From Acceleration to Saturation: Scaling Behavior of Bootstrapped Language Model Pretraining *

Seng Pei Liew†
SB Intuitions
Tokyo, Japan

Takuya Kato
SB Intuitions
Tokyo, Japan

Abstract

Bootstrapped pretraining, i.e., the reuse of a pretrained base model for further pretraining, such as continual pretraining or model growth, is promising at reducing the cost of training language models from scratch. However, its effectiveness remains unclear, especially when applied to overtrained base models. In this work, we empirically study the scaling behavior of bootstrapped pretraining and find that its scaling efficiency diminishes in a predictable manner: The scaling exponent with respect to second-stage pretraining tokens decreases logarithmically with the number of tokens used to pretrain the base model. The joint dependence on first- and second-stage tokens is accurately modeled by a simple scaling law. Such saturation effect reveals a fundamental trade-off in multi-stage pretraining strategies: the more extensively a model is pretrained, the less additional benefit bootstrapping provides. Our findings provide practical insights for efficient language model training and raise important considerations for the reuse of overtrained models.

1 Introduction

Large language models (LLMs) have recently shown astounding performance in various natural language processing tasks, reaching human-level capabilities in some cases [1, 2, 23]. However, training/pretraining these models from scratch is computationally expensive and time-consuming, requiring weeks to months even with powerful GPU clusters. To address this challenge *within the pretraining stage itself*, researchers have explored strategies for bootstrapping/reusing existing pretrained (base) models for various purposes. These include strategies to learn with new pretraining data, or to increase the model size, without starting from scratch. We refer such strategies that pretrain a model in multiple stages collectively as *bootstrapped pretraining*.

More specifically, we are interested in bootstrapped pretraining strategies that (1) improve domain-specific performance through *continual pretraining* (CPT) of the base model (see, e.g., [34]); or (2) scale model capacity via *model growth* techniques, which increase the model size reusing base model parameters to speed up training [13]. These strategies have been shown to be promising at accelerating performance improvement of LLMs while reducing the computational cost compared to pretraining from scratch.

On the other hand, neural networks could lose plasticity after training for a long time, thereby reducing their ability to learn from new data effectively [5, 3, 58]. Grown models may also similarly suffer from bad initialization, resulting in ineffective utilization of their enlarged capacity. Hence, it is unclear if the second stage of pretraining can be effectively applied to models that are overtrained, or whether the scaling behavior remains consistent across different training stages and model sizes.

*Accepted to NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling.

†sengpei.liew@sbintuitions.co.jp

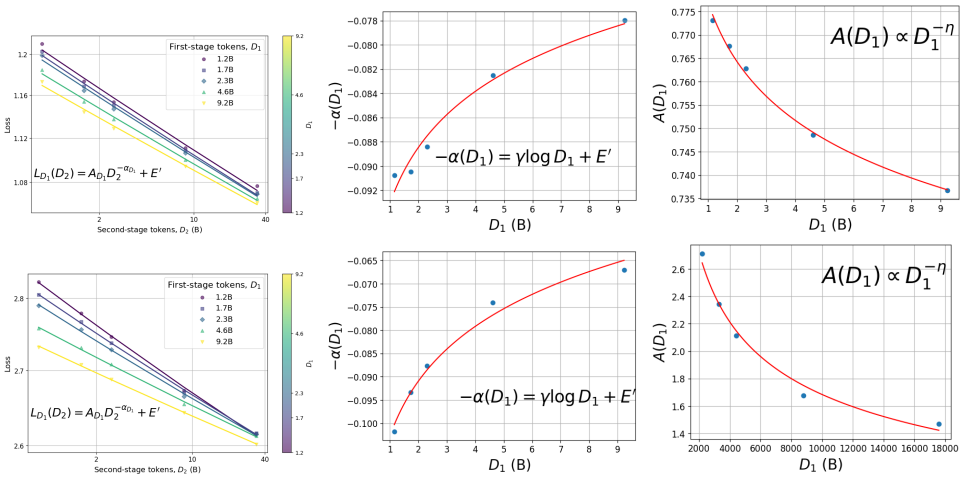

Figure 1: **Bootstrapped pretraining with overtrained base models leads to saturation in scaling behavior. Left: D_2 has power-law scaling.** We show scaling behavior of second-stage training tokens (D_2) for different values of first-stage tokens (D_1). **Middle: Interaction term explains decreasing exponents.** The fitted exponents in the left plots are used to fit Equation 5 as a function of D_1, and are shown to agree well with the functional form. **Right: Scaling factor has power-law scaling w.r.t. D_1. Top: Continual pretraining (CPT) on code data. Bottom: Model growth by stacking.**

In this paper, we seek to understand the scaling behavior of bootstrapped pretraining methods by running a multitude of controlled language modeling experiments. We find that, from the performance (cross-entropy loss) perspective, bootstrapping overtrained base models leads to *saturation* in scaling during the second-stage pretraining. Specifically, we show that this effect is *quantifiable* via scaling laws: **The scaling exponent (with respect to the number of training tokens in the second stage) decreases as the base model is trained for longer periods of time. The decrease is proportional to the logarithm of the number of training tokens invested to the first stage.** See Figure 1.

This *saturation* effect underscores a fundamental trade-off in multi-stage pretraining strategies: bootstrapping a more heavily pretrained base model does not necessarily lead to better performance. Our scaling laws enable a quantitative assessment of when bootstrapping is beneficial versus when training from scratch is preferable, offering practical guidance for efficient language model training.

Contributions. More technically, our main contributions are summarized as follows:

- We conduct extensive experiments on language models of various sizes and pretraining tokens/datasets, to study the scaling behaviors of bootstrapped pretraining methods, including continual pretraining and model growth.
- We formulate and empirically validate possible functional forms to quantify the the scaling behavior of these methods. We find that for a wide range of configuration of these methods, the following scaling relation holds:

$$L(D_1, D_2) = AD_1^{-\alpha_1} D_2^{-\alpha_2 + \alpha_3 \log D_1} + E$$

  where L is the validation loss after the second stage, D_1 and D_2 are the number of training tokens in the first and second stages, respectively, and $A, \alpha_1, \alpha_2, \alpha_3, E$ are *positive* constants. We denote the term $\alpha_3 \log D_1$ as the *interaction term*. The scaling exponent with respect to D_2 then quantifies the saturation effect of bootstrapped pretraining.
- We also formulate a joint scaling law incorporating model size in addition to dataset sizes, see Equation 7. Using these results, we discuss the practical implications of the scaling laws for efficient bootstrapping of language models. Particularly, we show how the scaling laws provide guidance on when to train from scratch instead of bootstrapping, and how compute optimality is affected by bootstrapped pretraining.

Notations used throughout the paper are summarized in Appendix A.

1.1 Related Work

Continual pretraining. CPT has been widely used to adapt existing LLMs to specific domains [61, 35, 36], including code [69, 71, 26] and mathematics [25, 53] domains, to be studied in this work. Systematic studies at scale are relatively fewer, and our methodology mainly follows [34].

Model growth. While [13] was the first model growth work in the deep learning era, the idea of growing neural networks from smaller ones can be traced to the 90s [20, 21]. More recent language model-related model growth methods include [24, 12, 65, 67, 55, 19, 70], which were subsequently systematically analyzed in [17], motivating our choice of model growth methods.

Scaling laws. We focus on modeling only the *final* loss, which has a power-law like behavior, a generic phenomenon not only occuring in neural networks [31, 37], but is also observed in other natural as well as man-made phenomena [15]. We shall additionally note that recent scaling law studies attempted to fit the whole loss curve albeit with more sophisticated functional forms [62, 66, 50].

Similar to CPT, the reduced capability of transfer learning of code data from models pretrained on natural language data was observed in [29], where the authors denoted as ossification. A similar phenomenon was also observed in the pretraining-instruction-tuning pipeline [59]. These studies however did not quantify the saturated scaling behavior. Previous fine-tuning scaling studies [45, 72, 7] did not observe the subtler saturation effects as well, which is perhaps due to the smaller scale (in terms of training tokens) of their experiments.

2 Experimental Setup

Model configuration. We consider decoder-only transformers [64] pretrained with an autoregressive language modeling objective. Our architecture is LLaMA-like [63], incorporating refinements such as SwiGLU activation functions [54] and rotary position embeddings [60]. We use the LLaMA tokenizer with a vocabulary size of 32,000. We consider a suite of model sizes up to 1.1B in our experiments. Table 3 of Appendix B contains the model configuration used in this paper.

Training configuration. All experiments are conducted using the Megatron-LM library [57]. We train models in mixed precision (bfloat16) using the AdamW optimizer [43], with maximum learning rates tuned separately for each model size. The learning rate follows the *warmup-stable-decay* (WSD) schedule [8, 33], with the final learning rate decayed to one tenth of the peak value. This schedule enables long training runs and facilitates checkpoint reuse for emulating shorter effective training budgets, reducing computational cost. In Appendix C, we also provide an ablation study with the cosine learning rate schedule to show that they achieve similar performance. We train each model for a number of tokens for up to roughly 200 times the model size in tokens in total (up to a few hundred B tokens overall). Tables 4 and 5 in Appendix summarize the main training configurations. See also Appendix B for further experimental details and Appendix C for training details and evaluation on downstream tasks.

Two stages of pretraining. In the context of bootstrapped pretraining, we are primarily interested in how the loss behaves with respect to the number of training tokens used in the two stages of pretraining:

- D_1: the number of tokens used in the **first-stage** pretraining (base model).
- D_2: the number of tokens used in the **second-stage** pretraining (CPT or model growth).

Base model. For the first-stage pretraining, we use the CommonCrawl portion of the Slimpajama-DC dataset [56], containing 368B tokens in total. The models trained in this stage serve as base checkpoints for the second-stage bootstrapped pretraining experiments.

Continual Pretraining. CPT refers to further training a pretrained model on a large dataset, typically billions of tokens, with the goal of improving performance on a new domain. We distinguish CPT from *instruction tuning*, which typically trains with smaller datasets (millions of tokens or fewer), and is thus harder to analyze the scaling behavior. For our experiments, we perform CPT on the base model on two domain-specific datasets: **Code corpus:** Stack/StarCoder [40]; and **mathematics corpus:** OpenWebMath [48]. Unless otherwise noted, we adopt the same optimizer and learning rate configurations as the first-stage pretraining.

Model growth. Model growth refers to increasing model capacity by adding new layers and/or expanding hidden dimensions, thereby increasing the number of trainable parameters. This strategy aims to leverage the representations learned during first-stage training, allowing the larger model to accelerate learning in the second stage. We investigate two model growth techniques shown to be the most effective (in terms of adding new parameters in the width and depth directions) in prior work [17] (see also Appendix C.5 for more details):

- **Width expansion:** Add new neurons to each layer while using *function-preserving initialization* (FPI), ensuring that the expanded model initially reproduces the behavior of the smaller model [12, 17].
- **Depth-wise stacking:** Add new layers to the top of the transformer by copying existing layer weights, thereby extending model depth [12, 17] .

After performing this procedure, the larger models are trained on the same dataset as the base model. For both bootstrapped pretraining scenarios in consideration, the validation loss is evaluated on a held-out set from the second-stage dataset. We illustrate the whole framework in Figure 2.

3 Formulating the Data Scaling Laws

In this Section, we derive possible functional forms for the scaling laws of bootstrapped pretraining methods, focusing on the number of training tokens as the main variable of interest. These are then fitted and compared empirically in later sections.

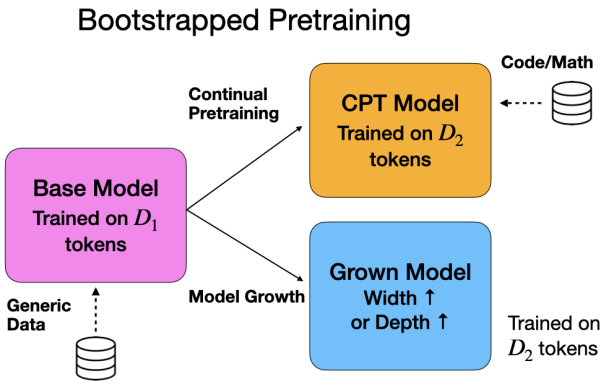

Figure 2: **Illustration of bootstrapped pretraining in consideration.** Bootstrapped pretraining consists of two stages: (1) first-stage pretraining of a base model for D_1 tokens on internet/generic data; (2) bootstrapping/second-stage pretraining via continual pretraining or model growth for D_2 tokens. Section 3 and 4 study and develop scaling laws as a function of these two variables (and additionally model size, N in Section 5) to predict the final loss after the second stage.

Power-law ansatz. We begin by assuming that the validation loss L follows the widely observed *power-law scaling* with respect to a single variable of interest, such as the number of training tokens or the model size [31, 30, 28]:

$$L = AX^{-\alpha} + E,$$

where X is the variable of interest, α is the scaling exponent, A is the scaling factor, and E is the irreducible loss due to the inherent entropy of the data distribution. [3]

Our goal is to find a functional form $L(D_1, D_2)$ that captures how the second-stage loss depends jointly on both stages. To derive a principled formulation, we impose two natural constraints:

Condition 1: For a fixed base model trained on D_1 tokens, the second-stage loss should follow a power law with respect to the number of tokens D_2:

$$L(D_1, D_2) = L_{D_1}(D_2) = AD_2^{-\alpha} + E. \tag{1}$$

This is consistent with classical neural scaling laws and reflects the expectation that, when starting from a fixed initialization, additional tokens improve performance predictably.

[3] Strictly speaking, we assume the form $L = \frac{A}{(X+1)^\alpha} + E$, which ensures finiteness at $X \to 0$. However, since in practice $X \gg 1$ (typically 10^6 or more), we approximate it as $L \approx AX^{-\alpha} + E$ for notational simplicity. Moreover, we use the same symbols A, α, and E to denote the scaling factor, exponent, and additive constant, respectively, for notational conveniences, although they may differ across different variables of interest.

Condition 2: For any fixed value of D_2, the loss should exhibit power-law behavior with respect to the number of first-stage tokens D_1: $L(D_1, D_2) = L_{D_2}(D_1) = AD_1^{-\alpha} + E$. This is consistent with the power-law ansatz and captures the intuition that a better-trained base model (i.e., larger D_1) should result in lower loss. Moreover, this condition implies that as $D_2 \to 0$, the loss should continuously approach that of the base model: $\lim_{D_2 \to 0} L(D_1, D_2) = AD_1^{-\alpha} + E$, which is a natural requirement for: [4]

- **CPT:** The second stage's initial loss should begin from the base model's (evaluated on second-stage dataset).
- **Function-preserving model growth:** When model capacity is expanded but initialized carefully, the initial loss should remain close to the base model's loss.

Together, these conditions lead to the following candidate formulations that jointly satisfy both requirements:

$$\textbf{Multiplicative:} \quad L(D_1, D_2) = AD_1^{-\alpha_1} D_2^{-\alpha_2 + \alpha_3 \log D_1} + E. \tag{2}$$

$$\textbf{Additive:} \quad L(D_1, D_2) = AD_1^{-\alpha_1} + FD_2^{-\alpha_2} + E. \tag{3}$$

$$\textbf{Hybrid:} \quad L(D_1, D_2) = \big(AD_1^{-\alpha_1} + F\big) D_2^{-\alpha_2} + E. \tag{4}$$

Note that since $D_1^{-\alpha_1} D_2^{-\alpha_2 + \alpha_3 \log D_1} = D_1^{-\alpha_2 + \alpha_3 \log D_2} D_2^{-\alpha_2}$, the multiplicative form with an interaction term satisfies our conditions, while other forms do not. We further note that multiplicative scaling laws have been observed in [45, 72] but without the interaction term. [41] empirically showed that sparse upcycling (training sparse mixture-of-experts (MoE) models reusing existing dense models) follows a scaling law similar to Equation 2 but under different motivation and conditions. Our work directly extends their findings to other pretraining paradigms. Equation 4 has also been studied in [4].

4 Data Scaling Laws

Fitting the scaling laws. To determine the functional form described in the previous Section that best describes the scaling behavior of bootstrapped pretraining methods, we use a model of size 0.1B, trained on a 5×5 grid of D_1, D_2. For CPT, we consider both code and mathematics datasets, while for model growth, we consider both width expansion and depth-wise stacking that double the size of the base model. [5]

Running the experiments as described above and obtaining the results, we fit the losses with the functional forms of Equations 2, 3, and 4. Following [32, 6], we perform optimization using the Huber loss ($\delta = 10^{-3}$) and the BFGS algorithm, to fit the logarithm of the loss via the LogSumExp trick applied to the RHS of functional forms. The leave-one-out root mean square error (RMS) serves as the goodness-of-fit metric.

As can be seen in Table 1, the multiplicative form with interaction, Equation 2, consistently achieves the lowest error across all methods and datasets, indicating that it best captures the underlying scaling behavior.

Bottom-up empirical evidence. To strengthen our proposal for Equation 2, we demonstrate through a *bottom-up* observational approach (instead of the *top-down* approach of deriving the scaling laws by imposing suitable conditions) that the multiplicative nature of the scaling law, including the interaction term, arises naturally from empirical data.

In Figure 1, we first perform power-law fits to show that the loss follows Equation 1 for fixed D_1, validating Condition 1 we impose for deriving the scaling laws. More importantly, we observe that the fitted scaling exponent of Equation 1 decreases as D_1 increases, with the model growth method more

[4] As in Footnote 3, we omit the $+1$ term in $D + 1$ expressions for notational simplicity, such that terms proportional to D_2 are finite as $D_2 \to 0$.

[5] We increase the size of the hidden dimension by $\sqrt{2}$ for width expansion; and we double the number of non-embedding layers for stacking.

pronouncedly so. To quantify this trend, we express (the minus of) the scaling exponent as a function of D_1, denoted by $-\alpha(D_1)$. A scatter plot of $-\alpha(D_1)$ reveals a clear logarithmic dependence:

$$-\alpha(D_1) = \gamma \log D_1 + E'. \tag{5}$$

This relationship fits the empirical data well, as shown in the same Figure. Substituting Equation 5 into Equation 1, we arrive at a term of the form $D_2^{-E'+\gamma \log D_1}$, providing direct empirical justification for the interaction term. This demonstrates that the interaction term is not a theoretical artifact, but rather a necessary component for capturing observed trends in the data. Additionally in the same Figure, we show that the multiplicative dependence, $A \propto D_1^{-\alpha_1}$ also arises from empirical observations.

We further show that the loss follows a power law with respect to D_1 for fixed D_2 in Figure 3 (with more plots in Appendix D). This validates Condition 2 we impose for deriving the scaling laws.

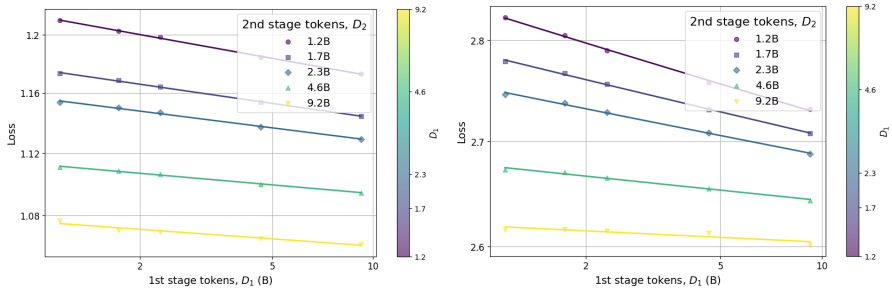

Figure 3: D_1 **has power-law scaling.** We show scaling behavior of first-stage training tokens (D_1) for different values of second-stage tokens (D_2), indicating that D_1 also has power-law scaling. Left: Continual pretraining on code data; Right: model growth by stacking. More plots in Appendix D.

4.1 Interpretations

Several quantitative and qualitative insights can be made from the scaling law. Fixing D_1, the effective scaling factor for D_2 becomes $AD_1^{-\alpha_1}$. As D_1 increases, this factor decreases, resulting in a lower initial loss at the start of second-stage pretraining, agreeing with the conventional wisdom that better-pretrained base models (larger D_1) provide stronger initialization for second-stage pretraining.

Table 1: **Multiplicative scaling law with interaction consistently achieves lowest error.** Leave-one-out RMS error ($\times 10^{-3}$) for fitting the loss for reusing a 0.1B model. Functional forms are from Equations 2, 3, and specific cases with $\alpha_3 = 0$.

RMS ($\times 10^{-3}$)	**Mul.**	**Mul. ($\alpha_3 = 0$)**	**Add.**	**Hyb.**
CPT (code)	**1.573**	2.095	3.913	2.245
CPT (math)	**1.737**	3.147	7.443	3.340
Expand	**2.790**	4.837	9.855	4.841
Stack	**2.845**	7.818	10.323	7.748

However, also at fixed D_1, the effective scaling exponent with respect to D_2 is given by $\alpha_2 - \alpha_3 \log D_1$. This implies that as the base model becomes more overtrained (i.e., larger D_1), the improvement in loss from additional second-stage tokens becomes increasingly marginal. In other words, the returns from increasing D_2 diminish, manifesting as saturation effects at higher D_1.

As shown in Table 7 in Appendix D, the fitted values of α_3 are consistently positive across all settings and datasets, reinforcing the generality of this saturation phenomenon (see also Section 5.1 for additional evidence). Also note that α_3 is typically at least an order of magnitude smaller than other exponents (with its effects accumulating logarithmically with D_1), which may explain why it has not been observed widely in previous scaling studies.

5 A Closer Look At The Scaling Behavior

We first perform a more detailed analysis of the scaling behavior of bootstrapped pretraining methods, by examining more variants of scaling law and training methods. The aim is to validate the robustness

of the multiplicative scaling law with interaction. We then propose and validate a joint scaling law of dataset and model sizes. The rest of the Section is subsequently dedicated to studying the practical implications of the scaling laws.

5.1 Variants of Bootstrapped Pretraining

Variant of CPT scaling law. When the dataset used for CPT is the same as the base model, we expect the scaling formula shown below holds:

$$L(D_1, D_2) = A(D_1 + D_2)^{-\alpha} + E \tag{6}$$

Hence, one may expect this holds for CPT on a different dataset as well, as assumed in [51, 66]. Fitting the above formula for the code (math) CPT scenario, we find that the RMS error is 0.0213 (0.0235), higher than the best ones in Table 1. Hence, our scaling law models CPT better, in contrast to previous assumptions, which may have overlooked the overtraining effects on the model.

Variants of CPT. We also consider CPT variants that are commonly used in practice [34]:

- **CPT with replay:** A portion of the first-stage data is mixed into the second-stage data, which is common for mitigating catastrophic forgetting. We consider a replay ratio of 0.25, i.e., 25% of the second-stage data is from the first stage.
- **CPT from stable phase:** We consider CPT starting from a base model checkpoint in the stable phase of the WSD learning rate schedule. This is to avoid adverse effects from re-warming the learning rate from a decayed value.

Variants of model growth. We further consider model growth variants.

- **Larger growth factor:** We consider a larger growth factor of 4 times (instead of 2).
- **Model growth from stable phase:** Similar to CPT, we train the stacked model starting from a base model checkpoint in the stable phase of the WSD learning rate schedule. This also avoids re-warming the learning rate from a decayed value.

We again conduct experiments using a 0.1B model, and fit the scaling laws with all possible functional forms as in Section 4 for the above variants. Results are summarized in Table 2, with fitted scaling exponents also shown in Table 7 of Appendix D. These results show that the propsed scaling law still has the lowest RMS, and variants like replay and larger growth factor do not change the functional form. Furthermore, the saturation effects are unlikely to be artifacts of re-warmings of the learning rate. Overall, the proposed scaling law is robust across datasets, configurations, and methods investigated.

5.2 Joint Scaling Incorporating Model Size

Let us first focus on CPT where the model size remains unchanged after bootstrapping. We proceed to extend the data scaling law to include model size N. To determine the functional form, we impose that the loss follows the well-grounded "Chinchilla" scaling law [32] (which jointly models the base model's loss with respect to dataset and model sizes) with respect to D_2 and N:

Table 2: **Multiplicative scaling law with interaction is robust over various configurations (RMS Error).** Leave-one-out RMS error ($\times 10^{-3}$) for fitting the loss for variants of bootstrapped pretraining. See Table 1 for definitions and Section 5 for variant details.

RMS ($\times 10^{-3}$)	**Mul.**	**Mul. ($\alpha_3 = 0$)**	**Add.**	**Hyb.**
CPT (replay)	**0.987**	2.222	4.646	2.223
CPT (stable)	**1.371**	2.366	4.315	2.367
Stack (x4)	**2.152**	6.557	8.866	6.667
Stack (stable)	**2.957**	8.757	10.559	8.822

$$L_{D_1}(D_2, N) = AD_2^{-\alpha} + N^{-\beta} + E.$$

The straightforward functional form that satisfies the Chinchilla-style scaling law is:

$$L(D_1, D_2, N) = AD_1^{-\alpha_1} D_2^{-\alpha_2 + \alpha_3 \log D_1} + BN^{-\beta} + E, \tag{7}$$

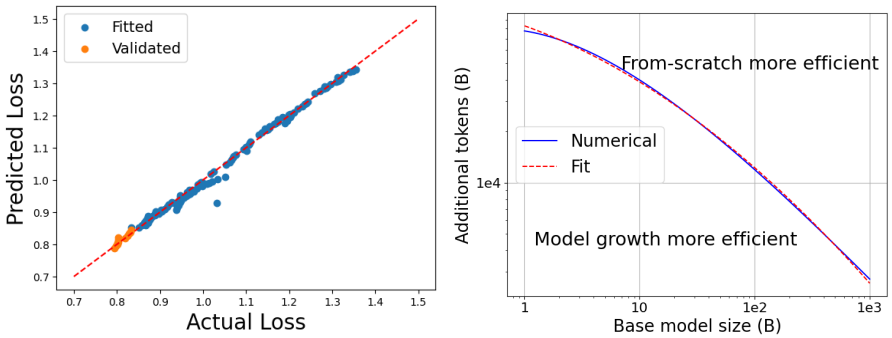

Figure 4: **Left: Joint scaling law fit for continual pretraining on code data.** Orange points indicate the 10% of data with lowest losses used for validation. **Right: Model growth efficiency decreases with sunk cost and model size.** For token budgets above the curve(s), training from scratch is more efficient than stacking-based model growth; for budgets below, model growth remains advantageous. Shown are the numerical (blue) and analytical (red) solutions of Equation 8.

which is also consistent with the conditions (power-law ansatz) imposed in Section 3.

Fitting the joint scaling law. We conduct experiments with base models of sizes 15 million (M), 44M, 0.1B, 0.2B, 0.5B and 1B, training them with different numbers of first and second-stage tokens as in Section 4. To fit the joint scaling law, we use the same fitting procedure as in Section 4, additionally taking N as an additional variable. In the left panel of Figure 4, we show that the formula fits the data well for CPT on code, and the goodness-of-fit to the validation data points also shows that it can be extrapolated to larger model and dataset sizes. See Table 8 in Appendix D.2 for the fitted coefficients. In the same Appendix, we further extend our results to model growth where model sizes increase.

5.3 Practical Implications

We consider two practical implications based on the extrapolation of the scaling laws.

Training from scratch versus bootstrapped pretraining. We show that by comparing the scaling laws with the base model scaling laws, one can determine whether bootstrapped pretraining is more efficient than training *from scratch* under different circumstances.

A key practical consideration is how much of the *initial investment* in pretraining, the so-called *sunk cost* (D_1), can be effectively leveraged in the second stage of training. This question has been explored in the context of MoE upcycling strategies [38, 41]. For example, it was found that upcycling yields benefits up to 120% of the sunk cost. That is, to match the performance of an upcycled MoE model that underwent an additional 0.4 trillion tokens of training after an initial 2T tokens, training a comparable MoE from scratch would require 2.4T tokens—representing an effective saving of 2T tokens.

To investigate this in model growth, we define D^* as the number of tokens required for training from scratch to match the performance of a grown model with the same sunk cost, following [41]:

$$L_{2N}^{\text{scratch}}(D^*) = L_N^{\text{grown}}(D_1 = D^*, D_2 = D^*) \tag{8}$$

where $L_{2N}^{\text{scratch}}(D)$ is the loss of a model of size $2N$ trained from scratch for D tokens, and $L_N^{\text{grown}}(D_1, D_2)$ is the loss of a model of size $2N$ grown from a base model of size N trained for D_1 tokens, and then trained for D_2 tokens. To obtain $L_{2N}^{\text{scratch}}(D)$, we fit the losses of models trained from scratch with the Chinchilla-style scaling law (see Table 8 in Appendix D.2 for the fitted coefficients).

The right panel of Figure 4 shows that D^* decreases with increasing model size, with D^* equal to 13T tokens for a 100B model. When $D_2 \lesssim D^*$, the required from-scratch tokens to catch up is more than 100% of D_1: model growth remains more efficient than training from scratch. However, beyond this threshold, from-scratch training becomes more efficient. Solving Equation 8, we can approximate the threshold D^* analytically as $D^* \simeq 13 \left(\frac{N}{10^{11}}\right)^{-0.6-0.04\log(N/10^{11})}$ T tokens.

For CPT on code data, we can perform similar extrapolation as well, finding that a 7B model trained from scratch begins to outperform bootstrapped models (pretrained on 10T tokens) once more than 300B tokens of domain-specific data are used. See Appendix E for details. These findings offer quantitative guidance for deciding between bootstrapped pretraining and training from scratch (or from intermediate checkpoints), depending on the extent to which the base model is overtrained, as well as the available budget.

Compute optimality of bootstrapped pretraining. For scaling studies of LLMs, the cost of training is often estimated using floating point operations (FLOPs), which we simply refer to as compute. The compute cost can be approximated as $C = 6ND$ [37]. One is often interested in training LLMs with the least amount of compute. For CPT, optimizing the compute based on Equation 2 leads to the scaling relations:

$$D_2^{\text{opt}} \propto C_2^{\frac{\beta}{\beta+\alpha_{\text{eff}}}}, \quad N^{\text{opt}} \propto C_2^{\frac{\alpha_{\text{eff}}}{\beta+\alpha_{\text{eff}}}} \tag{9}$$

where $\alpha_{\text{eff}} \coloneqq \alpha_2 - \alpha_3 \log D_1$. Notably, as D_1 increases, α_{eff} decreases, meaning larger models require more tokens for compute-optimal bootstrapped pretraining. We give a similar derivation for model growth in Appendix E. We leave empirical verification of these relationships to future work, primarily due to cost considerations (see Appendix B.4 for an estimate of GPU hours spent in the work).

6 Discussion and Conclusion

In this paper, we have shown that while bootstrapped pretraining methods can accelerate learning, they exhibit scaling saturation as the base model is overtrained. Our work bears practical implications as even relatively tiny models are overtrained to trillions of tokens nowadays [73, 18], and bootstrapped pretraining is widely used in practice. We encourage the community to report the amount of pretraining data used for the base model, and even make intermediate checkpoints available, which can be more useful than the final model for bootstrapped pretraining.

We have provided a fairly broad scaling study of two-stage pretraining and there are many potential future directions worth diving deeper into. Let us highlight a few of them.

Incorporating other factors into the scaling laws. We have not considered the scaling behavior with respect to other method-specific factors, such as the CPT replay ratio [51, 66] and the growth factor in model growth [17], as (1) our focus is on the more broadly applicable scaling behavior (i.e., scaling with respect to training tokens), and (2) adding more factors would increase the complexity of the scaling laws and the number of parameters to fit (and hence the amount of additional experiments/GPU hours/costs needed). We nevertheless expect that these factors can be incorporated quite naturally into our scaling laws which are power-law based.

Theory. Providing a theoretical explanation of the scaling laws is an important open question in the field, but existing work has been focusing on relatively simple theoretical setups like *power-law random feature models* to reproduce power-law scaling in deep neural networks [44, 11, 47]. To our best knowledge, there is no existing theory that can explain the change of scaling exponents with respect to the amount of pretraining data. In the current work, we have focused on the practical aspects of the scaling laws, following previous influential scaling law work [37, 32], and have not attempted to provide a theoretical explanation of the scaling laws as the current theoretical understanding of scaling laws is still limited. Nevertheless, it would be interesting to study if such a change of scaling exponents can be explained by existing theories or if new theoretical frameworks are needed.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Anthropic. Introducing the next generation of claude, 2024.

[3] Jordan Ash and Ryan P Adams. On warm-starting neural network training. *Advances in neural information processing systems*, 33:3884–3894, 2020.

[4] Matthew Barnett. An empirical study of scaling laws for transfer. *arXiv preprint arXiv:2408.16947*, 2024.

[5] Tudor Berariu, Wojciech Czarnecki, Soham De, Jorg Bornschein, Samuel Smith, Razvan Pascanu, and Claudia Clopath. A study on the plasticity of neural networks. *arXiv preprint arXiv:2106.00042*, 2021.

[6] Tamay Besiroglu, Ege Erdil, Matthew Barnett, and Josh You. Chinchilla scaling: A replication attempt. *arXiv preprint arXiv:2404.10102*, 2024.

[7] Louis Bethune, David Grangier, Dan Busbridge, Eleonora Gualdoni, Marco Cuturi, and Pierre Ablin. Scaling laws for forgetting during finetuning with pretraining data injection. *arXiv preprint arXiv:2502.06042*, 2025.

[8] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

[9] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. pages 2397–2430. PMLR, 2023.

[10] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7432–7439, 2020.

[11] Blake Bordelon, Alexander Atanasov, and Cengiz Pehlevan. A dynamical model of neural scaling laws. In *Proceedings of the 41st International Conference on Machine Learning*, pages 4345–4382, 2024.

[12] Cheng Chen, Yichun Yin, Lifeng Shang, Xin Jiang, Yujia Qin, Fengyu Wang, Zhi Wang, Xiao Chen, Zhiyuan Liu, and Qun Liu. bert2bert: Towards reusable pretrained language models. *arXiv preprint arXiv:2110.07143*, 2021.

[13] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[14] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[15] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[16] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[17] Wenyu Du, Tongxu Luo, Zihan Qiu, Zeyu Huang, Yikang Shen, Reynold Cheng, Yike Guo, and Jie Fu. Stacking your transformers: A closer look at model growth for efficient llm pre-training. *arXiv preprint arXiv:2405.15319*, 2024.

[18] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[19] Utku Evci, Bart van Merrienboer, Thomas Unterthiner, Max Vladymyrov, and Fabian Pedregosa. Gradmax: Growing neural networks using gradient information. *arXiv preprint arXiv:2201.05125*, 2022.

[20] Scott Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

[21] Scott E. Fahlman. The recurrent cascade-correlation architecture. In *Neural Information Processing Systems*, 1990.

[22] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024.

[23] Gemini Team Google. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[24] Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. Efficient training of bert by progressively stacking. In *International conference on machine learning*, pages 2337–2346. PMLR, 2019.

[25] Zheng Gong, Kun Zhou, Xin Zhao, Jing Sha, Shijin Wang, and Ji-Rong Wen. Continual pre-training of language models for math problem understanding with syntax-aware memory network, 2022.

[26] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

[27] Alex Hägele, Elie Bakouch, Atli Kosson, Leandro Von Werra, Martin Jaggi, et al. Scaling laws and compute-optimal training beyond fixed training durations. *Advances in Neural Information Processing Systems*, 37:76232–76264, 2024.

[28] Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B Brown, Prafulla Dhariwal, Scott Gray, et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

[29] Danny Hernandez, Jared Kaplan, Tom Henighan, and Sam McCandlish. Scaling laws for transfer. *arXiv preprint arXiv:2102.01293*, 2021.

[30] Joel Hestness, Newsha Ardalani, and Gregory Diamos. Beyond human-level accuracy: Computational challenges in deep learning. pages 1–14, 2019.

[31] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

[32] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *Proceedings of the 36th International Conference on Neural Information Processing Systems*, pages 30016–30030, 2022.

[33] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

[34] Adam Ibrahim, Benjamin Thérien, Kshitij Gupta, Mats Leon Richter, Quentin Gregory Anthony, Eugene Belilovsky, Timothée Lesort, and Irina Rish. Simple and scalable strategies to continually pre-train large language models. *Transactions on Machine Learning Research*.

[35] Joel Jang, Seonghyeon Ye, Changho Lee, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, and Minjoon Seo. Temporalwiki: A lifelong benchmark for training and evaluating ever-evolving language models. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 6237–6250. Association for Computational Linguistics, 2022.

[36] Joel Jang, Seonghyeon Ye, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, Stanley Jungkyu Choi, and Minjoon Seo. Towards continual knowledge learning of language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[37] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[38] Aran Komatsuzaki, Joan Puigcerver, James Lee-Thorp, Carlos Riquelme Ruiz, Basil Mustafa, Joshua Ainslie, Yi Tay, Mostafa Dehghani, and Neil Houlsby. Sparse upcycling: Training mixture-of-experts from dense checkpoints.

[39] Teven Le Scao, Thomas Wang, Daniel Hesslow, Stas Bekman, M Saiful Bari, Stella Biderman, Hady Elsahar, Niklas Muennighoff, Jason Phang, Ofir Press, Colin Raffel, Victor Sanh, Sheng Shen, Lintang Sutawika, Jaesung Tae, Zheng Xin Yong, Julien Launay, and Iz Beltagy. What language model to train if you have one million GPU hours? pages 765–782, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[40] Raymond Li, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, LI Jia, Jenny Chim, Qian Liu, et al. Starcoder: may the source be with you! *Transactions on Machine Learning Research*.

[41] Seng Pei Liew, Takuya Kato, and Sho Takase. Scaling laws for upcycling mixture-of-experts language models. In *Forty-second International Conference on Machine Learning*.

[42] Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. Logiqa: a challenge dataset for machine reading comprehension with logical reasoning. pages 3622–3628, 2021.

[43] Ilya Loshchilov, Frank Hutter, et al. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 5, 2017.

[44] Alexander Maloney, Daniel A Roberts, and James Sully. A solvable model of neural scaling laws. *arXiv preprint arXiv:2210.16859*, 2022.

[45] Hiroaki Mikami, Kenji Fukumizu, Shogo Murai, Shuji Suzuki, Yuta Kikuchi, Taiji Suzuki, Shin-ichi Maeda, and Kohei Hayashi. A scaling law for syn2real transfer: How much is your pre-training effective? pages 477–492. Springer, 2022.

[46] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, 2016.

[47] Elliot Paquette, Courtney Paquette, Lechao Xiao, and Jeffrey Pennington. 4+ 3 phases of compute-optimal neural scaling laws. *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[48] Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text. *arXiv preprint arXiv:2310.06786*, 2023.

[49] Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models. *Advances in Neural Information Processing Systems*, 37:100535–100570, 2024.

[50] Shikai Qiu, Lechao Xiao, Andrew Gordon Wilson, Jeffrey Pennington, and Atish Agarwala. Scaling collapse reveals universal dynamics in compute-optimally trained neural networks. *arXiv preprint arXiv:2507.02119*, 2025.

[51] Haoran Que, Jiaheng Liu, Ge Zhang, Chenchen Zhang, Xingwei Qu, Yinghao Ma, Feiyu Duan, Zhiqi Bai, Jiakai Wang, Yuanxing Zhang, et al. D-cpt law: Domain-specific continual pre-training scaling law for large language models. *Advances in Neural Information Processing Systems*, 37:90318–90354, 2024.

[52] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[53] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[54] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[55] Sheng Shen, Pete Walsh, Kurt Keutzer, Jesse Dodge, Matthew Peters, and Iz Beltagy. Staged training for transformer language models. In *International Conference on Machine Learning*, pages 19893–19908. PMLR, 2022.

[56] Zhiqiang Shen, Tianhua Tao, Liqun Ma, Willie Neiswanger, Zhengzhong Liu, Hongyi Wang, Bowen Tan, Joel Hestness, Natalia Vassilieva, Daria Soboleva, et al. Slimpajama-dc: Understanding data combinations for llm training. *arXiv preprint arXiv:2309.10818*, 2023.

[57] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[58] Shagun Sodhani, Sarath Chandar, and Yoshua Bengio. Toward training recurrent neural networks for lifelong learning. *Neural computation*, 32(1):1–35, 2020.

[59] Jacob Mitchell Springer, Sachin Goyal, Kaiyue Wen, Tanishq Kumar, Xiang Yue, Sadhika Malladi, Graham Neubig, and Aditi Raghunathan. Overtrained language models are harder to fine-tune. *arXiv preprint arXiv:2503.19206*, 2025.

[60] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[61] Yu Sun, Shuohuan Wang, Yu-Kun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. ERNIE 2.0: A continual pre-training framework for language understanding. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8968–8975. AAAI Press, 2020.

[62] Howe Tissue, Venus Wang, and Lu Wang. Scaling law with learning rate annealing. *arXiv preprint arXiv:2408.11029*, 2024.

[63] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[65] Peihao Wang, Rameswar Panda, Lucas Torroba Hennigen, Philip Greengard, Leonid Karlinsky, Rogerio Feris, David Daniel Cox, Zhangyang Wang, and Yoon Kim. Learning to grow pretrained models for efficient transformer training. *arXiv preprint arXiv:2303.00980*, 2023.

[66] Xingjin Wang, Howe Tissue, Lu Wang, Linjing Li, and Daniel Dajun Zeng. Learning dynamics in continual pre-training for large language models. *arXiv preprint arXiv:2505.07796*, 2025.

[67] Yite Wang, Jiahao Su, Hanlin Lu, Cong Xie, Tianyi Liu, Jianbo Yuan, Haibin Lin, Ruoyu Sun, and Hongxia Yang. Lemon: Lossless model expansion, 2023.

[68] Johannes Welbl, Nelson F Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. *W-NUT 2017*, page 94, 2017.

[69] Prateek Yadav, Qing Sun, Hantian Ding, Xiaopeng Li, Dejiao Zhang, Ming Tan, Parminder Bhatia, Xiaofei Ma, Ramesh Nallapati, Murali Krishna Ramanathan, Mohit Bansal, and Bing Xiang. Exploring continual learning for code generation models. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 782–792. Association for Computational Linguistics, 2023.

[70] Yiqun Yao, Zheng Zhang, Jing Li, and Yequan Wang. Masked structural growth for 2x faster language model pre-training, 2024.

[71] Daoguang Zan, Bei Chen, Dejian Yang, Zeqi Lin, Minsu Kim, Bei Guan, Yongji Wang, Weizhu Chen, and Jian-Guang Lou. CERT: continual pre-training on sketches for library-oriented code generation. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 2369–2375. ijcai.org, 2022.

[72] Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. When scaling meets llm finetuning: The effect of data, model and finetuning method.

[73] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

A Notations

We summarize main notations used in the paper.

- L: Cross-entropy loss or the (natural) logarithmic loss
- D: Dataset size in token
- N: Non-embedding model size or number of non-embedding parameters
- A, B, F: Scaling factors of the power law, independent of the variable under consideration
- E: Irreducible loss of the power law, independent of the variable under consideration
- α, β, γ: Scaling exponents of the power law, independent of the variable under consideration
- n_{layer}: number of layers of a model
- d_{model}: hidden dimension size of a model
- d_{MLP}: intermediate hidden dimension size of a model

B More on Architecture and Experimental Design

B.1 Megatron-LM Configuration

Infrastructure. Our experiments are performed on multiple nodes, each consisting of 8 NVIDIA H100 80 GB GPUs, interconnected via InfiniBand HDR. The software we use for training is the Megatron-LM library [57].

We use and modify the Megatron-LM (core v0.8.0) library for our experiments[6]. Models are trained with data type bfloat16. Other optimization libraries used include FlashAttention [16] and TransformerEngine[7]. See the example scripts provided in supplementary material.

B.2 Model Configuration

Let us elaborate more on our model configuration. The intermediate hidden dimension size, d_{MLP}, is set to be four times the hidden dimension size, i.e., $4d_{\text{model}}$. Bias is not used in the linear layers. We do not consider efficiency-motivated implementations like grouped query attention as well. Attention head number is chosen to increase with model size following standard practices. Other designs of the architecture follow Llama2's closely [63].

We vary model sizes keeping the ratio $n_{\text{layer}}/d_{\text{model}}$ to lie in the range 32 to 64, as in [37]. Smaller models for used for ablation studies. See Table 3 for the exact numbers for the model configuration.

Table 3: Models used in our study and their parametric details. Note that d_{MLP}, is set to be $4d_{\text{model}}$.

Model	n_{layer}	d_{model}	n_{head}	N
15M	9	320	4	14,751,680
44M	12	480	8	44,248,800
0.1B	15	640	8	98,323,840
0.2B	21	832	8	232,623,040
0.5B	26	1,120	16	521,889,760
1B	30	1,504	16	1,085,859,424

B.3 Training Configuration

As discussed in the main text, we adopt the WSD learning rate schedule for all experiments. The number of warmup steps is set approximately equal to the model size, as suggested by [49]. In the final phase of training, the learning rate decays linearly to 10% of its peak value, with the decay phase spanning roughly 10% of the total training steps, following the setup in [27]. To emulate varying token budgets, we save intermediate checkpoints at logarithmically spaced intervals.

[6] `https://github.com/NVIDIA/Megatron-LM`

[7] `https://github.com/NVIDIA/TransformerEngine`

Table 4: Training configuration used throughout the paper.

Configuration	Details
Context length	1,024
Embedding tying	False
Optimizer	AdamW [43]
Adam β_1	0.9
Adam β_2	0.95
Adam ϵ	1e-8
Weight decay	0.1
Gradient clipping	1.0

The general training configuration is summarized in Table 4, while model-specific hyperparameters, such as warmup iterations, initialization standard deviation ($\sqrt{2/5d_{\text{model}}}$ [39]), maximum training steps, batch size, and tuned learning rate, are provided in Table 5. Batch size is scaled with model size according to standard practice, without tuning for optimality. The number of tokens is increased with model size as well, such that the parameter-to-token ratio remains roughly constant across different model sizes. 6 We note that no specialized techniques for mitigating training instabilities, such as Z-loss or QK normalization, are employed, as such instabilities do not arise in our experiments.

Table 5: **Model-dependent training configuration**. "init. size" refers to the standard deviation of the normal distribution used for initializing the weights. "Max iter." refers to the maximum iteration run on the model.

Model	warmup iter.	init. size	Max iter.	batch size	LR
15M	200	0.035	17,600	128	8e-3
44M	200	0.029	17,600	256	4e-3
0.1B	200	0.025	17,600	512	4e-3
0.2B	400	0.022	35,200	512	2e-3
0.5B	800	0.019	70,400	512	4e-4
1B	800	0.016	70,400	1024	4e-4

B.4 GPU Hours and Costs

Instead of reporting the actual runtimes on our cluster, which varied in our experiments due to many factors affecting the cluster (number of available nodes, congestion, etc.), we give a theoretical estimate of total GPU hours used for obtaining the joint scaling law, which involves running the largest tested model with most training tokens in this paper.

The estimate is as follows. We calculate the FLOPs for training the largest first and second-stage models with maximum iterations using the $6ND$ approximation, ignoring the additional FLOPs required to continued pretrain models with shorter iterations (as we can reuse the intermediate checkpoints). We further assume that the per-second TFLOPs of the GPU is 400. We obtain around 10^9 TFLOPs. Taking into account of additional experimentation and ablation runs, we estimate that around 3000 GPU hours were used for the entire project. Assuming a cost of 2 USD per GPU hour, the total cost is around 6000 USD.

C Training Details

C.1 Ablation of Learning Rate Schedules

Here, we compare the performances of using WSD and the commonly used learning rate (LR) cosine schedules (decaying to 10% of the peak LR value) [63]. The model size we use is 0.1B, with the following training configuration: batch size 512, 4,000 training iterations, and 200 warmup iterations. We can see from Figure 5 that both schedules yield similar (with WSD achieving slightly better final loss) performances. This justifies our choice of using the WSD schedule throughout the paper.

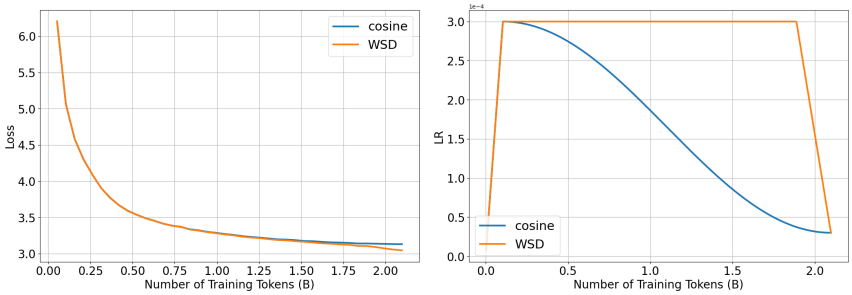

Figure 5: **WSD vs cosine LR schedule.** We show that the WSD LR schedule achieves similar (even slightly better) final loss compared to the more commonly used cosine LR schedule.

C.2 Ablation of Data Repetition

During model growth, we have used the same data as the base model for the second stage of training. This is because the base model is often trained with a large amount of data, and it is not always possible to obtain a large amount of additional data. However, this means that the second stage of training involves repeating the same data. To study the effect of data repetition, we have made an ablation study, where we compare the performance of using the same data as the base model and using different portion of the Slimpajama data. As shown in Figure 6, we find that the difference is small.

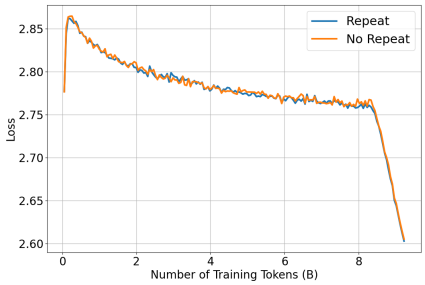

Figure 6: **Ablation of data repetition during model growth.** We compare the performance of using the same data as the base model and using different data for the second stage of training. The difference is small, indicating that data repetition does not affect the scaling behavior significantly.

C.3 Evaluation with Standard Benchmarks

We compare the performance of our trained 1B base model against existing models with similar sizes, Pythia [9] and TinyLlama [73], based on standard natural language processing benchmarks, ARC [14], lambada [46], logiqa [42], piqa [10], sciq [68], and winogrande [52].

Table 6 shows the results. We see that the model perform similarly to the open models, indicating that our models have been trained correctly.

C.4 Continual Pretraining Details

Using a lower LR for CPT is also a common practice [34]. As an ablation, we have experimented with using a constant LR schedule setting the value to be the final LR for the first-stage base model training, on a 0.1B model. We find that it yields worse performance (see Figure 7). Hence, we use the same LR (and LR schedule) as the base model for CPT.

C.5 Model Growth Details

For completeness, we provide more details on the model growth methods used in this work.

Table 6: **Benchmarks' performance comparison across models.** Reported scores are accuracies (normalized by byte length whenever applicable). The first two columns are scores of existing models. The last column is evaluation results of the largest base model trained with the most number of tokens in this work. Our models are evaluated with the LM Evaluation Harness v0.4.0 library [22].

Models	**Pythia-1B**	**TinyLlama-1.1B**	**Our 1B model**
Datasets	Pile	Slimpajama & Starcoder	Slimpajama
Tokens	100B	103B	74B
ARC-c	25.59	24.32	27.65
ARC-e	47.26	44.91	52.10
lambada	53.52	-	45.08
logiqa	29.49	-	26.11
piqa	69.31	67.30	65.89
sciq	77.3	-	78.10
winogrande	51.22	53.28	54.93
Avg.	50.53	-	49.98

Width expansion details. Let us elaborate more on the width expansion method used in this work. We first give a more precise defintion of function preservation. Let F be a function and G as the growth operator. The function preservation condition is defined as:

$$F(x) = G(F)(x), \forall x \in \mathcal{X}$$

where $\mathcal{X}$ is the input space.

When performing width expansion, the neuron values of each layer are expanded by duplicating the weights of existing neurons, and dividing the output weights by the growth factor.

Stacking details. We first note that stacking does not preserve function but is empirically found to work well in practice [17]. We use the recommended stacking procedure in [17], which is, letting M be the non-embedding part of the base model, the stacked model with growth factor k is given by:

$$M' = M \circ M \circ ... \circ M \quad (k \text{ times})$$

where $\circ$ denotes function composition. The embedding and final layer are then simply copied from the base model.

Low LR. We also experiment with using a lower constant LR for stacking as above, and find that it also yields worse performance (see Figure 7). Hence, we use the same LR (and LR schedule) as the base model for stacking as well.

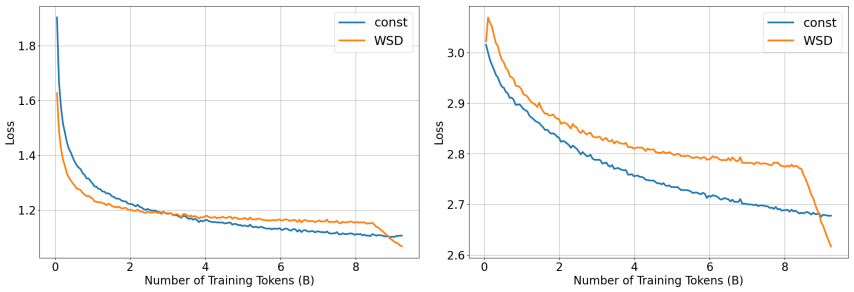

Figure 7: **Bootstrapped pretraining with lower LR.** We show that using the same LR as the base model achieves better final performance than using a lower LR for a 0.1B model continual pretrained on code data (left) and stacking (right).

D Scaling Laws Details

D.1 Fitted Exponents and Other Results

We show the fitted exponents of the multiplicative scaling laws studied in Table 1 and 2 in Table 7.

We further show that D_1 has power-law scaling in Figure 8, for CPT on mathematics data and model growth by expansion, which justifies the multiplicative form of the scaling laws.

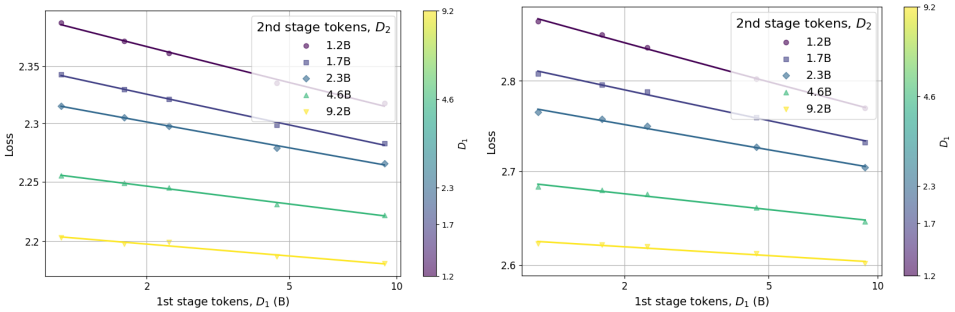

Figure 8: **Left: D_1 has power-law scaling.** We show scaling behavior of fist-stage training tokens (D_1) for different values of second-stage tokens (D_2), indicating that D_1 also has power-law scaling. From left to right: Continual pretraining on mathematics data, and model growth by expansion.

Table 7: **Fitted exponents for the multiplicative scaling law.** Corresponding $\alpha_1, \alpha_2, \alpha_3$ values for the multiplicative interaction fits shown in Table 7.

Variant	α_1	α_2	α_3
CPT (code)	0.106	0.146	0.004
CPT (math)	0.981	0.388	0.017
Expand	0.549	0.852	0.024
Stack	0.515	0.350	0.017
CPT (replay)	0.424	0.626	0.018
CPT (stable)	0.920	0.156	0.004
Stack (x4)	0.644	0.829	0.028
Stack (stable)	0.891	0.507	0.009

D.2 Joint Scaling Law

For model growth, we also fit the joint scaling law in Equation 7. Note that there is ambiguity in defining the model size N in the context of model growth. We consider N to be the size of the model before growth, and as we keep the growth factor fixed to be 2 in our experiments, the new model size after growth is $N' = 2N$. Therefore, N and N' differ by a constant factor of 2, which can be absorbed into the coefficient B in Equation 7. Unless stated otherwise, we use N in the fitting of the joint scaling law for model growth. In Figure 9, we show the fit of the joint scaling law for stacking.

We provide the fitted coefficients of the joint scaling law in Table 8. In addition to bootstrapped pretraining, we also show the fitted coefficients for base models trained from scratch on the same dataset as the second stage (code data for CPT, and the same data as the base model for model growth). For model growth, we consider a growth factor of 2, and fit the parameters with model size N before stacking for comparison conveniences. We further note that the fitted coefficients are produced by fitting the joint scaling law to *all* data points collected, including those used for validation in Figure 9.

D.3 Why Fitting Scaling Laws Separately?

We justify our decision to fit two separate scaling laws—one for models trained from scratch and another for bootstrapped pretraining—instead of employing a unified formulation that spans all stages.

First, the scaling behavior of models trained from scratch is expected to differ from that of models undergoing bootstrapping. Specifically, at the limit $D_1 \to 0$, grown models are initialized from a pretrained base model, whereas scratch-trained models start from random weights. This difference in initialization leads to distinct learning dynamics and, consequently, different scaling law parameters.

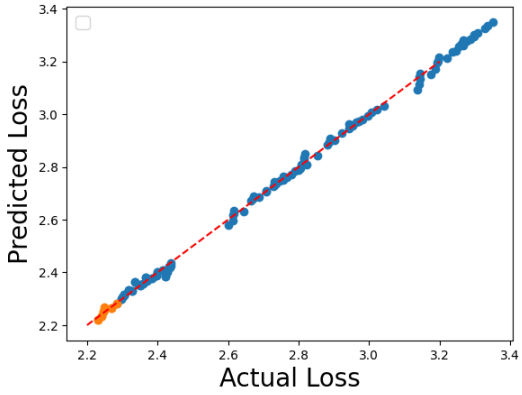

Figure 9: **Fits of the joint upcycling scaling law of stacking.** 10% of the collected data points with lowest losses are used for validation (orange points).

Table 8: **Fitted coefficients for joint scaling laws.** Note that for stacked models, we fit the coefficients with model size N before stacking for comparison conveniences.

	α/α_1	α_2	α_3	β	A	B	E
Base (Slimpajama)	0.092	-	-	0.105	10.383	10.085	0.041
From-scratch (code)	0.113	-	-	0.234	8.143	27.286	0.105
CPT (code)	0.048	0.126	0.001	0.238	15.062	27.234	0.105
Stack	0.087	0.119	0.003	0.173	33.394	22.471	0.041

Similarly, the scaling behavior of the base model differs from that of the second-stage training in the limit $D_2 \to 0$. As shown in Figure 7, second-stage training often begins with a *rewarming* phase, during which the loss initially increases before decreasing. This early instability deviates from the expected scaling of dense models, although the overall trend remains correlated—supporting the validity of Condition 1, due to function preservation and empirical observations that rewarming does not entirely disrupt loss behavior (as quoted from [15]: "In practice, few empirical phenomena obey power laws for all values of x").

Finally, our preliminary experiments indicate that a unified scaling law does not provide a satisfactory fit across both stages. We therefore opt to model them separately. We also note that there may exist alternative functional forms that could better capture the full range of behavior, but they may violate some of the well-established conditions for scaling laws (power law), and may overcomplicate the analysis or overfit the data; hence, we leave their exploration to future work.

E More Practical Implications

To further examine the trade-off between training from scratch and bootstrapped pretraining, we fit Chinchilla-style scaling laws to models trained *from scratch* on the same datasets used in our second-stage experiments, namely, code or math data for CPT, and the original dataset for model growth. In Figure 10 and 11, we plot the validation loss as a function of training tokens for both approaches, across a range of sunk costs, using a fixed model size of 7B. For CPT on code data, we find that models trained from scratch begin to outperform bootstrapped models (pretrained on 10T tokens) once more than 300B tokens of domain-specific data are used.

E.1 Compute Optimality of Model Growth

For model growth with growth factor 2, we want to scale N_2, D_2 optimally, $N_2^{\text{opt}}, D_2^{\text{opt}}$, given a FLOPs budget and fixing D_1, while minimizing the loss L, which we write as $L_{D_1}(D_2, N_1)$. Here,

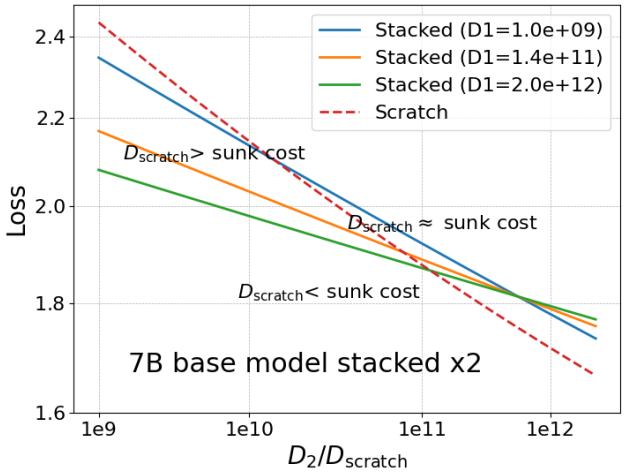

Figure 10: **Tokens required for from-scratch training to catch up with bootstrapped pretraining.** We compare loss-versus-token plots of from-scratch and bootstrapped pretraining at various base model training budgets (sunk costs). Stacking is only considered efficient when $D_{\text{scratch}} <$ sunk cost. We observe that the efficiency of stacking diminishes with sunk cost.

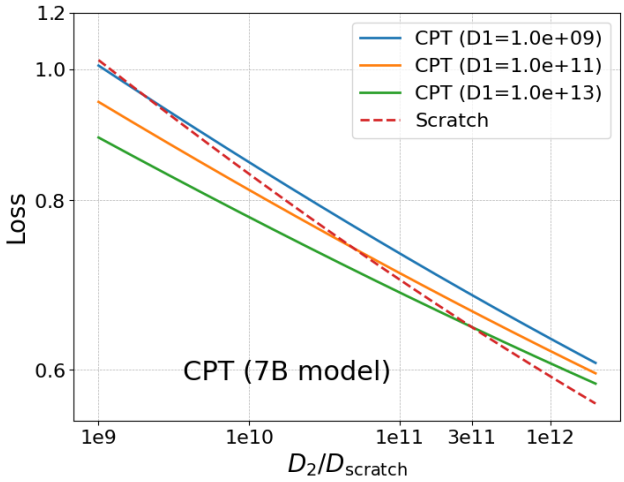

Figure 11: **Tokens required for from-scratch training to catch up with bootstrapped pretraining.** Conclusion similar to those made in Figure 10 can be made for CPT on code data, where the scaling efficiency decreases with first-stage training tokens.

N_1 is the model size before growth, and $N_2 = 2N_1$. This is equivalent to solving the following:

$$\frac{\partial}{\partial D_2} L_{D_1}(D_2, C_2/12D_2)\Big|_{D_2=D_2^{\text{opt}}} = 0,$$
$$\frac{\partial}{\partial N_1} L_{D_1}(C_2/12N_1, N_1)\Big|_{N_1=N_1^{\text{opt}}} = 0$$

where we have used $N_2 = 2N_1$ and $C_2 = 6N_2D_2$. Solving the above equations leads to

$$D_2^{\text{opt}} = G\left(\frac{C_2}{12}\right)^a,$$
$$N_1^{\text{opt}} = G^{-1}\left(\frac{C_2}{12}\right)^b$$

where

$$
\begin{aligned}
G &\coloneqq \left(\frac{A_{\text{eff}}\alpha_{\text{eff}}}{B\beta}\right)^{1/(\alpha_{\text{eff}}+\beta)} \\
a &\coloneqq \frac{\beta}{\alpha_{\text{eff}}+\beta} \\
b &\coloneqq \frac{\alpha_{\text{eff}}}{\alpha_{\text{eff}}+\beta} \\
A_{\text{eff}} &\coloneqq AD_1^{-\alpha_1} \\
\alpha_{\text{eff}} &\coloneqq \alpha_2 - \alpha_3 \log D_1
\end{aligned}
$$

We can henceforth relate D_2^{opt} and N_1^{opt} via

$$D_2^{\text{opt}} = G\left(GN_1^{\text{opt}}\right)^{a/b} \propto \left(N_1^{\text{opt}}\right)^{\beta/\alpha_2 - \alpha_3 \log D_1}$$

and

$$N_1^{\text{opt}} = G^{-1}\left(G^{-1}D_2^{\text{opt}}\right)^{b/a} \propto \left(D_2^{\text{opt}}\right)^{(\alpha_2 - \alpha_3 \log D_1)/\beta}$$