NEURIPS 2026 · arXiv:2512.06370

Greedy Alignment Principle for Optimizer Hyperparameter Selection

Choose **momentum** from gradients

Jaerin Lee & Kyoung Mu Lee

Computer Vision Lab, ASRI
Seoul National University

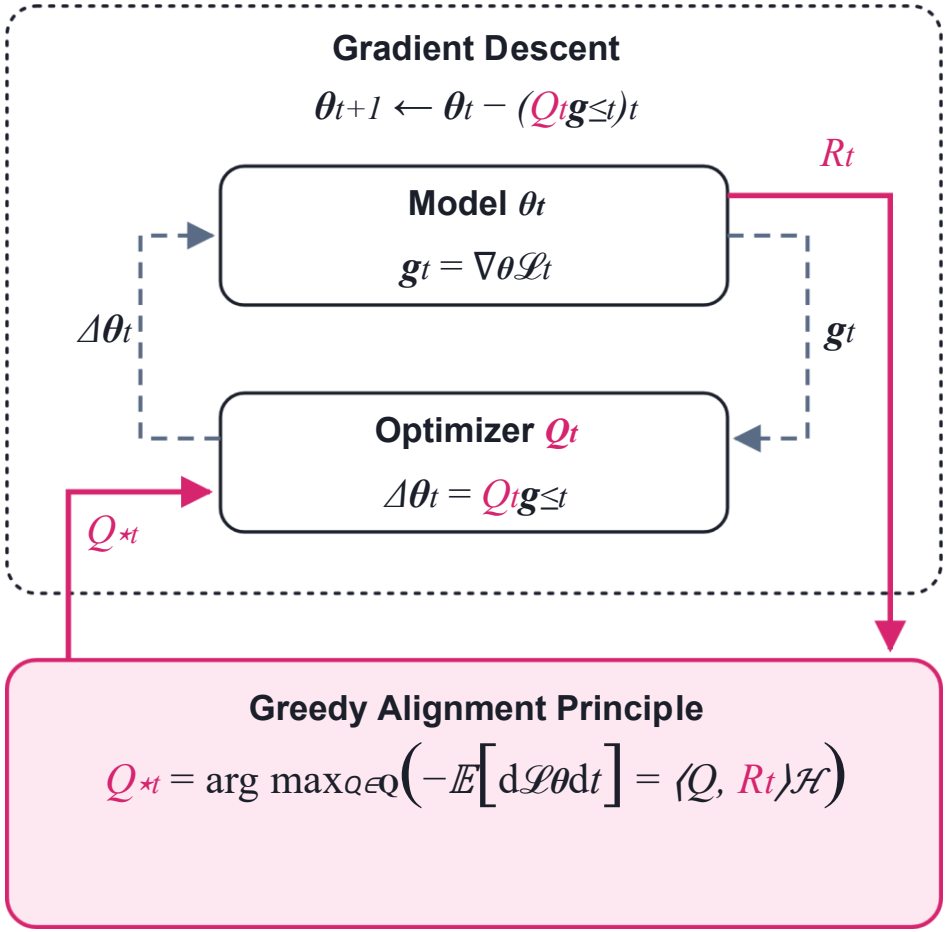


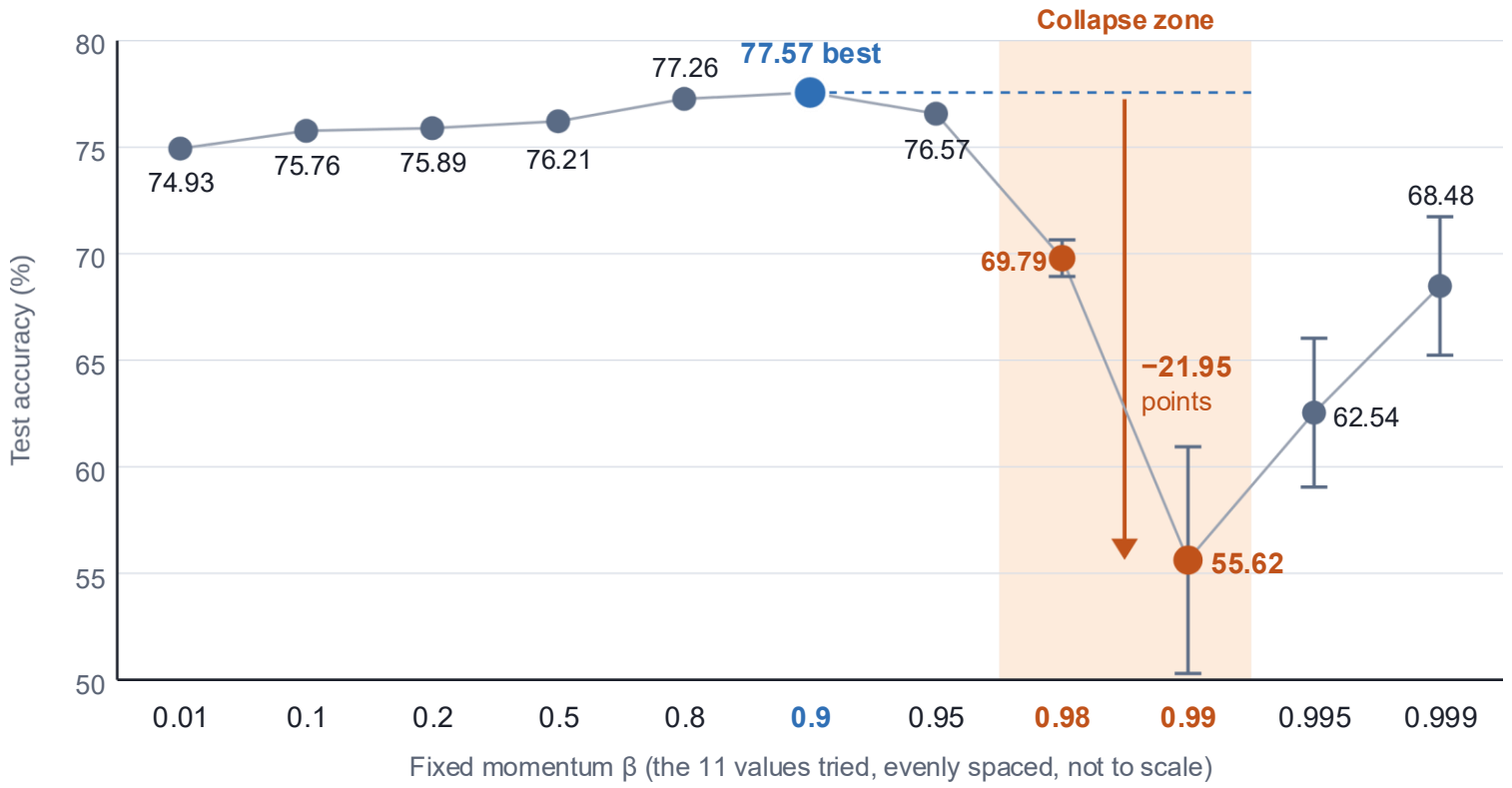
Figure 1, redrawn. The optimizer trains the model with updates $\Delta\boldsymbol{\theta}$. The model's gradient statistics **R** in turn pick the optimizer **Q⋆** from a set of candidates **Q**.

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

MOTIVATION

Picking the momentum β is still trial and error

SGD+Momentum, ResNet-18 on CIFAR-100 · test accuracy for each fixed β (dot = mean, whisker = ± std)



$$m_{\beta,t} = \beta\, m_{\beta,t-1} + g_{t\,\S3.1}$$

β sets how much of the old step **m** is kept when the new gradient **g** arrives.

77.57	best accuracy, at **β = 0.9**
55.62	at **β = 0.99**, about **22 points** lower for a change of 0.09
11	values of β tried, each one a **full training run**

The best β changes with the task and optimizer, so the sweep starts over each time.

IN PLAIN WORDS β sets how long past steps affect an update. Too little momentum stalls; too much overshoots the minimum.

Source: Lee & Lee, NeurIPS 2026, Table 13 (test acc. %, mean ± std), §1 and §3.1

GREEDY ALIGNMENT

Steps for the fastest loss decrease

Gradient $g = \nabla_\theta L$ points uphill. A step is $\theta \leftarrow \theta - u$.

Eq. 1

$$u_t^\star = \arg\max_{u_t \in U_t} g_t^\top u_t$$

Pick the candidate in U_t that lines up best with g.

Eq. 2

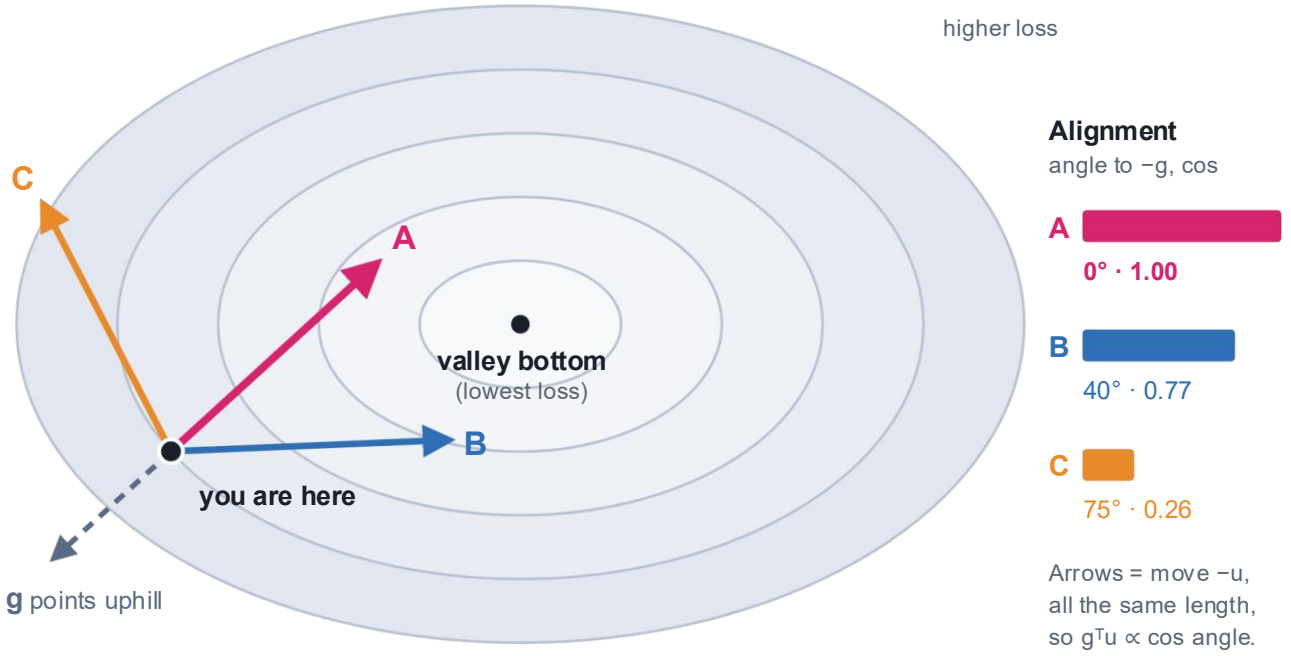
$$g_t^\top u_t = -\nabla_\theta L^\top d\theta_t = -dL_t$$

By the chain rule, alignment is **how fast the loss drops**.

ALREADY USED, BUT ONLY AS AN EDIT

Cautious optimizer and MGUP use $g^\top u$ to filter or reweight the output of a fixed optimizer. This paper uses it to choose the optimizer itself.

ILLUSTRATION

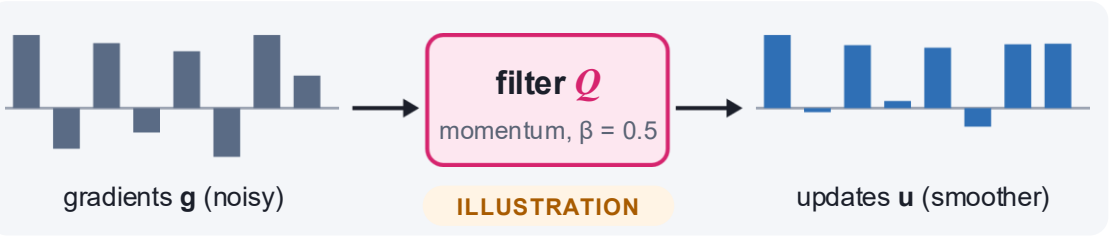


STEP CHOICE Using only the local slope, choose the step with the **fastest loss decrease**. Step A wins.

Source: Lee & Lee, NeurIPS 2026, Eq. 1–2 (p. 2). Hill, steps and angles are an illustration.

THE FILTER VIEW

An optimizer is a filter on the gradient stream



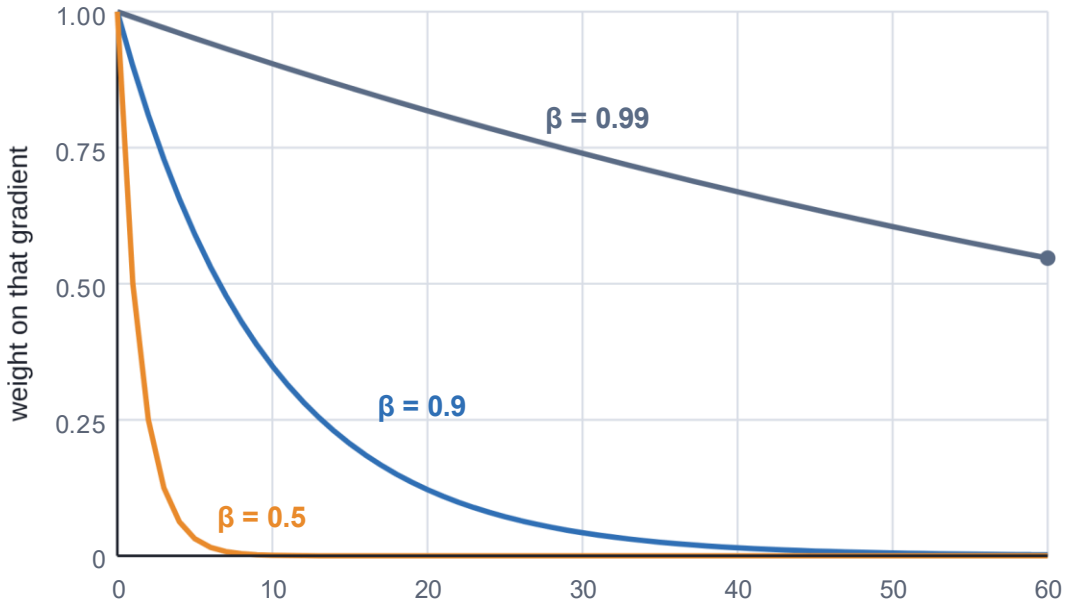
$$\boldsymbol{u}_t = (Q\boldsymbol{g})_t = \sum_{k=0}^{\infty} Q_{t,k}\, \boldsymbol{g}_{t-k} \qquad \text{Eq. 3, 6}$$

Each update mixes today's gradient with older ones. The weights $Q_{t,k}$ are the filter, and the filter is the optimizer.

$$Q_k = \eta\beta_k,\ (Q\boldsymbol{g})_t = \eta\sum_{k=0}^{\infty}\beta_k\boldsymbol{g}_{t-k} \qquad \text{Eq. 13}$$

EMA momentum is a one-pole filter. A gradient k steps old gets weight β_k, so it fades by a factor β every step.

Impulse response of EMA momentum, weight $\beta_k = Q_k/\eta$



Weight left after 60 steps: **β = 0.5** ≈ 0 · **β = 0.9** 0.0018 · **β = 0.99** 0.55

GRADIENT DECAY Each gradient affects later updates, with its weight multiplied by β at every step. Small β removes past gradients quickly; large β retains them longer.

Source: Lee & Lee, NeurIPS 2026, Eq. 3, 6, 13. Chart: exact plot of the Eq. 13 weights for k = 0–60. Top-left strip: example stream filtered with η = 1, β = 0.5.

THEORY · PROPOSITION 2.1

Learning power as an inner product of filter and gradient memory

Eq. 8

$$P_t(Q) = \mathbb{E}[\boldsymbol{g}_t^T \boldsymbol{u}_t] = \sum_{k=0}^{\infty} \mathrm{Tr}(Q_{\tau t,k} R_{t,k}) = \langle Q_t, R_t \rangle_{\mathcal{H}}$$

Learning power $P = -\mathbb{E}[dL/dt]$ is how fast the loss falls. It is **filter** × **gradient memory**, added over all lags.

$R_{t,k} = \mathbb{E}[\boldsymbol{g}_t \boldsymbol{g}_{\tau t-k}]$ compares today's gradient with k steps ago.

(♠)

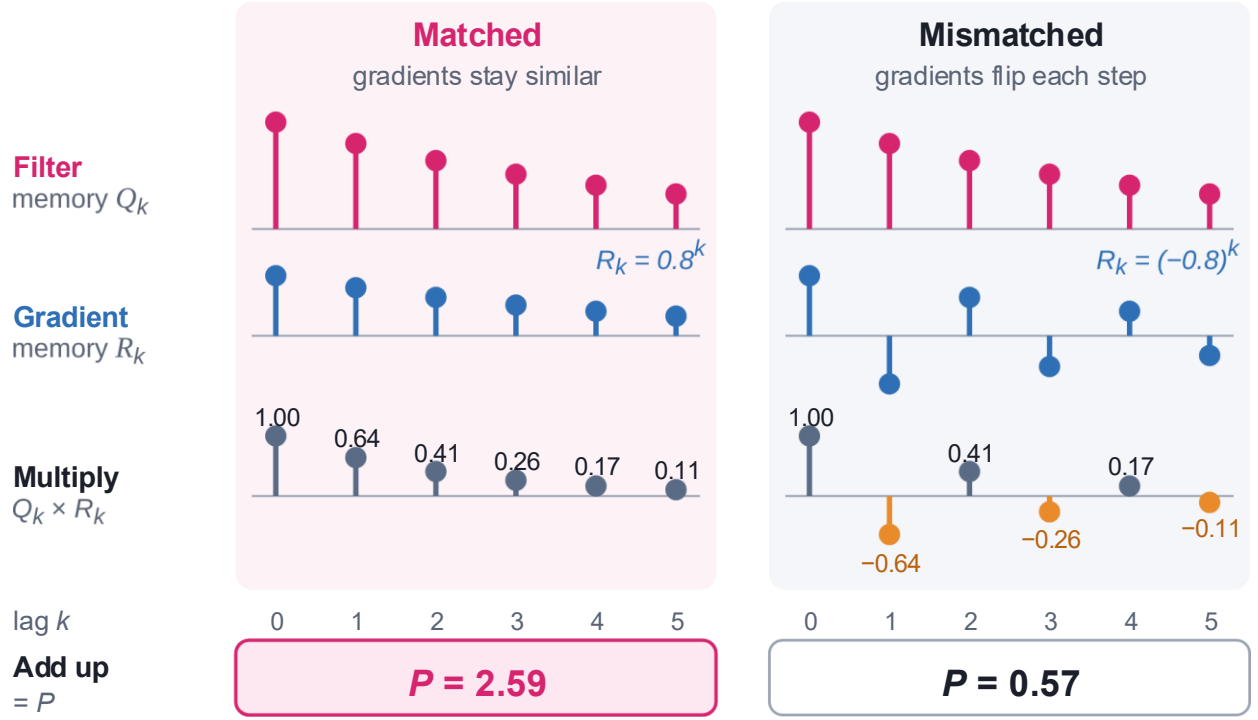
$$Q_t^\star = \arg\max_{Q \in \mathbf{Q}} \langle Q, R_t \rangle_{\mathcal{H}}$$

Pick the filter that lines up best with today's R_t. It is linear, so the search is convex if the set $\mathbf{Q}$ is convex and bounded.

MEMORY LENGTH

Momentum retains past steps. This helps when gradients keep the same direction and hurts when they alternate. Memory should last as long as gradients stay alike.

ILLUSTRATION Momentum filter $Q_k = 0.8^k$ against two gradient streams



No memory at all (plain SGD, $Q = 1, 0, 0, \ldots$) gives P = 1.00 for both streams. The same memory **helps** the matched pair and **hurts** the mismatched one.

Source: Lee & Lee, NeurIPS 2026, Prop. 2.1, Eq. 5–8 and (♠), p. 3. Exhibit: illustration, scalar gradients, lags k = 0–5 summed.

THEORY · THEOREM 2.2 & COROLLARY 2.3

Optimal optimizer existence, stability and corners

Theorem 2.2. A compact convex family has a greedy optimum, stable in R.

i EXISTENCE **The best choice exists**

The maximum learning power $P\star(R)$ is attained and finite for every gradient statistic R.

ii CONSTRUCTION **We can build it**

The optimal optimizer $Q\star$ is a subgradient (a slope) of $P\star$ at R. If it is unique, $Q\star = \nabla_R P\star(R)$.

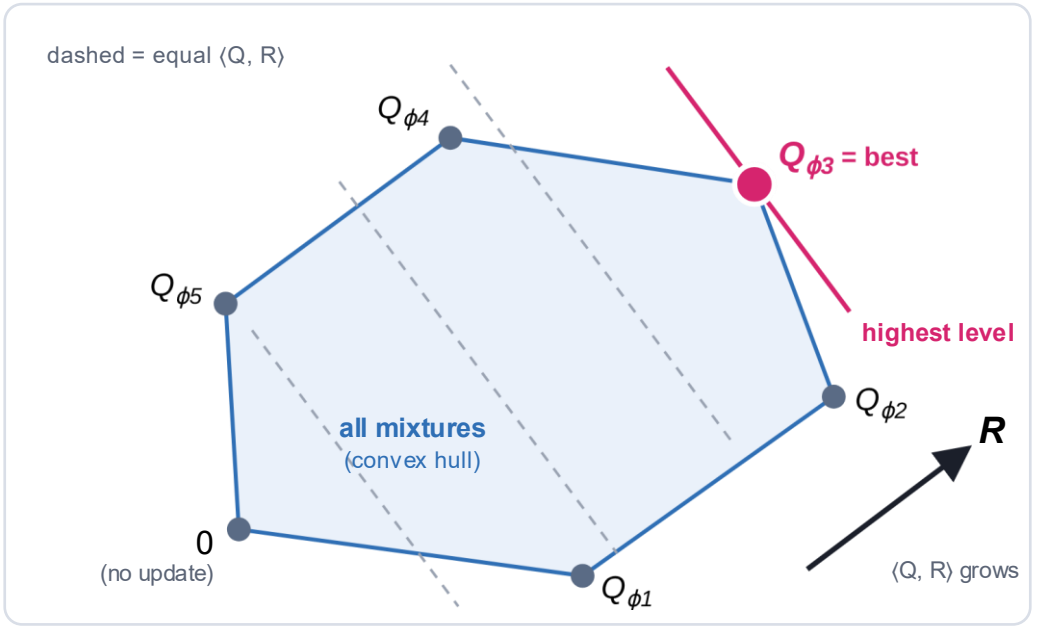
iii STABILITY **Small noise, small error**

$$|P\star(R) - P\star(\hat{R})| \le \|R - \hat{R}\|$$

Noisy **mini-batch** estimates $\hat{R}$ are usable.

Corollary 2.3. Scoring in the hyperparameterized set F is exact.

ILLUSTRATION



The linear objective $\langle Q, R \rangle$ always reaches its maximum at a **corner**. Score each candidate ϕ (and 0) and select the best.

CORNER SEARCH A linear score on this polygon is highest at a corner. We therefore test only the corners, one per setting ϕ, and keep the best.

Source: Lee & Lee, NeurIPS 2026, Theorem 2.2, Corollary 2.3 (Eq. 12), Table 17. Right panel is an illustration.

For SGD+Momentum, the best β equals the gradients' own memory ρ

THE RULE (THEOREM 3.1)

$$\beta_{t\star} = \arg\max_{\beta \in \mathcal{B}} \sqrt{1-\beta^2}\, \mathrm{E}[\boldsymbol{g}_t^{\top} \boldsymbol{m}_{\beta,t}] \quad \text{Eq. 14}$$

Pick the **β** whose momentum $\boldsymbol{m}$ best matches today's gradient $\boldsymbol{g}$.

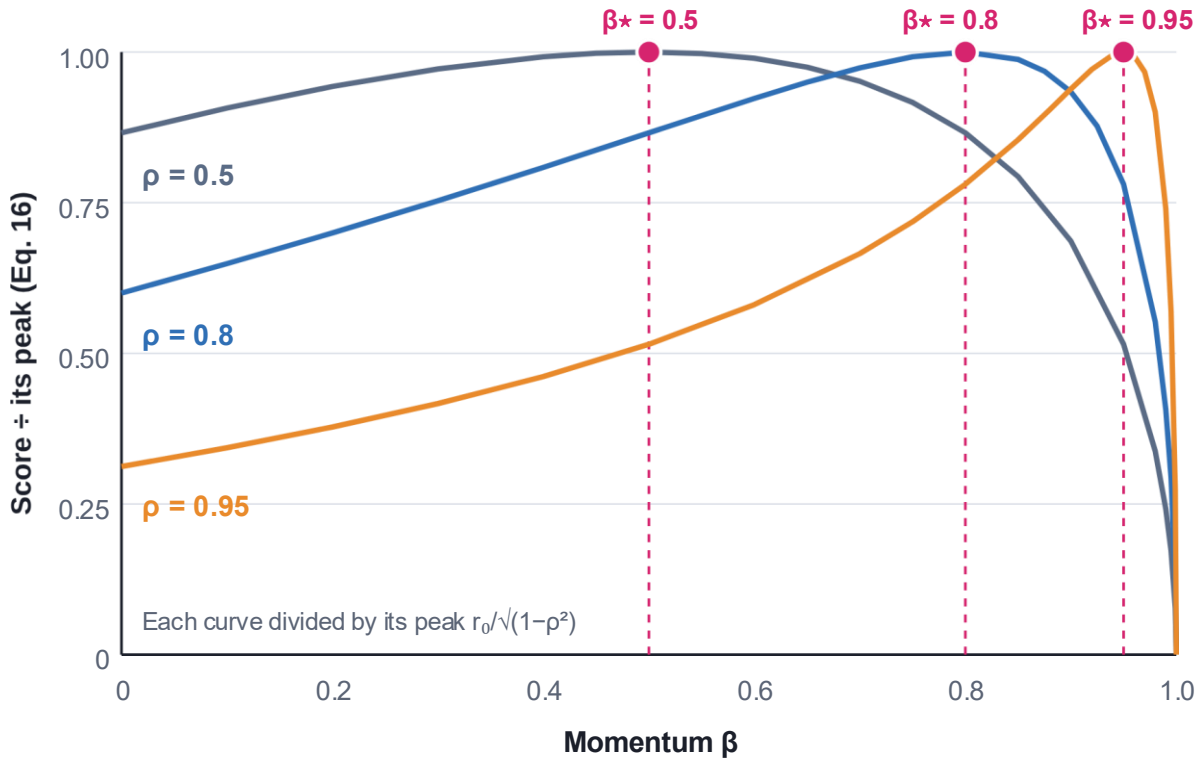
The factor $\sqrt{1-\beta^2}$ keeps the contest fair. It removes the trivial pull toward **no memory** (β = 0) or **infinite memory** (β = 1).

IF GRADIENTS FADE AT RATE ρ (COROLLARY 3.2)

Assume $r_k = \mathrm{E}[\boldsymbol{g}_t^{\top} \boldsymbol{g}_{t-k}] = r_0 \rho^k$ (Eq. 15)

$$\sqrt{1-\beta^2}\, \mathrm{E}[\boldsymbol{g}_t^{\top} \boldsymbol{m}_{\beta,t}] = r_0 \frac{\sqrt{1-\beta^2}}{1-\beta\rho} \quad \text{Eq. 16}$$

Highest at **$\beta_\star = \rho$**. With a cap $\bar{\beta}$, it is $\beta_\star = \max\{0, \min\{\bar{\beta}, \rho\}\}$.



MATCHING B When gradients keep the same direction for longer (big ρ), retain past steps longer (big β). When they change quickly, retain less.

Source: Lee & Lee, NeurIPS 2026 (arXiv:2512.06370), Theorem 3.1, Corollary 3.2, Eq. 14–16. Chart: exact plot of Eq. 16 for ρ = 0.5, 0.8, 0.95.

Greedy alignment for Adam and AdamW

$$(\beta 1\star, \beta 2\star)t = \underset{\beta 1,\, \beta 2 \in (0,1)2}{\arg\max} \sqrt{\frac{1+\beta 1}{1-\beta 1}} \frac{1}{\sqrt{\sum j\, cj-1}} \mathbb{E}[gt\top u\beta 1,\beta 2,t]$$

Eq. 19 · Corollary 3.3

WHAT WE CHOOSE | NORMALIZER | ALIGNMENT

	SGD + Momentum Theorem 3.1, Eq. 14	**Adam / AdamW** Corollary 3.3, Eq. 17–19
MEMORY	$mt = \beta mt-1 + gt$	$\mu t = \beta 1\mu t-1 + (1-\beta 1)gt\ vt = \beta 2vt-1 + (1-\beta 2)gt2$
STEP DIRECTION	$m\beta$, same scale for every parameter	$u = \mu \odot c-1$, each parameter divided by its own $cj = \sqrt{(vj + \epsilon)}$
NORMALIZER	$\sqrt{(1 - \beta 2)}$	$\sqrt{((1+\beta 1)/(1-\beta 1))} \cdot 1/\sqrt{(\sum j\, cj-1)}$
ALIGNMENT	$\mathbb{E}[gt\top m\beta]$	$\mathbb{E}[gt\top u\beta 1,\beta 2]$
FILTER VIEW	Optimal one-pole approximation of a **norm-bounded** optimizer	Optimal one-pole approximation of a **dynamic diagonal** optimizer with preconditioner $c-1$

STEP SCALING Adam measures each parameter's step with **its own ruler** cj. The choosing rule stays the same, so pick the setting that goes downhill fastest.

Source: Lee & Lee, NeurIPS 2026, Theorem 3.1 (Eq. 14), Corollary 3.3 (Eq. 17–19), p. 5–6. Adam's u omits bias correction; AdamW adds weight decay separately.

One greedy step is safe when the step is small

$A(\boldsymbol{u}) = \boldsymbol{g}^{\top}\boldsymbol{u}$ alignment score (linear estimate)

$D\eta(\boldsymbol{u}) = L(\theta) - L(\theta - \eta\boldsymbol{u})$ true loss drop after one step

$$|D\eta(\boldsymbol{u}) - \eta A(\boldsymbol{u})| \le \frac{Ls\eta 2}{2} \quad \|\boldsymbol{u}\|2 \qquad \text{Eq. 20}$$

The estimate's error is at most a term in **η²**. Halving the step cuts the error fourfold.

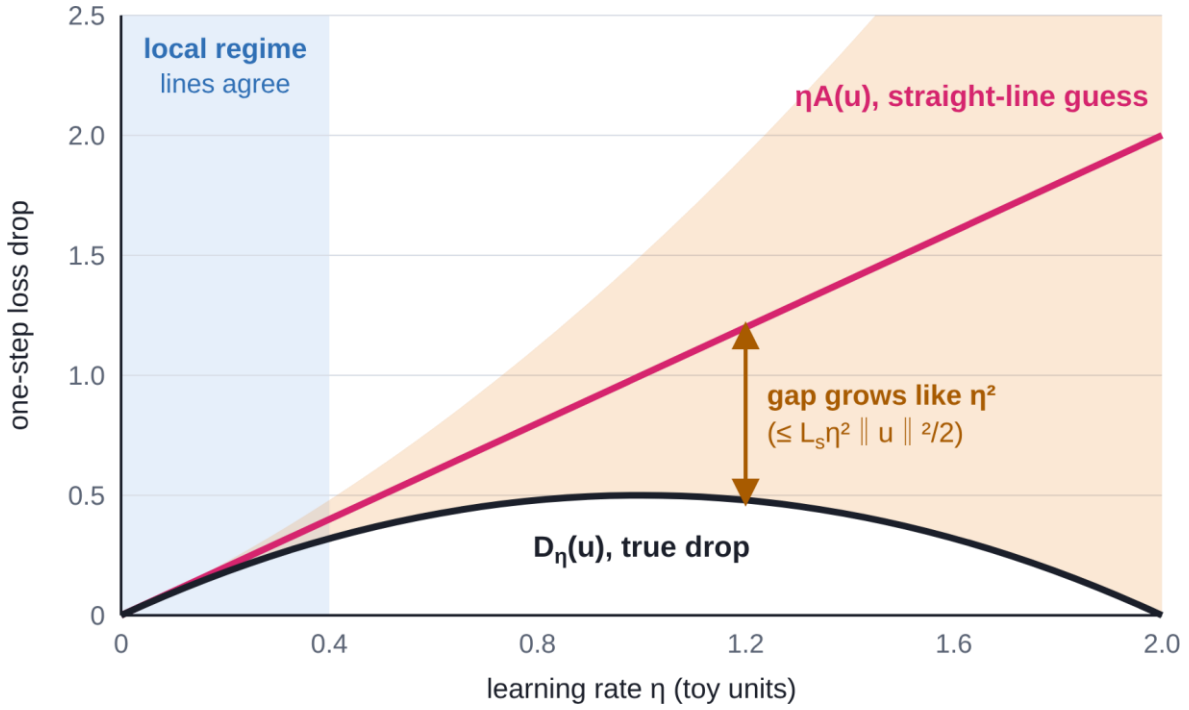
$$D\eta(\boldsymbol{u}\star) \ge \max \boldsymbol{v}\in U\, D\eta(\boldsymbol{v}) - Ls\eta 2\rho 2 \qquad \text{Eq. 21}$$

Picking the best-aligned update $\boldsymbol{u}\star$ loses at most $Ls\eta 2\rho 2$ against the best true drop, when every candidate has size ≤ ρ.

Maximizing $\boldsymbol{g}^{\top}\boldsymbol{u}$ is an O(η²)-accurate stand-in for the real one-step drop.

Toy loss $L(\theta) = \frac{1}{2}\theta^2$ at $\theta = 1$, update $\boldsymbol{u} = \boldsymbol{g}$, so $Ls = 1$

ILLUSTRATION



STEP SIZE A linear estimate describes a curved loss well only for a short step. With tiny steps, the estimated and actual loss drops match. With a large step, curvature makes them differ.

Source: Lee & Lee, NeurIPS 2026, Sec. 3.2, Theorem 3.4 (Eq. 20) and Corollary 3.5 (Eq. 21). Chart: illustration on a toy quadratic.

ALGORITHM · K-SWITCH

K-Switch runs K momenta and uses the one with best alignment

Algorithm 1 K-Switch SGD+Momentum

Require: η, $\{\beta_k\}_{k=1K}$

$\boldsymbol{\mu}_k \leftarrow \mathbf{0}$ for all k

for each step n **do**

  $\mathbf{g} \leftarrow \nabla_\theta \mathscr{L}(\boldsymbol{\theta})$

  for $k = 1, \ldots, K$ **do**

    $\boldsymbol{\mu}_k \leftarrow \beta_k \boldsymbol{\mu}_k + \mathbf{g}$

    $A_k \leftarrow \sqrt{1-\beta_{k2}}\, \mathbf{g}^{\top} \boldsymbol{\mu}_k$ **ADDED**

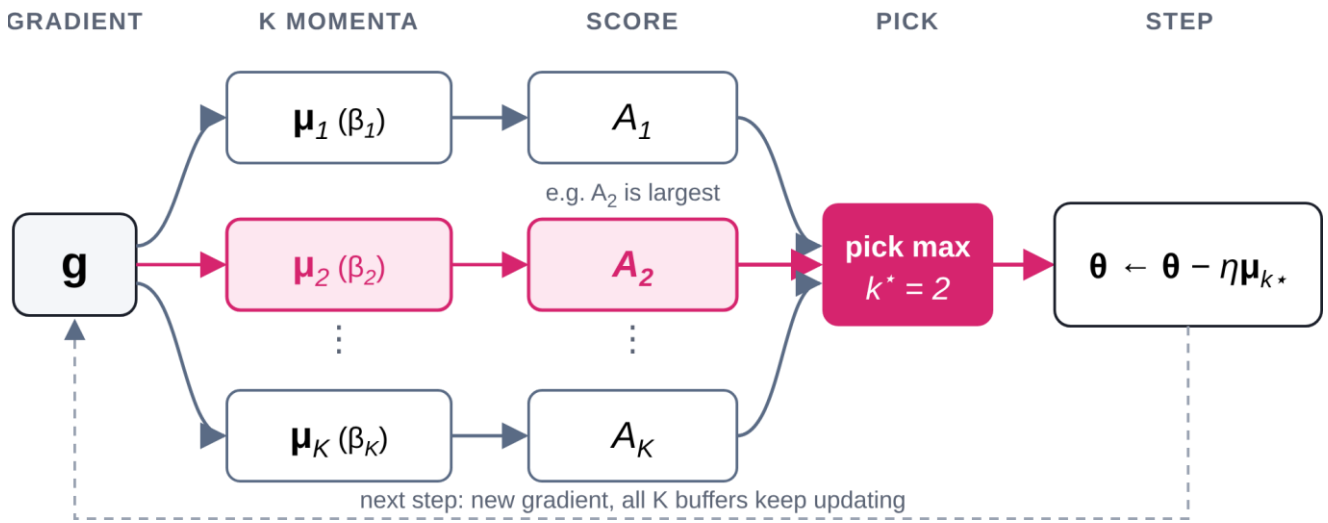
  end for

  $k_\star \leftarrow \arg\max_k A_k$ **ADDED**

  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \boldsymbol{\mu}_{k_\star}$

end for

Pink lines are the only change to plain SGD+Momentum. The factor $\sqrt{(1-\beta_{k2})}$ is used only for choosing; the step size η stays with the usual learning-rate scheduler.



COROLLARY 4.1 · EXACT

K-Switch is **not a heuristic**. Over the K normalized candidates (and any blend of them), picking the single best is the exact greedy optimum.

PROPOSITION 4.2 · STABLE

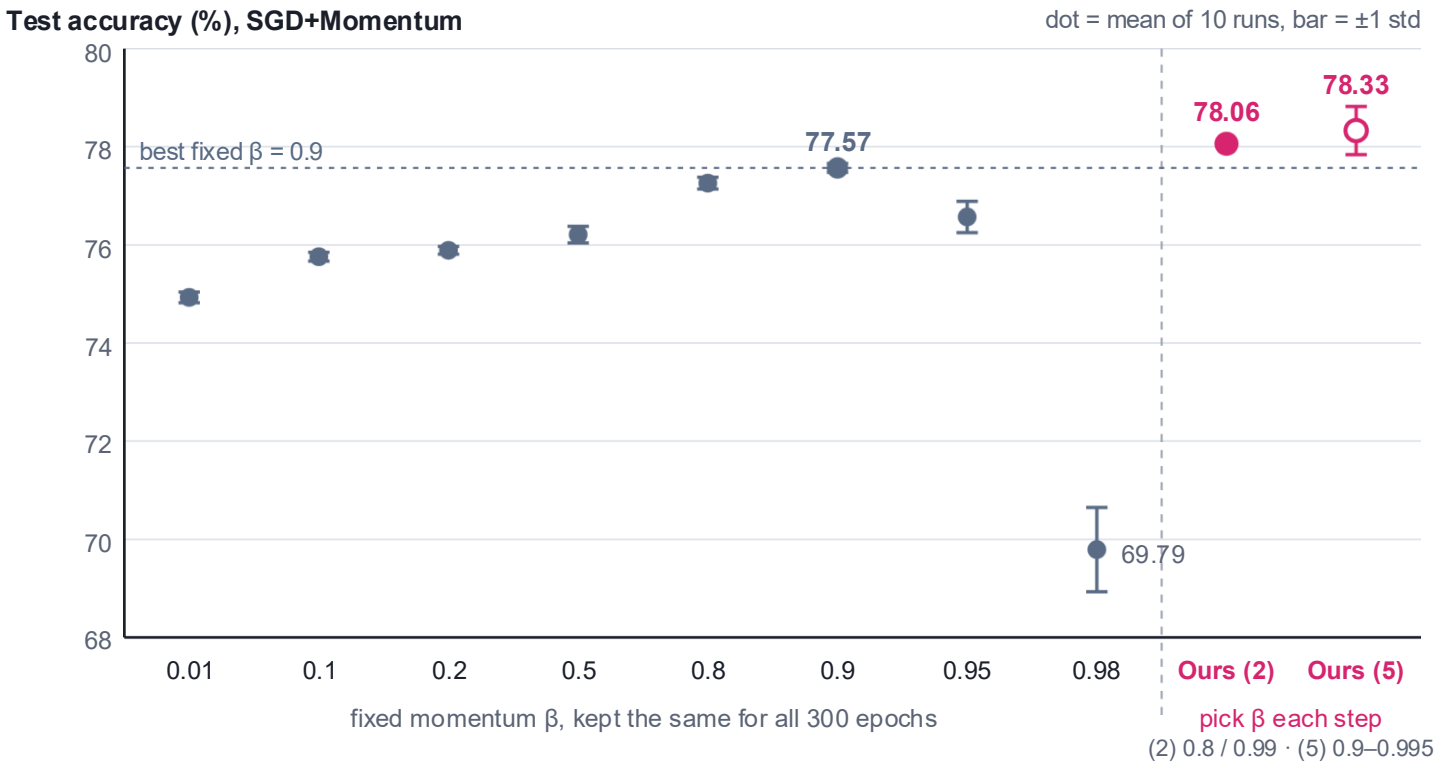
Mini-batch scores are noisy by at most ε. If the best candidate leads by **more than 2ε**, the pick is still correct. Experiments use K **= 2**.

AT EACH STEP At each step, compare a few candidate updates and use the one with the **fastest loss decrease**.

Source: Lee & Lee, NeurIPS 2026, Algorithm 1 (p.6), Sec. 4.1–4.2, Corollary 4.1, Proposition 4.2. Diagram example k⋆ = 2 is illustrative.

RESULTS · RESNET-18 ON CIFAR-100

Switching β every step beats every fixed β



SGD+Momentum (Tables 1, 13)

Method	Test acc. %	Train loss
Best fixed β = 0.9	77.57 ± 0.09	0.0078 ± 0.0001
Ours, 2 options	**78.06 ± 0.07**	**0.0080 ± 0.0001**
Ours, 5 options	78.33 ± 0.49	0.0073 ± 0.0001

Accuracy **+0.49 pt**; train loss about equal (0.0080 vs 0.0078). The 5-option switch costs about +100% compute.

Adam (Tables 2, 14)

Method	Test acc. %	Train loss
Best fixed (0.8, 0.999)	73.20 ± 0.21	0.0324 ± 0.0042
Ours, 2 options	**73.26 ± 0.31**	**0.0115 ± 0.0010**

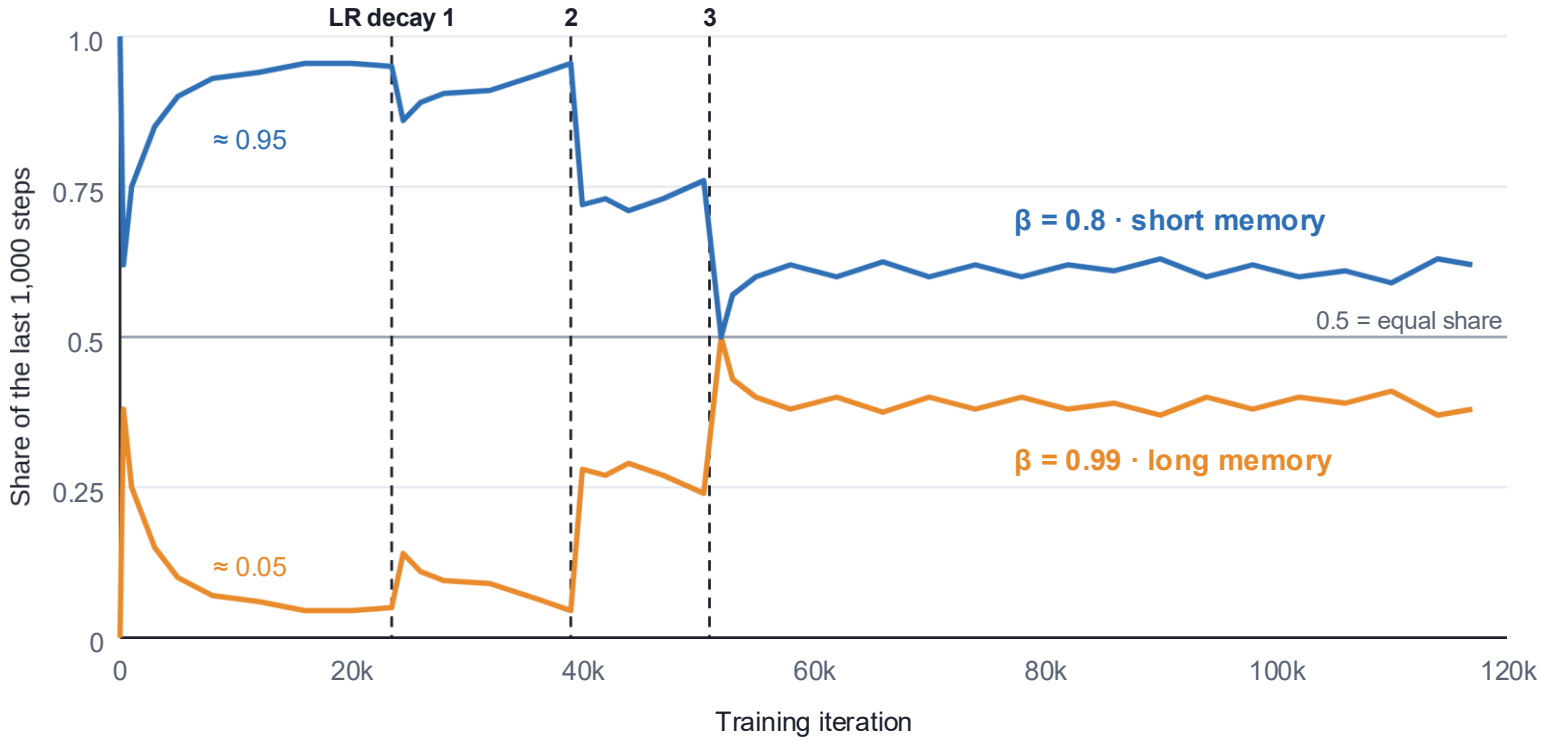
Accuracy is within the noise; train loss is **about 3× lower**.

TWO B OPTIONS Consider two settings, β = 0.8 and 0.99. Keeping β = 0.99 throughout training gives only **55.62%** accuracy. Switching to the setting that fits each step gives **78.06%**, better than any fixed setting.

Source: Lee & Lee, NeurIPS 2026, Figure 2 and Tables 1, 2, 13, 14 (ResNet-18, CIFAR-100, 10 runs, mean ± std). Fixed β ≥ 0.99 (55.62–68.48%) fall below the axis.

RESULTS · INSIDE ONE RUN

K-Switch β choices during training



Share of steps per phase

Approximate

Phase	β = 0.8	β = 0.99
Start 0 – 23.5k	≈ 0.95	≈ 0.05
After decay 1 23.5k – 39k	≈ 0.91	≈ 0.09
After decay 2 39k – 51k	≈ 0.73	≈ 0.27
After decay 3 51k – 117k	≈ 0.61	≈ 0.39

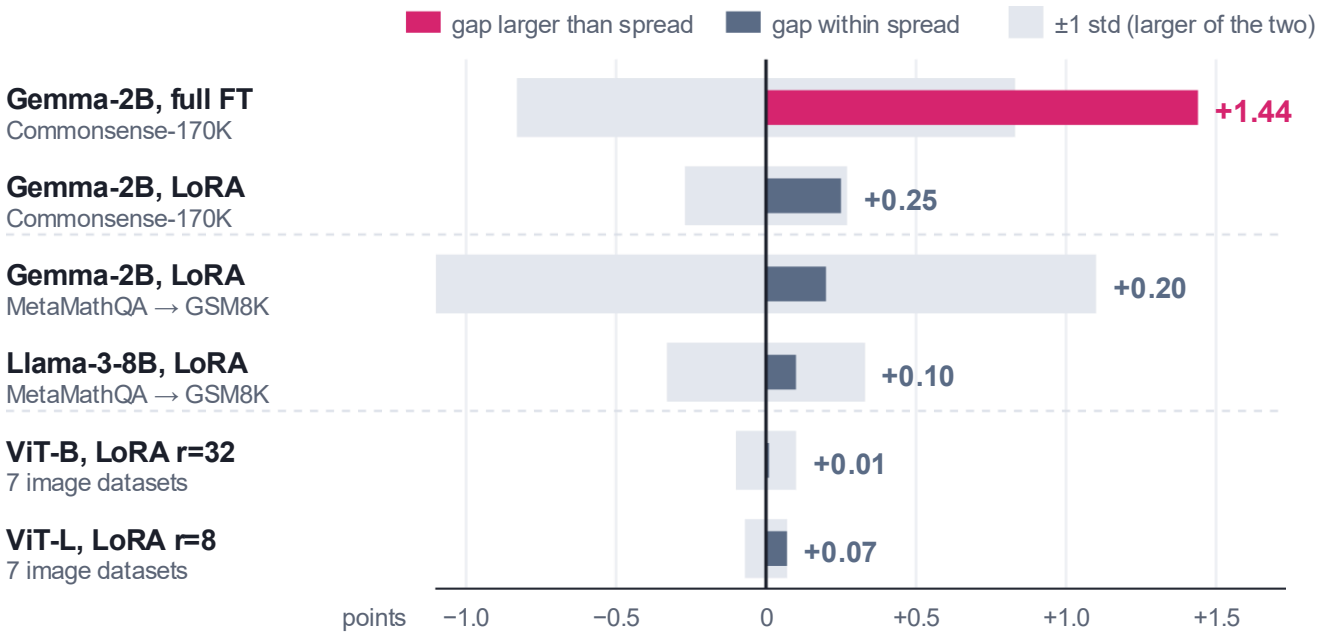
TRAINING PHASES Each step K-Switch chooses between **short memory** and **long memory**. Early on, it uses short memory almost every time. Later, it uses long memory about 4 steps in 10.

The preference drifts toward 0.5 and jumps at every LR decay, so no single fixed β suits the whole run.

Source: Lee & Lee, NeurIPS 2026, redrawn from Fig. 5a (approx.). SGD+Momentum, ResNet-18 on CIFAR-100, 1,000-step trailing window.

Large language models and vision transformers

Average test accuracy, **K-Switch minus best fixed β1** (percentage points)



Avg test acc. %	Best fixed	K-Switch	β1 pair
Commonsense-170K			
Gemma full FT	61.35 ± 0.27	**62.79 ± 0.83**	0.1, 0.99
Gemma LoRA	72.76 ± 0.17	**73.01 ± 0.27**	0.8, 0.95
GSM8K			
Gemma-2B	52.57 ± 1.10	**52.77 ± 0.93**	0.8, 0.99
Llama-3-8B	76.20 ± 0.33	**76.30 ± 0.31**	0.8, 0.99
ViT image classification			
ViT-B	83.95 ± 0.06	**83.96 ± 0.10**	0.5, 0.99
ViT-L	86.50 ± 0.05	**86.57 ± 0.07**	0.5, 0.99

Best fixed is the best result from an oracle sweep over many β1 values. **K-Switch** switches between two preset β1 values in one run.

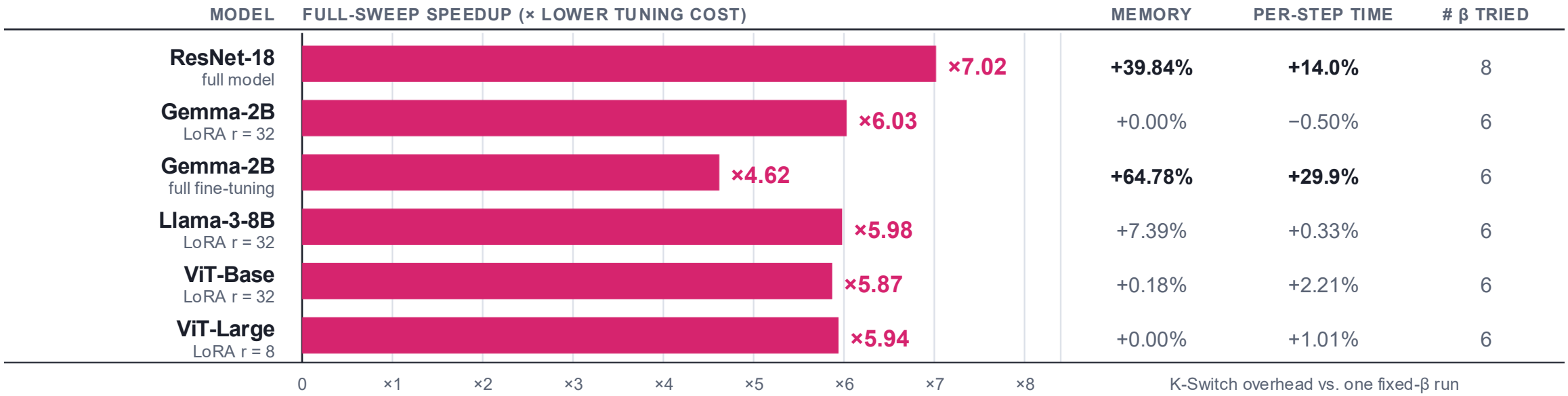
Commonsense full fine-tuning gains +1.44 points. The other five gaps are within one std, a tie.

ACROSS TASKS A sweep tests many settings and keeps the best. K-Switch chooses during one run and scores about the same, and clearly better on one test. A gap smaller than the usual variation across runs is not a real win.

Source: Lee & Lee, NeurIPS 2026, Tables 3–6 (mean ± std; GSM8K over 10 runs) and Tables 8–12 (β1 pairs). Gap = K-Switch − best fixed baseline.

RESULTS · TUNING COST

One K-Switch run replaces a full β sweep



Tuning cost drops by at least ×4.62. Overhead is small for LoRA and moderate for full-model runs.

$$speedup \approx \#\,\beta\ candidates 1 + step\ overhead$$

Gemma-2B full FT: 6 ÷ 1.299 = ×4.62 · ResNet-18: 8 ÷ 1.14 = ×7.02

RUN COUNT A sweep compares settings in **6 runs**, one after another. K-Switch compares settings during **one run**. That run is a little slower, but training happens only once.

Source: Lee & Lee, NeurIPS 2026, Table 7 (VRAM and per-step wall-clock time, same fused PyTorch/Triton optimizers). The speedup formula fits all six Table7 columns.

WRAP-UP

Promises and limits of the principle

TAKEAWAYS

1. **An optimizer is a filter**
   It mixes past gradients into one step.
   $$u_t = \sum_k Q_{t,k}\, g_{t-k}$$ Eq. 6

2. **Learning power is an alignment score**
   Choose the **β** whose filter best matches the gradient autocorrelation.
   $$P_t(Q) = \langle Q_t, R_t \rangle_{\mathcal{H}}$$ Eq. 8

3. **K-Switch needs one run, not a sweep**
   It matches or beats oracle-tuned fixed β. Commonsense-170K average **+0.25** (LoRA) and **+1.44** (full FT). Tuning cost cut by at least **×4.62**.

LIMITATIONS (SEC. 6)

1. **Greedy and first-order**
   It maximizes today's expected loss-drop rate. No global convergence or validation-loss guarantee.

2. **You still pick the candidate family**
   It tunes inside the family Q you supply. It does not choose the optimizer class.

3. **Adam is only approximated**
   Adam is nonlinear, so it is treated as a local time-varying filter.

4. **Memory and runtime grow with K**
   Up to **+64.78%** memory and **+29.9%** step time (Gemma-2B full FT), near zero with LoRA.

DURING TRAINING The optimizer is chosen from the **current gradient statistics** while it trains the model. Each update follows the local slope.

QUESTIONS
ironjr@snu.ac.kr
arXiv:2512.06370

Source: Lee & Lee, NeurIPS 2026, Sec. 5–7, Eq. 6, Eq. 8, Table 7