Memorization Capacity and Grokking in LoRA Adapters

Nguyen Tai*
Independent

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 both memorize and generalize by training on DataDecide OLMo checkpoints between 20M and 1B parameters. On random fact tables, we observe that an adapter stores about 1.2 bits per trainable parameter, a third of full training. Capacity follows a law conditioned on adapter size and base model quality and can be predicted on a held-out 1B base within 36% error. Pretraining can raise the most capacity when base models are small, and capacity alone does not decide whether an adapter can generalize to rule-based facts. On modular addition, an adapter groks only above a fixed count of high rank and sufficient base quality. Finally, we propose a simple method for finding the minimum required rank a task required from one full-finetuning run, observing that real-world tasks require roughly 15 times less rank than modular rules.

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 and Li [2025] previously find that neural networks can store about 2 bits per parameter from training on synthetic biographies. For LLMs, Morris et al. [2025] estimate LLMs to memorize around 3.6 bits of text per parameter, with grokking occurred 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

*Correspondence: tainguyen7597@gmail.com

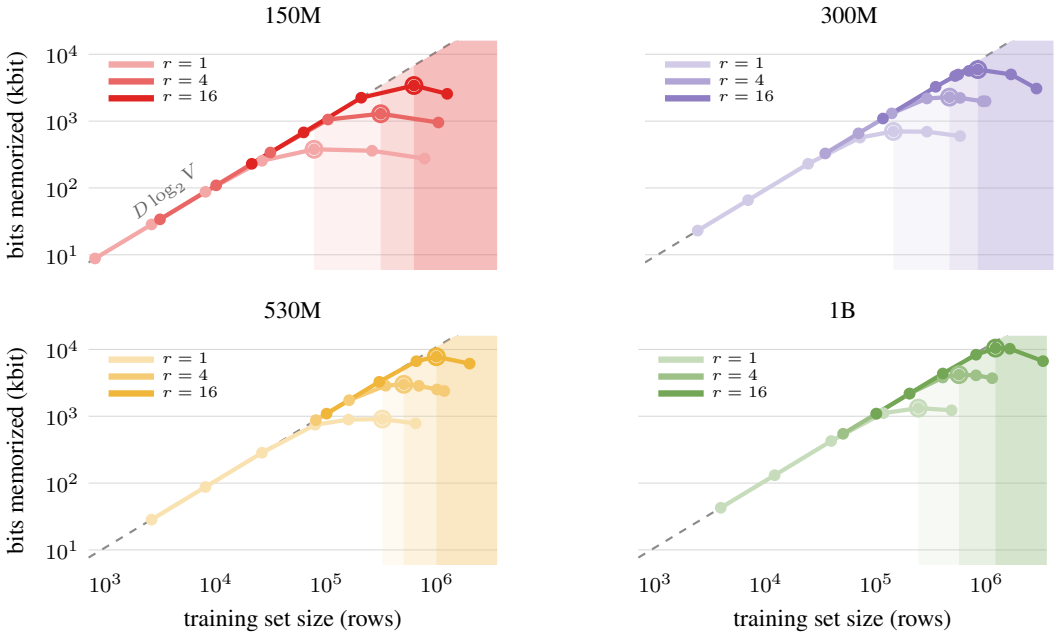

Figure 1: An adapter fills up, then forgets. Bits memorized against training set size on one shared table, one base per panel (150M, 300M, 530M, 1B), ranks 1, 4 and 16 from light to dark. Every curve rides the $D \log_2 V$ diagonal until it peaks at its capacity (hollow ring ○). The shaded band past each ring enters the forgetting regime, where new rows might overwrite stored ones.

measure LoRA capacity as a function of both the number of parameter and base model quality as measured by pretrained token count. We make the following contributions:

- **Capacity can peak but does not hold a ceiling.** On model sizes ranging from 20M-1B, we find a LoRA adapter to store about 1.2 bits per trainable parameter, roughly a third of full training. Stored bits can decline up to 36% once continuous training overwrites what model previously holds (hollow rings ○ in Figure 1).
- **A scaling law fit for capacity.** LoRA adapter has a multiplicative scaling law between the number of adapter parameters p and the base quality: $C = 0.453\, p^{0.764} N^{0.228}$. Base quality can be expressed via the number of base parameter N, number of pretraining tokens T, or familiarity with the task NLL. Our best fit can predict the capacity of a `(base, pretraining, rank)` combination of a held-out 1B within 36% error (Figure 3, Table 5).
- **Overtraining lifts capacity most for small bases.** More pretraining tokens raise the capacity of the same adapter, but there is diminishing return as the base grows, from $2\times$ at 20M to $1.1\times$ at 300M across the DataDecide checkpoints (Figure 3).
- **LoRA adapters grok slower than full finetuning.** In the over-capacity regime, a fully finetuned setup can grok much faster than a LoRA finetuning. At 20M on the synthetic set $a{+}b \bmod 97$, full finetuning crosses below chance-level loss within the first 5k steps, while an adapter at $r = 173$ takes about 100k steps to reach a perfect validation score. On the same setup, every rank up to 128 never crosses within 400k steps (Figure 4).
- **Rank tracks data structure.** The rank a task needs depends on its "structure" rather than the amount of data. An algebraic rule, which is resistant to compression, requires a smallest required rank $r^* = 9.3\, d^{0.579}$ singular update directions, from 90.1% (173 out of 192) at 20M to 29.9% (614 out of 2048) at 1B. This is in contrast to PopQA, which requires as few as 27 directions at every width ($25\times$ fewer), and concentrates trained updates in one to three directions.
- **A simple method for finding the required rank r^*.** On a budget, we can run full finetuning or choose an arbitrary big rank and read the spectrum to determine the minimal rank that can still reproduce facts (Algorithm 1).

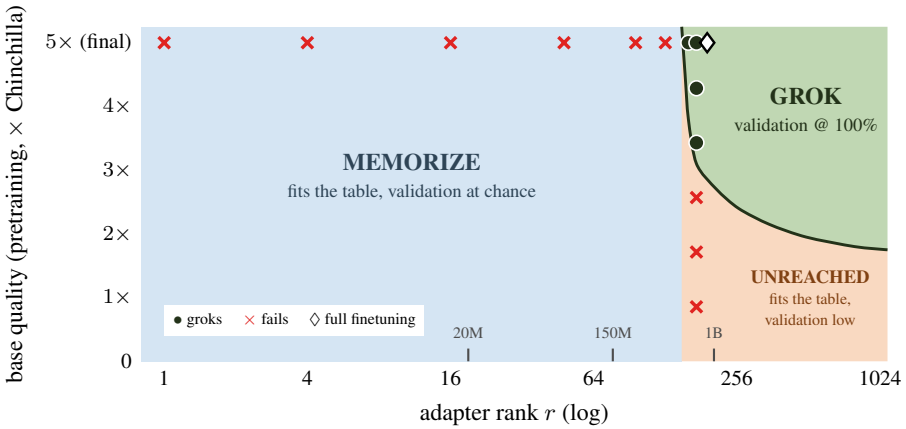

Figure 2: What decides grokking, at 20M. Every depth-1 $a+b \bmod 97$ run on adapter rank against base pretraining, with the boundary drawn as a hypothesis. Left of it an adapter memorizes. Right of it, it groks only on a base pretrained far enough, and below the curve the rule is representable but unreached. The boundary's height is measured at 20M only, between $2.6\times$ and $3.4\times$ Chinchilla. The wall stays at 96 to 160 directions as the base widens (Figure 4c), so the ranks LoRA is used at, $r = 0.1\,d$ at the ticks for 20M, 150M and 1B, clear it near 1B.

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:

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

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

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 V candidates. 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 $H(a_i \mid \hat{\theta}) = \mathrm{NLL}_2(a_i \mid q_i)$ to encode a_i. Equation 1 reduces to the difference of the two. Summing that over the training set gives us:

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

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, which is what bends every curve in Figure 1 back down past its peak, marked there by a hollow ring ∘.

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,

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

with D_c the same number in rows, and we report C/p over the adapter's p trainable parameters.

Scaling law. Given relevant parameters, we can find a functional form that predict the storage capacity of adapters $C(N, r)$. For instance, we can consider p trainable LoRA parameters set by

Table 1: Samples of synthetic facts, with answer token in bold.

`Thunder % pirate = **coconut**`	random table fact
`Kepler % Fern = **heir**`	$a+b \bmod 97$, trained: $45+13=58$
`Shakespeare % genius = **Prophet**`	trained: $37+10=47$
$\Rightarrow$ `Fern % Kepler = **heir**`	held out, follows from the rule: $13+45=58$

rank, N base parameters, and base model quality, such as the number of pretrained tokens T. On a specific task, we can also proxy base quality with the initial loss on the task. Our hypothesis is that the size of the low-rank weights and base quality can act multiplicatively in deciding capacity. We also posit $C(N, r)$ to likely follow a power law $C \propto p^a N^b$. In Section 4.2 and Table 5, we fit the law on different free variable combinations and can predict capacity on the held-out 1B scale up to high fidelity.

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 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 c is i.i.d. random. 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 $\mathrm{rect}(3000, 3000, 2000)$, 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 §5. 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 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. 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.

4 Results

4.1 Capacity peaks

Every cell taken far enough past its capacity turns over and loses 14 to 36% of its stored bits (Figure 1). The decline is real in absolute bits, not only per parameter, and it becomes visible only past about $10\times$ capacity, which is why an earlier reading of the region as a plateau held over its range. Handing an adapter more data past its peak costs stored bits.

4.2 The capacity law

Taking each curve's peak as that cell's capacity and fitting in log space gives

$$C \;=\; 0.453 \cdot p^{0.764} \cdot N^{0.228} \qquad R^2 = 0.990 \tag{4}$$

on twenty-two saturated cells across all six base sizes (Figure 3), the multiplicative form of §2. The adapter is the store. Doubling its parameters buys about $1.7\times$ the bits, and the exponent holds over a $16\times$ range of rank. The base is a multiplier with less than a third of that exponent, so a better base lifts every adapter on it but does not stand in for adapter parameters. Fitted without the 1B cells, the law places them within 36%. In density, saturated adapters store 0.45 to 1.89 bits per parameter, about a third of the 3.6 that Morris et al. [2025] measure for full models.

Base size is the quality variable that belongs in the law. Within one size, more pretraining adds capacity, most for the weakest base (Figure 3a), but pretraining tokens and the base's own loss add no out-of-sample lift once N is in the fit, and every form without a p term fails (Table 5, Appendix F).

4.3 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, D/D_c from 0.23 to 1.85, six base sizes and five pretraining checkpoints, every LoRA cell sat at or below 2.1% against chance while full finetuning reached 99.4 to 100% on 35 of 37 experiments on the same tables. The held-out loss tells a similar story: it falls from 19.7 bits to 7.0 and stops within 0.06 to 0.08 bits of uniform at every size and rank, meaning none of the rules were learned. The minimum comes at the first evaluation, after which held-out loss climbs to 13 to 16 bits while training loss goes to zero, so the adapter is memorizing rather than learning the rule slowly. The failure is not an artifact of the setup. Unfreezing the embeddings leaves the adapter at 1.3%, handing it every layernorm parameter leaves it at 1.0 to 1.7% over 48k steps, and budgets of 100k to 300k steps run 5 to 15 times past the step at which full finetuning generalizes.

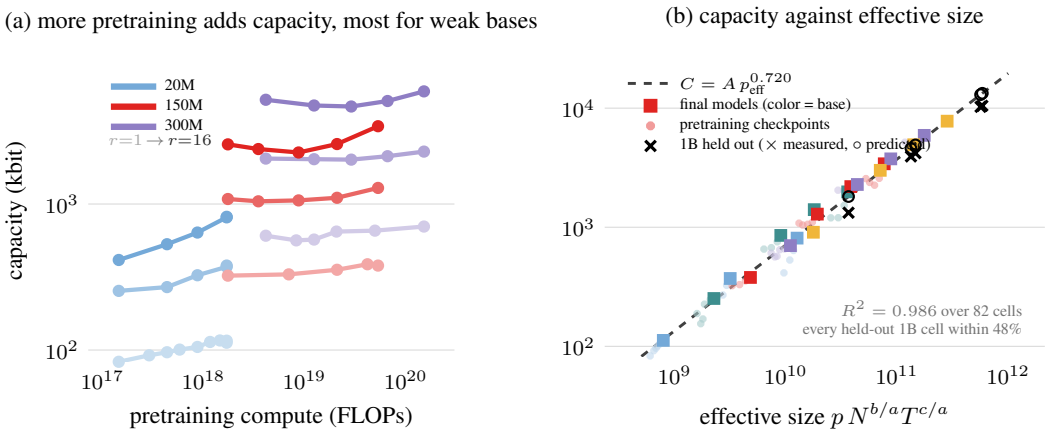

Figure 3: The capacity law, adapter storage times base quality. (a) Capacity against pretraining compute for 20M, 150M and 300M at different ranks (gradient). Every slope is small and positive and the lift is largest for the weakest base. (b) Folding adapter size p, base size N and pretraining tokens T into one effective size collapses all 82 cells onto one power law. We can predict the held-out 1B `(checkpoint, rank)` within 48%.

4.4 The rank wall

What is left is representability. At 20M, adapters at $m = 97$ with ranks 16 through 128 fit their table yet bottom out above uniform, and generalization switches on between $r = 128$ and $r = 160$ (Figure 4a). At that width 160 directions is $0.83\,d$, nearly the whole model, but wider bases move the width and not the wall (Figure 4c). At 60M every rank from 96 up groks, and at 150M $r = 96$ fits and never generalizes while $r = 160$ and 320 grok and hold. The rule fixes the number of directions it needs, 96 to 160 at every width tried, and width sets what fraction of the budget that is. The ranks LoRA is used at, near $r = 0.1\,d$, rise with width and cross the wall between 530M and 1B by extrapolation. Truncating full finetuning's own solution reads higher, 173 to 614 directions from 20M to 1B on $a{+}b \bmod 150$ with $r^* = 9.3\,d^{0.579}$ (Algorithm 1), because full finetuning spreads its solution across whatever width it has, so it bounds the requirement from above. The claim rests on one rule at three widths, and no adapter grokked the $m = 150$ tables at any rank tried.

4.5 The pretraining threshold

Representability is necessary and not sufficient. Up the $r = 173$ column of Figure 2, the same adapter on the same table fails on bases pretrained to 17%, 34% and 51% of the 20M run and groks and holds from 69% on. Once both conditions are met, the speed at which model can grok is set by headroom (Figure 4c). Full finetuning reaches the rule within 3k to 4k steps at every base width, while an adapter just above the wall requires 100k to 300k steps. The delay falls with rank above the wall and with width at fixed rank.

Weight decay is not what keeps the grok, since every value from 0 to 1 groks and holds at 20M. Two things do lose it. Deep LoRA [Yaras et al., 2024], a three-matrix factorization with a stronger low-rank bias, groks at the same rank and then collapses, and so does the most generous rank we tried, $r = 640$ at 150M. Appendix E shows the same sweeps in train loss, which rules out underfitting as the alternative reading.

4.6 Real tasks

Real tasks live far below the wall. PopQA needs 27 to 41 directions flat across a $10.7\times$ range of width, and a rank sweep moves its held-out gain by under 7% across a $16\times$ rank change. That distance from the wall, not capacity, is plausibly why low-rank adaptation works in production. The same split shows up inside the updates. Every real-task update we measured, multilingual injection, multitask instruction following [Wang et al., 2022] and long-tail factual QA, settles at an effective rank of 1 to 3, while random-fact memorization spreads across every direction it is given (Figure 5b). Memorization

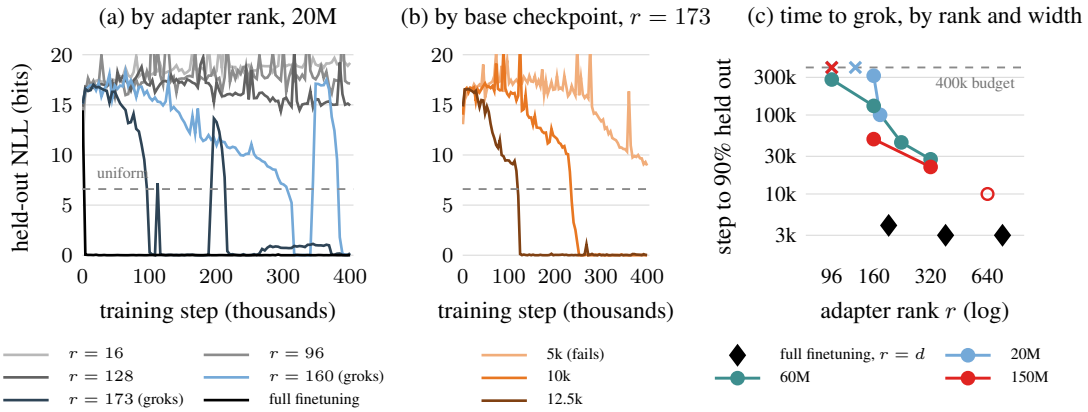

Figure 4: The two thresholds, and the speed of the grok, on $a+b \bmod 97$. Below the uniform line the LoRA adapter successfully groks the rules, above it the model has not reached grokking. (a) On the fully pretrained 20M base, ranks 16 through 128 memorize their random fact table yet bottom out above uniform, and only $r = 160$ and 173 drop to zero and stay. Full finetuning groks the fastest. (b) By base checkpoint at $r = 173$. The recipe groks from a base pretrained to step 10k+ and fails at 5k. (c) The number of steps required to reach held-out accuracy at 90%. Full finetuning takes a few thousand at every model width. Adapters can grok between 100k-300k steps, with the delay shrinking with more headroom and more width, 20M to 60M to 150M. The hollow marker represents $r = 640$ at 150M, which groks at 10k but eventually drifts back off the rule.

fills ranks and generalization does not, and truncating a skill's adapter to rank 1 keeps essentially all of its gain where truncating a memorization adapter costs bits at every discarded direction.

4.7 Identify where the bits live

We attribute storage by ablation, adapting one matrix group at a time at rank 16 and running each through the capacity protocol (Figure 6, Appendix C). The MLP pair holds essentially all the bits, while attention and the LM head store close to zero on their own, and the LM head is a large share of the parameters at small sizes, 45% at 20M against 7% at 1B. Counting projection parameters alone flattens the spread in bits per parameter across sizes from about 80% to about 14%. This means the apparent effectiveness of base size is mostly the LM head that is heavy in the number of params, but store few bits.

5 Finding the required rank

Section 4 showed two ways an adapter fails. Below the rank wall it cannot represent the rule. Above it, on a base without enough pretraining, it can represent the rule but not fully internalize it. Full finetuning provides the ceiling. It is the same update at full rank, and what it learns is the most an adapter on this base can learn. We train it once, take its converged update $\Delta W^\star$ on the matrices LoRA would adapt, and truncate it by SVD to each rank r. The required rank r^* reported in Section 4 is the smallest r at which the truncated update still solves the task (Algorithm 1). By Eckart-Young-Mirsky, the top-r truncation is the best rank-r approximation, so a surviving truncation proves that a rank-r adapter holds a solution whether or not training would reach it. Two cautions. A run that finds the rule can lose it later, so the update is taken at the best held-out step, and reinserted at full rank it must reproduce the run's own accuracy. On real tasks, whose targets are multi-token, r^* is the smallest rank keeping 90% of the held-out loss gain.

Figure 5 shows what the method returns. On the rule the required rank rises with width, from 173 directions at 20M to 614 at 1B, because full finetuning spreads its solution across whatever width it has, while on PopQA it stays at 27 to 41 directions across a $10.7\times$ range of width, a 6 to $25\times$ gap. Panel (b) shows the same split from inside trained adapters. Every real task puts its update into one or two directions, at effective rank 1.5 to 2.1, while random-fact memorization spreads across all sixteen it is given, so the rank a task needs and the rank an adapter uses agree on which regime a task is in.

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 $\mathcal{R}$

$\{\Delta W_t\} \leftarrow \text{FULLFINETUNE}(W_0)$ — 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**

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

end for

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

(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

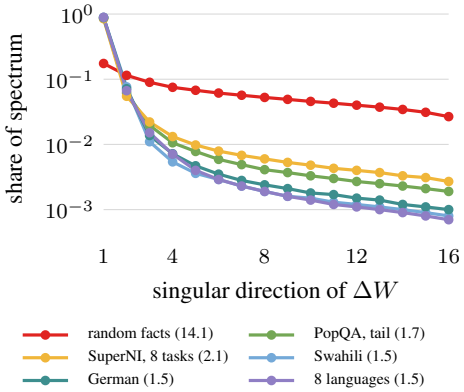

Figure 5: (a) The smallest rank at which a truncated full-finetuning update still solves the task, for the modular rule and for PopQA, by base width. (b) Singular-value spectra of trained rank-16 updates, averaged over the 65 adapted matrices; effective ranks are in parentheses. Random fact memorization spreads across all sixteen directions, while real tasks live in the first one or two directions.

The recipe follows. Train once at a generous rank, read the spectrum, truncate to r^*, and spend what is saved on data and the base.

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 and 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 and Lee [2024] characterize what low rank can express, but neither gives a stored-information figure. Schulman and Thinking Machines Lab [2025] map the regime where LoRA matches full finetuning; our width result says rule learning leaves it only when the base is too narrow to spare the rule its fixed count of directions, and their product-of-matrices dynamics may bear on why a deeper factorization's grok collapses where the two-matrix adapter's holds. Busbridge et al. [2025]'s capacity gap, a student capped by its fixed teacher, is the shape our law 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 multiplicative law in which the $\{A, B\}$ matrices carry about three times the exponent of the base size. This means the contribution of pretraining is helpful but also modest in improving LoRA capacity. 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.

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.

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.

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.

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.

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.

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.

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	$\text{rect}(3000, 3000, 2000)$, 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

The choice of SFT loss alone can move the headline from 0.05 to 0.44 bits per parameter to 1.5 at the same cell. Table 4 isolates it at 150M, rank 1, matched at every dataset size, crossing the training loss with the number of relations in the pool.

Table 4: Bits per adapter parameter at 150M, rank 1, matched at every dataset size. Sequence loss undercounts storage by $3.4\times$ and invents a $3.2\times$ effect of relation count that answer loss shows is not there.

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	$3.24\times$
answer token	$-\log p(a \mid q)$	1.521	1.574	$0.97\times$
ratio		$3.43\times$	$11.5\times$	

A row carries $\log_2 4000 \approx 12$ bits at its value. Under sequence loss the model also fits the entity, another 12 bits, and the relation, 4.3 or 9.6 bits, so most of the gradient goes to tokens that never reach the metric. Answer loss removes the effect, and with it relation count changes capacity by 3%. Every capacity number in this paper is trained and read at the answer token.

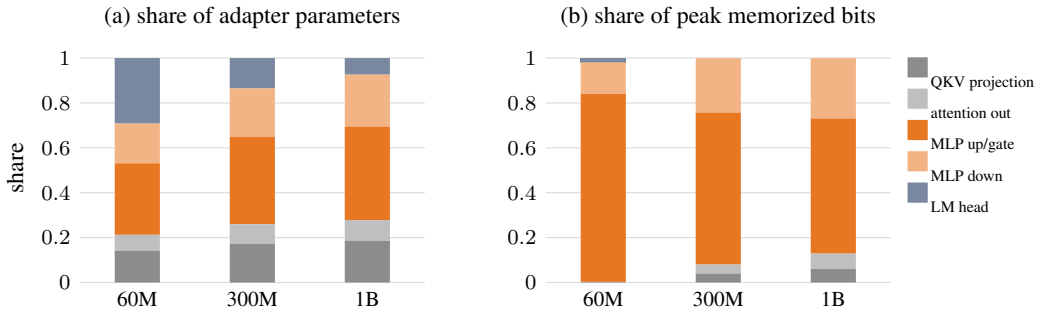

Figure 6: The allocation of LoRA's parameters responsible for storing the bits. Each matrix group ablation is run at rank 16. The MLP pair holds essentially all the bits while attention and the LM head store close to none. The LM head is a large share of the parameters at small sizes.

C Where the bits live, by matrix group

Figure 6 is the single-group ablation of §4.7, each matrix group adapted alone at rank 16.

D Spectral analyses of the trained updates

Each update decomposes by SVD, $\Delta W = \sum_i \sigma_i u_i v_i^\top$, and the shape of the σ_i says how the adapter spent its rank. We normalize the sixteen singular values of each rank-16 update to sum to one, average over the 65 adapted matrices, and summarize each spectrum by its effective rank, the perplexity $\exp(-\sum_i p_i \log p_i)$ of the normalized values. The tasks are PopQA [Mallen et al., 2023], eight Super-NaturalInstructions tasks [Wang et al., 2022], and Swahili, German and eight-language injection drawn from the Aya collection [Singh et al., 2024]. Every real task piles onto the first one or two directions, effective rank 1.5 to 2.1, while random-fact memorization spreads across all sixteen at 14.1 (Figure 5b). The eight-language adapter lands at 1.5, so the languages share one correction.

E Grokking in train and held-out loss

The two 20M sweeps of §4.4 and §4.5 in both train and held-out loss (Figure 7). Train loss reaches zero everywhere, so every setting stores the table, and only the held-out loss separates them. Figure 8 repeats the rank sweep at 60M and 150M, the runs behind Figure 4c.

F The full fit comparison

Table 5 fits every combination of the candidate drivers of the capacity law with 1B held out, the backing for $p \cdot N$ in §4.2.

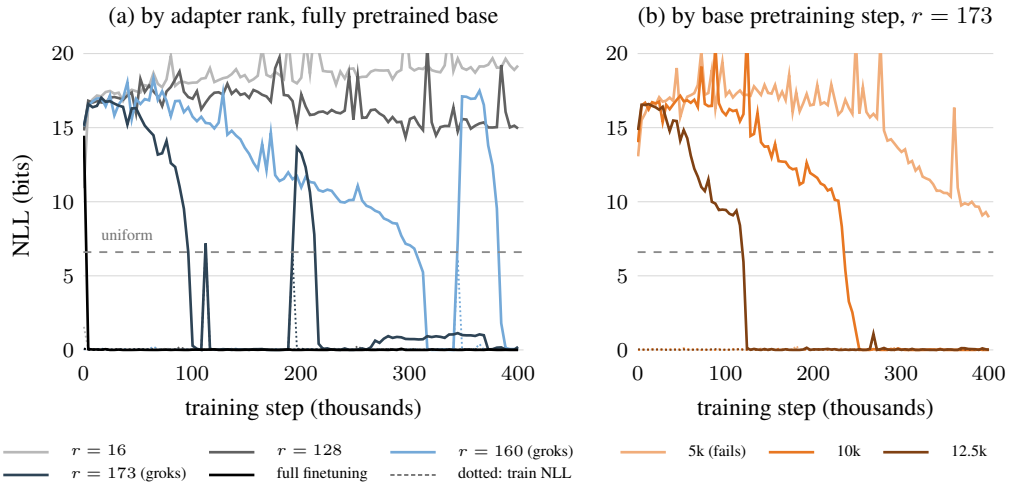

Figure 7: Every adapter fits its training half but does not always generalize. We show two 20M sweeps of Figure 4 in train NLL (dotted) and held-out NLL (solid). Train loss reaches zero within a few thousand steps at every rank and every checkpoint, and only held-out loss separates the settings, closing for $r = 160$ and 173 and for bases pretrained past step 10k.

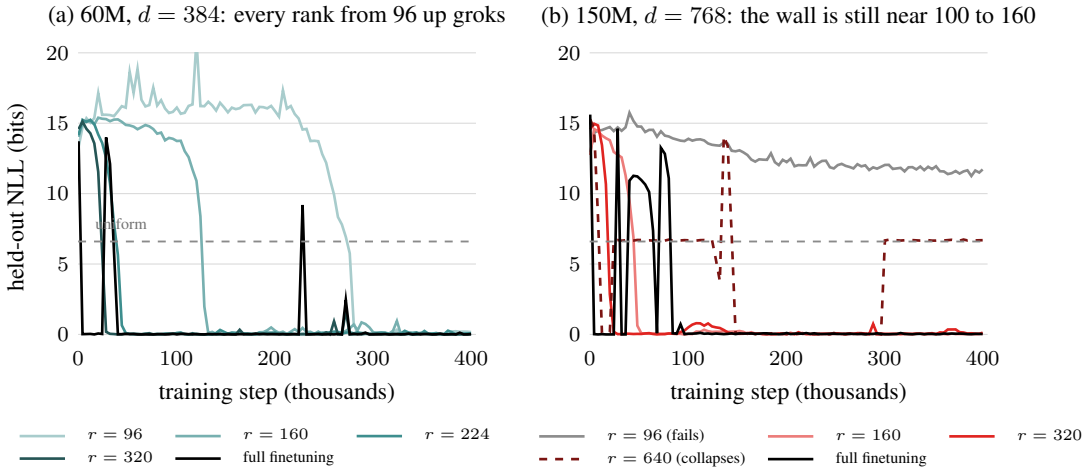

Figure 8: The rank wall on wider bases, held-out NLL on $a+b \bmod 97$. (a) At 60M every rank from 96 up groks, so the wall is at or below 96 directions. (b) At 150M $r = 96$ never leaves the confidently-wrong band while $r = 160$ and $r = 320$ grok and hold; the dashed $r = 640$ groks first and drifts back.

Table 5: Which proxy for base quality belongs in the capacity law. Each form is fit in log space with 1B held out; the cells column gives the split, and T forms are fit on the token-axis ladders together with the finals, so in-fit R^2 compares only within a block. $p \cdot N$ is the simplest form that generalizes, and every form without a storage term collapses.

Terms in the fit	Fitted law	Cells (fit + held)	In-fit R^2	1B held out
Adapter storage alone				
p	$C = 4.71 \cdot p^{0.902}$	19 + 3	0.922	55%
Storage × base-quality proxy				
$p \cdot N$ (**chosen**)	$C = 0.453 \cdot p^{0.764} \cdot N^{0.228}$	19 + 3	**0.990**	**36%**
$p \cdot N \cdot \mathrm{NLL}$	$C = 1.1{\times}10^{-12} \cdot p^{0.765} \cdot N^{0.212} \cdot \mathrm{NLL}^{9.093}$	19 + 3	0.992	36%
$p \cdot \mathrm{NLL}$	$C = 4.0{\times}10^{-33} \cdot p^{0.875} \cdot \mathrm{NLL}^{25.727}$	19 + 3	0.942	32%
$p \cdot N \cdot T$	$C = 0.0449 \cdot p^{0.720} \cdot N^{0.277} \cdot T^{0.080}$	77 + 5	0.986	48%
$p \cdot T$	$C = 0.132 \cdot p^{0.857} \cdot T^{0.168}$	77 + 5	0.919	41%
$p \cdot N \cdot \mathrm{NLL} \cdot T$	$C = 0.185 \cdot p^{0.720} \cdot N^{0.275} \cdot \mathrm{NLL}^{-0.508} \cdot T^{0.085}$	77 + 5	0.986	48%
Base-quality proxies alone, no storage				
N	$C = 71 \cdot N^{0.534}$	19 + 3	0.486	301%
$N \cdot \mathrm{NLL}$	$C = 6.4{\times}10^{-10} \cdot N^{0.518} \cdot \mathrm{NLL}^{8.646}$	19 + 3	0.488	317%
$N \cdot T$	$C = 0.813 \cdot N^{0.603} \cdot T^{0.123}$	77 + 5	0.572	317%
NLL	$C = 9.8{\times}10^{-72} \cdot \mathrm{NLL}^{59.751}$	19 + 3	0.110	518%