Generalization Lecture 1

Regularization and fine-tuning

Compute-efficient pre-training

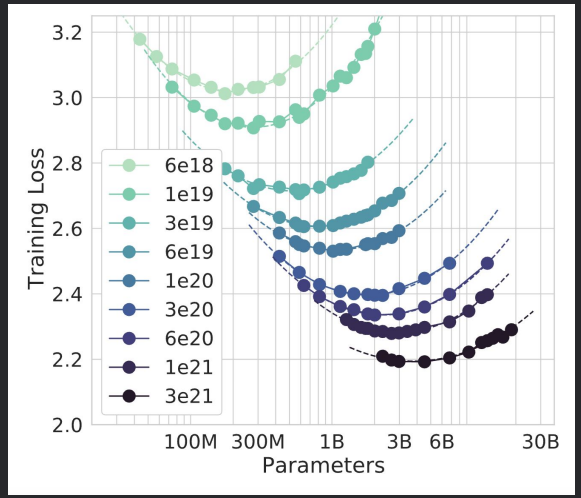
Since pre-training is so expensive, people reason about the best pre-training recipe under compute constraints.

Compute-efficient pre-training

Since pre-training is so expensive, people reason about the best pre-training recipe under compute constraints.

Example 1: Train compute

Chinchilla scaling: for a fixed compute budget, set parameter count N and token count D in a 1:20 ratio with no repetitions



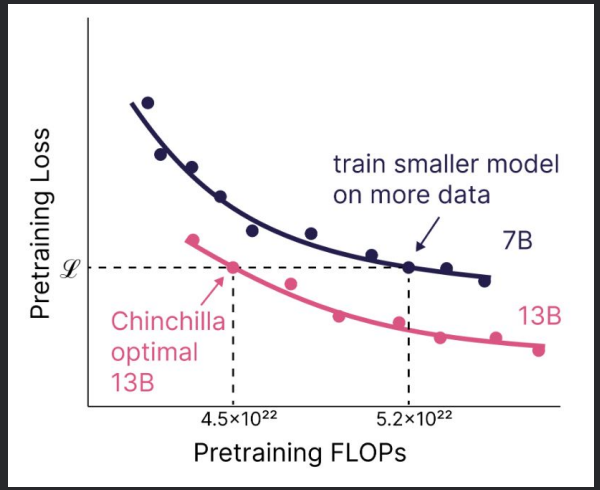
Kaplan et al, 2020 ; Hoffman et al, 2022

Compute-efficient pre-training

Since pre-training is so expensive, people reason about the best pre-training recipe under compute constraints.

Example 2: Inference compute

To lower inference costs with small models, common to over-train models with 1:2000 parameter to token ratio

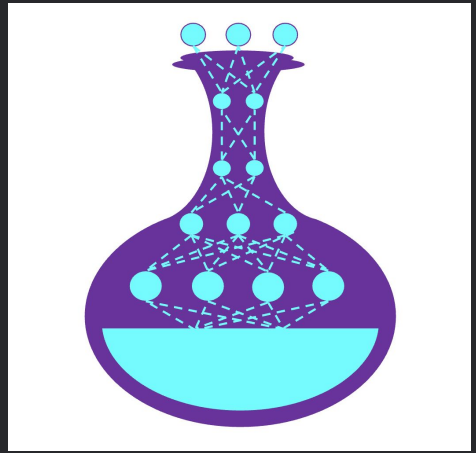


Sardana et al, 2025, Llama, Qwen, ...

Compute-efficient pre-training

Since pre-training is so expensive, people reason about the best pre-training recipe under compute constraints.

Example 3: Deep learning alchemy
We constrain both inference compute to be 30M parameters and train compute to be proportional to 600M tokens.



Infinite data regime

When we think about compute efficiency, we're implicitly assuming access to infinite data. In this regime, train loss is the same as test loss, we can not overfit to the train set, and we must generalize to the test set



Many domains are naturally data-constrained

Beyond text, many distributions genuinely have few samples



Agents (i.e. web navigation trajectories)



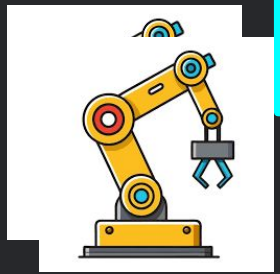
Finance (i.e. stock prices)



Biology (i.e. DNA sequences)



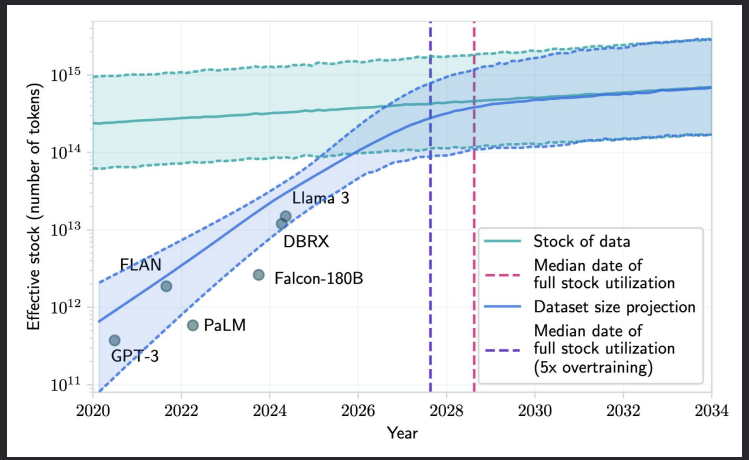
Personalization (i.e. personal information)



Robotics (i.e. robot trajectories)

Same fate for text

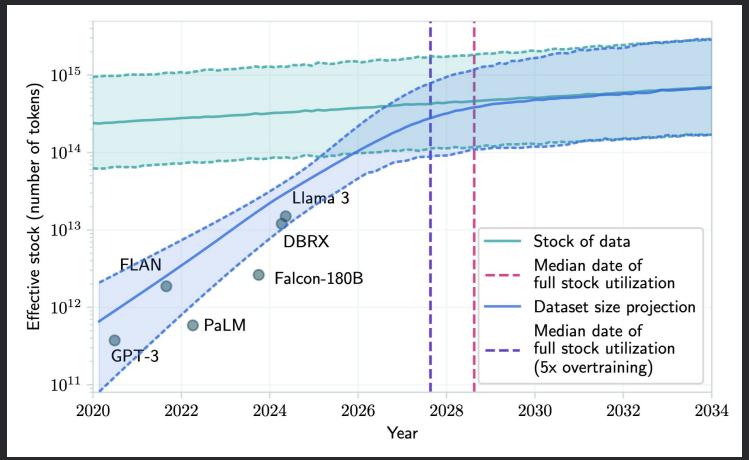
The amount of data available on the internet grows quite slowly (approximately 3% per year), and we can not assume that we will always have access to more.



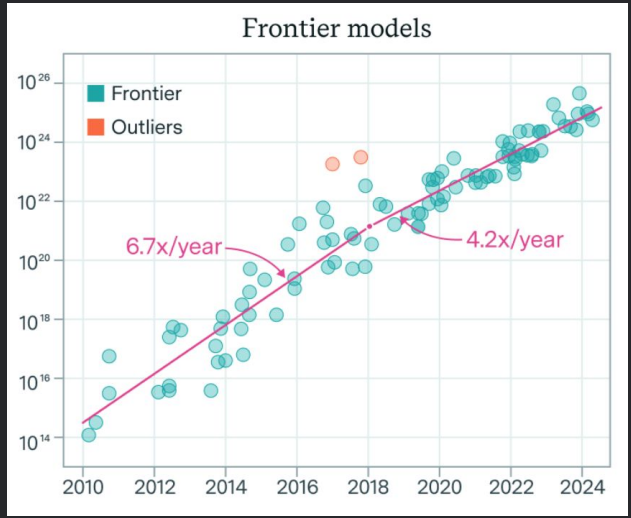
[Villalobos et al, 2024]

Observation: compute continues to increase

On the other hand, compute continues to rapidly increase, with pre-training spending 5x more compute per year



Villalobos et al, 2024

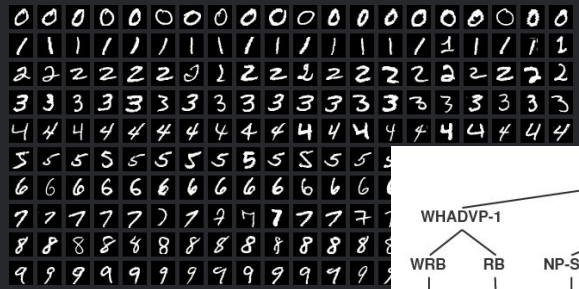


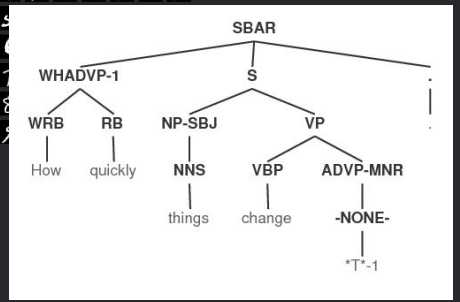
Sevilla and Roldán, 2024

How should one approach training when constrained by data and willing to spend compute?

Classical machine learning

This is closer to ML before the modern LLM era. Empirically, benchmarks such as MNIST and Penn Tree Bank often had few samples. Unfortunately, they didn't have as strong empirical rigor for scaling





Model	#Param	Validation	Test
Mikolov & Zweig (2012) – RNN-LDA + KN-5 + cache	9M‡	-	92.0
Zaremba et al. (2014) – LSTM	20M	86.2	82.7
Gal & Ghahramani (2016) – Variational LSTM (MC)	20M	-	78.6
Kim et al. (2016) – CharCNN	19M	-	78.9
Merity et al. (2016) – Pointer Sentinel-LSTM	21M	72.4	70.9
Grave et al. (2016) – LSTM + continuous cache pointer†	-	-	72.1
Inan et al. (2016) – Tied Variational LSTM + augmented loss	24M	75.7	73.2
Zilly et al. (2016) – Variational RHN	23M	67.9	65.4
Zoph & Le (2016) – NAS Cell	25M	-	64.0
Melis et al. (2017) – 2-layer skip connection LSTM	24M	60.9	58.3
Merity et al. (2017) – AWD-LSTM w/o finetune	24M	60.7	58.8
Merity et al. (2017) – AWD-LSTM	24M	60.0	57.3
Ours – AWD-LSTM-MoS w/o finetune	22M	58.08	55.97
Ours – AWD-LSTM-MoS	22M	**56.54**	**54.44**
Merity et al. (2017) – AWD-LSTM + continuous cache pointer†	24M	53.9	52.8
Krause et al. (2017) – AWD-LSTM + dynamic evaluation†	24M	51.6	51.1
Ours – AWD-LSTM-MoS + dynamic evaluation†	22M	**48.33**	**47.69**

Table 1: Single model perplexity on validation and test sets on Penn Treebank. Baseline results are obtained from Merity et al. (2017) and Krause et al. (2017). † indicates using dynamic evaluation.

Model	#Param	Validation	Test
Inan et al. (2016) – Variational LSTM + augmented loss	28M	91.5	87.0
Grave et al. (2016) – LSTM + continuous cache pointer†	-	-	68.9
Melis et al. (2017) – 2-layer skip connection LSTM	24M	69.1	65.9
Merity et al. (2017) – AWD-LSTM w/o finetune	33M	69.1	66.0
Merity et al. (2017) – AWD-LSTM	33M	68.6	65.8
Ours – AWD-LSTM-MoS w/o finetune	35M	66.01	63.33
Ours – AWD-LSTM-MoS	35M	**63.88**	**61.45**
Merity et al. (2017) – AWD-LSTM + continuous cache pointer †	33M	53.8	52.0
Krause et al. (2017) – AWD-LSTM + dynamic evaluation†	33M	46.4	44.3
Ours – AWD-LSTM-MoS + dynamical evaluation†	35M	**42.41**	**40.68**

Table 2: Single model perplexity over WikiText-2. Baseline results are obtained from Merity et al. (2017) and Krause et al. (2017). † indicates using dynamic evaluation.

Classical machine learning

From many experiments over history, we have developed a few principles for how we can extract value from limited data

1. Regularization (statistics)
2. Sharpness (optimization)
3. Description length (architecture)
4. Conditioning on less related data (fine-tuning, in-context learning)
5. Procuring more related data (augmentation, filtering)

Classical machine learning

From many experiments over history, we have developed a few principles for how we can extract value from limited data

1. **Regularization (statistics)**
2. Sharpness (optimization)
3. Description length (architecture)
4. **Conditioning on less related data (fine-tuning, in-context learning)**
5. Procuring more data (augmentation, filtering)

We will cover perspectives 1 and maybe 4 today and 2, 3, and 5 next week

Regularization concepts

Perspective from statistics - Bias Variance

Simplest example of generalization in statistics comes from linear regression (studied in Belkin et al, 2018, Hastie et al, 2021)

For tractable analysis, assume n data points come from:

$$\text{arbitrary } w \text{ in } \mathbb{R}^p, \qquad x^{(i)} \sim N(0, I_p), \qquad \varepsilon^{(i)} \sim N(0, \sigma^2), \qquad y^{(i)} = w^T x^{(i)} + \varepsilon^{(i)}$$

When $p < n$, the linear solution that minimizes mean squared error is given by $w^* = (X^T X)^{-1} X^T y$. This is called Ordinary Least Squares

Perspective from statistics - Bias Variance

When $p > n$, there are multiple solutions that would interpolate the train set and get zero loss. How can we know what will perform best on the test set?

Given no other information, we might want to pick the simplest solution. One operationalization is smallest norm

Perspective from statistics - Bias Variance

Interestingly, gradient descent, naturally converges to the minimum norm interpolator of the data points. *Inductive bias* or *implicit regularization* of the algorithm

$$\hat{\beta} = \arg\min \left\{ \|b\|_2 : b \text{ minimizes } \|y - Xb\|_2^2 \right\} = (X^T X)^+ X^T y$$

Proposition 1. *Initialize* $\beta^{(0)} = 0$*, and consider running gradient descent on the least squares loss, yielding iterates*

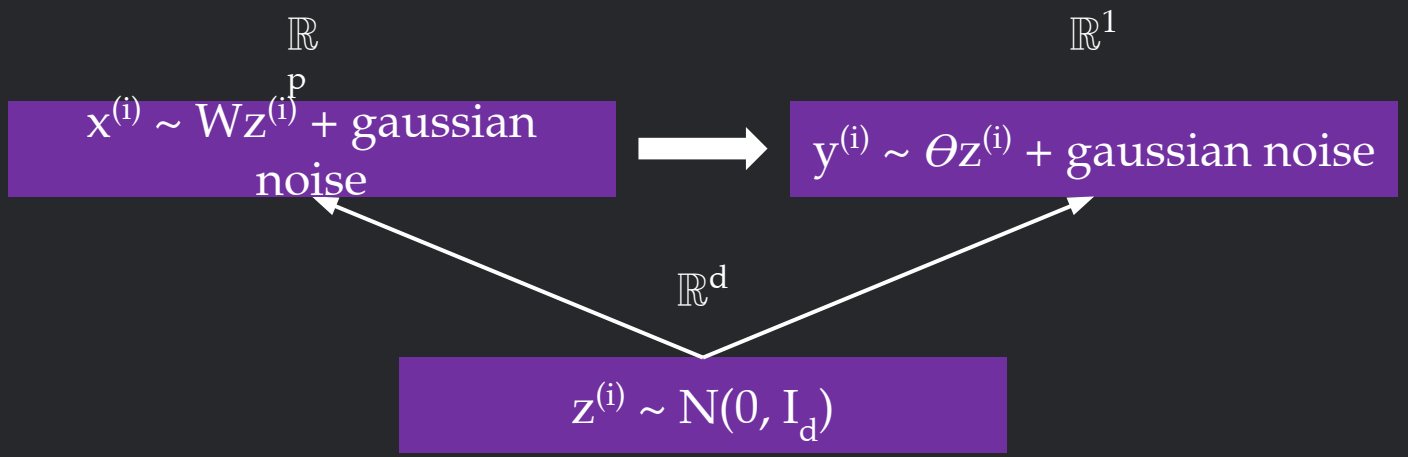
$$\beta^{(k)} = \beta^{(k-1)} + tX^T(y - X\beta^{(k-1)}), \quad k = 1, 2, 3, \ldots,$$

where we take $0 < t \leq 1/\lambda_{\max}(X^T X)$ *(and* $\lambda_{\max}(X^T X)$ *is the largest eigenvalue of* $X^T X$*). Then* $\lim_{k\to\infty} \beta^{(k)} = \hat{\beta}$*, the min-norm least squares solution in* (4)*.*

Proof. The choice of step size guarantees that $\beta^{(k)}$ converges to a least squares solution as $k \to \infty$, call it $\tilde{\beta}$. Note that $\beta^{(k)}$, $k = 1, 2, 3, \ldots$ all lie in the row space of X; therefore $\tilde{\beta}$ must also lie in the row space of X; and the min-norm least squares solution $\hat{\beta}$ is the unique least squares solution with this property. □

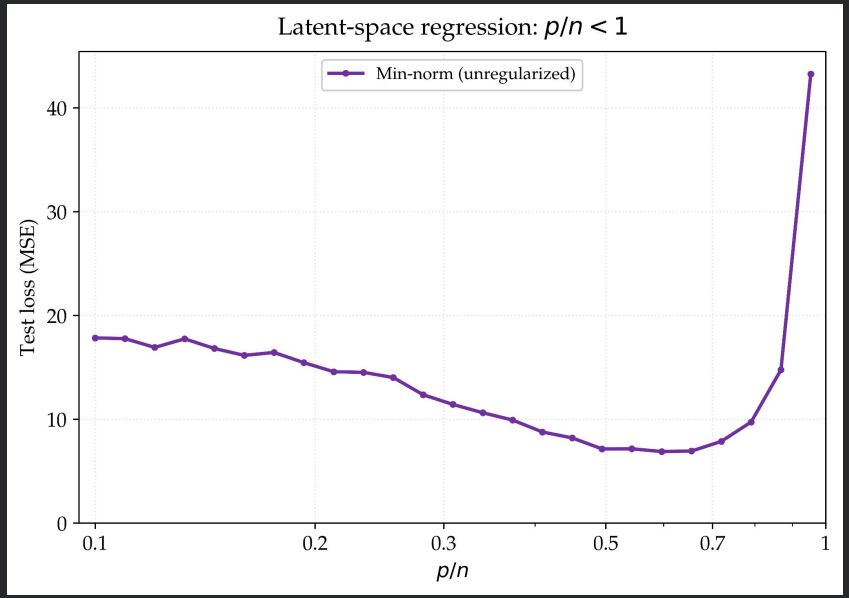
Perspective from statistics - Bias Variance

One drawback of the linear regression model is that the underlying problem dimension changes as we scale p (parameter count). To fix the underlying problem dimension, we introduce the latent space model



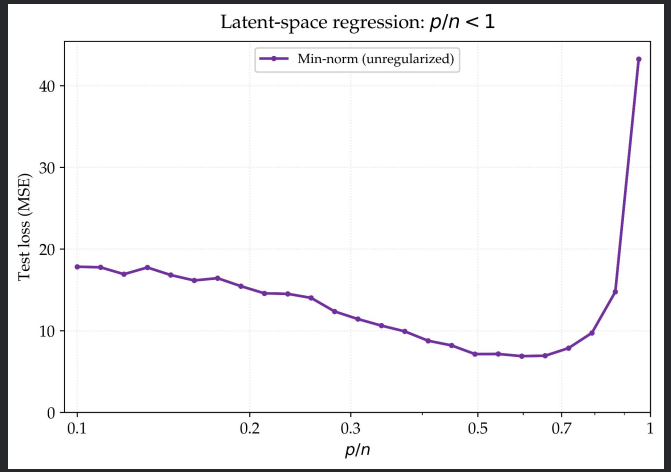
Perspective from statistics - Bias Variance

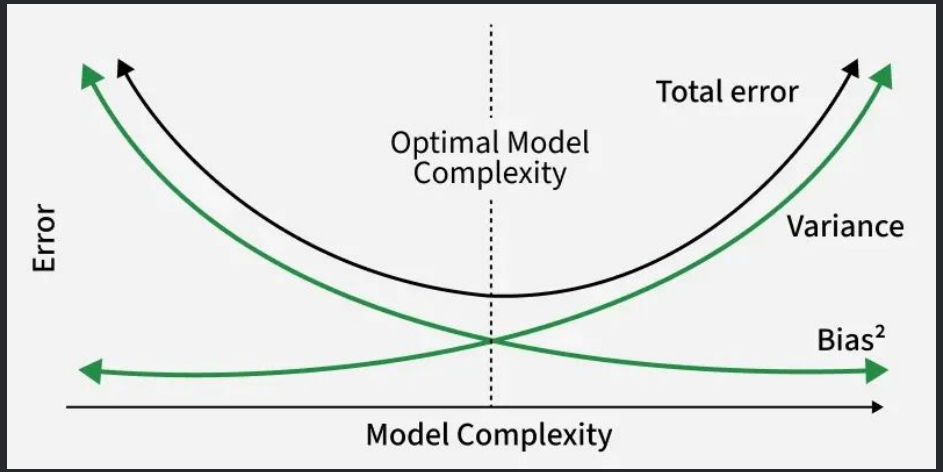
Under latent space, there's an issue where too low of a parameter count means you can't fit the target distribution and too high of a parameter count overfits to the noise in the data



Perspective from statistics - Bias Variance

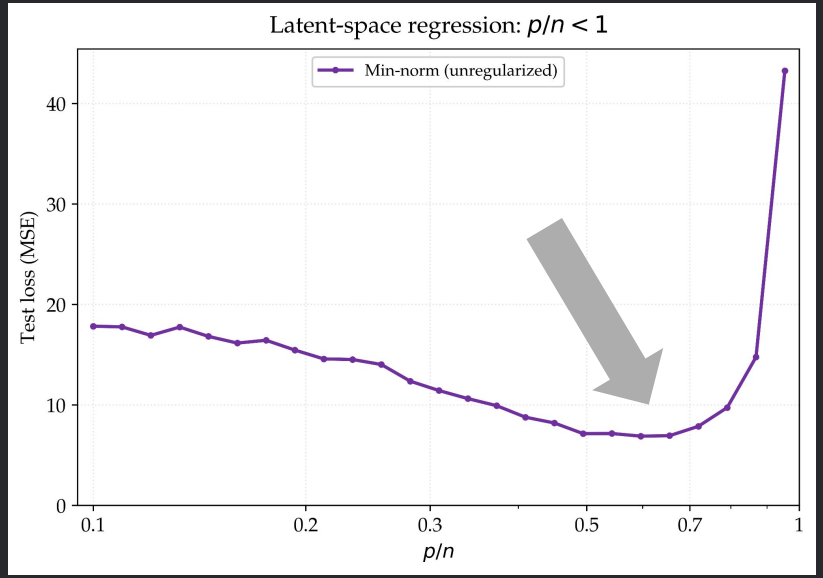
Under latent space, there's an issue where too low of a parameter count means you can't fit the target distribution (high bias^2) and too high of a parameter count overfits to the noise in the data (high variance)





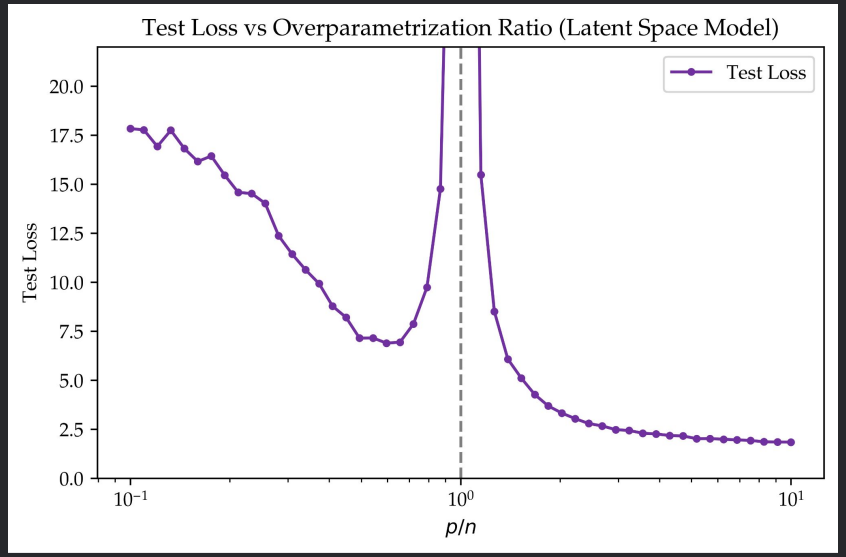
Perspective from statistics - Bias Variance

Therefore, we may regularize by picking an optimal model complexity: not too small, not too big. Many formulations of controlling complexity: PAC learning, VC dimension, Rademacher complexity, etc.



Perspective from statistics - Double Descent

One of the big surprises in deep learning is that bias variance is *not* an issue and you can keep training larger models. Even in our example, the inductive bias of minimum norm leads to benefits from overparameterization.



Perspective from statistics - Explicit Regularization

In the previous case, we relied on the implicit regularization from overparameterization to do well. However, if we are familiar with problem structure, we can derive better algorithms

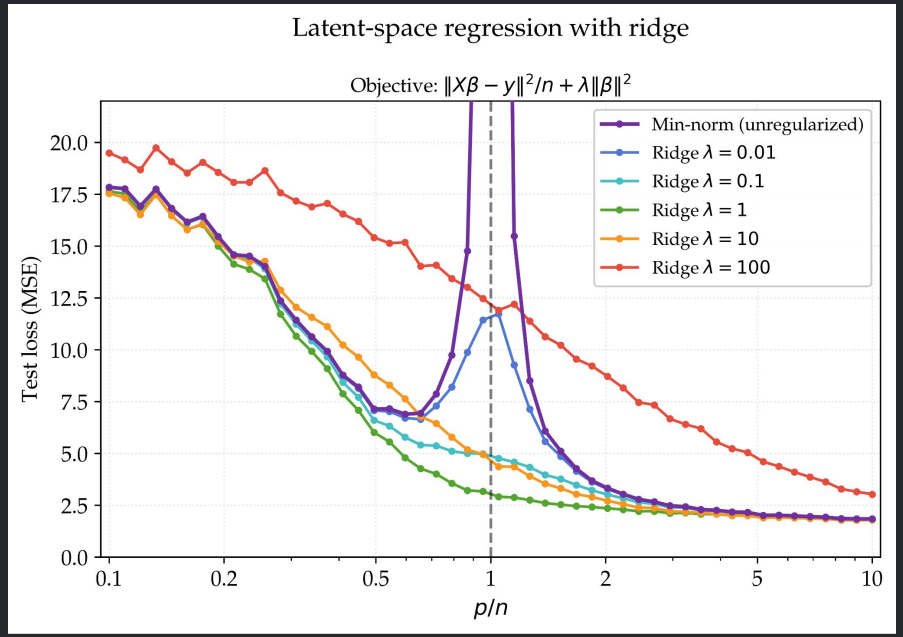
Assume we have a prior that the weights are small:

$$w \sim N(0, \tau^2 I_p), \qquad x^{(i)} \sim N(0, I_p), \qquad \varepsilon^{(i)} \sim N(0, \sigma^2), \qquad y^{(i)} = w^T x^{(i)} + \varepsilon^{(i)}$$

The optimal solution is now $w^* = (X^T X + (\sigma^2/\tau^2) I_p)^{-1} X^T y$. This is called ridge regression and is uniquely defined for all parameter counts

Perspective from statistics - Explicit Regularization

When we tune the strength of ridge regression, bias variance and double descent goes away and we simply get monotone scaling



Transferring to empirics

Since we don't know the distribution, we don't know the right complexity measure to regularize (parameter count and norm for linear regression). Nonetheless, we have two lessons

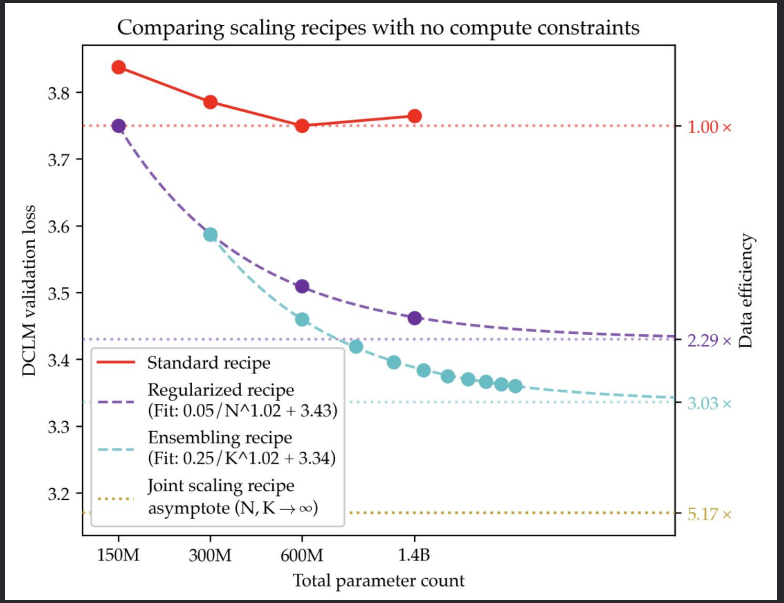
- Lesson 1: Neural network training may have some inductive bias that minimizes the right notion of complexity and improves generalization
- Lesson 2: We have to identify explicit forms of regularization that can improve performance and scaling behavior

Regularization empirics

Empirical investigation of regularization

How far can regularization take us?

We will follow Pre-training under infinite compute to see different regularization baselines. We will specifically develop scaling laws for how loss changes as compute increases (and approaches infinity)



Setting

We will operate under a limit of 200M training tokens from DCLM with a held-out validation set (Note: your assignment will be on DNA, not DCLM).

We will use a recipe similar to class (transformer, AdamW, etc.)

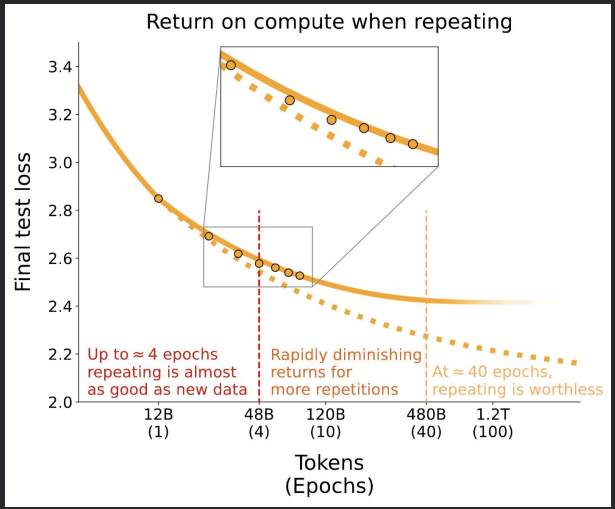
Since we are assuming infinite compute, we train much larger models than normal, defaulting to a N = 300M parameter model (30x larger than Chinchilla)

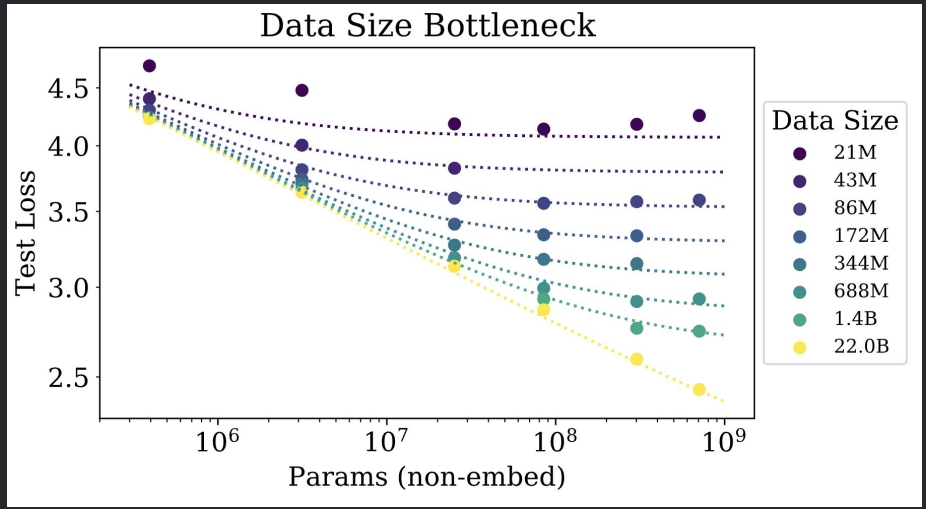
1. Current recipes

how well do existing methods handle data constraints?

1. Current recipes

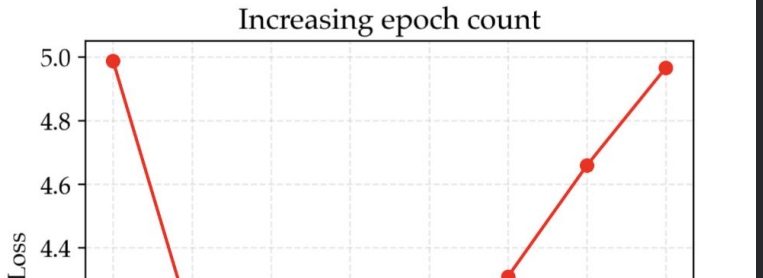
We start by considering some existing data-constrained pre-training recipes such as repeating the data [Muennighoff et al, 2023] and increasing parameter count [Kaplan et al, 2020 ; Hoffman et al, 2022]





1a. Current recipes: epoching

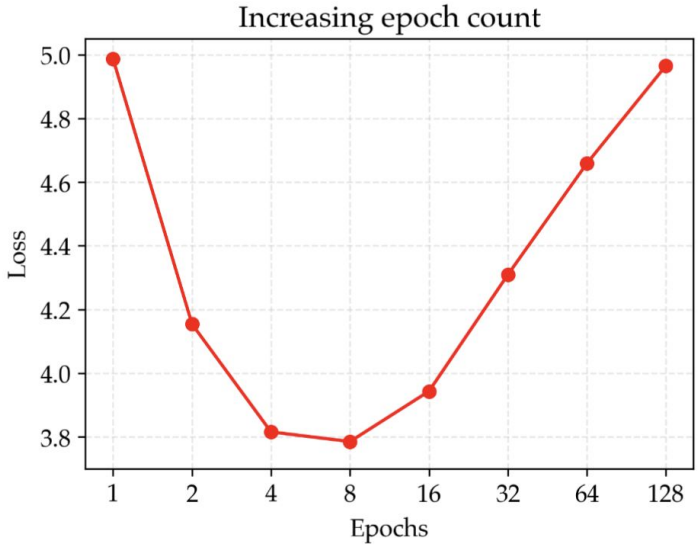
If we repeat the data too many times, we start overfitting



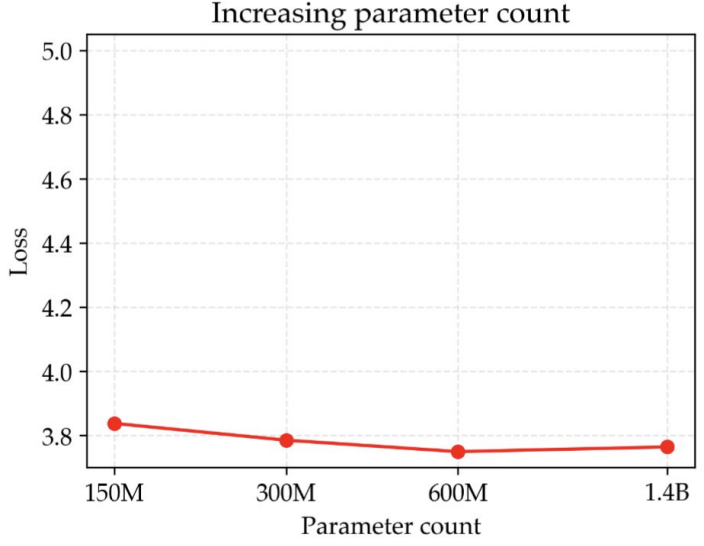
Tuned *H*	**1**	**8**	**128**
Learning rate	1e-3	1e-3	3e-3

1b. Current recipes: epoching + parameter scaling

Increasing the parameter count isn't very helpful, even after tuning epoch count



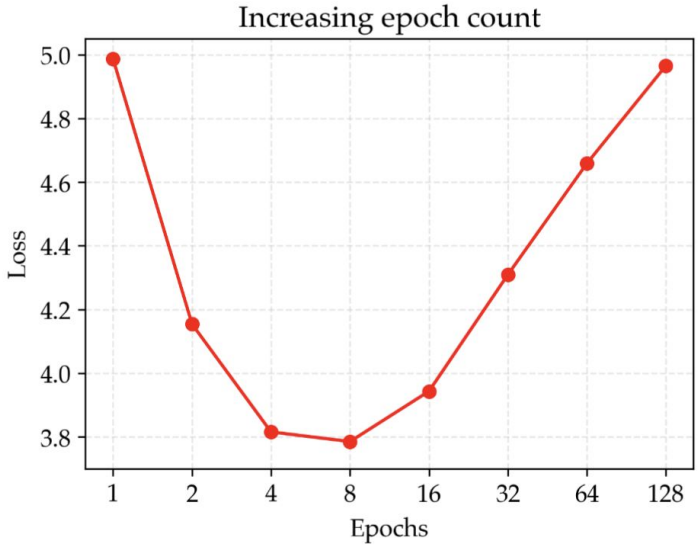
Tuned H	1	8	128
Learning rate	1e-3	1e-3	3e-3



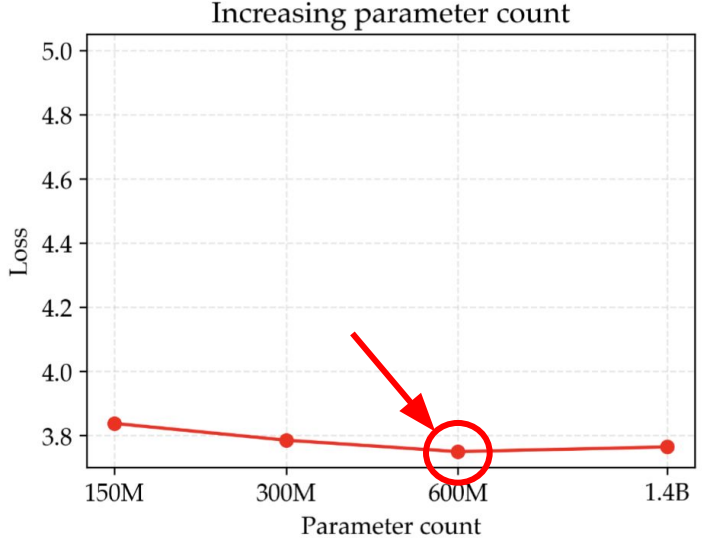
Tuned H	150M	300M	600M	1.4B
Learning rate	3e-3	1e-3	1e-3	3e-4
Epoch count	8	8	4	4

1b. Current recipes: epoching + parameter scaling

Is this the best possible performance under infinite compute?



Tuned H	1	8	128
Learning rate	1e-3	1e-3	3e-3



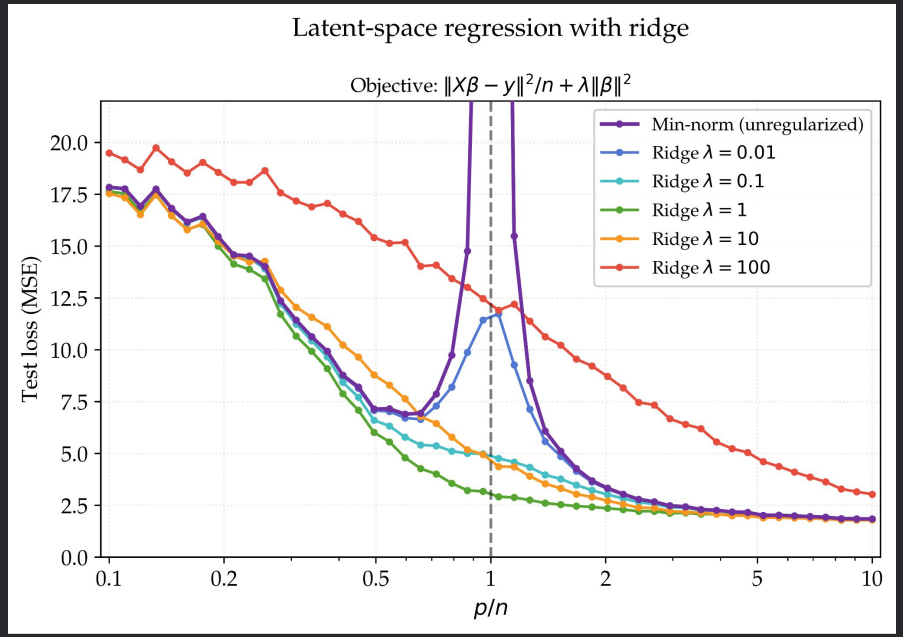
Tuned H	150M	300M	600M	1.4B
Learning rate	3e-3	1e-3	1e-3	3e-4
Epoch count	8	8	4	4

2. Regularized parameter scaling

can we get stronger/monotone/clean scaling?

2. Regularized parameter scaling

Recall our intuition that we simply need to regularize the norm of our model to prevent overfitting [Nakkiran et al, 2021]

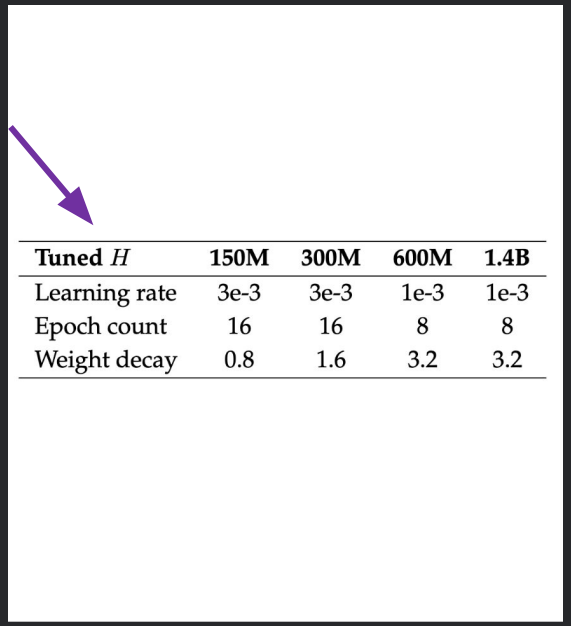


2. Regularized parameter scaling

We find that to get the best possible performance from these over-parameterized, epoched models, it is critical to regularize pre-training with much higher weight decay.

2. Regularized parameter scaling

We first jointly tune learning rate, epoch count, and weight decay for each parameter count by performing an expensive search for locally optimal hyper-parameters



Tuned H	150M	300M	600M	1.4B
Learning rate	3e-3	3e-3	1e-3	1e-3
Epoch count	16	16	8	8
Weight decay	0.8	1.6	3.2	3.2

2. Regularized parameter scaling

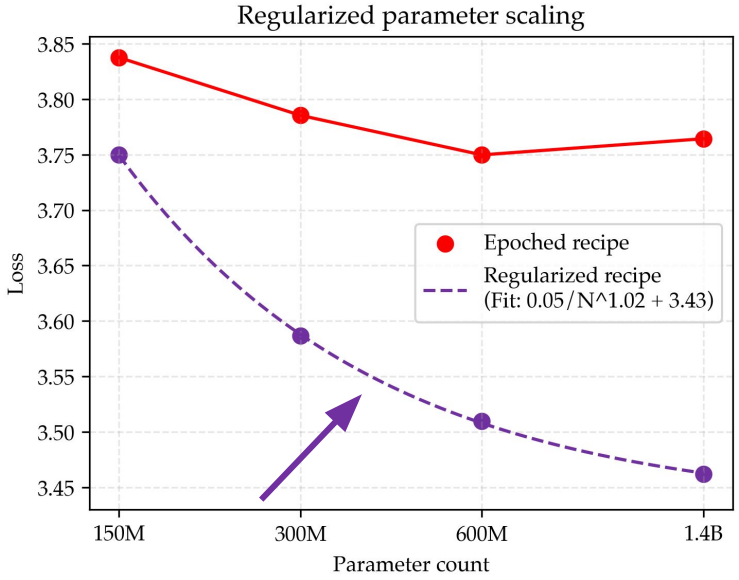
The optimal weight decay can be **over 30 times** higher than standard practice of 0.1, likely because our models are now larger than the data and we epoch

Tuned H	**150M**	**300M**	**600M**	**1.4B**
Learning rate	3e-3	3e-3	1e-3	1e-3
Epoch count	16	16	8	8
Weight decay	0.8	1.6	3.2	3.2



2. Regularized parameter scaling

After regularization, our loss follows clean power law scaling!

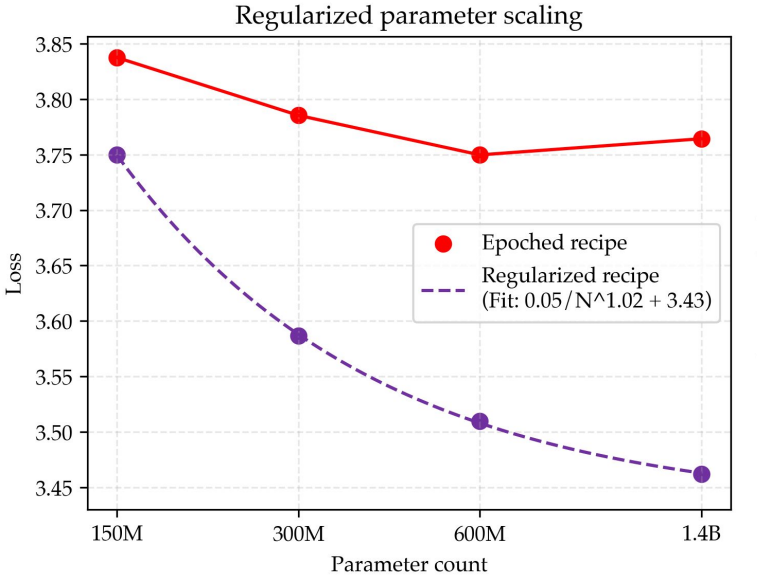


$$L = 0.05 / N^{1.02} + 3.43$$

Tuned H	150M	300M	600M	1.4B
Learning rate	3e-3	3e-3	1e-3	1e-3
Epoch count	16	16	8	8
Weight decay	0.8	1.6	3.2	3.2

2. Regularized parameter scaling

Our law has a model scaling exponent that is much faster than Chinchilla (1.02 vs 0.34). Exponent 1 is predicted by data-constrained scaling theory [Bahri et al, 2021]

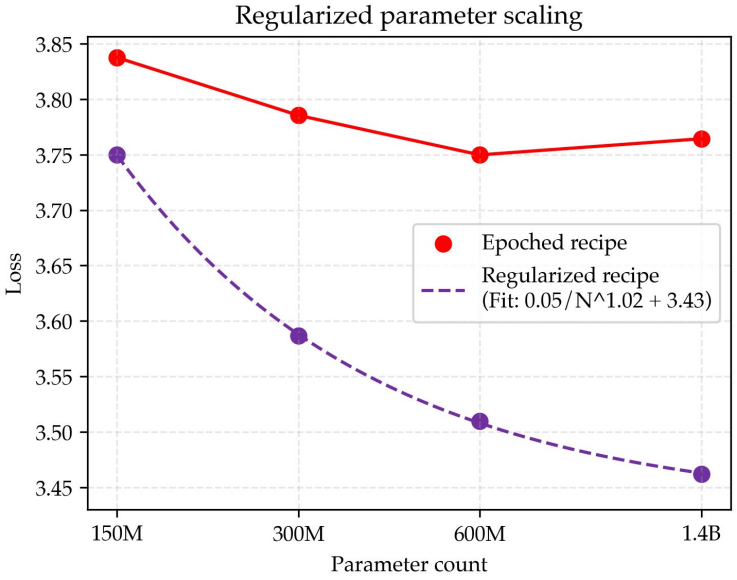


$$L = 0.05 / N^{1.02} + 3.43$$

Tuned H	150M	300M	600M	1.4B
Learning rate	3e-3	3e-3	1e-3	1e-3
Epoch count	16	16	8	8
Weight decay	0.8	1.6	3.2	3.2

2. Exponent scaling intuition

Our law has a model scaling exponent that is much faster than Chinchilla (1.02 vs 0.34)

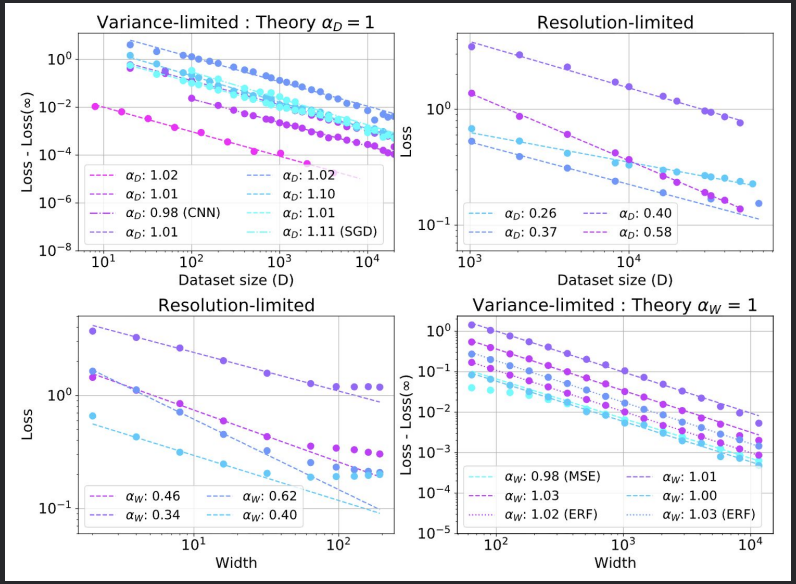


$$L = 0.05 / N^{1.02} + 3.43$$

Tuned H	150M	300M	600M	1.4B
Learning rate	3e-3	3e-3	1e-3	1e-3
Epoch count	16	16	8	8
Weight decay	0.8	1.6	3.2	3.2

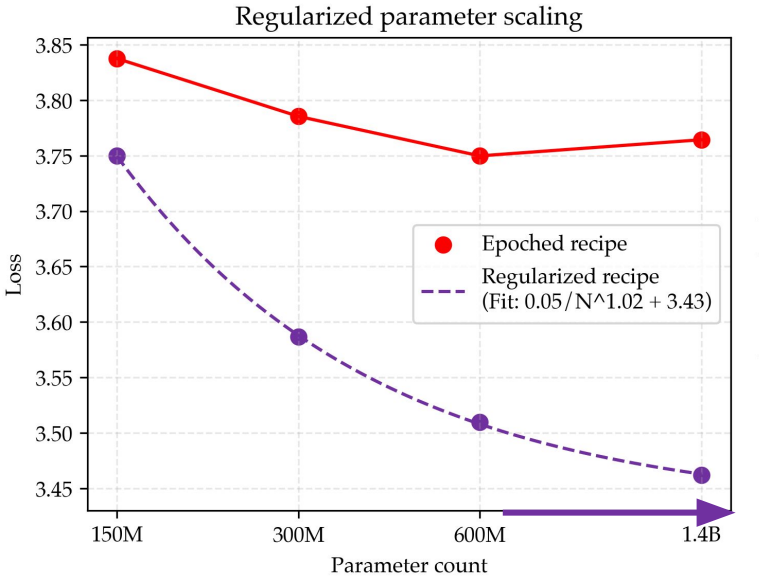
2. Exponent scaling intuition

Exponent 1 is predicted by data-constrained scaling theory [Bahri et al, 2021]. Specifically, when one resource is finite, the other resource scales with exponent 1, and when one is infinite, we expect a non-1 exponent



2. Regularized parameter scaling

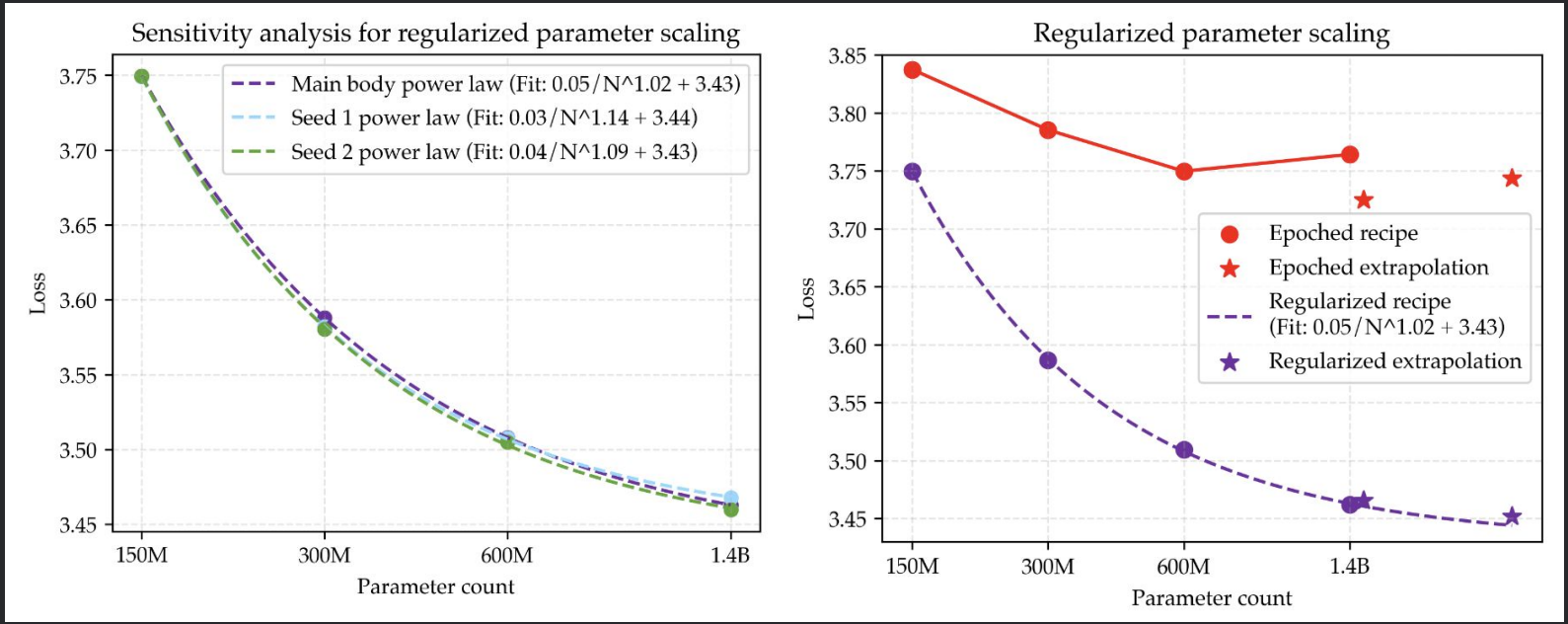
Thanks to our scaling law, we can estimate the best possible performance under infinite compute via the ***asymptote*** under infinite parameter count



$$L = 0.05 / N^{1.02} + 3.43$$

Tuned H	150M	300M	600M	1.4B
Learning rate	3e-3	3e-3	1e-3	1e-3
Epoch count	16	16	8	8
Weight decay	0.8	1.6	3.2	3.2

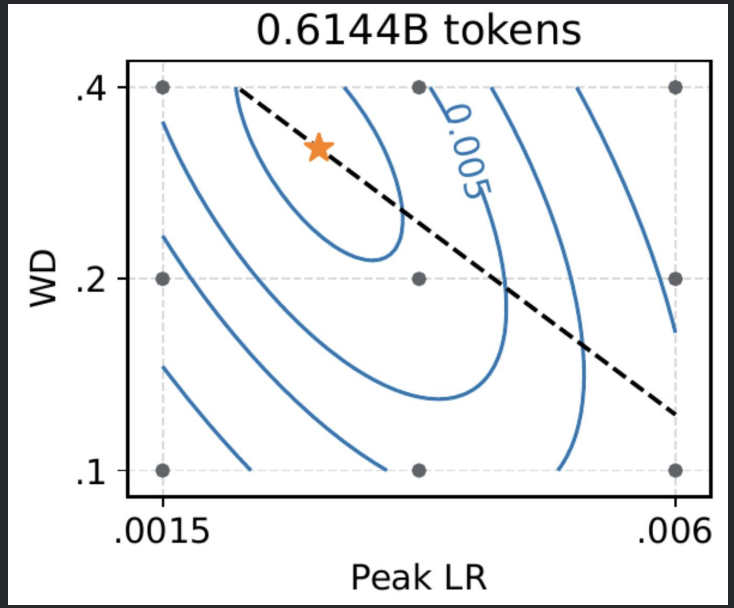
Sensitivity analysis + Extrapolation



2. What is weight decay doing?

From previous assignments, we know that weight decay is secretly just controlling optimization speed via norm. Is weight decay actually regularization?

In the assignment, you will build the relationship between weight decay, learning rate, and loss in the epoched regime. You will also test if Hyperball works well in this regime.



3. Ensemble scaling

can we improve upon the parameter scaling asymptote?

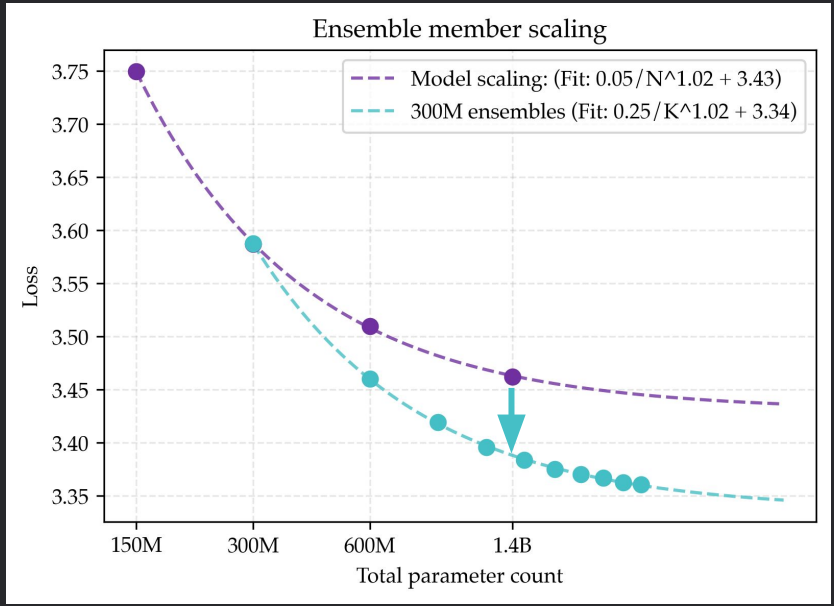
3. Ensemble member scaling

Parameter scaling is only one possible limit, are there others?

We investigate ensembling: instead of training bigger models, we train multiple models with different data orders + initializations and average logits at inference

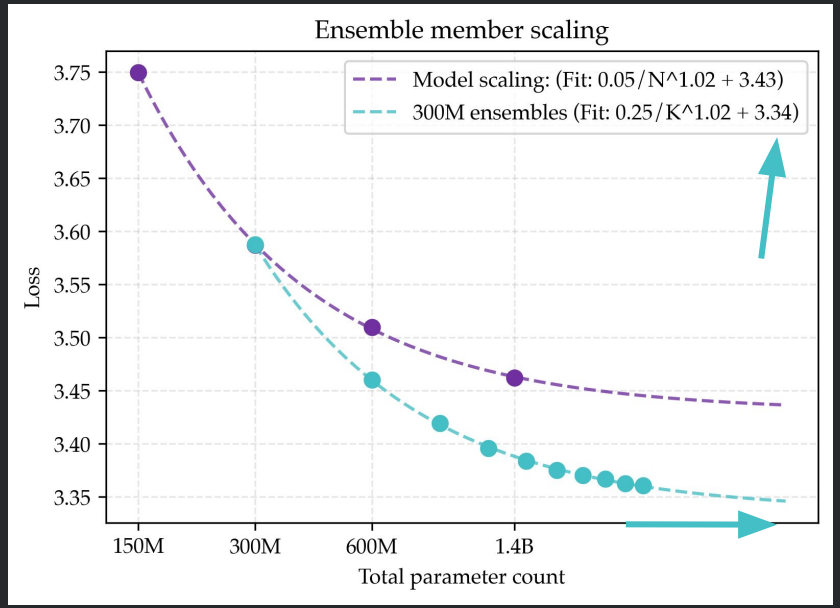
3. Ensemble member scaling: better asymptotes

Increasing ensemble member count out-performs increasing parameter count!
Instead of training bigger models, it's better to train multiple models



3. Ensemble member scaling: better asymptotes

We also get power law scaling for ensembles. The exponents are similar and the asymptote is lower, making it better with infinite compute



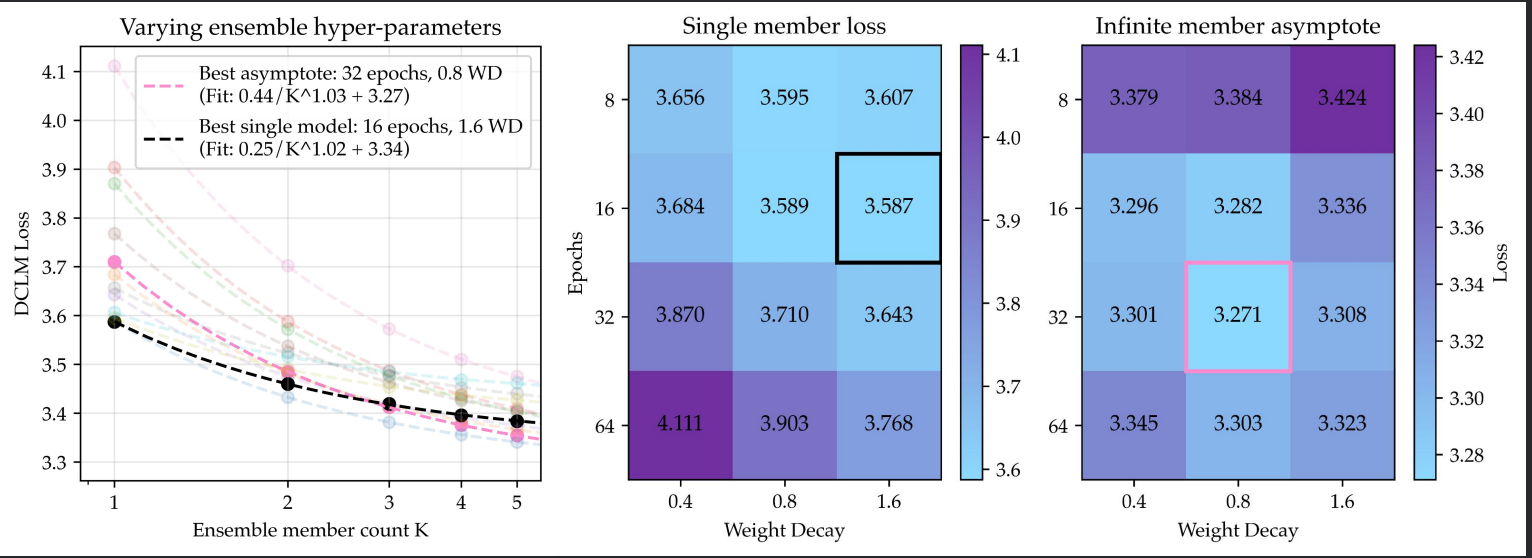
3. Ensemble member scaling: why are ensembles better?

Allen-Zhu and Li, 2023 suggest data is “multi-view”, where one of many different features correctly classifies the data, but the best performance comes from using all the features. Parameter scaling is biased to learning only one view of the data



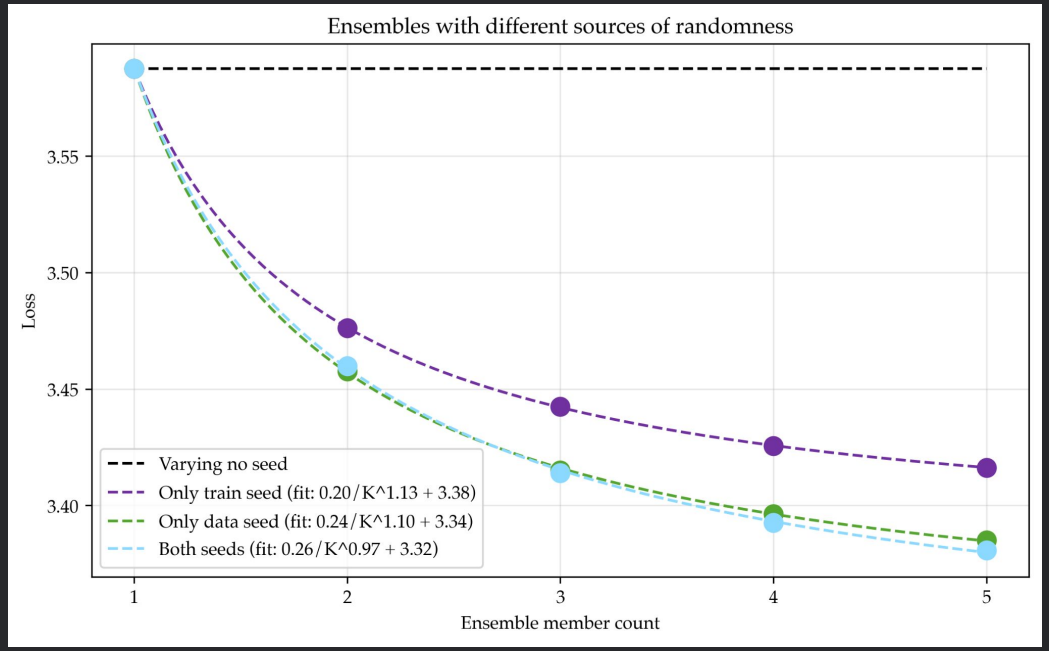
3. Ensemble member scaling: why are ensembles better?

We find evidence for multi-view data when tuning ensemble members: we get better asymptotes when each member is trained with more epochs and less regularization, with each model over-fitting in a different way



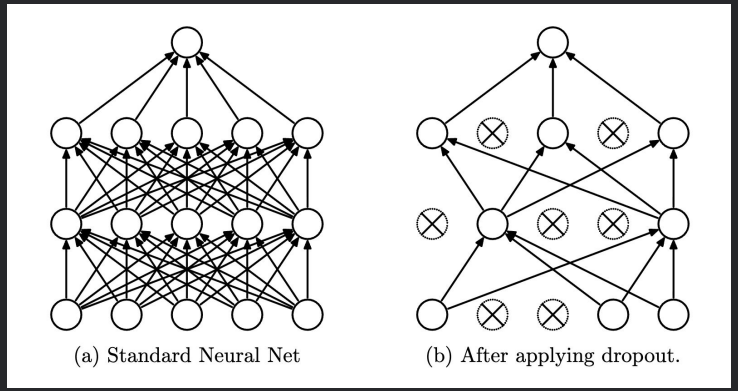
3. Seed science

One source of variance is sufficient for almost all of the benefit!



3. Dropout

Since ensembling is expensive, one can simulate ensembling by randomly masking a fraction of the activations every training step [Hinton et al, 2012]



In the assignment, you will test the difference between dropout and ensembling

4. Composing all recipes

how far can we push data-efficiency with all of our recipes?

4. Composing both ensemble and parameter scaling

Previously, we were comparing parameter and ensemble scaling. However, there is nothing that stops us from composing them. We characterize the best possible performance of both by computing the double limit

4. Composing both ensemble and parameter scaling

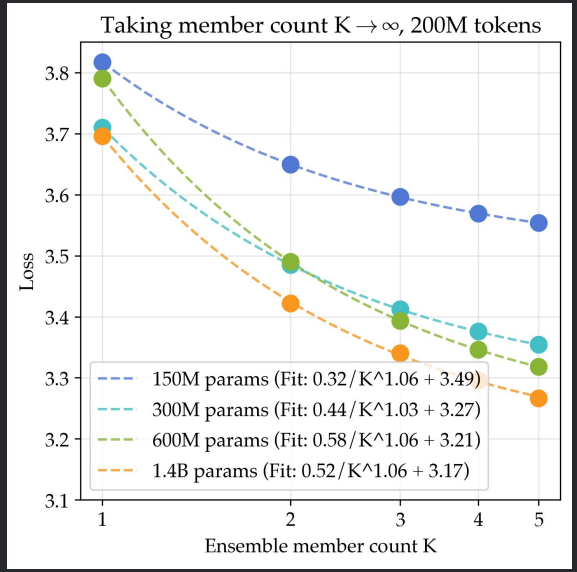
Previously, we were comparing parameter and ensemble scaling. However, there is nothing that stops us from composing them. We characterize the best possible performance of both by computing the double limit

$$\hat{\mathcal{L}}_D = \lim_{N\to\infty} \lim_{K\to\infty} \min_{H} \mathcal{L}\left(\mathcal{E}_{\mathcal{A}}\left(D, N, K, H\right)\right)$$

D tokens
N parameters
K ensemble members
H hyperparams (wd, lr, epochs)

4. Composing both ensemble and parameter scaling

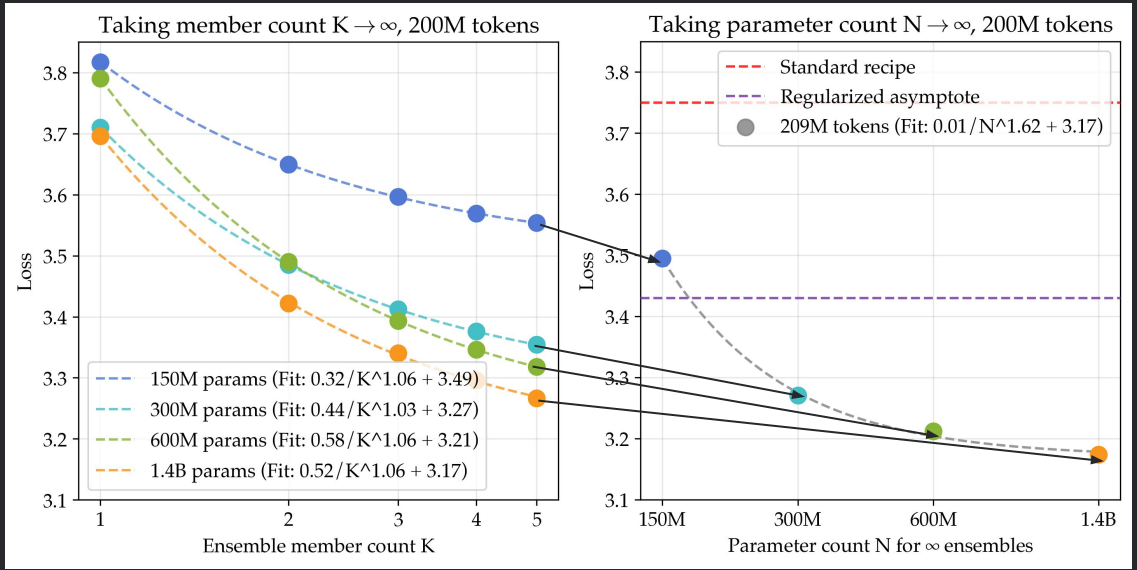
Left: We first send K → ∞ for fixed N, D with hparams selected for the asymptote



4. Composing both ensemble and parameter scaling

Left: We first send K → ∞ for fixed N, D with hparams selected for the asymptote

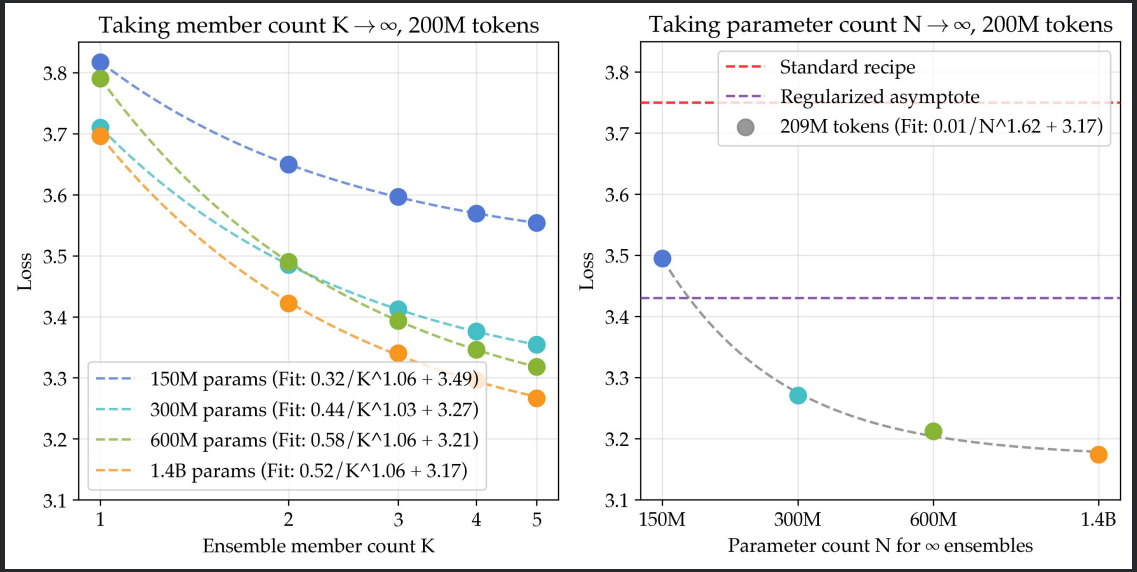
Right: We then send N → ∞ for fixed D and achieve a lower asymptote of 3.17



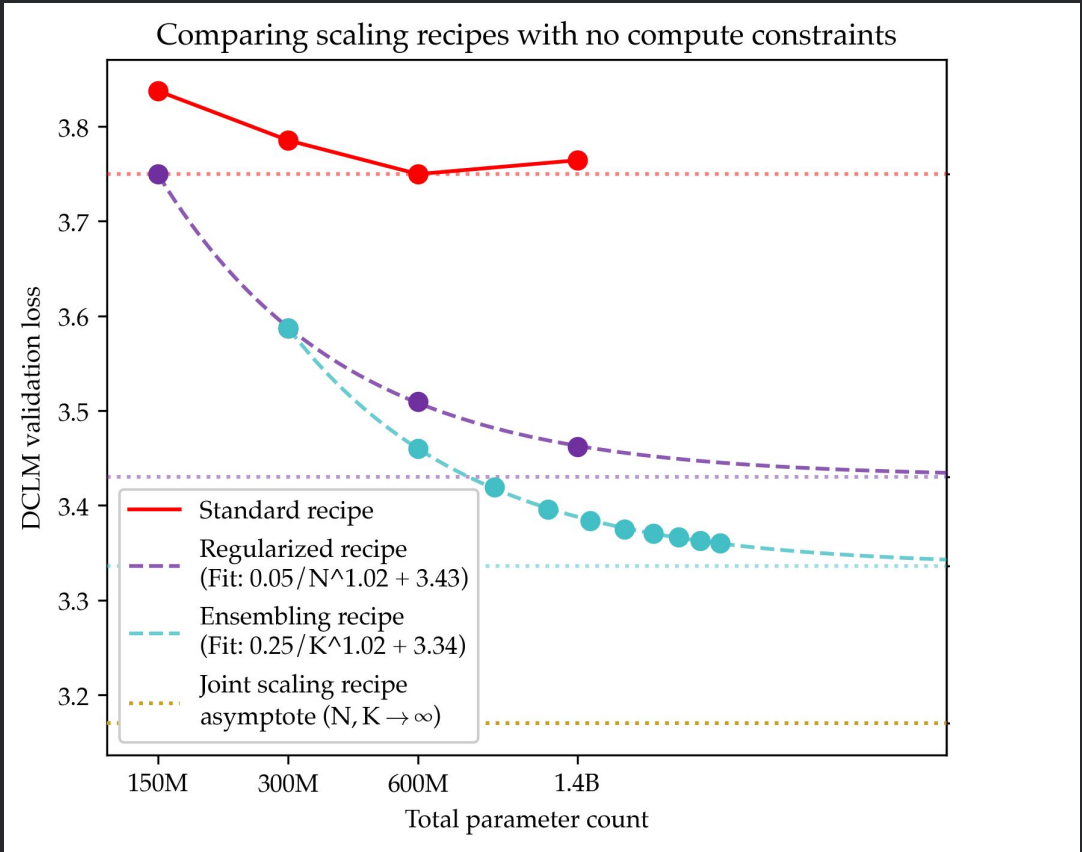
4. Composing both ensemble and parameter scaling

Left: We first send K → ∞ for fixed N, D with hparams selected for the asymptote

Right: We then send N → ∞ for fixed D and achieve a lower asymptote of 3.17



Summary of scaling recipes at 200M tokens



1. Current approaches over-fit

2. Regularization has a scaling law + asymptote

3. Ensembling has a lower asymptote

4. We can compose our recipes

5. Scaling seed token count

do our methods only help at small scale?

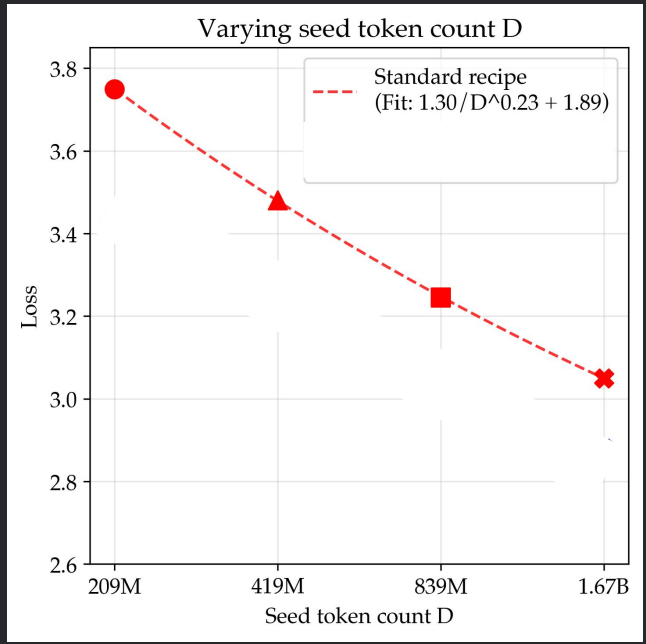
5. Scaling seed token count

Do these interventions only help at 200M tokens, or do they also help at larger token counts?

We scale up our experiments by estimating the best possible loss of epoching, regularizing, and ensembling at each token scale. We can then measure how much better our method is at different token scales.

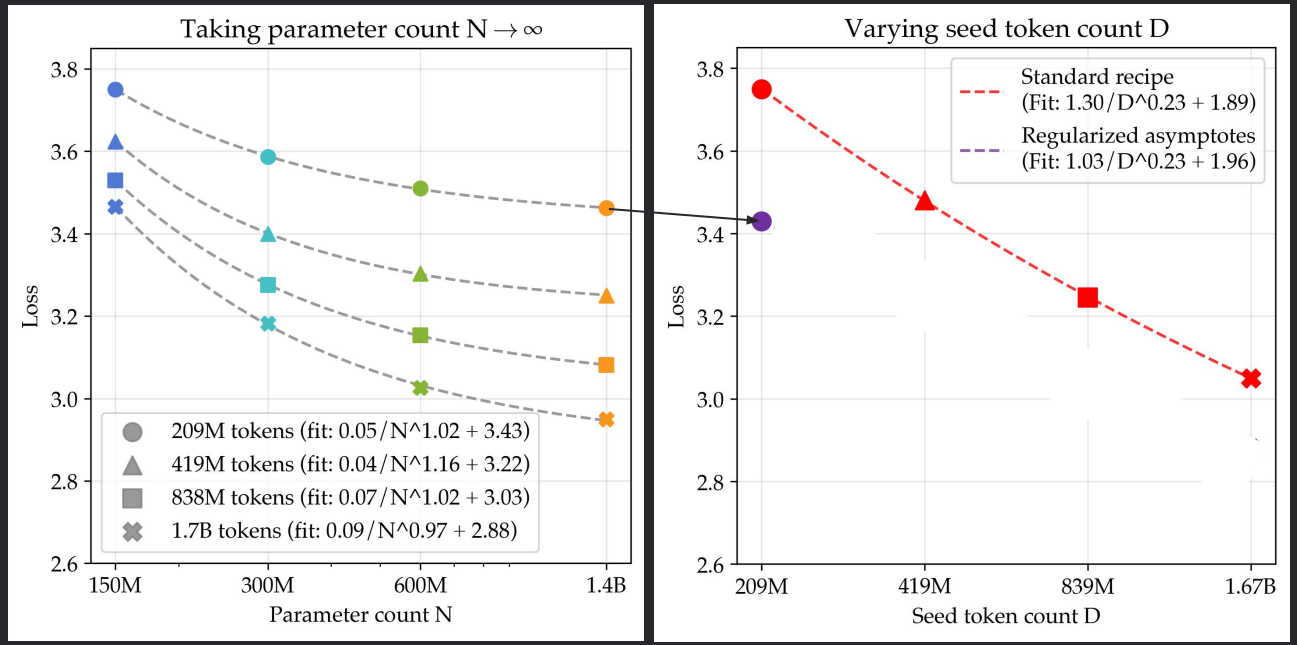
5. Scaling seed token count

For each seed token count, we find the best possible loss of the standard recipe and regularized recipe (asymptote)



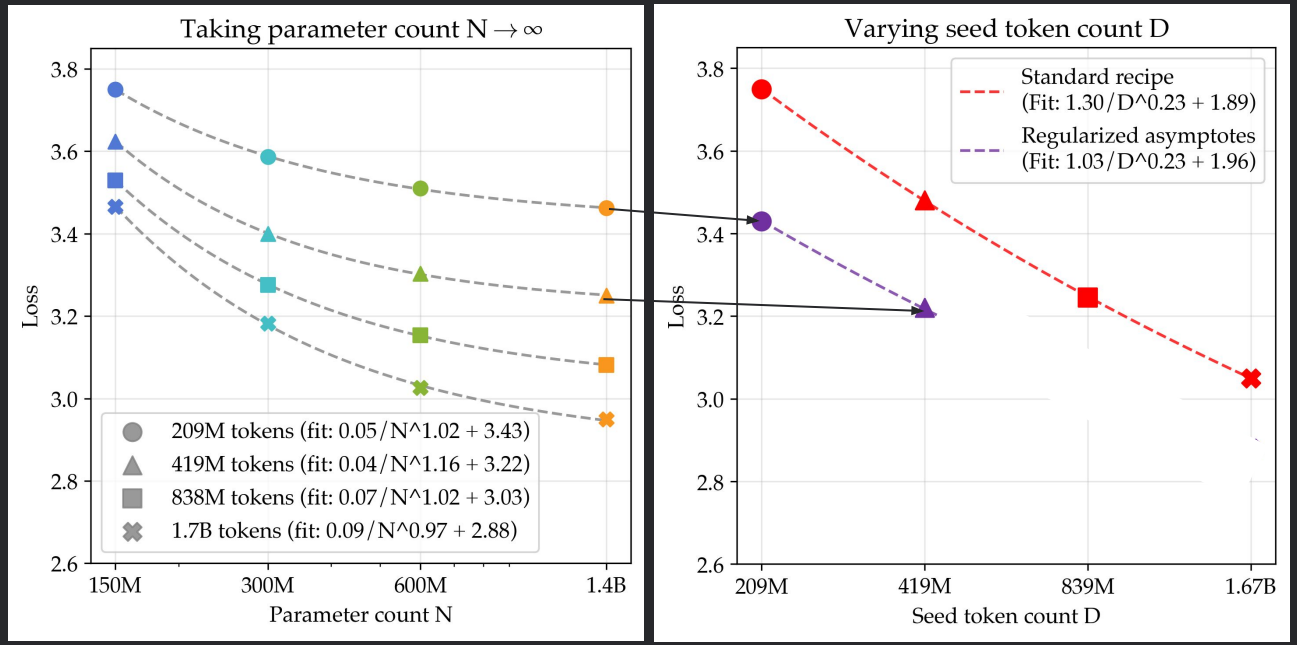
5. Scaling seed token count

For each seed token count, we find the best possible loss of the standard recipe and regularized recipe (asymptote)



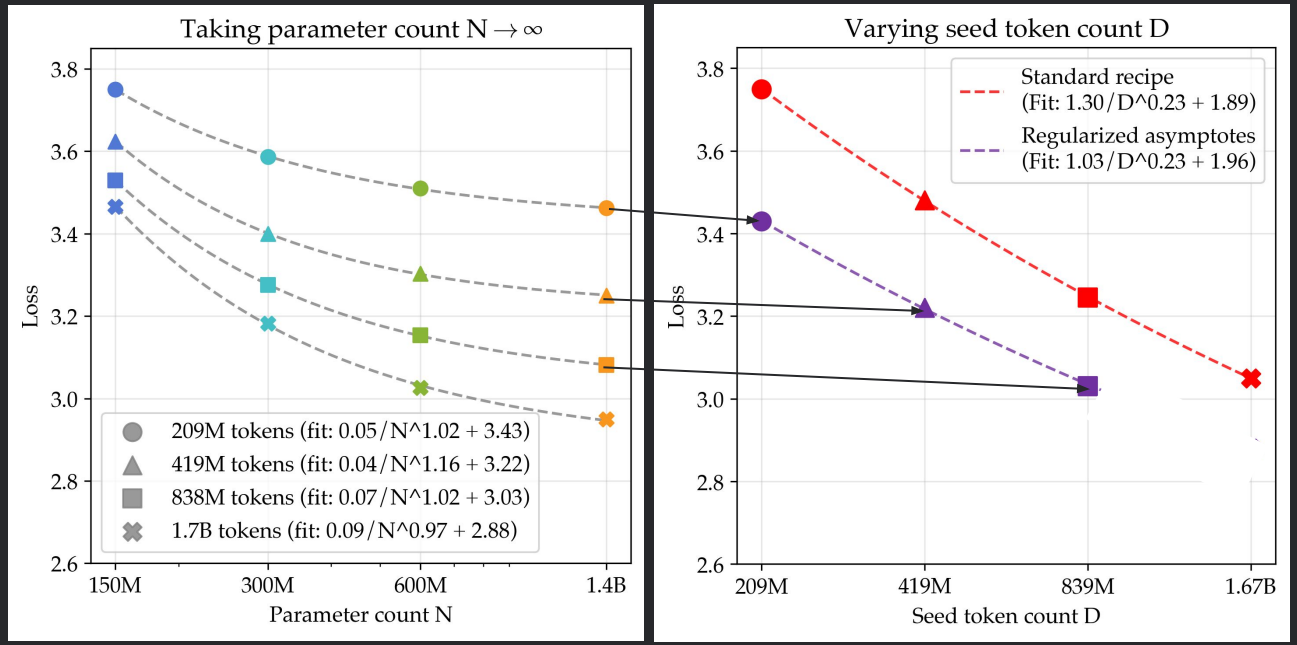
5. Scaling seed token count

For each seed token count, we find the best possible loss of the standard recipe and regularized recipe (asymptote)



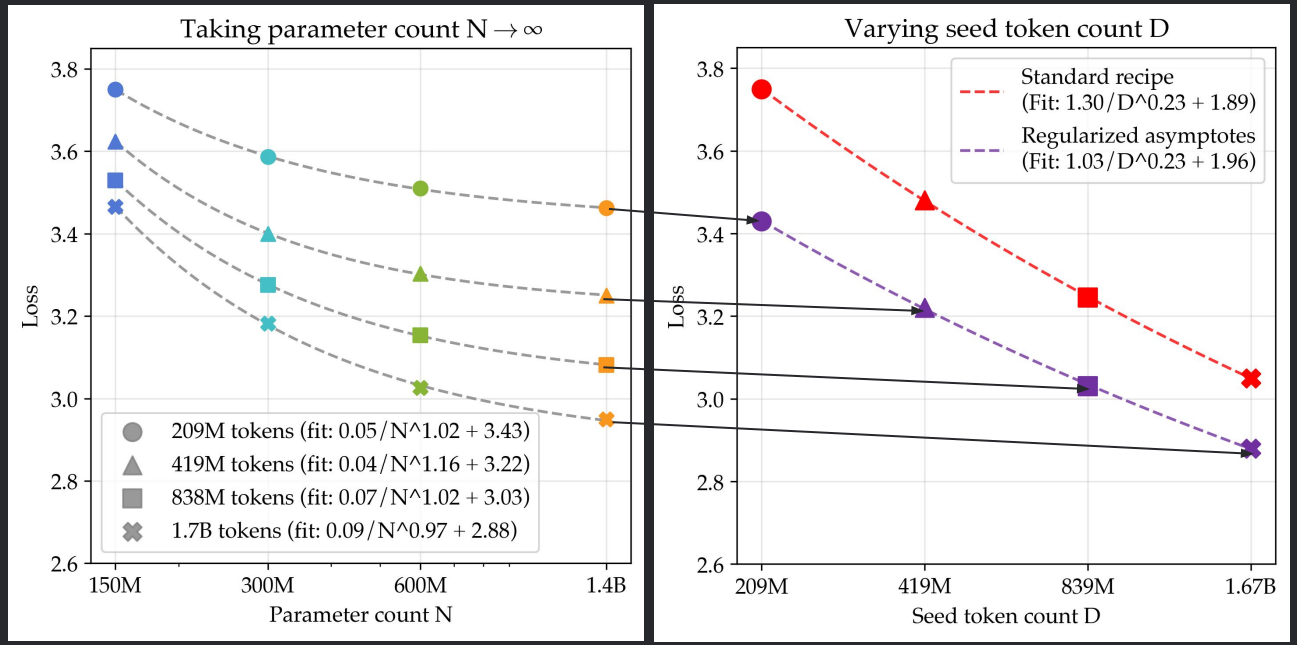
5. Scaling seed token count

For each seed token count, we find the best possible loss of the standard recipe and regularized recipe (asymptote)



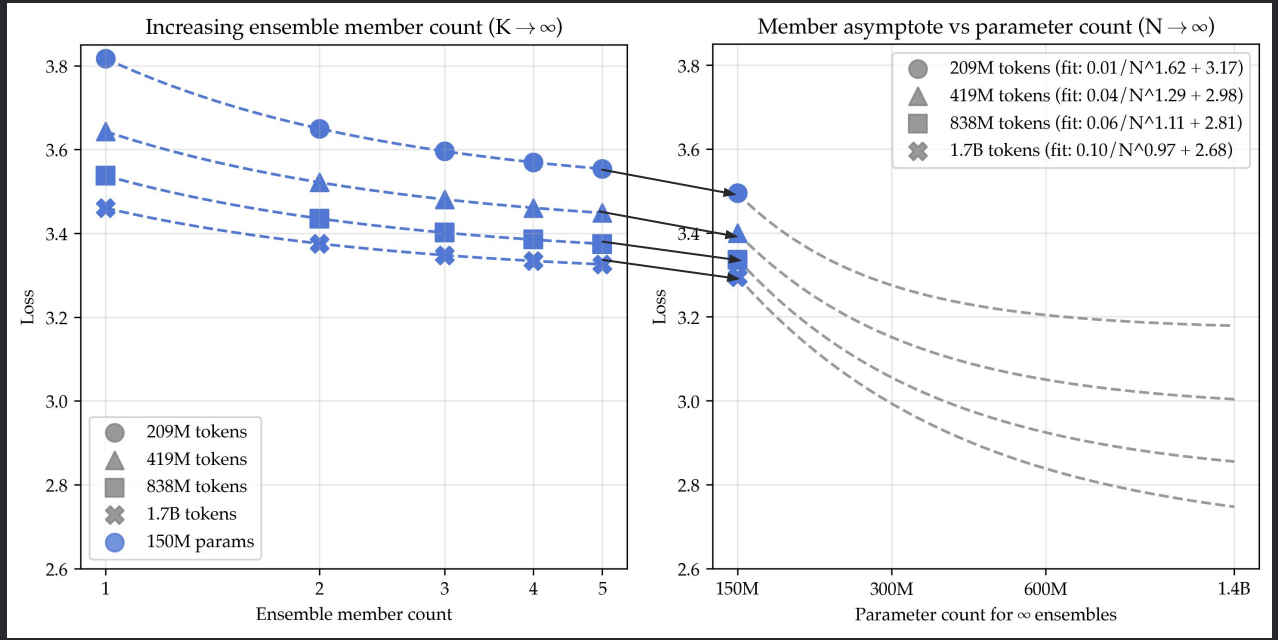
5. Scaling seed token count

For each seed token count, we find the best possible loss of the standard recipe and regularized recipe (asymptote)



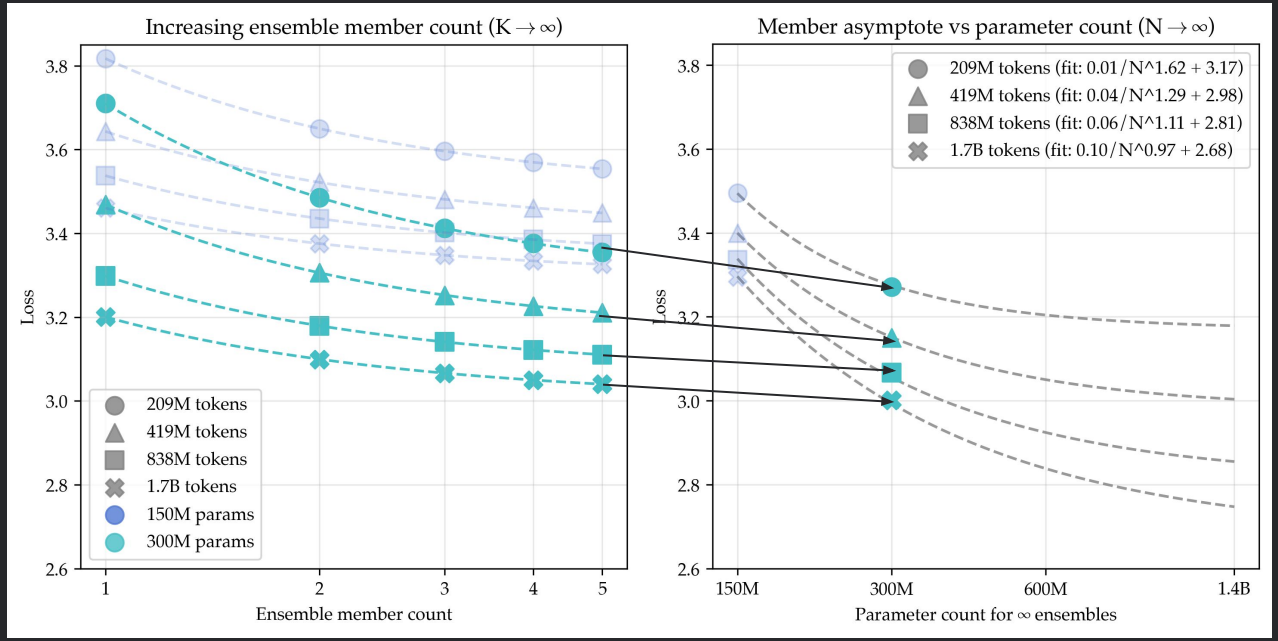
5. Scaling seed token count

For ensembles, for each D, we first send K → ∞ for fixed N with hparams selected for the asymptote.



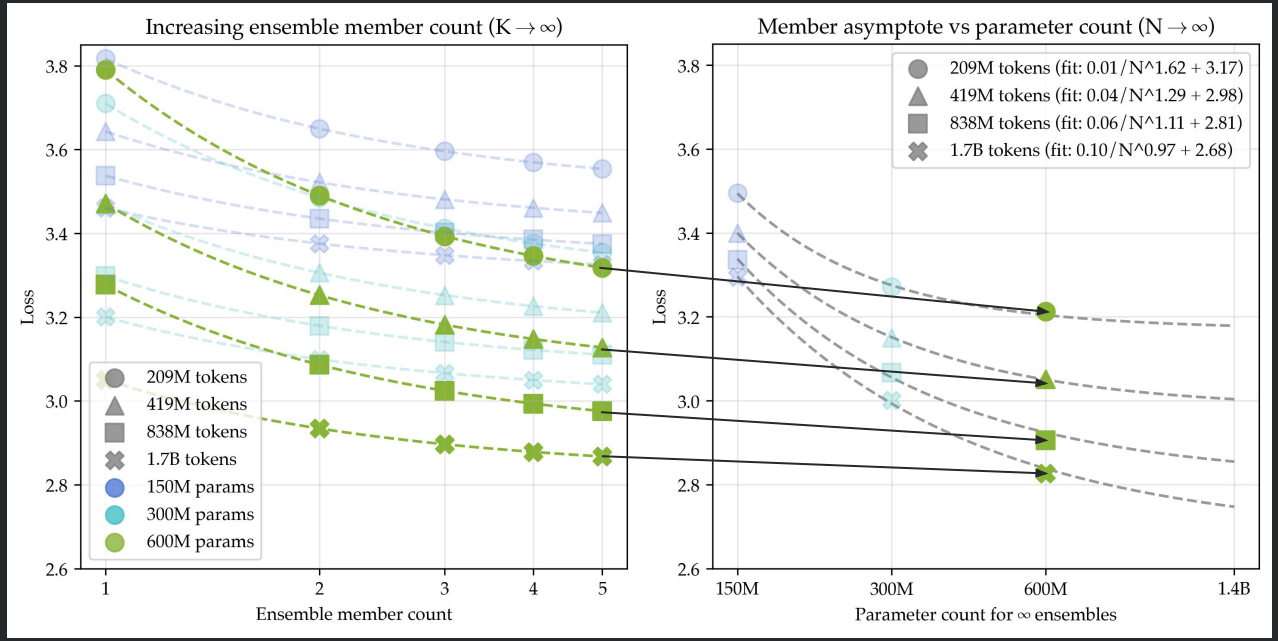
5. Scaling seed token count

For ensembles, for each D, we first send K → ∞ for fixed N with hparams selected for the asymptote.



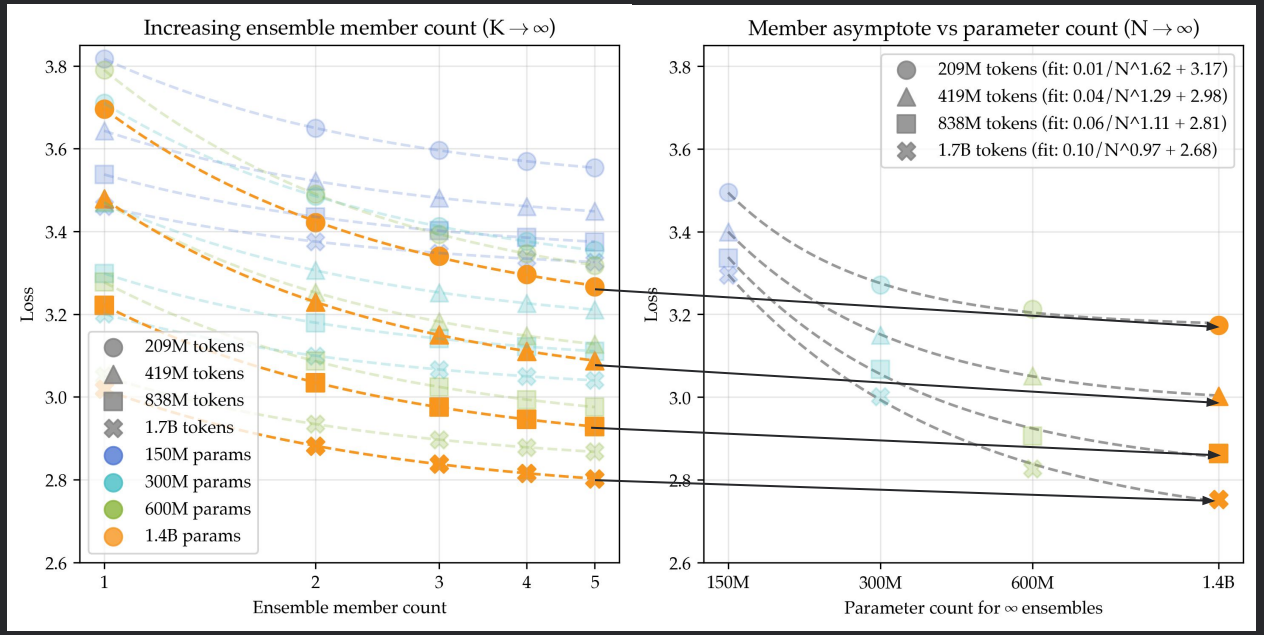
5. Scaling seed token count

For ensembles, for each D, we first send $K \to \infty$ for fixed N with hparams selected for the asymptote.

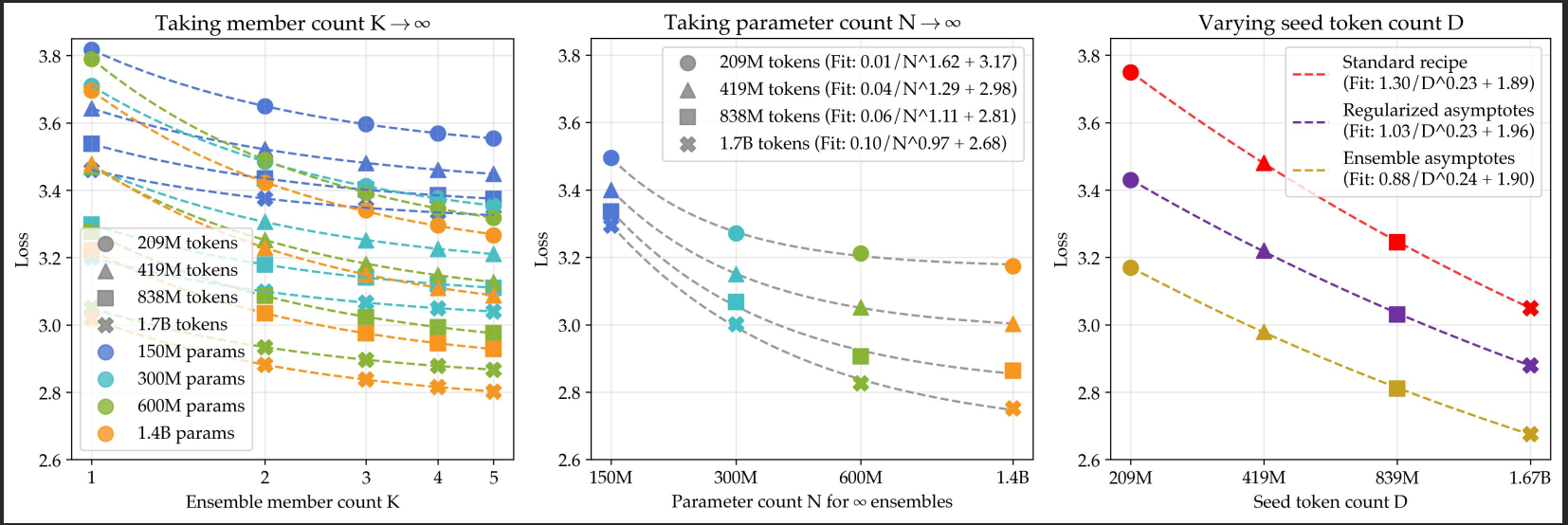


5. Scaling seed token count

For ensembles, for each D, we first send K → ∞ for fixed N with hparams selected for the asymptote.

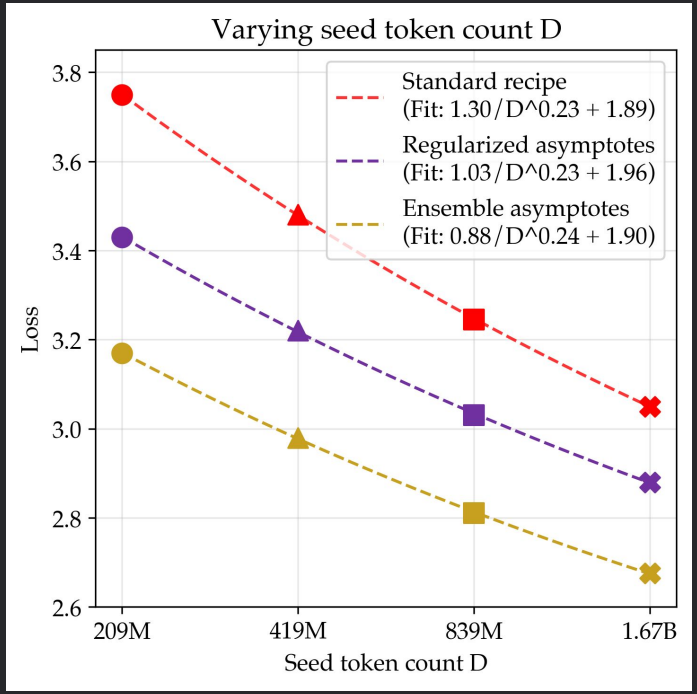


5. Scaling seed token count



5. Scaling seed token count

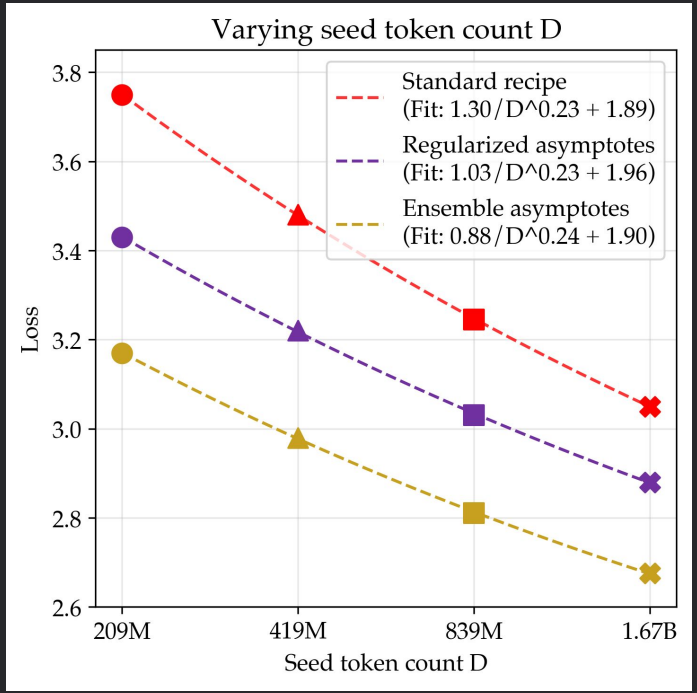
Summary of best possible loss for each method at each token count



5. Scaling seed token count

Summary of best possible loss for each method at each token count

Baseline needs ~5 times more data to match the performance of ensembling at 200M tokens

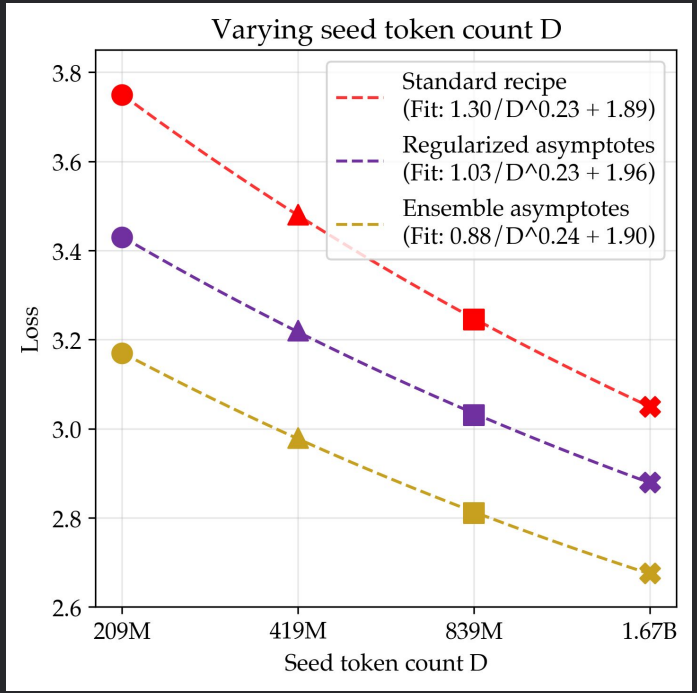


5. Scaling seed token count

Summary of best possible loss for each method at each token count

Baseline needs ~5 times more data to match the performance of ensembling at 200M tokens

Asymptotes and exponents are weirdly similar across methods, suggesting that both methods are a constant data-efficiency win across token scales



6. Distillation

do we need such large parameter counts at inference/training?

6. Distillation

Are the large parameter counts necessary for the final model and during training?

We find that via distillation, we can achieve data-efficiency gains without largely increasing parameter count

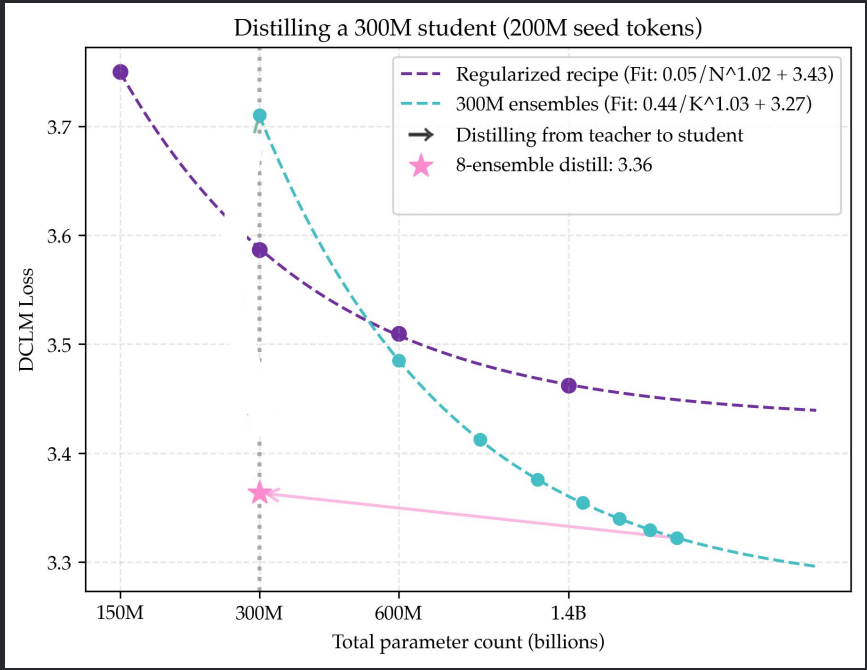
6. Distillation algorithm

We opt for the simplest algorithm of Sequence KD [Kim and Rush, 2016]

1. Train a teacher model M′ on D tokens
2. Sample from M′ unconditionally (i.e. no prompt) to generate a dataset of D′ tokens
3. Train a student model M from scratch on the mixture of D and D′

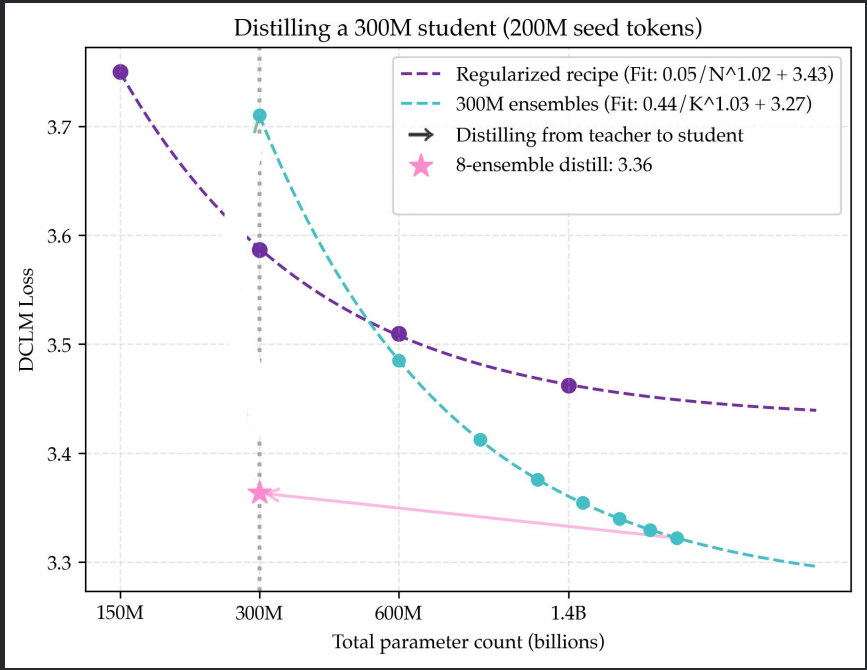
6a. Ensemble distillation

We follow this algorithm to distill an 8-ensemble into a single model, retaining most of the loss improvement and beating standard parameter scaling



6a. Ensemble distillation

Therefore, the benefit of ensembles + parameter scaling is more related to optimization dynamics rather than limited expressivity



6b. Self-distillation

Ensemble distillation removes ensembles at inference time. Do we need them at train time?

At first glance, this seems unlikely assuming that student models can not out-perform their teachers. In fact, papers claim that training on self-generations results in mode collapse [Dohmatob et al, 2024 ; Gerstgrasser et al, 2024 ; Shumailov et al, 2024]

6b. Self-distillation

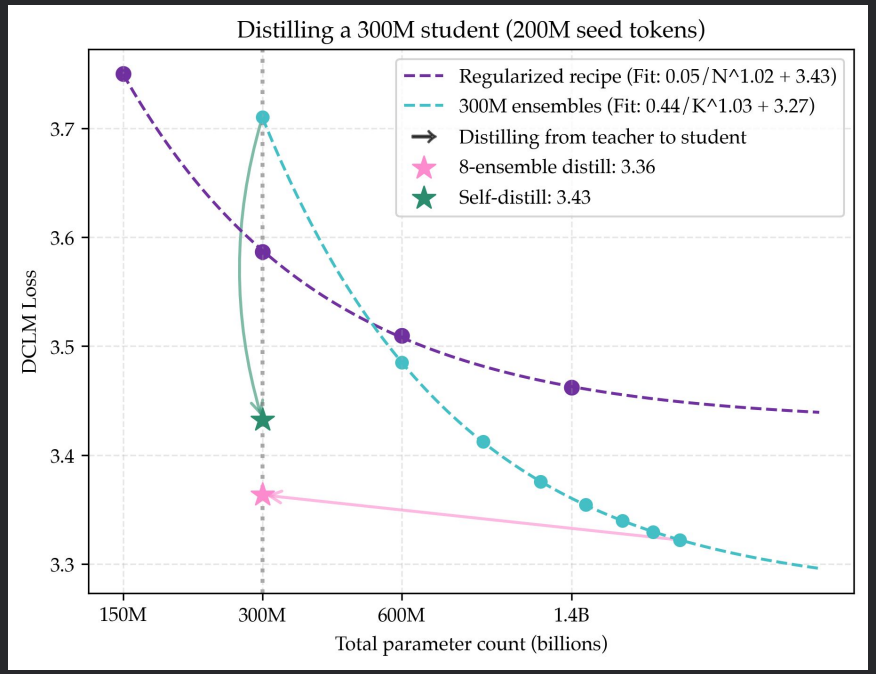
Ensemble distillation removes ensembles at inference time. Do we need them at train time?

At first glance, this seems unlikely assuming that student models can not out-perform their teachers. In fact, papers claim that training on self-generations results in mode collapse [Dohmatob et al, 2024 ; Gerstgrasser et al, 2024 ; Shumailov et al, 2024]

Surprisingly, we find that self-distillation (making the student and teacher exactly the same) *improves* performance!

6b. Self-distillation

Even though the teacher has the same architecture and parameter count as the student, the student out-performs the teacher

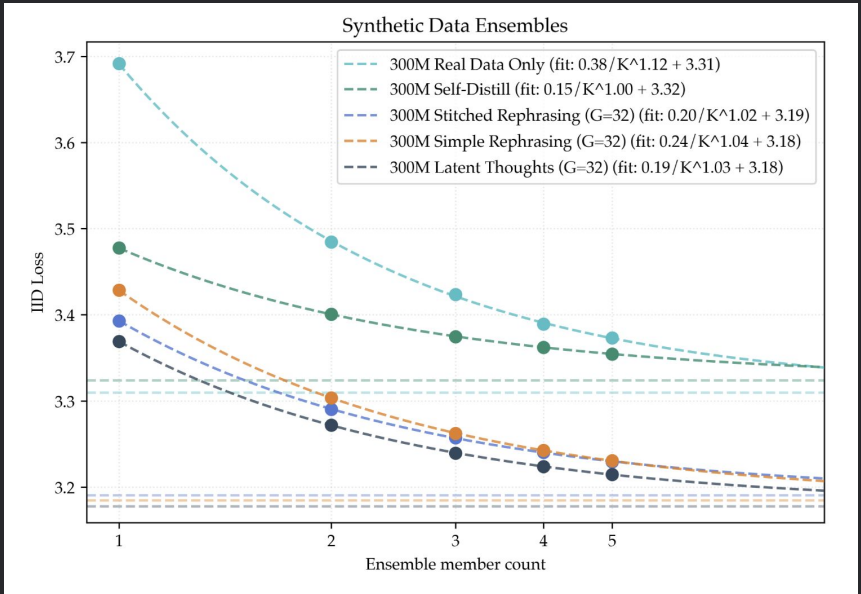


6b. Self-distillation

Allen-Zhu and Li, 2023 suggest that self-distillation can be seen as implicitly ensembling the trained teacher model and the fresh student model, suggesting a connection between distillation, synthetic data, and ensembling

6b. Self-distillation

We can test this prediction by ensembling multiple independently self-distilled models [Kim et al, 2026]. We find that the self-distillation does NOT compose with ensembling, suggesting they are equivalent benefits



7. Downstream benefits

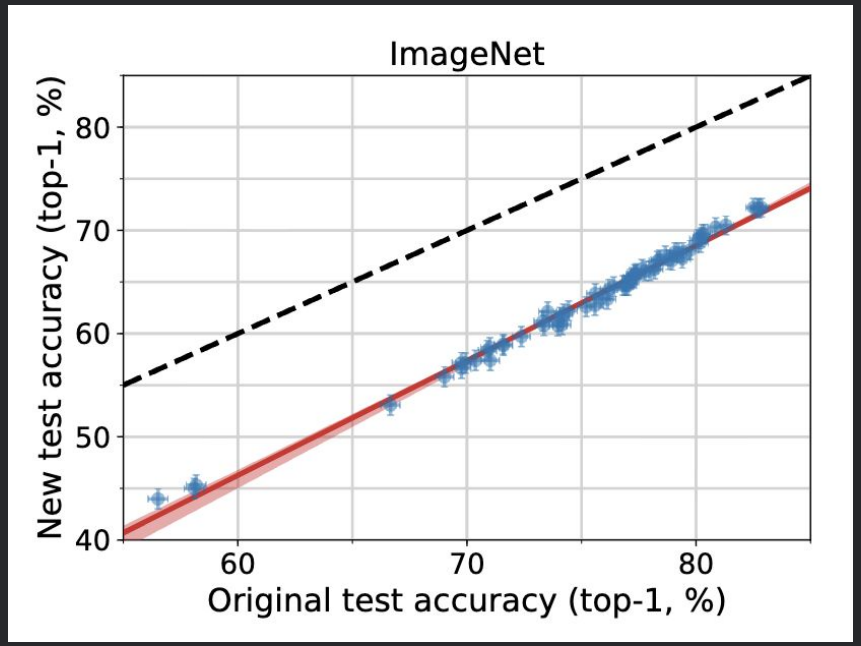
is it okay that we only looked at validation loss?

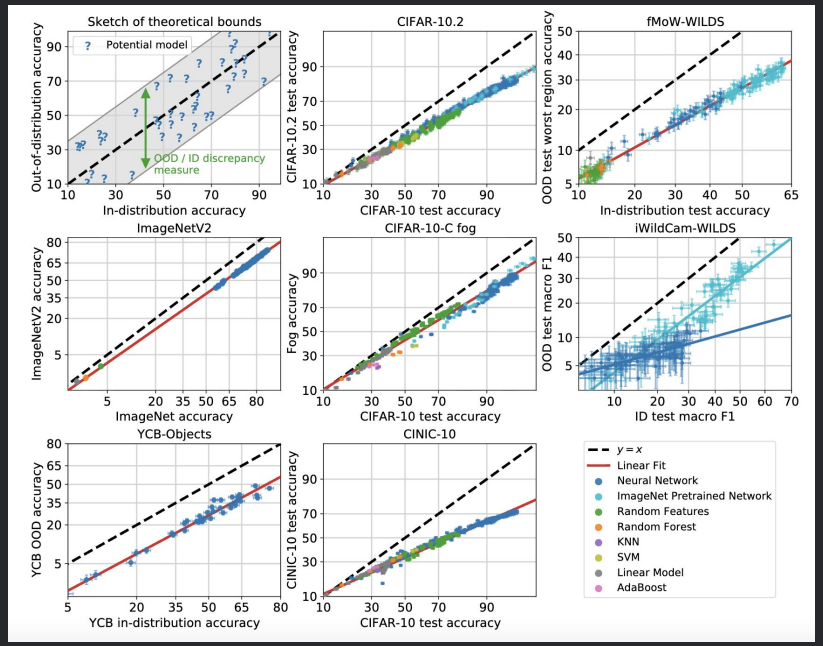
7: Out-of-distribution generalization

Does in-distribution loss map to out-of-distribution tasks we care about?
Historically, many algorithms tried to improve out-of-distribution performance via explicit regularization

7: Out-of-distribution generalization

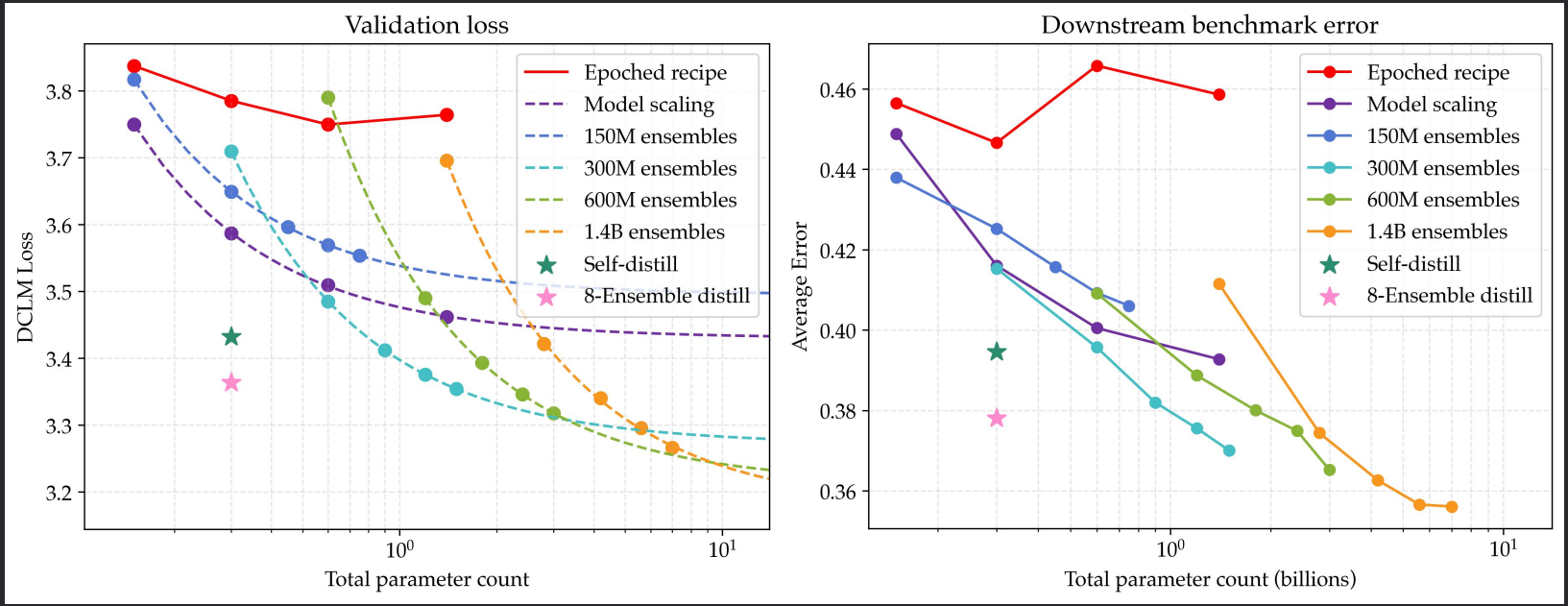
One important intuition is that for most algorithms, OOD performance is strongly correlated with ID performance [Recht et al, 2019 ; Miller et al, 2021]





7: Validation loss translates to downstream benchmarks

In our setting, better validation loss translates to better average performance on the small model benchmarks of ARC Easy, PiQA, and SciQ



Summary

We can compose various forms of regularization (limited epoch count, regularized parameter scaling, ensembling) to improve loss in the data-constrained regime

Suggests that a lot of the algorithms we developed for compute-efficient training may need to be reworked for the data-efficient setting. We will go over such algorithms next Tuesday

Next, we'll figure out why we're doing all this pre-training in the first place…

Fine-tuning concepts

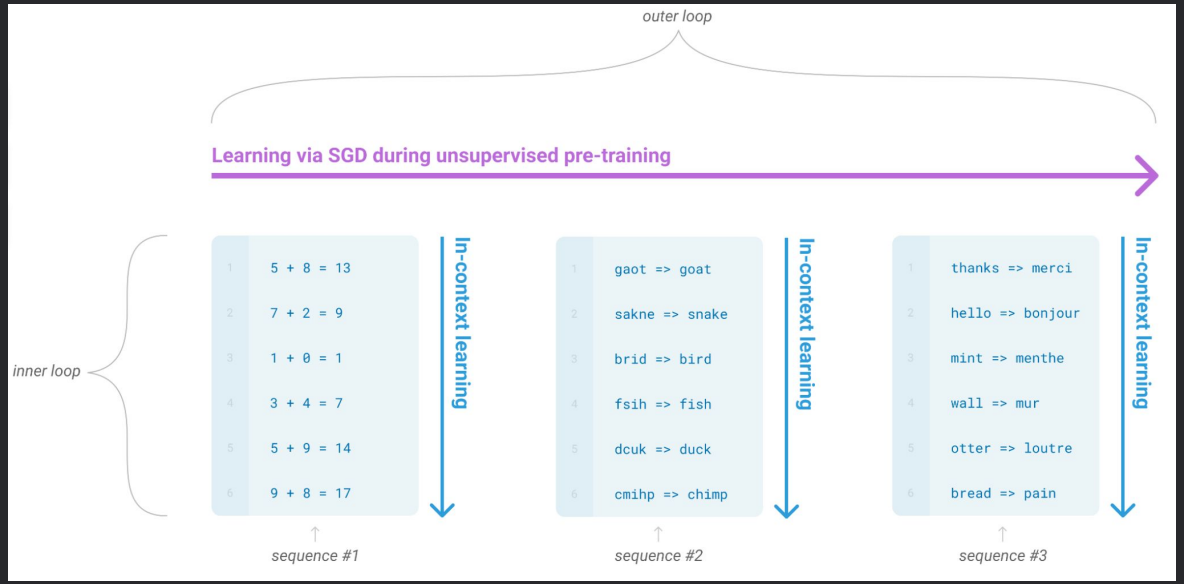
Training -> Pre-training

The main motivation for **pre**-training is to **pre**cede training on a target domain of interest (our “pre-training” has honestly just been “training” so far)

Though we’ve discussed how we can improve pre-training data efficiency, we’ll discuss how pre-training can improve fine-tuning data efficiency.

In-context learning

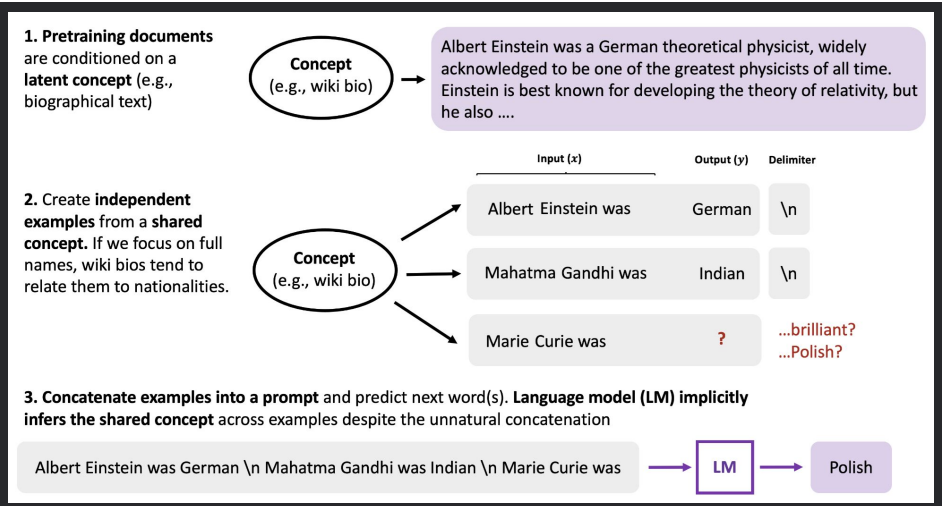
For a simpler case, we will consider taking a pre-trained model and giving our fine-tuning data (as (x, y) pairs) in-context. Remarkably, people find pre-trained models capable of in-context learning in this fashion



In-context learning

When you provide samples in-context, optimal pre-trained models would perform Bayesian inference to infer the fine-tuning task + solve it [Xie et al, 2021]

$$p(\text{output}|\text{prompt}) = \int_{\text{concept}} p(\text{output}|\text{concept}, \text{prompt}) p(\text{concept}|\text{prompt}) d(\text{concept}).$$



In-context learning

Because we pre-trained a strong model, we can implement both

(1) Implicitly infer the concept of interest

(1) Solve the task conditioned on knowing the concept

Importantly, implicit inference (Part 1) can be done in much fewer samples than actually learning the capability (Part 2)

$$p(\text{output}|\text{prompt}) = \int_{\text{concept}} p(\text{output}|\text{concept}, \text{prompt}) p(\text{concept}|\text{prompt}) d(\text{concept}).$$

Fine-tuning

We can operationalize fine-tuning as performing the same implicit inference with gradient descent. We can think of our pre-trained model as implementing inference and capabilities

$$f_{PT}(y \mid x) = \Sigma_i \, \alpha_i(x) \, f_i(y \mid x)$$

for various capabilities f_i (coding, writing, etc.) and $\alpha_i(x)$ that implicitly infers what capability x requires, ideally

$$\alpha_i(x) = P(\text{uses } f_i \mid x) * P(\text{uses } f_i)$$

Feature learning

$$f_{PT}(y \mid x) = \Sigma_i \alpha_i(x)\, f_i(y \mid x)$$

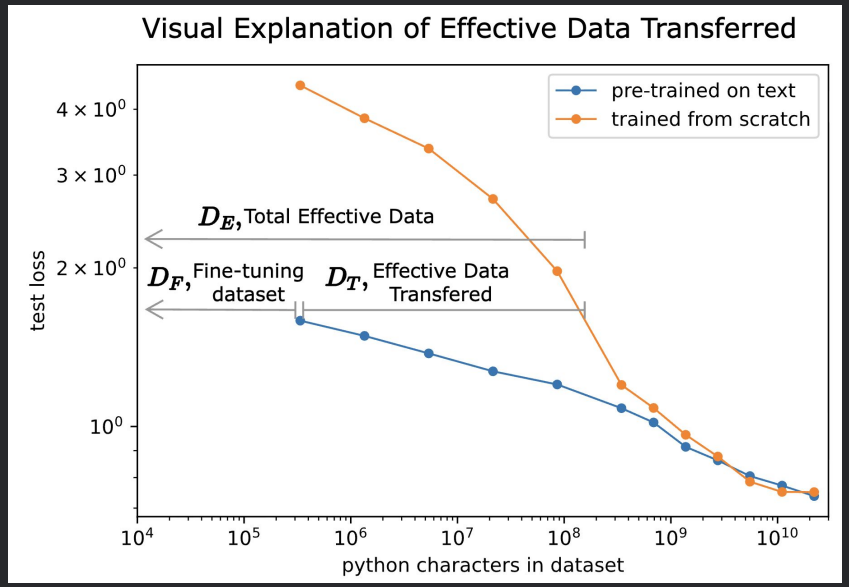
Fine-tuning can quickly improve loss by reweighting the mixture weights $\alpha_i(x)$ for the target task and also retaining many other general purpose capabilities.

We will soon see examples of how with the right methods, pre-training + fine-tuning can improve data efficiency in-distribution, retain prior capabilities, and improve performance out-of-distribution.

Fine-tuning empirics

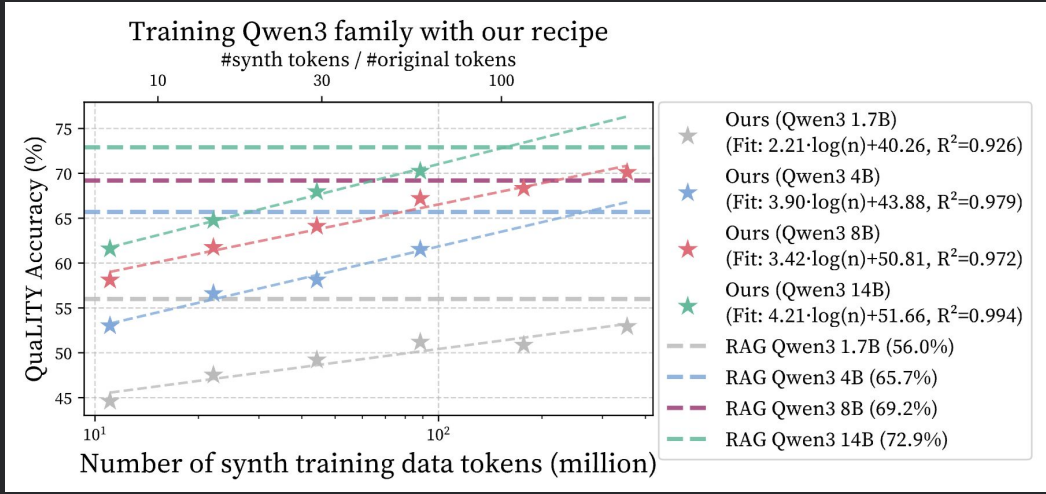
Fine-tuning increases effective samples

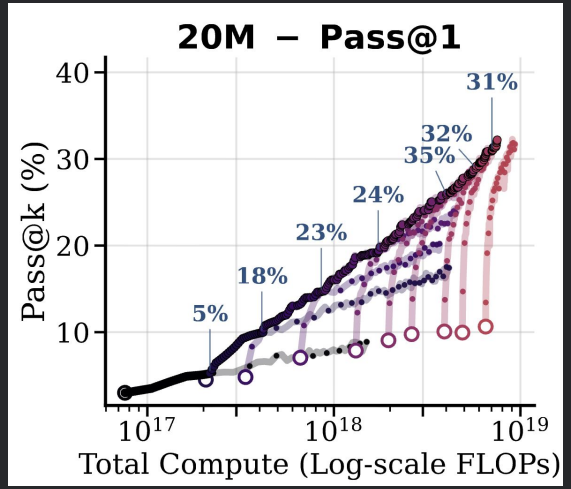
We can compare training on data from a target task with and without pre-training. Kaplan et al, 2021 finds that pre-training transfers a large amount of effective samples to fine-tuning, improving data efficiency by orders of magnitude



Better pre-training leads to better fine-tuning

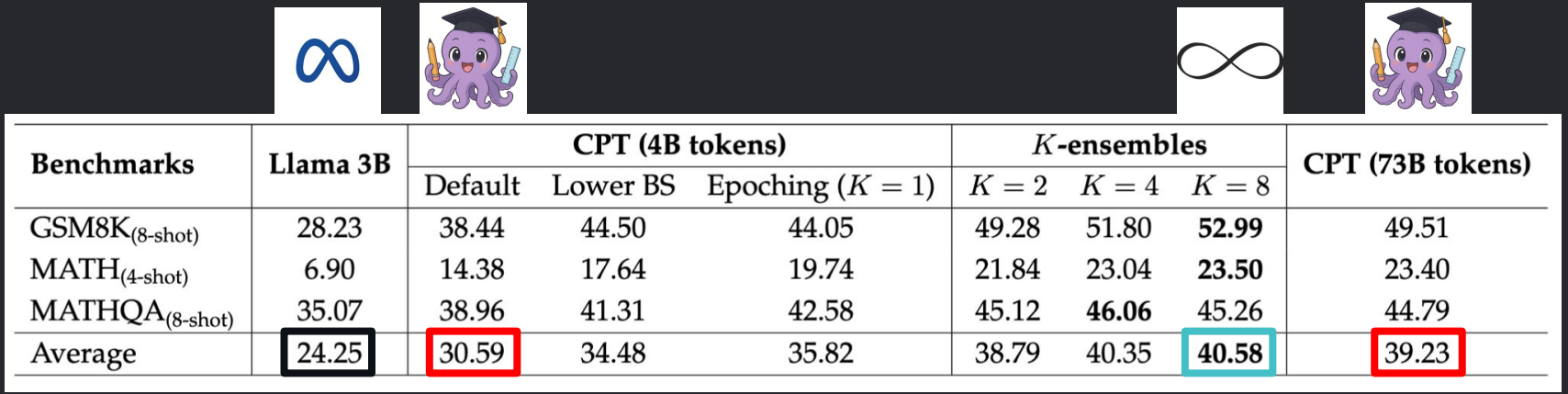
Pre-training not only improves fine-tuning data efficiency, it improves it exponentially. In Han et al, 2026, different student models have better fine-tuning slopes and achieve > 100x data efficiency. Same for Chess RL [Shen et al, 2026]





General data efficiency interventions

We can transfer our general data efficiency interventions from pre-training to fine-tuning as well. Kim et al, 2025 with 4B tokens outperforms standard training with 73B tokens (17.5x data-efficiency)



Benchmarks	Llama 3B	CPT (4B tokens)			K-ensembles			CPT (73B tokens)
		Default	Lower BS	Epoching ($K = 1$)	$K = 2$	$K = 4$	$K = 8$	
$\text{GSM8K}_{\text{(8-shot)}}$	28.23	38.44	44.50	44.05	49.28	51.80	**52.99**	49.51
$\text{MATH}_{\text{(4-shot)}}$	6.90	14.38	17.64	19.74	21.84	23.04	**23.50**	23.40
$\text{MATHQA}_{\text{(8-shot)}}$	35.07	38.96	41.31	42.58	45.12	**46.06**	45.26	44.79
Average	24.25	30.59	34.48	35.82	38.79	40.35	**40.58**	39.23

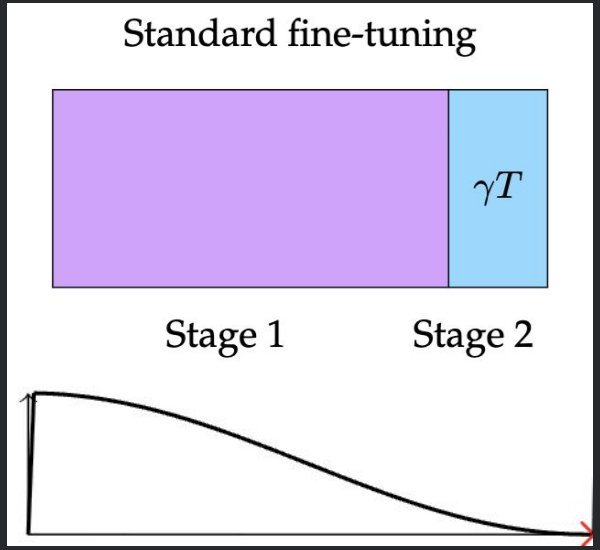
Challenges in fine-tuning

Fine-tuning comes with a unique set of issues

1. How can we best transfer pre-trained performance to the fine-tuned task?
2. How do we preserve our earlier pre-trained performance?
3. How do we improve performance on out-of-distribution evaluations?

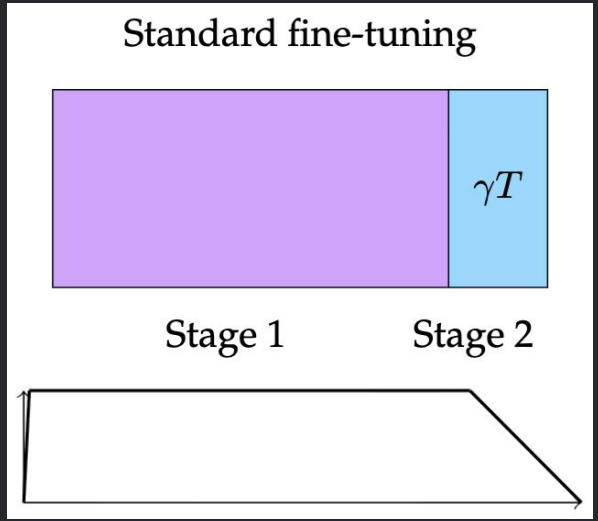
1. Transferring pre-trained performance

How should we handle optimization dynamics? If we kept a joint decay schedule for both stages, then fine-tuning data will be seen when loss is changing the least



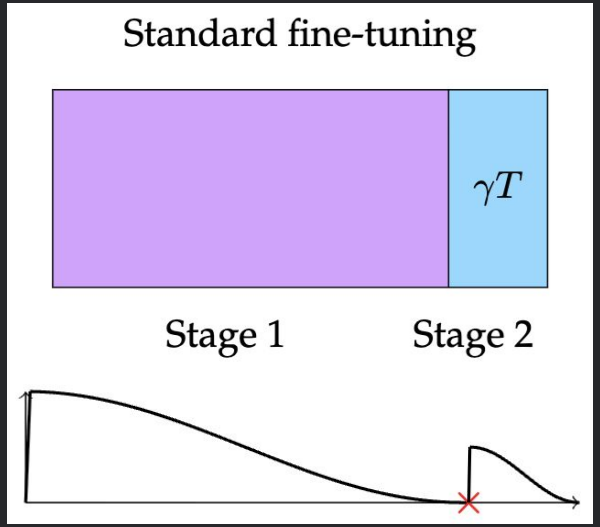
1. Transferring pre-trained performance

Ideally, you would use a WSD schedule for fine-tuning since the most progress happens during the final decay phase



1. Transferring pre-trained performance

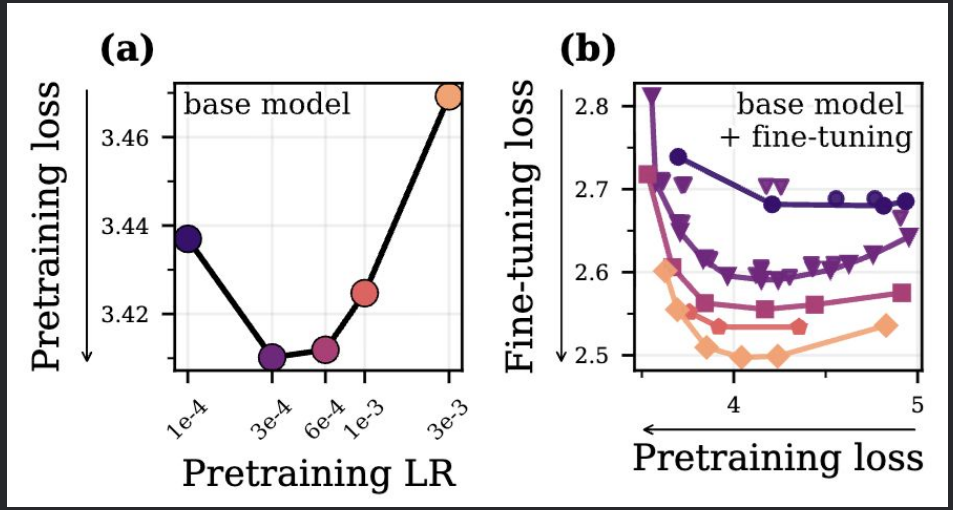
In practice, pre-trained models are mostly decayed and don't release optimizer state. Therefore, we have to compromise by rewarming up the model.



1. Transferring pre-trained performance

Even after controlling for this, pre-training might not perfectly predict the fine-tunability of a model.

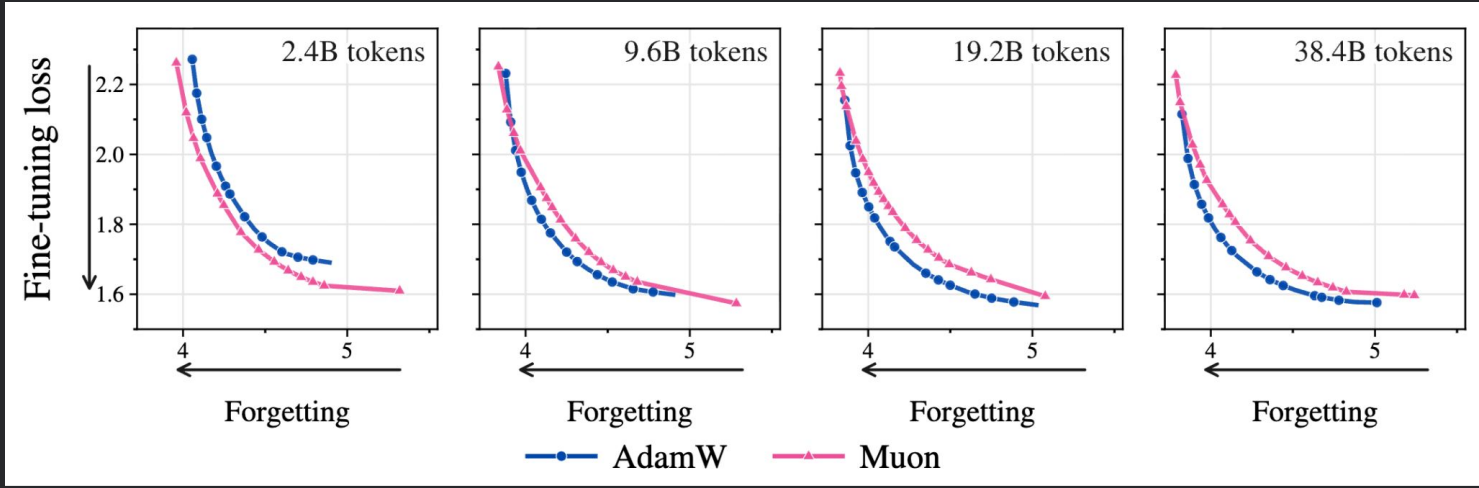
Watts et al, 2026 find flatter solutions (from higher LR) lead to better fine-tuning



1. Transferring pre-trained performance

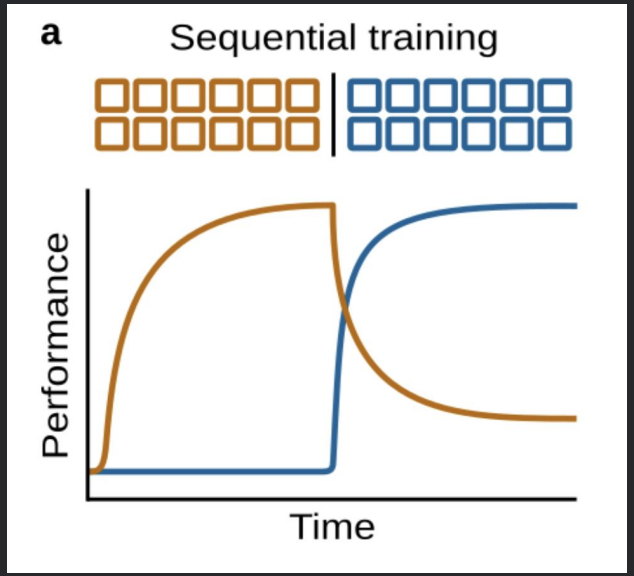
Even after controlling for this, pre-training might not perfectly predict the fine-tunability of a model.

Li et al, 2026 find that Muon can improve pre-training but hurt fine-tuning



2. Catastrophic forgetting

Another concern is that fine-tuning on the target task may cause “catastrophic forgetting” of the original task [McCloskey and Cohen, 1989]



2. Catastrophic forgetting

However, forgetting is almost a non-issue if you replay the original data [Schaul et al, 2015]. Simple demonstration from Marek et al, 2026 using replay data generated from the current model.

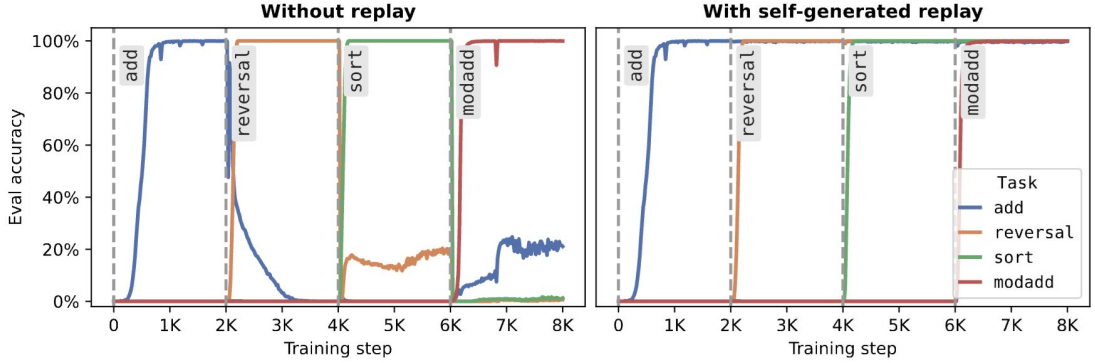
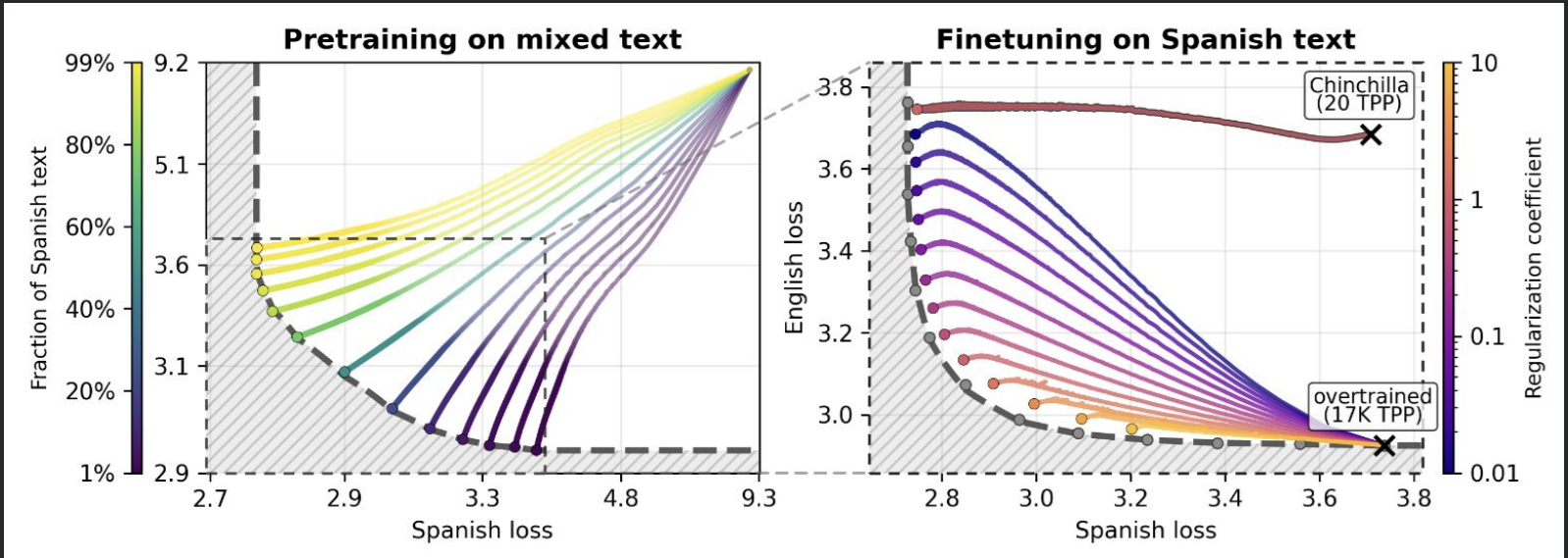


Figure 3: **Replay mitigates forgetting of language models under shifting tasks.** A small (2M) transformer language model is sequentially trained to perform add, `reversal`, `sort`, and modadd on 3-digit decimal inputs. **Left:** Using standard training, as the model learns a new task, its accuracy on prior tasks completely degrades. **Right:** By adding self-generated replay data, forgetting is entirely eliminated.

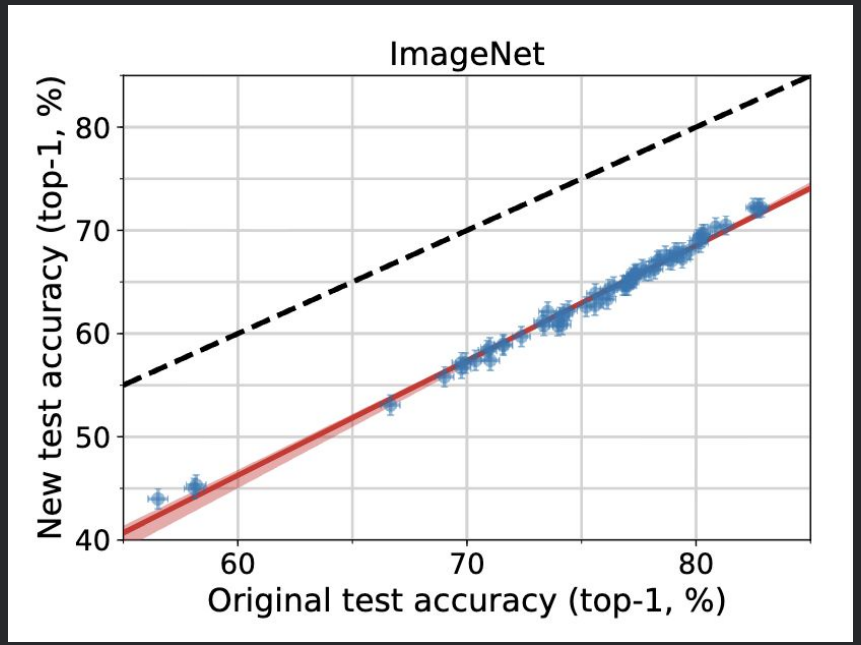
2. Catastrophic forgetting

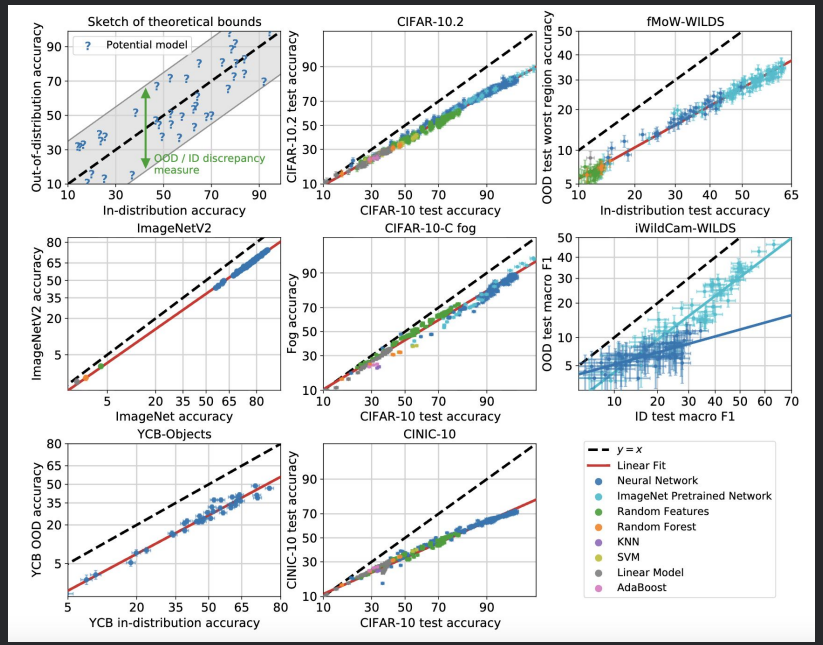
Marek et al, 2026 further quantify how forgetting is mostly induced by capacity. For example, its most prevalent during pre-training and least prevalent when fine-tuning a model that has not seen many tokens



3. Out-of-distribution generalization

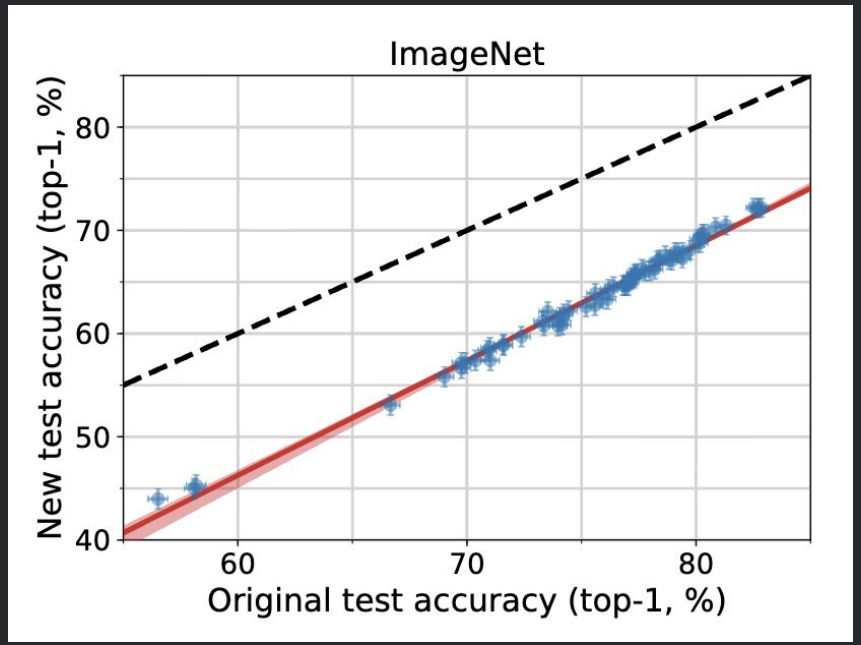
Recall the earlier plot for correlation between ID and OOD generalization. How does pre-training + fine-tuning influence this?

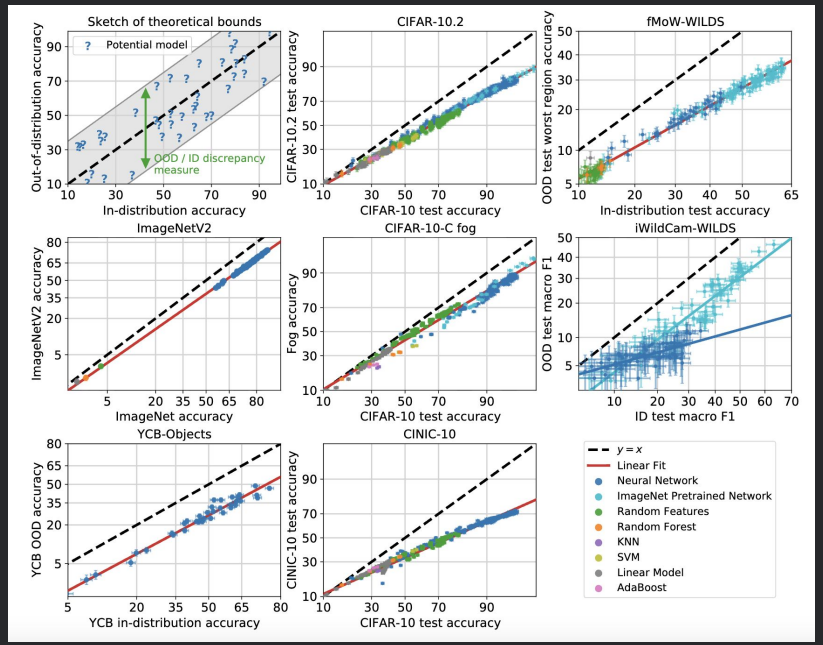




3. Out-of-distribution generalization

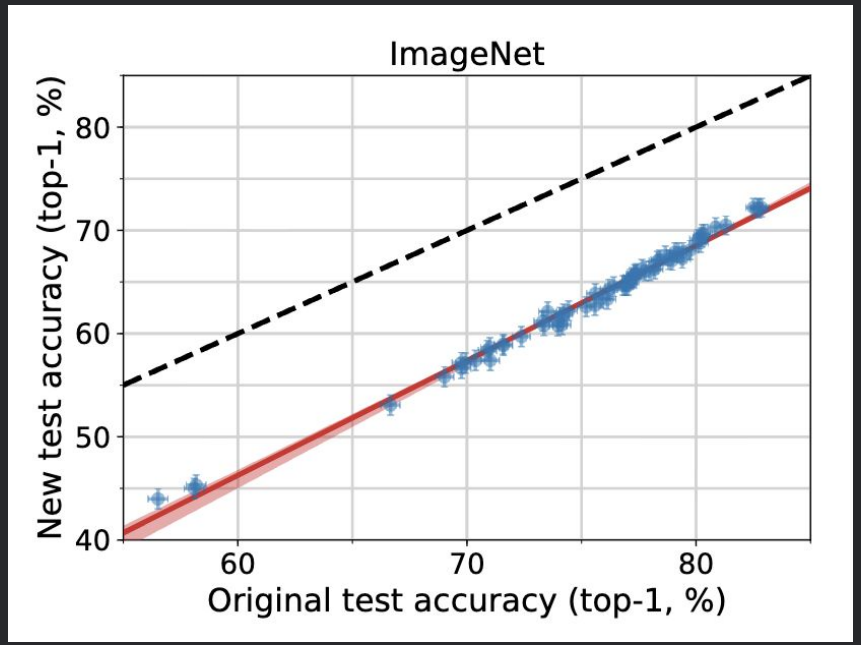
Notice an exception?

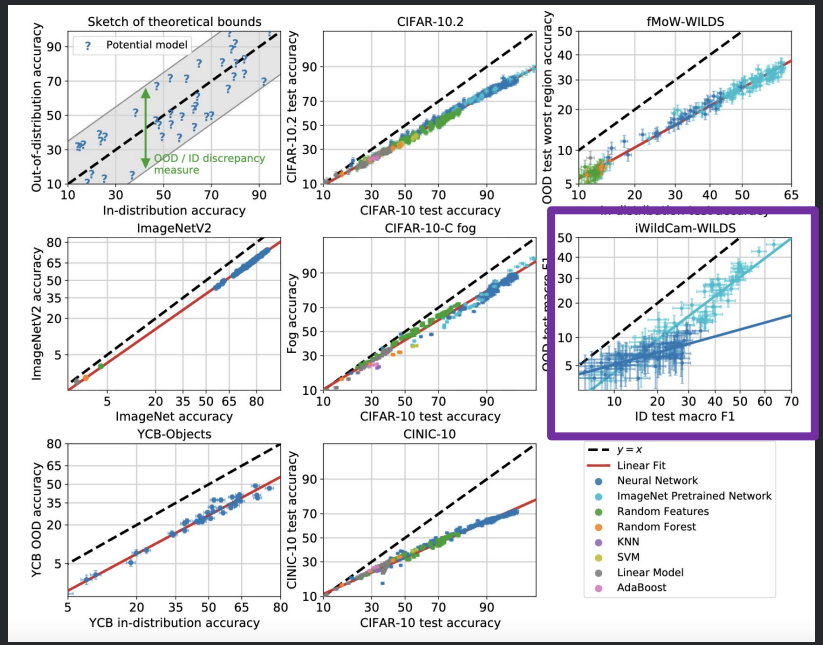




3. Out-of-distribution generalization

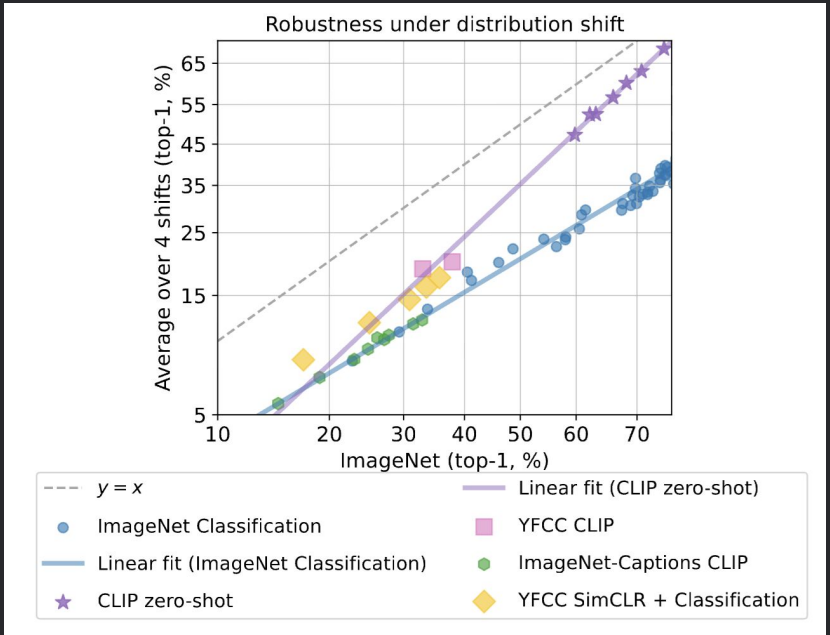
Notice an exception?





3. Out-of-distribution generalization

In general, after many attempts at improving OOD generalization, the most robust intervention is pre-training on more diverse data [Fang et al, 2022]



Summary

Treating our modeling task as fine-tuning improves performance

We can squeeze more value if we are careful about the handoff between stages

Catastrophic forgetting may appear problematic but can often be mitigated

Pre-training can improve OOD performance in qualitatively different ways

Next week

We now know that we need to revisit all of the choices in the deep learning stack with a fresh eye for data efficiency.

How can we develop data-efficient optimizers? (minimize sharpness?)
How can we develop data-efficient architectures? (minimize description length?)
How should we process our data (keep it and make more?)