EXPERIMENTAL EVALUATION OF ALGORITHMIC PROGRESS IN AI

Anonymous authors
Paper under double-blind review

ABSTRACT

Algorithms have been estimated to increase AI training efficiency by a factor of 10^4 (Ho et al, 2024). We run controlled ablation experiments and benchmark the gains in training efficiency from many recent training improvements, but find that they only account for a modest 10x of these gains at small scales. We postulate that significant gains come from algorithms improving with compute scale. We run scaling law experiments, ablating important algorithmic improvements, and find evidence that some may not only affect efficiency but also the scaling laws themselves. In the process, our study illustrate the interaction between algorithmic improvements and clarify how to think about algorithmic advances in AI. Finally, we suggest revising commonly held assumptions for modeling advances in AI algorithms and rethinking the role of algorithmic advances in AI.

1 INTRODUCTION

Algorithms have played an important role in machine learning progress. (Ho et al., 2024) estimated that algorithmic progress has contributed more than 4 orders of magnitude to compute scaling in training in the last 10 years. However, their contribution is hard to estimate. In this respect, they constitute the dark energy of training - an unseen quantity that is responsible for the expansion of deep learning. Here, we try to probe deeper into the root causes of algorithms. We find that in analogy with cosmological dark energy, algorithmic advances matter much more at larger scales. Our experiments provide compelling evidence that algorithms have had a strong impact on neural scaling laws. In the process, our analysis sheds light on the tenets of the dominant theory of algorithmic progress. In the dominant framework, algorithmic advances are seen as relatively smooth, continuous scale-independent advances that multiplicatively build on each other, leading to smooth exponential growth in effective compute over time. Ho et al. (2024) estimates that algorithmic progress increases effective compute around 3x per year. Using ablation experiments, our research calls into question many of these tenets.

The dominant framework in the algorithmic progress literature views algorithmic advances as multiplicative advances, which multiply the amount of effective compute and which don't have noticeable efficiency changes with scale Davidson et al. (2023) Ho et al. (2024). For instance, (Ho et al., 2024) estimates that the invention of the transformer led to a 60x increase in effective compute, i.e, previous architectures like LSTMS would have required 60 times the physical compute resources to match the transformer. This is based on research such as the (Hestness et al., 2017), which found no scaling exponent difference between architectures LSTMs and RHN (Recurrent Highways Networks), as well as between Adam and SGD optimizers. (Bansal et al., 2022) measured the data scaling exponent between Encoder-Decoder, Decoder only, and mixed LSTM-Transformer architectures and found little difference.

In parallel, ML theory has increasingly focused on the intrinsic properties of training data as the most important factor in determining the exponent in neural scaling laws (Michaud et al., 2023)(Bahri et al., 2024).

However, this paradigm has started to come under criticism. Sanderson et al. (2025) examines the compute multipliers from selected papers and speculates that algorithmic advances like the LSTM-Transformer transition had scale-dependent effects. There are also new AI architectures that are purported to have a larger/steeper scaling exponent (Liu et al., 2024).

2 Experiment Description

Broadly, we study two different "model" models. The first is a transformer-based model. The second is an LSTM based on the architecture and parameter . We try to keep as many things consistent as possible between the two models while adjusting some hyperparameters to keep optimal performance on each model, respectively. We base as many hyperparameter choices as possible on state-of-the-art parameters for small models in the literature. Sometimes we have to choose between different recommendations in the literature or choose different parameters to keep consistency between models. In these cases, we run variation experiments to justify our choices (see Appendix 12). We also run learning rate tunes for all our models at every scale.

Our experimental setup consists of a transformer and an LSTM-based model. We attempt to control for as many confounders as possible while using hyperparameters that are consistent with the respective architecture. All the models in this paper share the same tokenizer and use the same C4 dataset. Our architectural and hyperparameter choices for our transformer model are primarily based on those in Srećković et al. (2025), Hoffmann et al. (2022), and Porian et al. (2024). For parameters not specified in these papers, we do a small tuning study or find other relevant literature to base our parameters on. Our LSTM model is based on Melis et al. (2017), where we use the parameters from their word-level 10M parameter model. We do learning rate tunes across all our models at all scales. See Appendix 11.1 and the following discussion for more details on our experimental setup.

2.1 Transformer Model

For our transformer-based experiments, we use a vanilla transformer with rotary-based encodings. We keep a constant width to depth aspect ratio. Kaplan et al. (2020) used a $N : L \approx 100$ width-depth ratio (Dey et al., 2025). Given the scale of our experiments, we choose a ratio of $N : L \approx 16$. This gives us depth to width ratio similiar to that of other small transformers like Lan et al. (2019). Attention heads number is commonly set such that the dimension of each head stays at a constant size, such as 64 or 128 (Hoffmann et al., 2022)(Vaswani et al., 2017). Given the scale of our experiments, we choose a constant attention head dimension of 16. In our experience, setting a larger head dimension and therefore reducing the number of attention heads degraded performance at small scales. To scale our transformers, run a learning rate tune at each model size over the range: $[10^{-3}, 10^{-2.5}, 10^{-2}, 10^{-1.5}, 10^{-1}]$. Each learning rate tune consist of a small training run with a fixed token budget and fixed schedule (across sizes). We measure FLOPs by running PyTorch profiler over one minibatch and scaling according to gradient accumulation and number of steps. Our initialization, unless stated otherwis,e has 0 bias, all weights are sampled from $\mathcal{N}(0, 0.02)$. We scale the output projections initialization by $1/\sqrt{2L}$ where L is the number of layers as is done in Wang et al. (2024) as this is more suitable for the deeper networks in our study (Radford et al., 2019). We use pre-layer normalization (Xiong et al., 2020) and experiment with RMSNorm but find it has little effect on final loss or compute efficiency (see Section 3.5).

2.2 LSTM Model

For our LSTM model we choose a vanilla LSTM and do not analyze more recent variations developed after 2018 (see (Melis et al., 2019) and (Beck et al., 2024)). In addition, we do not include the many pre-2018 variations of LSTMs like GRUs as Greff et al. (2016) finds that these variants do not improve on the standard LSTM. We develop a setup based on state-of-the-art LSTMs before the invention of the Transformer using hyperparameters from Melis et al. (2017). We focus our efforts on a 2-layer LSTM, which seems to perform well in practice for many studies at many sizes (Reimers & Gurevych, 2017)(Melis et al., 2017). The layer number seems to scale very moderately with LSTM size (Melis et al., 2017) so we keep the layer number constant while scaling hidden dimension. In addition to standard LSTM optimizations ie, gradient clipping, and TBPTT (Truncated Back Propagation Through Time), we also use layer normalization. All other parameters we keep constant between LSTM and Transformer (see Appendix). Similiar, to our transformer model, for each scale we run a learning rate tune over: $[10^{-3}, 10^{-2.5}, 10^{-2}, 10^{-1.5}, 10^{-1}]$. We use standard LSTM initialization consisting of Xavier-uniform embedding weights, orthogonal recurrent weights, and 0 bias except for a forget gate bias of +1. We include a variational dropout with separate dropout rates for embedding matrices vs hidden matrices (Melis et al., 2017)(Merity et al., 2017).

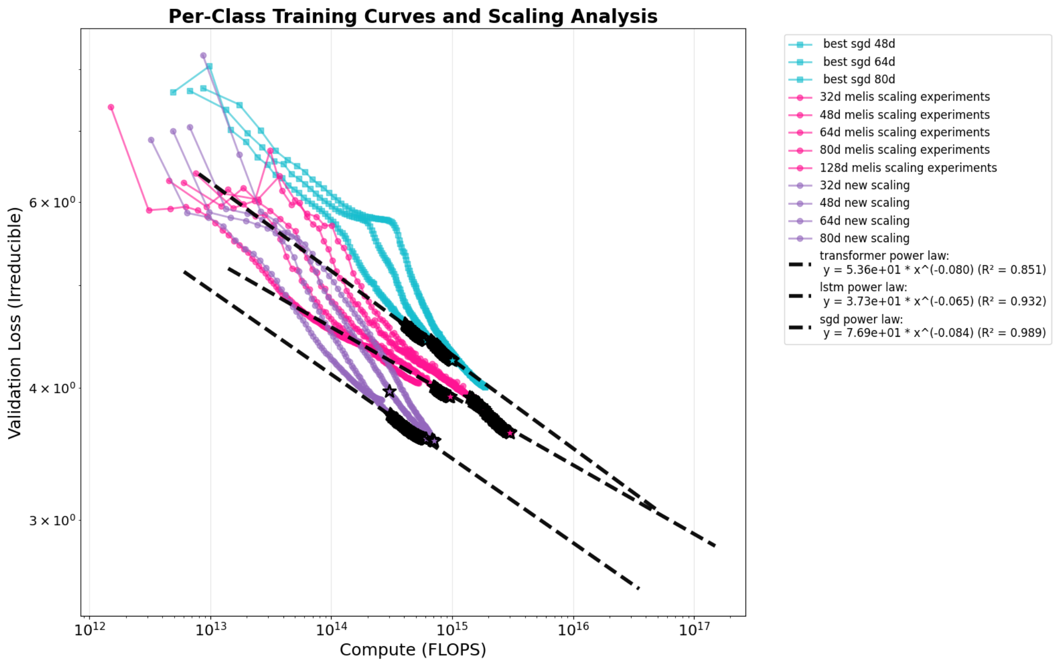

Figure 1: Graph of training runs for standard transformer, transformer using SGD, transformer with sinusoidal positional embeddings, and an LSTM.

2.3 Determining Compute Optimal Frontier

To determine the computationally optimal frontier, we use two techniques based on the methods outlined in Kaplan et al. (2020). First, we try to find a consistent power law envelope to the points traced out by the models in our training run. For each curve, we identify this region visually and then fit a power law to these points. In particular, we subtract out an irreducible loss of 1.8 and do a power law fit. For the second method, we take training runs and fit them to a scaling law form below basedChoshen et al. (2025). We use a similar procedure to Choshen et al. (2025) fitting on intermediate points using scipy curvefit, while keeping the more robust Huber loss as done in Hoffmann et al. (2022). Specifically, we remove the first third of training steps in order to fit more intermediate points as is done in Choshen et al. (2025).

The resulting scaling exponents using each method are illustrated in the table below.

Table 1: compute exponents

Name	γ (90% CI)	a (90% CI)	b (90% CI)
Transformer	-.3421 [-.344, -.340]	0.434 [0.430, 0.4369]	0.5657 [0.563, 0.569097]
Transformer SGD	-0.036 [-0.041174, -0.0327]	.861	.1385
LSTM	-.148 [-.1934, -0.1023]	0.1858 [0.12, 0.24]	0.8141 [0.759, 0.87148]

2.4 Discussion

Our scaling graphs hint that improvements in neural network architecture are not scale invariant and have increasing returns to scale effects. However, optimizer choices seem to be scale-invariant. While other algorithmic changes, like improved positional embeddings have decreasing returns to scale.

3 What Has Been the Impact of Particular Algorithmic Innovations?

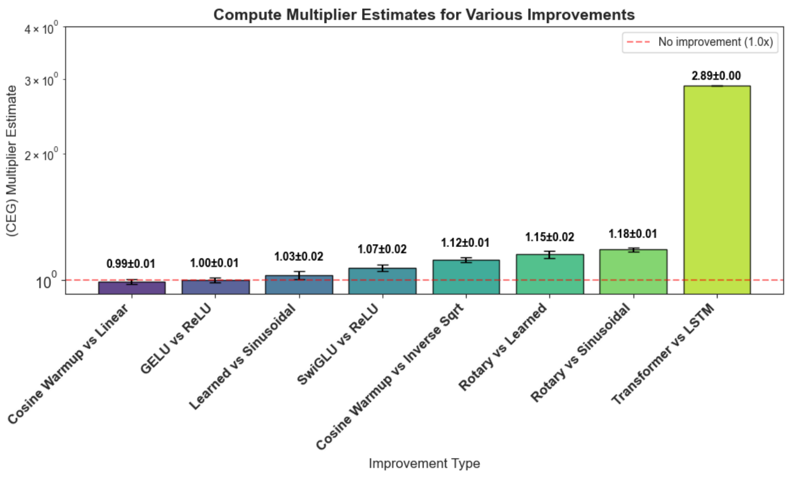

Figure 2: Compute Equivalent Gains/training efficiency gains for some of the algorithms measured in this study using a 3.6M parameter transformer model. Error bars are bootstraped by running 4 separate training runs with different seeds for each baseline.

Here, we try to get a fine-grained picture of algorithmic improvements in training by running a large series of ablation experiments. For instance, running a transformer with older activation functions like ReLU and with newer activation functions like SwiGLU. In each case, we run learning rate tunes where we sweep learning rates $10^{0.5}$ higher and lower than the learning rate of the model without the ablation. For each model, we run training with four different seeds to better identify the extent of the algorithmic improvement relative to noise. In order to run experiments with many seeds, all of these experiments are done with a small 3.6 million parameter transformer model (SwiGLU activation adds a small number of additional parameters).

We choose to examine activation functions, positional encodings, normalization techniques, learning rate schedules, initialization techniques, and optimizers. We leave out tokenizers and data quality enhancements as they are difficult to evaluate using perplexity. We examine literature that has benchmarked these contributions. However, in general, we find that estimating algorithmic improvements from the primary literature (i.e, from the papers that proposed the algorithmic improvement itself) is significantly higher than the secondary literature (i.e, estimates from papers that include multiple previous algorithms as benchmarks). The primary literature may have a bias to enlarge their innovation to make it seem more favorable. Experimentally, we find that in many of these categories (except initialization and normalization), performance has improved by $10\% - 20\%$ since the invention of the transformers at the scales we measure. We describe the effects of each of these innovations below and in Fig 2.

3.1 Activation Functions

Our study finds relatively little gain due to activation functions. Interestingly, our results suggest that ReLu outperforms GELU at the scales we test. We are not the first to show this result. ReLU marginally outperforms GELU in Shazeer (2020). Mirzadeh et al. (2023) finds that ReLU has a negligible impact on performance and recommends the reintroduction of ReLU into language models.

3.2 Positional Encoding

We find that positional encodings have a fairly large impact on training efficiency in transformers. We test three different types of positional encoding, including learned positional encoding. We estimate that rotary encoding constitutes an improvement of 2-3x in training efficiency over sinusoidal encoding originally employed in Vaswani et al. (2017). This is in line with previous work (Sanderson et al., 2025). However, we see that they were only a modest improvement over their predecessor's learned positional embeddings.

3.3 Learning Rate Schedules

Our test consists of 3 prominent learning rate schedules used in the last 10 years. These include an inverse square root learning rate decay schedule variations of which were used in T5Raffel et al. (2020) and Vaswani et al. (2017). We also implement a linear warmup/linear decay schedule as used in BERT ((Devlin et al., 2019)) and linear-warmup with cosine decay, which is a more recent default use in GPT-3 Brown et al. (2020). We measure some training efficiency gains due to learning rate schedulers. We observe a CEG gain of about 12% training efficiency gain between the cosine decay schedule and the inverse square root decay schedule, but notice little difference as long as the decay is faster than square root, which is the more important distinction (Zhang et al., 2025). Our results are roughly in line with Kaplan et al. (2020), which find minimal difference between state-of-the-art learning rate schedulers. However, Bergsma et al. (2025) finds that their learning rate scheduler has a 60% compute efficiency gain for 610M parameter model.

3.4 Initialization

We compare three different types of initialization. Xavier-uniform, Kaiming-normal, and transformed scaled initializations developed by Wang et al. (2024), which is our default. Wang et al. (2024) modifies the initialization developed by BERT by scaling output projections by $1/\sqrt{2L}$. In our transformer model, we find that these variations have a negligible impact on transformer pretraining efficiency. Papers like Bachlechner et al. (2021) developed an initialization technique which they say improves pretraining efficiency by 56% for a 12-layer transformer. However, much of the initialization literature we encountered concentrated on stabilizing training or tailoring training for specific architectures rather than broad efficiency improvement per-se.

3.5 Normalization Techniques

We test 3 different normalization techniques: pre-layer norm, post-layernorm, and rmsnorm. Pre-layernorm applies layernorm before multihead attention in the residual stream. This is generally seen as more stable than its predecessor post-layernorm. RMSNorm is an update to layernorm, which may have runtime benefits of $7\% - 64\%$ because it removes the mean centering step (Zhang & Sennrich, 2019). However, in these variations seem to have little impact on the computational efficiency of training.

3.6 Optimizer Adam vs SGD

The gap between Adam and SGD is a controversial subject. We find it difficult to scale SGD on transformers with our default batch size of 256. Sgd performs notably badly on transformers in comparison to Adam (Ahn et al., 2023) (Zhao et al., 2024). This is theorized to be due to Hessian heterogeneity Zhang et al. (2024). In contrast, Hestness et al. (2017) found that Adam and SGD scale similarly with respect to data on recurrent highway networks. More recent work finds that the Adam SGD gap is considerably smaller when high momentum is used and batch sizes are very small Srećković et al. (2025). Zhang et al. (2019) finds that Adam is compatible with much higher critical batch sizes than SGD. Using a smaller batch size of 64 we are able to get scaling behavior.

3.7 Dataset and Tokenizers

We do not do any tokenizer and dataset ablation experiments. In general, it is hard to compare performance between tokenizers and datasets as perplexity is a tokenizer and dataset-relative benchmark. However, there has been significant benchmark-based efficiency studies. Li et al. (2024) did a large-scale study comparing pretraining efficiency on new and historical datasets. They found that their optimal dataset was able to improve pre-training efficiency by a factor of 2-3 over the worst-performing historical dataset (C4). Similarly, optimized datasets like (Xie et al., 2023) have shown pretraining efficiency gains of 2.6x. In regards to tokenizer, Ali et al. (2024) finds that inefficient tokenization leads to additional training costs of 68%.

3.8 Kaplan to Chinchilla Scaling Laws

Kaplan et al. (2020) advocates for a smaller level of data scaling relative to parameters as compute scales relative to the chinchilla scaling procedure developed by Hoffmann et al. (2022). Subsequent literature found that Kaplan misapproximated scaling law exponents due to small sizes, not accounting for embedding parameters, and using a constant warmup size (Porian et al., 2024)Pearce & Song (2024). Hoffmann et al. (2022) accounted for all parameters and found that data and parameters should be scaled equally while scaling compute. We can estimate the compute efficiency gains of the switch from Kaplan to Chinchilla using the recommended data and compute allocations from each source. Here, we assume that computer scaling laws are governed by a chinchilla optimal form in reality, while model trainers allocate parameters and data according to mistaken Kaplan scaling laws (see Appendix 13). This form of algorithmic improvement has clear scale-dependent effects but has an interesting form where the training efficiency gap first converges, then diverges.

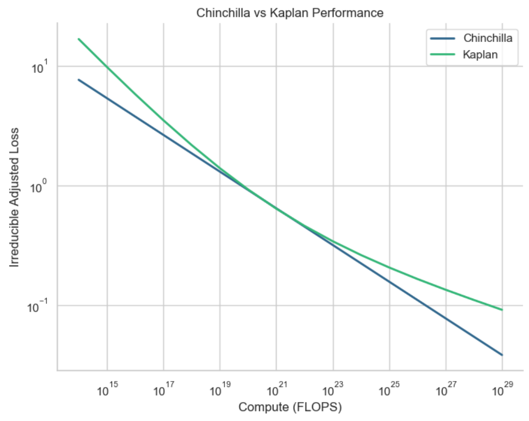

Figure 3: Difference in performance between models scaled with Kaplan vs Chinchilla recommendations. Interestingly, the compute gap first converges than diverges.

Are Algorithmic Improvements Multiplicative

4 Insights on Training Models and Scaling Law Analysis at Small Scales

5 Limitations and Further Work

We would like to reiterate the limitations of our current study and encourage further work in this field. First, we only address innovations that improve FLOP efficiency. Many of the key innovations in recent years have been innovations that improve speed and GPU utilization, like Flash Attention. This is also not addressed in Ho et al. (2024) and much of the algorithmic progress literature, which leaves out an important element of progress. Further, algorithmic improvements are heavily multidimensional and examining only efficiency gains gives a warped picture of the true benefits of algorithmic innovations.

In addition, our experiments are conducted at small scales compared to more recent scaling studies (i.e Hoffmann et al. (2022)). However, our studies are done at similar scales to prominent studies like Kaplan et al. (2020) and at similar or larger scales to the papers that introduced many of these innovations Melis et al. (2017).

Another key limitation is the fickle nature of hyperparameters. For instance, we found it nearly impossible to scale a transformer trained with sgd with a batch size over 128. We are more confident in our results because our performance aligns well with other studies like Porian et al. (2024) and Melis et al. (2017). However, our study leaves open the possibility of a ”golden” hyperparameter that could significantly lower the gap between the algorithmic improvements we measure or even change the experimental scaling relationship. For instance, SGD was long thought to be inherently inferior to Adam for Transformer training, but studies like Srećković et al. (2025) demonstrate a very small performance gap with the right hyperparameters.

In addition, we did not have sufficient compute to do scaling law studies for all the algorithms we studied in Section 11

6 The Nature of Algorithmic Progress in AI

7 Implications for AI Progress

7.1 What Portion of Algorithmic Progress is due to Scaling

8 Conclusion

9 LLM Usage

LLMs played a significant role in generating code for many of our experiments. All code was validated by the researchers in this paper. We also used LLMs to find relevant literature.

10 Reproducibility Statement

All code used in model training and model analysis is hosted here, along with setup instructions: https://anonymous.4open.science/r/Experimental_Progress-02EF/. The main body of our experiments section (see Sec 2) contains details on our training and analysis procedure. A detailed list of hyperparameters used is in Appendix 11.1.

References

Kwangjun Ahn, Xiang Cheng, Minhak Song, Chulhee Yun, Ali Jadbabaie, and Suvrit Sra. Linear attention is (maybe) all you need (to understand transformer optimization). *arXiv preprint arXiv:2310.01082*, 2023.

Mehdi Ali, Michael Fromm, Klaudia Thellmann, Richard Rutmann, Max Lübbering, Johannes Leveling, Katrin Klug, Jan Ebert, Niclas Doll, Jasper Buschhoff, et al. Tokenizer choice for llm training: Negligible or crucial? In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 3907–3924, 2024.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in Artificial Intelligence*, pp. 1352–1361. PMLR, 2021.

Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 121(27), June 2024. ISSN 1091-6490. doi: 10.1073/pnas.2311878121. URL http://dx.doi.org/10.1073/pnas.2311878121.

Yamini Bansal, Behrooz Ghorbani, Ankush Garg, Biao Zhang, Colin Cherry, Behnam Neyshabur, and Orhan Firat. Data scaling laws in nmt: The effect of noise and architecture. In *International Conference on Machine Learning*, pp. 1466–1482. PMLR, 2022.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *Advances in Neural Information Processing Systems*, 37:107547–107603, 2024.

Shane Bergsma, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Straight to zero: Why linearly decaying the learning rate to zero works best for llms, 2025. URL https://arxiv.org/abs/2502.15938.

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

Leshem Choshen, Yang Zhang, and Jacob Andreas. A hitchhiker's guide to scaling law estimation, 2025. URL https://arxiv.org/abs/2410.11840.

Tom Davidson, Jean-Stanislas Denain, Pablo Villalobos, and Guillem Bas. Ai capabilities can be significantly improved without expensive retraining, 2023. URL https://arxiv.org/abs/2312.07413.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Nolan Dey, Bin Claire Zhang, Lorenzo Noci, Mufan Li, Blake Bordelon, Shane Bergsma, Cengiz Pehlevan, Boris Hanin, and Joel Hestness. Don't be lazy: Completep enables compute-efficient deep transformers. *arXiv preprint arXiv:2505.01618*, 2025.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10): 2222–2232, 2016.

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.

Anson Ho, Tamay Besiroglu, Ege Erdil, David Owen, Robi Rahman, Zifan Carl Guo, David Atkinson, Neil Thompson, and Jaime Sevilla. Algorithmic progress in language models, 2024. URL https://arxiv.org/abs/2403.05812.

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, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

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, 2020. URL https://arxiv.org/abs/2001.08361.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Guha, Sedrick Scott Keh, Kushal Arora, et al. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282, 2024.

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*, 2024.

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Gábor Melis, Tomáš Kočiskỳ, and Phil Blunsom. Mogrifier lstm. *arXiv preprint arXiv:1909.01792*, 2019.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

Eric Michaud, Ziming Liu, Uzay Girit, and Max Tegmark. The quantization model of neural scaling. *Advances in Neural Information Processing Systems*, 36:28699–28722, 2023.

Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. Relu strikes back: Exploiting activation sparsity in large language models. *arXiv preprint arXiv:2310.04564*, 2023.

Tim Pearce and Jinyeop Song. Reconciling kaplan and chinchilla scaling laws. *arXiv preprint arXiv:2406.12907*, 2024.

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.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Nils Reimers and Iryna Gurevych. Optimal hyperparameters for deep lstm-networks for sequence labeling tasks, 2017. URL https://arxiv.org/abs/1707.06799.

Jack Sanderson, Teddy Foley, Spencer Guo, Anqi Qu, and Henry Josephson. Rethinking llm advancement: Compute-dependent and independent paths to progress. *arXiv preprint arXiv:2505.04075*, 2025.

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

Teodora Srećković, Jonas Geiping, and Antonio Orvieto. Is your batch size the problem? revisiting the adam-sgd gap in language modeling. *arXiv preprint arXiv:2506.12543*, 2025.

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.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(10):6761–6774, 2024.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy S Liang, Quoc V Le, Tengyu Ma, and Adams Wei Yu. Doremi: Optimizing data mixtures speeds up language model pretraining. *Advances in Neural Information Processing Systems*, 36:69798–69818, 2023.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pp. 10524–10533. PMLR, 2020.

Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Learning rate scheduling. https://d2l.ai/chapter_optimization/lr-scheduler.html, 2025. “Dive into Deep Learning” (version 1.0.3), accessed 20 September 2025.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Guodong Zhang, Lala Li, Zachary Nado, James Martens, Sushant Sachdeva, George Dahl, Chris Shallue, and Roger B Grosse. Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model. *Advances in neural information processing systems*, 32, 2019.

Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhiquan Luo. Why transformers need adam: A hessian perspective. *Advances in neural information processing systems*, 37:131786–131823, 2024.

Rosie Zhao, Depen Morwani, David Brandfonbrener, Nikhil Vyas, and Sham Kakade. Deconstructing what makes a good optimizer for language models. *arXiv preprint arXiv:2407.07972*, 2024.

11 Appendix

11.1 Model Architecture

The transformer model used tied input and output embeddings.

Table 2: Base hyperparameters used to train the Transformer Model

Hyperparameter	**Value**
Dropout	0.0
Sequence length	128
Stride	128
Batch size	256
Gradient Clipping	1.0
Weight decay	0.01
Learning Rate Schedule	cosine annealing
min learning rate	0.1x max lr
warmup fraction	10%
Initialization	Transformer Scaled
Activation Function	GeLU
SGD momentum	0.9
Default Optimizer	AdamW
Embedding Weight Tying	True
Tokenizer	GPT-2 Tokenizer
Vocabulary	50257
Dataset	C4

Table 3: Base hyperparameters used to train LSTM Model

Hyperparameter	**Value**
Number of layers	2
Sequence length	128
Stride	128
TBPTT length	64
TBPTT stride	64
Batch size	256
input dropout	0.2
hidden dropout	0.1
output dropout	0.2
learning Rate Schedule	cosine annealing
min learning rate	0.1x max lr
warmup fraction	10%
Weight decay	0.01
Gradient clipping	1.0
Initialization	LSTM (Orthogonal Hidden Matrices)
SGD momentum	0.9
Embedding Weight Tying	True
Tokenizer	GPT-2 Tokenizer
Vocabulary	50257
Dataset	C4

11.1.1 Hyperparameter Tuning

We tune the learning rate for the standard transformer, transformer with sgd, standard lstm, and lstm with sgd. We used a standard selection $[10^{-3}, 10^{-2.5}, 10^{-2}, 10^{-1.5}, 10^{-1}]$. We experimented with a wider range of learning rates but learning reates below 10^{-4} performed significantly worse.

11.1.2 Extended Model Discussion

We use standard variational dropout for the LSTM. We set dropout equal to 0 as was done in Chinchilla scaling laws (Hoffmann et al., 2022). We experimented with different learning rate schedules. However, in our experience they had little effect on either the standard transformer, or the transformer with sgd. This is in line with Kaplan et al. (2020).

11.2 Data Selection and Loading

We choose to the C4 dataset for both the transformer and LSTM. Hoffmann et al. (2022) found similar scaling exponents on their full dataset and on exclusively C4.

11.3 How Chinchilla Scaling Laws Change with Algorithm Advantages

12 Ablation and Variation Studies

Throughout this investigation, we have had to compromise between using the very best setups for any given model and keeping a fair comparison between models. Here, we have a table of ablation experiments at small scales to justify the choices we have made.

13 Kaplan-Style Scaling When the True Frontier is Chinchilla-Optimal

Setup. Assume the standard bi-variate loss

$$L(N,D) \;=\; E \;+\; A\,N^{-a} \;+\; B\,D^{-b}, \qquad a,b>0, \tag{1}$$

and training compute

$$C \;\propto\; N_{\text{core}}\,D, \tag{2}$$

where N_{core} excludes embeddings. Consider scaling along a path

$$D \;\propto\; N^{p} \qquad (p>0). \tag{3}$$

Then $C \propto N^{1+p}$, hence

$$N \;\propto\; C^{\frac{1}{1+p}}, \qquad D \;\propto\; C^{\frac{p}{1+p}}. \tag{4}$$

Compute-only decay along a path. Substituting (4) into (1) gives

$$L(C)-E \;\asymp\; A\,C^{-\frac{a}{1+p}} \;+\; B\,C^{-\frac{p\,b}{1+p}}, \tag{5}$$

so the effective compute exponent is

$$\gamma(p) \;=\; \frac{\min\{a,\; p\,b\}}{1+p}, \qquad L(C) \;\approx\; E \;+\; K\,C^{-\gamma(p)}. \tag{6}$$

Chinchilla-optimal path. Maximizing (6) balances the two terms:

$$a = p^{\star}b \;\Rightarrow\; p^{\star} = \frac{a}{b}, \qquad \gamma^{\star} = \gamma(p^{\star}) = \frac{ab}{a+b}. \tag{7}$$

If $a \approx b$, then $p^{\star} \approx 1$ (tokens $\propto$ parameters) and $\gamma^{\star} = \frac{a}{2}$.

Kaplan-style path under Chinchilla reality. Let a Kaplan-style schedule use $D \propto N^{p_{\text{K}}}$ with $p_{\text{K}} < 1$. Then

$$\gamma_{\text{K}} \;=\; \gamma(p_{\text{K}}) \;=\; \frac{\min\{a,\; p_{\text{K}}b\}}{1+p_{\text{K}}}. \tag{8}$$

Under $a \approx b$,

$$\gamma_{\text{K}} \;=\; \frac{p_{\text{K}}\,a}{1+p_{\text{K}}} \;<\; \frac{a}{2} \;=\; \gamma^{\star}, \qquad \frac{\gamma_{\text{K}}}{\gamma^{\star}} \;=\; \frac{2p_{\text{K}}}{1+p_{\text{K}}} < 1. \tag{9}$$

Example: $p_{\text{K}} = 0.74$ yields $\gamma_{\text{K}} \approx 0.74a/1.74 \approx 0.425a$ vs. $\gamma^{\star} = 0.5a$ (about 15% weaker exponent).

Geometry of the mismatch. Along $D \propto N^p$,

$$\frac{D}{N} \;\propto\; N^{p-1} \;\propto\; C^{\frac{p-1}{1+p}}. \tag{10}$$

Thus for $p_{\mathrm{K}} < 1$ the aspect ratio D/N shrinks with compute, increasingly undertraining larger models and yielding the smaller exponent in (9).

Embeddings. Let $N_{\text{total}} = N_{\text{core}} + N_{\text{emb}}$ and write $N_{\text{core}} = (1-s)N_{\text{total}}$ with embedding share $s \in [0, 1)$. Then

$$C \;\propto\; (1-s)\, N_{\text{total}}\, D. \tag{11}$$

Any bounded or slowly varying s rescales C by (asymptotically) a constant and does not affect the exponent $\gamma(p)$ in (6); embeddings change constants, not the compute-scaling exponent.