent progress in both convex and non-convex settings. However, its walltime is consistently worse or on par with the other evaluated optimizers.

[3] See Appendix D for details of all hyperparameters and their tuning, data splits, hardware used, compute resources, in-depth statistical significance tests, experiment accuracy results, ablation study, and link to test code.



(a) 3D Agent Trajectories

(b) Reward Comparison

Figure 5: Comparison of RL results in terms of agent trajectories and reward performance.

In the logistic regression and MLPs experiments, shown in Appendix D.7 and Appendix D.8, Milo$_{\text{LW}}$ achieves faster loss reduction along with higher F1 accuracy in convex and non-convex settings with these less complex problems (Figure 3a and Figure 3b). This illustrates the benefits of the layer-wise group-normalized updates in these less complex environments, leading to a more rapid loss decrease.

The ResNet18 experiment (in Appendix D.9) contrasts with the less complex experiments as Milo outperforms Milo$_{\text{LW}}$. Milo demonstrates robust performance with an almost on-par convergence rate with its competitors, while also demonstrating a high validation accuracy (Table 1). This suggests that while Milo$_{\text{LW}}$ is effective for capturing layer-specific dynamics, it can underperform in scenarios where global gradient invariance is more advantageous (e.g., CNNs [Ye et al., 2021]).

In the LLM experiment (in Appendix D.11), Milo$_{\text{LW}}$ is one of the better performing optimizers, demonstrating robust performance (Figure 4a and Figure 4b). However, it falls close behind AdamW and AdaGrad. This suggests that global grouping can wash out useful layer disparities in particular complex instances such as the LLM pre-training here.

In Appendix D.10, we evaluate the optimizers in a three-dimensional RL scenario (Figure 5a and Figure 5b). For this task, there were two variations, featuring the agents reaching a consistently set point and a randomly generated point. In the set goal task, most optimizers had a near 100% accuracy in reaching the goal. Whereas in the randomized setup, Milo$_{\text{LW}}$ achieves the highest success rate by the largest margin, with Milo, and AdaGrad following behind, with half of its success rate. This demonstrates the better performance of Milo$_{\text{LW}}$ in noisy or ill-conditioned tasks, with it also best maximizing the reward for these scenarios as well.

Scalability and Runtime. This study also evaluated the scalability and efficiency of Milo and Milo$_{\text{LW}}$, with it being shown that in a large variety of scenarios, their wall-clock time, memory usage, and computational overhead often come at the cost of their increase in performance. Though Milo was observed to maintain relatively on-par runtimes with other optimizers, Milo$_{\text{LW}}$ commonly sacrificed its runtime for better results in the more complex models, such as the LLM pre-training. Future work will involve addressing these runtime issues and further expanding the initial solutions in Section 3.2. A more in-depth analysis of the runtime metrics is provided in Appendix D.

Hyperparameter Tuning Behavior. Despite aggressive hyperparameter tuning, Milo and Milo$_{\text{LW}}$ proved to be largely invariant to changes in their hyperparameters. Due to this, the default values for Milo and Milo$_{\text{LW}}$ were employed for each experiment, with no tuning. This suggests an invariance to parameter changes, with the normalization acting as a self-regulator for the optimizer, giving it a "plug and play" usage that many other optimizers do not provide. In contrast, the other methods required careful tuning to reach the demonstrated accuracies, with their default implementation achieving much lower accuracies. We provide further exploration of this idea in Appendix D.12.

5.2 Theory to Empirical Results

In convex settings, experiments on logistic regression have demonstrated a monotonic decrease in training loss consistent with our descent inequality. The normalized updates enforce a stable step size regardless of raw gradient magnitude variations. Additionally, our theoretical analysis (Appendix B) shows that controlling the effective gradient norm (via the contraction factor ρ) and the alignment error (ε) reduces the regret from $R_{\text{SGD}}(T) \leq D\,G\,\sqrt{T}$ to $R_{\text{Milo}}(T) \leq D\,\tilde{G}\sqrt{T}$, with $\tilde{G} < G$. Empirical results (**??**) confirm that both Milo and its layer-wise variant (Milo$_{\text{LW}}$) consistently achieve lower training loss and higher accuracy on MNIST.

In non-convex settings, stable convergence trajectories were given for complex models like multilayer neural networks and CNNs, as our non-convex convergence theorem (Appendix C) guarantees that $\min_{t=1,\ldots,T} \mathbb{E}\left[\|\nabla F(w_t)\|^2\right] = O\left(\frac{1}{\sqrt{T}}\right)$, implying reduced gradient noise. Empirically, Figure 4a and Figure 4b show that Milo-based updates yield smoother loss curves and stable convergence despite the presence of multiple local minima and saddle points. Furthermore, the group normalization reduces gradient variance (as quantified by ρ and controlled via ϵ and $\sigma_{\min}$), which is reflected in lower oscillations and tighter convergence in the loss curves.

6 Limitations and Future Work

Limitations. While Milo demonstrates strong performance across tasks, several important questions remain. First, its behavior in large-scale fine-tuning regimes (e.g. LLM fine-tuning) has yet to be explored. Second, in densely parameterized modules, such as attention layers, per-group normalization may inadvertently suppress critical gradient signals. Last, by enforcing nearly constant step magnitudes, Milo trades off adaptivity for consistency; this could lead to overly conservative updates in rapidly changing or highly nonstationary loss landscapes.

Extensions and Variants. We believe our algorithm could be extended to cover and further evaluate these limitations in future work. Such extensions would include addressing the above limitations, exploring new versions such as Milo with decoupled weight decay, and further theoretical analysis to compare the similarities of base Milo with SGD and Milo$_{\text{LW}}$ with Adam. We plan to expand our implementation to cover those mentioned in future work.

7 Conclusion

This study introduces a new, high-performing optimizer applicable to diverse ML training scenarios. The major impact of our work is the increased stability, loss reduction, and reduced sensitivity to hyperparameters. These make it practical for practitioners looking for a "plug and play" optimizer that can retain or exceed the performance of other optimizers. Milo combines the stability and direct update dynamics of SGD with the variance-reducing effects of gradient normalization, making it effective across vision tasks, varying model scales, and noisy or ill-conditioned optimization settings. Furthermore, Milo$_{\text{LW}}$ offers enhanced layer-specific adaptation, excelling in settings where fine-grained adjustments are required due to heterogeneous gradient landscapes. It is to be noted that despite the improved performance, Milo comes at the cost of increased runtime, especially when considering the layer-wise version. Overall, the Magnitude-Invariant Learning Optimizer (Milo and Milo$_{\text{LW}}$) provides a robust, theoretically sound, and practically effective alternative to conventional optimizers.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631.

Bottou, L., Curtis, F. E., and Nocedal, J. (2018). Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311. Used for Assumption 1.2 (Bounded Gradient Norm).

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, F., and Zaremba, W. (2016). Openai gym. arXiv:1606.01540.

Crockford, D. (2006). The application/json media type for javascript object notation (json). RFC 4627.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159. Introduces AdaGrad and provides a detailed regret analysis of adaptive gradient methods, closely related to Adam's approach.

FareedKhan-dev (2025). FareedKhan-dev/train-llm-from-scratch: A straightforward method to train large language models from scratch. GitHub repository. Accessed: 2025-05-14.

Gao, L., Biderman, S., Black, S., Golding, C., Hoppe, T., Foster, J. P., He, H., Thite, A., Nabeshima, N., Tow, B., et al. (2020). The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Ginsburg, B., Castonguay, P., Hrinchuk, O., Kuchaiev, O., Lavrukhin, V., Leary, R., Li, J., Nguyen, H., Zhang, Y., and Cohen, J. M. (2019). Stochastic gradient methods with layer-wise adaptive moments for training of deep networks. *Preprint at* `https://arxiv.org/abs/1905.11286`.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Gower, R. M., Schmidt, M., Bach, F., and Richtárik, P. (2020). Variance-reduced methods for machine learning. *Proceedings of the IEEE*, 108(11):1968–1983. Shows empirical affect of variance reduction.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pages 1861–1870.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Hettinger, L., Bernstein, J., et al. (2022). Recent advances in optimizers for deep learning. *arXiv preprint arXiv:2207.01716*.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, volume 30.

Himmelblau, D. M. (1972). *Applied Nonlinear Programming*. McGraw-Hill.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30. Used for Lemma 1.1 (Concentration of Empirical Means).

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 448–456. Used for Lemma 1.4 (Bounded Variance Contraction).

Keskar, N. S., Nocedal, J., et al. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*. Introduces Adam, which computes adaptive learning rates for each parameter by combining momentum and adaptive scaling of gradients. This approach reduces variance and accelerates convergence compared to plain SGD.

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Technical report, University of Toronto.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. The MNIST dataset consists of 70,000 grayscale images of handwritten digits (0–9) and is commonly used for training and testing in the field of machine learning and computer vision.

lonePatient (2019). lonePatient/NovoGrad-pytorch: Pytorch implementation of the novograd optimizer. GitHub repository. Accessed: 2025-05-14.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *International Conference on Learning Representations*.

McKinney, W. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, pages 51–56.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3rd edition.

Molga, M. and Smutnicki, C. (2005). Test functions for optimization needs. Technical report, Faculty of Fundamentals of Technology, Wrocław University of Technology.

Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376. Used for Assumption 1.1 (Smoothness).

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

Python Software Foundation (2023). *Python Standard Library: itertools*. Version 3.10.

PyTorch Contributors (2023). torchvision: image and video datasets, transforms, and models for pytorch. `https://github.com/pytorch/vision`. Version 0.21.0.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning (ICML)*, pages 1139–1147.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5—rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

van der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Vershynin, R. (2018). *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge University Press. Used for Assumption 1.4 (Moment Conditions).

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. (2020). Scipy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272.

Waskom, M. (2021). Seaborn: Statistical data visualization.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, volume 30.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., and Brew, J. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.

Wu, Y. and He, K. (2018). Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19. Used for Assumption 1.3 (Variance Contraction via Normalization) and Lemma 1.3 (Robustness for Small σ_j).

Xu, J., Sun, X., Zhang, Z., Zhao, G., and Lin, J. (2019). Understanding and improving layer normalization. In *Advances in Neural Information Processing Systems*, volume 32, pages 5665–5675. Curran Associates, Inc. We reference this work to highlight limitations of per-layer normalization in handling heterogeneous gradient scales, motivating our group-normalization approach in Milo.

Ye, C., Zhou, X., McKinney, T., Liu, Y., Zhou, Q., and Zhdanov, F. (2021). Exploiting invariance in training deep neural networks. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence*. This research paper delves into the theoretical advantages of incorporating invariance into neural networks, a common characteristic of real-world data.

Yong, Y., Huang, E., Hua, L., and Zhang, C. (2020). Gradient centralization: A new optimization technique for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*. Introduces Gradient Centralization, an optimization technique that centralizes gradients to improve training stability and convergence.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

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

   Answer: [Yes]

   Justification: This paper provides extensive empirical and theoretical validation, which matches the claims made. These sections clearly state the contributions made by this paper and are further developed in the following sections.

   Guidelines:

   - The answer NA 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 NA 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: This paper dives heavily into the limitations of our proposed optimizer, highlighting the computational issues that come with it along with domains where is underperforms compared to other optimizers. Additionally, we also discuss potential limitations that we did not come across in our studies, but are plausible.

   Guidelines:

- The answer NA 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: For each of our theoretical results, we give an extensive setup covering assumptions, lemmas, theorems, definitions, etc. Additionally, we validate each of these where necessary and provide sketches for the proofs as well.

   Guidelines:

   - The answer NA 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: This paper provides in-depth details and self-contained pseudocode for the algorithm we propose, along with hyperparameters, data splits, hardware details, a fixed random seed (42), and more for reproducibility. Additionally, we intend to release the source code used for the experimentation done in this study.

   Guidelines:

   - The answer NA 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: We submit our code as supplemental material, and intend to include a GitHub link in the final paper version if accepted for publication. This will include a README and requirements.txt so users can fully recreate our testing environment. This will provide commands for execution, data splits, etc.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.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://nips.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, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The main text and appendices give full details on data splits, hyperparameter selection, optimizer configurations, and training procedures. Additionally, the configuration and training files are submitted with this work.

   Guidelines:

- The answer NA 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: We report error bars in all plots, representing the standard error of the mean (SEM) across independent runs, capturing variability due to random initialization and data splits. Error bars and p-values (from Welch's t-test) are calculated as described in the appendix, with all methods and assumptions clearly stated and referenced in the main text and figures.

   Guidelines:

   - The answer NA 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: For each experiment run, we record and report the type of compute resources used (CPU/GPU model, memory, and execution time), as well as detailed system information and per-run resource usage, which are included in the main text, appendices, and made available in the repository. We also provide the compute requirements for individual runs and the total experiment, enabling full reproducibility and transparency regarding computational resources.

   Guidelines:

   - The answer NA 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 submission is fully anonymized and no ethical concerns arise from a foundational optimizer study.

Guidelines:

- The answer NA 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: [NA]

Justification: This paper has no direct societal impact as it is foundational research, having the same level of impact as any other optimizer.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA 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., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: This paper poses no such risks, as this optimizer is foundational and does not stand to be misused. It introduces no high-risk data or models requiring controlled release.

Guidelines:

- The answer NA 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: All third-party methods and datasets are properly cited with original references and URLs, crediting the creators.

Guidelines:

- The answer NA 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: This paper introduces a new software asset, the Milo optimizer. Additionally, it provides extensive documentation alongside it.

Guidelines:

- The answer NA 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: [NA]

Justification: This paper does not involve crowdsourcing nor research with human subjects. The nature of our work does not involve it.

Guidelines:

- The answer NA 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: [NA]

Justification: This paper does not involve crowdsourcing nor research with human subjects. The nature of our work does not involve it.

Guidelines:

- The answer NA 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 rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: LLMs are not an important, original, or non-standard component of our research. They have no impact on our methodology other than training one in an experiment.

    Guidelines:

    - The answer NA 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 (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Appendices

Contents

A Nomenclature

Table 2: Nomenclature

Symbol	Description
Model Parameters and Tensors	
p_i	Parameter tensor.
$\text{layer}(p)$	Function mapping a parameter p to its layer identifier.
$\text{vec}(\cdot)$	Operation that flattens a tensor into a vector.
Gradients and Grouping	
g_i	Gradient of parameter p_i.
$\nabla f(p)$	Gradient of the objective function f with respect to p.
$g_t \in \mathbb{R}^n$	Full (raw) gradient at iteration t (vector of dimension n).
n	Total number of gradient components.
m	Number of disjoint groups into which g_t is partitioned.
j	Group index, $j \in \{1, 2, \ldots, m\}$.
G_j	jth group (subset) of gradient components from g_t.
$\lvert G_j \rvert$	Number of components in group G_j.
μ_j	Empirical mean of the components in group G_j.
σ_j^2	Empirical variance of the components in group G_j.
ϵ	Stability constant: small positive constant added in normalization ($\sigma_j + \epsilon$) to avoid division by zero.
$\tilde{g}_t$	Normalized gradient at iteration t, defined as $[\tilde{g}_t]_i = \frac{[g_t]_i - \mu_j}{\sigma_j + \epsilon}$ for $i \in G_j$.
ρ_j	Variance contraction factor for group G_j, defined as $\rho_j = \frac{\sigma_j^2}{(\sigma_j + \epsilon)^2}$.
ρ	Global constant such that $\rho_j \leq \rho < 1$ for all groups.
Layer-Wise Grouping (if applicable)	
$\mathcal{G}_l$	Set of parameters (or corresponding gradient indices) in layer l.
v_l	Flattened gradient vector for layer l.
$\lvert v_l \rvert$	Number of elements in v_l.
$v_l^{(j)}$	jth subgroup of v_l (when partitioning a layer's gradients).
$\mu_l^{(j)}$	Mean of subgroup j in layer l.
$\sigma_l^{(j)}$	Standard deviation of subgroup j in layer l.
$\tilde{v}_l$	Concatenated normalized gradient vector for layer l.
Optimization Variables and Algorithm Parameters	
w_t	Current model parameters (iterate) at iteration t.
f or F	Objective function (convex: f; non-convex: F).
η	Step size (learning rate).
T	Total number of iterations.
D	Diameter of the feasible set $\mathcal{X}$.
$\mathcal{X}$	Feasible set of model parameters (subset of $\mathbb{R}^d$).
G	Uniform upper bound on the gradient norm.
Theoretical Analysis Parameters	
δ_t	Alignment error vector defined by $\tilde{g}_t = \nabla f(w_t) + \delta_t$.
ε	Alignment error bound: a scalar such that $\lvert\langle \nabla f(w_t), \delta_t \rangle\rvert \leq \varepsilon \lVert \nabla f(w_t) \rVert^2$. specifically, it is bounded as $\lvert\langle \nabla f(w_t), \delta_t \rangle\rvert \leq \varepsilon \lVert \nabla f(w_t) \rVert^2$, with $\varepsilon = \frac{\epsilon}{\sigma_{\min}} C(\tau)$.
α	Constant in $(0, 1)$ controlling the ratio between ϵ and σ_j to ensure proper scaling.
$C(\tau)$	Constant that depends on the sub-Gaussian parameter τ of the gradient components.
$\Theta(\cdot)$	Tight asymptotic bound.
$O(\cdot)$	Big-O notation.
Regret Analysis	
$R(T)$	Cumulative regret over T iterations.
$R_{SGD}(T)$	Regret bound for standard SGD.
$R_{Milo}(T)$	Regret bound for Normalized SGD with Layer-Wise Grouping (Milo).

B Convergence Proof for Convex Settings

In this section we present the main convergence proof. We begin by introducing the definitions, assumptions, and lemmas that will be used throughout.

B.1 Proof Sketch of Theorem

Under standard L-smoothness, bounded-gradient, and variance-contraction/moment conditions, we obtain the following.

Proof Sketch of Theorem B. We outline the main steps under Assumptions B.1–B.4 and Lemmas B.4, B.2.

1. Smoothness Step. By L–smoothness,

$$f(w_{t+1}) \le f(w_t) + \langle \nabla f(w_t),\, w_{t+1} - w_t \rangle + \tfrac{L}{2}\|w_{t+1} - w_t\|^2.$$

Using the update $w_{t+1} = w_t - \eta \tilde{g}_t$ gives

$$f(w_{t+1}) \le f(w_t) - \eta \langle \nabla f(w_t), \tilde{g}_t \rangle + \tfrac{L\eta^2}{2}\|\tilde{g}_t\|^2.$$

2. Variance-Contraction Bound. By Assumption B.3 and Lemma B.4, each group satisfies $\|[\tilde{g}_t]_{G_j}\|^2 \le \rho\,|G_j|$. Summing yields

$$\|\tilde{g}_t\|^2 \le \rho\, n.$$

3. Alignment Control. Decompose $\tilde{g}_t = \nabla f(w_t) + \delta_t$. Lemma B.2 gives

$$|\nabla f(w_t), \delta_t\,| \le \varepsilon \|\nabla f(w_t)\|^2,$$

so

$$\nabla f(w_t), \tilde{g}_t\ \ge (1-\varepsilon)\|\nabla f(w_t)\|^2.$$

4. One-Step Progress. Combine the above bounds:

$$f(w_{t+1}) \le f(w_t) - \eta(1-\varepsilon)\|\nabla f(w_t)\|^2 + \tfrac{L\eta^2}{2}\rho\, n.$$

5. Telescoping Sum. Summing over $t = 1 \dots T$ and using $f(w_{T+1}) \ge f(w^*)$:

$$\sum_{t=1}^{T} \|\nabla f(w_t)\|^2 \le \frac{f(w_1) - f(w^*)}{\eta(1-\varepsilon)} + \frac{L\eta T}{2(1-\varepsilon)}\rho\, n.$$

6. Learning-Rate Choice. Setting $\eta = \Theta(1/\sqrt{T})$ balances the two terms and yields

$$\frac{1}{T}\sum_{t=1}^{T} \|\nabla f(w_t)\|^2 = O\big(T^{-1/2}\big).$$

7. Regret Comparison. Standard SGD regret is $O(GD\sqrt{T})$. Replacing G by $\sqrt{\rho n}$ gives Milo

$$R_{\mathrm{Milo}}(T) = O\big(D\sqrt{\rho n\, T}\big) < O(DG\sqrt{T}) = R_{\mathrm{SGD}}(T),$$

showing improved dependence on the variance.

B.2 Definitions and Assumptions

Definition B.1 (Convex Function)**.** *A function $f : \mathbb{R}^d \to \mathbb{R}$ is* convex *if for all $x, y \in \mathbb{R}^d$ and all $\lambda \in [0,1]$,*

$$\lambda f(x) + (1-\lambda) f(y) \ge f\big(\lambda x + (1-\lambda) y\big). \tag{B.1}$$

Remark B.1. *Notice that for a convex function f, one can lower bound f by a hyperplane at any point of its domain. In particular, if f is differentiable, then for all $x, y \in \mathbb{R}^d$,*

$$f(y) \ge f(x) + \nabla f(x)^T (y - x). \tag{B.2}$$

Assumption B.1 (Smoothness)**.** *The function $f : \mathbb{R}^d \to \mathbb{R}$ is differentiable and L-*smooth*; that is, for all $x, y \in \mathbb{R}^d$,*

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2. \tag{B.3}$$

Often, f is given as a finite sum $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$ with individual smoothness constants L_i; we denote $L_{\max} = \max\{L_1, \ldots, L_n\}$.

[See, e.g., [Nesterov, 1983] for the smoothness condition.]

Assumption B.2 (Bounded Gradient Norm)**.** *For every $x \in \mathcal{X}$ (the feasible set), the gradient is bounded:*

$$\|\nabla f(x)\|_2 \leq G, \tag{B.4}$$

with $G > 0$.

[This is standard in stochastic optimization; see, e.g., [Bottou et al., 2018].]

Assumption B.3 (Variance Contraction via Normalization)**.** *Let $g_t = \nabla f(w_t)$ be the gradient at iteration t. Partition g_t into disjoint groups $\{G_j\}_{j=1}^{m}$ (with $\sum_{j=1}^{m} |G_j| = n$). For each group G_j, define the empirical mean and variance as*

$$\mu_j = \frac{1}{|G_j|}\sum_{i \in G_j} [g_t]_i, \quad \sigma_j^2 = \frac{1}{|G_j|}\sum_{i \in G_j} \Big([g_t]_i - \mu_j\Big)^2. \tag{B.5}$$

The normalized gradient is then defined by

$$[\tilde{g}_t]_i = \frac{[g_t]_i - \mu_j}{\sigma_j + \epsilon}, \quad i \in G_j, \tag{B.6}$$

Because the mean is subtracted in Equation (B.6) *(centering the data), the variance of the normalized gradient in G_j becomes*

$$\tilde{\sigma}_j^2 = \frac{\sigma_j^2}{(\sigma_j + \epsilon)^2}. \tag{B.7}$$

Define the variance contraction factor as

$$\rho_j = \frac{\sigma_j^2}{(\sigma_j + \epsilon)^2}. \tag{B.8}$$

We assume that there exists a global constant $\rho \in [0, 1)$ such that

$$\rho_j \leq \rho, \quad \forall\, j. \tag{B.9}$$

[See, e.g., [Wu and He, 2018] for the effects of group normalization on variance contraction.]

Assumption B.4 (Moment Conditions)**.** *For each parameter tensor p with flattened gradient $v = \mathrm{vec}(g(p))$, assume:*

1. *(**Large Group Size**) Each group G_j (obtained by partitioning v) satisfies $|G_j| \geq N_0$ so that the empirical estimates*

$$\mu_j = \frac{1}{|G_j|}\sum_{i \in G_j} v_i, \quad \sigma_j^2 = \frac{1}{|G_j|}\sum_{i \in G_j} (v_i - \mu_j)^2 \tag{B.10}$$

   concentrate around the true moments.

2. *(**Bounded Higher-Order Moments**) There exists a constant M_k and some $k \geq 3$ such that*

$$\mathbb{E}[|v_i - \mu_j|^k] \leq M_k. \tag{B.11}$$

3. *(**Symmetry**) The distribution of v_i for $i \in G_j$ is symmetric (or nearly so) around μ_j.*

[Moment conditions as in [Vershynin, 2018].]

B.3 Auxiliary Lemmas

Lemma B.1 (Concentration of Empirical Means)**.** *Let $v_1, \dots, v_N$ be independent random variables in a group G_j with true mean μ and assume $|v_i - \mu| \le B$ almost surely. Then, for any $\delta > 0$,*

$$\Pr\left(\Big| \frac{1}{N} \sum_{i=1}^{N} v_i - \mu \Big| \ge \delta \right) \le 2 \exp\Big(-\frac{2N\delta^2}{B^2} \Big). \tag{B.12}$$

A similar bound holds for the empirical variance.

[This is an application of Hoeffding's inequality; see [Hoeffding, 1963].]

Lemma B.2 (Formal Alignment of Normalized Gradient)**.** *Assume that for a given group G_j the gradient components can be expressed as*

$$v_i = \mu_j + \sigma_j z_i, \quad i \in G_j, \tag{B.13}$$

where z_i are independent sub-Gaussian random variables with zero mean, unit variance, and sub-Gaussian parameter τ. Define the normalized elements by

$$\tilde{v}_i = \frac{v_i - \mu_j}{\sigma_j + \epsilon}. \tag{B.14}$$

Then,

$$\tilde{v}_i = \frac{\sigma_j}{\sigma_j + \epsilon}\, z_i. \tag{B.15}$$

Define the scaling factor $P_j := \frac{\sigma_j}{\sigma_j + \epsilon}$. If for a fixed $\alpha \in (0,1)$ it holds that $\epsilon \le \alpha\, \sigma_j$ (i.e. $P_j \ge \frac{1}{1+\alpha}$), then

$$1 - P_j \le \frac{\epsilon}{\sigma_j}. \tag{B.16}$$

Now, writing the decomposition

$$\tilde{g}_t = \nabla f(w_t) + \delta_t, \tag{B.17}$$

the above results (with appropriate concentration bounds) imply that

$$\big| \langle \nabla f(w_t), \delta_t \rangle \big| \le \varepsilon\, \|\nabla f(w_t)\|^2, \tag{B.18}$$

with $\varepsilon = \frac{\epsilon}{\sigma_{\min}}\, C(\tau)$ (here $\sigma_{\min} = \min_j \sigma_j$ and $C(\tau)$ is a constant depending on τ).

[See, e.g., [Duchi et al., 2011] for related analysis on adaptive and normalized gradient methods.]

Lemma B.3 (Robustness for Small σ_j with Explicit ϵ)**.** *Assume that for each group G_j either:*

1. *$\sigma_j \ge \sigma_{\min} > 0$, or*
2. *If $\sigma_j < \sigma_{\min}$, then $|v_i - \mu_j| \le \beta\, \sigma_{\min}$ for all $i \in G_j$,*

for some constants $\sigma_{\min} > 0$ and $\beta > 0$. Then, by choosing

$$\epsilon \le \gamma\, \sigma_{\min}, \tag{B.19}$$

for a given $\gamma \in (0,1)$, it follows that for every $i \in G_j$,

$$|[\tilde{g}_t]_i| \le \frac{\beta}{\gamma}. \tag{B.20}$$

Lemma B.4 (Bounded Variance Contraction)**.** *Let $\epsilon > 0$ be such that for every group G_j (and for all iterations) it holds that*

$$\epsilon \ge c\, \sigma_j, \tag{B.21}$$

for some constant $c > 0$. Then, the variance contraction factor satisfies

$$\rho_j = \frac{\sigma_j^2}{(\sigma_j + \epsilon)^2} \le \Big(\frac{1}{1+c} \Big)^2 < 1. \tag{B.22}$$

[This result is motivated by normalization techniques in [Ioffe and Szegedy, 2015].]

B.4 Proof of Convergence for Convex Settings

We now proceed with the convergence proof. The update rule for Milo is given (after normalization) by

$$w_{t+1} = w_t - \eta\, \tilde{g}_t, \tag{B.23}$$

where η is the learning rate and the normalized gradient $\tilde{g}_t$ is defined in eq. Equation (B.6). Throughout the proof we also use the following decomposition:

$$\tilde{g}_t = \nabla f(w_t) + \delta_t, \tag{B.24}$$

where δ_t represents the alignment error (as introduced in Lemma B.2).

Step 1: Smoothness Inequality. By Assumption B.1 (Smoothness), for any $x, y \in \mathbb{R}^d$ we have

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|^2. \tag{B.25}$$

Taking $x = w_t$ and $y = w_{t+1} = w_t - \eta\, \tilde{g}_t$ yields

$$f(w_{t+1}) \leq f(w_t) - \eta \langle \nabla f(w_t), \tilde{g}_t \rangle + \frac{L\eta^2}{2}\|\tilde{g}_t\|^2. \tag{B.26}$$

Here:

- $\nabla f(w_t)$ is the true gradient at w_t,
- $\tilde{g}_t$ is the normalized gradient from Equation (B.6), and
- $\langle \nabla \mathbf{f}(\mathbf{w_t}), \tilde{\mathbf{g}}_\mathbf{t} \rangle$: The inner product measuring the alignment between the true gradient and the normalized gradient.
- $\mathbf{L}$: The Lipschitz constant of ∇f, governing the curvature of f, as in Assumption B.1.

Step 2: Bounding the Norm of $\tilde{g}_t$. Within each group G_j, by the variance contraction property in Assumption B.3 and Lemma B.4,

$$\|[\tilde{g}_t]_{G_j}\|^2 = \frac{|G_j|\,\sigma_j^2}{(\sigma_j + \epsilon)^2} \leq \rho\,|G_j|. \tag{B.27}$$

Summing over all groups, we obtain

$$\|\tilde{g}_t\|^2 \leq \sum_j \rho\,|G_j| = \rho\, n. \tag{B.28}$$

Step 3: Handling the Alignment Error. We decompose the normalized gradient as

$$\tilde{g}_t = \nabla f(w_t) + \delta_t. \tag{B.29}$$

Then, by Lemma B.2 (Formal Alignment of Normalized Gradient), under the moment conditions (Assumption B.4) and the condition $\epsilon \leq \alpha\, \sigma_j$ for all j, we have

$$|\langle \nabla f(w_t), \delta_t \rangle| \leq \varepsilon\, \|\nabla f(w_t)\|^2, \tag{B.30}$$

with $\varepsilon = \frac{\epsilon}{\sigma_{\min}}\, C(\tau)$. Consequently,

$$\langle \nabla f(w_t), \tilde{g}_t \rangle \geq (1 - \varepsilon)\|\nabla f(w_t)\|^2. \tag{B.31}$$

Step 4: Inserting the Bounds. Substitute the alignment bound and the norm bound into the smoothness inequality:

$$f(w_{t+1}) \leq f(w_t) - \eta(1 - \varepsilon)\|\nabla f(w_t)\|^2 + \frac{L\eta^2}{2}\rho\, n. \tag{B.32}$$

Step 5: Telescoping. Summing from $t = 1$ to T yields

$$\sum_{t=1}^{T} \Big[f(w_t) - f(w_{t+1}) \Big] \geq \eta(1 - \varepsilon) \sum_{t=1}^{T} \|\nabla f(w_t)\|^2 - \frac{L\eta^2 T}{2}\rho\, n. \tag{B.33}$$

Since the sum telescopes to $f(w_1) - f(w_{T+1})$ and $f(w_{T+1}) \geq f(w^*)$ (where w^* is an optimal point), we have

$$f(w_1) - f(w^*) \geq \eta(1 - \varepsilon) \sum_{t=1}^{T} \|\nabla f(w_t)\|^2 - \frac{L\eta^2 T}{2}\rho\, n. \tag{B.34}$$

Dividing both sides by $\eta(1 - \varepsilon)T$ gives

$$\frac{1}{T} \sum_{t=1}^{T} \|\nabla f(w_t)\|^2 \leq \frac{f(w_1) - f(w^*)}{\eta(1 - \varepsilon)T} + \frac{L\eta}{2(1 - \varepsilon)}\rho\, n. \tag{B.35}$$

Step 6: Choosing the Learning Rate. By setting

$$\eta = \Theta\Big(\frac{1}{\sqrt{T}}\Big), \tag{B.36}$$

the two terms on the right-hand side are balanced, leading to an overall convergence rate of

$$\frac{1}{T} \sum_{t=1}^{T} \|\nabla f(w_t)\|^2 = O\Big(\frac{1}{\sqrt{T}}\Big). \tag{B.37}$$

Step 7: Regret Bound. Standard regret analysis for SGD yields

$$R_{\text{SGD}}(T) \leq \frac{D^2}{2\eta} + \frac{\eta G^2 T}{2}. \tag{B.38}$$

For Milo, if we denote by $\tilde{G}$ the effective bound on $\|\tilde{g}_t\|$ (with $\tilde{G} \leq \sqrt{\rho\, n}$), then the regret can be bounded as

$$R_{\text{Milo}}(T) \leq \frac{D^2}{2\eta} + \frac{\eta \tilde{G}^2 T}{2}. \tag{B.39}$$

Choosing $\eta = \frac{D}{\tilde{G}\sqrt{T}}$ gives

$$R_{\text{Milo}}(T) \leq D\,\tilde{G}\,\sqrt{T} < R_{\text{SGD}}(T) \leq D\,G\,\sqrt{T}, \tag{B.40}$$

thus demonstrating the advantage of variance reduction.

B.5 Discussion and Practical Relevance

The final convergence bound explicitly depends on the alignment error ε, the variance contraction factor ρ, and the learning rate η:

$$\frac{1}{T}\sum_{t=1}^{T} \|\nabla f(w_t)\|^2 \leq \frac{f(w_1) - f(w^*)}{\eta\,(1-\varepsilon)\,T} + \frac{L\,\eta}{2\,(1-\varepsilon)}\,\rho\, n. \tag{B.41}$$

By balancing the two terms with $\eta = \Theta(1/\sqrt{T})$, this yields the stated $O(1/\sqrt{T})$ convergence rate. Moreover, replacing the usual gradient bound G by $\sqrt{\rho\, n}$ in the standard SGD regret analysis gives

$$R_{\text{Milo}}(T) = O\big(D\sqrt{\rho\, n\, T}\big) < O\big(D\,G\,\sqrt{T}\big) = R_{\text{SGD}}(T),$$

demonstrating Milo's tighter dependence on variance.

Lemmas B.1 and B.2 show that under the moment conditions (Assumption B.4) and for sufficiently large group sizes, the empirical estimates μ_j, σ_j^2 concentrate around their true values and the normalized gradient remains well aligned with $\nabla f(w_t)$, yielding an alignment error $\varepsilon = O\big(\frac{\epsilon}{\sigma_{\min}}\big)$. Lemma B.4 guarantees a global variance contraction $\rho < 1$ whenever $\epsilon \geq c\,\sigma_j$, and Lemma B.3 ensures that even in low-variance regimes ($\sigma_j < \sigma_{\min}$) the updates stay bounded by choosing ϵ relative to $\sigma_{\min}$.

By making these parameter dependencies explicit, our analysis directly informs practical implementation:

1. Choose group sizes large enough to invoke concentration (Assumption B.4).
2. Select $\epsilon = O(\sigma_{\min})$ to control both alignment error ε and variance contraction ρ.
3. Verify sub-Gaussian or bounded-moment behavior of gradient components to satisfy the lemmas.

Empirical results (cf. Section 5) confirm that in realistic deep-learning settings these constants remain small, yielding smoother loss trajectories, reduced sensitivity to learning-rate tuning, and better generalization. This completes our self-contained convex convergence proof and highlights Milo's theoretical and practical advantages over standard SGD.

B.5.1 Practical Remarks.

In practice, unbiasedness holds up to $O(\epsilon/\sigma_{\min})$ when group sizes are large and gradient-component distributions are near-symmetric; variance is contracted by $\rho_j \leq (\sigma_j/(\sigma_j+\epsilon))^2 < 1$ whenever $\epsilon \geq c\,\sigma_j$; and even if some σ_j are tiny, choosing $\epsilon = O(\sigma_{\min})$ (Lemma B.3) keeps updates bounded.

C Convergence Proof for Non-Convex Settings

In this section, we present a detailed convergence analysis for Milo when the objective function is *non-convex*. Although nonconvexity precludes guarantees of reaching a global optimum, under standard smoothness and boundedness assumptions, we can establish a sublinear rate in terms of the (squared) gradient norm.

Proof Sketch of Theorem C. Under Assumptions C.1, C.2, and the variance contraction/moment conditions (Assumptions B.3, B.4), we outline the main steps.

1. Descent Lemma. By L–smoothness of F, for $w_{t+1} = w_t - \eta\tilde{g}_t$:

$$F(w_{t+1}) \leq F(w_t) - \eta\nabla F(w_t), \tilde{g}_t\ + \frac{L\eta^2}{2}\|\tilde{g}_t\|^2.$$

2. Alignment Bound. Decompose $\tilde{g}_t = \nabla F(w_t) + \delta_t$. From Lemma B.2,

$$\nabla F(w_t), \tilde{g}_t \ \geq (1-\varepsilon)\|\nabla F(w_t)\|^2.$$

3. Norm Control. Using variance contraction (Lemma B.4),

$$\|\tilde{g}_t\|^2 \leq \rho\, n.$$

4. One-Step Inequality. Combine the above into the descent:

$$F(w_{t+1}) \leq F(w_t) - \eta(1-\varepsilon)\|\nabla F(w_t)\|^2 + \tfrac{L\eta^2}{2}\rho\, n.$$

Rearrange:

$$\eta(1-\varepsilon)\|\nabla F(w_t)\|^2 \leq F(w_t) - F(w_{t+1}) + \tfrac{L\eta^2}{2}\rho\, n.$$

5. Telescoping Sum. Summing for $t = 0 \dots T-1$ and using $F(w_T) \geq F^*$:

$$\sum_{t=0}^{T-1} \|\nabla F(w_t)\|^2 \leq \frac{F(w_0) - F^*}{\eta(1-\varepsilon)} + \frac{L\eta T}{2(1-\varepsilon)}\rho\, n.$$

6. Rate via Step-Size. Choose $\eta = \min\{1/L,\, c/\sqrt{T}\}$ for constant $c > 0$. Then both terms scale as $O(T^{-1/2})$, yielding

$$\min_{t<T} \|\nabla F(w_t)\|^2 = O(T^{-1/2}).$$

□

C.1 Assumptions and Preliminaries

Let $F : \mathbb{R}^d \to \mathbb{R}$ be a (possibly) non-convex function. In addition to the variance reduction ingredients from the convex analysis, we assume the following:

Assumption C.1 (Smoothness for Non-Convex Functions)**.** *F is differentiable and L-smooth; that is, for all $x, y \in \mathbb{R}^d$,*

$$\|\nabla F(x) - \nabla F(y)\|_2 \leq L\|x - y\|_2. \tag{C.1}$$

Assumption C.2 (Bounded Gradient)**.** *For every $x \in \mathbb{R}^d$, the gradient is bounded:*

$$\|\nabla F(x)\|_2 \leq G, \tag{C.2}$$

with $G > 0$.

We also assume that the grouping and normalization procedure is defined exactly as in the convex case. That is, for iteration t we have

$$g_t = \nabla F(w_t), \tag{C.3}$$

which is partitioned into groups $\{G_j\}_{j=1}^m$ with

$$\mu_j = \frac{1}{|G_j|}\sum_{i \in G_j} [g_t]_i, \quad \sigma_j^2 = \frac{1}{|G_j|}\sum_{i \in G_j} \Big([g_t]_i - \mu_j\Big)^2, \tag{C.4}$$

and the normalized gradient is defined as

$$[\tilde{g}_t]_i = \frac{[g_t]_i - \mu_j}{\sigma_j + \epsilon}, \quad i \in G_j. \tag{C.5}$$

We assume that Assumptions B.3 and B.4 from the convex proof (regarding variance contraction via normalization and moment conditions) hold unchanged here.

C.2 Proof of Convergence for Non-Convex Settings

The Milo update is given by

$$w_{t+1} = w_t - \eta\, \tilde{g}_t. \tag{C.6}$$

Since F is L-smooth (Assumption C.1), we have the descent lemma:

$$F(w_{t+1}) \leq F(w_t) + \langle \nabla F(w_t), w_{t+1} - w_t \rangle + \frac{L}{2}\|w_{t+1} - w_t\|_2^2. \tag{C.7}$$

Substituting $w_{t+1} = w_t - \eta\, \tilde{g}_t$ into Equation (C.7), we obtain

$$F(w_{t+1}) \leq F(w_t) - \eta\, \langle \nabla F(w_t), \tilde{g}_t \rangle + \frac{L\eta^2}{2}\|\tilde{g}_t\|_2^2. \tag{C.8}$$

Decomposition and Alignment. We write the normalized gradient as

$$\tilde{g}_t = \nabla F(w_t) + \delta_t, \tag{C.9}$$

where δ_t is the error due to the nonlinearity of the normalization. Under the moment conditions (Assumption B.4) and by using Lemma B.2 (Formal Alignment of Normalized Gradient), we can bound the deviation:

$$\langle \nabla F(w_t), \tilde{g}_t \rangle \geq (1-\varepsilon)\|\nabla F(w_t)\|_2^2, \tag{C.10}$$

for some ε (dependent on $\epsilon/\sigma_{\min}$).

Bounding the Norm of $\tilde{g}_t$. As in the convex setting (see Lemma B.4), the group normalization yields

$$\|\tilde{g}_t\|_2^2 \leq \rho\, n, \tag{C.11}$$

where n is the total number of parameters and $\rho \in [0, 1)$ is the variance contraction factor.

Combining the Bounds. Substitute the alignment bound Equation (C.10) and the norm bound Equation (C.11) into Equation (C.8):

$$F(w_{t+1}) \leq F(w_t) - \eta\,(1-\varepsilon)\|\nabla F(w_t)\|_2^2 + \frac{L\eta^2}{2}\rho\, n. \tag{C.12}$$

Rearrange Equation (C.12) to obtain an inequality on the gradient norm:

$$\eta\,(1-\varepsilon)\|\nabla F(w_t)\|_2^2 \leq F(w_t) - F(w_{t+1}) + \frac{L\eta^2}{2}\rho\, n. \tag{C.13}$$

Telescoping over Iterations. Summing Equation (C.13) over $t = 0, \ldots, T-1$ results in:

$$\eta\,(1-\varepsilon)\sum_{t=0}^{T-1}\|\nabla F(w_t)\|_2^2 \leq F(w_0) - F(w_T) + \frac{L\eta^2}{2}\rho\, nT. \tag{C.14}$$

Since $F(w_T) \geq F^*$ (with $F^* = \min_w F(w)$ or a known lower bound), we have:

$$\eta\,(1-\varepsilon)\sum_{t=0}^{T-1}\|\nabla F(w_t)\|_2^2 \leq F(w_0) - F^* + \frac{L\eta^2}{2}\rho\, nT. \tag{C.15}$$

Dividing both sides of Equation (C.15) by $\eta\,(1-\varepsilon)T$ yields:

$$\frac{1}{T}\sum_{t=0}^{T-1}\|\nabla F(w_t)\|_2^2 \leq \frac{F(w_0) - F^*}{\eta\,(1-\varepsilon)T} + \frac{L\eta\,\rho\, n}{2(1-\varepsilon)}. \tag{C.16}$$

Choosing the Learning Rate. By selecting the step-size as

$$\eta = \min\Big\{\frac{1}{L},\, \frac{c}{\sqrt{T}}\Big\}, \tag{C.17}$$

for some constant $c > 0$, the two terms on the right-hand side of Equation (C.16) balance and both decay as $O(1/\sqrt{T})$. Consequently, we obtain

$$\min_{t=0,\ldots,T-1}\|\nabla F(w_t)\|_2^2 = O\Big(\frac{1}{\sqrt{T}}\Big), \tag{C.18}$$

which establishes that Milo converges (in terms of the squared gradient norm) at a sublinear rate despite nonconvexity.

C.3 Summary of the Proof

1. **Smoothness:** Using L-smoothness we derived
$$F(w_{t+1}) \leq F(w_t) - \eta\,\langle \nabla F(w_t), \tilde{g}_t \rangle + \frac{L\eta^2}{2}\|\tilde{g}_t\|_2^2.$$
2. **Alignment:** The error decomposition $\tilde{g}_t = \nabla F(w_t) + \delta_t$ combined with Lemma B.2 gives
$$\langle \nabla F(w_t), \tilde{g}_t \rangle \geq (1-\varepsilon)\|\nabla F(w_t)\|_2^2.$$
3. **Norm Bound:** Lemma B.4 ensures that $\|\tilde{g}_t\|_2^2 \leq \rho\, n$.
4. **Telescoping:** Summing over iterations and rearranging leads to
$$\frac{1}{T}\sum_{t=0}^{T-1}\|\nabla F(w_t)\|_2^2 \leq \frac{F(w_0) - F^*}{\eta\,(1-\varepsilon)T} + \frac{L\eta\,\rho\, n}{2(1-\varepsilon)}.$$
5. **Rate:** With the choice of step-size in Equation (C.17), we conclude that
$$\min_{t=0,\ldots,T-1}\|\nabla F(w_t)\|_2^2 = O\Big(\frac{1}{\sqrt{T}}\Big).$$

C.4 Discussion and Practical Implications

The preceding analysis shows that even when F is non-convex, Milo's normalized gradients drive a controlled descent of the objective and guarantee

$$\min_{t=0,\dots,T-1} \|\nabla F(w_t)\|_2^2 \;=\; O\big(T^{-1/2}\big).$$

The key technical ingredients are:

- **Smoothness:** L-smoothness ensures F does not change too abruptly, yielding the descent lemma.
- **Variance Contraction:** Group-wise normalization (Assumption B.3 and Lemma B.4) tightly controls $\|\tilde{g}_t\|$.
- **Alignment Control:** Lemma B.2 bounds the misalignment δ_t so that $\nabla F(w_t), \tilde{g}_t \;\geq\; (1 - \varepsilon)\|\nabla F(w_t)\|^2$.

By choosing $\eta = \min\{1/L,\, c/\sqrt{T}\}$, the telescoped one-step bounds give the claimed $O(T^{-1/2})$ rate in the squared gradient norm. Crucially, this leverages the same normalization strategy that yields variance contraction in the convex case, now coupled with careful alignment control in the non-convex regime.

This result provides rigorous theoretical backing for Milo's empirical success across both convex and non-convex problems, explaining why normalized updates can both stabilize training and accelerate convergence even on highly non-convex objectives.

D Experimentation

D.1 Experiment Hardware

Table 3: Static Hardware Configuration

Property	Value
Platform	Windows-11-10.0.22631-SP0
Processor	AMD Ryzen 9 9900X
Physical CPU cores	12
Logical CPU threads	24
Total system memory	61.6 GB
Disk capacity used	931.4 GB
CUDA available	True
Compute type	GPU
GPU count	1
GPU model	NVIDIA GeForce RTX 5070 Ti,
GPU memory	16.0 GB

D.2 Libraries Used

We rely on the following open-source software (version, license, URL, citation):

- **PyTorch** (v2.6.0; BSD 3-clause License; `https://pytorch.org`) [Paszke et al., 2019]
- **torchvision** (v0.21.0; BSD 3-clause License; `https://github.com/pytorch/vision`) [PyTorch Contributors, 2023]
- **seaborn** (v0.13.2; BSD License; `https://seaborn.pydata.org`) [Waskom, 2021]
- **matplotlib** (v3.10.0; PSF License; `https://matplotlib.org`) [Hunter, 2007]
- **NumPy** (v2.1.3; BSD License; `https://numpy.org`) [van der Walt et al., 2011]
- **NovoGrad** optimizer (preprint; Apache-2.0; `https://arxiv.org/abs/1905.11286`) [Ginsburg et al., 2019]
- **pandas** (v2.2.3; BSD License; `https://pandas.pydata.org`) [McKinney, 2010]
- **JSON module** (Python 3.10 stdlib; PSF License; `https://docs.python.org/3/library/json.html`) [Crockford, 2006]

- **Optuna** (v4.3.0; MIT License; `https://optuna.org`) [Akiba et al., 2019]
- **scikit-learn** (v1.6.1; BSD License; `https://scikit-learn.org`) [Pedregosa et al., 2011]
- **SciPy** (v1.15.1; BSD License; `https://scipy.org`) [Virtanen et al., 2020]
- **itertools** (Python 3.10 stdlib; PSF License; `https://docs.python.org/3/library/itertools.html`) [Python Software Foundation, 2023]
- **Gymnasium** (v1.1.0; MIT License; `https://gymnasium.farama.org`) [Brockman et al., 2016]
- **Transformers** (v4.30.0; Apache-2.0; `https://github.com/huggingface/transformers`) [Wolf et al., 2020]
- **train-llm-from-scratch** (MIT License; `https://github.com/FareedKhan-dev/train-llm-from-scratch`) [FareedKhan-dev, 2025]
- **NovoGrad-pytorch** (MIT License; `https://github.com/lonePatient/NovoGrad-pytorch`) [lonePatient, 2019]

D.3 Datasets

We conduct all experiments on the following publicly released datasets:

- **MNIST** (Public Domain; `http://yann.lecun.com/exdb/mnist/`) [LeCun et al., 1998]
- **CIFAR-10** (2009 Technical Report; MIT License; `https://www.cs.toronto.edu/~kriz/cifar.html`) [Krizhevsky and Hinton, 2009]
- **CIFAR-100** (2009 Technical Report; MIT License; `https://www.cs.toronto.edu/~kriz/cifar.html`) [Krizhevsky and Hinton, 2009]
- **The Pile** (Apache 2.0 License; `https://pile.eleuther.ai/`) [Gao et al., 2020]

D.4 Error Bar Computation

Error bars in our experiments quantify the uncertainty in each metric across multiple runs. They are computed as follows:

1. Calculation

- **Standard Error of the Mean (SEM):**

$$\mathrm{SEM} = \frac{\mathrm{SD}}{\sqrt{n}}$$

  where SD is the sample standard deviation and n is the number of independent runs.

- **95% Confidence Interval (CI):**

$$\mathrm{CI} = t_{0.975,\,n-1} \;\times\; \mathrm{SEM},$$

  with $t_{0.975,\,n-1}$ the two-sided critical value from the Student's t-distribution.

- For each epoch or step, we aggregate the metric values over n runs, compute the mean, SD, SEM, and then plot the mean $\pm$ CI as error bands.

2. Sources of Variability

- *Run-to-run stochasticity*: random initialization, data shuffling.
- *Optimizer/model differences*: varying update rules or architectures.
- *Data subsampling*: different train/validation/test splits.
- *Hardware noise*: floating-point precision and device variability.

3. Assumptions

- **Normality**: metric distributions are approximately Gaussian.
- **Independence**: each run is statistically independent.
- **Sufficient sample size**: $n > 1$ for reliable SEM and CI estimates.
- **Consistency**: identical hyperparameters and data splits across runs.

4. Implementation The function `calculate_statistics()` computes mean, SD, SEM, and CI for each metric, and `plot_seaborn_style_with_error_bars()` renders the resulting error bands in the figures.

D.5 Experiment 1: Loss Map Predictions

The objective of this experiment is to evaluate multiple optimizers on different loss maps to provide an understanding of their behavior in complex loss environments. In each optimization problem, the optimizers were run 5 times and their outputs were averaged, however, since it is a static environment there was never any variation in their behavior.

D.5.1 Himmelblau Function and Its Gradient

The Himmelblau function is defined as:

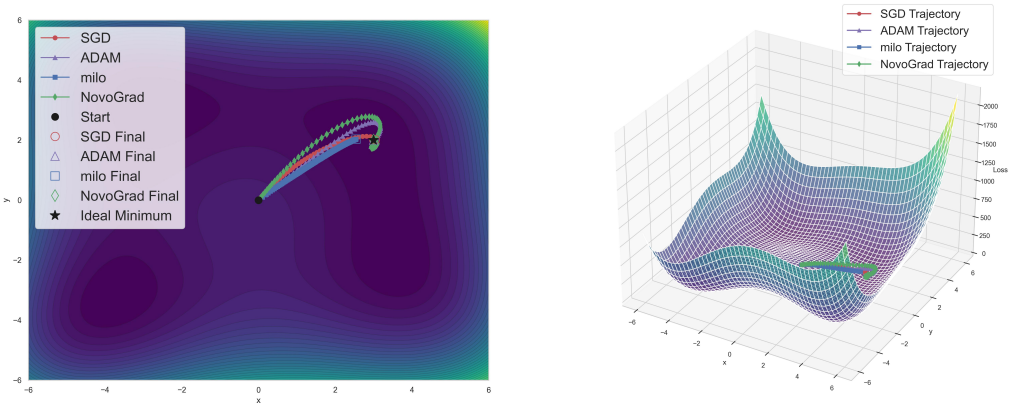
$$f(x_1, x_2) = \left(x_1^2 + x_2 - 11\right)^2 + \left(x_1 + x_2^2 - 7\right)^2. \tag{D.1}$$

For the analytic gradient, we define:

$$u = x_1^2 + x_2 - 11, \qquad v = x_1 + x_2^2 - 7. \tag{D.2}$$

Then, the partial derivatives are:

$$\frac{\partial f}{\partial x_1} = 4x_1\, u + 2v, \qquad \frac{\partial f}{\partial x_2} = 2u + 4x_2\, v. \tag{D.3}$$



(a) Himmelblau Loss 2D

(b) Himmelblau Loss 3D

Figure 6: Himmelblau Loss Function: 2D Contour vs. 3D Loss Map

Experimental Setup

- **Domain:** $[-6, 6] \times [-6, 6]$
- **Initial Point:** $(0, 0)$
- **Iterations:** 150

Observations and Analysis

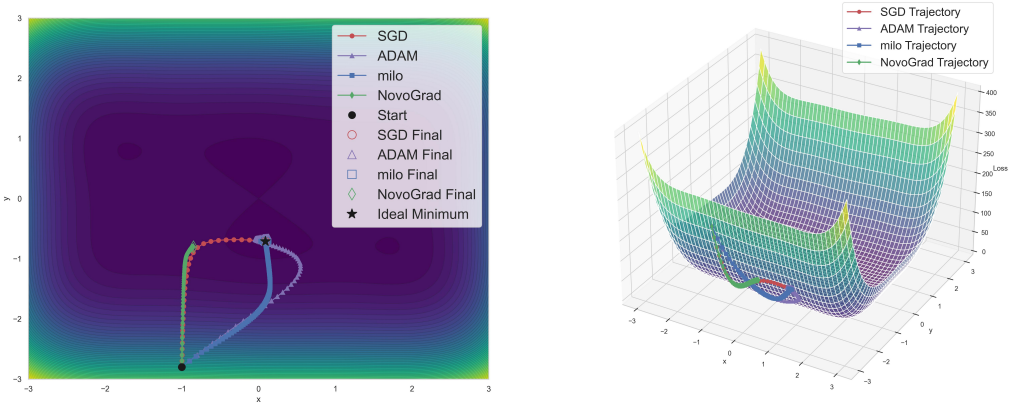
- **SGD:** The SGD trajectory follows a generally follows a more direct path, quickly descending into a low-loss valley, curving towards the end, and stabilizing near the minimum.
- **Adam:** Adam has a less direct path, overshooting and eventually curving around to reach the objective.
- **NovoGrad:** NovoGrad, despite taking normalized steps, overshoots with similar behavior to Adam, eventually curving around to reach the objective.
- **Milo:** Milo takes normalized steps, taking a more conservative path since the large gradients are scaled down, following a uniform trajectory toward the basin.

Key Takeaway: All four optimizers ultimately approach a global minimum, but SGD and Milo exhibit a more stable and direct convergence compared to Adam and NovoGrad.

D.5.2 Camel6 Function

The Camel6 function is defined by:

$$f(x_1, x_2) = \left(4 - 2.1\,x_1^2 + \frac{x_1^4}{3}\right) x_1^2 + x_1 x_2 + \left(-4 + 4\,x_2^2\right) x_2^2. \tag{D.4}$$



(a) Camel6 Loss 2D (b) Camel6 Loss 3D

Figure 7: Camel6 Loss Function: 2D Contour vs. 3D Loss Map

Experimental Setup

- **Domain:** $[-3, 3] \times [-3, 3]$
- **Initial Point:** $(-1, -2.8)$
- **Iterations:** 150

Observations and Analysis

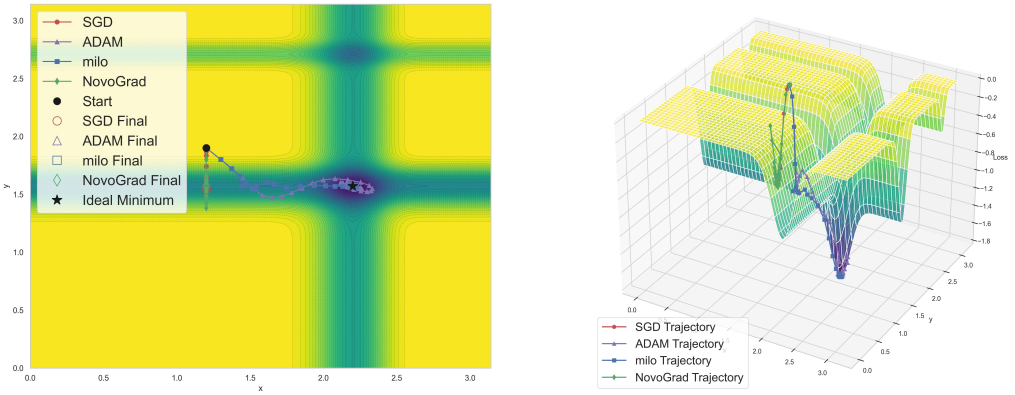
- **SGD:** The trajectory takes consistent steps, taking a largely indirect path to the local minima and eventually converging to the global.
- **Adam:** Similar to SGD Adam takes an indirect path to the local minima, then shifts to a smaller step size with a more direct path to the eventual global minima.
- **NovoGrad:** Novograd follows a similar path as SGD, however is comes to an early stop after it reaches the local minima, never reaching the global.
- **Milo:** With normalized updates, Milo takes consistent steps, initialy following the same path as Adam, but deviating towards the global minima for a more direct convergence than the other optimizers.

Key Takeaway: In the Camel6 landscape, Milo's consistent step sizes help it maintain stable progress even when facing plateaus, with SGD and Adam taking more indirect paths. NovoGrad never reaches convergence in this problem, instead settling in the local minima it quickly reached.

D.5.3 Michalewicz Function

For a d-dimensional input x and a parameter m (typically $m = 10$), the Michalewicz function is:

$$f(x) = -\sum_{i=1}^{d} \sin(x_i) \left[\sin\left(\frac{i\,x_i^2}{\pi}\right)\right]^{2m}. \tag{D.5}$$



(a) Michalewicz Loss 2D (b) Michalewicz Loss 3D

Figure 8: Michalewicz Loss Function: 2D Contour vs. 3D Loss Map

Experimental Setup

- **Domain:** $[-10, 10] \times [-10, 10]$
- **Initial Point:** $(2.4, -4.3)$
- **Iterations:** 150

Observations and Analysis

- **SGD:** For the Michalewicz experiment, SGD quickly became stuck in the local minima within the basin, never reaching the global.
- **Adam:** Adam's adaptive step size helps it overcome the local traps, reaching the global minima, however it takes a more indirect path and briefly overshoots the goal.
- **NovoGrad:** NovoGrad, similar to SGD, also becomes quickly stuck in the local minima within the basin.
- **Milo:** Milo successfully reaches the global minima, making consistent and direct progress towards the goal with no overshooting.

Key Takeaway: De Jong's function, with its many local minima, highlights Milo's strength in sustaining movement when the gradient is very small. Adam adapts well but again overshoots the goal before circling back. Additionally, it is shown that SGD and NovoGrad can become trapped in shallow basins.

D.5.4 De Jong Function 5

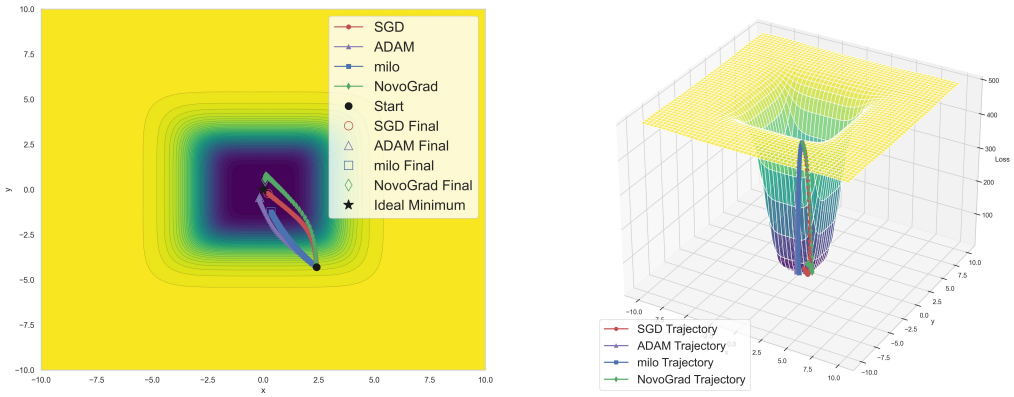
De Jong's fifth function is a multimodal test function defined for a 2-dimensional input vector $x = (x_1, x_2)$. It is constructed using a fixed grid of constants and is known for its complex landscape with many local minima. The function is given by:

$$f(x) = \left[0.002 + \sum_{i=1}^{25} \frac{1}{i + (x_1 - a_{1i})^6 + (x_2 - a_{2i})^6} \right]^{-1}, \tag{D.6}$$

where the constants a_{1i} and a_{2i} form a 5×5 grid of values from the set $\{-32, -16, 0, 16, 32\}$, specifically:

$$a_{1i} = \text{tile}([-32, -16, 0, 16, 32], 5),$$
$$a_{2i} = \text{repeat}([-32, -16, 0, 16, 32], 5).$$

This function is highly sensitive to small changes in x, making it useful for evaluating the performance of optimization algorithms on rugged search landscapes.



(a) DeJong Loss 2D (b) DeJong Loss 3D

Figure 9: DeJong's Fifth Function Loss Function: 2D Contour vs. 3D Loss Map

Experimental Setup

- **Domain:** $[0, \pi] \times [0, \pi]$
- **Initial Point:** $(1.2, 1.9)$
- **Iterations:** 250

Observations and Analysis

- **SGD:** Due to the steep descent structure of the function, SGD curves around the edge until it corrects towards the global minima, reaching it.
- **Adam:** Adam's adaptive updates help it follow a smoother descent, reaching the final global minima smoothly.
- **NovoGrad:** NovoGrad follows a similar path around the edge as SGD, eventually overshooting slightly before reaching the global minima.
- **Milo:** Milo's normalized update rule maintains a consistent step size even when the gradients are small, allowing steady progress; however, it stops right as the bottom of the basin is reached, not continuing further as the other optimizers do.

Key Takeaway:

D.5.5 Experiment 1 Analysis

The loss map experiments reveal that while SGD, Adam, NovoGrad, and Milo each have their strengths, the proposed Milo optimizer, with its normalized, group-based updates, can offer stable and consistent progress in challenging optimization landscapes. In particular, Milo shows promise in escaping flat regions and avoiding overshooting due to its consistent step size. However, as shown with DeJong, the consistent step size can potentially lead to it stopping earlier than desired.

Table 4: Key Experimental Settings, Experiments 2 - 4

Setting	Specification
Data splits	Validation: 15%, Test: 10%
Batch size	128
Epochs	10
Runs per optimizer	5
Hyperparameter tuning	5 trials per optimizer using log-uniform and uniform sampling over specified parameter grids
Base learning rates	`LOGISTIC`, `MULTILAYER`: 0.05; `DEEPCNN`, `RESNET18`: 0.005
Optimizers	Milo, Milo_LW, SGD, AdaGrad, AdamW, NovoGrad
Models & Datasets	`LOGISTIC`, `MULTILAYER`: MNIST; `RESNET18`: CIFAR-100
Transforms	MNIST: flatten to vector; CIFAR: tensor only (no augmentations)

D.6 Hyperparameter Tuning:

Table 5: Hyperparameter search spaces for all optimizers

Optimizer	Search Space
SGD	`lr`: log-uniform $[0.005, 0.1]$; `momentum`: uniform $[0.45, 0.99]$; `weight_decay`: log-uniform $[0.0005, 0.01]$
ADAGRAD	`lr`: log-uniform $[0.005, 0.1]$; `lr_decay`: uniform $[0.0, 0.1]$; `weight_decay`: log-uniform $[0.0005, 0.01]$; `eps`: log-uniform $[10^{-10}, 10^{-6}]$
ADAMW	`lr`: log-uniform $[0.005, 0.1]$; `weight_decay`: log-uniform $[0.0005, 0.01]$; `betas`: $\{(0.9, 0.98), (0.95, 0.99)\}$; `eps`: log-uniform $[10^{-9}, 10^{-6}]$
NovoGrad	`lr`: log-uniform $[0.005, 0.1]$; `betas`: $\{(0.9, 0.98), (0.95, 0.99)\}$; `weight_decay`: log-uniform $[0.0005, 0.01]$
Milo	– (default settings)
Milo_{LW}	– (default settings)

Table 6: Default learning rates and optimizer parameter settings

Component	Default Setting
Base learning rate	
LOGISTIC	0.05
MULTILAYER	0.05
DEEPCNN	0.005
RESNET18	0.005
Optimizer defaults	
SGD	`momentum`=0.9; `nesterov`=True; `weight_decay`=0.0001
ADAGRAD	`lr_decay`=0; `weight_decay`=0.0; `eps`=1e-10
ADAMW	`betas`=(0.9, 0.999); `eps`=1e-8; `weight_decay`=0.01
Milo	`layer_wise`=False; `scale_aware`=True; `scale_factor`=0.2; `momentum`=0.9; `use_cached_mapping`=False; `foreach`=True
Milo_{LW}	`layer_wise`=True; `scale_aware`=True; `scale_factor`=0.2; `momentum`=0.9; `use_cached_mapping`=True; `foreach`=True
NovoGrad	`betas`=(0.9, 0.99); `weight_decay`=0.001; `grad_averaging`=True

D.7 Experiment 2: Logistic Regression

This experiment evaluates a convex multi-class logistic regression using the MNIST Dataset ([LeCun et al., 1998]), just as was done in the study that proposed Adam. The goal is to compare optimizers with respect to training loss, convergence speed, and test accuracy. The model uses L2 regularization with a single linear layer mapping a 784-dimensional input to 10 classes. Visual results can be seen in Figure 3.

Table 7: Logistic Regression Model Architecture Layers

Layer	**Description**
Input Layer	Flattened input vector of dimension 784 (28×28)
Linear	Single fully connected layer mapping 784 to 10 (number of classes)

Table 8: Logistic Regression Training Results

Optimizer	**Epoch Group**	**Loss**	**Accuracy (%)**	**F1 Score**
Milo	1–4	0.2930	92.42	0.9233
	5–8	0.2732	93.07	0.9298
	9–10	0.2714	93.19	0.9310
Milo_{LW}	1–4	0.2594	92.87	0.9277
	5–8	0.2312	93.62	0.9354
	9–10	0.2258	93.75	0.9367
SGD	1–4	0.3076	91.47	0.9135
	5–8	0.2818	92.23	0.9212
	9–10	0.2773	92.34	0.9224
ADAGRAD	1–4	0.4866	88.03	0.8784
	5–8	0.4720	88.29	0.8810
	9–10	0.4687	88.33	0.8814
ADAMW	1–4	0.3103	91.48	0.9135
	5–8	0.2593	92.83	0.9273
	9–10	0.2490	93.18	0.9309
Novograd	1–4	0.3774	89.87	0.8973
	5–8	0.2928	91.83	0.9172
	9–10	0.2827	92.13	0.9202

Table 9: Average Runtime and Memory Increase for Logistic Regression Experiment (5 runs)

Optimizer	**Avg Runtime (s)**	**Avg Memory (MB)**
MILO	32.51	63.5
MILO_{LW}	32.66	2.0
SGD	29.71	0.0
ADAGRAD	29.76	0.0
ADAMW	30.03	16.4
NOVOGRAD	32.16	4.1

D.7.1 Statistical Analysis

Table 10: Pairwise Significance Tests for Validation Loss

Optimizer A	Optimizer B	Mean A	Mean B	Better	p-value	Sig.
MILO	MILO_{LW}	0.350833	0.304008	MILO_{LW}	8.13499e-08	***
MILO	SGD	0.350833	0.306714	SGD	3.62048e-06	***
MILO	ADAMW	0.350833	0.287661	ADAMW	9.31980e-07	***
MILO	ADAGRAD	0.350833	0.481010	MILO	9.90140e-11	***
MILO	NOVOGRAD	0.350833	0.309194	NOVOGRAD	4.23780e-06	***
MILO_{LW}	SGD	0.304008	0.306714	MILO_{LW}	1.32372e-01	
MILO_{LW}	ADAMW	0.304008	0.287661	ADAMW	9.06983e-05	***
MILO_{LW}	ADAGRAD	0.304008	0.481010	MILO_{LW}	2.49227e-13	***
MILO_{LW}	NOVOGRAD	0.304008	0.309194	MILO_{LW}	1.67816e-02	*
SGD	ADAMW	0.306714	0.287661	ADAMW	1.66218e-08	***
SGD	ADAGRAD	0.306714	0.481010	SGD	2.60849e-11	***
SGD	NOVOGRAD	0.306714	0.309194	SGD	2.46024e-02	*
ADAMW	ADAGRAD	0.287661	0.481010	ADAMW	4.96532e-11	***
ADAMW	NOVOGRAD	0.287661	0.309194	ADAMW	9.13035e-09	***
ADAGRAD	NOVOGRAD	0.481010	0.309194	NOVOGRAD	1.79728e-11	***

Table 11: Pairwise Significance Tests for Validation F1-Score

Optimizer A	Optimizer B	Mean A	Mean B	Better	p-value	Sig.
MILO	MILO_{LW}	0.912861	0.916955	MILO_{LW}	1.75279e-03	**
MILO	SGD	0.912861	0.914048	SGD	2.41945e-01	
MILO	ADAGRAD	0.912861	0.876378	MILO	5.42161e-10	***
MILO	ADAMW	0.912861	0.919612	ADAMW	5.82021e-05	***
MILO	NOVOGRAD	0.912861	0.913075	NOVOGRAD	7.63541e-01	
MILO_{LW}	SGD	0.916955	0.914048	MILO_{LW}	1.56224e-02	*
MILO_{LW}	ADAGRAD	0.916955	0.876378	MILO_{LW}	2.24596e-10	***
MILO_{LW}	ADAMW	0.916955	0.919612	ADAMW	9.94675e-03	**
MILO_{LW}	NOVOGRAD	0.916955	0.913075	MILO_{LW}	1.93199e-03	**
SGD	ADAGRAD	0.914048	0.876378	SGD	3.79011e-10	***
SGD	ADAMW	0.914048	0.919612	ADAMW	4.33298e-04	***
SGD	NOVOGRAD	0.914048	0.913075	SGD	2.51483e-01	
ADAGRAD	ADAMW	0.876378	0.919612	ADAMW	8.47333e-09	***
ADAGRAD	NOVOGRAD	0.876378	0.913075	NOVOGRAD	9.06177e-08	***
ADAMW	NOVOGRAD	0.919612	0.913075	ADAMW	3.71358e-05	***

D.8 Experiment 3: Multilayer Neural Networks

The objective of this experiment is to test the optimizer on a non-convex problem by using a multilayer perception. For this study, our model choice was driven by the previous Adam publication, using two fully connected layers with 1000 ReLU units each, with dropout and weight decay on a minibatch of 128, on the MNIST image dataset ([LeCun et al., 1998]).

Table 12: Multilayer NN Model Architecture Layers

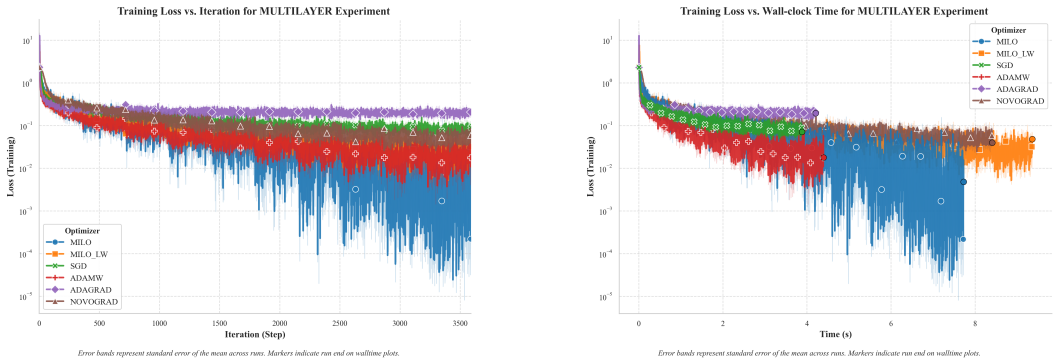
Layer	**Description**
Linear	Fully connected layer: input dimension 784 to hidden dimension 128
ReLU	Activation function
Dropout	Dropout layer with probability 0.2
Linear	Fully connected layer: 128 to 10 (number of classes)

Table 13: Multilayer Network Training Results

Optimizer	Epoch Group	Loss	Accuracy (%)	F1 Score
Milo	1–4	0.0777	98.24	0.9822
	5–8	0.0161	99.66	0.9966
	9–10	0.0050	99.90	0.9990
Milo_{LW}	1–4	0.1034	96.95	0.9692
	5–8	0.0372	98.88	0.9887
	9–10	0.0215	99.36	0.9936
SGD	1–4	0.1695	95.32	0.9528
	5–8	0.0921	97.63	0.9762
	9–10	0.0756	98.16	0.9815
ADAGRAD	1–4	0.2204	93.98	0.9391
	5–8	0.2013	94.53	0.9447
	9–10	0.1968	94.66	0.9461
ADAMW	1–4	0.0791	97.58	0.9756
	5–8	0.0259	99.14	0.9913
	9–10	0.0144	99.51	0.9951
Novograd	1–4	0.1655	95.24	0.9519
	5–8	0.0637	98.18	0.9817
	9–10	0.0461	98.72	0.9871

Table 14: Average Runtime and Memory Increase for Multilayer Experiment (5 runs)

Optimizer	Avg Duration (s)	Avg Memory (MB)
MILO	34.37	18.4
MILO_{LW}	36.05	0.0
SGD	30.28	0.0
ADAGRAD	30.74	0.0
ADAMW	30.87	0.0
NOVOGRAD	34.24	0.0



(a) MLP Train Loss

(b) MLP Train Walltime Loss

Figure 10: Comparison of MLP training cost by iterations and wall time.

D.8.1 Statistical Analysis

Table 15: Pairwise Significance Tests for Validation Loss (Final Epoch)

Optimizer A	Optimizer B	Mean A	Mean B	Better	p-value	Sig.
MILO	MILO_{LW}	0.225943	0.150747	MILO_{LW}	1.51986e-04	***
MILO	SGD	0.225943	0.104657	SGD	1.10944e-04	***
MILO	ADAMW	0.225943	0.121495	ADAMW	2.48203e-05	***
MILO	ADAGRAD	0.225943	0.215592	ADAGRAD	3.16927e-01	
MILO	NOVOGRAD	0.225943	0.0872035	NOVOGRAD	2.96707e-05	***
MILO_{LW}	SGD	0.150747	0.104657	SGD	4.70080e-04	***
MILO_{LW}	ADAMW	0.150747	0.121495	ADAMW	1.64005e-03	**
MILO_{LW}	ADAGRAD	0.150747	0.215592	MILO_{LW}	1.31936e-05	***
MILO_{LW}	NOVOGRAD	0.150747	0.0872035	NOVOGRAD	2.06945e-05	***
SGD	ADAMW	0.104657	0.121495	SGD	1.71513e-02	*
SGD	ADAGRAD	0.104657	0.215592	SGD	2.16145e-05	***
SGD	NOVOGRAD	0.104657	0.0872035	NOVOGRAD	9.02238e-04	***
ADAMW	ADAGRAD	0.121495	0.215592	ADAMW	7.75432e-07	***
ADAMW	NOVOGRAD	0.121495	0.0872035	NOVOGRAD	4.28650e-04	***
ADAGRAD	NOVOGRAD	0.215592	0.0872035	NOVOGRAD	1.16408e-06	***

Table 16: Pairwise Significance Tests for Validation F1-Score (Final Epoch)

Optimizer A	Optimizer B	Mean A	Mean B	Better	p-value	Sig.
MILO	MILO_{LW}	0.973116	0.968024	MILO	9.10593e-04	***
MILO	SGD	0.973116	0.971065	MILO	5.32615e-02	
MILO	ADAMW	0.973116	0.972800	MILO	8.14991e-01	
MILO	ADAGRAD	0.973116	0.941102	MILO	6.55427e-06	***
MILO	NOVOGRAD	0.973116	0.973147	NOVOGRAD	9.80252e-01	
MILO_{LW}	SGD	0.968024	0.971065	SGD	1.56224e-02	**
MILO_{LW}	ADAMW	0.968024	0.972800	ADAMW	6.94322e-03	**
MILO_{LW}	ADAGRAD	0.968024	0.941102	MILO_{LW}	3.03333e-05	***
MILO_{LW}	NOVOGRAD	0.968024	0.973147	NOVOGRAD	2.48226e-03	**
SGD	ADAMW	0.971065	0.972800	ADAMW	1.83324e-01	
SGD	ADAGRAD	0.971065	0.941102	SGD	3.85788e-05	***
SGD	NOVOGRAD	0.971065	0.973147	NOVOGRAD	8.87216e-02	
ADAMW	ADAGRAD	0.972800	0.941102	ADAMW	1.98429e-06	***
ADAMW	NOVOGRAD	0.972800	0.973147	NOVOGRAD	8.11459e-01	
ADAGRAD	NOVOGRAD	0.941102	0.973147	NOVOGRAD	3.13313e-06	***

D.9 Experiment 4: Convolutional Neural Networks (CNNs)

This experiment evaluates a deep Residual Network, specifically ResNet-18 [He et al., 2016], on the CIFAR-100 dataset [Krizhevsky and Hinton, 2009] to test optimizer performance in a finer-grained classification setting. Compared to CIFAR-10, CIFAR-100 has 100 target classes, which stresses both the representation capacity of the network and the optimizer's ability to generalize. We use the standard ResNet-18 architecture, modifying only the final linear layer to output 100 logits. Input images are augmented with 4-pixel zero-padding, then per-channel normalized to zero mean and unit variance. All training runs use a batch size of 128.

Table 17: Deep CNN Model Architecture Layers

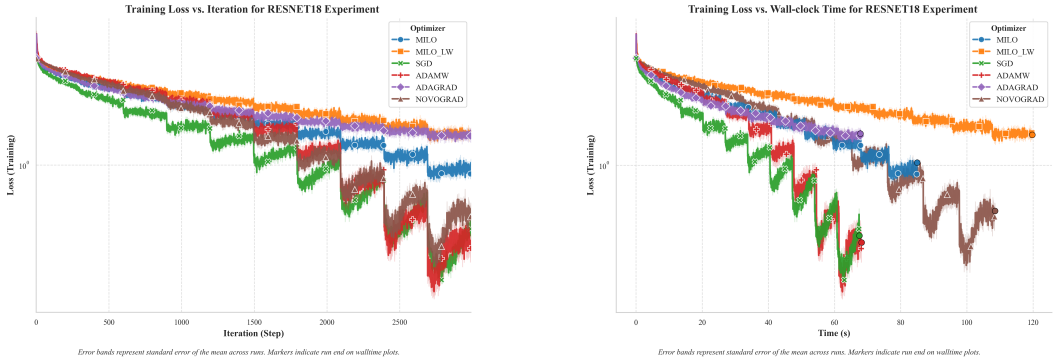
Layer	Description
Conv2d	Input: 3 channels, Output: 32 channels, Kernel size: 3, Padding: 1
ReLU	Activation function
Conv2d	Input: 32 channels, Output: 32 channels, Kernel size: 3, Padding: 1
ReLU	Activation function
MaxPool2d	Pooling layer with kernel size 2
Conv2d	Input: 32 channels, Output: 64 channels, Kernel size: 3, Padding: 1
ReLU	Activation function
Conv2d	Input: 64 channels, Output: 64 channels, Kernel size: 3, Padding: 1
ReLU	Activation function
MaxPool2d	Pooling layer with kernel size 2
Flatten	Converts feature maps into a 1D vector
Linear	Fully connected layer: 4096 (i.e. $64 \times 8 \times 8$) to 128
ReLU	Activation function
Linear	Fully connected layer: 128 to 10 (number of classes)

Table 18: ResNet-18 Training Results

Optimizer	Epoch Group	Loss	Accuracy (%)	F1 Score
Milo	1–4	2.6884	33.03	0.3130
	5–8	1.4598	61.59	0.6113
	9–10	0.7754	80.63	0.8064
Milo_{LW}	1–4	3.0691	23.40	0.2103
	5–8	2.0448	44.88	0.4389
	9–10	1.5241	57.17	0.5651
SGD	1–4	2.6914	34.53	0.3335
	5–8	1.2595	63.90	0.6395
	9–10	0.6575	81.21	0.8144
ADAGRAD	1–4	2.7093	32.94	0.3112
	5–8	1.7878	55.53	0.5470
	9–10	1.4423	65.83	0.6533
ADAMW	1–4	3.2084	22.69	0.2104
	5–8	1.1366	68.65	0.6847
	9–10	0.2509	92.53	0.9258
NOVOGRAD	1–4	3.1196	24.28	0.2205
	5–8	1.1359	66.98	0.6673
	9–10	0.3902	87.82	0.8780

Table 19: Average Runtime and Memory Increase for ResNet-18 Experiment (5 runs)

Optimizer	Avg Duration (s)	Avg Memory (MB)
MILO	148.53	59.4
MILO_{LW}	182.82	2.0
SGD	130.56	14.3
ADAGRAD	130.69	0.0
ADAMW	131.10	14.3
NOVOGRAD	172.25	4.1



(a) ResNet18 Train Loss

(b) ResNet18 Walltime Loss

Figure 11: Comparison of ResNet18 training cost by iterations and wall time.

D.9.1 Statistical Analysis

Table 20: Pairwise Significance Tests for Validation Loss

Opt. A	Opt. B	Mean A	Mean B	Better	p-value	Sig.
MILO	MILO_{LW}	2.09638	2.11451	MILO	0.428833	
MILO	SGD	2.09638	2.16893	MILO	0.044138	*
MILO	ADAGRAD	2.09638	2.48136	MILO	4.41550e-07	***
MILO	ADAMW	2.09638	3.38785	MILO	1.17780e-06	***
MILO	NOVOGRAD	2.09638	2.29087	MILO	0.000535	***
MILO_{LW}	SGD	2.11451	2.16893	MILO_{LW}	0.081725	
MILO_{LW}	ADAGRAD	2.11451	2.48136	MILO_{LW}	1.24390e-06	***
MILO_{LW}	ADAMW	2.11451	3.38785	MILO_{LW}	6.11940e-06	***
MILO_{LW}	NOVOGRAD	2.11451	2.29087	MILO_{LW}	0.001571	**
SGD	ADAGRAD	2.16893	2.48136	SGD	9.18010e-06	***
SGD	ADAMW	2.16893	3.38785	SGD	6.27030e-07	***
SGD	NOVOGRAD	2.16893	2.29087	SGD	0.009368	**
ADAGRAD	ADAMW	2.48136	3.38785	ADAGRAD	8.92350e-06	***
ADAGRAD	NOVOGRAD	2.48136	2.29087	NOVOGRAD	0.000613	***
ADAMW	NOVOGRAD	3.38785	2.29087	NOVOGRAD	6.09120e-07	***

Table 21: Pairwise Significance Tests for Validation F1-Score

Opt. A	Opt. B	Mean A	Mean B	Better	p-value	Sig.
MILO	MILO_{LW}	0.481343	0.441000	MILO	0.000321	***
MILO	SGD	0.481343	0.454009	MILO	0.003989	**
MILO	ADAGRAD	0.481343	0.365256	MILO	1.58180e-07	***
MILO	ADAMW	0.481343	0.411299	MILO	0.000352	***
MILO	NOVOGRAD	0.481343	0.492262	NOVOGRAD	0.113014	
MILO_{LW}	SGD	0.441000	0.454009	SGD	0.050683	
MILO_{LW}	ADAGRAD	0.441000	0.365256	MILO_{LW}	1.69300e-06	***
MILO_{LW}	ADAMW	0.441000	0.411299	MILO_{LW}	0.024678	*
MILO_{LW}	NOVOGRAD	0.441000	0.492262	NOVOGRAD	3.15430e-06	***
SGD	ADAGRAD	0.454009	0.365256	SGD	6.67760e-07	***
SGD	ADAMW	0.454009	0.411299	SGD	0.005071	**
SGD	NOVOGRAD	0.454009	0.492262	NOVOGRAD	0.000213	***
ADAGRAD	ADAMW	0.365256	0.411299	ADAMW	0.003599	**
ADAGRAD	NOVOGRAD	0.365256	0.492262	NOVOGRAD	4.01170e-08	***
ADAMW	NOVOGRAD	0.411299	0.492262	NOVOGRAD	0.000330	***

D.10 Experiment 5: Reinforcement Learning

Reinforcement Learning (RL) has become increasingly vital across various domains in recent years. RL enables agents to learn optimal behavior across a variety of environments through trial and error. This results in agents being more adaptable than traditional methods in complex environments. It plays a role in autonomous systems, sequential decision-making without supervision, and LLMS through Reinforcement Learning from Human Feedback (RLHF). Due to these growing uses, this study explores the performance of our optimizer in a three-dimensional space with stable consistent points, and randomly generated points. The purpose of this experiment is to study the optimizer's influence on reward-based actions and its adaptability. This experiment involved simulating a boxed environment, providing the agent with the observation of its current location and goal, and simulating 1000 episodes with 500 steps each. Each agent was trained 5 individual times, given 5 runs per training regimen, resulting in the final results being the averaged prediction of each agent across 25 runs, with the success rate being the rate the agent correctly reached the goal. The following results were observed.

Table 22: RL Experiment Training Parameters

Parameter	**Value**
Learning Rate	0.0005
Gamma (Discount Factor)	0.99
Episodes	1000
Log Interval	250 episodes
Initial Exploration	0.2
Final Exploration	0.01
Exploration Decay	0.998
Gradient Clip	0.5
Distance Threshold	0.05
Random Goal	True
Max Steps per Episode	500
Start Position	[0.0, 0.0, 0.0]
Goal Position	[1, 1, 1]

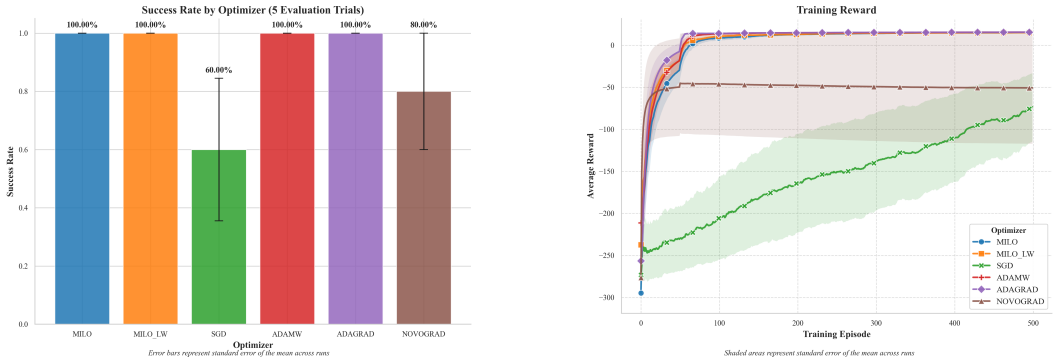
Table 23: Policy Network Architecture

Component	**Description**
Input Embedding	A linear layer mapping the input (observation) to 256 dimensions, followed by LayerNorm and ReLU activation.
Residual Blocks	4 residual blocks; each block consists of: • Linear ($256 \rightarrow 128$) • LayerNorm + ReLU • Dropout (0.1) • Linear ($128 \rightarrow 256$) • LayerNorm • Skip connection
Output Layers	A sequential bottleneck: Linear ($256 \rightarrow 128$) + LayerNorm + ReLU, Linear ($128 \rightarrow 64$) + LayerNorm + ReLU, and Linear ($64 \rightarrow$ output dimension), with final outputs scaled via `tanh` (multiplied by 0.1).

D.10.1 Set Result:

(a) 3D Agent Tractories (b) 2D Agent Tractories

Figure 12: 3D and 2D visuals of set RL agent trajectories.

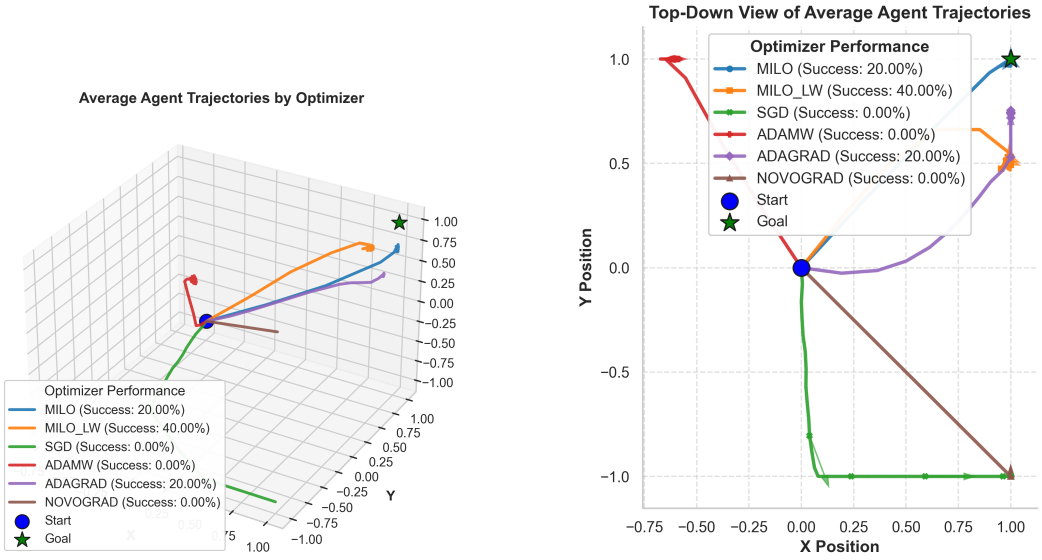


(a) RL Set Success Rates (b) Reward Comparision

Figure 13: Comparison of set success rates by optimizer and average reward by episode.

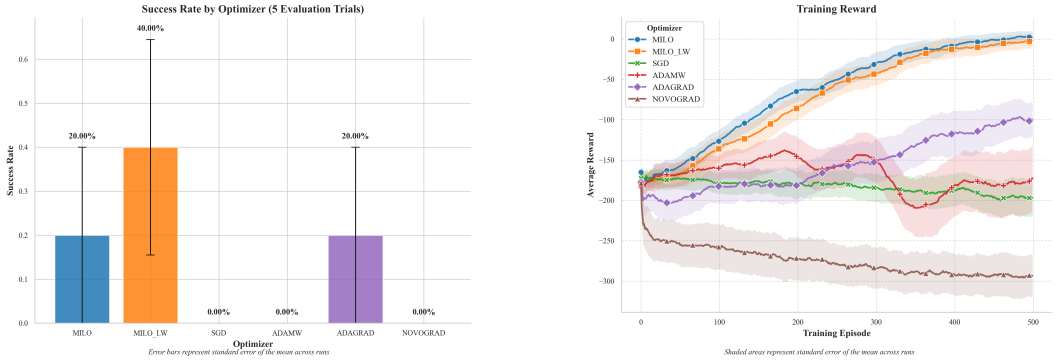
As shown in Figure 12 and Figure 13, each of the optimizers has a high success rate of reaching the set goal with no random initialization or generation of the goal. The optimizers with the highest performance proved to be AdamW, AdaGrad, Milo, and Milo_{LW}, each achieving a 100% accuracy. Overall, the set experiment demonstrates that each of the models can effectively learn the patterns generated in this 3D space.

D.10.2 Randomized Result:



(a) 3D Agent Tractories (b) 2D Agent Tractories

Figure 14: 3D and 2D visuals of set RL agent trajectories.



(a) RL Randomized Success Rates (b) Reward Comparison

Figure 15: Comparison of randomized success rates by optimizer and average reward by episode.

The results of the randomly generated start and goal differ drastically from the set, with each optimizer performing worse than before. However, each optimizer retains the relative performance shown previously regarding their performance against one another. As shown in Figure 14a, Figure 14b, and Figure 15a, Milo_{LW} has the highest accuracy, with a final success rate of 40%, being a large improvement over the success rates of the other optimizers. This performance is also shown in Figure 15b, with Milo_{LW} demonstrating a higher reward than the other optimizers.

D.11 Experiment 6: Large Language Model

LLMs have experienced significant growth in popularity in recent years, accompanied by numerous innovations in model architecture and optimization. This study explores the performance of our optimizer in a pretraining setting, using a small-scale Transformer model trained from scratch on a processed subset of The Pile dataset [Gao et al., 2020]. The model weights are randomly initialized at the start of training, and no pre-trained parameters are used at any stage. This setup allows us to evaluate optimizer behavior in a controlled language modeling task, where both model capacity and training steps are deliberately constrained to enable rapid comparison across runs.

To support this investigation, we adapt the open-source training framework of FareedKhan-dev (2025), which implements a simplified version of the Transformer architecture introduced by Vaswani et al. (2017). This architecture includes core components such as multi-head self-attention, layer normalization, and position-wise feed-forward networks, implemented natively in PyTorch. We extend the original implementation to support multiple optimizers and perform controlled training and evaluation across several runs. Visual results can be seen in Figure 4.

Table 24: Training Parameters

Parameter	Value	Description
VOCAB_SIZE	50304	Number of unique tokens in the vocabulary.
CONTEXT_LENGTH	64	Maximum sequence length for the model.
N_EMBED	64	Dimension of the embedding space.
N_HEAD	4	Number of attention heads in each transformer block.
N_BLOCKS	1	Number of transformer blocks in the model.
T_BATCH_SIZE	32	Number of samples per training batch.
T_CONTEXT_LENGTH	16	Context length for training batches.
T_TRAIN_STEPS	3000	Total number of training steps.
T_EVAL_STEPS	500	Frequency (in steps) for evaluation.
T_EVAL_ITERS	125	Number of iterations for evaluation.
T_LR_DECAY_STEP	1500	Step at which the learning rate decays.
T_LR	0.05	Initial learning rate for training.
T_LR_DECAYED	0.005	Learning rate after decay.
DEVICE	cuda	Device used for training.

Table 25: Model Architecture Layers

Layer	Description
Token Embedding	Converts input token indices into embeddings of dimension N_EMBED.
Positional Embedding	Adds positional information to the token embeddings for a maximum sequence length of CONTEXT_LENGTH.
Transformer Block	Comprises a multi-head self-attention mechanism (with N_HEAD heads) and a feed-forward network, incorporating residual connections and layer normalization.
Final Linear Projection	Projects the transformer output to logits over the vocabulary (transforming from N_EMBED to VOCAB_SIZE).

Table 26: Transformer Experiment Configuration

Component	Details
Model	Transformer
Vocabulary Size	50304
Embedding Dimension	64
Attention Heads	4
Transformer Blocks	1
Training Batch Size	32
Training Context Length	16
Total Training Steps	3000
Evaluation Frequency	Every 500 steps (125 iterations)
Learning Rate	Initial: 0.05, decays to 0.005 at step 1500

The results in this experiment, as shown in Figure 4a and Figure 4b, demonstrate that Milo and Milo$_{\text{LW}}$ demonstrate comparable performance to some of the industry standard used for training LLMs such as Adam and AdamW, with Milo$_{\text{LW}}$ and Milo achieving the lowest recorded loss and fastest reduction of loss as well. Additionally, since this was a more complex task, the wall time of each optimizer was logged as well, with Milo being comparable to Adam and AdamW, whereas Milo$_{\text{LW}}$ and NovoGrad took a longer training time.

D.12 Experiment 7: Ablation Study

To assess the impact of Milo's design choices, we conduct a comprehensive ablation study across two model architectures (MLP, ResNet-18) and two datasets (MNIST, CIFAR-10). Our goals are to isolate the effects of key hyperparameters—group size, learning rate, clipping norm, and scale-aware blending—and evaluate their impact on training stability, convergence speed, and sensitivity to hyperparameter selection.

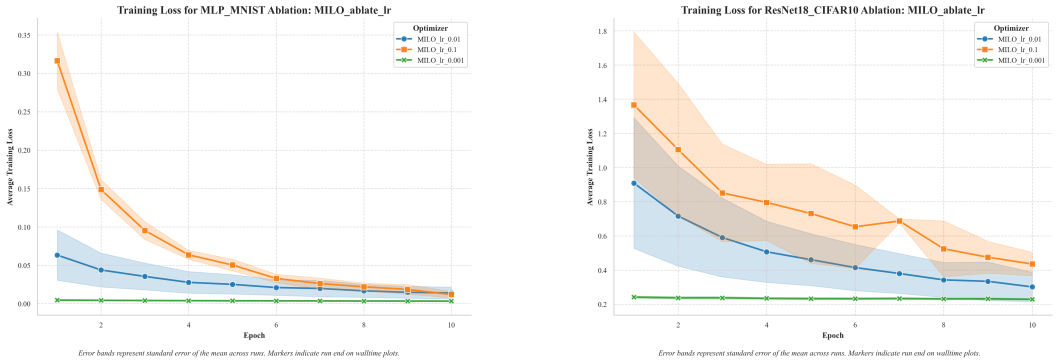
Experimental Setup. We evaluate two optimizer variants: standard Milo and its layer-wise variant Milo_{LW}. For each model–dataset pair, we establish a baseline configuration and systematically vary one hyperparameter at a time. The key parameters we will discuss include:

- **Learning Rate** (`lr`)**:** {0.0001, 0.001, 0.005, 0.01}
- **Momentum** (`momentum`)**:** {0.0, 0.5, 0.9}
- **Weight Decay** (`weight_decay`)**:** {0, 1e-4, 1e-3}
- **Epsilon** (`eps`)**:** {1e-8, 1e-5, 1e-2}
- **Scale-Aware Blending** (`scale_aware`, `scale_factor`)**:** {True, False}, {0.1,0.2,0.5} (Milo_{LW} only)

We also explore the other parameters used by the model in our code, but only discuss these here as they had the most notable results.

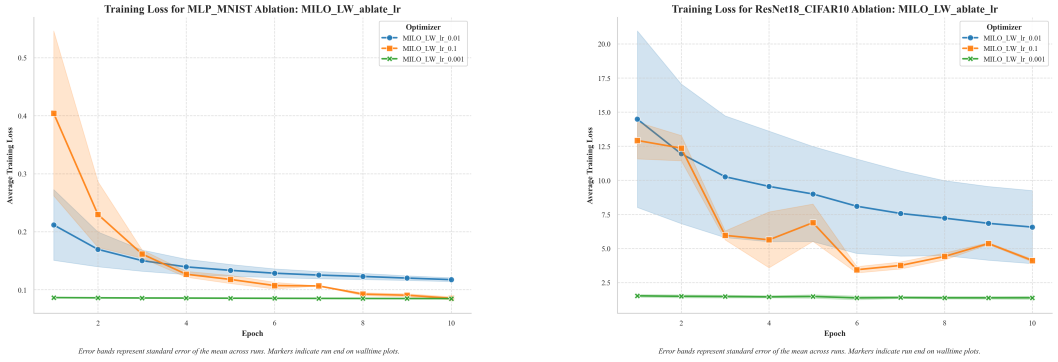
We measure:

- *Convergence Speed*: epochs to reach fixed loss threshold.
- *Stability*: variance in final loss and accuracy across seeds.
- *Hyperparameter Sensitivity*: performance variation relative to baseline.



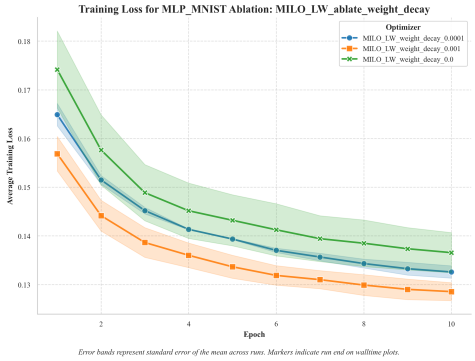
(a) Milo MLP Train Loss (b) Milo ResNet-18 Train Loss

Figure 16: Comparison of Milo LR Ablation in MLP and ResNet-18.

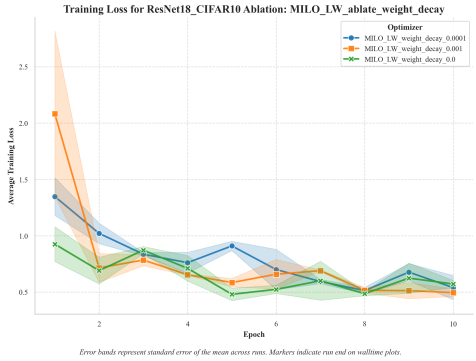


(a) Milo MLP Train Loss (b) Milo ResNet-18 Train Loss

Figure 17: Comparison of Milo_LW LR Ablation in MLP and ResNet-18.



(a) Milo MLP Train Loss



(b) Milo ResNet-18 Train Loss

Figure 18: Comparison of Milo_LW Weight Decay Ablation in MLP and ResNet-18.

Key Findings. We found that **learning rate** remains the most critical factor: optimal values vary by task and model, with Milo and Milo_{LW} exhibiting similar sensitivity profiles. **Momentum** and **scale-aware blending** have a moderate impact, influencing the convergence rate and final plateauing of the model. However, there is not as large of a deviation in accuracy as changes in the learning rate. Lastly, **weight decay**, **epsilon**, and a majority of the remaining hyperparameters were found to have minimal affect on the training. These results suggest that Milo's group normalization and adaptive components are robust across a wide range of hyperparameters, with the LR being the primary parameter needing to be tuned. This reduces the tuning burden relative to conventional optimizers, as discussed earlier in this paper.