Is the Number of Trainable Parameters All That Actually Matters?

Amélie Chatelain[1] **Amine Djeghri**[3*] **Daniel Hesslow**[1] **Julien Launay**[1,2]

Iacopo Poli[1]

[1]LightOn [2]LPENS, École Normale Supérieure [3]Sorbonne Université

`{firstname}@lighton.ai`

Abstract

Recent work has identified simple empirical scaling laws for language models, linking compute budget, dataset size, model size, and autoregressive modeling loss. The validity of these simple power laws across orders of magnitude in model scale provides compelling evidence that larger models are also more capable models. However, scaling up models under the constraints of hardware and infrastructure is no easy feat, and rapidly becomes a hard and expensive engineering problem. We investigate ways to tentatively *cheat* scaling laws, and train larger models for cheaper. We emulate an increase in effective parameters, using efficient approximations: either by *doping* the models with frozen random parameters, or by using fast structured transforms in place of dense linear layers. We find that the scaling relationship between test loss and compute depends only on the *actual* number of trainable parameters; scaling laws cannot be deceived by spurious parameters.

1 Introduction

Predictably linking model and dataset size with generalization error is a long-standing open question. Discrepancies between classical bias-variance trade-off models and modern practices have been identified [1], with phenomena such as deep double descent [2] providing glimpses into a deeper understanding. However, actionable insights for machine learning practitioners have remained elusive.

Recently, simple and general empirical scaling laws for deep learning models have been uncovered. Starting with first relationships for modern convolutional networks [3], guidelines informing optimal architectures–such as EfficientNet [4]–have been derived. Importantly, these laws can extrapolate model performance across scale, motivating the training of increasingly large and capable models [5].

We focus on seminal work on large generative language models [6], establishing and using scaling laws to predict the computational requirements for a specific performance level–as pioneered by [7]. This work showed that the performance of an auto-regressive language model follows a remarkably simple relationship, linking model and dataset size, compute budget, and modeling loss across many orders of magnitude in scale. This make it possible to answer questions central to the efficient training of extreme-scale models, such as: *(1) which model size achieves the best loss given a fixed compute budget*; or *(2) how many samples are necessary to train a model of a given size optimally.*

Unfortunately, these empirical laws also show that training models much larger than current ones comes at a prohibitive cost. State-of-the-art language models such as GPT-3 have required several thousand PF-days to train [8], and model size has plateaued since then [9, 10]. Proposed distributed

[*]Work done while at LightOn.

sharding methods for the training of such extreme-scale models only bridge the memory gap [11], and do not fundamentally change the required compute budget. Pending drastic hardware improvements, it is unlikely we will see a 100-1,000x increase in model size as quickly as we have seen in the past.

Accordingly, to enable sustained growth in model size and capabilities, we attempt to *cheat* scaling laws. To do so, we propose to fudge the effective number of parameters in the model. Instead of using only expensive fully trainable parameters, we explore two classes of efficient parameters motivated by potential efficient hardware implementations: either by *doping* the models with cheaper and simpler frozen parameters, or by implementing *structured* transforms in place of dense linear layers. We study these additions under the guise of scaling laws.

Frozen parameters Transformer models with fixed random parameters have previously been studied in Reservoir Transformers [12], where some layers of a RoBERTa model [13] are initialized randomly and then never updated. Using an *area under convergence curve* metric, they find this approach to lead to more efficient training. This is motivated by a broader line of work around random projections in machine learning, with applications in random kernel methods [14], unsupervised feature learning [15], and reservoir computing [16]. In a language model, frozen parameters are one third cheaper than fully trainable ones: assuming the backward pass is normally twice the cost in FLOP of the forward (backpropagation + update) [6], the update step is skipped for frozen parameters. Furthermore, multiplication with random matrices can be offloaded to specialized co-processors [17].

Structured parameters Structured efficient linear layer parametrizations have been proposed to accelerate fully-connected layers [18, 19, 20]. They leverage transforms with an efficient $\mathcal{O}(n \log n)$ implementation, such as the Hadamard Transform, Discrete Fourier Transform or the Discrete Cosine Transform. These structured layers have far fewer trainable parameters (on the order of n) but still emulate an operation equivalent to n^2 parameters. We initially hypothesize that the scaling laws will depend on the the emulated number of parameters n^2 rather than n, reducing the memory needs and compute budget for training the model, while maintaining identical modelling loss.

Contributions We study random and structured parameters in large auto-regressive language models, and show that scaling laws **cannot** be cheated using spurious parameters. More specifically, we find that in both cases, the empirical scaling laws obtained depend only on the *real* number of trainable parameters. Neither the frozen or structured parameters make pre-training more efficient.

2 Methods

All experiments are based on the training on a language modeling task of a decoder-only auto-regressive Transformer based on the GPT architecture [21]. Our only significant deviation from the architecture is the use of rotary embeddings [22], which enable increased performance. We performed initial pathfinding experiments without rotary embeddings and obtained similar results. Architecture details for the four model sizes considered are available in the supplementary.

The model is trained on French data obtained using a Common Crawl dump filtered by CCNet [23]. We use byte-level byte-pair encoding, with a vocabulary of 50,262 tokens. Our dataset contains 32 billion tokens, and is similar to the setup of FlauBERT [24]. All validation and test losses are obtained on the `fr-wiki` dataset, made of 0.5 billion tokens obtained from a dump of French Wikipedia.

All experiments were run in a distributed setup with four NVIDIA V100-16GB GPUs per node, on the Jean Zay supercomputer. We use a sharded data parallel strategy for training the larger models, similar to ZeRO-3 [25], and implemented with FairScale [26]. Structured transforms are implemented as optimized CUDA kernels and compiled in PyTorch extensions for best performance.

We estimate the compute budget in PF-day using the formula $C = 6NBS$ [6], where N is the number of trainable non-embedding parameters, B is the number of tokens per batch, S is the number of parameter updates, and 6 is the factor to account for the forward and backward passes. We verified our experimental setup, and in particular the choice of a French dataset in place of an English one, by reproducing the scaling law fit of [6] and obtaining a similar exponent – as shown in Figure 1.

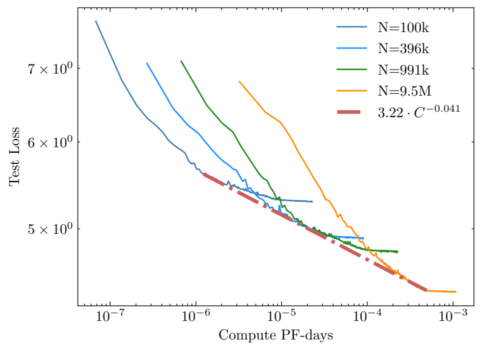

Figure 1: Loss against compute for different model sizes, with the empirical scaling law fit. We highlight that we find an exponent of -0.041, which is remarkably close to the -0.048 of [27], considering that it is obtained on a different dataset in a different language, with a different codebase.

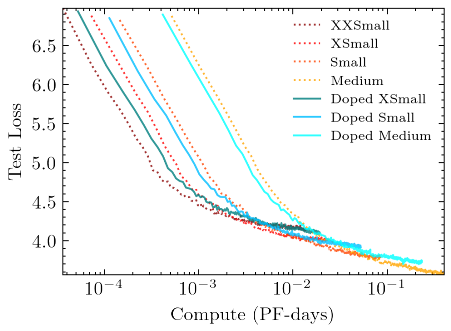

Figure 2: Loss against compute for random parameters costing two thirds as much as trainable ones, as in our implementation.

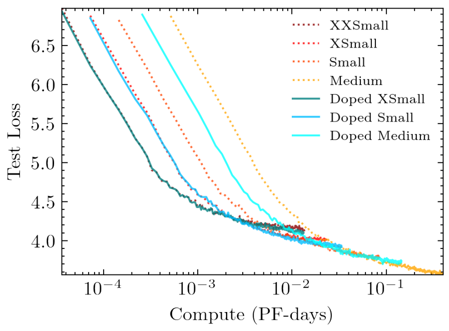

Figure 3: Loss against compute for an ideal random parameter cost of zero, achievable with offloading to dedicated hardware.

3 Doping with Frozen Random Parameters

3.1 Methods

Similar to [12] we replace the decoder layers–consisting of self-attention followed by a multilayer perceptron (MLP)–with a single frozen MLP. Indeed, we observed that keeping the entire frozen decoder layer lead to similar performances as keeping only the frozen MLP. The latter, which involves random matrix multiplications, could more easily be done on a specialized co-processor. In order to approximately preserve the same total number of parameters, we increase the hidden dimension of the feedforward network from $4 \times d_{\text{model}}$ to $6 \times d_{\text{model}}$. By training models with fixed parameters over multiple orders of magnitude, we establish the scaling laws both for the models with fixed random parameters and the baseline models.

3.2 Results

The cost of the random parameters depends on the choice of hardware, as well as any potential algorithm used to approximate them. To be able to compare the models with frozen random parameter with the baseline models, we plot the scaling laws for different costs of the random parameters.

Figure 2 shows scaling laws for random parameters costing two thirds as much as trainable parameters, which corresponds to skipping the parameter update step, and which is effectively achieved by our implementation. The baseline models (dotted lines) significantly outperform the models with random parameters (solid lines). Figure 3 shows the scaling behaviour assuming the best-case scenario of free random parameters: doped models reproduce the training curves of a model with the same amount of trainable parameters and no frozen parameters. We observe the same behavior on different scales

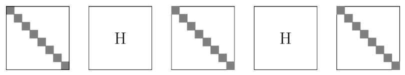

Figure 4: Adaptive FastFood replaces a linear layer by three learnable diagonal matrices and two Hadamard transforms.

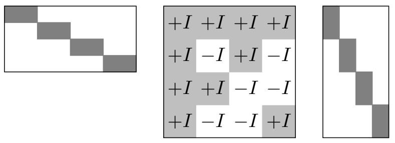

Figure 5: The MLP is replaced by two block diagonal matrices and one block Hadamard transform.

of models. In other words, the frozen parameters have no effect at all on the loss-compute curve of these models, and are effectively useless for pre-training.

4 Structured Efficient Linear Layers

Since frozen layers do not contribute to the loss-compute curve, we turn our attention to structured layers. Structured layers are a natural midpoint between fully frozen and fully trainable layers, implementing a smaller number of *structured* parameters, typically only linearly increasing with layer width, whereas fully-connected layers will increase quadratically. Importantly, and in contrast with sparse layers, structured layers can be efficiently implemented in modern hardware.

4.1 Methods

Random projections can be efficiently approximated by alternating diagonal matrices and Fast Hadamard transformations using FastFood [28], as shown in Figure 4. A vector can then be projected with time complexity $\mathcal{O}(n \log n)$, significantly speeding up the original $\mathcal{O}(n^2)$ operation. Using the approach of [19], we allow for some adaptability of the frozen layers by training the diagonal matrices. This operation dramatically reduces the number of parameters from $\mathcal{O}(n^2)$ down to $\mathcal{O}(n)$. Since in a language model, $n \sim 1000$, we approximate the cost of the structured operation to zero.

We also investigate using a combination of block diagonal matrices and block Hadamard matrices, as in Figure 5. The block diagonal matrices can transform nearby features, while the block Hadamard matrix introduce long-range correlations. The parameter reduction of such a structured MLP is given by the number of blocks, and is tunable. We set the number of blocks to 4, maintaining a relatively large number of trainable parameters in the MLP, with a cost per parameter identical to dense layers.

4.2 Results

In Figure 6, the FastFood structured model is supposed to emulate a *Small* model, but actually performs like an *XSmall* with a similar number of trainable parameters. Furthermore, in Figure 7, the behavior of the block structured model closely resembles that of a model with as many trainable parameters. Accordingly, structured models behave almost identically to fully-trainable models with similar number of trainable parameters, even though both the number of layers and the width of the model is different. The number of spurious parameters does not matter, and only the real one does.

5 Conclusion and outlooks

We showed that random parameters and structured transforms do not provide an advantage over dense fully trainable layers in the loss-compute landscape of autoregressive models. Remarkably, it appears that the scaling law for these models reliably depends *only* on the *actual* number of trainable parameters. Unfortunately, scaling laws are not easily cheated by using spurious parameters. Our investigation points to interesting questions: is the number of trainable parameters all that actually matters? And is this true for any model architecture or specific to autoregressive language models?

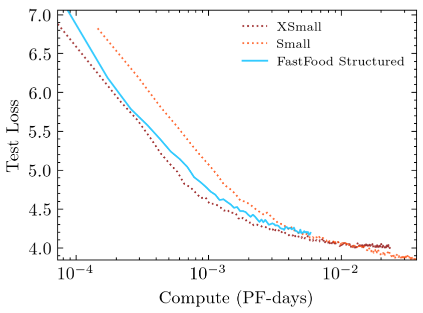

Figure 6: Loss against compute for a model where the linear layers are replaced with adaptive FastFood.

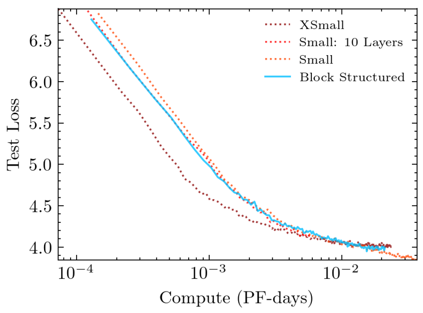

Figure 7: Loss against compute with the MLP replaced by block diagonal matrices and block Hadamard transform.

Acknowledgments and Disclosure of Funding

We thank Gustave Pariente and the LightOn team for useful feedback and support. This work was granted access to the HPC/AI resources of IDRIS under the allocation 2021-A0101012429 made by GENCI. Specifically, we thank the team of the Jean Zay supercomputer for their support.

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 860360

References

[1] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[2] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations*, 2019.

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

[4] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[5] Jonathan S Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. A constructive prediction of the generalization error across scales. *arXiv preprint arXiv:1909.12673*, 2019.

[6] 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. *CoRR*, abs/2001.08361, 2020.

[7] Joel Hestness, Newsha Ardalani, and Gregory Diamos. Beyond human-level accuracy: Computational challenges in deep learning. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, pages 1–14, 2019.

[8] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[9] Wei Zeng, Xiaozhe Ren, Teng Su, Hui Wang, Yi Liao, Zhiwei Wang, Xin Jiang, ZhenZhang Yang, Kaisheng Wang, Xiaoda Zhang, et al. Pangu-α: Large-scale autoregressive pretrained chinese language models with auto-parallel computation. *arXiv preprint arXiv:2104.12369*, 2021.

[10] Boseop Kim, HyoungSeok Kim, Sang-Woo Lee, Gichang Lee, Donghyun Kwak, Dong Hyeon Jeon, Sunghyun Park, Sungju Kim, Seonhoon Kim, Dongpil Seo, et al. What changes can large-scale language models bring? intensive study on hyperclova: Billions-scale korean generative pretrained transformers. *arXiv preprint arXiv:2109.04650*, 2021.

[11] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. *arXiv preprint arXiv:2104.07857*, 2021.

[12] Sheng Shen, Alexei Baevski, Ari S. Morcos, K. Keutzer, Michael Auli, and Douwe Kiela. Reservoir transformer. *ArXiv*, abs/2012.15045, 2020.

[13] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[14] Ali Rahimi, Benjamin Recht, et al. Random features for large-scale kernel machines. In *NIPS*, volume 3, page 5. Citeseer, 2007.

[15] Andrew M Saxe, Pang Wei Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y Ng. On random weights and unsupervised feature learning. In *ICML*, 2011.

[16] Jonathan Dong, Ruben Ohana, Mushegh Rafayelyan, and Florent Krzakala. Reservoir computing meets recurrent kernels and structured transforms. *Advances in Neural Information Processing Systems*, 33, 2020.

[17] Charles Brossollet, Alessandro Cappelli, Igor Carron, Charidimos Chaintoutis, Amélie Chatelain, Laurent Daudet, Sylvain Gigan, Daniel Hesslow, Florent Krzakala, Julien Launay, et al. Lighton optical processing unit: Scaling-up ai and hpc with a non von neumann co-processor. *arXiv preprint arXiv:2107.11814*, 2021.

[18] Marcin Moczulski, Misha Denil, Jeremy Appleyard, and Nando de Freitas. Acdc: A structured efficient linear layer. *arXiv preprint arXiv:1511.05946*, 2015.

[19] Zichao Yang, Marcin Moczulski, Misha Denil, Nando De Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1476–1483, 2015.

[20] Tri Dao, Nimit S Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. *arXiv preprint arXiv:2012.14966*, 2020.

[21] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

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

[23] Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. Ccnet: Extracting high quality monolingual datasets from web crawl data. *arXiv preprint arXiv:1911.00359*, 2019.

[24] Hang Le, Loïc Vial, Jibril Frej, Vincent Segonne, Maximin Coavoux, Benjamin Lecouteux, Alexandre Allauzen, Benoît Crabbé, Laurent Besacier, and Didier Schwab. Flaubert: Unsupervised language model pre-training for french. *arXiv preprint arXiv:1912.05372*, 2019.

[25] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE, 2020.

[26] Mandeep Baines, Shruti Bhosale, Vittorio Caggiano, Naman Goyal, Siddharth Goyal, Myle Ott, Benjamin Lefaudeux, Vitaliy Liptchinsky, Mike Rabbat, Sam Sheiffer, Anjali Sridhar, and Min Xu. Fairscale: A general purpose modular pytorch library for high performance and large scale training. `https://github.com/facebookresearch/fairscale`, 2021.

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

[28] Quoc Le, Tamás Sarlós, Alex Smola, et al. Fastfood-approximating kernel expansions in loglinear time. In *Proceedings of the international conference on machine learning*, volume 85, 2013.

Table 1: Baselines

Hyperparameter	XXSmall	XSmall	Small	Medium
Number of layers	3	6	12	24
d_{model}	768	768	768	1024
Attention heads	12	12	12	16
Context size	1024	1024	1024	1024
Dropout	0.1	0.1	0.1	0.1
Attention dropout	0.1	0.1	0.1	0.1
Total batch size	80	80	80	80
Optimizer	Adam	Adam	Adam	Adam
Peak learning rate	$5e-4$	$5e-4$	$5e-4$	$3e-4$
Learning rate decay	cosine	cosine	cosine	cosine
Warmup tokens	409.6M	409.6M	409.6M	409.6M
Decay tokens	32.8G	32.8G	32.8G	40.9G
Max training tokens	32G	32G	32G	32G
Weight decay	0.0	0.0	0.0	0.0
Gradient clipping	1.0	1.0	1.0	1.0

Table 2: Doped models

Hyperparameter	Doped XSmall	Doped Small	Doped Medium
Total number of layers	5	11	23
Number of trainable layers	3	6	12
Number of frozen layers	2	5	11
d_{model}	768	768	1024
Attention heads	12	12	16
Context size	1024	1024	1024
Dropout	0.1	0.1	0.1
Attention dropout	0.1	0.1	0.1
Total batch size	80	80	80
Optimizer	Adam	Adam	Adam
Peak learning rate	$5e-4$	$5e-4$	$3e-4$
Learning rate decay	cosine	cosine	cosine
Warmup tokens	409.6M	409.6M	409.6M
Decay tokens	32.8G	32.8G	40.9G
Max training tokens	32G	32G	32G
Weight decay	0.0	0.0	0.0
Gradient clipping	1.0	1.0	1.0

A Appendix

We include all the hyperparameter and architectural details for the baselines (Table 1), the doped models (Table 2, and the structured models (Table 3). The doped models are built by alternating trainable transformer and frozen MLP layers, so that the first and last layers of the models are always trainable.

Table 3: Structured Models

Hyperparameter	Structued Adaptive FastFood	Structued Block Diagonal
Number of layers	12	12
d_{model}	1024	1024
Attention heads	16	16
Context size	1024	1024
Dropout	0.1	0.1
Attention dropout	0.1	0.1
Total batch size	80	80
Optimizer	Adam	Adam
Peak learning rate	$5e-4$	$5e-4$
Learning rate decay	cosine	cosine
Warmup tokens	409.6M	409.6M
Decay tokens	32.8G	40.9G
Max training tokens	32G	32G
Weight decay	0.0	0.0
Gradient clipping	1.0	1.0