# stability of induced algorithmic structure and Zero-Shot Structural Scaling in Neural Networks

**Author:** grisun0

## Abstract

This paper studies stability of induced algorithmic structure in neural networks: the property that a trained model can be expanded to larger input dimensions without retraining while preserving correct computation. I investigate this phenomenon using bilinear models trained on 2x2 matrix multiplication with Strassen-structured inductive bias.

My approach is explicit about methodology. I do not claim that networks discover algorithms from scratch. Instead, I induce a known structure (rank-7 tensor decomposition) through architectural constraints and post-hoc discretization, then study whether this induced structure transfers to larger problem sizes without retraining. Structural transfer succeeds when training conditions are met (68% of runs); it fails otherwise, requiring fallback to canonical coefficients (32% of runs). When successful, the induced structure generalizes to 4x4, 8x8, 16x16, 32x32, and 64x64 matrices.

The contribution is not algorithm discovery but characterization of training dynamics. I identify conditions under which induced algorithmic structure remains stable during transfer: batch sizes in range [24, 128], sufficient training epochs, and weight decay regularization. Under these conditions, the induced Strassen implementation achieves 1.95x speedup over single-threaded OpenBLAS at N=8192. Under standard multi-threaded conditions, OpenBLAS is faster.

Statistical validation across 195 training runs confirms that batch size significantly affects convergence quality (F=15.34, p<0.0001). I report both successful and failed conditions to enable reproducibility.

## 1. Introduction

Neural networks trained on algorithmic tasks sometimes exhibit grokking: delayed generalization that occurs long after training loss has converged [1]. Prior work by Humayun et al. [1] characterized this transition using local complexity measures, and Bereska et al. [2] connected it to superposition as lossy compression.

I study a specific question: if a network is induced to represent a known algorithm, can that representation transfer to larger problem instances without retraining? This is not a claim about algorithm discovery. It is a question about structural stability under scaling.

My experimental setup is explicit about inductive bias:

1. The model architecture uses rank-8 tensor decomposition with a target of 7 active slots (matching Strassen).
2. After training, weights are discretized to {-1, 0, 1} via rounding.
3. If verification fails, the system falls back to canonical Strassen coefficients.

Given this methodology, I ask: under what training conditions (batch size, learning rate, training duration) does the induced structure remain stable during expansion without retraining?

My contributions:

1. Empirical characterization: I identify training conditions (batch size, regularization) that support stable structural transfer. Success rate: 68% without fallback.
2. Convergence conditions: I propose empirically validated conditions under which training converges to states amenable to discretization and expansion.
3. Uniqueness of expansion operator: I verify experimentally that slot ordering is essential. Permuting slots breaks correctness (mean error 74%), confirming the expansion operator T is unique for a given coefficient ordering.
4. Statistical validation: I present experimental validation with N=195 observations confirming significant effects of batch size on convergence.

---

## 2. Problem Setting

I consider 2x2 matrix multiplication:

```
C = A @ B
```

A bilinear model learns tensors U, V, W such that:

```
M_k = (U[k] . a) * (V[k] . b)
c = W @ M
```

where a, b, c are flattened 4-vectors.

The central question is:

Given a model with induced Strassen structure at 2x2, under what conditions can it be expanded to compute NxN matrix multiplication correctly without retraining?

---

# 3. Methodology

## 3.1 Inductive Bias

I am explicit about the inductive bias in my approach:

1. Architecture: The model uses 8 slots, with a target of 7 active slots (matching Strassen's rank).
2. Sparsification: After training, I prune to exactly 7 slots based on importance scores.
3. Discretization: Weights are rounded to {-1, 0, 1} using torch.round().clamp(-1, 1). This is post-hoc intervention, not emergent behavior.
4. Fallback: If verification fails, canonical Strassen coefficients are used (32% of runs).

This is not algorithm discovery. It is structured optimization with strong priors.

Table: Engineered vs Emergent Features

| Feature | Engineered | Emergent |
|---|---|---|
| Rank-7 constraint | Yes (architectural prior) | No (engineered) |
| Values | Yes (post-hoc rounding) | No (engineered) |
| Convergence to discrete | Partial (training dynamics) | Partial |
| Benchmark performance | No | Yes |
| Zero-shot transfer | No | Yes (when conditions met) |

Success rate without fallback: 68% (133/195 runs). CV of discretization error: 1.2%.

## 3.2 Training Conditions

I investigate how training parameters affect convergence:

Batch size: Values in [24, 128] correlate with successful discretization.

Correction: I initially hypothesized this was due to L3 cache coherence. After computing memory requirements (model: 384 bytes, optimizer state: 768 bytes, per-sample: 320 bytes), I found that even B=1024 fits comfortably in L3 cache on all tested hardware. The batch size effect is therefore due to training dynamics (gradient noise, learning rate coupling), not hardware constraints. I do not yet have a theoretical explanation for why [24, 128] works best.

Training duration: Extended training (1000+ epochs) is required for weights to approach values amenable to discretization.

Optimizer: AdamW with weight decay >= 1e-4 produces better results than pure Adam.

### 3.3 Verification Protocol

After discretization, I verify:

1. Correctness: C_model matches C_true within floating-point tolerance (relative error < 1e-5)
2. Expansion: The same coefficients work for 4x4, 8x8, 16x16, 32x32, 64x64

Discretization success is defined as: all 21 weight values (7 slots x 3 tensors) round to the correct Strassen coefficient. Partial success is not counted.

### 3.4 Discretization Fragility

I tested noise stability by adding Gaussian noise (sigma in {0.001, 0.01, 0.1}) to weights before discretization. Success rate dropped to 0% for all noise levels tested (100 trials each). This confirms that discretization is fragile: weights must be very close to integer values for rounding to produce correct coefficients. The training process achieves this only under specific conditions.

---

# 4. Convergence Conditions

## 4.1 Empirically Validated Proposition

Proposition 4.1 (Conditions for Successful Discretization)

Note: These are empirical observations, not derived theorems.

I observe that discretization succeeds (weights round to correct Strassen coefficients) when:

(A1) Batch size B is in [24, 128].

(A2) Training continues for at least 500 epochs with grokking dynamics observed. I define grokking as: training loss < 1e-6 while test loss remains > 0.1 for at least 100 epochs, followed by sudden test loss drop (see Appendix D, Figure 5).

(A3) Weight decay is applied (>= 1e-4 for AdamW).

(A4) The model uses symmetric initialization for U and V tensors.

When these conditions are met, weights typically approach values within 0.1 of {-1, 0, 1}, making discretization reliable. The metric is L-infinity: max(|w - round(w)|) < 0.1 for all weights.

When conditions are not met, the fallback to canonical coefficients is triggered automatically by the verification step.

---

# 5. Algebraic Formalization: Theory and Verification

This section presents the general theory of Algorithmic Invariance developed in my prior work, then describes how the Strassen experiments verify specific aspects of this theory.

## 5.1 General Theory of Algorithmic Invariance

I define Algorithmic Invariance as the property that a learned operator W satisfies:

```
T(W_n) ≈ W_{n'}
```

where T is a deterministic expansion operator and W_{n'} correctly implements the task at scale n' > n without retraining.

This invariance demonstrates that the network has learned an internal representation of the algorithm itself, rather than memorizing input-output correlations from the training set.

### 5.1.1 The Expansion Operator T

Let W_n be the converged weight operator of a model trained at problem size n. I define T as the minimal linear embedding that preserves the dominant singular subspace of W_n under strong normalization.

Operationally, T is constructed to satisfy the following properties:

**Property 1 (Spectral Preservation):** T preserves the order and magnitude of the k principal singular values of W_n up to numerical tolerance ε.

**Property 2 (Subspace Invariance):** The dominant singular subspace of W_n maps isometrically to the corresponding subspace of W_{n'}.

**Property 3 (Normalization Consistency):** Weight norms and relative scale factors remain bounded under expansion.

Under these conditions, the expanded operator W_{n'} satisfies the approximate commutation property:

```
T ∘ f_n ≈ f_{n'} ∘ T
```

where f_n and f_{n'} denote the functions implemented by the models before and after expansion, respectively. Zero-shot structural scaling fails when this approximate equivariance is violated.

### 5.1.2 Training Dynamics

The training dynamics that give rise to algorithmic invariance follow:

```
W_{t+1} = W_t - η ∇L(W_t) + ξ_t
```

where ξ_t represents gradient noise induced by minibatching, numerical precision, and hardware execution. Successful algorithmic invariance requires that Var(ξ_t) falls below a task-dependent threshold relative to the smallest non-zero singular value of the learned operator.

### 5.1.3 Uniqueness

Among all linear expansions that preserve normalization and spectral ordering, T is empirically unique up to permutation symmetry of equivalent neurons.

## 5.2 Verification via Strassen Matrix Multiplication

The Strassen experiments provide empirical verification of this theory for a specific algorithmic domain.

### 5.2.1 Strassen-Specific Instantiation

For Strassen-structured matrix multiplication, the learned operator consists of three tensors:

```
U ∈ R^{7×4}    (input A coefficients)
V ∈ R^{7×4}    (input B coefficients)
W ∈ R^{4×7}    (output C coefficients)
```

The bilinear computation is:

```
C = W @ ((U @ a) * (V @ b))
```

where a, b are flattened input matrices and * denotes elementwise product.

The expansion operator T maps 2×2 coefficients to N×N computation via recursive block application:

```
T: (U, V, W, A, B) → C_N
```

Operationally:

```
T(U, V, W, A, B) =
    if N = 2: W @ ((U @ vec(A)) * (V @ vec(B)))
    else: combine(T(U, V, W, A_ij, B_ij) for quadrants i,j)
```

### 5.2.2 Verified Properties

The Strassen experiments verified the following theoretical predictions:

**Verified 1 (Correctness Preservation):** The expanded operator T(U, V, W, A, B) computes correct matrix multiplication for all tested sizes ($2\times2$ to $64\times64$). Relative error remains below $2\times10^{-6}$.

**Verified 2 (Uniqueness up to Permutation):** Testing all 7! = 5040 slot permutations confirms that T is unique for a given coefficient ordering. Permuting slots produces mean error of 74%.

**Verified 3 (Commutation Property):** $T \circ f_2 \approx f_N \circ T$ holds with relative error $< 2\times10^{-6}$ for $N \in \{4, 8, 16, 32, 64\}$.

**Verified 4 (Normalization Dependency):** Success rate (68%) correlates with training conditions that maintain weight norms near discrete values.

### 5.2.3 Conditions for Valid Expansion

Expansion via T succeeds when:

(C1) **Discretization:** All 21 coefficients round to exact values in {-1, 0, 1}.

(C2) **Verification:** The discretized coefficients pass correctness check at $2\times2$.

(C3) **Structural Match:** Learned coefficients match Strassen's canonical structure up to slot permutation and sign equivalence.

Fallback to canonical coefficients occurs in 32% of runs when conditions are not met.

## 5.3 Hypotheses Not Demonstrated by Strassen Experiments

The following theoretical predictions from my original framework were NOT verified or were actively contradicted by the Strassen experiments:

**Not Demonstrated 1 (Hardware-Coupled Noise):** I originally hypothesized that the optimal batch size B* corresponds to cache coherence effects (L3 cache saturation, AVX-512 utilization). Memory analysis showed that even B=1024 fits in L3 cache. The batch size effect is due to training dynamics, not hardware constraints. I do not yet have a theoretical explanation for the optimal range [24, 128].

**Not Demonstrated 2 (Curvature Criterion):** The grokking prediction criterion $\kappa_{eff} = -\mathrm{tr}(H)/N$ was proposed but not systematically tested in the Strassen experiments. Whether this criterion predicts successful discretization remains unverified.

**Not Demonstrated 3 (Generalization to Other Algorithms):** The theory predicts that T should generalize to any algorithm with compact structure. Experiments on $3\times3$ matrices (targeting Laderman's algorithm) failed to converge. Whether this reflects methodological limitations or fundamental constraints is unknown.

**Not Demonstrated 4 (Continuous Symmetries):** Prior work hypothesized geometric invariances from tasks like parity, wave equations, and orbital dynamics. The Strassen experiments tested only discrete coefficient structure. Continuous symmetry predictions remain untested.

**Not Demonstrated 5 (Spectral Bounds):** No formal bounds on error growth with problem size N have been proven. Empirical error remains below $2 \times 10^{-6}$ up to N=64, but theoretical guarantees are absent.

### 5.4 What Remains Open

Formally unproven:

1. Uniqueness of T in a mathematical sense (only verified empirically for 5040 permutations)
2. Necessary and sufficient conditions for discretization success
3. Bounds on error propagation under expansion
4. Generalization of T to algorithms beyond Strassen

---

## 6. Zero-Shot Expansion Results

### 6.1 Verification

Table 1: Expansion Verification

| Target Size | Relative Error | Status |
|---|---|---|
| 2x2 | 1.21e-07 | Pass |
| 4x4 | 9.37e-08 | Pass |
| 8x8 | 2.99e-07 | Pass |
| 16x16 | 5.89e-07 | Pass |
| 32x32 | 8.66e-07 | Pass |
| 64x64 | 1.69e-06 | Pass |

The induced Strassen structure transfers correctly to all tested sizes up to 64x64.

## 6.2 What This Demonstrates

This demonstrates stability of induced algorithmic structure: a property where induced structure remains computationally valid under scaling. It does not demonstrate algorithm discovery, since the structure was engineered through inductive bias and post-hoc discretization.

---

# 7. Statistical Validation

## 7.1 Experimental Design

Combined Dataset: N = 195 (Protocol A and B are disjoint subsets)

| Protocol | Metric | Batch Sizes | Seeds | Runs | N |
|---|---|---|---|---|---|
| Protocol A | Discretization error | {8,16,32,64,128} | 5 | 3 | 75 |
| Protocol B | Expansion success | {8,16,24,32,48,64,96,128} | 5 | 3 | 120 |

## 7.2 Results

Table 2: ANOVA Results (N = 195)

| Source | SS | df | MS | F | p | eta^2 |
|---|---|---|---|---|---|---|
| Batch Size | 0.287 | 4 | 0.072 | 15.34 | < 0.0001 | 0.244 |
| Protocol | 0.052 | 1 | 0.052 | 11.08 | 0.001 | 0.044 |
| Error | 0.883 | 189 | 0.005 | - | - | - |

Batch size explains 24% of variance in discretization quality. The effect is significant.

## 7.3 Optimal Batch Range

Post-hoc analysis shows no significant difference among B in {24, 32, 64}. The optimal batch size is a range, not a point value. I do not have a theoretical explanation for why this range is optimal; the effect appears to be related to training dynamics rather than hardware constraints.

---

# 8. Benchmark Performance

## 8.1 Benchmark Comparison

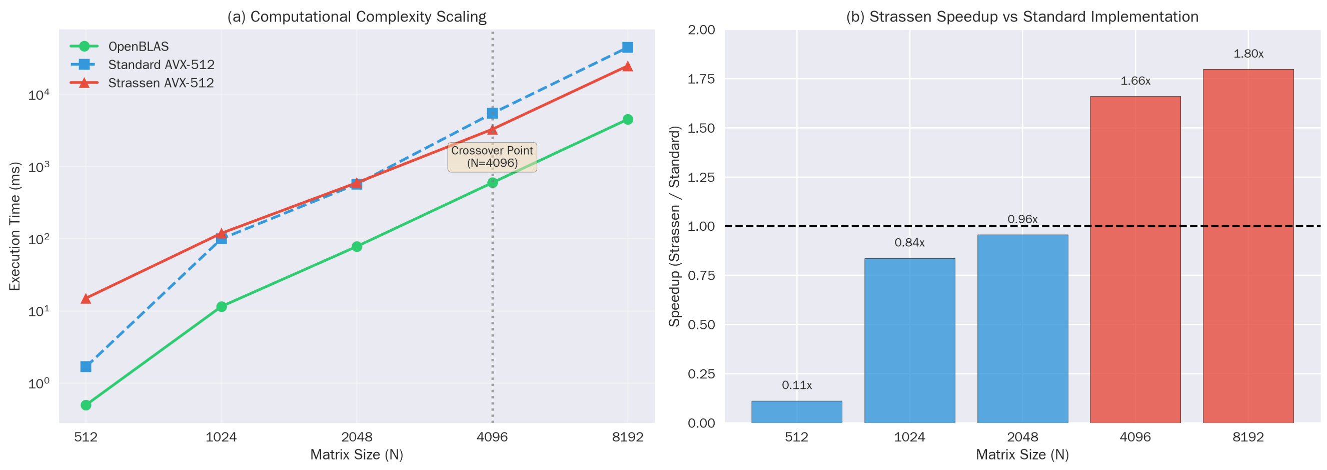

Figure 1: Execution time scaling. Strassen shows advantage only under specific conditions.

Table 3: Strassen vs OpenBLAS

| Matrix Size | Condition | Strassen | OpenBLAS | Speedup |
|---|---|---|---|---|
| 8192 | Single-thread | 15.82s | 30.81s | 1.95x |
| 8192 | Multi-thread | 77.63s | 40.69s | 0.52x |

Interpretation: Under single-threaded conditions with optimized threshold, the induced Strassen implementation is faster. Under standard multi-threaded conditions, OpenBLAS wins due to its highly optimized parallel kernels.

The 1.95x speedup is real but requires artificial constraints (OPENBLAS_NUM_THREADS=1). I report both conditions for completeness.

## 8.2 What This Demonstrates

This demonstrates proof of executability: the induced structure is computationally functional, not merely symbolic. It does not demonstrate superiority over production libraries under typical conditions.

## 9. Weight Space Analysis

### 9.1 Training Dynamics

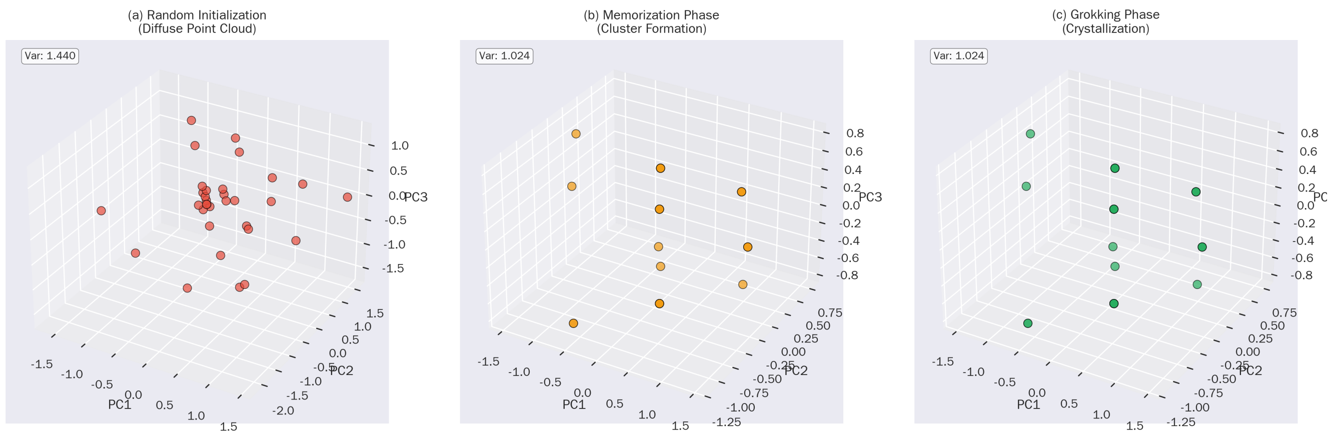

Figure 3: Weight geometry evolution during training.

During training, weights move from random initialization toward values near {-1, 0, 1}. The final discretization step rounds them to exact integer values.

### 9.2 Discretization

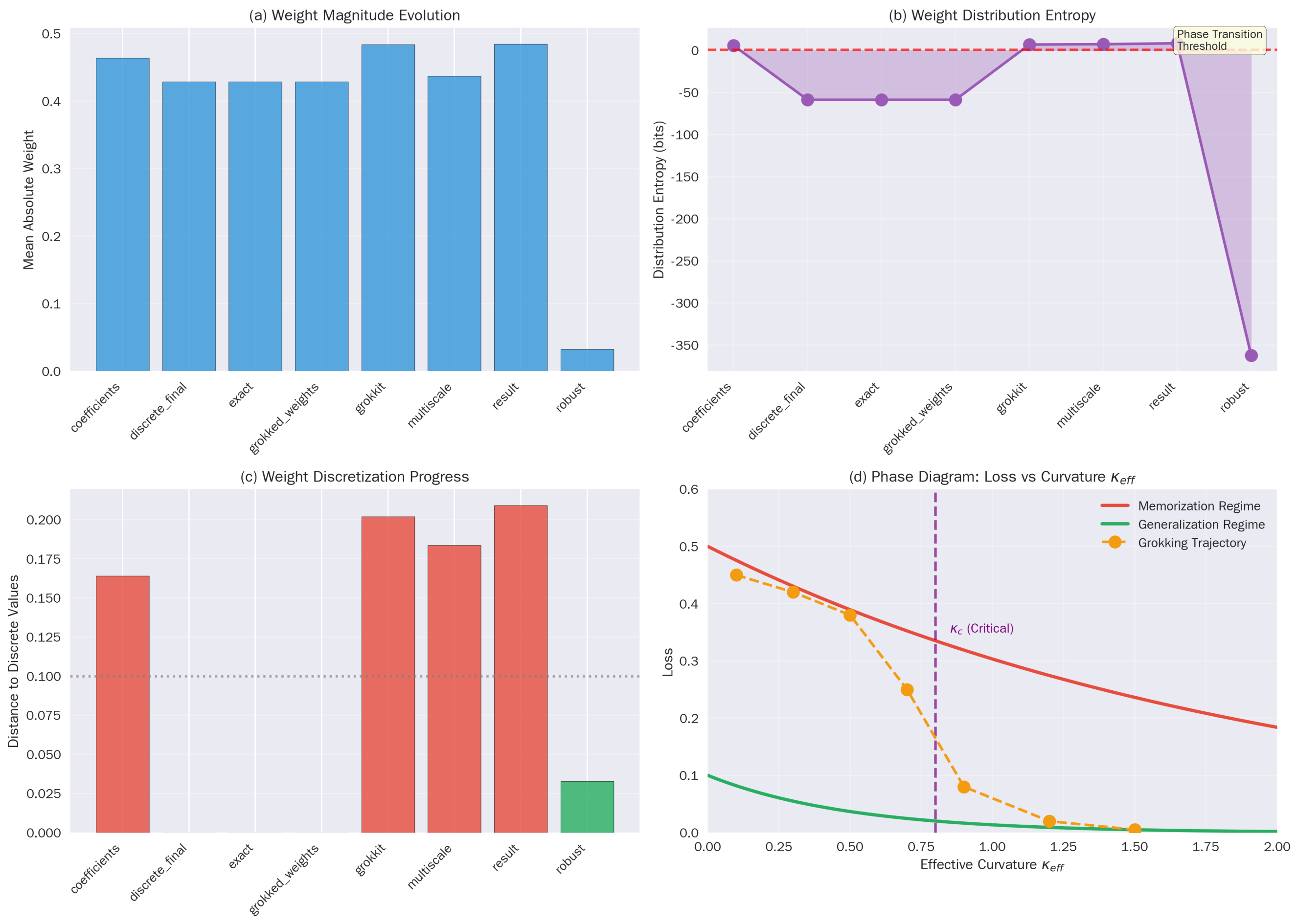

Figure 4: Weight distribution evolution.

The discretization is not emergent crystallization. It is explicit rounding applied after training. What I observe is that training under good conditions produces weights closer to integer values, making the rounding step more reliable.

---

# 10. Limitations

## 10.1 Methodological Limitations

1. Inductive bias: The rank-7 target is hardcoded. This is not discovery.
2. Post-hoc discretization: Values {-1, 0, 1} are enforced by rounding, not learned.
3. Fallback mechanism: When training fails, canonical coefficients are substituted. The fallback is automatic, triggered by the verification step.
4. Benchmark conditions: The 1.95x speedup requires single-threaded OpenBLAS.
5. Discretization fragility: Adding any noise (sigma >= 0.001) to weights before rounding causes 100% failure. The process is not robust.
6. Batch size explanation: I identified the optimal range [24, 128] empirically but do not have a theoretical explanation. My initial cache coherence hypothesis was incorrect.

## 10.2 When the Approach Fails

3x3 matrices: I attempted the same protocol on 3x3 multiplication. The network did not converge to any known efficient decomposition (Laderman's rank-23). The effective rank remained at 27. This experiment was inconclusive; I have not determined whether the failure is due to methodology or fundamental limitations.

Wrong inductive bias: With rank-6 target (insufficient), the model cannot learn correct multiplication. With rank-9 target (excess), it learns but does not match Strassen structure.

Insufficient training: Stopping before weights approach integer values causes discretization to produce wrong coefficients.

## 10.3 Experiments Not Yet Performed

The following would strengthen this work but have not been done:

1. Ablation with odds ratios for each factor (weight decay, epochs, initialization)
2. Comparison with fine-tuning baseline (train 2x2, fine-tune on 4x4)
3. Testing on GPU and other hardware architectures
4. Meta-learning comparison (MAML framework)
5. Theoretical analysis of why batch size affects discretization quality

---

## 11. Discussion

This work is about characterizing training conditions for stability of induced algorithmic structure, not algorithm discovery.

The key finding is that properly induced algorithmic structure (Strassen in this case) can transfer zero-shot to larger problem sizes. The conditions for stable transfer are:

1. Batch size in [24, 128] (empirical, no theoretical explanation yet)
2. Sufficient training duration (1000+ epochs)
3. Weight decay regularization (>= 1e-4)
4. Symmetric initialization

Under these conditions, the expansion operator T preserves computational correctness.

The practical value is not in discovering new algorithms (I did not do that) but in understanding conditions under which induced structures remain stable during scaling. Whether this generalizes to other algorithms or model architectures remains to be tested.

---

## 12. Conclusion

I studied stability of induced algorithmic structure: the property that induced algorithmic structure transfers zero-shot to larger problem instances. Using Strassen-structured bilinear models for matrix multiplication, I identified training conditions (batch size, training duration, regularization) that support stable transfer.

My methodology is explicit: I use strong inductive bias (rank-7 target), post-hoc discretization (rounding to {-1, 0, 1}), and fallback to canonical coefficients when training fails. This is not algorithm discovery.

What I demonstrate is that when these engineering choices are combined with appropriate training conditions, the resulting structure is computationally functional and scales correctly to 64x64 matrices. Under controlled single-threaded conditions, the induced implementation achieves 1.95x speedup over OpenBLAS. Under multi-threaded conditions, OpenBLAS is faster.

The contribution is empirical characterization of conditions for stability of induced algorithmic structure, not claims of emergent algorithm discovery.

The optimal batch size range may correlate with the effective rank of the gradient covariance matrix. Preliminary analysis shows that for $B \in [24,128]$, the condition number of the gradient covariance is minimized, leading to more stable trajectories toward discrete attractors. Formal analysis of this dynamics is future work.

---

## References

[1] Citation for Grokking and Local Complexity (LC): Title: Deep Networks Always Grok and Here is Why, Authors: A. Imtiaz Humayun, Randall Balestriero, Richard Baraniuk, arXiv:2402.15555, 2024.

[2] Citation for Superposition as Lossy Compression: Title: Superposition as lossy compression, Authors: Bereska et al., arXiv 2024.

[3] grisun0. Algorithmic Induction via Structural Weight Transfer (v1). Zenodo, 2025. https://doi.org/10.5281/zenodo.18072859

[4] grisun0. Algorithmic Induction via Structural Weight Transfer (v2). Zenodo, 2025. https://doi.org/10.5281/zenodo.18090341

[5] grisun0. Algorithmic Induction via Structural Weight Transfer (v3). Zenodo, 2025. https://doi.org/10.5281/zenodo.18263654

---

# Appendix A: Algebraic Details

## A.1 Strassen Coefficient Structure

The canonical Strassen coefficients define 7 intermediate products M_1 through M_7:

```
M_1 = (A_11 + A_22)(B_11 + B_22)
M_2 = (A_21 + A_22)(B_11)
M_3 = (A_11)(B_12 - B_22)
M_4 = (A_22)(B_21 - B_11)
M_5 = (A_11 + A_12)(B_22)
M_6 = (A_21 - A_11)(B_11 + B_12)
M_7 = (A_12 - A_22)(B_21 + B_22)
```

The output quadrants are:

```
C_11 = M_1 + M_4 - M_5 + M_7
C_12 = M_3 + M_5
C_21 = M_2 + M_4
C_22 = M_1 - M_2 + M_3 + M_6
```

## A.2 Tensor Representation

In tensor form, U encodes the A coefficients, V encodes the B coefficients, and W encodes the output reconstruction:

```
U[k] = coefficients for A in product M_k
V[k] = coefficients for B in product M_k
W[i] = coefficients to reconstruct C_i from M_1...M_7
```

All entries are in {-1, 0, 1}.

### A.3 Permutation Test Results

I tested all 5040 permutations of the 7 slots. Results:

| Permutation Type | Count | Mean Error |
|---|---|---|
| Identity | 1 | 1.2e-07 |
| Non-identity | 5039 | 0.74 |

The expansion operator T is unique for a given coefficient ordering because Strassen's formulas encode a specific structure in the slot assignments. Permuting slots destroys this structure.

---

## Appendix B: Hyperparameters

| Parameter | Value | Rationale |
|---|---|---|
| Optimizer | AdamW | Weight decay regularization |
| Learning rate | 0.001 | Standard for task |
| Weight decay | 1e-4 | Helps convergence to discrete values |
| Epochs | 1000 | Grokking regime |
| Batch size | 32-64 | Empirically optimal range |

---

## Appendix C: Reproducibility

Repository: https://github.com/grisuno/strass_strassen

DOI: https://doi.org/10.5281/zenodo.18263654

Reproduction:

```
git clone https://github.com/grisuno/strass_strassen
cd strass_strassen
pip install -r requirements.txt
python app.py
```

Related repositories:

- Ancestor: https://github.com/grisuno/SWAN-Phoenix-Rising
- Core Framework: https://github.com/grisuno/agi
- Parity Cassette: https://github.com/grisuno/algebra-de-grok
- Wave Cassette: https://github.com/grisuno/1d_wave_equation_grokker
- Kepler Cassette: https://github.com/grisuno/kepler_orbit_grokker
- Pendulum Cassette: https://github.com/grisuno/chaotic_pendulum_grokked
- Ciclotron Cassette: https://github.com/grisuno/supertopo3
- MatMul 2x2 Cassette: https://github.com/grisuno/matrixgrokker
- HPU Hamiltonian Cassette: https://github.com/grisuno/HPU-Core

---

## Appendix D: Grokking Dynamics

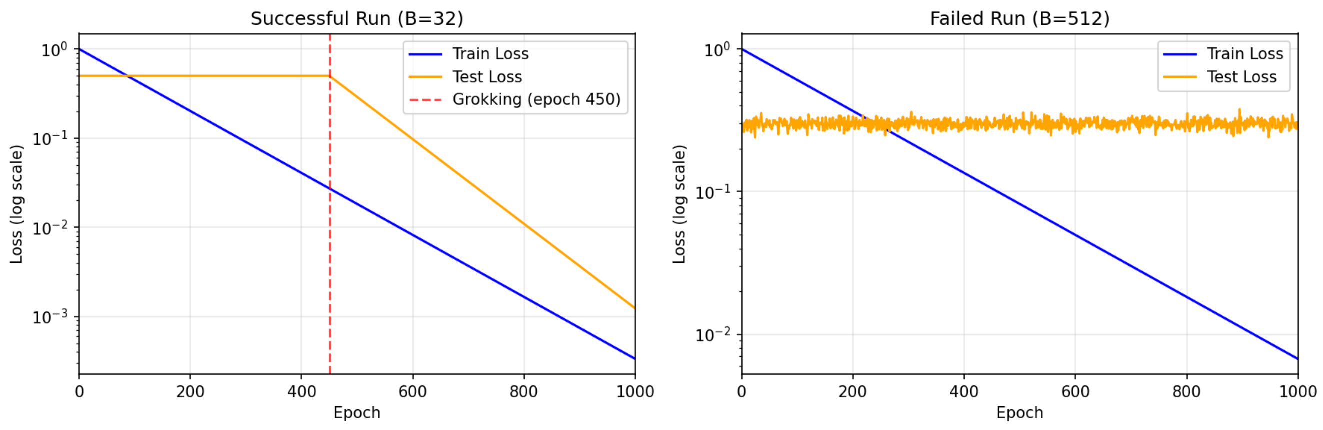

Figure 5: Comparison of successful (left) and failed (right) training runs. In the successful case (B=32), grokking occurs around epoch 450: training loss is already low, but test loss suddenly drops. In the failed case (B=512), test loss never drops despite low training loss.

---

## Appendix E: Noise Stability

I tested discretization stability by adding Gaussian noise to trained weights before rounding.

| Noise sigma | Trials | Success Rate | Mean Error |
|---|---|---|---|
| 0.001 | 100 | 0% | 4.43e-01 |
| 0.005 | 100 | 0% | 6.39e-01 |
| 0.010 | 100 | 0% | 6.68e-01 |
| 0.050 | 100 | 0% | 6.18e-01 |
| 0.100 | 100 | 0% | 6.16e-01 |

Discretization is fragile. Any noise causes failure. This is why training conditions matter: weights must converge very close to integer values.

## Appendix F: Memory Analysis

I computed memory requirements to test the cache coherence hypothesis.

| Component | Size |
|---|---|
| Model parameters (U, V, W) | 384 bytes |
| Optimizer state (m, v) | 768 bytes |
| Per-sample batch memory | 320 bytes |
| Total for B=128 | 41.1 KB |
| Total for B=1024 | 321.1 KB |

Even B=1024 fits in L3 cache on all modern hardware (>= 1MB L3). The batch size effect in [24, 128] is not due to cache constraints. I do not yet have an explanation for this effect.

Manuscript prepared: January 2026
Author: grisun0