Optimal Scaling Needs Optimal Norm

author names withheld

Under Review for the Workshop on High-dimensional Learning Dynamics, 2025

Abstract

Despite recent progress in optimal hyperparameter transfer under model and dataset scaling, no unifying explanatory principle has been established. For Adam and Scion optimizers, we discover that *joint* optimal scaling across model and dataset sizes is conditioned on a single invariant: the operator norm of the output layer. Across models with up to 1.3B parameters trained on up to 138B tokens, the optimal learning rate/batch size pair (η^*, B^*) consistently has the same operator norm value — a phenomenon we term *norm transfer*. This constant norm condition is necessary but not sufficient: while for each dataset size, multiple (η, B) reach the optimal norm, only a unique (η^*, B^*) achieves the best loss. As a sufficient condition, we provide the first measurement of (η^*, B^*) scaling with dataset size, finding it consistent with theoretical predictions. We share practical insights on norm-guided optimal scaling and release our Distributed Scion (`Disco`) implementation with logs from over two thousand runs to support research on LLM training dynamics.

1. Introduction

Progress in Large Language Models (LLMs) has been largely driven by scale: enlarging model and training-data size consistently yields more capable systems [24, 29], but at growing computational cost. Achieving *optimal scaling* — configuring hyperparameters optimally as scale grows — is thus essential to push the model frontier further.

Several powerful yet disparate methods address this challenge. Theoretically grounded frameworks like Maximum Update Parametrization (μP) [60] transfer optimal hyperparameters across model scales, while empirical scaling laws [33] provide rules of thumb where theory is absent, e.g. for dataset size scaling. A unifying principle covering *both* model and dataset scaling dimensions, however, remains elusive.

Recently, norm-based optimization [6, 44] has offered a new lens on training dynamics, reframing optimization as control of the operator norms of weights and gradient updates and enabling insights deeper than the loss curve alone. This raises a natural question: **can the norm-based perspective unify optimal model and dataset size scaling?**

By tracking layer norms across thousands of experiments, we answer affirmatively, with the following discoveries:

- **Unifying invariant for optimal scaling.** The operator norm of the output layer $\|W_{\text{out}}\|_{\text{RMS}\to\infty}$ (Definition 1) at the optimal learning rate (η) and batch size (B) is invariant — i.e. "transfers" — across model (via width and depth) and dataset scaling (Fig. 1), for both Scion and Adam optimizers. This *norm transfer* is a *necessary* condition for optimality, but not sufficient: multiple non-optimal (η, B) pairs can reach the same norm value (Fig. 4).

- **Scaling rules for the Scion optimizer.** As a *sufficient* condition, we empirically find $\eta^*(B, D) \propto B^{0.62} \cdot D^{-0.56}$, matching Adam's square-root rule. The optimal batch size scales as $B^*(D) \propto D^{0.45\pm0.07}$, yielding $\eta^*(D) \propto D^{-0.28\pm0.07}$, consistent with theoretical predictions [51]. For fixed D, one can trade $\eta^* \leftrightarrow B^*$ via $\eta \propto \sqrt{B}$ within a low-sensitivity region around the optimal norm (Fig. 3), enabling larger-batch training at no performance cost.
- **Distributed Scion/Muon and experimental logs.** We release `Disco`, a distributed Scion/Muon implementation compatible with modern parallelization strategies, along with norm logs from over two thousand training runs.

2. Preliminaries

In a series of work [5–7, 62] it was shown how by controlling gradient update norms during optimization one can essentially perform the steepest gradient descent under these norms (Appendix B). After being built into optimizers – with prominent examples being Muon [28] and Scion [44] – it yielded not only more stable training, but also improved model performance at scale [49, 55, 58]. Importantly, at its root the approach embeds *zero-shot hyperparameter transfer*: the property of preserving optimal hyperparameters across model scales. This allows to use small (proxy) models for hyperparameter tuning with optimality guarantees of their transfer to larger models [60].

Such norm-based optimization naturally puts matrix norms into the spotlight of model training. Specifically, [7] introduces:

Definition 1 (Induced operator norm[1]) *Given a matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and two normed vector spaces $(\mathbb{R}^{d_{\text{in}}}, \|\cdot\|_\alpha)$ and $(\mathbb{R}^{d_{\text{out}}}, \|\cdot\|_\beta)$, the "$\alpha$ to β" induced operator norm is given by:*

$$\|W\|_{\alpha\to\beta} = \max_{x\in\mathbb{R}^{d_{\text{in}}}} \frac{\|Wx\|_\beta}{\|x\|_\alpha}, \tag{1}$$

where of particular interest for us would be:

$$\|W\|_{\text{RMS}\to\infty} := \max_i d_{\text{in}} \|\text{row}_i(W)\|_{\text{RMS}}, \tag{2}$$

with $\text{row}_i(.)$ denoting the i-th row of a matrix.

While these norms are explicitly constrained for model weights updates via special transformations of gradients (e.g. Equations (9) to (11)), the norm of model weights themselves are not constrained within this approach.

In this work, we start from an observation that keeping track of these norms and analysing patterns therein can provide insights on the model training dynamics and its optimality. Specifically, we are interested in the problem of joint optimal model and data scaling with the main question: how do optimal hyperparameter configurations relate to each other at different scales?

To answer this, we train multiple Llama 3 models (see more details on our training setup in Appendix C) on text up to a token horizon (i.e. total number of training tokens) of 138B; starting from 69M model size and scaling up to $\times$12 in width (to 1.3B parameters), up to $\times$32 in depth (to 168M). Notably, we employ a `norm-everywhere` approach, inspired by the concept of well-normedness in [32], where the input to every linear layer is normalized. For efficient training we developed a distributed version of the Scion optimizer (`Disco`), as described in Appendix E.

For each of the model and data scales, we vary global learning rate η (shared across layer groups; per-layer-group tuning is analyzed in Appendix K) and batch size B, and analyze model weight norms following our methodology in Appendix F. In essence, for each scale we perform statistical fits and extract optimal norm and hyperparameter values, thus tracking the weight norm dynamics along the optimal scaling trajectory. While in principle one could track weight norms of any layer, we focus on the output layer norm $\|W_{\text{out}}\|_{\text{RMS}\to\infty}$ as the most natural choice.[2] We ablate this choice for other layers and norms in Appendix T, finding it not affecting our main findings.

3. Results

3.1. Output norm dynamics

First of all, learning rate scans reveal an optimal output norm for each batch size and horizon, with the norm increasing monotonically with η (Fig. 2). Since we use unconstrained Scion (without weight decay), norms grow with gradient steps (Appendix J). We also show that they can be bounded via weight decay (Appendix M) or spectral clipping [42]. Strikingly, this growth is *piecewise linear* in log-log scale: across all batch sizes, the slope changes abruptly near norm values of 2^6–2^7 and again at 2^9–2^{10}, beyond which the dynamics enter a "turbulence" regime – possibly the same phenomenon recently observed in loss curves [41]. Finally, η sets the curves' offset while B controls their "decoupling": curves with equal η but different B coincide early on, then split at the 2^6–2^7 transition more sharply for larger B, before reconverging to a common slope.

3.2. Optimal norm transfer

We summarise learning rate scans across batch sizes, horizons and model widths/depths in Fig. 1 (extended plots in Appendix U and Appendix V). Each point uses the optimal η^* for its batch size, horizon and model. We report observations per scaling direction together with ablations.

Data scaling: For each horizon, a single optimal batch size yields an optimal output norm $\|W_{\text{out}}\|_{\text{RMS}\to\infty} = 2^{7.0\pm0.2}$, and this value transfers across horizons. We call this *norm transfer*: the optimal (η, B) pair for any horizon necessarily hits $\approx 2^7$. The optimal batch size grows with the horizon, which we revisit in Sec. 3.3. The same behavior holds when changing the dataset, as we show on the Thai and Russian Fineweb-2 partitions [43] (Appendix N).

Model width scaling: The spectral condition (Eq. 2) is designed to preserve the optimal norm. Accordingly, scaling width by $\times 12$ at fixed horizon (Fig. 1(*b*)) yields nested "μP-style" curves with a shared optimal norm and decreasing loss.

Model depth scaling: Surprisingly, scaling depth by $\times 32$ in our setup also preserves the optimal norm, despite not using any established depth-transfer techniques [9, 15, 61]. In Appendix O we show that all such techniques induce learning rate transfer, but our choice (no residual scaling factors, pre-residual layers initialised with a $1/\sqrt{2N_{\text{layers}}}$ rescaling) gives the lowest loss. We attribute this to the uniform norm treatment of our `norm-everywhere` approach (Appendix C).

Momentum & learning rate decay: Both Scion with non-zero momentum and a decaying schedule – more practically relevant – also exhibit norm transfer (Appendix S). Momentum reduces sensitivity to batch size, with multiple B reaching the same optimal norm and loss (Fig. 15); learning rate decay analogously reduces sensitivity to η (Fig. 19(*b*)).

2. It is invariant to both width and depth scaling, it is the most sensitive to learning rate tuning (Appendix K), and it can be viewed as a linear classifier on the learned hidden representations – making it "representative" for the entire model.

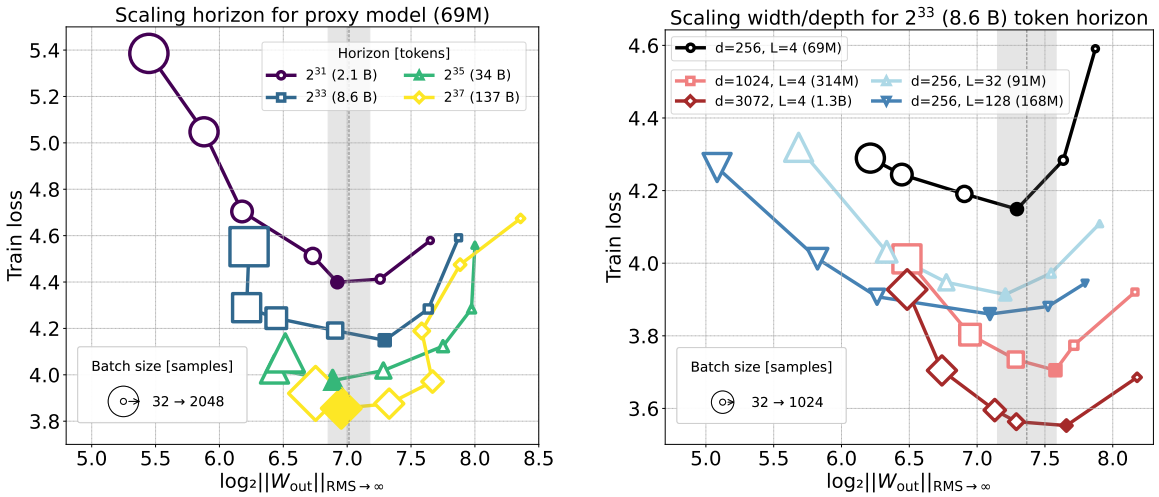


Figure 1: **Training loss vs. output layer norm across batch sizes. (a)** Fixed proxy model (69M parameters) while increasing token horizon from 2^{31} to 2^{37}. **(b)** Fixed token horizon 2^{33} while scaling width/depth of the proxy model as indicated in the legend. Each batch size point (increasing from 32 in $\times 2$ steps, reflected by marker size) has its learning rate optimally tuned. The optimal batch size per horizon/model configuration is indicated by the filled marker. All curves share optimal norm at 7.0 ± 0.2 across horizons and 7.4 ± 0.2 across models (grey band).

Adam optimizer: For the proxy model we run Adam with momenta ($\beta_1 = 0.9, \beta_2 = 0.95$) and without ($\beta_1 = \beta_2 = 0$, sign gradient descent). Both show clear norm transfer (Appendix P): without momentum the optimal norm matches the without-momentum Scion ($\approx 2^7$), while with momentum it shifts to $\approx 2^{11}$. This supports the common norm-based view of optimization [7] and suggests our Scion observations carry over to Adam.

Choice of norm and layer: While we focus on the output layer's $\|.\|_{\text{RMS}\to\infty}$ norm (Sec. 2), one could equally monitor other layers or operator norms. Substituting it with $\|W_{\text{out}}\|_{\text{RMS}\to\text{RMS}}$ or with the input-layer norm $\|W_{\text{in}}\|_{1\to\text{RMS}}$ also yields norm transfer across horizons (Appendix T), indicating that the phenomenon is not specific to a particular (layer, norm) pair.

Normalization layers: Firstly, we ablate placements of `RMSNorm` (without trainable parameters) for the proxy model (Appendix R). With Scion (momentum $\alpha = 0.1$), removing residual normalization causes divergence (Fig. 14(*b*)). QK-norm + residuals + output layer norm matches the `norm-everywhere` learning rate profile, showing MLP- and VO-norm are redundant. Residuals + output is slightly better but more η-sensitive, while QK-norm reduces η-sensitivity, consistent with [59].

Therefore, for further scaling ablations we pick the residuals-only configuration. Fig. 13 shows norm transfer still holds, but at a markedly lower optimal norm ($\approx 2^3$, one order of magnitude below `norm-everywhere`); legends in Fig. 14(*b*) pinpoint the output layer norm as the source of this drop. While we do not use this configuration for the main results, we leave it as an interesting open question for future work, whether lower-norm models are preferable [42].

3.3. Optimal (η, B) scaling rule

The norm guidance leaves open how to pick (η, B) for a given horizon; here we ask what is the *sufficient* condition for optimality. Fig. 4 shows that the optimal-norm condition is necessary but not sufficient: for each horizon, every batch size reaches the optimal-norm region $\|W_{\text{out}}\|_{\text{RMS}\to\infty} \in [2^{6.8}, 2^{7.2}]$ at a sufficiently high η. Therefore, we unfold Fig. 1(*a*) to expose the optimal η^* that was profiled away, and study how it varies with B at fixed D and across horizons. Fig. 3 shows $\log_2 \eta^*(B, D) = \alpha \log_2 B + \beta \log_2 D + \gamma$ fit results (circled markers are per-horizon optima). We observe several things:

- The fit gives $\alpha = 0.62 \pm 0.05$, $\beta = -0.56 \pm 0.05$, consistent with the established square-root scaling in B [38] and D [8] for Adam. Like [2, 48], we see no surge phenomenon [34], i.e. no transition from $\eta^* \propto \sqrt{B}$ to $\eta^* \propto 1/\sqrt{B}$ above a critical batch size [63]; [27] accounts for this via mean-field theory.
- Different B yield different losses, with a horizon-dependent optimum $B^*(D)$ (circled markers, transparency for relative loss). B^* grows with D: on an extended horizon set we measure $B^*(D) \propto D^{0.45\pm0.07}$ (Appendix L), consistent with Adam [4, 33] and recent theoretical predictions for $\sqrt{D}$ scaling [51]. Combining $B^*(D) \propto D^{0.45}$ with the fit gives $\eta^*(D) \propto D^{-0.28\pm0.07}$, again consistent with $D^{-1/4}$ predictions [51].
- Existence of a single optimal $B^*(D)$ per horizon caps the usable device count for pretraining: beyond it, more devices either hurt throughput (smaller per-device microbatch) or loss (leaving the optimal-B region) – suggesting an open question of whether this limit can be bypassed.
- In practice, (η^*, B^*) extends to an *optimal region* $(\eta^* \pm \Delta\eta, B^* \pm \Delta B)$ of near-optimal runs (opacity in Fig. 3). We connect this to learning-rate sensitivity [59], here rephrased as *norm sensitivity*, set by the flatness of the horizon curve (Fig. 1(*a*)) around the optimal norm. Within it, one can trade η for B via $\eta \propto \sqrt{B}$, allowing flexibility such as training at larger batch sizes.

4. Conclusion and Discussion

We demonstrate that norm analysis of model weights is a powerful measure guiding joint optimal scaling across both model and dataset dimensions. Empirically, we (1) study how layer norms evolve with the choice of (η, B, D) and how to tune them to desired values (Sec. 3.1); (2) show that an optimal hyperparameter configuration must hit a predefined output layer norm to be transferable across data and model scales (Sec. 3.2); and (3) derive optimal scaling rules $\eta^*(D), B^*(D)$ that yield optimal loss (Sec. 3.3).

While our empirical observations are consistent with theoretical predictions [51], we still don't yet know how they connect to our main finding: optimal trajectories in the (data, model) scaling plane must remain on a constant-norm *manifold* [5]. This raises further questions:

- What keeps the optimal scaling trajectory on the constant-norm manifold? What is its structure?
- How can the constant-norm condition be leveraged as an emerging inductive bias to optimize training?

We don't yet have answers, but believe our study scratches the surface of exciting phenomena to be further understood.

References

[1] Kwangjun Ahn, Byron Xu, Natalie Abreu, Ying Fan, Gagik Magakyan, Pratyusha Sharma, Zheng Zhan, and John Langford. Dion: Distributed orthonormalized updates. *arXiv preprint arXiv:2504.05295*, 2025.

[2] Essential AI, :, Ishaan Shah, Anthony M. Polloreno, Karl Stratos, Philip Monk, Adarsh Chaluvaraju, Andrew Hojel, Andrew Ma, Anil Thomas, Ashish Tanwer, Darsh J Shah, Khoi Nguyen, Kurt Smith, Michael Callahan, Michael Pust, Mohit Parmar, Peter Rushton, Platon Mazarakis, Ritvik Kapila, Saurabh Srivastava, Somanshu Singla, Tim Romanski, Yash Vanjani, and Ashish Vaswani. Practical efficiency of muon for pretraining. *arXiv preprint arXiv:2505.02222*, 2025.

[3] Noah Amsel, David Persson, Christopher Musco, and Robert M. Gower. The polar express: Optimal matrix sign methods and their application to the muon algorithm, 2025. URL https://arxiv.org/abs/2505.16932.

[4] Shane Bergsma, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Power lines: Scaling laws for weight decay and batch size in llm pre-training. *arXiv preprint arXiv:2505.13738*, 2025.

[5] Jeremy Bernstein. Modular manifolds. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250926. https://thinkingmachines.ai/blog/modular-manifolds/.

[6] Jeremy Bernstein and Laker Newhouse. Modular duality in deep learning. *arXiv preprint arXiv:2410.21265*, 2024.

[7] Jeremy Bernstein and Laker Newhouse. Old optimizer, new norm: An anthology. *arXiv preprint arXiv:2409.20325*, 2024.

[8] Johan Bjorck, Alon Benhaim, Vishrav Chaudhary, Furu Wei, and Xia Song. Scaling optimal lr across token horizons. *arXiv preprint arXiv:2409.19913*, 2025.

[9] Blake Bordelon, Lorenzo Noci, Mufan Bill Li, Boris Hanin, and Cengiz Pehlevan. Depthwise hyperparameter transfer in residual networks: Dynamics and scaling limit. *arXiv preprint arXiv:2309.16620*, 2023.

[10] Franz Louis Cesista. Muon and a selective survey on Steepest Descent in Riemannian and non-Riemannian Manifolds, April 2025. URL http://leloykun.github.io/ponder/steepest-descent-non-riemannian/.

[11] Enea Monzio Compagnoni, Tianlin Liu, Rustem Islamov, Frank Norbert Proske, Antonio Orvieto, and Aurelien Lucchi. Adaptive methods through the lens of sdes: Theoretical insights on the role of noise. *arXiv preprint arXiv:2411.15958*, 2024.

[12] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[13] DeepSeek-AI, :, Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, Huazuo Gao, Kaige Gao, Wenjun Gao, Ruiqi Ge, Kang Guan, Daya Guo, Jianzhong Guo, Guangbo Hao, Zhewen Hao, Ying He, Wenjie

Hu, Panpan Huang, Erhang Li, Guowei Li, Jiashi Li, Yao Li, Y. K. Li, Wenfeng Liang, Fangyun Lin, A. X. Liu, Bo Liu, Wen Liu, Xiaodong Liu, Xin Liu, Yiyuan Liu, Haoyu Lu, Shanghao Lu, Fuli Luo, Shirong Ma, Xiaotao Nie, Tian Pei, Yishi Piao, Junjie Qiu, Hui Qu, Tongzheng Ren, Zehui Ren, Chong Ruan, Zhangli Sha, Zhihong Shao, Junxiao Song, Xuecheng Su, Jingxiang Sun, Yaofeng Sun, Minghui Tang, Bingxuan Wang, Peiyi Wang, Shiyu Wang, Yaohui Wang, Yongji Wang, Tong Wu, Y. Wu, Xin Xie, Zhenda Xie, Ziwei Xie, Yiliang Xiong, Hanwei Xu, R. X. Xu, Yanhong Xu, Dejian Yang, Yuxiang You, Shuiping Yu, Xingkai Yu, B. Zhang, Haowei Zhang, Lecong Zhang, Liyue Zhang, Mingchuan Zhang, Minghua Zhang, Wentao Zhang, Yichao Zhang, Chenggang Zhao, Yao Zhao, Shangyan Zhou, Shunfeng Zhou, Qihao Zhu, and Yuheng Zou. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

[14] Nolan Dey, Quentin Anthony, and Joel Hestness. The practitioner's guide to the maximal update parameterization, Sep 2024. URL https://www.cerebras.ai/blog/the-practitioners-guide-to-the-maximal-update-parameterization.

[15] 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.

[16] Katie Everett, Lechao Xiao, Mitchell Wortsman, Alexander A. Alemi, Roman Novak, Peter J. Liu, Izzeddin Gur, Jascha Sohl-Dickstein, Leslie Pack Kaelbling, Jaehoon Lee, and Jeffrey Pennington. Scaling exponents across parameterizations and optimizers. *arXiv preprint arXiv:2407.05872*, 2024.

[17] Wei Feng, Will Constable, and Yifan Mao. Getting started with fully sharded data parallel (fsdp2), 2025. URL https://docs.pytorch.org/tutorials/intermediate/FSDP_tutorial.html.

[18] Oleg Filatov, Jan Ebert, Jiangtao Wang, and Stefan Kesselheim. Time transfer: On optimal learning rate and batch size in the infinite data limit. *arXiv preprint arXiv:2410.05838*, 2025.

[19] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2018.

[20] Diego Granziol, Stefan Zohren, and Stephen Roberts. Learning rates as a function of batch size: A random matrix theory approach to neural network training. *arXiv preprint arXiv:2006.09092*, 2021.

[21] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary,

Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin

Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[22] Tom Gunter, Zirui Wang, Chong Wang, Ruoming Pang, Andy Narayanan, Aonan Zhang, Bowen Zhang, Chen Chen, Chung-Cheng Chiu, David Qiu, Deepak Gopinath, Dian Ang Yap,

Dong Yin, Feng Nan, Floris Weers, Guoli Yin, Haoshuo Huang, Jianyu Wang, Jiarui Lu, John Peebles, Ke Ye, Mark Lee, Nan Du, Qibin Chen, Quentin Keunebroek, Sam Wiseman, Syd Evans, Tao Lei, Vivek Rathod, Xiang Kong, Xianzhi Du, Yanghao Li, Yongqiang Wang, Yuan Gao, Zaid Ahmed, Zhaoyang Xu, Zhiyun Lu, Al Rashid, Albin Madappally Jose, Alec Doane, Alfredo Bencomo, Allison Vanderby, Andrew Hansen, Ankur Jain, Anupama Mann Anupama, Areeba Kamal, Bugu Wu, Carolina Brum, Charlie Maalouf, Chinguun Erdenebileg, Chris Dulhanty, Dominik Moritz, Doug Kang, Eduardo Jimenez, Evan Ladd, Fangping Shi, Felix Bai, Frank Chu, Fred Hohman, Hadas Kotek, Hannah Gillis Coleman, Jane Li, Jeffrey Bigham, Jeffery Cao, Jeff Lai, Jessica Cheung, Jiulong Shan, Joe Zhou, John Li, Jun Qin, Karanjeet Singh, Karla Vega, Kelvin Zou, Laura Heckman, Lauren Gardiner, Margit Bowler, Maria Cordell, Meng Cao, Nicole Hay, Nilesh Shahdadpuri, Otto Godwin, Pranay Dighe, Pushyami Rachapudi, Ramsey Tantawi, Roman Frigg, Sam Davarnia, Sanskruti Shah, Saptarshi Guha, Sasha Sirovica, Shen Ma, Shuang Ma, Simon Wang, Sulgi Kim, Suma Jayaram, Vaishaal Shankar, Varsha Paidi, Vivek Kumar, Xin Wang, Xin Zheng, Walker Cheng, Yael Shrager, Yang Ye, Yasu Tanaka, Yihao Guo, Yunsong Meng, Zhao Tang Luo, Zhi Ouyang, Alp Aygar, Alvin Wan, Andrew Walkingshaw, Andy Narayanan, Antonie Lin, Arsalan Farooq, Brent Ramerth, Colorado Reed, Chris Bartels, Chris Chaney, David Riazati, Eric Liang Yang, Erin Feldman, Gabriel Hochstrasser, Guillaume Seguin, Irina Belousova, Joris Pelemans, Karen Yang, Keivan Alizadeh Vahid, Liangliang Cao, Mahyar Najibi, Marco Zuliani, Max Horton, Minsik Cho, Nikhil Bhendawade, Patrick Dong, Piotr Maj, Pulkit Agrawal, Qi Shan, Qichen Fu, Regan Poston, Sam Xu, Shuangning Liu, Sushma Rao, Tashweena Heeramun, Thomas Merth, Uday Rayala, Victor Cui, Vivek Rangarajan Sridhar, Wencong Zhang, Wenqi Zhang, Wentao Wu, Xingyu Zhou, Xinwen Liu, Yang Zhao, Yin Xia, Zhile Ren, and Zhongzheng Ren. Apple intelligence foundation language models. *arXiv preprint arXiv:2407.21075*, 2024.

[23] Jacob Hilton, Karl Cobbe, and John Schulman. Batch size-invariance for policy optimization. *arXiv preprint arXiv:2110.00641*, 2022.

[24] 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. *arXiv preprint arXiv:2203.15556*, 2022.

[25] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, Xinrong Zhang, Zheng Leng Thai, Kaihuo Zhang, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

[26] Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro Von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. *arXiv preprint arXiv:2405.18392*, 2024.

[27] Su Jianlin. Rethinking learning rate and batch size (part 3): Muon, Sep 2025. URL https://kexue.fm/archives/11285.

[28] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

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

[30] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2017.

[31] Jeonghoon Kim, Byeongchan Lee, Cheonbok Park, Yeontaek Oh, Beomjun Kim, Taehwan Yoo, Seongjin Shin, Dongyoon Han, Jinwoo Shin, and Kang Min Yoo. Peri-ln: Revisiting normalization layer in the transformer architecture. *arXiv preprint arXiv:2502.02732*, 2025.

[32] Tim Large, Yang Liu, Minyoung Huh, Hyojin Bahng, Phillip Isola, and Jeremy Bernstein. Scalable optimization in the modular norm. *arXiv preprint arXiv:2405.14813*, 2024.

[33] Houyi Li, Wenzhen Zheng, Qiufeng Wang, Hanshan Zhang, Zili Wang, Shijie Xuyang, Yuantao Fan, Zhenyu Ding, Haoying Wang, Ning Ding, Shuigeng Zhou, Xiangyu Zhang, and Daxin Jiang. Predictable scale: Part i – optimal hyperparameter scaling law in large language model pretraining. *arXiv preprint arXiv:2503.04715*, 2025.

[34] Shuaipeng Li, Penghao Zhao, Hailin Zhang, Xingwu Sun, Hao Wu, Dian Jiao, Weiyan Wang, Chengjun Liu, Zheng Fang, Jinbao Xue, Yangyu Tao, Bin Cui, and Di Wang. Surge phenomenon in optimal learning rate and batch size scaling. *arXiv preprint arXiv:2405.14578*, 2024.

[35] Wanchao Liang, Tianyu Liu, Less Wright, Will Constable, Andrew Gu, Chien-Chin Huang, Iris Zhang, Wei Feng, Howard Huang, Junjie Wang, et al. Torchtitan: One-stop pytorch native solution for production ready llm pretraining. In *The Thirteenth International Conference on Learning Representations*, 2025.

[36] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

[37] Ilya Loshchilov, Cheng-Ping Hsieh, Simeng Sun, and Boris Ginsburg. ngpt: Normalized transformer with representation learning on the hypersphere. *arXiv preprint arXiv:2410.01131*, 2025.

[38] Sadhika Malladi, Kaifeng Lyu, Abhishek Panigrahi, and Sanjeev Arora. On the sdes and scaling rules for adaptive gradient algorithms. *arXiv preprint arXiv:2205.10287*, 2024.

[39] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

[40] Meta AI. Introducing llama 4: Advancing multimodal intelligence, 2025. URL https://ai.meta.com/blog/llama-4-multimodal-intelligence/.

[41] Andrei Mircea, Supriyo Chakraborty, Nima Chitsazan, Milind Naphade, Sambit Sahu, Irina Rish, and Ekaterina Lobacheva. Training dynamics underlying language model scaling laws: Loss deceleration and zero-sum learning. *arXiv preprint arXiv:2506.05447*, 2025.

[42] Laker Newhouse, R. Preston Hess, Franz Cesista, Andrii Zahorodnii, Jeremy Bernstein, and Phillip Isola. Training transformers with enforced lipschitz constants. *arXiv preprint arXiv:2507.13338*, 2025.

[43] Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro Von Werra, and Thomas Wolf. Fineweb2: One pipeline to scale them all–adapting pre-training data processing to every language. *arXiv preprint arXiv:2506.20920*, 2025.

[44] Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. Training deep learning models with norm-constrained lmos. *arXiv preprint arXiv:2502.07529*, 2025.

[45] Thomas Pethick, Wanyun Xie, Mete Erdogan, Kimon Antonakopoulos, Tony Silveti-Falls, and Volkan Cevher. Generalized gradient norm clipping & non-euclidean (l_0, l_1)-smoothness. *arXiv preprint arXiv:2506.01913*, 2025.

[46] Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models. *arXiv preprint arXiv:2406.19146*, 2025.

[47] Artem Riabinin, Egor Shulgin, Kaja Gruntkowska, and Peter Richtárik. Gluon: Making muon & scion great again! (bridging theory and practice of lmo-based optimizers for llms). *arXiv preprint arXiv:2505.13416*, 2025.

[48] Naoki Sato, Hiroki Naganuma, and Hideaki Iiduka. Convergence bound and critical batch size of muon optimizer. *arXiv preprint arXiv:2507.01598*, 2025.

[49] Andrei Semenov, Matteo Pagliardini, and Martin Jaggi. Benchmarking optimizers for large language model pretraining. *arXiv preprint arXiv:2509.01440*, 2025.

[50] Yikang Shen, Matthew Stallone, Mayank Mishra, Gaoyuan Zhang, Shawn Tan, Aditya Prasad, Adriana Meza Soria, David D. Cox, and Rameswar Panda. Power scheduler: A batch size and token number agnostic learning rate scheduler. *arXiv preprint arXiv:2408.13359*, 2024.

[51] Egor Shulgin, Dimitri von Rütte, Tianyue H Zhang, Niccolò Ajroldi, Bernhard Schölkopf, and Antonio Orvieto. Deriving hyperparameter scaling laws via modern optimization theory. *arXiv preprint arXiv:2603.15958*, 2026.

[52] Samuel L. Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent. *arXiv preprint arXiv:1710.06451*, 2018.

[53] Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset. *arXiv preprint arXiv:2412.02595*, 2025.

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

[55] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

[56] Maksim Velikanov, Ilyas Chahed, Jingwei Zuo, Dhia Eddine Rhaiem, Younes Belkada, and Hakim Hacid. Learnable multipliers: Freeing the scale of language model matrix layers. *arXiv preprint arXiv:2601.04890*, 2026.

[57] Shuche Wang, Fengzhuo Zhang, Jiaxiang Li, Cunxiao Du, Chao Du, Tianyu Pang, Zhuoran Yang, Mingyi Hong, and Vincent YF Tan. Muon outperforms adam in tail-end associative memory learning. *arXiv preprint arXiv:2509.26030*, 2025.

[58] Kaiyue Wen, David Hall, Tengyu Ma, and Percy Liang. Fantastic pretraining optimizers and where to find them. *arXiv preprint arXiv:2509.02046*, 2025.

[59] Mitchell Wortsman, Peter J. Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D. Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale transformer training instabilities. *arXiv preprint arXiv:2309.14322*, 2023.

[60] Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

[61] Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs vi: Feature learning in infinite-depth neural networks. *arXiv preprint arXiv:2310.02244*, 2023.

[62] Greg Yang, James B. Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2024.

[63] Hanlin Zhang, Depen Morwani, Nikhil Vyas, Jingfeng Wu, Difan Zou, Udaya Ghai, Dean Foster, and Sham Kakade. How does critical batch size scale in pre-training? *arXiv preprint arXiv:2410.21676*, 2025.

[64] Jingwei Zuo, Maksim Velikanov, Ilyas Chahed, Younes Belkada, Dhia Eddine Rhayem, Guillaume Kunsch, Hakim Hacid, Hamza Yous, Brahim Farhat, Ibrahim Khadraoui, Mugariya Farooq, Giulia Campesan, Ruxandra Cojocaru, Yasser Djilali, Shi Hu, Iheb Chaabane, Puneesh Khanna, Mohamed El Amine Seddik, Ngoc Dung Huynh, Phuc Le Khac, Leen AlQadi, Billel Mokeddem, Mohamed Chami, Abdalgader Abubaker, Mikhail Lubinets, Kacper Piskorski, and Slim Frikha. Falcon-h1: A family of hybrid-head language models redefining efficiency and performance. *arXiv preprint arXiv:2507.22448*, 2025.

Appendix A. Related Work

Hyperparameters with model scaling [60] showed how to transfer optimal hyperparameters from a small to a large model in a principled way via Maximal Update Parametrization (μP). [16] later showed that such transfer is also possible in other parametrizations. [15, 61] extended the method towards model scaling in depth. Empirically, scaling laws on how to set optimal hyperparameters as a function of compute [13], loss [25] or model size [46] were measured.

Hyperparameters with data scaling Remains poorly understood theoretically: [52] showed for SGD how to adjust learning rate and batch size by modelling optimization trajectory as a stochastic differential equation (SDE). Largely, the problem has been approached experimentally by measuring hyperparameter scaling rules as a function of the dataset size [4, 18, 25, 33, 50].

(η, B) **scaling rules** Historically, studies of interaction between learning rate and batch size emerged as an experimental effort to scale batch size without losing performance [19, 23, 30]. Later, a deeper understanding has been built from various theoretical angles: SDE [11, 38], loss curvature [39], random matrix theory [20].

Norm-based optimization Starting from the spectral condition [62], the approach of transforming gradient updates based on norm assumptions was fully established in [6, 32], and recently explored in constraining weights themselves [42]. The steepest descent view allowed for connections with manifold learning [10] and optimizer design [47]. This line of work has led to Muon [28] and Scion [44, 45], along with improvements [1, 3], and benchmarks [49, 58] thereof.

Appendix B. Overview of norm-based optimization

Recently, a fundamental shift in the field of optimal scaling occurred with the work of [62]. It changed the focus from model parametrizations towards the norm perspective by showing that Maximum Update Parametrization (μP) [60] can be derived from a more fundamental principle: enforcing a *spectral condition* on the model weights and their updates during the training. We briefly explain the idea behind these concepts below.

μP introduces theoretically grounded scaling rules for hyperparameters as a function of model width in order to ensure "maximal" feature learning in the infinite width limit. This way, the model is guaranteed to learn meaningful features while remaining stable as one scales up its size. As an important by-product, it was found that models with different widths, once parameterized within μP, all share the same optimal hyperparameters (e.g. learning rate) — therefore allowing for what is known as *zero-shot hyperparameter transfer*. This property has been extensively used for the past years to ensure optimal model scaling by tuning hyperparameters for a small (proxy) model, and then effortlessly transferring them to a larger one [14, 22, 40, 64].

In turn, the spectral condition specifies bounds on the norms of weights and weight updates that are necessary to ensure feature learning. More formally:

Definition 2 (Spectral condition) *Consider applying a gradient update $\Delta W_\ell \in \mathbb{R}^{d^\ell_{\text{out}} \times d^\ell_{\text{in}}}$ to the ℓth weight matrix $W_\ell \in \mathbb{R}^{d^\ell_{\text{out}} \times d^\ell_{\text{in}}}$ for a layer $\ell = 1, \ldots, L$. The spectral norms of these matrices should satisfy*

$$\|W_\ell\|_* = \Theta\left(\sqrt{\frac{d^\ell_{\text{out}}}{d^\ell_{\text{in}}}}\right) \qquad \text{and} \qquad \|\Delta W_\ell\|_* = \Theta\left(\sqrt{\frac{d^\ell_{\text{out}}}{d^\ell_{\text{in}}}}\right), \tag{3}$$

where $\|W\|_*$ is the spectral norm, also equal to the largest singular value of W, and $\|x\|_{\text{RMS}} = \|x\|_2/\sqrt{d}$. The symbol Θ is employed following the "Big-O" notation, indicating scaling behaviour (in this case, "constant"[3]) w.r.t. infinite width limit $d \to +\infty$. If conditions in Definition 2 are met, the zero-shot hyperparameter transfer is guaranteed and the model is being trained in the μP regime.

Let us rewrite Definition 2 in a more "natural" way as:

$$\|W_\ell\|_{\text{RMS}\to\text{RMS}} = \Theta(1) \qquad \text{and} \qquad \|\Delta W_\ell\|_{\text{RMS}\to\text{RMS}} = \Theta(1), \tag{4}$$

where we follow [32] and introduce the core concept of this work:

Definition 3 (Induced operator norm) *Given a matrix $W \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$ and two normed vector spaces $(\mathbb{R}^{d_{\text{in}}}, \|\cdot\|_\alpha)$ and $(\mathbb{R}^{d_{\text{out}}}, \|\cdot\|_\beta)$, the "$\alpha$ to β" induced operator norm is given by:*

$$\|W\|_{\alpha\to\beta} = \max_{x\in\mathbb{R}^{d_{\text{in}}}} \frac{\|Wx\|_\beta}{\|x\|_\alpha}. \tag{5}$$

The operator norms we are most interested in will be:

$$\|W\|_{1\to\text{RMS}} \coloneqq \max_j \|\text{col}_j(W)\|_{\text{RMS}}\,, \tag{6}$$

$$\|W\|_{\text{RMS}\to\text{RMS}} \coloneqq \sqrt{d_{\text{in}}/d_{\text{out}}}\,\|W\|_*\,, \tag{7}$$

$$\|W\|_{\text{RMS}\to\infty} \coloneqq \max_i d_{\text{in}}\,\|\text{row}_i(W)\|_{\text{RMS}}\,, \tag{8}$$

where $\text{row}_i(.)$ and $\text{col}_j(.)$ denote the i-th row and j-th column of a matrix. In order to control the operator norms, [6] derived *duality maps*, i.e. transformation rules of the gradients induced by a given norm. Applying these transformations not only keeps the gradient updates within the required bound (e.g. Eq. 4), but also ensures the steepest descent under the chosen norm [7]. For the norms in Equations (6) to (8), the corresponding duality maps for the gradient G with singular value decomposition (SVD) $G = U\Sigma V^\top$ are:

$$\|.\|_{1\to\text{RMS}}: \quad \text{col}_j(G) \mapsto \frac{\text{col}_j(G)}{\|\text{col}_j(G)\|_{\text{RMS}}} \tag{9}$$

$$\|.\|_{\text{RMS}\to\text{RMS}}: \quad G \mapsto \sqrt{\tfrac{d_{\text{out}}}{d_{\text{in}}}} \times UV^\top \tag{10}$$

$$\|.\|_{\text{RMS}\to\infty}: \quad \text{row}_i(G) \mapsto \frac{1}{d_{\text{in}}}\frac{\text{row}_i(G)}{\|\text{row}_i(G)\|_{\text{RMS}}} \tag{11}$$

where the $\|.\|_{\text{RMS}\to\infty}$ norm was added by [44]. Moreover, they wrapped the norm-based approach outlined above into a Scion optimizer.

Within Scion, one has to assign an operator norm to each layer, e.g. out of those in Equations (6) to (8). The corresponding duality maps determine how raw gradients should be transformed for those layers before the optimizer updates the weights. For simplicity, layers are typically grouped as input, hidden, and output, and norms are assigned to these groups. Importantly, model weights are not explicitly transformed within Scion; only the weight updates are, via duality maps.

One prominent example of the norm-based view on model optimization is the Muon optimizer [28], which proved to outperform Adam at scale [36, 57] and showed great performance for models

3. Formally, $f(x) = \Theta(g(x))$ if there are constants $A, B > 0$ such that $A \cdot g(x) \leq f(x) \leq B \cdot g(x)$.

up to 1T parameters [55]. Muon can be viewed as a specific instantiation of Scion: it optimizes hidden layers under $\|.\|_{\text{RMS}\to\text{RMS}}$ assumption, and uses Adam for the remaining parameters. However, only in the case with no exponential moving average does Adam coincide with the steepest descent in "max-of-max norm" [7]. Since this is uncommon in practice, no "natural" norm applies, making Muon hard to analyze through the norm lens. By contrast, Scion naturally incorporates the norm perspective, updating every layer with an assigned, layer-specific norm, making it easy to interpret.

In practice, using norm-based optimizers as of now looks like a free lunch: they require only one momentum buffer[4] (compared to two for Adam), result in better performance with almost no computational overhead in large-scale distributed scenarios, and by design have zero-shot hyperparameter transfer built in. Moreover, the norm-based approach provides more insights into the dynamics of the model training: optimizer-assigned norms can be used naturally to monitor the training dynamics on a per-layer basis.

Appendix C. Training setup

In all experiments, we use the Llama 3 architecture [21] and `torchtitan` training framework [35]. Most of the experiments are performed on the model with a total size of 69M trainable parameters (including input/output embedding layers), hereafter referred to as proxy model. For additional ablations in Sec. 3.2, we scale up the model up to $\times 12$ in width (to 1.3B parameters) and up to $\times 32$ in depth (to 168M). Notably, we employ a `norm-everywhere` approach, inspired by the concept of well-normedness in [32] and the recent line of work [31, 37]. Effectively, we ensure that the input x to every `Linear` layer is normalized to $||x||_{\text{RMS}} = 1$ by a preceding `RMSNorm` layer without learnable parameters. More details on model configurations are provided in Appendix D.

As optimizer, we use Scion [44] with the following configuration (unless stated otherwise):

- Unconstrained version (without weight decay),
- Norm assumptions: $\|.\|_{1\to\text{RMS}} \Rightarrow \|.\|_{\text{RMS}\to\text{RMS}} \Rightarrow \|.\|_{\text{RMS}\to\infty}$ for input $\Rightarrow$ hidden $\Rightarrow$ output layers,
- Learning rate η: grid with $2^{0.5}$ step for the proxy model, and 2^1 step for the width/depth-scaled-up models,
- momentum $\mu = 0$, without Nesterov momentum,
- no warmup, constant learning rate schedule,
- $\epsilon = 1e^{-20}$ (used in gradient normalisation),
- orthogonalization of gradients for hidden layers ($\|.\|_{\text{RMS}\to\text{RMS}}$ norm assumption) with Newton-Schulz algorithm for $n_{\text{iter}} = 5$ with original Muon coefficients $a, b, c = (3.4445, -4.7750, 2.0315)$ [28].

Furthermore, we developed its distributed version, which natively integrates into `torchtitan`, supports FSDP/DDP/TP/EP/CP/PP strategies, and greatly speeds up the training at scale compared to the standard implementation. We make it openly available and provide more details in Appendix E.

4. Or even none, see `ScionLight` [44].

For pretraining, we use a high-quality partition of the Nemotron-CC dataset [53], Llama 3 tokenizer [21] with a vocabulary size of 128,256 (after padding) and a context window of 4096. All the models are pretrained with the causal language modelling task. Unless stated otherwise, a constant learning rate schedule without warmup and without decay is used. This allows us, for a given set of hyperparameters, to perform a single long run and evaluate progressively larger dataset sizes, rather than conducting several runs for each dataset individually, thereby substantially reducing computational costs [25, 26].

Appendix D. Model training configurations

- Proxy model, 69M parameters: 4 hidden layers with $d_{\text{model}} = 256$, Multi-Head Attention with $n_{\text{heads}} = 4$ and $n_{\text{kv-heads}} = 4$, SwiGLU activation function with MLP expansion factor $f_{\text{MLP}} = 2.75$, RoPE with $\theta = 10000$ [54], Llama 3 tokenizer with vocabulary size of 128 256 (after padding) [21], input and output embedding layers are not tied.
- $\times 4(12)$ wider model, 314M (1.3B) parameters: same as proxy, except $d_{\text{model}} = 1024$ (3072). In width scaling, we keep fixed $d_{\text{head}} = 64$ and scale the number of heads accordingly.
- $\times 8(32)$ deeper model, 91M (168M) parameters: same as proxy, except 32 (128) hidden layers.
- Semi-orthogonal initialization for hidden linear layers and row-wise normalized Gaussian initialization for input/output embedding layers [44]. Initialisation of the last layer of both MLP and attention blocks (those with the output being added with the residual stream) is multiplied by $1/\sqrt{2N_{\text{layers}}}$.
- Dropout disabled, no biases in all `Linear` layers, no weight sharing between input and output embedding layers.
- `norm-everywhere`: normalise input to every `Linear` layer via `RMSNorm` without learnable parameters with $\epsilon = 1e^{-20}$. Effectively, this corresponds to Pre-LN setup with QK-norm plus three additional normalisation layers: V-norm, O-norm (before output projection matrix in Attention block), and MLP-norm (after SwiGLU and before the last MLP layer). Residual connections, including the ones injecting the input embedding layer information, remain intact.
- Random seeds:
  - For all proxy model runs in Sec. 3.2 and Sec. 3.3: `30`
  - For all width/depth-scaled-up model runs: interleaved `30` + `3034` (every 2^2 step is `30`, every other 2^2 step is `3034`)
  - For layout scans in Fig. 6(*a*) and Fig. 7: averaging over `30` + `3034` + `303409` for the three "core" learning rate values ($\{2^{-4}, 2^{-6}, 2^{-8}\}$ for $B = 32$, $\{2^{-2}, 2^{-4}, 2^{-6}\}$ for $B = 128$, $\{2^{-1}, 2^{-3}, 2^{-5}\}$ for $B = 512$), `3034` + `303409` for the rest
  - For layout scans in Fig. 6(*b*): `30`
- `torchtitan` codebase, [35], FSDP2 [17], FlashAttention-2 [12]

Appendix E. Distributed Scion

We implemented a distributed version of Scion/Muon. In this section, we briefly describe the implementation. We assume that the vectorized momentum buffer update is performed before applying the actual weight update.

E.1. DDP-Disco

As a warm-up, we first consider the DDP case (note that a DDP-based version of Muon has already been implemented in `modded-nanogpt`[5]). Our implementation differs slightly from theirs, as we do not explicitly apply communication–computation overlap for DDP.

Algorithm 1: Disco `step_ddp`
Input: Parameters $\{p_i\}_{i=0}^{P-1}$ with $P = |\{p\}|$, world size M, local rank r
bucket_size $\leftarrow M$ total_buckets $\leftarrow \lceil P/M \rceil$ *global_updates* $\leftarrow$ array of length P

`/* Step 1: Compute local updates */`
for $i = 0$ **to** $P - 1$ **do**
  if $i \bmod M = r$ **then**
    $g_i \leftarrow \text{GETMOMENTUM}(p_i)$ $u_i \leftarrow \text{LMO}(g_i)$ *global_updates*$[i] \leftarrow u_i$
  end
end

`/* Step 2: Communicate updates in buckets */`
for $b = 0$ **to** *total_buckets* -1 **do**
  start_idx $\leftarrow b \cdot M$ *end_idx* $\leftarrow \min(\textit{start\_idx} + M,\ P)$ *my_idx* $\leftarrow$ *start_idx* $+r$ **if** *my_idx* $<$ *end_idx* **then**
    $u_{\text{send}} \leftarrow$ *global_updates*[*my_idx*]
  else
    $u_{\text{send}} \leftarrow 0$
  end
  $\{u_j\}_{j=0}^{M-1} \leftarrow \text{ALLGATHER}(u_{\text{send}})$ **for** $j = 0$ **to** *end_idx* $-$ *start_idx* -1 **do**
    global_updates[*start_idx* $+j$] $\leftarrow u_j$
  end
end

`/* Step 3: Apply updates vectorized */`
$\text{APPLYUPDATES}(\{p_i\}_{i=0}^{P-1}, \textit{global\_updates})$

Helper functions:

- $\text{GETMOMENTUM}(p)$: returns the momentum of p from the momentum buffer.
- $\text{LMO}(g)$: runs the LMO based on the chosen norm of p.
- $\text{ALLGATHER}(u)$: gathers one tensor u from each rank in the data-parallel group.
- $\text{APPLYUPDATES}(\{p\}, \{u\})$: applies the global updates $\{u\}$ to the parameters $\{p\}$ in a single vectorized operation.

5. https://github.com/KellerJordan/modded-nanogpt

Notice this version works out-of-the-box for PP+DDP, as we could let each PP(Pipeline Parallelism) stage only manage the parts of the model that the current PP stages needed for forward and backward.

To make it work with TP, one needs to do an extra all-gather in the local update loop.

E.2. FSDP-Disco

Here, "FSDP" refers to a combination of `FSDP2` with arbitrary parallelisms, including Data Parallelism (DP), Context Parallelism (CP), Expert Parallelism (EP), Tensor Parallelism (TP), and Pipeline Parallelism (PP). In this section, we restrict our discussion to FSDP and EP (via DP2EP). In principle, there is no need to treat DP and PP separately: one only needs to all-gather the full gradient before communication in the FSDP case to ensure compatibility with TP.

We assume the design of this work, which applies an $\|.\|_{1\to\mathrm{RMS}}$ norm for the LLM's embedding layer and an $\|.\|_{\mathrm{RMS}\to\infty}$ norm for the output linear layer. (`SignNorm` is also acceptable and remains compatible if one strictly follows Scion's design.)

The `FSDP2` implementation in PyTorch shards weights and gradients along the tensor's first dimension. We discuss Disco under this assumption and further assume that each tensor or matrix corresponds to a single layer. Consequently, fused tensors such as `fused_QKV` in attention layers or `fused_W13` in SwiGLU are not supported.

Under these hypotheses, we can classify parameters into three groups: `embedding`, `experts`, and (pure-)`fsdp`. For updates, no extra communication is required for `embedding` and `experts` parameters, thanks to the `Shard(0)` strategy in `FSDP2`.

Algorithm 2: Disco `step_embedding`
Input: Embedding parameters $\{p_i\}_{i=0}^{P-1}$

`/* Initialise updates storage */`
updates $\leftarrow$ array of length P

`/* get momentum and compute LMO update on local shards */`
for $i = 0$ **to** $P-1$ **do**
  $g_i \leftarrow \textsc{GetMomentum}(p_i)$ $u_i \leftarrow \mathrm{LMO}(g_i)$ *updates*$[i] \leftarrow u_i$
end

`/* Apply updates vectorized */`
$\textsc{ApplyUpdates}(\{p_i\}_{i=0}^{P-1},$ *updates*$)$

Noting that MoE expert weights are typically laid out as either $(\text{total\_experts}, d_{\text{out}}, d_{\text{in}})$ or $(\text{total\_experts}, d_{\text{in}}, d_{\text{out}})$, we apply a transpose in the latter case to ensure that the output dimension comes first. In an FSDP + DP2EP setting, each gradient passed to `LMO` is therefore a 3D tensor with layout $(\text{local\_experts}, d_{\text{out}}, d_{\text{in}})$. Accordingly, SVD or Newton-Schulz-based algorithms must correctly handle batched inputs.

Algorithm 3: Disco `step_experts`
Input: Expert parameters $\{p_i\}_{i=0}^{P-1}$, transpose flag *transpose*

`/* Initialise updates storage */`
updates $\leftarrow$ array of length P

`/* get momentum and compute LMO update on local shards */`
for $i = 0$ **to** $P - 1$ **do**
  $g_i \leftarrow$ GETMOMENTUM(p_i) $u_i \leftarrow$ BATCHEDLMO$(g_i;$ *transpose_experts* $=$ *transpose*$)$ *updates*$[i] \leftarrow u_i$
end

`/* Apply updates vectorized */`
APPLYUPDATES$(\{p_i\}_{i=0}^{P-1},$ *updates*$)$

Algorithm 4: Disco `step_fsdp`
Input: FSDP-sharded parameters $\{p_i\}_{i=0}^{P-1}$, world size M over `fsdp`, local rank r
bucket_size $\leftarrow M$ total_buckets $\leftarrow \lceil P/M \rceil$ *global_updates* $\leftarrow$ array of length P

for $b = 0$ **to** *total_buckets* -1 **do**
  start $\leftarrow b \cdot M$; *end* $\leftarrow \min($*start* $+ M,\ P)$ *my_idx* $\leftarrow$ *start* $+ r$
  for $j = 0$ **to** $M - 1$ **do**
    $i \leftarrow$ *start* $+ j$ **if** $i <$ *end* **then**
      $g_i \leftarrow$ GETMOMENTUM(p_i) `// row-sharded by FSDP` *send_list*[j] $\leftarrow g_i$
    else
      send_list[j] $\leftarrow 0$ `// zero padding`
    end
  end
  recv_list $\leftarrow$ ALLTOALL$($*send_list*$)$
  $g^\star \leftarrow$ CONCATROWS$($*recv_list*$)$ `// reconstruct full gradient for` $p_{i^\star}$
  $u^\star \leftarrow \mathrm{LMO}(g^\star)$
  updates_send_list $\leftarrow$ SPLITROWS$(u^\star,\ M)$ `// split` $u^\star$ `by rows` *updates_recv_list* $\leftarrow$ ALLTOALL$($*updates_send_list*$)$
  for $j = 0$ **to** *end* $-$ *start* -1 **do**
    global_updates[*start* + j] $\leftarrow$ *updates_recv_list*[j]
  end
end

`/* Single vectorized apply */`
APPLYUPDATES$(\{p_i\}_{i=0}^{P-1},$ *global_updates*$)$

Helper functions:

- ALLTOALL(*list*): list-based ALLTOALL over `dp_shard_cp`.
- CONCATROWS(*list*): concatenates row-shards into a full tensor.
- SPLITROWS(u, M): splits u into M contiguous row blocks.

Appendix F. Optimal norm measurement

F.1. General idea

Our initial intuition was that for a given model and data scale, there is always some optimal norm value, corresponding to some optimal hyperparameter choice. To establish this, we focus on the output layer with the Scion-assigned $\|.\|_{\mathrm{RMS}\to\infty}$ norm (hereafter referred to as *output norm*), motivated in the main text (Sec. 2); we also ablate this choice in Appendix T. The choice of $\|.\|_{\mathrm{RMS}\to\infty}$ norm is further motivated by [6] as mapping from a "natural" continuous RMS norm semantics for hidden model representations onto a discrete vocabulary. Since by default we disable momentum and any regularization, we are only left with learning rate (η) and batch size (B) as hyperparameters to tune for optimality.

To extract the optimal hyperparameter configuration and the corresponding optimal norm, we run an (η, B) grid search for a given model and a given pretraining dataset size (hereafter referred to as horizon D, measured in tokens), and evaluate the model performance with training loss (cross-entropy of the next token prediction). Since we train in a non-repeating "infinite-data" regime, training loss faithfully reflects model performance and its generalization. First, we examine how the optimal norm, associated to a (η^*, B^*) configuration optimal for a given horizon, changes as the horizon increases. Then, we fix the horizon and scale up the model in width and depth, repeating the same optimal norm measurement. This way, we study both model and dataset scaling directions.

Practically, for every batch size we are interested in "marginalising" or "profiling" across learning rates, i.e. picking the optimal one and the corresponding output norm (see Appendix D for details on the grid and random seed variations). However, an empirically lowest-loss point across the learning rate grid turned out to be a statistically noisy estimate; therefore, for each batch size, we perform a fit to the distribution of training loss vs. output norm across learning rates. Finally, we extract the optimal norm value from the fitted curve and the corresponding learning rate from the nearest data point to the fitted optimum. We provide more details on the fitting procedure below.

F.2. Fitting & loss smoothing

After naïvely taking the empirical optimum across the learning rate grid (e.g. as the one emphasized with dashed black lines in Fig. 2), we found that the corresponding norm scans, although still indicating norm transfer, are quite noisy (e.g. compare Fig. 1(*a*) vs. Fig. 17a). From Fig. 2 we noted that data points in loss vs. norm plot resemble parabola if plotted in log-log scale. Furthermore, we know by design that at initialization (step 0) the output norm equals to 1, and train loss equals to 11.765. With this, we chose to perform a constrained fit with a second-order polynomial function in log-log scale $\log(\text{loss}) = a\log(\text{norm})^2 + b\log(\text{norm}) + c$, where the free term c is fixed at precisely the loss value at initialization. We do this using weighted least squares fitting with `np.linalg.lstsq`, where the weighting is done with inverse uncertainties coming from *loss smoothing*, described below. For robustness, only seven data points around the empirical optimum are used in the fit. The optimal loss and norm values are then extracted as the parabola optimum coordinates. Optimal learning rate is taken from the data point closest to the fitted optimum. Results of such fits for Fig. 1 can be found in Fig. 18.

Since running several random seeds is computationally intensive, we perform *loss smoothing* to estimate the loss variance and make loss estimates more robust. Essentially, for a given horizon point, instead of taking its loss value, we average it with the previous and next evaluated points

(67M tokens away, or e.g. 128 steps from each other with $B = 128$). Empirically estimated standard deviation is then used in the fits as described above. We employ loss smoothing only for small batch sizes $B \leq 128$, as those having large loss variance, and for large token horizons $D \geq 2^{33}$, to stay in the region of largely converged loss (which can be locally linearly approximated) and therefore not bias the estimate.

In order to get variance estimate in Fig. 3 without running several random seed runs, we vary the fitting procedure outlined above (with/without fitting, with/without loss smoothing, with/without constraint to loss at initialization), thus resulting in 6 total variations. For each of those we track how optimal norm/loss/learning rate changes, and propagate this variance to plotting and downstream analysis.

Appendix G. Illustration of optimal norm for fixed batch size B and horizon D

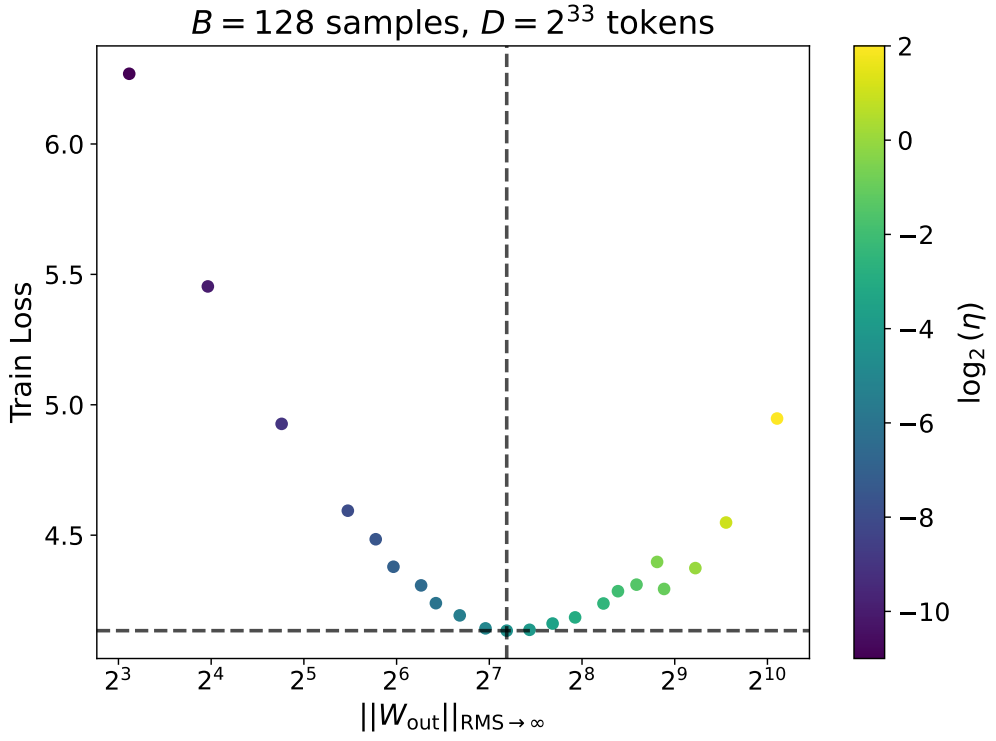


Figure 2: **Interplay of training loss, output layer norm $\|W_{\text{out}}\|_{\text{RMS}\to\infty}$ and learning rate**. Results are for the proxy model (69M parameters), batch size $B = 128$ samples and horizon $D = 2^{33}$ tokens. Points are colored by $\log_2(\eta)$ where η is the learning rate. Black dashed lines mark the optimal configuration with minimum training loss.

Appendix H. Optimal learning rate per batch size across horizons

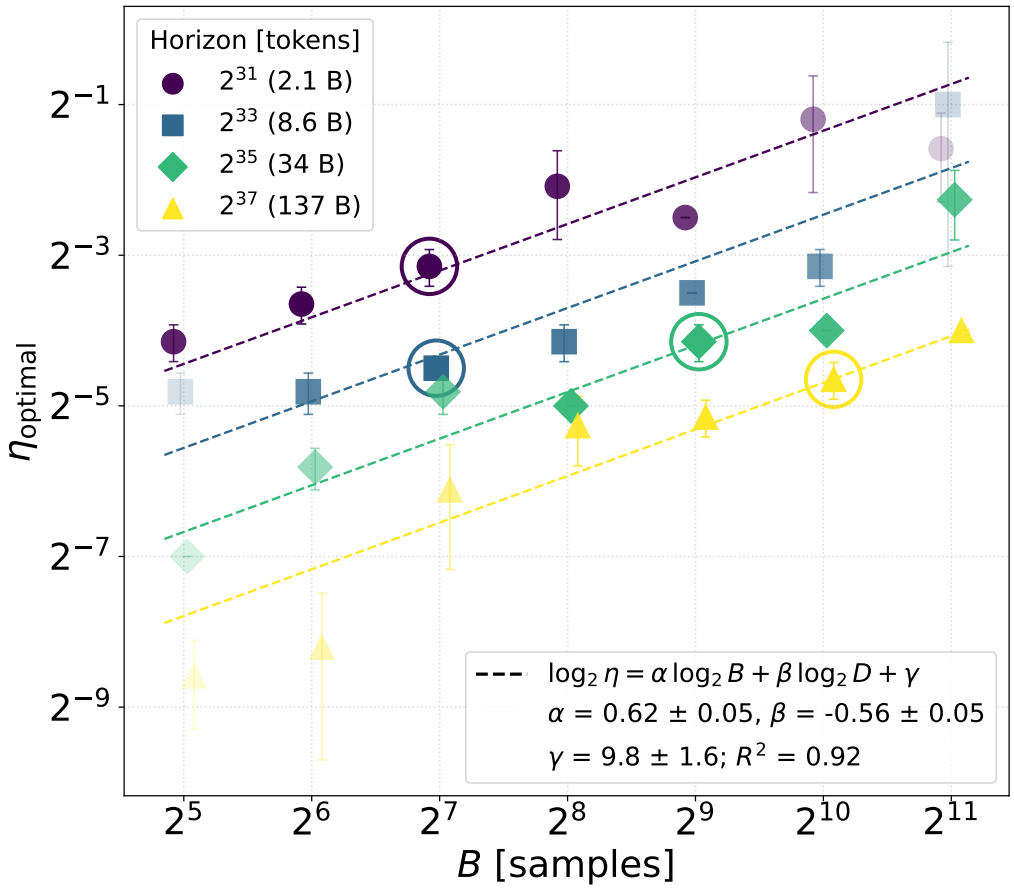


Figure 3: **Optimal learning rate per batch size across horizons.** Circled markers indicate optimal (η^*, B^*) with the lowest loss. Within a horizon, marker transparency linearly interpolates between the lowest- and highest-loss runs. Error bars show systematic variation from the fitting method (Appendix F). Dashed lines are a joint linear regression with $\log_2 \eta^* \sim \log_2 B + \log_2 D$. Fit is performed on an extended set of horizons, not shown here for clarity (see Fig. 8(*a*)).

Appendix I. Optimal norm can be reached with multiple (η, B) configurations

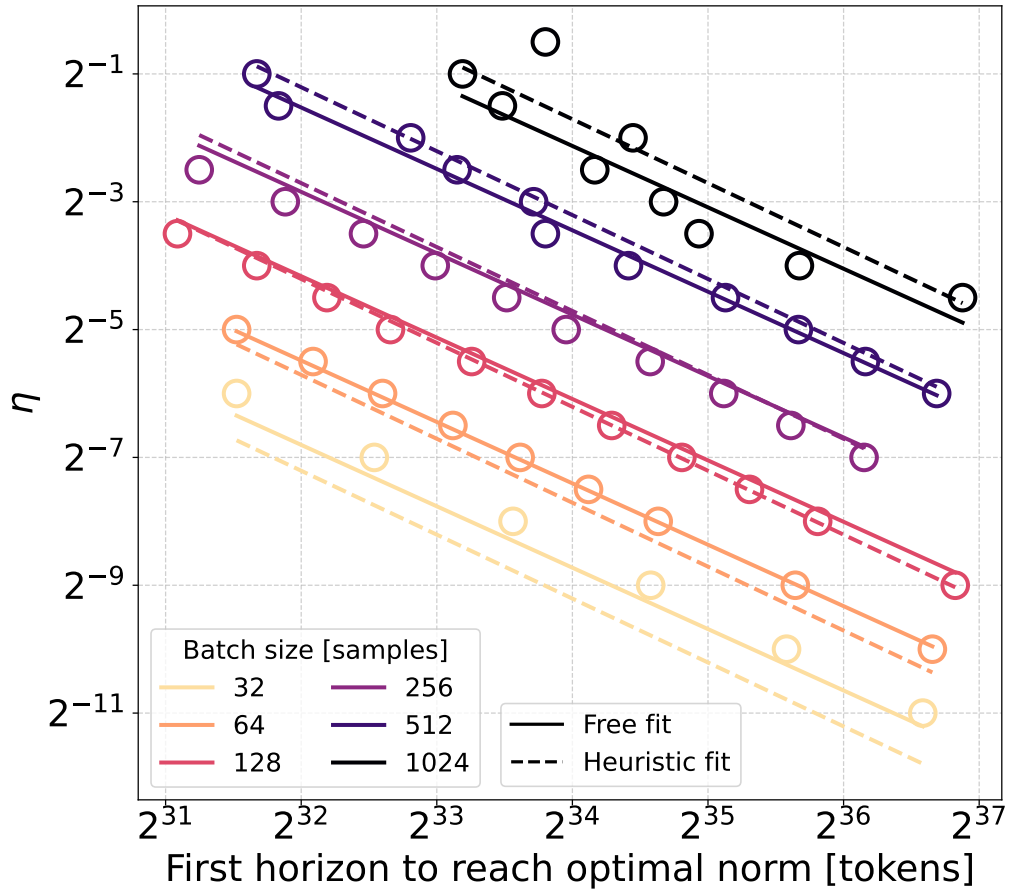
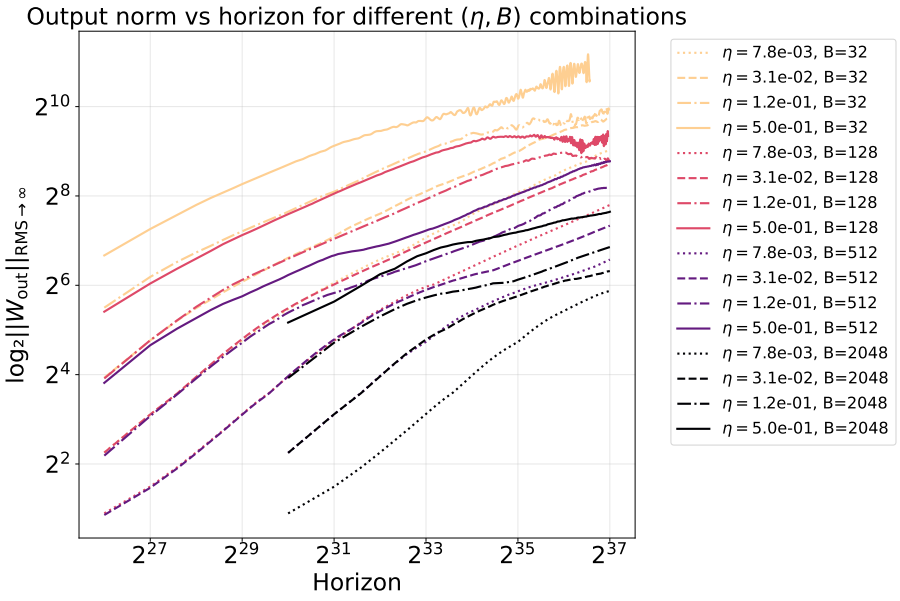
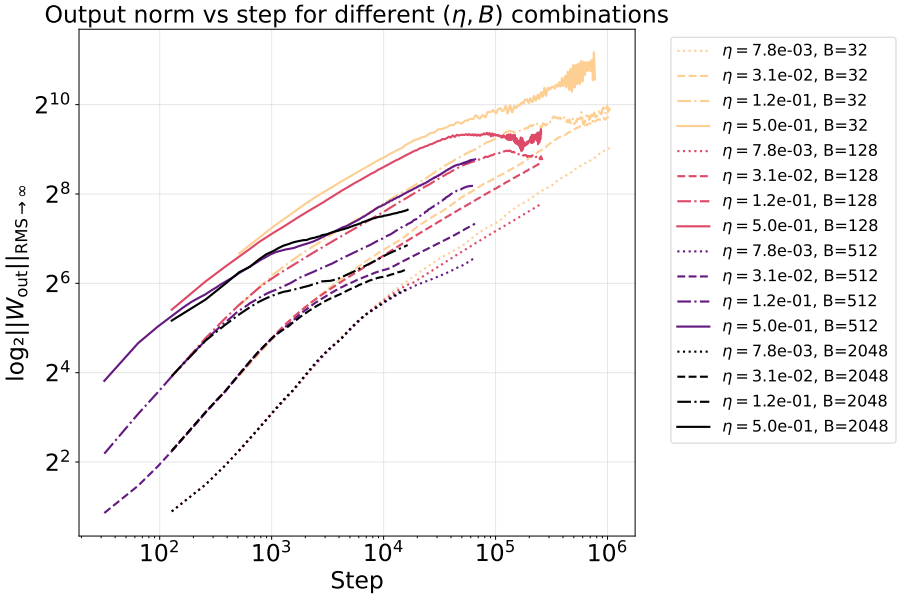


Figure 4: (η, B) **combinations that reach the optimal norm** $\|W_{\text{out}}\|_{\text{RMS}\to\infty} = 2^{7.0\pm0.2}$ **for a given token horizon.** Colours denote batch size (B); the y-axis is learning rate (η). Solid and dashed lines denote free and heuristic fits (described in text). See also Fig. 3 in the main text for the corresponding (η^*, B) scan across horizons.

Appendix J. Output norm evolution with different (η, B)



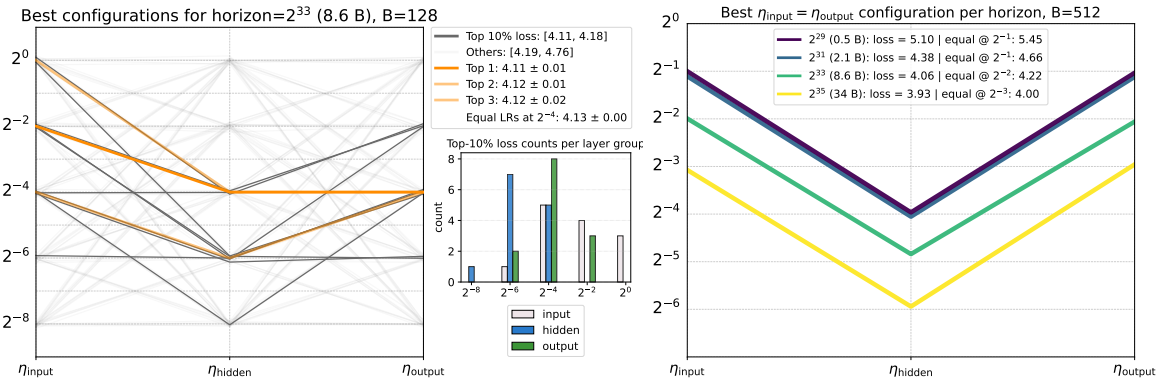
(*a*)



(*b*)

Figure 5: **Growth of the output layer norm $\|W_{\text{out}}\|_{\text{RMS}\to\infty}$ vs. horizon, in tokens (a) and number of steps (b).** Results are for the proxy model (69M parameters). Each curve is a (learning rate η, batch size B) pair, with B measured in samples: colour encodes batch size and line style encodes learning rate, as described in the legend.

Appendix K. Optimal per-layer-group learning rate



(*a*) (*b*)

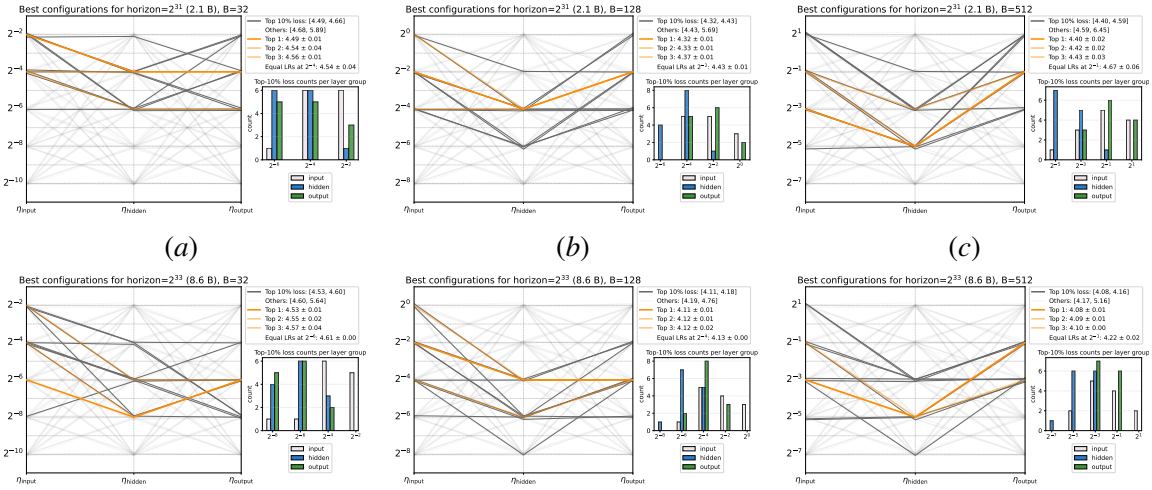
Figure 6: **(a) Parallel-coordinates view of per-layer-group learning rate tuning.** Results are for the proxy model (69M parameters) and batch size $B = 128$ samples, averaged across random seeds as described in Appendix D. Dark gray lines are the top 10% runs (loss 4.11–4.18); light gray lines are the remainder (loss 4.19–4.76). Orange traces highlight the three best settings. The inset histogram shows the distribution of top 10% counts for each layer group. **(b) Best learning rate layouts per training horizon under the constraint $\eta_{\text{input}} = \eta_{\text{output}}$.** Results are for the proxy model (69M parameters) and batch size $B = 512$ samples. All horizons favor a V-shaped layout with η_{hidden} smaller than the input/output learning rates by the same $\times 1/8$ factor. In the legend we also report loss for the optimal $\eta_{\text{input}} = \eta_{\text{hidden}} = \eta_{\text{output}} \equiv \eta$ layout ("equal @η").

In the main experiments, we approached scaling from a "global" learning rate point of view. However, this may not be the case, and intricate dynamics can emerge where various layers require different learning rates at different scales to be trained optimally, thus questioning our conclusions so far. Here, we explore if this is the case.

Fig. 6(*a*) presents results for a proxy model (69M parameters), fixed data horizon (8.6B tokens) and fixed batch size ($B = 128$ samples, optimal for this horizon) where we run grid search over learning rate values $\eta \in \{2^{-8}, 2^{-7}, \ldots, 2^{0}\}$ for input (token embedding), output (linear projection onto vocabulary) and hidden (all the other) layers, averaged across random seeds (Appendix D). We observe that there is little optimal learning rate imbalance across layer groups, and uniform learning rate assignment results in the same loss as the optimal configurations within uncertainties. Furthermore, from the width of the optimal nodes count histograms per layer groups, we conclude that the output layer is the most sensitive to learning rate mistuning, with the sensitivity progressively decreasing for hidden and then input layers. In this context, noteworthy is the work of [56] where the authors also use norm-based view and propose to automate the tuning procedure with learnable multipliers, yielding automatic optimal norm alignment across layers.

From analysing Fig. 6(*a*) and additional ones for different batch sizes (Fig. 7) we found that the configuration $\eta_{\text{input}} : \eta_{\text{output}} : \eta_{\text{hidden}} = 1 : 1/8 : 1$ is always among the top 10%. This symmetry simplifies the learning scan and notably contradicts the optimal configurations suggested in [44]

and [47]. To study dynamics with horizon scaling, we perform the learning rate grid scan same as in Fig. 6(*a*) but with constraining $\eta_{\text{input}} = \eta_{\text{output}}$ [6], for the proxy model with $B = 512$. Fig. 6(*b*) illustrates the results, where we see the optimal hidden ratio ($\eta_{\text{input}}/\eta_{\text{hidden}} = 1/8$) transfer across horizons, as well as that it brings loss improvement w.r.t. a constant learning rate baseline. Lastly, we note that again, due to the optimizer design, we expect these observations to hold true under model width scaling.



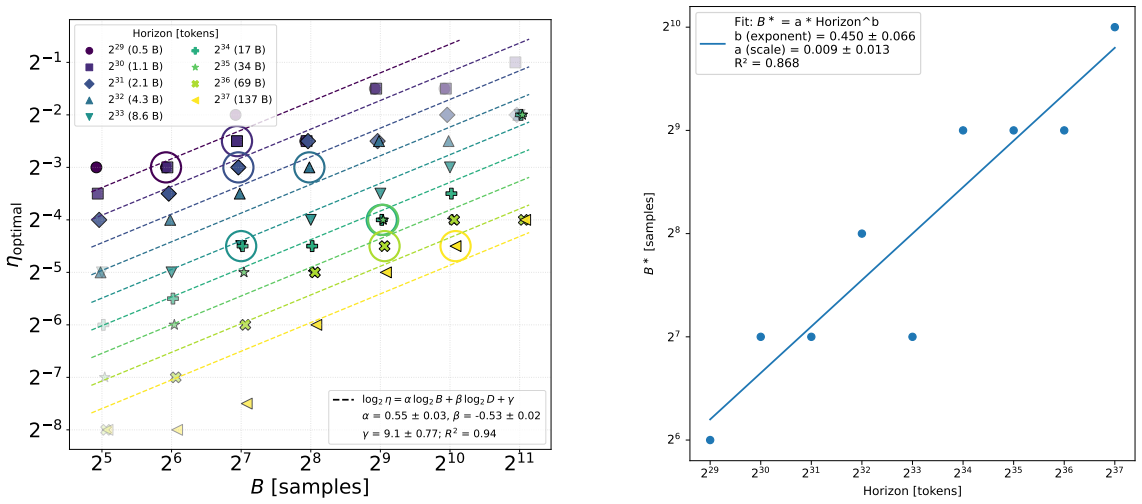
(*a*) (*b*) (*c*)

(*d*) (*e*) (*f*)

Figure 7: **Extended version of Fig. 6(*a*) with additional batch sizes and horizons.** Top (bottom) row: $D = 2^{31}(2^{33})$ token horizons. Batch sizes, in samples: $B = 32$ (left), $B = 128$ (middle), $B = 512$ (right). Performance is averaged across random seeds as described in Appendix D. Note that the optimal B^* is 128 for both $D = 2^{31}$ and $D = 2^{33}$ according to Fig. 3.

6. In terminology of [6] this corresponds to *mass* tuning.

Appendix L. Optimal $B^*(D)$ measurement

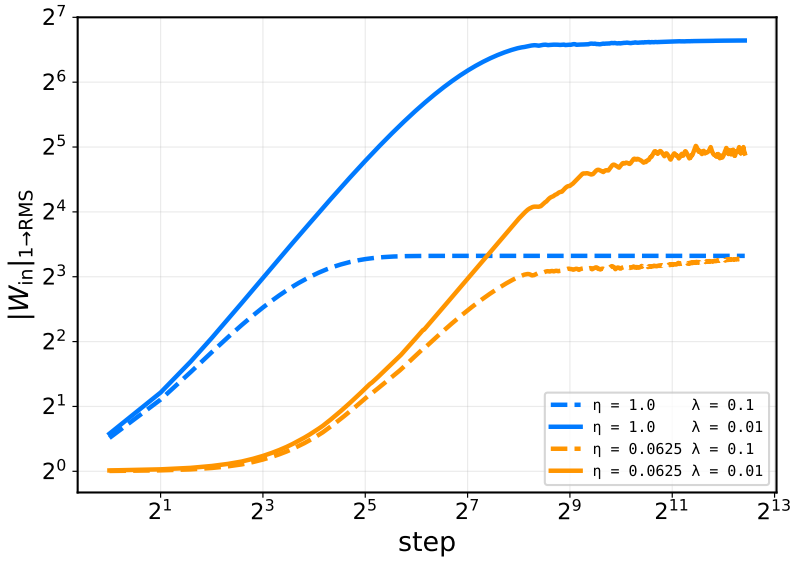


(*a*) (*b*)

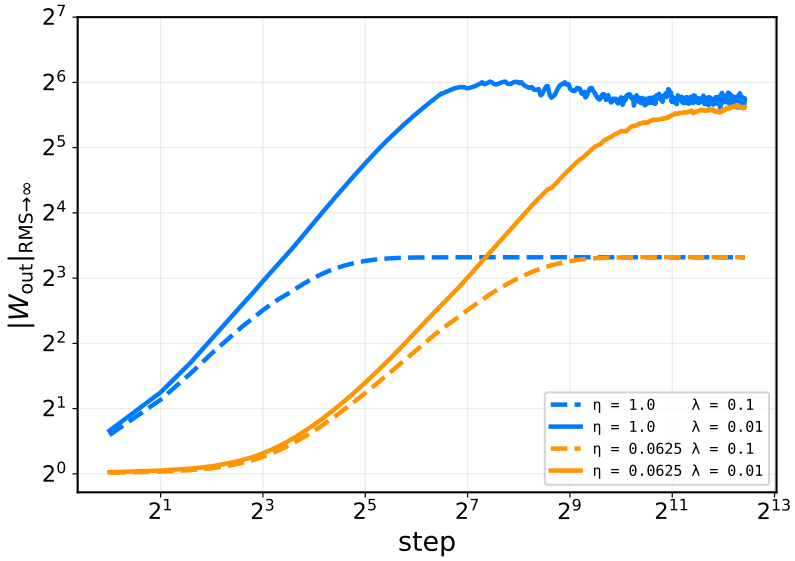
Figure 8: **(a)** Same as Fig. 3, but with extended set of horizons. **(b)** Optimal batch size B^* vs. horizon, as extracted from (a)). The line is a power-law fit (described in legend).

Fig. 3, for the sake of clarity and simplicity, illustrates only four horizons. This is not really sufficient to extract precisely the scaling of optimal (η^*, B^*) (circled markers) with D, as it would mean fitting of four data points. We therefore perform the ordinary least squares (OLS) fit on the extended set of 9 horizons from Fig. 8(*a*), effectively fitting the x-coordinate of the circled markers with a line. We model optimal batch size dependency on horizon D as a power law $B^*(D) = aD^b$ and present results on Fig. 8(*b*). We extract $B^* \propto D^{0.45\pm0.07}$, consistent with the square-root scaling.

Appendix M. Norm constraint with weight decay



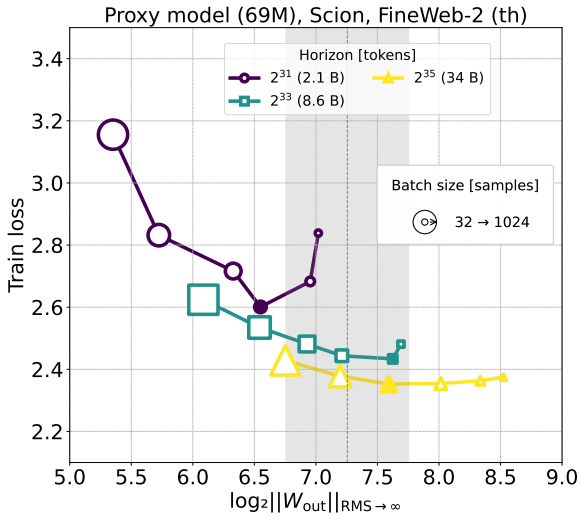
(*a*)



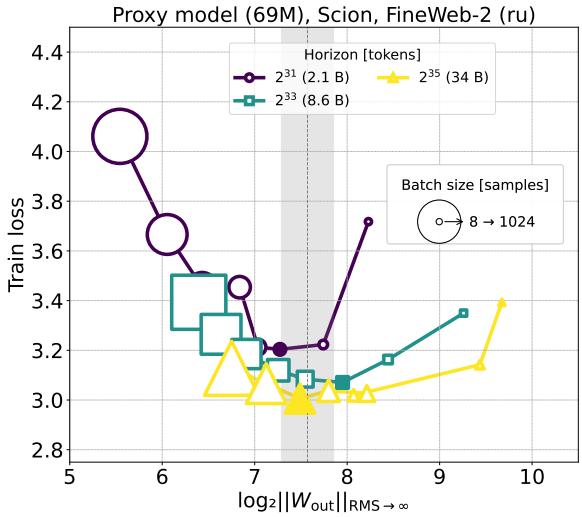
(*b*)

Figure 9: **Operator norm against number of gradient update steps.** Fixed batch size $B = 32$, momentum $\mu = 0.1$, two values of learning rate $\eta = \{0.0625, 1.\}$ and two values of weight decay $\lambda = \{0.01, 0.1\}$ (applied as in Pethick et al. [44]), for a proxy model (69M parameters). **(a)** $\|W_{\text{in}}\|_{1\to\text{RMS}}$ norm **(b)** $\|W_{\text{out}}\|_{\text{RMS}\to\infty}$. We see for $\lambda = 0.1$ both norms converging to $1/\lambda$, while for $\lambda = 0.01$ asymptotic values are not conclusive.

Appendix N. Ablation with FineWeb-2 dataset

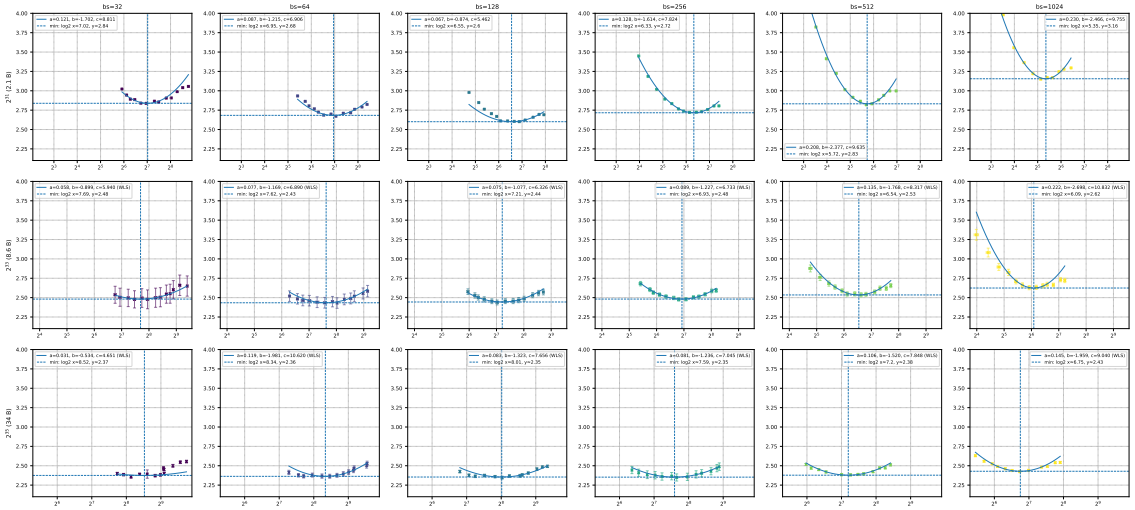


(*a*)

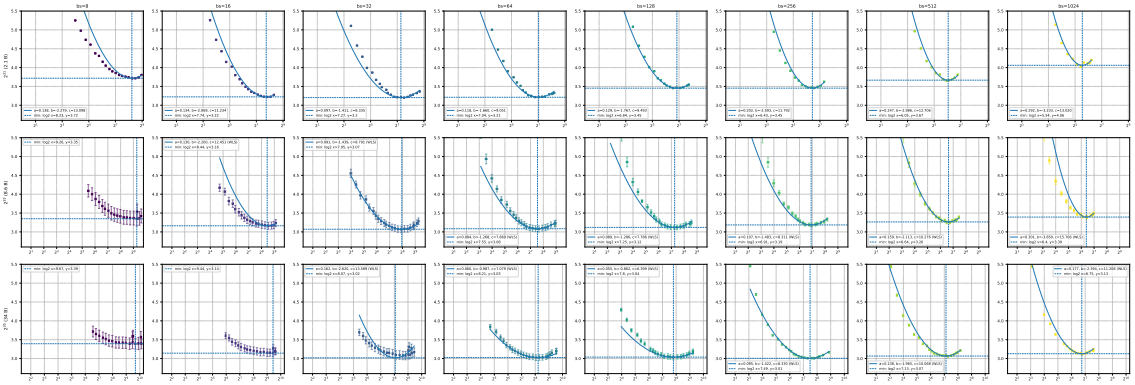


(*b*)

Figure 10: **Same as Fig. 1(*a*) but using the FineWeb-2 dataset: (a)** Thai partition, **(b)** Russian partition. We note that while for the Russian language the three horizon curves are nested inside of each other and share the same optimal norm, for the Thai partition the first horizon is off. This may be due to being an early phase of training or statistical fluctuation. For completeness, we provide the individual learning rate scans and fits used to produce this plot in Fig. 11.



(*a*)



(*b*)

Figure 11: **Individual $\|W_{\text{out}}\|_{\text{RMS}\to\infty}$ norm scans for various batch sizes B (columns) across various horizons D (rows), for the FineWeb-2 dataset: (a)** Thai partition, **(b)** Russian partition.

Appendix O. Depth transfer study

Model: same as our proxy model (Appendix D), with the only difference in the head configuration: $n_{\text{query\_heads}} = 2$, $n_{\text{kv\_heads}} = 1$. We run a combination of two ablations: (i) weight initialisation depth-wise scaling (via gains/variance), and (ii) residual branch summation ratios.

For weight *initialisation*, the depth-wise scaling factors are applied to **only** the output linear projection of attention and SwiGLU. We compare three flavours of depth init scaling: `identity` (baseline), `total-depth`, and `relative-depth`, defined by multiplying the gain σ by

$$\sigma* = \begin{cases} 1/\sqrt{2N_{\text{layers}}} & \text{scale by } \texttt{total-depth}, \\ 1/\sqrt{2\,l_i} & \text{scale by } \texttt{relative-depth}, \\ 1 & \text{scale by } \texttt{identity}. \end{cases} \tag{12}$$

where N_{layers} is the total number of Transformer blocks, and $l_i \in \{1, \ldots, 2N_{\text{layers}}\}$ is the relative depth of the current block; σ is the scaled orthogonal gain, $\sigma = \sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}}$, for hidden weights $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$.

Each transformer block is assigned depth 2, since attention and FFN sub-blocks each count as depth 1. When using `relative-depth`, the depth of all FFN blocks can be offset by 1.

For depth-wise residual scaling, we write the residual connection in transformer as:

$$Y \;=\; \alpha \cdot X \;+\; \beta \cdot \text{Block}\big(\text{Norm}(X)\big), \tag{13}$$

where X is the block input and $\text{Block}(\cdot)$ denotes either self-attention or a FFN, and Norm is RMSNorm in our setup.

We consider three depth-wise residual scaling schemes:

$$(\alpha, \beta) = \begin{cases} \left(\frac{2N_{\text{layers}}-1}{2N_{\text{layers}}}, \frac{1}{2N_{\text{layers}}}\right) & \text{scale by } \texttt{depth-normalized}, \\ \left(1, \frac{1}{2N_{\text{layers}}}\right) & \text{scale by } \texttt{completeP}, \\ (1, 1) & \text{scale by } \texttt{identity}. \end{cases} \tag{14}$$

`depth-normalized` [32] scales both the residual and block contributions proportionally to depth. `completeP` [15] preserves the residual branch while scaling down the block contribution by depth. `identity` corresponds to the conventional unscaled residual formulation.

We fixed batch size (B) to 32 samples, the sequence length to 4096, and the number of training steps to 2048. Experiments were conducted using proxy models with depths $N_{\text{layers}} \in \{2, 16, 64\}$. For all models, we performed a sweep over the learning rate $\{2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}\}$.

We report the final-step losses in Table 1, Table 2, and Table 3 for the three depths, respectively, with the two lowest losses highlighted. From the perspective of learning rate transfer, we find that with our optimizer, the optimal learning rate consistently remains around 2^{-2}, regardless of weight initialisation or residual scaling. We also observe that combining `total-depth` weight initialisation with `identity` residual scaling yields a negligible improvement compared to using `identity` weight initialisation.

Table 1: 2 layers ($B = 32$, steps=2048)

Residual init	**Residual multiplier**	**Learning rate η**				
		2^{-4}	2^{-3}	2^{-2}	2^{-1}	2^{0}
`total-depth`	`identity`	4.20	**4.11**	**4.09**	4.13	4.22
`total-depth`	`depth-normalized`	4.20	**4.12**	**4.11**	4.17	4.21
`total-depth`	`completeP`	4.22	**4.15**	**4.16**	4.17	4.28
`identity`	`identity`	4.19	**4.10**	**4.10**	4.13	4.23
`identity`	`depth-normalized`	4.22	**4.12**	**4.12**	4.13	4.21
`identity`	`completeP`	4.21	**4.15**	**4.13**	4.16	4.24
`relative-depth`	`identity`	4.20	**4.11**	**4.09**	4.13	4.23
`relative-depth`	`depth-normalized`	4.20	**4.13**	**4.11**	4.16	4.25
`relative-depth`	`completeP`	4.21	**4.16**	**4.14**	4.18	4.24

Table 2: 16 layers ($B = 32$, steps=2048)

Residual init	**Residual multiplier**	**Learning rate η**				
		2^{-4}	2^{-3}	2^{-2}	2^{-1}	2^{0}
`total-depth`	`identity`	3.81	**3.75**	**3.73**	3.77	3.88
`total-depth`	`depth-normalized`	3.85	**3.79**	**3.80**	3.84	3.92
`total-depth`	`completeP`	3.87	**3.82**	**3.82**	3.85	3.94
`identity`	`identity`	3.81	**3.74**	**3.75**	3.79	3.89
`identity`	`depth-normalized`	3.83	**3.78**	**3.78**	3.83	3.92
`identity`	`completeP`	3.86	**3.81**	**3.81**	3.85	3.94
`relative-depth`	`identity`	3.82	**3.79**	**3.74**	3.80	3.90
`relative-depth`	`depth-normalized`	3.84	**3.79**	**3.80**	3.83	3.95
`relative-depth`	`completeP`	3.88	**3.82**	**3.82**	3.85	3.95

Table 3: 64 layers (B=32, steps=2048)

Residual init	**Residual multiplier**	**Learning rate η**				
		2^{-4}	2^{-3}	2^{-2}	2^{-1}	2^{0}
`total-depth`	`identity`	3.67	**3.60**	**3.60**	3.65	3.79
`total-depth`	`depth-normalized`	3.71	**3.65**	**3.65**	3.69	3.80
`total-depth`	`completeP`	3.72	**3.67**	**3.67**	3.72	3.82
`identity`	`identity`	3.70	**3.63**	**3.62**	3.66	3.78
`identity`	`depth-normalized`	3.70	**3.64**	**3.64**	3.69	3.80
`identity`	`completeP`	3.70	**3.70**	**3.67**	3.72	3.82
`relative-depth`	`identity`	3.70	**3.61**	**3.61**	3.67	3.82
`relative-depth`	`depth-normalized`	3.71	**3.65**	**3.65**	3.69	3.80
`relative-depth`	`completeP`	3.72	**3.68**	**3.67**	3.73	3.83

Appendix P. Adam optimizer

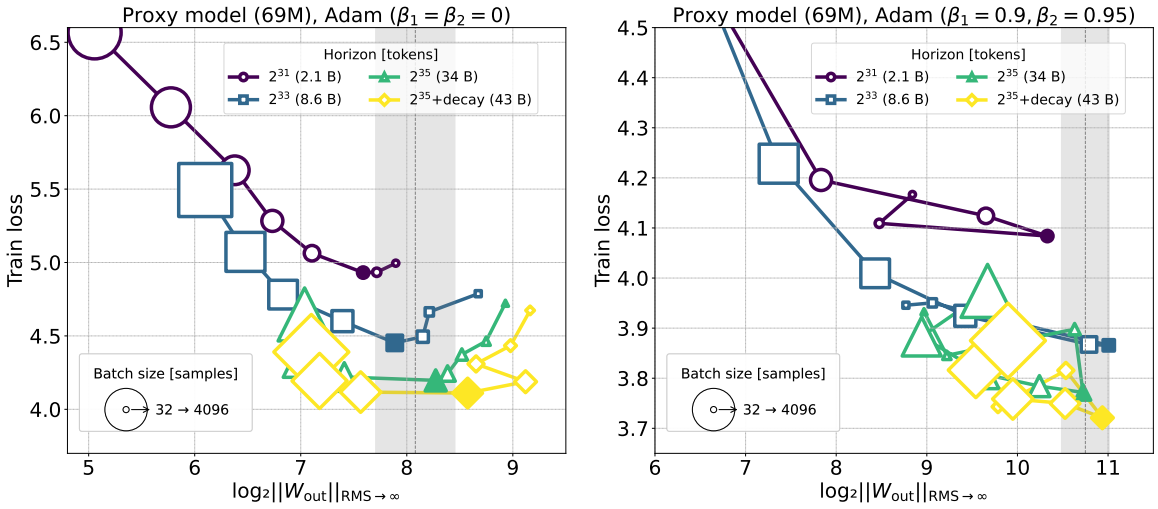


Figure 12: **Same as Fig. 1(*a*) but using the Adam optimizer** (no weight decay, warmup for $2^{29} \approx 537$M tokens across all batch sizes, followed by constant learning rate schedule with linear decay to 0 for 20% of the total schedule): **(a)** $\beta_1 = \beta_2 = 0$, **(b)** $\beta_1 = 0.9$, $\beta_2 = 0.95$. For the no-momentum version (a) we observe norm transfer at the same optimal norm value as our main experiment with Scion (Fig. 1(*a*)). For the version with momentum (b), norm transfer is also present (with reduced sensitivity to batch size choice similarly to Scion), but notably at a higher optimal norm value.

Appendix Q. Residuals-only data scaling

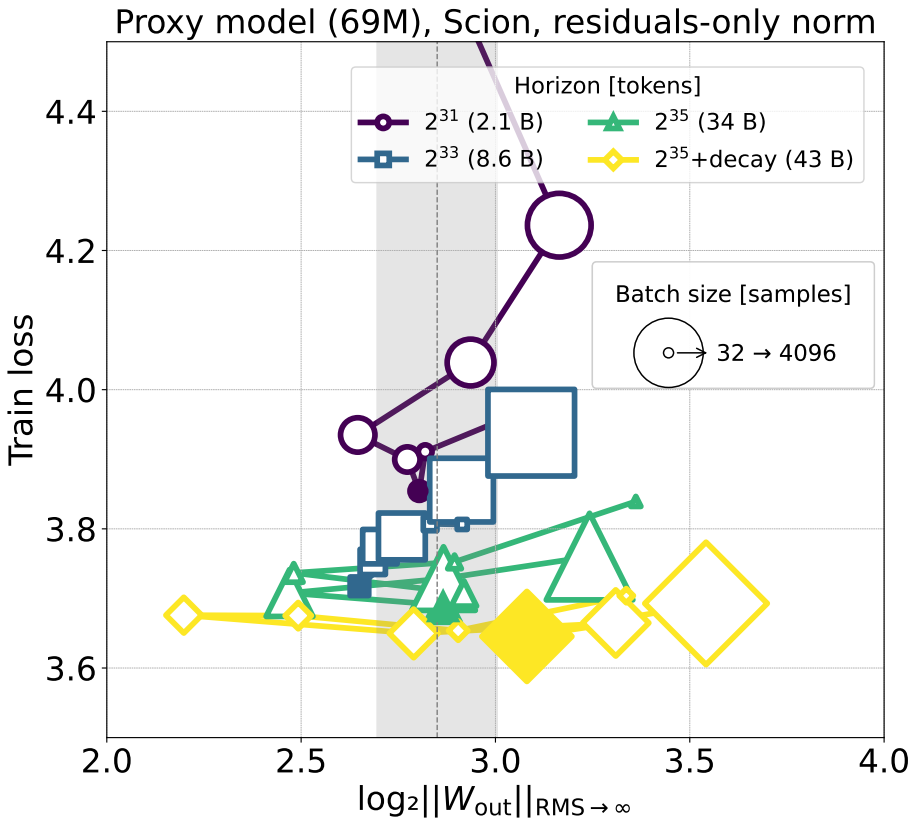
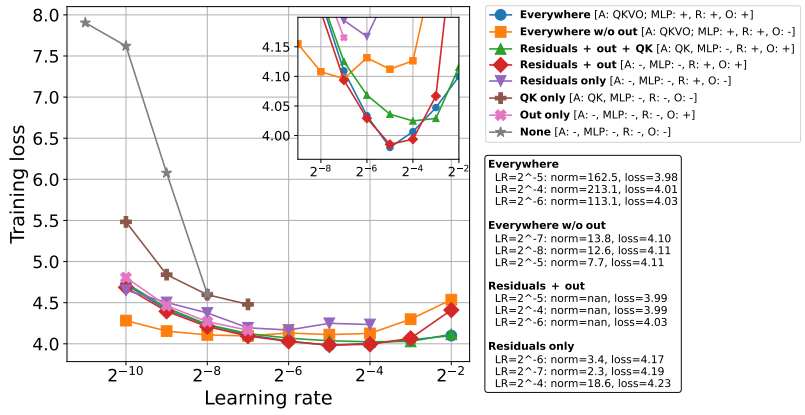
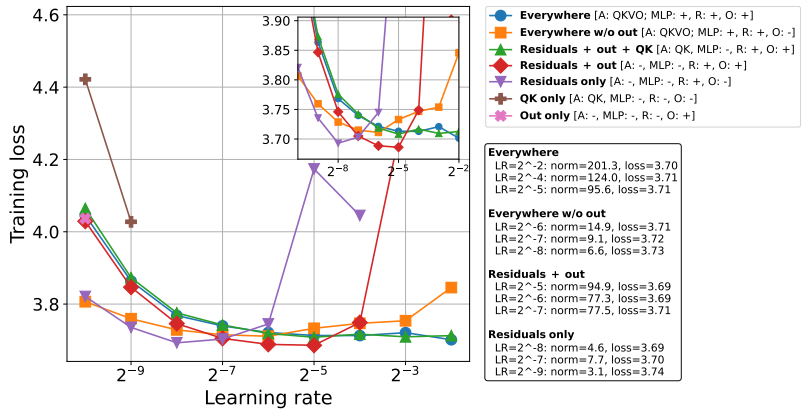


Figure 13: **Same as Fig. 1(*a*) but leaving in the proxy model only residuals normalization layers** (`RMSNorm` without trainable parameters normalizing inputs to every attention and MLP blocks).

Appendix R. Normalization layers



(*a*)



(*b*)

Figure 14: **Learning rate scan for various layer normalization strategies for the proxy model and the Scion optimizer: (a)** without momentum $\alpha = 1.0$, **(b)** with momentum $\alpha = 0.1$. All runs share the same batch size $B = 256$ [sequences] and the same learning rate schedule: no warmup, constant phase (2^{35} tokens), linear decay to 0 for 20% of the total horizon (0.25×2^{35} tokens). Learning rate plotted on the X axis corresponds to the learning rate of the constant phase. Markers which are not present on the plot mean that the training diverged.
Notations in the legend corresponds to normalising 1) for the attention block (A) QKV: outputs of the QKV projection matrices, O: inputs to the output projection matrix; 2) MLP: inputs to the last linear layer; 3) residuals (R): inputs to both attention and MLP blocks; 4) Output layer (O): inputs to the final layer projecting the model dimension onto the vocabulary. Additional legend block shows the final norm ($\|W_{\text{out}}\|_{\text{RMS}\to\infty}$) value and training loss for the top-3 optimal learning rate runs within each normalization strategy.

Appendix S. Momentum & learning rate decay

In this set of experiments we set momentum to 0.1 (which is by default disabled in the main text) and firstly run the same horizon scaling experiment for the proxy model (69M parameters) with the constant learning rate schedule and evaluate at the same horizons $D = \{2^{31}, 2^{33}, 2^{35}, 2^{37}\}$ as Fig. 1(*a*). The results are presented in Fig. 15(*a*). Here we perform loss smoothing in the same way as for the no-momentum scenario, but do not perform the fitting, i.e. for each batch size we take the optimal norm from the empirically best performing learning rate run. We find that the curves look more like "blobs", where multiple batch sizes give almost the same performance and are centered around the optimal norm (which also transfers across horizons). Also the loss difference between horizons is not well-pronounced as in the no-momentum scenario.

Then, we add learning rate decay, where we start from checkpoints of the horizons specified above, assume that that constitutes 75% of the total horizon, and linearly decay learning rate to 0 for the rest 25%. Likewise, we smooth loss values and take optimum value per batch size across empirical ones on the learning rate grid. In Fig. 15(*b*) we see that there is potentially a slight drift of the optimal norm with horizon scaling. However, after examining individual scans (Fig. 19) we surprisingly found that for long horizons the learning rate decay smooths out the norm optimum: a broad range (factor $\times 4 - 8$ in norm) of learning rates results in the same loss. Hence, there is no longer a single optimal norm, but rather a sizeable range, indicating that learning rate decay significantly reduces norm sensitivity. Therefore, we conclude that the seaming drift in Fig. 15(*b*) is not significant.

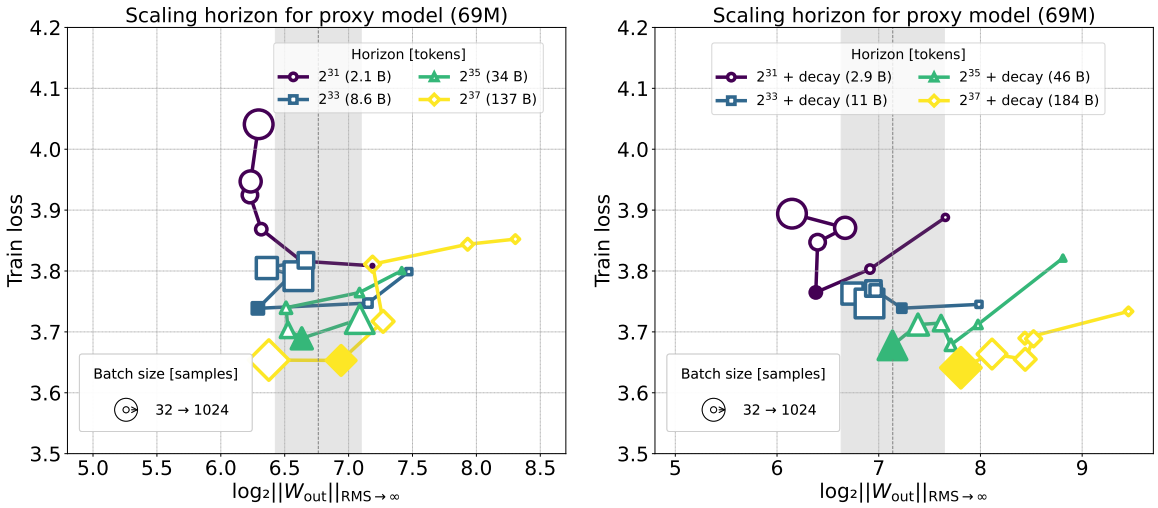


Figure 15: **Same as Fig. 1(*a*) but with momentum = 0.1. (a)** Without learning rate decay. **(b)** With linear learning rate decay to 0 for extra 25% of the total horizon.

Appendix T. Norm choice

In this Section, we ablate if it is only the output layer norm that induces norm transfer. We replot Fig. 1(*a*), with loss smoothing and without fitting, but now where we use $\|.\|_{\text{RMS}\to\text{RMS}}$ norm of the output or $\|.\|_{1\to\text{RMS}}$ of the input layers instead default $\|.\|_{\text{RMS}\to\infty}$ of the output layer. We observe in Fig. 16 (see also individual norm scans in Fig. 20) that interestingly both norms induce norm transfer.

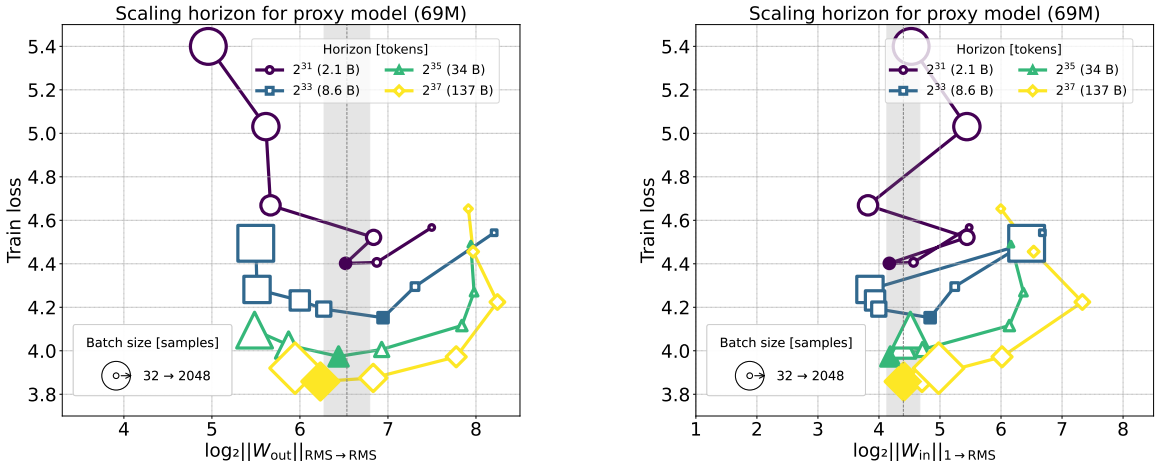


Figure 16: **Same as Fig. 1(*a*) but with $\|W_{\text{out}}\|_{\text{RMS}\to\infty}$ norm for the X-axis changed to: (a)** $\|W_{\text{out}}\|_{\text{RMS}\to\text{RMS}}$ (output layer). **(b)** $\|W_{\text{in}}\|_{1\to\text{RMS}}$ (input layer).

Appendix U. Supplementary plots to Fig. 1

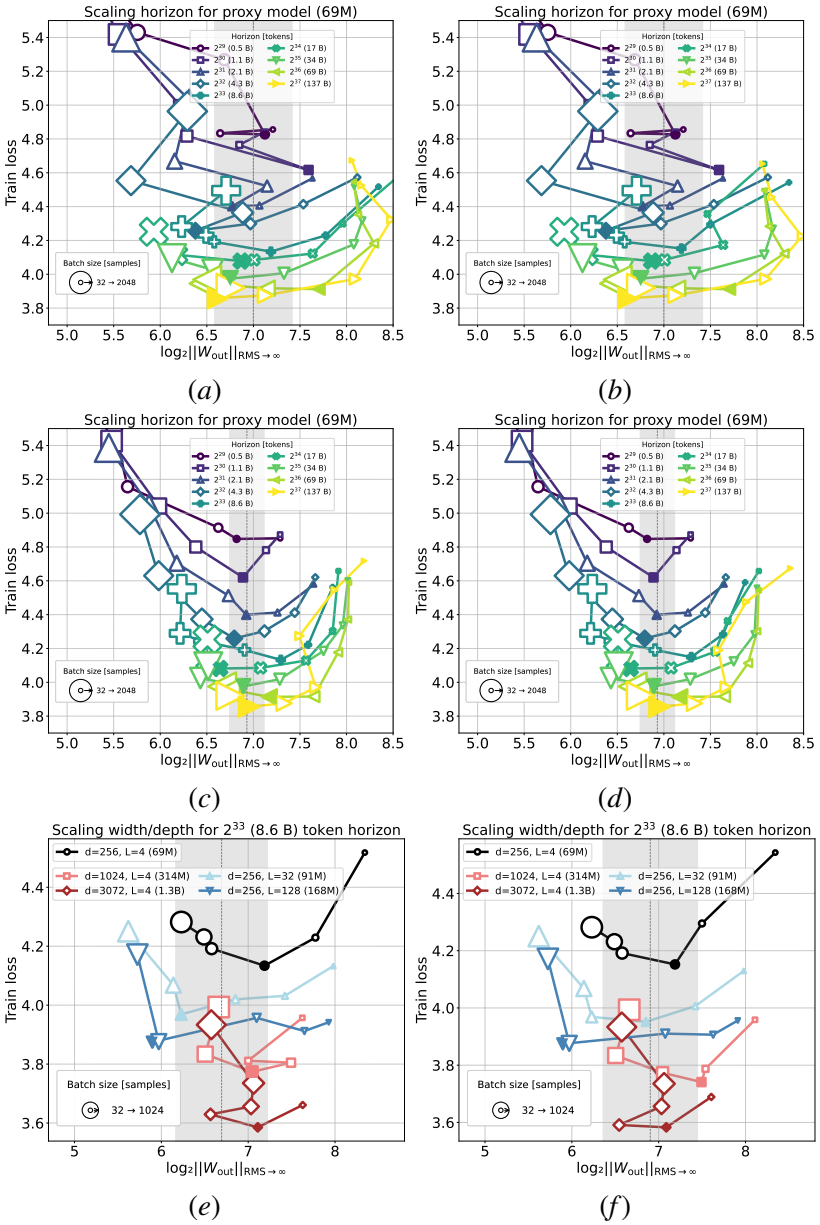


Figure 17: **(a)** Fig. 1(*a*) with an extended set of horizons, raw data (i.e. no loss smoothing, no fitting, see Appendix F for details on fitting and loss smoothing). **(b)** Same as (a) + loss smoothing. **(c)** Same as (a) + fitting. **(d)** Same as (a) + fitting + loss smoothing. **(e)** Fig. 1(*b*), raw data (no loss smoothing, no fitting). **(f)** Same as (e) + loss smoothing.

Appendix V. Individual norm scans and fits

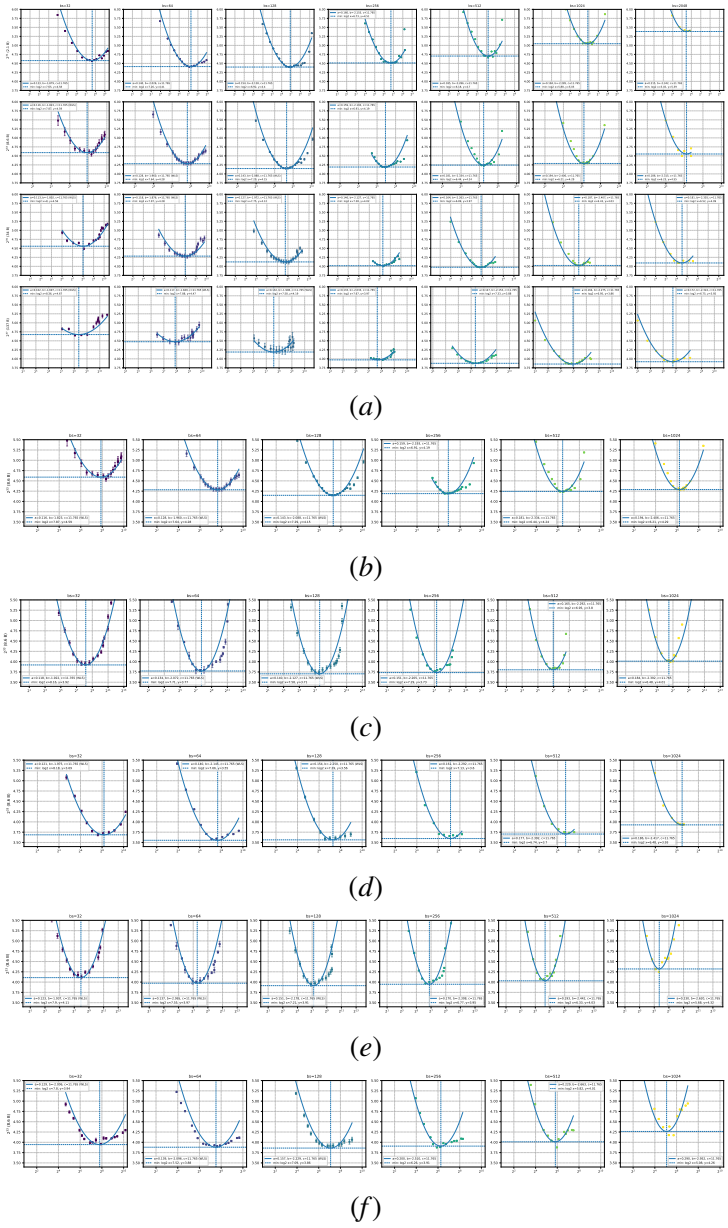
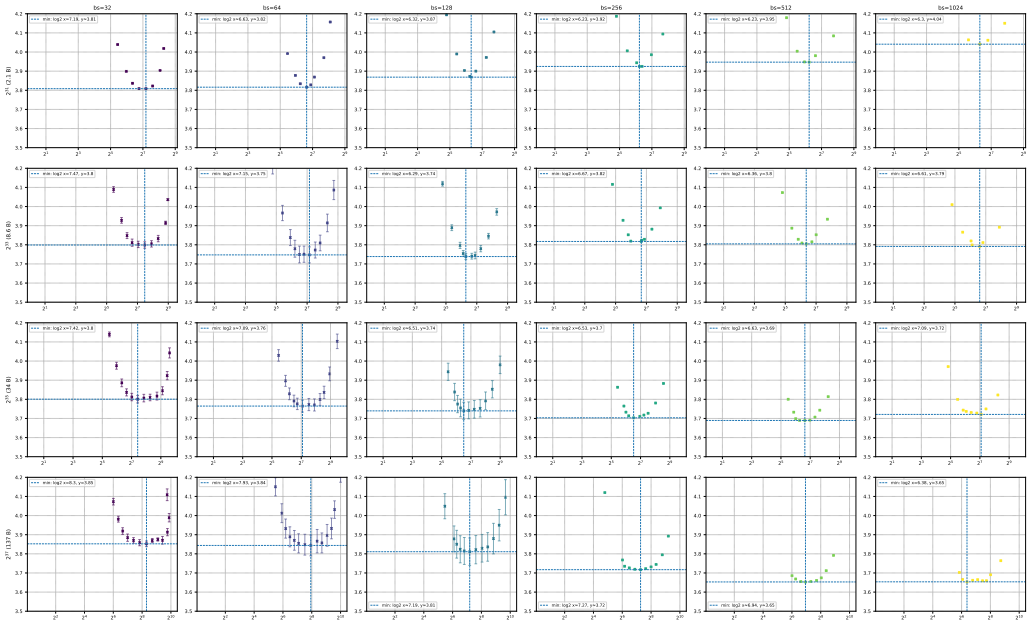
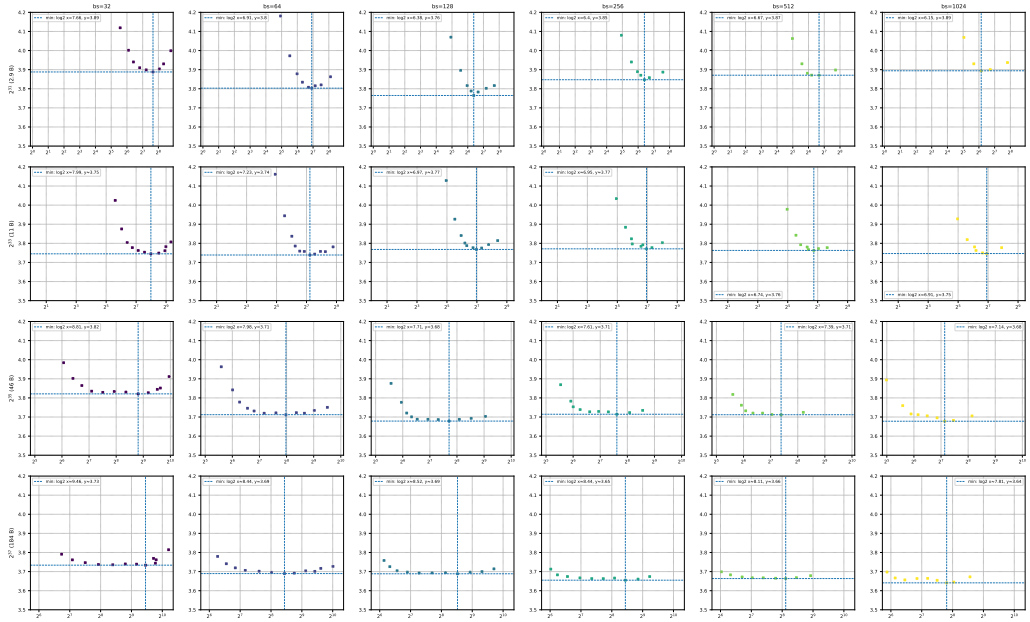


Figure 18: **Individual norm scans for various batch sizes B (columns), across various horizons D in (a), across various models in (b) (rows).** We plot train loss (Y-axis) against the output layer operator norm $\|W_{\text{out}}\|_{\text{RMS}\to\infty}$, where each point corresponds to a different learning rate run and error bars correspond to loss smoothing variance (see Appendix F). The best-loss point for each (B, D) is pinpointed with the blue dashed line, fitted curves are shown with blue solid lines. These fit results are used for: **(a)** Fig. 1(a) and Fig. 3, **(b)** Fig. 1(b), from top to bottom rows: proxy, $\times$4-width, $\times$12-width, $\times$8-depth, $\times$32-depth.

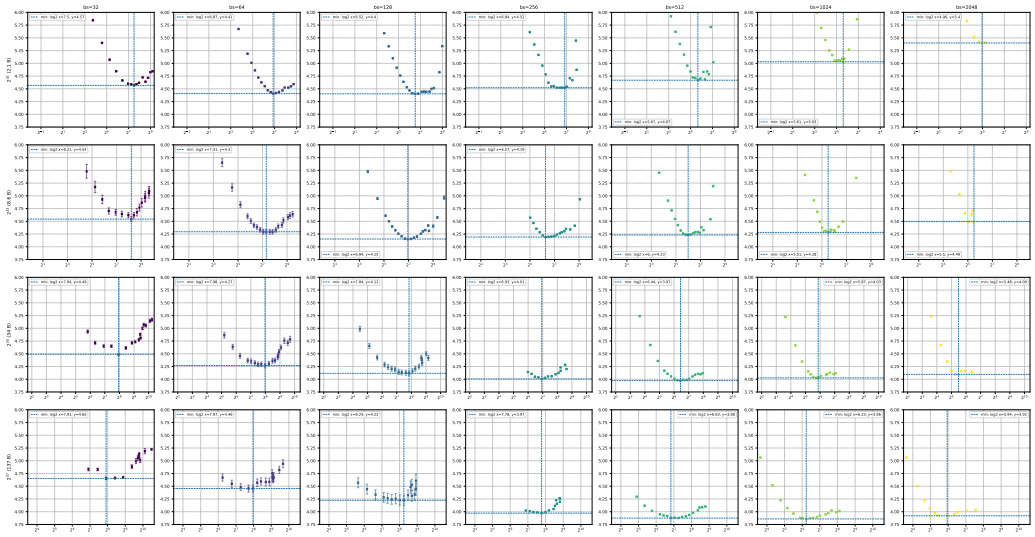


(*a*)

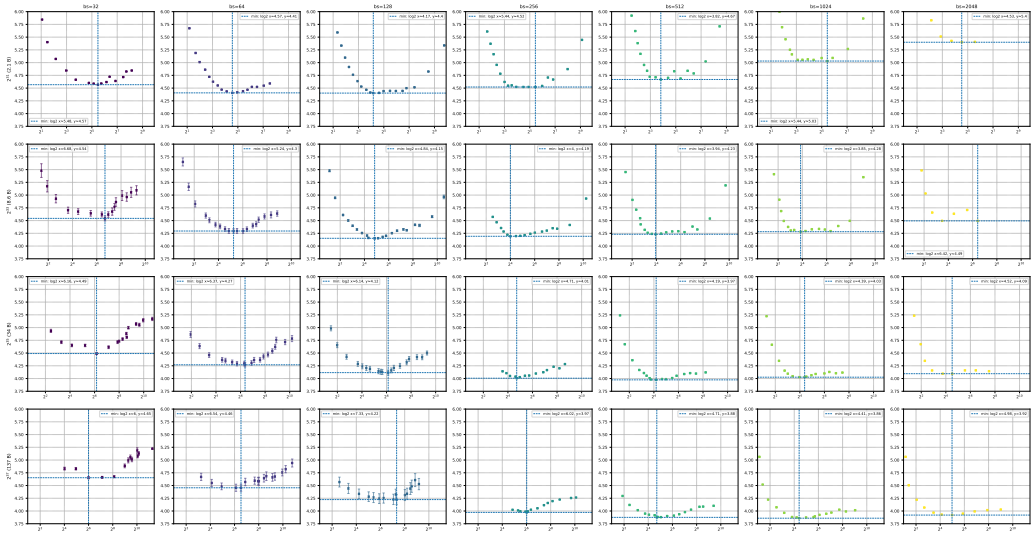


(*b*)

Figure 19: **Individual output norm $\|W_{\text{out}}\|_{\text{RMS}\to\infty}$ scans for various batch sizes B (columns) across various horizons D (rows). (a)** with momentum = 0.1, no learning rate decay. **(b)** with momentum = 0.1, with learning rate decay linearly to 0 for 25% of total horizon.



(*a*)



(*b*)

Figure 20: **Individual norm scans for various batch sizes B (columns) across various horizons D (rows). (a)** For $\|W_{\text{out}}\|_{\text{RMS}\to\text{RMS}}$ (output layer). **(b)** For $\|W_{\text{in}}\|_{1\to\text{RMS}}$ (input layer).