Two Levels of Learned Structure

Why Checkpoints Are Not Complete Artifacts

[Author Name]

January 19, 2026

Abstract

Trained neural networks admit two distinct levels of description. **Level 1** (Behavioral State) is defined by the model weights and fully determines inference. **Level 2** (Modification Semantics) determines which operations on the model preserve coherent function. Level 2 depends on training trajectory information that cannot be recovered from weights alone.

We prove that Level 2 is distinct from Level 1 with a minimal experiment. We resume training on a converged 4-layer transformer under two conditions: one preserving optimizer state, one resetting it. The weights are identical at the moment of intervention. The outcomes diverge: the preserved condition remains stable at 99%, while the reset condition collapses to 10%–60% accuracy within steps. Same weights, different modification semantics.

This distinction has security implications. Any workflow that treats a checkpoint as a complete artifact (fine-tuning, model merging, training resumption) implicitly assumes that Level 1 determines Level 2. Our results show this assumption is false. Operations that appear safe under weight-only analysis can fail catastrophically because they violate constraints that are invisible in the checkpoint.

We characterize Level 1 via a $G \times S$ decomposition: the QK circuit (Geometry, determining attention routing) and the OV/MLP weights (Slack, determining margin allocation) are separable under intervention but jointly specialized during training. Transplanting Geometry between converged models causes accuracy to collapse from 99.99% to 0.02%. Clamping Slack degrades noise tolerance linearly. Both components stabilize progressively, with Geometry consolidating before Slack. At 4 layers, this consolidation becomes stochastic: models exhibit collapse/recovery oscillations, and stability emerges probabilistically rather than deterministically.

The trust boundary for safe model modification is the training provenance, not the checkpoint.

1 The Two-Level Distinction

When practitioners save a model checkpoint, they typically save the weight tensors and, optionally, the optimizer state. The weights are treated as the artifact; the optimizer state is treated as scaffolding that assisted training but is not part of the model itself. This intuition is wrong.

The weights determine inference. Given an input, the forward pass is fully specified by the weights. In this sense, the weights *are* the model. We call this **Level 1: Behavioral State**.

But inference is only one operation. Other operations include fine-tuning, continued pretraining, model merging, and checkpoint averaging. These operations modify the weights. The question is: which modifications preserve coherent function?

The answer depends on information that is not present in the weights. We call this **Level 2: Modification Semantics**. Level 2 determines the set of transformations under which the model remains well-behaved. It is a property of the training trajectory, not the final checkpoint.

1.1 Level 1: Behavioral State

Level 1 is defined by the weight tensors θ. It fully determines the input-output mapping $f_\theta(x)$. Level 1 is observable: given a checkpoint, you can compute inference, measure activations, and characterize behavior. All standard interpretability analysis (probing, activation patching, circuit analysis) operates at Level 1.

Level 1 is *necessary* for any use of the model. You cannot run inference without weights.

1.2 Level 2: Modification Semantics

Level 2 is defined by the relationship between the current weights and the trajectory that produced them. It determines which operations on the weights will succeed and which will fail.

Consider two copies of the same checkpoint. One resumes training with the original optimizer state; the other resumes with a fresh optimizer. The weights are identical at the moment of intervention. If Level 1 fully determined modification semantics, the outcomes would be identical. They are not (Section 2).

Level 2 is *not observable* from the checkpoint alone. You cannot determine, by inspecting weights, whether a particular fine-tuning operation will succeed or fail. The checkpoint does not contain enough information.

1.3 The Security Boundary

A **trust boundary** is the interface beyond which assumptions cannot be verified. In software security, trust boundaries separate code that has been audited from code that has not.

For neural networks, the trust boundary is commonly assumed to be the checkpoint. If you have the weights, you can (in principle) verify behavior through testing and interpretability analysis. This assumption is correct for Level 1 properties: you can verify what the model does on any input.

The assumption fails for Level 2 properties. You cannot verify, by inspecting the checkpoint, whether:

- Fine-tuning will preserve capabilities or cause collapse
- Merging two models will produce coherent behavior
- Continued training will remain stable or enter oscillatory regimes
- The model sits in a sharp minimum that will be escaped under perturbation

Any workflow that treats the checkpoint as a complete artifact implicitly assumes Level 1 determines Level 2. This assumption is a security vulnerability. The true trust boundary is the training provenance: the full specification of how the model arrived at its current state.

2 The Minimal Proof: Level 2 $\neq$ Level 1

We prove that Level 2 is distinct from Level 1 with a single experiment. The logic is simple: if Level 1 fully determined Level 2, then two systems with identical Level 1 states would have identical modification semantics. We show they do not.

2.1 Experimental Setup

We trained transformers on modular addition ($a+b \mod 113$), a task known to exhibit grokking [7]. We used a standard architecture (128 dimensions, 4 attention heads per layer, 512-dimensional

MLP) with weight decay 1.0 to induce the grokking regime. Models were trained until convergence (>99% validation accuracy).

At convergence, we created two conditions:

- **CONTROL**: Resume training with both weights and optimizer state preserved
- **RESET**: Resume training with weights preserved but optimizer state reinitialized

The weights are identical at the moment of intervention. Any difference in outcomes must arise from information not present in the weights.

2.2 Results: 1-Layer Models

At one layer, the effect is modest:

Condition	Initial Acc	Final Acc	Trajectory
CONTROL	99.2%	99.8%	Stable
RESET	99.2%	99.6%	Minor dip, recovery

Table 1: **1-Layer Optimizer Ablation.** At one layer, resetting optimizer state causes a transient perturbation but the model recovers. The modification semantics are similar.

2.3 Results: 4-Layer Models

At four layers, the effect is catastrophic:

Condition	Initial	Step 1	Step 5	Pattern
CONTROL	99.5%	99.5%	99.5%	Stable
RESET	99.5%	**63.8%**	**10.8%**	Collapse
RESET (0.1× LR)	99.5%	98.6%	99.3%	Minor dip
RESET (0.01× LR)	99.5%	99.5%	99.5%	Stable

Table 2: **4-Layer Optimizer Ablation.** At four layers, resetting optimizer state with full learning rate causes immediate collapse (99.5% → 10.8%). Reducing learning rate mitigates the effect. The weights are identical across all conditions at Step 0.

The RESET condition at full learning rate shows accuracy dropping to 10%–60% within the first few optimization steps, then entering an oscillatory regime. The model does not recover.

2.4 Interpretation

The weights are identical at the moment of intervention. The forward pass is identical. Level 1 is identical.

The outcomes under continued training diverge completely: stable maintenance versus catastrophic collapse.

Therefore, Level 1 does not determine Level 2. The modification semantics of a checkpoint depend on information that is not present in the weights.

2.5 What the Optimizer State Encodes

We are not claiming that optimizer state is a third component of the model alongside weights. At inference time, optimizer state does not exist; only the weights determine behavior.

Rather, the optimizer state is *evidence* that Level 2 exists and is distinct from Level 1. The optimizer state (Adam's momentum and variance estimates) encodes information about the local curvature of the loss landscape along the recent training trajectory. This curvature information determines which step sizes and directions are safe.

When optimizer state is reset, this curvature information is lost. The optimizer takes steps that are appropriate for a generic point in weight space but inappropriate for this specific point, which sits in a region with sharp curvature in certain directions. At one layer, the landscape is smooth enough that the model recovers. At four layers, the landscape has sharp eigenvalue spikes, and the uninformed steps immediately eject the model from its stable basin.

Reducing learning rate works because it reduces the step size below the threshold at which sharp curvature causes instability. Warmup works because it allows the optimizer to rebuild curvature estimates before taking large steps.

The optimizer state is not part of the model. But it encodes information required for safe modification of the model. This is the defining feature of Level 2.

3 Level 1 Structure: The Geometry × Slack Decomposition

Having established that Level 2 exists, we now characterize Level 1. What structure do the weights encode? How does this structure determine behavior?

We adopt the framework of Elhage et al. [2], which decomposes attention heads into two circuits: the **QK circuit** (query-key, determining attention patterns) and the **OV circuit** (output-value, determining how attended tokens affect the residual stream). We extend this decomposition to the full model by defining:

- **Geometry (G)**: The QK circuit parameters (W_Q, W_K) across all layers. Geometry determines attention routing: which tokens attend to which other tokens.
- **Slack (S)**: The OV circuit parameters (W_O, W_V) and MLP weights. Slack determines how information is transformed and how strongly the model commits to its predictions (the margin at the decision boundary).

The behavioral state is jointly determined by G and S. We write this as $G \times S$ to indicate that both components are necessary and that they interact.

3.1 Geometry is Causal

We test whether Geometry causally determines behavior by transplanting it between models. If G is merely correlated with behavior, swapping it should have limited effect. If G is causal, swapping it should disrupt function.

Protocol. We trained two models on an interleaved sequence task to identical convergence (99.99% accuracy):

- Model A: Standard cross-entropy loss
- Model B: Cross-entropy with noise injection during training

Both models solve the task perfectly. Their attention patterns are similar (cosine similarity 0.79). Their residual stream representations are orthogonal (cosine similarity -0.002).

We created a hybrid model by transplanting the QK parameters from Model B into Model A, leaving A's OV and MLP weights unchanged.

Condition	Params Swapped	Accuracy	Δ
Model A (baseline)	0	99.99%	–
Swap 1 head	192	47.1%	−52.9%
Swap 2 heads	384	11.7%	−88.3%
Swap all heads	768	**0.02%**	−99.97%

Table 3: **Routing Swap Causes Collapse.** Transplanting QK parameters between converged models causes accuracy to fall to chance (0.02%). The Slack learned under Model A's Geometry is incompatible with Model B's Geometry. (n=5 seeds)

The hybrid model performs at chance. It is worse than both parents. The QK parameters from B are exactly preserved (parameter drift 0.0), but the representations produced are orthogonal to both A and B.

This result establishes that Geometry is causal: changing the routing changes the behavior, even when the downstream processing (Slack) is unchanged. It also establishes that G and S are jointly specialized: Slack learned under one Geometry cannot function under a different Geometry.

We replicated this experiment at 2 layers with the same result: swapping all QK parameters across both layers caused accuracy to fall to 0.0%.

3.2 Slack is Pre-Allocated

Models maintain large margins between correct and incorrect logits. Is this margin a computational necessity, or excess capacity that provides robustness?

No gain reflex. We subjected a trained model to noise injection at inference time (no retraining) and measured the response.

Condition	Noise (σ)	Accuracy	Margin	Reflex?
Baseline	0.0	99.9%	8.55	–
Mild	0.3	99.9%	8.47	No
Moderate	2.0	99.2%	5.74	No
Severe	3.0	82.9%	3.02	No

Table 4: **Absence of Dynamic Compensation.** Under noise injection, margin decreases monotonically. The model does not increase margin to compensate for the perturbation. Robustness draws down a fixed buffer. (n=5 seeds)

If the model possessed a compensatory mechanism to increase margin under stress, we would expect margin to increase or stabilize. Instead, margin decreases monotonically with noise magnitude. The model is a passive system drawing down a pre-allocated buffer.

Margin is structurally necessary. We clamped the residual stream magnitude (post-LayerNorm, pre-output head) to varying fractions of its natural value, preserving direction while attenuating magnitude.

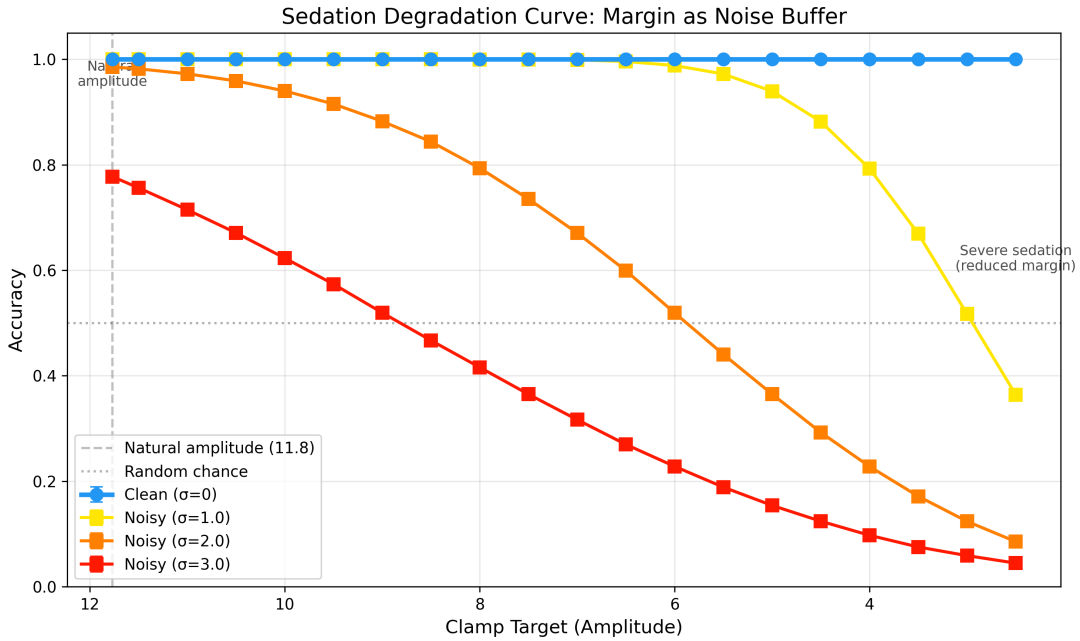

Figure 1: **Margin Serves as Robustness Buffer.** We clamped residual stream magnitude to varying fractions of natural amplitude. Under clean conditions ($\sigma = 0$), accuracy is insensitive to clamping; the task can be solved with 20% of natural margin. Under noise, accuracy tracks clamping linearly. The excess margin is a pre-allocated noise buffer, not computational waste. (n=5 seeds, shaded regions: 95% CI)

Clean accuracy remains high even at 20% of natural amplitude. The margin far exceeds what is required for the task. Under noise, accuracy degrades linearly with clamping. The excess margin was serving as a buffer against perturbation.

Precision drives margin. We trained models on tasks with varying precision requirements (modular arithmetic with different moduli) at fixed width.

Task	**Modulus**	**Margin**	A_{norm}
Interleaved	–	+3.1	1.01
Modular Add	7	+5.8	1.01
Modular Add	113	+6.4	1.04
Modular Add	227	+6.5	1.04

Table 5: **Precision Drives Margin.** Higher-precision tasks induce higher margins during training. The margin is allocated prophylactically based on task demands, not generated dynamically at inference.

Higher-precision tasks drive higher margins. The model allocates margin during training based on task demands. This allocation is prophylactic: it happens before any test-time perturbation.

3.3 G and S are Separable but Interdependent

The routing swap experiment demonstrates that we can intervene on G while holding S constant. The clamping experiment demonstrates that we can intervene on S (attenuating margin) while holding G constant. In this sense, G and S are separable: we can manipulate them independently.

But the routing swap also demonstrates that G and S are interdependent: Slack learned under one Geometry does not function under a different Geometry. The two components are jointly specialized during training. You cannot mix and match G and S from different models.

This separability-under-intervention combined with interdependence-under-training is what we mean by the $G \times S$ decomposition. The components are structurally distinct (different parameter groups, different functions) but functionally coupled (co-trained, non-transferable).

4 How Level 1 Structure Emerges

The $G \times S$ decomposition describes the structure of converged models. But how does this structure arise during training? The answer has implications for Level 2: understanding *when* structure consolidates helps explain *why* modification semantics depend on training trajectory.

4.1 Progressive Specialization

We tracked parameter velocity ($v_t = \|\theta_t - \theta_{t-1}\|_2$) for Geometry (QK parameters) and Slack (OV/MLP parameters) throughout training on modular addition.

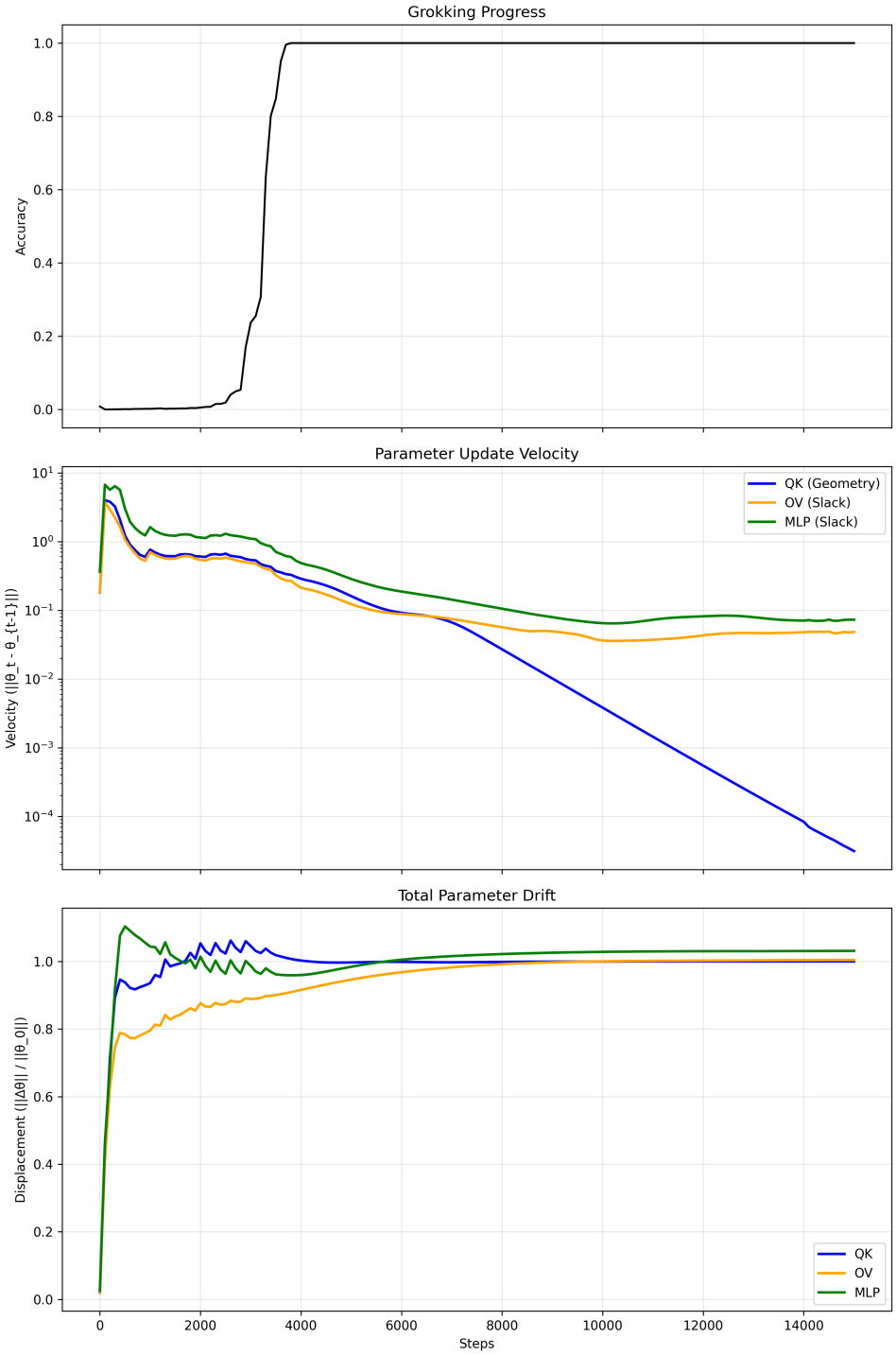

Figure 2: **Geometry Stabilizes Before Slack.** Parameter velocity over training. Blue: QK circuit (Geometry). Red: OV/MLP (Slack). Geometry velocity drops to near-zero by step 4000, while Slack continues to evolve until step 15000. The model establishes routing structure first, then optimizes margin allocation within that structure.

The training process exhibits a clear phase structure:

1. **Plastic Phase (Steps 0–1400):** Both G and S evolve rapidly. The model is fitting training data. No generalization yet.

2. **Geometry Consolidation (Steps 1400–4000):** QK velocity drops sharply while OV/MLP

velocity remains high. The routing structure is stabilizing. This phase corresponds to circuit formation in Nanda et al.'s analysis of grokking [7].

3. **Slack Optimization (Steps 4000–15000):** QK velocity is near-zero. OV/MLP continues to adjust. The model optimizes margin allocation within the now-fixed routing structure.

4. **Full Consolidation (Steps 15000+):** Both G and S have stabilized. Training is complete.

This temporal ordering (G before S) explains why the routing swap fails so catastrophically: by the time Slack is learned, it has been specialized for thousands of steps under a specific Geometry. That specialization cannot be undone by transplantation.

4.2 The Early-Phase Subspace

If Geometry consolidates early, what is the relationship between early Geometry and final Slack? We tested this by freezing QK parameters early in training (step 1000, before generalization) and then training Slack under two different conditions.

Protocol.

- Train model to step 1000 (100% train accuracy, 0.3% validation accuracy)
- Freeze QK parameters ("Early-Phase Geometry")
- Train two separate models from this frozen Geometry:
  - **Anchor**: Standard cross-entropy loss
  - **Probe**: Cross-entropy plus orthogonality penalty $\lambda \cdot |\cos(S_{anchor}, S_{probe})|$

Depth	Model	Accuracy	Grokking Step	CosSim
1-Layer	Anchor	100%	∼2500	1.00
1-Layer	Probe	100%	∼2500	**0.00**
2-Layer	Anchor	100%	∼17000	1.00
2-Layer	Probe	100%	∼15000	**0.00**

Table 6: **Early-Phase Geometry Permits Orthogonal Slack.** Under frozen Early-Phase Geometry, two models achieve 100% accuracy with perfectly orthogonal residual representations. Early G defines a solution subspace, not a single point. (n=5 seeds)

Both models achieve 100% validation accuracy. Both successfully grok. Yet their final Slack allocations are perfectly orthogonal (cosine similarity 0.00).

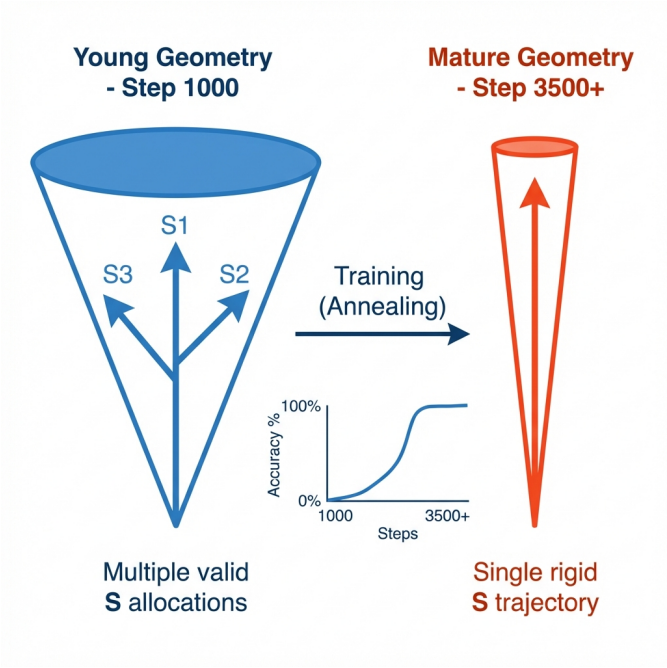

Figure 3: **Progressive Specialization.** Early Geometry defines a broad subspace of valid Slack configurations. Training specializes the model into one specific path through this subspace. Mature Geometry admits only one Slack allocation.

This result establishes that early Geometry defines a **subspace** of valid solutions. Multiple orthogonal Slack allocations can achieve perfect accuracy under the same routing. Training selects one path through this subspace.

4.3 Consolidation and Plasticity Loss

Achille et al. [1] demonstrated that deep networks exhibit "critical periods" during which deficits cause permanent impairment. They attributed this to loss of "Information Plasticity" as network structure consolidates.

Our results provide a mechanistic account of this consolidation. During the critical period (before step 4000 in our setup), Geometry is plastic and defines a broad subspace. After consolidation, Geometry is fixed and the subspace collapses to a single trajectory.

This consolidation is why Level 2 constraints exist. The modification semantics of a checkpoint depend on *which path through the early subspace the model took*. Two models with identical final accuracy may have arrived via different trajectories, and these trajectories determine which modifications are safe.

Concretely: if a model's Geometry consolidated while the optimizer was tracking a particular curvature structure, that curvature structure is part of the model's modification semantics. Discarding it (as in optimizer reset) can cause the model to take steps that are incompatible with the consolidated structure.

The checkpoint contains the endpoint of the trajectory. It does not contain the trajectory itself. This is the fundamental reason why Level 1 does not determine Level 2.

5 Depth Creates Metastability

At 1–2 layers, the $G \times S$ framework describes a clean, deterministic process: Geometry stabilizes, Slack optimizes within it, and the model converges. At 4 layers, this deterministic picture breaks

down. Training dynamics become stochastic, and stability emerges probabilistically rather than reliably.

This metastability compounds the Level 2 problem. At shallow depth, Level 2 constraints are about trajectory information (optimizer state, curvature estimates). At greater depth, Level 2 constraints include the stochastic outcome of basin escape, which may not be reproducible even with identical trajectory information.

5.1 The Metastable Regime

We trained 4-layer models on modular addition using the same protocol as Section 4, with Early-Phase Geometry frozen at the auto-detected critical period (90% validation accuracy threshold with hysteresis, typically around step 1500).

Metric	Value	Interpretation
Stability Rate	40–50%	Stochastic outcome
Mean Collapses	6.4 ± 3.1	High variance
Time-to-Stability	∼19,500 steps	When achieved
Final Accuracy Std	44%	Bimodal distribution

Table 7: **4-Layer Stability Characterization.** At 4 layers, only 40–50% of seeds achieve stable convergence. The 44% standard deviation on final accuracy reflects a bimodal outcome: models either lock into a stable basin (∼100% accuracy) or remain in an oscillatory regime (∼0% accuracy). (5 seeds)

Training trajectories exhibit repeated collapse/recovery cycles. A model may achieve 100% accuracy, collapse to 0.9%, recover to 100%, collapse again, and either eventually stabilize or continue oscillating indefinitely. The outcome cannot be predicted from early training behavior.

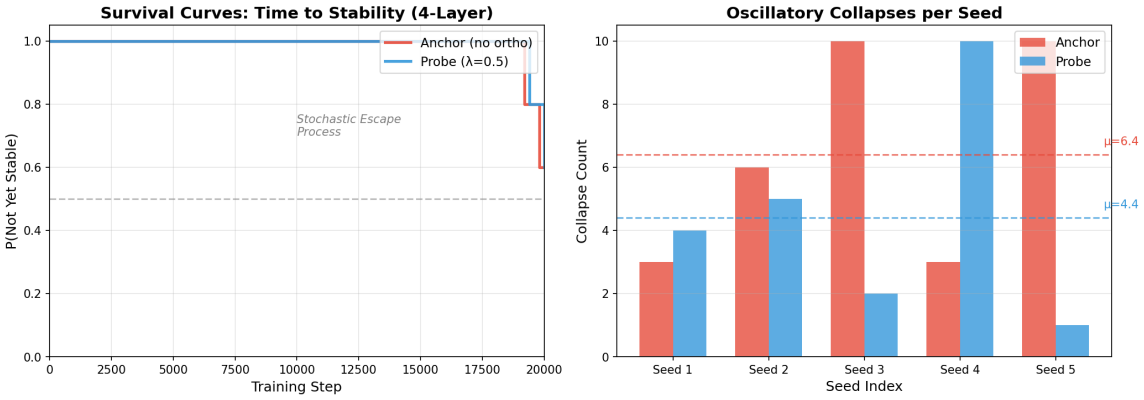

Figure 4: **Stochastic Escape at 4 Layers.** Left: Time-to-stability curves. Models are "not yet stable" until they suddenly lock in (step-function escape). Right: Collapse counts vary widely across seeds (2–10), and collapse count does not predict final stability.

5.2 Loss Engineering Does Not Help

A natural hypothesis is that regularization might help models escape the metastable regime. We tested orthogonality penalties ($\lambda \in \{0, 0.05, 0.3\}$) across 8 seeds per condition.

λ	Stability Rate	Collapses	Mean Accuracy
0.0 (baseline)	**50%**	6.1 ± 2.7	0.594 ± 0.42
0.05	25%	6.0 ± 4.4	0.562 ± 0.37
0.3	25%	6.5 ± 5.0	0.490 ± 0.35

Table 8: **Orthogonality Penalties Reduce Stability.** The baseline condition ($\lambda = 0$) achieves the highest stability rate (50%). Adding orthogonality penalties reduces stability (25%) and degrades mean accuracy. Collapse counts remain unchanged. (8 seeds per condition)

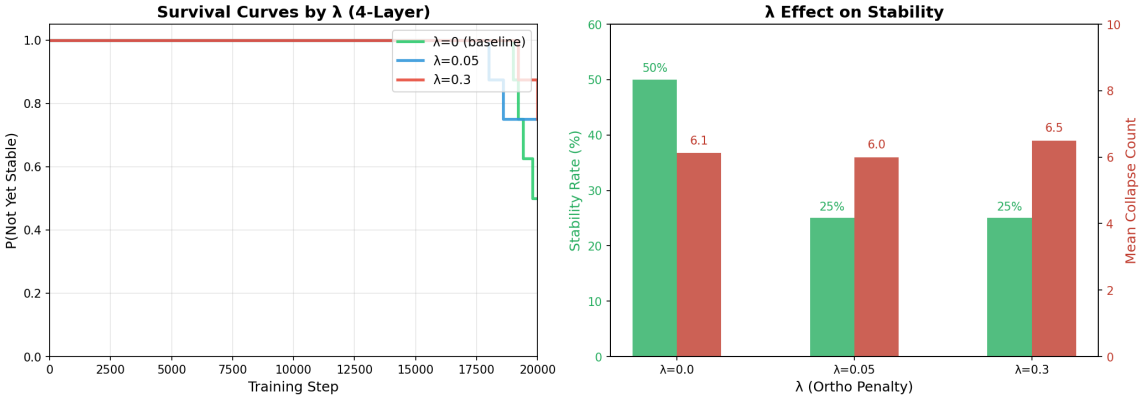

Figure 5: **Lambda Sweep Results.** Left: Survival curves by λ. Baseline ($\lambda = 0$) achieves highest final stability. Right: Summary statistics showing stability rate and accuracy degrading with increasing λ.

The baseline condition (no regularization) achieves the highest stability rate. Adding orthogonality penalties *reduces* stability. Collapse counts remain unchanged across conditions ($\sim$6), suggesting that the penalty affects trajectory dynamics without changing the underlying landscape topology.

We ran a 2×2 factorial (6 seeds × 4 conditions) to disentangle the effects of reference coupling (training alongside an anchor model) and orthogonality penalty:

Condition	Reference	Ortho	Stability	Collapses
CE Only	×	×	50%	6.2 ± 2.9
CE + EMA-Self	×	×	50%	$\mathbf{1.8 \pm 2.3}$
Anchor (no penalty)	✓	×	**67%**	8.0 ± 2.7
Anchor + Ortho	✓	✓	50%	7.5 ± 1.5

Table 9: **2×2 Factorial.** Reference coupling provides modest stability benefit (+8%). Orthogonality penalty reduces stability (−8%). EMA smoothing reduces collapse frequency but does not improve escape probability. (6 seeds per condition)

Main effects:

- Reference coupling: +8% stability (50% → 58% average)
- Orthogonality penalty: −8% stability (58% → 50% average)
- EMA smoothing: Reduces collapse frequency (6.2 → 1.8) but does not improve escape probability

Neither intervention reliably improves stability. The modest benefit of reference coupling may come from weak directional bias toward stable basins. The penalty consistently destabilizes, likely through gradient interference. EMA smooths trajectories without changing the landscape topology.

5.3 Implications for Scale

These results suggest that metastability is a property of the loss landscape at depth, not a failure of optimization dynamics. Loss term engineering modulates trajectories but cannot alter the underlying basin structure.

If metastability increases with depth (as our 1→2→4 layer progression suggests), training outcomes at frontier scales may be fundamentally stochastic. Two runs with identical hyperparameters and different seeds may converge to qualitatively different solutions. The variance in training outcomes documented at scale [4] may reflect this intrinsic stochasticity rather than optimization noise.

For Level 2, metastability means that modification semantics may depend on stochastic factors that are not encoded anywhere. Even with full training provenance (weights, optimizer state, random seed, hyperparameters), the outcome of continued training may be unpredictable because the model sits on a ridge where small perturbations determine which basin it falls into.

6 Security Implications

The distinction between Level 1 and Level 2 has immediate implications for the security of machine learning workflows. Any operation that assumes checkpoints are complete artifacts is making an implicit assumption that Level 1 determines Level 2. Our results show this assumption is false.

6.1 The Trust Boundary

In security analysis, a trust boundary separates verified from unverified components. For neural network deployments, the trust boundary is commonly assumed to be the model checkpoint: if you have the weights, you can verify behavior through testing and interpretability.

This assumption is correct for inference (Level 1). Given a checkpoint, you can:

- Compute the output for any input
- Measure internal activations
- Verify behavior on held-out test sets
- Apply interpretability techniques (probing, activation patching, circuit analysis)

The assumption fails for modification (Level 2). Given only a checkpoint, you cannot:

- Predict whether fine-tuning will preserve capabilities
- Determine whether the model sits in a stable or metastable basin
- Know which learning rates and optimizers are safe
- Verify that model merging will produce coherent behavior

The true trust boundary for modification operations is the training provenance: the checkpoint plus the optimizer state plus the training trajectory information required to resume safely. Vendors who ship weights without this information are shipping an artifact that cannot be reliably modified.

6.2 Unsafe Operations

Several common workflows implicitly assume Level 1 determines Level 2:

Fine-tuning without optimizer state. Our minimal proof (Section 2) demonstrates that resuming training without optimizer state can cause catastrophic collapse at depth. Fine-tuning workflows that start from a downloaded checkpoint with a fresh optimizer are vulnerable to this failure mode. The standard mitigation (warmup, reduced learning rate) works by reducing step size below the threshold at which sharp landscape curvature causes instability, but practitioners typically apply these mitigations without understanding why they are necessary.

Model merging. Techniques that combine weights from multiple models (averaging, SLERP, task arithmetic) assume that the resulting weights will produce coherent behavior. Our routing swap experiment (Section 3) demonstrates that this assumption is false: combining Geometry from one model with Slack from another produces a hybrid that performs at chance. Model merging can succeed when the models being merged share a common ancestor and have not diverged too far, but there is no principled way to determine the safe merging radius from checkpoints alone.

Checkpoint averaging. Averaging checkpoints from different points in training assumes that the average of valid solutions is itself valid. This assumption can fail when the checkpoints sit in different basins or when the averaging crosses a basin boundary. The metastability results (Section 5) suggest that at depth, checkpoints may sit on ridges where small perturbations (including averaging) determine basin membership.

Training resumption after failure. When training is interrupted (hardware failure, preemption, timeout), the standard practice is to resume from the last checkpoint. If the optimizer state was not saved or was corrupted, the resumed run may diverge from the original trajectory. Our results suggest this divergence is mild at shallow depth but potentially catastrophic at greater depth.

6.3 The Supply Chain Problem

Model providers (Hugging Face, model APIs, research releases) typically distribute checkpoints without optimizer state. The optimizer state is large (2× model size for Adam) and is not required for inference. This practice is reasonable for inference-only use cases.

For modification use cases (fine-tuning, continued pre-training, adaptation), the absence of optimizer state creates a supply chain vulnerability. The recipient receives an artifact that appears complete (the weights) but lacks information required for safe modification (the trajectory context).

This is analogous to distributing compiled binaries without debug symbols. The binary runs correctly, but debugging and modification become unreliable. The difference is that for neural networks, the "debug information" (optimizer state, training trajectory) is not merely convenient but *necessary* for certain operations.

Providers who intend their models to be fine-tuned should consider distributing optimizer state or, at minimum, documenting the learning rate schedules and warmup protocols that are safe for their checkpoints.

6.4 Detection and Verification

Can you determine, before performing an operation, whether it will violate Level 2 constraints?

Partial answer: warmup as a boundary probe. Warmup schedules that start with very low learning rates and gradually increase can serve as a probe for Level 2 constraints. If the model tolerates gradual increase to a target learning rate, that learning rate is likely safe. If the model becomes unstable during warmup, the target rate exceeds the Level 2 tolerance. This is a heuristic, not a guarantee.

Open problem: pre-modification audits. Ideally, practitioners could audit a checkpoint before modification to determine which operations are safe. This would require characterizing the local loss landscape (curvature, basin depth, distance to basin boundaries) from the checkpoint alone. Current techniques for loss landscape analysis are computationally expensive and do not scale to frontier models. Developing efficient pre-modification audits is an open research problem.

Practical recommendation: defensive defaults. In the absence of reliable pre-modification audits, practitioners should assume that any modification operation may fail and use defensive defaults:

- Always use warmup when fine-tuning or resuming training
- Start with learning rates at least 10× smaller than training
- Monitor validation metrics continuously during early fine-tuning
- Preserve optimizer state when checkpointing during long runs
- Test fine-tuned models on held-out capability benchmarks before deployment

These practices are already common in large-scale training but are often treated as optimization folklore rather than security requirements. Our results suggest they should be treated as mandatory safeguards for Level 2 compliance.

7 Related Work

Transformer Circuits. Elhage et al. [2] introduced the mathematical framework we build on, decomposing attention heads into QK (routing) and OV (processing) circuits. Subsequent work identified specific circuits for induction [8], indirect object identification [10], and other behaviors. We adopt their decomposition and extend it to training dynamics: asking when circuits form, how they consolidate, and whether they can be modified after training.

Grokking. Power et al. (2022) discovered delayed generalization ("grokking") in transformers trained on algorithmic tasks. Nanda et al. [7] reverse-engineered the learned algorithm (Fourier multiplication for modular addition) and identified three training phases: memorization, circuit formation, and cleanup. Our work extends this analysis by identifying which parameters change during each phase (QK during circuit formation, OV/MLP during cleanup) and showing that the phase structure has implications for modification semantics.

Information Plasticity. Achille et al. [1] demonstrated that deep networks exhibit critical periods during which deficits cause permanent impairment. They attributed this to "Information Plasticity" loss as network structure consolidates. Our $G \times S$ framework provides mechanistic grounding: Geometry consolidates first, defining a subspace; Slack then optimizes within that subspace. The "non-transferability" we observe in converged models is a consequence of plasticity loss after the critical period.

Training Dynamics. Thilak et al. [9] identified the "Slingshot Mechanism" (cyclic oscillations between stable and unstable training regimes) in grokking models. Our 4-layer experiments show that at depth, these oscillations become stochastic: stability emerges probabilistically rather than deterministically. Gross et al. [3] modeled late-stage optimization as a stochastic process near a stationary distribution, which helps explain why optimizer reset causes the effects we observe.

Self-Repair. McGrath et al. [6] documented "self-repair" mechanisms by which models compensate for ablated components through backup heads. We do not contest these findings. Our claim is narrower: there is no dynamic margin amplification at the decision boundary. Compensation operates through information rerouting, not gain modulation. The pre-allocated margin (Slack) is a fixed buffer, not an adaptive response.

Model Merging. Recent work on model merging [5, 11] develops techniques for combining weights from multiple models. Our routing swap results provide a causal account of why naive merging fails: Slack learned under one Geometry is incompatible with different Geometry. Successful merging requires that the models being merged share compatible $G \times S$ structure, which is more likely when they share a common training ancestor.

8 Limitations

Scale. Our experiments use 1–4 layer transformers on controlled tasks (modular addition, interleaved sequences). Whether the Level 1/Level 2 distinction holds at frontier scale (billions of parameters, hundreds of layers) remains an open question. The metastability we observe at 4 layers may intensify, diminish, or qualitatively change at greater depth.

Task domain. We focus on algorithmic tasks that exhibit grokking. These tasks have sharp phase transitions that make the Level 2 effects visible. Language modeling on TinyStories (8 layers) showed smooth convergence without metastability. The Level 2 constraints we identify may be weaker or differently structured for statistical tasks that lack grokking dynamics.

Sample sizes. The metastability characterization (Section 5) uses 5–8 seeds per condition. The qualitative phenomena (collapse/recovery oscillations, stochastic escape) are consistent across all seeds, but the exact stability rates (40% vs 50% vs 67%) are estimates with wide confidence intervals. Larger-scale studies would sharpen these estimates.

Optimizer specificity. Our experiments use Adam with standard hyperparameters. The Level 2 effects may differ for other optimizers (SGD, AdamW with different weight decay, LAMB, etc.). The specific curvature information encoded in Adam's variance estimates may not generalize to optimizers with different state structures.

Causality of optimizer state. We demonstrate that optimizer state is *informative* about modification semantics (identical weights, different outcomes). We do not demonstrate that optimizer state is *sufficient*. Other trajectory information (learning rate schedule history, batch ordering, etc.) may also contribute to Level 2 constraints.

9 Conclusion

Trained neural networks admit two levels of description. Level 1 (Behavioral State) is defined by the weights and determines inference. Level 2 (Modification Semantics) determines which operations on the model preserve coherent function and depends on training trajectory information that cannot be recovered from weights alone.

We proved Level 2 is distinct from Level 1 with a minimal experiment: two 4-layer transformers with identical weights but different outcomes under continued training. Preserving optimizer state maintains stability; resetting it causes collapse. Same weights, different modification semantics.

We characterized Level 1 structure via a $G \times S$ decomposition. Geometry (the QK circuit) determines attention routing; Slack (OV/MLP) determines margin allocation. Transplanting Geometry between models causes collapse to chance. Clamping Slack degrades noise tolerance. Geometry consolidates before Slack during training, defining a subspace that narrows to a single trajectory as training progresses.

At depth, the Level 2 problem compounds. Four-layer models exhibit metastable dynamics: collapse/recovery oscillations with stochastic escape to stability. Loss term engineering does not reliably improve stability rates. The phenomenon reflects landscape topology rather than optimization dynamics.

The practical implication is that checkpoints are not complete artifacts for modification operations. The trust boundary for safe modification is the training provenance, not the weights alone. Fine-tuning, model merging, and training resumption are all operations that require

Level 2 information. Practitioners who perform these operations without that information are accepting risk they may not be aware of.

The defensive recommendations follow directly: use warmup, reduce learning rates, preserve optimizer state, and monitor for capability degradation during fine-tuning. These practices are already common in large-scale training, but they are typically treated as optimization folklore. Our results suggest they should be treated as security requirements.

The broader implication is that the machine learning community's mental model of trained models is incomplete. We think of models as weight tensors that determine behavior. This is correct for inference. For modification, models are positions in a landscape with local structure that is not encoded in the weights. Operations that appear safe under weight-only analysis can fail because they violate constraints that are invisible in the checkpoint. Recognizing this distinction is the first step toward workflows that respect it.

References

[1] Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep networks. In *International Conference on Learning Representations*, 2019.

[2] Nelson Elhage, Neel Nanda, Catherine Olsson, et al. A mathematical framework for transformer circuits. In *Transformer Circuits Thread*, 2021.

[3] David J Gross, Neel Nanda, et al. Weight fluctuations in deep linear neural networks and a derivation of the inverse-variance flatness relation. *arXiv preprint arXiv:2403.00322*, 2024.

[4] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[5] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, et al. Editing models with task arithmetic. In *ICLR*, 2023.

[6] Thomas McGrath et al. The hydra effect: Emergent self-repair in language model computations. *arXiv preprint*, 2023.

[7] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations*, 2023.

[8] Catherine Olsson, Nelson Elhage, Neel Nanda, et al. In-context learning and induction heads. In *Transformer Circuits Thread*, 2022.

[9] Vimal Thilak, Etai Littwin, Shuangfei Zhai, Omid Saremi, Roni Paiss, and Joshua Susskind. The slingshot mechanism: An empirical study of adaptive optimizers and the grokking phenomenon. *arXiv preprint arXiv:2206.04817*, 2022.

[10] Kevin Wang et al. Interpretability in the wild: A circuit for indirect object identification in gpt-2 small. In *ICLR*, 2023.

[11] Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. In *NeurIPS*, 2023.