Shortening the Loss Plateau

Jet Yue jyue@ucsd.edu
Tommy Li wal016@ucsd.edu
Samuel Cho sjc006@ucsd.edu
Tianhao Wang tianhaowang@ucsd.edu

Abstract

Transformers often exhibit two distinct optimization phenomena during training: an early training-loss plateau, where loss remains near a suboptimal value before rapidly decreasing, and a later grokking phase, where training accuracy saturates while test accuracy improves only after a long delay. Although both involve stalled progress, they occur at different stages of learning and reflect different underlying dynamics.

We first replicate the training-loss plateau and confirm prior observations of representation collapse, repetition bias, and delayed formation of structured attention maps. We show that introducing task diversity through multi-task training significantly shortens this plateau, suggesting that diverse objectives encourage more robust representations and faster convergence.

We then study grokking separately, evaluating the effects of task diversity, different optimizers, and initialization constraints. Our results indicate that while the training plateau and grokking are mechanistically distinct, interventions that promote stable and structured representations can accelerate both phases of learning.

Website: https://jetyue04.github.io/loss-plateau/
Code: https://github.com/jetyue04/loss-plateau

1 Introduction

Despite their widespread success, Transformer architectures frequently encounter severe optimization stagnation during training. These stalls typically manifest as extended flat regions in the learning curves, representing a major bottleneck in both computational efficiency and model development. Rather than treating these plateaus as a single unified problem, our work investigates them as two fundamentally distinct phenomena that emerge at entirely different stages of a model's lifecycle. Specifically, we focus on the early *training-loss plateau*—where the model struggles to form foundational representations—and the later *grokking phase*—where the model quickly memorizes training data but requires a massive, delayed optimization period to finally generalize. By dissecting both phases, we aim to uncover their unique mechanistic drivers and identify targeted interventions to accelerate the learning process.

1.1 Loss Plateau

The training loss plateau refers to an extended phase during which training loss remains near a suboptimal value before abruptly decreasing. Prior work attributes this behavior to representation collapse, repetition bias in token embeddings, and delayed formation of structured attention maps Gopalani and Hu (2025).

We replicate this phenomenon on moving-window tasks (MWS, MWP) using shallow Transformers. During the plateau, we observe clear signs of representation collapse and repetition bias in the embeddings, and attention structure forms slowly during this period, supporting the hypothesis that delayed alignment of representations contributes to stalled optimization.

We then evaluate whether **task diversity** Kim et al. (2025) can shorten the plateau. Our results show that multi-task training significantly reduces plateau duration compared to single-task training. Full results and figures are presented in Section 3.

1.2 Grokking

In addition to the training-loss plateau, we study grokking, a phenomenon in which models achieve near-zero training loss early but exhibit delayed generalization Power et al. (2022). Our baseline run on the modular division task shows that training accuracy rises to 100% rapidly, while validation accuracy remains flat for a prolonged period before being able to fully generalize. So what causes this delayed transition from overfitting to generalization, and can it be accelerated? Understanding and controlling grokking has practical implications for reducing the computational cost of training models and for revealing the mechanisms by which neural networks develop structured internal representations.

Power et al. (2022) first identified and named this phenomenon by training small transformers on modular arithmetic tasks, observing that models would first memorize the training

set before suddenly surging toward perfect generalization. They also found that weight decay was essential to the emergence of grokking. Future theoretical research has suggested that this transition is driven by a "dichotomy" of implicit biases during training, where early phases prioritize an overfitting solution or simple memorization of training samples while later phases shift toward discovering low-rank or structurally efficient generalizing solutions Lyu et al. (2024). They also suggest that changing weight decay as well as initialization scale, can either delay or speed up the transition from overfitting to generalization.

Although prior work briefly hinted at how these factors could accelerate grokking, we are specifically interested in how grokking can be accelerated. In this work, we investigate other factors that have not yet been fully explored in this context, with a primary focus on **task diversity**, specifically training on multiple modular arithmetic tasks rather than a single one. We use synthetically generated modular arithmetic datasets, following the setup of Power et al. (2022), in which all possible equations of the form $a \odot \equiv c \pmod{p}$ are enumerated for a prime modulus $p = 97$ and operations $\odot \in \{\div, +, -, \times\}$. This synthetic setting is well-suited for studying grokking because the data is fully controlled, the ground truth structure is known, and generalization can be measured exactly. We train a small decoder-only transformer on these tasks and vary the combination of tasks presented during training, measuring the effect on grokking speed. Our results show that multi-task training substantially accelerates grokking across all tasks, with certain task combinations — particularly those involving modular division and multiplication together — producing the most extreme acceleration.

We also investigate the effect of optimizer choice and highly constrained initialization schemes. Switching from AdamW to SGD introduces gradient noise that accelerates grokking by roughly 60%, while sparse or small-scale initialization effectively eliminates the grokking delay entirely—achieving generalization in ∼1,050 steps versus the baseline's 334,000. Taken together, our results suggest that both forms of optimization stagnation can be shortened by promoting diverse representations and restricting initial network capacity.

2 Methods

2.1 Loss Plateau

2.1.1 Model Architecture

We use a shallow Transformer consisting of a single attention layer with **linear causal attention** and a 2-layer feedforward MLP. Each token is embedded and summed with an absolute positional embedding. Pre-LayerNorm is applied before the attention and MLP blocks, and a final linear layer maps hidden states to logits for sequence generation. The MLP uses a GELU activation, with intermediate dimensionality four times the hidden dimension. For sequence generation, **greedy decoding** is employed, selecting the token with the highest logit at each step.

Formally, for a sequence of tokens $(s_1, \ldots, s_L)$, the model computes:

$$\mathrm{TF}_\theta(s_1, \ldots, s_L) = \mathrm{LM} \circ (\mathrm{Id} + \mathrm{MLP}) \circ (\mathrm{Id} + \mathrm{Attn}) \circ \mathrm{Embed}(s_1, \ldots, s_L),$$

where `Embed` produces token plus positional embeddings, `Attn` is the causal linear attention operation:

$$[\mathrm{Attn}(h_1, \ldots, h_L)]_i = W_O \left(\sum_{j=1}^{i} (h_j^\top W_K^\top W_Q h_i) W_V h_j \right), \quad W_O, W_K, W_Q, W_V \in \mathbb{R}^{d \times d},$$

and `LM` is the output projection to the vocabulary. Pre-LayerNorm is applied before the attention and MLP blocks, and residual connections ensure stable training.

Embedding Layer (`Embed`): Each token s_i is mapped to a d-dimensional vector h_i using learned token embeddings plus an absolute positional embedding:

$$h_i = \mathrm{TokenEmbed}(s_i) + \mathrm{PosEmbed}(i).$$

Feedforward MLP (`MLP`): The MLP is 2-layer with GELU activation and hidden size $4d$:

$$\mathrm{MLP}(x) = W_2 \, \mathrm{GELU}(W_1 x + b_1) + b_2.$$

Output Projection (`LM`): The final hidden states are mapped to logits for sequence generation:

$$\mathrm{logits}_i = W_{\mathrm{LM}} h_i + b_{\mathrm{LM}}, \quad W_{\mathrm{LM}} \in \mathbb{R}^{|V| \times d}.$$

Greedy Decoding: The next token is selected as the one with the highest logit:

$$\hat{s}_{i+1} = \arg\max_k \mathrm{logits}_i[k].$$

2.1.2 Training Procedure

Models are trained **online**, drawing a fresh batch of 256 sequences at each step. The training objective is standard **next-token cross-entropy loss** over the output portion of each sequence. Accuracy is measured on the generated output sequence.

For multi-task experiments, batches contain sequences from multiple tasks, with samples distributed evenly across tasks. To ensure **fair comparisons** between single-task and multi-task setups, we fixed the following:

- **Batch size:** constant across all experiments.
- **Vocabulary size:** identical for single-task and multi-task runs.
- **Model architecture:** the same 1-layer, 1-head Transformer is used throughout.
- **Number of examples per task:** comparisons are made based on the number of examples seen per task, rather than raw training steps, to account for the increased task diversity in multi-task batches.

Sequences follow the general format:

$$x_1, x_2, \ldots, x_n, [\mathbf{SEP}]_k, y_1, y_2, \ldots, y_n,$$

where $[\mathbf{SEP}]_k$ denotes the task-specific separator token ($k = 0$ for single-task training, $k \geq 1$ for multi-task batches). Multi-task batches combine sequences from different tasks to promote **diverse representations**, while keeping the number of examples per task fixed.

2.1.3 Algorithmic Tasks

We use several deterministic sequence-to-sequence tasks, including:

- **Moving Window Sum (MWS):** $y_i = x_1$ if $i = 1$, else $(x_{i-1} + x_i) \bmod p$.
- **Moving Window Product (MWP):** $y_i = x_1$ if $i = 1$, else $(x_{i-1} \cdot x_i) \bmod p$.
- **Moving Window Difference (MWD):** $y_i = x_1$ if $i = 1$, else $(x_i - x_{i-1}) \bmod p$.
- **Prefix Sum (PS):** $y_i = \sum_{j=1}^{i} x_j \bmod p$.

Sequence length is $n = 16$, modulus $p = 17$, and the vocabulary includes integers 0 through $p + (\#\text{tasks}) - 1$. Task-specific separator tokens ensure that multi-task batches are distinguishable.

2.2 Grokking

2.2.1 Model Architecture

We employ a decoder-only Transformer with causal self-attention for predicting modular arithmetic results. The architecture consists of token embeddings summed with learned positional encodings (fixed to 5 positions for our input sequences), followed by stacked Transformer encoder layers with causal masking, and a final linear projection to vocabulary logits.

Specifically, the model uses multi-head self-attention with h heads, each operating on dimension d_{model}/h, followed by a feedforward network with hidden dimension $4 \times d_{\text{model}}$ and ReLU activation. Pre-LayerNorm is applied before both the attention and feedforward sublayers, with residual connections around each.

Formally, for an input sequence of tokens $(t_1, \ldots, t_L)$, the model computes:

$$\text{TF}_\theta(t_1, \ldots, t_L) = \text{Linear}_{\text{out}} \circ \text{TransformerStack} \circ (\text{TokenEmbed} + \text{PosEmbed})(t_1, \ldots, t_L),$$

where `TokenEmbed` and `PosEmbed` produce token and positional embeddings respectively, `TransformerStack` applies n_{layers} transformer encoder layers with causal masking, and $\texttt{Linear}_{\text{out}}$ projects the final hidden state of the last token to vocabulary logits.

For our experiments, we use $d_{\text{model}} = 128, h = 4$ attention heads, and $n_{\text{layers}} = 2$, with dropout set to 0.0 to facilitate grokking observations.

2.2.2 Training Procedure

Models are trained using **step-based training** rather than epoch-based training, allowing precise control over optimization steps. We employ the AdamW optimizer with learning rate $\eta = 10^{-3}$, weight decay $\lambda = 10^{-3}$, and $\beta = (0.9, 0.98)$. The training objective is **cross-entropy loss** on the predicted result token.

Weight decay is essential for observing grokking. Without sufficient regularization, models memorize the training set without later generalizing. Each training batch contains 512 randomly sampled equations, and training proceeds for 400,000 steps with evaluation every 50 steps.

For multi-task experiments, training data is shuffled such that batches contain equations from multiple tasks. To ensure **fair comparisons** between single-task and multi-task setups, we scale the total number of training steps proportionally with the number of tasks. Specifically, we train for an additional 400,000 steps per additional task (e.g., a four-task run trains for $400,000 \times 4 = 1,600,000$ steps). This ensures that the number of training examples seen per task remains consistent across all experimental conditions, including the baseline. We control for the following additional factors:

- **Batch size:** constant at 512 across all experiments.
- **Vocabulary size:** identical across conditions, comprising special tokens (operator separator, equals sign), task identifier tokens ($\langle DIV \rangle, \langle ADD \rangle, \langle SUB \rangle, \langle MULT \rangle$), and number tokens (0 through $p-1$).
- **Model architecture:** identical across all task configurations.
- **Data split:** 50% train-validation split applied to each task independently, ensuring validation sets remain task-separated for per-task evaluation.

Validation accuracy is tracked separately per task rather than aggregated, allowing us to observe each task's individual generalization behaviors when trained alongside different task combinations. We detect grokking as the point at which a given task first achieves $\geq 95\%$ validation accuracy.

To enable direct comparison across experiments with different numbers of tasks and total step counts, we report all results in terms of **training progress percentage**, or the fraction of total training steps completed, rather than absolute step count. This ensures that grokking timings remain directly comparable regardless of the total length of each run.

Equation sequences follow the format:

$$\langle \text{TASK} \rangle, \text{num}_x, \text{op}, \text{num}_y, \text{eq}, \text{num}_{\text{result}},$$

where $\langle \text{TASK} \rangle \in \{\langle DIV \rangle, \langle ADD \rangle, \langle SUB \rangle, \langle MULT \rangle\}$ identifies the operation, x and y are operands, and the model predicts the result token. The model receives the first five tokens as input and predicts the final result token.

2.2.3 Modular Arithmetic Tasks

We focus on four fundamental modular arithmetic operations over $\mathbb{Z}_p$ with prime modulus $p = 97$:

- **Modular Division (DIV):** $(x, y) \mapsto x \cdot y^{-1} \bmod p$ (for $y \neq 0$).
- **Modular Addition (ADD):** $(x, y) \mapsto (x + y) \bmod p$.
- **Modular Subtraction (SUB):** $(x, y) \mapsto (x - y) \bmod p$.
- **Modular Multiplication (MULT):** $(x, y) \mapsto (x \cdot y) \bmod p$.

For division, $x \in \{0, \ldots, p-1\}$ and $y \in \{1, \ldots, p-1\}$ (excluding division by zero), yielding $97 \times 96 = 9{,}312$ unique equations. For the other three operations, both $x, y \in \{0, \ldots, p-1\}$, yielding $97 \times 97 = 9{,}409$ equations each. The vocabulary size is $|V| = 103$: 6 special tokens plus 97 number tokens.

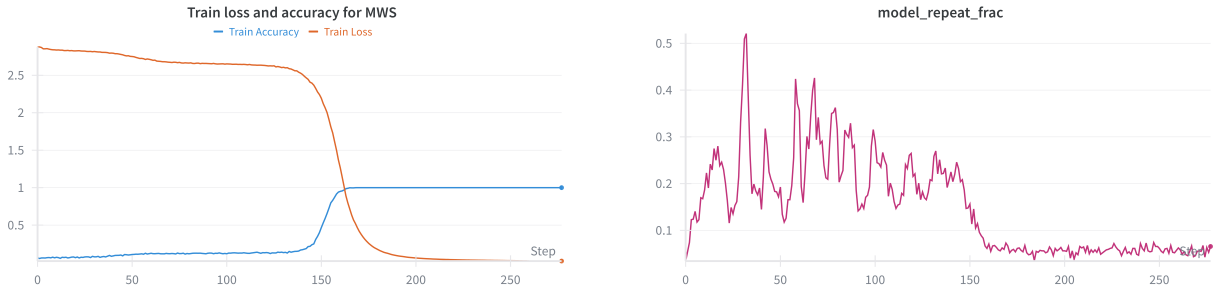
Task-specific separator tokens ($\langle$DIV$\rangle$, $\langle$ADD$\rangle$, etc.) enable the model to distinguish operations in multi-task settings. We track grokking by monitoring when validation accuracy first exceeds 95% for each task independently, allowing us to observe task-dependent generalization dynamics in multi-task training scenarios.

3 Results

3.1 Loss Plateau

3.1.1 Baseline Replication: Moving Window Sum

We replicated the training-loss plateau on MWS using a 1-layer, 1-head Transformer. Figure 1 shows the training loss remaining near a suboptimal value for many steps before rapidly decreasing. Repetition bias is clearly visible in the token embeddings during this period. Figure 2 shows the cosine similarity between token embeddings and the attention map structure; both remain unstructured during the plateau and only converge once loss begins to fall.



(a) Training and test loss with accuracy overlay

(b) Repetition bias in embeddings

Figure 1: Baseline replication results for the one-layer, one-head transformer on MWS.

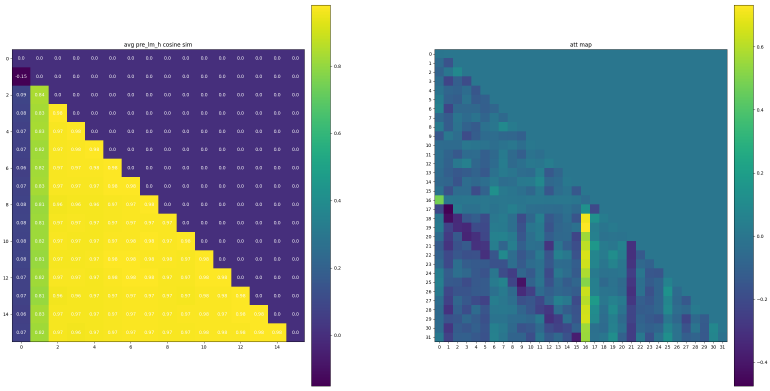


Figure 2: Cosine similarity between token embeddings (left) and attention map (right) during the plateau.

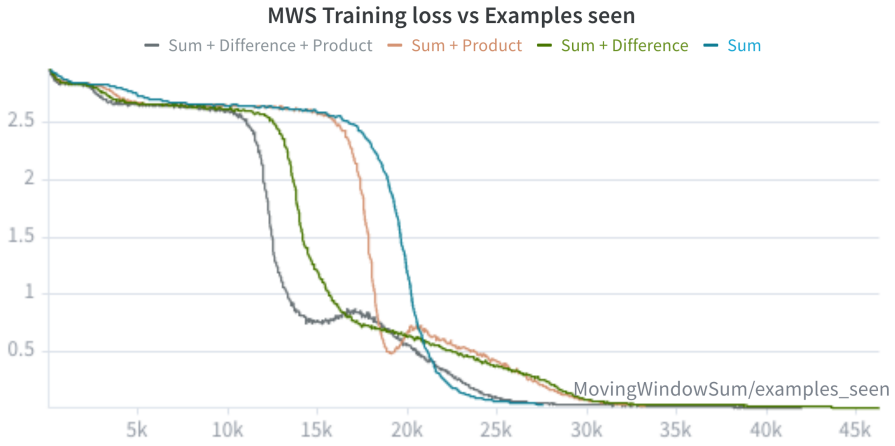
3.1.2 Task Diversity

We train the model on multiple algorithmic tasks simultaneously (e.g., MWS, MWP, MWD, PS) using balanced batches. This setup examines whether shared representations across tasks can accelerate the loss plateau on algorithmic tasks. To ensure fairness, the overall training batch size is kept constant across experiments. When multiple tasks are included in a batch, the number of training samples is divided evenly among all tasks, and each task is uniquely identified by a separate separator token. Notably, the loss plateau can emerge multiple times: the model often learns one task first, then others sequentially. We measure the plateau in terms of the total number of training samples observed until each task is fully learned.

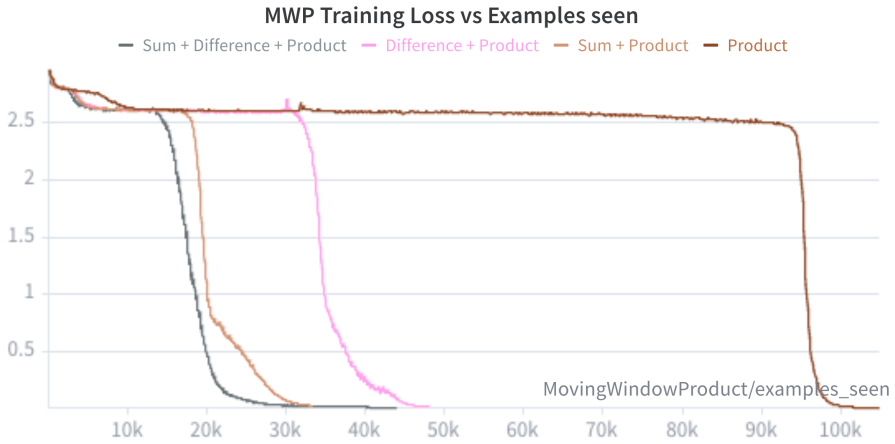
As shown in Fig. 3, the duration of the loss plateau shortens significantly as more tasks are incorporated into training. This acceleration is summarized in Table 1. We can observe that for MWS and MWD, the plateau shortens moderately, whereas for MWP, the acceleration is especially pronounced. Furthermore, as more tasks are incorporated, the speedup generally increases across all tasks. For example, for MWP, the plateau drops from 93,568 examples (single-task) to 13,288 examples with three tasks, corresponding to a 63% speedup. This suggests that multi-task training encourages the model to learn shared representations that generalize across tasks, reducing the number of samples required to reach the loss plateau for each individual task.

Sequential Learning Effects: In addition to shortening the plateau, multi-task training changes the shape of the loss curve. Instead of a single extended plateau, the loss often exhibits multiple rises or spikes, corresponding to the sequential learning of different tasks within the batch. Each spike reflects a temporary increase in loss as the model adjusts to the requirements of a new task, after which the loss decreases again as the model integrates the new information. This pattern illustrates that the plateau is no longer a monolithic block

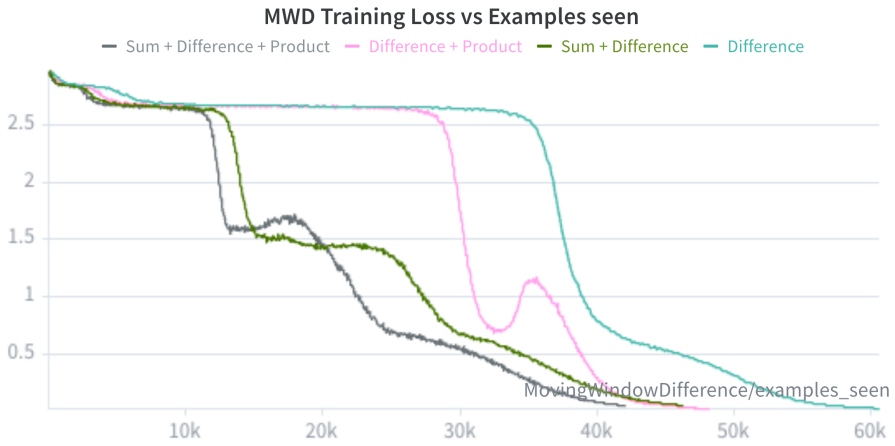
of stalled learning but a series of task-specific adaptation phases, highlighting the dynamic interplay of learning across multiple objectives.



(a) Moving Window Sum



(b) Task 2



(c) Moving Window difference

Figure 3: Multi-task training reduces the loss plateau for each task compared to single-task training.

Table 1: Plateau Reduction Summary. For each task and multi-task configuration, the table shows the number of training examples seen until the loss plateau ends. The last column reports the average % speedup relative to the single-task baseline, computed as $1-$(Current/Baseline).

#Tasks	MWS	MWP	MWD	AVG % Speedup
1	16,256	93,568	34,560	N/A
2	13,600	24,096	19,680	44.55%
3	10,542	13,288	10,794	63.24%

3.1.3 Transfer Learning

Since incorporating task diversity shortens the loss plateau and allows the model to learn tasks using fewer training samples, we investigate this phenomenon via **transfer learning**. We pretrain the model on a simpler task before fine-tuning on a more difficult task. Here, we examine two scenarios: when the tasks share the same target function and when the tasks share the same attention map.

We observed that pretraining significantly shortens the loss plateau. The attention matrices (**K, Q, V**) remain largely stable, while their respective bias terms shift significantly. These observations suggest that the attention mechanism captures general structural patterns of the sequence, while the MLP layer flexibly adapts to task-specific output requirements.

Same Target Function: Prefix Sum → MWS
Pretraining on **Prefix Sum** and then fine-tuning on **Moving Window Sum (MWS)** eliminates the loss plateau entirely, as shown in Fig. 4. Here, the attention map is reorganized while the target function is shared between the two tasks. The attention map is able to be reformed within 10 steps, as shown in Fig. 5. Examination shows that the attention matrix weights remain stable while bias terms shift significantly. The MLP weights also remain stable. Freezing the MLP layer and rerunning the experiment reproduces the same phenomenon. The loss plateau can be replicated if the attention layer is reinitialized.

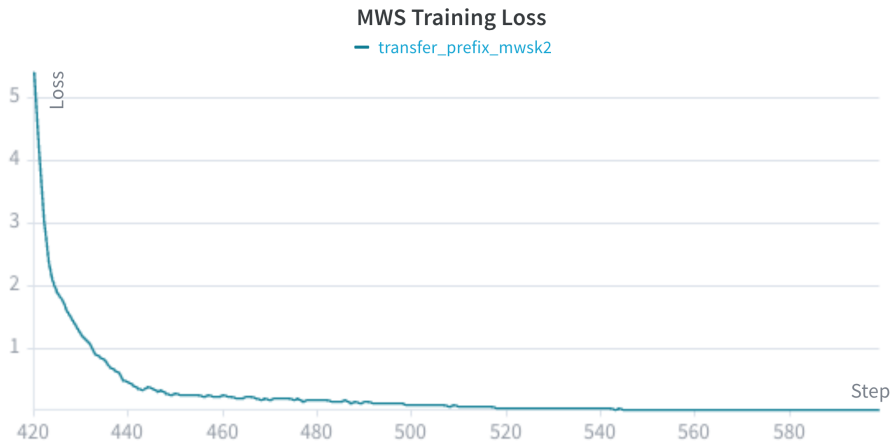


Figure 4: Attention maps for transfer learning from Prefix Sum to MWS. Left: initial attention map. Right: final attention map after fine-tuning.

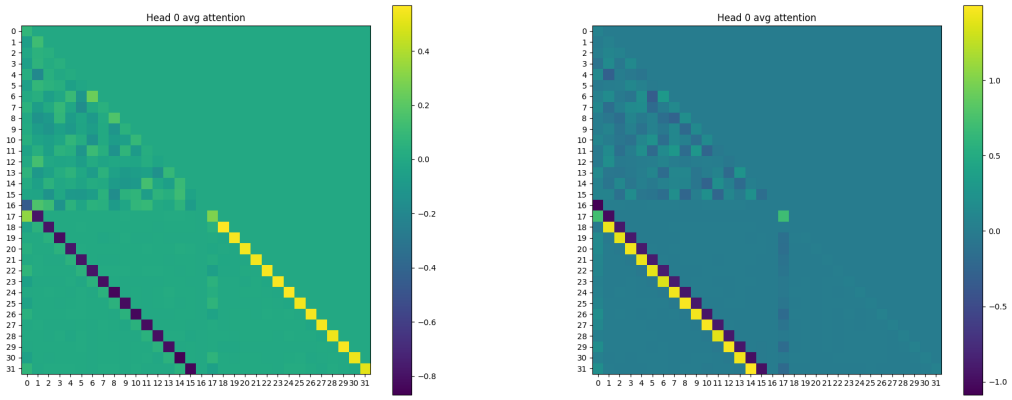


Figure 5: Detailed attention maps for transfer learning from Prefix Sum to MWS. Left: initial attention map. Right: final attention map after fine-tuning.

Same Attention Map: MWS → MWP

Pretraining on **MWS** and fine-tuning on **Moving Window Product (MWP)** significantly shortens the loss plateau, as shown in Fig. 6. Here, the attention map remains largely unchanged, while the target functions differ between tasks. Analysis indicates that attention matrix weights remain stable, while the MLP layer weights shift significantly to accommodate the new target function.

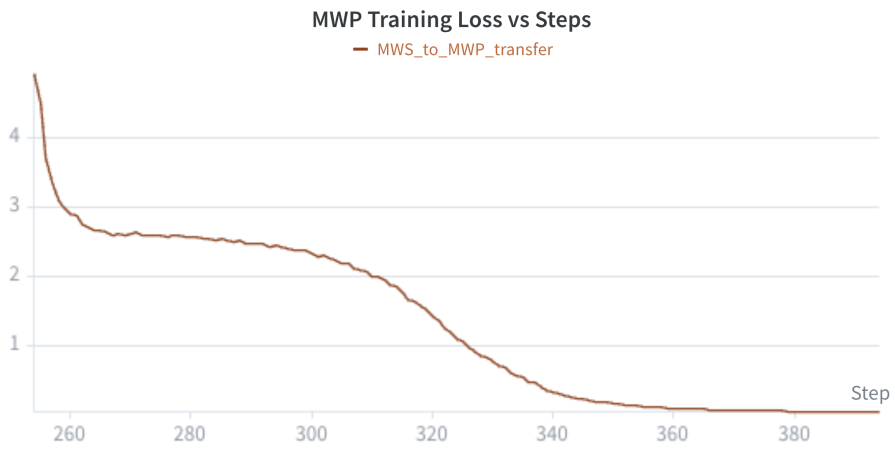


Figure 6: Loss vs steps curve for transfer learning from MWS to MWP.

These results complement our Task Diversity experiments, showing that prior exposure to related tasks enables the model to generalize more quickly and reduces training requirements. Visual inspection of attention maps shows that the overall pattern of attended positions is preserved, but the magnitude of certain attention components shifts slightly to accommodate the new task. Overall, these experiments demonstrate that **transfer learning effectively leverages previously learned representations to reduce the loss plateau**. When the target function is shared, the plateau can be entirely eliminated, and when only the attention structure is shared, the plateau shortens significantly. This highlights the complementary roles of the attention mechanism in capturing general sequence patterns and the MLP in task-specific adaptation.

3.2 Grokking

We first reproduced the results from Power et al. (2022) on a modular division task, serving as our baseline.

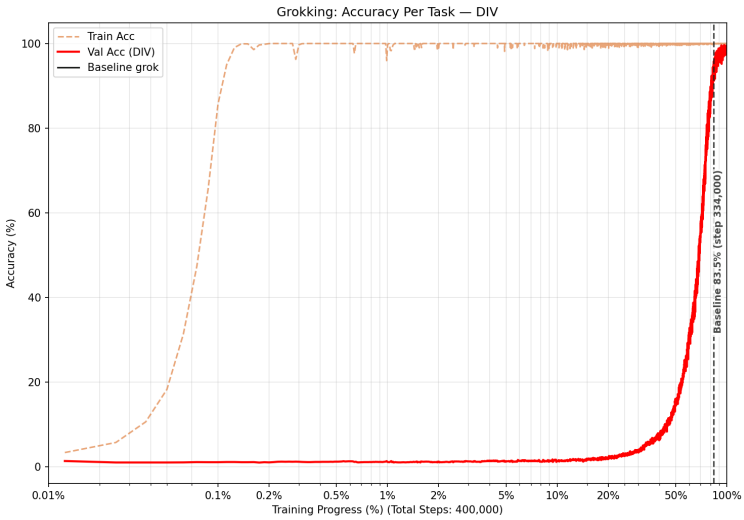


Figure 7: Baseline implementation of "grokking"—the sudden transition from memorization to generalization—during the training of a neural network on a modular division (DIV) task. The dashed orange line represents training accuracy and the solid red line represents validation accuracy. Note the logarithmic scale on the x-axis, which highlights the temporal gap between perfect training performance and true generalization. The dashed vertical line marks the moment it reaches 95% validation accuracy.

We then implemented task diversity by training all four arithmetic tasks at the same time to see its effect on grokking acceleration.

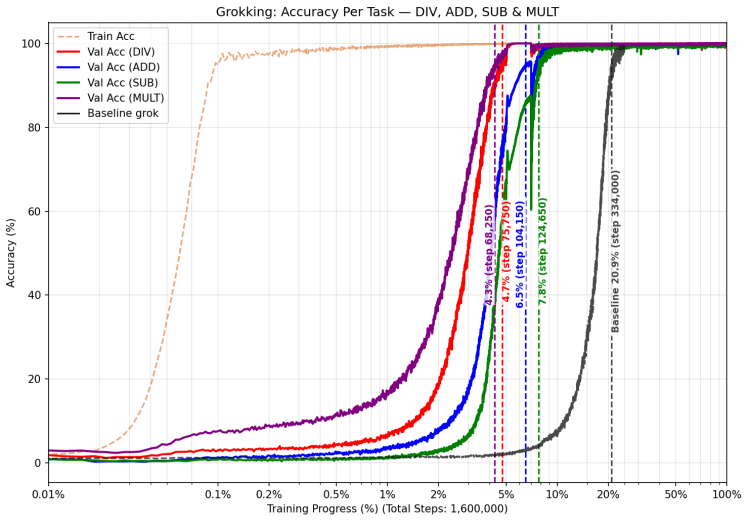
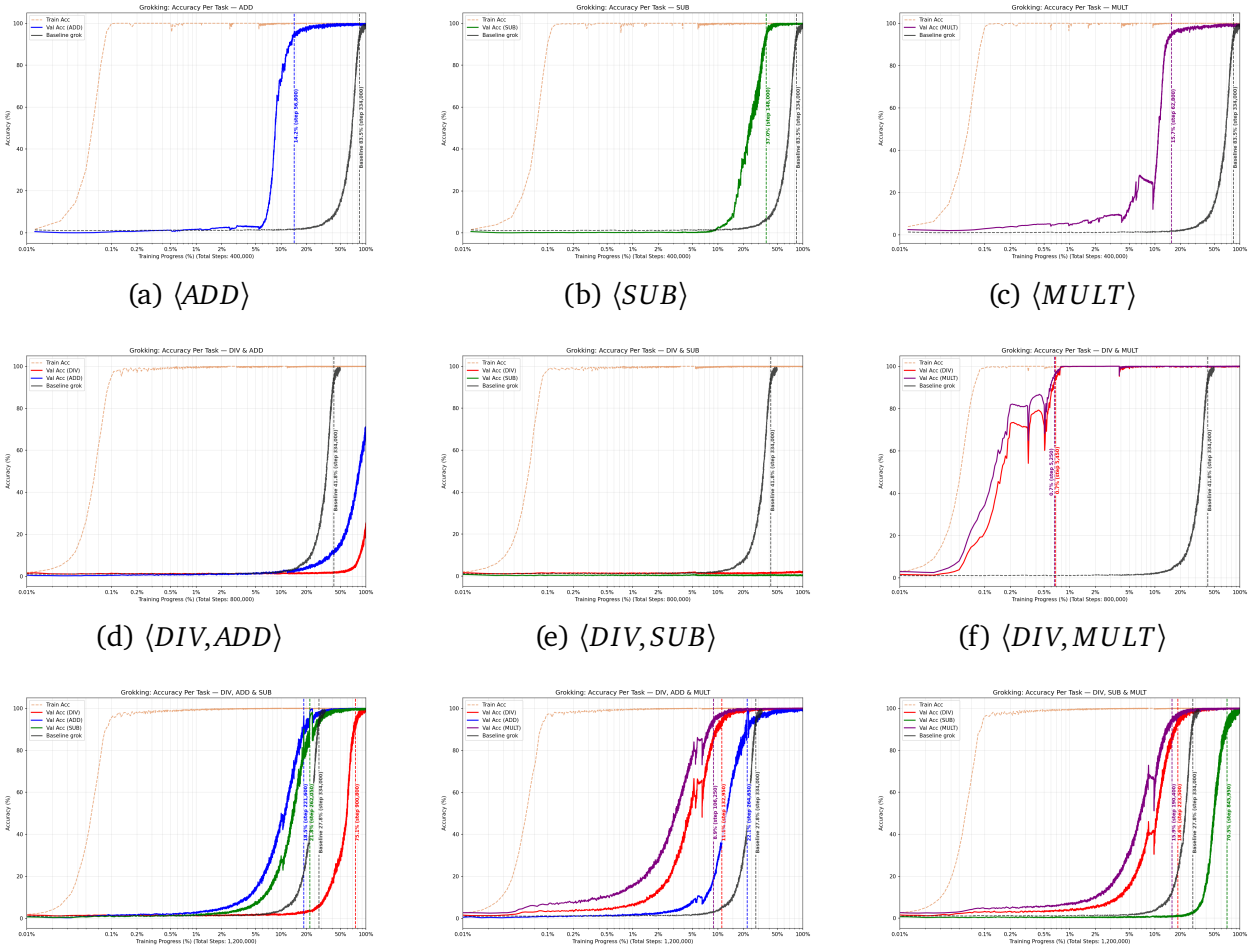


Figure 8: Model trained on all four arithmetic tasks simultaneously (division, addition, subtraction, and multiplication). The black curve represents our baseline model trained only on modular division. Modular division (red) is trained alongside addition (blue), subtraction (green), and multiplication (purple). The dashed vertical lines mark the moment the validation accuracies for each specific task reached 95%, with the color corresponding to the respective task's validation curve. The training accuracy is still represented by the dashed orange line, and the x-axis is in the same logarithmic scale.

Finally, we trained models on every task combination containing the baseline modular division task to further investigate the effect of specific singular and multi-task training on grokking acceleration.



(a) $\langle ADD \rangle$ (b) $\langle SUB \rangle$ (c) $\langle MULT \rangle$

(d) $\langle DIV, ADD \rangle$ (e) $\langle DIV, SUB \rangle$ (f) $\langle DIV, MULT \rangle$

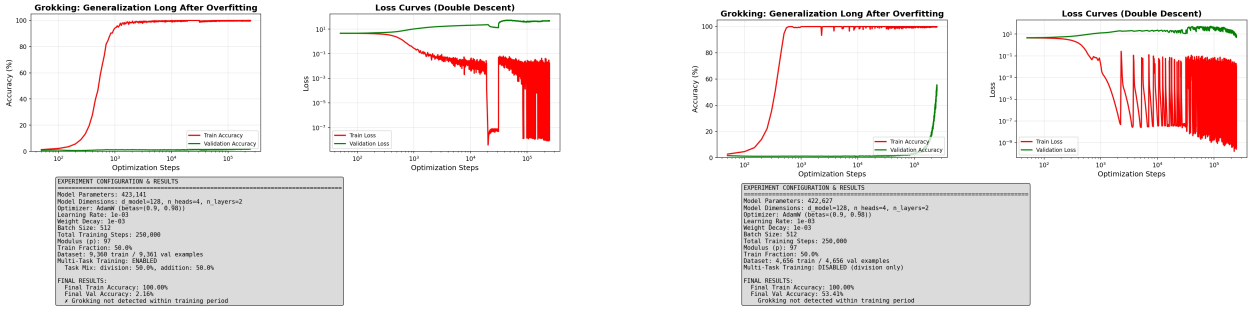
(g) $\langle DIV, ADD, SUB \rangle$ (h) $\langle DIV, ADD, MULT \rangle$ (i) $\langle DIV, SUB, MULT \rangle$

Figure 9: 3×3 grid illustrating how different combinations of arithmetic operations (Division (red), Addition (blue), Subtraction (green), Multiplication (purple)) influence the speed of generalization. Each panel compares a specific multi-task training setup against the baseline modular division task (black curve). Again, the vertical dashed lines in each plot indicate the specific step at which each task reaches the generalization threshold (95%), allowing for a direct comparison of grokking timings across different task combinations. The training accuracy is still represented by the dashed orange line, and the x-axis is in the same logarithmic scale.

3.3 Batch Balancing, Optimizer, and Initialization

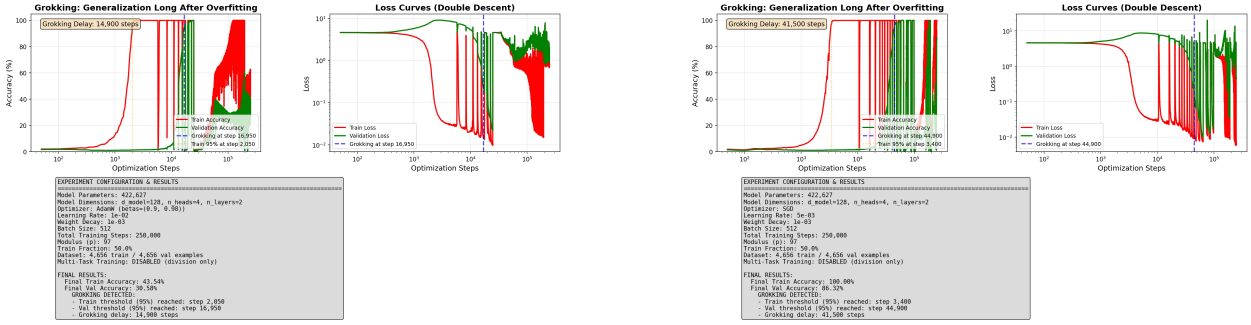
Further experiments isolated the effect of batch composition, optimizer choice, and constrained initialization on grokking. Figure 10 shows that randomized 50/50 multi-task

splits (Division/Addition) fail to grok entirely, stalling at 2.16% validation accuracy, while strictly balanced batches (256 examples per task per step) successfully stabilize training, achieving 53.41% validation accuracy. Figure 11 compares SGD at two learning rates against the AdamW baseline. Figure 12 shows that sparse or small-scale initialization schemes reduce the grokking delay from ∼334,000 steps to ∼1,050 steps.



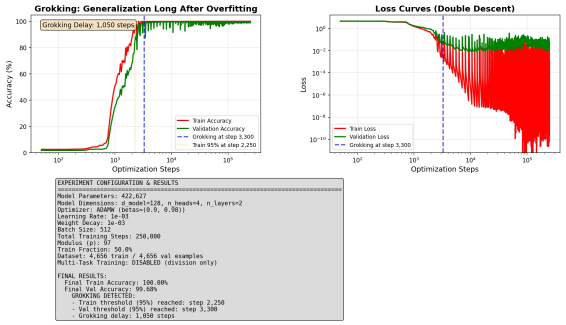
(a) Randomized 50/50 (Fails to Grok) (b) Balanced 50/50 (Successful Grokking)

Figure 10: The necessity of exact batch balancing. Randomized sampling causes catastrophic interference, while strict 256/256 batch balancing rescues generalization.

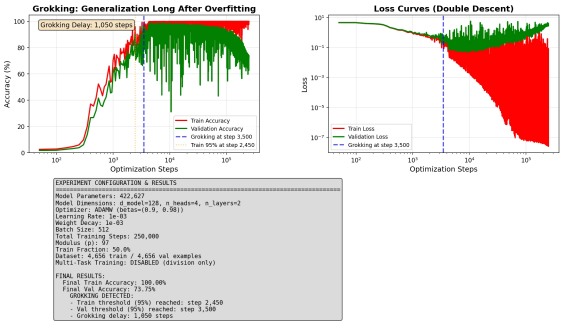


(a) SGD LR 0.01 (14.9k steps, unstable) (b) SGD LR 0.005 (41.5k steps, stable)

Figure 11: The effect of SGD learning rate on grokking speed and stability compared to AdamW.



(a) Sparse Initialization (sparsity=0.9)



(b) Small Initialization (scale=0.01)

Figure 12: Restricted initialization schemes completely eliminate the grokking delay, achieving generalization in merely $\sim 1,050$ steps.

4 Discussion

4.1 Loss Plateau

Our replication confirms the three key mechanisms identified by Gopalani and Hu:

1. **Representation collapse:** Token embeddings cluster during the early phase, reducing the model's ability to differentiate inputs.

2. **Repetition bias:** The model tends to repeat recent tokens, further degrading output quality.

3. **Slow attention map formation:** The attention map remains diffuse initially, limiting early token-level computation.

Effect of Multi-Task Training: Exposure to multiple tasks (MWS, MWP, MWD, PS) accelerates learning and shortens the plateau. The duration of the plateau decreases as more tasks are incorporated, suggesting that the model may leverage common structure across tasks to learn more efficiently. This observation aligns with Kim et al., who show that task diversity can reduce the length of learning plateaus.

Effect of Transfer Learning: Pretraining on a simpler or related task alleviates plateau dynamics. Our observations show that during fine-tuning, the attention weight matrices (**K, Q, V**) remain largely stable, while the corresponding bias terms shift significantly. This indicates that the attention map itself is not strictly constrained, contrary to prior suggestions that attention limits plateau exit.

For example, pretraining on Prefix Sum allows the model to rapidly adapt to Moving Window Sum (MWS), effectively eliminating the plateau. Similarly, pretraining on MWS accelerates learning on Moving Window Product (MWP), even though the target functions differ. These results suggest that while the attention weights are mostly preserved, adaptation occurs primarily through bias terms and potentially other components such as hidden representations, enabling faster task-specific learning.

These results suggest a revised perspective: while attention structures provide inductive

bias for sequence processing and embeddings encode token-level information, the emergence of the plateau may be constrained primarily by the dynamics of hidden representation formation. Multi-task and transfer learning accelerate these dynamics, allowing faster convergence and improved learning efficiency.

4.2 Effects of Task Diversity (Grokking)

In this section, we discuss the impact of training on multiple modular arithmetic tasks simultaneously on the speed at which grokking occurs.

4.2.1 Baseline Reproduction

As seen in Figure 7, in our baseline run consisting of training on only the modular division task for 400,000 steps, we observe that the training accuracy reaches perfect to near-perfect accuracy very early in training, at only 0.1-0.2% training progress. This indicates that the model has successfully memorized the training set. In contrast, we can see that the validation accuracy of the division task remains near chance levels for the majority of training before grokking, rising toward 100% after a period of memorization. In our run, the model reached 95% validation accuracy, or completed grokking, at 83.5% of training progress, or 334,000 out of 400,000 steps. These results are consistent with the findings of Power et al. (2022), who similarly observed a sharp generalization transition occurring long after memorization on modular arithmetic tasks, confirming that our baseline reproduces the original grokking phenomenon.

4.2.2 Multi-Task Training Accelerates Grokking

Now we examine how these grokking behaviors change when additional tasks are introduced. In Figure 8, we can see that grokking occurs significantly faster when all three modular addition, subtraction, and multiplication are trained alongside the division task. Similar to our baseline, the training accuracy reaches near-perfect accuracy almost instantly, at 0.1-0.2% training progress. However, our results clearly show that the validation accuracies for each task reach near-perfect accuracy before the baseline does. More specifically, the multiplication task grokked the fastest, at just 4.3% of training progress (68,250) steps, followed by division at 4.7% (75,750), then addition at 6.5% (104,150 steps), and finally subtraction at 7.8% (124,650 steps). These results show that training on multiple arithmetic tasks simultaneously substantially accelerates grokking across all tasks. We hypothesize that this occurs because of the similar modular properties behind all four arithmetic tasks, encouraging the model to learn more general solutions rather than task-specific ones. Introducing diverse arithmetic tasks to the model promotes regularization toward learning the underlying common structure of modular arithmetic, speeding up generalization. Although this multi-task acceleration effect has not been directly studied in the context of grokking, the study on the loss plateau by Kim et al. (2025) proposes a similar hypothesis as to why

task diversity shortens generalization delays. This is consistent with the broader grokking literature, which notes the emergence of the underlying structures of mathematical objects when training Power et al. (2022).

4.2.3 Effects of Specific Task Combinations

We now look at the results across different task combinations. Looking first at single-task runs for $\langle ADD \rangle$, $\langle SUB \rangle$, and $\langle MULT \rangle$, achieving grokking at around 14.2%, 37.0%, and 15.7% training progress, respectively, which were all significantly faster than the baseline division task that achieved grokking at around 83.5% training progress for the same amount of total steps (400,000). This makes sense as modular division is generally considered a more complex task due to its unique mathematical properties that make it less straightforward to learn. For runs with two tasks, ran for 800,000 steps, we find that grokking actually slows down for both $\langle DIV, ADD \rangle$, and $\langle DIV, SUB \rangle$, with each task failing to generalize more efficiently compared to the baseline. However, the $\langle DIV, MULT \rangle$ run exhibited the fastest grokking across all experiments, with both tasks at just around 0.7% for both tasks compared to the baseline's 41.8%. This may be due to similar mathematical properties between modular division and multiplication that division does not share with addition or subtraction. For combinations of three tasks, we again get varying results. For the $\langle DIV, ADD, MULT \rangle$ run, all three tasks were able to achieve grokking faster than the baseline, but the acceleration was not as drastic, with only a 5.7% gap in grokking occurrence between the addition task and the baseline task. For both the $\langle DIV, ADD, SUB \rangle$ and $\langle DIV, SUB, MULT \rangle$ runs, we observe that two of the arithmetic tasks exhibit grokking faster than the baseline, while the remaining task grokked slower than the baseline, these tasks being division and subtraction, respectively. We again theorize that this is due to the mathematical properties of the tasks involved. For example, the directional asymmetry subtraction introduces may make it more difficult for the model to learn the underlying patterns in the data, causing some tasks to generalize more slowly than they would in isolation or in different task combinations.

4.2.4 Limitations and Future Work

There are some limitations of our work that are worth acknowledging. First, all experiments are conducted with a fixed model size ($d_{model} = 128$, 2 layers, 4 heads) and a single prime modulus ($p = 97$); it is unclear whether the observed acceleration effects would hold for different model sizes or moduli. This holds true for other hyperparameters as well, such as learning rate, weight decay, and training data fraction. Second, our hypothesis that the underlying structure of modular arithmetic tasks drives grokking acceleration is thus far qualitative. We do not provide concrete, interpretable evidence to directly support this claim. That said, these limitations open up several possibilities for future work. A logical extension would be to apply interpretability tools, such as attention visualization or probing classifiers, to examine whether multi-task models learn different internal representations compared to single-task models, which could provide direct evidence for our common struc-

ture hypothesis. Additionally, experiments with varying model capacities, choice of prime p, and other hyperparameters could help establish how our findings generalize beyond the specific setup studied here. Finally, investigating other forms of task diversity beyond modular arithmetic could also reveal whether the acceleration effect we observe is present in other settings or specific to ours.

4.3 Optimizer Trade-offs: SGD vs. AdamW

We evaluated the effect of optimizer noise by switching from AdamW to SGD (see Figure 11). Using SGD with a high learning rate ($LR = 0.01$) induced grokking in just $14,900$ steps (a roughly $7\times$ speedup over the typical AdamW baseline), but the training was highly unstable, resulting in catastrophic forgetting immediately after generalization. Lowering the learning rate to 0.005 stabilized the training. The model grokked after $41,500$ steps, retaining its accuracy. This represents a $\sim 60\%$ reduction in the grokking delay compared to AdamW, demonstrating that the gradient noise inherent in SGD can help the model escape the memorization phase faster. Intuitively, the larger, noisier gradient steps in SGD prevent the optimizer from settling into flat memorization minima and instead push the model toward lower-norm, more generalizable solutions—consistent with the implicit bias arguments of Lyu et al. (2024).

4.4 Network Initialization: Eliminating the Plateau

Our most significant finding concerns network initialization (see Figure 12). We hypothesized that standard initialization provides excess capacity that encourages early memorization.

- **Sparse Initialization:** By initializing the network with mostly zero weights ($sparsity = 0.9$), the grokking delay was reduced to merely $1,050$ steps.
- **Small Initialization:** Similarly, initializing weights with a very small uniform scale ($scale = 0.01$) also resulted in a grokking delay of exactly $1,050$ steps.
- **Low-Rank Initialization:** Restricting the initial rank ratio of weight matrices to 0.5 and 0.3 severely impaired the network, capping validation accuracy at 3.48% and 1.40%, respectively—suggesting that extreme rank constraints remove the capacity needed to represent generalizable solutions at all.

These interventions represent an approximate $100\times$ speedup over the baseline, essentially eliminating the grokking plateau. The contrast between sparse/small initialization (which succeeds) and low-rank initialization (which fails) is instructive: what matters is not reducing the number of parameters, but starting with weights of low magnitude so that the optimizer is not committed to any high-capacity memorization solution from the start.

5 Conclusion

This work investigated two distinct optimization plateaus in Transformer training—the early training-loss plateau and the later grokking phase—and showed that both can be substantially shortened through targeted interventions.

For the early training-loss plateau, we confirmed the three characteristics identified by Gopalani and Hu: representation collapse, repetition bias, and slow attention map organization. Our experiments demonstrate that task diversity through multi-task training on moving-window tasks (MWS, MWP, MWD, PS) accelerates learning and shortens the plateau. Incorporating multiple tasks appears to allow the model to leverage common structure across tasks, facilitating faster development of informative hidden representations. These observations align with the theoretical framework of Kim et al., who argue that task diversity can reduce the duration of learning plateaus by leveraging the common structure between tasks.

Transfer learning further alleviates plateau dynamics. Pretraining on a simpler or related task enables rapid adaptation to a new task, with attention weight matrices (K, Q, V) remaining largely stable while bias terms shift significantly. This indicates that the attention map itself is not strictly constrained, and adaptation occurs primarily through bias adjustments and possibly hidden state modifications. For example, pretraining on Prefix Sum accelerates learning on Moving Window Sum (MWS), while pretraining on MWS shortens the plateau on Moving Window Product (MWP). These findings highlight that embeddings and attention provide the inductive bias for sequence processing, while multi-task and transfer learning mainly support faster emergence of informative hidden representations and bias adjustments.

For the grokking phase, our experiments revealed three complementary acceleration strategies. First, multi-task training on modular arithmetic tasks accelerates grokking across all operations, with the $\langle DIV, MULT \rangle$ combination producing the most extreme speedup—suggesting that structurally related tasks promote shared generalization mechanisms. Second, switching from AdamW to SGD introduces gradient noise that helps escape the memorization minimum, achieving a ∼60% reduction in grokking delay at a stable learning rate. Third, and most dramatically, sparse or small-scale initialization eliminates the grokking delay almost entirely—reducing it from ∼334,000 steps to ∼1,050 steps, a 100× speedup—by preventing the network from ever committing to a high-capacity memorization solution.

Taken together, these findings suggest a unified principle: both forms of optimization stagnation are driven by the model settling into degenerate, high-capacity intermediate states. Interventions that promote representation diversity (multi-task training) or restrict initial capacity (constrained initialization, optimizer noise) effectively prevent or shorten these stalls. Future work should examine whether these findings transfer to larger models, other moduli, and tasks beyond modular arithmetic, and should apply interpretability tools to directly verify the representation-level mechanisms hypothesized here.

6 Contributions

Jet Yue: Focused on the early training-loss plateau. Replicated loss-plateau baselines, verified representation collapse and attention-map behavior. Drafted the early plateau sections of the report.

Samuel Cho: Investigated task diversity and task-specific grokking dynamics. Developed fair-comparison balanced batching. Mapped the independent grokking speeds of all four arithmetic operations over 400k+ training steps. Designed and executed Low-Rank and initial Sparsity sweeps.

Tommy Li: Led experiments on accelerating grokking via optimizers, demonstrating the speed-stability trade-off between AdamW and SGD (accelerating grokking to 41.5k steps). Refined the Sparse and Small initialization methodologies, discovering the specific hyperparameters that reduced the grokking delay by 100x (to 1,050 steps).

All work was guided and mentored by Dr. Tianhao Wang.

References

Gopalani, Pulkit, and Wei Hu. 2025. "What Happens During the Loss Plateau? Understanding Abrupt Learning in Transformers." [Link]

Kim, Jaeyeon, Sehyun Kwon, Joo Young Choi, Jongho Park, Jaewoong Cho, Jason D. Lee, and Ernest K. Ryu. 2025. "Task Diversity Shortens the ICL Plateau." [Link]

Lyu, Kaifeng, Jikai Jin, Zhiyuan Li, Simon S. Du, Jason D. Lee, and Wei Hu. 2024. "Dichotomy of Early and Late Phase Implicit Biases Can Provably Induce Grokking." [Link]

Power, Alethea, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. 2022. "Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets." [Link]