LoRA Memorization Scaling Laws

Anonymous authors
Paper under double-blind review

Abstract

LoRA adapters have become the canonical method for injecting new knowledge and behavior into a pretrained model, yet their physical limits are not yet fully understood. This work measures the capacity of adapters to memorize and generalize by training on synthetic datasets of both random facts and hidden modular rules. On the DataDecide OLMo checkpoints between 20M and 1B parameters, we observe that an adapter stores up to 2.2 bits per projection parameter, with an efficiency that falls with rank and rises with base model quality. Our memorization scaling laws predict the target held-out 1B x r rank within 11% errors. While capacity alone does not decide whether an adapter can generalize to modular rules, it requires a sufficient number of directions to represent facts and can grok faster on a pretrained base checkpoint of higher quality.

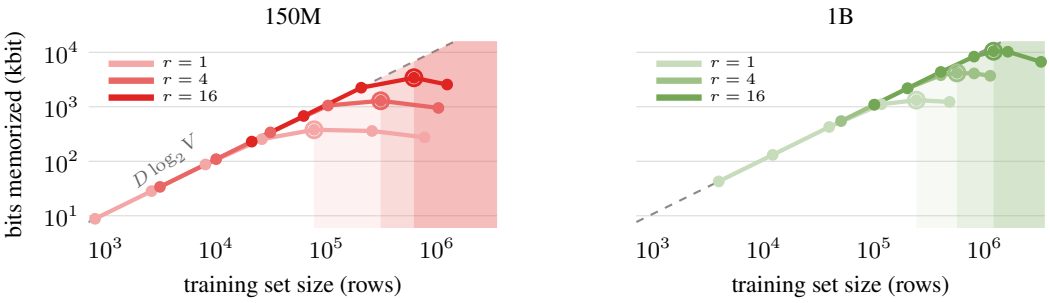


Figure 1: An adapter fills up, then forgets. Bits memorized against training set size on one shared table for the 150M and 1B bases, ranks 1, 4 and 16 from light to dark. Each curve follows the $D \log_2 V$ diagonal until its peak, the capacity (hollow ring ○), and the shaded band past the ring is the forgetting regime. All six bases are in Appendix C.

1 Introduction

While pretraining distills human knowledge into a large-scale model, post-training is a key step for adapting large language models (LLM) to downstream tasks and shaping their behavior. Currently, due to the high costs of compute and data required in pretraining, downstream adaptation is often the only avenue to inject new knowledge into a model, via algorithms like supervised finetuning (SFT) and reinforcement learning (RL).

In (Hu et al., 2022), task-specific weight updates are discovered to be very low-rank, leading to the wide adoption of LoRA adapters. Specifically, for a frozen pretrained weight $W_0 \in \mathbb{R}^{d\times k}$, LoRA reparameterizes the finetuned weights as $W_\Delta = BA$, with trainable matrices $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$, and rank $r \ll \min(d, k)$. Since A and B only cost $r(d+k)$ parameters, instead of dk, it is much more memory- and bandwidth-efficient to train, store, and move around a LoRA adapter compared to full finetuning. For instance, an inference server can maintain many $\{A, B\}$ in memory to swap and batch together payload requests. In the continual learning setting, LoRA weights could also serve as a form of memory (Back et al., 2026).

Despite this flexibility, neural networks are known to have physical limitations. Allen-Zhu & Li (2025) previously find that neural networks can store about 2 bits per parameter from training on

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

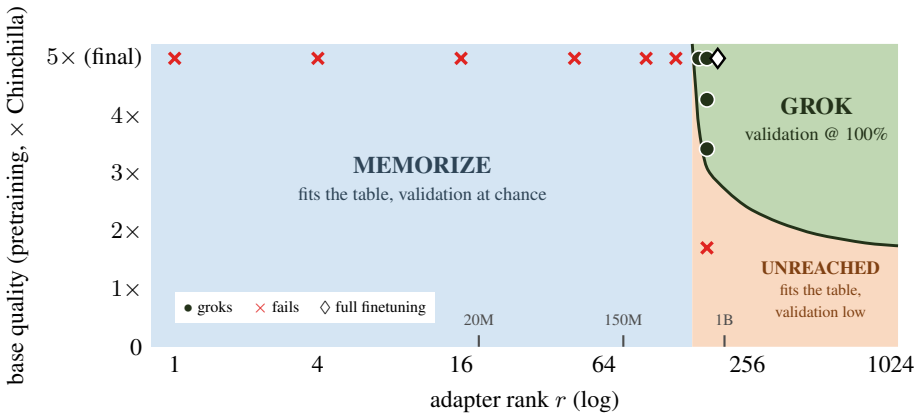


Figure 2: Grokking outcomes at 20M for adapters on $a+b \bmod 97$, by adapter rank and base pretraining, with the boundary drawn as a hypothesis. To the left of it, an adapter memorizes. To the right, it groks only on a base pretrained far enough, and below the curve the rule is representable but unreached. The boundary's height is measured only at 20M, between $1.7\times$ and $3.4\times$ Chinchilla at a 400k-step budget. Ticker marks $r = 0.1\,d$ for three different widths.

synthetic biographies. For LLMs, Morris et al. (2025) estimate LLMs to memorize around 3.6 bits of text per parameter, with grokking occurring once the size of the dataset exceeds capacity. In this work, we study the limitations of LoRA adapters in storing new knowledge acquired entirely via low-rank post-training.

On a synthetic set of both random and rule-based facts, we isolate the storage patterns of language models and LoRA adapters. Leveraging the DataDecide OLMo suite (Magnusson et al., 2025), we measure LoRA capacity as a function of both the number of parameters and base model quality as measured by its held-out loss. We make the following contributions:

- **Capacity peaks, then declines.** A LoRA adapter stores about 2.1 bits per projection parameter at rank 1, falling to about 1.1 at rank 16. Given more rows than it can hold, it loses up to 36% of its stored bits, and exact recall fails first, at about half the peak (Figure 1).
- **MLP layers store most of the bits.** Adapted one group at a time, the MLP projections hold nearly all of an adapter's stored bits, while attention and the LM head hold almost none. There is also a synergy: adapting all of the layers together stores more than the sum of the groups adapted separately, up to three times more at 60M (Appendix D).
- **A capacity scaling law.** Capacity is linear in the adapter's projection parameters p, discounted by a rank efficiency $\eta(r) = 1/(1 + r/r_0)$, and scaled by a base quality factor $g(\ell)$ in the base's C4 loss: $C = k\,p\,\eta(r)\,g(\ell)$ (Equation 6). Fit on bases up to 530M, it predicts a held-out 1B base within 11% on average. Pretraining raises capacity until the base's loss falls below about 5 bits per token (Figure 3).
- **Rank and pretraining decide grokking more than storage capacity.** On $a+b \bmod 97$, an adapter groks only with about 100 to 160 directions, a count that stays fixed as the width grows fourfold, and only on a base pretrained far enough. Even then it needs 100k to 300k steps, against 3k to 4k for full finetuning (Figures 2 and 5).
- **Real tasks need few directions.** PopQA keeps its gain with 27 to 41 directions at every width, 6 to 25 times fewer than the modular rule, and trained real-task adapters use only 1 to 3 directions (Figure 4).

2 Defining capacity

We treat capacity as compression, following Morris et al. (2025). The information a trained model $\hat{\theta}$ holds about a datapoint x is the number of bits the model saves encoding it. This is measured as

$H(x) - H(x \mid \hat{\theta})$, where $H(x)$ is x's own code length and $H(x \mid \hat{\theta})$ is x's code length as encoded by the model. A code built from a model distribution spends $-\log_2 p$ bits on an outcome it assigns probability p, making the bits encoded a negative log-likelihood, $H(x \mid \hat{\theta}) = -\log_2 p(x \mid \hat{\theta})$. Since this quantity conflates two contributions, information specific to x and representation of x in the model, Morris et al. separate the latter by defining the unintended memorization as:

$$\text{mem}_U(x) \;=\; H(x \mid \theta) - H(x \mid \theta, \hat{\theta}), \tag{1}$$

and capacity is the largest total of it a model can hold over any training set, $\max_X \sum_{x \in X} \text{mem}_U(x)$.

Let a training set size D be $X = \{(q_i, a_i)\}_{i=1}^{D}$ containing pairs of `(query, answer)`. We can remove generalization from Equation 1 if we were to draw a_i uniformly and independently from a set V candidate words. We construct such a dataset in §4. The reference model θ then becomes a uniform distribution that encodes every answer by the same $H(a_i \mid \theta) = \log_2 V$ bits. Accordingly, the trained adapter spends $H(a_i \mid \hat{\theta}) = \text{NLL}_2(a_i \mid q_i)$ to encode a_i. Equation 1 reduces to the difference of the two. Summing that over the training set gives us:

$$\text{bits}(D) \;=\; \sum_{i=1}^{D} \text{mem}_U(a_i) \;=\; \sum_{i=1}^{D} \Big(\log_2 V - \text{NLL}_2(a_i \mid q_i) \Big), \tag{2}$$

as the bits the adapter stores out of the $D \log_2 V$ amount of data. Different from the precedent work, we do not cap the floor of a datapoint's contribution at zero. If a fact cannot be recalled, the adapter's NLL on x exceeds $\log_2 V$ and counts towards the sum as *negative* bits, so every curve in Figure 1 bends back down past its peak, marked by a hollow ring $\circ$.

In this setup, below some dataset size, we can reasonably expect the adapter to store every row and the total bits to steadily climb the diagonal $D \log_2 V$. Once storage saturates, bits might decrease as new rows overwrite old ones. This forms the peak of our capacity,

$$C \;=\; \max_{D} \text{bits}(D), \qquad D_c = C / \log_2 V, \tag{3}$$

with D_c the same number in rows, and we report C/p over the adapter's p parameters on the block projections. The LM head adapter stores nothing on its own (Appendix D) and is left out of p.

Grokking. Beyond storing raw knowledge, neural networks are powerful because of their ability to generalize beyond seen data, giving them emergent capabilities. Let the answers now follow a rule, $a_i = f(q_i)$ for a fixed f such as $a + b \bmod m$, we can ask whether the model can answer queries it has not seen. A model can fit the train rows in two ways, by storing them or by finding the rule, and only the second can solve held-out rows. Grokking is the delayed transition from the first to the second (Power et al., 2022), in which training accuracy reaches 100% early while held-out accuracy stays at chance for many steps before making the jump. We draw three hypotheses that decide whether the transition happens. Morris et al. observe that grokking begins when the table exceeds capacity, $D \log_2 V > C$, forcing the only path of compressing the rules in order to fit more facts. Under the representability hypothesis, the rule occupies some number of directions r^*, and an adapter of rank $r < r^* < d$ (layer width) cannot physically represent. Under the reachability hypothesis, rank is necessary but not sufficient, and an adapter finds the rule only on a base of sufficient quality. We read the outcome from the held-out loss against the uniform value $\log_2 m$. Below that value, the model has learned parts of the rule; above it, the model has failed to grok. Figure 2 demonstrates the three regimes in our experiments.

3 Memorization scaling laws

A scaling law establishes a formula fit that predicts an outcome from quantities known before training begins. It helps us tell how much memorization a LoRA adapter can store given informative axes in post-training fact injection. Here, outcome is the capacity C defined in Equation 3, the number of bits an adapter can hold. Naturally, the quantities known in advance are the base model width d and depth L, the adapter rank r, and, where available, the base model held-out loss ℓ on natural language text (in bits per token) at the checkpoint the adapter is used. The first three set $p = 20\, dL\, r$, the number of adapter parameters on the block projections, where our ablations show store most of the facts (Appendix D) and ℓ measures the base quality.

Table 1: Rows from the two synthetic tables, answers in bold. Every row is the text `x % y = z` over single-token words; in the rule table each word stands for a number.

random table

Thunder % pirate = **coconut**		

rule table, $a+b \bmod 97$ Kepler 45, Fern 13, heir 58, Shakespeare 37, genius 10

Kepler % Fern = **heir**	trained, $45+13 = 58$	
heir % Kepler = **vas**	trained, $58+45 = 103$, $103-97 = 6$	
Shakespeare % genius = **Prophet**	trained, $37+10 = 47$	
$\Rightarrow$ Fern % genius = **Batt**	held out, $13+10 = 23$	

The constants of the law are fit on the bases from 20M to 530M, with DataDecide 1B checkpoints as the held-out target. Each factor below follows from one observation in Figure 3. We build our scaling law step by step.

Store density. One simple intuition for the law is that more LoRA parameters can store more facts, and that capacity grows proportionally to the number of adapter parameters (doubling p also doubles the storage capacity). If that holds, bits per parameter, C/p, should be the same for every single base model size. At a fixed rank, we indeed observe this pattern. For example, at rank 1, every base from 60M to 1B stores about 2.05 bits per parameter, even when the 1B has 5x the number of parameters of the 60M (Figure 3a). For comparison, a full model stores about 3.6 bits per parameter on the same measure (Morris et al., 2025). Therefore, within a rank,

$$C \;=\; k \cdot p, \tag{4}$$

where k is the store density, the bits one projection parameter stores on a fully trained base, before considering a rank discount in the next paragraph. We count p over the attention and MLP projections, leaving the LM head adapter (a linear map on the base final hidden state) from this count. Adapted one group at a time, we observe the MLP pair holds nearly all of the bits while the head store close to none (Figure 7, Appendix D), consistent with work locating facts in the feed-forward layers (Geva et al., 2021; Meng et al., 2022). As a linear map, the head adapter can fit at most about d random rows, a few thousand, against the hundreds of thousands stored in the block projections. Since its size tracks the vocabulary rather than the width (the head is 45% of the adapter at 20M but 7% at 1B), counting it would make the density appear to grow with base size, inhibiting an accurate fit.

Rank efficiency. Store density does not hold constant across ranks. At 150M, rank 16 has sixteen times the parameters of rank 1 (2.95M vs. 184k), but its peak stores nine times the bits (3411 against 379 kbit). Bits per parameter fall from about 2.1 at rank 1 to 1.2 at rank 16, and every base falls along the same curve of diminishing returns (Figure 3a).

In fact, on log-log axes the curve flattens at low rank rather than following a power law throughout. This tells us that each added direction stores less than the rank that comes before (with the size of the random dataset changing accordingly). This suggests we should fit the density as two factors: how much a parameter stores when rank is small, and how much rank efficiency regresses when rank increases. Beyond store density k from Equation 4, we additionally fit the shape $\eta(r)$ as the share of storage a parameter still holds at rank r,

$$C \;=\; k \cdot p \cdot \eta(r), \qquad \eta(r) = \frac{1}{1 + r/r_0}, \tag{5}$$

This is the simplest curve that is flat at small r and falls at $1/r$ at large r, containing a single constant. Setting $r = r_0$ makes the denominator 2, so r_0 is the rank at which a parameter stores half of k. Note that $\eta \in [0, 1]$ and η equals 1 only as $r \to 0$. This means k is the density in that limit and $\eta(1) = 0.94$, meaning rank 1 keeps 94% of density. At 150M and rank 16, $\eta(16) = 0.48$, so each parameter stores about half of k.

For fully trained bases of DataDecide, Equation 5 is the full law. Fitting it to the 22 final checkpoints gives $k = 2.17$ bits per parameter and $r_0 = 14.8$ ($R^2 = 0.996$), with an error of 7% for predicting the held-out final 1B checkpoint.

Base quality factor. The quality of the base model, such as pretraining checkpoints recorded at various FLOPs, adds a third axis to adapter capacity. At the same rank, we observe that an adapter

trained on an earlier base checkpoint stores fewer bits, with the base held-out loss holding predictive power (Figure 3b). At 20M, the final checkpoint (5.8 bits per token on C4 validation set) holds 0.76 of the capacity $k\,p\,\eta(r)$, and the first checkpoint (7.0 bits per token) holds about half. From 60M and above, the final checkpoints sit at 0.87 to 0.97.

Adapters store more on overtrained base models, but we see similar diminishing returns, as with rank density. The earliest checkpoints give the smallest capacity. However, once a base;s C4 loss drops below about 5 bits per token, more pretraining makes little difference, and the final checkpoints (5x Chinchilla) from 60M and up are already in that range. We model this with:

$$C \;=\; k \cdot p \cdot \eta(r) \cdot g(\ell), \qquad g(\ell) = \frac{1}{1 + (\ell/\ell_0)^m}. \tag{6}$$

Setting $\ell = \ell_0$ makes the denominator 2, so ℓ_0 is the loss at which capacity halves. $g(\ell)$ does not exceed 1 and equals 1 only at $\ell = 0$, so the fit decides where the fall occurs via ℓ_0 and the sharpness of the capcity fall via is h m. When $m = 1$, we have the same shape as η. The fitted $m = 5.8$ makes the drop far sharper than the fall in rank. g goes from 0.9 to 0.1 over a factor of about 2 in loss, from 4.9 to 10.4 bits per token, where η needs a factor of 81 in rank.

We also tried a power law and an exponential in ℓ. Neither flattens at low loss, and both double the held-out prediction error. A logistic in ℓ fits as well as Equation 6 (Table 5). Below about 5 bits per token, g is within 10% of 1.

Fitting and testing the law. We fit Equation 6 in log space to all 76 points, 22 final checkpoints and 54 pretraining checkpoints. This gives us the constants $k = 2.46$ bits per parameter, $r_0 = 11.3$, $\ell_0 = 7.1$ bits per token and $m = 5.8$, with $R^2 = 0.994$ and a mean error of 8% (Figure 3c). k comes out higher than in Equation 5 because even the final checkpoints have g below 1. To test the law, we refit it on 20M to 530M and predict the five 1B points. The mean error is 11%. All five predictions are slightly high, by 2 to 4% at rank 16 and by 12 to 17% at ranks 1 and 4, so the law overpredicts 1B slightly and consistently (Figure 3d). Base size N, pretraining tokens T and the base's loss on the task itself do not improve the fit once p and ℓ are in (Table 5).

4 Experimental setup

In this section, we discuss the datasets and the model ladder we use to assess model memorization and grokking capacity.

Synthetic (random) tables. The capacity ladders train on one shared random operation table, $\mathrm{rect}(A, B, V)$, rows of the form $a \circ b = c$ where a and b range over blocks of A and B symbols and c is drawn i.i.d. uniformly from a block of V answer symbols. V is therefore the number of candidate answers, and the $\log_2 V$ of Equation 2 is the content of one row. The A, B, V symbol blocks are disjoint, so no symbol appears on both sides of a row and no row says anything about another. Every answer has to be stored. We take each symbol as a single token sampled from the base's own vocabulary. This allows us to isolate memorization patterns from the fact that model might have to learn new token embeddings. Our default dimension is $\mathrm{rect}(3000, 3000, 2000)$, so $V = 2000$ and each row carries 11 bits, giving us nine million cells and 98.7 Mbit of text content. This scale provides enough headroom to evaluate capacity at the 1B scale and big ranks, where operation format puts capacity and grokking in the same units. Table 1 shows an example of the data.

Rule-based tables. The grokking experiments use modular addition, $a + b = c \pmod m$ with $m \in \{97, 150, 200, 250\}$ and half the table held out, mimicking the setting of Power et al. (2022). These rows read exactly like the random table. The difference is that each symbol now stands for a number and a single rule fixes every answer, so an adapter that has learned the rule can complete a row it never saw, as the last line of Table 1 shows.

Real tasks. Beyond synthetic tables, we use two real datasets to comparatively measure the required rank (Appendix E). PopQA (Mallen et al., 2023) is long-tail factual question answering, where each query names an entity and the answer is a related fact. Additionally, the multilingual Aya collection (Singh et al., 2024) is another natural test bed given that our base checkpoints are pretrained almost exclusively on English. While both of these tasks do not demand as many representational ranks as

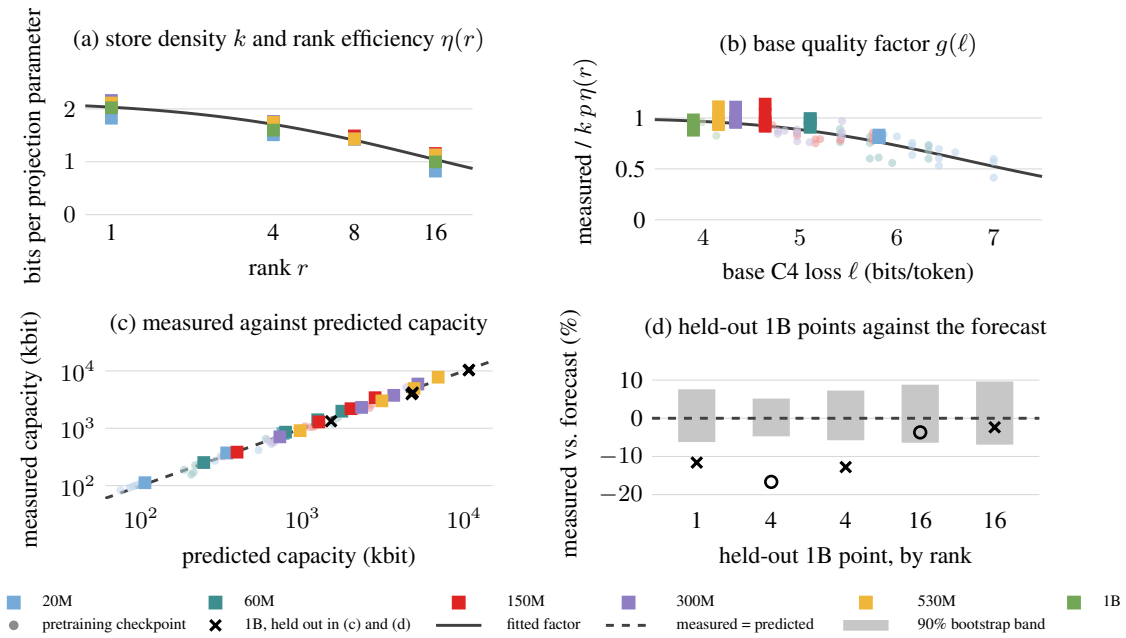


Figure 3: The capacity law. (a) Bits per projection parameter against rank at the final checkpoints, with the fitted $k\,\eta(r)$. (b) Measured capacity divided by $k\,p\,\eta(r)$ against the base's C4 loss for all 76 points, with the fitted $g(\ell)$. (c) Measured against predicted capacity, dashed line at equality. The law is fit on 20M to 530M, so the 1B points ($\times$) are extrapolations. (d) Each held-out 1B point relative to its forecast, with the 90% band of 400 bootstrap refits; hollow marks are pretraining checkpoints.

rule-based tables, they theoretically still inject almost entirely new knowledge via post-training. This condition makes them desirable for our study.

Base models. Every base is a DataDecide OLMo checkpoint (Magnusson et al., 2025; Groeneveld et al., 2024) on the `dolma1.7` data ablation recipe, six sizes from 20M to 1B spanning d_{model} 192-2048. Each of these base comes with intermediate checkpoints on a step grid with a final checkpoint at $5\times$ Chinchilla-optimal. DataDecide allows us to draw a connection to pretraining, and for each checkpoint we take the base's C4 validation loss from its published evaluations as the quality variable of the capacity law. Throughout the work, we treat N as the non-embedding parameter count and keep the base at `bfloat16`.

Applying adapters. LoRA is attached to all four projection groups of every block, including the fused QKV and the LM head. The embeddings and layernorms remain frozen.

Training. Every run uses answer-token loss. Capacity cells train for 200 epochs at batch 512 and run a ladder of dataset sizes placed at multiples of the cell's own predicted capacity, from $0.03\times$ to $30\times$. Rule-learning cells use the optimizer of Power et al. (2022) verbatim, run up to 400k steps and evaluate every 500, and full finetuning with frozen embeddings and layernorms runs at every cell as a control. We list all hyperparameters in Appendix A.

5 Discussion

5.1 Capacity is a peak

Every cell taken far enough past its capacity turns over and loses a portion of its stored bits (Figure 1). For instance, the 150M rank-16 adapter stores the most at 3.4 Mbit when trained on 618,000 rows. At twice as many rows, it stores only 2.6 Mbit. At the largest sizes in our results, the cells had lost

(a) required rank of a full-finetuning update

		$a+b \bmod 150$		PopQA		
base	d	r^*	r^*/d	r^*	r^*/d	gap
20M	192	173	0.90	29	0.15	6×
60M	384	307	0.80	31	0.08	10×
150M	768	461	0.60	31	0.04	15×
300M	1024	563	0.55	41	0.04	14×
530M	1344	672	0.50	27	0.02	25×
1B	2048	614	0.30	41	0.02	15×

(b) spectrum of a trained rank-16 adapter

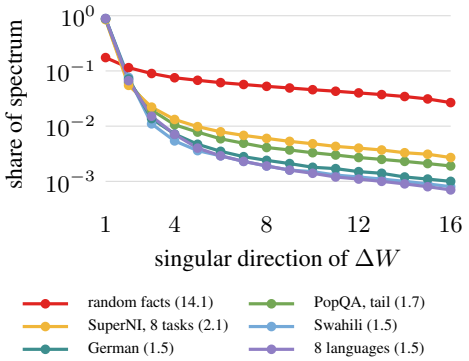


Figure 4: (a) The smallest rank at which a truncated full-finetuning update still solves the task, for the modular rule and for PopQA, by base width. (b) Singular-value spectra of trained rank-16 updates, averaged over the 65 adapted matrices, with effective ranks in parentheses.

up to 36% of their peak. Since each training runs for a fixed 200 epochs (each sample repeated 200 times), the bit loss comes from overwritten facts inside LoRA, rather than any artifacts of data or training configurations.

5.2 Capacity does not decide grokking

Under the capacity hypothesis, generalization should begin once the table no longer fits. We observe no such behavior. Across three capacity regimes at (20M, rank 1), capacity lines D/D_c from 0.23-1.85, six base sizes and five pretraining checkpoints, every LoRA cell sits at or below 2.1% against chance. Meanwhile, full finetuning reaches 99.4 to 100% on 35 of 37 experiments on the same tables. The held-out loss agrees. It falls from 19.7 bits to 7.0 and stops within 0.06-0.08 bits of uniform at every size and rank, meaning none of the rules were ever learned. These adapters all sit below the rank a rule requires (§5.4). Above that rank, it is possible for adapters to grok.

We tried several ways in search for grokking behavior, but none were able to make validation accuracy better than chance. This includes making the embeddings and layernorm trainable (1.3% and 1.7% validation acc), and training for longer from 100k to 300k steps.

5.3 Real tasks need few directions

We choose tasks where knowledge injection theoretically separates from the knowledge of the base model, such as long-tail PopQA facts and Aya languages (Swahili, German) that the model has not seen, being pretrained almost exclusively on English. Despite this, the chosen real-world tasks require far less rank than our synthetic rules. Specifically, we take the top singular directions of a full-finetuning update by truncated singular value decomposition (SVD) and count the number of directions needed to keep 90% of its held-out loss improvement. At 1B, PopQA requires 41 directions while the modular rule dataset requires 614 (Figure 4a, Appendix E). Training adapters directly agrees: a 16× change in rank moves PopQA's held-out gain by less than 7%. Raw storing capacity is not the culprit since Equation 6 shows that 100,000 facts of 11 bits require only a rank around 3-4 for the 150M base.

We can also ask how many directions a trained adapter actually uses. For each rank-16 adapter we take the singular values of its update, normalize them to sum to one, and compute the exponential of their entropy. This effective rank is 16 when all directions carry equal weight, and 1 when a single direction carries all of it. Adapters for multilingual injection, multitask instruction following (Wang et al., 2022) and long-tail factual QA have an effective rank of 1 to 3, while random-fact adapters reach 14.1 of 16 (Figure 4b). In other words, when an adapter is compressed to rank 1, real-world tasks can keep nearly all of their gains, while truncated random-fact adapters lose stored bits with every direction removed.

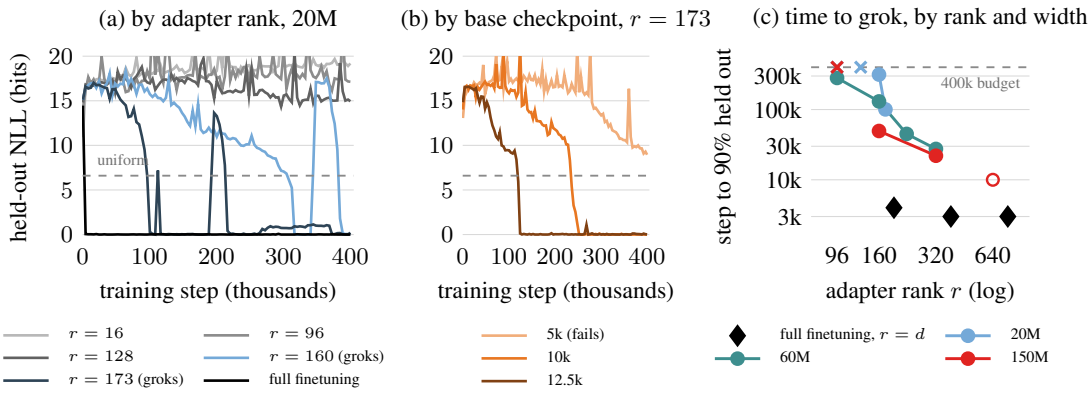


Figure 5: Held-out NLL on $a+b \bmod 97$; below the dashed uniform line the adapter has the rule. (a) By adapter rank on the fully pretrained 20M base, with full finetuning in black. (b) By base checkpoint at $r = 173$. (c) Steps to reach 90% held-out accuracy against rank for 20M, 60M and 150M, with full finetuning as diamonds; the hollow marker is $r = 640$ at 150M, which groks at 10k steps and later collapses.

5.4 Grokking requires a minimum rank

An adapter can only learn a rule if a sufficient rank is able to represent it. At 20M on $a+b \bmod 97$, adapters of rank 16 through 128 memorize their training cells but never generalize, while ranks 160 and 173 grok (Figure 5a). At 20M, 160 directions is 83% of the width, so the rule could need most of the model. Wider bases show it does not. If the requirement were a fixed fraction of the width, a 60M base would need about 320 directions and a 150M base about 640. Instead, every rank from 96 up groks at 60M, and at 150M rank 96 fails while 160 and 320 grok. We conclude that our rule set needs roughly 100-160 number directions, regardless of model width. A wider base therefore makes grokking easier, because the same count is a smaller share of its rank budget. In practice, LoRA adapters often use a rank near a tenth of the width, which only reaches 100 to 160 directions for bases around 1B.

Truncated SVD of full finetuning provides a higher count of directions at 173 at 20M, before rising to 614 at 1B (Appendix E). This is an upper bound. Full finetuning is free to spread its solution over the whole width , so its truncation keeps more directions than a low-rank adapter often needs.

5.5 Base quality induces faster grokking

Enough rank is not sufficient. At 20M and rank 173, the same adapter on the same table fails on a base pretrained to 34% of its run and groks on bases from 69% on (Figure 2, Figure 5b). However, even when both conditions hold, adapters still grok far more slowly than full finetuning, in 100k to 300k steps against 3k to 4k steps (Figure 5c), with more rank or more width shortens the delay. The largest rank we run, $r = 640$ at 150M, actually groks and later loses the rule, which speaks to optimizer instability that can occur for this task. We describe this in Appendix G.

Recipe and budget. All rule-learning runs use one seed, a learning rate of 10^{-3}, weight decay 1, $\alpha = r$ and at most 400k steps. Near both thresholds the delay grows steeply, and a few runs below them were still improving when training stops. Therefore, we acknowledge that the thresholds partially reflect the budget. Grokking is known to depend on weight decay and learning rate (Power et al., 2022; Liu et al., 2023; Nanda et al., 2023), and we have not measured how far either threshold moves under a different recipe.

6 Related work

Morris et al. (2025) define capacity in bits and measure LLMs to store near 3.6 per parameter, with generalization beginning once the dataset exceeds capacity. This framing inspires our investigation

into LoRA adapters, but we depart on the reference model (§2); Allen-Zhu & Li (2025) reach about 2 bits per parameter by an independent route.

On the adapter side, LoRA is Hu et al. (2022), and the observation that finetuning has a tiny intrinsic dimension goes back to Aghajanyan et al. (2021). Biderman et al. (2024) find LoRA learns less than full finetuning on knowledge-heavy tasks, which our spectrum reading places in the memorization regime, and Zeng & Lee (2024) characterize what low rank can express, but neither gives a stored-information figure. Schulman & Thinking Machines Lab (2025) map the regime where LoRA matches full finetuning. Zhang et al. (2024) fit power laws for finetuning loss in data, model and adapter size and find adapter parameters a weak lever, which our rank efficiency puts in bits, and Zhang et al. (2025) track a mutual-information metric between base and adapter hidden states across base size, rank and prompt length, positing a sum of power laws without fitting one or measuring stored information. Busbridge et al. (2025)'s capacity gap, a student capped by its teacher, is the shape our base quality factor takes.

The grokking tasks and optimizer come from Power et al. (2022), though our regime is not theirs, with delay factors of 3.1 to 20 against their roughly 1000 on bases that are pretrained rather than initialized from scratch. Nanda et al. (2023) show structure forming during the flat plateau, reminding us to have a long patience for flat validation before model could realize grokking. We leverage DataDecide (Magnusson et al., 2025) OLMo models (Groeneveld et al., 2024) in this work.

7 Conclusion

Our work studies the capacity of a LoRA adapter to store and grok. We find that LoRA adapters can generalize given an adequate condition of both rank dimension and base model quality, although this regime is highly sensitive. In memorization bits, adapters reach for peak rather than maintaining a ceiling, and follow a law that is linear in the adapter projection parameters, discounted by rank, and scaled by a quality factor in the base held-out loss. Pretraining matters until the base C4 loss reaches about 5 bits per token, and beyond that a better base is only a bigger store per rank. Whether a task fits in low rank is decided by the structure of the task, and that structure sets a count of directions required for an effective post-training setup. In practice, real tasks settle into very few directions and can survive aggressive truncation compared to random algebraic rules, legitimizing low-rank reparameterization of full finetuning. Instead of cost-saving measures that involve reducing LoRA ranks, the budget should instead be spent on the data and on improving base model quality. Alternatively, one can explore architectural decisions for post-training, such as dedicated memory layers for storing facts (Lample et al., 2019; Berges et al., 2024) or adapter designs that prioritizes depth over width for potentially better grokking.

AI use statement

We use generative AI assistant (Claude from Anthropic) throughout this work for drafting and editing texts, Appendix writing and organization, and writing code to create figures. The research questions, the experimental design, the training runs and the interpretation of the results are done by the authors. All AI-assisted text, code and numbers were reviewed by the author, and every reported number is computed by the released analysis scripts from the run logs. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Ethics statement

All experiments train adapters on synthetic tables of random and rule-based facts and on public datasets (PopQA, Super-NaturalInstructions and the Aya collection) with publicly released base checkpoints (DataDecide). No human subjects, private data or personally identifiable information are involved.

Reproducibility statement

Appendix A lists the base models, the data tables and every training hyperparameter of the two measurement families, and Appendix B the loss and the capacity computation. The rows of Table 1 regenerate from the released data generator with the run's seed. The capacity cells, grokking runs,

truncation results and fit scripts, including the bootstrap behind Figure 3d, are in the supplementary material, together with the scripts that regenerate the plot data behind every figure.

REFERENCES

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, 2021.

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.3, knowledge capacity scaling laws. In *International Conference on Learning Representations*, 2025.

Seungju Back, Dongwoo Lee, Naun Kang, Taehee Lee, S. K. Hong, Youngjune Gwon, and Sungjin Ahn. Understanding LoRA as knowledge memory: An empirical analysis. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. arXiv:2603.01097.

Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen-tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale. *arXiv preprint arXiv:2412.09764*, 2024.

Dan Biderman, Jacob Portes, Jose Juan Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, Cody Blakeney, and John P. Cunningham. LoRA learns less and forgets less. *Transactions on Machine Learning Research*, 2024.

Dan Busbridge, Amitis Shidani, Floris Weers, Jason Ramapuram, Etai Littwin, and Russ Webb. Distillation scaling laws. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. OLMo: Accelerating the science of language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Ziming Liu, Eric J. Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *International Conference on Learning Representations*, 2023.

Ian Magnusson, Tai Nguyen, Ben Bogin, David Heineman, Jena D. Hwang, Luca Soldaini, Akshita Bhagia, Jiacheng Liu, Dirk Groeneveld, Oyvind Tafjord, Noah A. Smith, Pang Wei Koh, and Jesse Dodge. DataDecide: How to predict best pretraining data with small experiments. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

John X. Morris, Chawin Sitawarin, Chuan Guo, Narine Kokhlikyan, G. Edward Suh, Alexander M. Rush, Kamalika Chaudhuri, and Saeed Mahloujifar. How much do language models memorize? *arXiv preprint arXiv:2505.24832*, 2025.

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.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

John Schulman and Thinking Machines Lab. LoRA without regret. `https://thinkingmachines.ai/blog/lora/`, 2025. Thinking Machines Lab blog.

Shivalika Singh, Freddie Vargus, Daniel D'souza, Börje F. Karlsson, Abinaya Mahendiran, Wei-Yin Ko, Herumb Shandilya, Jay Patel, Deividas Mataciunas, Laura O'Mahony, et al. Aya dataset: An open-access collection for multilingual instruction tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

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.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. Super-NaturalInstructions: Generalization via declarative instructions on 1600+ NLP tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022.

Can Yaras, Peng Wang, Laura Balzano, and Qing Qu. Compressible dynamics in deep overparameterized low-rank learning & adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Yuchen Zeng and Kangwook Lee. The expressive power of low-rank adaptation. In *International Conference on Learning Representations*, 2024.

Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. When scaling meets LLM finetuning: The effect of data, model and finetuning method. In *International Conference on Learning Representations*, 2024.

Jing Zhang, Hui Gao, Peng Zhang, Shuzhen Sun, Chang Yang, and Yuexian Hou. The scaling law for LoRA base on mutual information upper bound. *arXiv preprint arXiv:2501.03152*, 2025.

Table 2: The base-model ladder.

Base	d_{model}	Layers	Non-embed params ($16d^2L$)	Vocab	Pretraining tokens	Precision
20M	192	16	9,437,184	50,304	1.9 B	bf16
60M	384	16	37,748,736	50,304	5.7 B	bf16
150M	768	12	113,246,208	50,304	15.0 B	bf16
300M	1024	16	268,435,456	50,304	30.0 B	bf16
530M	1344	16	462,422,016	50,304	53.0 B	bf16
1B	2048	16	1,073,741,824	50,304	100.0 B	bf16

Table 3: Hyperparameters for the two measurement families.

Setting	Capacity measurement	Rule learning
Table	$\mathrm{rect}(3000, 3000, 2000)$, random	$a{+}b \bmod m$
LoRA ranks	1, 4, 8, 16	swept to locate r^* (1 to 640), $r = 16$ across small m
LoRA α	1.0 $(\alpha/r = 1/r)$	r $(\alpha/r = 1)$
Target modules	fused QKV, attention out, MLP up/gate, MLP down, LM head	
Precision	bf16 base, fp32 adapter	
Sequence	one equation per row, query $\approx$ 6 tokens, 1 answer token scored	
Loss	answer token only	
Batch	512 rows	512 equations
Optimizer	AdamW, betas (0.9, 0.95)	AdamW, betas (0.9, 0.98)
Learning rate	3×10^{-3}	10^{-3}
Weight decay	0.1	1.0
LR schedule	cosine to lr/10	constant, 10-step warmup
Budget	200 epochs	up to 400k steps, eval every 500
Grad clip	1.0	1.0
Symbols (V)	2000 values	m (modulus)
Frozen	entire base; the full-FT control instead freezes only embeddings and layernorms	
Seed	42	42

A Base models and hyperparameters

Bases are the DataDecide `dolma1_7` OLMo ladder with the GPT-NeoX 50k tokenizer (Table 2), loaded at the final pretraining checkpoint unless a token-axis sweep names an earlier one. Non-embedding parameters are the four projections per block, $16d^2L$, and bases load in bf16 with adapter parameters in fp32. Table 3 gives the recipe for both measurement families.

B Loss computation

Measured capacity depends strongly on which tokens the training loss covers. Table 4 compares two losses at 150M and rank 1, at matched dataset sizes, on fact tables with 20 or 800 relations. With the loss on the whole sequence, the adapter stores 0.44 bits per parameter with 20 relations and 0.14 with 800. With the loss on the answer token only, it stores 1.52 and 1.57.

Table 4: Bits per adapter parameter at 150M and rank 1, at matched dataset sizes, for two training losses and two relation counts.

training loss	definition	20 relations	800 relations	ratio
whole sequence	$-\sum_{t=1}^{T} \log p(x_t \mid x_{<t})$	0.444	0.137	3.24×
answer token	$-\log p(a \mid q)$	1.521	1.574	0.97×
ratio		3.43×	11.5×	

Each row carries $\log_2 4000 \approx 12$ bits in its answer. Under sequence loss the model is also trained to predict the entity, another 12 bits, and the relation, 4.3 or 9.6 bits, so most of the gradient goes to tokens the capacity metric never reads, and more relations take a larger share. With answer loss,

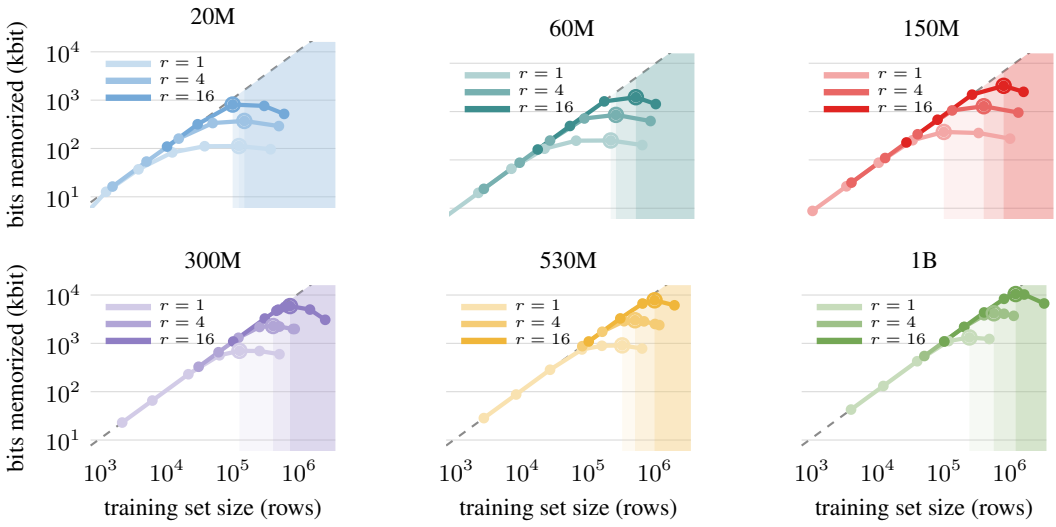


Figure 6: Capacity ladders for all six bases, in the style of Figure 1: bits memorized against training set size, ranks 1, 4 and 16 from light to dark, hollow rings at each rank's capacity, and the shaded band past each ring the forgetting regime.

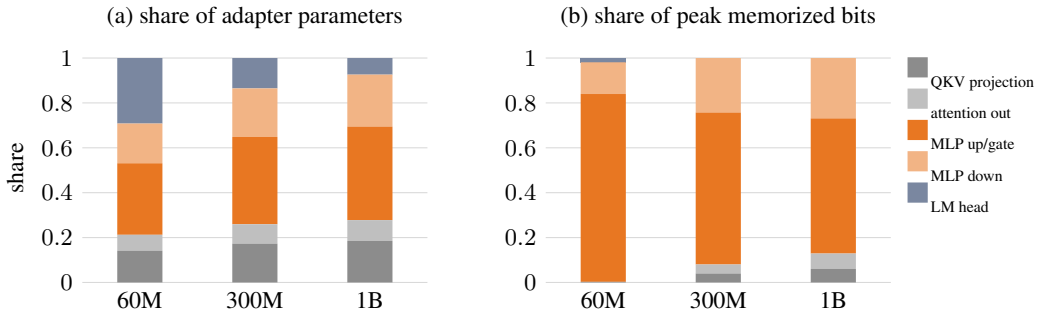


Figure 7: Share of adapter parameters (a) and of peak memorized bits (b) by matrix group, each group adapted alone at rank 16, for the MLP pair, attention and the LM head.

relation count changes capacity by only 3%. Every capacity number in this paper uses answer loss for both training and measurement.

C Capacity curves for every base

Figure 6 shows the capacity curves of Figure 1 for all six bases. At every width, each curve follows the $D \log_2 V$ diagonal, peaks at its capacity, and then declines.

D Where the bits live, by matrix group

We adapt one matrix group at a time at rank 16 and run each through the capacity protocol (Figure 7). The MLP pair holds nearly all of the bits, and attention and the LM head store close to none on their own. The groups do not add. Adapted one at a time they sum to 33%, 69% and 85% of the full adapter's capacity at 60M, 300M and 1B, so attention stores nothing alone but lets the MLP store more, most at small widths.

Algorithm 1 The required rank r^* of a task, from one full finetuning run

Require: base W_0, task with held-out metric $\mathcal{A}$, rank grid $\mathcal{R}$

$\{\Delta W_t\} \leftarrow \textsc{FullFinetune}(W_0)$ — ΔW = LoRA set at max rank

$\Delta W^\star \leftarrow \Delta W_{t^\star},\ t^\star = \arg\max_t \mathcal{A}(W_0 + \Delta W_t)$ — a run can lose the rule late; take its best held-out step

assert $\mathcal{A}(W_0 + \Delta W^\star) \approx \mathcal{A}_{\text{run}}$ — full-rank reinsert must reproduce the run

for $r \in \mathcal{R}$ **do**

  $\mathcal{A}_r \leftarrow \mathcal{A}\big(W_0 + \mathrm{SVD}_r(\Delta W^\star)\big)$ — Eckart–Young: best rank-r solution, no retraining

end for

return $r^* = \min\{r : \mathcal{A}_r \geq \frac{1}{2}\}$ — keep smallest r

E Finding the required rank

An adapter can fail to learn a task for two reasons. Its rank may be too small to represent any solution, or training may not reach a solution it could represent. Truncation separates the two. We fully finetune the base once, take the update $\Delta W^\star$ on the matrices LoRA would adapt, and replace it with its truncated SVD at each rank r (Algorithm 1). By the Eckart–Young–Mirsky theorem this is the best rank-r approximation of $\Delta W^\star$, so if the truncated update still solves the task, a rank-r adapter can represent a solution. The required rank r^* is the smallest such r. For rules, solving means keeping at least half of full finetuning's held-out accuracy. For real tasks, whose answers span several tokens, it means keeping 90% of the held-out loss improvement. A run can find a rule and lose it later, so we take the update at the best held-out step, and we check that reinserting the untruncated update reproduces the run's accuracy.

F Spectral analyses of the trained updates

Each rank-16 update decomposes by SVD as $\Delta W = \sum_i \sigma_i u_i v_i^\top$. We normalize its sixteen singular values to sum to one, average the normalized spectra over the 65 adapted matrices, and report the effective rank $\exp(-\sum_i p_i \log p_i)$. The tasks are PopQA (Mallen et al., 2023), eight Super-NaturalInstructions tasks (Wang et al., 2022), and Swahili, German and eight-language injection from the Aya collection (Singh et al., 2024). Real-task adapters have an effective rank of 1.5 to 2.1, and random-fact adapters 14.1 (Figure 4b). The eight-language adapter has an effective rank of 1.5, no more than a single language, so the eight languages share the same few directions.

G Grokking in train and held-out loss

Figure 8 shows the two 20M sweeps of §5.4 and §5.5 in both train and held-out loss. Train loss reaches zero in every run, so every adapter fits its training cells, and only the held-out loss separates the runs that grok from those that do not. Figure 9 repeats the rank sweep at 60M and 150M; these runs give Figure 5c.

Collapse after grokking. Two runs grok and later lose the rule: $r = 640$ at 150M, and Deep LoRA (Yaras et al., 2024), a three-matrix factorization, at $r = 173$ on 20M. In both, train loss first reaches exactly zero, and weight decay then shrinks the adapter's parameter norm by half or more, from 175 to 79 and from 61 to 34. Within the next 30k steps the norm jumps to about 400, and train and held-out loss both return to the uniform value over the 97 answers, so the adapter outputs the same distribution for every row. The runs that keep the rule avoid one of these conditions. The $r = 320$ run at 150M never reaches exactly zero train loss and its norm stays near 180, and the $r = 173$ run without weight decay keeps a norm that only grows. This matches the slingshot instability of adaptive optimizers when gradients vanish (Thilak et al., 2022), which predicts that $r = 640$ would keep the rule with a decaying learning rate or without weight decay.

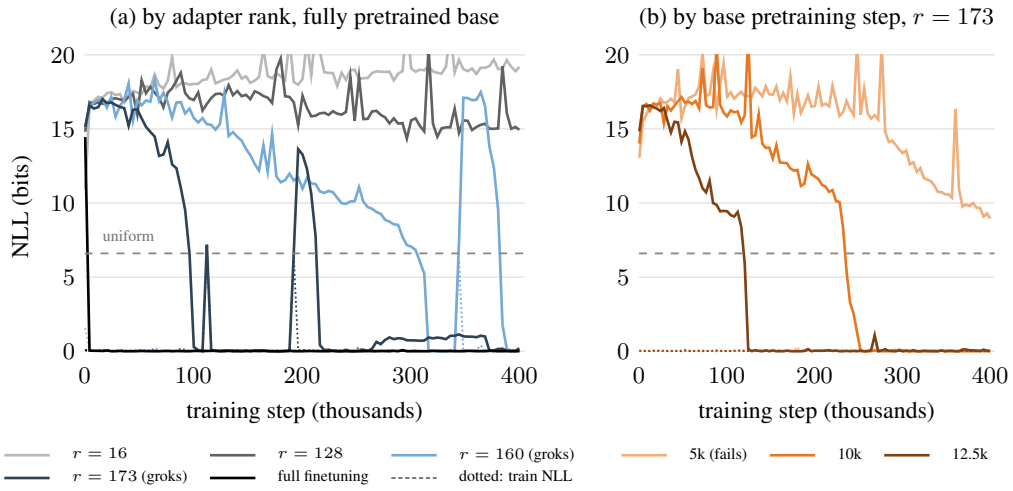


Figure 8: The two 20M sweeps of Figure 5 in train NLL (dotted) and held-out NLL (solid), by adapter rank on the fully pretrained base (a) and by base pretraining step at $r = 173$ (b).

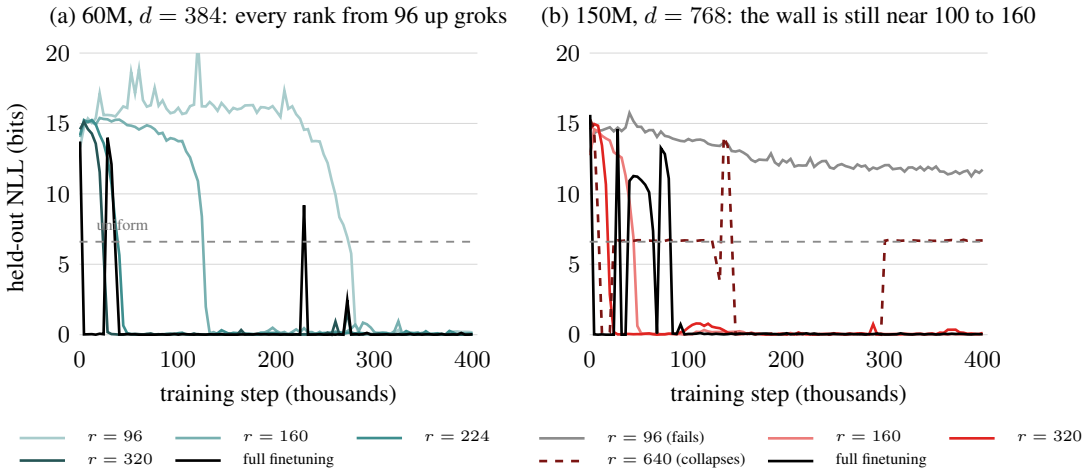


Figure 9: Held-out NLL on $a+b \bmod 97$ by adapter rank at 60M (a) and 150M (b); the dashed curve at 150M is $r = 640$.

H THE FULL FIT COMPARISON

Table 5 compares candidate forms of the law, each fit with 1B held out. The first three blocks are power laws in total adapter parameters, base size N, pretraining tokens T and the base's loss on the task. The last block uses the form of Equation 6, with p excluding the LM-head adapter. All fits use least squares in log space. Over 400 bootstrap resamples of the fit points, the 5th to 95th percentiles of the constants are k 2.28 to 2.74, r_0 10.2 to 13.0, ℓ_0 6.9 to 7.4 and m 3.7 to 8.8 for Equation 6, and k 2.08 to 2.28 and r_0 12.1 to 17.3 for Equation 5. Refitting with a Huber loss changes every constant by less than 4%. Dropping the 20% of points with the highest capacity and refitting gives $k = 2.41$, $r_0 = 10.2$, $\ell_0 = 7.2$ and $m = 6.5$, and the refit predicts the dropped points within 10%.

Range of validity. All capacity runs use $\alpha = 1$, one seed, ranks 1 to 16 and C4 losses between 3.9 and 7.0 bits per token, and the constants are fit within that range. ℓ_0 lies at the upper edge of the loss range, so the quality factor is never observed below one half. At $\alpha = 1$ the update is scaled by α/r, so higher ranks train with smaller effective steps, and η may describe this recipe rather than adapters in general. Separating the two requires a sweep over α that we have not run.

Table 5: Candidate forms of the capacity law, each fit in log space with 1B held out and scored by the mean error on the held-out 1B points. In the first three blocks p counts every adapter parameter, and forms with T are fit over the pretraining checkpoints as well, so R^2 compares only within a block. In the last block p excludes the LM-head adapter and the held-out set is the five 1B points.

Terms in the fit	Fitted law	Points (fit + held)	R^2	Held-out mean error
Adapter storage alone				
p	$C = 4.71 \cdot p^{0.902}$	19 + 3	0.922	32%
Storage × base-quality proxy				
$p \cdot N$	$C = 0.453 \cdot p^{0.764} \cdot N^{0.228}$	19 + 3	0.990	27%
$p \cdot N \cdot \mathrm{NLL}$	$C = 1.1{\times}10^{-12} \cdot p^{0.765} \cdot N^{0.212} \cdot \mathrm{NLL}^{9.093}$	19 + 3	0.992	28%
$p \cdot \mathrm{NLL}$	$C = 4.0{\times}10^{-33} \cdot p^{0.875} \cdot \mathrm{NLL}^{25.727}$	19 + 3	0.942	23%
$p \cdot N \cdot T$	$C = 0.0449 \cdot p^{0.720} \cdot N^{0.277} \cdot T^{0.080}$	77 + 5	0.986	35%
$p \cdot T$	$C = 0.132 \cdot p^{0.857} \cdot T^{0.168}$	77 + 5	0.919	23%
$p \cdot N \cdot \mathrm{NLL} \cdot T$	$C = 0.185 \cdot p^{0.720} \cdot N^{0.275} \cdot \mathrm{NLL}^{-0.508} \cdot T^{0.085}$	77 + 5	0.986	35%
Base-quality proxies alone, no storage				
N	$C = 71 \cdot N^{0.534}$	19 + 3	0.486	142%
$N \cdot \mathrm{NLL}$	$C = 6.4{\times}10^{-10} \cdot N^{0.518} \cdot \mathrm{NLL}^{8.646}$	19 + 3	0.488	146%
$N \cdot T$	$C = 0.813 \cdot N^{0.603} \cdot T^{0.123}$	77 + 5	0.572	112%
NLL	$C = 9.8{\times}10^{-72} \cdot \mathrm{NLL}^{59.751}$	19 + 3	0.110	241%
Projection storage × rank efficiency × base quality factor				
$p\,\eta(r)$, finals (Eq. 5)	$k = 2.17,\ r_0 = 14.8$	19 + 5	0.996	7%
$p\,\eta(r)\,g(\ell)$ (Eq. 6)	$k = 2.46,\ r_0 = 11.3,\ \ell_0 = 7.1,\ m = 5.8$	71 + 5	0.994	11%
$p\,\eta(r)$, all checkpoints	$k = 1.96,\ r_0 = 12.2$	71 + 5	0.980	12%
$p\,\eta(r)\,e^{-t\ell}$	$k = 5.27,\ r_0 = 11.2,\ t = 0.18$	71 + 5	0.992	20%
$p\,\eta(r)\,\ell^{-t}$	$k = 9.33,\ r_0 = 11.2,\ t = 0.93$	71 + 5	0.992	24%
$p\,\eta(r)\,/\,(1 + e^{(\ell-\ell_0)/w})$	$k = 2.57,\ r_0 = 11.3,\ \ell_0 = 7.0,\ w = 1.18$	71 + 5	0.994	11%