

Pre-training under infinite compute

Konwoo Kim[∞], Suhas Kotha[∞], Percy Liang, Tatsunori Hashimoto
Stanford University

Abstract

Since compute grows much faster than web text available for language model pre-training, we ask how one should approach pre-training under fixed data and no compute constraints. We first show that existing data-constrained approaches of increasing epoch count and parameter count eventually overfit, and we significantly improve upon such recipes by properly tuning regularization, finding that the optimal weight decay is $30\times$ larger than standard practice. Since our regularized recipe monotonically decreases loss following a simple power law in parameter count, we estimate its best possible performance via the **asymptote** of its scaling law rather than the performance at a fixed compute budget. We then identify that ensembling independently trained models achieves a significantly lower loss asymptote than the regularized recipe. Our best intervention combining epoching, regularization, parameter scaling, and ensemble scaling achieves an asymptote at 200M tokens using $5.17\times$ less data than our baseline, and our data scaling laws predict that this improvement persists at higher token budgets. We find that our data efficiency gains can be realized at much smaller parameter counts as we can distill an ensemble into a student model that is $8\times$ smaller and retains 83% of the ensembling benefit. Finally, our interventions designed for validation loss generalize to downstream benchmarks, achieving a 9% improvement for pre-training evals and a $17.5\times$ data efficiency improvement over continued pre-training on math mid-training data. Our results show that simple algorithmic improvements can enable significantly more data-efficient pre-training in a compute-rich future.

1 Introduction

Language model pre-training has historically been studied under compute constraints at training [Hoffmann et al., 2022, Kaplan et al., 2020] and inference [Brown et al., 2024, Sardana et al., 2025, Snell et al., 2024] while assuming access to unlimited web text. However, web data increases by $1.03\times$ per year, whereas compute spent on pre-training grows by $4\times$ per year [Sevilla and Roldán, 2024, Villalobos et al., 2024]. In anticipation of a regime where compute vastly exceeds data, we ask:

How should one approach pre-training when constrained by data and unconstrained by compute?

To establish a baseline, we fix a seed training corpus of 200M tokens of web text and evaluate a *standard recipe* following existing data-constrained approaches of repeating data [Muennighoff et al., 2023] and increasing parameter count [Kaplan et al., 2020] (Section 2). We find that either too many epochs or too many parameters results in the loss eventually increasing due to overfitting. This bounds the performance improvements we can get from tuning this recipe, even if we were willing to spend more compute in exchange for a better model.

We instead get predictable monotone scaling in parameter count by considering a *regularized recipe* (Section 3). Currently, regularization used in pre-training is often adopted from existing recipes, for example defaulting to a weight decay of 0.1 from Brown et al. [2020]. We find this amount to

[∞]Equal contribution.

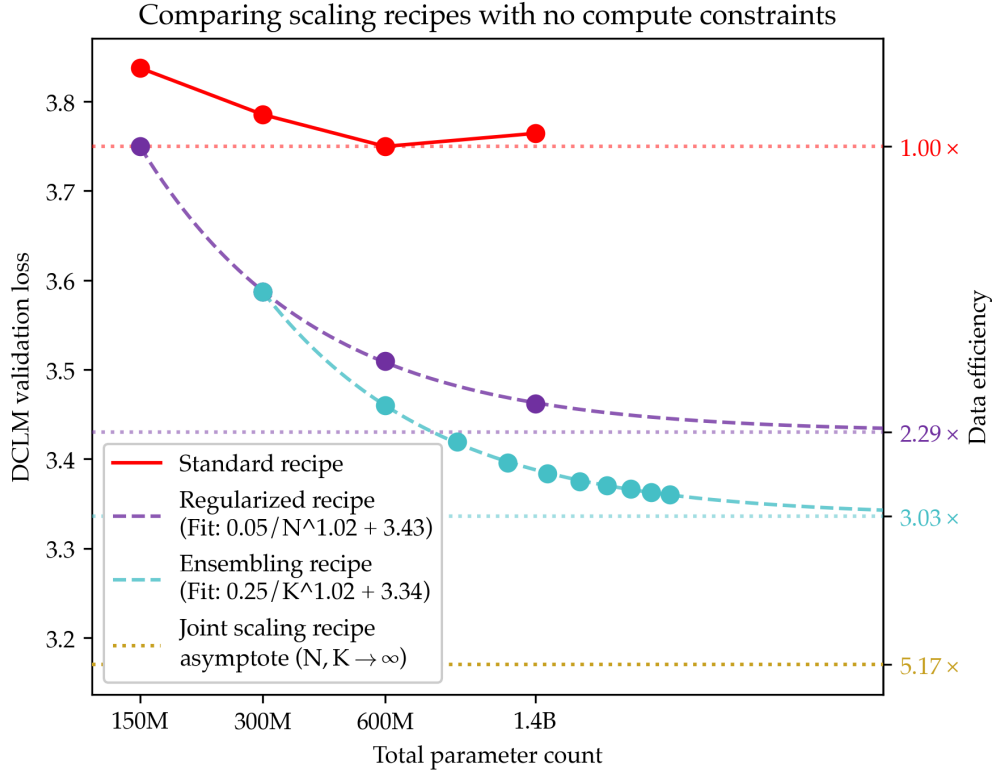


Figure 1: **Comparing scaling recipes with no compute constraints.** To simulate a data-constrained future, we restrict models to only have access to 200M training tokens. (1) Standard pre-training recipes overfit with too many epochs or parameters, even if we tune the epoch count at each parameter count N (2) By correctly tuning regularization for each parameter count, loss monotonically decreases following a power law in parameter count N . We predict the best possible loss of the regularized recipe by the **asymptote** of its power law (3) Instead of scaling N , we achieve a lower asymptote by ensembling K models of a fixed parameter count as K approaches infinity. (4) Composing parameter scaling and ensemble scaling further improves the asymptote, and we estimate that the baseline would need $5.17\times$ more data to match its loss, even with infinite compute. We additionally find that these data efficiency improvements hold for larger token counts, distilled models, downstream benchmarks, and continued pre-training (Sections 5, 6, 7).

be inadequate for preventing overfitting under data constraints and that the optimal weight decay is $30\times$ larger than standard practice for our most over-parameterized models. After jointly tuning weight decay, learning rate, and epoch count at each parameter count N , loss closely follows a power law in N for parameter-to-token ratios $140\times$ larger than Chinchilla, as shown in Figure 1.

Normally, researchers would compare two recipes such as regularized and standard by evaluating performance at different train or inference compute budgets [Brown et al., 2024, Hoffmann et al., 2022, Kaplan et al., 2020, Sardana et al., 2025, Snell et al., 2024]. However, this does not reflect our interest in the best possible performance when constrained by data and unconstrained by compute. Since the loss of the regularized recipe continues to decrease as parameter count increases, we are interested in the limit of the loss as parameter count goes to infinity. More generally, we propose evaluating monotone scaling recipes by the **asymptote** of their scaling law (for example, 3.43 for the

regularized recipe as seen in Figure 1). By preferring recipes with lower loss asymptotes, we can train better models at sufficiently high compute budgets.

Though taking the parameter count to infinity is one possible limit under infinite compute, we ask if we can design recipes with even lower asymptotes. We consider an alternative *ensembling recipe* where instead of training a larger model, we average the logits of K independently trained models, each of a fixed size [Dietterich, 2000] (Section 4). As shown in Figure 1, the ensembling recipe achieves a lower loss asymptote as ensemble member count approaches infinity compared to the regularized recipe as model parameter count approaches infinity. This implies that at sufficiently high parameter counts, it is advantageous to train multiple smaller models (e.g. two 300M models) instead of a single larger model (e.g. one 600M model). Furthermore, we find that ensembling and parameter scaling compose, achieving a lower asymptote when following the *joint scaling recipe* of taking both the number of members and the parameters of each member to infinity.

Since our previous experiments were on the scale of 200M tokens, we study how our recipes scale across higher seed token counts and find that the asymptotes themselves follow a scaling law (Section 5). Our estimates indicate that the joint scaling recipe achieves its 200M asymptote with $5.17\times$ less data than the standard recipe. Importantly, extrapolation of our data scaling laws indicate that the data efficiency improvements will persist at higher token counts.

Though the asymptotes of our recipes benefit the most from large parameter counts, we find that distillation [Hinton et al., 2015, Kim and Rush, 2016] allows us to retain most of the loss improvements without increasing parameter count at inference. Specifically, we find that distilling an 8-ensemble into a single 300M model retains 83% of the ensembling loss improvement over the best regularized 300M model and even outperforms the asymptote of the regularized recipe. We also find that self-distilling a 300M teacher into a student of the same architecture surprisingly reduces loss, improving data efficiency without ever explicitly training a model of higher parameter count.

Finally, we confirm that our validation loss improvements translate to improvements on downstream benchmarks (Section 7). Ensembles with better validation loss perform better on downstream benchmarks, with our best ensemble outperforming our best unregularized model by 9% on average over PIQA, SciQ, and ARC Easy (standard pre-training benchmarks for models at our scale [Thrush et al., 2025]). We also test the immediate applicability of our interventions for continued pre-training (CPT) on the MegaMath-Web-Pro dataset from Wang et al. [2025]. We find that with only 4B tokens, the ensembling recipe (i.e. an ensemble of epoched models) outperforms default CPT on the full 73B tokens following their training hyperparameters, resulting in a $17.5\times$ data efficiency improvement. We open-source all of our runs on WandB and our code on Github.

2 Standard pre-training

Historically, pre-training has focused on training the best possible models subject to compute or parameter constraints. Under train compute constraints, scaling recipes like Chinchilla recommend jointly increasing data and model size with $20\times$ more tokens than parameters [Hoffmann et al., 2022, Kaplan et al., 2020]. Under parameter constraints for cheaper inference and fine-tuning, current practice opts for over-training language models relative to Chinchilla with token counts $2000\times$ larger than the parameter count or distilling from preexisting larger models [Busbridge et al., 2025, Gadre et al., 2024, Grattafiori et al., 2024, Sardana et al., 2025].

Many prior works implicitly assume no constraint over the number of tokens and always train on fresh data. In this paper, we are instead interested in studying how algorithms fare under data constraints, preventing us from continuing to jointly scale token and parameter count. To study

data-constrained pre-training in its purest form, we return to the classical statistical formulation of learning, lifting all other constraints such as train compute and parameter count. To formulate our objective, we formalize standard pre-training as a training routine $\mathcal{A}$ that accepts arguments such as token count D , parameter count N , epoch count E to produce a model M with loss $\mathcal{L}(M)$. Any arguments that are not explicitly specified are passed through hyperparameter tuple H . Our objective for pre-training with D tokens of data unconstrained by compute becomes

$$\mathcal{L}_D^* = \min_H \mathcal{L}(\mathcal{A}(D, H))$$

To measure the performance of an algorithm, we construct a controlled pre-training environment with a limited amount of web data from DCLM [Li et al., 2025]. Since we are interested in algorithms that spend orders of magnitude more compute than Chinchilla scaling at a fixed data budget, we default to a smaller token count of 200M tokens and test whether our findings hold across higher token counts in Section 5. For evaluation, we defer to loss on a held-out i.i.d. validation set since this is shown to correlate with downstream pre-trained capabilities in our analysis in Section 7 and prior work [Chen et al., 2025, Gadre et al., 2024, Thrush et al., 2025]. To best represent standard practice, we follow a standard auto-regressive recipe, using Llama-style architecture, AdamW optimizer with cosine learning rate schedule, context length 4096, etc. (full details in Appendix A)

2.1 Evaluating existing data-constrained recipes

Since the amount of fresh data is limited, we build a standard recipe of increasing repetition count [Muennighoff et al., 2023] and parameter count [Kaplan et al., 2020]. Since there is unlimited compute, we depart from compute-efficient practice by training models that are much larger relative to the token count, defaulting to 300M parameter models for a 200M seed token constraint.

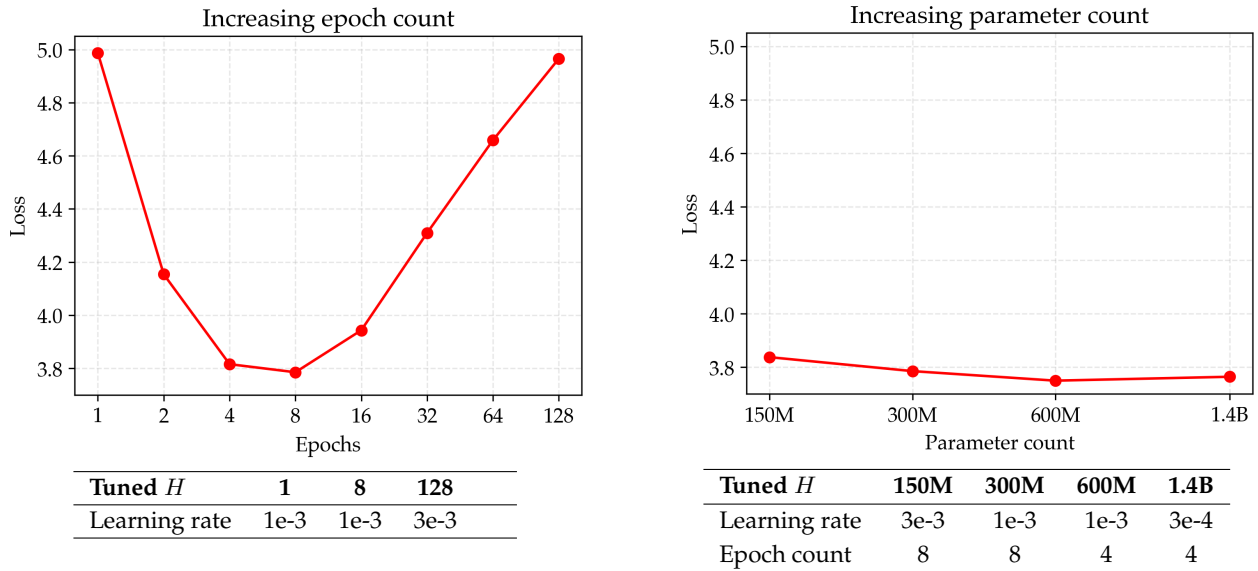


Figure 2: **Evaluating standard recipe of epoching and parameter scaling for 200M tokens.** Left: Though repeating the data lowers the loss, too many repetitions results in overfitting for 300M models. Right: We try increasing parameter count, tuning the epoch count at each parameter count. We similarly find that loss starts increasing. Moreover, increasing the parameter count 10× improves the loss by less than 0.1.

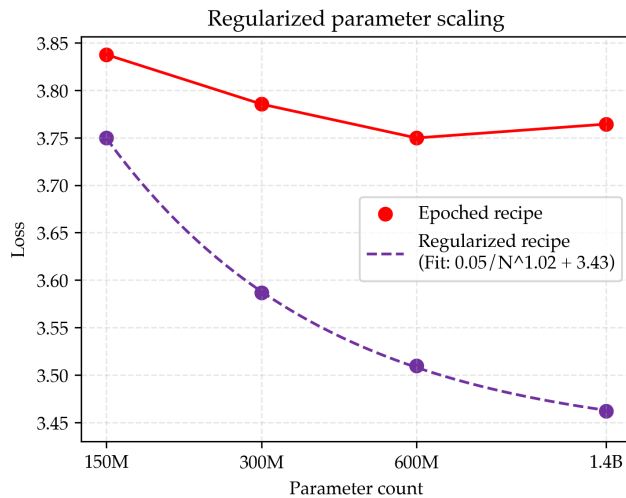
We first consider increasing the epoch count E at a fixed parameter count, taking $E \times$ more training compute. We find that for sufficiently high epoch counts, the models start overfitting and loss starts increasing (Figure 2, left), indicating that the data can not be epoched forever. These experiments disagree with the functional form of the decay-based scaling law in Muennighoff et al. [2023], which posits that loss monotonically decreases in epoch count. Their work acknowledges this discrepancy and removes most runs that overfit when fitting their scaling law, discussed in their Appendix D.

Since epoch count cannot be increased arbitrarily without increasing loss, we turn to increasing parameter count. To establish a competitive scaling recipe, we jointly tune epoch count and learning rate for each parameter count (described in Appendix B.1). Even after tuning these hyperparameters, increasing parameter count does not significantly decrease loss, and the 1.4B model performs worse than the 600M model. This is consistent with the single-pass findings in Kaplan et al. [2020], Figure 9 which show that increasing parameter count eventually starts increasing loss for fixed data. It is likely that both increasing repetition count and parameter count result in overfitting the train set, detailed in Appendix B.5.

3 Regularized parameter scaling

We show that to get the best performance from these over-parameterized, epoched models, it is critical to regularize pre-training with much higher weight decay than standard practice. To jointly tune weight decay, learning rate, and epoch count, we perform an extensive search for “locally optimal” hyperparameters where increasing or decreasing any hyperparameter does not result in a better model. To find these hyperparameters, we use a coordinate descent algorithm inspired by Wen et al. [2025] (more details in Appendix B.1). We find that over-parametrized models need much higher weight decay, over $30\times$ larger than the standard practice of 0.1 (Figure 3, right table).

With this tuning, loss follows monotone scaling in parameter count for models up to $140\times$ larger than Chinchilla as shown in Figure 3. This agrees with theory for over-parameterized regression that predicts that even when loss does not monotonically decrease due to double descent, the loss will monotonically decrease when regularization is optimally tuned [Advani and Ganguli, 2016, Canatar et al., 2021, Nakkiran et al., 2021, Simon et al., 2024]. In Appendix B.2, we show how our locally-optimal tuning procedure is critical to achieve this monotone scaling.



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

Figure 3: Power law scaling from jointly tuning regularization. We compare the standard recipe in Figure 2 (red line) to our regularized recipe that jointly tunes learning rate, epoch count, and weight decay (purple line). After tuning with regularization, the loss decreases proportional to $\approx \frac{1}{N}$. The power law predicts that as parameter count goes to infinity, the best model achieves 3.43 loss.

To capture how increasing parameter count improves loss, we fit a power law with an asymptote as

$$\hat{\mathcal{L}}_{D,N} := \frac{A_D}{N^{\alpha_D}} + E_D$$

where we fit free variables A_D, α_D, E_D . Fitting this law to the runs across four parameter counts results in $\hat{\mathcal{L}}_{200M,N} = \frac{0.05}{N^{1.02}} + 3.43$ ¹. This exponent of 1.02 for parameter scaling is quite high considering that Chinchilla finds a parameter scaling exponent of 0.34. This suggests that when we better leverage the data, there is faster improvement from using larger models.

However, if we truly had infinite compute, the monotone scaling law suggests that we should increase the parameter count N as much as we can, independent of the scaling exponent. This is different from the train compute-constrained regime where increasing N comes at the cost of training on less data and can hurt performance. To characterize the best possible performance when unconstrained by compute, we are interested in the loss under the limit as parameter count goes to infinity. Assuming the power law fit, we can estimate the limit as $\lim_{N \rightarrow \infty} \hat{\mathcal{L}}_{D,N}$, equivalent to the asymptote E_D . The asymptote for the regularized recipe law predicts that the best possible model it can train achieves loss 3.43. Because of run-to-run variance of the points forming the power law, we share a sensitivity analysis in Appendix H.1 and find that the estimated asymptotes vary by at most 0.02 loss across 3 seeds.

4 Ensemble scaling

The regularized recipe offers a straightforward way to improve performance by taking $N \rightarrow \infty$. Is this the best possible loss under infinite compute, or can different training algorithms better leverage the data? In this section, we consider simple ensembling [Dietterich, 2000]: independently train K models and average their logits for generation, formalized in Section 4.1. In Section 4.2, we show how ensembling can outperform parameter scaling at fixed parameter counts and under the limit as total parameter count approaches infinity. In Section 4.3, we construct our best recipe composing regularized parameter scaling and ensemble scaling by taking the limit as both $N, K \rightarrow \infty$.

4.1 Formalizing ensembles

The ensembling pre-training algorithm $\mathcal{E}$ accepts a standard pre-training algorithm $\mathcal{A}$ and trains K members that are identical except for random seed Z_i controlling the data order and model initialization. The output of the ensembling algorithm is a model that averages the logits of the K models, computed by querying all K models. More formally, we define the ensembling algorithm as

$$\mathcal{E}_{\mathcal{A}}(D, N, K, H) = \text{LogitAvg} \left(\{ \mathcal{A}(D, N, Z_i, H) \}_{i \in [K]} \right)$$

for randomness Z_i , where LogitAvg produces a model with likelihood for a sequence x given by

$$\text{LogitAvg} \left(M_{i \in [K]} \right) (x) \propto \exp \left(\frac{1}{K} \sum_{i \in [K]} \log (M(x)) \right)$$

The number of FLOPs needed to generate from or evaluate an ensemble is simply the sum of the costs for all members. Since the number of FLOPs in a forward pass is approximately linear in

¹For this power law and all following ones, the units of parameters and tokens will be in billions for cleaner visualization and comparison. This only affects the numerator of the scaling law, not its asymptote or exponent.

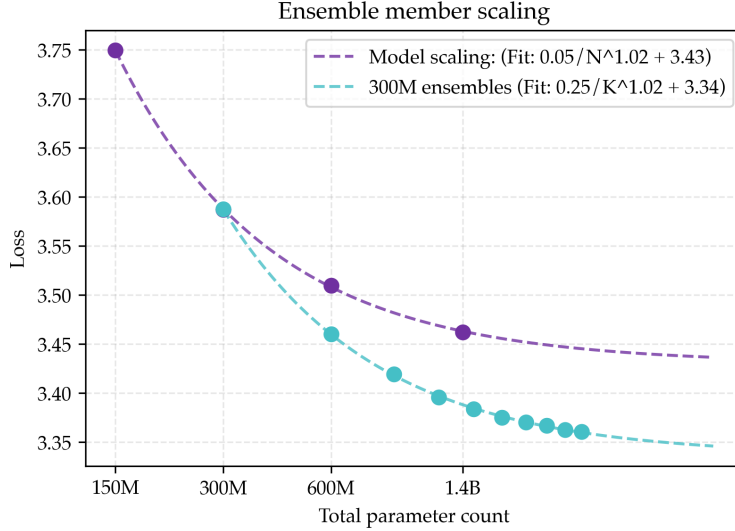


Figure 4: **Comparing scaling parameter count vs scaling ensemble member count.** Instead of scaling the parameter count of a single model, we can train an ensemble of smaller models and scale the number of ensemble members (resulting in NK total parameters for K ensemble members). Scaling up member count K can similarly be fit by a power law with exponent approximately 1. Importantly, this law achieves a better asymptote than scaling up parameter count.

parameter count [Hoffmann et al., 2022, Kaplan et al., 2020], we will consider an ensemble’s total parameter count as NK when comparing it to standard pre-training.

Scaling ensembles raises two significant questions. In Section 4.2, we ask whether the asymptote under $K \rightarrow \infty$ for fixed N outperforms the asymptote under $N \rightarrow \infty$ for $K = 1$. In Section 4.3, we ask whether we can compose both recipes by measuring the asymptote under $N, K \rightarrow \infty$.

4.2 Scaling member count instead of parameter count

We compare the regularized and ensembling recipes under the best regularized hyperparameters from Section 3. In Figure 4, we find that the excess loss for the ensembling recipe decreases close to a rate of $\frac{1}{K}$, similar to how the excess loss for the regularized recipe decreases at a rate close to $\frac{1}{N}$ in Figure 3. However, under the limit of infinite compute, the asymptote of ensembling ($N = 300\text{M}$, $K \rightarrow \infty$) is 3.34, which is lower than the asymptote of the regularized recipe ($N \rightarrow \infty$, $K = 1$), which is 3.43. This implies that for sufficiently large parameter counts, it is advantageous to train multiple small models (e.g. two 300M models) instead of a single large model (e.g. one 600M model). In fact, even the $K = 3$ ensemble outperforms the asymptote of the regularized recipe.

Why does ensembling improve over standard parameter scaling? Allen-Zhu and Li [2023] shows that ensembling helps when the data can be well-classified with one of many unique features but is best classified when using all such features. Under this “multi-view” structure, they find that training a single model is biased towards only learning one feature, whereas each member of an ensemble happens to learn different features when independently trained.

Given the success of ensembles, we study how to tune their hyperparameters. In Figure 4, the ensemble members were trained using the best hyperparameters for a single model, but this may

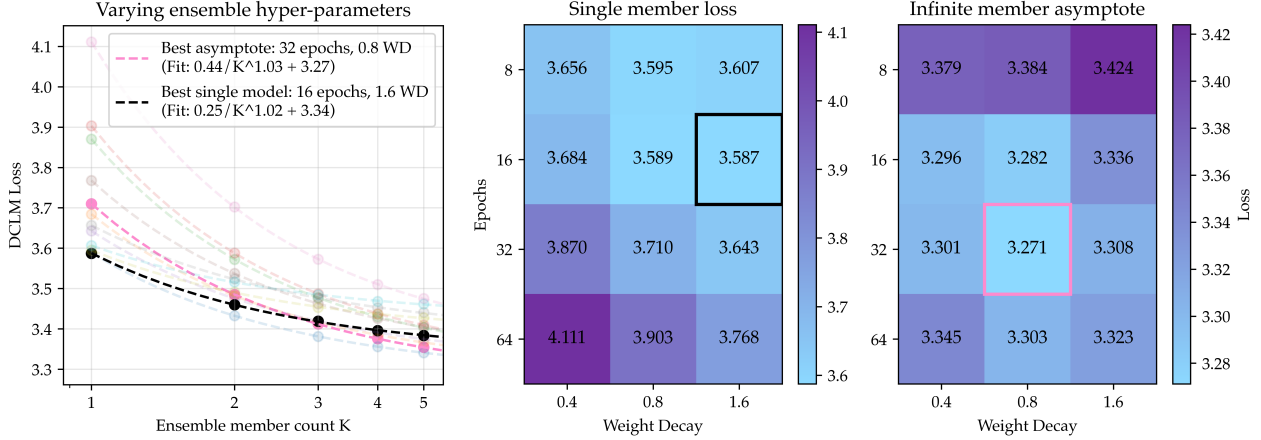


Figure 5: **Tuning hyperparameters of ensemble members for lowest asymptote under $K \rightarrow \infty$.** We construct ensembles for different K when varying epoch count and weight decay. We find that the ranking between hyperparameters changes across K (left) and that the infinite member asymptote benefiting from more epochs and less weight decay per member.

not be optimal for large K ensembles. More formally,

$$\underbrace{\operatorname{argmin}_H \mathcal{L}(\mathcal{A}(D, N, H)) = \operatorname{argmin}_H \mathcal{L}(\mathcal{E}_\mathcal{A}(D, N, K=1, H))}_{\text{best for single model}} \neq \underbrace{\operatorname{argmin}_H \lim_{K \rightarrow \infty} \mathcal{L}(\mathcal{E}_\mathcal{A}(D, N, K, H))}_{\text{best for ensemble member asymptote}}$$

In Figure 5, we train ensembles for $K \in [5]$ as we vary weight decay and epoch count². To estimate the best hyperparameters as $K \rightarrow \infty$, we fit a power law and refer to the asymptote. We find that the ranking between hyperparameters changes depending on K (Figure 5, left). Importantly, the optimal hyperparameters for $K = 1$ (black) are not the best asymptote under $K \rightarrow \infty$ (pink), which benefits from more epochs and less weight decay. This intuitively corresponds to each member being more overfit to the data, which might be helpful to learn different views of the data. Selecting the hyperparameters when considering $K \rightarrow \infty$ improves the ensembling asymptote from 3.34 to 3.27 (Figure 5, right).

Another design choice for our ensembles is the source of randomness Z_i . Though we vary both the data order and model initialization, we find that either one by itself obtains most of the benefits of ensembling and reserve a full analysis to Appendix C.1.

4.3 Joint scaling recipe composing parameter and ensemble scaling

Although the ensembling recipe outperforms parameter scaling, we can compose both by taking the number of members and the size of each member to infinity ($N, K \rightarrow \infty$). To estimate the best possible loss of a joint scaling recipe, we take two limits:

$$\hat{\mathcal{L}}_D = \lim_{N \rightarrow \infty} \lim_{K \rightarrow \infty} \min_H \mathcal{L}(\mathcal{E}_\mathcal{A}(D, N, K, H))$$

As long as $\min_H \mathcal{L}(\mathcal{E}_\mathcal{A}(D, N, K, H))$ monotonically decreases in N and K when the other variable is fixed, the value does not depend on the order we take the limits. We choose this particular order since we found it results in the most convenient hyperparameter tuning (discussed in Appendix C.4).

²We find that optimal learning rate is generally consistent across K , Appendix C.2

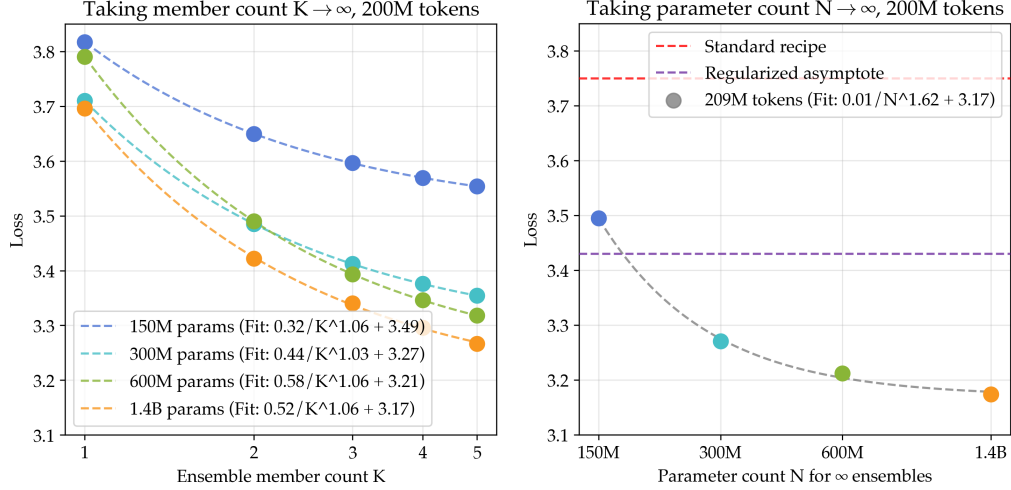


Figure 6: **Composing the regularized recipe and ensembling recipe under the double limit.** Left: For each parameter count, we fit a power law on the loss as K increases. We select hyperparameters that result in low asymptotes instead of loss at small K . Right: We take the asymptotes from the left plot and fit a power law to capture how the asymptote changes for ensembles of larger models. This asymptote estimates the best possible loss under the joint scaling recipe.

Due to pragmatic experimental constraints, we cannot search for locally optimal hyperparameters in the same way we did for parameter scaling. We instead estimate the best ensemble hyperparameters via the heuristic of the locally optimal regularized hyperparameters with $2 \times$ epochs and $0.5 \times$ weight decay, since we found this works best across almost all of our scales (Appendix C.2).

In Figure 6, we show how we take this double limit. On the left, we vary K for different fixed parameter counts N , fitting power laws that estimate loss as K increases. We then take the asymptotes of these four power laws and plot them on the right with respect to N . Though the asymptote of 150M ensembles is actually worse than the regularized asymptote, the ensemble asymptote decreases as N increases, beating the regularized asymptote for sufficiently large N . We then fit a second power law, allowing us to extrapolate how the estimated asymptote for $K \rightarrow \infty$ changes as we take $N \rightarrow \infty$. Our final estimate for the best possible loss is 3.17, which is a significant improvement over the loss of 3.43 for the regularized pre-training recipe and 3.75 for the unregularized recipe.

5 Scaling the seed token count under infinite compute

Our previous experiments show large improvements in loss for 200M tokens and we ask whether these findings generalize to larger token counts. To test this, we measure the best possible loss of the standard recipe, regularized recipe, and joint scaling recipe at higher token counts (Sections 5.1, 5.2, 5.3). To contextualize the loss improvements, we measure the **data efficiency** of a recipe by interpolating how much data the standard recipe would need to match its asymptote. At 200M seed tokens, we estimate that the regularized recipe and joint scaling recipe are $2.29 \times$ and $5.17 \times$ more data efficient than the standard recipe. We then fit data-scaling laws to our estimates of the best possible loss to extrapolate how recipes would perform for higher seed token counts. Although our tiered data scaling laws are bound to be noisy, they share similar exponents and asymptotes, which would imply that the data efficiency improvement is constant across all data scales (Section 5.4).

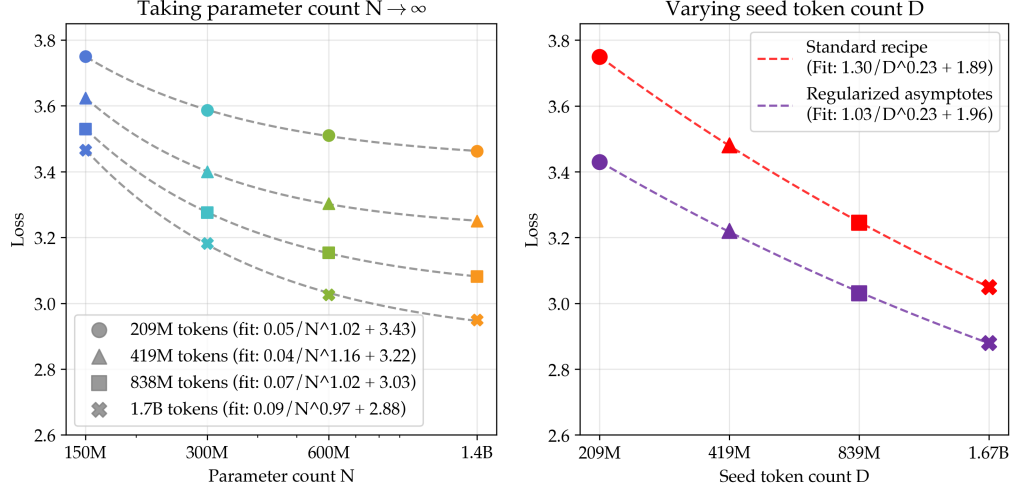


Figure 7: **Scaling the seed token count for single model pre-training.** We first consider the best possible loss of the standard recipe tuning epochs and parameters (red, right). We then consider the best possible loss of the regularized recipe by fitting parameter scaling laws across four token counts, shown on the left. The asymptotes of these laws form the purple points on the right. We further fit data-scaling laws to extrapolate performance as seed token count increases.

5.1 Tuning the standard recipe

We first detail how we construct the data scaling law for the standard recipe. For a given seed token count D , we estimate the best possible loss by tuning learning rate, epoch count, and parameter count. As we found in Section 2, increasing parameter count eventually results in validation loss increasing so we cannot build monotonic scaling laws. Therefore, we search for the best parameter count and hyperparameters for 200M, 400M, 800M, and 1.6B tokens (detailed in Appendix D.1). These four losses form the red points in Figure 7, right. Given these four estimates of the best possible loss at each token count, we can fit the data scaling power law shown as the red line using

$$\hat{\mathcal{L}}_D := \frac{A}{D^\alpha} + E$$

5.2 Scaling parameter count

We now characterize the best possible loss of the regularized recipe by estimating $\lim_{N \rightarrow \infty} \min_H \mathcal{L}(\mathcal{A}(D, N, H))$ via the asymptote of the power law as described in Section 3. Since we now have to compute asymptotes to build the points for the data scaling law, we follow a two step procedure as visualized in Figure 7.

1. **Varying parameter count (left plot).** We first follow the locally-optimal hyperparameter search detailed in Section 3 to find the best possible models for four parameter counts across four seed token counts. These are the 16 points in Figure 7, left.
2. **Varying token count (right plot).** For each seed token count, we can fit a power law and use the asymptote E_D as an estimate of $\mathcal{L}_D^*$. These four (D, E_D) tuples form the purple points in Figure 7, right. We then fit a data scaling power law over these asymptotes, shown as the purple line.

Measuring data efficiency. Since we are data-constrained, we care about the loss improvement at a fixed value of D , unlike our compute scaling laws where we care about the asymptote. Visually inspecting the plot suggests that the standard recipe would need approximately twice as much data to match the performance of regularized scaling across all of our token scales. To formalize the data efficiency improvement of algorithm $\mathcal{A}_2$ over algorithm $\mathcal{A}_1$ at a token count D , we compute the effective data D' that $\mathcal{A}_1$ would need to match $\mathcal{A}_2$. After interpolating D' via the data scaling law of $\mathcal{A}_1$, we report the data efficiency improvement as $\frac{D'}{D}$. This metric characterizes the asymptote of the regularized recipe as $2.29\times$ more data efficient than the standard recipe at 200M tokens. Since our tiered asymptote estimation can be unreliable, we note that even without any extrapolation, the best 1.4B model at 200M tokens is $2.09\times$ more data efficient than our baseline.

5.3 Scaling member and parameter count

We repeat the above procedure for ensembles by taking $\lim_{N \rightarrow \infty} \lim_{K \rightarrow \infty} \min_H \mathcal{L}(\mathcal{E}_{\mathcal{A}}(D, N, K, H))$ for each seed token count D following Section 4.3. This results in the following three step procedure, visualized in Figure 8.

1. **Varying parameter count (left plot).** In Figure 8, left, we picture the losses of ensembles across our 4 token counts, 4 parameter counts, and 5 member counts with hyperparameters selected for better asymptotes. For each of the 16 pairs of D and N , we use a power law to extrapolate the performance as we take member count $K \rightarrow \infty$.
2. **Varying ensemble member count (middle plot).** The asymptotes of the above 16 limits are visualized in Figure 8, middle. Given these asymptotes, we can fit a second set of 4 power laws that extrapolate the loss as we take parameter count $N \rightarrow \infty$.
3. **Varying token count (right plot).** The asymptotes of these 4 limits are visualized by the gold points on Figure 8, right. We then fit a data scaling law, depicted by the gold line.

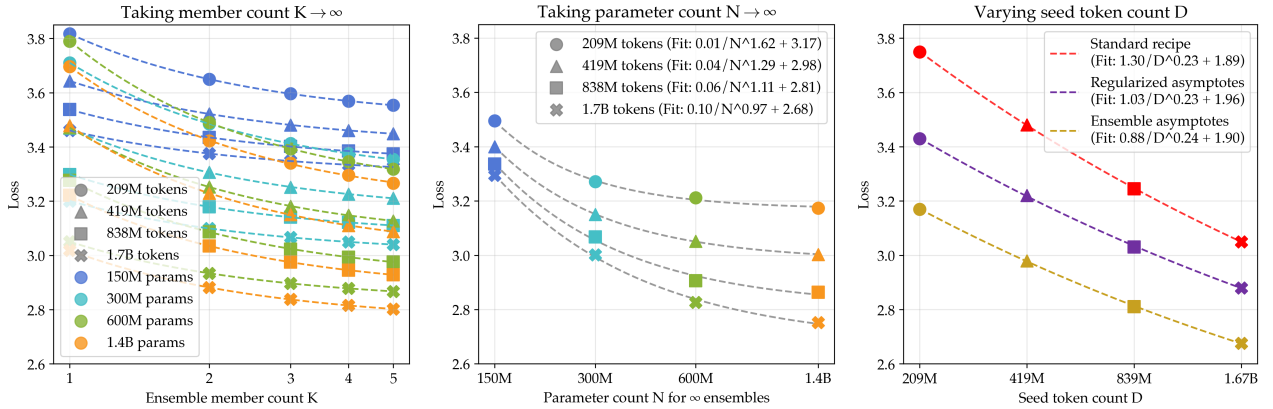


Figure 8: Scaling the seed token count for ensembles. Left: For a fixed parameter count and token count, we fit a power law in K , with hyperparameters optimized for the asymptote. Middle: We take the asymptote of the 16 laws on the left and fit a power law to measure how the asymptote changes in N . Right: We take the asymptote of the 4 laws in the middle and fit a power law to measure how the asymptote of asymptotes changes in D . We find over $2\times$ data efficiency wins over the regularized recipe and $5\times$ data efficiency wins over the standard recipe at all tested token counts.

At 200M tokens, the asymptote of the joint scaling recipe is $5.17\times$ more data efficient than the standard recipe. Without taking asymptotes, our best ensemble of five 1.4B models is itself $3.75\times$ more data efficient.

5.4 Scaling data

We compare the data scaling laws across our three recipes to measure whether the data efficiency improvements persist across token count. Though the data scaling laws are expected to be noisy, these laws predict that all three scaling recipes decay at a similar rate with exponents between 0.23 and 0.24 and asymptotes between 1.89 and 1.96. Asymptotic statistics suggests that the asymptotes are equal if the algorithms achieve Bayes-optimal error under infinite data and compute, in which case their loss would be the entropy of text [Shannon, 1951, Van der Vaart, 2000]. In the case where the asymptote E and exponent α of the laws are the same for algorithms A_1, A_2 , there is a constant data efficiency improvement at all token counts determined by the numerators A_1, A_2 , equal to $(A_2/A_1)^{\frac{1}{\alpha}}$. Our preliminary analysis suggests that our data efficiency improvements will not disappear across all data scales even if they perform similarly under infinite data.

6 Data efficiency under parameter constraints

The asymptotes of the regularized and ensembling recipes rely on arbitrarily high parameter models. For example, the smallest ensemble that achieves a loss of 3.37 needs 1.2B total parameters. In this section, we investigate whether high parameter counts are necessary for data efficiency, either for the final model or for training. In Section 6.1, we show that we can distill an ensemble of 2.4B total parameters into a 300M student, preserving 83% of the loss improvement without increasing the parameter count of the final model. In Section 6.2, we show that by self-distilling a 300M teacher model into a student model of the same parameter count and architecture, we can train a student that outperforms the teacher, removing the need for large parameter counts at training.

6.1 Reducing final parameter count via ensemble distillation

So far, the benefit of ensembling is only realized at a high enough parameter count. Therefore, even if our best scaling recipe helps in the limit as $N, K \rightarrow \infty$, it is not immediately clear whether the scaling recipe helps train models that are small relative to D . However, it is known that better large models can improve the performance of smaller models through knowledge distillation [Hinton et al., 2015]. In fact, many smaller models today are pre-trained through distilling large models [Goyal et al., 2025, Grattafiori et al., 2024, Team et al., 2025a, Yang et al., 2025]. Since we are not bound by train compute, we can first pre-train a data-efficient teacher M' using our existing recipes and then distill M' to train a student M via sequence-level knowledge distillation [Kim and Rush, 2016]:

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

In Figure 9, we show the student model (pink star) obtained from using an 8-ensemble of 300M models (right-most blue point) with loss 3.32. Despite the $8\times$ smaller inference compute budget, our distillation procedure attains a loss of 3.36, preserving 83% of the ensemble improvement over the regularized 300M model loss of 3.57 (purple point). Our student outperforms the regularized recipe asymptote and even matches the loss of a 4-ensemble of 300M models (details in Appendix E).

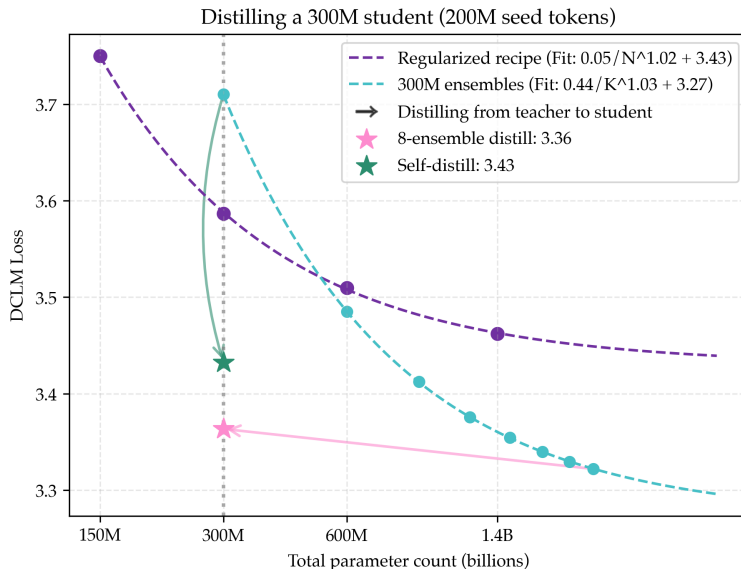


Figure 9: **Ensemble distillation and self-distillation.** We can compress our data efficiency gains into smaller models through distillation. Distilling an 8-ensemble teacher into a 300M student retains most of the loss improvement (pink star) and outperforms the asymptote of model scaling. We also find that self-distillation with a 300M teacher and 300M student (green star) is surprisingly effective, matching the asymptote of the regularized recipe without increasing parameter count at training.

6.2 Reducing train parameter count via self-distillation

The success of ensemble distillation shows how the final model does not need to have a high parameter count and pushes the loss-vs-parameter tradeoff to the left. Is it possible to train a small model (i.e. 300M) better than our recipes without high parameter count at train time as well? We consider this question in the context of self-distillation where the teacher and student are of the same size and architecture. A priori, it seems impossible for a student to outperform its teacher following arguments such as the data processing inequality. In fact, recent papers discuss how training a new student model on model generations can result in model degradation via model collapse [Dohmatob et al., 2024, Gerstgrasser et al., 2024, Shumailov et al., 2024, Taori and Hashimoto, 2022].

On the contrary, we find that correctly distilling a 300M teacher into a fresh student of the same architecture can *improve* loss. Crucially, by mixing together the D real tokens and D' synthetic tokens, we find that we avoid collapse and train a student that vastly outperforms its teacher. In Figure 9, we show how using a single 300M model as a teacher (blue point) results in a 300M student model (green star) that outperforms the best 300M model from standard pre-training (purple point).

Why does self-distillation help? Allen-Zhu and Li [2023] provide theory interpreting self-distillation as implicitly ensembling the teacher and the freshly initialized student. The connection between ensembling and self-distillation is further reflected in their strong empirical performance. Beyond just our experiments, recent work suggests that synthetic data augmented from the original pre-training data can provide data efficiency wins [Allen-Zhu and Li, 2024, Maini et al., 2024, Ruan et al., 2025, Su et al., 2025, Team et al., 2025b, Yang et al., 2024]. We view self-distillation as a different type of synthetic data that does not require human priors such as hand-crafted invariances, reward signals, or prompting strategies, making it appealing for scalable data-constrained pre-training.

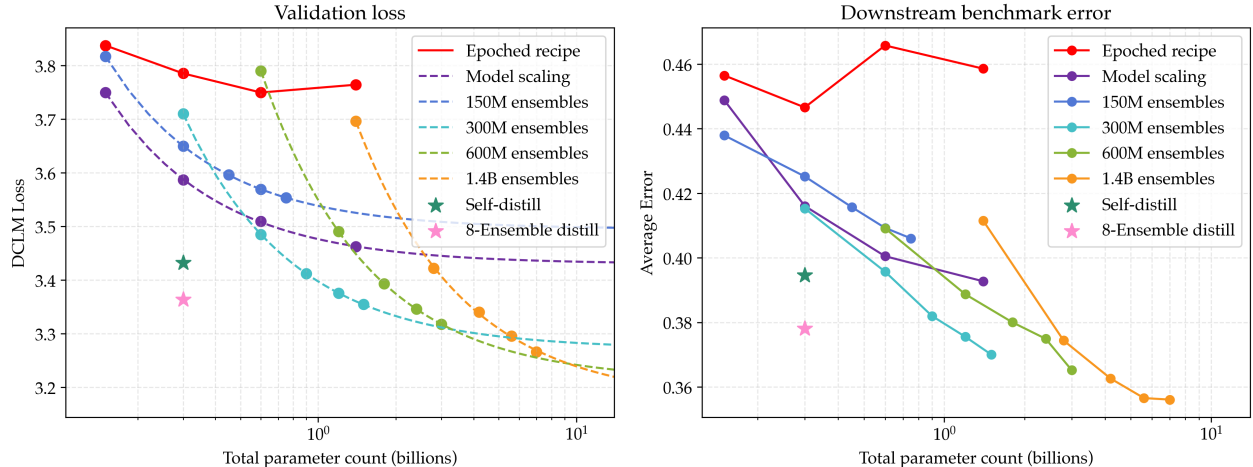


Figure 10: **Performance of pre-trained models on downstream tasks.** We have thus far been using validation loss (left) to separate whether models are better pre-trained models or not. We evaluate the same models and ensembles on downstream benchmarks (right). Models with lower validation loss have lower average error across downstream benchmarks.

7 Downstream tasks

7.1 Downstream benchmarks

So far, we have only analyzed pre-training validation loss as a proxy for model quality. Although loss is known to correlate with capabilities of interest [Chen et al., 2025, Gadre et al., 2024, Thrush et al., 2025], it may not be perfectly reflective. Therefore, we test our model’s general capabilities using downstream benchmarks. Since we need evaluations that are informative for models at our scale, we take all the accuracy-based benchmarks from Thrush et al. [2025], namely PIQA [Bisk et al., 2019], SciQ [Welbl et al., 2017], and ARC Easy [Clark et al., 2018]. Notably, we did not evaluate on *any* benchmarks until the end of the project after we selected the best recipes following validation loss, making these benchmarks a strong test of generalization.

In Figure 10, we show both the validation loss (left) and downstream benchmark error (right) of our pre-trained models, ensembles, and distilled models for 200M tokens. Without our intervention of regularization, the standard recipe does not strongly benefit from parameter scaling. Adding in regularization (purple points) makes downstream accuracy scale smoothly with diminishing returns, similar to validation loss. For ensembles, we observe similar trends as validation loss with increasing N and K improving performance. Furthermore, our distilled models improve upon all other 300M parameter models. Overall, our best ensemble outperforms our best unregularized model by over 9% on average and our best distilled model outperforms the unregularized 300M model by around 7%, marking a large improvement over standard pre-training. A full breakdown of results is available in Appendix F.

7.2 Continued pre-training

We demonstrate the immediate applicability of our findings in settings outside of pre-training from scratch by improving the data efficiency of existing CPT recipes. We adopt the setup from Wang et al. [2025] of performing continued pre-training on Llama 3.2 3B Base [Grattafiori et al., 2024]

with the MegaMath-Web-Pro mid-training dataset. To simulate a data-constrained setting, we restrict ourselves to only 4B seed tokens of the full 73B tokens. We evaluate math performance by accuracy on a subset of representative benchmarks from Wang et al. [2025]: GSM8K [Cobbe et al., 2021], MATH [Hendrycks et al., 2021], and MathQA [Amini et al., 2019].

In Table 1, we show that many of our results for pre-training from scratch transfer to continued pre-training setting. We start with a CPT baseline (Default) from the reference hyperparameters of Wang et al. [2025] which gives a 6.34% improvement over the base model. We then apply our interventions from Appendix B.4 and Section 3 by decreasing batch size and epoching, and we find that weight decay was not helpful for CPT. Our final single model baseline provides an additional 5.23% lift in average accuracy on top of the reference hyperparameters. See Appendix G for details.

We further show that ensembling eight epoched models provides an additional 4.76% gain in average accuracy over just a single epoched CPT model ($K = 1$). We observe that average accuracy scales with increasing ensemble member count, and our best ensembles exceed the performance of a baseline continually pre-trained on the full 73B tokens, providing a $17.5\times$ data efficiency win.

Table 1: **Data efficiency improvements on OctoThinker.** We take reasoning mid-training data from Wang et al. [2025] and apply our data efficiency interventions of reducing batch size (Appendix B.4), epoching data, and ensembling multiple models. Our best ensemble utilizing 4B tokens outperforms vanilla CPT on 73B tokens following the original paper’s training hyperparameters, resulting in a $17.5\times$ data efficiency improvement.

Benchmarks	Llama 3B	CPT (4B tokens)			K -ensembles			CPT (73B tokens)
		Default	Lower BS	Epoching ($K = 1$)	$K = 2$	$K = 4$	$K = 8$	
GSM8K _(8-shot)	28.23	38.44	44.50	44.05	49.28	51.80	52.99	49.51
MATH _(4-shot)	6.90	14.38	17.64	19.74	21.84	23.04	23.50	23.40
MATHQA _(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

8 Related Work

Scaling laws. Much of the success of modern language model pre-training was built upon scaling laws which accurately predict performance at a given resource budget [Cortes et al., 1993, Henighan et al., 2020, Hestness et al., 2017, 2019, Hoffmann et al., 2022, Kaplan et al., 2020, Rosenfeld et al., 2019, Ruan et al., 2024, Sorscher et al., 2023]. Past work has studied the scaling properties of machine learning under various constraints such as data and compute [Goyal et al., 2024, Muennighoff et al., 2023], hardware precision [Kumar et al., 2024], parameter count [Gadre et al., 2024, Sardana et al., 2025, Springer et al., 2025], and test-time compute [Brown et al., 2024, Snell et al., 2024]. We first show that scaling laws in past work [Muennighoff et al., 2023] do not account for over-fitting of standard recipes with too many epochs and rectify this via regularization. We also show that even though early work in double descent suggests that over-parameterized deep learning does not have clean scaling laws due to double descent [Belkin et al., 2019, Hastie et al., 2020, Nakkiran et al., 2019], we can get clean scaling via tuning regularization, agreeing with theory in over-parameterized regression [Advani and Ganguli, 2016, Canatar et al., 2021, Nakkiran et al., 2021, Simon et al., 2024]. Finally, we propose asymptote estimation of scaling laws as a new metric to evaluate the performance of pretrained models under infinite compute.

Ensembling. Ensembling is a well-established algorithm that boosts performance across many settings [Dietterich, 2000]. Past work has established the success of ensembling deep networks for uncertainty estimation [Lakshminarayanan et al., 2017], image classification [Garipov et al., 2018, Huang et al., 2017], and reinforcement learning [van Hasselt et al., 2015]. Ensembling has been shown to have surprising connections to other successful techniques including distillation [Allen-Zhu and Li, 2023] and residual networks [Veit et al., 2016] and are shown to follow power laws [Lobacheva et al., 2021]. On the other hand, ensembling is sometimes not believed to outperform parameter scaling in certain theoretical models of scaling [Ruben et al., 2024, Vyas et al., 2023]. We show how the simplest form of ensembling can be adopted for pre-training and then build scaling laws to characterize their loss. See Appendix C.3 for further discussion on related alternatives including Mixture of Experts and weight-averaging.

Distillation. Distillation spends compute to produce strong models with lower inference and fine-tuning costs [Hinton et al., 2015]. Though we use sequence knowledge distillation [Kim and Rush, 2016] as a preliminary demonstration, there are many better distillation algorithms such as using more supervision via logits [Sanh et al., 2020] and minimizing different divergences between the teacher and student [Agarwal et al., 2024, Gu et al., 2024]. Past work has further quantified the scaling properties of distillation [Busbridge et al., 2025], and distillation is increasingly used to pre-train the most performant small language models [Goyal et al., 2025, Team et al., 2025a, Yang et al., 2025]. For self-distillation, there is recent work showing how training on self-generated inputs can be harmful [Dohmatob et al., 2024, Shumailov et al., 2024, Taori and Hashimoto, 2022]. Gerstgrasser et al. [2024] is the most optimistic work, suggesting that training on self-generated data can be helpful in limited scenarios, though their comparisons are neither compute-matched nor data-matched. We find that self-distillation can actually be helpful over optimal regularized pre-training if done correctly, aligned with prior evidence from data-constrained deep learning [Mobahi et al., 2020, Zhang et al., 2019]. Notably, Allen-Zhu and Li [2023] draws a connection between how self-distillation can be viewed as implicitly performing ensembling and distillation. We can interpret distillation as a form of synthetic data, and unlike recent work on synthetic data to improve data efficiency [Allen-Zhu and Li, 2024, Maini et al., 2024, Ruan et al., 2025, Su et al., 2025, Team et al., 2025b, Yang et al., 2024], distillation requires minimal human priors such as a trusted reward function or known augmentation invariance.

Classical data-constrained deep learning. Historically, many machine learning benchmarks before the era of large language models were data-constrained [Deng et al., 2009, Lecun et al., 1998, Marcus et al., 1993, Warstadt et al., 2023], resulting in many effective algorithms. For example, the best models on Penn Tree Bank utilized dynamic evaluation [Krause et al., 2017, Mikolov et al., 2010], ensembling and model averaging [Takase et al., 2018, Zaremba et al., 2015], regularization (drop-out, weight decay, weight tying, lower batch size) [Gal and Ghahramani, 2016, Merity et al., 2017, Zaremba et al., 2015], data augmentation [Shi et al., 2021, Xie et al., 2017], and novel architectures [Grave et al., 2016, Yang et al., 2018, Zilly et al., 2017]. We revisit a few of these algorithms and advocate for future work to explore all others.

Modern data-constrained pre-training. There are several more recent works which study interventions for data-efficient pre-training. These works have shown the benefit of epoching [Muennighoff et al., 2023], rephrased synthetic data [DatologyAI et al., 2025, Maini et al., 2024, Ruan et al., 2024, Yang et al., 2024], diffusion language models [Ni et al., 2025, Prabhudesai et al., 2025], and energy-based models [Gladstone et al., 2025]. Some of these methods such as rephrased synthetic data have been adopted to train the strongest open-source models [Team et al., 2025b, Yang et al., 2025].

This collection of recent work does not fully leverage additional compute: most of the papers do not optimally epoch the models and none of the papers tune regularization, build scaling laws to estimate the infinite parameter limits, or build scaling laws as data increases to estimate data efficiency. We also hope that the performance gains we demonstrate through ensembling and self-distillation are orthogonal to those of existing work and can compose for better performance.

9 Discussion

Notably, the algorithms we have considered (parameter scaling, regularization, ensembling, distillation) mirror classical techniques from when deep learning was data-constrained by limited sentences [Marcus et al., 1993, Warstadt et al., 2023], images [Deng et al., 2009, Lecun et al., 1998], etc. The success of such simple methods suggests that there is free lunch on the table for data-efficient pre-training, encouraging us to revisit basic decisions such as objective [Ni et al., 2025, Prabhudesai et al., 2025], architecture [Gladstone et al., 2025], and data augmentation [Maini et al., 2024]. Since available compute grows far more quickly than available data, we are excited by algorithms that can better leverage additional compute for better performance, in line with The Bitter Lesson [Sutton, 2019]. We hope that by carefully evaluating scaling recipes via their asymptotes instead of their compute-constrained performance, we can design algorithms that are better prepared for a data-constrained future.

10 Acknowledgements

We thank Steven Cao, Sam Park, Jacob Mitchell Springer, Kaiyue Wen, Yu Sun, Nathan Hu, Meena Jagadeesan, Luke Bailey, Neil Band, Sally Zhu, Ben Spector, and Audrey Xie for their helpful discussions or feedback on the paper draft.

This work is a part of the [Marin Project](#) and the compute is supported by the Google TPU Research Cloud (TRC). TH was supported by a grant by HAI, DSO labs, gifts from Open Philanthropy, Amazon, Schmidt Sciences, the Tianqiao and Chrissy Chen Foundation and a grant under the NSF CAREER IIS-2338866, ONR N00014-24-1-2609, and DARPA Cooperative Agreement HR00112520013. PL was supported by DARPA Cooperative Agreement HR00112520013. This work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred.

References

- M. Advani and S. Ganguli. Statistical mechanics of optimal convex inference in high dimensions. *Phys. Rev. X*, 6:031034, Aug 2016. doi: 10.1103/PhysRevX.6.031034. URL <https://link.aps.org/doi/10.1103/PhysRevX.6.031034>.
- R. Agarwal, N. Vieillard, Y. Zhou, P. Stanczyk, S. Ramos, M. Geist, and O. Bachem. On-policy distillation of language models: Learning from self-generated mistakes, 2024. URL <https://arxiv.org/abs/2306.13649>.
- S. K. Ainsworth, J. Hayase, and S. Srinivasa. Git re-basin: Merging models modulo permutation symmetries, 2023. URL <https://arxiv.org/abs/2209.04836>.
- Z. Allen-Zhu and Y. Li. Towards understanding ensemble, knowledge distillation and self-distillation in deep learning, 2023. URL <https://arxiv.org/abs/2012.09816>.
- Z. Allen-Zhu and Y. Li. Physics of language models: Part 3.1, knowledge storage and extraction, 2024. URL <https://arxiv.org/abs/2309.14316>.
- A. Amini, S. Gabriel, P. Lin, R. Koncel-Kedziorski, Y. Choi, and H. Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms, 2019. URL <https://arxiv.org/abs/1905.13319>.
- M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, July 2019. ISSN 1091-6490. doi: 10.1073/pnas.1903070116. URL <http://dx.doi.org/10.1073/pnas.1903070116>.
- T. Besiroglu, E. Erdil, M. Barnett, and J. You. Chinchilla scaling: A replication attempt, 2024. URL <https://arxiv.org/abs/2404.10102>.
- Y. Bisk, R. Zellers, R. L. Bras, J. Gao, and Y. Choi. Piqa: Reasoning about physical commonsense in natural language, 2019. URL <https://arxiv.org/abs/1911.11641>.
- B. Brown, J. Juravsky, R. Ehrlich, R. Clark, Q. V. Le, C. Ré, and A. Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling, 2024. URL <https://arxiv.org/abs/2407.21787>.
- T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020. URL <https://arxiv.org/abs/2005.14165>.
- D. Busbridge, A. Shidani, F. Weers, J. Ramapuram, E. Littwin, and R. Webb. Distillation scaling laws, 2025. URL <https://arxiv.org/abs/2502.08606>.
- A. Canatar, B. Bordelon, and C. Pehlevan. Spectral bias and task-model alignment explain generalization in kernel regression and infinitely wide neural networks. *Nature Communications*, 12(1), May 2021. ISSN 2041-1723. doi: 10.1038/s41467-021-23103-1. URL <http://dx.doi.org/10.1038/s41467-021-23103-1>.
- Y. Chen, B. Huang, Y. Gao, Z. Wang, J. Yang, and H. Ji. Scaling laws for predicting downstream performance in llms, 2025. URL <https://arxiv.org/abs/2410.08527>.

- P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge, 2018. URL <https://arxiv.org/abs/1803.05457>.
- K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems, 2021. URL <https://arxiv.org/abs/2110.14168>.
- C. Cortes, L. D. Jackel, S. Solla, V. Vapnik, and J. Denker. Learning curves: Asymptotic values and rate of convergence. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1993. URL https://proceedings.neurips.cc/paper_files/paper/1993/file/1aa48fc4880bb0c9b8a3bf979d3b917e-Paper.pdf.
- F. D’Angelo, M. Andriushchenko, A. Varre, and N. Flammarion. Why do we need weight decay in modern deep learning?, 2024. URL <https://arxiv.org/abs/2310.04415>.
- DatologyAI, :, P. Maini, V. Dorna, P. Doshi, A. Carranza, F. Pan, J. Urbanek, P. Burstein, A. Fang, A. Deng, A. Abbas, B. Larsen, C. Blakeney, C. Bannur, C. Baek, D. Teh, D. Schwab, H. Mongstad, H. Yin, J. Wills, K. Mentzer, L. Merrick, R. Monti, R. Adiga, S. Joshi, S. Das, Z. Wang, B. Gaza, A. Morcos, and M. Leavitt. Beyondweb: Lessons from scaling synthetic data for trillion-scale pretraining, 2025. URL <https://arxiv.org/abs/2508.10975>.
- J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.
- T. G. Dietterich. Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems*, MCS ’00, page 1–15, Berlin, Heidelberg, 2000. Springer-Verlag. ISBN 3540677046.
- E. Dohmatob, Y. Feng, A. Subramonian, and J. Kempe. Strong model collapse, 2024. URL <https://arxiv.org/abs/2410.04840>.
- K. Everett, L. Xiao, M. Wortsman, A. A. Alemi, R. Novak, P. J. Liu, I. Gur, J. Sohl-Dickstein, L. P. Kaelbling, J. Lee, and J. Pennington. Scaling exponents across parameterizations and optimizers, 2024. URL <https://arxiv.org/abs/2407.05872>.
- S. Y. Gadre, G. Smyrnis, V. Shankar, S. Gururangan, M. Wortsman, R. Shao, J. Mercat, A. Fang, J. Li, S. Keh, R. Xin, M. Nezhurina, I. Vasiljevic, J. Jitsev, L. Soldaini, A. G. Dimakis, G. Ilharco, P. W. Koh, S. Song, T. Kollar, Y. Carmon, A. Dave, R. Heckel, N. Muennighoff, and L. Schmidt. Language models scale reliably with over-training and on downstream tasks, 2024. URL <https://arxiv.org/abs/2403.08540>.
- Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks, 2016. URL <https://arxiv.org/abs/1512.05287>.
- L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac’h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou. The language model evaluation harness, 07 2024. URL <https://zenodo.org/records/12608602>.

- T. Garipov, P. Izmailov, D. Podoprikin, D. Vetrov, and A. G. Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns, 2018. URL <https://arxiv.org/abs/1802.10026>.
- M. Gerstgrasser, R. Schaeffer, A. Dey, R. Rafailov, H. Sleight, J. Hughes, T. Korbak, R. Agrawal, D. Pai, A. Gromov, D. A. Roberts, D. Yang, D. L. Donoho, and S. Koyejo. Is model collapse inevitable? breaking the curse of recursion by accumulating real and synthetic data, 2024. URL <https://arxiv.org/abs/2404.01413>.
- A. Gladstone, G. Nanduru, M. M. Islam, P. Han, H. Ha, A. Chadha, Y. Du, H. Ji, J. Li, and T. Iqbal. Energy-based transformers are scalable learners and thinkers, 2025. URL <https://arxiv.org/abs/2507.02092>.
- S. Goyal, P. Maini, Z. C. Lipton, A. Raghunathan, and J. Z. Kolter. Scaling laws for data filtering – data curation cannot be compute agnostic, 2024. URL <https://arxiv.org/abs/2404.07177>.
- S. Goyal, D. Lopez-Paz, and K. Ahuja. Distilled pretraining: A modern lens of data, in-context learning and test-time scaling, 2025. URL <https://arxiv.org/abs/2509.01649>.
- A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Roziere, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Wyatt, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Guzmán, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Thattai, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. Kloumann, I. Misra, I. Evtimov, J. Zhang, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Prasad, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, K. El-Arini, K. Iyer, K. Malik, K. Chiu, K. Bhalla, K. Lakhotia, L. Rantala-Young, L. van der Maaten, L. Chen, L. Tan, L. Jenkins, L. Martin, L. Madaan, L. Malo, L. Blecher, L. Landzaat, L. de Oliveira, M. Muzzi, M. Pasupuleti, M. Singh, M. Paluri, M. Kardas, M. Tsimpoukelli, M. Oldham, M. Rita, M. Pavlova, M. Kambadur, M. Lewis, M. Si, M. K. Singh, M. Hassan, N. Goyal, N. Torabi, N. Bashlykov, N. Bogoychev, N. Chatterji, N. Zhang, O. Duchenne, O. Çelebi, P. Alrassy, P. Zhang, P. Li, P. Vasic, P. Weng, P. Bhargava, P. Dubal, P. Krishnan, P. S. Koura, P. Xu, Q. He, Q. Dong, R. Srinivasan, R. Ganapathy, R. Calderer, R. S. Cabral, R. Stojnic, R. Raileanu, R. Maheswari, R. Girdhar, R. Patel, R. Sauvestre, R. Polidoro, R. Sumbaly, R. Taylor, R. Silva, R. Hou, R. Wang, S. Hosseini, S. Chennabasappa, S. Singh, S. Bell, S. S. Kim, S. Edunov, S. Nie, S. Narang, S. Raparthy, S. Shen, S. Wan, S. Bhosale, S. Zhang, S. Vandenhende, S. Batra, S. Whitman, S. Sootla, S. Collot, S. Gururangan, S. Borodinsky, T. Herman, T. Fowler, T. Sheasha, T. Georgiou, T. Scialom, T. Speckbacher, T. Mihaylov, T. Xiao, U. Karn, V. Goswami, V. Gupta, V. Ramanathan, V. Kerkez, V. Gonguet, V. Do, V. Vogeti, V. Albiero, V. Petrovic, W. Chu, W. Xiong, W. Fu, W. Meers, X. Martinet, X. Wang, X. Wang, X. E. Tan, X. Xia, X. Xie, X. Jia, X. Wang, Y. Goldschlag, Y. Gaur, Y. Babaei, Y. Wen, Y. Song, Y. Zhang, Y. Li, Y. Mao, Z. D. Coudert, Z. Yan, Z. Chen, Z. Papakipos, A. Singh, A. Srivastava, A. Jain, A. Kelsey, A. Shajnfeld, A. Gangidi, A. Victoria, A. Goldstand, A. Menon, A. Sharma, A. Boesenberg, A. Baevski, A. Feinstein, A. Kallet, A. Sangani, A. Teo, A. Yunus, A. Lupu, A. Alvarado, A. Caples, A. Gu, A. Ho, A. Poulton, A. Ryan, A. Ramchandani, A. Dong, A. Franco, A. Goyal, A. Saraf, A. Chowdhury, A. Gabriel, A. Bharambe, A. Eisenman, A. Yazdan, B. James, B. Maurer, B. Leonhardi, B. Huang, B. Loyd, B. D. Paola, B. Paranjape, B. Liu,

- B. Wu, B. Ni, B. Hancock, B. Wasti, B. Spence, B. Stojkovic, B. Gamido, B. Montalvo, C. Parker, C. Burton, C. Mejia, C. Liu, C. Wang, C. Kim, C. Zhou, C. Hu, C.-H. Chu, C. Cai, C. Tindal, C. Feichtenhofer, C. Gao, D. Civin, D. Beaty, D. Kreymer, D. Li, D. Adkins, D. Xu, D. Testuggine, D. David, D. Parikh, D. Liskovich, D. Foss, D. Wang, D. Le, D. Holland, E. Dowling, E. Jamil, E. Montgomery, E. Presani, E. Hahn, E. Wood, E.-T. Le, E. Brinkman, E. Arcaute, E. Dunbar, E. Smothers, F. Sun, F. Kreuk, F. Tian, F. Kokkinos, F. Ozgenel, F. Caggioni, F. Kanayet, F. Seide, G. M. Florez, G. Schwarz, G. Badeer, G. Swee, G. Halpern, G. Herman, G. Sizov, Guangyi, Zhang, G. Lakshminarayanan, H. Inan, H. Shojanazeri, H. Zou, H. Wang, H. Zha, H. Habeeb, H. Rudolph, H. Suk, H. Aspegren, H. Goldman, H. Zhan, I. Damla, I. Molybog, I. Tufanov, I. Leontiadis, I.-E. Veliche, I. Gat, J. Weissman, J. Geboski, J. Kohli, J. Lam, J. Asher, J.-B. Gaya, J. Marcus, J. Tang, J. Chan, J. Zhen, J. Reizenstein, J. Teboul, J. Zhong, J. Jin, J. Yang, J. Cummings, J. Carvill, J. Shepard, J. McPhie, J. Torres, J. Ginsburg, J. Wang, K. Wu, K. H. U, K. Saxena, K. Khandelwal, K. Zand, K. Matosich, K. Veeraraghavan, K. Michelena, K. Li, K. Jagadeesh, K. Huang, K. Chawla, K. Huang, L. Chen, L. Garg, L. A. L. Silva, L. Bell, L. Zhang, L. Guo, L. Yu, L. Moshkovich, L. Wehrstedt, M. Khabsa, M. Avalani, M. Bhatt, M. Mankus, M. Hasson, M. Lennie, M. Reso, M. Groshev, M. Naumov, M. Lathi, M. Keneally, M. Liu, M. L. Seltzer, M. Valko, M. Restrepo, M. Patel, M. Vyatskov, M. Samvelyan, M. Clark, M. Macey, M. Wang, M. J. Hermoso, M. Metanat, M. Rastegari, M. Bansal, N. Santhanam, N. Parks, N. White, N. Bawa, N. Singhal, N. Egebo, N. Usunier, N. Mehta, N. P. Laptev, N. Dong, N. Cheng, O. Chernoguz, O. Hart, O. Salpekar, O. Kalinli, P. Kent, P. Parekh, P. Saab, P. Balaji, P. Rittner, P. Bontrager, P. Roux, P. Dollar, P. Zvyagina, P. Ratanchandani, P. Yuvraj, Q. Liang, R. Alao, R. Rodriguez, R. Ayub, R. Murthy, R. Nayani, R. Mitra, R. Parthasarathy, R. Li, R. Hogan, R. Battey, R. Wang, R. Howes, R. Rinott, S. Mehta, S. Siby, S. J. Bondu, S. Datta, S. Chugh, S. Hunt, S. Dhillon, S. Sidorov, S. Pan, S. Mahajan, S. Verma, S. Yamamoto, S. Ramaswamy, S. Lindsay, S. Lindsay, S. Feng, S. Lin, S. C. Zha, S. Patil, S. Shankar, S. Zhang, S. Zhang, S. Wang, S. Agarwal, S. Sajuyigbe, S. Chintala, S. Max, S. Chen, S. Kehoe, S. Satterfield, S. Govindaprasad, S. Gupta, S. Deng, S. Cho, S. Virk, S. Subramanian, S. Choudhury, S. Goldman, T. Remez, T. Glaser, T. Best, T. Koehler, T. Robinson, T. Li, T. Zhang, T. Matthews, T. Chou, T. Shaked, V. Vontimitta, V. Ajayi, V. Montanez, V. Mohan, V. S. Kumar, V. Mangla, V. Ionescu, V. Poenaru, V. T. Mihailescu, V. Ivanov, W. Li, W. Wang, W. Jiang, W. Bouaziz, W. Constable, X. Tang, X. Wu, X. Wang, X. Wu, X. Gao, Y. Kleinman, Y. Chen, Y. Hu, Y. Jia, Y. Qi, Y. Li, Y. Zhang, Y. Zhang, Y. Adi, Y. Nam, Yu, Wang, Y. Zhao, Y. Hao, Y. Qian, Y. Li, Y. He, Z. Rait, Z. DeVito, Z. Rosnbrick, Z. Wen, Z. Yang, Z. Zhao, and Z. Ma. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.
- E. Grave, A. Joulin, and N. Usunier. Improving neural language models with a continuous cache, 2016. URL <https://arxiv.org/abs/1612.04426>.
- Y. Gu, L. Dong, F. Wei, and M. Huang. Minillm: Knowledge distillation of large language models, 2024. URL <https://arxiv.org/abs/2306.08543>.
- T. Hastie, A. Montanari, S. Rosset, and R. J. Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation, 2020. URL <https://arxiv.org/abs/1903.08560>.
- D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL <https://arxiv.org/abs/2103.03874>.
- T. Henighan, J. Kaplan, M. Katz, M. Chen, C. Hesse, J. Jackson, H. Jun, T. B. Brown, P. Dhariwal, S. Gray, C. Hallacy, B. Mann, A. Radford, A. Ramesh, N. Ryder, D. M. Ziegler, J. Schulman, D. Amodei, and S. McCandlish. Scaling laws for autoregressive generative modeling, 2020. URL <https://arxiv.org/abs/2010.14701>.

- J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang, and Y. Zhou. Deep learning scaling is predictable, empirically, 2017. URL <https://arxiv.org/abs/1712.00409>.
- J. Hestness, N. Ardalani, and G. Diamos. Beyond human-level accuracy: Computational challenges in deep learning, 2019. URL <https://arxiv.org/abs/1909.01736>.
- G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network, 2015. URL <https://arxiv.org/abs/1503.02531>.
- J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre. Training compute-optimal large language models, 2022. URL <https://arxiv.org/abs/2203.15556>.
- G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get m for free, 2017. URL <https://arxiv.org/abs/1704.00109>.
- K. Jordan. On the variance of neural network training with respect to test sets and distributions, 2024. URL <https://arxiv.org/abs/2304.01910>.
- J. Juravsky, A. Chakravarthy, R. Ehrlich, S. Eyuboglu, B. Brown, J. Shetaye, C. Ré, and A. Mirhoseini. Tokasaurus: An llm inference engine for high-throughput workloads. <https://scalingintelligence.stanford.edu/blogs/tokasaurus/>, 2025.
- J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models, 2020. URL <https://arxiv.org/abs/2001.08361>.
- N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2017. URL <https://arxiv.org/abs/1609.04836>.
- Y. Kim and A. M. Rush. Sequence-level knowledge distillation, 2016. URL <https://arxiv.org/abs/1606.07947>.
- B. Krause, E. Kahembwe, I. Murray, and S. Renals. Dynamic evaluation of neural sequence models, 2017. URL <https://arxiv.org/abs/1709.07432>.
- T. Kumar, Z. Ankner, B. F. Spector, B. Bordelon, N. Muennighoff, M. Paul, C. Pehlevan, C. Ré, and A. Raghunathan. Scaling laws for precision, 2024. URL <https://arxiv.org/abs/2411.04330>.
- B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles, 2017. URL <https://arxiv.org/abs/1612.01474>.
- Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.
- Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-49430-0. doi: 10.1007/3-540-49430-8_2. URL https://doi.org/10.1007/3-540-49430-8_2.

- J. Li, A. Fang, G. Smyrnis, M. Ivgi, M. Jordan, S. Gadre, H. Bansal, E. Guha, S. Keh, K. Arora, S. Garg, R. Xin, N. Muennighoff, R. Heckel, J. Mercat, M. Chen, S. Gururangan, M. Wortsman, A. Albalak, Y. Bitton, M. Nezhurina, A. Abbas, C.-Y. Hsieh, D. Ghosh, J. Gardner, M. Kilian, H. Zhang, R. Shao, S. Pratt, S. Sanyal, G. Ilharco, G. Daras, K. Marathe, A. Gokaslan, J. Zhang, K. Chandu, T. Nguyen, I. Vasiljevic, S. Kakade, S. Song, S. Sanghavi, F. Faghri, S. Oh, L. Zettlemoyer, K. Lo, A. El-Nouby, H. Pouransari, A. Toshev, S. Wang, D. Groeneveld, L. Soldaini, P. W. Koh, J. Jitsev, T. Kollar, A. G. Dimakis, Y. Carmon, A. Dave, L. Schmidt, and V. Shankar. Datacomp-1m: In search of the next generation of training sets for language models, 2025. URL <https://arxiv.org/abs/2406.11794>.
- J. Liu, J. Su, X. Yao, Z. Jiang, G. Lai, Y. Du, Y. Qin, W. Xu, E. Lu, J. Yan, Y. Chen, H. Zheng, Y. Liu, S. Liu, B. Yin, W. He, H. Zhu, Y. Wang, J. Wang, M. Dong, Z. Zhang, Y. Kang, H. Zhang, X. Xu, Y. Zhang, Y. Wu, X. Zhou, and Z. Yang. Muon is scalable for llm training, 2025. URL <https://arxiv.org/abs/2502.16982>.
- E. Lobacheva, N. Chirkova, M. Kodryan, and D. Vetrov. On power laws in deep ensembles, 2021. URL <https://arxiv.org/abs/2007.08483>.
- P. Maini, S. Seto, H. Bai, D. Grangier, Y. Zhang, and N. Jaitly. Rephrasing the web: A recipe for compute and data-efficient language modeling, 2024. URL <https://arxiv.org/abs/2401.16380>.
- M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL <https://aclanthology.org/J93-2004/>.
- M. Marek, S. Lotfi, A. Somasundaram, A. G. Wilson, and M. Goldblum. Small batch size training for language models: When vanilla sgd works, and why gradient accumulation is wasteful, 2025. URL <https://arxiv.org/abs/2507.07101>.
- S. McCandlish, J. Kaplan, D. Amodei, and O. D. Team. An empirical model of large-batch training, 2018. URL <https://arxiv.org/abs/1812.06162>.
- S. Merity, N. S. Keskar, and R. Socher. Regularizing and optimizing lstm language models, 2017. URL <https://arxiv.org/abs/1708.02182>.
- T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. Recurrent neural network based language model. volume 2, pages 1045–1048, 09 2010. doi: 10.21437/Interspeech.2010-343.
- H. Mobahi, M. Farajtabar, and P. L. Bartlett. Self-distillation amplifies regularization in hilbert space, 2020. URL <https://arxiv.org/abs/2002.05715>.
- N. Muennighoff, A. M. Rush, B. Barak, T. L. Scao, A. Piktus, N. Tazi, S. Pyysalo, T. Wolf, and C. Raffel. Scaling data-constrained language models, 2023. URL <https://arxiv.org/abs/2305.16264>.
- P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever. Deep double descent: Where bigger models and more data hurt, 2019. URL <https://arxiv.org/abs/1912.02292>.
- P. Nakkiran, P. Venkat, S. Kakade, and T. Ma. Optimal regularization can mitigate double descent, 2021. URL <https://arxiv.org/abs/2003.01897>.

- J. Ni, the, and team. Diffusion language models are super data learners. <https://jinjieni.notion.site/Diffusion-Language-Models-are-Super-Data-Learners-239d8f03a866800ab19> 2025. Notion Blog.
- M. Prabhudesai, M. Wu, A. Zadeh, K. Fragkiadaki, and D. Pathak. Diffusion beats autoregressive in data-constrained settings, 2025. URL <https://arxiv.org/abs/2507.15857>.
- J. S. Rosenfeld, A. Rosenfeld, Y. Belinkov, and N. Shavit. A constructive prediction of the generalization error across scales, 2019. URL <https://arxiv.org/abs/1909.12673>.
- Y. Ruan, C. J. Maddison, and T. Hashimoto. Observational scaling laws and the predictability of language model performance, 2024. URL <https://arxiv.org/abs/2405.10938>.
- Y. Ruan, N. Band, C. J. Maddison, and T. Hashimoto. Reasoning to learn from latent thoughts, 2025. URL <https://arxiv.org/abs/2503.18866>.
- B. S. Ruben, W. L. Tong, H. T. Chaudhry, and C. Pehlevan. No free lunch from random feature ensembles, 2024. URL <https://arxiv.org/abs/2412.05418>.
- V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020. URL <https://arxiv.org/abs/1910.01108>.
- N. Sardana, J. Portes, S. Doubov, and J. Frankle. Beyond chinchilla-optimal: Accounting for inference in language model scaling laws, 2025. URL <https://arxiv.org/abs/2401.00448>.
- J. Sevilla and E. Roldán. Training compute of frontier ai models grows by 4-5x per year, 2024. URL <https://epoch.ai/blog/training-compute-of-frontier-ai-models-grows-by-4-5x-per-year>. Accessed: 2025-08-21.
- C. Shannon. Prediction and entropy of printed english. *Bell system technical journal*, 30(1):50–64, 1951.
- H. Shi, K. Livescu, and K. Gimpel. Substructure substitution: Structured data augmentation for nlp, 2021. URL <https://arxiv.org/abs/2101.00411>.
- I. Shumailov, Z. Shumaylov, Y. Zhao, N. Papernot, R. Anderson, and Y. Gal. Ai models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759, 2024.
- J. B. Simon, D. Karkada, N. Ghosh, and M. Belkin. More is better in modern machine learning: when infinite overparameterization is optimal and overfitting is obligatory, 2024. URL <https://arxiv.org/abs/2311.14646>.
- S. P. Singh and M. Jaggi. Model fusion via optimal transport, 2023. URL <https://arxiv.org/abs/1910.05653>.
- S. L. Smith, E. Elsen, and S. De. On the generalization benefit of noise in stochastic gradient descent, 2020. URL <https://arxiv.org/abs/2006.15081>.
- C. Snell, J. Lee, K. Xu, and A. Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. URL <https://arxiv.org/abs/2408.03314>.
- B. Sorscher, R. Geirhos, S. Shekhar, S. Ganguli, and A. S. Morcos. Beyond neural scaling laws: beating power law scaling via data pruning, 2023. URL <https://arxiv.org/abs/2206.14486>.

- J. M. Springer, S. Goyal, K. Wen, T. Kumar, X. Yue, S. Malladi, G. Neubig, and A. Raghunathan. Overtrained language models are harder to fine-tune, 2025. URL <https://arxiv.org/abs/2503.19206>.
- D. Su, K. Kong, Y. Lin, J. Jennings, B. Norick, M. Kliegl, M. Patwary, M. Shoeybi, and B. Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset, 2025. URL <https://arxiv.org/abs/2412.02595>.
- C. Summers and M. J. Dinneen. Nondeterminism and instability in neural network optimization, 2021. URL <https://arxiv.org/abs/2103.04514>.
- R. Sutton. The bitter lesson. <http://www.incompleteideas.net/IncIdeas/BitterLesson.html>, 2019. Blog post.
- S. Takase, J. Suzuki, and M. Nagata. Direct output connection for a high-rank language model, 2018. URL <https://arxiv.org/abs/1808.10143>.
- R. Taori and T. B. Hashimoto. Data feedback loops: Model-driven amplification of dataset biases, 2022. URL <https://arxiv.org/abs/2209.03942>.
- G. Team, A. Kamath, J. Ferret, S. Pathak, N. Vieillard, R. Merhej, S. Perrin, T. Matejovicova, A. Ramé, M. Rivière, L. Rouillard, T. Mesnard, G. Cideron, J. bastien Grill, S. Ramos, E. Yvinec, M. Casbon, E. Pot, I. Penchev, G. Liu, F. Visin, K. Kenealy, L. Beyer, X. Zhai, A. Tsitsulin, R. Busa-Fekete, A. Feng, N. Sachdeva, B. Coleman, Y. Gao, B. Mustafa, I. Barr, E. Parisotto, D. Tian, M. Eyal, C. Cherry, J.-T. Peter, D. Sinopalnikov, S. Bhupatiraju, R. Agarwal, M. Kazemi, D. Malkin, R. Kumar, D. Vilar, I. Brusilovsky, J. Luo, A. Steiner, A. Friesen, A. Sharma, A. Sharma, A. M. Gilady, A. Goedeckemeyer, A. Saade, A. Feng, A. Kolesnikov, A. Bendebury, A. Abdagic, A. Vadi, A. György, A. S. Pinto, A. Das, A. Bapna, A. Miech, A. Yang, A. Paterson, A. Shenoy, A. Chakrabarti, B. Piot, B. Wu, B. Shahriari, B. Petrini, C. Chen, C. L. Lan, C. A. Choquette-Choo, C. Carey, C. Brick, D. Deutsch, D. Eisenbud, D. Cattle, D. Cheng, D. Paparas, D. S. Sreepathihalli, D. Reid, D. Tran, D. Zelle, E. Noland, E. Huizenga, E. Kharitonov, F. Liu, G. Amirkhanyan, G. Cameron, H. Hashemi, H. Klimczak-Plucińska, H. Singh, H. Mehta, H. T. Lehari, H. Hazimeh, I. Ballantyne, I. Szpektor, I. Nardini, J. Pouget-Abadie, J. Chan, J. Stanton, J. Wieting, J. Lai, J. Orbay, J. Fernandez, J. Newlan, J. yeong Ji, J. Singh, K. Black, K. Yu, K. Hui, K. Vodrahalli, K. Greff, L. Qiu, M. Valentine, M. Coelho, M. Ritter, M. Hoffman, M. Watson, M. Chaturvedi, M. Moynihan, M. Ma, N. Babar, N. Noy, N. Byrd, N. Roy, N. Momchev, N. Chauhan, N. Sachdeva, O. Bunyan, P. Botarda, P. Caron, P. K. Rubenstein, P. Culliton, P. Schmid, P. G. Sessa, P. Xu, P. Stanczyk, P. Tafti, R. Shivanna, R. Wu, R. Pan, R. Rokni, R. Willoughby, R. Vallu, R. Mullins, S. Jerome, S. Smoot, S. Girgin, S. Iqbal, S. Reddy, S. Sheth, S. Pöder, S. Bhatnagar, S. R. Panyam, S. Eiger, S. Zhang, T. Liu, T. Yacovone, T. Liechty, U. Kalra, U. Evci, V. Misra, V. Roseberry, V. Feinberg, V. Kolesnikov, W. Han, W. Kwon, X. Chen, Y. Chow, Y. Zhu, Z. Wei, Z. Egyed, V. Cotruta, M. Giang, P. Kirk, A. Rao, K. Black, N. Babar, J. Lo, E. Moreira, L. G. Martins, O. Sanseviero, L. Gonzalez, Z. Gleicher, T. Warkentin, V. Mirrokni, E. Senter, E. Collins, J. Barral, Z. Ghahramani, R. Hadsell, Y. Matias, D. Sculley, S. Petrov, N. Fiedel, N. Shazeer, O. Vinyals, J. Dean, D. Hassabis, K. Kavukcuoglu, C. Farabet, E. Buchatskaya, J.-B. Alayrac, R. Anil, Dmitry, Lepikhin, S. Borgeaud, O. Bachem, A. Joulin, A. Andreev, C. Hardin, R. Dadashi, and L. Hussenot. Gemma 3 technical report, 2025a. URL <https://arxiv.org/abs/2503.19786>.
- K. Team, Y. Bai, Y. Bao, G. Chen, J. Chen, N. Chen, R. Chen, Y. Chen, Y. Chen, Y. Chen, Z. Chen, J. Cui, H. Ding, M. Dong, A. Du, C. Du, D. Du, Y. Du, Y. Fan, Y. Feng, K. Fu, B. Gao, H. Gao, P. Gao, T. Gao, X. Gu, L. Guan, H. Guo, J. Guo, H. Hu, X. Hao, T. He, W. He, W. He, C. Hong,

- Y. Hu, Z. Hu, W. Huang, Z. Huang, Z. Huang, T. Jiang, Z. Jiang, X. Jin, Y. Kang, G. Lai, C. Li, F. Li, H. Li, M. Li, W. Li, Y. Li, Y. Li, Z. Li, Z. Li, H. Lin, X. Lin, Z. Lin, C. Liu, C. Liu, H. Liu, J. Liu, J. Liu, L. Liu, S. Liu, T. Y. Liu, T. Liu, W. Liu, Y. Liu, Y. Liu, Y. Liu, Y. Liu, Z. Liu, E. Lu, L. Lu, S. Ma, X. Ma, Y. Ma, S. Mao, J. Mei, X. Men, Y. Miao, S. Pan, Y. Peng, R. Qin, B. Qu, Z. Shang, L. Shi, S. Shi, F. Song, J. Su, Z. Su, X. Sun, F. Sung, H. Tang, J. Tao, Q. Teng, C. Wang, D. Wang, F. Wang, H. Wang, J. Wang, J. Wang, J. Wang, S. Wang, S. Wang, Y. Wang, Y. Wang, Y. Wang, Y. Wang, Y. Wang, Z. Wang, Z. Wang, Z. Wang, C. Wei, Q. Wei, W. Wu, X. Wu, Y. Wu, C. Xiao, X. Xie, W. Xiong, B. Xu, J. Xu, J. Xu, L. H. Xu, L. Xu, S. Xu, W. Xu, X. Xu, Y. Xu, Z. Xu, J. Yan, Y. Yan, X. Yang, Y. Yang, Z. Yang, Z. Yang, Z. Yang, H. Yao, X. Yao, W. Ye, Z. Ye, B. Yin, L. Yu, E. Yuan, H. Yuan, M. Yuan, H. Zhan, D. Zhang, H. Zhang, W. Zhang, X. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Y. Zhang, Z. Zhang, H. Zhao, Y. Zhao, H. Zheng, S. Zheng, J. Zhou, X. Zhou, Z. Zhou, Z. Zhu, W. Zhuang, and X. Zu. Kimi k2: Open agentic intelligence, 2025b. URL <https://arxiv.org/abs/2507.20534>.
- T. Thrush, C. Potts, and T. Hashimoto. Improving pretraining data using perplexity correlations, 2025. URL <https://arxiv.org/abs/2409.05816>.
- A. W. Van der Vaart. *Asymptotic statistics*, volume 3. Cambridge university press, 2000.
- H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning, 2015. URL <https://arxiv.org/abs/1509.06461>.
- A. Veit, M. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks, 2016. URL <https://arxiv.org/abs/1605.06431>.
- P. Villalobos, A. Ho, J. Sevilla, T. Besiroglu, L. Heim, and M. Hobbhahn. Will we run out of data? limits of llm scaling based on human-generated data, 2024. URL <https://arxiv.org/abs/2211.04325>.
- N. Vyas, A. Atanasov, B. Bordelon, D. Morwani, S. Sainathan, and C. Pehlevan. Feature-learning networks are consistent across widths at realistic scales, 2023. URL <https://arxiv.org/abs/2305.18411>.
- Z. Wang, F. Zhou, X. Li, and P. Liu. Octothinker: Mid-training incentivizes reinforcement learning scaling, 2025. URL <https://arxiv.org/abs/2506.20512>.
- A. Warstadt, L. Choshen, A. Mueller, A. Williams, E. Wilcox, and C. Zhuang. Call for papers – the babylm challenge: Sample-efficient pretraining on a developmentally plausible corpus, 2023. URL <https://arxiv.org/abs/2301.11796>.
- J. Welbl, N. F. Liu, and M. Gardner. Crowdsourcing multiple choice science questions, 2017. URL <https://arxiv.org/abs/1707.06209>.
- K. Wen, D. Hall, T. Ma, and P. Liang. Fantastic pretraining optimizers and where to find them, 2025. URL <https://arxiv.org/abs/2509.02046>.
- M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, and L. Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time, 2022. URL <https://arxiv.org/abs/2203.05482>.
- Z. Xie, S. I. Wang, J. Li, D. Lévy, A. Nie, D. Jurafsky, and A. Y. Ng. Data noising as smoothing in neural network language models, 2017. URL <https://arxiv.org/abs/1703.02573>.

- A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu. Qwen3 technical report, 2025. URL <https://arxiv.org/abs/2505.09388>.
- G. Yang, E. J. Hu, I. Babuschkin, S. Sidor, X. Liu, D. Farhi, N. Ryder, J. Pachocki, W. Chen, and J. Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer, 2022. URL <https://arxiv.org/abs/2203.03466>.
- Z. Yang, Z. Dai, R. Salakhutdinov, and W. W. Cohen. Breaking the softmax bottleneck: A high-rank rnn language model, 2018. URL <https://arxiv.org/abs/1711.03953>.
- Z. Yang, N. Band, S. Li, E. Candès, and T. Hashimoto. Synthetic continued pretraining, 2024. URL <https://arxiv.org/abs/2409.07431>.
- W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization, 2015. URL <https://arxiv.org/abs/1409.2329>.
- L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation, 2019. URL <https://arxiv.org/abs/1905.08094>.
- J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber. Recurrent highway networks, 2017. URL <https://arxiv.org/abs/1607.03474>.

Contents

1	Introduction	1
2	Standard pre-training	3
2.1	Evaluating existing data-constrained recipes	4
3	Regularized parameter scaling	5
4	Ensemble scaling	6
4.1	Formalizing ensembles	6
4.2	Scaling member count instead of parameter count	7
4.3	Joint scaling recipe composing parameter and ensemble scaling	8
5	Scaling the seed token count under infinite compute	9
5.1	Tuning the standard recipe	10
5.2	Scaling parameter count	10
5.3	Scaling member and parameter count	11
5.4	Scaling data	12
6	Data efficiency under parameter constraints	12
6.1	Reducing final parameter count via ensemble distillation	12
6.2	Reducing train parameter count via self-distillation	13
7	Downstream tasks	14
7.1	Downstream benchmarks	14
7.2	Continued pre-training	14
8	Related Work	15
9	Discussion	17
10	Acknowledgements	17
A	Problem setting	29
B	Standard pre-training details	30
B.1	Locally optimal hyperparameters	30
B.2	Ablating on coordinate descent	31
B.3	Tuned hyperparameters	32
B.4	Hyperparameter ablations	33

B.5	Overfitting analysis	33
C	Ensembling details	35
C.1	Seed science	35
C.2	Hyperparameter tuning for ensembles	35
C.3	Alternatives to ensembling	36
C.3.1	Mixture-of-Experts	36
C.3.2	Model soups	38
C.4	Order of limits	38
D	Data scaling	39
D.1	Epoch tuned baseline	39
E	Distillation details	40
E.1	Data generation	40
E.2	Hyperparameters	40
E.3	Mixing data ablation	41
F	Downstream task details	41
F.1	Downstream tasks	41
F.2	Hyperparameter tuning	41
G	Continued pre-training	42
G.1	Hyperparameters	42
G.2	CPT soups	43
H	Power laws	43
H.1	Sensitivity analysis	43
H.2	Fitting laws	44

A Problem setting

Pre-training algorithm. We instantiate the pre-training algorithm $\mathcal{A}$ using a standard pre-training recipe developed through the Marin project (<https://marin.community>) by following the best practices shared publicly and found internally.

- **Optimizer.** We train with AdamW, either with a default of 0.1 or a tuned weight decay. We set other hyperparameters to standard defaults ($\beta_1 = 0.9, \beta_2 = 0.95, \epsilon = 10^{-8}$). We clip the norm of the gradient at 1.
- **Learning rate.** We use a cosine learning rate schedule with a warmup for the first 1% of training, decaying to 0 by the end of training. We always tune learning rate for all of our experiments.

Every run has its own learning rate schedule and we never report the loss before the learning rate anneals to zero in the main body .

- **Architecture.** We train Llama-style auto-regressive language models. We specify the main architectural choices in Table 2. When scaling models, we change the initialization scheme to have variance inversely proportional to the hidden dimension (this is known to outperform μ P [Yang et al., 2022] within our framework: <https://github.com/marin-community/marin/issues/621>. For other architectural choices, we default to SiLU activations, untied word embeddings, and rotary position embeddings. We note that the non-standard scaling for the 1.4B model is a consequence of using presets in our pre-training framework.
- **Systems.** We train in mixed-precision with parameters in fp32 and compute + output in bf16. Most of our jobs were run on v4-64 or v4-128 TPUs, with bitwise-determinism for handling preemption and promoting reproducibility.
- **Data.** We pre-train using DCLM data [Li et al., 2025]. We keep a fixed validation set of 1024 sequences (4 million tokens) across all experiments for clear comparison. When increasing the size of the train pool, we ensure that smaller pools are a subset of larger pools.
- **Data order.** We generate a random permutation of the windows after tokenization and use this same permutation across epochs. We note that performance might have been better if we used a unique permutation every epoch but keep this fixed across models which further reduces randomness of training.

Parameter	150M	300M	600M	1.4B
Context Length	4096	4096	4096	4096
Hidden Dimension	512	768	1024	2048
Intermediate Dimension	1792	2688	7168	7168
Attention Heads	8	12	16	16
KV Heads	8	12	8	8
Layers	6	12	24	16

Table 2: Model architecture configurations for different model sizes. We default to the 300M model if not specified.

B Standard pre-training details

B.1 Locally optimal hyperparameters

We are interested in finding the best setting of hyperparameters (e.g. learning rate, epoch count, and weight decay) for a fixed parameter count and token count in the data-efficient pre-training setting. To make this search problem tractable, we first discretize the space of hyperparameters (e.g. only try epoch counts that are powers of 2). Under this discretization, it is prohibitively expensive to try every possible hyperparameter selection within our grid. Therefore, we search for **locally-optimal hyperparameters** as defined below.

Definition 1 (Locally-optimal hyperparameters). *We define the neighborhood $\mathcal{B}(H)$ of hyperparameter tuple H containing m variables as the $2m$ neighbors from incrementing/decrementing exactly one of the*

variables. We say H is locally optimal if and only if

$$\forall H' \in \mathcal{B}(H), \mathcal{L}(\mathcal{A}(D, N, H)) \leq \mathcal{L}(\mathcal{A}(D, N, H'))$$

Under certain assumptions, the locally-optimal hyperparameters are also globally optimal (for example, if the loss was convex in each dimension when the other variables are fixed). Though this may seem like a big assumption, we did not observe counter-examples to this in early experiments. One can verify this property for the single model losses presented in Figure 17.

Assuming this property, we use the following search procedure

1. Seed the search with initial runs around a best guess for optimal hyperparameters (heuristically set by us, using all runs until this point in time)
2. Take the best run so far and run its neighbors.
3. If any of its neighbors is better than the current candidate, the current candidate is sub-optimal. Repeat step 2 with the new best run.
4. If this run is better than all of its neighbors, we consider it “certified” and terminate the search procedure.

Though this procedure is quite expensive, we found it better than other natural heuristics which don’t rigorously follow this procedure separately for each parameter and token count (Appendix B.2). Furthermore, it seems to give clean scaling in both loss and hyperparameters, suggesting that the hyperparameter optimization landscape is nice enough for this coordinate descent algorithm to work. We note that this is a simplified version of the assumptions used in Wen et al. [2025].

Our specific discretization was forcing learning rate to be 1 or 3 times a power of 10, epoch count to be an integer power of 2, and weight decay to be either 0.0 or an integer power of 2 times 0.1. To further restrict the search space, we set bounds based on initial experiments tuning these hyperparameters, with a maximum learning rate of 3e-3, maximum weight decay of 6.4, and a maximum epoch count of 64 (these bounds only triggered for three of our searches).

B.2 Ablating on coordinate descent

For the parameter scaling experiments, we jointly tune learning rate, epoch count, and weight decay. We find this joint tuning necessary for clean scaling. To demonstrate this, we consider three alternatives and show their scaling in Figure 11.

1. **Fixing weight decay 0.1 and tuning learning rate and epoch count (red).** This baseline has already been shown to fail in Section 2.1.
2. **Jointly tuning weight decay only for 150M (green).** Here, we jointly tune the weight decay for the 150M model, finding the optimal value is 0.8. We then assume this is the optimal value for higher parameter counts and correspondingly tune learning rate and epoch count. We find that this scaling is not even monotonic.
3. **Jointly tuning epoch count only for 150M (blue).** Here, we jointly tune the epoch count for the 150M model, finding the optimal value is 16. We then assume this is the optimal value for higher parameter counts and correspondingly tune learning rate and weight decay. We find that this scaling plateaus much faster than the regularized recipe.

This shows the importance of jointly tuning both weight decay and epoch count at each model scale instead of blindly hoping for transfer.

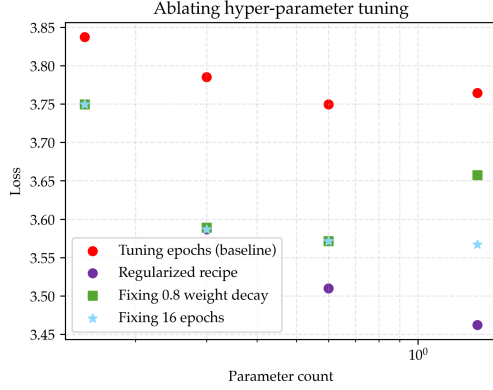


Figure 11: **Ablating joint tuning procedure.** We generally jointly tune learning rate, epoch count, and weight decay separately for each parameter count (purple). We show that trying to naively transfer hyperparameters across scales is a bad idea. Red: fixing weight decay to 0.1 (default regularization). Green: assuming that the optimal weight decay for 150M models (0.8) is optimal across N . Blue: assuming that the optimal epoch count for 150M models (16) is optimal across N .

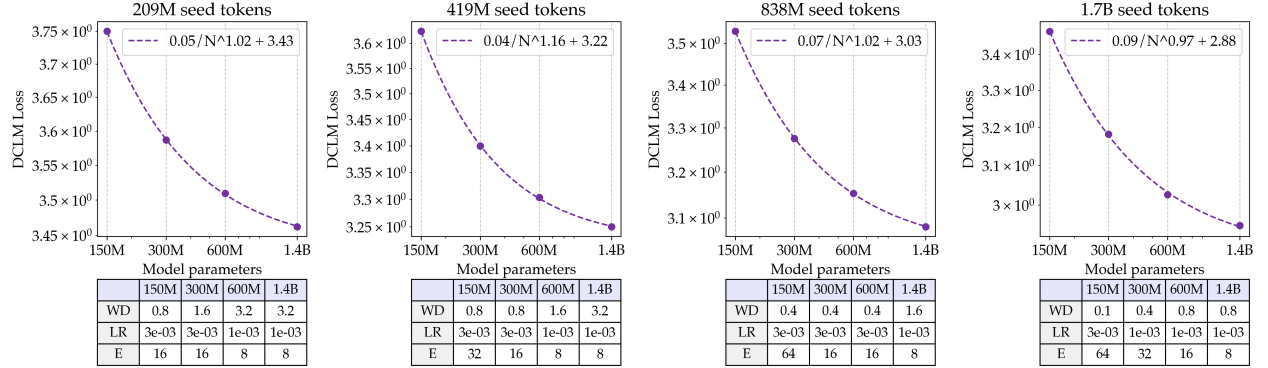


Figure 12: **Tuned hyperparameters for regularized scaling.** We show the optimal hyperparameters tuned separately for each parameter and token count. We find that as parameter count increases, optimal weight decay goes up, optimal epoch count goes down, and optimal learning rate goes down. We find the trends for weight decay and epoch count hold when token count decreases.

B.3 Tuned hyperparameters

In Figure 12, we share the locally optimal hyperparameters we found for 4 different token counts and 4 different parameter counts. We notice a few trends when looking at the resulting hyperparameters and power laws.

- The optimal learning rate decreases for larger models, noted by prior work [Everett et al., 2024, Yang et al., 2022]. The optimal learning rate does not strongly depend on the number of tokens.
- The optimal weight decay increases for larger models. Similarly, the optimal weight decay decreases for larger token counts. When fixing the parameter-to-token ratio, the weight decay stays around 0.8.
- The optimal epoch count decreases for larger models. Similarly, the optimal epoch count increases for larger token counts. When fixing the parameter-to-token ratio, the epoch count stays around 16.
- The power laws fit across all token counts share similar scaling exponents close to 1. This holds even though almost every model is over-parameterized at 200M tokens and under-parameterized at 1.6B tokens.

B.4 Hyperparameter ablations

We perform additional ablations on hyperparameters to understand their role beyond their optimal values. We start from a recipe of single epoch pre-training with 0.1 weight decay and tuned learning rate, and build our way up to tuning all hyperparameters.

Batch size. It is known that when optimizing for throughput, it is best to train at the “critical batch size” to best utilizes hardware [McCandlish et al., 2018]. However if we drop this constraint and instead measure performance for a fixed number of data points, we find that it is best to use smaller batch sizes, as shown in Figure 13, left, corroborating prior work in optimization [Keskar et al., 2017, LeCun et al., 1998, Marek et al., 2025, Smith et al., 2020]. We use a batch size of 64, which is the smallest size that is practical for our hardware.

Weight decay. It is known that regularization can further improve generalization when repeating data [D’Angelo et al., 2024]. Figure 13, right shows how varying the weight decay impacts epoched models (1.4B parameters for 8 epochs vs 300M parameters for 16 epochs). More over-parametrized models require a larger amount of regularization (3.2 vs 1.6). Without optimally tuning weight decay, one may draw the incorrect conclusion that larger models are worse than smaller models in the data-constrained setting. We also reproduce findings that higher weight decay enables a higher optimal learning rate and epoch count, shown by contrasting Figure 2 and Figure 3.

We find that increasing weight decay strongly changes the training dynamics. Though the train and validation losses decrease much slower at the start of training with high weight decay, they decrease rapidly by the end of training, eventually beating the run with low weight decay. In Figure 14, we visualize this for the best run with weight decay 0.1 and the best run tuning weight decay. This phenomenon also holds when using a higher weight decay on top of the best 0.1 weight decay hyperparameters. This is in line with previous findings in optimization research [D’Angelo et al., 2024, Liu et al., 2025, Wen et al., 2025] and suggests that we should only look at the performance at the end of training.

B.5 Overfitting analysis

In Section 2.1, we discuss how introducing too many epochs or parameters results in validation loss going up which we believe is due to over-fitting. In Figure 15, we track train loss for the interventions of increasing epoch count and parameter count. On the left, we show how increasing epoch count monotonically decreases train loss but eventually results in validation loss going up. On the right, we show how increasing parameter count results in erratic changes in train loss. We hypothesize this is because the optimal epoch count changes from 8 for the first two models to 4 for the last two models. We find that when we restrict all models to only use 4 epochs, train loss decreases monotonically and validation loss still goes up, suggesting over-fitting.

We note that another reason over-fitting may be happening is because our 1.4B model trades depth for width. We did not recognize our model scaling was non-standard until the majority of our experiments had finished because these were the default settings in our pre-training framework. We do not think this is a severe issue since correctly tuning weight decay seems to correct for the fact that this architecture has less layers. Moreover, the large improvement from weight decay is also suggestive of the fact that larger models are over-fitting.

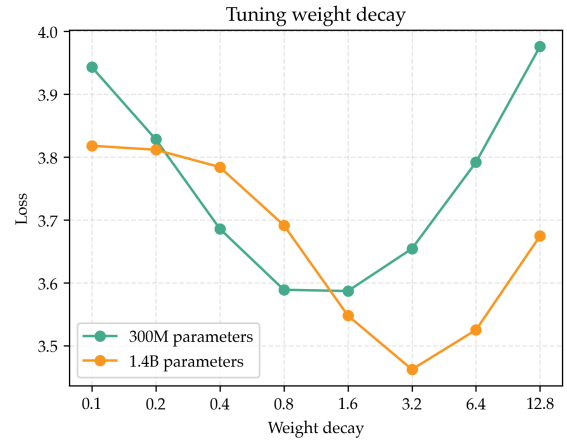
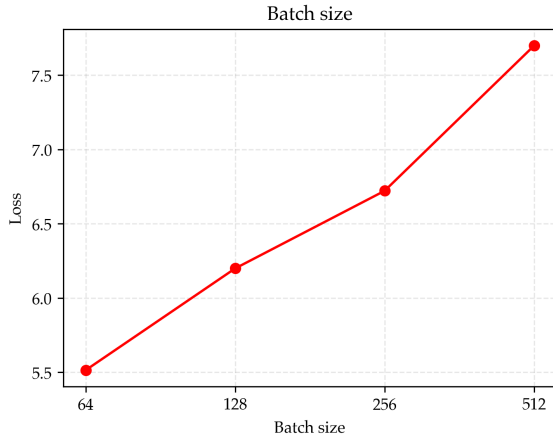


Figure 13: **Re-evaluating hyperparameters for standard pre-training.** Left: smaller batch sizes are better, we stop at 64 since this is the smallest our hardware practically supports (shown for 1 epoch training with 0.1 weight decay and a fixed learning rate of $3e-3$). Right: weight decay helps, and the optimal weight decay is higher for larger models (300M is 16 epochs $3e-3$ learning rate, 1.4B is 8 epochs $1e-3$ learning rate).

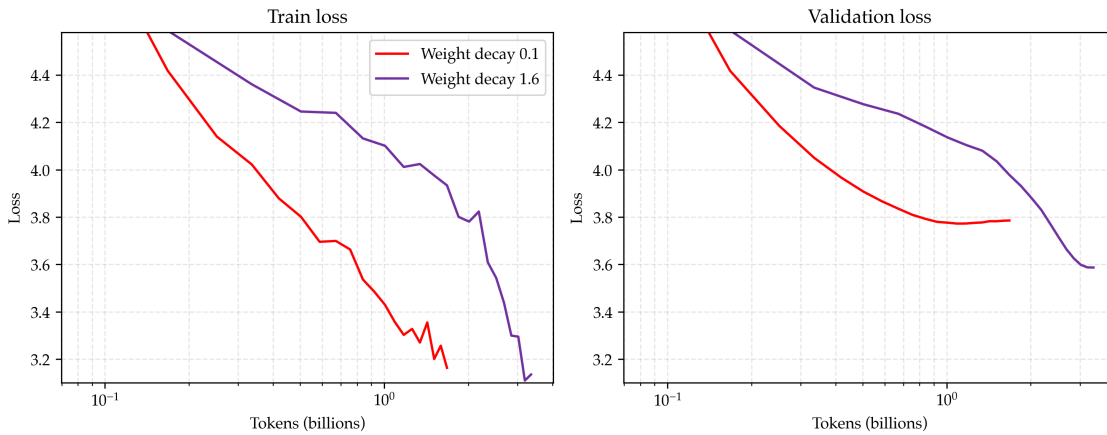


Figure 14: **Loss trajectories for different weight decays.** We compare the best run with default weight decay (8 epochs, $1e-3$ learning rate, 0.1 weight decay) and the best run with tuned weight decay (16 epochs, $3e-3$ learning rate, 1.6 weight decay) for 200M tokens and 300M parameters. We find that loss for runs with high weight decay decreases much more slowly at the start of training, but quickly decreases near the end of training.

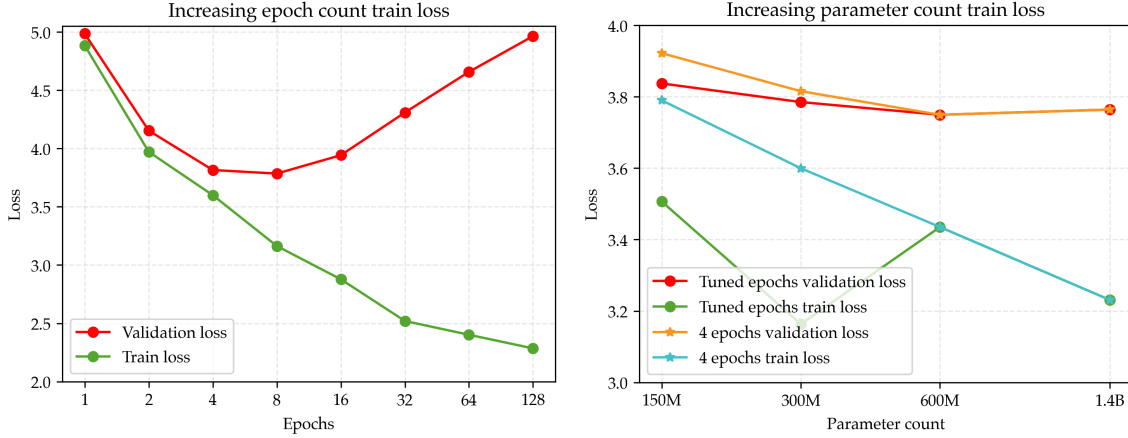


Figure 15: **Train losses for epoching and parameter scaling.** Left: Increasing epoch count results in train loss decreasing while validation loss starts increasing. Right: Increasing parameter count does not always decrease loss, potentially due to optimal epoch count changing (8, 8, 4, 4). Train loss monotonically goes down when restricted to 4 epochs.

C Ensembling details

C.1 Seed science

For our training runs, the randomness only comes from the sampled initialization (train seed) and shuffled data order (data seed). We first characterize the run-to-run variance when varying both seeds, only train seed, or only data seed. We train 5 models for each of the three randomness options. When using the optimal hyperparameters for a 300M model with 200M tokens, the standard deviation is estimated to be 0.008207 for both seeds, 0.007605 for only train seed, and 0.007213 for only data seed. This is in line with prior work that shows how only a small amount of instability is needed to induce the majority of the variance over a final model’s loss [Jordan, 2024, Summers and Dinneen, 2021].

We now measure the loss of ensembles with these sources of variance in Figure 16. Either of these sources delivers most of the benefit of ensembling, with data order helping more. Considering the marginal benefit of introducing additional sources of randomness, we didn’t strongly explore adding more sources of randomness during training.

C.2 Hyperparameter tuning for ensembles

Similar to parameter scaling, we hope to find the locally optimal hyperparameters for a given parameter count and token count. However, we care about the hyperparameters as $K \rightarrow \infty$, not at $K = 1$ (as discussed in Section 4.2). Since it is too experimentally prohibitive to search for locally optimal hyperparameters for the asymptote, we study how the hyperparameters change relative to the optimal H for single models.

We repeat the analysis in Figure 5 for 3 different parameter counts as well as 2 different learning rates at 300M parameters, shown in Figure 17. We find that for all of the displayed settings, our best run comes from an ensemble with the same learning rate, half weight decay, and double epoch count relative to the optimal single model hyperparameters. For the three parameter counts we

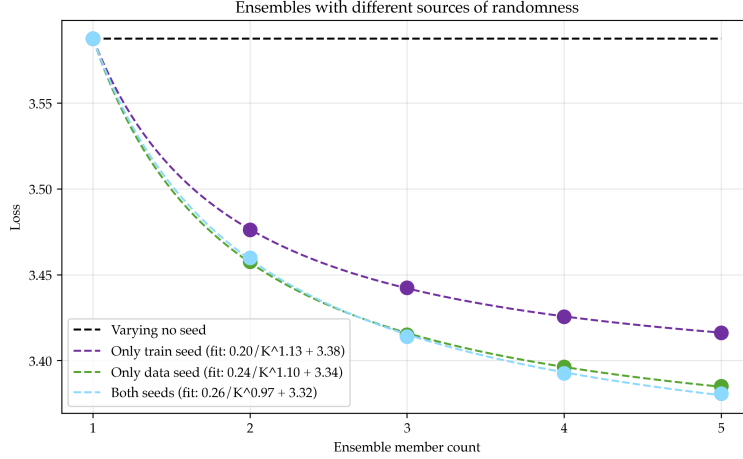


Figure 16: **Sources of randomness for ensembling.** Only varying train or data seed is enough to induce the benefits of ensembling. Varying data seed (i.e. order) is better between these two.

display, we verify that these are locally optimal hyperparameters. In fact, across all of the ensembles we trained across our scales (including many that are not directly visualized), we find only one counter-example to this heuristic. This occurs for 1.4B models trained on 200M tokens, where the hyperparameters that minimize the asymptote do not halve the weight decay. We suspect this change occurs because this is our most over-parameterized setting, and our scaling laws use this best run over the heuristic for this single setting.

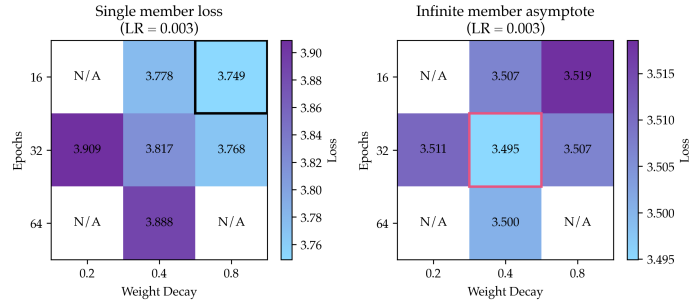
C.3 Alternatives to ensembling

The success of ensembling suggests alternative parameterizations that might also boost data efficiency. We discuss commonly considered ones here, which we were not able to get to outperform ensembling.

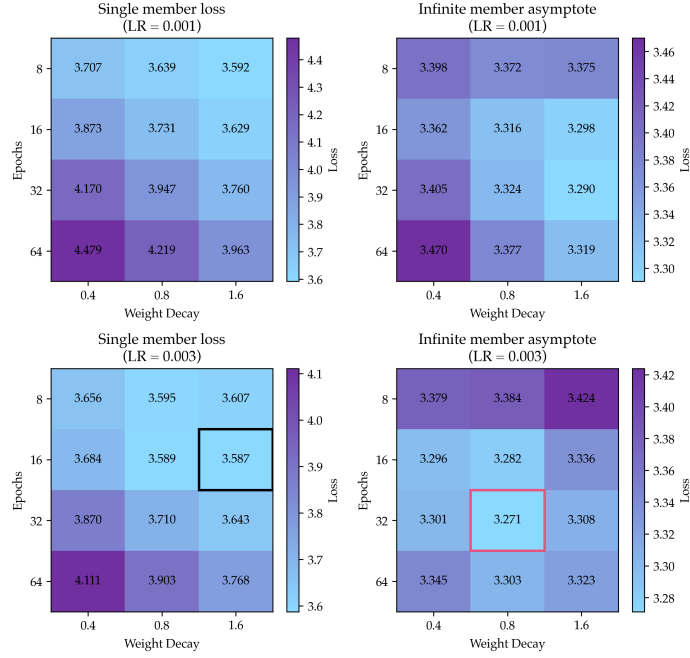
C.3.1 Mixture-of-Experts

Ensembling may qualitatively seem similar to training with Mixture-of-Experts (MoE). However, we find an important distinction: when training an ensemble, the learning trajectory for each model is completely independent of each other, whereas for a MoE, it is still a single learning trajectory. Unfortunately, the intuition from [Allen-Zhu and Li \[2023\]](#) suggests that the sparsity of the MoE architecture is not guaranteed to benefit from “multi-view” data in the way ensembles do. In their paper, they consider a simplified analogue where they construct a model that runs the ten models in parallel and takes the gradient step through the ensemble jointly. They find that this barely improves performance over a single model. In early experiments, we were able to reproduce this phenomenon, with a jointly trained 10 ensemble of models outperforming a single model by only 0.02 loss. Therefore, we hypothesize that if MoE’s were to help, their benefits would come from the drop-out aspect instead of the sparsity aspect, which does not require MoE’s (note that we did not tune or consider drop-out in this work, though we expect it to further help performance). We hope future experiments can settle this intuition more concretely.

Ensemble tuning for $N = 150M$, $D = 209M$



Ensemble tuning for $N = 300M$, $D = 209M$



Ensemble tuning for $N = 600M$, $D = 209M$

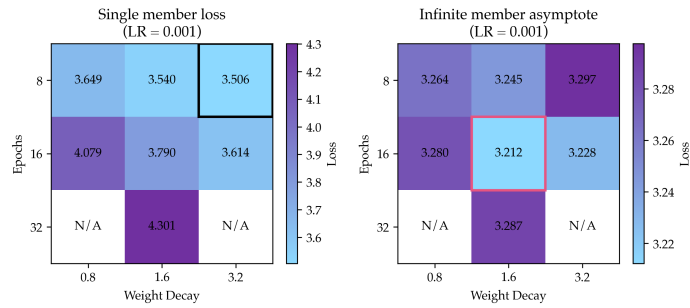


Figure 17: **Single model and asymptote loss when varying epoch count and weight decay for different model sizes, token counts, and learning rates.** We display an extended version of Figure 5 for 200M tokens with 150M, 300M (sub-optimal and optimal LR), and 600M parameter models. For these settings, the “double epoch, half weight decay” heuristic correctly predicts the best ensemble. This heuristic is consistent with all of our parameter and token counts except for our most over-parameterized setting.

C.3.2 Model soups

Prior works have shown that averaging the weights of independent training runs can result in better models [Wortsman et al., 2022]. However, we note that most success from averaging weights comes at fine-tuning, not pre-training. We replicate these results in our own settings, with model soups achieving close to random performance on downstream benchmarks (Table 5) but slightly outperforming ensembles in continued pre-training (Table 7).

One intuition for this discrepancy is that models need to be in the same “loss basin” for averaging to help final performance, and pre-trained models enter different loss basins [Ainsworth et al., 2023, Singh and Jaggi, 2023]. Past studies also design compute-efficient algorithms for merging models trained from scratch, but they find that the more expensive procedure of distillation outperforms their method [Singh and Jaggi, 2023]. Since we are in the infinite compute regime, we opt to use distillation over model merging for the best performance.

C.4 Order of limits

In Section 4.3, we are interested in computing the best possible performance of ensembles as N and K both go to ∞ . There are a couple of different ways to compute this, some of which are enumerated below

Double Limit 1 (Our Approach). We first solve for the limit as $K \rightarrow \infty$ by tuning asymptotes and then solve for the outer limit via a second power law over the inner asymptotes.

$$\lim_{N \rightarrow \infty} \lim_{K \rightarrow \infty} \min_H \mathcal{L}(\mathcal{E}_A(D, N, K, H))$$

Hypothetical Double Limit 2. We can flip the above order and instead take $N \rightarrow \infty$ before $K \rightarrow \infty$, corresponding to

$$\lim_{K \rightarrow \infty} \lim_{N \rightarrow \infty} \min_H \mathcal{L}(\mathcal{E}_A(D, N, K, H))$$

Hypothetical Double Limit 3. Following literature in compute-optimal scaling, we can find the best possible performance for a given compute budget C and take $C \rightarrow \infty$.

$$\lim_{C \rightarrow \infty} \min_{\substack{H, N, K \\ \text{s.t. FLOPs}(D, N, K, H) = C}} \mathcal{L}(\mathcal{E}_A(D, N, K, H))$$

We believe that our approach is experimentally much more convenient than the hypothetical approaches even though they are equivalent in output under assumptions. We share our reasoning by answering the following questions comparing the approaches.

The core assumption that we will make is that $f(N, K) = \min_H \mathcal{L}(\mathcal{E}_A(D, N, K, H))$ is monotone in N and K when the other is fixed. Across all of our experiments, we do not observe any contradictions to this assumption as long as we tune regularization (for examples, refer to Figure 7 and Figure 8).

- **Are Double Limit 1 and 2 mathematically equivalent?** Mathematically, both limits are equivalent. For a quick proof, define $k_i := \lim_{K \rightarrow \infty} f(D, N)$ and $n_i := \lim_{N \rightarrow \infty} f(D, N)$. By monotonicity, both k_i and n_i exist and are non-increasing sequences. Define $k = \lim_{N \rightarrow \infty} k_i$ and $n = \lim_{K \rightarrow \infty} n_i$ which exist for the same reason. Note that $f(N, K) \leq n_i$, and since limits preserve inequality, it follows that $k_i \leq n$, and further follows that $k \leq n$. By repeating this argument in the reverse direction, it follows that $k = n$.

- **Are Double Limits 1 and 2 mathematically equivalent to 3?** If we make the same monotonicity assumption as earlier, then we have that the minimization problem is monotonic in C . With this, we can apply a similar argument as above to show that the limit converges and is equivalent to k and n .
- **How do we solve for the inner limit in Double Limit 1?** It is computationally prohibitive to tune all the hyperparameters for each choice of N and K . Therefore, we would prefer searching for optimal hyperparameters once per choice of N to fit the outer limit. Since we are only interested in the performance as $K \rightarrow \infty$ increases, we can reasonably approximate this via our hyperparameter heuristic from Appendix C.2 without ever determining the optimal hyperparameters at lower values of K .
- **Why do we prefer Double Limit 1 to Double Limit 2?** We first make the observation that tuning hyperparameters depends on N , but if we follow the previous bullet’s asymptote tuning heuristic, tuning hyperparameters does not depend on any finite value of K . Therefore, Approach 2 would still have to fit hyperparameters for each N, K , whereas Approach 1 avoids this by fitting it once per N .
- **Why do we not use Double Limit 3?** In Double Limit 1, we keep the hyperparameter as the scaling axis, instead of Double Limit 3 which sets compute as the scaling axis. When we choose to scale the hyperparameter, we can use our locally optimal hyperparameter search algorithm to find the best possible performance for that hyperparameter. This is difficult when scaling compute, since our hyperparameters such as model size and epoch count influence the compute spent during pre-training. Our preference reflects how practitioners typically use Chinchilla Approach 3 (fitting loss for best run given N, D , closer to Double Limit 1) over Chinchilla Approach 1 (fitting the envelope of the runs, closer to Double Limit 3) [Besiroglu et al., 2024].

D Data scaling

D.1 Epoch tuned baseline

Unlike regularized parameter scaling and ensemble scaling where we estimate the best possible loss via asymptotes (Section 5), epoch tuning eventually over-fits. To estimate the best possible loss under epoch tuning for each token count, we use the following procedure.

1. For a fixed token count D and fixed parameter count N , search for locally optimal hyperparameters while fixing weight decay to be 0.1
2. Perform this for all parameter counts N and token counts D

After following this procedure, we found that 600M models and 1.4B models were within ≈ 0.02 loss of each other and were much better than the other models, as shown in Figure 18. Across all our token scales, 600M models slightly outperformed 1.4B models, which we discuss in B.5. Therefore, we take the 600M performance as an estimate of the best possible loss under regularized parameter scaling. We believe the performance of this algorithm would be better under better width/depth/architectures, though we expect this benefit to translate to our other recipes as well.

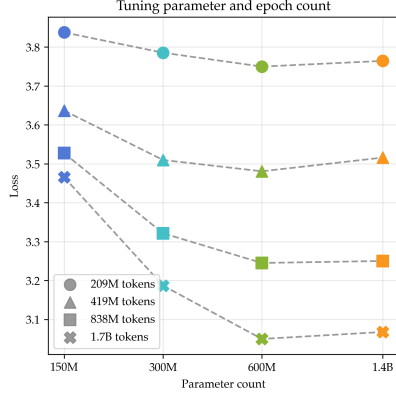


Figure 18: **Scaling seed token count for epoching and parameter scaling.** For each seed token count, we train models of varying N with jointly tuned learning rate and epoch count with a weight decay of 0.1. We take the best possible model at each token count as an estimate of the best performance under the standard recipe.

E Distillation details

E.1 Data generation

For all distillation experiments, we generate teacher data from the same model family: K -ensembles of 300M models with optimal hyperparameters for asymptotic performance. We choose to perform self-distillation with a 1-ensemble from this family rather than the 300M model from the regularized recipe to cleanly isolate the effect of distilling from a stronger teacher. In practice, we don’t observe a significant difference: self-distillation from the 1-ensemble (blue point, Figure 9) gives a loss of 3.43 while self-distillation from the 300M regularized recipe (purple point, Figure 9) gives a loss of 3.44.

Because we are unconstrained by train compute, optimal distillation should never epoch on teacher data and instead generate more. We pre-generate a large pool of teacher distillation data by sampling unconditionally with temperature 1 using a high-throughput inference engine designed for batched workloads [Juravsky et al., 2025]. For generating ensemble teacher distillation data, we experiment with inferencing both the logit averaged ensemble as well as the individual members. We observe better student performance using the individual members.

E.2 Hyperparameters

For both ensemble distillation and self-distillation, we search for optimal hyperparameters using a procedure similar to Appendix B.1. Our distillation recipe also introduces a new hyperparameter which we refer to as the mixing ratio: the ratio of batches of real data to synthetic data. A mixing ratio of 1 : 1 indicates that we take the same number of gradient steps on real data as teacher-generated data. For example, if we have 209M tokens of real data that we wish to epoch on 16 times, a 1 : 1 mixing ratio would require $209 \cdot 10^6 \times 16 = 3.3\text{B}$ tokens of teacher-generated distillation data. We find tuning the mixing ratio to be important for performance.

We detail the exact values of hyperparameters in Table 3. Interestingly, we observe that optimal weight decay for distillation is lower than that of our regularized recipes, in line with standard practice. In addition, we find that ensemble distillation admits a higher optimal mixing ratio, likely due to the greater diversity from the teacher’s synthetic data. Our ensemble distillation run trains on a total of $16 \times 209 \cdot 10^6 \times (1 + 9) = 33.4\text{B}$ tokens, while our self-distillation run trains on a total of $16 \times 209 \cdot 10^6 \times (1 + 3) = 13.4\text{B}$ tokens. Due to limitations in inferencing, we only generate 10B tokens each of ensemble distillation and self-distillation data, so our ensemble distillation may epoch up to 3 times on the teacher data.

Parameter	Ensemble Distill	Self-Distill
Learning Rate	3e-3	3e-3
Weight Decay	0.1	0.1
Mixing Ratio	1 : 9	1 : 3
Epochs	16	16

Table 3: Optimal hyperparameters for ensemble and self-distillation.

E.3 Mixing data ablation

We provide a token-matched ablation for the effect of mixing in the real pre-training data when doing self-distillation. As in Appendix E.2, we start with the same pool of 10B pre-generated tokens from a 300M 1-ensemble. Perfect distillation into a student model of the same size (with an infinite amount of teacher data) would achieve the same loss as the teacher.

We compare self-distillation with and without mixing in real data. For mixing in real data, we epoch the real data 16 times and use a 1 : 1 mixing ratio so that the total number of tokens we train on is less than 10B. For no mixing, we simply train on a subset of the pre-generated pool. Both methods use the same learning rate and batch size, train on a total of $16 \times 209 \cdot 10^6 \times (1 + 1) = 6.688\text{B}$ tokens, and never repeat the synthetic teacher data. For no mixing, we additionally search over weight decay.

Table 4 shows that without mixing in real pre-training data, self-distillation is substantially worse than the teacher model (as one might expect). Mixing data allows for self-distillation to exceed the teacher model.

	Teacher Model	Self-Distill (1 : 1 mixing)	Self-Distill (No mixing)
Val Loss	3.7103	3.4373	4.0693

Table 4: Effect of mixing real pre-training data for self-distillation.

F Downstream task details

F.1 Downstream tasks

We provide a full breakdown of downstream benchmark scores per model type in Table 5. We use `lm-evaluation-harness` [Gao et al., 2024] for our evaluations. Our evaluation code is available at <https://github.com/konwook/lm-eval-ensemble>.

F.2 Hyperparameter tuning

We find that hyperparameter tuning from validation loss transfers to downstream benchmarks as well. Figure 19 (left) shows how adding heavy regularization with weight decay (with a fixed learning rate of 3e-3) shifts the overfitting point based on validation loss to the right and down. We observe a similar effect in Figure 19 (right), although the overfitting threshold (in epochs) is twice the threshold observed for validation loss.

Model type		ARC-Easy (%)	PIQA (%)	SciQ (%)	Avg (%)
Unregularized model scaling	150M	40.95 $\pm$ 1.01	59.68 $\pm$ 1.14	62.40 $\pm$ 1.53	54.35 $\pm$ 0.72
	300M	41.96 $\pm$ 1.01	61.15 $\pm$ 1.14	62.90 $\pm$ 1.53	55.34 $\pm$ 0.72
	600M	39.86 $\pm$ 1.00	59.90 $\pm$ 1.14	60.50 $\pm$ 1.55	53.42 $\pm$ 0.72
	1.4B	40.61 $\pm$ 1.01	60.39 $\pm$ 1.14	61.40 $\pm$ 1.54	54.14 $\pm$ 0.72
Model scaling	150M	41.29 $\pm$ 1.01	60.17 $\pm$ 1.14	63.90 $\pm$ 1.52	55.12 $\pm$ 0.72
	300M	44.28 $\pm$ 1.02	61.81 $\pm$ 1.13	69.10 $\pm$ 1.46	58.39 $\pm$ 0.70
	600M	47.10 $\pm$ 1.02	63.06 $\pm$ 1.13	69.70 $\pm$ 1.45	59.95 $\pm$ 0.70
	1.4B	45.66 $\pm$ 1.02	63.82 $\pm$ 1.12	72.70 $\pm$ 1.41	60.73 $\pm$ 0.69
150M ensembles	$K = 1$	42.85 $\pm$ 1.02	60.88 $\pm$ 1.14	64.90 $\pm$ 1.51	56.21 $\pm$ 0.72
	$K = 2$	44.61 $\pm$ 1.02	62.02 $\pm$ 1.13	65.80 $\pm$ 1.50	57.48 $\pm$ 0.71
	$K = 3$	44.91 $\pm$ 1.02	62.08 $\pm$ 1.13	68.30 $\pm$ 1.47	58.43 $\pm$ 0.71
	$K = 4$	45.83 $\pm$ 1.02	62.19 $\pm$ 1.13	69.20 $\pm$ 1.46	59.07 $\pm$ 0.70
	$K = 5$	45.50 $\pm$ 1.02	62.30 $\pm$ 1.13	70.40 $\pm$ 1.44	59.40 $\pm$ 0.70
300M ensembles	$K = 1$	44.36 $\pm$ 1.02	62.95 $\pm$ 1.13	68.10 $\pm$ 1.47	58.47 $\pm$ 0.71
	$K = 2$	46.72 $\pm$ 1.02	63.87 $\pm$ 1.12	70.70 $\pm$ 1.44	60.43 $\pm$ 0.70
	$K = 3$	47.77 $\pm$ 1.02	64.74 $\pm$ 1.11	72.90 $\pm$ 1.41	61.80 $\pm$ 0.69
	$K = 4$	48.36 $\pm$ 1.03	65.89 $\pm$ 1.11	73.10 $\pm$ 1.40	62.45 $\pm$ 0.69
	$K = 5$	49.33 $\pm$ 1.03	65.67 $\pm$ 1.11	74.00 $\pm$ 1.39	63.00 $\pm$ 0.68
600M ensembles	$K = 1$	45.92 $\pm$ 1.02	62.84 $\pm$ 1.13	68.50 $\pm$ 1.47	59.09 $\pm$ 0.71
	$K = 2$	47.56 $\pm$ 1.02	64.04 $\pm$ 1.12	71.80 $\pm$ 1.42	61.13 $\pm$ 0.69
	$K = 3$	48.44 $\pm$ 1.03	64.25 $\pm$ 1.12	73.30 $\pm$ 1.40	62.00 $\pm$ 0.69
	$K = 4$	49.03 $\pm$ 1.03	64.80 $\pm$ 1.11	73.70 $\pm$ 1.39	62.51 $\pm$ 0.69
	$K = 5$	50.34 $\pm$ 1.03	64.80 $\pm$ 1.11	75.30 $\pm$ 1.36	63.48 $\pm$ 0.68
1.4B ensembles	$K = 1$	43.56 $\pm$ 1.02	64.20 $\pm$ 1.12	68.80 $\pm$ 1.47	58.85 $\pm$ 0.70
	$K = 2$	47.26 $\pm$ 1.02	65.13 $\pm$ 1.11	75.30 $\pm$ 1.36	62.56 $\pm$ 0.68
	$K = 3$	49.33 $\pm$ 1.03	65.40 $\pm$ 1.11	76.50 $\pm$ 1.34	63.74 $\pm$ 0.67
	$K = 4$	48.86 $\pm$ 1.03	66.38 $\pm$ 1.10	77.80 $\pm$ 1.31	64.35 $\pm$ 0.67
	$K = 5$	49.71 $\pm$ 1.03	66.38 $\pm$ 1.10	77.10 $\pm$ 1.33	64.39 $\pm$ 0.67
Distillation (300M)	Self	46.68 $\pm$ 1.02	62.35 $\pm$ 1.13	72.60 $\pm$ 1.41	60.54 $\pm$ 0.69
	Ensemble	48.44 $\pm$ 1.03	62.84 $\pm$ 1.13	75.30 $\pm$ 1.36	62.19 $\pm$ 0.68
Model soups	$K = 2$	26.56 $\pm$ 0.91	54.84 $\pm$ 1.16	24.70 $\pm$ 1.36	35.37 $\pm$ 0.67
	$K = 4$	24.96 $\pm$ 0.89	55.28 $\pm$ 1.16	23.90 $\pm$ 1.35	34.71 $\pm$ 0.66

Table 5: Benchmark accuracies of all methods using 200M tokens on ARC-Easy, PIQA, and SciQ with averages. Entries are value $\pm$ SE in percentage points.

G Continued pre-training

G.1 Hyperparameters

Hyperparameters for continued pre-training baselines are shown in Table 6. The 73B CPT run uses the default hyperparameters from [Wang et al., 2025], except for learning rate which we tuned

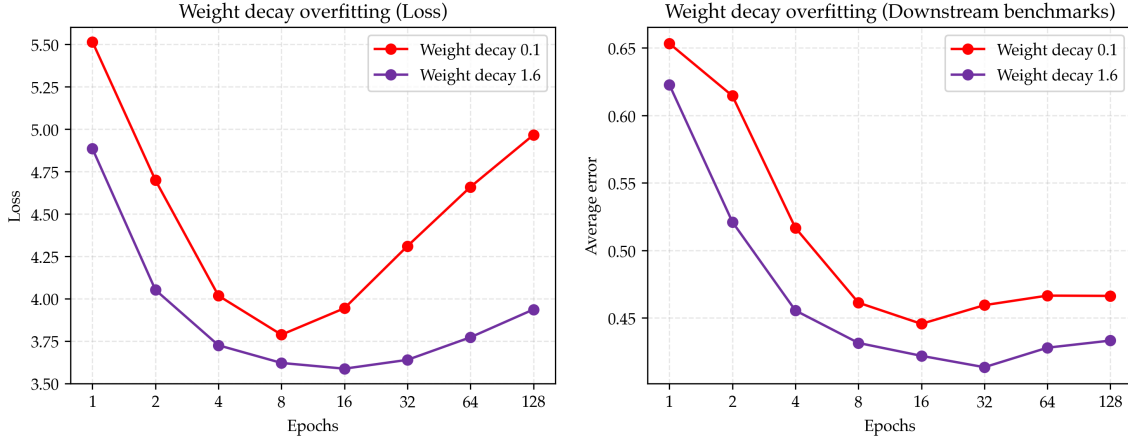


Figure 19: **Effect of regularization on overfitting for downstream benchmarks.** Downstream benchmarks also reflect the benefit of heavy regularization on performance. The effect of overfitting on downstream benchmarks (right) appears at twice the epoch count compared to validation loss (left).

ourselves. The individual members of the K -ensembles use the same hyperparameters as the standard recipe.

Parameter	Default	Lower BS	Epoching
Learning Rate	3e-5	3e-5	3e-5
Weight Decay	0.1	0.1	0.1
Batch Size	512	64	64
Epochs	1	1	4

Table 6: Hyperparameters for continued pre-training.

G.2 CPT soups

We ablate the performance of model soups compared to ensembling in our continued pre-training setting by averaging the weights of the members instead of ensembling them. Unlike standard pre-training, CPT soups perform strongly and slightly outperform ensembles as we increase the number of averaged models (Table 7).

H Power laws

H.1 Sensitivity analysis

To test whether our asymptote estimation is reliable due to run-to-run variance, we conduct a sensitivity analysis for regularized parameter scaling and ensembling, shown in Figure 20.

To test parameter scaling, we fit three power laws to all the models trained where each power law uses a different seed (governing data order and model initialization). Though the scaling laws change per seed, they remain relatively consistent, with the asymptotes staying close together. This

Table 7: Continually pre-trained ensembles vs. soups

Benchmarks	Llama 3B	K -ensembles			K -soups		
		$K = 2$	$K = 4$	$K = 8$	$K = 2$	$K = 4$	$K = 8$
GSM8K _(8-shot)	28.23	49.28	51.80	52.99	49.73	53.83	54.96
MATH _(4-shot)	6.90	21.84	23.04	23.50	22.40	23.02	23.72
MATHQA _(8-shot)	35.07	45.12	46.06	45.26	44.59	46.10	45.33
Average	24.25	38.79	40.35	40.58	38.91	40.98	41.34

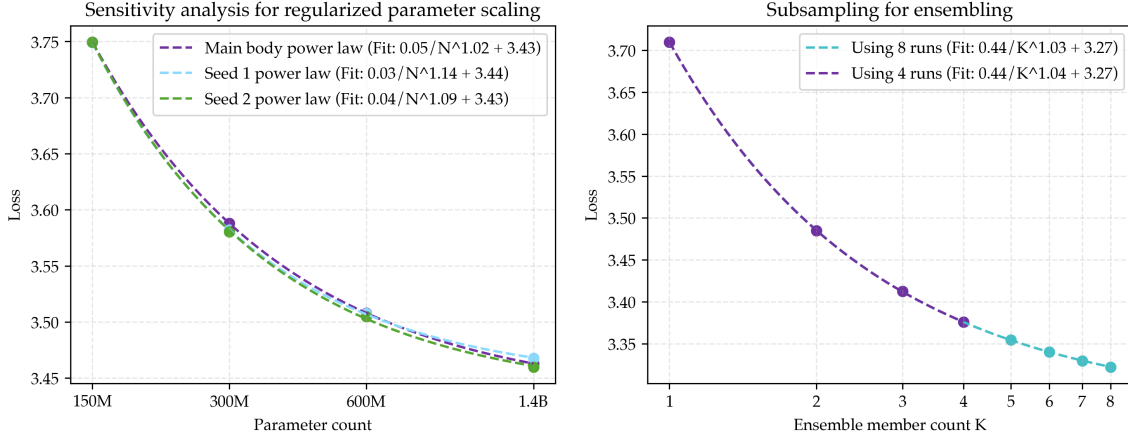


Figure 20: **Sensitivity analysis.** Left: When re-fitting the regularized power law across two additional seeds, we find that the asymptote stays relatively stable. Right: When subsampling the number of points for the ensemble scaling law, we find that the power law barely changes.

is encouraging, as the standard deviation in asymptotes is close to the run-to-run standard deviation for 300M models (Appendix C.1).

Since we have more runs for ensembling, we test the reliability of ensembling by subsampling the number of members. When fitting a power law using up to four ensemble members, we find an extremely similar law to using up to eight ensemble members. Qualitatively, over the course of our experiments, we found that the scaling law for ensembling is a lot more stable than the scaling law for parameter scaling.

We note that this is a limited stress-test and that it is likely our asymptote estimation procedure is quite noisy. Furthermore, we note that this does not test our two-tier and three-tier power laws for joint scaling of parameters, members, and data, nor does it test our settings where our best run is further from the asymptote. We advise taking these asymptotes with a grain of salt and interpreting them as rough estimates.

H.2 Fitting laws

To fit our power laws, we use `scipy.optimize.curve_fit`, either with no initial conditions and bounds or with `p0=[1.0, 0.5, 2.0]` and `bounds=([0, 0, 0], [np.inf, np.inf, np.inf])`. We note that unlike prior work where such parameters have been found to be impor-

tant [[Besiroglu et al., 2024](#), [Hoffmann et al., 2022](#)], we did not find them to be critical considering how our fits are simple and over 1 dimension.