Scaling Laws for Code: A More Data-Hungry Regime

Anonymous ACL submission

Abstract

Code Large Language Models (LLMs) are revolutionizing software engineering. However, scaling laws that guide the efficient training are predominantly analyzed on Natural Language (NL). Given the fundamental differences like strict syntax between code and NL, it is unclear whether these laws are directly applicable to code. To address this gap, we conduct the first large-scale empirical study of scaling laws for code, comprising 117 experimental runs with model sizes from 0.2B to 3.8B and training tokens from 2B to 128B. We fit the Chinchilla law and the Farsser law. First, the results show that the more expressive Farseer law offers greater accuracy. Second, the analysis reveals that Code LLMs scale effectively with model size. Crucially, code represents a more data-hungry regime, requiring a substantially higher data-to-parameter ratio than NL. Finally, two additional sets of experiments on code-NL mixtures show that NL benefits resource-constrained scenarios, but becomes a detriment at higher compute budgets.

1 Introduction

Code Large Language Models (LLMs) trained on large-scale code corpora have achieved remarkable code-related capabilities (Chen et al., 2021; Li et al., 2022; Roziere et al., 2023; Li et al., 2023; Huang et al., 2025; OpenAI et al., 2025; Hui et al., 2024). Built upon these LLMs, applications such as OS, GUI, and Terminal Agents significantly enhance developer productivity and substantially impact the field (Hu et al., 2025; Tang et al., 2025; Team, 2025). The fuel for this technological revolution is the continuous growth of data and model size, which also incurs substantial computational costs (Kaplan et al., 2020; Hoffmann et al., 2022). Training frontier LLMs requires thousands of petaflop/s-days and millions of dollars, making it impractical to conduct ablation experiments on the largest models, whether on the structure, data, or training strategies (Brown et al., 2020; DeepSeek-AI et al., 2024; Yang et al., 2025).

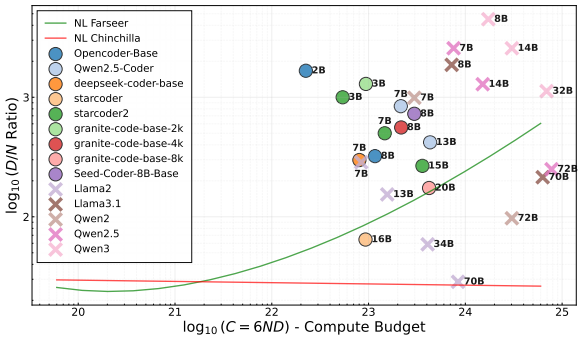

Figure 1: The relationship between compute budget (C) and the optimal data-to-parameter ratio (D/N) for various LLMs. The Farseer and Chinchilla curves are derived from natural language data. Code LLMs is represented as circles while general LLMs is cross.

Scaling laws provide a theoretical foundation to address this challenge (Kaplan et al., 2020; Hoffmann et al., 2022). By using results from smaller models to fit empirical formulas, scaling laws describe the relationship between model performance (typically the validation loss, L) and factors like model size (N), dataset size (D), and compute (C). They provide important references of resource allocation on the data and model for training general-purpose natural language (NL) LLMs (Brown et al., 2020; Team, Kimi et al., 2025). However, code, as a highly structured data type, has statistical properties that fundamentally differ from NL: it has strict syntax, complex long-range dependencies, and unique vocabulary distributions (Allamanis et al., 2018). This raises the question: *Can scaling laws developed for NL be applied to code?*

Exploring this open question is particularly important. As shown in Figure 1, Code LLMs tend to be concentrated in smaller sizes, with most prominent models being 32B or smaller. *Does model scaling for Code LLMs plateau more quickly?* Futher-

more, Code LLMs often require a larger data-to-parameter (D/N) ratio, which Figure 1 shows deviates significantly from the predictions of NL scaling laws. *Does this imply that Code LLMs fundamentally require more code data?* Answering these questions is crucial for understanding and guiding the future development of code pretraining.

To address these questions, we conduct the first systematic explorations of scaling laws specifically for code. Our study comprises 117 experiments with N ranging from 0.2B to 3.8B and D from 2B to 128B tokens. These models are trained on a curated public code corpus, using a consistent architecture and optimized hyperparameters for each run, and their losses are evaluated on a high-quality, held-out validation set. By fitting these results to both the Chinchilla and Farseer formulations, we validate that the more expressive **Farseer law accurately models code's scaling behavior**, achieving a significantly better fit on experiment runs and higher predictive precision on validation points. The fitted Farseer surface reveals a key dynamic: **contrary to initial expectations, Code LLMs scale robustly with** N. Further analysis of the compute-optimal D/N ratio reveals that **code is significantly more "data-hungry" than NL**. For a given N, a Code LLM requires substantially more data to reach its optimal performance. We attribute this to the inherent repetitiveness of code. Massive token volumes are needed to introduce new applications. This insight provides a compelling explanation for the smaller size of typical code LLMs: it is likely a consequence of the scarcity of high-quality code data and practical low-latency inference requirements like code completion, rather than a lack of benefit from scaling model parameters.

Given the relative abundance of NL data and some practice of training Code LLMs on code-NL mixtures (Lozhkov et al., 2024; Li et al., 2023), we also investigate the potential to augment code model training with NL. We conduct a total of 234 additional experiments, comprising a full 117-run sweep for each of the two mixture ratios (70%/30% and 30%/70% code-NL). The results indicate that when the volume of code data is limited, incorporating a moderate amount of NL data can indeed enhance performance on code. However, this benefit diminishes and eventually becomes a detriment as the proportion of NL data increases or as more pure code data becomes available.

Our contributions are summarized as follows:

- To our knowledge, we conduct the first scaling law for code, demonstrating that its loss is highly predictable and can be accurately modeled by an existing law formulation.
- We find that Code LLMs exhibit good scaling properties with respect to N. The compute-optimal D/N ratio for code is significantly larger than for NL.
- We reveal a reversal of effects for code-NL mixtures: NL provides a performance boost in low-compute settings but degrades performance at larger scales, providing clear guidance on when to use mixed-data strategies.

2 Methodology

2.1 Background and Motivation

Scaling laws are empirical formulas that guide the efficient training of LLMs by describing the relationship between L and key factors like N, D, and C. Foundational work focused primarily on NL, such as the Chinchilla law (Hoffmann et al., 2022), proposed a simple power-law relationship:

$$L(N, D) = E + \frac{A}{N^a} + \frac{B}{D^b} \tag{1}$$

. While powerful, its constant exponents (a, b) mean N and D are scaled proportionally, failing to capture the nuanced interplay where larger models learn more efficiently from data. To address this, a recent refinement, the Farseer law (Li et al., 2025b) has introduced more expressive formulations, where the scaling exponents are themselves functions of N, such as:

$$L(N, D) = e^{s \cdot N^q + S} + e^{B \cdot N^b + Q} \cdot D^{-e^{A \cdot N^a + E}} \tag{2}$$

. Possessing such a law enables the calculation of the compute-optimal D/N ratio for any fixed C, thereby guiding efficient resource allocation.

However, while these laws have proven effective for NL, their direct applicability to code is questionable. Code possesses fundamentally different statistical properties, including strict syntax and complex long-range dependencies (Allamanis et al., 2018). Beyond these theoretical distinctions, empirical observations suggest that NL-derived laws are a poor fit. As illustrated in Figure 1, the D/N ratios of prominent code LLMs deviate significantly from the optimal values predicted by NL scaling laws. This discrepancy highlights the need for a

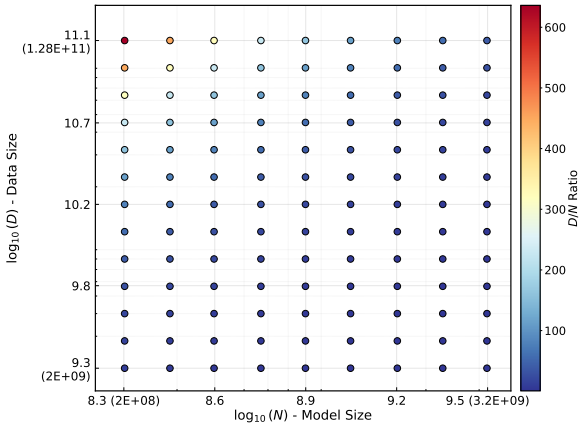

Figure 2: A scatter plot of our 117 experimental points (colored by D/N ratio), showing model parameters (N) versus training tokens (D).

dedicated investigation to establish a scaling law tailored specifically for code, providing more accurate guidance for training future Code LLMs.

2.2 Experimental Design

Our experimental design is a budget-conscious adaptation of the comprehensive methodology from Farseer (Li et al., 2025b). Given that a full replication is computationally prohibitive, our design focuses on a strategic selection of configurations that are optimized to balance deep informational gain with a feasible computational cost.

Sampling Strategy. Given the significant computational cost of training, we carefully curate a set of 117 configurations by sampling log-uniformly across N and D. We constrain N to a range of 0.2B to 3.8B. The lower bound is chosen because models smaller than 0.2B contribute minimally to fitting, while the upper bound considers the current size distribution of Code LLMs. Furthermore, we prune extreme D/N ratios, including both very high and very low, from the original Farseer design. This retains D in a range of 2B to 128B. Figure 2 visualizes the distribution across N, D, and D/N ratio of the final 117 experimental points.

Model Architecture and Parameterization. To strictly isolate the effects of scale, all 117 models in our study share a consistent decoder-only Transformer architecture. This architecture incorporates modern, effective components for large-scale training, including SwiGLU activations, Rotary Position Embeddings, and RMSNorm (Shazeer, 2020; Su et al., 2021; Zhang and Sennrich, 2019). To determine the specific hyperparameters such as model dimension, number of attention heads, and layer count for each model, we employ the deterministic procedure from Farseer (Li et al., 2025b). This ensures that all models are well-proportioned by maintaining near-optimal architectural aspect ratios, making them both structurally efficient and directly comparable. A complete table of all model configurations is available in Appendix A.

Training Hyperparameters. To ensure that each model is trained efficiently without a cost-prohibitive hyperparameter search, we follow StepLaw's optimal hyperparameter scaling rules to set near-optimal learning rates, global batch sizes (GBZ) from N and D (Li et al., 2025a). Crucially, StepLaw validates its scaling rules specifically on a code pretraining recipe, confirming the applicability to our study. We use the AdamW optimizer (Loshchilov and Hutter, 2019) with a cosine learning rate decay schedule for all experiments.

Train Set and Validation Set. We utilize the high-quality code pre-training data from Opencoder (Huang et al., 2025), which is sourced from public GitHub repositories and has undergone meticulous deduplication and filtering. To create a more balanced programming language distribution, we downsample the initial 1216.93B token corpus, resulting in our final 895.51B token train set. For evaluation, our validation set is a 6.3M token internal codebase from Xuyang et al. (2025). This dataset consists of production-grade code used in real-world applications, ensuring its practical relevance, and has been rigorously checked to guarantee no overlap with the train set. Full details about the two set are provided in Appendix B.

Computational Setup. All experiments are conducted on a cluster of NVIDIA H100-80GB GPUs. Once the GBZ is determined, we configure the number of GPUs and the per-GPU micro batch size by managing the trade-off between minimizing runtime (which favors more GPUs) and maximizing Model FLOPs Utilization (MFU, which favors fewer, more saturated GPUs). The final configurations for each run utilized between 8 and 128 GPUs. The total compute expended is approximately 13,600 H100 GPU-days. Further scheduling details are in Appendix C.

3 Experiment Results

3.1 Scaling Behavior

After training, the validation loss is computed for each of the 117 models. As a prerequisite for fitting a scaling law, we first qualitatively verify that

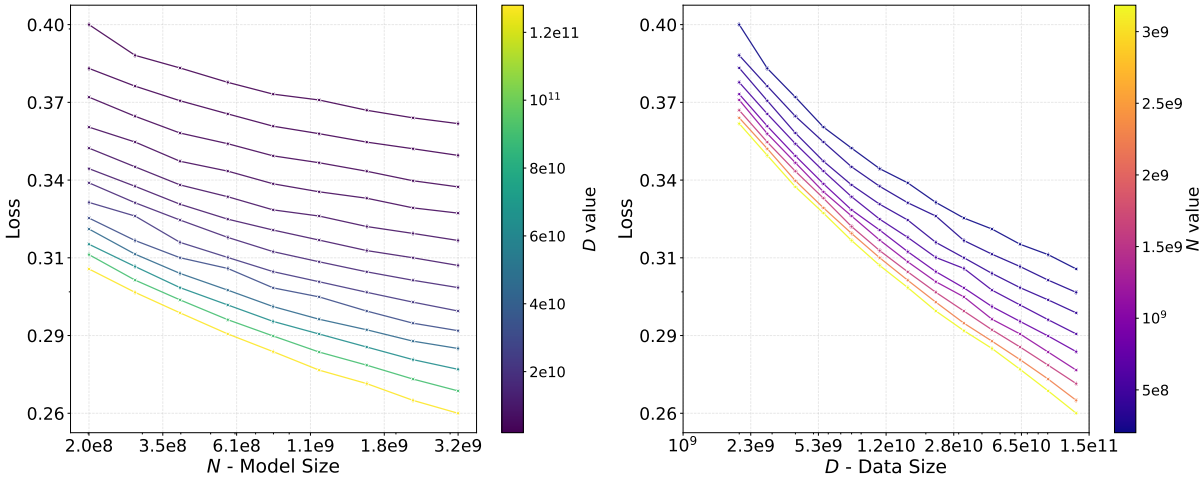

Figure 3: Scaling behavior of loss across different model sizes (N) and data sizes (D). (a) Loss decreases with increasing N for fixed D values. (b) Loss decreases with increasing D for fixed N values. The color gradients represent different D values (left) and N values (right). Both plots use double-logarithmic axes.

D/N	N (B)	D (B)	GBZ	GPUs	MBZ	PL_F	PL_C	Loss	RE_F(‰)	RE_C(‰)
20	6.37	127	640	160	2	0.259271	0.265707	0.256833	**9.49**	34.55
150	2.27	341	1080	120	9	0.253488	0.262330	0.253786	**1.17**	33.67
424	1.34	567	1456	112	13	0.255846	0.262939	0.258546	**10.44**	16.99

Table 1: Validation results for three model configurations trained at a large compute budget. **GPUs** indicates the number of GPUs used for the run. **GBZ** and **MBZ** refer to the Global and Micro Batch Size. $\mathbf{PL_F}$ and $\mathbf{RE_F}$ denote the predicted loss and relative error from Farseer, respectively, with $\mathbf{PL_C}$ and $\mathbf{RE_C}$ representing the counterparts for Chinchilla. **Loss** is the empirically measured validation loss.

the model performance scales predictably with N and D. Figure 3 provides a comprehensive visualization of this behavior. Figure 3(a) shows that for a fixed D, the validation loss monotonically decreases as N increases, a trend that holds consistently across all tested D. Symmetrically, Figure 3(b) demonstrates that for any given N, the loss also smoothly declines as D grows. The near-linear trends on these log-log plots indicate approximate power-law relationships between loss and N/D, consistent with scaling-law assumptions. These clear and consistent trends suggest that performance on code exhibits stable scaling behavior, thus providing an empirical foundation for our subsequent quantitative analysis.

3.2 Scaling Law Fitting

To quantitatively model the observed scaling behavior, we fit our 117 experimental data points to the Chinchilla and the Farseer, yielding the specific formulations shown in Equation 3 and Equation 4:

$$L(N, D) = 0.2193 + \frac{534.374}{N^{0.4853}} + \frac{76.0743}{D^{0.2983}} \quad (3)$$

$$\begin{aligned} L(N,D) = &\exp(-0.0047 \cdot N^{0.239} - 0.8188) \\ &+ \exp(62.8936 \cdot N^{-0.0614} - 14.0414) \\ &\cdot D^{-\exp(-0.0209 \cdot N^{0.1943} - 0.1826)} \end{aligned} \quad (4)$$

The Farseer formulation provides a better fit to our data, exhibiting a lower mean relative error than the Chinchilla model (0.82‰ vs. 1.03‰). This more accurate law also makes a fundamentally different asymptotic prediction: an irreducible loss of zero, in contrast to the 0.2 limit from Chinchilla, suggesting there may be no theoretical performance ceiling for scaling (Details in Appendix D). However, the ultimate test of a scaling law includes its power to extrapolate and predict performance in new, large-scale scenarios, which motivated our dedicated validation experiment.

3.3 Scaling Law Validation

Within available computational resources, we design a rigorous validation experiment at a fixed large compute budget of $C = 5.36 \times 10^{21}$FLOPs.

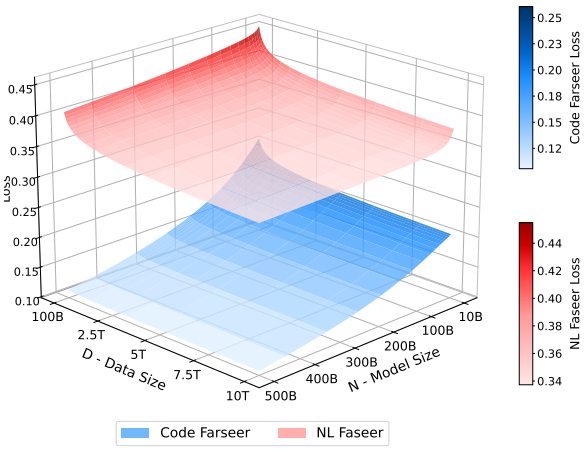

Figure 4: 3D visualization of the fitted Farseer scaling-law surfaces for Code (blue) and Natural Language (orange) over (N, D).

Three distinct D/N ratios are selected: (1) D/N=20, the established optimum for NL Chinchilla, to test the direct applicability of NL-derived heuristics; (2) D/N=150, the predicted optimum for code from Farseer (Equation 4), it is also really close to Chinchilla law's prediction thus run once for efficiency; and (3) D/N=424, an extreme data-heavy as a upper-bound point to evaluate the extrapolation capabilities of equations.

Table 1 presents the predictions from both equations alongside the empirical results and offers two primary findings. First, optimal training derived from NL is suboptimal for code. This is evidenced by the empirical losses, where the model trained at $D/N = 150$ achieves a significantly lower loss than the one at the NL-optimal ratio of $D/N = 20$. However, this does not imply an infinite appetite for data, as the loss at $D/N = 424$ is higher again. Second, Farseer demonstrates better predictive accuracy. It outperforms Chinchilla across all three validation points, achieving a remarkably low relative error of just 1.17‰at the empirically confirmed optimum. The prediction errors at the extreme D/N ratios, while still low at approximately 1%, are relatively higher. This is likely because such settings are less common in practice and thus are less represented in our initial 117 fitting points.

In summary, our experiments demonstrate that while NL training recipes are suboptimal for code, code performance is highly predictable under the Farseer formulation. We emphasize that our primary goal is to identify the most suitable *form* of the scaling law for code, rather than providing universal parameters, as the specific values are inherently tied to the experimental setup and may vary subtly across different implementations

4 Analysis

Having established in Section 3.3 that the Farseer law provides an accurate model for code scaling, we now use this formulation to analyze the underlying dynamics and derive key insights.

4.1 Scaling Surface

Figure 4 shows the fitted Farseer surfaces for Code vs. NL over (N, D). The visual inspection reveals some fundamental differences.

Code Has Lower Intrinsic Loss. The validation losses for code and NL models are computed on their own validation sets. In our fitted surfaces, code loss is generally lower than NL across a broad range of (N, D). This suggests a difference in intrinsic entropy between the two modalities. Although code can be logically complex, its statistical regularities, like strict syntax, standard templates, keywords, and common programming idioms, are often more prevalent than in NL (Zhu et al., 2024b). This view aligns with evidence from speculative decoding, where code tokens are easier to predict than NL tokens, enabling higher acceleration (Leviathan et al., 2023). Differences across programming languages also appear smaller than across NL, making code tokens easier to learn and predict (Roziere et al., 2023; Hasan et al., 2025).

Code Exhibits More Sustained Scalability. Although code has a lower absolute loss, its surface has still sustained steepness. As visualized in Figure 4, while the NL surface begins to visibly flatten at larger scales, the code surface maintains a more consistent and steep downward gradient across the entire range. This indicates that, within our experimental scope, code models are not approaching a saturation point. Instead, they continue to derive significant performance benefits from further increases in either N or D, highlighting the vast potential for scaling even larger Code LLMs.

Code Has Different Scaling Priorities. Contrary to our initial hypothesis that code would plateau faster along N, we observe excellent scaling along the N axis: gains from increasing N are larger for code than NL. Whereas for D, achieving a comparable reduction in loss requires a substantially larger volume of additional data. This asymmetry may be attributed to the fundamental structure of code. Code is governed by a finite set

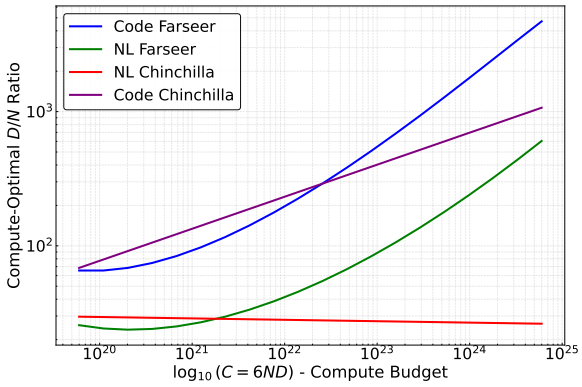

Figure 5: The compute-optimal D/N ratio vs. compute budget C.

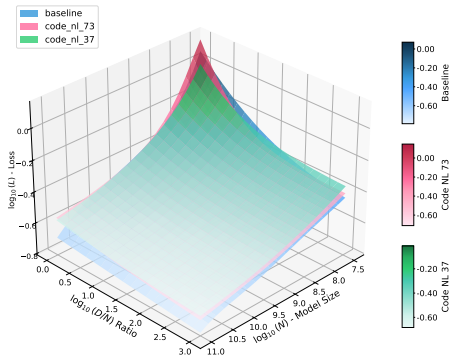

Figure 6: 3D visualization comparing scaling surfaces for different data mixtures: baseline (100% code), `code_nl_73` (70% code + 30% NL), and `code_nl_37` (30% code + 70% NL). D/N ratio only considers the number of code tokens.

of syntactic rules and structural patterns (Winskel, 1993). On one hand, a bigger model can better understand how basic programming constructs combine to form complex algorithms (Shi et al., 2024; Liu et al., 2024). On the other hand, because these underlying rules are repetitive across the corpus, a vast number of new tokens is needed to encounter genuinely novel application scenarios, leading to less efficient D-scaling (Casalnuovo et al., 2019).

4.2 Optimal D/N Ratio

Based on the scaling landscapes, we derive the optimal D/N ratio as a function of C in Figure 5.

Code LLMs Tend to Be More Data-Hungry. Across the compute budgets we studied, the optimal D/N ratio for code is consistently higher than for NL, and this gap widens with scale. We attribute this to the lower average information density of large-scale, conventional code corpora, such as GitHub data used in our training. Due to the inherent repetitiveness, the model masters common patterns in the early stages of training. While further convergence can be achieved by stacking more data by expanding to cover more scenarios, the process is highly costly. Therefore, we posit that a more effective path to improvement lies in enhancing data quality, for example, through targeted upsampling of high-complexity code, rather than just blindly increasing data quantity.

The Optimal D/N Ratio Grows Exponentially with C. The optimal D/N ratio for code is not static but grows progressively with C. While NL Chinchilla proposes a near-constant D/N ratio, Figure 5 reveals that even the Chinchilla law fit on code data exhibits a clear upward trend. This growth is linear because Chinchilla assumes that N does not affect the dynamics of D. However, Farseer better captures the scaling dynamics and posits that a larger N accelerates learning from D, thus demanding more data for convergence. This results in a super-linear growth of the optimal D/N ratio, confirming that the demand for data accelerates at higher compute scales.

D/N May Help Explain Smaller SOTA Code LLMs. This high data requirement helps explain why mainstream code LLMs are generally smaller than their NL counterparts. The primary bottleneck appears to be the lack of sufficient data to optimally train a massive model, rather than diminishing returns from scaling the model size itself. In practice, model size is further constrained by the limited availability of code data and by factors like inference cost, especially for low-latency applications like code completion. Nevertheless, larger Code LLMs can still deliver significant performance benefits when sufficient data and compute are available like Qwen3-Coder (Yang et al., 2025).

4.3 Data Mixing

Section 4.2 reveals that Code LLMs are significantly more data-hungry, a practical challenge given the relative scarcity of high-quality code data. Furthermore, while it is common industry practice to train on proprietary code-NL mixtures (Li et al., 2023; Lozhkov et al., 2024), the impact of these ratios on scaling behavior is largely unexplored. These considerations motivate our investigation into whether NL corpora can be strategically leveraged to improve code model performance.

To this end, our computational budget allows for two additional full experimental sweeps, for

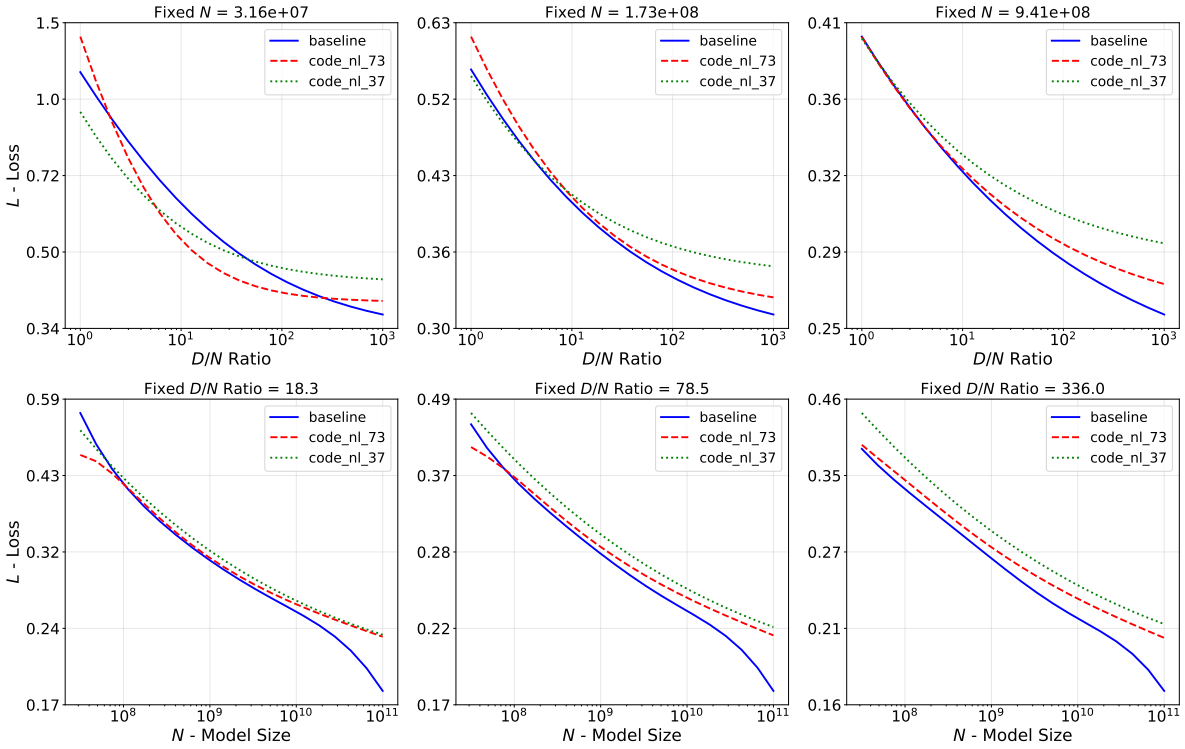

Figure 7: 2D slices of three scaling law surfaces. Top Row: Loss as a function of the D/N ratio for three fixed N. Bottom Row: Loss as a function of N for three fixed D/N ratios. D/N ratio only considers the number of code tokens and does not include the NL data.

which we select two distant points on the mixture spectrum to maximize the observable differences: (1) **70% Code + 30% NL**: A code-dominant mixture, denoted as `code_nl_73`. (2) **30% Code + 70% NL**: An NL-dominant mixture, denoted as `code_nl_37`. For each mixture, we replicate our full set of 117 experimental runs, covering the same range of N and D as the baseline. This rigorous approach allows us to fit a new, distinct Farseer law for each condition, enabling a direct, comprehensive and fair comparison of their scaling behaviors. To ensure this comparison is practical, it is crucial to note that Figure 6 and Figure 7 define D and the D/N ratio using only the number of code tokens, not the total token count of mixtures.

Figure 6 compares the fitted Farseer surfaces for baseline and two data mixtures, revealing a scale-dependent trade-off. In the low-compute regime (i.e., at smaller N and D/N ratios), the surfaces have a crossover phenomenon. Particularly, the `code_nl_73` can achieve a lower loss than the pure code baseline. However, as the scale of computing increases, the baseline quickly becomes and remains superior. Furthermore, the baseline surface maintains a steeper and more consistent downward gradient. In contrast, the surfaces for the mixtures begin to show signs of saturation, with their rate of loss reduction diminishing more quickly.

To further dissect the scaling dynamics of the data mixtures, we plot 2D slices of the 3D surfaces in Figure 7. These slices allow for a more granular analysis of performance under controlled conditions, yielding several key insights. The top panels show that the benefit of NL data is most pronounced for smaller models. The top-left panel (Fixed $N = 31.6$M) provides clear evidence of the crossover phenomenon suggested by the 3D surfaces. At low D/N ratios (e.g., < 40), both the `code_nl_73` and `code_nl_37` achieve lower loss than the baseline. This empirically validates that when code data is limited, a moderate amount of NL data can indeed enhance a smaller model's performance on code tasks. We hypothesize this is because the limited in-domain data is insufficient to learn robust representations, and the diverse NL data acts as a powerful regularizer and provides transferable world knowledge (Chen et al., 2021). The `code_nl_73` is particularly effective, maintaining its advantage until the D/N ratio approaches 300. The `code_nl_37`, while optimal in a very small and impractical $D/N < 5$ regime, quickly falls behind. However, as N increases to 173M

and further to 0.94B (middle and right panels), the model's capacity to learn from pure code improves, and the mixtures' advantages almost completely vanish. At $N = 0.94$B, the baseline is superior across all D/N ratios. For these larger models, which are high-capacity, the potential benefit of knowledge transfer from NL may be outweighed by the cost of distributional shift. The introduction of out-of-domain data can act as a performance impediment from the outset.

This pattern is even more pronounced when observing scaling with N at fixed D/N ratios in the bottom row plots of Figure 7. At a low, data-scarce ratio of $D/N = 18.3$ (bottom-left panel), the `code_nl_73` mixture is again significantly stronger than the baseline for smaller models ($N < 0.1$B), reinforcing the hypothesis that NL data provides a crucial boost when in-domain data is insufficient. As the D/N ratio increases to 78.5 and then to 336, the in-domain data becomes more and more. The need for the NL supplement diminishes, and its role shifts from a helpful regularizer to an out-of-domain distraction. Consequently, at $D/N = 336$ (bottom-right panel), the three curves become stably stratified, with the baseline performing the best, followed by `code_nl_73`, and then `code_nl_37`, with no crossovers observed.

In summary, these 2D projections provide a clear prescription for the role of NL data in code pre-training. It acts as a valuable supplement in low-resource settings, where it likely provides a regularizing effect and transferable knowledge to smaller models trained on limited in-domain data. However, this benefit is a trade-off. As model capacity or the availability of in-domain data increases, the performance cost of the distributional shift outweighs the benefits, and pure, in-domain data becomes the unambiguous key to achieving optimal performance at scale.

5 Related Work

Neural Scaling Laws. Kaplan et al. (2020) first established the power-law relationship between L and factors like C, N, and D, sparking the industry's focus on aggressively scaling up model size. This trend is refined by Chinchilla (Hoffmann et al., 2022), which argued that many LLMs were significantly undertrained. They introduced the concept of compute-optimal scaling, proposing that N and D should be scaled proportionally. Subsequent research has expanded beyond these core variables to incorporate more fine-grained factors, including inference costs (Touvron et al., 2023; Su et al., 2024; Sardana et al., 2024), Mixture-of-Experts (MoE) architectures (Zhao et al., 2025; Ludziejewski et al., 2025; Tian et al., 2025), data quality and distribution (Shuai et al., 2024; Chen et al., 2025; Qin et al., 2025; Liu et al., 2025), and the specific downstream capabilities (Lin et al., 2024; Roberts et al., 2025). Most recently, Farseer (Li et al., 2025b) challenged the fixed optimal D/N ratio, demonstrating that it should increase with C and proposing a more expressive formulation to capture this dynamic.

Large Language Models for Code. Code LLMs have progressed rapidly alongside general-purpose LLMs. They are predominantly trained on corpora from GitHub, StackExchange, and other code-centric web sources. Early work continue-pretrain the NL LLMs to acquire code capabilities (Chen et al., 2021; Chowdhery et al., 2023; Anil et al., 2023; Zhu et al., 2024a; Team et al., 2024; Hui et al., 2024). Subsequently, stronger Code LLMs are trained from scratch on pure code or code-heavy mixtures (Li et al., 2022; Roziere et al., 2023; Nijkamp et al., 2023; Fried et al., 2023; Allal et al., 2023; Guo et al., 2024; Mishra et al., 2024; Seed et al., 2025; Huang et al., 2025).

Despite these advances, scaling laws for code remain poorly understood. Current code models are often trained on heuristic settings without a strong empirical basis. We first addresses this gap. We analyze models trained on pure code and controlled code-NL mixtures to establish a principled guide for the compute-optimal training of foundation models for code.

6 Conclusion

In this work, we conducted the first large-scale empirical study of scaling laws for code. We first find that the Farseer paradigm can accurately predict the scaling properties of code, which demonstrates that code adheres to existing scaling law frameworks. The analysis of scaling properties reveals that code scales robustly with model size, but exhibits a significant demand for data. This is further validated by the fact that the compute-optimal D/N ratio for code is substantially higher than for NL, and that this gap widens as the compute budget increases. Finally, code-NL mixtures experiments reveal that while NL data can be beneficial in low-resource scenarios, pure code data remains superior as model size increases or when in-domain data is abundant.

Limitations

While this study provides foundational insights into the scaling laws of code, it is subject to several limitations. First, due to computational constraints, our analysis is based on a curated set of 117 experimental points rather than the 400+ configurations used in the original Farseer study. Although our strategic selection covers the most critical regions, a larger set of fitting points could further refine the precision of our fitted law. Second, our validation experiments, while substantial, did not extend to the extreme scales of the largest industrial models, leaving the predictive accuracy at even higher orders of magnitude of compute as a subject for future work. Finally, our investigation into code-NL mixtures was focused on two distant points (70/30 and 30/70) to maximize observable differences. A more fine-grained study with additional ratios is needed to map the entire mixture spectrum and identify a more precise optimal blend.

Ethical Considerations

The data used is from Huang et al. (2025); Xuyang et al. (2025); Li et al. (2025b) and others are solely from publicly accessible project resources on reputable websites, ensuring that no sensitive information is included. We have taken care to acknowledge the original authors by properly citing their work.

References

Loubna Ben Allal, Raymond Li, Denis Kocetkov, Chenghao Mou, Christopher Akiki, Carlos Munoz Ferrandis, Niklas Muennighoff, Mayank Mishra, Alex Gu, Manan Dey, and 1 others. 2023. Santacoder: don't reach for the stars! *arXiv preprint arXiv:2301.03988*.

Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. 2018. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):1–37.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, and 1 others. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Casey Casalnuovo, Kenji Sagae, and Prem Devanbu. 2019. Studying the difference between natural and programming language corpora. *Empirical Software Engineering*, 24(4):1823–1868.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Zhengyu Chen, Siqi Wang, Teng Xiao, Yudong Wang, Shiqi Chen, Xunliang Cai, Junxian He, and Jingang Wang. 2025. Revisiting scaling laws for language models: The role of data quality and training strategies. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23881–23899.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and 1 others. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, and 1 others. 2024. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. 2023. Incoder: A generative model for code infilling and synthesis. In *The Eleventh International Conference on Learning Representations*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, and 1 others. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Md Mahade Hasan, Muhammad Waseem, Kai-Kristian Kemell, Jussi Rasku, Juha Ala-Rantala, and Pekka Abrahamsson. 2025. Assessing small language models for code generation: An empirical study with benchmarks. *arXiv preprint arXiv:2507.03160*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, and 3 others. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, and 1 others. 2025. Os agents: A survey on mllm-based agents for computer, phone and browser use. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7436–7465.

Siming Huang, Tianhao Cheng, Jason Klein Liu, Weidi Xu, Jiaran Hao, Liuyihan Song, Yang Xu, Jian Yang, Jiaheng Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Xianzhen Luo, Qiufeng Wang, Yuan-Tao Fan, Qingfu Zhu, Zhaoxiang Zhang, Yang Gao, Jie Fu, and 7 others. 2025. OpenCoder: The open cookbook for top-tier code large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33167–33193, Vienna, Austria. Association for Computational Linguistics.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*.

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

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. *arXiv preprint arXiv:2211.17192*.

Houyi Li, Wenzheng Zheng, Qiufeng Wang, Hanshan Zhang, Zili Wang, Shijie Xuyang, Yuantao Fan, Zhenyu Ding, Haoying Wang, Ning Ding, Shuigeng

Zhou, Xiangyu Zhang, and Daxin Jiang. 2025a. Predictable scale: Part I, step law – optimal hyperparameter scaling law in large language model pretraining. In *Proceedings of the Conference*.

Houyi Li and 1 others. 2025b. Predictable scale: Part II, Farseer: A refined scaling law in large language models. *arXiv preprint arXiv:2506.10972*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, and 48 others. 2023. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, and 7 others. 2022. Competition-level code generation with AlphaCode. *Science*, 378:1092–1097.

Jiayi Lin, Hande Dong, Yutao Xie, and Lei Zhang. 2024. Scaling laws behind code understanding model. *arXiv preprint arXiv:2402.12813*.

Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Zhe Zhao, Xi Lin, Xialiang Tong, Mingxuan Yuan, Zhichao Lu, Zhenkun Wang, and 1 others. 2024. A systematic survey on large language models for algorithm design. *arXiv preprint arXiv:2410.14716*.

Qian Liu, Xiaosen Zheng, Niklas Muennighoff, Guangtao Zeng, Longxu Dou, Tianyu Pang, Jing Jiang, and Min Lin. 2025. Regmix: Data mixture as regression for language model pre-training. In *The Thirteenth International Conference on Learning Representations*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Tianyang Liu, Misha Laskin, and 1 others. 2024. StarCoder2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

Jan Ludziejewski, Maciej Pióro, Jakub Krajewski, Maciej Stefaniak, Michał Krutul, Jan Małaśnicki, Marek Cygan, Piotr Sankowski, Kamil Adamczewski, Piotr Miłoś, and 1 others. 2025. Joint moe scaling laws: Mixture of experts can be memory efficient. *arXiv preprint arXiv:2502.05172*.

Mayank Mishra, Matt Stallone, Gaoyuan Zhang, Yikang Shen, Aditya Prasad, Adriana Meza Soria, Michele Merler, Parameswaran Selvam, Saptha Surendran, Shivdeep Singh, and 1 others. 2024. Granite code models: A family of open foundation models for code intelligence. *arXiv preprint arXiv:2405.04324*.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis. In *The Eleventh International Conference on Learning Representations*.

OpenAI, Ahmed El-Kishky, Alexander Wei, Andre Saraiva, Borys Minaiev, Daniel Selsam, David Dohan, Francis Song, Hunter Lightman, Ignasi Clavera, Jakub Pachocki, Jerry Tworek, Lorenz Kuhn, Lukasz Kaiser, Mark Chen, Max Schwarzer, Mostafa Rohaninejad, Nat McAleese, o3 contributors, and 6 others. 2025. Competitive programming with large reasoning models. *arXiv preprint arXiv:2502.06807*.

Zeyu Qin, Qingxiu Dong, Xingxing Zhang, Li Dong, Xiaolong Huang, Ziyi Yang, Mahmoud Khademi, Dongdong Zhang, Hany Hassan Awadalla, Yi R Fung, and 1 others. 2025. Scaling laws of synthetic data for language models. *arXiv preprint arXiv:2503.19551*.

Nicholas Roberts, Niladri S. Chatterji, Sharan Narang, Mike Lewis, and Dieuwke Hupkes. 2025. Compute optimal scaling of skills: Knowledge vs reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 13295–13316, Vienna, Austria. Association for Computational Linguistics.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, and 6 others. 2023. Code Llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. 2024. Beyond chinchilla-optimal: accounting for inference in language model scaling laws. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

ByteDance Seed, Yuyu Zhang, Jing Su, Yifan Sun, Chenguang Xi, Xia Xiao, Shen Zheng, Anxiang Zhang, Kaibo Liu, Daoguang Zan, and 1 others. 2025. Seed-coder: Let the code model curate data for itself. *arXiv preprint arXiv:2506.03524*.

Noam Shazeer. 2020. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Ben Shi, Michael Tang, Karthik R Narasimhan, and Shunyu Yao. 2024. Can language models solve olympiad programming? In *First Conference on Language Modeling*.

Xian Shuai, Yiding Wang, Yimeng Wu, Xin Jiang, and Xiaozhe Ren. 2024. Scaling law for language models training considering batch size. *arXiv preprint arXiv:2412.01505*.

Hui Su, Zhi Tian, Xiaoyu Shen, and Xunliang Cai. 2024. Unraveling the mystery of scaling laws: Part i. *arXiv preprint arXiv:2403.06563*.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*.

Fei Tang, Haolei Xu, Hang Zhang, Siqi Chen, Xingyu Wu, Yongliang Shen, Wenqi Zhang, Guiyang Hou, Zeqi Tan, Yuchen Yan, Kaitao Song, Jian Shao, Weiming Lu, Jun Xiao, and Yueting Zhuang. 2025. A survey on (m)llm-based gui agents. *Preprint*, arXiv:2504.13865.

CodeGemma Team, Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A Choquette-Choo, Jingyue Shen, Joe Kelley, and 1 others. 2024. Codegemma: Open code models based on gemma. *arXiv preprint arXiv:2406.11409*.

The Terminal-Bench Team. 2025. Terminal-bench: A benchmark for ai agents in terminal environments.

Team, Kimi, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, and 1 others. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

Changxin Tian, Kunlong Chen, Jia Liu, Ziqi Liu, Zhiqiang Zhang, and Jun Zhou. 2025. Towards greater leverage: Scaling laws for efficient mixture-of-experts language models. *arXiv preprint arXiv:2507.17702*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Glynn Winskel. 1993. *The formal semantics of programming languages: an introduction*. MIT press.

Shijie Xuyang, Xianzhen Luo, Tianhao Cheng, Zheng Chu, Houyi Li, ziqi wang, Siming Huang, Qingfu Zhu, Qiufeng Wang, Xiangyu Zhang, Shuigeng Zhou, and Wanxiang Che. 2025. Is compression really linear with code intelligence? *Preprint*, arXiv:2505.11441.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32.

Guoliang Zhao, Yuhan Fu, Shuaipeng Li, Xingwu Sun, Ruobing Xie, An Wang, Weidong Han, Zhen Yang, Weixuan Sun, Yudong Zhang, and 1 others. 2025. Towards a comprehensive scaling law of mixture-of-experts. *arXiv preprint arXiv:2509.23678*.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, and 1 others. 2024a. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*.

Yuqi Zhu, Jia Li, Ge Li, YunFei Zhao, Jia Li, Zhi Jin, and Hong Mei. 2024b. Hot or cold? adaptive temperature sampling for code generation with large language models. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press.

A Model Configurations

The architectural hyperparameters for a representative set of the 117 models are enumerated in Table 2. In accordance with the methodology outlined in Section 2.3, each model's architecture was algorithmically derived from its target non-embedding parameter count (N). This deterministic mapping maintains structural consistency and near-optimal aspect ratios across scales, thereby isolating the effects of scale from architectural variance.

B Dataset and Validation Details

We use the pretraining corpus released by OpenCoder (Huang et al., 2025) as the training set, which covers public GitHub content up to November 2023. The dataset has already undergone strict deduplication and rule-based filtering, following best practices established for large-scale open code corpora such as The Stack (Lozhkov et al., 2024). The strong performance of OpenCoder indicates the coverage and quality of this corpus. We remove synthetic data to better match the natural distribution of real-world code.

After tokenization with our in-house tokenizer, the raw corpus contains 1216.93B tokens. Since knowledge density does not scale proportionally with raw volume across programming languages, we downsample languages with high redundancy and extreme volume to improve balance and pretraining efficiency. The resulting train set contains 895.51B tokens. Table 3 reports the detailed language composition and summary statistics.

For the scaling-law experiments, training sets at different D are obtained as nested subsets of the same pool. We shuffle the entire pool once with a fixed random seed and take progressively larger contiguous prefixes to form each budget. This design keeps the distribution stable across budgets.

To evaluate trained models, we measure loss on a high-quality validation set drawn from a closed-source internal codebase (Xuyang et al., 2025). It is written by experienced engineers for production use, which aims to ensure realism and quality. Its closed-source nature reduces the risk of contamination from the public training pool. We further perform exact and near-duplicate checks against the training pool to guard against accidental overlap. Although no single source can perfectly represent the entire distribution of code, this validation set provides a consistent benchmark with checks to minimize contamination across all models. The final validation set contains 6.3M tokens.

For data-mixing experiments, the NL train data we used is from Li et al. (2025b).

C Computational Setup

All experiments are conducted on a cluster of NVIDIA H100-80GB GPUs. A meticulous resource allocation strategy is employed to maximize computational efficiency while adhering to the optimal training hyperparameters. For each run, the optimal global batch size (GBZ) is calculated by N and D through a compute-aware heuristic. The maximum per-GPU micro batch size (MBZ), is then determined based on the model's size and training sequence length. This value can be slightly lower in multi-GPU setups. While using more GPUs can accelerate experiments, it often reduces Model FLOPs Utilization (MFU). We carefully manage this trade-off.

The ideal number of GPUs, approximated by GBZ / MBZ, often requires adjustment due to hardware constraints such as integer divisibility. In making the adjustment, the priority is to maintain the GBZ as close to the optimal number as possible, given its critical impact on scaling law dynamics. Across our experimental runs, this strategy resulted in configurations using 8 to 128 GPUs, with a total compute equivalent to approximately 13,600 A100 GPU-days.

D Irreducible Loss Limit Derivation

A key theoretical divergence between the Chinchilla and Farseer scaling laws lies in their prediction of the model's performance limit. The Chinchilla formulation posits a non-zero irreducible loss, E, while the Farseer law's structure allows for a limit that can be zero. This difference is a direct consequence of their mathematical structures. For our fitted Chinchilla law (Equation 3), the limit is a substantial non-zero constant:

$$\lim_{N,D\to\infty}\left(0.2193+\frac{534.374}{N^{0.4853}}+\frac{76.0743}{D^{0.2983}}\right)=0.2193 \tag{5}$$

For our specific fitted Farseer law (Equation 4), while one term decays to zero, the second term converges to a very small, non-zero constant, making

Model	N	D_{count}	D_{range}	d_{model}	d_{ff}	N_{head}	N_{layer}	$N_{\text{with_emb}}$
1	201M	13	[2B, 128B]	1024	2728	16	16	335M
2	284M	13	[2B, 128B]	1152	3032	18	18	435M
3	398M	13	[2B, 128B]	1280	3472	20	20	566M
4	568M	13	[2B, 128B]	1472	3888	23	22	761M
5	798M	13	[2B, 91B]	1600	4264	25	26	1.01B
6	1.13B	13	[2B, 128B]	1792	4832	28	29	1.36B
7	1.61B	13	[2B, 128B]	2048	5448	32	32	1.88B
8	2.27B	13	[2B, 128B]	2304	6064	36	36	2.58B
9	3.18B	13	[4B, 128B]	2560	6952	40	40	3.52B

Table 2: Architectural specifications for a representative subset of the models used in our scaling law experiments. All models share a consistent decoder-only Transformer architecture. N: Number of non-embedding parameters; D_{count}: The number of distinct dataset sizes this model architecture was trained on; D_{range}: The range of training tokens used for this architecture; d_{model}: The hidden dimension of the model; d_{ff}: The intermediate dimension of the feed-forward network; N_{head}: The number of attention heads; N_{layer}: The number of Transformer layers; $N_{\text{with_emb}}$: The total number of model parameters, including token embeddings.

the final limit non-zero:

$$\begin{aligned}\lim_{N,D\to\infty} L(N,D) &= \lim_{N\to\infty} e^{-0.0047N^{0.239}-0.8188} \\ &\quad + \lim_{N,D\to\infty} e^{62.8936N^{-0.0614}-14.0414} \\ &\quad \times D^{-e^{-0.0209N^{0.1943}-0.1826}} \\ &= 0 + e^{-14.0414} \cdot \lim_{D\to\infty} D^0 \\ &= e^{-14.0414} \approx 8.00 \times 10^{-7}\end{aligned} \tag{6}$$

While our specific fit results in a near-zero irreducible loss rather than a true zero, it still stands in stark contrast to the much larger value predicted by the Chinchilla model. This motivates a deeper look into the theoretical argument for why such a limit should be possible, as it explores why, for models with sufficient context, the empirical entropy of a finite dataset can be expected to approach zero.

From an information-theoretic perspective, this discussion is centered on the entropy rate h of the data, which can be expressed as the limit of the conditional entropy:

$$h = \lim_{n\to\infty} H(X_n|X_1,\ldots,X_{n-1}) \tag{7}$$

The central question is whether h must be a positive constant. While this may be true for an idealized, infinite data generating process, it is plausible to argue that the empirical entropy rate of any finite training corpus trends towards zero as the modeled context length n becomes sufficiently large. To make this abstract concept more concrete, let us examine the practical realities of modern transformers.

Consider a model with a typical context window of $n = 4096$ tokens. Its objective is to predict the 4096-th token given the prefix of 4095 tokens. The space of all possible prefixes is vast, on the order of V^{4095} where V is the vocabulary size. Given the finite size of any real-world training corpus (e.g., trillions of tokens), this number is minuscule compared to the space of possible prefixes. Consequently, it is highly improbable that a specific, long prefix of 4095 tokens appears more than once in the entire dataset.

For the vast majority of training instances, the model is therefore presented with a prefix that is empirically unique. This uniqueness implies that the token that follows it is also unique from the dataset's point of view, making it deterministic in this context. In such cases, the empirical conditional probability for the next token is effectively 1 for a single outcome. The conditional entropy for such a unique prefix is therefore zero:

$$\begin{aligned}&H(X_{4096}|X_1,\ldots,X_{4095}) \\ &\quad = -\sum_x P(x|\text{prefix}) \log_2 P(x|\text{prefix}) \\ &\quad \approx -(1 \cdot \log_2 1) = 0\end{aligned} \tag{8}$$

A unique prefix leading to a deterministic next token is not a rare edge case but the dominant scenario when dealing with large context windows on finite data. This prevalence of empirically deterministic sequences suggests that as a model's context capacity increases, the average empirical

Language	Original (B)	Sample	Sampled (B)	Weight(%)
C	52.33	1.00	52.33	5.84
C++	67.43	1.00	67.43	7.53
C#	66.53	1.00	66.53	7.43
Go	12.76	1.00	12.76	1.43
HTML	260.27	0.05	13.01	1.45
Java	148.29	0.50	74.14	8.28
JavaScript	75.84	1.00	75.84	8.47
Others	307.22	1.00	307.22	34.31
PHP	75.67	1.00	75.67	8.45
Python	74.53	1.00	74.53	8.32
Jupyter	15.04	1.00	15.04	1.68
Stack v2	55.46	1.00	55.46	6.19
LeetCode	5.56	1.00	5.56	0.62
Total	**1,216.93**	**–**	**895.51**	**100.00**

Table 3: Composition and sampling configuration of the training set by language.

entropy it is tasked with modeling should decrease. Therefore, a scaling law whose functional form permits a limit of or near zero, like Farseer, may provide a more accurate theoretical foundation for transformers with large context windows. Such models have the capacity to leverage long-range, near-deterministic patterns that are inherent to any finite data collection.