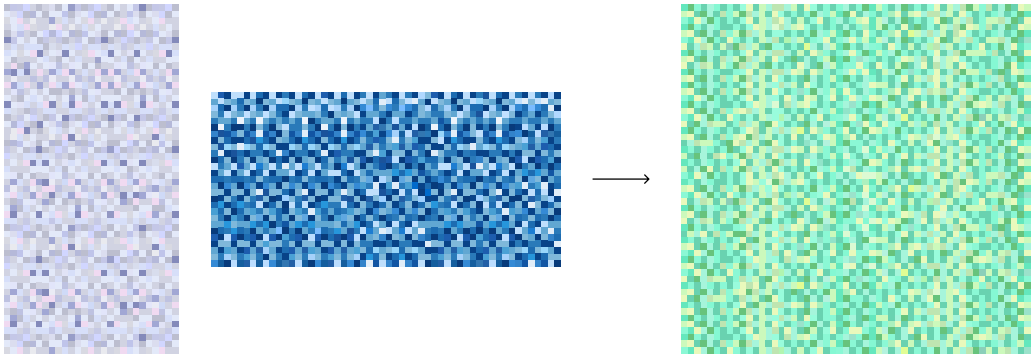
LoRA Without Regret

John Schulman in collaboration with others at Thinking Machines

Sep 29, 2025



Today's leading language models contain upwards of a trillion parameters, pretrained on tens of trillions of tokens. Base model performance keeps improving with scale, as these trillions are necessary for learning and representing all the patterns in written-down human knowledge.

In contrast, post-training involves smaller datasets and generally focuses on narrower domains of knowledge and ranges of behavior. It seems wasteful to use a terabit of weights to represent updates from a gigabit or megabit of training data. This intuition has motivated parameter efficient fine-tuning (PEFT), which adjusts a large network by updating a much smaller set of parameters.

The leading PEFT method is low-rank adaptation, or LoRA. LoRA replaces each weight matrix W from the original model with a modified version $W' = W + \gamma BA$, where B and A are matrices that together have far fewer parameters than W, and γ is a constant scaling factor. In effect, LoRA creates a low-dimensional representation of the updates imparted by fine-tuning.

LoRA may offer advantages in the cost and speed of post-training, and there are also a few operational reasons to prefer it to full fine-tuning (henceforth, FullFT):

- **Multi-tenant serving.** Since LoRA trains an adapter (i.e., the A and B matrices) while keeping the original weights unchanged, a single inference server can keep many adapters (different model versions) in memory and sample from them simultaneously in a batched way.[1] Modern inference engines such as vLLM and SGLang implement this feature.
- **Layout size for training.** When fine-tuning the whole model, the optimizer state needs to be stored along with the original weights, often at higher precision. As a result, FullFT usually requires an order of magnitude more accelerators than sampling from the same model does, and thus a different layout.[2] Since LoRA trains far fewer weights and uses far less memory, it can be trained on a layout only slightly larger than what is used for sampling. This makes training more accessible, and often more efficient.
- **Ease of loading and transfer.** With fewer weights to store, LoRA adapters are fast and easy to set up or transfer between machines.

These reasons are sufficient to explain the growing popularity of LoRA since the publication of the original LoRA paper in 2021.[3] However, the literature is unclear on how well LoRA performs relative to FullFT.

There is agreement that LoRA underperforms in settings that resemble pre-training,[4] namely those with very large datasets that exceed the storage limits of LoRA parameters. But for dataset sizes that are typical in post-training, LoRA has sufficient capacity to store the essential information. However, this fact makes no guarantees regarding sample efficiency and compute efficiency. The question is: *can LoRA match the performance of full fine-tuning, and if so, under which conditions?*

In our experiments, we find that indeed, when we get a few key details right, LoRA learns with the same sample efficiency as FullFT and achieves the same ultimate performance.

What matters for LoRA

This article covers a series of supervised fine-tuning and reinforcement learning experiments we conducted to determine the conditions under which LoRA matches FullFT efficiency. To this end, we did a few things differently from previous experiments on LoRA:

- We investigated the general relationship between training set size and number of LoRA parameters, rather than focusing on specific datasets and tasks.
- In supervised learning, we measured *log loss* rather than employing sampling-based evals, with the same goal of generality in mind. Log loss measurement gives clean results and scaling laws over ranges of training steps and training parameters.

We find that:

- For supervised fine-tuning on small-to-medium-sized instruction-tuning and reasoning datasets, LoRA performs the same as full fine-tuning.
- For datasets that exceed LoRA capacity, LoRA underperforms FullFT. Rather than the loss reaching a distinct floor that it can't go below, LoRA results in worse training efficiency that depends on the relationship between model capacity to dataset size.
- In some scenarios, LoRA is less tolerant of large batch sizes than full fine-tuning — it pays a larger penalty in loss as batch size increases beyond some point. This penalty is not mitigated by increasing the LoRA rank; it is a property of the product-of-matrices parametrization, which has different training dynamics than optimizing the original weight matrix.
- Even in small data settings, LoRA performs better when applied to all weight matrices, especially MLP and MoE layers. Attention-only LoRA underperforms even when we match the number of trainable parameters by using higher rank for attention-only LoRA.
- LoRA performs equivalently to FullFT for reinforcement learning even with small ranks. We find that RL requires very low capacity, a result we anticipated based on information-theoretical arguments.

We also studied the impact of hyperparameters used for LoRA on its learning rate relative to full fine-tuning. We examine some invariances in hyperparameters like init scales and multipliers, and explain why the 1/r prefactor makes the optimal learning rate (LR) approximately independent of rank. We also show experimentally how the optimal LR for LoRA relates to the optimal LR for FullFT.

The outcome of our experiments is the characterization of a "low-regret regime" where LoRA performs similarly to FullFT in terms of dataset size and LoRA parameters. We found this regime covers most post-training scenarios, opening the door to the use of efficient fine-tuning in many applications.

Methods and results

We designed our experiments to measure in detail the relative performance of LoRA compared to FullFT across a range of conditions. Here are some details of our experimental setup:

- We varied the LoRA rank over three orders of magnitude, with rank between 1 and 512, and compared these to full fine-tuning.
- To eliminate potential confounds from using a suboptimal learning rate, we swept the LR for each experimental condition. We used constant learning rate schedule (no warmup or cooldown).
- Our experiments used Llama 3 series models[5] and Qwen3 models[6], including a mixture of experts (MoE) model.
- The main supervised learning experiments used the Tulu3[7] and OpenThoughts3[8] datasets, focused on instruction following and reasoning, respectively. The two sets differ significantly in scope, structure, and application, supporting the generality of our results.
- Our RL experiments used mathematical reasoning tasks with answer correctness as the reward.

LoRA rank

We trained for a single epoch on the Tulu3 dataset and a subset of the OpenThoughts3 datasets. For each dataset and model size, we swept over LoRA rank and learning rate. In the plots below, we draw one colored line for each rank, where the line is obtained by taking the pointwise minimum over all learning rates at each training step:

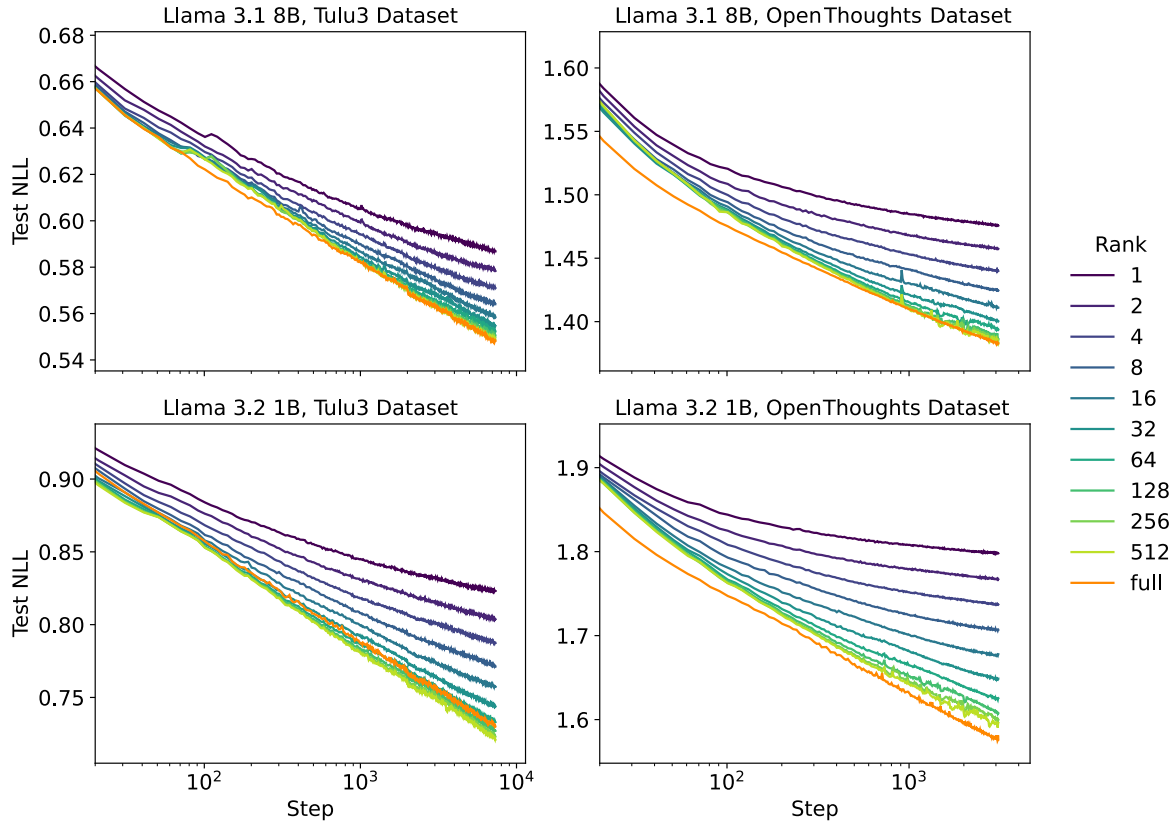


Figure 1: LoRA training curves for various ranks on Tulu3 and OpenThoughts3 datasets. FullFT and high-rank LoRAs have similar learning curves with loss decreasing linearly with the logarithm of steps. Lower-rank LoRAs fall off the minimum-loss curve when the adapter runs out of capacity. In the bottom plots (1B model) high-rank LoRA performs better than FullFT on one dataset and worse on the other. There might be some random variation in how LoRA performs on different datasets, due to differences in training dynamics or generalization behavior.

We see that FullFT and high-rank LoRAs have similar learning curves with loss decreasing linearly with the logarithm of the number of steps. Medium and low-rank LoRAs fall off the minimum-loss learning curves at some threshold of steps that correlates with rank. Intuitively, learning slows down when the adapter runs out of capacity, which in turn is determined by rank.

Next, we plot how loss changes with LR to check that our sweep covers the best learning rate for each rank.

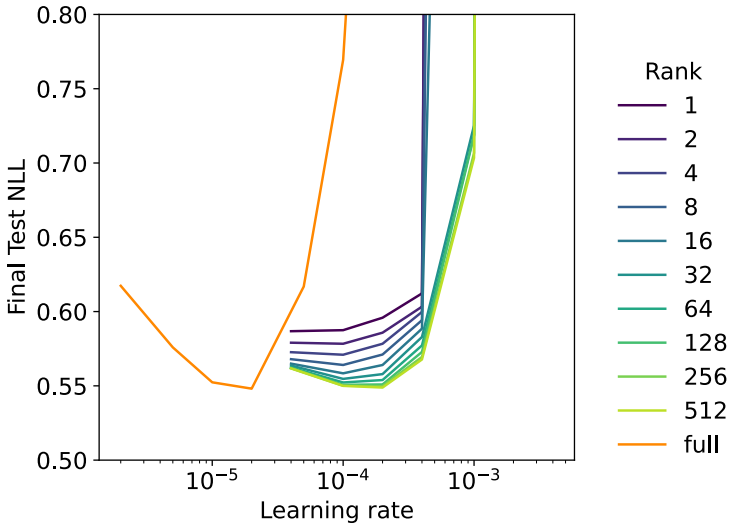


Figure 2: Learning rate versus final loss for various LoRA ranks on Tulu3. Minimum loss is approximately the same for high rank LoRA and FullFT. Optimal LR is 10 times higher for LoRA.

We find that the optimal learning rate for FullFT is lower by a factor of 10 than for high-rank LoRAs.[9] We'll return to this in our discussion of LoRA hyperparameters later on.

The optimal LR seems to be similar for all the LoRA runs across different ranks; we give a theoretical explanation for this finding below. However, there does seem to be some rank dependence, with lower optimal LR for rank=1 than for higher-rank LoRAs. The optimal LR changes by a factor of less than 2 between rank=4 and rank=512.

Batch size effects

We found that in some settings, LoRA is less tolerant of large batch sizes than FullFT. The performance gap grows with larger batch sizes, independent of rank. For this next experiment, we used a small 10,000-example subset of OpenThoughts3.

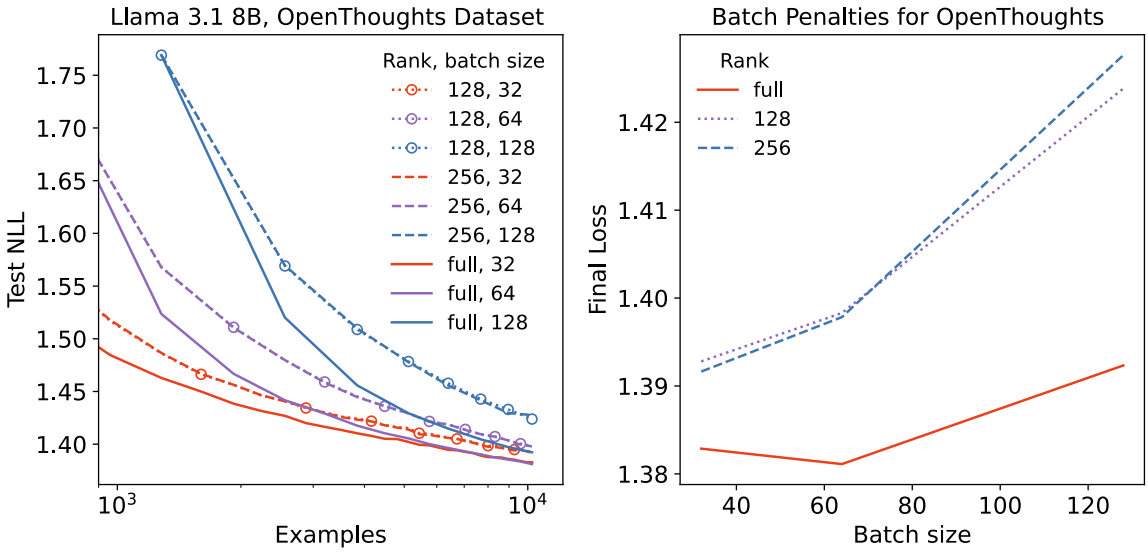


Figure 3: Batch size effects on LoRA vs FullFT performance. Left: Learning curves for different batch sizes show a persistent gap between LoRA (dashed) and FullFT (solid) at large batch sizes. Right: Final loss as a function of batch size shows LoRA pays a larger penalty for increased batch size.

The left-hand plot in Figure 3 shows a persistent gap between the LoRA (dashed lines) and FullFT (solid line) learning curves at large batch sizes. The gap is smaller and shrinks over time for the smaller batch size of 32.

The right-hand chart plots final loss as a function of batch size. We see the gap in loss for LoRA increasingly diverging from FullFT for larger batch sizes.

The learning gap at large batches doesn't seem to depend on rank, but rather seems to be a property of LoRA. The likely reason is that the product-of-matrices parametrization (BA) has less favorable optimization dynamics on this dataset than the full matrix (W). However, both LoRA and FullFT achieve their best loss at smaller batch sizes, so this gap may not matter as much in practice.

Layers Where LoRA Is Applied

We investigated the effects of applying LoRA to different layers in the network. The original paper by Hu et al. recommended applying LoRA only to the attention matrices, and many subsequent papers followed suit, though a recent trend has been to apply it to all layers.[10] Indeed, we achieved far better results when applying LoRA to all layers, in particular, the MLP (including MoE) layers. In fact, applying LoRA to the attention matrices shows no additional benefits beyond applying it to the MLPs only.[11]

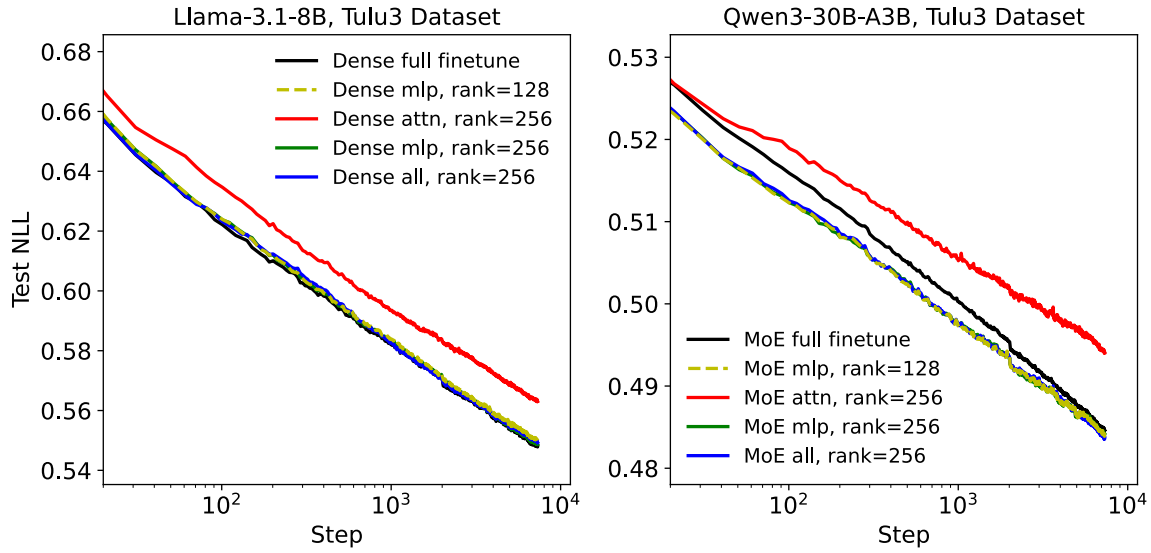


Figure 4: Attention-only LoRA significantly underperforms MLP-only LoRA, and does not further improve performance on top of LoRA-on-MLP. This effect holds for a dense model (Llama-3.1-8B) and a sparse MoE (Qwen3-30B-A3B-Base).

The underperformance of attention-only LoRA is not explained by having fewer parameters. In this particular case, attention-only with rank 256 underperforms MLP-only with rank 128, despite them having approximately the same number of parameters. (Compare the bold numbers in the table below.)

LoRA configuration	Params
mlp, rank=256	0.49B
attn, rank=256	**0.25B**
all, rank=256	0.70B
mlp, rank=128	**0.24B**

Parameter counts for LoRA on Llama-3.1-8B

For the MoE experiment, we trained a separate LoRA on each expert, with the rank of each equal to the total rank divided by the number of active experts (equal to 8 for Qwen3 MoE). This scaling keeps the ratio of LoRA parameters to FullFT parameters the same for MoE layers as for other layers.

We did similar experiments comparing different LoRA layers in two additional settings: (1) supervised learning on a small subset of the OpenThoughts3 dataset with rank=256, and (2) reinforcement learning on the MATH dataset. We describe our

experimental setup in the following section. Attention-only LoRA underperforms MLP-only LoRA (which performs similarly to MLP+attention) in these settings as well.

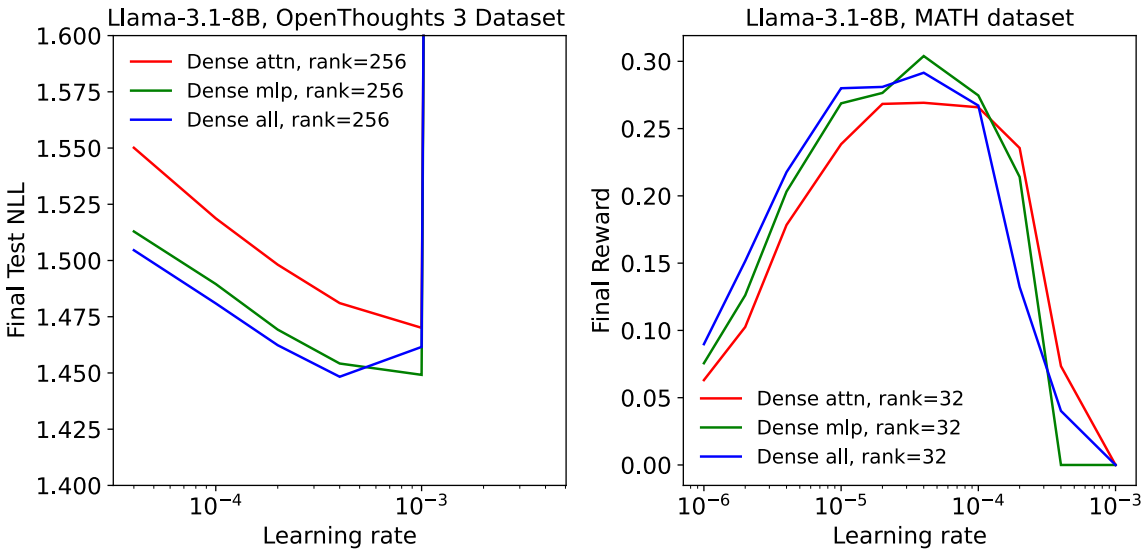


Figure 5: Learning rate vs final loss or reward, when varying which layers we apply LoRA to.

Reinforcement learning

A key finding from our experiments is that LoRA fully matches the learning performance of FullFT when running policy gradient algorithms for reinforcement learning, even with ranks as low as 1.

For these experiments, we used a basic policy gradient algorithm with an importance sampling correction; objective $= \sum_t \frac{p_{\text{learner}}}{p_{\text{sampler}}} Adv_t$.[12] We used a GRPO-like centering scheme[13] where we sample multiple completions per problem and subtract the mean reward per group.

Figure 6 (below) shows LR sweeps on the MATH[14] and GSM[15] datasets, using typical hyperparameters for each. We used the Llama-3.1-8B base model as Qwen2.5 and Qwen3 are known to have been pretrained on data that improves their math performance, as described by the Qwen tech reports[16], which makes it harder to measure what is being learned only during RL.

LoRA shows a wider range of performant learning rates and arrives at the same peak performance as FullFT (black line), at least within the precision limits afforded by the noisiness of RL.

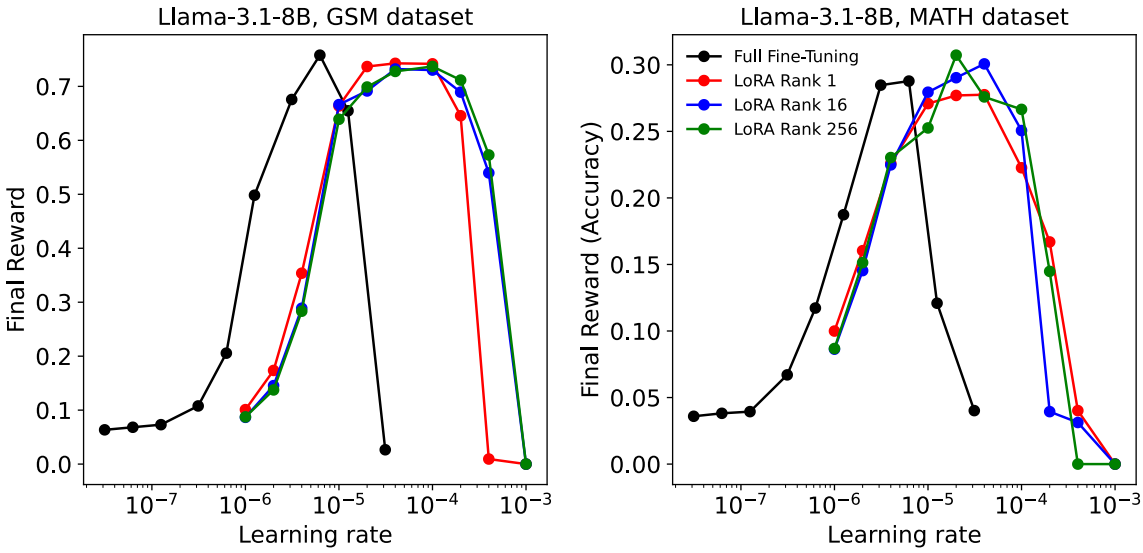


Figure 6: Learning rate vs final reward (accuracy) when doing RL on grade school math (GSM, left) or MATH (right) dataset.

This result is anticipated by an information-theoretic argument. Supervised learning arguably provides O(number of tokens) bits per episode. In contrast, in policy gradient methods, learning is driven by the advantage function which provides only O(1) bits per episode. When each episode contains thousands of tokens, RL absorbs ~1000 times less information per token in training than supervised learning does.

We can use more precise numbers based on our experiments. In the MATH example, we trained on ~10,000 problems with 32 samples per problem. Assuming each completion yields a single bit of information, the whole training process only needs to absorb 320,000 bits. Rank-1 LoRA for Llama-3.1-8B already has 3M parameters[17], almost 10 times that number. Even at rank-1, LoRA has more than enough capacity to absorb all the information provided during training.

As another point of comparison, DeepSeek-R1-Zero was trained on 5.3M episodes[18], corresponding to 5.3M bits of information. This is less than the number of parameters in a low-rank LoRA, and we predict that the results can be replicated with LoRA.

For additional validation of our findings of LoRA's effectiveness in reasoning RL, we carried out larger-scale experiments with Qwen3-8b-base on the DeepMath dataset[19] as it is much larger than the MATH dataset and in general contains harder problems. To speed up experiments, we restricted the samples to a length of 8192 tokens for training and evaluation. This sample length allows for backtracking and reasoning but limits the performance, relative to longer chain-of-thought.

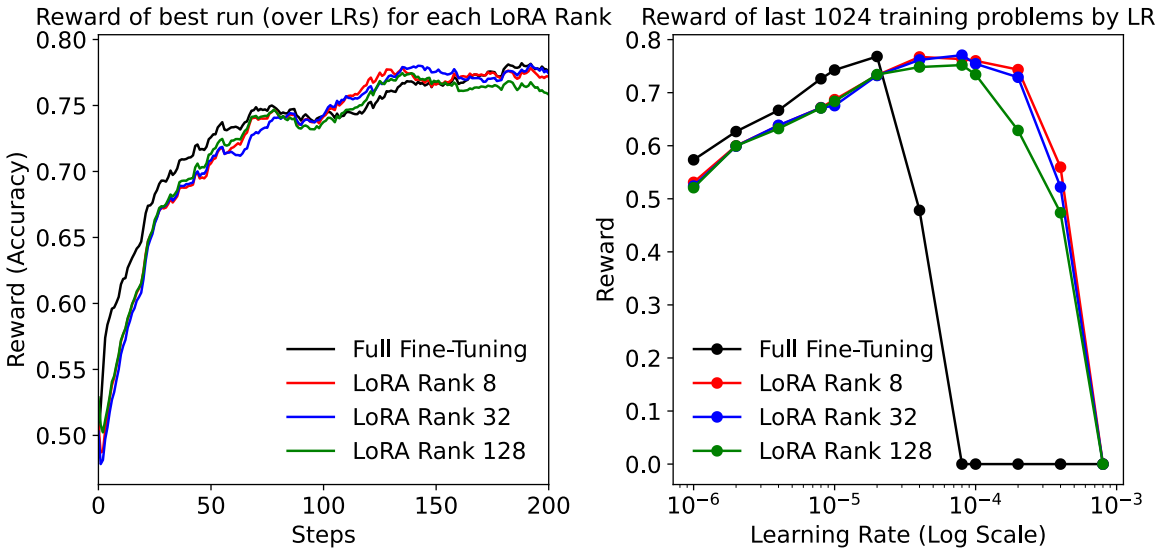


Figure 7: Experiments on the DeepMath dataset with Qwen3-8b-base. In the left plot, we show the learning curve for different ranks and full fine-tuning. For each of these settings, we show the best learning rate, which results in the highest final performance. On the right, we plot learning rate vs final performance. As in our previous math experiments, LoRA seems to have a wider peak of near-optimal learning rates.

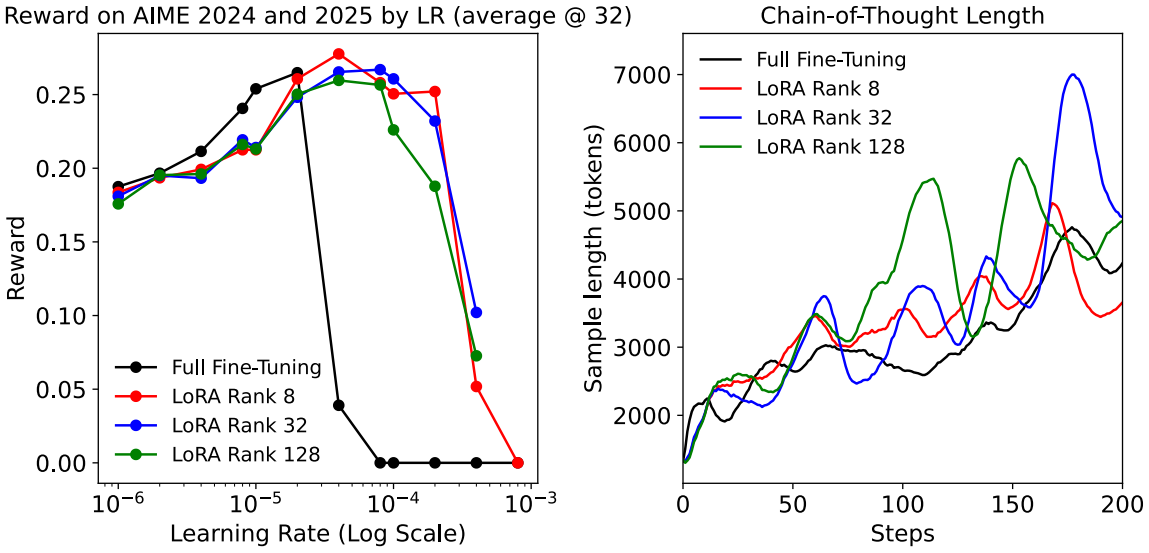


Figure 8: Additional plots from experiments on the DeepMath dataset with Qwen3-8b-Base. The left plot shows the benchmark scores on the AIME test set, which is more challenging than the training set. The right plot shows the chain-of-thought (CoT) length over training steps, which can be seen as a sign of learning to reason.

We observe that when picking the optimal learning rates for each setting, training progresses in an almost identical way for LoRAs with different sizes and full fine-tuning. Moreover, we see similar findings when we evaluate the models on the held-out problems of AIME 2024 and AIME 2025. Furthermore, we observe similar qualitative behavior from the LoRA and full-finetuning runs: both develop advanced reasoning behaviors such as backtracking, self-verification and in-context exploration, which is visible in the lengthening of the model CoTs.

Setting LoRA hyperparameters

One barrier to LoRA adoption is the necessity to choose optimal hyperparameters, which are different from ones optimized for FullFT. In this section, we show that this problem isn't as daunting as it appears at first glance and discuss our findings related to hyperparameter choice.

Optimal learning rate and rank

Following Hu et al., we consider the following parametrization for LoRA:

$$W' = W + \frac{\alpha}{r} BA$$

Where r is the LoRA rank, α is the LoRA scaling factor, and A, B are the LoRA weight matrices (of rank r). We use $\alpha = 32$ for the experiments in this article, following standard practice from other implementations.

The $1/r$ scaling factor makes the optimal learning rate approximately independent of rank. In fact, a stronger condition holds – the learning curve is exactly the same at the beginning of training, regardless of rank. This effect is striking, and in our experiments the closeness of the learning curves for different ranks had us worried that a bug caused the rank parameter to be ignored. It follows that in a short training regime, the optimal LR is also independent of rank. However, as we showed above in our plots of learning rate vs loss (Figure 2), optimal LR has some rank-dependence in the longer-training regime.

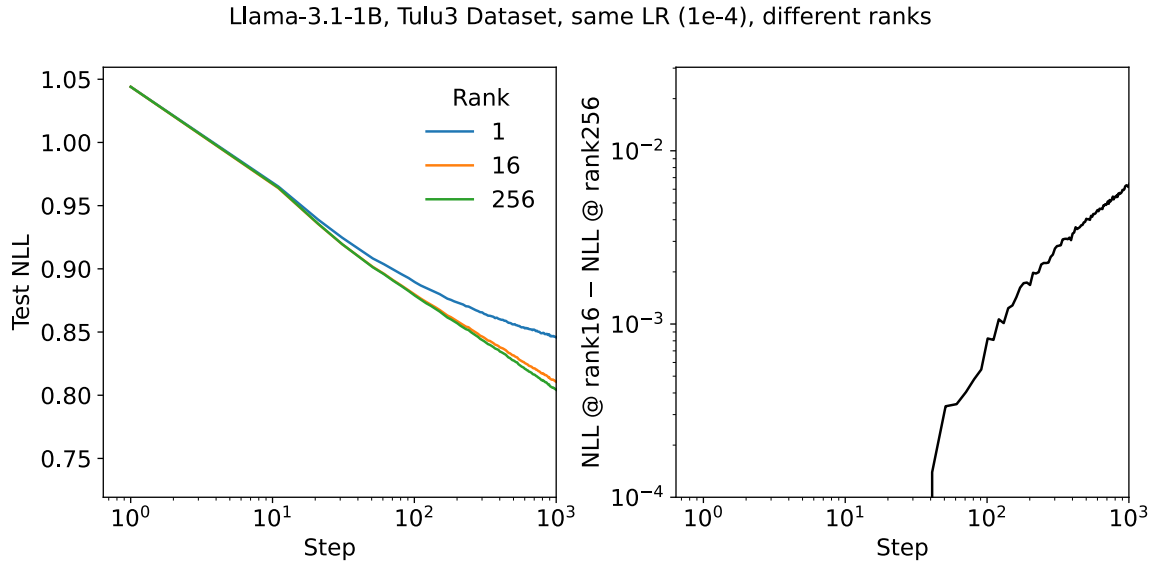


Figure 9: These plots look at the differences in the learning curves, early in training, for different ranks with the same learning rate. On the left, we show the learning curves. The right shows the difference between rank 16 and 256, which grows over

time. Strangely, it is negative (though tiny) for the first few steps, so that part of the curve is missing from the plot.

We can partly explain this result by looking at the expected update to the LoRA matrix after the very first training update. We can think of the LoRA product BA as the sum of r rank-1 outer products: $BA = \sum_{i=1}^{r} b_i a_i^T = \sum_{i=1}^{r} \Delta_i$, where we define $\Delta_i = b_i a_i^T$. Here, $\partial \text{Loss}/\partial \Delta_i$ is the same for all i; however the gradients $\partial \text{Loss}/\partial b_i$ and $\partial \text{Loss}/\partial a_i$ will depend on the initialization ($\partial \text{Loss}/\partial b_i$ depends on a_i, for example). Since the initialization of a_i and b_i do not depend on rank, it follows that $\mathbb{E}[\Delta_i]$ is the same for all i and does not depend on rank. At the first step of training, the expected update from each of these terms is equal and independent of the rank. It follows that $(1/r)\sum_{i=1}^{r} \Delta_i$ is just a sample average of r terms with the same expectation, so the expectation of the average, i.e., the change to the adapter $(1/r)BA$, doesn't depend on the rank.

Parametrization invariances

There are four hyperparameters potentially applicable to LoRA:

1. The scale factor α which appears in α/r.
2. The learning rate for the down-projection matrix A, LR_A
3. The learning rate for the up-projection matrix B, LR_B.
4. The initialization scale of matrix A, init_A. For a random initialization, this is the standard deviation of A's initial elements. Matrix B is initialized to zero, so there is no need to define init_B.

Having to tune four different parameters may seem overwhelming. However, invariances in the training dynamics mean that two of these are redundant, and learning behavior is determined by two. We show this invariance by noting that when training with Adam and $\varepsilon = 0$, the[20] optimization process is invariant to the following two-parameter transformation.

For $p, q > 0$:

- $\alpha \to \frac{1}{pq} \cdot \alpha$
- $\text{init}_A \to p \cdot \text{init}_A$
- $LR_A \to p \cdot LR_A$
- $LR_B \to q \cdot LR_B$

Since two degrees of freedom out of the four don't affect the learning process, we are left with a 2D parameter space. We can choose different bases for this 2D space, such

as the following one which lends itself to a straightforward interpretation:

- $\alpha \cdot \text{init}_A \cdot LR_B$. This determines the scale of initial updates, or, equivalently, the initial slope of the learning curve. Since B is initialized to zero, LR_A and the initial updates to A are irrelevant.
- init_A / LR_A. Since Adam updates the elements of A by approximately LR_A at each step, this timescale parameter determines the number of steps it takes to significantly transform A away from its initial state.

We can reinterpret some proposals from previous work on LoRA in terms of this basis.

LoRA+[21] proposes to use different LRs on A and B, with a higher rate for B. Expressed in terms of our basis above, increasing LR_B is equivalent to increasing init_A / LR_A so that A changes on a longer timescale.

Unsloth's LoRA Hyperparameter Guide recommends using higher values of α for high-rank LoRA, e.g. by avoiding the $1/r$ scaling. This is also equivalent to increasing init_A / LR_A. When we increase α, LR_A and LR_B need to be lowered in compensation to get the same update size. This in turn simply makes LR_A smaller relative to init_A.

In our experiments, we used the standard parametrization used in the Huggingface `peft` library[22] proposed by Hu et al: a uniform distribution for A with scale $1/\sqrt{d_{in}}$, zero initialization for B, the same LR for both, and $\alpha = 32$. We were unable to improve on these hyperparameters in our experimentation.

Optimal learning rates for LoRA vs. FullFT

Our experiments showed that the optimal LR for LoRA is consistently 10x the one used for FullFT in the same application, for both supervised learning and reinforcement learning. This shows up in every U-shaped plot of performance (loss or reward) charted against learning rate. This observation should make it more straightforward to transfer learning hyperparameters from FullFT to LoRA.

We don't yet have an adequate theoretical explanation for this observation. We can attempt to derive this result from the facts that optimal LoRA LR is invariant to rank and that full-rank LoRA is directly comparable to FullFT. This analysis suggests a LR ratio of the model's hidden size divided by $2 \cdot \alpha$, which doesn't match the empirical result of the optimal ratio being fixed at 10 independent of the base model.

For our empirical analysis, we conducted an LR sweep of 14 different Llama and Qwen models for both LoRA and FullFT on the Tulu3 dataset. From those sweeps, we fit a

function that predicts the optimal learning rate based on the model's hidden size and an indicator of whether it's Llama or Qwen. The functional form used was:

$$\text{LR} = M_{\text{LoRA}} \cdot \left(\frac{2000}{\text{hidden size}}\right)^{\text{model pow}+\text{LoRA pow}}$$

Where:

- M_{LoRA} is a multiplier applied when LoRA is used (1 if FullFT)
- model pow is an exponent adjustment, calculated separately for each model source (Llama and Qwen)
- LoRA pow is an additional exponent adjustment for LoRA
- hidden size is the dimension of the residual stream of the model.

We scored a predicted learning rate by using linear interpolation to predict the loss, based on the data from our sweep, and rated the parameters by summing the predicted loss over the 14 problems. Our optimization found a multiplier of 9.8 for LoRA over FullFT, and different dependence on hidden_size for Qwen3 and Llama models, but LoRA LRs had the same dependence on hidden_size as FullFT LRs, i.e., the optimization found $\text{LoRA pow} = 0$.

Learning rates in short and long runs

The typical initialization of LoRA creates an implicit schedule of change in the effective learning rate. This leads to differences between short and long training runs, and some differences in the shape of learning curves compared to FullFT.

At the start of training, B is initialized to zero. While B is very small, changes in A have negligible effects on the adapter BA which is added to the original network weights. As B grows larger, updates to A start to have a bigger impact on the network outputs, with the effective learning rate increasing over the course of training as B approaches A in scale. We found that by the end of the full training runs on the Tulu3 and OpenThoughts datasets, the B matrices ended up with larger spectral norms than the A matrices.

This implies that the optimal LR should be set higher for shorter training runs. Preliminary evidence suggests an optimal multiplier around 15x over the FullFT for short runs[23], converging to the aforementioned 10x multiplier for longer runs.

Discussion

We want to move beyond our empirical results to discuss some broader considerations related to LoRA performance and applicability that would be of interest to both researchers and builders.

First, let us examine in more depth our main result, namely the two conditions under which LoRA performs similarly to full fine-tuning:

1. LoRA is applied to all layers of the network, especially the MLP/MoE layers which house most of the parameters.
2. LoRA works well when not capacity constrained, i.e., the number of trainable parameters exceeds the amount of information to be learned, which can be estimated in terms of dataset size.

When (1) is satisfied, we get similar learning dynamics to FullFT at the very start of training. Then, as per (2), LoRA continues to look like FullFT until we start reaching capacity limits.

Why LoRA might be needed on all layers

As we showed earlier, if we put LoRA on only the attention layers, we get slower learning even in the tiny-data regime.

One possible explanation could come from thinking about the empirical neural tangent kernel (eNTK) as an approximation of what happens when we do a small amount of fine-tuning, following Malladi et al.[24] eNTK is based on the dot products of gradients, specifically gradients $g_i = \partial/\partial\theta \log p(\text{token}_i|\text{prefix}_i)$, and $K(i,j) = g_i \cdot g_j$. As a consequence, the layers with the most parameters will typically have the most influence on the kernel. The paper also points out the eNTK for LoRA is approximately the same as that for full fine-tuning, when you train all the layers. So LoRA training $\approx$ eNTK(LoRA) $\approx$ eNTK(FullFT) $\approx$ FullFT. The approximation eNTK(LoRA) $\approx$ eNTK(FullFT) only holds when we apply LoRA to the layers that contain most of the parameters which make up the dot products.

How much capacity is needed by supervised and reinforcement learning?

Past work[25] has shown that neural networks can store 2 bits per parameter. These results pertain to the maximum amount of information absorbed in the long-training limit, not to the compute efficiency or rate of learning.

The 2-bits-per-parameter result relied on synthetic datasets cleverly constructed to contain a precise amount of information. It's not as straightforward to estimate the information content required for a given realistic learning problem. One classic observation is that when minimizing log-loss, the total log-loss measured during the first epoch of training provides a measurement of the dataset's description length. That is, an upper bound for the number of bits required to memorize the dataset. LLM datasets usually have a loss of around 1 bit (0.69 nats) per token, depending on dataset and model size.

This estimate measures the capacity required to perfectly memorize the dataset, which overestimates the actual capacity needed for "generalizable" learning that reduces log-loss on test data. Measuring the capacity requirements of supervised learning and how these interact with the number of trainable parameters is an open question for future work.

For RL, we claimed that policy gradient algorithms learn roughly 1 bit of information per episode, given that there's a single reward value at the end of the episode. This isn't a fundamental property of RL, as other algorithms could conceivably learn a lot more from each episode. For example, model-based RL algorithms train the learning agent to predict the observations and build a world model, potentially extractingmore information per episode. The claim of 1-bit-per-episode may only apply narrowly to policy gradient algorithms.

We can sharpen the bits-counting argument in information-theoretic terms. Consider an episode, consisting of a trajectory τ and final reward, as a message (i.e., a noisy channel) that provides some information about the unknown reward function R. We'll condition on the current policy and training history and look at the mutual information between the policy gradient estimator and R. The REINFORCE update is $G = S \cdot \mathrm{Adv}$ with $S = \nabla \log p_\theta(\tau)$. S is independent of R given the history, so the only R-dependent component is the scalar advantage.

By the data processing inequality:

$$I(G; R|\text{history}) \leq I((S, \mathrm{Adv}); R|\text{history}) = I(\mathrm{Adv}; R|S, \text{history}) \leq H(\mathrm{Adv}).$$

If we quantize the advantage into B bins, then $H(\mathrm{Adv}) \lesssim \log(B)$. That is, the number of bits of useful information gleaned per episode is $O(1)$, independent of model size. These bits tell us which member of a discrete set of reward functions (or, equivalently,

optimal-policy classes) we're in. This analysis of mutual information mirrors what's used in some theoretical analysis of optimization algorithms.[26] Note that this estimate is an *upper* bound on the information absorbed by training; the actual amount learned will depend on the policy initialization and other details. For example, if we initialize with a policy that doesn't get any reward, then the entropy of the advantage is zero (not log(B)), and it won't learn anything.

Compute efficiency advantage of LoRA

Our experiments above measured learning progress against the number of training steps, but we may also be interested in the *compute efficiency* of different methods. We calculate that LoRA takes slightly more than ⅔ of the FLOPs that full fine-tuning does per pass. As a result, it will often outperform FullFT on compute efficiency overall.

We derive this ⅔ ratio by analyzing the FLOPs used in the forward–backward pass on a given weight matrix. These operations account for the vast majority of FLOPs in neural network models. We use the following notation:

- $W \in \mathbb{R}^{N \times N}$ is a weight matrix
- $x \in \mathbb{R}^N$ is an input vector
- $y = Wx \in \mathbb{R}^N$ is an output vector
- $\bar{x}, \bar{y} \in \mathbb{R}^N$ are the gradients of the loss with respect to x and y, computed in the backward pass
- $\bar{W} \in \mathbb{R}^{N \times N}$ is the gradient of the loss with respect to W

Full fine-tuning performs the following operations:

Forward

1. $y = Wx$ (N^2 multiply–adds)

Backward

2. $\bar{x} = W^T \bar{y}$ (N^2 multiply–adds)
3. $\bar{W} \mathrel{+}= x\bar{y}^T$ (N^2 multiply–adds)

The forward pass requires N^2 multiply-adds, and the backward pass requires another $2 \cdot N^2$ for $3N^2$ total. Training, which requires both, thus uses 3 times the FLOPs of forward-only inference.

With LoRA, we replace W by $W + BA$, where $B \in \mathbb{R}^{N\times R}$ and $A \in \mathbb{R}^{R\times N}$, with $R \ll N$. Since we only update $\bar{A}$ and $\bar{B}$, we replace the third step of updating $\bar{W}$ with a much cheaper operation. A and B are $N \cdot R$ matrices, so the full forward-backward computation on each requires $3NR$ multiply-adds instead of $3N^2$ for W. The total for both is $6NR$. We also perform the forward-backward pass on Wx and $\bar{x}$, equivalent to the first two steps of FullFT. The total number of multiply-adds is $2N^2 + 6NR$. With $R \ll N$, this is slightly more than $\frac{2}{3}$ of $3N^2$.

If we plotted LoRA performance over FLOPs[27] instead of training steps, it would show a clear advantage over FullFT.

Open questions

There are several questions related to our results that we would love to see investigated in the future:

- Sharpening our predictions of LoRA performance and the precise conditions under which it matches full fine-tuning. We have roughly characterized the regime of equal performance and can estimate the required capacity in terms of tokens or episodes, but we can't yet make accurate forecasts.
- Our theoretical understanding of LoRA learning rates and training dynamics is limited. A fuller theory that explains the ratio between LoRA and FullFT learning rates would be valuable.
- How do LoRA variants such as PiSSA[28] perform when measured according to the methodology in this article?
- There are various options for applying LoRA to MoE layers. LoRA users would benefit from an investigation into how well they perform, and how compatible each approach is with methods like tensor parallelism and expert parallelism that are important for large MoE models.

Closing thoughts

At Thinking Machines, we believe in the power of fine-tuning to advance AI usefulness in many domains of expertise. Our interest in LoRA is driven by a goal of making this power widely accessible and easily customizable to specific needs.

Aside from its practical uses, research on LoRA has also led us to deeper investigations of model capacity, dataset complexity, and sample efficiency. Looking at how learning speed and performance depend on capacity provides a lens for studying fundamental questions in machine learning. We look forward to advancing this study in the future.

Acknowledgements

We thank Dan Alexander Biderman, Weizhu Chen, Daniel Han, and Sadhika Malladi for their insightful feedback on an earlier draft of this post.

Citation

Please cite this work as:

```
Schulman, John and Thinking Machines Lab, "LoRA Without Regret",
Thinking Machines Lab: Connectionism, Sep 2025.
```

Or use the BibTeX citation:

```
@article{schulman2025lora,
  author = {John Schulman and Thinking Machines Lab},
  title = {LoRA Without Regret},
  journal = {Thinking Machines Lab: Connectionism},
  year = {2025},
  note = {https://thinkingmachines.ai/blog/lora/},
  doi = {10.64434/tml.20250929},
}
```