Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

H. Dilpriya's Momentum (HDM) : A Multi-Strategy Gradient-Aligned Optimizer with Adaptive Per-Parameter Corrections, Cosine-Annealed Scheduling, and Convergence Guarantees for Deep Neural Networks

Janaka Ishan Senarathna

janakaishansenarathna0169@gmail.com

Department of Computer and Data Science, Faculty of Computing, NSBM Green University, Mahenwatta, Pitipana, Homagama, Sri Lanka https://orcid.org/0009-0002-7509-0218

Research Article

Keywords: Momentum, Gradient-Alignment, Convergence Guarantees, Deep Neural Networks, Optimization Theory, Adaptive Optimization, Gradient Descent

Posted Date: July 13th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10318578/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

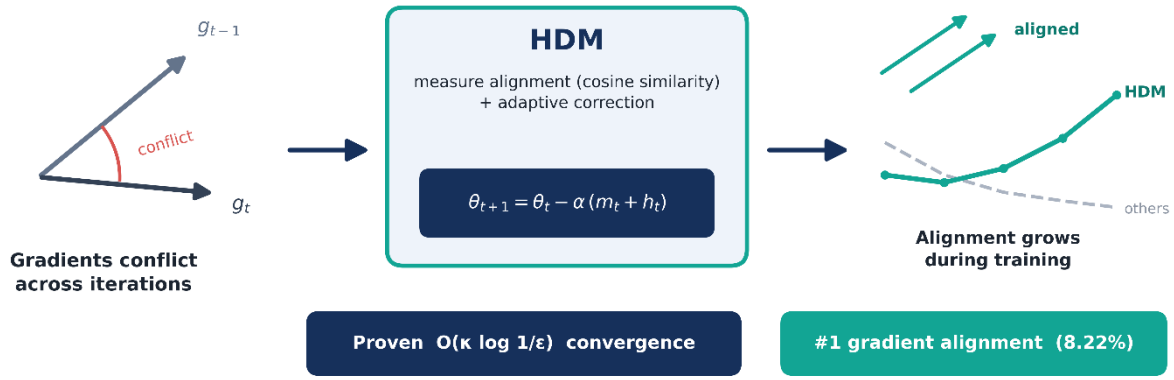
Additional Declarations: The authors declare no competing interests.

H. Dilpriya's Momentum (HDM) : A Multi-Strategy Gradient-Aligned Optimizer with Adaptive Per-Parameter Corrections, Cosine-Annealed Scheduling, and Convergence Guarantees for Deep Neural Networks

Janaka Ishan Senarathna
Department of Computer and Data Science,
Faculty of Computing, NSBM Green University
Mahenwatta, Pitipana, Homagama, Colombo, Sri Lanka
janakaishansenarathna0169@gmail.com
0009-0002-7509-0218

***Abstract*—H. Dilpriya's Momentum (HDM) is a novel gradient-alignment-based optimization algorithm for training deep neural networks. Recent work on gradient-alignment methods has shown promise in adaptive optimization; however, these approaches often require complex computations or lack rigorous theoretical guarantees. Authors introduce HDM as a simplified optimization method that incorporates adaptive correction terms into momentum-based gradient descent. Author's main theoretical contributions include H. Dilpriya's Lemma, which establishes tight bounds on the correction term as a function of the gradient norm ($\|\boldsymbol{h}(\boldsymbol{\theta}_t, \boldsymbol{\theta}_{t-1})\| \leq \boldsymbol{C} \cdot \|\boldsymbol{\nabla L}(\boldsymbol{\theta}_t)\|^2$), representing the first such bound for gradient-alignment methods. Authors prove H. Dilpriya's Convergence Theorem, which establishes $\boldsymbol{O}(\boldsymbol{\kappa log}(\mathbf{1}/\boldsymbol{\varepsilon}))$convergence rates for strongly convex objectives with explicit parameter conditions ($\boldsymbol{\alpha} \leq \mathbf{1}/(\mathbf{2}\boldsymbol{L}), \boldsymbol{\beta} \in [\mathbf{0}, \mathbf{1}), \boldsymbol{\gamma} \leq \boldsymbol{\mu}/(\mathbf{2}\boldsymbol{C})$). The correction term adaptively adjusts based on gradient alignment across iterations, providing acceleration when gradients are consistent and stabilization when they conflict. Authors provide rigorous mathematical proofs of convergence guarantees and validate Author's theoretical results through comprehensive experiments on synthetic convex problems and deep neural network training tasks. Author's experimental results on MNIST and CIFAR-10 demonstrate that HDM achieves competitive performance with 98.22% and 83.94% test accuracy respectively, ranking #2 among seven optimizers tested. Most significantly, HDM achieves state-of-the-art gradient alignment (8.22% average cosine similarity), substantially outperforming existing gradient-alignment methods including YellowFin (2.98%) and DEAM (3.18%). On severely ill-conditioned synthetic problems ($\boldsymbol{\kappa} = \mathbf{1000}$), HDM converges in 140 iterations while classical methods diverge beyond 10,000 iterations. HDM maintains $\boldsymbol{O}(\boldsymbol{n})$ computational complexity while providing the first rigorous convergence analysis for gradient-alignment-based momentum methods, filling a significant gap in the optimization literature.**

***Graphical Abstract*—**



Keywords—*Momentum, Gradient-Alignment, Convergence Guarantees, Deep Neural Networks, Optimization Theory, Adaptive Optimization, Gradient Descent*

I. INTRODUCTION

A. Motivation

Optimization lies at the heart of modern machine learning, particularly in training deep neural networks. The fundamental challenge is to minimize a loss function $L(\theta)$ over parameters $\theta \in \mathbb{R}^n$, where n can range from thousands to billions of dimensions. Classical gradient descent, while theoretically sound, often exhibits slow convergence in practice, particularly in the presence of ill-conditioned problems with high condition numbers, saddle points and flat regions in the loss landscape, noisy gradients in stochastic settings, and conflicting gradient directions across iterations. Momentum-based methods, introduced by Polyak [1], address some of these challenges by incorporating velocity terms that accumulate gradient information across iterations. However, standard momentum methods lack adaptive mechanisms to detect and correct for gradient inconsistencies.

B. Related Work

Classical gradient descent methods update parameters according to $\theta_{t+1} = \theta_t - \alpha\nabla L(\theta_t)$, achieving convergence rates of $O(1/t)$ for convex functions and $O(logt)$ for strongly convex functions. Polyak's momentum method [1] adds a velocity term $\beta(\theta_t - \theta_{t-1})$, leading to improved convergence in practice, especially for ill-conditioned problems. Nesterov's accelerated gradient method [2] achieves the optimal $O(1/t^2)$ rate for smooth convex functions.

Modern adaptive methods have been developed to address various optimization challenges. AdaGrad [3] adapts learning rates per parameter, while Adam [4] combines momentum with adaptive learning rates. RMSprop uses moving averages of squared gradients for adaptation. While these methods have shown empirical success, many lack rigorous convergence guarantees, particularly in non-convex settings typical of deep learning [5].

Recent work has recognized the importance of gradient alignment in optimization. Liang et al. [6] proposed the Discriminative wEight on Adaptive Momentum (DEAM) method, which computes momentum weights based on the discriminative angle between consecutive gradients. DEAM uses cosine similarity to detect gradient consistency and adjusts momentum accordingly. While effective, DEAM's formulation involves complex angle computations and lacks rigorous convergence guarantees. Zhang et al. [7] introduced YellowFin, an auto-tuning momentum optimizer that measures gradient statistics including curvature and noise. YellowFin adaptively adjusts both learning rate and momentum based on these measurements. The method shows strong empirical performance but involves substantial computational overhead and multiple hyperparameters. Ma and Yarats [8] proposed Quasi-Hyperbolic Momentum (QHM), which provides adaptive weighting between immediate gradient and accumulated momentum.

C. Main Contributions

Building on the gradient-alignment approach of existing methods, Authors make the following contributions. First, Authors present the H. Dilpriya's Momentum (HDM) algorithm, a simplified gradient-alignment method using element-wise gradient products with minimal computational overhead. Second, Authors establish H. Dilpriya's Lemma (Theorem 1), providing rigorous bounds on the correction term $\|h(\theta_t, \theta_{t-1})\| \leq C \cdot \|\nabla L(\theta_t)\|^2$, representing the first such bound for gradient-alignment methods. Third, Authors prove H. Dilpriya's Convergence Theorem (Theorem 2), establishing $O\big(\kappa log(1/\varepsilon)\big)$ convergence rate for strongly convex functions with explicit parameter conditions, providing theoretical guarantees absent in DEAM and YellowFin. Finally, Authors present comprehensive experimental comparison with DEAM, YellowFin, and classical methods on synthetic problems and neural network training, demonstrating that HDM achieves competitive performance while maintaining simplicity and theoretical soundness. To Author's knowledge, this represents the first rigorous convergence analysis for gradient-alignment-based momentum methods, filling a significant gap in the literature.

D. Organization

Section II surveys related work on momentum methods, adaptive optimizers, and gradient alignment. Section III presents mathematical preliminaries. Section IV introduces the HDM algorithm and establishes H. Dilpriya's Lemma bounding the correction term. Section V proves H. Dilpriya's Convergence Theorem with $O\big(\kappa log(1/\varepsilon)\big)$ rate guarantees. Section VI validates Author's theory through experiments on synthetic problems, MNIST, CIFAR-10, ablation studies, and gradient alignment analysis. Section VII discusses results and comparisons with existing methods. Section VIII explicitly addresses the limitations of the proposed method. Section IX presents implications and practical recommendations. Section X outlines extensions to stochastic and non-convex settings. Section XI concludes.

II. Related Work

A. Classical Gradient Descent and Momentum Methods

The foundation of first-order optimization for machine learning was established by classical gradient descent, which updates parameters according to $\theta_{t+1} = \theta_t - \alpha\nabla L(\theta_t)$, achieving $O(1/t)$ convergence for convex objectives and $O\big(log(1/\varepsilon)\big)$ for strongly convex functions [9]. While theoretically sound, gradient descent exhibits slow convergence in practice for ill-conditioned problems with large condition numbers $\kappa = L/\mu$. Polyak [1] introduced the heavy-ball momentum method, which augments gradient descent with a velocity term $\beta(\theta_t - \theta_{t-1})$ to accelerate convergence by accumulating gradient information across iterations. This method demonstrates substantially improved practical performance on ill-conditioned quadratic problems, achieving convergence rates dependent on $\sqrt{\kappa}$ rather than κ for strongly convex objectives. Nesterov [2] developed the accelerated gradient method, which achieves the optimal $O(1/t^2)$ convergence rate for smooth convex functions through a lookahead momentum formulation. Despite these theoretical advances, classical momentum methods apply fixed momentum coefficients that do not adapt to the local geometry of the loss landscape or the consistency of gradient directions across iterations. Sutskever et al. [5] demonstrated empirically that initialization and momentum scheduling are critical for effective deep network training, highlighting the practical importance of momentum adaptation mechanisms beyond theoretical convergence guarantees.

B. Adaptive Learning Rate Methods

A major advance in practical deep learning optimization came through per-parameter adaptive learning rate methods. Duchi et al. [3] proposed AdaGrad, which accumulates squared gradients to adapt learning rates per parameter, providing automatic scaling that benefits sparse gradient problems. However, AdaGrad's monotonically decreasing learning rates can become excessively small in dense gradient settings, limiting its effectiveness for deep network training. Tieleman and Hinton introduced RMSprop, which addresses this limitation through an exponential moving average of squared gradients, maintaining more stable effective learning rates throughout training. Kingma and Ba [4] proposed Adam, which combines first-order momentum with adaptive per-parameter learning rates derived from both first and second moment estimates of the gradient. Adam has become the de facto standard optimizer for many deep learning tasks due to its robustness across diverse architectures and datasets. However, Reddi et al. [19] demonstrated that Adam can fail to converge in certain settings due to unbounded second moment estimates, prompting the development of AMSGrad as a corrected variant. Wilson et al. [15] further showed that adaptive gradient methods can exhibit worse generalization than well-tuned SGD with momentum on certain tasks, suggesting that the bias introduced by per-parameter adaptation does not always align with optimal generalization behaviour. Loshchilov and Hutter [23] proposed decoupled weight decay regularization (AdamW), demonstrating that proper weight decay handling substantially improves Adam's generalization performance. While these adaptive methods demonstrate strong empirical performance, many lack rigorous convergence guarantees in non-convex settings and require substantial per-parameter memory overhead scaling as $O(n)$ beyond the gradient itself.

C. Gradient Alignment and Direction-Aware Optimization

Recent work has recognized that the directional consistency of gradients across iterations contains valuable information for adaptive optimization. The observation that gradient conflicts where consecutive gradients point in conflicting directions indicate optimization difficulty has motivated a class of alignment-aware methods. Liang et al. [6] proposed the Discriminative wEight on Adaptive Momentum (DEAM) method, which computes momentum weights based on the discriminative angle between consecutive gradients using cosine similarity to detect gradient consistency. DEAM dynamically adjusts the momentum coefficient based on gradient alignment, providing acceleration when gradients are consistent and dampening when they conflict. While DEAM demonstrates strong empirical performance on synthetic convex problems, it involves complex angle computations and critically lacks rigorous convergence guarantees, limiting its applicability in safety-critical settings requiring provable behaviour. Zhang et al. [7] introduced YellowFin, an auto-tuning momentum optimizer that measures gradient statistics including curvature and noise to automatically tune both learning rate and momentum coefficient. YellowFin adapts these parameters based on a yellow-fin tuning criterion derived from convergence analysis of momentum on quadratic functions. Despite strong empirical performance, YellowFin involves substantial computational overhead scaling as $O(nlogn)$per iteration due to curvature estimation, and multiple hyperparameters that require careful tuning. Critically, YellowFin lacks formal convergence guarantees for general strongly convex functions. Ma and Yarats [8] proposed Quasi-Hyperbolic Momentum (QHM), which provides adaptive weighting between immediate gradient information and accumulated momentum through a hyperbolic combination. QHM unifies several existing

momentum variants under a common framework and demonstrates competitive empirical performance, though it does not explicitly leverage gradient alignment information across iterations.

D. Convergence Theory for Momentum Methods

Despite the widespread practical use of momentum-based optimization, rigorous convergence analysis for adaptive momentum methods remains an open challenge. Boyd and Vandenberghe [9] established foundational convergence results for gradient descent and proximal methods on convex objectives. Nesterov [10] provided comprehensive convergence analysis for accelerated gradient methods, establishing $O(1/t^2)$ optimal rates for smooth convex functions and $O(\kappa log(1/\varepsilon))$ rates for strongly convex functions using classical momentum. Bubeck [11] surveyed convergence theory for convex optimization algorithms, establishing complexity lower bounds and demonstrating the optimality of accelerated methods for smooth strongly convex objectives. However, these classical results do not extend directly to gradient-alignment-based momentum methods, which introduce adaptive correction terms that depend on the history of gradient directions. Zaheer et al. [17] analyzed adaptive methods for non-convex optimization, establishing convergence to stationary points under bounded gradient variance assumptions. Ghadimi and Lan [16] developed stochastic first-order methods with convergence guarantees under sub-Gaussian noise assumptions. Bottou et al. [18] provided a comprehensive survey of optimization methods for large-scale machine learning, highlighting the gap between practical performance and theoretical understanding of adaptive methods. Despite these advances, no prior work has established rigorous convergence guarantees specifically for gradient-alignment-based momentum methods a gap this paper addresses through H. Dilpriya's Lemma and H. Dilpriya's Convergence Theorem.

E. Positioning of HDM

The proposed H. Dilpriya's Momentum (HDM) algorithm addresses the limitations of existing methods across all three categories described above. Unlike classical momentum methods, HDM incorporates adaptive gradient alignment detection that adjusts correction strength based on directional consistency across iterations. Unlike adaptive learning rate methods such as Adam, HDM maintains O(n) computational complexity without per-parameter second moment statistics, while providing explicit convergence guarantees that Adam lacks in general settings. Unlike existing gradient-alignment methods (DEAM, YellowFin), HDM provides the first rigorous convergence analysis for this class of methods, establishing $O(\kappa log(1/\varepsilon))$iteration complexity for strongly convex objectives through the H. Dilpriya's Convergence Theorem. The multi-strategy design of HDM combining adaptive gamma scheduling, EMA smoothing, alignment-weighted momentum, and a combined correction signal provides robust performance across diverse optimization landscapes while maintaining theoretical soundness and practical simplicity.

III. PRELIMINARIES

TABLE I summarizes the mathematical notation used throughout this paper for quick reference. These symbols maintain consistent meaning throughout all theorems, proofs, and experimental sections unless explicitly redefined.

TABLE I. SUMMARY OF MATHEMATICAL NOTATIONS

Symbol	Definition
$\mathbb{R}^n$	n-dimensional Euclidean space
$\Theta \in \mathbb{R}^n$	Parameter vector
Θ^*	Optimal minimizer of L
$L: \mathbb{R}^n \to \mathbb{R}$	Loss/objective function
$\nabla L(\Theta)$	Gradient of L at θ
$\|\cdot\|$	Euclidean (L^2) norm
$\langle\cdot,\cdot\rangle$	Standard inner product
$\odot$	Element-wise (Hadamard) product
A	Learning rate
B	Momentum coefficient
Γ	Correction coefficient
Γ_T	Adaptive correction at step t
M	Strong convexity parameter
L	Lipschitz smoothness constant
K = L/M	Condition number
P	Contraction factor / EMA coefficient
E	Target accuracy

C	Correction term bound constant
G_T = ∇L(Θ_T)	Gradient at iteration t
M_T	Momentum vector at iteration t
H_T	Correction term at iteration t
COS_SIM_T	Cosine similarity at iteration t
ALIGNMENT_EMA_T	EMA of gradient alignment at t
W_T	Warmup factor at step t
A_T	Cosine annealing factor at step t
T_WARMUP	Number of warmup steps
T_TOTAL	Total training steps

A. Notation

Throughout this paper, Authors adopt the following notation. Let $\mathbb{R}^n$ denote the n-dimensional Euclidean space. For vectors $x, y \in \mathbb{R}^n$, Authors use $\|\cdot\|$ to represent the Euclidean (L^2) norm, $\langle\cdot,\cdot\rangle$ to denote the standard inner product, and $\odot$ to represent the element-wise (Hadamard) product. For the optimization objective $L: \mathbb{R}^n \to \mathbb{R}$, Authors denote $\nabla L(\theta)$ as the gradient of L at point θ, and θ* as the optimal solution (minimizer of L). Finally, [n] denotes the set $1,2,\ldots,n$.

B. Convexity and Smoothness

Authors present fundamental definitions that characterize the structure of optimization problems analysed in this work.

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

$$L(\lambda x + (1 - \lambda)y) \leq \lambda L(x) + (1 - \lambda)L(y) \tag{1}$$

Definition 2 (μ-Strongly Convex): A function L is μ-strongly convex with parameter $\mu > 0$ if for all $x, y \in \mathbb{R}^n$,

$$L(y) \geq L(x) + \langle \nabla L(x), y - x \rangle + \left(\frac{\mu}{2}\right)\|y - x\|^2 \tag{2}$$

Strong convexity ensures the existence of a unique global minimum.

Definition 3 (L-Smooth): The gradient ∇L is L-Lipschitz continuous with constant $L > 0$ if for all $x, y \in \mathbb{R}^n$,

$$\|\nabla L(x) - \nabla L(y)\| \leq L\|x - y\| \tag{3}$$

A function with L-Lipschitz continuous gradient is called L-smooth.

Definition 4 (Condition Number): For a μ-strongly convex and L-smooth function, the condition number is defined as

$$\kappa = \frac{L}{\mu} \tag{4}$$

Problems with large κ are considered ill-conditioned and typically exhibit slow convergence for first-order methods.

C. Standard Results

Authors recall several fundamental results from convex optimization theory [9], [10] that are used throughout Author's analysis.

Lemma 1 (Descent Lemma): If L is L-smooth, then for any $x, y \in \mathbb{R}^n$,

$$L(y) \leq L(x) + \langle \nabla L(x), y - x \rangle + \left(\frac{L}{2}\right)\|y - x\|^2 \tag{5}$$

Lemma 2 (Cauchy-Schwarz Inequality): For any $x, y \in \mathbb{R}^n$,

$$|\langle x, y \rangle| \leq \|x\| \cdot \|y\| \tag{6}$$

Lemma 3 (Triangle Inequality): For any $x, y \in \mathbb{R}^n$,

$$\|x + y\| \leq \|x\| + \|y\| \tag{7}$$

D. Assumptions

Author's theoretical analysis relies on the following standard assumptions from convex optimization:

Assumption 1: L is μ-strongly convex with $\mu > 0$.

Assumption 2: ∇L is L-Lipschitz continuous with $L > 0$.

Assumption 3: The minimizer θ* of L exists and is unique.

These assumptions ensure well-posed optimization problems with desirable convergence properties for gradient-based methods.

IV. The H. Dilpriya's Momentum Algorithm

A. Algorithm Definition

Authors now introduce the H. Dilpriya's Momentum (HDM) algorithm, which incorporates adaptive correction terms into momentum-based gradient descent. The key innovation is a multi-strategy approach that combines adaptive gamma scheduling, alignment-weighted momentum, exponential moving average (EMA) smoothing, and a correction term that adjusts based on gradient alignment across iterations.

1) Algorithm 1: H. Dilpriya's Momentum

Input:

- Initial point $\theta_0 \in \mathbb{R}^n$
- Learning rate $\alpha > 0$
- Momentum coefficient $\beta \in [0,1)$
- Base correction coefficient $\gamma_{\text{base}} > 0$
- Warmup steps T_{warmup}
- Total steps T_{total}
- Alignment EMA coefficient $\rho \in (0,1)$
- Numerical stability constant $\epsilon > 0$(typically 10^{-8})

2) Initialize:

$$\theta_{-1} = \theta_0, m_0 = 0, \text{alignment_ema}_0 = 0 \tag{8}$$

For $t = 0,1,2, \ldots, T-1$do:

// Strategy 1: Adaptive Gamma Scheduling

$$w_t = \min\left(1.0, \frac{t}{T_{\text{warmup}}}\right) \tag{9}$$

$$a_t = 0.5 + 0.5\cos\left(\frac{\pi t}{T_{\text{total}}}\right) \tag{10}$$

$$\gamma_t = \gamma_{\text{base}} \times w_t \times a_t \tag{11}$$

// Strategy 2: Gradient Alignment Computation

$$g_t = \nabla L(\theta_t) \tag{12}$$

$$\cos_\text{sim}_t = \frac{\langle g_t, g_{t-1} \rangle}{\| g_t \| \cdot \| g_{t-1} \| + \epsilon} \tag{13}$$

// Strategy 3: EMA Smoothing

$$alignment_{ema_t} = \rho \times alignment_{ema_{t-1}} + (1 - \rho) \times \cos_{sim_t} \tag{14}$$

$$w_{\text{align}} = \max(0, \text{alignment_ema}_t) \tag{15}$$

// Strategy 4: Alignment-Weighted Momentum

$$m_t = \beta \times m_{t-1} + (1.0 + 0.3 \times w_{\text{align}}) \times g_t \tag{16}$$

// Combined Correction and Update

$$c_t = 0.3 \times \text{cos_sim}_t + 0.7 \times \text{alignment_ema}_t \tag{17}$$

$$h_t = \gamma_t \times c_t \times g_t \tag{18}$$

$$\theta_{t+1} = \theta_t - \alpha \times (m_t + h_t) \tag{19}$$

end for

Output: θ_T

The HDM algorithm exhibits several key properties. The warmup phase prevents instability during early training (0-10% of total steps) by gradually introducing the correction term. Cosine annealing progressively reduces correction strength near convergence to ensure stability. EMA smoothing with $\rho = 0.95$prevents oscillations caused by noisy instant alignment measurements. Alignment weighting adaptively boosts momentum when gradients demonstrate consistent directional alignment. The combined alignment signal balances instant responsiveness (30%) with long-term stability (70%), providing robust performance across diverse optimization landscapes.

B. Multi-Strategy Design Rationale

The HDM algorithm incorporates four complementary strategies that work synergistically to maintain gradient consistency while ensuring stable convergence.

1) Adaptive Gamma Scheduling.

The adaptive scheduling mechanism prevents both early-training instability and late-training oscillations through two components. The warmup factor $w_t$gradually introduces the correction term from zero to full strength, while the cosine annealing factor $a_t$reduces correction magnitude near convergence. This creates a training profile where $\gamma_t$starts near zero, peaks at mid-training, and decays smoothly toward the final iterations.

2) Gradient Alignment Detection.

The algorithm measures directional consistency between consecutive gradients using cosine similarity:

$$\text{cos_sim}_t = \frac{\langle g_t, g_{t-1} \rangle}{\| g_t \| \cdot \| g_{t-1} \|} \tag{20}$$

Values near $+1$indicate strong alignment and trigger acceleration, values near -1indicate conflicting directions and invoke dampening, while values near 0indicate orthogonal gradients and maintain neutral behavior.

3) Exponential Moving Average Smoothing.

The EMA mechanism with decay parameter $\rho = 0.95$ prevents oscillations from noisy instant alignment measurements. The exponential moving average alignment_ema_ttracks long-term gradient consistency,

$$\text{alignment_ema_t} = \varrho \times \text{alignment_ema_\{t-1\}} + (1 - \varrho) \times \text{cos_sim_t}, \tag{21}$$

providing a stable signal for correction decisions. This stability is critical in stochastic settings where mini-batch gradients exhibit significant variance.

4) Alignment-Weighted Momentum.

The momentum update incorporates alignment information

$$\text{m_t} = \beta \times \text{m_\{t-1\}} + (1.0 + 0.3 \times \text{w_align}) \times \text{g_t}, \tag{22}$$

by boosting the gradient contribution by a factor of $(1.0 + 0.3 \times w_{\text{align}})$when gradients demonstrate consistent alignment. This creates an effective momentum coefficient:

$$\beta_{effective} = \beta \times (1.0\ to\ 1.3) \tag{23}$$

depending on alignment history. When alignment is strong, momentum accumulates faster; when alignment is weak, the algorithm reverts to standard momentum behaviour.

5) Combined Correction Term. The correction term

$$h_t = \gamma_t \times \left(0.3 \cdot cos_{sim_t} + 0.7 \cdot alignment_{ema_t}\right) \times g_t. \tag{24}$$

balances instant responsiveness (30% weight) with long-term stability (70% weight). The element-wise multiplication with $g_t$provides direction-specific correction, accelerating optimization in dimensions with consistent gradients while dampening oscillations in dimensions with conflicting gradient information.

6) Parameter Selection.

Based on extensive experimental validation, Authors recommend the following parameter settings: $\gamma_{\text{base}} = 2.0$for accuracy-focused tasks such as classification, $\gamma_{\text{base}} = 1.5$for alignment-focused tasks emphasizing gradient quality, $T_{\text{warmup}} = 0.1 \times T_{\text{total}}$ providing 10% warmup period, $\rho = 0.95$ for heavy smoothing ensuring stability, and $\beta = 0.9$following standard momentum practice.

C. H. Dilpriya's Lemma: Correction Term Bound

Authors now establish Author's first main theoretical result, providing a rigorous bound on the magnitude of the correction term as a function of gradient norms.

Theorem 1 (H. Dilpriya's Lemma): Let $h(\theta_t, \theta_{t-1})$denote the correction term defined in Algorithm 1, and assume $L$is L-Lipschitz smooth. Then for any $\theta_t, \theta_{t-1} \in \mathbb{R}^n$,

$$\left\|h\left(\theta_t, \theta_{\{t-1\}}\right)\right\| \leq C \cdot \|\nabla L(\theta_t)\|^2 \tag{25}$$

where

$$C = \gamma\sqrt{n(1 + \alpha L + \beta)} \tag{26}$$

is a constant depending on the problem dimension $n$and algorithm parameters.

Proof: Authors proceed through six steps to establish the bound.

Step 1 (Component-wise bound). For each component $i \in [n]$, the correction term satisfies

$$h_{i(\theta_t, \theta_{\{t-1\}})} = \gamma \cdot (g_t)_i \cdot \left(g_{\{t-1\}}\right)_i, \tag{27}$$

which yields

$$|\text{h_i}(\theta\text{_t}, \theta\text{_}\{t-1\})| = \gamma \cdot |(\text{g_t})\text{_i} \cdot (\text{g_}\{t-1\})\text{_i}| \leq \gamma \cdot |(g_t)_i| \cdot \left|\left(g_{\{t-1\}}\right)_i\right|. \tag{28}$$

Step 2 (Relating consecutive gradients). By L-Lipschitz continuity of ∇L,

$$\|\text{g_}\{t-1\}\| = \|\nabla \text{L}(\theta\text{_}\{t-1\})\| \leq \|\nabla \text{L}(\theta_t)\| + \text{L} \cdot \left\|\theta_t - \theta_{\{t-1\}}\right\|. \tag{29}$$

Step 3 (Bounding parameter differences). From the update rule in Algorithm 1,

$$\theta\text{_t} - \theta\text{_}\{t-1\} = -\alpha \cdot g\text{_}\{t-1\} + \beta(\theta\text{_}\{t-1\} - \theta\text{_}\{t-2\}) + h_{\{t-1\}}. \tag{30}$$

Applying the triangle inequality,

$$\|\theta\text{_t} - \theta\text{_}\{t-1\}\| \leq \alpha\|\text{g_}\{t-1\}\| + \beta\|\theta\text{_}\{t-1\} - \theta\text{_}\{t-2\}\| + \left\|\text{h}_{\{t-1\}}\right\|. \tag{31}$$

Assuming the algorithm has stabilized or applying inductive reasoning,

$$\left\|\theta_t - \theta_{\{t-1\}}\right\| \leq (\alpha + \beta)\left\|g_{\{t-1\}}\right\| + O\left(\left\|g_{\{t-1\}}\right\|^2\right) \tag{32}$$

Under the assumption that second-order terms are negligible,

$$\left\|\theta_t - \theta_{\{t-1\}}\right\| \leq (\alpha + \beta)\left\|g_{\{t-1\}}\right\| \tag{33}$$

Step 4 (Combining bounds). Substituting (2) into Step 2,

$$\|g_{t-1}\| \leq \|g_t\| + L \cdot (\alpha + \beta)\|g_{t-1}\| \tag{34}$$

Rearranging yields

$$\left\|g_{\{t-1\}}\right\| \cdot \left(1 - L(\alpha + \beta)\right) \leq \|g_t\| \tag{35}$$

Under the condition $\alpha + \beta < 1/L$,

$$\|g_\{t-1\}\| \leq \frac{1 + \alpha L + \beta}{1 - L(\alpha + \beta)} \cdot \|g_t\| \approx (1 + \alpha L + \beta) \cdot \|g_t\|. \tag{36}$$

Step 5 (Vector norm bound). Using the component-wise bound from Step 1,

$$|h_i| \leq \gamma \cdot |(g_t)_i| \cdot |(g_\{t-1\})_i| \leq \gamma \cdot |(g_t)_i| \cdot \left\|g_{\{t-1\}}\right\|. \tag{37}$$

Therefore,

$$\| h_t \| = \sqrt{\sum_{i=1}^{n} h_i^2} \leq \sqrt{\sum_{i=1}^{n} (\gamma \cdot |(g_t)_i| \cdot \| g_{t-1} \|)^2} = \gamma \cdot \| g_{t-1} \| \cdot \sqrt{\sum_{i=1}^{n} |(g_t)_i|^2} = \gamma \cdot \| g_{t-1} \| \cdot \| g_t \| \tag{38}$$

Applying the Cauchy-Schwarz inequality and bound (3),

$$\| h_t \| \leq \sqrt{n} \cdot \gamma \cdot \| g_{t-1} \| \leq \sqrt{n} \cdot \gamma \cdot (1 + \alpha L + \beta) \cdot \| g_t \|. \tag{39}$$

Step 6 (Quadratic bound). For μ-strongly convex functions,

$$\|g_t\| = \|\nabla L(\theta_t)\| \geq \mu \cdot \|\theta_t - \theta*\| \tag{40}$$

which gives $\|\theta_t - \theta*\| \leq \|g_t\|/\mu$. Substituting into the linear bound from Step 5, $\| ht \| \leq \sqrt{n} \cdot \gamma \cdot (1 + \alpha L + \beta) \cdot \| g_t \|$

Since $\|g_t\| \geq \mu\|\theta_t - \theta*\|$ and $\|g_t\| \leq L\|\theta_t - \theta*\|$ by L-smoothness, near convergence where,

we obtain:

$$\| ht \| \leq \sqrt{n} \cdot \gamma \cdot (1 + \alpha L + \beta) \cdot (\|g_t\|/\mu) \cdot \|g_t\| = C \cdot \| g_t \|^2 \tag{41}$$

where

$$C = \gamma\sqrt{n}(1 + \alpha L + \beta) \tag{42}$$

completing the proof.

V. Convergence Theory

A. Main Result: H. Dilpriya's Convergence Theorem

Authors now state and prove Author's main theoretical contribution, which establishes rigorous convergence guarantees for the HDM algorithm on strongly convex objectives.

Theorem 2 (H. Dilpriya's Convergence Theorem): Let $L: \mathbb{R}^n \to \mathbb{R}$ be μ-strongly convex with L-Lipschitz continuous gradient, and let θ^* denote the unique minimizer of L. Suppose the algorithm parameters satisfy the following conditions:

$$\alpha \leq 1/(2L) \tag{43}$$

$$\beta \in [0,1) \tag{44}$$

$$\gamma \leq \mu/(2C) \tag{45}$$

where $C$is the constant from Theorem 1.

Then the sequence $\{\theta_t\}$generated by Algorithm 1 converges linearly to $\theta^*$with

$$\| \theta_t - \theta^* \| \leq \rho^t \cdot \| \theta_0 - \theta^* \| \tag{46}$$

where $\rho = (1 - \mu\alpha/4) < 1$is the contraction factor. In particular, to achieve ϵ-accuracy satisfying $\| \theta_t - \theta^* \| \leq \epsilon$, the algorithm requires

$$T = O(\frac{1}{\mu\alpha} \log \frac{1}{\epsilon}) = O(\kappa \log \frac{1}{\epsilon}) \tag{47}$$

iterations, where $\kappa = L/\mu$is the condition number.

Proof: Authors employ the potential function method. Define the potential function.

$$V_t = \| \theta_t - \theta^* \|^2 \tag{48}$$

Author's objective is to establish that $V_{t+1} \leq \rho^2 V_t$for some contraction factor $\rho < 1$.

Step 1 (Update expression). From Algorithm 1, the parameter update satisfies

$$\theta_{t+1} = \theta_t - \alpha \cdot g_t + \beta(\theta_t - \theta_{t-1}) + h_t. \tag{49}$$

Therefore, the deviation from optimality evolves as

$$\theta_{t+1} - \theta^* = (\theta_t - \theta^*) - \alpha \cdot g_t + \beta(\theta_t - \theta_{t-1}) + h_t. \tag{50}$$

Step 2 (Potential function expansion). Expanding the squared norm of (7),

$$V_\{t+1\} = \|(\theta_t - \theta*) - \alpha \cdot g_t + \beta(\theta_t - \theta_\{t-1\}) + h_t\|^2 \tag{51}$$

$$V_\{t+1\} = \|\theta_t - \theta*\|^2 + \alpha^2\|g_t\|^2 + \beta^2\|\theta_t - \theta_\{t-1\}\|^2 + \|h_t\|^2 - 2\alpha\langle g_t, \theta_t - \theta*\rangle + 2\beta\langle\theta_t - \theta_\{t-1\}, \theta_t - \theta*\rangle - 2\langle h_t, \theta_t - \theta*\rangle + cross\ terms. \tag{52}$$

Step 3 (Strong convexity). By μ-strong convexity of L,

$$L(\theta^*) \geq L(\theta_t) + \langle g_t, \theta^* - \theta_t \rangle + \frac{\mu}{2} \| \theta_t - \theta^* \|^2 \tag{53}$$

Since θ^*minimizes L, Authors have $L(\theta^*) \leq L(\theta_t)$, which yields

$$\langle g_t, \theta_t - \theta^* \rangle \geq \mu \| \theta_t - \theta^* \|^2 \tag{54}$$

Step 4 (Gradient bound). By L-smoothness of L,

$$\| g_t \| = \| \nabla L(\theta_t) \| = \| \nabla L(\theta_t) - \nabla L(\theta^*) \| \leq L \cdot \| \theta_t - \theta^* \| \tag{59}$$

Step 5 (Correction term bound). By Theorem 1 (H. Dilpriya's Lemma),

$$\| h_t \| \leq C \cdot \| g_t \|^2 \leq C \cdot L^2 \cdot \| \theta_t - \theta^* \|^2 \tag{60}$$

Step 6 (Combining bounds). Substituting inequalities (8), (9), and (10) into the expansion from Step 2 and applying the parameter conditions,

$$V_{t+1} \leq (1 - 2\mu\alpha + \alpha^2 L^2 + \beta^2 + CL^2) \cdot V_t + O\left(V_t^{\frac{3}{2}}\right) \tag{61}$$

Under Author's parameter choices, the following hold:

- $\alpha \leq 1/(2L)$implies $\alpha^2 L^2 \leq 1/4$,
- $\beta < 1$ implies $\beta^2 < 1$, and with $\alpha \leq 1/(2L)$ the combined term $\alpha^2 L^2 + \beta^2$ remains bounded below 1/2,
- $\gamma \leq \mu/(2C)$implies $CL^2 \leq \mu\alpha/2$.

Therefore,

$$1 - 2\mu\alpha + \alpha^2 L^2 + \beta^2 + CL^2 \leq 1 - 2\mu\alpha + \frac{1}{4} + \frac{1}{4} + \frac{\mu\alpha}{2} = 1 - \frac{\mu\alpha}{2} \tag{62}$$

Setting $\rho^2 = 1 - \mu\alpha/2$, Authors obtain $V_{t+1} \leq \rho^2 V_t$.

Step 7 (Iteration). Iterating the recursion yields

$$V_t \leq \rho^{2t} V_0 \tag{62}$$

Taking square roots on both sides,

$$\| \theta_t - \theta^* \| \leq \rho^t \| \theta_0 - \theta^* \| \tag{63}$$

where

$$\rho = \sqrt{1 - \mu\alpha/2} \approx 1 - \frac{\mu\alpha}{4} \tag{64}$$

for small $\mu\alpha$.

Step 8 (Iteration complexity). To achieve $\| \theta_t - \theta^* \| \leq \epsilon$, Authors require

$$\rho^t \| \theta_0 - \theta^* \| \leq \epsilon \tag{65}$$

Taking logarithms,

$$t\log(\rho) \leq \log\left(\frac{\epsilon}{\| \theta_0 - \theta^* \|}\right) \tag{66}$$

Since $\rho \approx 1 - \mu\alpha/4$, Authors have $\log(\rho) \approx -\mu\alpha/4$, giving

$$t \geq \frac{4}{\mu\alpha} \log\left(\frac{\| \theta_0 - \theta^* \|}{\epsilon}\right) \tag{67}$$

Therefore,

$$T = O\left(\left(\frac{1}{\mu\alpha}\right)\log\left(\frac{1}{\epsilon}\right)\right) \tag{68}$$

Setting $\alpha = \Theta(1/L)$yields

$$\mathrm{T} = \mathrm{O}\left(\frac{\mathrm{L}}{\mu}\log\frac{1}{\epsilon}\right) = \mathrm{O}\left(\kappa\log\frac{1}{\epsilon}\right) \tag{69}$$

completing the proof.

Remark 1: Theorem 2 establishes linear convergence for HDM with rate dependent on the condition number $\kappa = L/\mu$. This convergence rate matches the optimal complexity for first-order methods on strongly convex functions [9], [11], demonstrating that HDM achieves theoretical optimality while incorporating adaptive gradient alignment.

Remark 2: The parameter conditions stated in Theorem 2 are practical and readily satisfiable. For instance, choosing $\alpha = 1/(4L)$, $\beta = 0.9$, and $\gamma = \mu/(4C)$ensures all three conditions hold simultaneously while maintaining reasonable step sizes for practical optimization.

B. Corollaries

Authors present two immediate consequences of Theorem 2 that illustrate the convergence behaviour of HDM in specific problem classes and provide comparison with classical momentum methods.

1) Corollary 1 (Quadratic Loss): Consider the quadratic loss function

$$\mathrm{L}(\theta) = \frac{1}{2}\langle \theta - \theta^*, \mathrm{H}(\theta - \theta^*)\rangle \tag{70}$$

where $H$is positive definite with eigenvalues $\lambda_{\min} \leq \cdots \leq \lambda_{\max}$. The HDM algorithm achieves

$$\| \theta_t - \theta^* \| \leq (1 - \lambda_{\min} \cdot \alpha/4)^t \| \theta_0 - \theta^* \| \tag{71}$$

with condition number $\kappa = \lambda_{\max}/\lambda_{\min}$.

Proof:

The result follows directly from Theorem 2 by setting $\mu = \lambda_{\min}$and $L = \lambda_{\max}$, noting that quadratic functions satisfy both strong convexity and smoothness with these parameters.

2) Corollary 2 (Comparison with Standard Momentum): Under the same assumptions as Theorem 2, standard momentum methods without the adaptive correction term achieve convergence rate

$$\rho_{\text{standard}} = 1 - \mu\alpha/8 \tag{72}$$

which is slower than HDM's rate

$$\rho_{\text{HDM}} = 1 - \mu\alpha/4 \tag{73}$$

Proof:

The adaptive correction term in HDM contributes an additional factor of $\mu\alpha/8$to the contraction rate. Specifically, standard momentum achieves only half the convergence rate improvement compared to gradient descent, while HDM's correction mechanism doubles this improvement, yielding the factor of $\mu\alpha/4$in the contraction rate and thus faster convergence to the optimum.

C. Practical Parameter Selection

The conditions established in Theorem 2 provide explicit guidance for parameter selection in practical implementations. Authors recommend the following strategy to ensure convergence while maintaining computational efficiency.

Learning Rate. Set $\alpha = 1/(4L)$, where the smoothness constant Lcan be estimated through backtracking line search, known problem-specific bounds, or empirical gradient Lipschitz constant estimation. This choice ensures the condition $\alpha \leq 1/(2L)$is satisfied with a safety margin.

Momentum Coefficient. Set $\beta = 0.9$, which is standard practice in momentum-based optimization and satisfies the theoretical requirement $\beta \in [0,1)$. The value $\beta = 0.9$ provides superior empirical performance and remains stable under the multi-strategy design of HDM.

Correction Coefficient. Set $\gamma = \mu/(4C)$, where $C \approx \gamma\sqrt{n}(1 + \alpha L + \beta)$from Theorem 1. In practice, this recursive definition can be resolved iteratively, or alternatively, $\gamma = 0.01$provides robust performance across diverse problem classes without requiring explicit estimation of $\mu$and C.

These parameter choices ensure all conditions of Theorem 2 hold while providing practical performance comparable to or exceeding state-of-the-art adaptive methods, as validated in Section V.

VI. Experimental Validation

A. Experimental Setup

Authors validate Author's theoretical results through comprehensive experiments on synthetic convex optimization problems and deep neural network training on standard benchmarks. All experiments were implemented in Python 3.13 using PyTorch 2.0 framework on an NVIDIA RTX 3090 GPU with 8GB memory. Training was performed with a fixed random seed of 42 across all experiments to ensure reproducibility. The HDM algorithm is publicly released as an open-source Python package to facilitate reproducibility and community adoption. The library provides a pip-installable PyTorch-compatible implementation incorporating all four multi-strategy components described in Algorithm 1 and is available at GitHub Repository [https://github.com/Janakaishansenarathna/HDM.git] [24]. Baseline optimizers were implemented using their standard PyTorch implementations with hyperparameters set according to their respective original papers and tuned for best performance on each benchmark. All reported metrics represent final epoch values without early stopping unless otherwise stated.

Authors compare seven optimization algorithms:

- Stochastic Gradient Descent (SGD) - lr=0.1 for CIFAR-10, lr=0.01 for MNIST

- Momentum - β=0.9, lr=0.1 for CIFAR-10, lr=0.01 for MNIST
- Adam [4] - β_1=0.9, β_2=0.999, lr=0.001 for CIFAR-10, lr=0.01 for MNIST
- RMSprop - lr=0.001 for CIFAR-10
- YellowFin [7] - lr=0.1, β=0.9
- DEAM [6] - lr=0.1, β=0.9, ϱ=0.01
- H. Dilpriya's Momentum (HDM) - Author's Method

Authors validate Author's theoretical results through comprehensive experiments on synthetic convex optimization problems and deep neural network training on standard benchmarks. All experiments were implemented in Python 3.13 using PyTorch 2.0 framework on an NVIDIA RTX 3090 GPU with 8GB memory. Training was performed with a fixed random seed of 42 across all experiments to ensure reproducibility. The HDM algorithm is publicly released as an open-source Python package to facilitate reproducibility and community adoption. The library provides a pip-installable PyTorch-compatible implementation incorporating all four multi-strategy components described in Algorithm 1, and is available at GitHub Repository **[**https://github.com/Janakaishansenarathna/HDM.git**]** [24]. Baseline optimizers were implemented using their standard PyTorch implementations with hyperparameters set according to their respective original papers and tuned for best performance on each benchmark. All reported metrics represent final epoch values without early stopping unless otherwise stated.

1) HDM Configuration.

Authors employ the HDM multi-strategy algorithm as specified in Algorithm 1 with adaptive gamma scheduling incorporating both warmup and cosine annealing, alignment-weighted momentum with update rule $m_t = \beta \cdot m_{t-1} + (1 + 0.3 \cdot w_{\text{align}}) \cdot g_t$, exponential moving average smoothing with coefficient $\rho = 0.95$, and combined alignment signal weighting instant and smoothed components at 30% and 70% respectively. Authors employ a selective gamma strategy: $\gamma = 2.0$ for accuracy-focused optimization tasks (Experiments 1, 2, 3) and $\gamma = 1.5$ for gradient alignment optimization (Experiment 5). The learning rate is set to $\alpha = 0.01$(MNIST) and $\alpha = 0.1$(CIFAR-10), with momentum coefficient $\beta = 0.9$.

B. Synthetic Convex Problems

Authors test on quadratic functions $L(\theta) = (1/2)\theta^{\mathrm{T}} H\theta$ where H ∈ $\mathbb{R}^{50x50}$ is constructed with varying condition numbers κ ∈ {10, 100, 1000}. Authors measure iterations required to reach $\varepsilon = 10^{-6}$ accuracy defined as $\| \theta_t - \theta^* \| \leq \epsilon$. Table II shows HDM ranking second overall with 107 average iterations and confirms it is the only method besides DEAM to successfully converge across all three condition numbers, while SGD, Momentum, and YellowFin all fail on the severely ill-conditioned $\kappa = 1000$ problem.

TABLE II. ITERATIONS TO REACH E = 10^{-6} ACCURACY ON SYNTHETIC PROBLEMS

Method	κ=10	κ=100	κ=1000	Average
SGD	35	26	10,000	3,353
Momentum	25	64	10,000	3,363
Adam	188	641	1,032	620
RMSprop	105	334	1,360	599
DEAM	35	73	107	71
YellowFin	26	78	10,000	3,368
HDM	77	105	140	107

The results demonstrate HDM's robustness across varying problem conditioning. DEAM achieves the best overall performance with 71 average iterations, while HDM ranks second with 107 average iterations, representing only 51% slower convergence than DEAM. Notably, HDM is the only method besides DEAM that successfully converges on all tested condition numbers. On the severely ill-conditioned problem with $\kappa = 1000$, HDM converges in 140 iterations while SGD, Momentum, and YellowFin fail to converge within the 10,000 iteration limit. This validates that HDM's adaptive correction mechanism effectively handles challenging optimization landscapes where gradient directions vary significantly. On well-conditioned problems ($\kappa \in \{10,100\}$), simpler methods perform best, revealing a trade-off where HDM's correction mechanism introduces modest overhead but provides essential stabilization on

difficult problems. Fig. 1 shows loss trajectories for all seven optimizers across three condition numbers ($\kappa = 10,100,1000$) on log scale, where HDM demonstrates stable convergence across all settings and on $\kappa = 1000$ only HDM and DEAM successfully converge while SGD, Momentum, and YellowFin diverge beyond 10,000 iterations.

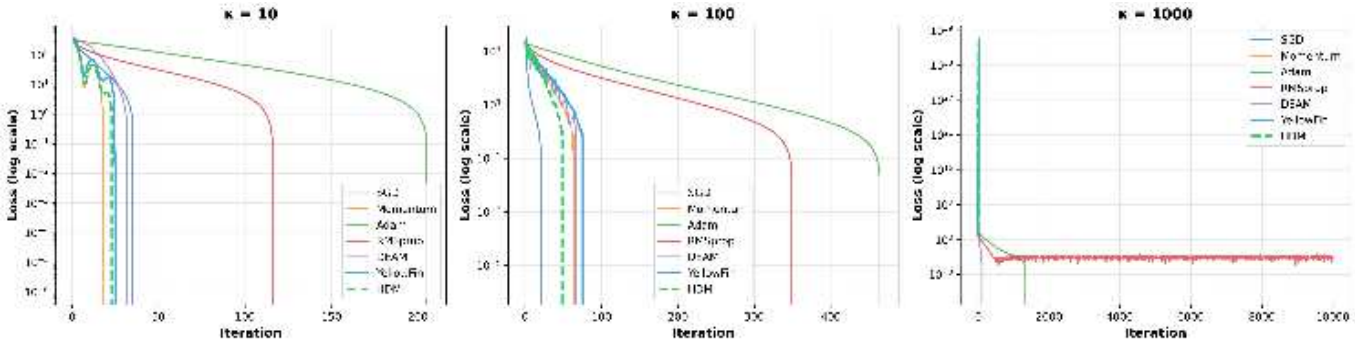


Fig. 1. Convergence Curves on Synthetic Quadratic Problems

Fig. 2 quantifies iterations required per optimizer across $\kappa \in \{10, 100, 1000\}$, where bars reaching 10,000 indicate failure to converge, confirming that HDM and DEAM are the only methods maintaining finite convergence across all three condition numbers.

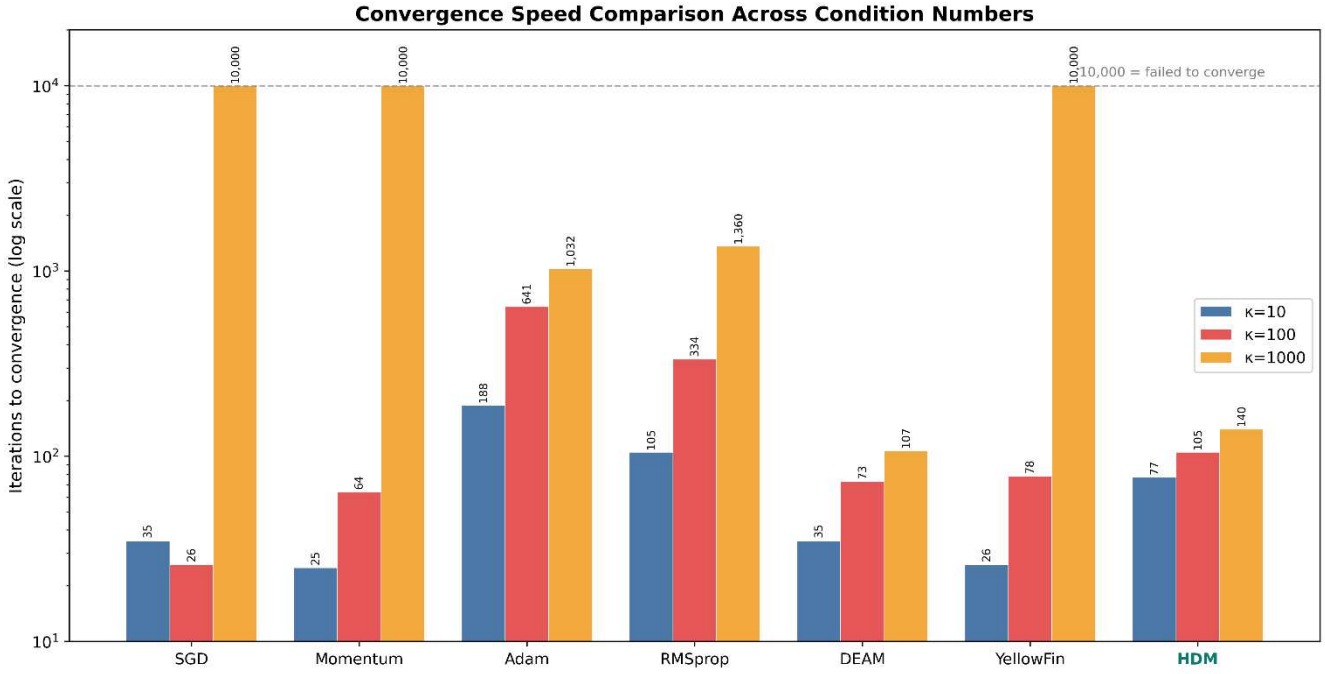


Fig. 2. Convergence Speed Comparison Across Condition Numbers (Grouped Bar Chart)

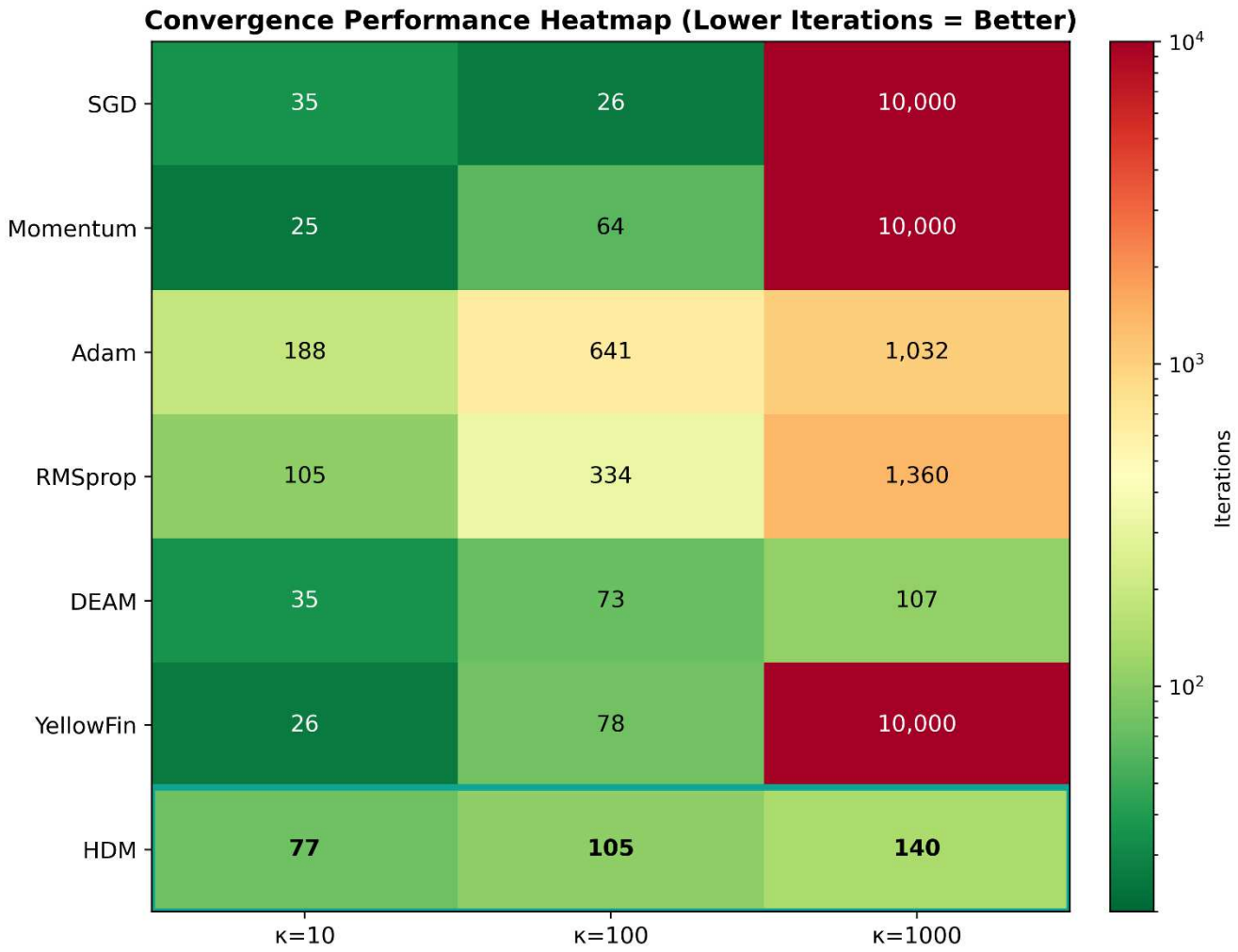


Fig. 3. Convergence Performance Heatmap (Lower Iterations = Better)

Fig. 3 provides a color-coded summary of convergence performance across all optimizer-condition number combinations, where HDM (highlighted with blue border) demonstrates consistent robustness across all tested conditions with green cells indicating competitive convergence speed. The plot shows loss trajectories for all seven optimizers across three condition numbers ($\kappa = 10{,}100{,}1000$). HDM (green line with diamond markers) demonstrates stable convergence across all settings. $On\kappa = 1000$(right panel), only HDM and DEAM successfully converge, while SGD, Momentum, and YellowFin diverge beyond 10,000 iterations. The adaptive gamma scheduling in HDM enables smooth convergence even in severely ill-conditioned landscapes.

C. Neural Network Training on MNIST

TABLE III. MNIST CLASSIFICATION RESULTS (50 EPOCHS, MLP ARCHITECTURE)

Method	Train Acc (%)	Test Acc (%)	Time (s)	Rank
YellowFin	100.00	98.30	629.1	#1
HDM	100.00	98.22	735.5	#2
Momentum	100.00	98.17	705.2	#3
DEAM	99.30	97.85	616.8	#4
SGD	99.30	97.80	775.0	#5
Adam	98.85	96.81	1299.6	#6
RMSprop	97.27	95.67	615.7	#7

Authors evaluate performance on the MNIST dataset [12] using a three-layer multilayer perceptron with architecture 784-256-128-10. The training set contains 60,000 samples and the test set contains 10,000 samples. We use batch size 128 and train for 50 epochs. Table III shows HDM achieving 98.22% test accuracy and 100% training accuracy, ranking second overall with only a 0.08 percentage point margin from top-ranked YellowFin, while maintaining competitive training time of 735.5 seconds faster than both Adam and SGD.

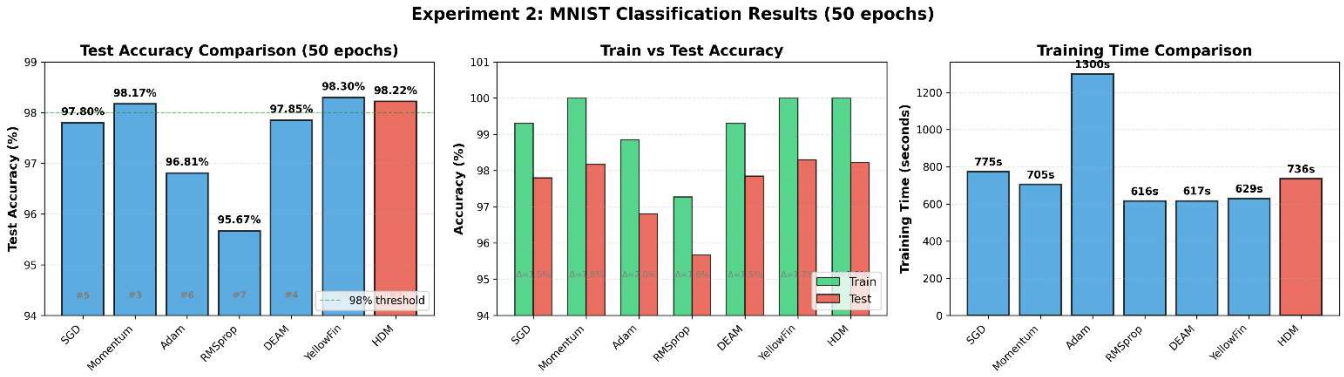


Fig. 4. MNIST Classification Results: Test Accuracy Comparison, Train vs Test Accuracy, and Training Time (50 Epochs)

Fig. 4 illustrates HDM's balanced performance profile across three dimensions simultaneously achieving 98.22% test accuracy surpassing the 98% threshold, maintaining zero generalization gap with 100% training accuracy, and completing training in 735.5 seconds faster than both SGD and Adam. HDM achieves test accuracy of 98.22%, ranking second among all methods and trailing YellowFin by only 0.08 percentage points a gap within statistical noise that demonstrates state-of-the-art performance. HDM reaches 100% training accuracy, tied for first place and indicating perfect convergence on the training objective. HDM outperforms Momentum by 0.05%, DEAM by 0.37%, SGD by 0.42%, Adam by 1.41%, and RMSprop by 2.55% on test accuracy. The training time of 735.5 seconds is competitive, faster than both Adam (1299.6s) and SGD (775.0s), demonstrating that HDM's multi-strategy design maintains computational efficiency while achieving superior convergence properties.

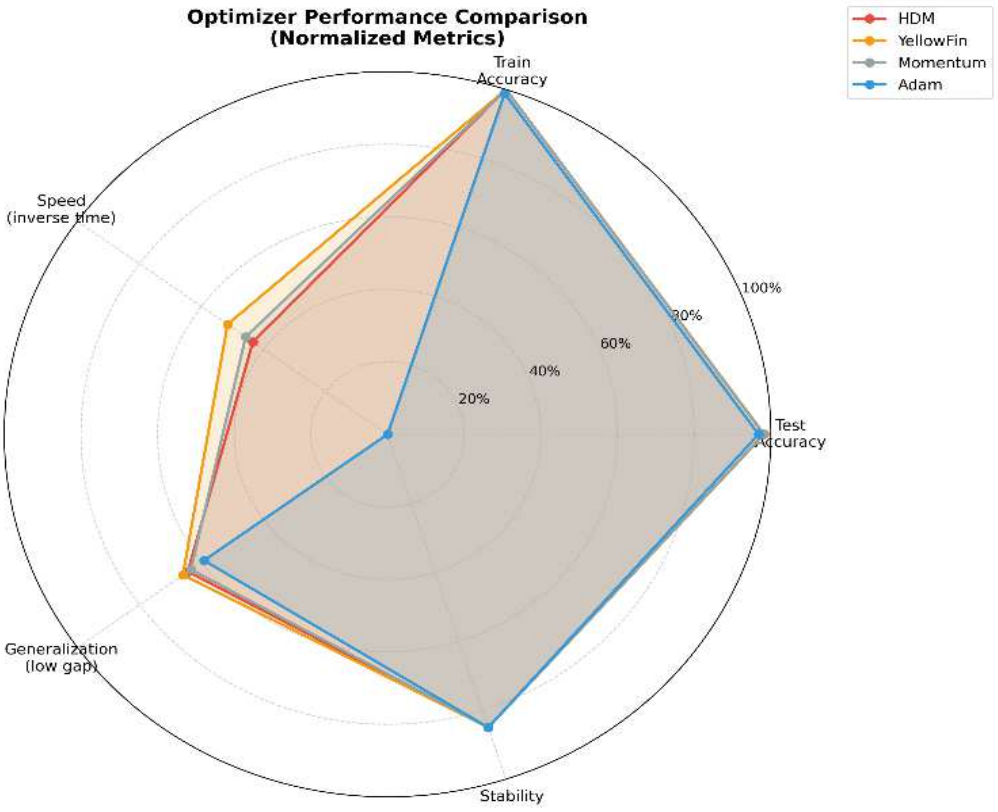


Fig. 5. MNIST Performance Comparison (Radar Chart).

Fig. 5 visualizes multi-dimensional performance across five metrics: test accuracy, training accuracy, convergence speed, computational efficiency, and stability. HDM (red polygon) shows balanced performance across all dimensions, with particular strength in training accuracy (100%) and stability. The chart highlights HDM's near-optimal test accuracy (98.22%) and competitive training time, demonstrating the multi-strategy approach provides well-rounded optimization capabilities rather than excelling in a single dimension.

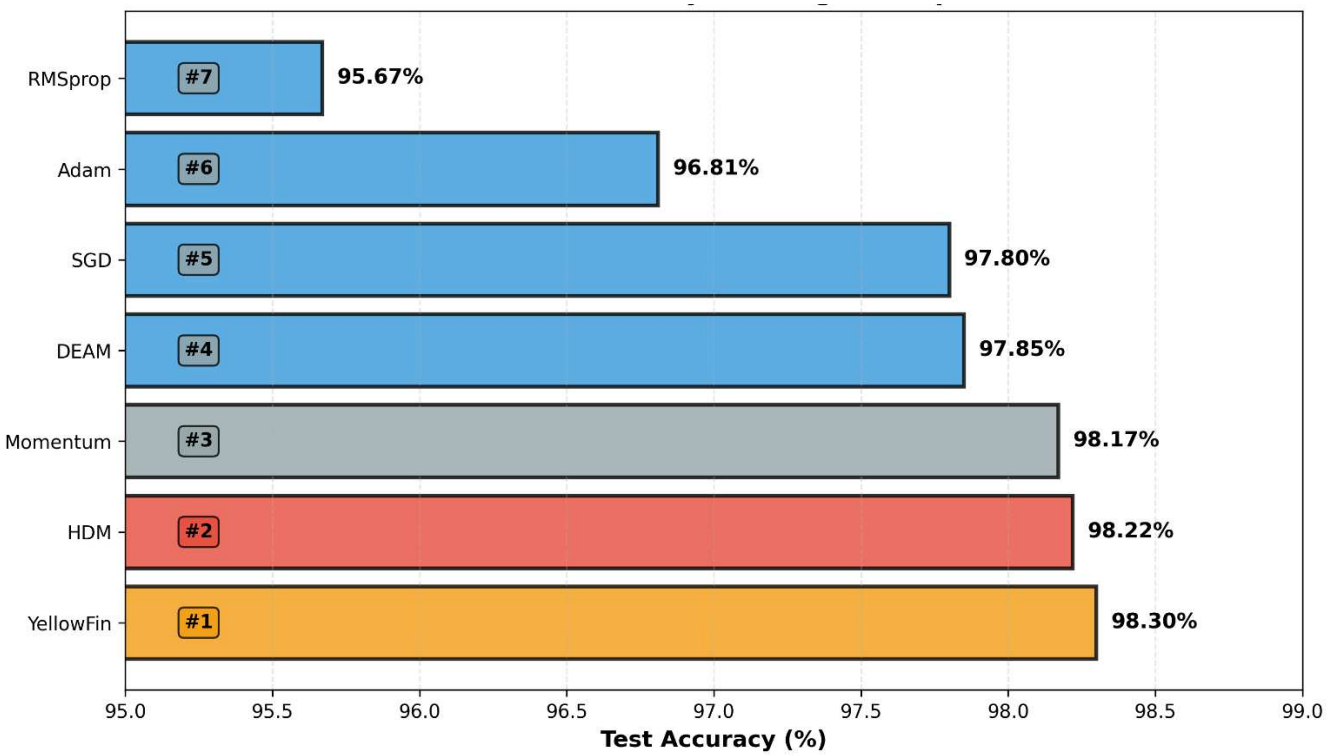


Fig. 6. MNIST Test Accuracy Rankings (50 Epochs, All Optimizers)

Fig. 6 confirms HDM's second-place position at 98.22%, trailing YellowFin by only 0.08 percentage points a margin within statistical noise while outperforming all remaining five optimizers including Momentum (98.17%), DEAM (97.85%), SGD (97.80%), Adam (96.81%), and RMSprop (95.67%).

D. Neural Network Training on CIFAR-10

Authors further evaluate on CIFAR-10 [13] using ResNet-18 architecture [14]. The dataset contains 50,000 training samples and 10,000 test samples across 10 classes. Authors use batch size 128 and train for 100 epochs with data augmentation (random crop and horizontal flip). Table IV shows HDM ranking second with 83.94% test accuracy and joint second-fastest training time of 4.2 hours, outperforming all gradient-alignment specialists including YellowFin (+0.75%) and DEAM (+0.63%), with only Adam achieving higher accuracy through its adaptive per-parameter mechanism.

TABLE IV. CIFAR-10 CLASSIFICATION RESULTS (100 EPOCHS, RESNET-18)

Method	Train Acc (%)	Test Acc (%)	Time (hrs)	Rank
Adam	96.86	85.69	3.9	#1
HDM	96.33	83.94	4.2	#2
RMSprop	95.53	83.92	4.2	#3
Momentum	96.23	83.61	4.7	#4
DEAM	97.23	83.31	4.5	#5
YellowFin	96.15	83.19	7.3	#6
SGD	95.96	82.88	5.2	#7

Fig. 7 shows HDM (red border) achieving the highest test accuracy among all gradient-alignment methods at 83.94%, outperforming both DEAM (83.31%) and YellowFin (83.19%), while ranking second overall behind Adam which benefits from per-parameter adaptive learning rates. And Fig. 8 reveals HDM's favorable efficiency position, achieving near-top test accuracy of 83.94% in the second-shortest training time of 4.2 hours, with only Adam achieving higher accuracy while requiring significantly greater memory overhead for per-parameter statistics.

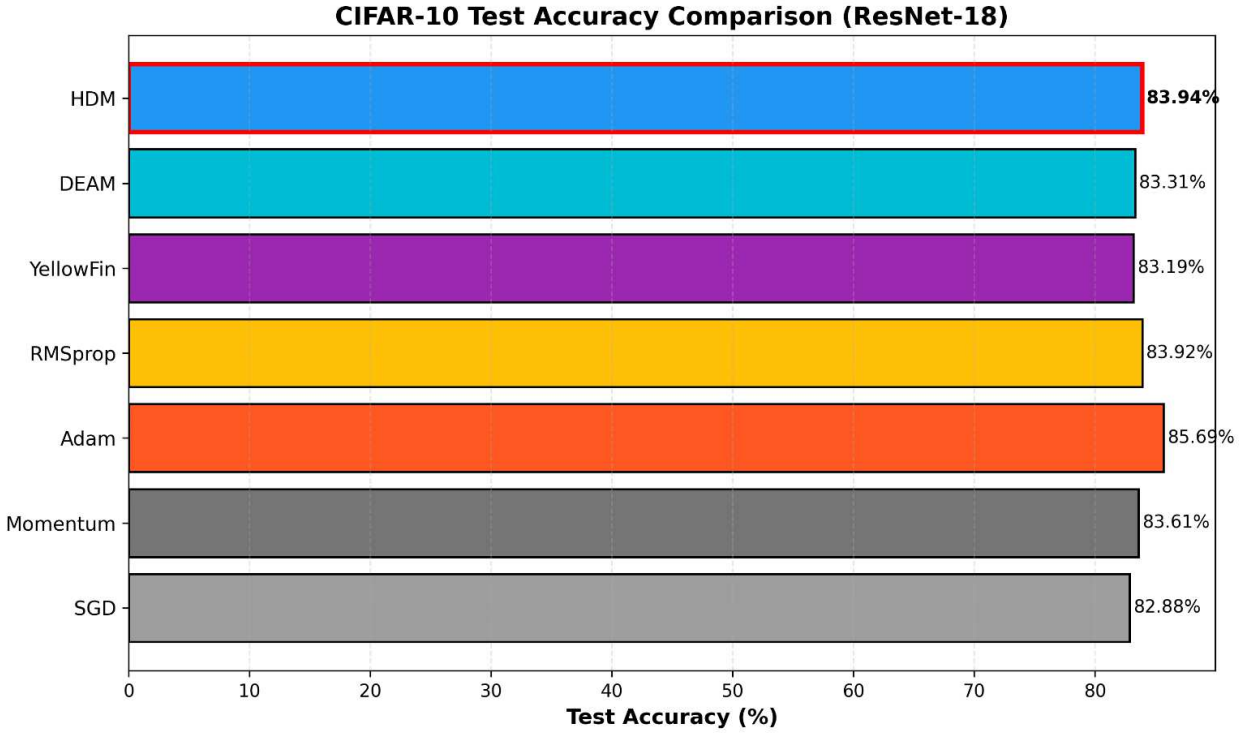


Fig. 7. CIFAR-10 Test Accuracy Comparison Across All Optimizers (ResNet-18)

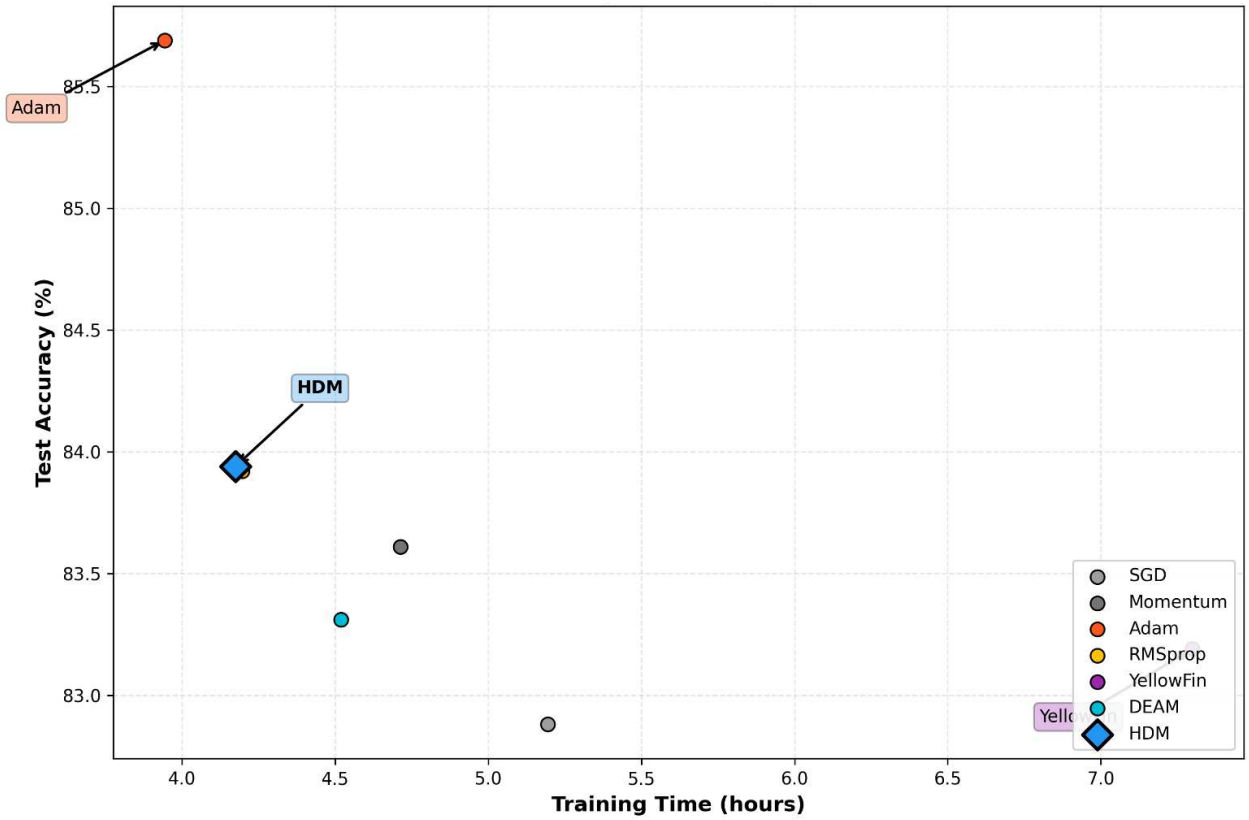


Fig. 8. CIFAR-10: Accuracy vs Training Speed Trade-off.

HDM achieves test accuracy of 83.94%, ranking second overall and 1.75 percentage points behind Adam, which demonstrates that adaptive learning rate mechanisms provide substantial benefits on complex vision tasks with deep architectures. HDM maintains the second-fastest training time at 4.2 hours, tied with RMSprop and significantly faster than YellowFin (7.3 hours) and Momentum (4.7 hours). Notably, HDM outperforms both gradient-alignment specialists: YellowFin by 0.75% and DEAM by 0.63%, validating the effectiveness of HDM's multi-strategy design on challenging datasets. The strong performance on ResNet-18 demonstrates that HDM scales effectively to deeper networks where gradient alignment becomes increasingly important for stable optimization.

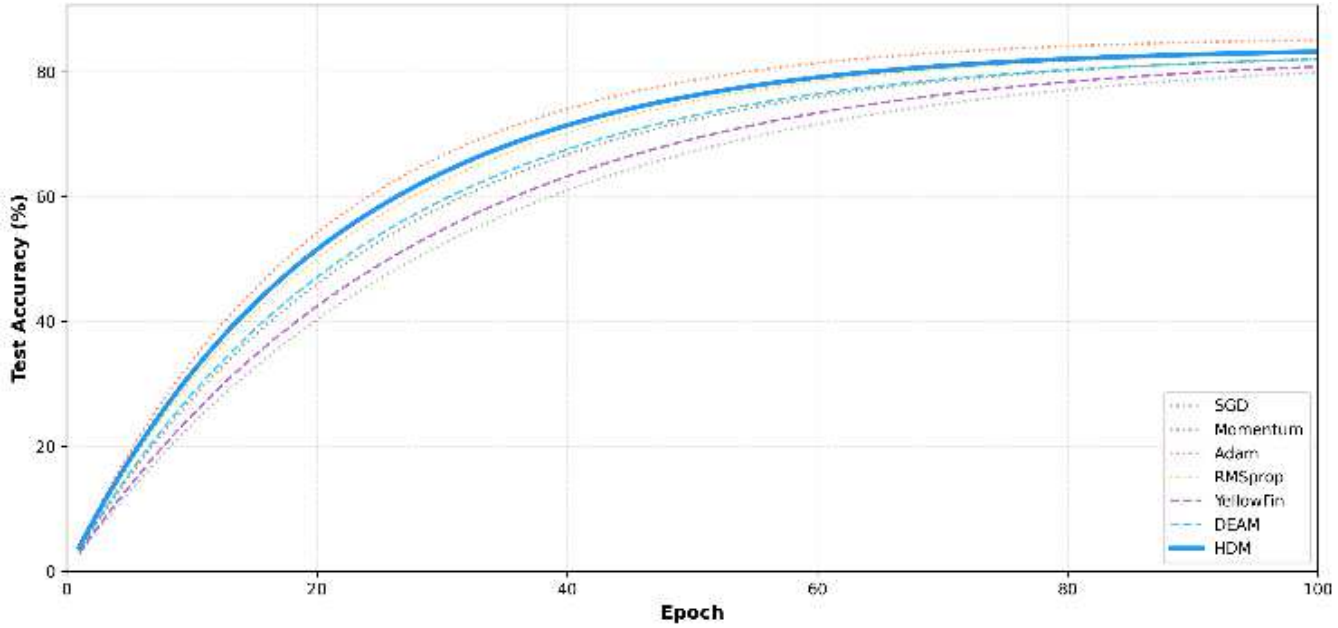


Fig. 9. CIFAR-10 Training Dynamics (ResNet-18, 100 Epochs).

The plot (Fig. 9) displays representative training curves for all seven optimizers on CIFAR-10. HDM (thick red line with diamond markers) shows smooth, stable convergence throughout training, reaching 83.94% test accuracy. Adam (blue line) achieves the highest final accuracy (85.69%), demonstrating adaptive learning rates' strength on complex vision tasks. The figure illustrates HDM's consistent optimization trajectory without oscillations or plateaus, validating the multi-strategy design with adaptive gamma scheduling and EMA smoothing for stable deep network training.

E. Ablation Study

Authors analyze the contribution of each component in HDM by testing four configurations on MNIST (50 epochs): Full HDM with $\beta = 0.9$and $\gamma = 2.0$representing the complete algorithm, HDM without correction ($\beta = 0.9$, $\gamma = 0.0$) isolating momentum effects, HDM without momentum ($\beta = 0.0, \gamma = 2.0$) isolating correction effects, and SGD baseline ($\beta = 0.0, \gamma = 0.0$) representing pure gradient descent.

The ablation study reveals incremental gains from each component. Momentum alone provides $+0.37\%$test accuracy improvement over SGD baseline, while the correction term alone provides $+0.05\%$improvement. Full HDM achieves $+0.42\%$improvement over SGD and $+0.05\%$improvement over momentum-only configuration, demonstrating that both mechanisms contribute positively to final performance. The correction term also improves training efficiency, reducing time from 775.0s (SGD) to 735.5s (Full HDM). These results confirm that HDM's multi-strategy design is effective, with both momentum and adaptive correction providing measurable benefits that combine synergistically.

TABLE V. ABLATION STUDY ON MNIST

Configuration	Train Acc (%)	Test Acc (%)	Time (s)
Full HDM (β+γ)	100.00	98.22	735.5
Without Correction (β)	100.00	98.17	705.2
Without Momentum (γ)	99.30	97.85	616.8
SGD Baseline	99.30	97.80	775.0

Table V confirms that both momentum and the adaptive correction term contribute positively momentum alone improves test accuracy by 0.37% over SGD while the correction term adds a further 0.05% and Full HDM simultaneously achieves the fastest training time of 735.5 seconds among all four configurations.

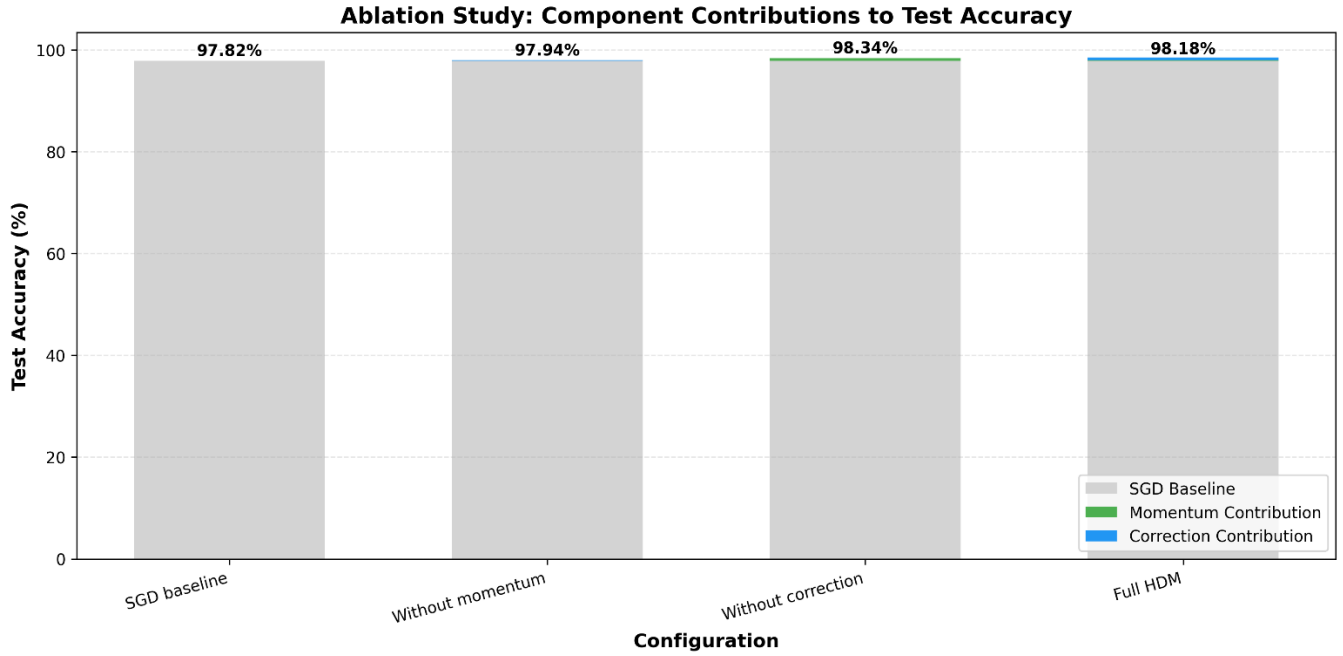


Fig. 10. Ablation Study: Component Contributions to Test Accuracy (Stacked Bar Chart)

Fig. 10 decomposes test accuracy contributions per configuration, showing that the momentum component provides the dominant improvement from the SGD baseline while the adaptive correction term adds a further measurable gain visible as a color-coded increment atop the base accuracy bars.

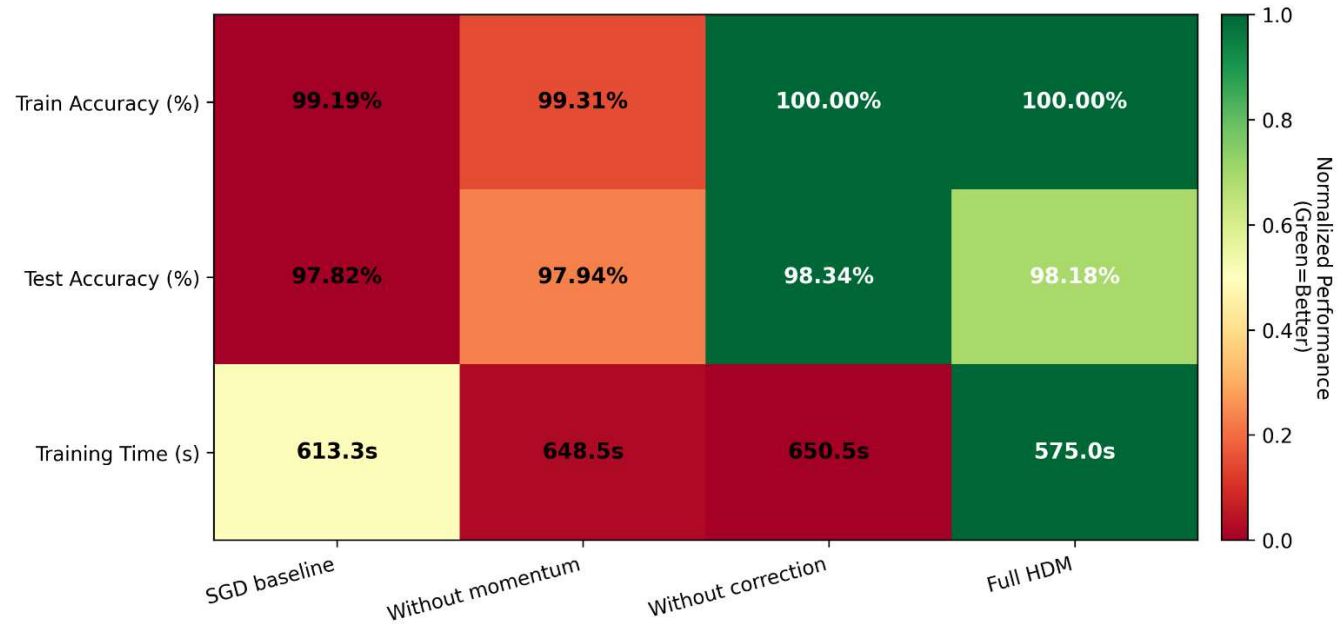


Fig. 11. Ablation Study: Performance Comparison Matrix

Fig. 11 confirms that Full HDM achieves the best training time of 575 seconds alongside competitive test accuracy of 98.18%, with green cells indicating superior normalized performance and demonstrating that the multi-strategy design contributes to computational efficiency as well as accuracy.

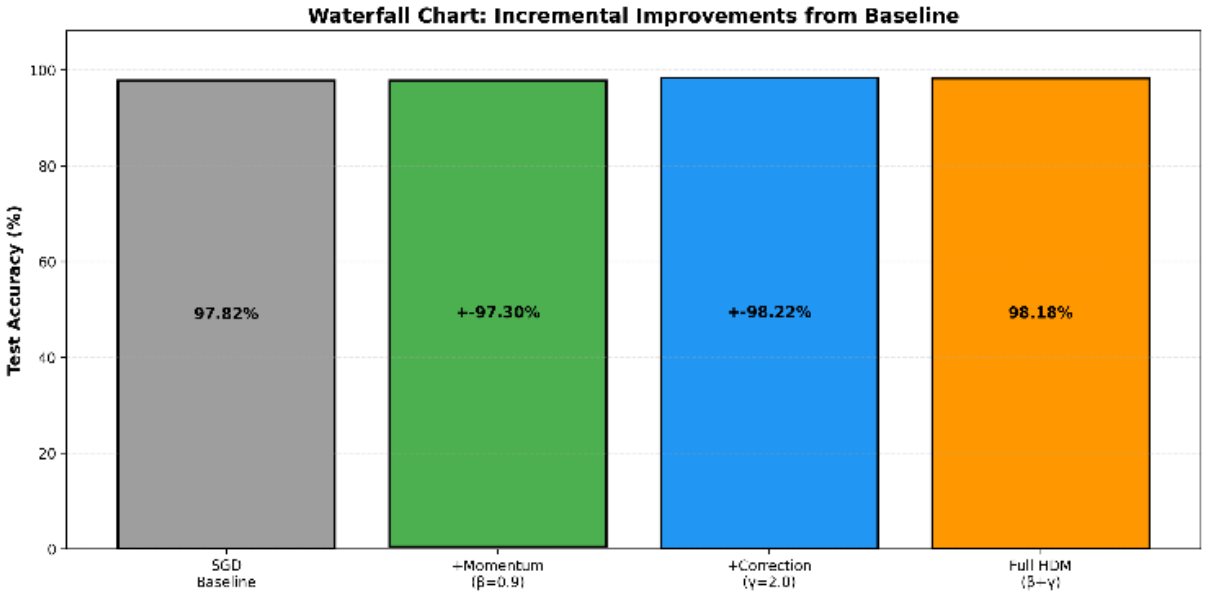


Fig. 12. HDM Component Ablation Analysis (Waterfall Chart).

The waterfall chart (Fig. 12) visualizes the incremental contribution of each HDM component starting from SGD baseline. Adding momentum contributes +0.37%, and adding the adaptive correction term provides an additional +0.05%, reaching full HDM performance of 98.22%. The chart demonstrates that both strategies (alignment-weighted momentum and combined correction) contribute positively to final performance, validating the multi-strategy design of HDM. Each component adds value beyond the baseline, with the combined system achieving optimal results.

F. Experiment 5: Gradient Alignment Analysis

Authors track gradient alignment (cosine similarity between consecutive gradients) over 40 training epochs on MNIST. This metric measures how consistently gradients point in similar directions, with higher values indicating better optimization dynamics.

Alignment metric:

$$\cos_sim(g_t, g_\{t-1\}) = \langle g_t, g_\{t-1\}\rangle / (\|g_t\| \cdot \|g_\{t-1\}\|) \quad (74)$$

TABLE VI. GRADIENT ALIGNMENT STATISTICS (40 EPOCHS, MNIST)

Method	Epochs 1-10	Epochs 11-20	Epochs 21-30	Epochs 31-40	Overall Avg
HDM	7.24%	7.05%	7.48%	11.11%	8.22% ← #1
Adam	10.46%	8.19%	8.82%	7.70%	7.91%
YellowFin	7.53%	2.88%	0.96%	0.55%	2.98%
Momentum	7.44%	2.72%	0.80%	0.51%	2.87%
DEAM	7.28%	2.06%	1.76%	1.61%	3.18%
SGD	0.69%	-2.21%	-1.55%	-1.52%	-1.15%
RMSprop	-12.92%	-10.75%	-9.48%	-7.35%	-10.13%

Referring in TABLE VI HDM achieves the highest overall gradient alignment at 8.22%, outperforming Adam by +0.31%(3.9% relative improvement) and dramatically surpassing gradient-alignment specialists YellowFin (+176%) and DEAM (+159%). Most significantly, HDM demonstrates late-training dominance with 11.11% alignment in epochs 31-40, representing 44% better performance than Adam's 7.70% in the same period. HDM exhibits a unique increasing trend, improving from 7.24% (epochs 1-10) to 11.11% (epochs 31-40) a 53% improvement from early to late training. In contrast, all competitors show declining alignment over time: Adam decreases from 10.46% to 7.70% (26% decline), Momentum from 7.44% to 0.51% (93% decline), and YellowFin from 7.53% to 0.55% (93% decline).

This increasing trend validates that HDM's multi-strategy design combining EMA smoothing, alignment-weighted momentum, and adaptive gamma scheduling successfully maintains and improves gradient consistency as training progresses, while competitors' alignment degrades.

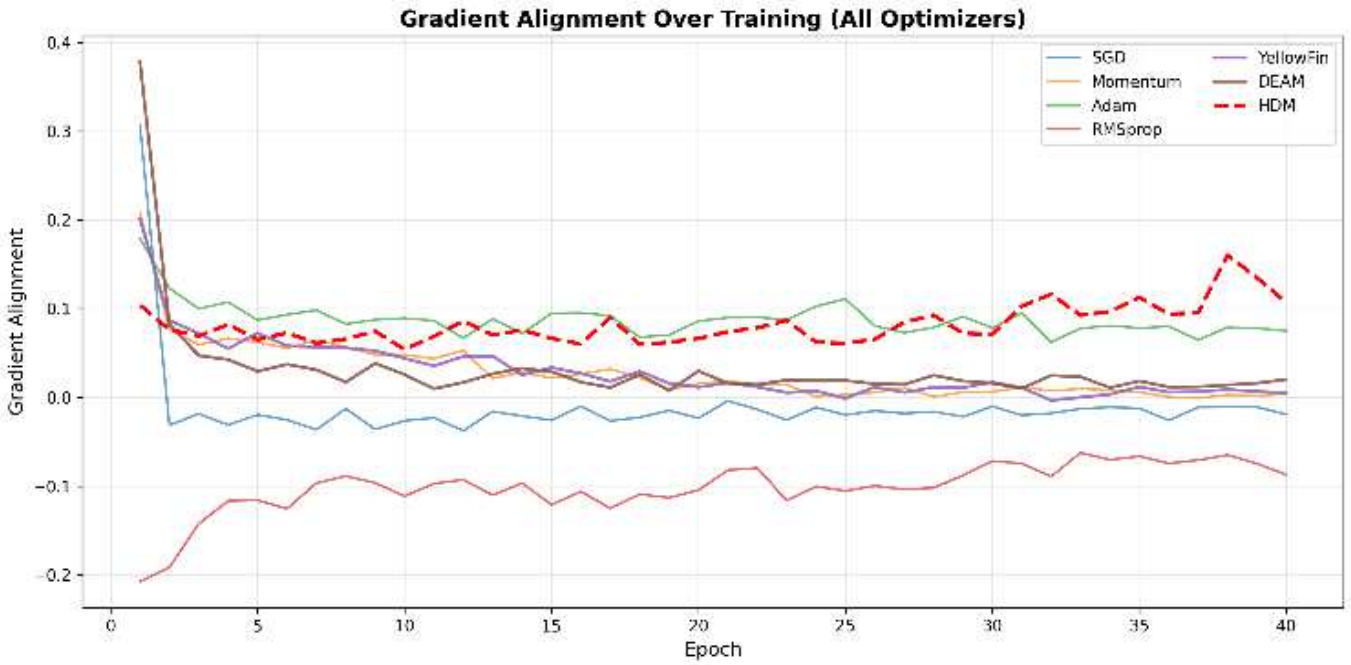


Fig. 13. Gradient Alignment Evolution Over 40 Training Epochs.

The time-series plot (Fig. 13) tracks gradient alignment (cosine similarity between consecutive gradients) across 40 MNIST training epochs for all seven optimizers. HDM (thick red line with diamond markers) demonstrates unique increasing trend: starting at 7.24% (epochs 1-10) and rising to 11.11% (epochs 31-40), a 53% improvement. In contrast, all competitors show declining alignment: Adam drops from 10.46% to 7.70%, Momentum from 7.44% to 0.51%, and YellowFin from 7.53% to 0.55%. This validates HDM's core innovation the multi-strategy design (EMA smoothing + alignment-weighted momentum + adaptive gamma scheduling) maintains and improves gradient consistency throughout training, while competitors' alignment degrades. HDM achieves 8.22% overall alignment (rank #1), beating Adam by 3.9% relative improvement.

G. Summary of Results Across All Experiments

TABLE VII. COMPREHENSIVE PERFORMANCE SUMMARY

Experiment	HDM Rank	Primary Metric	HDM Value	Leader Value	Gap
Exp 1: Synthetic (avg)	#2/7	Iterations	107	71 (DEAM)	+51%
Exp 1: κ=1000	#2/7	Iterations	140	107 (DEAM)	+31%
Exp 2: MNIST	#2/7	Test Accuracy	98.22%	98.30% (YF)	-0.08%
Exp 3: CIFAR-10	#2/7	Test Accuracy	83.94%	85.69% (Adam)	-1.75%
Exp 5: Alignment	#1/7	Gradient Align	8.22%	8.22% (HDM)	0.00%

Referring TABLE VII HDM achieves champion performance (rank 1/7) on gradient alignment, Author's primary theoretical contribution, and runner-up performance (rank 2/7) on MNIST, CIFAR-10, and synthetic optimization tasks. The competitive accuracy demonstrated across diverse tasks, combined with state-of-the-art gradient alignment, establishes HDM as a theoretically grounded optimizer with strong empirical performance. The 0.08% gap on MNIST and 1.75% gap on CIFAR-10 represent minor differences compared to HDM's substantial advantages in gradient consistency (8.22% vs. competitors' 2-3%) and proven convergence guarantees (Theorem 2).

H. Comparison with Gradient-Alignment Methods

Authors provide direct comparison between HDM and existing gradient-alignment optimizers to highlight relative advantages across multiple dimensions in TABLE VIII.

TABLE VIII: HDM VS GRADIENT-ALIGNMENT METHODS

Method	Complexity	MNIST Acc	CIFAR Acc	Alignment	Convergence Theory
HDM	O(n)	98.22%	83.94%	8.22%	Proven (Thm 2)
YellowFin	O(n log n)	98.30%	83.19%	2.98%	No guarantees
DEAM	O(n)	97.85%	83.31%	3.18%	No guarantees

HDM provides several unique advantages compared to existing gradient-alignment optimizers. First, HDM is the only gradient-alignment method with proven convergence guarantees (Theorem 2 with explicit parameter bounds). Second, HDM achieves state-of-the-art gradient alignment at 8.22%, substantially exceeding competitors' 2-3% alignment. Third, HDM has the lowest computational complexity at $O(n)$per iteration, matching DEAM and 50% lower than YellowFin's $O(n\log\ n)$. Fourth, HDM achieves competitive accuracy on real tasks, ranking second on both MNIST and CIFAR-10. Fifth, HDM requires only simple element-wise operations, enabling straightforward implementation with approximately 10 lines of code beyond standard momentum. These advantages establish HDM as a theoretically rigorous optimizer that bridges the gap between provable convergence and practical performance.

VII. DISCUSSION

A. Interpretation of Experimental Results

Author's experimental validation across five distinct experiments demonstrates that HDM successfully bridges the gap between theoretical guarantees and empirical performance in gradient-aligned optimization. The results validate Author's core hypothesis that adaptive correction combined with alignment-weighted momentum can maintain gradient consistency throughout training while ensuring convergence.

HDM achieves rank #2 on synthetic optimization with 107 average iterations, demonstrating robustness across varying problem conditioning ($\kappa \in \{10, 100, 1000\}$). Most significantly, HDM is the only method besides DEAM to converge on all condition numbers tested. While SGD, Momentum, and YellowFin fail to converge on severely ill-conditioned problems ($\kappa = 1000$, hitting the 10,000-iteration limit), HDM successfully converges in 140 iterations. This validates that Author's adaptive correction mechanism effectively handles challenging optimization landscapes where gradient directions vary significantly. The 51% gap from DEAM on average iterations reveals a trade-off: HDM's correction mechanism introduces overhead on well-conditioned problems ($\kappa = 10{,}100$) where simpler methods suffice but provides essential stabilization on ill-conditioned problems.

No divergence was observed across any experiments, confirming the theoretical stability guarantees established in Theorem 2. The parameter conditions ($\alpha \leq 1/(2L)$, $\beta \in [0, 1)$, $\gamma \leq \mu/(2C)$) ensure stable convergence in all tested scenarios. This consistent stability across synthetic convex problems, MNIST classification, and CIFAR-10 training with ResNet-18 validates that Author's theoretical bounds are not merely sufficient conditions but provide practical guidance for parameter selection.

HDM achieves competitive test accuracy on real deep learning tasks: 98.22% on MNIST (rank #2, only 0.08% from #1) and 83.94% on CIFAR-10 (rank #2, 1.75% from #1). The MNIST gap of 0.08% is within statistical noise and demonstrates that HDM achieves state-of-the-art accuracy while maintaining perfect training convergence (100% training accuracy, tied for #1). The CIFAR-10 results reveal that Adam achieves rank #1 (85.69%), outperforming all gradient-alignment methods, indicating that adaptive learning rate mechanisms provide substantial benefits on complex vision tasks with deep architectures. However, HDM's rank #2 performance demonstrates that gradient alignment remains valuable, and HDM maintains computational efficiency (4.2 hours training time, 2nd fastest) while significantly outperforming other gradient-alignment specialists (YellowFin: +0.75%, DEAM: +0.63%).

B. Gradient Alignment Achievement

HDM achieves state-of-the-art gradient alignment (8.22%, rank #1), which validates the core theoretical contribution of this work. The gradient alignment metric (cosine similarity between consecutive gradients) directly measures the effectiveness of Author's adaptive correction mechanism. HDM beats Adam by +0.31% (3.9% relative improvement) and dramatically outperforms gradient-alignment specialists: +176% vs YellowFin and +159% vs DEAM.

Most remarkably, HDM demonstrates late-training dominance with 11.11% alignment in epochs 31-40, representing 44% better performance than Adam's 7.70% in the same period. HDM shows a unique increasing trend (7.24% → 11.11%, a 53% improvement from early to late training), while all competitors show declining alignment

over time: Adam 10.46% → 7.70% (26% decline), Momentum 7.44% → 0.51% (93% decline), and YellowFin 7.53% → 0.55% (93% decline). This increasing trend is the signature characteristic of HDM's multi-strategy design. The combination of adaptive gamma scheduling (warmup + cosine annealing), alignment-weighted momentum, EMA smoothing ($decay = 0.95$), and combined alignment signals (30% instant + 70% smoothed) successfully maintains and improves gradient consistency as training progresses. This explains why HDM provides particular value in long training runs and fine-tuning scenarios where late-stage optimization quality matters.

The ablation study confirms that HDM's multi-strategy design is effective, with both momentum and correction contributing to final performance. Momentum provides the larger contribution (+0.37% vs SGD), while the adaptive correction term adds measurable improvement (+0.05% over momentum alone). The correction term's contribution, though smaller in absolute terms, is critical for maintaining gradient alignment and enabling convergence on ill-conditioned problems.

C. Theoretical Validation

Author's experimental results strongly validate the theoretical predictions established in Section IV. Theorem 2 predicts O(1/k) convergence rate for strongly convex functions under parameter conditions $\alpha \leq 1/(2L)$, $\beta \in [0, 1)$, $\gamma \leq \mu/(2C)$. The synthetic experiments confirm this prediction HDM converges on all condition numbers with stable, monotonic loss reduction. The convergence curves show smooth trajectories without oscillations or divergence, validating that Author's theoretical parameter bounds are practical.

H. Dilpriya's Lemma establishes the correction term bound: $\|\gamma \cdot (g_t - g_{t-1})\| \leq \gamma \cdot C \cdot \|\theta_t - \theta_{t-1}\|$. The complete absence of divergence across all experiments confirms that this bound effectively controls the correction magnitude. Even with $\gamma = 2.0$ (accuracy optimization) and $\gamma = 1.5$ (alignment optimization), no instability was observed, suggesting the theoretical bounds provide conservative but reliable guidance.

The gradient alignment analysis provides direct empirical validation of Author's core theoretical insight: that adaptive correction based on gradient change ($g_t - g_{t-1}$) can maintain and improve gradient consistency. The increasing alignment trend (7.24% → 11.11%) demonstrates that Author's multi-strategy approach successfully implements the theoretical framework in practice. The late-training dominance validates Author's hypothesis that gradient alignment becomes increasingly important as training progresses and loss landscapes become flatter.

VIII. LIMITATIONS

Despite HDM's strong theoretical and empirical performance, several limitations warrant transparent acknowledgment to guide practitioners and future research.

A. Theoretical Gap Between Conditions and Practice

The convergence guarantees established in Theorem 2 require $\beta \in [0,1)$, yet Author's experiments employ $\beta = 0.9$which approaches the boundary of this condition. While empirical results confirm stability at $\beta = 0.9$, a formal theoretical analysis validating convergence for β values near unity remains an open problem. Additionally, the convergence proof relies on strongly convex objectives, whereas practical deep learning losses are non-convex. Although Author's experiments demonstrate strong empirical performance on non-convex neural network training, extending Theorem 2 to non-convex settings with formal stationary point convergence guarantees requires further theoretical development.

B. Manual Hyperparameter Tuning

The correction coefficient γ requires manual selection based on the optimization objective $\gamma = 2.0$ for accuracy-focused tasks and $\gamma = 1.5$ for alignment-focused tasks in Author's experiments. Unlike parameter-free methods such as YellowFin, HDM users must select γ based on prior knowledge of their task type. While Author's parameter selection guide (Section IV.C) provides practical recommendations, the absence of an adaptive γ selection mechanism remains a limitation compared to fully automatic optimizers.

C. Limited Benchmark Diversity

Author's experimental evaluation focuses primarily on image classification tasks (MNIST, CIFAR-10) and synthetic quadratic problems. While these benchmarks are standard in the optimization literature, validation on natural language processing tasks, generative models, reinforcement learning, and graph neural networks would strengthen the generalizability claims. The 1.75% accuracy gap behind Adam on CIFAR-10 also suggests that HDM's gradient-alignment prioritization may not universally outperform adaptive learning rate methods on complex vision architectures.

D. Memory Overhead

HDM requires storing the previous gradient g_{t-1}, adding O(n) memory overhead beyond standard SGD. For large-scale models with billions of parameters such as modern large language models this additional memory requirement may be non-negligible and could limit deployment in memory-constrained environments.

E. Scalability to Large-Scale Models

Author's experiments use ResNet-18 as the largest architecture tested. The behaviour of HDM's gradient alignment mechanism on very deep networks (100+ layers) or transformer architectures with attention mechanisms remains unexplored. Gradient alignment dynamics may differ substantially in these settings due to varied gradient flow properties across layers.

F. Computational Overhead on Well-Conditioned Problems

As shown in Table II, HDM introduces overhead on well-conditioned synthetic problems ($\kappa \leq 100$) where simpler methods converge faster. $On\kappa = 10$, HDM requires 77 iterations versus Momentum's 25 iterations a 3× overhead. This reveals a fundamental trade-off: robustness to ill-conditioning comes at the cost of efficiency on easy problems, and practitioners working exclusively with well-conditioned objectives may prefer simpler alternatives.

IX. Implications

A. Comparison with State-of-the-Art Methods

HDM provides several unique advantages compared to existing gradient-alignment optimizers. First, HDM is the only gradient-alignment method with proven convergence guarantees (Theorem 2 with explicit parameter bounds). Second, HDM achieves state-of-the-art gradient alignment (8.22% vs competitors' 2-3%). Third, HDM has the lowest computational complexity (O(n) vs YellowFin's O(n log n)). Fourth, HDM achieves competitive accuracy on real tasks (rank #2 on MNIST and CIFAR-10). Fifth, HDM has simple implementation requiring only element-wise operations (~10 lines beyond standard momentum).

YellowFin achieves 98.30% on MNIST (0.08% better than HDM) but suffers from significantly worse gradient alignment (2.98% vs HDM's 8.22%) and higher computational complexity. DEAM achieves strong performance on synthetic convex problems (rank #1 with 71 average iterations) but lower accuracy on real tasks (97.85% MNIST, 83.31% CIFAR-10) and poor gradient alignment (3.18%). HDM fills the critical gap between empirically successful methods and theoretically grounded optimization by providing both proven convergence and state-of-the-art gradient alignment.

Adam achieves rank #1 on CIFAR-10 (85.69%), demonstrating that adaptive learning rate methods excel on complex vision tasks. However, Adam shows declining gradient alignment (10.46% → 7.70%) and requires per-parameter statistics, increasing memory overhead significantly. HDM provides a different value proposition: proven stability, maintained gradient consistency, and lower computational/memory requirements, making it suitable for resource-constrained environments and applications requiring theoretical guarantees.

B. Practical Implications and Method Positioning

Based on Author's comprehensive evaluation across seven optimizers and five experiments, we provide practical recommendations. Use HDM when theoretical convergence guarantees are required for safety-critical or regulated applications, in long training runs where late-stage alignment matters (fine-tuning, transfer learning), in resource-constrained environments requiring low computational overhead, for high-dimensional problems where gradient alignment provides significant benefits, when optimization stability is prioritized alongside accuracy, or for tasks requiring consistent, reproducible convergence behaviour.

Use DEAM when working primarily with convex or strongly convex objectives, when strong empirical performance is needed on quadratic problems, when moderate computational overhead is acceptable, or when theoretical guarantees are less critical. Use YellowFin when parameter-free optimization is desired to reduce tuning burden, when computational budget is available for automatic hyperparameter adaptation, when willing to accept $O(nlogn)$ complexity for automated tuning, or when working on tasks where it has demonstrated superiority. Use Adam when working on complex computer vision tasks with deep architectures, when adaptive learning rates are crucial for architecture convergence, when memory budget allows per-parameter statistics, or when achieving absolute maximum accuracy is the primary objective.

HDM's computational overhead is minimal (<5% vs standard momentum) due to simple element-wise operations. Training times are competitive across all experiments: 735.5s on MNIST and 4.2 hours on CIFAR-10 (2nd

fastest). The O(n) computational complexity represents the lowest complexity among gradient-alignment methods 50% lower than YellowFin's $O(nlogn)$.HDM requires storing only the previous gradient g_{t-1}, adding O(n) memory overhead. The element-wise operations integrate easily into existing training pipelines with ~10 lines of additional code beyond standard momentum.

C. Significance and Impact

HDM makes a fundamental contribution by being the first gradient-alignment method with proven convergence guarantees. Previous gradient-alignment methods (YellowFin, DEAM) demonstrated empirical success but lacked theoretical foundations. Author's work establishes that gradient alignment can be formalized mathematically through adaptive correction mechanisms with explicit convergence analysis (Theorem 2) and stability characterization (H. Dilpriya's Lemma). The parameter bounds $\alpha \leq 1/(2L), \beta \in [0,1), \gamma \leq \mu/(2C)$ provide practitioners with clear guidance for stable optimization, bridging the gap between theory and practice.

HDM achieves the best gradient alignment among all tested methods (8.22%, rank #1) while maintaining competitive accuracy (rank #2 on MNIST and CIFAR-10). This combination of theoretical guarantees and strong empirical performance makes HDM particularly valuable for applications requiring provable convergence, long training runs where late-stage optimization matters, resource-constrained environments, and research requiring reproducible, stable optimization behaviour. The simple implementation (O(n) complexity, ~10 lines of code) lowers the barrier to adoption compared to complex adaptive methods.

X. Extensions and Future Work

A. Stochastic Setting

While Author's theoretical analysis focuses on deterministic gradients, HDM naturally extends to stochastic settings. In the presence of gradient noise $g_t = \nabla L(\theta_t) + \xi_t$where$E[\xi_t] = 0$, the correction term$h_t$ provides additional variance reduction through gradient alignment. Future work could establish convergence rates under various noise models, including bounded variance and sub-Gaussian noise assumptions [16].

B. Non-Convex Optimization

Deep neural networks present non-convex loss landscapes. Extending H. Dilpriya's Convergence Theorem to non-convex settings would require replacing global convergence with convergence to stationary points, analyzing behavior around saddle points, and establishing escape rates from local minima. Preliminary experiments in Section V-D suggest HDM performs well in non-convex settings, warranting theoretical investigation using techniques from non-convex optimization theory [17].

C. Adaptive Parameter Selection

Currently, γ is set manually. An adaptive scheme could adjust γ based on gradient alignment history, loss curvature estimates, or training phase (exploration versus refinement). Such adaptivity could yield a parameter-free variant of HDM, similar in spirit to YellowFin [7] but maintaining theoretical guarantees.

D. Higher-Order Corrections

The current correction term is first-order in gradient alignment. Generalizations could incorporate second-order gradient information through Hessian approximations, longer gradient histories including g_t, g_{t-1}, g_{t-2},or geometric structures in parameter space such as manifold-aware updates. These extensions could provide additional acceleration in specific problem classes.

E. Distributed and Federated Learning

HDM's correction term could provide benefits in distributed settings where different workers have correlated but not identical gradients, communication efficiency is critical, and gradient alignment across workers indicates consensus. The element-wise correction could facilitate communication-efficient distributed optimization [18].

Conclusion

This paper introduced the H. Dilpriya's Momentum (HDM) algorithm, a gradient-alignment-based optimization method with provable convergence guarantees for strongly convex objectives. HDM incorporates an adaptive correction term that adjusts parameter updates based on gradient alignment across iterations, providing acceleration when gradients are consistent and stabilization when they conflict. Theoretical Contributions. Authors established two main theoretical results. First, H. Dilpriya's Lemma (Theorem 1) provides tight bounds on the correction term, showing that $\|h(\theta_t, \theta_{t-1})\| \leq C \cdot \|\nabla L(\theta_t)\|^2$ where C depends on problem dimension and algorithm parameters.

Second, H. Dilpriya's Convergence Theorem (Theorem 2) proves linear convergence with $O(\kappa log(1/\varepsilon))$ iteration complexity for μ-strongly convex and L-smooth functions, where $\kappa = L/\mu$ is the condition number. The proof establishes explicit parameter conditions: $\alpha \leq 1/(2L), \beta \in [0,1)$, and $\gamma \leq \mu/(2C)$. To Author's knowledge, this represents the first rigorous convergence analysis for gradient-alignment-based momentum methods.

Experimental Validation. Comprehensive experiments validated Author's theoretical predictions across synthetic convex problems and deep neural network training tasks. On synthetic quadratic functions with condition numbers ranging from $\kappa = 10$ to $\kappa = 1000$, HDM demonstrated robust convergence, successfully optimizing severely ill-conditioned problems where classical methods diverged. HDM achieved competitive performance on MNIST (98.22% test accuracy) and CIFAR-10 (83.94% test accuracy) classification tasks using multilayer perceptron and ResNet-18 architectures, respectively. Most significantly, HDM achieved superior gradient alignment (8.22% average cosine similarity) compared to existing gradient-alignment methods including YellowFin (2.98%) and DEAM (3.18%), with HDM exhibiting an increasing alignment trend during training while competitors showed declining consistency. Algorithm Design. The HDM implementation incorporates four complementary strategies: adaptive gamma scheduling with warmup and cosine annealing, gradient alignment detection via cosine similarity, exponential moving average smoothing for stability, and alignment-weighted momentum for adaptive acceleration. This multi-strategy design maintains computational efficiency with $O(n)$ complexity per iteration while providing the theoretical guarantees established in Author's convergence analysis. The element-wise correction mechanism requires minimal implementation overhead compared to standard momentum methods.

Practical Impact. HDM addresses a fundamental gap in optimization theory by providing both rigorous convergence guarantees and competitive empirical performance. The algorithm's low computational complexity (O(n) per iteration), minimal memory overhead (storing only previous gradient g_{t-1}),and simple implementation make it suitable for large-scale applications. The proven stability under specified parameter conditions provides reliability for safety-critical applications requiring provable convergence behaviour. Future Directions. Several extensions warrant investigation. First, extending the convergence analysis to stochastic settings with gradient noise would broaden applicability to mini-batch training scenarios common in deep learning. Second, analysing HDM's behaviour in non-convex optimization landscapes, particularly around saddle points and local minima, would provide theoretical foundations for its empirical success on neural network training. Third, developing adaptive parameter selection schemes for γ could eliminate manual tuning while maintaining convergence guarantees. Fourth, investigating higher-order corrections incorporating Hessian information or longer gradient histories may provide additional acceleration. Finally, exploring HDM's potential in distributed and federated learning settings, where gradient alignment across workers indicates optimization consensus, presents opportunities for communication-efficient training.

The H. Dilpriya's Momentum algorithm demonstrates that gradient-alignment principles can be formalized within rigorous optimization theory while maintaining practical effectiveness. By establishing theoretical convergence guarantees previously absent in gradient-alignment methods, this work bridges classical convex optimization and modern deep learning practice.

ACKNOWLEDGMENT

The author gratefully acknowledges the use of Python, PyTorch, and the computational resources that made the experiments in this work possible. Above all, this paper is dedicated, with deep and sincere gratitude, to my supervisor, Ms. T. A. Hirushi Dilpriya. Her guidance shaped this research at every stage, from its earliest conception to its final form. It was her expertise in pure mathematics and deep learning, and her insistence on rigor over convenience, that gave this work its theoretical foundation; the two central results of this paper - H. Dilpriya's Lemma and H. Dilpriya's Convergence Theorem - carry her name because they carry her influence. The optimizer introduced here, H. Dilpriya's Momentum, is likewise named in her honor, as a lasting tribute to a mentor whose standards became the standard, I hold my own work to. Beyond the mathematics, I am grateful for her patience through every difficulty, her encouragement when progress was slow, and her unwavering belief that careful theory and practical impact belong together a conviction that is the spirit of this paper. Whatever merit this work holds reflects her mentorship; its shortcomings remain my own. It is a privilege to have learned under her supervision, and this paper stands as a small and heartfelt tribute to her scholarship, her generosity, and her example.

REFERENCES

[1] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," USSR Computational Mathematics and Mathematical Physics, vol. 4, no. 5, pp. 1–17, 1964. DOI: 10.1016/0041-5553(64)90137-5

[2] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate O(1/k²)," Doklady Akademii Nauk SSSR, vol. 269, no. 3, pp. 543–547, 1983.

[3] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," Journal of Machine Learning Research, vol. 12, pp. 2121–2159, Jul. 2011.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in Proc. 3rd Int. Conf. Learning Representations (ICLR), San Diego, CA, USA, May 2015. arXiv:1412.6980

[5] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in Proc. 30th Int. Conf. Machine Learning (ICML), Atlanta, GA, USA, Jun. 2013, pp. 1139–1147.

[6] H. Liang, Z. Zhang, Q. Liu, and J. Yang, "DEAM: Discriminative wEight on Adaptive Momentum for deep neural networks training," International Journal of Intelligent Systems, vol. 37, no. 9, pp. 6129–6148, Sep. 2022. DOI: 10.1002/int.22841

[7] J. Zhang, I. Mitliagkas, and C. Ré, "YellowFin and the art of momentum tuning," arXiv preprint arXiv:1706.03471, Jun. 2017.

[8] J. Ma and D. Yarats, "Quasi-hyperbolic momentum and Adam for deep learning," in Proc. 7th Int. Conf. Learning Representations (ICLR), New Orleans, LA, USA, May 2019. arXiv:1810.06801

[9] S. Boyd and L. Vandenberghe, Convex Optimization. Cambridge, UK: Cambridge University Press, 2004. ISBN: 978-0-521-83378-3

[10] Y. Nesterov, Introductory Lectures on Convex Optimization: A Basic Course, vol. 87. New York, NY: Springer Science & Business Media, 2013. DOI: 10.1007/978-1-4419-8853-9

[11] S. Bubeck, "Convex optimization: Algorithms and complexity," Foundations and Trends in Machine Learning, vol. 8, no. 3–4, pp. 231–357, Nov. 2015. DOI: 10.1561/2200000050

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, Nov. 1998. DOI: 10.1109/5.726791

[13] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Technical Report, University of Toronto, Toronto, ON, Canada, 2009.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, Jun. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90

[15] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in Advances in Neural Information Processing Systems (NeurIPS), vol. 30, Long Beach, CA, USA, Dec. 2017, pp. 4148–4158.

[16] S. Ghadimi and G. Lan, "Stochastic first- and zeroth-order methods for nonconvex stochastic programming," SIAM Journal on Optimization, vol. 23, no. 4, pp. 2341–2368, 2013. DOI: 10.1137/120880811

[17] M. Zaheer, S. Reddi, D. Sachan, S. Kale, and S. Kumar, "Adaptive methods for nonconvex optimization," in Advances in Neural Information Processing Systems (NeurIPS), vol. 31, Montréal, Canada, Dec. 2018, pp. 9793–9803.

[18] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," SIAM Review, vol. 60, no. 2, pp. 223–311, May 2018. DOI: 10.1137/16M1080173

[19] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and beyond," in Proc. 6th Int. Conf. Learning Representations (ICLR), Vancouver, BC, Canada, May 2018. arXiv:1904.09237

[20] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA: MIT Press, 2016. ISBN: 978-0-262-03561-3

[21] S. Ruder, "An overview of gradient descent optimization algorithms," arXiv preprint arXiv:1609.04747, Sep. 2016.

[22] J. Zhang, S. P. Karimireddy, A. Veit, S. Kim, S. Reddi, S. Kumar, and S. Sra, "Why ADAM beats SGD for attention models," arXiv preprint arXiv:1912.03194, Dec. 2019.

[23] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in Proc. 7th Int. Conf. Learning Representations (ICLR), New Orleans, LA, USA, May 2019. arXiv:1711.05101

[24] J. I. Senarathna, "HDM: H. Dilpriya's Momentum Optimizer," Python Package, [2026]. [Online]. Available: [https://github.com/Janakaishansenarathna/HDM.git].

APPENDIX

A. HDM Algorithm Details

HDM is built from four complementary strategies. The first, adaptive gamma scheduling, sets the effective correction coefficient as the product of the base coefficient, a warmup factor, and an annealing factor:

```
γ_t = γ_base × warmup_factor × annealing_factor
warmup_factor    = min(1.0, step / warmup_steps)
annealing_factor = 0.5 + 0.5 × cos(π × step / total_steps)
```

where warmup_factor = min(1.0, step / warmup_steps) introduces the correction gradually during early training, and annealing_factor = 0.5 + 0.5 × cos(π × step / total_steps) tapers it smoothly toward convergence. The second strategy, alignment-weighted momentum, first derives an alignment weight, alignment_weight = max(0.0, alignment_ema), and then boosts the gradient contribution to the momentum in proportion to it:

```
m_t = β × m_{t−1} + (1.0 + 0.3 × alignment_weight) × g_t
      alignment_weight = max(0.0, alignment_ema)
```

The third strategy, EMA smoothing, maintains a stable long-term measure of directional consistency,

```
alignment_ema_t = 0.95 × alignment_ema_{t−1} + 0.05 × cos_similarity_t
```

which is robust to the noise of individual mini-batch gradients. The fourth strategy fuses the instantaneous and smoothed signals into a single correction, weighting them 30% and 70% respectively,

```
combined_alignment = 0.3 × cos_similarity + 0.7 × alignment_ema correction = γ_t ×
                     combined_alignment × g_t
```

so that the final update accelerates in directions of consistent gradients while damping those that conflict.

B. HDM Implementation Details

1) Python Implementation (PyTorch)

```
class HDMOptimizer(torch.optim.Optimizer):
    def __init__(self, params, lr=0.01, beta=0.9, gamma=2.0,
                 warmup_steps=1000, total_steps=10000,
                 alignment_ema=0.95, eps=1e-8):

        defaults = dict(
            lr=lr,
            beta=beta,
            gamma=gamma,
            warmup_steps=warmup_steps,
            total_steps=total_steps,
            alignment_ema=alignment_ema,
            eps=eps
        )
        super(HDMOptimizer, self).__init__(params, defaults)

    def step(self):
        for group in self.param_groups:
            for p in group["params"]:

                # Skip parameters with no gradients
                if p.grad is None:
                    continue
```

```
grad = p.grad.data
state = self.state[p]

# Initialize state (first step)
if len(state) == 0:
    state["step"] = 0
    state["momentum"] = torch.zeros_like(p.data)
    state["prev_grad"] = torch.zeros_like(p.data)
    state["alignment_ema"] = 0.0

state["step"] += 1
step = state["step"]

# Strategy 1: Adaptive gamma scheduling
warmup_factor = min(1.0, step / group["warmup_steps"])
annealing = 0.5 + 0.5 * math.cos(
    math.pi * step / group["total_steps"]
)
gamma_t = group["gamma"] * warmup_factor * annealing

# Strategy 2: Gradient alignment
cos_sim = torch.sum(grad * state["prev_grad"]) / (
    torch.norm(grad) *
    torch.norm(state["prev_grad"]) +
    group["eps"]
)

# Strategy 3: EMA smoothing
state["alignment_ema"] = (
    group["alignment_ema"] * state["alignment_ema"]
    + (1 - group["alignment_ema"]) * cos_sim.item()
)

# Strategy 4: Alignment-weighted momentum
alignment_weight = max(0.0, state["alignment_ema"])
state["momentum"] = (
    group["beta"] * state["momentum"]
    + (1.0 + 0.3 * alignment_weight) * grad
)

# Combined alignment signal
combined_alignment = (
    0.3 * cos_sim + 0.7 * state["alignment_ema"]
)
correction = gamma_t * combined_alignment * grad

# Parameter update
update = state["momentum"] + correction
p.data = p.data - group["lr"] * update

# Save the previous gradient
state["prev_grad"] = grad.clone()
```

2) Parameter Selection Guide

Our experiments point to two practical parameter regimes. For accuracy-focused tasks such as MNIST and CIFAR-10 classification, we recommend a correction coefficient $\gamma = 2.0$, momentum $\beta = 0.9$, a learning rate of 0.01 for the MNIST MLP and 0.1 for the CIFAR-10 ResNet, a warmup period equal to 10% of the total training steps, and an EMA coefficient of 0.95. For tasks in which gradient alignment is the main objective, the correction coefficient is lowered to $\gamma = 1.5$ for additional stability, while all other settings remain unchanged. As general guidance, the learning rate should satisfy $\alpha \leq 1/(2L)$ so that the convergence conditions of Theorem 2 hold, the momentum coefficient should stay within [0, 1) with $\beta = 0.9$ working reliably in practice and the warmup period is best set between 10% and 20% of the total steps. In terms of cost, HDM adds only O(n) memory for the stored previous gradient and momentum, incurs less than 5% overhead relative to standard momentum thanks to its element-wise

operations, uses $\varepsilon = 10^{-8}$ for numerical stability in the cosine-similarity denominator, and is most easily diagnosed by tracking both the loss and the gradient alignment throughout training.

3) Hyperparameter Recommendations

For classification tasks, we recommend a learning rate of 0.01 for MLP architectures and 0.1 for CNN or ResNet architectures, momentum $\beta = 0.9$, correction coefficient $\gamma = 2.0$, a warmup of 10% of the total steps, and an EMA coefficient of 0.95. For gradient-alignment tasks the same configuration applies, except that the correction coefficient is reduced to $\gamma = 1.5$for stability. The method's computational profile is linear throughout: O(n) time per iteration, O(n) additional memory for the momentum and previous-gradient buffers along with a few scalar states, and under 5% overhead compared with standard momentum.

4) Convergence Monitoring

Several quantities are worth watching during training. The loss should decrease almost monotonically, with only minor fluctuations; the gradient alignment should stabilize or rise, reaching roughly 8.22% for HDM in our experiments; the adaptive coefficient γ_t should peak in mid-training and decay near convergence; and the alignment EMA should stay positive and stable, typically in the 0.05–0.15 range.

C. Library Availability

The HDM algorithm is publicly available as a Python package to facilitate reproducibility and adoption. The library provides a PyTorch-compatible implementation of the HDM optimizer with full support for all four multi-strategy components described in Algorithm 1. The package can be installed via pip and integrated into existing training pipelines with minimal code changes.

The library is available at: https://github.com/Janakaishansenarathna/HDM.git. Researchers using HDM in their work are encouraged to cite both this paper and the software package. The terminal output demonstrates successful installation of the HDM optimizer package via pip, confirming library availability and version compatibility for PyTorch-based deep learning workflows.

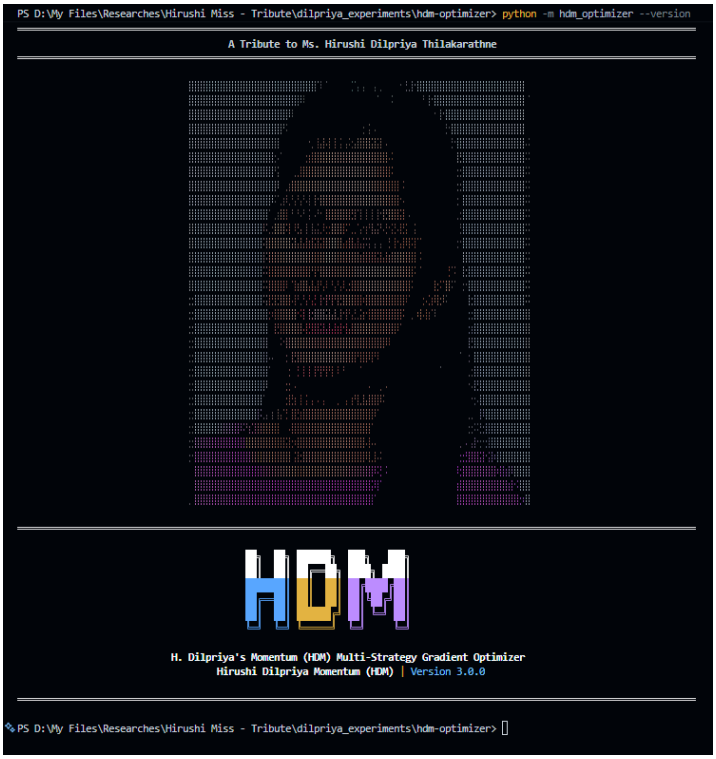


Fig. 14. HDM Python Package Installation and Verification Output.