Do Deep Networks Forget Initialization? A Forgetting-Time View of Practical Inductive Bias

Mohua Das* **Pierfrancesco Beneventano***

Shibshankar Dey **Gareth H. McKinley** **Tomaso Poggio**

Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

Randomly initialized neural networks induce a prior over functions, but the predictor used in practice is produced only after training. We ask how much of this initial bias survives the training pipeline. To make the question measurable, we introduce *initialization memory*: the dependence of the validation-selected predictor on the scale of the random initialization. We perform controlled experiments on BatchNorm ResNet 9/56/110 (CIFAR-10) and NanoGPT (OpenWebText) where initialization memory already sharply separates training regimes. Low-learning-rate SGD interpolates while still remembering its initialization: ResNet-9 with batch size $b = 128$, test accuracy varies by 26.5 percentage points across initialization scales despite $\geq 99.5\%$ training accuracy. In contrast, Adam-family methods largely erase the dependence. SGD can also be made to forget when larger learning rates are paired with explicit L_2 norm control. We interpret these findings in terms of the time scale of forgetting: gradient-flow-like dynamics can preserve initialization memory, whereas stochastic finite-step effects, explicit norm decay, and adaptive preconditioning erase it on scales governed by the size of explicit or implicit regularization. The practical inductive bias of a trained network is therefore not the architectural/initialization prior alone, but the architectural prior after being filtered by the forgetting dynamics of the training pipeline. On our grid, training procedures with stronger implicit or explicit regularization tend to both improve generalization and reduce the σ_w-spread. Code is available at `github.com/mohuadas-neurips/Initialization-Bias-Deep-Neural-Network`.

1 Introduction

Modern neural networks are too expressive a priori for performance to be explained by capacity alone. Thus, the relevant object is not merely the hypothesis class, but the training pipeline that selects one function from it [1]. In practice, this pipeline includes data preprocessing, architecture, initialization, optimizer, batching, explicit regularization, and training time. Understanding performance, therefore, requires understanding not only the prior induced by the architecture, but also how training transforms that prior. This article investigates a precise subquestion:

What is the role of initialization in explaining performance?

Initialization is a natural place to look for such bias. From a dynamical systems perspective, the initial condition, in the absence of regularization, determines which region of parameter space the trajectory explores (basin of attraction) and which solution is ultimately selected [2, 3]. In simplified linear and homogeneous networks, initialization is also known to control the implicit bias of gradient-based training [4, 5]. A line of work studies the function prior of random networks: before seeing labels,

*Equal contribution. Correspondence to `{mohuadas, pierb}@mit.edu`

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

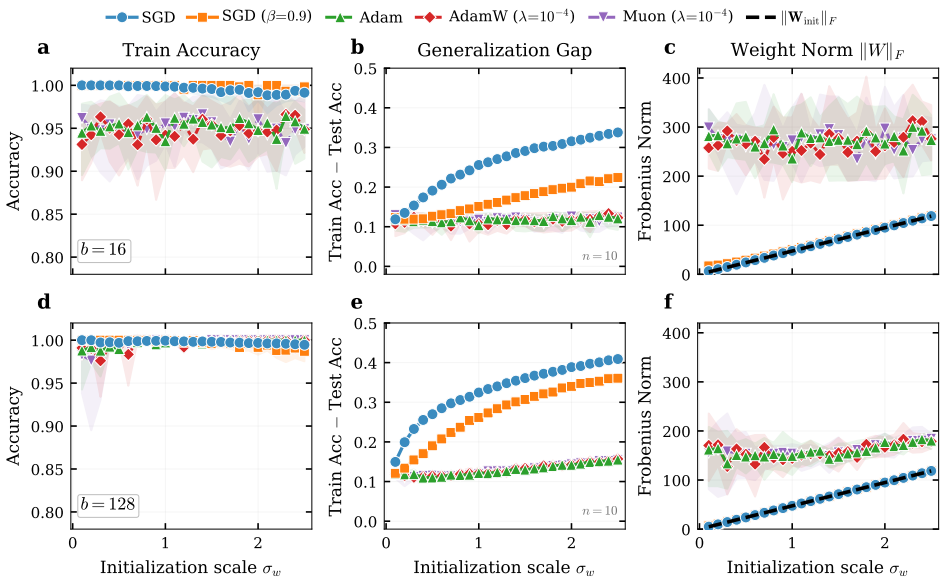


Figure 1: **SGD remembers initialization; Adam-family methods forget.** ResNet-9 under a shared low-learning-rate training procedure. Each curve shows the mean over $n = 10$ seeds; shaded bands indicate the $10^{\text{th}} - 90^{\text{th}}$ percentile range. SGD interpolates, but its generalization gap grows with σ_w. Adam, AdamW, and Muon show substantially weaker dependence on σ_w. The norm panels show radial memory: SGD retains sensitivity to the initial norm (dashed line), whereas over this range ($\sigma_w \leq 2.5$) adaptive methods move toward a common final norm scale; at much larger σ_w they do not (Appendix E.2). Here "memory" refers to the returned predictor's dependence on σ_w: a common final norm indicates reduced radial dependence, not that training has erased the initial parameters.

architectures, and initialization schemes assign much higher probability to some functions than to others, often favoring simple ones [6–8]. Mingard et al. [7] distinguish the first-order question of why overparameterized deep neural networks (DNNs) generalize at all from the second-order question of how to further improve the performance of already-generalizing models. Our focus is on one concrete part of that bridge: whether the simplicity bias present at initialization survives training strongly enough to remain visible in the final predictor and in practical performance. Answering this requires studying (i) what happens after optimization begins and (ii) whether practical performance is affected by these geometric biases. The simplicity bias present at initialization may be preserved, distorted, or erased by the subsequent training dynamics. Thus, the practical question is not only whether random networks have a simplicity bias, but whether that bias remains visible in the predictor selected by a modern training pipeline. The question, given these works, becomes:

When does training remember initialization's bias, when does it forget it, and on what time scale? How does this geometric simplicity bias resolve in practical performance?

In apparent contrast with the literature above, a different intuition is common in large-scale model training. There, initialization is often treated less as a source of final inductive bias and more as a mechanism for stabilizing optimization: preventing exploding or vanishing signals, enabling depth, and making training predictable at scale. This perspective underlies variance-preserving initialization schemes, random-matrix and dynamical-mean-field analyses of signal propagation [9–17], and modern parameterization theory such as maximal-update parameterization, μP [18–23]. From this viewpoint, improvements attributed to initialization may come primarily from making training possible or stable, rather than from a persistent preference among functions.

Recent evidence that random seeds and initialization can affect language-model training—both during fine-tuning [24] and during pretraining [25–28]—further sharpens the timeline of the issue. Related controlled studies of language-model training pipelines and architecture choices reinforce that large-scale behavior is shaped by more than the architecture alone [29, 30].

Our contribution. As we argue above, there is substantial work showing that random networks and SGD-trained networks are biased toward simple functions, and there is separate work showing that normalization, SGD noise, regularization, and finite-step discretization strongly alter optimization trajectories [31–35] (further related work in Appendix H). What is still comparatively under-articulated is when initialization-induced simplicity survives training and when it is forgotten. That is the precise gap our paper occupies.

This article is an empirical study in deliberately controlled settings: ResNets on CIFAR-10 and GPT-2 on OpenWebText. We perform extensive ablations across initialization scales, optimizers, batch sizes, depths, training horizons, and explicit regularization. We introduce initialization memory as our diagnostic: the extent to which the predictor returned by a training pipeline still depends on the initialization scale σ_w. Our diagnostic tracks one controlled projection of the initialization, its *scale* σ_w, rather than the full functional prior: whether this projection remains visible in, or is reduced by, the training procedure. Regarding CIFAR-10, our main grid uses BatchNorm ResNets; normalization and augmentation checks (BatchNorm, LayerNorm, and an unnormalized (NoNorm) ResNet) appear in Appendix E. Regarding language, we extend the study to transformer pretraining: GPT-2 (124M, nanoGPT) on OpenWebText, sweeping σ_w over SGD-momentum, Adam, and Muon under a fixed 600M-token budget. Notation is summarized in Table 2 (Appendix A.1).

1. **A controlled phase diagram of initialization-scale memory.** We show that training procedures differ sharply in whether the returned predictor still depends on initialization scale.
   - Low-learning-rate SGD can interpolate while retaining large initialization-scale memory: for ResNet-9 at $b = 128$, every run reaches $\geq 99.5\%$ training accuracy during training, yet test accuracy at the validation-selected checkpoint varies by 26.5 percentage points across σ_w.
   - In the same diagnostic grid, Adam, AdamW, and Muon instead largely erase this dependence.

   Hyperparameter ablations and depth stress tests show that the failure mode changes across regimes: poor forgetting can appear both as interpolation with poor generalization in shallower networks, or as degraded trainability in deeper ones.
2. **A separation between interpolation, training horizon, and initialization-scale forgetting.** We show that erasing initialization-scale dependence is not implied by fitting the labels or by extending the same low-learning-rate (low LR) dynamics. A $5{,}000$-epoch low-LR SGD control leaves the initialization-scale spread essentially unchanged. SGD can nevertheless be made to forget when the recipe supplies larger effective movement (larger implicit regularization) or explicit regularization as weight decay, showing that forgetting is a property of the full training recipe rather than the optimizer name. In particular, we show that the training procedure forgets more with a larger learning rate and regularization, or a smaller batch size.
3. **A forgetting-timescale mechanism.** We organize the results by cumulative optimizer clocks:

$$\mathcal{T}_{\mathrm{SGD}} = \frac{1}{b}\sum_{k<K}\eta_k^2, \qquad \mathcal{T}_{L_2} = \lambda\sum_{k<K}\eta_k, \qquad \mathcal{T}_{\mathrm{adapt}} = \sum_{k<K}\eta_k.$$

   These time scales track stochastic finite-step effects, explicit norm decay, and adaptive preconditioning. They explain why epoch count and interpolation time are not reliable proxies for erasing initialization-scale dependence, and are supported by the repair map, sensitivity-collapse plots, and minimal conservation-law model.

Importantly, we see that the main takeaways align with the (implicit) regularization literature:

Gradient-flow-like training can remember initialization; stochasticity, adaptivity, and norm control erase it on optimizer-dependent time scales.

2 Experimental Design

We use the initialization scale as a controlled perturbation of the training pipeline. For each optimizer–batch-size pair, we sweep the global scale σ_w of the random kernel initialization while holding all other choices fixed. If the predictor returned by the same training and checkpoint-selection rule varies across σ_w, the procedure has retained initialization-scale memory; if the dependence is small, this component of the initial condition has been erased.

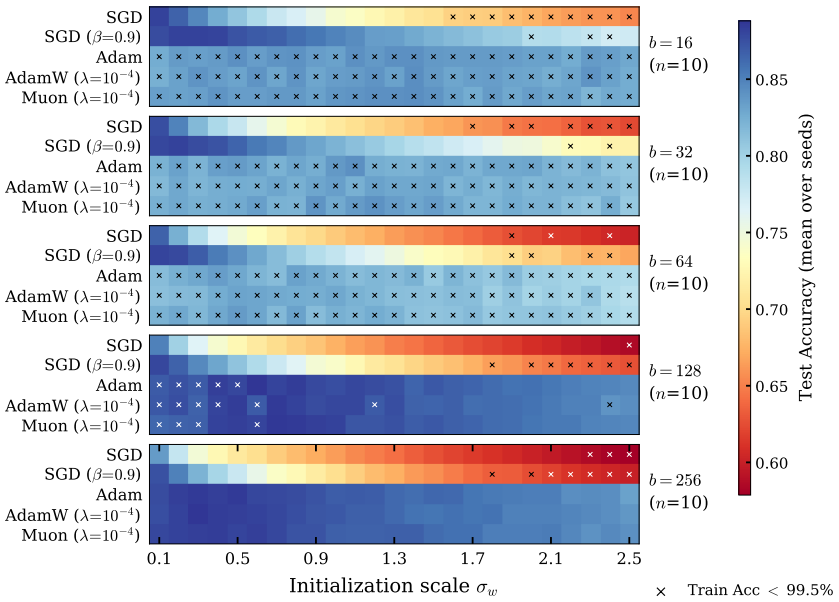


Figure 2: **Large-batch fixed-epoch regimes forget initialization more slowly.** ResNet-9 test accuracy at τ_{best}, averaged over $n = 10$ seeds. Each panel corresponds to one batch size $b \in \{16, 32, 64, 128, 256\}$; within each panel, rows are optimizers and columns are initialization scales. Cells marked $\times$ did not reach 99.5% mean training accuracy at the best-validation-loss checkpoint τ_{best}. All cells are plotted and included in Mem; the $\times$ is a diagnostic annotation, not a filter used to compute initialization memory. SGD shows a strong left-to-right degradation that becomes more pronounced at larger batch sizes; adaptive methods remain much flatter.

Formally, let Σ be the finite set of scales in the sweep and let $\mathcal{R}$ denotes the full training procedure: architecture, data split, optimizer, batch size, schedule, regularization, horizon, and checkpoint rule. Let $A_{\mathcal{R},K}(\sigma, s)$ be the predictor returned after K updates from scale σ and seed s. For metric m, define

$$\mathsf{Mem}_m^{\text{ret}}(\mathcal{R}, K; \Sigma) = \max_{\sigma \in \Sigma} \mathbb{E}_s[m(A_{\mathcal{R},K}(\sigma, s))] - \min_{\sigma \in \Sigma} \mathbb{E}_s[m(A_{\mathcal{R},K}(\sigma, s))].$$

In the main experiments, m is the test accuracy, and the returned predictor is the best-validation-loss checkpoint.

Diagnostic. *Initialization-scale memory is the finite-horizon dependence of the returned predictor on one controlled initialization perturbation, σ_w. It is a property of the full training-and-selection procedure.*

Controlled setting: vision. We use CIFAR-10 BatchNorm ResNets. The main ResNet-9 grid fixes the data split, architecture, learning-rate schedule, and training horizon, and varies only initialization scale, optimizer, and batch size. We sweep $\sigma_w \in \{0.10, 0.20, \ldots, 2.50\}$, optimizers SGD, SGD with momentum, Adam, AdamW, and Muon, and $b \in \{16, 32, 64, 128, 256\}$, with 10 seeds per configuration. Convolutional and dense kernels are drawn from $W_{ij} \sim \mathcal{N}(0, \sigma_w^2/\text{fan}_{\text{in}})$, so $\sigma_w = 1$ is the fan-in normal baseline and σ_w is the only initialization quantity swept in the main grid. Biases are drawn from $\mathcal{N}(0, 0.2^2)$ and are not rescaled by σ_w; BatchNorm affine parameters use the framework defaults and are not swept. $\eta_0 = 10^{-3}$ with cosine decay for 300 epochs. Ablations over learning rate, L_2 regularization, training length, normalization, augmentation, and depth (ResNet-56, ResNet-110, R9-AvgPool) are reported in Appendices C–E; full hyperparameters are in Appendix A.

Controlled setting: language. To test generality beyond vision, we additionally pretrain GPT-2 (124M, nanoGPT) on OpenWebText for a fixed 600M-token budget ($\approx$4575 iterations), with 3 seeds for each $\sigma_w \in \{0.5, 1.0, 2.0, 4.0\}$. Here σ_w multiplies the standard deviation of the default GPT-2 initialization for all normal-initialized weights, so $\sigma_w = 1$ is the standard initialization. We compare SGD with momentum (0.9), Adam, and Muon, all with weight decay 0; here the metric m is validation loss, and we additionally report the final-to-initial Frobenius norm ratio $r_f = \|W_f\|_F / \|W_i\|_F$.

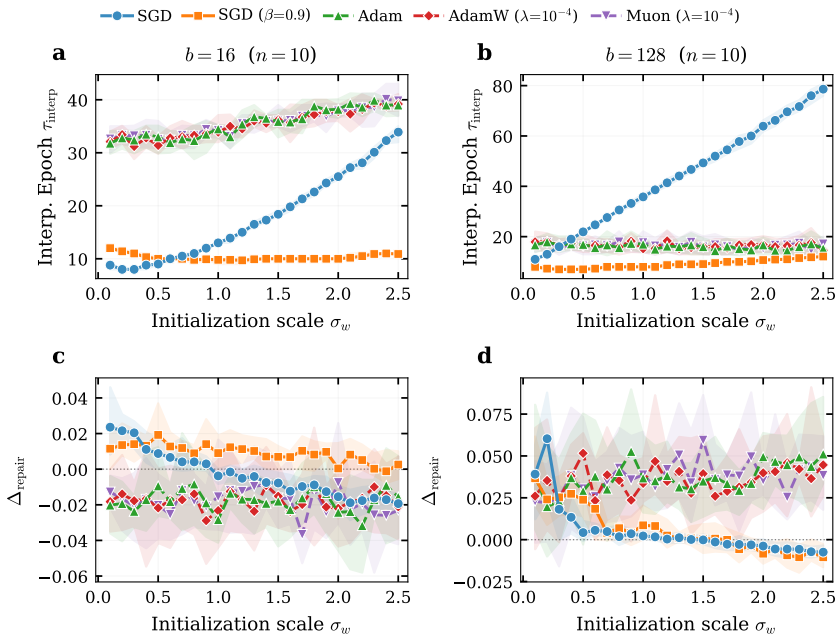


Figure 3: **Interpolation is not forgetting.** (a,b) Interpolation epoch τ_{interp} versus σ_w. SGD's interpolation time grows sharply with σ_w, especially at large batch sizes; Adam's is nearly flat. (c,d) Repair gap $\Delta_{\text{repair}} = \text{ValAcc}_{\tau_{\text{best}}} - \text{ValAcc}_{\tau_{\text{interp}}}$. At large batch ($b \geq 128$) Adam-family methods continue to gain validation accuracy after interpolation ($\tau_{\text{best}} \gg \tau_{\text{interp}}$, $\Delta_{\text{repair}} > 0$). At small batch ($b \leq 64$) they reach τ_{best} *before* interpolating, so the formal repair gap can be negative even though validation accuracy keeps improving (see Appendix. B). Vanilla SGD's repair gap stays near zero: validation accuracy is flat between τ_{best} and τ_{interp}, even as both grow with σ_w.

Metrics. We report test accuracy at the best-validation-loss checkpoint τ_{best}. We also track the interpolation epoch, $\tau_{\text{interp}} = \min\{t : \text{TrainAcc}_t \geq 99.5\%\}$, the trainable-kernel Frobenius norm $\|W\|_F = \left(\sum_\ell \|W^{(\ell)}\|_F^2\right)^{1/2}$, and the checkpoint repair gap, $\Delta_{\text{repair}} = \text{ValAcc}_{\tau_{\text{best}}} - \text{ValAcc}_{\tau_{\text{interp}}}$. The interpolation epoch measures when the labels are fit. The norm measures radial memory of the initial scale. The repair gap measures how validation accuracy changes between interpolation and the selected checkpoint. We define *initialization memory* 'Mem' operationally, as the dependence of the returned predictor on the initialization scale—the spread of its test accuracy across the σ_w sweep. This is distinct from, and does not imply, geometric erasure of the initial parameters: a small Mem means only that a seed-averaged performance metric varies little across the σ_w sweep; it does not establish that training has erased the initial parameter vector (its coordinates or its norm). We therefore read Mem together with absolute accuracy (a non-functional optimizer can show low spread at low accuracy), and we report explicit distance-from-initialization diagnostics—$\Delta(t) = \|W(t) - W_{\text{init}}\|_F$, $d(t) = \Delta(t)/\|W_{\text{init}}\|_F$, and direction $\cos(W(t), W_{\text{init}})$—in Appendix E.2.

3 Memory of Initialization: Optimizers and Hyperparameters

We now compare how much initialization-scale memory different procedures retain. The central contrast is that, under the shared low-learning-rate diagnostic procedure, vanilla SGD leaves σ_w visible at the selected checkpoint, whereas Adam, AdamW, and Muon largely erase it. Two diagnostics refine this comparison: the kernel norm tests whether the radial scale has been overwritten, and the checkpoint analysis separates fitting the labels from forgetting the initial scale.

3.1 SGD *vs.* Adam and Muon

Figure 1 exemplifies the basic optimizer contrast; Figure 2 extends it to the full $\sigma_w \times b$ grid.

Vision. *Vanilla SGD remembers where it started*: at $b = 128$ it reaches 99.5% training accuracy during training across the entire σ_w sweep, yet its test accuracy at the validation-selected checkpoint τ_{best} ranges from 85.1% at $\sigma_w = 0.1$ down to 58.6% at $\sigma_w = 2.5$, a 26.5 pp spread between two runs that, by training accuracy alone, appear to have learned the same data. *Momentum is only a partial cure*: at small batch it reduces the spread substantially (from 22.8 to 10.9 pp at $b = 16$), but at $b = 128$ under the same low-LR procedure it barely helps.

Adam-family methods largely wash out initialization scale: with the same data, learning rate, and epoch budget, Adam, AdamW, and Muon have b=128 spreads of only 4.3, 4.7, and 4.0 pp, respectively, and Adam's b=16 spread is 1.6 pp against SGD's 22.8 pp. The fact that *Adam itself* (no weight decay) matches AdamW and Muon (decoupled λ=10^{-4}) shows the effect is not driven by the optimizer-default weight decay (Appendix A).

Language. SGD-momentum has the largest val-loss spread (1.11) and stays near its initial norm scale ($r_f \approx 1$), while Adam reduces the dependence (0.44) and Muon is almost flat (0.06) (Table. 1). The norm ratios clarify the mechanism and match the radial picture of Fig. 1: Muon moves far from initialization, whereas Adam is robust yet retains radial scale at large σ_w ($r_f \to 1.04$)—it reduces the predictor's dependence on σ_w without contracting to one common final norm.

Weight Norm. The norm panels in Figure 1 (c,f) show the same distinction in parameter space. Under low-LR SGD, $\|W_{\tau_{\text{best}}}\|_F$ remains coupled to $\|W_{\text{init}}\|_F$: larger initial kernels lead to larger selected kernel norms. Over the main-grid range ($\sigma_w \leq 2.5$), Adam, AdamW, and Muon instead move different initial scales toward a common norm range; this is not a universal attractor, since at σ_w=10 Adam's final norm grows to 638 (Appendix E.2). We therefore read $\|W_{\tau_{\text{best}}}\|_F$ as a radial memory diagnostic, not as a measure of total distance traveled.

Table 1: GPT-2 (124M) on OpenWebText: validation loss and norm ratio $r_f = \|W_f\|_F / \|W_i\|_F$ across initialization scales (3 seeds). The SGD-vs-adaptive contrast from the vision grid persists: SGD-momentum has the largest val-loss spread and stays near its initial norm, while Adam and Muon are far flatter.

	SGD-momentum		Adam		Muon	
σ_w	val loss	r_f	val loss	r_f	val loss	r_f
0.5	4.48	1.20	3.56	3.09	3.35	29.5
1.0	4.24	1.05	3.56	1.59	3.34	10.1
2.0	4.41	1.01	3.64	1.17	3.34	7.45
4.0	5.35	1.00	4.00	1.04	3.40	3.86
spread	**1.11**	—	**0.44**	—	**0.06**	—

Thus, forgetting initialization is therefore not the event of fitting the labels. Under low-LR SGD, interpolation is reached before radial memory has been erased, and little subsequent repair occurs. The behavior of the Adam-family is batch-dependent: at large batch ($b \geq 128$) the best-validation-loss checkpoint is reached *after* interpolation, and the continued post-interpolation movement closes the σ_w gap; at small batches ($b \leq 64$) the best-validation-loss checkpoint is reached *before* interpolation, i.e., gap is already closed in the pre-interpolation phase (Figure 3, Appendix. B).

Message 1. *Under a shared low-learning-rate procedure, Adam(W) and Muon largely reduce initialization-scale memory, while SGD retains it.*

3.2 Dependence on Hyperparameters

In Figure 1, we use a deliberately diagnostic low-learning-rate SGD baseline, not a tuned SGD procedure. The natural objection is therefore simple: perhaps SGD only needs more time or better hyperparameters. In this section, we analyze how forgetting the initialization depends on the hyperparameters.

Figure 4 tests this directly with a targeted ResNet-9 sweep over training length, learning rate, and explicit L_2 regularization. See Appendix C for more details. Precisely, we see that:

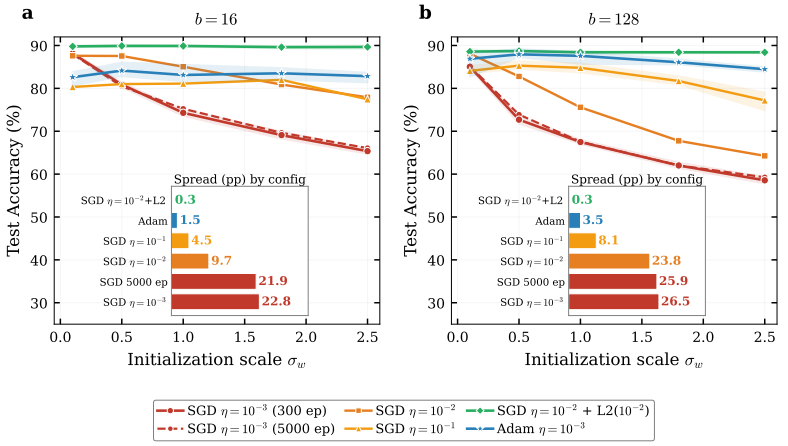


Figure 4: **What helps SGD forget initialization?** Test accuracy vs. σ_w for (a) b=16, and (b) b=128; inset shows the spread (pp) per configuration, sorted ascending. Long training alone leaves the curves nearly unchanged; larger learning rates and especially moderate LR with explicit L_2 sharply reduce the spread. Spreads use the 5-point grid $\sigma_w \in \{0.1, 0.5, 1.0, 1.8, 2.5\}$ [Table 8]; these agree with the 25-point spreads of §3 to within ~ 1 pp.

- **Weight decay and regularization.** Explicit regularization is most effective once the learning rate is large enough to move the iterate substantially. With $\eta_0 = 10^{-2}$ and cosine decay, adding $\lambda_{L_2} = 10^{-2}$ reduces the spread to 0.3 pp at both $b = 16$ and $b = 128$ (Table 8).
- **Learning rate.** Increasing the SGD learning rate helps, especially at a small batch. At $b = 16$, raising η_0 from 10^{-3} to 10^{-2} reduces the spread from 22.8 to 9.7 pp, and $\eta_0 = 10^{-1}$ reduces it to 4.5 pp. At $b = 128$, the same intervention is weaker: $\eta_0 = 10^{-2}$ still leaves 23.8 pp, and $\eta_0 = 10^{-1}$ leaves 8.1 pp. Large batches need stronger forgetting mechanisms.
- **Batch size.** Figure 2 extends the comparison to the full $\sigma_w \times b$ grid. We find that *large batch slows forgetting under a fixed-epoch budget.* For SGD, initialization sensitivity remains large across the entire range of batch sizes, with spreads of 22.8 pp at $b = 16$, 26.5 pp at $b = 128$, and 25.7 pp at $b = 256$; as b grows, the failure pattern also becomes more orderly and the absolute test-accuracy degradation more severe. Adaptive methods with the same low-LR training procedure still erase initialization memory more effectively than SGD, but their robustness weakens as the training regime becomes larger-batch and deeper. Adam, indeed, shows the same directional trend, but much more weakly: its spread increases from 1.6 pp at $b = 16$ to 5.3 pp at $b = 256$.
- **Normalization and augmentation.** Replacing BatchNorm with LayerNorm or adding standard data augmentation reduces the SGD spread but does not eliminate it; Adam remains robust under all these configurations (Appendix E). The adaptive-vs-SGD contrast itself does not require BatchNorm—it holds under both BatchNorm and LayerNorm—but graceful behavior at very large σ_w does depend on having *some* normalization: an unnormalized (NoNorm) ResNet loses it, with Adam falling from 84.9% at σ_w=1 to 21.8% at σ_w=10 (Appendix E.1).

The standard recipe removes the measured spread. Under a well-tuned SGD recipe (lr= 0.1, momentum= 0.9, weight decay= 5×10^{-4}, augmentation, 200 epochs), the test accuracies at the two tested endpoints $\sigma_w \in \{0.1, 2.5\}$ coincide at b=128 (94.2% at both; difference < 0.05 pp, i.e. 0.0 pp after rounding; Table 14, Appendix F). In this setting, the regularizers that produce strong generalization also remove the measurable initialization-scale dependence.

Depth changes the failure mode. Figure 5 extends the initialization sweep to ResNet-56, ResNet-110, and the pooling-matched R9-AvgPool control. The main effect is not a monotonic increase in spread with depth. At $b = 128$, the SGD spread is 26.5 pp for ResNet-9, 24.2 pp for ResNet-56, and 20.4 pp for ResNet-110. What changes more decisively is the absolute failure mode: ResNet-9 typically still interpolates and then generalizes poorly, whereas deeper ResNets increasingly fail to train well at large σ_w. Full numerical results (test and train accuracy, interpolation epochs) and the corresponding train accuracy figure [Figure 7] are provided in Appendix D. The regime, therefore, shifts from memorization without good generalization to degraded trainability itself.

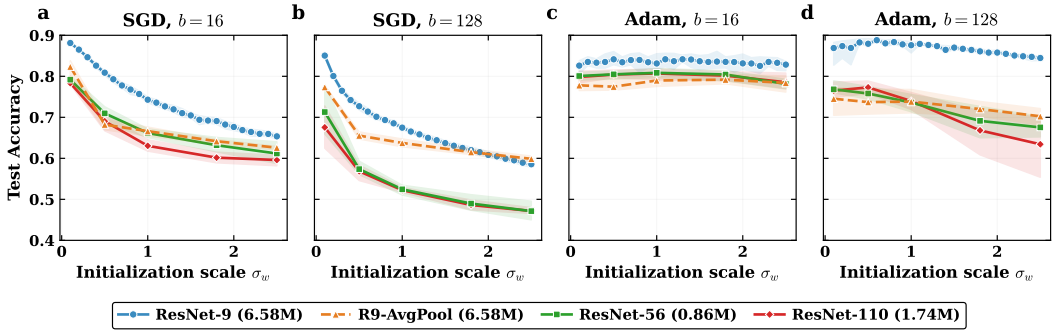


Figure 5: **Depth makes poor forgetting dynamics more damaging; pooling is a partial confound.** Test accuracy versus σ_w for ResNet-9, R9-AvgPool, ResNet-56, and ResNet-110. Switching pooling strategy reduces ResNet-9 performance, but the 9-layer R9 remains more robust than ResNet-56, suggesting that depth and optimization difficulty contribute beyond pooling alone.

SGD can forget, but only when the training procedure makes it forget. The repair map changes the interpretation of the optimizer comparison. The message is not that Adam is intrinsically capable of forgetting while SGD is not. Rather, Adam forgets automatically in this grid, while SGD forgets only when its training pipeline supplies both movement and radial control. Initialization robustness is therefore a property of the full training pipeline, not of the optimizer name alone.

Message 2. *Memory of initialization is, in general, a property of the whole training procedure, not a precise subset. Larger batch size, or smaller regularization or learning rate, increases initialization memory.*

4 The Time Scale of Forgetting

Sections 3 showed *what* is remembered and forgotten; this section asks *why*. The key idea is that epoch count is not the natural unit of forgetting: the relevant clock is the accumulated strength of the regularization—implicit or explicit—that acts along the trajectory.

4.1 The time scale is regularization

Backward error view. Backward error analysis views a discrete optimizer as approximately following a modified continuous-time dynamics [36, 37]. In ML, this viewpoint writes the discrete update as a gradient flow on a modified objective

$$\mathcal{L}_{\mathrm{mod}} \;=\; \mathcal{L} \;+\; \rho_{\mathcal{A}}(\eta, b, \lambda, \ldots)\, \mathcal{R}_{\mathcal{A}} \;+\; \text{higher-order terms},$$

where $\mathcal{R}_{\mathcal{A}}$ is the implicit or explicit regularizer associated with algorithm $\mathcal{A}$: implicit gradient regularization for finite-step GD [32], covariance/Fisher-type regularization for minibatch SGD [33, 34], momentum-amplified regularization [38], and geometry-dependent implicit bias for adaptive methods [39]. What matters is not $\rho_{\mathcal{A}}$ itself, but its accumulated strength along the trajectory: the events that erase initialization-dependent quantities are exactly the perturbation terms of $\mathcal{L}_{\mathrm{mod}}$ accumulating over many steps.

Forgetting timescales. For the mechanisms our experiments isolate, the leading cumulative scales along a schedule $(\eta_k)_{k<K}$ are

$$\mathcal{T}_{\mathrm{SGD}} = \frac{1}{b}\sum_{k<K}\eta_k^2, \qquad \mathcal{T}_{L_2} = \lambda\sum_{k<K}\eta_k, \qquad \mathcal{T}_{\mathrm{adapt}} = \sum_{k<K}\eta_k.$$

The first scale corresponds to the minibatch finite-step/covariance effect: the η^2/b scaling is the cumulative strength of the SGD modified-loss correction [33, 34]. The second is explicit radial norm decay, which contracts initialization-dependent norm at rate $\lambda\eta$ per step. The third is the first-order adaptive transport scale: adaptive preconditioners do not preserve the symmetries that make the gradient-flow imbalance conserved, so they can move initialization-dependent quantities at order η per step [39]. For constant learning rate these reduce to $K\eta^2/b$, $K\eta\lambda$, and $K\eta$. We use these

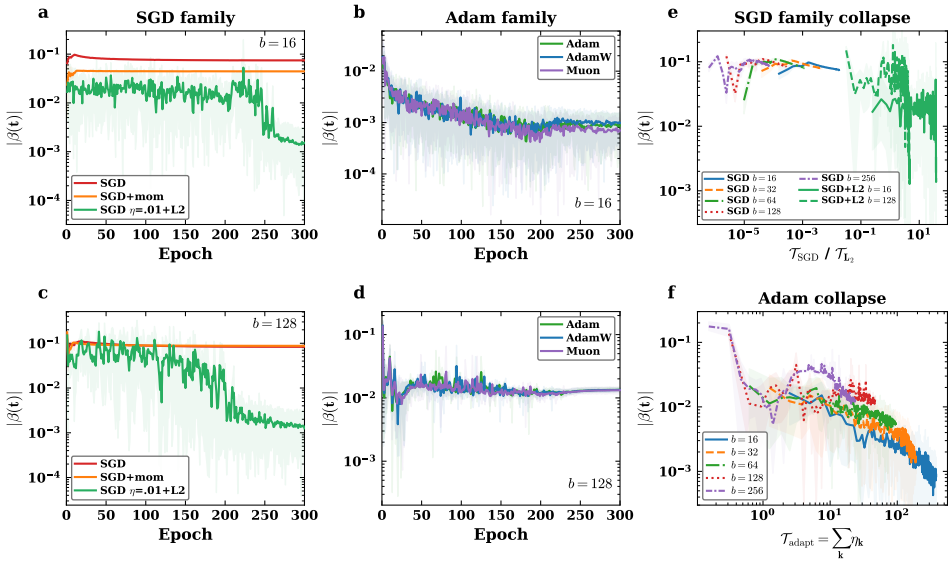


Figure 6: **Initialization sensitivity decays on regularization timescales, not epoch count.** Here $|\beta(t)| = |d\,\mathrm{ValAcc}/d\sigma_w|$ is the *initialization sensitivity*: the per-epoch slope of validation accuracy with respect to the initialization scale (larger = more memory), estimated by ordinary least squares over the σ_w sweep. (a–d) $|\beta(t)|$ versus epoch for the SGD family (left) and adaptive methods (right) at b=16 (top) and b=128 (bottom). Vanilla SGD (red) and SGD+momentum (orange) stay flat; Adam/AdamW/Muon decay steadily; SGD with η=10^{-2}, L_2=10^{-2} (green) reaches the adaptive noise floor. (e) Vanilla SGD $|\beta|$ versus $\mathcal{T}_{\mathrm{SGD}}$=$(1/b)\sum_k \eta_k^2$ across all five batch sizes (colored lines); SGD+L_2 versus $\mathcal{T}_{L_2}$=$\lambda\sum_k \eta_k$ at b=16, 128 (green). (f) Adam $|\beta|$ versus $\mathcal{T}_{\mathrm{adapt}}$=$\sum_k \eta_k$ across all five batch sizes. Curves approximately collapse under the proposed rescaling.

expressions as an organizing principle for the experiments, not as a complete theorem for nonlinear BatchNorm ResNets.

4.2 From timescales to data

A linear-network sanity check. These timescales are visible in the smallest nontrivial homogeneous network: the two-parameter scalar problem $(abx - y)^2$ with $a, b \in \mathbb{R}$. (Appendix G proves the conservation, finite-step, L_2, and adaptive-preconditioning statements below for two-factor and deep linear networks). Gradient flow preserves the imbalance $D = a^2 - b^2$ exactly, so the norm at convergence depends on initialization. Explicit L_2 decay kills D at rate $e^{-2\lambda t}$, giving a timescale $O(\lambda\eta K)$. Discrete minibatch SGD breaks the conservation at second order in η; the batch-dependent part of the leakage scales as $O(\eta^2 K/b)$ [35, 40, 41]. Adaptive preconditioning, by contrast, generically breaks this first-order cancellation; the resulting $O(\eta K)$ clock for adaptive methods is proved in our minimal model (Appendix G) and is suggested by the implicit-bias analyses of [38, 39]. In the regimes studied here, the effective clocks are ordered roughly as

$$\mathcal{T}_{\mathrm{SGD}} \ll \mathcal{T}_{L_2} \lesssim \mathcal{T}_{\mathrm{adapt}},$$

once explicit L_2 is large enough to matter. This ordering matches the observed forgetting hierarchy: low-LR SGD remembers, SGD with sufficient L_2 forgets, and adaptive methods forget fastest under the diagnostic grid. In other words, the formulas explain the empirical hierarchy without requiring a universal ordering: low-LR SGD has a tiny $\mathcal{T}_{\mathrm{SGD}}$, explicit L_2 becomes effective when $\lambda\sum_k \eta_k$ is order one or larger, and adaptive methods have a first-order movement clock $\sum_k \eta_k$.

Numerical check. The timescale arithmetic explains the repair map of Section 3.2. Ignoring cosine decay and using $K \approx E\, n_{\mathrm{train}}/b$, low-LR SGD at b=128 has $\mathcal{T}_{\mathrm{SGD}} \approx 7.3\cdot 10^{-4}$ after 300 epochs and only $\approx 1.2\cdot 10^{-2}$ after 5,000 epochs; both regimes retain $\sim$26 pp of spread. At b=16 and η=10^{-2}, the same timescale reaches ≈ 4.7 and the spread drops sharply. Explicit L_2 shows the same threshold: at b=128, η=10^{-2} with λ=10^{-3} gives $\mathcal{T}_{L_2} \approx 0.94$ and still leaves 17.4 pp, whereas λ=5×10^{-3} gives $\mathcal{T}_{L_2} \approx 4.7$ and collapses the spread below 1 pp. Adaptive methods erase initialization-scale dependence without the SGD-style sweep, consistent with the $K\eta$ scaling of $\mathcal{T}_{\mathrm{adapt}}$.

Empirical $|\beta(t)|$ collapse. If these timescales are the right dynamical clocks, then initialization sensitivity at different batch sizes should collapse when plotted against $\mathcal{T}$ rather than epoch count. We test this with $|\beta(t)| = |d\,\mathrm{ValAcc}/d\,\sigma_w|$ at each epoch, estimated by ordinary least squares regression across the 25-point σ_w sweep (10 seeds).

Figure 6(a–d) shows $|\beta|$ versus epoch: vanilla SGD stays flat near 10^{-1}; Adam, AdamW, and Muon decay by 1–2 orders of magnitude; the SGD ablation with $\eta{=}10^{-2}$ and $L_2{=}10^{-2}$ reaches the adaptive noise floor. Panels (e–f) replot against the cumulative timescale $\mathcal{T}$ for all five batch sizes ($b \in \{16, 32, 64, 128, 256\}$): under $\mathcal{T}_{\mathrm{SGD}}$ the vanilla-SGD curves approximately collapse, and adding SGD with L_2 (plotted against $\mathcal{T}_{L_2}$) shows the same decay trajectory as the adaptive methods. Panel (f) confirms that Adam curves at all five batch sizes align under $\mathcal{T}_{\mathrm{adapt}}$. The collapse is not exact—$|\beta|$ is a noisy estimator and the timescale derivations assume linear, scale-invariant dynamics—but the qualitative alignment across a $16\times$ range of batch sizes is clear.

Message 3. *The right unit of forgetting is accumulated regularization, not epoch count. Initialization memory gives an empirical readout of implicit and explicit regularization: the optimizer is not only fitting the data, but also deciding how quickly the initial condition stops mattering.*

5 Conclusion

An apparent paradox, and its resolution. A line of work argues that the parameter–function map of deep networks is strongly biased toward simple functions, and that this architectural prior already accounts for generalization [6, 7, 42]. Our results sharpen this picture in a way that initially looks paradoxical: the training regimes that preserve initialization bias (low-LR SGD) leave generalization fragile to the initialization, while those that erase it (Adam, well-tuned SGD with weight decay) generalize well across the tested scales. The standard ResNet recipe ($\eta{=}0.1$, momentum, $\lambda{=}5{\times}10^{-4}$, augmentation) gives the same test accuracy, 94.2%, at both tested endpoints $\sigma_w \in \{0.1, 2.5\}$ at $b{=}128$, among the best in our study (Appendix F). In our settings, the Occam's razor that matters is therefore not the one built into the parameter–function map at initialization; it is the one imposed by the regularizers accumulated along the trajectory—and the mechanisms that close the σ_w gap tend to be the ones that produce good generalization.

Implications, limitations, and future work. Our controlled study (6,250+ runs across optimizers, batch sizes, depths, regularizers, normalizations, and training lengths) shows that interpolation does not imply forgetting, long, low-LR training does not erase memory, and that forgetting requires accumulated regularization on optimizer-dependent timescales ($K\eta^2/b$, $K\eta\lambda$, $K\eta$). A practical, somewhat counterintuitive consequence is that designing initialization *scales* purely "for predictor performance" brings little payoff once the pipeline includes sufficient regularization: any initialization-scale advantage is erased on the same timescale as the one that delivers generalization. The initialization design remains valuable as a *trainability* device—stabilizing optimization, enabling depth, supporting shorter training horizons and controlling the signal propagation—but it should not be expected to leave a persistent fingerprint under modern, regularization-rich procedures. Conversely, in gradient-flow-like regimes (small η, large b, no weight decay) initialization-induced bias does survive, and changing the initialization scale becomes a real lever on the final predictor.

Our core grid is BatchNorm ResNets on CIFAR-10, with LayerNorm/NoNorm ablations (Appendix E) and a GPT-2/OpenWebText pretraining study as initial generality checks, and the timescale expressions are organizing principles derived from linear and scale-invariant arguments rather than full nonlinear theorems—limitations shared with the implicit-bias literature we build on [32–34, 39]. Our σ_w probe is also one-dimensional, leaving bias distributions, layer-wise scalings, and μP parameterizations unstudied. The most pressing extensions are to larger-scale settings (ImageNet, ViTs, and—building on our GPT-2 study—larger-scale LLM pretraining and fine-tuning), where recent seed-dependence results—across pretraining [25–28] and fine-tuning [24]—suggest initialization memory may be both measurable and consequential, and to tighten the backward-error framework into a quantitative theory of forgetting time for homogeneous and approximately scale-invariant networks.

Acknowledgments

MD and GHM acknowledge financial support from 3M; PB and TP are supported by McGovern Institute of Brain Research, TP is supported by McDermott Endowed Chair. The authors declare no competing interests.

References

[1] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[2] Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Westview Press, 2 edition, 2015.

[3] Morris W. Hirsch, Stephen Smale, and Robert L. Devaney. *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. Academic Press, 3 edition, 2013.

[4] Hancheng Min, Salma Tarmoun, René Vidal, and Enrique Mallada. On the explicit role of initialization on the convergence and implicit bias of overparametrized linear networks. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7760–7768. PMLR, 2021.

[5] Oria Gruber and Haim Avron. On the role of initialization on the implicit bias in deep linear networks, 2024.

[6] Guillermo Valle-Pérez, Chico Q. Camargo, and Ard A. Louis. Deep learning generalizes because the parameter–function map is biased towards simple functions. In *International Conference on Learning Representations*, 2019.

[7] Chris Mingard, Henry Rees, Guillermo Valle-Pérez, and Ard A. Louis. Deep neural networks have an inbuilt occam's razor. *Nature Communications*, 16:220, 2025.

[8] Thomas Fink. Deep-layered machines have a built-in occam's razor. *arXiv preprint arXiv:2603.01217*, 2026.

[9] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256. PMLR, 2010.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[11] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[12] Samuel S. Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. In *International Conference on Learning Representations*, 2017.

[13] Jeffrey Pennington, Samuel S. Schoenholz, and Surya Ganguli. Resurrecting the sigmoid in deep learning through dynamical isometry: Theory and practice. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[14] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[15] Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[16] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of CNNs: How to train 10,000-layer vanilla convolutional neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5393–5402. PMLR, 2018.

[17] Minmin Chen, Jeffrey Pennington, and Samuel S. Schoenholz. Dynamical isometry and a mean field theory of RNNs: Gating enables signal propagation in recurrent neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 873–882. PMLR, 2018.

[18] Greg Yang and Edward J. Hu. Tensor programs IV: Feature learning in infinite-width neural networks. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11727–11737. PMLR, 2021.

[19] Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs V: Tuning large neural networks via zero-shot hyperparameter transfer. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[20] Blake Bordelon and Cengiz Pehlevan. Self-consistent dynamical field theory of kernel evolution in wide neural networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2023(11):114009, 2023.

[21] Blake Bordelon and Cengiz Pehlevan. Dynamics of finite width kernel and prediction fluctuations in mean field neural networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2024(10):104021, 2024.

[22] Blake Bordelon and Cengiz Pehlevan. Deep linear network training dynamics from random initialization: Data, width, depth, and hyperparameter transfer. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 4968–4997. PMLR, 2025.

[23] Clarissa Lauditi, Blake Bordelon, and Cengiz Pehlevan. Adaptive kernel predictors from feature-learning infinite limits of neural networks. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 32617–32648. PMLR, 2025.

[24] Jesse Dodge, Gabriel Ilharco, Roy Schwartz, Ali Farhadi, Hannaneh Hajishirzi, and Noah A. Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *CoRR*, abs/2002.06305, 2020.

[25] Oskar van der Wal, Pietro Lesci, Max Müller-Eberstein, Naomi Saphra, Hailey Schoelkopf, Willem H. Zuidema, and Stella R. Biderman. PolyPythias: Stability and outliers across fifty language model pre-training runs. In *International Conference on Learning Representations*, 2025.

[26] Finlay Fehlauer, Kyle Mahowald, and Tiago Pimentel. Convergence and divergence of language models under different random seeds. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 32982–32991, Suzhou, China, 2025. Association for Computational Linguistics.

[27] Yao Tong, Haonan Wang, Siquan Li, Kenji Kawaguchi, and Tianyang Hu. SeedPrints: Fingerprints can even tell which seed your large language model was trained from. In *International Conference on Learning Representations*, 2026. Poster.

[28] Siquan Li, Yao Tong, Haonan Wang, and Tianyang Hu. Transformers are born biased: Structural inductive biases at random initialization and their practical consequences, 2026.

[29] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.1, knowledge storage and extraction. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 1067–1077. PMLR, 2024.

[30] Zeyuan Allen-Zhu. Physics of language models: Part 4.1, architecture design and the magic of canon layers. In *Proceedings of the 39th Conference on Neural Information Processing Systems*, NeurIPS '25, 2025. Full version available at `https://ssrn.com/abstract=5240330`.

[31] Zhiyuan Li, Kaifeng Lyu, and Sanjeev Arora. Reconciling modern deep learning with traditional optimization analyses: The intrinsic learning rate. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[32] David G. T. Barrett and Benoit Dherin. Implicit gradient regularization. In *International Conference on Learning Representations*, 2021.

[33] Samuel L. Smith, Benoit Dherin, David G. T. Barrett, and Soham De. On the origin of implicit regularization in stochastic gradient descent. In *International Conference on Learning Representations*, 2021.

[34] Pierfrancesco Beneventano. On the trajectories of sgd without replacement. *arXiv preprint arXiv:2312.16143*, 2023.

[35] Pierfrancesco Beneventano, Andrea Pinto, and Tomaso Poggio. How neural networks learn the support is an implicit regularization effect of SGD. 2024.

[36] David F. Griffiths and J. M. Sanz-Serna. On the scope of the method of modified equations. *SIAM Journal on Scientific and Statistical Computing*, 7(3):994–1008, 1986.

[37] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*, volume 31 of *Springer Series in Computational Mathematics*. Springer, Berlin, Heidelberg, 2 edition, 2006.

[38] Avrajit Ghosh, He Lyu, Xitong Zhang, and Rongrong Wang. Implicit regularization in heavy-ball momentum accelerated stochastic gradient descent. *arXiv preprint arXiv:2302.00849*, 2023.

[39] Matias D. Cattaneo, Jason Matthew Klusowski, and Boris Shigida. On the implicit bias of Adam. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 5862–5906. PMLR, 2024.

[40] Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[41] Pierfrancesco Beneventano and Blake Woodworth. Gradient descent converges linearly to flatter minima than gradient flow in shallow linear networks. *arXiv preprint arXiv:2501.09137*, 2025.

[42] Chris Mingard, Joar Skalse, Guillermo Valle-Pérez, David Martínez-Rubio, Vladimir Mikulik, and Ard A. Louis. Neural networks are a priori biased towards boolean functions with low entropy, 2020.

[43] Vladimir N. Vapnik and Alexey Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16(2):264–280, 1971.

[44] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[45] Peter L. Bartlett. The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536, 1998.

[46] Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482, 2002.

[47] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Proceedings of The 28th Conference on Learning Theory*, volume 40 of *Proceedings of Machine Learning Research*, pages 1376–1401. PMLR, 2015.

[48] Peter L. Bartlett, Dylan J. Foster, and Matus J. Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[49] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

[50] Devansh Arpit, Stanislaw Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 233–242. PMLR, 2017.

[51] Preetum Nakkiran, Gal Kaplun, Dimitris Kalimeris, Tristan Yang, Benjamin L. Edelman, Fred Zhang, and Boaz Barak. SGD on neural networks learns functions of increasing complexity. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[52] Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1225–1234. PMLR, 2016.

[53] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[54] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. A PAC-bayesian approach to spectrally-normalized margin bounds for neural networks. In *International Conference on Learning Representations*, 2018.

[55] Yiding Jiang, Dilip Krishnan, Hossein Mobahi, and Samy Bengio. Predicting the generalization gap in deep networks with margin distributions. In *International Conference on Learning Representations*, 2019.

[56] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2020.

[57] Mengjia Xu, Akshay Rangamani, Qianli Liao, Tomer Galanti, and Tomaso Poggio. Dynamics in deep classifiers trained with the square loss: Normalization, low rank, neural collapse, and generalization bounds. *Research*, 6:0024, 2023.

[58] Tomer Galanti, Mengjia Xu, Liane Galanti, and Tomaso Poggio. Norm-based generalization bounds for sparse neural networks. *Advances in Neural Information Processing Systems*, 36:42482–42501, 2023.

[59] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[60] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017.

[61] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1019–1028. PMLR, 2017.

[62] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. In *International Conference on Learning Representations*, 2021.

[63] David A. McAllester. PAC-bayesian model averaging. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 164–170, 1999.

[64] Gintare Karolina Dziugaite and Daniel M. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*, 2017.

[65] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P. Adams, and Peter Orbanz. Non-vacuous generalization bounds at the ImageNet scale: A PAC-bayesian compression approach. In *International Conference on Learning Representations*, 2019.

[66] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International Conference on Learning Representations*, 2018.

[67] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[68] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations*, 2020.

[69] Peter L. Bartlett, Philip M. Long, Gábor Lugosi, and Alexander Tsigler. Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 117(48):30063–30070, 2020.

[70] Radford M. Neal. *Bayesian Learning for Neural Networks*, volume 118 of *Lecture Notes in Statistics*. Springer, 1996.

[71] Christopher K. I. Williams. Computing with infinite networks. In *Advances in Neural Information Processing Systems*, volume 9, 1996.

[72] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. In *International Conference on Learning Representations*, 2018.

[73] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[74] Kamaludin Dingle, Chico Q. Camargo, and Ard A. Louis. Input–output maps are strongly biased towards simple outputs. *Nature Communications*, 9:761, 2018.

[75] Giacomo De Palma, Bobak Kiani, and Seth Lloyd. Random deep neural networks are biased towards simple functions. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[76] Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.

[77] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[78] Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, 3 edition, 2008.

[79] Satwik Bhattamishra, Arkil Patel, Varun Kanade, and Phil Blunsom. Simplicity bias in transformers and their ability to learn sparse Boolean functions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5767–5791, Toronto, Canada, 2023. Association for Computational Linguistics.

[80] Michael Hahn and Mark Rofin. Why are sensitive functions hard for transformers? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14973–15008, Bangkok, Thailand, 2024. Association for Computational Linguistics.

[81] Bhavya Vasudeva, Deqing Fu, Tianyi Zhou, Elliott Kau, Youqi Huang, and Vatsal Sharan. Transformers learn low sensitivity functions: Investigations and implications. In *International Conference on Learning Representations*, 2025.

[82] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5301–5310. PMLR, 2019.

[83] Roman Novak, Yasaman Bahri, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: An empirical study. In *International Conference on Learning Representations*, 2018.

[84] Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2596–2604. PMLR, 2019.

[85] Boris Hanin and David Rolnick. Deep ReLU networks have surprisingly few activation patterns. In *Advances in Neural Information Processing Systems*, volume 32, pages 359–368, 2019.

[86] Benoit Dherin, Michael Munn, Mihaela Rosca, and David G. T. Barrett. Why neural networks find simple solutions: The many regularizers of geometric complexity. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[87] Maria Refinetti, Alessandro Ingrosso, and Sebastian Goldt. Neural networks trained with SGD learn distributions of increasing complexity. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 28843–28863. PMLR, 2023.

[88] Etienne Boursier and Nicolas Flammarion. Simplicity bias and optimization threshold in two-layer ReLU networks, 2024.

[89] Yedi Zhang, Andrew M. Saxe, and Peter E. Latham. Saddle-to-saddle dynamics explains a simplicity bias across neural network architectures. In *International Conference on Learning Representations*, 2026. Poster.

[90] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations*, 2014.

[91] Yaniv Blumenfeld, Dar Gilboa, and Daniel Soudry. Beyond signal propagation: Is feature diversity necessary in deep neural network initialization? In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 960–969. PMLR, 2020.

[92] Greg Yang. Tensor programs I: Wide feedforward or recurrent neural networks of any architecture are gaussian processes, 2020.

[93] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456. PMLR, 2015.

[94] Sanjeev Arora, Kaifeng Lyu, and Zhiyuan Li. Theoretical analysis of auto rate-tuning by batch normalization. In *International Conference on Learning Representations*, 2019.

[95] Zhiyuan Li, Kaifeng Lyu, and Sanjeev Arora. Reconciling modern deep learning with traditional optimization analyses: The intrinsic learning rate. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[96] James Hardy Wilkinson. *Rounding Errors in Algebraic Processes*. Number 32 in Notes on Applied Science. Her Majesty's Stationery Office, London, 1963.

[97] James Hardy Wilkinson. *The Algebraic Eigenvalue Problem*. Monographs on Numerical Analysis. Clarendon Press, Oxford, 1965.

[98] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2 edition, 2002.

[99] M. P. Calvo, Ander Murua, and J. M. Sanz-Serna. Modified equations for ODEs. In Peter E. Kloeden and Kenneth J. Palmer, editors, *Chaotic Numerics*, volume 172 of *Contemporary Mathematics*, pages 63–74. American Mathematical Society, Providence, RI, 1994.

[100] Tony Shardlow. Modified equations for stochastic differential equations. *BIT Numerical Mathematics*, 46(1):111–125, 2006.

[101] Konstantinos C. Zygalakis. On the existence and the applications of modified equations for stochastic differential equations. *SIAM Journal on Scientific Computing*, 33(1):102–130, 2011.

[102] Arnaud Debussche and Erwan Faou. Weak backward error analysis for SDEs. *SIAM Journal on Numerical Analysis*, 50(3):1735–1752, 2012.

[103] Yuanyuan Feng, Tingran Gao, Lei Li, Jian-Guo Liu, and Yulong Lu. Uniform-in-time weak error analysis for stochastic gradient descent algorithms via diffusion approximation. *Communications in Mathematical Sciences*, 18(1):163–188, 2020.

[104] Qianxiao Li, Cheng Tai, and Weinan E. Stochastic modified equations and adaptive stochastic gradient algorithms. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2101–2110. PMLR, 2017.

[105] Qianxiao Li, Cheng Tai, and Weinan E. Stochastic modified equations and dynamics of stochastic gradient algorithms I: Mathematical foundations. *Journal of Machine Learning Research*, 20(40):1–47, 2019.

[106] Taiki Miyagawa. Toward equation of motion for deep neural networks: Continuous-time gradient descent and discretization error analysis. In *Advances in Neural Information Processing Systems*, volume 35, pages 37778–37791, 2022.

[107] Mihaela Rosca, Yan Wu, Chongli Qin, and Benoit Dherin. On a continuous time model of gradient descent dynamics and instability in deep learning. *arXiv preprint arXiv:2302.01952*, 2023.

[108] Matias D Cattaneo and Boris Shigida. Modified loss of momentum gradient descent: Fine-grained analysis. *arXiv preprint arXiv:2509.08483*, 2025.

[109] Stefano Di Giovacchino, Desmond J. Higham, and Konstantinos C. Zygalakis. Backward error analysis and the qualitative behaviour of stochastic optimization algorithms: Application to stochastic coordinate descent. *Journal of Computational Dynamics*, 11(4):453–467, 2024.

[110] Matias Cattaneo and Boris Shigida. How memory in optimization algorithms implicitly modifies the loss. *Advances in Neural Information Processing Systems*, 38:156059–156096, 2026.

[111] Matias D Cattaneo and Boris Shigida. The effect of mini-batch noise on the implicit bias of adam. *arXiv preprint arXiv:2602.01642*, 2026.

[112] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *Journal of Machine Learning Research*, 19(70):1–57, 2018.

[113] Suriya Gunasekar, Jason D. Lee, Daniel Soudry, and Nathan Srebro. Implicit bias of gradient descent on linear convolutional networks. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[114] Kaifeng Lyu and Jian Li. Gradient descent maximizes the margin of homogeneous neural networks. In *International Conference on Learning Representations*, 2020.

[115] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[116] Stephan Mandt, Matthew D. Hoffman, and David M. Blei. Stochastic gradient descent as approximate bayesian inference. *Journal of Machine Learning Research*, 18(134):1–35, 2017.

[117] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

A Notation and Experimental Details

A.1 Notation

Table 2: **Notation.** Symbols used throughout the paper.

Symbol	Meaning	Symbol	Meaning
b	Mini-batch size.	Mem	Init. memory: predictor spread over σ_w.
σ_w	Initialization scale multiplier.	τ_ε	Forgetting time: min K with $\mathsf{Mem} \le \varepsilon$.
η_k	Learning rate at step k.	$\mathcal{T}_{\mathrm{SGD}}$	SGD-noise clock $\frac{1}{b}\sum_k \eta_k^2$.
λ	L_2 regularization strength.	$\mathcal{T}_{L_2}$	Explicit-L_2 clock $\lambda \sum_k \eta_k$.
K	Number of parameter updates.	$\mathcal{T}_{\mathrm{adapt}}$	Adaptive clock $\sum_k \eta_k$.
W	Trainable kernel tensors.	$\rho_{\mathcal{A}}$	Per-step reg. coeff. of $\mathcal{A}$.
$\|W\|_F$	Kernel Frobenius norm (Fig. 1).	$\mathcal{R}_{\mathcal{A}}$	Regularizer (implicit/explicit) of $\mathcal{A}$.
μP	Maximal-update parameterization.	$R_{\mathcal{A}}$	Accumulated $\sum_k \rho_{\mathcal{A}}$ along training.

A.2 Experimental Details

This appendix records the setups for all experiments. The CIFAR-10 ResNet experiments are implemented in Keras 3 / TensorFlow on a single NVIDIA GPU per run (H100, H200, or L40S); the GPT-2 experiments use nanoGPT (PyTorch). A summary of the main-grid optimizers is given in Table 3; per-experiment paragraphs follow.

Common ingredients (all CIFAR-10 ResNet experiments).

- *Data.* CIFAR-10 with the standard $50{,}000$ training and $10{,}000$ test images. We hold out a fixed $10{,}000$-image validation set from the training partition using permutation seed 42, giving a $40{,}000/10{,}000/10{,}000$ train/validation/test split that is identical across every experiment. Inputs are scaled to $[0, 1]$.
- *Initialization.* Convolutional and dense kernels are drawn from $W_{ij} \sim \mathcal{N}\big(0, \sigma_w^2/\mathrm{fan_{in}}\big)$, so $\sigma_w = 1$ corresponds to a fan-in normal baseline. Biases are drawn from $\mathcal{N}(0, 0.2^2)$ and are not rescaled by σ_w.
- *Loss.* Sparse categorical cross-entropy from logits.
- *Checkpoint selection.* For each run we save and report metrics at the validation-loss-minimizing epoch τ_{best}, and record $\tau_{\mathrm{interp}} = \min\{t : \mathrm{TrainAcc}_t \ge 99.5\%\}$ (except in Table 9, which states its own architecture-specific threshold) and the kernel Frobenius norm $\|W\|_F$.
- *Reproducibility.* Each run sets Python, NumPy, and TensorFlow random seeds to the same integer.

Main ResNet-9 grid (§3, App. B). ResNet-9 with BatchNorm, no augmentation, 300 epochs. Learning rate follows the cosine schedule $\eta_k = \eta_0\big[\alpha + (1-\alpha)\frac{1}{2}(1 + \cos(\pi k/K))\big]$ with $\eta_0 = 10^{-3}$, $\alpha = 0.01$, and total updates $K = 300\,\lceil 40{,}000/b \rceil$. The grid varies three axes: $\sigma_w \in \{0.10, 0.20, \ldots, 2.50\}$ (25 values), optimizer in {SGD, SGD-momentum, Adam, AdamW, Muon} (Table 3), and batch size $b \in \{16, 32, 64, 128, 256\}$. Each of the $25 \times 5 \times 5 = 625$ configurations is run for 10 seeds $\{0, 1, 42, 123, 456, 789, 999, 2024, 2025, 2026\}$, for a total of $6{,}250$ trained models.

$5{,}000$**-epoch SGD (§3, §4).** Vanilla SGD ($\eta = 10^{-3}$, momentum 0) trained for $5{,}000$ epochs *with constant learning rate* (no cosine decay), no augmentation, no weight decay or L_2. Five scales $\sigma_w \in \{0.1, 0.5, 1.0, 1.8, 2.5\}$, batch sizes $b \in \{16, 128\}$, single seed experiment.

LR$\times L_2$ **sweep for SGD (§3.2, Table 8).** Vanilla SGD (no momentum), 300 epochs, cosine schedule. Twelve configurations $\eta \in \{10^{-3}, 10^{-2}, 10^{-1}\}$ paired with explicit kernel L_2 regularization $\lambda \in \{0, 10^{-3}, 5\times10^{-3}, 10^{-2}\}$, at five scales $\sigma_w \in \{0.1, 0.5, 1.0, 1.8, 2.5\}$ and batch sizes $b \in \{16, 128\}$.

Norm and augmentation (App. E, Fig. 8). Two normalization variants – BatchNorm, LayerNorm – under standard data augmentation (random horizontal flip + 4-pixel reflective padding + random 32×32 crop). 300 epochs cosine, $\eta = 10^{-3}$, no weight decay. Two optimizers (SGD without momentum, Adam), $\sigma_w \in \{0.1, 0.5, 1.0, 1.8, 2.5\}$, $b \in \{16, 128\}$, 5 seeds.

Best-procedure comparison (App. F, Table 14). 200 **epochs**, cosine schedule, with augmentation. Two procedures: SGD ($\eta = 0.1$, momentum 0.9, weight decay 5×10^{-4}) and Adam ($\eta = 10^{-3}$, no weight decay), at the two extreme scales $\sigma_w \in \{0.1, 2.5\}$ and batch sizes $b \in \{16, 128, 256\}$, 10 seeds.

Depth (App. D, Fig. 5). ResNet-56, ResNet-110 (CIFAR-style architectures, $6n + 2$ layers with $n = 9$ and $n = 18$ respectively), and an R9-AvgPool control that replaces ResNet-9's global max-pool with global average-pool. All use 300 epochs cosine, $\eta_0 = 10^{-3}$, the five main-grid optimizers, $\sigma_w \in \{0.1, 0.5, 1.0, 1.8, 2.5\}$, and $b \in \{16, 128\}$, with 5 seeds (ResNet-9 reference values come from the 10-seed main grid). ResNet-56 and ResNet-110 use early stopping with patience 30 (without altering the cosine schedule); R9-AvgPool does not.

GPT-2 pretraining (§2, Table 1). GPT-2 (124M parameters; 12 layers, 12 heads, width 768, pre-LayerNorm) implemented in nanoGPT and trained from scratch on OpenWebText (GPT-2 BPE tokenizer) for a fixed budget of 600M tokens (≈ 4575 iterations, $\approx 131{,}072$ tokens per iteration). Initialization follows nanoGPT: all normal-initialized weights (token/position embeddings and all linear layers) use standard deviation 0.02, residual output projections use $0.02/\sqrt{2L}$, and biases are zero. The scale multiplier σ_w rescales every one of these standard deviations, so $\sigma_w = 1$ is the standard initialization; LayerNorm parameters are not rescaled. We sweep $\sigma_w \in \{0.5, 1.0, 2.0, 4.0\}$ with 3 seeds each and compare three optimizers, all with weight decay 0: SGD with momentum 0.9 ($\eta = 0.3$), Adam ($\eta = 6 \times 10^{-4}$), and Muon ($\eta = 1.2 \times 10^{-2}$, momentum 0.95). All three use linear warmup over $\max(100, \lfloor K/20 \rfloor)$ iterations followed by cosine decay to $\eta/10$ at iteration $K \approx 4575$; exact per-cell commands are given by `experiments/llm_pretrain/sweep.py` in the released code. We report the final validation loss and the final-to-initial norm ratio $r_f = \|W_f\|_F / \|W_i\|_F$.

Extended sweep and SGD+L_2 comparisons (Tables 11, 12, 13). BatchNorm and unnormalized ResNet-9, b=128, flip+crop augmentation, 300 epochs, cosine decay to $0.01\,\eta_0$, 5 seeds, $\sigma_w \in \{0.1, 0.5, 1.0, 1.8, 2.5, 3.5, 5.0, 10.0\}$. SGD: $\eta_0 = 10^{-3}$, no momentum, no regularization. Adam: $\eta_0 = 10^{-3}$, no weight decay. SGD+L_2: $\eta_0 = 10^{-2}$, no momentum, with an explicit (coupled) L_2 penalty $\lambda \sum \|W\|_F^2$ on all kernel weights, $\lambda = 10^{-2}$; this is the SGD+L_2 configuration of Figure 4.

Optimizer settings. Table 3 lists the per-optimizer settings used in the main grid. All five optimizers share $\eta_0 = 10^{-3}$ with cosine decay. The SGD family is run with zero weight decay; AdamW and Muon retain their default decoupled weight decay $\lambda = 10^{-4}$ as part of the optimizer definition. Adam, which has *no* weight decay, exhibits the same small spreads as AdamW and Muon, so the forgetting effect attributed to adaptive methods is not driven by AdamW's or Muon's $\lambda = 10^{-4}$.

Table 3: **Main-grid optimizer settings.** All optimizers use $\eta_0 = 10^{-3}$ with cosine decay ($\alpha = 0.01$) over 300 epochs. Adam-family $\beta_1 = 0.9$ (momentum-like) and $\beta_2 = 0.999$ (variance) are the Keras defaults.

Optimizer	η_0	Momentum	Weight decay λ	Notes
SGD	10^{-3}	0	0	—
SGD with momentum	10^{-3}	0.9	0	—
Adam	10^{-3}	β_1=0.9	0	β_2=0.999 (Keras default)
AdamW	10^{-3}	β_1=0.9	10^{-4}	Decoupled weight decay (default)
Muon	10^{-3}	—	10^{-4}	Output dense and embeddings excluded

Reproducibility. For each run, Python, NumPy, and TensorFlow random seeds are set to a single integer, so initialization, minibatch order, and all stochastic operations are jointly determined by it.

B Supplementary Tables

Tables 4 and 5 report validation accuracy at three training checkpoints for the two extreme initialization scales, $\sigma_w = 0.1$ and $\sigma_w = 2.5$. The *repair gap* $\Delta_{\text{repair}} := \text{ValAcc}_{\tau_{\text{best}}} - \text{ValAcc}_{\tau_{\text{interp}}}$ quantifies how much validation accuracy changes between the interpolation epoch and the best-validation-loss epoch. Its sign, however, is largely determined by the *ordering* of τ_{best} and τ_{interp}: because

validation accuracy generally increases during training, configurations where $\tau_{\text{best}} < \tau_{\text{interp}}$ (i.e. the lowest validation loss occurs before the model memorizes the training set) yield a mechanically negative Δ_{repair}. This is the dominant regime for adaptive optimizers at small batch sizes, where the model achieves its best-calibrated predictions early ($\tau_{\text{best}} \approx 5$–$7$) but does not interpolate until epoch 30–40; accuracy continues to improve even as cross-entropy loss rises. In contrast, at large batch sizes ($b \geq 128$), adaptive optimizers have $\tau_{\text{best}} \gg \tau_{\text{interp}}$, and the positive Δ_{repair} reflects genuine post-interpolation restructuring that benefits generalization. SGD exhibits near-zero Δ_{repair} at both extremes: its validation accuracy is essentially flat between τ_{best} and τ_{interp}, even when both grow with σ_w, consistent with the stopping-time stagnation discussed in §3.

Table 4: Validation accuracy at three checkpoints for $\sigma_w = 0.1$. Values are mean $\pm$ std over $n = 10$ seeds.

b	Optimizer	τ_{interp}	τ_{best}	Val Acc$_{\tau_{\text{interp}}}$	Val Acc$_{\tau_{\text{best}}}$	Val Acc$_{300}$	Δ_{repair}
16	SGD	8.8 ± 0.4	10.5 ± 0.7	0.862 ± 0.019	0.886 ± 0.003	0.890 ± 0.002	$+0.024 \pm 0.019$
16	SGD ($\beta = 0.9$)	12.0 ± 0.0	14.4 ± 2.0	0.870 ± 0.008	0.881 ± 0.004	0.889 ± 0.002	$+0.012 \pm 0.009$
16	Adam	31.8 ± 2.0	6.0 ± 1.3	0.851 ± 0.009	0.831 ± 0.015	0.879 ± 0.002	-0.020 ± 0.017
16	AdamW	32.1 ± 1.7	5.2 ± 1.0	0.844 ± 0.017	0.825 ± 0.012	0.880 ± 0.002	-0.018 ± 0.021
16	Muon	32.6 ± 1.9	6.6 ± 1.2	0.849 ± 0.008	0.836 ± 0.009	0.879 ± 0.003	-0.013 ± 0.011
32	SGD	9.0 ± 0.0	11.4 ± 1.2	0.868 ± 0.007	0.882 ± 0.003	0.883 ± 0.002	$+0.013 \pm 0.008$
32	SGD ($\beta = 0.9$)	10.0 ± 0.0	11.4 ± 1.1	0.874 ± 0.005	0.882 ± 0.004	0.889 ± 0.002	$+0.009 \pm 0.008$
32	Adam	28.9 ± 2.7	5.8 ± 1.0	0.849 ± 0.008	0.825 ± 0.012	0.883 ± 0.001	-0.024 ± 0.017
32	AdamW	28.5 ± 2.6	5.4 ± 1.0	0.849 ± 0.010	0.819 ± 0.018	0.882 ± 0.002	-0.030 ± 0.023
32	Muon	29.8 ± 1.6	5.7 ± 1.3	0.850 ± 0.012	0.827 ± 0.011	0.883 ± 0.003	-0.023 ± 0.016
64	SGD	9.1 ± 0.3	11.7 ± 1.3	0.815 ± 0.024	0.869 ± 0.003	0.869 ± 0.003	$+0.055 \pm 0.025$
64	SGD ($\beta = 0.9$)	9.0 ± 0.0	10.9 ± 0.9	0.868 ± 0.005	0.885 ± 0.002	0.888 ± 0.001	$+0.016 \pm 0.004$
64	Adam	23.3 ± 3.1	5.8 ± 1.6	0.843 ± 0.013	0.821 ± 0.018	0.884 ± 0.002	-0.022 ± 0.018
64	AdamW	25.4 ± 4.7	6.4 ± 3.7	0.851 ± 0.010	0.817 ± 0.025	0.884 ± 0.002	-0.034 ± 0.024
64	Muon	24.0 ± 3.9	7.7 ± 4.6	0.830 ± 0.025	0.825 ± 0.023	0.883 ± 0.002	-0.005 ± 0.015
128	SGD	11.0 ± 0.0	14.5 ± 2.0	0.817 ± 0.036	0.856 ± 0.004	0.856 ± 0.002	$+0.039 \pm 0.037$
128	SGD ($\beta = 0.9$)	8.0 ± 0.0	10.0 ± 0.5	0.847 ± 0.018	0.884 ± 0.002	0.886 ± 0.003	$+0.037 \pm 0.018$
128	Adam	16.7 ± 2.7	34.6 ± 22.1	0.833 ± 0.041	0.872 ± 0.022	0.885 ± 0.003	$+0.038 \pm 0.035$
128	AdamW	17.9 ± 2.6	34.6 ± 23.8	0.848 ± 0.026	0.874 ± 0.021	0.886 ± 0.002	$+0.026 \pm 0.037$
128	Muon	17.1 ± 3.7	32.6 ± 23.0	0.844 ± 0.020	0.867 ± 0.022	0.886 ± 0.003	$+0.023 \pm 0.021$
256	SGD	14.1 ± 0.3	21.1 ± 2.9	0.744 ± 0.064	0.841 ± 0.003	0.841 ± 0.003	$+0.097 \pm 0.063$
256	SGD ($\beta = 0.9$)	8.4 ± 0.5	10.7 ± 0.5	0.844 ± 0.018	0.878 ± 0.001	0.880 ± 0.003	$+0.034 \pm 0.018$
256	Adam	11.5 ± 0.7	40.5 ± 29.5	0.846 ± 0.014	0.882 ± 0.009	0.885 ± 0.002	$+0.036 \pm 0.015$
256	AdamW	11.5 ± 1.3	53.5 ± 30.3	0.843 ± 0.020	0.878 ± 0.007	0.884 ± 0.001	$+0.034 \pm 0.023$
256	Muon	11.6 ± 1.0	56.1 ± 32.9	0.837 ± 0.023	0.882 ± 0.005	0.885 ± 0.003	$+0.045 \pm 0.023$

B.1 Best-validation versus final checkpoint

To test whether the optimizer contrast depends on the checkpoint rule, we log *test* accuracy at both τ_{best} and the final epoch τ_{final}=300 (main-grid recipe, b=128, 5 seeds). The spread over the tested scales is essentially unchanged between the two checkpoints (Table 6).

B.2 Seed-level distribution

Table 7 shows the per-seed test accuracy behind the 26.5 pp SGD spread (main grid, b=128) at three representative scales. The dependence is a smooth, monotone shift of all seeds, not bistability: across the full 25-point grid, the largest per-scale standard deviation is 0.76 pp and the largest single-seed deviation from its σ_w-mean is 1.82 pp. On the extended sweep (App. E.2), the largest per-scale seed standard deviation is ≤ 0.5 pp for SGD, ≤ 1.6 pp for Adam, and ≤ 0.2 pp for SGD+L_2, far below the σ_w-driven spreads.

Table 5: Validation accuracy at three checkpoints for $\sigma_w = 2.5$ (large initialization). At this extreme, SGD's test accuracy collapses to $\sim$58% at $b = 128$ while adaptive optimizers retain $\sim$85%; the contrast with Table 4 directly quantifies initialization bias. Values are mean $\pm$ std over $n = 10$ seeds.

b	Optimizer	τ_{interp}	τ_{best}	Val Acc$_{\tau_{\text{interp}}}$	Val Acc$_{\tau_{\text{best}}}$	Val Acc$_{300}$	Δ_{repair}
16	SGD	33.9 ± 1.3	15.0 ± 2.3	0.679 ± 0.003	0.660 ± 0.005	0.699 ± 0.003	-0.019 ± 0.005
16	SGD ($\beta = 0.9$)	10.9 ± 0.3	12.0 ± 2.5	0.779 ± 0.004	0.781 ± 0.011	0.794 ± 0.002	$+0.003 \pm 0.010$
16	Adam	39.0 ± 2.1	6.4 ± 1.9	0.848 ± 0.012	0.832 ± 0.011	0.886 ± 0.001	-0.016 ± 0.018
16	AdamW	39.2 ± 1.4	6.5 ± 1.9	0.853 ± 0.011	0.832 ± 0.016	0.886 ± 0.002	-0.021 ± 0.019
16	Muon	39.8 ± 2.5	6.4 ± 1.3	0.853 ± 0.010	0.830 ± 0.015	0.886 ± 0.002	-0.023 ± 0.022
32	SGD	39.0 ± 1.2	20.8 ± 3.0	0.644 ± 0.005	0.628 ± 0.007	0.665 ± 0.005	-0.016 ± 0.004
32	SGD ($\beta = 0.9$)	10.1 ± 0.3	9.4 ± 1.6	0.729 ± 0.007	0.727 ± 0.008	0.749 ± 0.003	-0.002 ± 0.010
32	Adam	40.8 ± 2.9	7.2 ± 3.2	0.837 ± 0.014	0.812 ± 0.021	0.883 ± 0.002	-0.025 ± 0.031
32	AdamW	37.7 ± 3.5	7.2 ± 2.3	0.843 ± 0.011	0.818 ± 0.018	0.883 ± 0.003	-0.025 ± 0.027
32	Muon	40.0 ± 3.4	6.7 ± 2.1	0.846 ± 0.013	0.811 ± 0.018	0.883 ± 0.002	-0.035 ± 0.021
64	SGD	51.2 ± 1.5	35.3 ± 3.3	0.614 ± 0.005	0.605 ± 0.006	0.637 ± 0.006	-0.009 ± 0.003
64	SGD ($\beta = 0.9$)	10.2 ± 0.4	8.3 ± 1.6	0.680 ± 0.008	0.676 ± 0.009	0.706 ± 0.004	-0.004 ± 0.009
64	Adam	31.5 ± 4.4	7.9 ± 3.4	0.833 ± 0.012	0.799 ± 0.021	0.876 ± 0.004	-0.034 ± 0.025
64	AdamW	32.3 ± 4.6	6.9 ± 3.3	0.826 ± 0.016	0.795 ± 0.023	0.877 ± 0.002	-0.031 ± 0.030
64	Muon	30.9 ± 5.7	7.4 ± 3.4	0.828 ± 0.014	0.795 ± 0.022	0.876 ± 0.002	-0.033 ± 0.030
128	SGD	78.6 ± 3.3	59.6 ± 4.9	0.595 ± 0.006	0.587 ± 0.006	0.612 ± 0.006	-0.007 ± 0.003
128	SGD ($\beta = 0.9$)	12.1 ± 0.3	8.3 ± 1.6	0.640 ± 0.004	0.630 ± 0.009	0.671 ± 0.005	-0.011 ± 0.005
128	Adam	15.3 ± 3.1	55.6 ± 24.2	0.799 ± 0.024	0.851 ± 0.004	0.863 ± 0.004	$+0.051 \pm 0.023$
128	AdamW	14.6 ± 1.8	68.6 ± 30.2	0.808 ± 0.023	0.852 ± 0.004	0.860 ± 0.008	$+0.045 \pm 0.025$
128	Muon	17.1 ± 2.8	69.5 ± 30.8	0.812 ± 0.028	0.850 ± 0.003	0.859 ± 0.005	$+0.038 \pm 0.029$
256	SGD	153.3 ± 8.7	124.8 ± 15.0	0.582 ± 0.004	0.578 ± 0.006	0.587 ± 0.004	-0.004 ± 0.003
256	SGD ($\beta = 0.9$)	16.5 ± 0.5	12.9 ± 2.1	0.608 ± 0.005	0.601 ± 0.007	0.642 ± 0.005	-0.007 ± 0.007
256	Adam	8.0 ± 0.0	80.1 ± 21.8	0.787 ± 0.007	0.840 ± 0.010	0.861 ± 0.006	$+0.053 \pm 0.009$
256	AdamW	8.1 ± 0.3	94.0 ± 22.4	0.783 ± 0.011	0.844 ± 0.008	0.861 ± 0.007	$+0.061 \pm 0.012$
256	Muon	8.0 ± 0.0	89.3 ± 13.5	0.786 ± 0.008	0.848 ± 0.010	0.865 ± 0.004	$+0.062 \pm 0.008$

Table 6: Seed-mean test accuracy (%) at τ_{best} and at τ_{final}=300 (BatchNorm ResNet-9, main-grid recipe, b=128, 5 seeds).

	SGD		Adam		SGD+L_2	
σ_w	τ_{best}	τ_{final}	τ_{best}	τ_{final}	τ_{best}	τ_{final}
0.1	85.1	85.1	86.4	88.3	88.6	88.4
0.5	72.8	74.1	88.6	88.9	88.6	88.3
1.0	67.7	69.5	88.1	88.5	88.5	88.4
1.8	62.3	64.5	86.3	87.0	88.5	88.3
2.5	59.0	61.2	84.5	85.4	88.3	88.1
spread	26.1	23.9	4.1	3.5	0.3	0.3

Table 7: Per-seed test accuracy (%) at τ_{best}; SGD, b=128, main grid.

σ_w	s0	s1	s42	s123	s456	s789	s999	s2024	s2025	s2026	mean	std
0.10	85.16	85.45	84.89	85.11	85.21	85.38	85.26	84.88	84.42	84.69	85.05	0.32
1.00	67.76	66.87	67.93	68.41	67.60	67.48	67.12	67.08	67.08	67.47	67.48	0.47
2.50	59.25	58.42	59.18	59.93	58.43	58.47	58.08	58.19	57.84	57.77	58.56	0.69

C Learning-Rate Ablation

Table 8: **Full LR $\times$ L2 sweep for SGD on ResNet-9** (no momentum). Test accuracy (%) at the best-validation-loss epoch for each $(\eta, \lambda_{\mathrm{L2}}, \sigma_w)$ configuration. The "Spread" column reports $\max_{\sigma_w} \mathrm{acc} - \min_{\sigma_w} \mathrm{acc}$ (in percentage points); "Train" is the mean training accuracy across all σ_w values. Entries with $\pm$ show mean $\pm$ std over $n = 10$ seeds; entries marked † are from a single representative seed (seed 42). **Bottom rows**: Adam baseline and SGD long training (5000 ep, constant LR) for comparison.

			Test Accuracy (%) at $\sigma_w =$						
η	λ_{L2}	Sched.	0.1	0.5	1.0	1.8	2.5	Spread	Train
Batch size $b = 16$									
10^{-3}	0	cos	88.1±0.3	80.9±0.4	74.3±0.6	69.1±0.9	65.4±0.8	22.8	99.6
10^{-3}	10^{-3}	cos	88.1	82.5	77.9	74.3	72.1	16.0	100.0†
10^{-3}	5×10^{-3}	cos	88.7	86.4	85.5	82.7	82.1	6.6	100.0†
10^{-3}	10^{-2}	cos	88.4	87.1	86.1	85.6	86.2	2.8	100.0†
10^{-2}	0	cos	87.6	87.6	85.0	80.9	77.9	9.7	100.0†
10^{-2}	10^{-3}	cos	90.6	90.7	90.5	90.5	90.3	0.4	100.0†
10^{-2}	5×10^{-3}	cos	90.4	90.6	90.6	90.3	90.1	0.5	100.0†
10^{-2}	10^{-2}	cos	89.8±0.2	89.9±0.2	89.9±0.2	89.6±0.2	89.7±0.4	0.3	100.0
10^{-1}	0	cos	80.3	81.0	81.1	82.0	77.5	4.5	90.8†
10^{-1}	10^{-3}	cos	90.4	90.7	90.5	90.8	90.8	0.5	100.0†
10^{-1}	5×10^{-3}	cos	89.9	89.6	89.4	90.1	89.9	0.7	100.0†
10^{-1}	10^{-2}	cos	88.1	88.9	88.2	88.1	88.5	0.8	100.0†
Adam η=10^{-3}		cos	82.6±1.5	84.1±1.5	83.1±1.6	83.5±1.4	82.8±1.0	1.5	95.3
SGD η=10^{-3} (5k ep)		const	88.0	80.5	75.2	69.6	66.1	21.9	99.9†

			Test Accuracy (%) at $\sigma_w =$						
η	λ_{L2}	Sched.	0.1	0.5	1.0	1.8	2.5	Spread	Train
Batch size $b = 128$									
10^{-3}	0	cos	85.0±0.3	72.7±0.7	67.5±0.5	62.0±0.5	58.6±0.7	26.5	99.8
10^{-3}	10^{-3}	cos	85.8	73.7	69.9	65.0	62.0	23.8	100.0†
10^{-3}	5×10^{-3}	cos	85.1	74.9	71.4	66.9	64.0	21.1	100.0†
10^{-3}	10^{-2}	cos	85.3	75.0	72.5	69.2	66.8	18.5	100.0†
10^{-2}	0	cos	88.1	82.8	75.6	67.8	64.3	23.8	99.6†
10^{-2}	10^{-3}	cos	88.4	85.0	78.5	73.5	71.1	17.4	100.0†
10^{-2}	5×10^{-3}	cos	89.1	88.8	88.2	88.2	88.2	0.9	100.0†
10^{-2}	10^{-2}	cos	88.6±0.3	88.7±0.1	88.4±0.3	88.4±0.3	88.4±0.1	0.3	100.0
10^{-1}	0	cos	84.1±1.9	85.3±1.2	84.8±0.8	81.7±1.4	77.2±2.1	8.1	98.8
10^{-1}	10^{-3}	cos	89.0	89.3	89.5	89.5	88.6	0.9	100.0†
10^{-1}	5×10^{-3}	cos	89.1	89.0	89.1	89.3	89.1	0.3	100.0†
10^{-1}	10^{-2}	cos	88.5	88.6	88.5	88.8	88.3	0.6	100.0†
Adam η=10^{-3}		cos	86.9±2.3	87.9±2.4	87.6±1.4	86.1±0.5	84.5±0.5	3.5	99.7
SGD η=10^{-3} (5k ep)		const	85.1	73.9	67.6	62.0	59.3	25.9	99.9†

D Depth Comparison: Full Results

Figures 5 and 7 display test and train accuracy as a function of initialization scale σ_w for four architectures: ResNet-9 (6.58M parameters), R9-AvgPool (6.58M), ResNet-56 (0.86M), and ResNet-110 (1.74M). Each panel fixes one optimizer (SGD or Adam) and one batch size ($b = 16$ or $b = 128$); lines show the mean over seeds and shaded regions indicate the 10th–90th percentile range.

Table 9 reports the numerical values plotted in Figure 5, together with train accuracy at τ_{best} and the spread (difference between the best and worst mean test accuracy across σ_w values). The lower part of the table records the interpolation epoch τ_{interp} (first epoch where train accuracy reaches the architecture-specific threshold stated in Table 9) and the best-validation-loss epoch for each configuration.

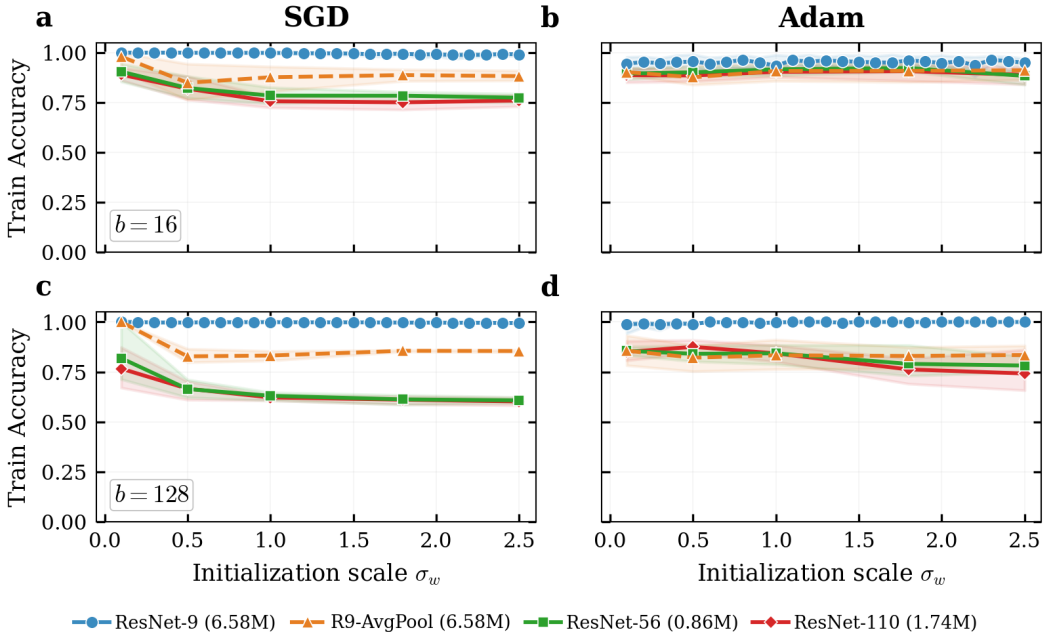


Figure 7: Train accuracy vs. initialization scale σ_w (same layout as Figure 5). Under SGD at $b = 128$, ResNet-56 and ResNet-110 fail to interpolate at large σ_w, while ResNet-9 maintains $> 99\%$ train accuracy throughout.

Table 9: Depth comparison: training and generalization metrics across architectures. All runs use η=10^{-3} with cosine decay, no weight decay, 300 epochs. Values are mean $\pm$ std over 10 seeds (ResNet-9, from the main grid) or 5 seeds (R9-AvgPool, ResNet-56, ResNet-110). Test and train accuracy are measured at τ_{best}. τ_{interp}: first epoch where train accuracy $\geq 99.5\%$ for ResNet-56/110, and $\geq 99.99\%$ for ResNet-9 and R9-AvgPool (for ResNet-9 at the 99.5% threshold, see Tables 4–5); "—" indicates the threshold was not reached. Spread $= \max_{\sigma_w} \text{test_acc} - \min_{\sigma_w} \text{test_acc}$.

Batch size $b = 16$		σ_w=0.1		σ_w=0.5		σ_w=1.0		σ_w=1.8		σ_w=2.5		
Architecture	Opt.	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Spread
ResNet-9	SGD	88.1 ± 0.3	100.0 ± 0.0	80.9 ± 0.4	100.0 ± 0.0	74.3 ± 0.6	99.9 ± 0.2	69.1 ± 0.9	99.5 ± 1.2	65.4 ± 0.8	99.1 ± 0.8	22.8
	Adam	82.6 ± 1.5	94.5 ± 3.5	84.1 ± 1.5	95.6 ± 3.0	83.1 ± 1.6	93.5 ± 3.8	83.5 ± 1.4	96.2 ± 2.5	82.8 ± 1.0	94.9 ± 2.8	1.5
R9-AvgPool	SGD	82.3 ± 1.9	98.0 ± 4.2	68.2 ± 1.1	84.9 ± 7.5	66.6 ± 1.0	87.7 ± 5.4	64.2 ± 0.6	88.8 ± 2.3	62.6 ± 0.3	88.2 ± 2.1	19.8
	Adam	77.8 ± 1.4	90.3 ± 2.3	77.5 ± 1.8	87.7 ± 4.8	79.0 ± 1.0	90.8 ± 3.8	79.2 ± 1.3	90.9 ± 4.2	78.4 ± 1.7	91.2 ± 4.5	1.7
ResNet-56	SGD	79.2 ± 1.1	90.5 ± 3.8	71.0 ± 1.7	82.3 ± 4.6	66.2 ± 1.1	78.5 ± 3.1	63.2 ± 1.6	78.4 ± 2.1	61.2 ± 2.1	77.4 ± 1.9	18.0
	Adam	80.1 ± 0.9	89.5 ± 2.6	80.5 ± 0.8	90.2 ± 2.2	80.8 ± 1.0	91.7 ± 2.0	80.4 ± 1.0	92.3 ± 2.1	78.1 ± 1.2	88.4 ± 3.1	2.7
ResNet-110	SGD	78.3 ± 1.0	89.0 ± 3.0	69.0 ± 1.8	81.9 ± 4.5	63.1 ± 1.2	75.6 ± 2.2	60.2 ± 1.2	75.1 ± 3.2	59.6 ± 1.2	76.0 ± 2.4	18.7
	Adam	79.8 ± 1.4	88.7 ± 3.4	80.4 ± 1.0	88.6 ± 2.5	80.7 ± 1.1	90.4 ± 2.5	80.2 ± 1.2	90.7 ± 3.4	78.6 ± 2.2	88.9 ± 4.0	2.1

Batch size $b = 128$		σ_w=0.1		σ_w=0.5		σ_w=1.0		σ_w=1.8		σ_w=2.5		
Architecture	Opt.	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Spread
ResNet-9	SGD	85.0 ± 0.3	100.0 ± 0.0	72.7 ± 0.7	99.7 ± 0.5	67.5 ± 0.5	99.9 ± 0.1	62.0 ± 0.5	99.7 ± 0.2	58.6 ± 0.7	99.5 ± 0.2	26.5
	Adam	86.9 ± 2.3	98.8 ± 2.6	87.9 ± 2.4	99.0 ± 3.3	87.6 ± 1.4	99.7 ± 0.7	86.1 ± 0.5	100.0 ± 0.0	84.5 ± 0.5	100.0 ± 0.0	3.5
R9-AvgPool	SGD	77.4 ± 0.4	100.0 ± 0.0	65.5 ± 0.8	82.8 ± 3.1	63.8 ± 0.8	83.2 ± 2.5	61.5 ± 0.6	85.6 ± 0.6	59.9 ± 0.6	85.5 ± 0.5	17.5
	Adam	74.5 ± 3.2	85.6 ± 6.6	73.7 ± 2.6	82.1 ± 5.9	73.8 ± 2.5	83.3 ± 6.2	72.0 ± 2.5	83.0 ± 5.7	70.3 ± 1.7	83.5 ± 5.5	4.3
ResNet-56	SGD	71.3 ± 4.9	81.9 ± 11.0	57.4 ± 1.5	66.4 ± 3.9	52.4 ± 1.0	63.0 ± 1.8	49.0 ± 1.6	61.2 ± 2.3	47.1 ± 1.9	60.7 ± 1.3	24.2
	Adam	76.8 ± 1.2	85.5 ± 2.6	75.8 ± 1.4	84.1 ± 3.5	73.6 ± 2.1	84.3 ± 4.4	69.1 ± 3.5	79.0 ± 6.5	67.5 ± 2.3	78.2 ± 3.8	9.3
ResNet-110	SGD	67.5 ± 4.0	76.6 ± 8.7	56.8 ± 1.9	66.6 ± 4.8	52.2 ± 0.6	62.2 ± 1.3	48.6 ± 1.0	61.0 ± 1.7	47.2 ± 0.7	60.2 ± 1.6	20.4
	Adam	76.5 ± 1.6	85.0 ± 4.3	77.3 ± 1.3	87.5 ± 3.2	74.0 ± 2.0	84.3 ± 3.4	66.8 ± 4.7	76.2 ± 5.5	63.4 ± 5.9	74.2 ± 8.3	13.9

		σ_w=0.1		σ_w=0.5		σ_w=1.0		σ_w=1.8		σ_w=2.5	
Architecture	Opt.	τ_{interp}	Best ep.	τ_{interp}	Best ep.	τ_{interp}	Best ep.	τ_{interp}	Best ep.	τ_{interp}	Best ep.
$b = 16$											
ResNet-9	SGD	11.0 ± 0.7	10.5 ± 0.7	13.8 ± 1.4	10.5 ± 1.4	32.2 ± 3.2	10.7 ± 1.3	93.5 ± 10.2	14.2 ± 2.4	203.6 ± 30.4	15.0 ± 2.3
	Adam	155.2 ± 4.8	6.0 ± 1.3	154.6 ± 9.4	6.6 ± 2.3	153.1 ± 10.2	5.4 ± 2.1	157.6 ± 10.0	7.3 ± 1.9	161.5 ± 7.2	6.4 ± 1.9
R9-AvgPool	SGD	32.1 ± 3.5	23.8 ± 11.2	67.2 ± 6.6	7.2 ± 2.1	161.4 ± 11.5	11.6 ± 2.1	—	20.2 ± 1.5	—	27.3 ± 1.9
	Adam	173.2 ± 7.2	5.8 ± 0.4	168.6 ± 10.9	5.1 ± 1.3	166.4 ± 4.5	5.0 ± 1.1	167.5 ± 6.8	4.6 ± 1.1	166.2 ± 9.4	4.7 ± 1.1
ResNet-56	SGD	48.1 ± 3.2	13.3 ± 4.5	67.2 ± 1.5	12.1 ± 2.8	115.7 ± 3.7	20.3 ± 2.4	—	41.7 ± 4.7	—	66.5 ± 8.4
	Adam	78.1 ± 3.6	10.0 ± 1.8	78.7 ± 3.4	10.1 ± 1.9	77.4 ± 4.2	10.5 ± 1.7	78.4 ± 6.3	10.8 ± 1.5	81.0 ± 5.2	8.8 ± 2.3
ResNet-110	SGD	43.7 ± 1.8	10.9 ± 2.2	63.5 ± 2.8	12.3 ± 2.1	110.3 ± 4.3	18.5 ± 1.6	252.0 ± 32.5	38.9 ± 5.0	—	66.8 ± 10.5
	Adam	74.2 ± 3.4	10.1 ± 2.2	73.6 ± 2.8	9.3 ± 1.8	73.5 ± 4.2	10.2 ± 2.0	74.7 ± 3.9	10.2 ± 2.1	75.9 ± 3.6	9.6 ± 1.7
$b = 128$											
ResNet-9	SGD	12.8 ± 0.4	14.5 ± 2.0	32.9 ± 1.3	22.0 ± 3.2	61.9 ± 2.8	39.3 ± 2.0	117.4 ± 9.6	51.8 ± 5.1	222.1 ± 23.2	59.6 ± 4.9
	Adam	25.7 ± 8.4	34.6 ± 22.1	22.8 ± 5.4	35.7 ± 24.0	22.1 ± 4.3	46.3 ± 30.2	19.8 ± 1.8	45.1 ± 26.0	19.1 ± 4.2	55.6 ± 24.2
R9-AvgPool	SGD	18.9 ± 0.6	20.0 ± 1.4	62.8 ± 3.2	26.7 ± 2.3	169.8 ± 4.4	54.0 ± 4.5	—	118.4 ± 2.6	—	224.2 ± 17.7
	Adam	103.2 ± 15.2	7.2 ± 2.1	81.3 ± 17.8	5.0 ± 1.6	75.3 ± 14.0	4.2 ± 1.5	63.9 ± 10.4	3.6 ± 1.0	62.4 ± 10.2	3.5 ± 1.4
ResNet-56	SGD	51.6 ± 4.8	24.9 ± 25.7	114.6 ± 7.2	25.0 ± 4.3	—	45.4 ± 5.0	—	93.9 ± 18.7	—	177.3 ± 57.8
	Adam	69.7 ± 3.9	12.1 ± 1.9	66.0 ± 3.9	9.9 ± 2.2	71.2 ± 7.1	10.9 ± 3.3	79.0 ± 6.4	8.2 ± 2.2	86.5 ± 8.9	8.1 ± 2.1
ResNet-110	SGD	48.2 ± 2.6	12.4 ± 5.5	109.0 ± 7.1	24.3 ± 3.8	236.7 ± 9.1	47.4 ± 6.5	—	98.5 ± 9.5	—	194.0 ± 46.7
	Adam	70.4 ± 2.7	13.3 ± 3.1	69.6 ± 8.0	13.5 ± 3.0	74.3 ± 4.8	10.6 ± 2.4	83.3 ± 5.2	8.5 ± 2.1	94.2 ± 4.9	9.0 ± 2.9

E Normalization and Data Augmentation

The main experiments use BatchNorm without data augmentation to isolate the effect of initialization scale from other regularization mechanisms. A natural question is whether switching the normalization layer or adding standard data augmentation eliminates initialization memory. To test this, we train ResNet-9 on CIFAR-10 under three configurations: (i) BatchNorm with augmentation (random horizontal flip + random crop with 4-pixel padding), (ii) LayerNorm with the same augmentation, and (iii) the original BatchNorm baseline without augmentation. All other hyperparameters are identical to the main grid ($\eta = 10^{-3}$, cosine decay, no weight decay, 300 epochs, $\sigma_w \in \{0.1, 0.5, 1.0, 1.8, 2.5\}$). Results are averaged over 5 seeds.

Figure 8 and Table 10 show the results. Data augmentation substantially reduces initialization memory under SGD (spread drops from 22.8 to 8.3 pp at $b = 16$, and from 26.5 to 18.9 pp at $b = 128$ for BatchNorm), but does not eliminate it. LayerNorm behaves similarly to BatchNorm: the SGD spread is 5.1 pp at $b = 16$ and 15.2 pp at $b = 128$. Under Adam, all three configurations show small spreads (≤ 3.5 pp), confirming that the forgetting property of Adam is robust to the choice of normalization and augmentation.

These results reinforce the main finding: neither the specific normalization scheme nor data augmentation is sufficient to erase the initialization memory under low-learning-rate SGD, though both reduce its magnitude.

Table 10: Normalization comparison on ResNet-9 CIFAR-10. "BN + Aug" and "LN + Aug" use data augmentation (random flip + crop) with BatchNorm and LayerNorm respectively; "BN (no aug)" is the main-grid baseline without augmentation. All use η=10^{-3}, cosine decay, no weight decay, 300 epochs. Values: mean $\pm$ std over 5 seeds (aug) or 10 seeds (baseline). Spread $= \max_{\sigma_w} \overline{\text{test_acc}} - \min_{\sigma_w} \overline{\text{test_acc}}$ (pp).

Batch size $b = 16$		σ_w=0.1		σ_w=0.5		σ_w=1.0		σ_w=1.8		σ_w=2.5		
Configuration	Opt.	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Spread
BN + Aug	SGD	92.7 ± 0.8	99.7 ± 0.6	89.9 ± 0.6	99.2 ± 0.5	87.8 ± 0.6	98.3 ± 0.7	85.9 ± 0.3	97.6 ± 0.6	84.4 ± 0.2	95.5 ± 0.3	8.3
	Adam	88.5 ± 0.9	94.7 ± 1.5	89.3 ± 0.3	94.9 ± 0.4	89.6 ± 0.6	95.8 ± 1.2	89.6 ± 0.5	95.7 ± 1.0	89.9 ± 0.5	96.7 ± 1.1	1.4
LN + Aug	SGD	89.6 ± 0.7	98.5 ± 0.8	88.9 ± 0.7	98.0 ± 1.2	87.5 ± 0.5	97.6 ± 1.3	84.9 ± 0.4	94.9 ± 0.9	84.4 ± 0.1	94.5 ± 0.5	5.1
	Adam	86.5 ± 0.4	94.6 ± 0.9	87.9 ± 0.6	95.2 ± 1.1	88.4 ± 0.6	95.5 ± 1.5	89.0 ± 0.5	95.9 ± 1.3	88.5 ± 0.2	95.0 ± 1.1	2.5
BN (no aug)	SGD	88.1 ± 0.3	100.0 ± 0.0	80.9 ± 0.4	100.0 ± 0.0	74.3 ± 0.6	99.9 ± 0.2	69.1 ± 0.9	99.5 ± 1.2	65.4 ± 0.8	99.1 ± 0.8	22.8
	Adam	82.6 ± 1.5	94.5 ± 3.5	84.1 ± 1.5	95.6 ± 3.0	83.1 ± 1.6	93.5 ± 3.8	83.5 ± 1.4	96.2 ± 2.5	82.8 ± 1.0	94.9 ± 2.8	1.5

Batch size $b = 128$		σ_w=0.1		σ_w=0.5		σ_w=1.0		σ_w=1.8		σ_w=2.5		
Configuration	Opt.	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Spread
BN + Aug	SGD	91.7 ± 0.2	100.0 ± 0.0	86.3 ± 0.4	96.2 ± 1.0	83.3 ± 0.3	92.3 ± 0.1	77.1 ± 0.4	82.1 ± 0.3	72.7 ± 0.4	76.2 ± 0.4	18.9
	Adam	88.7 ± 0.5	95.2 ± 0.9	89.0 ± 0.8	95.2 ± 1.5	89.1 ± 1.3	94.9 ± 2.7	88.6 ± 1.2	95.1 ± 2.1	88.2 ± 1.2	95.2 ± 1.8	0.9
LN + Aug	SGD	89.6 ± 0.3	99.7 ± 0.1	87.5 ± 0.2	97.7 ± 0.1	83.3 ± 0.2	89.6 ± 0.2	78.3 ± 0.3	80.5 ± 0.4	74.4 ± 0.4	74.3 ± 0.6	15.2
	Adam	86.5 ± 1.1	93.4 ± 1.6	88.9 ± 0.3	96.5 ± 0.8	89.5 ± 0.6	97.2 ± 1.1	89.4 ± 0.6	96.9 ± 1.3	89.2 ± 0.8	97.0 ± 1.4	3.0
BN (no aug)	SGD	85.0 ± 0.3	100.0 ± 0.0	72.7 ± 0.7	99.7 ± 0.5	67.5 ± 0.5	99.9 ± 0.1	62.0 ± 0.5	99.7 ± 0.2	58.6 ± 0.7	99.5 ± 0.2	26.5
	Adam	86.9 ± 2.3	98.8 ± 2.6	87.9 ± 2.4	99.0 ± 3.3	87.6 ± 1.4	99.7 ± 0.7	86.1 ± 0.5	100.0 ± 0.0	84.5 ± 0.5	100.0 ± 0.0	3.5

E.1 Unnormalized (NoNorm) ResNet-9

The BatchNorm/LayerNorm checks above leave open whether the adaptive-vs-SGD contrast requires normalization. We therefore repeat the extended sweep without BatchNorm (ResNet-9, b=128, flip+crop augmentation, 5 seeds, $\sigma_w \in \{0.1, 1.0, 2.5, 5.0, 10.0\}$). Table 11 reports test accuracy at τ_{best}.

The contrast does *not* require BatchNorm: without it, Adam still has the broadest usable range, while SGD and SGD+L_2 collapse to near chance away from $\sigma_w \approx 1$. Graceful behavior at very large σ_w *does* require some normalization: BatchNorm Adam stays at 85.5% at σ_w=10, whereas NoNorm Adam falls from 84.9% at σ_w=1 to 21.8%.

E.2 Distance and direction from initialization

To address the coordinate-level question of whether low Mem reflects geometric erasure of the initial parameters, we record, per epoch, the kernel distance from initialization on an extended BatchNorm ResNet-9 sweep (b=128, augmentation, 5 seeds, $\sigma_w \in \{0.1, 0.5, 1.0, 1.8, 2.5, 3.5, 5.0, 10.0\}$, 120 runs; base recipe as in this appendix): the raw distance $\Delta(t) = \|W(t) - W_{\text{init}}\|_F$, the normalized

Table 11: Test accuracy (%) at τ_{best} on the extended ResNet-9 sweep (b=128, augmentation, 5 seeds), with and without BatchNorm. Chance on CIFAR-10 is 10%.

	NoNorm			BatchNorm		
σ_w	SGD	Adam	SGD+L_2	SGD	Adam	SGD+L_2
0.1	10.0	52.5	10.0	91.7	89.1	94.5
1.0	73.3	84.9	75.5	83.3	89.3	94.3
2.5	35.4	84.1	10.0	72.8	88.7	94.3
5.0	10.0	41.0	10.0	63.3	87.6	94.1
10.0	10.0	21.8	10.0	54.6	85.5	94.2

Table 12: Distance from initialization on the extended BatchNorm ResNet-9 sweep (b=128, augmentation, 5 seeds). Each cell is $\Delta(t) = \|W(t) - W_{\text{init}}\|_F$ with the normalized $d(t) = \Delta(t)/\|W_{\text{init}}\|_F$ in parentheses (kernel weights; t=300 is the final epoch). Low-LR SGD barely moves ($d \leq 0.69$); Adam traverses a large but bounded distance; SGD+L_2 drives $d \to 1$ by collapsing the norm.

	SGD			Adam			SGD+L_2		
σ_w	t=5	60	300	t=5	60	300	t=5	60	300
0.1	1.1 (0.23)	3.0 (0.63)	3.3 (0.69)	68 (14.4)	312 (65.9)	453 (95.5)	3.4 (0.71)	6.6 (1.39)	5.0 (1.06)
1.0	1.1 (0.02)	3.4 (0.07)	5.3 (0.11)	47 (0.99)	290 (6.12)	438 (9.23)	13.2 (0.28)	46.0 (0.97)	47.3 (1.00)
2.5	1.3 (0.01)	3.9 (0.03)	5.9 (0.05)	44 (0.37)	277 (2.34)	428 (3.61)	32.2 (0.27)	115 (0.97)	118 (1.00)
5.0	1.8 (0.008)	5.3 (0.02)	7.9 (0.03)	48 (0.20)	268 (1.13)	421 (1.78)	64.0 (0.27)	231 (0.97)	237 (1.00)
10.0	2.5 (0.005)	7.5 (0.02)	11.3 (0.02)	58 (0.12)	269 (0.57)	424 (0.89)	128 (0.27)	462 (0.97)	474 (1.00)

$d(t) = \Delta(t)/\|W_{\text{init}}\|_F$, and the direction $\cos(W(t), W_{\text{init}})$. Table 12 reports $\Delta(t)$ with $d(t)$ in parentheses.

In practice, the combination of small early displacement and $r_f = \|W_f\|/\|W_i\| \approx 1$ (SGD in Table 13) is a cheap warning sign of a run that retains its initialization scale; it is not a calibrated predictor of accuracy.

The trajectories separate the optimizers. Low-LR **SGD barely moves** ($\Delta(t) < 12$, $d(t) \leq 0.69$ at every scale), making a small update around its initial coordinates. **Adam traverses a large but bounded distance** ($\Delta(t) \to \approx$420–455, nearly independent of σ_w): a $\sim$95$\times$ move relative to $\|W_{\text{init}}\|$ at σ_w=0.1 but only $\sim$0.9$\times$ at σ_w=10. **SGD+L_2 collapses the norm**, driving $\|W_f\| \to \approx$2 so that $d(t) \to 1$. Direction is consistent: $\cos(W(t), W_{\text{init}}) \geq 0.8$ throughout for SGD; for Adam it drops below 0.5 early at small σ_w but the initial direction stays visible at large σ_w ($\cos(W_f, W_i)$=0.75 at σ_w=10); for SGD+L_2 it crosses 0.5 at progressively later epochs. These coordinate-level views *corroborate* the operational Mem ordering rather than redefine it: SGD, the strongest-retention optimizer, also has the largest test-accuracy spread. The ordering is seed-stable (std of $d(t$=300) $\lesssim$ 0.08, of $\cos(W_f, W_i) \leq 0.001$ across 5 seeds).

Extending the sweep to σ_w=10 (so $\|W_{\text{init}}\|_F \approx 474$, above Adam's usual final-norm band) shows the behavior is neither active downward contraction nor catastrophic failure for Adam (Table 13): Adam stays usable (85.5% at σ_w=10) with returned-predictor spread $\approx$ 3.8 pp over $\sigma_w \in [0.1, 10]$ versus $\approx$ 37 pp for SGD and $\approx$ 0.3 pp for SGD+L_2, so the adaptive-vs-SGD contrast persists at this 4$\times$ scale. Adam does not contract the large initial norm ($\|W_f\|$ rises to 638); active downward contraction comes only from explicit L_2, which maps $\|W_i\| \approx 474$ to $\|W_f\| \approx 2.0$ while keeping 94.2% accuracy.

Table 13: Extending the sweep to σ_w=10 (BatchNorm ResNet-9, b=128). Final-epoch kernel norm $\|W_f\|_F$ and test accuracy at τ_{best} ($\|W_i\|_F$ in parentheses next to σ_w). Adam stays usable at very large σ_w but does not contract the large initial norm; explicit L_2 actively collapses $\|W_f\|$ to ≈ 2.

	Adam		SGD		SGD+L_2	
σ_w ($\|W_i\|$)	$\|W_f\|$	test%	$\|W_f\|$	test%	$\|W_f\|$	test%
0.1 (4.7)	453	89.1	5.7	91.7	1.9	94.5
1.0 (47.4)	441	89.3	47.5	83.3	1.9	94.3
2.5 (118.6)	447	88.7	118.5	72.8	1.9	94.3
5.0 (237.1)	488	87.6	237.0	63.3	2.0	94.1
10.0 (474.3)	638	85.5	474.0	54.6	2.0	94.2

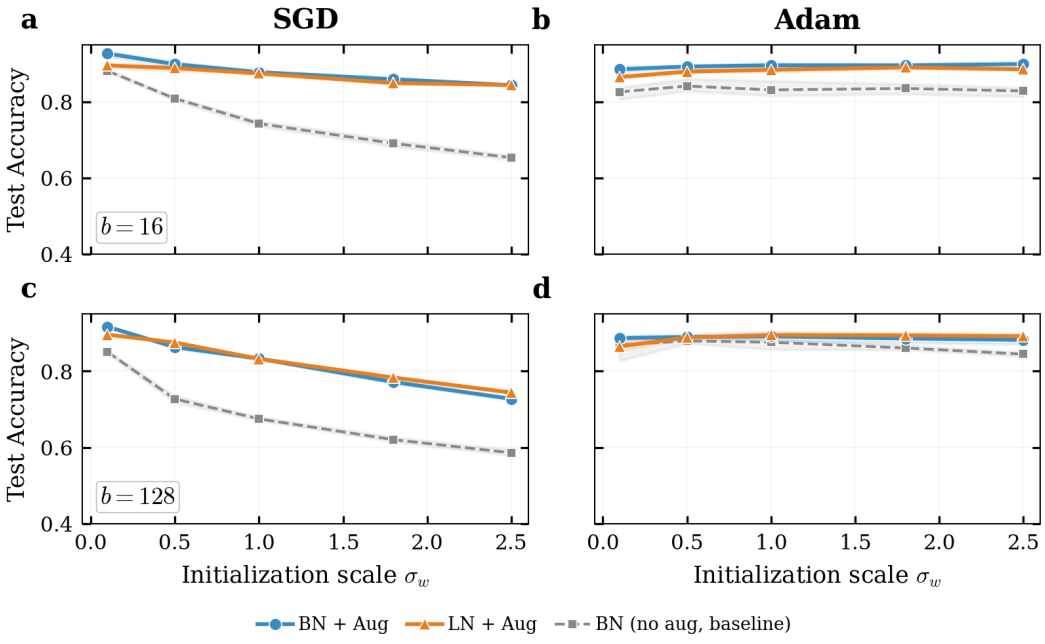


Figure 8: Test accuracy vs. initialization scale σ_w for ResNet-9 under BatchNorm + augmentation (blue), LayerNorm + augmentation (orange), and the BatchNorm baseline without augmentation (grey dashed). Lines: mean over seeds; shaded bands: 10th–90th percentile.

F Best-Training Pipeline Comparison

The preceding experiments deliberately use a minimal training configuration (η=10^{-3}, no momentum, no weight decay, no augmentation) to isolate the role of initialization scale. A natural concern is whether initialization memory persists under a standard, well-tuned training procedure. Table 14 addresses this by comparing two configurations at the extreme initialization scales $\sigma_w \in \{0.1, 2.5\}$: (i) SGD with lr=$0.1$, momentum=$0.9$, weight decay $= 5 \times 10^{-4}$, and data augmentation—the standard ResNet procedure—and (ii) Adam with lr=10^{-3}, no weight decay, and the same augmentation.

At b=128, SGD (best) achieves 94.2% test accuracy at both σ_w=0.1 and 2.5 (difference 0.0 pp after rounding), so the standard training procedure leaves no measurable initialization-scale dependence between the two tested endpoints. Adam also shows a small gap (0.5 pp at b=128; ≤ 0.6 pp across batch sizes). At b=16, the SGD procedure exhibits high seed-to-seed variance: with η=0.1 and only 16 samples per step, the effective per-sample step size is large enough that some seeds experience a near-divergence in the first epoch (loss > 50 vs. $\sim$15 for other seeds) and fail to recover fully within 200 epochs. This is a training-stability issue—addressable by learning-rate warmup—not an initialization-memory effect; the gap between σ_w values remains small even for the affected seeds.

Table 14: **Best-recipe comparison.** SGD (standard recipe: η=0.1, momentum=0.9, L_2 kernel reg. $\lambda = 5 \times 10^{-4}$) vs. Adam ($\eta$=$10^{-3}$, no WD), both with data augmentation and cosine decay, 200 epochs on ResNet-9 / CIFAR-10. Values: mean $\pm$ std over n=10 seeds. τ_{interp}: first epoch with train accuracy $\geq 99.5\%$ ("—" if not reached within 200 epochs); format is σ_w=0.1 / σ_w=2.5. τ_{best}: best-validation-loss epoch. $|\Delta|$: absolute difference between σ_w=0.1 and σ_w=2.5 test accuracy.

		Test Acc. (%)		Train Acc. (%)		Epochs		
Optimizer	b	σ_w=0.1	σ_w=2.5	σ_w=0.1	σ_w=2.5	τ_{interp}	τ_{best}	$\lvert\boldsymbol{\Delta}\rvert$
SGD (best)	16	86.5±6.4	86.7±3.0	90.2±7.7	90.2±4.2	— / —	198±2 / 198±1	0.2
	128	**94.2±0.1**	**94.2±0.2**	100.0±0.0	100.0±0.0	166±1 / 167±2	200±1 / 200±1	**0.0**
	256	93.8±0.5	93.9±0.3	100.0±0.0	100.0±0.0	154±3 / 153±2	198±2 / 199±1	0.1
Adam	16	89.5±0.7	90.1±0.5	100.0±0.0	100.0±0.0	90±2 / 90±3	17±3 / 23±4	0.6
	128	88.5±0.9	89.0±0.7	100.0±0.0	100.0±0.0	72±3 / 84±3	20±5 / 32±8	0.5
	256	89.1±1.0	88.6±0.8	100.0±0.0	100.0±0.0	68±4 / 83±3	28±9 / 38±10	0.5

G Minimal Linear Model for Initialization Memory

This appendix gives the calculations behind the timescale discussion in Section 4. The goal is not a complete theory of BatchNorm ResNets, but a minimal homogeneous model showing why gradient-flow-like dynamics can preserve initialization-dependent quantities and why explicit norm decay, finite steps, stochasticity, and adaptive preconditioning act on different clocks.

Throughout this appendix, we use ϕ for the loss as a function of the product represented by the linear network. This avoids overloading the layer index. All losses are assumed continuously differentiable, and all gradients are evaluated at the current product. In the two-factor case the product is $W = UV$. In the deep-linear case the product is

$$F(W_1, \dots, W_L) = W_L W_{L-1} \cdots W_1.$$

All discrete updates below are simultaneous updates of all factors.

For stochastic statements, $\mathbb{E}_k[\cdot]$ denotes conditional expectation given the current iterate. We write a product-space minibatch gradient estimate as

$$G_k = \bar{G}_k + \xi_k, \qquad \mathbb{E}_k[\xi_k] = 0, \qquad \mathbb{E}_k\|\xi_k\|_F^2 = O(1/b),$$

where $\bar{G}_k$ is the full product-space gradient and b is the minibatch size. In the deep-linear stochastic statements, the layerwise gradient estimates are induced from the same product-space estimate G_k by the chain rule. This shared G_k assumption is essential: the first-order cancellations below need not hold for arbitrary independent layerwise perturbations.

Gradient flow preserves imbalance. Consider a two-layer linear network $W = UV$, with

$$U \in \mathbb{R}^{d_{\text{out}} \times r}, \qquad V \in \mathbb{R}^{r \times d_{\text{in}}}, \qquad \mathcal{L}(U, V) = \phi(UV).$$

Let

$$G = \nabla_W \phi(W).$$

Then

$$\nabla_U \mathcal{L} = GV^\top, \qquad \nabla_V \mathcal{L} = U^\top G,$$

so Euclidean gradient flow is

$$\dot{U} = -GV^\top, \qquad \dot{V} = -U^\top G.$$

Define the imbalance

$$D(U, V) = U^\top U - VV^\top \in \mathbb{R}^{r \times r}.$$

Proposition 1 (Two-factor gradient flow preserves imbalance)**.** *Along Euclidean gradient flow for the two-factor linear network,*

$$\frac{d}{dt} D(U(t), V(t)) = 0.$$

Proof. We compute

$$\begin{aligned} \frac{d}{dt} U^\top U &= \dot{U}^\top U + U^\top \dot{U} \\ &= (-GV^\top)^\top U + U^\top (-GV^\top) \\ &= -VG^\top U - U^\top GV^\top. \end{aligned}$$

Similarly,

$$\begin{aligned} \frac{d}{dt} VV^\top &= \dot{V}V^\top + V\dot{V}^\top \\ &= (-U^\top G)V^\top + V(-U^\top G)^\top \\ &= -U^\top GV^\top - VG^\top U. \end{aligned}$$

The two derivatives are identical, hence their difference is zero:

$$\frac{d}{dt}\left(U^\top U - VV^\top\right) = 0.$$

□

Thus gradient flow exactly preserves an initialization-dependent quantity: $D(U(t), V(t)) = D(U(0), V(0))$. In this minimal homogeneous model, initialization memory is a conserved charge.

The conserved imbalance fixes the final norm in the scalar model. The preceding invariant is not merely formal: in the scalar case it directly determines the factor norm of any converged solution.

Proposition 2 (Scalar two-factor norm is fixed by conserved imbalance)**.** *Consider scalar parameters $a, b \in \mathbb{R}$ with predictor $p = ab$. Suppose gradient flow converges to a point with product $p_\star$. Let*

$$D_0 = a_0^2 - b_0^2.$$

Since $D = a^2 - b^2$ is conserved, the final squared Euclidean norm satisfies

$$\|(a_\infty, b_\infty)\|_2^2 = a_\infty^2 + b_\infty^2 = \sqrt{D_0^2 + 4p_\star^2}.$$

Therefore, among solutions with the same product $p_\star$, the final factor norm is a monotone function of $|D_0|$. The final norm thus retains initialization memory.

Proof. Let

$$x = a_\infty^2, \qquad y = b_\infty^2.$$

Conservation of imbalance gives

$$x - y = D_0,$$

and convergence to product $p_\star$ gives

$$xy = (a_\infty b_\infty)^2 = p_\star^2.$$

Therefore

$$(x + y)^2 = (x - y)^2 + 4xy = D_0^2 + 4p_\star^2.$$

Since $x + y \geq 0$, we obtain

$$x + y = \sqrt{D_0^2 + 4p_\star^2}.$$

Substituting back $x = a_\infty^2$ and $y = b_\infty^2$ gives the claim. □

Continuous-time L_2 decay contracts imbalance. Now add coupled continuous-time weight decay:

$$\dot{U} = -GV^\top - \lambda U, \qquad \dot{V} = -U^\top G - \lambda V, \qquad \lambda \geq 0.$$

Proposition 3 (Continuous-time L_2 decay erases imbalance)**.** *Under the dynamics above,*

$$\frac{d}{dt} D(t) = -2\lambda D(t), \qquad D(t) = e^{-2\lambda t} D(0).$$

Proof. The gradient-dependent terms are the same as in the previous calculation and still cancel in the difference. The decay terms give

$$\frac{d}{dt} U^\top U = \text{gradient terms} - 2\lambda U^\top U,$$
$$\frac{d}{dt} VV^\top = \text{same gradient terms} - 2\lambda VV^\top.$$

Subtracting yields

$$\dot{D}(t) = -2\lambda \left(U^\top U - VV^\top \right) = -2\lambda D(t).$$

Solving this linear differential equation gives

$$D(t) = e^{-2\lambda t} D(0).$$

□

Thus explicit norm control contracts the imbalance on the continuous-time clock λt. In discrete training, the gradient-flow time corresponding to a learning-rate schedule (η_k) is approximately $\sum_k \eta_k$, so the natural L_2 clock is

$$\mathcal{T}_{L_2} = \lambda \sum_{k<K} \eta_k = K\eta\lambda \quad \text{for constant } \eta.$$

Coupled discrete weight decay. The corresponding discrete calculation makes the same clock explicit. Consider the two-factor update

$$U_{k+1} = (1 - \eta_k \lambda) U_k - \eta_k G_k V_k^\top, \qquad V_{k+1} = (1 - \eta_k \lambda) V_k - \eta_k U_k^\top G_k.$$

Let

$$c_k = 1 - \eta_k \lambda.$$

Proposition 4 (Discrete weight decay contracts imbalance up to finite-step leakage)**.** *For the coupled discrete update above,*

$$D_{k+1} = c_k^2 D_k + \eta_k^2 \left(V_k G_k^\top G_k V_k^\top - U_k^\top G_k G_k^\top U_k \right).$$

Proof. Expand

$$U_{k+1}^\top U_{k+1} = c_k^2 U_k^\top U_k - c_k \eta_k \left(V_k G_k^\top U_k + U_k^\top G_k V_k^\top \right) + \eta_k^2 V_k G_k^\top G_k V_k^\top.$$

Similarly,

$$V_{k+1} V_{k+1}^\top = c_k^2 V_k V_k^\top - c_k \eta_k \left(U_k^\top G_k V_k^\top + V_k G_k^\top U_k \right) + \eta_k^2 U_k^\top G_k G_k^\top U_k.$$

The first-order terms are identical and cancel after subtraction. Hence

$$\begin{aligned} D_{k+1} &= U_{k+1}^\top U_{k+1} - V_{k+1} V_{k+1}^\top \\ &= c_k^2 \left(U_k^\top U_k - V_k V_k^\top \right) + \eta_k^2 \left(V_k G_k^\top G_k V_k^\top - U_k^\top G_k G_k^\top U_k \right), \end{aligned}$$

which is the stated identity. □

Ignoring the second-order finite-step leakage, the multiplicative decay over K updates is

$$\prod_{k<K} (1 - \eta_k \lambda)^2.$$

When $\eta_k \lambda \ll 1$, this is approximated by

$$\prod_{k<K} (1 - \eta_k \lambda)^2 \approx \exp\left(-2\lambda \sum_{k<K} \eta_k \right).$$

This is the discrete counterpart of the continuous-time clock $\mathcal{T}_{L_2} = \lambda \sum_k \eta_k$.

Finite-step Euclidean updates leak imbalance at second order. Without explicit L_2 decay, simultaneous Euclidean gradient updates do not preserve imbalance exactly. However, the failure of conservation starts only at second order in the step size.

Proposition 5 (Two-factor finite-step leakage is second order)**.** *For the simultaneous update*

$$U_{k+1} = U_k - \eta_k G_k V_k^\top, \qquad V_{k+1} = V_k - \eta_k U_k^\top G_k,$$

one has the exact identity

$$D_{k+1} - D_k = \eta_k^2 \left(V_k G_k^\top G_k V_k^\top - U_k^\top G_k G_k^\top U_k \right).$$

In particular, all first-order terms in η_k cancel exactly.

Proof. This is the previous proposition with $\lambda = 0$, equivalently $c_k = 1$. Expanding directly,

$$U_{k+1}^\top U_{k+1} = U_k^\top U_k - \eta_k \left(V_k G_k^\top U_k + U_k^\top G_k V_k^\top \right) + \eta_k^2 V_k G_k^\top G_k V_k^\top,$$

and

$$V_{k+1} V_{k+1}^\top = V_k V_k^\top - \eta_k \left(U_k^\top G_k V_k^\top + V_k G_k^\top U_k \right) + \eta_k^2 U_k^\top G_k G_k^\top U_k.$$

The first-order terms are identical. Subtracting leaves exactly the second-order expression. □

This calculation proves an order statement, not a monotonicity statement: finite steps can move the conserved quantity at order η_k^2, but the sign of the movement depends on the current factors and gradient.

Deep linear gradient flow preserves all adjacent imbalances. The same conservation law holds at every hidden layer of a deep linear network. Let

$$F(W_1, \dots, W_L) = W_L W_{L-1} \cdots W_1, \qquad \mathcal{L}(W_1, \dots, W_L) = \phi(F),$$

where

$$W_j \in \mathbb{R}^{d_j \times d_{j-1}}, \qquad j = 1, \dots, L.$$

For $j = 1, \dots, L-1$, define the adjacent-layer imbalance

$$D_j = W_j W_j^\top - W_{j+1}^\top W_{j+1} \in \mathbb{R}^{d_j \times d_j}.$$

Proposition 6 (Deep linear gradient flow preserves adjacent imbalances)**.** *Assume ϕ is differentiable. Along Euclidean gradient flow*

$$\dot{W}_j = -\nabla_{W_j} \mathcal{L}, \qquad j = 1, \dots, L,$$

one has

$$\frac{d}{dt} D_j(t) = 0, \qquad j = 1, \dots, L-1.$$

With coupled continuous-time weight decay,

$$\dot{W}_j = -\nabla_{W_j} \mathcal{L} - \lambda W_j,$$

the imbalances satisfy

$$\frac{d}{dt} D_j(t) = -2\lambda D_j(t), \qquad D_j(t) = e^{-2\lambda t} D_j(0).$$

Proof. For each layer j, define

$$A_j = W_L W_{L-1} \cdots W_{j+1}, \qquad B_j = W_{j-1} \cdots W_1,$$

with the convention that empty products are identity matrices. Then

$$F = A_j W_j B_j.$$

Let

$$G = \nabla_F \phi(F).$$

By the chain rule,

$$\nabla_{W_j} \mathcal{L} = A_j^\top G B_j^\top.$$

We use the identities

$$A_j = A_{j+1} W_{j+1}, \qquad B_{j+1} = W_j B_j.$$

Let

$$H_j = \nabla_{W_j} \mathcal{L} = A_j^\top G B_j^\top.$$

Then

$$\begin{aligned} H_j W_j^\top &= A_j^\top G B_j^\top W_j^\top \\ &= W_{j+1}^\top A_{j+1}^\top G B_{j+1}^\top \\ &= W_{j+1}^\top H_{j+1}, \end{aligned}$$

and

$$\begin{aligned} W_j H_j^\top &= W_j B_j G^\top A_j \\ &= B_{j+1} G^\top A_{j+1} W_{j+1} \\ &= H_{j+1}^\top W_{j+1}. \end{aligned}$$

Therefore, under gradient flow,

$$\begin{aligned} \frac{d}{dt}(W_j W_j^\top) &= -H_j W_j^\top - W_j H_j^\top \\ &= -W_{j+1}^\top H_{j+1} - H_{j+1}^\top W_{j+1}. \end{aligned}$$

On the other hand,

$$\frac{d}{dt}(W_{j+1}^\top W_{j+1}) = -H_{j+1}^\top W_{j+1} - W_{j+1}^\top H_{j+1}.$$

The two derivatives are identical, so

$$\frac{d}{dt}\left(W_j W_j^\top - W_{j+1}^\top W_{j+1}\right) = 0.$$

This proves conservation.

With weight decay, the gradient-dependent terms are unchanged and still cancel. The additional terms are

$$-2\lambda W_j W_j^\top \qquad \text{and} \qquad -2\lambda W_{j+1}^\top W_{j+1},$$

so

$$\dot{D}_j = -2\lambda\left(W_j W_j^\top - W_{j+1}^\top W_{j+1}\right) = -2\lambda D_j.$$

Solving gives $D_j(t) = e^{-2\lambda t} D_j(0)$. □

Finite-step leakage in deep linear networks. The deep-linear finite-step calculation is the discrete analogue of the previous conservation law. The key point is that the layerwise gradient estimates must be induced by the same product-space gradient estimate.

For a product-space gradient estimate G_k, define

$$A_{j,k} = W_{L,k}\cdots W_{j+1,k}, \qquad B_{j,k} = W_{j-1,k}\cdots W_{1,k},$$

and

$$H_{j,k} = A_{j,k}^\top G_k B_{j,k}^\top.$$

Thus $H_{j,k}$ is the layer-j gradient estimate induced by G_k.

Proposition 7 (Deep finite-step leakage is second order)**.** *Consider simultaneous Euclidean updates*

$$W_{j,k+1} = W_{j,k} - \eta_k H_{j,k}, \qquad j = 1,\dots,L.$$

Then, for $j = 1,\dots,L-1$,

$$D_{j,k+1} - D_{j,k} = \eta_k^2\left(H_{j,k}H_{j,k}^\top - H_{j+1,k}^\top H_{j+1,k}\right).$$

In particular, all first-order terms in η_k cancel exactly.

Proof. Expanding the first term in $D_{j,k+1}$,

$$\begin{aligned} W_{j,k+1}W_{j,k+1}^\top = W_{j,k}W_{j,k}^\top &- \eta_k\left(H_{j,k}W_{j,k}^\top + W_{j,k}H_{j,k}^\top\right) \\ &+ \eta_k^2 H_{j,k}H_{j,k}^\top. \end{aligned}$$

Expanding the second term,

$$\begin{aligned} W_{j+1,k+1}^\top W_{j+1,k+1} = W_{j+1,k}^\top W_{j+1,k} &\\ - \eta_k\left(H_{j+1,k}^\top W_{j+1,k} + W_{j+1,k}^\top H_{j+1,k}\right) &\\ + \eta_k^2 H_{j+1,k}^\top H_{j+1,k}. & \end{aligned}$$

It remains to verify that the first-order terms are identical. Since

$$H_{j,k} = A_{j,k}^\top G_k B_{j,k}^\top,$$

and

$$A_{j,k} = A_{j+1,k}W_{j+1,k}, \qquad B_{j+1,k} = W_{j,k}B_{j,k},$$

we have

$$\begin{aligned} H_{j,k}W_{j,k}^\top &= A_{j,k}^\top G_k B_{j,k}^\top W_{j,k}^\top \\ &= W_{j+1,k}^\top A_{j+1,k}^\top G_k B_{j+1,k}^\top \\ &= W_{j+1,k}^\top H_{j+1,k}, \end{aligned}$$

and

$$\begin{aligned} W_{j,k}H_{j,k}^\top &= W_{j,k}B_{j,k}G_k^\top A_{j,k} \\ &= B_{j+1,k}G_k^\top A_{j+1,k}W_{j+1,k} \\ &= H_{j+1,k}^\top W_{j+1,k}. \end{aligned}$$

Therefore the entire first-order contribution in the expansion of $W_{j,k+1}W_{j,k+1}^\top$ is identical to the first-order contribution in the expansion of $W_{j+1,k+1}^\top W_{j+1,k+1}$. Subtracting the two expansions cancels all first-order terms and leaves exactly

$$D_{j,k+1} - D_{j,k} = \eta_k^2\left(H_{j,k}H_{j,k}^\top - H_{j+1,k}^\top H_{j+1,k}\right).$$

□

The minibatch-dependent leakage clock. The previous proposition is deterministic: it holds for any product-space gradient estimate G_k. To isolate the batch-size-dependent part, write

$$G_k = \bar{G}_k + \xi_k, \qquad \mathbb{E}_k[\xi_k] = 0, \qquad \mathbb{E}_k\|\xi_k\|_F^2 = O(1/b).$$

Define

$$\bar{H}_{j,k} = A_{j,k}^\top \bar{G}_k B_{j,k}^\top, \qquad \Delta_{j,k} = A_{j,k}^\top \xi_k B_{j,k}^\top.$$

Then

$$H_{j,k} = \bar{H}_{j,k} + \Delta_{j,k}, \qquad \mathbb{E}_k[\Delta_{j,k}] = 0.$$

Taking conditional expectation in the finite-step leakage identity gives

$$\begin{aligned}\mathbb{E}_k[D_{j,k+1} - D_{j,k}] = \eta_k^2 \left(\bar{H}_{j,k}\bar{H}_{j,k}^\top - \bar{H}_{j+1,k}^\top \bar{H}_{j+1,k}\right) \\ + \eta_k^2 \mathbb{E}_k\left[\Delta_{j,k}\Delta_{j,k}^\top - \Delta_{j+1,k}^\top \Delta_{j+1,k}\right].\end{aligned}$$

The first line is the deterministic full-batch finite-step leakage. It is second order in η_k, but it is not batch-size dependent. The second line is the minibatch-dependent stochastic correction.

On any portion of the trajectory where the operator norms of the factors are bounded, the assumption $\mathbb{E}_k\|\xi_k\|_F^2 = O(1/b)$ implies

$$\mathbb{E}_k\|\Delta_{j,k}\|_F^2 = O(1/b)$$

for every fixed layer j. Therefore the batch-dependent part of the conditional expected leakage is $O(\eta_k^2/b)$ per update. Summing over updates gives the stochastic finite-step clock

$$\mathcal{T}_{\mathrm{SGD}} = \frac{1}{b}\sum_{k<K} \eta_k^2 = \frac{K\eta^2}{b} \quad \text{for constant } \eta.$$

This clock measures the cumulative size of the minibatch-dependent second-order correction. It is an order statement; it does not assert that the correction has a fixed sign.

Generic preconditioning creates a first-order imbalance term. Adaptive methods apply a preconditioned update rather than the Euclidean gradient update. The following calculation shows exactly where the first-order conservation cancellation can fail.

Let

$$Q_{j,k} = P_{j,k}(H_{j,k}),$$

where $P_{j,k}$ is a layerwise or coordinatewise preconditioning map, possibly depending on the current iterate and optimizer state. Consider updates

$$W_{j,k+1} = W_{j,k} - \eta_k Q_{j,k}.$$

Proposition 8 (Preconditioning creates a first-order imbalance term)**.** *For the preconditioned update above,*

$$\begin{aligned}D_{j,k+1} - D_{j,k} = -\eta_k\Big[Q_{j,k}W_{j,k}^\top + W_{j,k}Q_{j,k}^\top - Q_{j+1,k}^\top W_{j+1,k} - W_{j+1,k}^\top Q_{j+1,k}\Big] \\ + \eta_k^2\left[Q_{j,k}Q_{j,k}^\top - Q_{j+1,k}^\top Q_{j+1,k}\right].\end{aligned}$$

For Euclidean gradient descent, $Q_{j,k} = H_{j,k}$, and the first-order bracket vanishes by the chain-rule identity proved above. For a general adaptive preconditioner, $Q_{j,k}$ need not satisfy that identity, so imbalance can change at order η_k per update.

Proof. Expanding the first adjacent Gram matrix,

$$\begin{aligned}W_{j,k+1}W_{j,k+1}^\top = W_{j,k}W_{j,k}^\top - \eta_k\left(Q_{j,k}W_{j,k}^\top + W_{j,k}Q_{j,k}^\top\right) \\ + \eta_k^2 Q_{j,k}Q_{j,k}^\top.\end{aligned}$$

Expanding the second adjacent Gram matrix,

$$\begin{aligned}W_{j+1,k+1}^\top W_{j+1,k+1} = W_{j+1,k}^\top W_{j+1,k} - \eta_k\left(Q_{j+1,k}^\top W_{j+1,k} + W_{j+1,k}^\top Q_{j+1,k}\right) \\ + \eta_k^2 Q_{j+1,k}^\top Q_{j+1,k}.\end{aligned}$$

Subtracting the second expansion from the first gives the stated identity. If $Q_{j,k} = H_{j,k}$, the first-order bracket is zero by the identities

$$H_{j,k}W_{j,k}^\top = W_{j+1,k}^\top H_{j+1,k}, \qquad W_{j,k}H_{j,k}^\top = H_{j+1,k}^\top W_{j+1,k}$$

proved in the finite-step Euclidean case. □

This calculation proves that the order of imbalance movement can change from second order to first order under preconditioning. It does not claim that every adaptive optimizer monotonically contracts imbalance; rather, it identifies the algebraic cancellation that Euclidean gradient flow and Euclidean finite-step updates enjoy, and shows that generic preconditioning need not preserve it.

Scalar example of first-order breaking. The first-order effect is already visible in the scalar two-factor model. Let $p = ab$ and $g = \phi'(ab)$. The Euclidean gradients are

$$h_a = gb, \qquad h_b = ga.$$

Consider preconditioned updates

$$a_{k+1} = a_k - \eta\alpha_k g b_k, \qquad b_{k+1} = b_k - \eta\beta_k g a_k,$$

where $\alpha_k, \beta_k > 0$. Then

$$\begin{aligned} D_{k+1} - D_k &= a_{k+1}^2 - b_{k+1}^2 - (a_k^2 - b_k^2) \\ &= -2\eta g a_k b_k(\alpha_k - \beta_k) + \eta^2 g^2 \left(\alpha_k^2 b_k^2 - \beta_k^2 a_k^2\right). \end{aligned}$$

Thus, whenever

$$g a_k b_k(\alpha_k - \beta_k) \neq 0,$$

the imbalance changes at first order in η. Euclidean gradient descent corresponds to $\alpha_k = \beta_k = 1$, in which case the first-order term vanishes.

This scalar example justifies the adaptive clock

$$\mathcal{T}_{\mathrm{adapt}} = \sum_{k<K} \eta_k = K\eta \quad \text{for constant } \eta$$

as the natural scale on which preconditioned updates can move initialization-dependent imbalance.

Summary. In this minimal homogeneous model, Euclidean gradient flow exactly conserves initialization-dependent imbalances. In the scalar two-factor case, the conserved imbalance directly fixes the final factor norm among solutions with the same product, giving a precise notion of radial initialization memory. Coupled L_2 decay contracts the imbalance on the clock

$$\mathcal{T}_{L_2} = \lambda \sum_{k<K} \eta_k.$$

Euclidean finite-step updates break the conservation law only at second order; the minibatch-dependent part of the conditional expected leakage accumulates on the clock

$$\mathcal{T}_{\mathrm{SGD}} = \frac{1}{b} \sum_{k<K} \eta_k^2.$$

Generic adaptive preconditioning can break the conservation law at first order, giving the movement clock

$$\mathcal{T}_{\mathrm{adapt}} = \sum_{k<K} \eta_k.$$

These calculations are not a theorem for nonlinear BatchNorm ResNets: they do not prove monotone forgetting, nor do they determine test accuracy. Rather, they identify the conservation law and the optimizer-dependent orders at which different training mechanisms can move or contract initialization-dependent quantities. This mechanism matches the empirical hierarchy observed in Section 4: gradient-flow-like low-LR SGD remembers initialization, increasing the learning rate or adding explicit L_2 can reduce memory by enlarging the relevant clocks, and adaptive methods erase initialization-scale dependence much faster in the diagnostic grid.

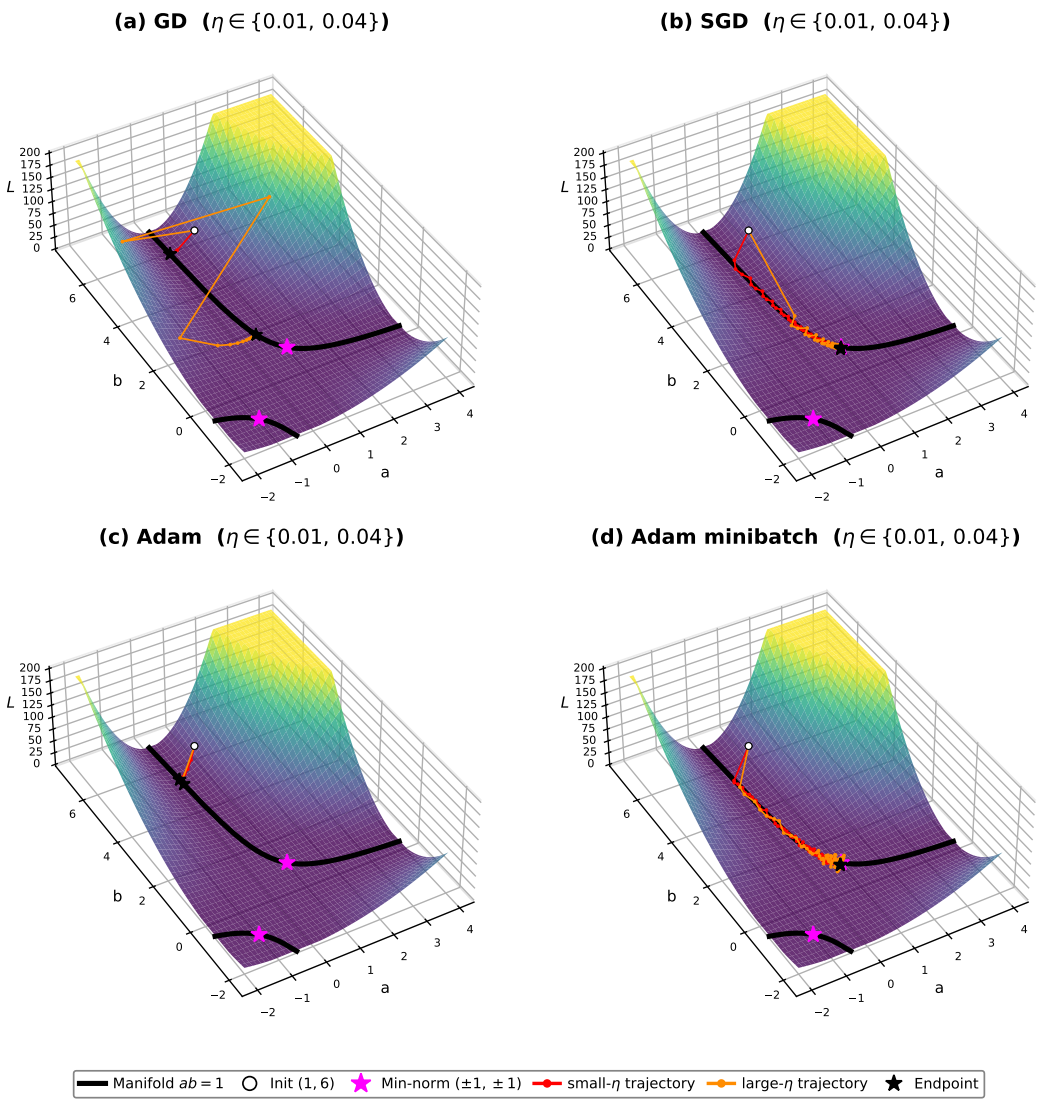


Figure 9: **Optimizer trajectories on the scalar loss $(ab-1)^2$, initialization** $(a_0, b_0) = (1, 6)$**.** All four panels use the same two learning rates $\eta \in \{0.01, 0.04\}$ (red and orange, respectively); only the optimizer changes. The black curve is the manifold of global minima $ab = 1$; magenta stars mark the minimum-norm solutions $(\pm 1, \pm 1)$. **(a)** Gradient descent (20 steps): the small-η trajectory stays near the initialization ($D \approx -35$, close to gradient-flow conservation), while the large-η trajectory moves toward lower norm ($D \approx -3$), illustrating the $O(\eta^2 K)$ finite-step leakage. **(b)** SGD with cyclic minibatch targets $\{0, 1, 2\}$ (up to 50,000 steps): both learning rates converge to the minimum-norm solution ($D \approx 0$), consistent with the $\mathcal{T}_{\mathrm{SGD}} = K\eta^2/b$ clock. **(c)** Full-batch Adam (5,000 steps): the preconditioner moves D from -35 toward ≈ -27, confirming first-order $O(\eta)$ movement of the imbalance. However, convergence stalls because the full-batch gradient $g = 2(ab-1) \to 0$ at the manifold. This implies that the numerator goes to 0 but the denominator is lowerbounded by $\epsilon > 0$. Thus the $O(\eta)$ coefficient vanishes and D freezes at a nonzero value. **(d)** Minibatch Adam (up to 200,000 steps): adding stochasticity restores forgetting—both learning rates reach $D \approx 0$. Individual minibatch gradients remain nonzero at $ab = 1$, keeping the adaptive clock active.

H Further Related Work

Purpose of this appendix. The main text cites only the papers needed to motivate the central definitions and claims. This appendix gives the broader historical map. The question "why do neural networks not overfit?" has been approached through several partially overlapping mechanisms: classical capacity control, margins and norms, flatness, PAC-Bayes and compression, interpolation and benign overfitting, function-space priors, spectral and geometric simplicity, stable signal propagation at initialization, and optimizer-dependent implicit bias. Our paper sits at the intersection of two views. On one hand, random architectures and initialization schemes define nontrivial priors over functions. On the other hand, the training pipeline transforms, preserves, or erases those priors. Initialization memory is our proposed diagnostic for measuring how much of the initial prior survives training.

The common thread. The literature can be read as a progression from static explanations to dynamical explanations:

$$\text{hypothesis class} \quad \rightarrow \quad \text{trained predictor} \quad \rightarrow \quad \text{training pipeline} \quad \rightarrow \quad \text{memory of initialization}.$$

Classical theory controls the size of the class; post-hoc complexity measures study the final predictor; implicit-bias work studies the optimizer's selection rule; simplicity-bias work studies the prior at initialization. Our contribution is to connect the last two: we ask whether the initialization-induced prior remains visible after optimization.

H.1 From capacity to pipeline-dependent generalization

Classical complexity control. Classical learning theory explains generalization by controlling effective capacity. VC theory and statistical learning theory formalize uniform convergence in terms of hypothesis-class complexity [43, 44]. Later refinements replaced raw parameter counting by data- and norm-dependent quantities, including margins, Rademacher complexities, and weight norms [45–48]. This line is historically essential because it already separates the number of trainable parameters from the actual complexity of the learned predictor. However, worst-case capacity control alone does not explain why highly overparameterized networks can fit random labels yet generalize well on natural labels.

The random-label challenge. The modern crisis point was the observation that standard deep networks can interpolate both natural labels and random labels under essentially the same architecture and optimization machinery [1, 49]. This shifted attention away from the hypothesis class alone and toward the complete pipeline that selects one interpolating solution among many. Follow-up work showed that deep networks tend to learn simple or structured patterns before memorizing idiosyncratic noise [50, 51]. The relevant object is therefore not merely the set of functions representable by the architecture, but the algorithmic process by which training selects one of them.

Stability and training time. Algorithmic stability gives one route from optimization to generalization. [52] showed that stochastic gradient methods with controlled numbers of steps can be stable, giving a formal sense in which the trajectory itself participates in capacity control. This perspective is especially relevant for our work because initialization memory is also trajectory-dependent: a fixed architecture may either preserve or erase initialization scale depending on update count, batch size, learning rate, and regularization.

H.2 Post-hoc explanations: norms, margins, flatness, PAC-Bayes, and compression

Norm and margin explanations. A large body of work attempts to explain generalization by measuring properties of the trained predictor rather than the raw hypothesis class. Norm-based and margin-based bounds for neural networks include path-norm, spectral-norm, and margin-normalized quantities [47, 48, 53–55]. Large-scale empirical studies have compared many proposed complexity measures and found that some correlate with generalization better than others, while many fail outside narrow settings [56]. Related norm-based generalization results connect products of layerwise Frobenius norms to Rademacher complexity and generalization bounds [57, 58]. These results provide complementary theoretical motivation for studying weight-scale dependence, although our experiments track the concatenated Frobenius norm rather than the product of layerwise norms.

Flatness and sharpness. The flat-minima view dates back to [59] and became central again in the large-batch generalization debate [60]. Because sharpness is not invariant to parameter rescaling, later work showed that naive sharpness measures can be misleading in deep networks [61]. Sharpness-aware minimization was subsequently proposed as an explicit algorithmic route to flatter solutions [62]. In our setting, BatchNorm scale invariance makes this issue particularly relevant: rescaling weights can leave the represented function nearly unchanged while substantially changing the optimization geometry.

PAC-Bayes and compression. PAC-Bayes gives another way to turn algorithmic or posterior concentration into generalization statements [54, 63–65]. Compression-based analyses similarly argue that trained networks generalize when they can be compressed without a large loss of performance [66]. These approaches support a broad message: overparameterization is not itself fatal if the training procedure selects a small or compressible subset of effective functions. The initialization-memory viewpoint adds a dynamical question: whether the selected subset is still determined by the initial function prior, or whether training has overwritten it.

H.3 Interpolation, double descent, and benign overfitting

Interpolation is not necessarily overfitting. The double-descent literature recast interpolation as a regime to be understood rather than automatically avoided. [67] proposed that the classical bias–variance curve extends into a second descent beyond the interpolation threshold, and [68] demonstrated double descent as a function of model size, data size, and training time in modern deep learning systems. Benign-overfitting theory shows that, even in simple linear models, interpolating predictors can generalize under appropriate spectral conditions on the data distribution [69]. Our experiments refine this picture for deep networks: interpolation of the training set is not the same as forgetting initialization. Low-learning-rate SGD can interpolate while retaining large initialization memory.

H.4 Function-space priors and simplicity bias at initialization

Bayesian and infinite-width priors. The idea that random neural networks define a nontrivial function predates the current simplicity-bias literature. [70] studied Bayesian neural networks and the connection between random networks and Gaussian processes; [71] analyzed computation with infinite neural networks; later work made Gaussian-process and neural-tangent-kernel limits central tools for analyzing wide networks [72, 73]. These function-space limits show that architecture and initialization define a prior over functions before training.

Algorithmic simplicity bias of parameter–function maps. A more recent line argues that many parameter–function maps are intrinsically biased toward simple outputs. [74] showed that broad classes of input–output maps can be strongly biased toward low-complexity outputs. [6] applied this idea to neural networks, arguing that the parameter–function map of deep networks is biased toward simple functions and that this can yield PAC-Bayesian explanations of generalization. [75] proved simplicity-bias results for random wide ReLU networks on Boolean inputs. Related work studies a priori biases toward low-entropy Boolean functions [42], and [7] sharpened the claim that deep networks have an inbuilt Occam-like bias. These works establish initialization and architecture as genuine sources of inductive bias. They do not, however, determine how much of that bias survives optimization. That survival question is the focus of our paper.

Algorithmic complexity and Lempel–Ziv measures. Several simplicity-bias papers operationalize simplicity through computable proxies for Kolmogorov complexity. The relevant background includes Lempel–Ziv complexity and universal compression [76, 77], as well as the broader theory of Kolmogorov complexity [78]. In the neural-network setting, these measures are often used not because they are the only possible definition of simplicity, but because they provide a concrete way to compare the output complexity of functions induced by random parameter draws. Our work is agnostic about which simplicity metric is definitive: the question we isolate is whether any initialization-induced simplicity bias remains visible after training.

Transformer simplicity bias. Simplicity bias is not restricted to convolutional or fully connected networks. Work on Transformers has studied biases toward sparse or low-sensitivity Boolean

functions [79–81]. Recent work further argues that Transformers already contain structural inductive biases at random initialization [28]. These results are relevant to the broader question of whether large modern architectures prefer simple rules, but they do not by themselves determine whether training preserves the initialization-induced prior. Initialization memory is intended to measure this survival question directly.

H.5 Technical notions of simplicity and their training dynamics

Frequency, sensitivity, regions, and geometry. The literature uses several inequivalent notions of simplicity. Spectral-bias or frequency-principle results show that neural networks often learn low-frequency components before high-frequency components [82]. Sensitivity-based work measures input-output Jacobians and local robustness around the data manifold [83]. Region-counting work studies the number of linear regions or activation patterns in ReLU networks, showing that typical trained or initialized networks use far fewer regions than worst-case expressivity bounds allow [84, 85]. Geometric-complexity work measures variation of the learned function through a Dirichlet-energy-like quantity and shows that many regularizers control this geometry [86]. These notions are not identical, but they share a common theme: practical deep networks tend to realize functions much simpler than the worst-case architecture permits.

Simplicity along training. Recent work has begun to study not only the simplicity of random networks or final predictors, but also the order in which different complexities are learned. Neural networks trained with SGD can learn distributions of increasing complexity over time [87]. Two-layer ReLU models exhibit simplicity bias and optimization thresholds that can be analyzed in controlled settings [88]. Saddle-to-saddle dynamics have also been proposed as a mechanism for simplicity bias across architectures [89]. These works are close in spirit to our paper because they treat simplicity as a dynamical phenomenon rather than a static property of the architecture alone.

Support learning as implicit regularization. A particularly relevant recent result is [35], who study how neural networks learn the support of the target function. They show that mini-batch SGD can shrink irrelevant input weights in the first layer, while full-batch gradient descent requires explicit regularization to obtain the same effect. Their mechanism is a second-order implicit regularization effect of SGD that depends on step size and batch size. This is directly aligned with our perspective: stochastic finite-step effects can erase or restructure parts of the initialization-dependent geometry, whereas gradient-flow-like dynamics may preserve them.

H.6 Initialization as a dynamical boundary condition

Basins and classical dynamical systems. The informal intuition that initialization chooses a basin of attraction comes from classical dynamical systems: the long-time behavior of an autonomous flow can depend strongly on its initial condition and on the basins of attraction of stable invariant sets [2, 3]. In neural-network training, this language is only an analogy: high-dimensional nonconvex losses, stochastic updates, normalization layers, and time-varying learning rates make the dynamics much richer than a simple gradient flow. Nevertheless, the basin viewpoint motivates the empirical question we study: whether the final trained predictor remains measurably dependent on its initial condition.

Initialization in implicit-bias theory. Several theoretical works show that initialization can affect convergence and implicit bias, even in simplified linear or homogeneous models. [4] analyzes the explicit role of initialization in overparameterized linear networks. [5] studies how initialization affects the implicit bias of deep linear networks. These works complement our minimal linear-network calculation: they show that initialization is not merely a nuisance parameter, but can control the geometry of the solution selected by gradient-based training.

H.7 Initialization for trainability and signal propagation

Variance propagation and dynamical isometry. A historically distinct literature treats initialization primarily as a trainability device. Xavier and Kaiming initializations were designed to stabilize forward and backward signal propagation through deep networks [9, 10]. Deep-linear-network analyses showed how initialization affects learning dynamics and training plateaus [90]. Mean-field

and random-matrix analyses of signal propagation identified ordered and chaotic regimes in random deep networks [11, 12], while dynamical-isometry results emphasized controlling the singular-value distribution of the input-output Jacobian at initialization [13, 16, 17]. Hanin and collaborators studied how architecture and initialization determine trainability, including the onset of exploding and vanishing gradients [14, 15]. This line explains why initialization can enable training without necessarily claiming that the final predictor should remain close to the initial prior.

Beyond signal propagation. Signal propagation is not the only relevant initialization criterion. [91] argue that feature diversity at initialization can matter beyond the stability of forward and backward signals. This is conceptually close to our motivation: initialization can affect not only whether training is numerically stable, but also what information and geometry are available to the optimizer early in training.

Dynamical mean-field and kernel-evolution perspectives. Mean-field and dynamical field theories provide another route to analyzing training from random initialization. Work on kernel evolution in wide neural network models, how predictors move away from their initial random-kernel description during feature learning [20, 21]. Recent analyses of deep linear networks from random initialization connect data, width, depth, and hyperparameter transfer [22], while feature-learning infinite limits yield adaptive kernel predictors [23]. These works are part of the same broader shift from static initialization priors to the dynamics by which training modifies those priors.

Parameterization and scaling. Modern scaling theory further emphasizes parameterization as part of the training pipeline. Tensor Programs and maximal-update parameterization show that stable feature learning and hyperparameter transfer require carefully chosen width scalings [18, 19, 92]. In this view, initialization is tightly coupled to learning-rate scales and update magnitudes. Our experiments echo this philosophy in a finite-network setting: initialization scale, learning rate, batch size, and regularization jointly determine whether the initial condition is remembered.

H.8 Normalization, scale invariance, and effective learning rates

BatchNorm and scale-invariant optimization. Batch normalization was introduced as a way to accelerate training and reduce sensitivity to initialization [93]. Subsequent theory emphasized that normalized networks contain scale-invariant parameter blocks, for which rescaling weights can leave the represented function nearly unchanged while altering the effective optimization dynamics [94, 95]. The intrinsic-learning-rate view is particularly close to our mechanism: for scale-invariant parameters, the effective angular step size scales inversely with the squared norm. This makes initialization scale a useful tracer in BatchNorm networks, because it is partly hidden from the forward map while remaining visible to the optimizer.

H.9 Backward error analysis and modified equations

Historical context. Backward error analysis is a classical tool in numerical analysis for understanding what a numerical algorithm actually solves. Its linear-algebra form is closely associated with Wilkinson's work on roundoff error and eigenvalue computations: rather than measuring only the forward error with respect to the original problem, one asks whether the computed output is the exact solution of a nearby perturbed problem [96–98]. For time-stepping methods, the analogous question is whether a discrete integrator is the exact time-h map of a nearby differential equation. This is the method of modified equations. Foundational analyses by [36] and subsequent developments in geometric numerical integration made this viewpoint central for understanding the qualitative behavior of ODE solvers [37, 99]. In Hamiltonian and structure-preserving integration, backward error analysis explains why finite-step methods may preserve modified invariants or modified Hamiltonians for long times, even when they do not exactly solve the original continuous system.

Stochastic modified equations. A stochastic analogue was developed for numerical SDEs and weak approximation. Modified-equation methods for SDEs show that a discrete stochastic scheme can be interpreted, in a weak sense, as solving a nearby stochastic dynamics whose drift, diffusion, or invariant measure has been perturbed by the step size [100–102]. This line is important for modern optimization because constant-step stochastic algorithms are not merely noisy versions of gradient

flow; their invariant behavior and local stability can depend on finite-step corrections. Uniform-in-time weak-error analyses for SGD make this connection explicit by using tools motivated by backward error analysis for stochastic differential equations [103].

BEA in optimization and machine learning. The ML use of modified-equation ideas began by treating stochastic gradient algorithms as discrete dynamical systems whose continuous-time approximations should include step-size-dependent correction terms. [104] introduced stochastic modified equations for adaptive stochastic gradient algorithms, and [105] developed the mathematical foundations for SGD, momentum SGD, and stochastic Nesterov methods. In deep-learning generalization, [32] used backward error analysis to show that finite learning rates in gradient descent induce an implicit gradient regularization term, while [33, 34] extended this perspective to SGD and derived minibatch-dependent finite-step regularization. Subsequent work used related discretization-error or modified-dynamics viewpoints to study deep-network gradient descent [106], momentum [38, 107, 108], stochastic coordinate descent [109], and the implicit bias of Adam and RMSProp [39, 110, 111]. These works support the conceptual move made in our paper: raw epoch count is not the natural dynamical unit. The effects that erase initialization memory are governed by accumulated step-size, stochasticity, preconditioning, and regularization timescales, such as

$$\frac{1}{b}\sum_k \eta_k^2, \qquad \lambda\sum_k \eta_k, \qquad \sum_k \eta_k.$$

In this sense, backward error analysis provides the mathematical language for our "forgetting-time" interpretation: finite-step optimization follows a nearby dynamics, and the perturbation terms of that nearby dynamics can act as implicit regularizers that overwrite parts of the initialization-induced geometry.

H.10 Optimization as solution selection and forgetting

Implicit bias of gradient descent. A large body of literature shows that optimization selects among interpolating solutions. In separable linear classification, gradient descent on exponential-type losses converges in direction to the max-margin classifier [112]. For linear convolutional networks, the implicit bias depends on architecture and depth [113]. Deep matrix factorization exhibits an implicit tendency toward low-rank solutions [40]. For homogeneous neural networks, gradient descent can be related to margin maximization [114]. These works support the view that optimization does not merely minimize loss; it chooses a particular geometry among many interpolating predictors.

SGD noise, batch size, and finite-step effects. The stochasticity and discretization of SGD also affect the selected solution. Large-batch training was linked to sharp minima and generalization gaps [60], while other work argued that the number of updates and high-learning-rate phase can be central to closing this gap [115]. Constant-step SGD can be viewed as a stochastic process with an approximate stationary distribution [116]. As reviewed in Appendix H.9, backward error analysis gives a complementary view: finite-step gradient methods approximately follow the gradient flow of a modified objective [32], and the SGD correction can be minibatch-dependent [33]. These works motivate our timescale language: raw epoch count is not the natural unit of forgetting; update count, learning rate, batch size, and explicit decay determine how quickly training leaves the initialization-controlled regime.

Adaptive methods. Adaptive optimizers use a geometry different from Euclidean gradient descent. [117] showed that adaptive methods can select different solutions from SGD and can generalize differently even when they optimize training loss well. Our results are consistent with this broader message: Adam-family methods do not merely train faster in our grid; they erase initialization-scale dependence more readily. We interpret this as a forgetting property of the optimizer geometry, not as a universal claim that Adam is always preferable to SGD.

H.11 Random seeds and initialization effects in language models

Fine-tuning and pretraining variability. Recent NLP work makes seed dependence concrete in large pretrained models. [24] showed that fine-tuning BERT can vary substantially across random seeds, with both weight initialization and data order contributing to downstream variance. For decoder-only pretraining, [25] introduced multiple Pythia-style pretraining runs across seeds and model sizes,

finding broadly stable dynamics but also identifiable outlier runs. [26] studied convergence and divergence of language models across random seeds through token-level distributional comparisons. [27] go further, showing that models can retain fingerprints of their training seed. These papers motivate the same broad question as ours in a different setting: which parts of final performance are due to the initial condition, and which are erased by training?

Controlled language-model training and architecture design. The Physics of Language Models series studies controlled mechanisms in language-model training, including knowledge storage and extraction [29] and architecture design choices in transformer training [30]. These works are not direct substitutes for an initialization-scale sweep, but they reinforce the broader point that large-scale model behavior is shaped by a coupled training pipeline rather than by architecture alone.

Why our main setting is deliberately controlled. Our main grid uses CIFAR-10 BatchNorm ResNets because the goal is to sweep initialization scale, optimizer, batch size, learning rate, regularization, depth, and seeds extensively; we complement it with a GPT-2 (124M) pretraining study on OpenWebText (§2, Table 1) as an initial check that the SGD-vs-adaptive contrast transfers to LayerNorm transformers. Neither study is meant to replace evidence at the scale of modern LLMs. Instead, the controlled setting isolates the mechanism: initialization can be a function prior, a trainability device, and a boundary condition for optimization; the observed performance effect depends on whether the training pipeline remembers that boundary condition.

H.12 Position of this work

The missing dynamical link. Prior work has studied architecture-induced priors, simplicity bias, trainability at initialization, optimizer implicit bias, scale-invariant optimization, and large-batch effects. We connect these themes through one measurable quantity,

$$\mathsf{Mem}_{\mathrm{acc}}(\mathcal{R}, K),$$

The dependence of the returned predictor on the initialization scale under a fixed training procedure $\mathcal{R}$. The resulting statement is neither that initialization always matters nor that it disappears universally. Initialization matters when the training dynamics remember it.

Summary. The literature above identifies many reasons why neural networks may avoid overfitting: classical capacity control, margins and norms, flat minima, compression, Bayesian priors, benign overfitting, function-space simplicity, spectral/geometric bias, stable signal propagation, and optimizer-induced implicit bias. Our contribution is not to replace these explanations. It is to add a dynamic measurement that connects them:

$$\text{initialization prior} \quad \xrightarrow{\text{training pipeline}} \quad \text{trained predictor.}$$

Initialization memory measures how much of the left-hand side survives the arrow. In this sense, initialization is neither universally decisive nor universally irrelevant. It matters exactly when the training dynamics remember it.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: the abstract and introduction make three main claims, each supported in the body of the paper: (i) low-learning-rate SGD retains strong initialization memory while Adam-family methods largely erase it—this is demonstrated by the controlled phase diagram across optimizers, batch sizes (Section 3, Figures 1–2); (ii) interpolation and long training do not imply forgetting—a $5{,}000$-epoch low-LR SGD control leaves the spread essentially unchanged, and SGD can be made to forget only with larger learning rates or explicit L_2 norm control (Section 3.2, Figure 4); (iii) forgetting is governed by cumulative optimizer-dependent timescales ($K\eta^2/b$, $K\eta\lambda$, $K\eta$), supported by numerical timescale arithmetic, the $|\beta(t)|$ data collapse across five batch sizes, and a minimal linear-network conservation-law model (Section 4, Figure 6, Appendix G). Scope is stated explicitly: the main grid uses CIFAR-10 BatchNorm ResNets, with LayerNorm/NoNorm ResNet checks and a GPT-2 (124M)/OpenWebText pretraining study as generality checks; Mem probes only the initialization *scale*, and the timescale expressions are described as organizing principles rather than complete nonlinear theorems.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Limitations are discussed explicitly in the conclusion (Section 5, "Implications, limitations, and future work" paragraph). The paper acknowledges: (i) the main grid uses BatchNorm ResNets on CIFAR-10; normalization variants and the GPT-2 (124M) study are smaller generality checks (5 and 3 seeds); (ii) the timescale expressions ($K\eta^2/b$, $K\eta\lambda$, $K\eta$) are organizing principles derived from linear and scale-invariant arguments, not complete nonlinear theorems—a limitation shared with the implicit-bias literature the paper builds on; (iii) the σ_w probe is one-dimensional, leaving bias distributions, layer-wise scalings, and μP parameterizations unstudied; (iv) Mem is an operational, scale-only diagnostic and does not measure geometric erasure of the initial parameters; the best-recipe comparison uses only two σ_w endpoints and the $5{,}000$-epoch control a single seed; (v) extensions to ImageNet, ViTs, and larger-scale LLM pretraining are left to future work.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.

- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

   Justification: All theoretical results are stated and proved in Appendix G. The appendix contains eight propositions, each with explicit assumptions and a self-contained proof: (1) two-factor gradient flow preserves imbalance (Euclidean gradient flow, differentiable loss); (2) conserved imbalance fixes the final norm in the scalar model (convergence to a point with product $p_\star$); (3) continuous-time L_2 decay contracts imbalance ($\dot{D} = -2\lambda D$); (4) discrete weight decay contracts imbalance up to finite-step leakage (coupled simultaneous updates); (5) two-factor finite-step leakage is second order in η_k (simultaneous Euclidean updates, no weight decay); (6) deep linear gradient flow preserves all adjacent imbalances (differentiable loss, arbitrary depth); (7) deep finite-step leakage is second order (simultaneous updates, shared product-space gradient estimate—this assumption is stated explicitly as essential); (8) preconditioning creates a first-order imbalance term (generic layerwise preconditioner $Q_{j,k} = P_{j,k}(H_{j,k})$). The stochastic extension uses the standard decomposition $G_k = \bar{G}_k + \xi_k$ with $\mathbb{E}_k[\xi_k] = 0$ and $\mathbb{E}_k\|\xi_k\|_F^2 = O(1/b)$, stated at the start of the appendix. The paper clearly distinguishes these order-of-magnitude results from a full convergence theorem: "These calculations are not a theorem for nonlinear BatchNorm ResNets: they do not prove monotone forgetting, nor do they determine test accuracy" (Appendix G, Summary paragraph).

   Guidelines:

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

4. **Experimental result reproducibility**

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

Answer: [Yes]

Justification: The datasets (CIFAR-10 and OpenWebText) are publicly available. The GPT-2 (124M) OpenWebText setup, including architecture, initialization scaling, token budget, Adam learning rate, and seed count, is described in Appendix A. Seed counts differ by experiment (10 for the main grid, 5 for the depth, normalization, and extended sweeps, 3 for GPT-2) and are stated with each table. The paper fully discloses all architectures (ResNet-9, ResNet-56, ResNet-110), optimizers (SGD, SGD+momentum, Adam, AdamW, Muon), hyperparameters (learning rates, batch sizes, weight decay coefficients, momentum parameters), training budgets (300 and 5,000 epochs), learning-rate schedules (cosine decay), initialization-scale sweep ($\sigma_w \in \{0.10, 0.20, \ldots, 2.50\}$), number of seeds per configuration (n=10), checkpoint-selection rule (best validation loss), and all diagnostic definitions (initialization memory Mem, interpolation epoch τ_{interp}, repair gap Δ_{repair}, sensitivity coefficient $|\beta(t)|$, forgetting timescales $\mathcal{T}_{\mathrm{SGD}}$, $\mathcal{T}_{L_2}$, $\mathcal{T}_{\mathrm{adapt}}$) in Section 2 and Appendix A. Ablation configurations (learning rate, L_2, normalization, augmentation, depth) are detailed in Appendices C–E. The toy-model optimizer implementations follow the standard definitions given in Appendix G. This provides sufficient detail to faithfully reproduce the main results and conclusions.

Guidelines:

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

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

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

   Answer: [Yes]

Justification: Code for the CIFAR-10 ResNet experiments, extended sweeps, associated analysis scripts, and the GPT-2 sweep driver is publicly available at `https://github.com/mohuadas-neurips/Initialization-Bias-Deep-Neural-Network`. The GPT-2 experiments build on nanoGPT.

Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Section 2 reports all training and evaluation details: architectures (ResNet-9, ResNet-56, ResNet-110, R9-AvgPool), optimizers (SGD, SGD+momentum, Adam, AdamW, Muon), hyperparameters ($\eta_0{=}10^{-3}$ with cosine decay, batch sizes $b \in \{16, 32, 64, 128, 256\}$, weight decay values, momentum coefficients), initialization-scale sweep ($\sigma_w \in \{0.10, 0.20, \ldots, 2.50\}$), number of seeds ($n{=}10$), training horizon (300 epochs, with a 5,000-epoch control), and checkpoint-selection rule (best validation loss). The main grid is explicitly described as diagnostic rather than best-tuned; targeted ablations over learning rate, L_2 regularization, normalization, augmentation, and depth are reported in Appendices C–E. Full per-optimizer hyperparameter tables are provided in Appendix A. The GPT-2 (124M) OpenWebText setup, including architecture, initialization scaling, token budget, Adam learning rate, and seed counts, is described in Appendix A. Seed counts differ by experiment (10 for the main grid, 5 for the depth, normalization, and extended sweeps, 3 for GPT-2) and are stated with each table.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: All main experiments are run with $n{=}10$ independent seeds per configuration. Figure 1 reports means with shaded 10^{th}–90^{th} percentile bands across seeds. The

initialization-memory diagnostic (Mem) is defined as the max–min spread of seed-averaged test accuracy across the σ_w sweep, providing a robust summary statistic. The sensitivity coefficient $|\beta(t)|$ in Figure 6 is estimated by ordinary least-squares regression across the 25-point σ_w grid (10 seeds each). Appendix ablations (normalization, augmentation) use 5 seeds. Numerical values (test accuracy, train accuracy, interpolation epochs, spreads) are tabulated in Appendices B–F with per-configuration seed counts stated.

Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: The vision experiments are implemented in Keras 3 / TensorFlow, while the GPT-2 experiments use nanoGPT / PyTorch. Experiments were conducted on NVIDIA GPUs. The main experimental grid comprises $6{,}250$+ runs across optimizers, batch sizes, initialization scales, depths, and seeds. Individual runs use standard ResNet architectures on CIFAR-10 (300 or 5,000 epochs), which are modest in scale.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The research conforms to the NeurIPS Code of Ethics. The work is methodological—studying how initialization memory depends on the training pipeline in deep networks—and uses only the standard public CIFAR-10 and OpenWebText datasets. It involves no human subjects, no sensitive data, and presents no foreseeable ethical issues.

Guidelines:

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

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: This work is methodological: it studies when and why training pipelines preserve or erase initialization-induced bias in deep networks. The contributions are empirical and diagnostic, using only the standard CIFAR-10 and OpenWebText benchmark. The paper introduces no deployment-ready systems, no new datasets, and no application-specific methods, so we identify no direct path to societal harm.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper releases no pre-trained generative models, scraped datasets, large language models, or other assets with high misuse potential.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: CIFAR-10 and OpenWebText are cited and used under their standard terms. All software (Keras 3, TensorFlow, NumPy, nanoGPT and PyTorch) is open-source. Optimizers and prior methods are cited at first use. No license restrictions are violated.

    Guidelines:

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

13. **New assets**

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

    Answer: [N/A]

    Justification: The paper introduces no new datasets, pre-trained models, or code packages. All experimental protocols and diagnostic definitions are fully specified in the paper and appendices so they can be reimplemented from the text alone.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper involves no crowdsourcing or human-subjects research.

Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper involves no human-subjects research.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: GPT-2 (124M) is trained from scratch as an object of study; no LLM is used as a component of the method or its evaluation.

    Guidelines:

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