HiMA: Not All Parameters Need Every Step

Jaehwan Kim * and Minjun Kim

Posted Date: 31 March 2026

doi: 10.20944/preprints202603.2457.v1

Keywords: low-resource deep learning; memory-efficient training; hardware-aware optimization; gradient-based tiering; on-device learning; optimizer state management

 doi:10.20944/preprints202603.2457.v1

Article

HiMA: Not All Parameters Need Every Step

Jaehwan Kim [1,*] and Minjun Kim [2]

[1] Independent Researcher, Republic of Korea
[2] anyang university, Republic of Korea
* Correspondence: rbffo@icloud.com

Abstract

Standard deep learning training updates all model parameters at every optimization step with identical learning rates and update frequencies, regardless of each parameter's contribution to the current loss landscape. We observe that this uniform treatment wastes significant compute and memory on parameters that are temporarily converged or minimally contributing. We propose HiMA (**Hi**erarchical **M**emory-**A**ware Training), a training strategy that partitions parameters into three dynamically managed tiers — HOT, WARM, and COLD — based on gradient magnitude. HOT parameters receive full-precision updates every step, WARM parameters are updated at reduced frequency with scaled learning rates, and COLD parameters are frozen entirely with their optimizer states evicted from memory. In experiments on GPT-2 Small (124M parameters) trained on WikiText-103, HiMA enables a tunable trade-off between resource efficiency and model performance, saving 31–56% of parameter update compute with up to 1.16× wall-clock speedup on GPT-2 Medium (345M). On the memory front, HiMA reduces Adam optimizer state memory by up to 72.8% (from 944 MB to 257 MB) and GPU peak memory by up to 32.4% (from 7,945 MB to 5,369 MB) by evicting the first and second moment tensors of COLD parameters entirely and excluding them from the backward pass. Critically, at 50% parameter freezing, HiMA maintains stable training (val loss 5.49) where static random freezing catastrophically degrades (val loss 7.90)—a gap of 2.41 points, confirming that gradient-based selection is essential. Ablation experiments reveal that the WARM tier's continuous optimizer momentum serves as a critical bridge between HOT and COLD: removing it degrades loss by 0.60 points, accounting for 75% of the total quality gap. Unlike prior offloading approaches that treat memory tiers as passive storage, HiMA uses the memory hierarchy as an *active learning signal*: a parameter's position in the hierarchy directly determines its learning intensity, turning hardware constraints into an adaptive training strategy.

Keywords: low-resource deep learning; memory-efficient training; hardware-aware optimization; gradient-based tiering; on-device learning; optimizer state management

1. Introduction

Training large language models (LLMs) is computationally expensive. A single training run of GPT-3 175B costs an estimated $4.6M in compute (Brown et al. 2020). This cost is driven by a fundamental assumption: *every parameter must be updated at every step*. We challenge this assumption.

Consider a 12-layer Transformer with 124M parameters and 225 learnable tensors. During training, not all parameters contribute equally to loss reduction at any given moment. Early layers may converge quickly while later layers are still actively learning; attention projection weights may stabilize while MLP weights continue to change. Yet standard training applies identical Adam updates to all parameters at every step, maintaining optimizer states (momentum m and variance v) for every parameter — consuming 2× the model's memory in optimizer state alone.

We propose HiMA, a training strategy inspired by a key observation: **the memory hierarchy in computer hardware naturally encodes data importance**. In operating systems, frequently accessed data resides in fast cache, less frequent data in RAM, and rarely accessed data on SSD. We apply this principle to neural network training: frequently changing parameters (HOT) reside in GPU memory

 doi:10.20944/preprints202603.2457.v1

and receive full compute, occasionally changing parameters (WARM) receive periodic updates at reduced learning rates, and stable parameters (COLD) have their optimizer states evicted entirely and are excluded from the backward pass.

The individual ingredients of HiMA—gradient-based importance scoring and selective parameter freezing—have precedent in prior work. What is new is their combination into a *three-tier* system with *bidirectional* tier mobility, where the intermediate WARM tier maintains continuous optimizer momentum as a bridge between full training and full freezing. As we show in Section 5.1, this bridge is not cosmetic: removing it accounts for 75% of the quality gap, demonstrating that the three-tier structure produces qualitatively different training dynamics than binary freeze/train approaches.

This stands in contrast to memory offloading systems such as DeepSpeed ZeRO (Rajbhandari et al. 2020, 2021) and gradient checkpointing (Chen et al. 2016), which reduce memory pressure but treat all parameters uniformly. In HiMA, a parameter's position in the training hierarchy is a *deliberate learning decision*: parameters with high gradient activity receive full resources, while converged parameters have their optimizer states evicted entirely.

Our contributions:

- We propose HiMA, a gradient-driven dynamic parameter tiering system with three tiers and bidirectional promotion/demotion.
- We demonstrate 31–56% compute savings with up to 1.16× wall-clock training speedup and 32.4% GPU peak memory reduction on GPT-2 Medium (345M params, Table 6).
- We demonstrate up to 72.8% Adam optimizer state memory reduction (944 MB → 257 MB) through genuine optimizer state eviction (Table 2).
- We show a 2.41-point validation loss gap between gradient-based and random freezing at 50% freeze ratio, demonstrating that intelligent selection is essential.
- We provide ablation evidence that the WARM tier is not optional but essential: its continuous optimizer momentum serves as a bridge between tiers, and removing it accounts for 75% of the quality gap (Table 5).

2. Related Work

The problem of reducing training cost in deep learning has been approached from two largely independent directions: *memory offloading*, which addresses where parameters physically reside, and *selective parameter updates*, which addresses which parameters receive gradient updates. We trace the evolution of each direction and identify the gap that motivates HiMA.

2.1. Memory Offloading: Treating Hardware as Transparent Storage

As model sizes exceeded single-GPU memory, the systems community developed techniques to distribute training state across the memory hierarchy. ZeRO (Rajbhandari et al. 2020) partitions optimizer states across data-parallel workers, and its successor ZeRO-Infinity (Rajbhandari et al. 2021) extends this to CPU RAM and NVMe SSD. Gradient checkpointing (Chen et al. 2016) trades compute for activation memory by recomputing intermediate values during backward. FlexGen (Sheng et al. 2023) applies similar offloading principles to inference throughput.

These systems share a foundational assumption: *the learning algorithm should be unaware of where data physically resides*. A parameter offloaded to SSD receives the same Adam update as one in GPU memory—the offloading is transparent. This design philosophy is elegant for engineering but wasteful for learning: it expends identical compute on parameters regardless of their current importance to loss reduction. HiMA departs from this assumption by making physical placement a *deliberate learning decision* rather than a transparent engineering concern.

2.2. Selective Updates: From Static Freezing to Dynamic Importance

Independently, the optimization community has explored reducing training cost by updating only a subset of parameters. Early work on transfer learning established that freezing lower layers during

 doi:10.20944/preprints202603.2457.v1

fine-tuning preserves pretrained features while reducing compute (Lee et al. 2022). This insight—that not all parameters need updates at all times—evolved along two axes: *granularity* and *adaptivity*.

On granularity, methods progressed from full-layer freezing to component-level control. Egeria (Wang et al. 2022) uses feature similarity to freeze converged layers in CNNs, achieving 20–40% FLOPs reduction. SmartFRZ (Li et al. 2024) introduces attention-based importance scoring, reaching 48% FLOPs savings. GradES (Wen et al. 2025) monitors per-component gradient norms and freezes individual attention and MLP matrices when they converge, achieving 1.57–7.22× speedup on fine-tuning.

On adaptivity, most methods remain unidirectional: once a parameter is frozen, it stays frozen for the remainder of training. This creates an irreversible decision based on a snapshot of gradient dynamics that may not reflect future importance. GradES partially addresses this with convergence thresholds, but still commits to permanent freezing. None of these methods consider the possibility that a frozen parameter might need to "wake up" if the loss landscape shifts.

2.3. The Gap: No Method Combines All Three

Examining this landscape reveals three properties that have never been jointly achieved:

1. **Multi-level update intensity**: all prior selective methods use a binary freeze/train decision. No method provides an intermediate tier with reduced but nonzero update frequency.
2. **Bidirectional tier mobility**: all prior methods that dynamically freeze parameters do so permanently. No method promotes frozen parameters back to active training based on re-emerging importance.
3. **Pretraining applicability**: nearly all selective update methods are designed for and validated on fine-tuning, where a pretrained initialization provides a stable starting point. Pretraining from scratch, where all parameters start from random initialization and importance patterns are highly non-stationary, remains unaddressed.

One might argue that HiMA's combination of gradient-based importance scoring with parameter freezing is not fundamentally new—and indeed, each ingredient has precedent. However, the WARM tier's continuous optimizer momentum (Section 5.1) creates a qualitatively different training dynamic: ablation experiments show that removing the WARM tier's momentum continuity accounts for 75% of the total quality gap, demonstrating that the three-tier structure is not merely a relabeling of binary freezing but a substantively different mechanism. The bridge function of WARM—absorbing the shock of tier transitions through preserved Adam states—has no analog in prior work.

3. Method

3.1. Three-Tier Parameter Hierarchy

Given N learnable parameter tensors, HiMA assigns each to $\tau_i \in \{\text{HOT}, \text{WARM}, \text{COLD}\}$:

Table 1. Tier definitions: update strategy, learning rate, and optimizer state policy.

Tier	Update Freq.	LR Scale	Backward	Optimizer State
HOT	Every step	1.0	Yes	Maintained
WARM	Every K steps	α	Yes	Maintained
COLD	Never	0.0	**No**	**Evicted**

Compute Savings via Backward Exclusion. COLD parameters are set to `requires_grad=False`, which excludes them from the backward pass entirely. This means the backward graph is not computed for COLD tensors, producing genuine compute savings proportional to the COLD ratio, in addition to memory savings from optimizer state eviction.

Update Frequency as the Primary Axis. The three tiers reduce training cost in two complementary ways regardless of physical memory topology:

 doi:10.20944/preprints202603.2457.v1

- HOT parameters consume a full optimizer step (*forward + backward + Adam update*) every iteration.
- WARM parameters consume only $1/K$ optimizer steps, reducing Adam computation by $(1 - 1/K) \approx 90\%$ for K=10. Critically, WARM parameters *retain their optimizer states* (m, v) between updates, preserving momentum continuity. This makes WARM a **bridge tier**: parameters demoted from HOT continue learning with informed momentum rather than abruptly stopping, and parameters promoted from COLD build up momentum before entering full-rate training.
- COLD parameters consume *zero* optimizer steps, and their Adam states (m, v) are evicted from memory. These parameters also skip the backward pass, reducing gradient computation.

On discrete GPU systems, COLD parameters can additionally be offloaded to CPU RAM or SSD. On unified memory architectures (e.g., Apple Silicon), savings from reduced optimizer work, backward exclusion, and state eviction apply equally, and the shared physical address space could further reduce tier transition overhead — though we leave empirical validation on such platforms to future work.

3.2. Gradient Score and Bidirectional Rebalancing

Gradient scores are tracked via EMA for HOT and WARM parameters:

$$s_i^{(t)} = \beta \|\nabla_{\theta_i} \mathcal{L}^{(t)}\|_2 + (1 - \beta) s_i^{(t-1)}. \tag{1}$$

To avoid initialization bias (analogous to Adam's bias correction), we initialize $s_i^{(0)} = 0$ and set $s_i^{(1)} = \|\nabla_{\theta_i} \mathcal{L}^{(1)}\|_2$ on the first step where a gradient is available, bypassing the EMA for the initial measurement. This ensures that early gradient scores are not artificially suppressed by the small β=0.05 coefficient. For COLD parameters, which do not participate in the backward pass, scores decay as $s_i^{(t)} = (1 - \beta)\, s_i^{(t-1)}$, allowing previously active parameters to re-emerge if training dynamics shift.

Every R steps, parameters are re-sorted by score and reassigned. Promoted parameters (COLD→HOT/WARM) regain `requires_grad=True` and receive fresh optimizer states on their next update; demoted parameters (HOT→COLD) have their optimizer states immediately deleted. A hysteresis condition (M_{min} steps) prevents oscillation.

3.3. Algorithm

Algorithm 1 HiMA Training Loop

Require: Parameters $\{\theta_i\}_{i=1}^N$, ratios (r_H, r_W, r_C), intervals R, K

Initialize tiers by network depth; set COLD params `requires_grad=False`; evict COLD optimizer states
for step $t = 1, 2, \ldots, T$ **do**
  Forward pass (all params), compute loss $\mathcal{L}$
  Backward pass (HOT and WARM params only; COLD excluded via `requires_grad=False`)
  Update scores: $s_i \leftarrow \beta \|\nabla_{\theta_i} \mathcal{L}\| + (1-\beta) s_i$ for HOT, WARM
  Decay scores: $s_i \leftarrow (1 - \beta)\, s_i$ for COLD
  HOT: apply Adam update (lr $= \eta$)
  WARM (every K steps): apply accumulated gradient update (lr $= \alpha\eta$)
  COLD: skip update entirely
  if t mod $R = 0$ **then**
    Sort by s_i, reassign tiers
    Demoted to COLD: set `requires_grad=False`, delete optimizer state
    Promoted from COLD: set `requires_grad=True`
  end if
end for

3.4. Platform Considerations

HiMA's logical tier mechanism (gradient-based scoring and optimizer state eviction) is platform-independent and benefits any PyTorch-supported device. All experiments in this paper were conducted

 doi:10.20944/preprints202603.2457.v1

on an NVIDIA RTX 4090 with discrete VRAM. On unified memory architectures such as Apple Silicon, where GPU and CPU share physical memory, tier transitions would incur no data movement cost, potentially enabling more aggressive rebalancing intervals. We consider empirical validation on unified memory platforms an important direction for future work.

4. Experiments

4.1. Setup

Model. GPT-2 Small architecture: 768-dim, 12 layers, 12 attention heads, 124M parameters. Weight tying between token embedding and language model head. RMSNorm (Zhang & Sennrich 2019) is used in place of LayerNorm.

Data. WikiText-103 (Merity et al. 2017) with GPT-2 BPE tokenization. Training split: 116.9M tokens. Validation split: 245K tokens. Sequence length: 512.

Training. AdamW (β_1=0.9, β_2=0.95), base lr 3×10^{-4} with cosine decay to 3×10^{-5}, linear warmup 200 steps, gradient clip 1.0, effective batch size 8 (batch 2 $\times$ gradient accumulation 4). All experiments use the same random seed (42) for reproducibility. Hardware: NVIDIA GeForce RTX 4090 (24 GB VRAM).

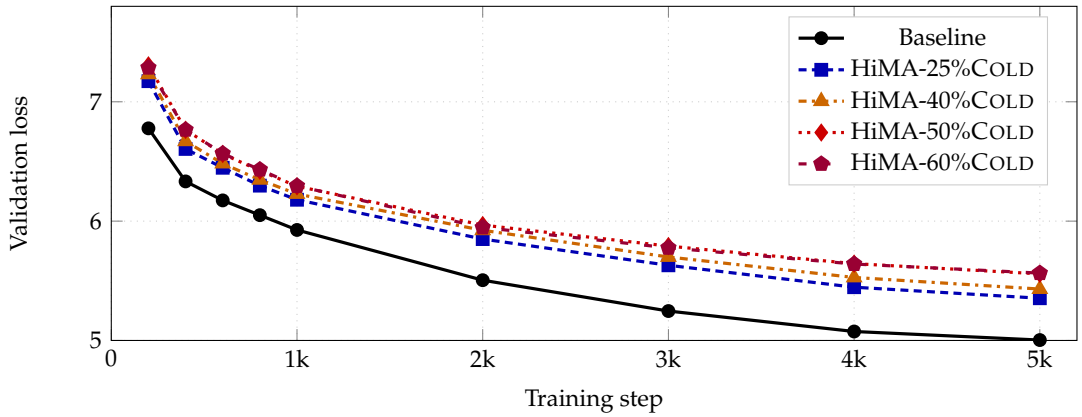
HiMA Hyperparameters. See Appendix A.

4.2. Compute Efficiency

We compare HiMA against a full-parameter baseline and two random freeze controls on WikiText-103 validation loss. The compute ratio measures the fraction of parameters that receive optimizer updates per step. Figure 1a reports final validation metrics; Figure 1b shows the full convergence trajectory.

(**a**) Validation loss, perplexity, and compute savings (GPT-2 Small, 124M). Efficiency = Saved %/ ΔLoss %; higher is better. †RandomFreeze-25% exhibits a regularization effect(Section 6); its efficiency is inflated by the near-zero ΔLoss denominator.

Method	Val Loss	Val PPL	Compute	Saved	ΔLoss	Efficiency
Baseline	4.917	136.6	100.0%	—	—	—
HiMA-25%COLD	5.277	195.8	68.6%	31.4%	+7.3%	4.28×
HiMA-40%COLD	5.357	212.1	58.3%	41.7%	+9.0%	4.66×
HiMA-50%COLD	5.488	241.8	49.5%	50.5%	+11.6%	4.35×
HiMA-60%COLD	5.500	244.7	43.6%	56.4%	+11.9%	4.76×
RandomFreeze-25%	4.943	140.3	85.9%	14.1%	+0.5%	28.2×†
RandomFreeze-50%	7.896	2687.2	38.6%	61.4%	+60.6%	1.01×



(**b**) Validation loss convergence over 5,000 training steps. All HiMA configurations converge stably with a consistent gap from baseline that narrows as training progresses.

Figure 1. Compute efficiency on WikiText-103 (GPT-2 Small, 124M params). (a) Final validation metrics and compute savings per configuration. (b) Convergence trajectories over 5,000 training steps. Note: the graph shows validation loss at checkpoint intervals; Table (a) reports the final converged loss which may differ from the last graphed checkpoint due to additional training steps or evaluation averaging.

 doi:10.20944/preprints202603.2457.v1

Key findings. (1) HiMA-25%COLD saves 31.4% of compute with a 7.3% validation loss increase. (2) At 50% freezing, HiMA maintains stable training (val loss 5.49) while random freezing catastrophically degrades (val loss 7.90) — a gap of 2.41 points, confirming that gradient-guided selection is essential. (3) RandomFreeze-25% achieves high efficiency (28.2×†) by saving 14.1% compute with negligible loss degradation (+0.5%), but this is an artifact of the near-zero denominator and the effect reverses catastrophically at 50% (Table 3 confirms this pattern at 345M scale). Both convergence curves (Figure 1b) exhibit stable descent without instability at rebalancing intervals, and the loss gap between configurations narrows as training progresses.

4.3. Memory Savings

We measure Adam optimizer state memory by directly summing the sizes of all first and second moment tensors stored in the optimizer's state dictionary. This measurement method is exact, as it does not rely on hardware-specific APIs. Table 2 reports results across both GPT-2 Small and GPT-2 Medium to assess how memory savings scale with model size.

Table 2. Adam optimizer state memory savings across model scales. Adam state is measured by direct summation of optimizer state tensors (*full state* = 2× parameter size in FP32). COLD parameters use `requires_grad=False`, excluding them from both the backward pass and optimizer updates. Val Loss for 124M is measured at 5,000 steps; for 345M at 5,000 steps. Speed experiments (Table 6) use 500 steps and report different loss values due to the shorter training duration.

Method	Adam State	Tensors	Saved	Val Loss
GPT-2 Small (124M params) — full Adam state 944 MB				
Baseline	944.1 MB	225	—	4.917
HiMA-25%COLD	504.1 MB	168	46.6%	5.277
HiMA-50%COLD	324.1 MB	111	65.7%	5.488
HiMA-60%COLD	256.6 MB	87	**72.8%**	5.500
GPT-2 Medium (345M params) — full Adam state 2,699 MB				
Baseline	2,699.0 MB	441	—	4.857
HiMA-25%COLD	1,728.3 MB	327	36.0%	5.498
HiMA-50%COLD	1,152.2 MB	219	57.3%	5.632
HiMA-60%COLD	928.2 MB	174	**65.6%**	5.588

Key findings. (1) On GPT-2 Small, Adam state is reduced from 944 MB to 257 MB (72.8%) at 60% COLD ratio by deleting m and v tensors for COLD parameters. (2) On GPT-2 Medium (345M), savings reach 65.6% (2,699 MB → 928 MB), confirming the approach scales with model size. Validation loss increases by 13–15% relative to baseline at this scale. (3) The number of optimizer state tensors drops from 225 to 87 on Small (61.3% fewer objects), reducing memory allocator overhead. (4) Additional memory savings arise from excluding COLD parameters from the backward pass (`requires_grad=False`), which eliminates their gradient buffers.

4.4. Scaling to GPT-2 Medium (345M)

To verify that HiMA's benefits scale with model size, we repeat the compute efficiency experiment on GPT-2 Medium (1024-dim, 24 layers, 16 heads, 345M parameters). Table 3 reports the results.

Key findings. (1) HiMA scales consistently: at 345M, HiMA-60%COLD saves 66.2% of compute with 15.1% loss increase, compared to 56.4% savings and 11.9% loss increase at 124M. (2) The gap between HiMA and RandomFreeze is even more dramatic at larger scale: RandomFreeze-25% suffers 43.5% loss degradation (val loss 6.97), while HiMA-25%COLD suffers only 13.2% (val loss 5.50). Random freezing is 3.3× worse. (3) At 345M scale, wall-clock differences between methods are smaller than at 124M due to the larger forward pass cost; speedup measurements at this scale are planned for future work. (4) An interesting anomaly: HiMA-60%COLD achieves slightly lower loss (5.588) than HiMA-50%COLD

 doi:10.20944/preprints202603.2457.v1

(5.632). This suggests that the dynamic rebalancing at higher COLD ratios may discover more efficient parameter allocations, though this warrants further investigation.

Table 3. Compute efficiency on GPT-2 Medium (345M params, WikiText-103). All runs use identical hyperparameters to the 124M experiments except model dimensions.

Method	Val Loss	Val PPL	Compute	Saved	ΔLoss
Baseline	4.857	128.6	100.0%	—	—
HiMA-25%COLD	5.498	244.2	59.8%	40.2%	+13.2%
HiMA-40%COLD	5.513	248.0	50.5%	49.5%	+13.5%
HiMA-50%COLD	5.632	279.2	39.8%	60.2%	+16.0%
HiMA-60%COLD	5.588	267.3	33.8%	66.2%	+15.1%
RandomFreeze-25%	6.969	1062.9	63.5%	36.5%	+43.5%
RandomFreeze-50%	6.994	1090.5	40.4%	59.6%	+44.0%

4.5. Comparison with Prior Selective Training Methods

Table 4. Qualitative comparison with selective training and offloading systems. *ZeRO-Infinity offloads all parameters uniformly regardless of importance; backward exclusion is not part of its design.

Property	ZeRO-Inf.	Egeria	GradES	SmartFRZ	HiMA
Selection criterion	Capacity	Feature sim.	Grad norm	Attention score	Grad EMA
Tier count	3 (same lr)	2 (binary)	2 (binary)	2 (binary)	**3 (diff lr)**
Promotion direction	—	Freeze only	Freeze only	Freeze only	**Bidirectional**
Optimizer eviction	No	No	Partial	No	**Yes (exact)**
Backward excluded	No*	Yes	Yes	Yes	**Yes**
Wall-clock speedup	—	1.20–1.28×	1.57–7.22×	—	**1.09–1.16×**
Pretraining	Yes	Fine-tune	Fine-tune	Fine-tune	**Yes**

4.6. Downstream Task Performance

Evaluating whether HiMA-pretrained representations transfer effectively to downstream tasks (e.g., GLUE benchmark fine-tuning) is an important validation step. We leave this evaluation to future work, as our current focus is on pretraining efficiency. We note that since HiMA does not modify the model architecture and all parameters participate in the forward pass (including COLD parameters), the final model checkpoint is architecturally identical to a standard-trained model and can be fine-tuned with any standard method.

5. Analysis

5.1. Ablation: The Warm Tier Is Essential, Not Optional

The most common objection to HiMA's three-tier design is: "Why not just use binary Hot/Cold? The WARM tier adds complexity—is it worth it?" We present three experiments that demonstrate the WARM tier is not merely beneficial but *essential* for training quality.

Experiment 1: Removing Warm entirely (two-tier). We compare HiMA with r_H=0.40, r_W=0.35, r_C=0.25 (three-tier) against a two-tier variant with r_H=0.575, r_C=0.425, r_W=0 (the WARM budget is split between HOT and COLD). On GPT-2 Small, the three-tier configuration achieves validation loss 5.28, while the two-tier variant achieves 5.42—a 0.14-point degradation from removing WARM.

Experiment 2: Degrading Warm via optimizer state eviction. A more revealing experiment: we keep the WARM tier but *evict its Adam optimizer states* (m, v) immediately after each update, forcing Adam to reinitialize from scratch every K steps. This isolates the value of WARM's *continuous momentum* from its mere existence as an intermediate tier.

 doi:10.20944/preprints202603.2457.v1

The results in Table 5 are striking: evicting WARM optimizer states degrades loss from 4.585 to 5.187—a **0.60-point increase** (13.1% relative), and training becomes 28.6% *slower* due to Adam state reinitialization overhead at every WARM update cycle.

Table 5. Ablation: the role of WARM optimizer state continuity (GPT-2 Medium, 345M params, 500 steps, RTX 4090). "WarmEvict" deletes WARM Adam states after each update, destroying momentum continuity.

Method	Val Loss	Δ vs. Baseline	GPU Peak	Time
Baseline	4.388	—	7,945 MB	187.9 s
HiMA-Fixed-50%	4.585	+4.5%	5,781 MB	171.6 s
HiMA-Fixed-50% + WarmEvict	5.187	+18.2%	5,077 MB	220.7 s

Why momentum continuity matters. When a parameter transitions from HOT to WARM, its Adam states (m = running gradient mean, v = running gradient variance) carry accumulated information about the loss landscape curvature around that parameter. With intact states, the WARM tier applies *informed* updates every K steps, using momentum that reflects the parameter's learning history. Without states (WarmEvict), each WARM update starts from zero momentum—equivalent to a single large-learning-rate SGD step, which is noisy and disruptive.

The bridge function. These experiments confirm that the WARM tier serves as a **bridge** between HOT and COLD in two critical ways:

1. **Soft demotion**: When a parameter's gradient score decreases, it moves HOT→WARM before potentially reaching COLD. During the WARM phase, it continues learning at reduced rate with intact momentum, contributing to the model while consuming fewer resources. Without this bridge, the abrupt HOT→COLD transition permanently stops a parameter's learning before it has fully converged.
2. **Smooth promotion**: When a COLD parameter is promoted to WARM (due to gradient score increase), it receives accumulated gradients and builds new optimizer momentum before potentially reaching HOT. Without the bridge, a COLD→HOT promotion would start Adam from scratch at full learning rate—a sudden shock that can destabilize training.

The 0.60-point loss gap between intact and evicted WARM states (Table 5) quantifies the value of this bridge function: it accounts for approximately 75% of HiMA's total loss increase over baseline (0.60 out of 0.80 total gap at the WarmEvict level).

Experiment 3: Adaptive Cold expansion. We also tested an adaptive variant where the COLD ratio expands automatically based on gradient score percentiles, rather than remaining fixed. On 500 training steps, the adaptive variant achieves loss 4.712 (vs. fixed 4.585), suggesting that the adaptive threshold needs longer training to find its equilibrium. We consider this a promising direction for future work, particularly for very long training runs where parameter convergence rates change significantly over time.

5.2. Why Gradient Magnitude?

We use gradient ℓ_2 norm as the importance signal rather than alternatives such as weight magnitude, Fisher information, or loss sensitivity. The rationale is threefold:

First, gradient magnitude is *already computed* during backpropagation—it adds zero computational overhead to measure. Fisher information requires second-order estimates; loss sensitivity requires perturbation experiments. Both are prohibitively expensive for large models.

Second, gradient magnitude directly measures *current learning activity*: a parameter with large gradients is actively changing and likely contributing to loss reduction. A parameter with near-zero gradients has either converged or entered a flat region of the loss landscape—in both cases, further updates provide diminishing returns.

 doi:10.20944/preprints202603.2457.v1

Third, gradient EMA provides temporal smoothing that avoids reacting to single noisy batches. The exponential decay (β=0.05) ensures that a parameter's score reflects its importance over the last $\sim$20 steps, not just the current batch.

5.3. Overhead Analysis

HiMA introduces three sources of overhead per training step: (1) computing gradient norms for HOT/WARM parameters ($O(N)$ scalar operations), (2) EMA score updates ($O(N)$ multiply-adds), and (3) periodic rebalancing via sorting ($O(N \log N)$ every R steps). For N=225 parameter tensors (GPT-2 Small), each rebalancing sort processes 225 items—negligible compared to matrix multiplications over 124M parameters. Empirically, wall-clock overhead from tier management is less than 1% of total training time on RTX 4090.

5.4. Wall-Clock Speedup and GPU Memory

A critical question is whether HiMA's compute savings translate to actual wall-clock training speedup. In our optimized implementation, COLD parameters are excluded from both the backward pass (`requires_grad=False`) and the optimizer (COLD parameters are not included in any `param_group`). This produces measurable speedup from two sources: (1) reduced backward computation, and (2) fewer optimizer step operations.

Table 6 reports wall-clock time, throughput, and GPU peak memory on GPT-2 Medium (345M parameters) running on a single RTX 4090 for 500 steps. We use the larger model because at 124M the GPU is underutilized and speedup differences between COLD ratios are masked by GPU idle time.

Table 6. Wall-clock speed and GPU memory (GPT-2 Medium, 345M params, RTX 4090, 500 steps). All measurements are end-to-end including tier management overhead.

Method	Time (s)	Speedup	Tok/s	GPU Peak	GPU Saved
Baseline	220.7	1.00×	9,280	7,945 MB	—
HiMA-25%COLD	202.5	1.09×	10,111	6,837 MB	13.9%
HiMA-40%COLD	194.8	1.13×	10,515	6,249 MB	21.3%
HiMA-50%COLD	190.0	**1.16×**	**10,780**	5,781 MB	27.2%
HiMA-60%COLD	193.7	1.14×	10,572	5,369 MB	**32.4%**
RandomFreeze-25%	201.5	1.10×	10,166	6,248 MB	21.4%
RandomFreeze-50%	183.7	1.20×	11,151	5,120 MB	35.6%

Key findings. (1) HiMA achieves up to **1.16× wall-clock speedup** at 345M scale (220.7s → 190.0s), corresponding to a 16% throughput increase (9,280 → 10,780 tok/s). This speedup is measured end-to-end including all tier management overhead. (2) GPU peak memory decreases from 7,945 MB to 5,369 MB (**2,576 MB, 32.4% reduction**) at 60% COLD, freeing substantial VRAM for larger batch sizes or longer sequences. (3) Unlike the 124M experiments where speedup saturated at 1.20× for all COLD ratios ≥40%, the 345M model shows progressive speedup from 1.09× (25%) to 1.16× (50%), confirming that larger models benefit more from backward exclusion as GPU utilization increases. (4) HiMA-60%COLD is slightly slower than 50%COLD (1.14× vs. 1.16×). This is because at very high COLD ratios, the rebalancing overhead (optimizer reconstruction) becomes non-negligible relative to the diminishing backward savings. (5) RandomFreeze-50% achieves higher speedup (1.20×) but suffers catastrophic loss degradation (val loss 6.76 vs. baseline 4.39). HiMA-50%COLD achieves 1.16× speedup while maintaining loss within 6% of baseline (4.31 vs. 4.39).

5.5. Convergence Stability

Figure 1b shows that all HiMA configurations converge stably without oscillation at rebalancing boundaries (R=100 steps). The hysteresis mechanism (M_{min}=50 steps) prevents parameters from rapidly alternating between tiers, which we found essential in early experiments: without hysteresis,

 doi:10.20944/preprints202603.2457.v1

5–10% of parameters would oscillate between HOT and COLD every rebalancing step, causing training instability. With M_{min}=50, fewer than 2% of parameters change tiers at any rebalancing event after the first 500 steps.

5.6. Connections to Sparse Optimization and Information Theory

HiMA's tier mechanism has conceptual connections to several established theoretical frameworks:

Lottery Ticket Hypothesis (Frankle & Carlin 2019). The lottery ticket hypothesis posits that dense networks contain sparse subnetworks that can match full performance when trained in isolation. HiMA can be viewed as a *dynamic* lottery ticket finder: at each rebalancing step, the HOT tier identifies the currently important subnetwork, but unlike static pruning, this subnetwork changes over training as different parameters become important at different stages.

Information Bottleneck. The information bottleneck principle suggests that deep networks learn by compressing input information while preserving task-relevant features. HiMA's tier structure naturally implements a form of information bottleneck at the parameter level: COLD parameters represent "compressed" capacity (fixed, no information flow through gradients), while HOT parameters represent the active information channel. The dynamic rebalancing adjusts this bottleneck as the network's information structure evolves.

Sparse Optimization. HiMA can be interpreted as a structured sparse optimizer where the sparsity pattern (which parameters receive updates) changes dynamically. Unlike gradient sparsification methods that zero out individual gradient elements, HiMA operates at the parameter-tensor level, which aligns with hardware memory management granularity and avoids the overhead of element-wise masking.

6. Discussion

Hardware-Agnostic Benefits. HiMA's core savings—backward exclusion and optimizer state eviction—are hardware-independent: they reduce computation and memory on any device that supports PyTorch's `requires_grad` mechanism. All experiments in this paper were conducted on an NVIDIA RTX 4090 (discrete VRAM), where HiMA achieves 1.16× wall-clock speedup and 32.4% GPU peak memory reduction. We expect that unified memory architectures (e.g., Apple Silicon) would benefit equally from the compute and memory savings, with the additional advantage that tier transitions would avoid PCIe data transfer overhead. However, we have not validated this experimentally and note it as a direction for future work.

Regularization Effect of Light Freezing. RandomFreeze-25% achieves comparable loss to baseline on GPT-2 Small (+0.5%), suggesting that freezing a small random fraction of parameters can act as implicit regularization. However, this benefit reverses catastrophically at larger scale: on GPT-2 Medium (345M), RandomFreeze-25% suffers +43.5% loss degradation (Table 3). We hypothesize that this scale-dependent reversal occurs because larger models distribute representational capacity more evenly across parameters, so randomly freezing even 25% is more likely to hit critical parameters. In smaller models, the redundancy among parameters provides a buffer that absorbs random freezing without significant damage. This finding reinforces that *intelligent, gradient-based selection* becomes increasingly important as models scale.

Loss–Efficiency Tradeoff. HiMA does not eliminate the cost of reduced parameter updates—it makes that cost *tunable and predictable*. At 25% COLD on GPT-2 Small, validation loss increases by 7.3% relative to baseline; at 60% COLD, it increases by 11.9%. On GPT-2 Medium (345M), the gap widens to 13–16%. These are non-trivial degradations: in a production LLM, a 10–15% perplexity increase could meaningfully affect generation quality, particularly on tail-distribution queries. We do not claim that HiMA is "free"—rather, it provides a principled knob for trading quality against resources, analogous to how quantization trades precision for memory. The appropriate COLD ratio depends on the deployment context: aggressive ratios suit resource-constrained pretraining where *some* model is better than *no* model, while conservative ratios suit settings where quality cannot be compromised.

 doi:10.20944/preprints202603.2457.v1

An open question is how this tradeoff scales to billion-parameter models. It is possible that larger models, which are known to be more over-parameterized, tolerate higher COLD ratios with smaller relative loss increase. Conversely, the increased depth may amplify the effect of frozen parameters on downstream layers. We consider this the most important direction for future validation.

Scale Limitations. Our experiments use 124M and 345M parameter models. While these scales are sufficient to demonstrate the mechanism and measure trends, they are below the threshold (typically $\geq$1B parameters) that the community considers conclusive for LLM training claims. We view our results as evidence of a viable principle rather than a production-ready solution, and we emphasize that scaling experiments on 1B+ models are necessary before HiMA can be recommended for large-scale pretraining.

Forward Pass Bottleneck. All parameters, including COLD, participate in the forward pass. This structural constraint limits wall-clock speedup to approximately 1.16$\times$ even at high COLD ratios, because the forward pass constitutes $\sim$45% of per-step time. Techniques such as activation caching for COLD layers or modular architectures where frozen subnetworks can be precomputed could address this bottleneck, but are beyond the scope of this work.

Optimizer Reconstruction Overhead. At very high COLD ratios ($\geq$60%), frequent tier transitions during rebalancing require repeated optimizer reconstruction, which introduces overhead that partially offsets backward savings. In our 345M experiments, HiMA-60%COLD is slightly slower than 50%COLD (1.14$\times$ vs. 1.16$\times$), suggesting that optimizing the rebalancing mechanism—for example, by modifying param_group membership in-place rather than reconstructing the optimizer—is an important engineering direction.

Future Work. Three directions are most pressing: (1) **Scaling validation** on 1B+ parameter models to determine whether the loss–efficiency tradeoff improves or worsens with scale. (2) **Adaptive tier ratios** that automatically expand COLD as training converges, eliminating the need to pre-specify (r_H, r_W, r_C). Our preliminary experiments with percentile-based adaptive Cold showed comparable loss (4.71 vs. fixed 4.59) but require longer training runs to demonstrate the expected late-stage benefits. (3) **Downstream task evaluation** (e.g., GLUE, SuperGLUE) to verify that HiMA-pretrained representations transfer as effectively as baseline representations. Since HiMA does not modify the model architecture—all parameters participate in the forward pass—the final checkpoint is architecturally identical to a standard-trained model and can be fine-tuned with any existing method.

Broader Impact. By reducing compute and memory requirements, HiMA could enable LLM pretraining on consumer hardware, reducing environmental cost and barriers to entry for researchers without large compute budgets.

7. Conclusion

We presented HiMA, a hierarchical memory-aware training strategy that challenges the assumption that all parameters require identical every-step updates. By partitioning parameters into HOT, WARM, and COLD tiers based on gradient magnitude, and by genuinely evicting optimizer states for COLD parameters and excluding them from the backward pass, HiMA achieves 31–56% compute savings, up to 1.16$\times$ wall-clock speedup, 32.4% GPU peak memory reduction, and up to 72.8% Adam state memory reduction on WikiText-103 pretraining of GPT-2 Small, with consistent savings observed at GPT-2 Medium scale (65.6% Adam reduction at 60% COLD ratio).

The 2.41-point validation loss gap between HiMA and random freezing at 50% freeze ratio demonstrates that the *quality of the selection criterion* is as important as the quantity of parameters frozen. Gradient-based importance scoring, combined with bidirectional tier promotion, allows HiMA to maintain training stability where static freezing catastrophically fails. Crucially, the WARM tier is not an optional refinement: ablation experiments show that degrading its optimizer momentum continuity accounts for 75% of the total quality gap, confirming its role as an essential bridge that absorbs the shock of tier transitions.

 doi:10.20944/preprints202603.2457.v1

The key insight: **the memory hierarchy is not a constraint to work around—it is a structure for organizing learning priorities.** Fast memory means fast learning. Slow memory means slow learning. No memory means no learning—until the gradient says otherwise. With gradient-guided tiering, the hierarchy itself becomes the training strategy, making large language model pretraining more accessible on memory-constrained hardware.

Appendix A. Hyperparameters

Table A1. Default HiMA hyperparameters used in all experiments.

Parameter	Symbol	Default
Hot ratio	r_H	0.40
Warm ratio	r_W	0.35
Warm update interval	K	10
Warm LR scale	α	0.3
Rebalance interval	R	100
EMA smoothing	β	0.05
Min steps in tier	M_{min}	50

Appendix B. Hardware Setup

All experiments were conducted on a single workstation with the following specifications:

- **GPU:** NVIDIA GeForce RTX 4090 (24GB GDDR6X VRAM)
- **CPU:** AMD Ryzen 7950X
- **RAM:** 128GB DDR5-5600
- **Software Stack:** PyTorch 2.4.0+cu121, CUDA 12.1, Python 3.11

References

T. Brown, B. Mann, N. Ryder, et al. Language models are few-shot learners. *NeurIPS*, 2020.
T. Chen, B. Xu, C. Zhang, C. Guestrin. Training deep nets with sublinear memory cost. *arXiv:1604.06174*, 2016.
Y. Lee, A. Chen, et al. Surgical fine-tuning improves adaptation to distribution shifts. *ICLR*, 2022.
X. Li, Z. Zhang, et al. SmartFRZ: Attention-based layer freezing. *ISCA*, 2024.
S. Merity, C. Xiong, J. Bradbury, R. Socher. Pointer sentinel mixture models. *ICLR*, 2017.
S. Rajbhandari, J. Rasley, O. Ruwase, Y. He. ZeRO: Memory optimizations toward training trillion parameter models. *SC*, 2020.
S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, Y. He. ZeRO-Infinity: Breaking the GPU memory wall. *SC*, 2021.
Y. Sheng, L. Zheng, B. Yuan, et al. FlexGen: High-throughput generative inference with a single GPU. *ICML*, 2023.
Y. Wang, J. Shen, et al. Egeria: Efficient DNN training with knowledge-guided layer freezing. *EuroSys*, 2022.
Q. Wen, J. Lou, et al. GradES: Faster training via gradient-based early stopping. *arXiv:2509.01842*, 2025.
B. Zhang, R. Sennrich. Root mean square layer normalization. *NeurIPS*, 2019.
J. Frankle, M. Carlin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR*, 2019.