u-μP: The Unit-Scaled Maximal Update Parametrization

author names withheld

Under Review for the Workshop on High-dimensional Learning Dynamics, 2024

Abstract

The recent Maximal Update Parametrization (μP) enables the hyperparameters for small models to transfer directly to large ones, substantially reducing the cost of training by avoiding expensive sweeps at scale. We present a new scheme, u-μP, which improves upon μP by combining it with Unit Scaling, a method for designing models that makes them easy to train in low-precision. The two techniques have a natural affinity: μP ensures that the scale of activations is independent of model size, and Unit Scaling ensures that the starting-scale of these activations is one (along with weights and gradients). This synthesis opens the door to a simpler scheme, whose default values are near-optimal. This in turn facilitates a more efficient sweeping strategy, with u-μP models reaching a lower loss than comparable μP models and working out-of-the-box in FP8.

1. Introduction

Finding good hyperparameters (HPs) is critical to effective training, yet doing so for modern large language models (LLMs) is challenging. The huge scale of models and datasets makes performing multiple runs over a set of candidate HPs prohibitively expensive. The Maximal Update Parametrization (μP) aims to make HP values *consistent* across model sizes, allowing practitioners to sweep HPs on a small-scale *proxy* model and transfer them to a larger *target* model [27, 30].

μP has become a popular choice for LLM pre-training [4, 5, 9, 13] (though recent leading models have not disclosed this training information). Unfortunately users have not always found

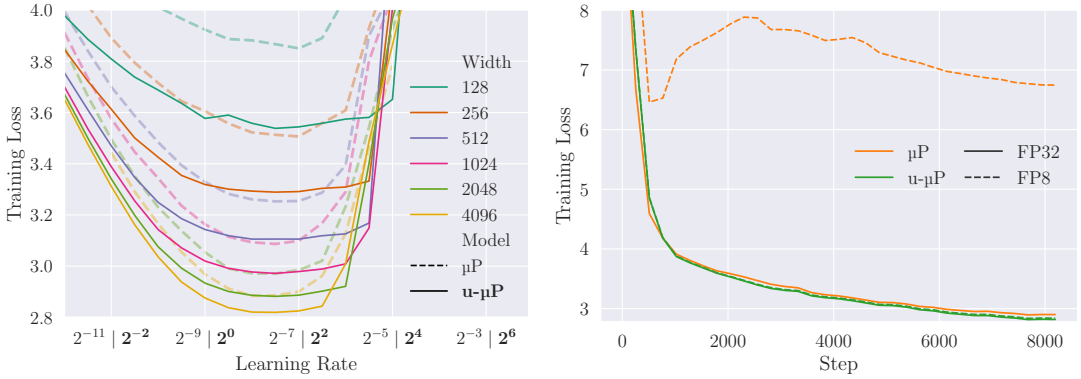

Figure 1: (Left) Using default HPs, u-μP models have lower losses and wider basins than μP models, while still transferring the optimal learning rate across width. (Right) FP8 training via a simple cast. μP uses HPs found by random grid search here, whereas u-μP uses default HPs, only sweeping the LR. Width $= 4096$, $\eta = (2^{-7.5}, 2^{1.5})$ for (μP, u-μP).

μP to provide effective transfer in practice [1, 12], with a gap between the theory and its effective application. We propose an improved scheme: Unit-Scaled Maximal Update Parametrization (u-μP).

Our method combines μP with another recent training innovation, Unit Scaling [3]. Originally designed to facilitate simple low-precision training, Unit Scaling proposes the principle of unit variance at initialization for activations, weights and gradients. In doing so it relies on the same mechanism as μP: applying scalar multipliers to linear layers to counteract the effects of changing model size. This allows for a synthesis of the two methods and facilitates several improvements. u-μP retains the HP-transfer property of μP and the out-of-the-box FP8 training of Unit Scaling (Figure 1), but also simplifies the scaling rules (Table 1) and provides a more principled and interpretable set of HPs (Appendix D). The default values of these HPs are near-optimal, giving lower losses than for μP (Figure 1) and opening up more efficient sweeping strategies (Figure 2, right).

2. Background

2.1. The Maximal Update Parametrization

As model sizes have grown, the behavior of neural networks during training in the limit of infinite width has become an active field of research. One of the most important results in this area is the classification of infinite-width limits under all possible *abc-parametrizations* [27]. An abc-parametrization assumes that the training dynamics for a weight W of a model are given by

$$W(t) = Aw(t), \quad w(0) \sim \mathcal{N}(0, B), \quad w(t+1) - w(t) = C\Phi_t(\nabla\mathcal{L}_0, ..., \nabla\mathcal{L}_t),$$

and provides a scheme for scaling the weight multiplier A, the initial variance B, and the learning rate C with some exponent of the network width.

The authors prove that the only abc-parametrization (up to symmetry) that allows all model features to evolve non-trivially in the infinite-width limit without blowing up, is μP. We outline μP's parametrization rules in Table 1. This has significant implications for training at scale, as it suggests that non-μP approaches will eventually be ineffective.

Another consequence of μP is that optimal hyperparameters transfer between different model sizes, a process known as μTransfer [30]. This is not the case under the Standard Parametrization (SP) and has led to its adoption for LLM training. Yang et al. [31] introduce a similar scheme for depth known as depth-μP[1], based on residual branch multipliers and learning rates (also see Table 1).

2.2. Unit Scaling

Unit Scaling is a paradigm to facilitate training with low-precision number formats. The use of these formats on modern AI hardware can bring substantial efficiency gains [7, 18], but requires extra care to guarantee stable numerics. As explained in Section E, existing efforts on FP8 training employ per-tensor scaling techniques, which are cumbersome to implement and incur additional overheads.

Unit Scaling takes a different approach, offering a paradigm for designing models that makes them inherently less likely to generate out-of-range values during training. It does so by ensuring that all activations, weights and gradients have *unit scale* (i.e. $\sigma = 1$) at initialization. As a consequence, unit scaled models can be trained in low-precision via a simple cast operation.

This is achieved through the application of scalar multipliers to every operation in the forward and backward passes, counteracting the scale-changing effect of operations like matrix multiplications.

1. When we refer to μP elsewhere in the paper we implicitly assume the modifications introduced in depth-μP.

For example, the rule when multiplying a vector by a square matrix of length d is to re-scale the result by $1/\sqrt{d}$.

3. Combining μP with Unit Scaling

In this section we derive our new u-μP scheme. The final set of scaling rules is given in Table 1.

Table 1: **The μP and u-μP schemes (expressed in absolute terms)**

	Weight type	Input	Output	Hidden	Residual
μP	Init. Var.	σ_{init}^2	σ_{init}^2	$\sigma_{\text{init}}^2 \frac{\text{base-fan-in}}{\text{fan-in}}$	—
	Multiplier	1	$\frac{\text{base-fan-in}}{\text{fan-in}}$	1	$\sqrt{\frac{\text{base-depth}}{\text{depth}}}$ *
	Adam LR Mult.	1	1	$\frac{\text{base-fan-in}}{\text{fan-in}}$	$\sqrt{\frac{\text{base-depth}}{\text{depth}}}$
u-μP	Init. Var.	1	1	1	—
	Multiplier	1	$\frac{1}{\text{fan-in}}$ †	$\frac{1}{\sqrt{\text{fan-in}}}$	$\frac{1}{\sqrt{\text{depth}}}$ *
	Adam LR Mult.	$\frac{1}{\sqrt{\text{fan-out}}}$	1	$\frac{1}{\sqrt{\text{fan-in}}}$	$\frac{1}{\sqrt{\text{depth}}}$
μP	Associated HPs	(8): $\eta, \hat{\eta}_{\text{emb}}, \text{base-width}, \text{base-depth}, \sigma_{\text{init}}, \alpha_{\text{emb}}, \alpha_{\text{attn}}, \alpha_{\text{output}}$			
u-μP	Associated HPs	(6): $\eta, \alpha_{\text{residual}}, \alpha_{\text{residual-attn-ratio}}, \alpha_{\text{ffn-act}}, \alpha_{\text{attn}}, \alpha_{\text{output}}$			

*Residual mults are applied to the end of each branch, rather than the output of linear layers.
†To maintain unit scale we apply $1/\sqrt{\text{fan-in}}$ scaling in the backward pass (see Appendix A.4).

3.1. From relative to absolute scaling

We seek to implement the rules specified by μP in a way that maintains unit scale for all tensors at initialization. Our first challenge is that whereas Unit Scaling provides *absolute* initialization values and multipliers, μP only specifies a *parametrization*. Yang et al. [30] define a parametrization as 'a rule for how to change hyperparameters when the widths of a neural network change, [not] how to set the hyperparameters for any specific width'. To satisfy Unit Scaling we require our u-μP scheme to provide absolute scaling rules[2], so our first step is to derive an absolute-value implementation of μP. We do this by combining μP's 'base shapes' and initialization HPs with its standard scaling rules, resulting in Table 1.

The non-unit initialization of μP here violates Unit Scaling. However, due to the symmetry of abc-parametrizations (see Section 2.1), we can move scales between the initialization, multiplier and learning rate to attain any desired initialization, choosing variance of 1 to give unit-scaled weights. This results in the intermediate scheme shown in Table 2, Appendix A.1.

We impose two further changes to simplify the scheme and facilitate better transfer: dropping the 'base shapes' and σ_{init} HPs, and changing the input LR rule to $1/\sqrt{\text{fan-out}}$ (these changes are justified in Appendices A.2 and A.3). This results in the final u-μP scheme given in Table 1.

The input and hidden multiplier rules that result from this scheme are exactly those specified by Unit Scaling (a reflection of its close relationship with μP). The u-μP output multiplier breaks Unit Scaling temporarily, but this is not a problem in practice as output mis-scaling does not propagate in

2. In this sense u-μP is not strictly a *parametrization*, but we retain the term for simplicity.

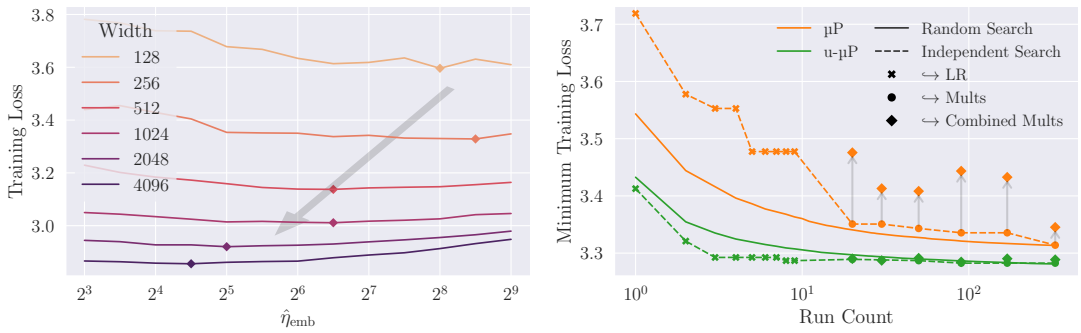

Figure 2: (Left) μP width transfer of the input LR multiplier $\hat{\eta}_{\text{emb}}$, showing poor transfer and motivating the change to $1/\sqrt{\text{fan-out}}$ scaling in u-μP. (Right) a comparison of random versus independent HP search strategies on μP and u-μP, showing that u-μP parameters are more amenable to independent search, and that combining the results of independent searches of μP mults harms performance.

the forward pass and output gradients can be corrected by $1/\sqrt{\text{fan-in}}$ scaling thanks to the *cut-edge rule* (see Appendix A.4). The only other point at which Unit Scaling is violated is the $1/d_{\text{head}}$ attention scaling, but this again is temporary. Otherwise, our resulting u-μP scheme has unit-scale weights, gradients and activations, while satisfying μP.

3.2. A maximally independent set of hyperparameters

The original set of μP HPs in Table 1 is convoluted and there is complex interplay and redundancy between some HPs (see Appendix F.1), in the sense that they effectively control the same scale in the model. We aim for a set of HPs that is maximally expressive while having no such redundancy. Besides the global learning rate parameter η, we end up with the following HPs:

- Residual branch multipliers: α_{residual} and ratio $\alpha_{\text{residual-attn-ratio}}$ (see eq. (19)).
- Multipliers before non-homogeneous operations: α_{attn}, $\alpha_{\text{ffn-act}}$, $\alpha_{\text{loss-softmax}}$ (see eq. (25)).

This set of HPs, although smaller than the set of original μP HPs, generates a rich set of training dynamics. Furthermore, it has no redundancy and each multiplier controls a meaningful scale in the model. For a more detailed explanation, see Appendix D.

4. Experiments

We provide empirical results to support our claim that u-μP's HPs facilitate more efficient sweeping than μP while retaining good HP transfer, and demonstrate that it provides out-of-the-box FP8 training without dynamic scaling. Our experiments use the Llama [23] architecture, trained on WikiText-103 [15] and evaluated using final training cross-entropy loss. Further details on experiments are given in Appendix F, and a guide to new unit-scaled ops required for this architecture in Appendix B.

Hyperparameter search The set of u-μP HPs has been chosen such that their effects should be independent (see Appendix F.1), with their default ($= 1$) values also providing unit-scale inputs to associated functions. To test the benefit of this, we evaluate two sweeping strategies on μP and u-μP. First, a random grid search following [4, 5, 9, 30], which should perform well in the presence of HP interactions. Second, an independent search which performs a line search for each HP in parallel, then combines the optima. Figure 2 (right) shows the results: u-μP is amenable to both

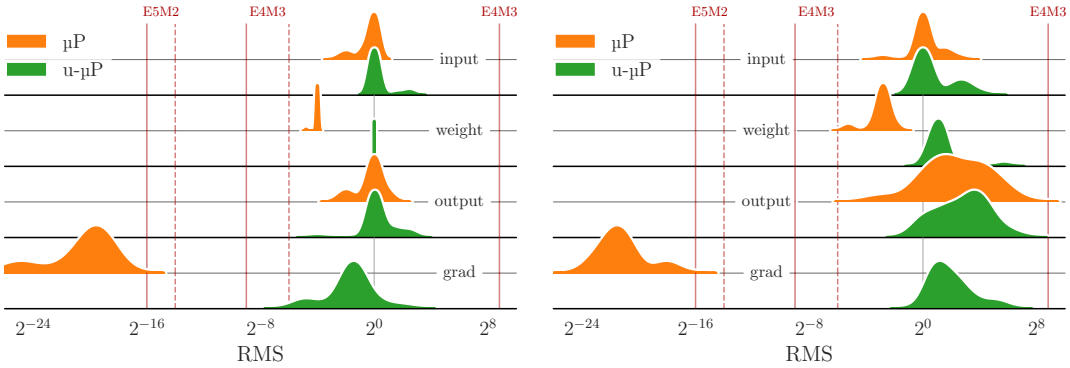

Figure 3: Per-tensor RMS $= \sqrt{\sigma^2 + \mu^2}$ across u-μP and μP models at initialization (left) and after training (right). u-μP tensors have RMS that starts close to 1 and remains within E4M3 range at the end of training. Dashed and solid red lines show each format's min. normal and subnormal values.

methods, while μP does not perform well under independent search. Hence u-μP enables more efficient sweeping strategies than μP, with near-optimal loss after just the LR portion of the search.

Hyperparameter transfer Figure 1 (left) and Figure 6 demonstrate that u-μP retains the learning rate transfer properties of μP. Using the default mult values, u-μP models are able attain the optimal loss for μP models of twice the width. Figure 2 (left), Figures 4 and 5 show that u-μP's change to the input scaling rule improves width transfer and absolute loss.

Numerical properties u-μP's scaling rules and multipliers are designed to maintain unit variance of activations, weights and gradients at initialization. Hence, these tensors are initialized well within the range of FP8 formats and, empirically, do not drift out of range during training (see Figure 3). Based on this, we propose and test a simple proof-of-concept scheme for FP8 training with u-μP. For every linear module, we cast input, weight and grad-output tensors to FP8—without any scaling—prior to the matrix multiplication. Some extra care is required for the inputs to FFN and self-attention final projections; the details of our scheme may be found in Appendix E. Figure 1 (right) and Figure 7 show that FP8 u-μP converges close to the baseline value, while μP fails to train.

5. Related work

The Tensor Programs series of papers [24–31] introduce the mathematical framework from which μP is derived. [4, 5, 9, 13] employ μP to derive the HPs for LLM training, with [12] exploring the transfer properties of μP variants. [3] introduce Unit Scaling, which has similarities to [10, 33] in its activation function scaling, and to [6, 8] in its approach to initialization.

6. Conclusions

We demonstrate the effectiveness of our improved scheme, which applies the principles of Unit Scaling to μP to form u-μP. We retain the HP transfer property of μP and benefit from the simple low-precision training brought by Unit Scaling. This allows FP8 via a simple cast, bringing substantial efficiency gains. Our u-μP HP scheme is more principled than that used for μP, leading to a simpler sweeping strategy. Indeed the default multipliers associated with u-μP models are near-optimal in our experiments, reaching a lower loss than heavily-tuned μP models. The combination of these benefits indicates that u-μP can be a valuable component in the simple, stable and efficient training of LLMs.

References

[1] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. The Falcon series of open language models. *CoRR*, abs/2311.16867, 2023. doi: 10.48550/ARXIV.2311.16867. URL https://doi.org/10.48550/arXiv.2311.16867.

[2] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. volume 116, page 15849–15854. Proceedings of the National Academy of Sciences, July 2019. doi: 10.1073/pnas.1903070116. URL http://dx.doi.org/10.1073/pnas.1903070116.

[3] Charlie Blake, Douglas Orr, and Carlo Luschi. Unit scaling: Out-of-the-box low-precision training. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 2548–2576. PMLR, 2023. URL https://proceedings.mlr.press/v202/blake23a.html.

[4] Nolan Dey, Gurpreet Gosal, Zhiming Chen, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, and Joel Hestness. Cerebras-GPT: Open compute-optimal language models trained on the cerebras wafer-scale cluster. *CoRR*, abs/2304.03208, 2023. doi: 10.48550/ARXIV.2304.03208. URL https://doi.org/10.48550/arXiv.2304.03208.

[5] Nolan Dey, Daria Soboleva, Faisal Al-Khateeb, Bowen Yang, Ribhu Pathria, Hemant Khachane, Shaheer Muhammad, Zhiming Chen, Robert Myers, Jacob Robert Steeves, Natalia Vassilieva, Marvin Tom, and Joel Hestness. BTLM-3B-8K: 7B parameter performance in a 3B parameter model. *CoRR*, abs/2309.11568, 2023. doi: 10.48550/ARXIV.2309.11568. URL https://doi.org/10.48550/arXiv.2309.11568.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and D. Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pages 249–256. JMLR.org, 2010. URL http://proceedings.mlr.press/v9/glorot10a.html.

[7] Graphcore. C600 IPU-Processor PCIe Card, 2022. URL https://www.graphcore.ai/products/c600.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society, 2015. doi: 10.1109/ICCV.2015.123. URL https://doi.org/10.1109/ICCV.2015.123.

[9] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, Xinrong Zhang, Zhen Leng Thai, Kai 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. *CoRR*, abs/2404.06395, 2024. doi: 10.48550/ARXIV.2404.06395. URL https://doi.org/10.48550/arXiv.2404.06395.

[10] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 971–980, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/5d44ee6f2c3f71b73125876103c8f6c4-Abstract.html.

[11] Oleksii Kuchaiev, Boris Ginsburg, Igor Gitman, Vitaly Lavrukhin, Carl Case, and Paulius Micikevicius. OpenSeq2Seq: extensible toolkit for distributed and mixed precision training of sequence-to-sequence models. *CoRR*, abs/1805.10387, 2018. URL http://arxiv.org/abs/1805.10387.

[12] Lucas D. Lingle. A large-scale exploration of μ-transfer. *CoRR*, abs/2404.05728, 2024. doi: 10.48550/ARXIV.2404.05728. URL https://doi.org/10.48550/arXiv.2404.05728.

[13] Zhengzhong Liu, Aurick Qiao, Willie Neiswanger, Hongyi Wang, Bowen Tan, Tianhua Tao, Junbo Li, Yuqi Wang, Suqi Sun, Omkar Pangarkar, Richard Fan, Yi Gu, Victor Miller, Yonghao Zhuang, Guowei He, Haonan Li, Fajri Koto, Liping Tang, Nikhil Ranjan, Zhiqiang Shen, Xuguang Ren, Roberto Iriondo, Cun Mu, Zhiting Hu, Mark Schulze, Preslav Nakov, Tim Baldwin, and Eric P. Xing. LLM360: towards fully transparent open-source llms. *CoRR*, abs/2312.06550, 2023. doi: 10.48550/ARXIV.2312.06550. URL https://doi.org/10.48550/arXiv.2312.06550.

[14] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

[15] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Byj72udxe.

[16] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=r1gs9JgRZ.

[17] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellempudi, Stuart F. Oberman, Mohammad Shoeybi, Michael Y. Siu, and Hao Wu. FP8 formats

for deep learning. *CoRR*, abs/2209.05433, 2022. doi: 10.48550/ARXIV.2209.05433. URL https://doi.org/10.48550/arXiv.2209.05433.

[18] Nvidia. Nvidia H100 Tensor Core GPU, 2022. URL https://www.nvidia.com/en-gb/data-center/h100.

[19] NVIDIA. Using fp8 with transformer engine. https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/examples/fp8_primer.html, 2024.

[20] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=Hkuq2EkPf.

[21] Noam Shazeer. GLU variants improve transformer. *CoRR*, abs/2002.05202, 2020. URL https://arxiv.org/abs/2002.05202.

[22] Jianlin Su, Murtadha H. M. Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568: 127063, 2024. doi: 10.1016/J.NEUCOM.2023.127063. URL https://doi.org/10.1016/j.neucom.2023.127063.

[23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023. doi: 10.48550/ARXIV.2302.13971. URL https://doi.org/10.48550/arXiv.2302.13971.

[24] Greg Yang. Tensor programs I: wide feedforward or recurrent neural networks of any architecture are gaussian processes. *CoRR*, abs/1910.12478, 2019. URL http://arxiv.org/abs/1910.12478.

[25] Greg Yang. Tensor programs II: neural tangent kernel for any architecture. *CoRR*, abs/2006.14548, 2020. URL https://arxiv.org/abs/2006.14548.

[26] Greg Yang. Tensor programs III: neural matrix laws. *CoRR*, abs/2009.10685, 2020. URL https://arxiv.org/abs/2009.10685.

[27] Greg Yang and Edward J. Hu. Tensor programs IV: feature learning in infinite-width neural networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 11727–11737. PMLR, 2021. URL http://proceedings.mlr.press/v139/yang21c.html.

[28] Greg Yang and Etai Littwin. Tensor programs IIb: Architectural universality of neural tangent kernel training dynamics. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 11762–11772. PMLR, 2021. URL http://proceedings.mlr.press/v139/yang21f.html.

[29] Greg Yang and Etai Littwin. Tensor programs IVb: Adaptive optimization in the infinite-width limit. *CoRR*, abs/2308.01814, 2023. doi: 10.48550/ARXIV.2308.01814. URL https://doi.org/10.48550/arXiv.2308.01814.

[30] 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. *CoRR*, abs/2203.03466, 2022. doi: 10.48550/ARXIV.2203.03466. URL https://doi.org/10.48550/arXiv.2203.03466.

[31] Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs VI: feature learning in infinite-depth neural networks. *CoRR*, abs/2310.02244, 2023. doi: 10.48550/ARXIV.2310.02244. URL https://doi.org/10.48550/arXiv.2310.02244.

[32] Chao Yu and Zhiguo Su. Symmetrical gaussian error linear units (sgelus). *CoRR*, abs/1911.03925, 2019. URL http://arxiv.org/abs/1911.03925.

[33] Peiwen Yuan and Changsheng Zhu. Normalized activation function: Toward better convergence. *CoRR*, abs/2208.13315, 2022. doi: 10.48550/ARXIV.2208.13315. URL https://doi.org/10.48550/arXiv.2208.13315.

[34] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12360–12371, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html.

Contents

Appendix A. From μP and Unit scaling to u-μP

Although u-μP provides the benefits of both μP and u-μP and is faithful to almost all of their rules, it includes some specific changes that deviate from the parent schemes. In this section we give additional details about u-μP as well as highlighting and explaining the changes from μP and Unit Scaling.

A.1. An intermediate scheme

The first step from μP to u-μP is enabled by the abc-symmetry (see Lemma J.1 in [30]). The initial hidden weight variance $\text{base-fan-in}/\text{fan-in}$ from Table 1 can be replaced by 1 if we change at the same time:

- Multiplier: $1 \rightarrow \sqrt{\frac{\text{base-fan-in}}{\text{fan-in}}}$
- Adam LR Mult.: $\frac{\text{base-fan-in}}{\text{fan-in}} \rightarrow \sqrt{\frac{\text{base-fan-in}}{\text{fan-in}}}$

Together with setting $\sigma_{\text{init}} = 1$ we get the intermediate scheme shown in Table 2.

Table 2: **μP scheme after abc symmetry and normalized init. variance**

	Weight type	Input	Output	Hidden	Residual
μP	Init. Var.	1	1	1	—
	Multiplier	1	$\frac{\text{base-fan-in}}{\text{fan-in}}$	$\sqrt{\frac{\text{base-fan-in}}{\text{fan-in}}}$	$\sqrt{\frac{\text{base-depth}}{\text{depth}}}$
	Adam LR Mult.	1	1	$\sqrt{\frac{\text{base-fan-in}}{\text{fan-in}}}$	$\sqrt{\frac{\text{base-depth}}{\text{depth}}}$

A.2. Dropping base shapes

The next step in our transition from the μP scheme in Table 2 to our u-μP scheme in Table 1 is the removal of the 'base shapes' HPs, base-fan-in and base-depth. It should be noted that although we refer to base-fan-in and base-depth in the same way as the multiplier HPs, these are not intended to be swept, but rather chosen according to taste. The stated purpose of the 'base shapes' in Yang et al. [30] is to enable a model to behave unchanged at a particular size, yet still scale according to μP rules as the model-size changes.

We argue that this is not necessary or desirable for u-μP based on the following:

1. Though in some cases it may be useful to align a μP model to some 'base' model in the manner described above, all uses of μP in the literature simply set the base shapes those of a small model and perform an HP sweep on top of it. This sweep alters the dynamics of the base model before scaling-up, so there is little sense in which the base shapes maintain the behavior of some smaller model as we scale. At best the base shapes can be seen as giving a useful starting point to the HP sweep.

2. In contrast u-μP offers a *principled* approach to the base dynamics of the model: that all tensor-scales should have unit variance. This can be seen as an alternative to the 'base shapes' approach, which substitutes this principle for alignment with non-μP models at a particular scale.

3. The effect of base shapes on the weight multipliers is a constant. Our set of u-μP multipliers is able to express any set of constant weight multipliers in the network, so our HP sweep is in effect testing what might happen were we to introduce base shapes into u-μP.

4. The effect of base shapes on the LR multipliers is again a constant factor, which is applied to all hidden weights for width, and residual weights for depth. This applies to the large majority of weights in the model in a similar fashion, making it closely linked to a global shift in the LR. As the LR is swept anyway, we conjecture that the effect of base shapes on the model's LRs is minimal.

5. Finally, we argue for simplicity. If we can remove these additional HPs without detriment, then our scaling rules become much simpler: each depends on a single factor.

With this change implemented, our scaling rules become those shown in Table 3.

Table 3: **μP scheme after abc symmetry, normalized init. variance and dropping of base shapes**

	Weight type	Input	Output	Hidden	Residual
μP	Init. Var.	1	1	1	—
	Multiplier	1	$\frac{1}{\text{fan-in}}$	$\frac{1}{\sqrt{\text{fan-in}}}$	$\frac{1}{\sqrt{\text{depth}}}$
	Adam LR Mult.	1	1	$\frac{1}{\sqrt{\text{fan-in}}}$	$\frac{1}{\sqrt{\text{depth}}}$

A.3. A new embedding scaling rule

Our only fundamental deviation from μP in terms of scaling is changing the embedding LR ($\hat{\eta}_{\text{emb}}$) from not being scaled to being scaled by $\sqrt{1/\text{fan-out}}$ (i.e. $\sqrt{1/\text{width}}$). This is based on an empirical observation: in our u-μP experiments we noticed that the embedding learning rate parameter does not transfer well with width.

This is shown in Figure 4 (left), where our best $\hat{\eta}_{\text{emb}}$ at the 256 width ($2^{-2.5}$) does not transfer to the 2048 width. However, applying a $\sqrt{1/\text{width}}$ scaling arrives at the optimal LR for 2048 width (2^{-4}). The same experiment with our new scheme is shown in Figure 4 (right). This effect can be seen more strongly with μP, which we demonstrate in the body of the paper in Figure 2 (left).

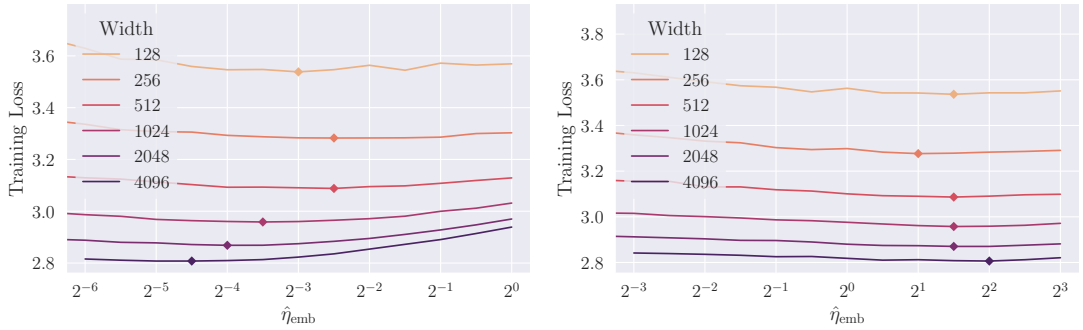

Figure 4: Transfer of the input LR multiplier $\hat{\eta}_{\text{emb}}$ over width. (Left) Modified u-μP *without* our $1/\sqrt{\text{fan-out}}$ input LR scaling rule. (Right) Standard u-μP *with* our $1/\sqrt{\text{fan-out}}$ input LR scaling rule.

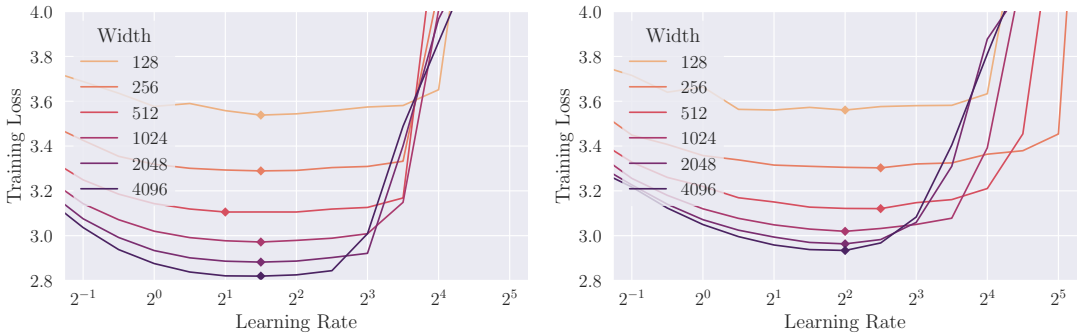

Figure 5: LR transfer over width. (Left) Standard u-μP *with* our $1/\sqrt{\text{fan-out}}$ input LR scaling rule. (Right) Modified u-μP *without* our $1/\sqrt{\text{fan-out}}$ input LR scaling rule. Without our proposed rule, u-μP would have worse transfer and absolute loss at large width.

Without this rule-change u-μP models have degraded performance, as demonstrated in Figure 5. Note that this phenomenon of poor *embedding* LR transfer only has a minor effect on the *global* LR transfer, but significantly degrades the loss at scale. We have observed similar issues for μP models, where sweeping an embedding LR multiplier can also impair the performance of the scaled-up model. We leave a potential theoretical explanation of why the original μP rules do not give transfer here to future work. It may be the case that at much larger scales the original rule is more appropriate, though this is unlikely to be of practical use.

A.4. Forward multipliers vs. backward multipliers

Table 1 shows the scaling rules for weight multipliers in u-μP. If we are more precise, these values are actually post-op forward multipliers, i.e. a multiplier α in a linear layer f gets applied as $h \mapsto \alpha \cdot f(h)$. This is because for low-precision training, we cast the input h to FP8 in the forward pass and a pre-op multiplication $\alpha \cdot h$ might go out of FP8 range if the multiplier is too small or too large. Conversely, we want α to act as a pre-op multiplier in the backward pass, so we have $g_h \mapsto \alpha \cdot \text{grad}_f(g_h)$ because in this case the gradient g_h gets cast to FP8. The same logic applies to residual multipliers, with the branch multiplier implemented after the residual path in the forward pass, and before the residual path in the backward pass.

This is straightforward when the forward and backward multipliers are the same, but nevertheless crucial to implement correctly in order to enable stable FP8 forward and backward passes. However, there are two instances in u-μP where forward and backward multiplier are different:

1. **Weight gradients.** While model weights under Unit Scaling are correctly scaled in the forward pass, the weight gradient computation involves a summation over the batch size b, hence from a Unit Scaling perspective we need to apply a scaling in b to keep the weight gradient unit scaled. We choose the scaling

$$g_w \mapsto b^{-\frac{1}{2}} g_w,$$

that works well in practice. The discrepancy between forward and backward computation can be easily resolved post-hoc in the optimizer function that calculates the weight update from

its gradients. In the case of Adam, no adjustment is needed because of its scale invariance property.

2. **Readout layer**. In order to satisfy μP and prevent logits from blowing up as the network width increases, the readout layer has a forward multiplier of $1/\text{fan-in}$. This contradicts Unit Scaling, but is not too problematic in the forward pass since we are only under-scaled at the first step of training and then have logits of order 1. Also, this happens at the very end of the network and has no significant effect on any subsequent operations, since the loss computation usually stays in higher precision. In the backward pass however, using this multiplier leads to all gradients in the model becoming under-scaled throughout training. The fix is to use the backward multiplier $\sqrt{1/\text{fan-in}}$ instead. Again, this produces "mathematically incorrect" gradients. Because the readout layer is not on a residual branch and the backward pass is linear, we can again easily compensate for this static factor in the optimizer.

A unified explanation for why different forward and backward scale are admissible for weights and readout layer, but not for operations on residual branches, is given by the cut-edge-rule (see Section 5.1 in [3]).

Appendix B. Additional Unit-Scaled Ops

For the sake of our experiments, a set of unit-scaled ops are required for Llama-style [23] transformer architectures. We are able to utilize many of the ops given in the Unit Scaling paper [3]. However, this selection lacks some of the features required to implement modern LLMs. To address this, in this section we outline a series of new unit-scaled ops for each of our required architectural features.

The presentation here is derived from that of the Unit Scaling Compendium given in Blake et al. [3, appendix G]. This makes reference to the factors $\alpha, \beta_1, \ldots, \beta_k$. α is the output scaling factor in the forward pass, and β_i are the scaling factors for the gradient of the op's inputs in the backward pass. For each op, a value or rule is provided for determining the required mult to ensure unit scale. The correct value for these multipliers is derived by analyzing the scaling behavior of each op, given some reasonable distributional assumptions about the input and incoming gradient tensors.

We provide a summary of these results in Table 4, which can be seen as an extension of Table A.2. from the Unit Scaling paper.

Unit scaled dot-product attention The Unit Scaling paper considers the attention layer scaling in terms of its separate components: the various matmul operations and the internal softmax. Linear operations are scaled using the standard rule, and the softmax scaling is given a $\alpha = \beta = s$ factor.

From an implementation perspective, the self-attention layer is more typically broken down into weight-matmuls and a fused scale-dot-product attention operation. This is the case we handle here, accounting for three complicating factors not considered in the Unit Scaling paper:

1. As we use a decoder-style transformer in our experiments, our softmax operation has a causal mask applied to its input.

2. We follow the μP guidance of using $1/d_{\text{head}}$ scaling in our self-attention layer, rather than the usual $1/\sqrt{d_{\text{head}}}$.

3. We place a α_{attn} multiplier immediately before the softmax, which is an HP that users may tune.

Table 4: Table of unit scaling factors for ops required for Llama, building on Table A.2. from the Unit Scaling paper [3].

Op	Unit scaling factors
$\text{attention}(q, k, v) =$ $\quad \text{softmax}\left(\alpha_{\text{attn}}\, d_{\text{head}}^{-1}\,(qk^\top)\, c_{\text{mask}}\right) v$	$\alpha = \beta_q = \beta_k = \beta_v =$ $\quad 1/\,\text{log\_interpolate}\left(\frac{1}{1+\frac{4d_{\text{head}}}{\alpha_{\text{attn}}^2}}, 1, \sqrt{\frac{\log(s)}{s}}\right)$
$\text{gated\_silu}(x_{\text{in}}, x_{\text{gate}}) =$ $\quad x_{\text{in}} \odot x_{\text{gate}} \odot \text{sigmoid}(\alpha_{\text{ffn-act}}\, x_{\text{gate}})$	$\alpha = \beta_{x_{\text{in}}} = \beta_{x_{\text{gate}}} =$ $\quad 1/\,\text{log\_interpolate}\left(\frac{1}{1+\frac{1}{\alpha_{\text{ffn-act}}^2}}, \frac{1}{\sqrt{2}}, \frac{1}{2}\right)$
$\text{residual\_add}(x_{\text{resid.}}, x_{\text{skip}}) = a\, x_{\text{resid.}} + b\, x_{\text{skip}}$	$a = \sqrt{\frac{\tau}{\tau+1}}$, $b = \sqrt{\frac{1}{\tau+1}}$ (see D.1.2 for full details, inc. values for τ.)
$\text{RoPE}(x)$	$\alpha = \beta = 1$ (i.e. no scaling)
$\text{RMSNorm}(x)$ (non-trainable, see [12])	$\alpha = \beta = 1$ (i.e. no scaling)

As a result our dot-product attention takes the form:

$$\text{attention}(q, k, v) = \text{softmax}\left(\alpha_{\text{attn}} \cdot d_{\text{head}}^{-1} \cdot (q \cdot k^\top) \cdot c_{\text{mask}}\right) \cdot v$$

The addition of an HP before the softmax introduces an additional challenge for Unit Scaling, as our scaling multipliers will need to account for this value when preserving unit scale.

This operation is sufficiently complex that we found an empirical model of its scale to be more accurate than any mathematically-derived rule (future work may consider justifying our model mathematically). We find that the scale of dot-product attention is approximately

$$\sigma(\text{attention}(q, k, v)) = \text{log\_interpolate}\left(\frac{1}{1+\frac{4d_{\text{head}}}{\alpha_{\text{attn}}^2}}, 1, \sqrt{\frac{\log(s)}{s}}\right)$$

where

$$\text{log\_interpolate}(\alpha, b_{\text{upper}}, b_{\text{lower}}) = e^{\alpha \log(b_{\text{upper}}) + (1-\alpha) \log(b_{\text{lower}})}.$$

The corresponding scaling rule is therefore to divide by this factor in both the forward and backward pass, as outlined in Table 4.

SwiGLU FFN Llama uses a SwiGLU [21] layer for its FFN, which introduces two new operations for us to unit-scale: a SiLU [32] (a.k.a. swish [20]) operation and an element-wise multiplication. We take a similar approach to our dot-product attention, and consider unit-scaling the following fused operation:

$$\text{gated\_silu}(x_{\text{in}}, x_{\text{gate}}) = x_{\text{in}} \odot x_{\text{gate}} \odot \text{sigmoid}(\alpha_{\text{ffn-act}}\, x_{\text{gate}})$$

For the surrounding weight-matmuls we follow the standard Unit Scaling rules.

Again, we use an empirical model of the scale of this op, which is surprisingly similar to the dot-product attention model:

$$\sigma(\text{gated\_silu}(x_{\text{in}}, x_{\text{gate}})) = \text{log\_interpolate}\left(\frac{1}{1+\frac{1}{\alpha^2_{\text{ffn-act}}}}, \frac{1}{\sqrt{2}}, \frac{1}{2}\right),$$

dividing through by this factor to get our scaling rule.

Residual layers Our implementation of residual layers for u-μP is more complex than other operations, as adjustments are required to:

1. Make pre-norm residual networks support Unit Scaling (see Appendix C).
2. Introduce our new, principled residual HPs (see Appendix D).

Our residual layer scheme is presented in full in D.1.2. For readers interested in our justification for this, see the sections noted above.

As mentioned in Appendix A.4, we also follow the example of Unit Scaling and delay the application of our residual multiplier in the backward pass to the base of the branch (see Blake et al. [3], Figure 3c). This does not change the model, and enables unit scale to be maintained on the residual branch regardless of the value of the multiplier.

RoPE embeddings We also require a unit-scaled implementation of Rotary Position Embeddings (RoPE [22]), which are applied just before the scaled dot-product attention operation. As RoPE essentially consists of pair-wise rotations of elements by different degrees, we observe no meaningful scale-change as a result of it's application, and hence leave it unchanged.

RMSNorm Following Lingle [12] we opt to use a non-trainable version of RMSNorm [34], in order to facilitate better transfer. As a result, we also leave this operation unchanged. Were a trainable RMSNorm to be used, the recipe would follow closely that of the LayerNorm presented in the original Unit Scaling Compendium.

Scale constraints One final, minor deviation from the scheme outlined in the Unit Scaling paper is the way in which we apply scale constraints (see their Section 5.2). The essence of scale constraints is that for perfect unit scaling, sometimes the ideal scale for the forward pass differs from those in the backward pass. In some special cases (e.g. at the ends of the network) the use of different scales can be valid, but in the general case a single scale must be agreed upon. The solution in the Unit Scaling paper is to use the geometric mean of the forward and backward scales.

We propose instead to simply use the forward scale over the backward scale(s) in these cases. We do so for the following reasons:

1. For these architectures we find empirically that where there is a disparity in ideal forward and backward scales, it is not large.
2. By taking the forward scale, we can ensure strict unit-scale in the forward pass.

The value of the latter point is in terms of what it means for the interpretation of our u-μPmultiplier HPs. Consider the $\alpha_{\text{ffn-act}}$ multiplier; with strict unit scale we can say that the standard deviation of activations immediately before this multiplier is 1. Therefore the standard deviation immediately

after is $\alpha_{\text{ffn-act}}$. As this multiplier is (by design) the last operation before the ffn activation function, we can say that the interpretation of $\alpha_{\text{ffn-act}}$ is simply to set the input standard deviation to the FFN's activation function. Similar arguments can be made for other u-μP multiplier HPs. This interpretation only holds because we use the forward-scale in our constraints.

Appendix C. Unit scaled pre-norm residual layers

The popular pre-norm residual network architecture is simple to implement, but problematic to combine with Unit Scaling. It exhibits scale-growth in the skip-stream at initialization, due to the repeated addition of residual connections without subsequent normalization. Here we present a surprising and useful finding: that for any pre-norm model there exists a mathematically-equivalent model where this scale-growth is eliminated, through the careful re-scaling of residual connections. This is a crucial preliminary step for our u-μP HP scheme, which is discussed in Appendix D (readers only interested in the final result may skip ahead to D.1.2).

C.1. Scale growth in pre-norm residual networks

Let's consider a pre-norm residual network of depth L:

$$R_0(x) = r_0 x, \tag{1}$$

$$R_l(x) = r_l f_l(R_{l-1}(x)) + R_{l-1}(x), \quad l = 1, .., L \tag{2}$$

$$R_{L+1}(x) = f_{L+1}(R_L(x)) \tag{3}$$

with embedding multiplier r_0 and residual branch multipliers r_l for $l = 1, .., L$. To satisfy pre-norm, all f_l are zero-homogeneous functions, i.e. $f_l(\lambda x) = f_l(x)$.

The scale of the skip-stream at initialization as a result of Equation (2) is

$$\sigma(R_l) = \sqrt{r_l^2 \sigma(f_l)^2 + \sigma(R_{l-1})^2} > \sigma(R_{l-1}), \quad l = 1, .., L \tag{4}$$

assuming $r_l^2 \sigma(f_l)^2 > 0$. This shows that scale inevitably grows with the addition of each residual layer.

This scale-growth is clearly incompatible with unit scaling, which aims for $\sigma(R_l) = 1$ for $l = 0, .., L + 1$. In the following we present an elegant solution to this problem making use of a symmetry transformation available in pre-norm residual architectures.

C.2. Residual symmetry in pre-norm architectures

To resolve the problem of scale shift in residual networks demonstrated by eq. (4), we try a slightly more general ansatz:

$$\hat{R}_0(x) = x, \tag{5}$$

$$\hat{R}_l(x) = a_l f_l(\hat{R}_{l-1}(x)) + b_l \hat{R}_{l-1}(x), \tag{6}$$

$$\hat{R}_{L+1}(x) = f_{L+1}(\hat{R}_L(x)) \tag{7}$$

with coefficients a_l, b_l. We want to choose these coefficients so that the outputs of $\hat{R}_l$ are unit-scaled if the outputs $f_l, \hat{R}_{l-1}$ are. A similar calculation as in eq. (4) leads to the sufficient condition

$$a_l^2 + b_l^2 = 1,$$

which can be easily satisfied. Having restored Unit Scale, we are faced with another issue. It seems that Equations (5) to (7) describes a different network than Equations (1) to (3), whereas ideally the relation from input to final output should be unchanged when converting the network to Unit Scaling.

Note that the coefficients a_l, b_l are not uniquely defined yet, so our mathematical intuition tells us that we should find an additional constraint to get a unique solution. To find this constraint, let us consider our original residual network in Equations (1) to (3) and analyze how the variance propagates through the network if we assume all operations f_l satisfy Unit Scaling and $\sigma(x) = 1$. Let σ_{l-1}^2 denote the variance of R_{l-1}. Then a simple inductive calculation shows that

$$\sigma_{l-1}^2 = \sum_{i=0}^{l-1} r_i^2.$$

By Equation (2) the output of R_l adds a quantity of variance r_l^2 from the residual connection and a quantity of variance σ_{l-1}^2 from the skip connection. Intuitively, the ratio of these variances should be more important for the overall network dynamics than the absolute scales. Thus our constraint becomes preserving the ratio of variances from the original model, through our choice of a_l, b_l:

$$\frac{a_l^2}{b_l^2} = \frac{\sigma(r_l f_l)^2}{\sigma_{l-1}^2} = \frac{r_l^2}{\sum_{i=0}^{l-1} r_i^2} =: \tau_l,$$

which (up to sign) uniquely defines our multipliers a_l, b_l as

$$a_l = \sqrt{\frac{\tau_l}{\tau_l + 1}}, \quad b_l = \sqrt{\frac{1}{\tau_l + 1}} \tag{8}$$

In summary, we propose the modified residual network

$$\hat{R}_0(x) = x, \tag{9}$$

$$\hat{R}_l(x) = \sqrt{\frac{\tau_l}{\tau_l + 1}} f_l(\hat{R}_{l-1}(x)) + \sqrt{\frac{1}{\tau_l + 1}} \hat{R}_{l-1}(x), \tag{10}$$

$$\hat{R}_{L+1}(x) = f_{L+1}(\hat{R}_L(x)), \tag{11}$$

$$\tau_l = \frac{r_l^2}{\sum_{i=0}^{l-1} r_i^2}. \tag{12}$$

Our main result of this section is that this transformation is indeed a mathematical equivalence under a simple additional structural assumption:

Lemma C.1 *Consider R_l, $\hat{R}_l$ defined as in Equations (2) and (10). Then $\hat{R}_l = R_l / \sqrt{\sum_{i=0}^{l} r_i^2}$ for all $l = 0, .., L$.*

Remarkably, this result does not assume the individual network operations f_l to actually satisfy Unit Scaling. It is purely a consequence of the pre-norm residual structure. However, only under Unit Scaling can the factors τ_l be interpreted as the ratio of variances between skip and residual branch.

As a consequence of the lemma, the final residual output $R_L(x)$ is the same as in our original network up to a fixed multiplier. Due to the zero-homogeneity of the final output function f_{L+1} this gives $\hat{R}_{L+1} = f_{L+1}\left(R_L(x) / \sqrt{\sum_{i=0}^{l} r_i^2}\right) = f_{L+1}(R_L(x)) = R_{L+1}$, proving the mathematical equivalence of our residual scheme.

Modern LLM architectures like Llama [23] are pre-norm residual networks of this kind. Hence they admit a faithful unit-scaled reparametrization.

C.3. Unit Scaling for transformer residuals

The above scheme describes Unit Scaling for arbitrary pre-norm residual networks. We now apply it to the case of (pre-norm) transformer residual layers.

We can describe a transformer in terms of the residual network given in Equations (1) to (3). Our f_l functions alternate between self-attention layers and feed-forward layers. Implementations differ in the handling of how residual multipliers r_l correspond to HPs. In many cases practitioners simply ignore these r_l, but for the sake of expressivity we assume the two types of residual layer each have their own HP, as well as the embedding. In other words,

$$r_l = \begin{cases} \alpha_{\text{emb}} & l = 0 \\ \alpha_{\text{attn-residual}} & l \text{ is odd} \\ \alpha_{\text{ffn-residual}} & l \text{ is even, and } l > 0. \end{cases}$$

To convert this to a Unit Scaled network we apply Equations (9) to (12), from which can derive the following closed-form expression for τ_l:

$$\tau_l = \begin{cases} \dfrac{\alpha_{\text{attn-residual}}}{\alpha_{\text{emb}} + \ell\alpha_{\text{attn-residual}} + \ell\alpha_{\text{ffn-residual}}} & l \text{ is odd} \\ \dfrac{\alpha_{\text{ffn-residual}}}{\alpha_{\text{emb}} + (\ell+1)\alpha_{\text{attn-residual}} + \ell\alpha_{\text{ffn-residual}}} & l \text{ is even.} \end{cases}$$

where $\ell = \lfloor \frac{l-1}{2} \rfloor$.

C.4. Proof of Lemma C.1

Proof This is proved by induction. For the base-case $l = 1$, we have $\tau_1 = r_1^2/r_0^2$, giving

$$\begin{aligned} \hat{R}_1(x) &= \sqrt{\frac{\tau_l}{\tau_l + 1}} f_1(x) + \sqrt{\frac{1}{\tau_l + 1}} x \\ &= (r_1 f_1(x) + r_0 x)/\sqrt{r_0^2 + r_1^2} \\ &= R_1/\sqrt{r_0^2 + r_1^2}. \end{aligned}$$

Then if the statement holds for $l-1$ we have

$$
\begin{aligned}
\hat{R}_l(x) &= \sqrt{\frac{\tau_l}{\tau_l+1}} f_l(\hat{R}_{l-1}(x)) + \sqrt{\frac{1}{\tau_l+1}} \hat{R}_{l-1}(x) \\
&= \frac{r_l}{\sqrt{\sum_{i=0}^{l} r_i^2}} f_l(\hat{R}_{l-1}(x)) + \frac{\sqrt{\sum_{i=0}^{l-1} r_i^2}}{\sqrt{\sum_{i=0}^{l} r_i^2}} \hat{R}_{l-1}(x) \\
&= \left(r_l f_l(\hat{R}_{l-1}(x)) + \sqrt{\sum_{i=0}^{l-1} r_i^2}\, \hat{R}_{l-1}(x) \right) / \sqrt{\sum_{i=0}^{l} r_i^2} \\
&= \left(r_l f_l(R_{l-1}(x)) + \sqrt{\sum_{i=0}^{l-1} r_i^2}\, \frac{R_{l-1}(x)}{\sqrt{\sum_{i=0}^{l-1} r_i^2}} \right) / \sqrt{\sum_{i=0}^{l} r_i^2} \\
&= \left(r_l f_l(R_{l-1}(x)) + R_{l-1}(x) \right) / \sqrt{\sum_{i=0}^{l} r_i^2} \\
&= R_l(x) / \sqrt{\sum_{i=0}^{l} r_i^2}
\end{aligned}
$$

■

Appendix D. Hyperparameters for u-μP

Here we explain the HPs from Section 3.2 in more detail. Our desiderata for the u-μP HPs are as follows:

1. **Unit scale:** For every choice of HPs we satisfy Unit Scaling, meaning that the variance at initialization throughout the entire model is 1.
2. **Interpretable HPs:** Each HP value determines a dynamic of the model at initialization that we consider important, giving them a clear interpretation.
3. **Fully expressive:** They result in a model which is as expressive as a standard pre-norm transformer network, meaning that for any model expressed by Equations (1) to (3), there is a choice of HPs that forms a mathematically-equivalent u-μP residual model.
4. **No redundancy:** Removing any HP results in a strictly less expressive model.

D.1. Residual branch multipliers α_{residual}, $\alpha_{\text{residual-attn-ratio}}$

In this section we present our u-μP residual branch multipliers. They can be viewed as a reparametrization of the original residual multipliers in C.3. We begin by explaining our heuristic for our new set of residual HPs and then combine this with the residual branch re-scaling derived in the previous section, which gives our u-μP residual scheme.

D.1.1. IMPROVED HPS FOR TRANSFORMER RESIDUALS

In Section 3.2 we refer to a new pair of u-µP HPs, α_{residual} and $\alpha_{\text{residual-attn-ratio}}$, which we use for residual layers. Here we define them and make the case for including them as part of our u-µP scheme. To avoid cluttered notation, in this section we rename

$$\alpha_{\text{residual}} = \alpha_r, \quad \alpha_{\text{residual-attn-ratio}} = \alpha_\rho.$$

There is no clear convention for the set of multipliers in a standard residual model. Hence we adopt the most generous and straightforward group of multipliers, $(\alpha_{\text{emb}}, \alpha_{\text{attn-residual}}, \alpha_{\text{ffn-residual}})$, as in Section C.3. For simplicity we rename

$$\alpha_{\text{emb}} = \alpha_e, \quad \alpha_{\text{attn-residual}} = \alpha_a \quad \alpha_{\text{ffn-residual}} = \alpha_f.$$

To make the presentation more clear, we derive our new HPs using the standard residual scheme from Equations (1) to (3). For the actual unit scaled implementation one needs to transform the multipliers following Equations (9) to (12), which we do in Section D.1.2.

Our new multipliers satisfy the following properties that $(\alpha_e, \alpha_a, \alpha_f)$ do not:

1. They have an intuitive interpretation for the multiplier values in the context of the residual output $R_L(x)$, such that each controls a dynamic in the model that we consider important.
2. The number of multipliers is minimized, under the constraint that expressivity is maintained.
3. The most effective choice of one multiplier depends as little as possible on the choice of the other multiplier(s).

To facilitate our analysis, we can view the transformer residual output as the sum of three terms:

$$\begin{aligned}
R_L &= R_L^{(e)} + R_L^{(a)} + R_L^{(f)}, \\
R_L^{(e)} &:= \alpha_e x, \\
R_L^{(a)} &:= \sum_{l=1}^{L/2} \frac{\alpha_a}{\sqrt{L}} f_{2l-1}(R_{2l-1}(x)), \\
R_L^{(f)} &:= \sum_{l=1}^{L/2} \frac{\alpha_f}{\sqrt{L}} f_{2l}(R_{2l}(x)), \\
R_L^{(r)} &:= R_L^{(a)} + R_L^{(f)},
\end{aligned}$$

Note that we have added in the depth-µP multipliers here, though a similar analysis can be performed for non-depth-µP models. As above, f_l functions alternate between self-attention layers and feed-forward layers.

With respect to point 1., we propose two new multipliers that correspond to dynamics in the network which we suggest are important to control at initialization. The first is the ratio of the scale of the residuals' contributions to those of the embedding, $\alpha_r = \sigma(R_L^{(r)})/\sigma(R_L^{(e)})$. The second is the ratio of the scale of the attention-residuals' contributions to those of the feed-forward-residuals, $\alpha_\rho = \sigma(R_L^{(a)})/\sigma(R_L^{(f)})$. We now demonstrate how the existing $(\alpha_e, \alpha_a, \alpha_f)$ multipliers can be replaced by (α_r, α_ρ).

Let us first examine these two quantities under the standard set of HPs. Residual functions of the same kind have the same expected output scale at initialization in pre-norm networks. Hence we denote the output scale $\sigma(f_l(R_l))$ of self-attention functions as σ_a, and of feed-forward functions as σ_f. We thus have the following scales at the output:

$$\begin{aligned}
\sigma(R_L^{(e)}) &= \alpha_e \sigma(x), \\
\sigma(R_L^{(a)}) &= \frac{\alpha_a}{\sqrt{L}} \sigma\left(\sum_{i=1}^{L/2} f_{2l-1}(R_{2l-1})\right) = \frac{\alpha_a \sigma_a}{\sqrt{2}}, \\
\sigma(R_L^{(f)}) &= \frac{\alpha_f}{\sqrt{L}} \sigma\left(\sum_{i=1}^{L/2} f_{2l}(R_{2l})\right) = \frac{\alpha_f \sigma_f}{\sqrt{2}}, \\
\sigma(R_L^{(r)}) &= \sqrt{\sigma(R_L^{(a)})^2 + \sigma(R_L^{(f)})^2} = \frac{\sqrt{(\alpha_a \sigma_a)^2 + (\alpha_f \sigma_f)^2}}{\sqrt{2}},
\end{aligned}$$

meaning our new multipliers are equal to:

$$\begin{aligned}
\alpha_\rho &= \frac{\alpha_a}{\alpha_f} \frac{\sigma_a}{\sigma_f}, \\
\alpha_r &= \frac{\sqrt{(\alpha_a \sigma_a)^2 + (\alpha_f \sigma_f)^2}}{\sqrt{2}\, \alpha_e \sigma(x)}, \\
&= \sqrt{\frac{\alpha_\rho^2 + 1}{2}} \frac{\sigma_f}{\sigma(x)} \frac{\alpha_f}{\alpha_e}.
\end{aligned}$$

The original α_a, α_f multipliers can then be written in terms of α_r, α_ρ:

$$\begin{aligned}
\alpha_a &= \alpha_\rho \alpha_f \frac{\sigma_f}{\sigma(\sigma_a)} \\
\alpha_f &= \alpha_r \alpha_e \frac{\sigma(x)}{\sigma_f} \sqrt{\frac{2}{\alpha_\rho^2 + 1}}
\end{aligned}$$

We have replaced two of the three original multipliers, but still have a dependence on α_e here in our expressions for α_f and $R_L^{(e)}$, which we now remove by dividing it out of our residual branches and embedding. We use the hat $(\hat{\cdot})$ symbol to denote terms that have been divided-through by α_e. This new system of equations is equivalent to our old one thanks to the zero-homogeneity of the final post-residual layer:

$$\begin{aligned}
R_{L+1}(x) &= f_{L+1}(R_L^{(e)} + R_L^{(r)}) \\
&= f_{L+1}((R_L^{(e)} + R_L^{(r)})/\alpha_e) \\
&= f_{L+1}(\hat{R}_L^{(e)} + \hat{R}_L^{(r)})
\end{aligned}$$

This gives $\hat{R}_L^{(e)} = \alpha_e x / \alpha_e = x$, removing our first occurrence of α_e. Following the division through $\hat{R}_L^{(r)}$ results in:

$$
\begin{aligned}
\hat{R}_L^{(r)} &= \hat{R}_L^{(a)} + \hat{R}_L^{(f)}, \\
\hat{R}_L^{(a)} &:= \sum_{l=1}^{L/2} \frac{\hat{\alpha}_a}{\sqrt{L}} f_{2l-1}(R_{2l-1}), \\
\hat{R}_L^{(f)} &:= \sum_{l=1}^{L/2} \frac{\hat{\alpha}_f}{\sqrt{L}} f_{2l}(R_{2l}), \\
\hat{\alpha}_a &= \alpha_\rho \hat{\alpha}_f \frac{\sigma_f}{\sigma_a}, \\
\hat{\alpha}_f &= \alpha_r \frac{\sigma(x)}{\sigma_f} \sqrt{\frac{2}{\alpha_\rho^2 + 1}}.
\end{aligned}
$$

This system of equations is the same as the original, but with the two α_e terms dropped, meaning our model's multipliers can be expressed in terms of only α_r and α_ρ. Using the above equations, any pair of values for (α_r, α_ρ) can be translated back into an equivalent set of values for $(\alpha_e, \alpha_a, \alpha_f)$ such that the output $R_{L+1}(x)$ is the same, meaning that our multipliers are no less expressive than the original set. This satisfies our desired property of minimizing the number of multipliers while maintaining expressivity.

We can simplify further in the case of unit-scaled models, which are designed such that $\sigma(x), \sigma_a, \sigma_f$ are all 1 at initialization. In this case our re-parametrization becomes:

$$\hat{\alpha}_a = \alpha_\rho \hat{\alpha}_f, \tag{13}$$

$$\hat{\alpha}_f = \alpha_r \sqrt{\frac{2}{\alpha_\rho^2 + 1}}, \tag{14}$$

$$\hat{\alpha}_e = 1. \tag{15}$$

This is the basis of our claim that Unit Scaling enables this more intuitive set of multipliers. Not only do the multipliers α_r and α_ρ represent important dynamics in the network at initialization (the ratio of residual-to-embedding scales, and the ratio of attention-to-feed-forward scales), but it's only via unit scaling that these equations become simple enough to implement in practice. Using equations Equations (13) to (15) for a non-unit scaled network may still be effective, but the interpretation we've given to α_r and α_ρ no longer hold.

Our final desired property is an empirical one: that the most effective choice of one multiplier depends as little as possible on the choice of the other multiplier(s). We demonstrate that our multipliers satisfy this property better than the standard set of residual multipliers (see Fig 10 and Fig 11).

D.1.2. THE FULL U-μP RESIDUAL SCHEME

Here we give the full definition of our u-μP residual scheme, summarizing the results of previous sections. A general pre-norm transformer is implemented as:

$$R_0(x) = c\,x, \tag{16}$$
$$R_l(x) = a_l f_l(R_{l-1}(x)) + b_l R_{l-1}(x), \quad l = 1, .., L \tag{17}$$
$$R_{L+1}(x) = f_{L+1}(R_L(x)), \tag{18}$$

where a_l, b_l and c are scalar multipliers, and the f_l alternate between self-attention and feed-forward layers. We consider our baseline set of residual HPs here to be $(\alpha_{\text{emb}}, \alpha_{\text{attn-residual}}, \alpha_{\text{ffn-residual}})$, which we implement (assuming depth-μP branch scaling) as:

$$a_l = \begin{cases} \frac{\alpha_{\text{attn-residual}}}{\sqrt{L}} & l \text{ is odd (self-attention)} \\ \frac{\alpha_{\text{ffn-residual}}}{\sqrt{L}} & l \text{ is even (feed-forward)} \end{cases}$$
$$b_l = 1$$
$$c = \alpha_{\text{emb}}.$$

The corresponding u-μP set of residual HPs is $(\alpha_{\text{residual}}, \alpha_{\text{residual-attn-ratio}})$, which we implement as:

$$a_l = \sqrt{\frac{\tau_l}{\tau_l + 1}} \tag{19}$$
$$b_l = \sqrt{\frac{1}{\tau_l + 1}} \tag{20}$$
$$c = 1, \tag{21}$$
$$\tau_l = \frac{1}{\sqrt{L}} \cdot \begin{cases} \frac{\hat{\alpha}_a}{1 + \ell\hat{\alpha}_a + \ell\hat{\alpha}_f} & l \text{ is odd} \\ \frac{\hat{\alpha}_f}{1 + (\ell+1)\hat{\alpha}_a + \ell\hat{\alpha}_f} & l \text{ is even} \end{cases}, \quad \ell = \left\lfloor \frac{l-1}{2} \right\rfloor \tag{22}$$
$$\hat{\alpha}_a = \hat{\alpha}_f\, \alpha_{\text{residual-attn-ratio}} \tag{23}$$
$$\hat{\alpha}_f = \sqrt{\frac{2}{\alpha^2_{\text{residual-attn-ratio}} + 1}}\, \alpha_{\text{residual}}. \tag{24}$$

This is the u-μP residual scheme. It satisfies the three properties that we initially set out to achieve: the variance at initialization of our $R_l(x)$ is always 1, our HPs have a clear and useful interpretation, and our scheme is as expressive as the baseline (which is neither unit-scaled or has interpretable HPs).

D.2. Multipliers for non-homogeneous ops $\alpha_{\text{attn-softmax}},\ \alpha_{\text{ffn-act}},\ \alpha_{\text{loss-softmax}}$

In this section we derive the rest of our u-μP multipliers. We want to identify the minimal set of multipliers that can still express all different choices of pre-op scales in the model. The crucial observation is that every pre-scale multiplier α of an operation $h \mapsto f(\alpha h)$ can be propagated through the network if f is k-homogeneous for some $k > 0$, i.e. $f(\alpha x) = \alpha^k f(x)$. We can iterate this along the computational path until either the next operation is non-homogeneous, we are at the end of a residual path, or the next operation is 0-homogeneous (e.g. a norm). In the first case

the accumulated scales are absorbed in the pre-op scale of the non-homogeneous operation (where we introduce a multiplier), in the second case they are absorbed in the residual addition for that branch (where we again introduce a multiplier), and in the final case the scale disappears (so we start over). We now go through the Llama forward computation and follow this paradigm to identify our multipliers in Table 5.

Table 5: A walkthrough of the Llama architecture, showing how our $\alpha_{\text{attn-softmax}}$, $\alpha_{\text{ffn-act}}$ and $\alpha_{\text{loss-softmax}}$ multipliers are derived via an analysis of scale-propagation.

Op	**Scale propagation behavior**
Embedding	We already saw in the previous section that the embedding multiplier can be absorbed in the residual multipliers.
Attention RMSNorm	This operation is 0-homogeneous and thus we start over.
Query and key	Query and key itself are linear, hence their weight multipliers get propagated.
Query-Key matmul	The query-key matrix multiplication is 2-homogeneous when viewed as function of the concatenated query-key vector. Hence it propagates the scale.
Softmax	The softmax operation is non-homogeneous. Thus the pre-op scale of the softmax becomes our first multiplier $\alpha_{\text{attn-softmax}}$.
Value	The value layer is linear and hence propagates its scale.
Softmax-value matmul	This operation is linear in all arguments and hence propagates the scale.
Attention projection	This operation is linear and lies at the end of the attention residual path. Hence there are no more multipliers in the attention block.
FFN RMSNorm	This operation is 0-homogeneous and thus we start over.
FFN input scale	The input layer is linear, hence its weight multiplier gets propagated.
Sigmoid input	This function is non-homogeneous and thus we have another multiplier $\alpha_{\text{ffn-act}}$.
SiLU weight	This layer is also linear and propagates the scale.
Product	The entry-wise multiplication of the outputs of sigmoid, input layer and SiLU weight is homogeneous and thus propagates the scale.
FFN output	This layer is linear and at the end of the residual path. Hence there are no more multipliers in the FFN residual block.
Output RMSNorm	This operation is 0-homogeneous and thus we start over.
Output head	This layer is linear, hence its weight multiplier gets propagated.
Loss	The cross-entropy loss is non-homogeneous and leads to our final multiplier $\alpha_{\text{loss-softmax}}$.

In summary, we have three multipliers $\alpha_{\text{attn-softmax}}$, $\alpha_{\text{ffn-act}}$, $\alpha_{\text{loss-softmax}}$ that are applied in the softmax, sigmoid and loss function via:

$$f_{\text{softmax}}(q, k) = \text{softmax}(\alpha_{\text{attn-softmax}} \cdot d_{\text{head}}^{-1} \cdot (q \cdot k^t)), \tag{25}$$

$$f_{\text{act}}(h) = \text{sigmoid}(\alpha_{\text{ffn-act}} \cdot h), \tag{26}$$

$$f_{\text{loss-softmax}}(h, x_{\text{targets}}) = \text{CE}(\alpha_{\text{loss-softmax}} \cdot h, x_{\text{targets}}) \tag{27}$$

We do not explicitly show the derivation of the residual multipliers here, as they undergo a change in accordance with D.1.1 before we get our final α_{residual} and $\alpha_{\text{residual-attn-ratio}}$.

Our analysis from above shows that these three multipliers together with the residual multipliers are as expressive as the full set of pre-ops multipliers in the whole transformer architecture while having no redundancy, i.e. a change in one of the multipliers cannot equivalently be expressed in terms of changes to the other multipliers.

Appendix E. Details on FP8 training

E.1. Background

While full precision (FP32) floating point arithmetic has been the default in the machine learning community for many years, the ever-growing scale of models and datasets has led to a push towards lower-precision formats. Training involving FP16 formats has seen major uptake in large-scale Transformer training, but requires extra care to guarantee stable numerics. Certain quantities, such as optimizer states, are routinely kept in full precision and loss scaling techniques [11, 16] are employed to keep gradients in the representable range of the FP16 format. More recently, BFLOAT16 training has mitigated these scaling issues for 16-bit formats. However, no such alternative is available for narrower formats, such as FP8.

Recently released AI accelerators have introduced native hardware support for FP8 arithmetic [7, 18]. However, the further reduced precision and range of these formats introduces additional numerical challenges which have not been addressed conclusively by the literature.

While FP16 arithmetic can be used throughout the complete forward-backward pass through the model, efforts on FP8 training have focused on performing the computationally most expensive operations, matrix multiplications, in FP8. This means that input tensors are cast to an FP8 format prior to a matrix multiplication, the result of which is produced in a higher-precision format again.

Micikevicius et al. [17] propose two different FP8 formats for deep learning, assigning different numbers of bits to the exponent and the mantissa, respectively. The `E4M3` format uses 4 bits for the exponent and 5 bits for the mantissa, whereas the `E5M2` format uses 5 bits for the exponent, prioritizing range over relative precision. (The maximum representable value of `E5M2` is 57344 compared to 448 for `E4M3`.) The authors recommend to cast activations and weights to `E4M3` and activation gradients (computed during the backward pass) to `E5M2`.

Finally, existing attempts at FP8 training employ per-tensor scaling techniques [17]. A scalar factor is extracted prior to casting a tensor to FP8 and multiplied back onto the output. In pseudo-code that reads:

```
a = scale(A)
b = scale(B)
A = to_fp8(A / a)
```

```
B = to_fp8(B / b)
C = (a * b) * matmul(A, B)
```

where we assume that `matmul` takes inputs in FP8 and directly produces the output in higher precision. An obvious choice for the scaling factor is to rescale the maximum absolute value of a tensor to the maximum value representable by the FP8 format [17]. However, this requires computing "absmax" prior to performing the matrix multiplication, which hinders an efficient implementation. To circumvent that, a so-called delayed scaling technique has been proposed [19], which tracks the scale of each tensor over time, allowing us to implement the rescaling and matrix multiplication in an efficient fused kernel.

E.2. Details on FP8 scheme

We demonstrate that u-µP enables a simple scheme for FP8 training, without the need for cumbersome per-tensor scaling. Each linear module in a model induces three matrix-matrix products: one during the forward pass to compute the output and two during the backward pass, computing gradients w.r.t. the weights and the inputs, respectively. The tensors participating in these matrix-matrix products are the input activations, the weights and the gradients w.r.t. the output activations. Unit scaling ensures unit scale for all three tensors at initialization.

Empirically, the scales of these tensors do not drift too much during training with the exception of the input tensors to the FFN and self-attention final projections, which grow considerably (see Appendix F.3, in particular Figure 8). This is consistent across different HP settings (see Figure 9).

Based on this observation, we test a simple scheme for FP8 training. For every matrix multiplication, we cast the input, weight and grad-output tensors to `E4M3`, with the exception of the inputs to FFN and self-attention final projections, which are cast to `E5M2` to accommodate their growing scale. The output of each matrix multiplication is produced directly in the higher-precision format (FP32 in our case). No loss scaling or per-tensor scaling is applied.

Note that we conducted our experiments with simulated FP8 numerics, quantizing inputs to a matrix multiplication as if they were cast to FP8, while allowing the multiplication to be computed on hardware that does not have native FP8 support.

Appendix F. Experimental Details

Our experiments are all in the setting of autoregressive language model training, a domain that has proven a useful testing ground for general machine learning techniques in the model-capacity constrained regime (the under-fitting regime described by Belkin et al. [2]). Our evaluation metric is final training cross-entropy loss. This has the benefits of low variance and of separating the concerns of downstream training and regularization, which are not in scope for this work. This follows the precedent of Yang et al. [30] who also report training loss. As our model training is not in the over-fitting regime, we expect training loss to track validation (and have seen so empirically). Default training settings are given in Table 6.

To compare µP and u-µP with the Llama architecture on a larger dataset, we modify the implementation provided by Yang et al. [30] for µP and implement u-µP in the same framework.

Dataset	WikiText-103 [15]
Sequence Length	256
Vocab Size	32000
Training Set Tokens	138 M
Architecture	Llama [23] (Transformer, PreNorm, RMSNorm, SwiGLU, RoPE, "untied" embeddings), non-trainable RMSNorm parameters.
Width	256
Depth	4
Head Dimension	64
Batch size	64
Training steps	8192 (0.97 epochs)
LR schedule	Cosine to 10%, 2000 steps warm-up
Optimizer	AdamW $(\beta_1, \beta_2, \epsilon) = (0.9, 0.999, 10^{-8})$
Weight Decay	2^{-13}, independent [14]
Dropout	0.0
μP HP Search Range	$\eta \in [2^{-10}, 2^{-6}]$ $\hat{\eta}_{\text{emb}} \in [2^0, 2^8]$ $\sigma_{\text{init}}, \alpha_{\text{emb}}, \alpha_{\text{attn}}, \alpha_{\text{output}} \in [2^{-2}, 2^2]$
u-μP HP Search Range	$\eta \in [2^{-1}, 2^3]$ $\alpha_{\text{attn}} \in [2^{-2}, 2^2]$ $\alpha_{\text{residual}}, \alpha_{\text{residual-attn-ratio}}, \alpha_{\text{ffn-act}}, \alpha_{\text{output}} \in [2^{-3}, 2^3]$
μP HP Defaults	$\sigma_{\text{init}} = \alpha_{\text{emb}} = \alpha_{\text{attn}} = \alpha_{\text{output}} = \hat{\eta}_{\text{emb}} = 1$
u-μP HP Defaults	$\alpha_{\text{residual}} = \alpha_{\text{residual-attn-ratio}} = \alpha_{\text{ffn-act}} = \alpha_{\text{output}} = \alpha_{\text{attn}} = 1$

Table 6: Default hyperparameters and training settings.

F.1. Hyperparameter Independence

Our analysis indicates that μP's hyperparameters have overlapping effects on dynamics-defining scales within the model, while u-μP attempts to isolate their effect. We hypothesize that this effect should be visible in the final loss—the effects of u-μP's hyperparameters should be more separable than that of μP's.

In our first test of this hypothesis, we construct pairs of hyperparameters, and perform a coarse 2D sweep for each pair (Figure 10, Figure 11). These results show some visual dependence between μP hyperparameters as a diagonal structure in the grids, such as $(\hat{\eta}_{\text{emb}}, \sigma_{\text{init}})$ and $(\eta, \alpha_{\text{attn}})$. We quantify this difference by evaluating the increase in loss on a given row by using the argmin hyperparameter from a different row of the grid, compared with the actual minimum and averaged over all grids. This metric gives an average loss increase of 0.08 for μP versus 0.03 for u-μP. This suggests a quantifiable improvement in hyperparameter separability, but note that the metric may conflate this with the flatness of the optimum.

The second test is more directly practical. We compare two hyperparameter search methods on μP and u-μP. The first is a random grid search, which samples configurations without replacement from a grid defined over all hyperparameters. After performing a single search, we can simulate the effect of a shorter search by taking a random sample of the results. The second method is an independent search, which consists of the following phases:

1. Perform a 1D line search for an optimal learning rate, with other hyperparameters set to their default (9 runs).
2. For each hyperparameter in parallel, perform a 1D line search (330 runs).
3. Combine the best settings from step 2, and re-evaluate (6 runs).

Each 1D line search can be done on an iteratively refined grid, to provide an incremental improvement as the number of runs increases.

Our results from this test in Figure 2 (right) show that the first LR sweep is much more efficient for u-μP since the default hyperparameters are better. For this reason, the 1D line search can outperform a random grid. We also observe that the final step of combining optimum hyperparameters is very harmful to μP, while it shows only a slight degradation for u-μP, which was expected as a regression to the mean.

F.2. Hyperparameter Transfer

We compare learning rate transfer for μP and u-μP in Figures 1 and 6, over a logarithmic grid of spacing $2^{1/2}$, with 3 runs for each point. We observe:

1. u-μP transfer of LR over width, training steps, batch size and depth is similar to or better than μP, when starting from default parameters.
2. u-μP and μP both show increased variance when the learning rate is too high (visible in wide confidence intervals in Figure 6).
3. The default settings for u-μP are better than those of μP, especially when scaling width and training duration (steps or batch size).

F.3. Numerical Properties

Our analysis of the numerical properties of u-μP focuses on the RMS statistics of tensors that we wish to cast to FP8: linear module input activations, weights and output gradients. RMS captures the larger of the mean and scale of a distribution, and as such can be a good test of whether the tensor is likely to suffer range (clipping) errors in low-precision number formats.

Figure 3 shows the distribution of statistics over all linear modules in the model, and Figure 8 shows RMS on a per-tensor basis, as it evolves during training. From these, we note:

1. μP has gradients and weights with low RMS, at risk of FP8 underflow, whereas u-μP starts with $\mathrm{RMS} \approx 1$.
2. Many input activations do not grow RMS during training (due to a preceding non-trainable RMSNorm), however the attention out projection and FFN down projection have unconstrained input activations that grow considerably during training.

3. The decoder weight grows during training. Since it is preceded by a RMSNorm, the model may require scale growth in order to increase the scale of softmax inputs. Other weights grow slightly during training.

4. Gradients grow quickly but stabilize, except for attention out projection and FFN down projection, whose gradients shrink as the inputs grow.

We also evaluate how RMS growth is affected by model and training hyperparameters in the tensors that showed the highest end-training RMS, shown in Figure 9. This shows that the main parameter affecting scale growth is learning rate, with end-training RMS increasing to the right of the optimal LR basin, as training becomes unstable. End-training RMS is remarkably stable as width, depth, training steps and batch size are independently increased.

We therefore propose the FP8 scheme described in Appendix E, which works for u-μP without any dynamic scaling or exponent bias search (see Figure 1 (left), Figure 7).

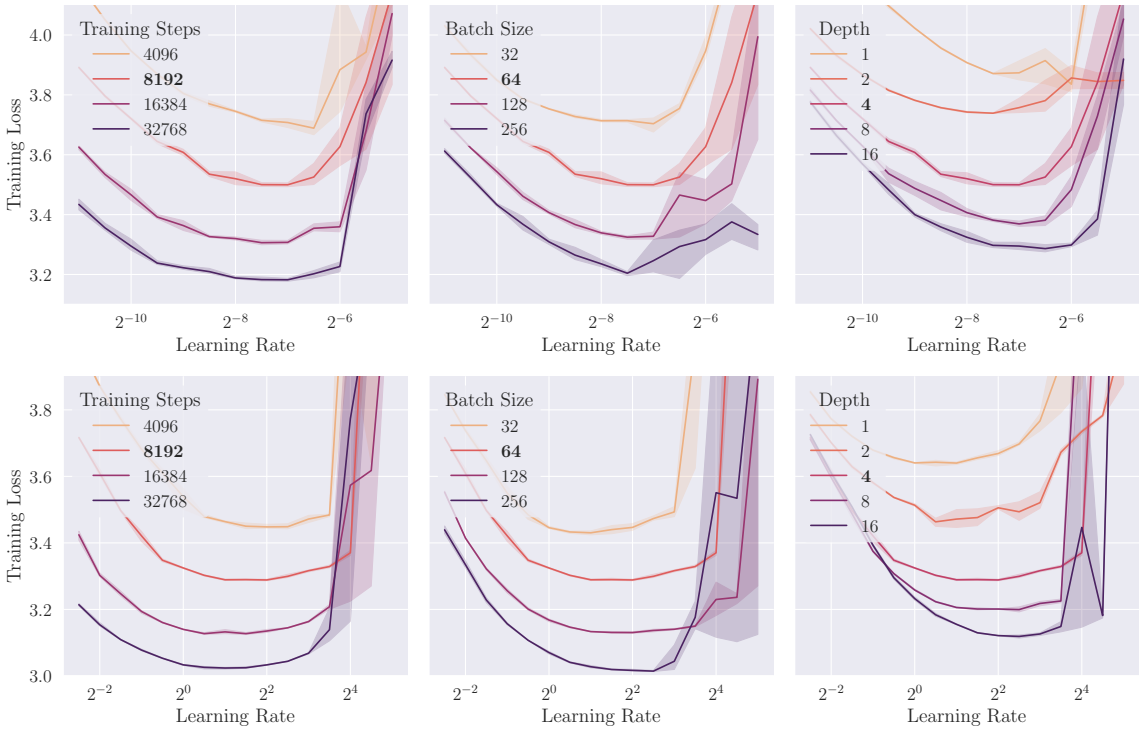

Figure 6: Learning rate transfer for μP (top) and u-μP (bottom), over training steps, batch size and depth. See Figure 1 (left) for transfer over width. The **default** shape parameter for other panels is shown in bold. The shaded area shows the 95% confidence interval for the mean.

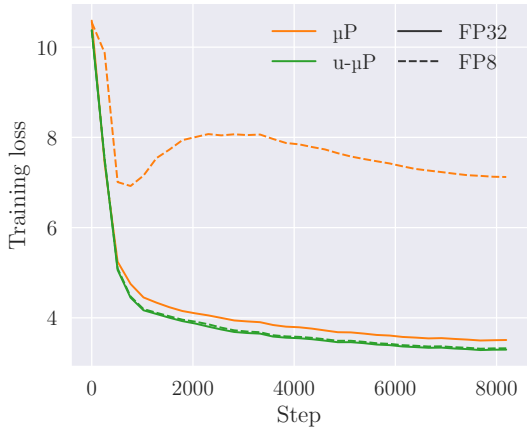

Figure 7: FP8 training by direct cast, width 256, default hyperparameters, $\eta = (2^1, 2^{-8})$ for (u-μP, μP).

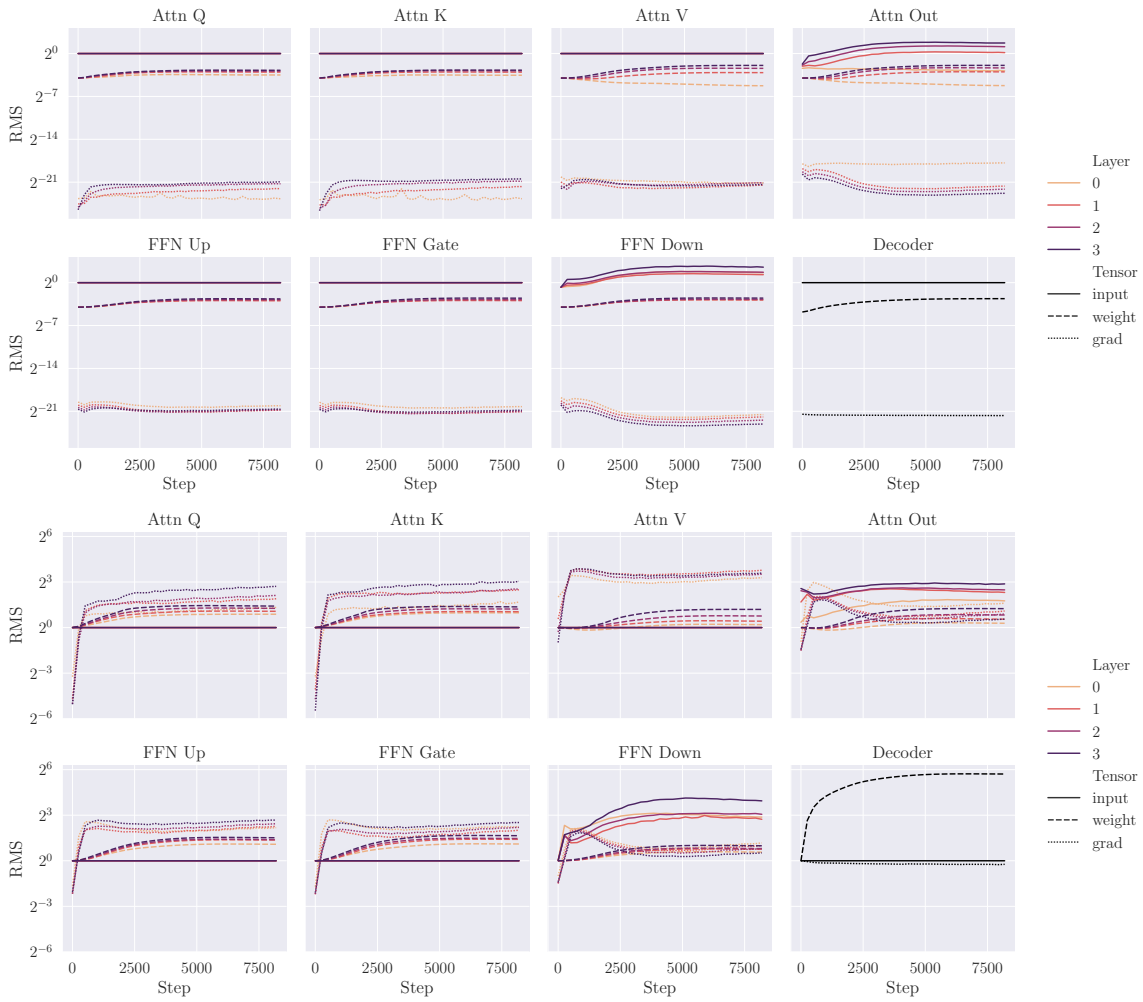

Figure 8: RMS during training, for all parametrized matmul inputs, for μP (top) and u-μP (bottom). Model width 256, default hyperparameters, $\eta = (2^1, 2^{-8})$ for (u-μP, μP).

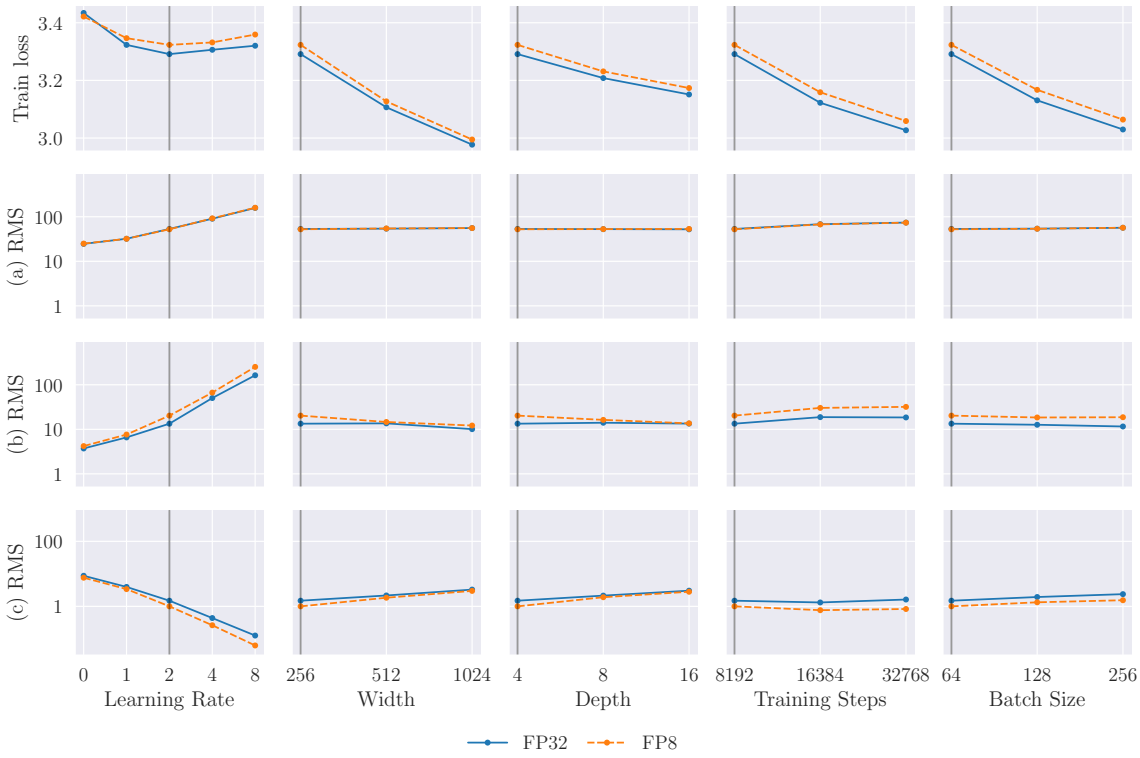

Figure 9: The effect of hyperparameters on FP8 training loss and on the end-training RMS of various tensors: (a) decoder weight, (b) last-layer FFN down-projection input and (c) last-layer FFN down-projection output gradient. Only learning rate has a substantial effect on the end-training RMS. Vertical lines show the default setting of that hyperparameter, as used for all other plots.

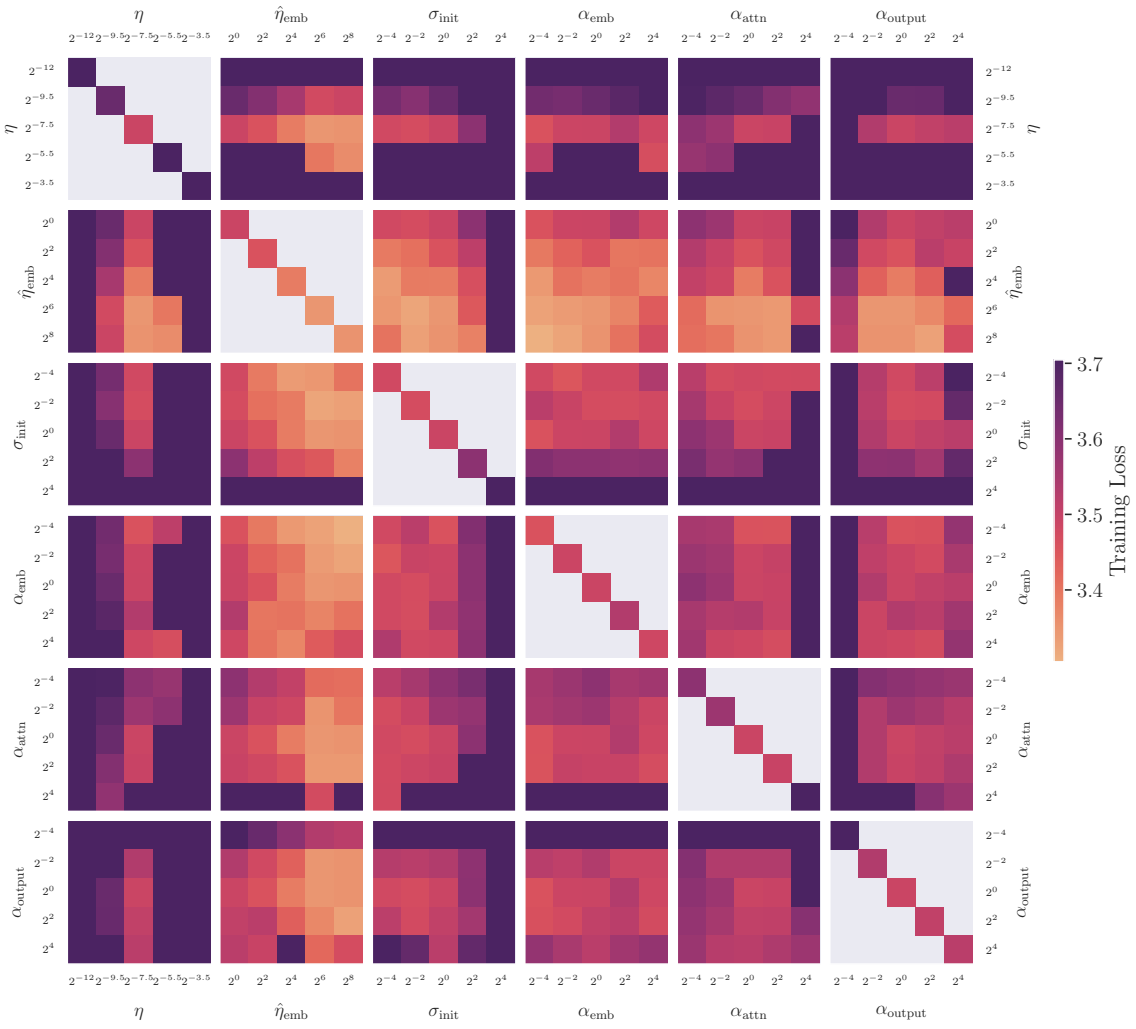

Figure 10: Hyperparameter coupling sweep for μP. Note strong coupling between optima, e.g. in the cases of $(\eta_{\text{emb}}, \sigma_{\text{init}})$ and $(\eta, \alpha_{\text{attn}})$. See also: u-μP, Figure 11. Across all grids, the average training loss degradation from using the optimum from the wrong row/column is 0.08, which is worse than u-μP (0.03).

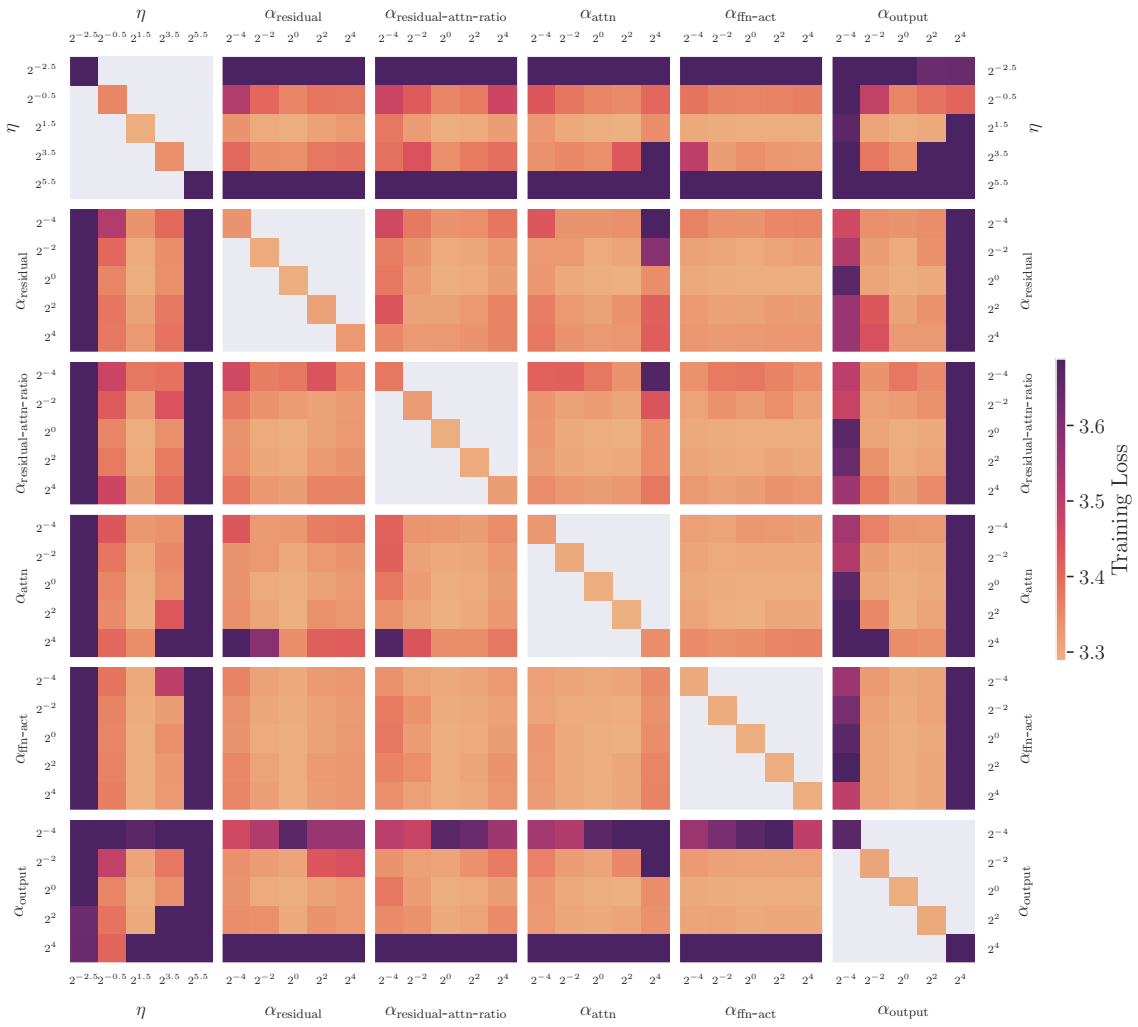

Figure 11: Hyperparameter coupling sweep for u-μP. Note less coupling than with μP, see Figure 10. Across all grids, the average training loss degradation from using the optimum from the wrong row/column is 0.03, which is better than μP (0.08).