Training Tiny Language Models: An Underexplored Regime Where Conventional Wisdom is Incomplete

Arya Wu, Xilin Wang, Gavin Yang

1. Introduction

Researchers have produced many training heuristics for large language models. The Chinchilla scaling laws established compute-optimal token-to-parameter ratios [1]. Learning rate scaling rules prescribed how to adjust hyperparameters with batch size [2]. The maximal update parameterization (muP) promised zero-shot hyperparameter transfer across model scales [3]. These results, derived from models ranging from hundreds of millions to hundreds of billions of parameters, now constitute the field's conventional wisdom.

We argue that this wisdom does not transfer to small-scale LLMs. Models below 100M parameters constitute a distinct training regime. This regime has largely been ignored. In this regime, established heuristics may not work as expected. Recent work inspired us to argue this claim: Godey et al. [4] demonstrate that small models suffer from a "softmax bottleneck" causing late-training performance degradation absent in larger models; Diehl Martinez et al. [5] show that small models exhibit slow and unstable convergence throughout training; and both MiniCPM [6] and SmolLM2 [7] report that standard hyperparameter choices require substantial modification at small scales. Although models in the works mentioned above may not strictly be below 100M, these findings motivate us to find evidence directly related to our claim.

Our experiments training 30M–75M parameter models reinforce the common theme of these findings. Learning rate extrapolations, batch-size scaling rules, and Chinchilla-optimal training durations that work reliably at 130M+ parameters failed non-monotonically at smaller scales. Small language models are increasingly important for edge deployment, as testbeds for architectural innovation, and for community-driven efficiency research. If our scaling laws cannot be trusted below 100M parameters, the field needs new empirical understanding of this regime.

2. Batch Size for Small-Scale Training

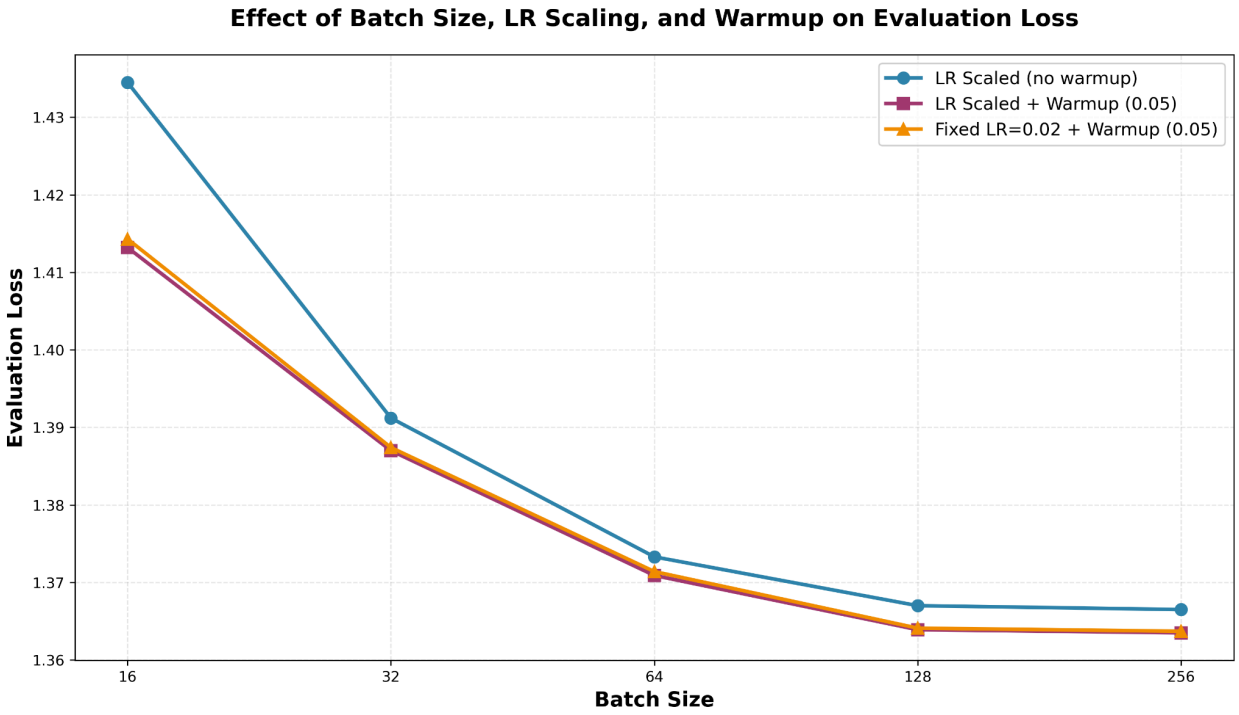
Batch size has long been recognized as a critical lever in model training. Larger batches enable faster wall-clock training through parallelism, and a body of research has established practices for exploiting this without sacrificing model quality [8].

Two heuristics have been widely adopted: critical batch size—the threshold beyond which larger batches yield diminishing returns, namely the point that minimizes the number of training tokens required for a target loss [9]; and the square root learning rate scaling rule—when increasing

batch size by a factor of k, multiply the learning rate by √k. The latter, originally derived from random matrix theory for adaptive optimizers like Adam [10], has been validated on models from BERT to GPT [11]. Similar trends appear in the Marin Speedrun project, where experiments on 130M+ parameter models confirm the rule's effectiveness [12].

But do these heuristics transfer to smaller scales? We investigated this question by training 50-million-parameter models under a 2× Chinchilla-optimal regime across batch sizes from 16 to 256. Our findings reveal that the intuitions developed at large scale can be misleading when applied to tiny models.

The following figure plots C4-BN BPB loss against batch sizes under three settings.



2.1 Critical batch size is larger than expected for tiny models

Across all experimental configurations, the lowest evaluation loss occurred at batch size 256. This contradicts the prevailing intuition that critical batch size should *increase* with model scale. Prior experiments for Muon show 130M models degrading when batch size increases from 128 to 256, while 1.2B models tolerate both equally well under 1× Chinchilla settings. By this logic, 50M models should plateau or degrade at even smaller batch sizes – however, our result reveals its critical batch size to be potentially larger than 256 (due to computational constraints we did not test batch sizes beyond).

The standard explanation for critical batch size is that larger batches stabilize gradient estimates toward the true gradient, reducing the steps needed to reach a target loss—but only up to a

point, beyond which further increases provide no additional benefit as gradient estimates are stable enough [8]. Our results suggest that for 50M models, gradient estimates remain noisy even at batch size 128, contrary to the assumption that smaller models should exhibit less gradient variance. This finding cautions against blindly applying insights from large-scale training to small-scale regimes without empirical validation.

2.2 Square root scaling for learning rate provides marginal benefit at best

We compared a scaled learning rate configuration (yellow line, applying $\sqrt{k}$ scaling from a base batch size of 128) against a fixed learning rate across all batch sizes (purple line). The difference was negligible with an average gap of 0.03%: at batch size 128, the scaled configuration achieved 1.3639 BPB versus 1.3641 for the fixed rate. Interestingly, the discrepancy between two settings diminishes as batch size increases.

Prior work on 130M+ models claims that square root scaling preserves training dynamics across batch sizes. Yet their empirical evidence is less direct than the theoretical foundation suggests: studies on GPT models and Marin-style models plotted learning rate × √batch size against evaluation loss, with no direct demonstration of the benefits from setting *LR / sqrt(BatchSize)* to be constant [12]. Our controlled comparison—holding all else constant while toggling the scaling rule—provides direct evidence that this widely-believed heuristic may not be as essential as claimed. We do not argue that learning rate scaling is useless at small scales, only that its necessity warrants scrutiny. Hyperparameter interactions at small scales may simply differ from those at large scales, potentially obviating the need for scaling entirely.

2.3 Linear warmup scaling consistently improves training

Standard Muon configurations on the Marin leaderboard omit warmup steps, and prior investigations have briefly brought up the potential effect of warmups without empirical guidance [12]. We found that adding a warmup ratio of 0.05 (linearly scaling with training steps) yielded lower evaluation losses across all batch sizes from 16 to 256. The improvement margin persisted as batch size increased.

Our training curves reveal a consistent pattern: the warmup configuration initially lags behind its counterpart but gradually overtakes it, ultimately achieving lower final loss. This suggests that warmup enables more stable optimization trajectories less susceptible to early plateaus. We did not systematically search for optimal warmup ratios or schedules; nonetheless, this experiment underscores the transferability problem, in which a practice deemed unnecessary at 130M+ parameters may prove beneficial at 50M.

3. How Long Should You Train Your Tiny Language Model?

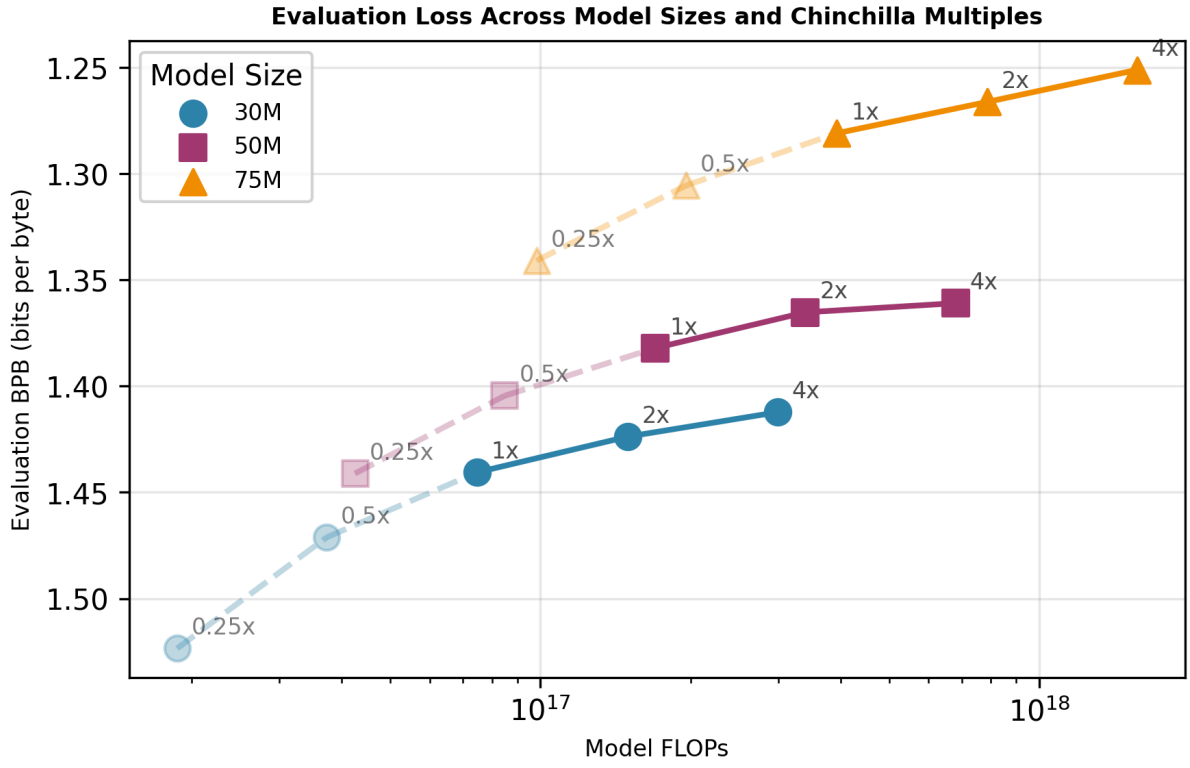
Training a language model costs millions of dollars, making one question critically important: given a fixed compute budget, how should we balance model size and training duration to maximize quality? Since total training cost is approximately proportional to the number of parameters multiplied by the number of training tokens, every training run involves deciding

whether to spend compute on more parameters or more data. The influential Chinchilla [1] paper provided an answer. After training hundreds of models across a wide range of sizes and token budgets, Hoffmann et al. [1] reported an optimal token-to-parameter ratio of roughly 20 tokens per parameter. This "20× rule" became the dominant heuristic, shaping both academic benchmarks and early industrial LLM development.

However, industry practice has shifted toward overtraining models far beyond Chinchilla-optimal ratios. Epoch AI [13] reports that the ratio of training data to parameters in open-weight LLMs has grown 3.1× per year since 2022, with recent models trained on 20× more tokens per parameter than Chinchilla's optimal ratio. The Llama family demonstrates this escalation: Llama-1 65B adhered to the prescribed 20× ratio, but Llama-2 70B pushed to 30× and Llama-3 70B exploded to 200× (15 trillion tokens). This trend is driven in part by the demand for powerful, smaller models in the 1B-70B parameter range that are easier and cheaper to finetune and deploy.

This raises a natural question: should we overtrain tiny models (<100M params) beyond Chinchilla-optimal, following industry trends with larger models?

To investigate, we train 30M, 50M, and 75M parameter models at 0.25x, 0.5x, 1× Chinchilla-optimal (20 tokens/parameter), 2× (40 tokens/parameter), and 4× (80 tokens/parameter) using the Muon optimizer. We plot evaluation C4-BN bits-per-byte (BPB) loss against model FLOPs, defined as the total number of FLOPs required to train the model.



3.1 Extended training continues improving tiny models with no saturation observed.

Across all model sizes (30M, 50M, 75M), we observe consistent quality improvements as we increase training duration from 1× to 2× to 4× Chinchilla-optimal. Crucially, we find no evidence of a saturation point: loss curves show no signs of plateauing, suggesting that further extended training would continue yielding quality gains.

While compute constraints prevented us from exploring higher multiples, the absence of saturation hints that extreme overtraining—as demonstrated by Sardana et al. (2024) [14] who trained small models up to 500× Chinchilla to match larger models—might also benefit tiny models.Whether tiny models (<100M parameters) exhibit similar gains at extreme training durations remains an open question for future work.

This finding validates the core motivation behind industry's overtraining strategy: longer training does produce better small models, making the extra training cost potentially worthwhile for deployment scenarios where smaller model size reduces inference costs.

3.2 Chinchilla-Optimal is NOT compute-efficient for tiny models: at this scale, model capacity dominates.

While extended training improves absolute quality (Finding 1), it does not provide a compute-efficient path to better models. At fixed FLOP budgets, larger models at lower D/N ratios consistently dominate smaller models at higher D/N ratios, across a wide range of training durations. The performance gaps between model sizes far exceed the marginal gains from extended training within a single model size.

But is Chinchilla-optimal (D/N≈20) itself the right target at tiny scale? To investigate this, we test sub-Chinchilla training regimes at 0.5× and 0.25× (D/N ≈ 10 and 5). Recent work challenges Chinchilla's universality—Farseer [15] introduces refined scaling laws demonstrating that Chinchilla's rule of thumb applies only at moderate compute budgets ($C \approx 10^{20}$–10^{21} FLOPs) and breaks down at both extremes. Our experiments confirm this: At roughly 2×10^{17} FLOPs, Chinchilla's framework would recommend a 50M model at 1× (D/N ≈ 20, BPB 1.40). Yet a 75M model trained at just 0.25× (D/N ≈ 5, BPB 1.35) achieves *better* performance with *less* compute. This pattern persists across budgets. For any fixed FLOP limit, allocating compute to a *larger* model with *minimal* training reliably outperforms allocating it to a smaller model at D/N ≈ 20.

Extrapolating our performance curves suggests an optimal D/N below 5. However, such low ratios yield poor absolute performance (BPB > 1.5), rendering them impractical. In practice, this means the tiny-scale regime offers no meaningful compute–performance tradeoff: to achieve good results, one should simply train the *largest* model that fits the FLOP budget, regardless of D/N.

3.3 Extreme hyperparameter instability blocks systematic exploration.

Despite the promise of extended training, we observed severe and unpredictable hyperparameter sensitivity that makes systematic exploration impractical. Learning rates extrapolated from reported optimal 130M-scale Muon configurations—adjusted for model size

and training duration—often failed catastrophically, even for seemingly minor differences in configuration.

We observed two forms of non-monotonic instability. Across model sizes, 4× Chinchilla rates worked for 30M and 75M but diverged for 50M, which only stabilized after reducing Muon LR from 0.016 to 0.006. Within a single model size (75M), the same extrapolated rates worked at 1× and 4× Chinchilla but failed at 2×, converging only after reducing Muon LR from 0.018 → 0.012 and Adam LR from 0.0028 → 0.0020.

This non-monotonic behavior—where similar configurations require drastically different hyperparameters—turns systematic exploration into trial-and-error and prevents reliable transfer across scales. Future work should develop principled scaling laws for tiny-model hyperparameters, enabling efficient exploration of extended training regimes.

4. Conclusion

We have argued that small-model training below 100M parameters constitutes a distinct empirical regime where the field's established training heuristics fail. Our experiments demonstrated this failure across two dimensions. First, batch-size scaling rules derived from large-model training do not transfer: the learning rate adjustments that stabilize training at scale produced inconsistent or degraded results at 30M–75M parameters. Second, Chinchilla-optimal training durations do not apply at a tiny scale: model capacity dominates, and allocating compute according to D/N ≈ 20 does not reliably yield the best results. While extended training continues to improve absolute performance, achieving these gains requires careful and sometimes non-monotonic hyperparameter tuning, as small changes can cause catastrophic failures.

These findings have practical implications. Small language models are no longer merely stepping stones to larger systems. They are deployment targets for edge devices, efficient testbeds for architectural research, and the substrate for community-driven efficiency research. The implicit assumption that insights transfer downward from large-scale training has allowed the sub-100M regime to remain underexplored. Our results suggest this assumption is unsafe, and that the field would benefit from systematic investigation of small-model training dynamics as a research problem.

4.1 Limitations

Our experiments were conducted within the Marin Speedrun framework, and this choice introduces limitations that warrant explicit discussion.

First, our results reflect a specific and potentially idiosyncratic computational environment. We encountered out-of-memory failures that required modifying JAX environment variables: flags controlling memory preallocation and garbage collection behavior. These modifications, while necessary to run experiments, produced measurably different training dynamics—a troubling interaction that we could not fully characterize. The fact that low-level runtime configuration can

significantly affect model training outcomes, without clear documentation of which settings the reference runs employed, undermines confidence in cross-submission comparisons. Speedrun leaderboards implicitly present results as commensurable, but hidden environmental dependencies call this into question.

Second, the Marin framework's reliance on JAX introduces complexity that hampers debugging and reproducibility. JAX's compilation model, while offering performance benefits, makes it difficult to isolate whether unexpected behaviors stem from optimizer dynamics or compilation artifacts. When our 50M model collapsed at 4× Chinchilla while the 30M and 75M models did not, distinguishing between a genuine optimizer problem and a framework-specific issue required substantial effort with limited tooling. PyTorch's eager execution model, while potentially slower, would have permitted more transparent diagnosis. For a competition aimed at discovering generalizable training insights, the choice of framework matters: complexity that obscures failure modes reduces the scientific value of negative results.

Third, and more broadly, speedrun competitions optimize for a metric that may not align with scientific understanding. Submissions are incentivized to exploit hardware-specific optimizations, framework quirks, and hyperparameter configurations that win on the leaderboard but do not generalize. A competition that aims to advance training efficiency would benefit from stricter controls: standardized environments, comprehensive mandatory configuration disclosure, and evaluation across multiple hardware targets.

These limitations do not invalidate our findings. The failures we observed are real, and the patterns we document are unlikely to be pure artifacts. But they do constrain interpretation. Our claim is not that Muon or the Marin framework are fundamentally flawed, but that small-model training exposes fragilities in optimizers, in heuristics, and in evaluation infrastructure. Larger-scale training may simply dismiss these fragilities with excess capacity. The field's path forward requires both better empirical understanding of small-model dynamics and more rigorous experimental frameworks in which to develop that understanding.

References

[1] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, et al. 2022. Training Compute-Optimal Large Language Models. In NeurIPS 2022. arXiv:2203.15556

[2] Priya Goyal, Piotr Dollár, Ross Girshick, et al. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. arXiv:1706.02677

[3] Greg Yang, Edward J. Hu, et al. 2022. Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer. In NeurIPS 2022. arXiv:2203.03466

[4] Nathan Godey, Éric de la Clergerie, and Benoît Sagot. 2024. Why Do Small Language Models Underperform? Studying Language Model Saturation via the Softmax Bottleneck. arXiv:2404.07647

[5] Rhys Diehl Martinez, Edoardo Maria Ponti, and Christopher D. Manning. 2024. Tending Towards Stability: Convergence Challenges in Small Language Models. In Findings of EMNLP 2024. arXiv:2410.11451

[6] Shengding Hu, Yuge Tu, Xu Han, et al. 2024. MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies. arXiv:2404.06395

[7] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, et al. 2025. SmolLM2: When Smol Goes Big—Data-Centric Training of a Small Language Model. arXiv:2502.02737

[8] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. 2018. An Empirical Model of Large-Batch Training. arXiv:1812.06162

[9] Naoki Sato, Hiroki Naganuma, and Hideaki Iiduka. 2025. Convergence Bound and Critical Batch Size of Muon Optimizer. arXiv:2507.01598

[10] Diego Granziol, Stefan Zohren, and Stephen Roberts. 2022. Learning Rates as a Function of Batch Size: A Random Matrix Theory Approach to Neural Network Training. Journal of Machine Learning Research 23(173), 1–65. arXiv:2006.09092

[11] Sadhika Malladi, Kaifeng Lyu, Abhishek Panigrahi, and Sanjeev Arora. 2022. On the SDEs and Scaling Rules for Adaptive Gradient Algorithms. In NeurIPS 2022. arXiv:2205.10287

[12] Marin Community. 2024. Marin Speedrun: Batch Size and Learning Rate Scaling. https://github.com/marin-community/marin/issues/1565

[13] Epoch AI. 2024. Training Tokens per Parameter. https://epoch.ai/data-insights/training-tokens-per-parameter

[14] Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. 2024. Beyond Chinchilla-Optimal: Accounting for Inference in Language Model Scaling Laws. In ICML 2024. arXiv:2401.00448

[15] Houyi Li, Wenzhen Zheng, Qiufeng Wang, Zhenyu Ding, Haoying Wang, Zili Wang, Shijie Xuyang, Ning Ding, Shuigeng Zhou, Xiangyu Zhang, and Daxin Jiang. 2025. Predictable Scale: Part II, Farseer: A Refined Scaling Law in Large Language Models. arXiv:2506.10972