OPT2024: 16th Annual Workshop on Optimization for Machine Learning

u-μP: The Unit-Scaled Maximal Update Parametrization

Charlie Blake, Josef Dean, Luke Y. Prince, Carlo Luschi, Douglas Orr
Graphcore CHARLIEB@GRAPHCORE.AI

Constantin Eichenberg, Lukas Balles, Björn Deiseroth, Samuel Weinbach
Aleph Alpha CONSTANTIN.EICHENBERG@ALEPH-ALPHA-IP.AI

Andres Felipe Cruz-Salinas
Cohere (work done while at Aleph Alpha)

Abstract

The Maximal Update Parametrization (μP) aims to make the optimal hyperparameters (HPs) of a model independent of its size, allowing them to be swept using a cheap proxy model rather than the full-size target model. 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 activations, weights and gradients begin training with a scale of one. 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

The algorithmic challenges of training large language models (LLMs) can be framed in terms of stability. We consider this in three forms: feature learning stability, which ensures that parts of the model do not learn too fast or slow relative to each other, hyperparameter stability, which ensures that the optimal HPs for small models remain unchanged as the model size grows, and numerical stability, which ensures that floating-point representations during training stay within the range of a given number format.

The Maximal Update Parametrization (μP) [31, 34] targets the first two sources of instability. μP defines a set of scaling rules that in principle make a model's optimal HP values consistent across model sizes and ensure 'maximal feature learning' in the infinite-width limit. The practical benefits of this are that models continue to improve as they get larger, and that practitioners can re-use a set of HP values (especially the learning rate) found for a small *proxy* version of their model, on a larger *target* model.

However, in practice μP does not necessarily provide the kind of simple, stable scaling which a user might expect. To address this, we propose the Unit-Scaled Maximal Update Parametrization (u-μP). u-μP combines μP with another closely-related training innovation, Unit Scaling [2]. μP ideally provides consistent training dynamics across model sizes, but says little about what those dynamics should be. Unit Scaling addresses this by proposing an ideal principle for dynamics: unit variance for all activations, weights and gradients. Unit Scaling was initially designed to ensure stable numerics, but in the context of μP the principle of unit-scale brings many additional benefits.

These include a more efficient approach to HP search, improved HP transfer, and the facilitation of a simpler HP scheme, all of which we demonstrate here, alongside the ability to perform *out-of-the-box* FP8 training.

2. Background

2.1. The Maximal Update Parametrization

Tensor Programs V [34] defines a parametrization as 'a rule for how to change [HPs] when the widths of a neural network change'. They show that µP is the only parametrization that gives 'maximal feature learning' in the limit, whereas standard parametrization (SP) has imbalanced learning.

µP can be defined in terms of the *abc-parametrization*. This describes a model in terms of three scaling factors per-weight-tensor: A_W, B_W, C_W, where $W_t = A_W \cdot w_t, w_0 \sim \mathcal{N}(0, B_W^2), w_{t+1} = w_t + C_W \cdot \Phi_t$ (Φ_t is the weight update at time t). Each factor is a function of the model-width. The particular scaling factors used for µP are given in Table 4 (see Appendix D.1). A key property of these parametrizations is that they are subject to *abc-symmetry*, which states that a model's dynamics (under Adam) are invariant to changes of the form $A_W \times= \theta, B_W \div= \theta, C_W \div= \theta$ for a fixed θ.

A consequence of improved stability is that learning dynamics under µP are ideally independent of model-size, as are optimal HPs. This facilitates *µTransfer*, the process of training smaller proxy models to evaluate candidate HPs, taking the best-performing ones to train a larger target model.

2.2. Unit Scaling

Unit Scaling [2] is a method for designing models that can be trained in low-precision via a simple cast operation. Unit scaled models insert *static* scaling factors after each op in both forward and backward passes, such that each weight, activation and gradient tensor has unit variance at initialization.

This is a useful criterion as it places values around the center of floating-point formats' absolute range. This applies to all tensors, meaning every operation in the network requires a scaling factor that ensures unit-scaled outputs, assuming unit-scaled inputs. We provide an example of deriving the Unit Scaling rule for a matmul op in Appendix D.3, resulting in the scaling factor: $1/\sqrt{d_{\text{fan-in}}}$. We accompany this example with a full recipe for applying Unit Scaling to an arbitrary model.

Table 1: Scaling rules for µP versus u-µP, *including* associated HPs (assuming the *extended* set in Table 6). These rules constitute the definition of u-µP, along with the unit-scaled ops in Appendix B.

	ABC-multiplier		Weight Type: Input	Weight Type: Hidden	Weight Type: Output	Residual
µP	parameter	(A_W)	α_{emb}	1 (or α_{attn})	$\alpha_{\text{out}} \frac{\text{base-fan-in}}{\text{fan-in}}$	$\sqrt{\frac{\text{base-depth}}{\text{depth}}}$
	initialization	(B_W)	σ_{init}	$\sigma_{\text{init}} \sqrt{\frac{\text{base-fan-in}}{\text{fan-in}}}$	σ_{init}	—
	Adam LR	(C_W)	$\eta\, \hat{\eta}_{\text{emb}}$	$\eta \frac{\text{base-fan-in}}{\text{fan-in}}$	η	$\sqrt{\frac{\text{base-depth}}{\text{depth}}}$
u-µP	parameter*	(A_W)	1	$\frac{1}{\sqrt{\text{fan-in}}}$	$\frac{1}{\text{fan-in}}$	$\frac{1}{\sqrt{\text{depth}}}$
	initialization	(B_W)	1	1	1	—
	Adam LR	(C_W)	$\eta \frac{1}{\sqrt{\text{fan-out}}}$	$\eta \frac{1}{\sqrt{\text{fan-in}}}$	η	$\frac{1}{\sqrt{\text{depth}}}$

*u-µP's α HPs are associated with operations, not weights, so are not included here (see Appendix E.1).

In contrast, other low-precision methods require dynamic calculation of scaling factors during training, adding complexity (see Appendix D.2). As unit-scaled models begin with all tensors having an 'ideal' starting scale, over time this appears to be sufficient to keep them within numerical range, despite the scale-changes resulting from training.

3. The Unit-Scaled Maximal Update Parametrization

Whereas Unit Scaling provides rules for scaling all operations, µP only does so for parametrized ones. It's these operations we need to address to arrive at a unified scheme, resolving differences in the scaling rules each recommends. We begin with the expressions for the A_W, B_W, C_W scaling factors in Equation (11), and substitute in the µP rules defined in Table 4. This results in a complete implementation of µP, shown in the top half of Table 1. We set out to turn this into a valid Unit Scaling scheme, which requires unit initializations $B_W = 1$, and matmuls scaled by $A_W = 1/\sqrt{\text{fan-in}}$.

Our first step is to drop the σ_W and base-fan-in HPs entirely, and associate the α_W HPs with certain functions instead of weights (as outlined in Appendix B). This new HP scheme is designed to satisfy four criteria: minimal cardinality, maximal expressivity, minimal interdependency and interpretability.

With these changes applied, hidden weights now have: $A_W, B_W, C_W = (1, 1/\sqrt{\text{fan-in}}, \eta/\text{fan-in})$ which differs from our Unit Scaling criteria. However, using the abc-symmetry we can shift scales by a factor of $\sqrt{\text{fan-in}}$, arriving at a unit-scaled scheme: $(1/\sqrt{\text{fan-in}}, 1, \eta/\sqrt{\text{fan-in}})$.

Our output layer also has unit initialization, but $A_W = 1/\text{fan-in}$. This differs from the Unit Scaling rule, but in the forward pass this is permissible as there are no subsequent matmuls of a transformer. In the backward pass this mis-scaling would propagate, so we apply the desired $1/\sqrt{\text{fan-in}}$ factor. Using different forward and backward scales in this way is usually not allowed, but is valid for output layers due to the cut-edge rule (see [2], Section 5.1).

The final change we make is to the input LR scaling rule. In Figure 1 we show empirically that µP's $c_W = 1$ rule is mis-specified for embeddings, with the optimal value scaling with $1/\sqrt{\text{fan-out}}$

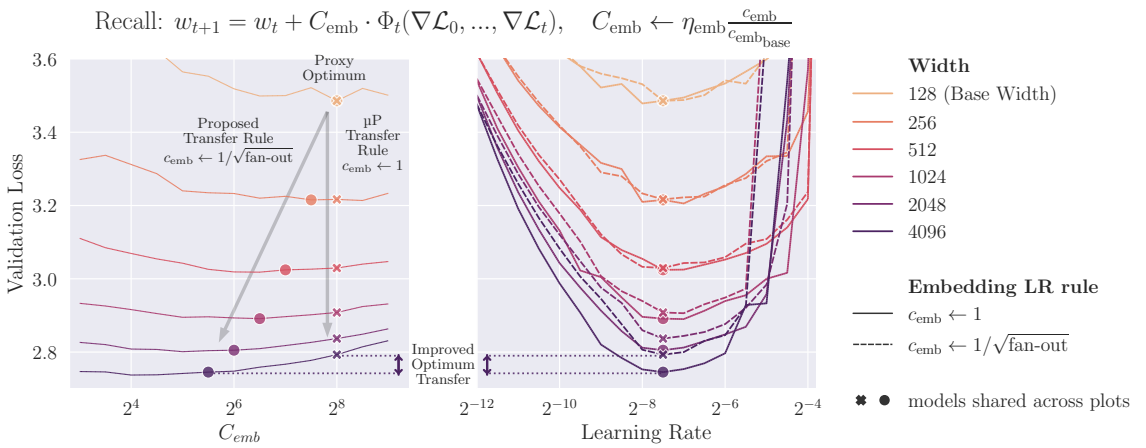

Figure 1: (Left) holding the embedding LR multiplier (C_{emb}) constant, vs. scaling with $\sqrt{1/\text{width}}$, both with a fixed global LR. This suggests the µP embedding LR rule (c_{emb}) should follow the latter scaling. (Right) we test this by sweeping the global LR under the two scaling rules. The new rule leads to lower loss on large models. (Dot/cross markers represent the same runs across both graphs).

instead, which we adopt for u-μP. With these changes made, we arrive at our final u-μP scheme, given in Table 1. It's important to note that the scaling rules in this table must be combined with the standard Unit Scaling rules for other non-matmul operations (covered in Appendix B).

4. Experiments

Our experiments use the Llama [24] architecture trained on WikiText-103 [15] (full settings are given in Table 2). In accordance with our analysis of requirements for effective μTransfer in Appendix C, we remove parameters from norm layers and use independent AdamW.

4.1. Hyperparameter search

Our new HP scheme, designed for improved separability, enables HPs to be swept more efficiently. This is shown in Figure 4 (a). Random search is the standard approach in the literature, which we compare to our proposed *independent search* (outlined in Appendix A.3) which sweeps HPs independently and combines them.

Independent search begins with a simple LR sweep. This alone is sufficient for u-μP to reach near-optimal loss (using only 9 runs). During this phase other HPs are fixed at 1, which for u-μP means that the inputs to operations are generally unit-scaled. Consequently, we conclude that unit scale at initialization is close to ideal scaling for effective learning here. In contrast μP still requires non-LR HPs to be swept to attain a reasonable loss. The 'combined mults' phase causes the loss to spike for μP. This is due to the HP dependencies shown in Figure 5, which mean HPs cannot be swept independently and used together, necessitating random search which can require hundreds of runs.

4.2. Hyperparameter transfer

We demonstrate that our scheme provides HP transfer, with Figure 4(b) showing LR transfer across width, Figure 2 showing it across training steps, batch size and depth, and Figure 8 for other HPs. We find that the optimal LR is constant for all widths under u-μP. The optimal LR is also approximately constant for steps, batch size and depth. This means we can scale our proxy model down across all axes and maintain LR transfer. Of these, width appears most stable and depth least.

Whereas μP sees diminishing returns for larger widths in Figure 4(b), u-μP continues to benefit from width, with the 2048 u-μP model matching the 4096 μP model. We attribute this to our improved embedding LR rule. The optimal values found for non-LR HPs are all close to 1. In practice this means that dropping these HPs entirely is potentially viable for similar models and training setups.

4.3. Numerical properties

Figure 3 shows the RMS over all linear modules, with more detailed analysis in Figures 9 and 10. RMS captures the larger of the mean and scale of a distribution, and as such can be a good test of whether a tensor is likely to suffer over/underflow in low-precision number formats. Detailed analysis of these statistics is given in Appendix A.5, with our results supporting the central thesis of Unit Scaling: that tensors are well-scaled at initialization and largely remain so across training.

Based on this we propose our FP8 scheme: we cast the input, weight and grad-output tensors for each matmul to FP8 `E4M3`, except the inputs to FFN and self-attention final projections, which are cast to FP8 `E5M2` to accommodate their growing scale. This simply requires FP8 casts to be inserted into the model, avoiding more complex scaling methods that have been proposed for FP8.

4.4. FP8 training

We now show that u-µP can indeed be trained in our FP8 scheme. Figure 4 (c) demonstrates the application of our FP8 scheme to training at width 4096. We use the HPs suggested by the sweep in Figure 4 (a), but transferred to the larger model-width. µP fails entirely under our FP8 scheme due to gradient underflow, indicating the need for a more complex scaling scheme under regular µP, whereas using just this simple cast u-µP trains in FP8 with only a small increase in validation loss.

5. Related Work

Techniques introduced to facilitate FP8 training include those covered in Appendix D.2 and more [14, 21, 25]. These largely concern the quantizing of activations, weights and gradients, though [20] also explore FP8 optimizer states and comms. [27] show that unstable training dynamics can result from attention logit growth (fixed by QK-norm [4]) and from divergent output logits (fixed by z-loss [3]). [1] investigate how pre-training settings affect instabilities during post-training quantization.

6. Conclusions

We present an improved version of µP, underpinned by the principle of Unit Scaling. This provides the simple low-precision training that comes with unit-scale, but also provides a platform for solving other problems with µP. These include more efficient HP search under u-µP, which can even drop non-LR HPs and still reach near-optimal loss. Our improved embedding LR scaling facilitates better performance at large widths, and we see strong HP transfer across width, depth, batch size and steps. Overall, u-µP simplifies and strengthens the practical application of µP, and provides further evidence that the principle of Unit Scaling is beneficial for model design.

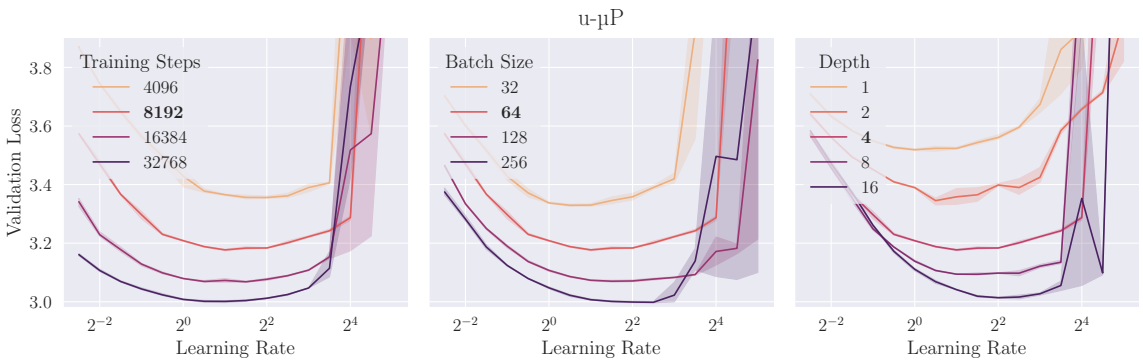

Figure 2: LR transfer for u-µP over training steps, batch size and depth. The **default** shape parameter for other panels is shown in bold. The shaded area shows the 95% confidence interval for the mean.

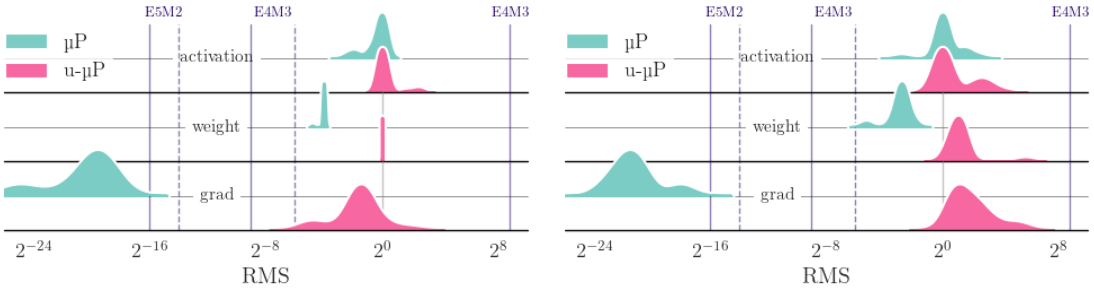

Figure 3: Per-tensor RMS $= \sqrt{\sigma^2 + \mu^2}$ across u-μP and μP models at initialization (left) and after training (right). u-μP tensors have RMS that starts close to 1 and remains within E4M3 range at the end of training. Dashed and solid red lines show each format's min. normal and subnormal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

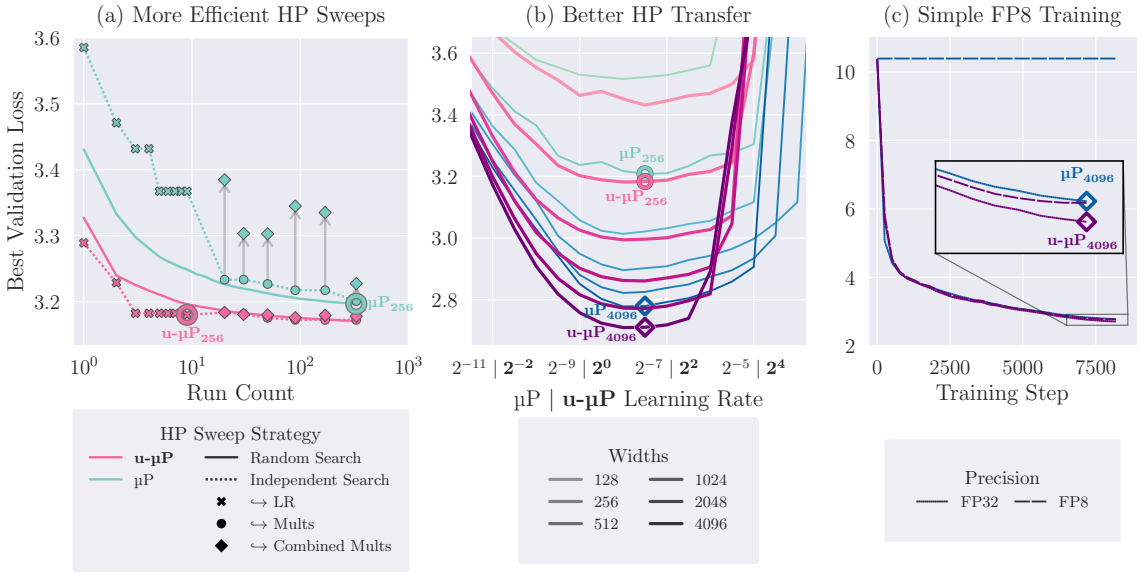

Figure 4: **(a)** Two different HP sweeping processes used for μP and u-μP proxy models. Unlike μP, u-μP admits independent (1D) search due to careful HP design. The first part of independent search is an LR sweep, which alone reaches near-optimal loss for u-μP. **(b)** Using the best proxy HPs from (a), we train many models at different widths and LRs. The best LR for width 256 is ~optimal for 4096, showing LR transfer along with lower loss. **(c)** We re-train with a simple un-scaled `.to(float8)` cast on matmul inputs. This would fail for other models, but u-μP trains with minimal degradation.

References

[1] Arash Ahmadian, Saurabh Dash, Hongyu Chen, Bharat Venkitesh, Stephen Zhen Gou, Phil Blunsom, Ahmet Üstün, and Sara Hooker. Intriguing properties of quantization at scale. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/6c0ff499edc529c7d8c9f05c7c0ccb82-Abstract-Conference.html.

[2] 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.

[3] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113, 2023. URL http://jmlr.org/papers/v24/22-1144.html.

[4] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme Ruiz, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd van Steenkiste, Gamaleldin Fathy Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Collier, Alexey A. Gritsenko, Vighnesh Birodkar, Cristina Nader Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Pavetic, Dustin Tran, Thomas Kipf, Mario Lucic, Xiaohua Zhai, Daniel Keysers, Jeremiah J. Harmsen, and Neil Houlsby. Scaling vision transformers to 22 billion parameters. 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 7480–7512. PMLR, 2023. URL https://proceedings.mlr.press/v202/dehghani23a.html.

[5] 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.

[6] Nolan Dey, Shane Bergsma, and Joel Hestness. Sparse maximal update parameterization: A holistic approach to sparse training dynamics. *CoRR*, abs/2405.15743, 2024. URL http://arxiv.org/abs/2405.15743.

[7] Hadi Esmaeilzadeh, Emily R. Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In Ravi R. Iyer, Qing Yang, and Antonio González, editors, *38th International Symposium on Computer Architecture (ISCA 2011), June 4-8, 2011, San Jose, CA, USA*, pages 365–376. ACM, 2011. doi: 10.1145/2000064.2000108. URL https://doi.org/10.1145/2000064.2000108.

[8] 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.

[9] IEEE Computer Society. IEEE standard for floating-point arithmetic. pages 1–84, July 2019. doi: 10.1109/IEEESTD.2019.8766229.

[10] Arthur Jacot, Clément Hongler, and Franck Gabriel. Neural tangent kernel: Convergence and generalization in neural networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 8580–8589, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/5a4be1fa34e62bb8a6ec6b91d2462f5a-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] 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.

[14] Naveen Mellempudi, Sudarshan Srinivasan, Dipankar Das, and Bharat Kaul. Mixed precision training with 8-bit floating point. *CoRR*, abs/1905.12334, 2019. URL http://arxiv.org/abs/1905.12334.

[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] Badreddine Noune, Philip Jones, Daniel Justus, Dominic Masters, and Carlo Luschi. 8-bit numerical formats for deep neural networks. *CoRR*, abs/2206.02915, 2022. doi: 10.48550/ARXIV.2206.02915. URL https://doi.org/10.48550/arXiv.2206.02915.

[19] NVIDIA. Transformer Engine. https://github.com/NVIDIA/TransformerEngine, 2024.

[20] Houwen Peng, Kan Wu, Yixuan Wei, Guoshuai Zhao, Yuxiang Yang, Ze Liu, Yifan Xiong, Ziyue Yang, Bolin Ni, Jingcheng Hu, Ruihang Li, Miaosen Zhang, Chen Li, Jia Ning, Ruizhe Wang, Zheng Zhang, Shuguang Liu, Joe Chau, Han Hu, and Peng Cheng. FP8-LM: training FP8 large language models. *CoRR*, abs/2310.18313, 2023. doi: 10.48550/ARXIV.2310.18313. URL https://doi.org/10.48550/arXiv.2310.18313.

[21] Sergio P. Perez, Yan Zhang, James Briggs, Charlie Blake, Josh Levy-Kramer, Paul Balanca, Carlo Luschi, Stephen Barlow, and Andrew Fitzgibbon. Training and inference of large language models using 8-bit floating point. *CoRR*, abs/2309.17224, 2023. doi: 10.48550/ARXIV.2309.17224. URL https://doi.org/10.48550/arXiv.2309.17224.

[22] Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Srinivasan, Xiaodong Cui, Wei Zhang, and Kailash Gopalakrishnan. Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks. 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 4901–4910, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/65fc9fb4897a89789352e211ca2d398f-Abstract.html.

[23] Thomas N. Theis and H.-S. Philip Wong. The end of Moore's law: A new beginning for information technology. *Comput. Sci. Eng.*, 19(2):41–50, 2017. doi: 10.1109/MCSE.2017.29. URL https://doi.org/10.1109/MCSE.2017.29.

[24] 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.

[25] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7686–7695, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/335d3d1cd7ef05ec77714a215134914c-Abstract.html.

[26] Xi Wang and Laurence Aitchison. How to set adamw's weight decay as you scale model and dataset size. *CoRR*, abs/2405.13698, 2024. doi: 10.48550/ARXIV.2405.13698. URL https://doi.org/10.48550/arXiv.2405.13698.

[27] 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. *CoRR*, abs/2309.14322, 2023. doi: 10.48550/ARXIV.2309.14322. URL https://doi.org/10.48550/arXiv.2309.14322.

[28] 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.

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

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] Greg Yang, James B. Simon, and Jeremy Bernstein. A spectral condition for feature learning. *CoRR*, abs/2310.17813, 2023. doi: 10.48550/ARXIV.2310.17813. URL https://doi.org/10.48550/arXiv.2310.17813.

[36] 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.

Appendix A. Additional experimental details

A.1. Experimental Setup

Our experimental analysis of u-µP was conducted by adapting the codebase used for Tensor Programs V, allowing us to compare µP and u-µP in the same setting. We change various experimental settings from the µP paper to make our experiments better reflect standard training procedures, particularly the dataset which we switch from WikiText-2 to the larger WikiText-103 [15]. Where not specified otherwise, the default setting used in our experiments are given in Table 2. These also represent the settings of our proxy model.

Dataset	WikiText-103 [15]
Sequence length	256
Vocab size	32000
Training set tokens	138M
Architecture	Llama [24] (Transformer, PreNorm, RMSNorm, SwiGLU, RoPE, "untied" embeddings), non-trainable RMSNorm parameters.
Width	256 (scaled up to 4096)
Depth	4
Number of heads	4 (scaled up to 64)
Head dimension	64
Total parameters	$19.5M$ (scaled up to 1.07B)
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 [13]
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 2: Default hyperparameters and training settings.

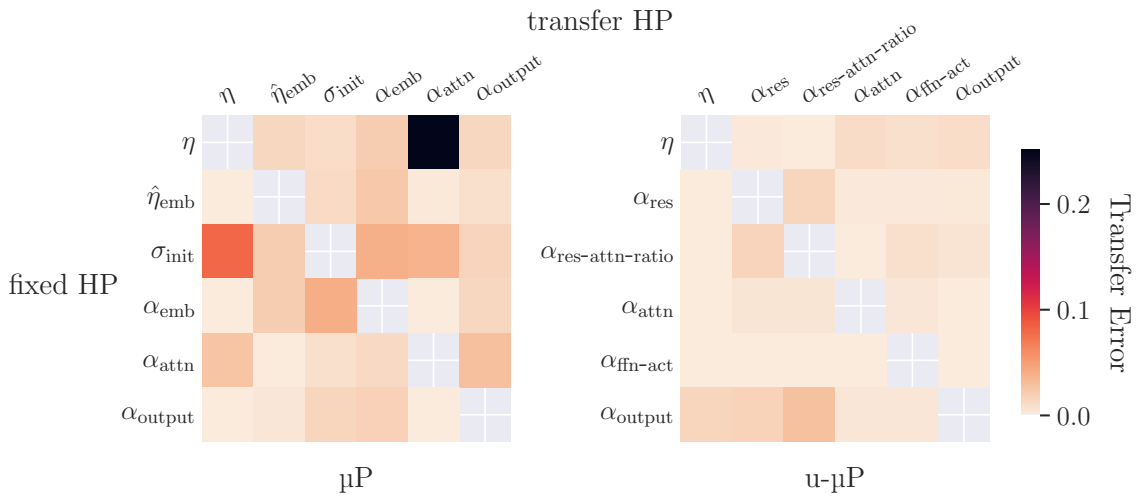

Figure 5: A visualization of the dependencies between pairs of HPs under each scheme. Transfer error measures the extent to which the optimal value of the transfer HP depends on the fixed HP (see Algorithm 1). On average, μP has a transfer error of 0.03, whereas u-μP has 0.005.

A.2. Hyperparameter interdependence

Our principled approach to HPs (see Appendix E.1 for details) contains the requirement that their optimal values should depend minimally on the value of other HPs. Here we investigate this empirically, conducting a 2D sweep over every pair of HPs for μP and u-μP.

To derive an empirical measure of HP dependency, we introduce the notion of *transfer error* (see Algorithm 1). This considers a pair of HPs, with one 'fixed' and the other for 'transfer'. We take the best value of the transfer HP for each non-optimal value of the fixed HP, and use it with the optimal value of the fixed HP. The transfer error is the difference between the losses obtained and the minimum loss. Figure 5 shows this measure for each pair of HPs under μP and u-μP, reflecting the improvement in HP dependency as a result of our scheme. This gives u-μP a reduced risk of small transfer errors leading to large degradations, and the potential to sweep HPs in a more separable way.

Algorithm 1 Transfer Error

Require: A 'fixed' HP with candidate values $F = \{f_1, \cdots, f_n\}$, a 'transfer' HP with candidate values $T = \{t_1, \cdots, t_m\}$, a function that gives the final validation loss for the pair of HPs $L : F \times T \to \mathbb{R}$ (assuming all other HPs are fixed at default values).

```
err ← 0
f*, t* ← argmin(L)
for f in F do
    if f ≠ f* then
        t ← argmin(L(f))
        err += L(f*, t) − L(f*, t*)
    end if
end for
return err/(n − 1)
```

A.3. Hyperparameter search

Here we outline the particular search processes used for our μP and u-μP HP sweeps in Figure 4 (a). The *random search* samples uniformly from a grid defined over all *extended* HPs (extended HP sets are defined in Table 6, with grid values defined in Table 2). We perform the random search over 339 runs, each of which is a full training of the width-256 proxy model. We then simulate the effect of shorter searches at various run-counts by taking a random sample of the results, resulting in the smooth curve over run-count shown.

The *independent search* consists of the following phases:

1. Perform a 1D line search for an optimal learning rate, with other hyperparameters set to their default values (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).

The number of runs in the 1D line search is an order of magnitude higher than is required in practice. We do so to form a fair comparison with the random search, which benefits from this large number of runs. The number of runs for the 1D line search could be reduced further by using binary search, though this would require sequential runs and limit the extent of parallelism.

A.4. Hyperparameter transfer experiments

Baseline μP transfer Figure 6 is a companion plot to Figure 2 in the body of the paper, showing the LR transfer of the baseline μP model over the same axes. u-μP shows marginally more stable HP transfer here relative to the baseline, and at a consistently lower loss.

LR transfer over width The transfer experiments shown in Figure 4 (b) use the non-LR HPs found in Figure 4 (a) (indicated by the circled points), rather than using default HP values. For the u-μP sweep we take the HPs at the end of the LR portion of the independent search, as these are already close-to-optimal, and means only 9 runs were required in the sweep. In contrast, for μP it is necessary to use the results of the random search over a large number of runs.

LR transfer over other axes For the training steps, batch size and depth transfer experiments in Figure 2, all HP values are fixed to 1 except LR which is swept. As with width transfer, u-μP outperforms μP here using these default HP values. Reducing training steps is done by fixing the number of warm-up steps (at 2000) and still cosine-decaying the learning rate to 10%; all that changes is the number of post-warm-up steps. We found this to be more effective than cutting-short the decay schedule. For both Figure 4 (b) and Figure 2 we sweep the LR over a logarithmically-spaced grid of step $2^{1/2}\times$, with 3 runs for each point.

Additionally, in Figure 7 we show learning rate transfer over sequence length for both μP and u-μP fixing either tokens per batch or sequences per batch. In both scenarios u-μP shows not only better absolute training performance, but also better transfer behavior as sequence length increases. Since our default proxy sequence length is 256, using μP to transfer to sequence length 2048 would result in minimal improvements or even a degradation in validation loss, whereas the u-μP shows much greater and more consistent improvements.

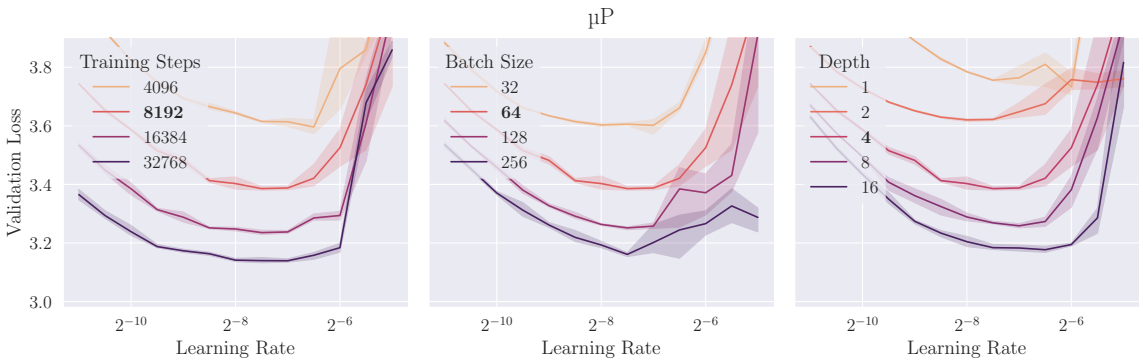

Figure 6: Learning rate transfer for µP over training steps, batch size and depth. For u-µP results, see Figure 2 in the body of the paper.

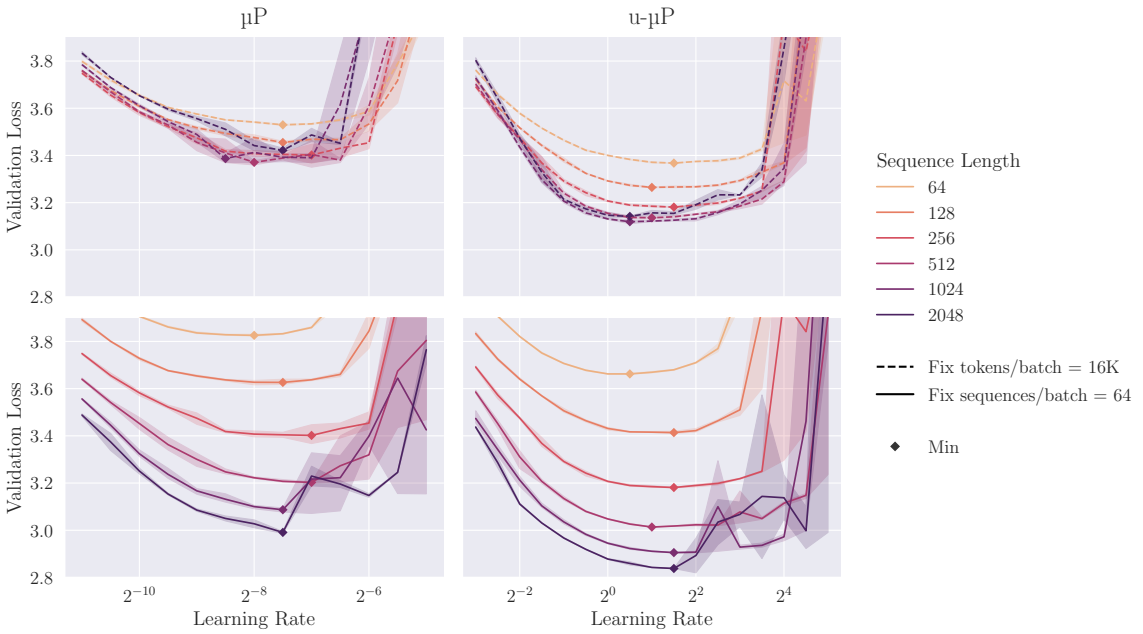

Figure 7: Transfer of learning rate over sequence length for µP (left) and u-µP (right). As sequence length varies, we can fix the number of tokens per batch by inversely varying the number of sequences per batch (top). Alternatively we can fix the sequences per batch and allow the number of tokens per batch to vary with sequence length (bottom). In the latter case, larger sequence lengths mean the model sees more tokens during training, though as per Table 2 this translates to >1 epoch on WikiText-103 when sequence length goes above 256.

Other HP transfer over width For our non-LR HP transfer results in Figure 8, we note that good transfer under µP has not been demonstrated for all HPs used in the literature. This is particularly true for the $\hat{\eta}_{\text{emb}}$ HP, which has poor transfer under µP. Our investigation here led to our identification of the need to adjust the embedding LR scaling rule as shown in Figure 1. In many cases users have not swept this HP, but instead swept the corresponding parameter multiplier α_{emb}. How this HP interacts

with the embedding LR scaling problem identified (and our proposed fix) remains to be explored, though we note in Figure 8 that it also appears to have poor transfer.

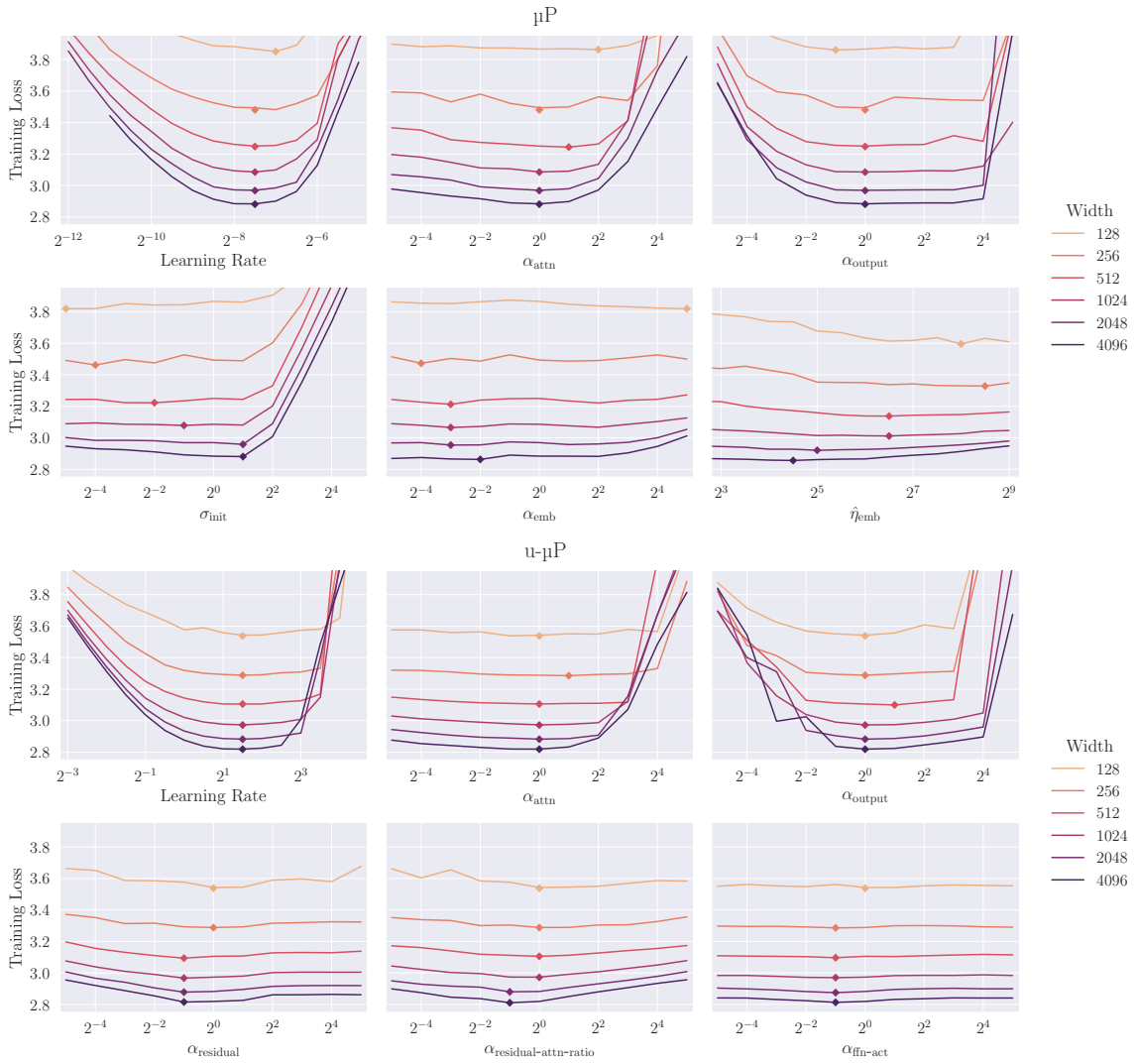

Figure 8: Transfer of model hyperparameters over width for μP (top) and u-μP (bottom). When one hyperparameter is being swept, all others are fixed at 1, with the exception of Learning Rate $\eta = (2^{1.5}, 2^{-7.5})$ for (u-μP, μP).

A.5. Numerical properties

Our analysis of the numerical properties of u-μP focuses on the RMS of tensors that we wish to cast to FP8: linear module input activations, weights and output gradients. From the RMS training statistics plots in Figures 3 and 9 we note that

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 10. 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.

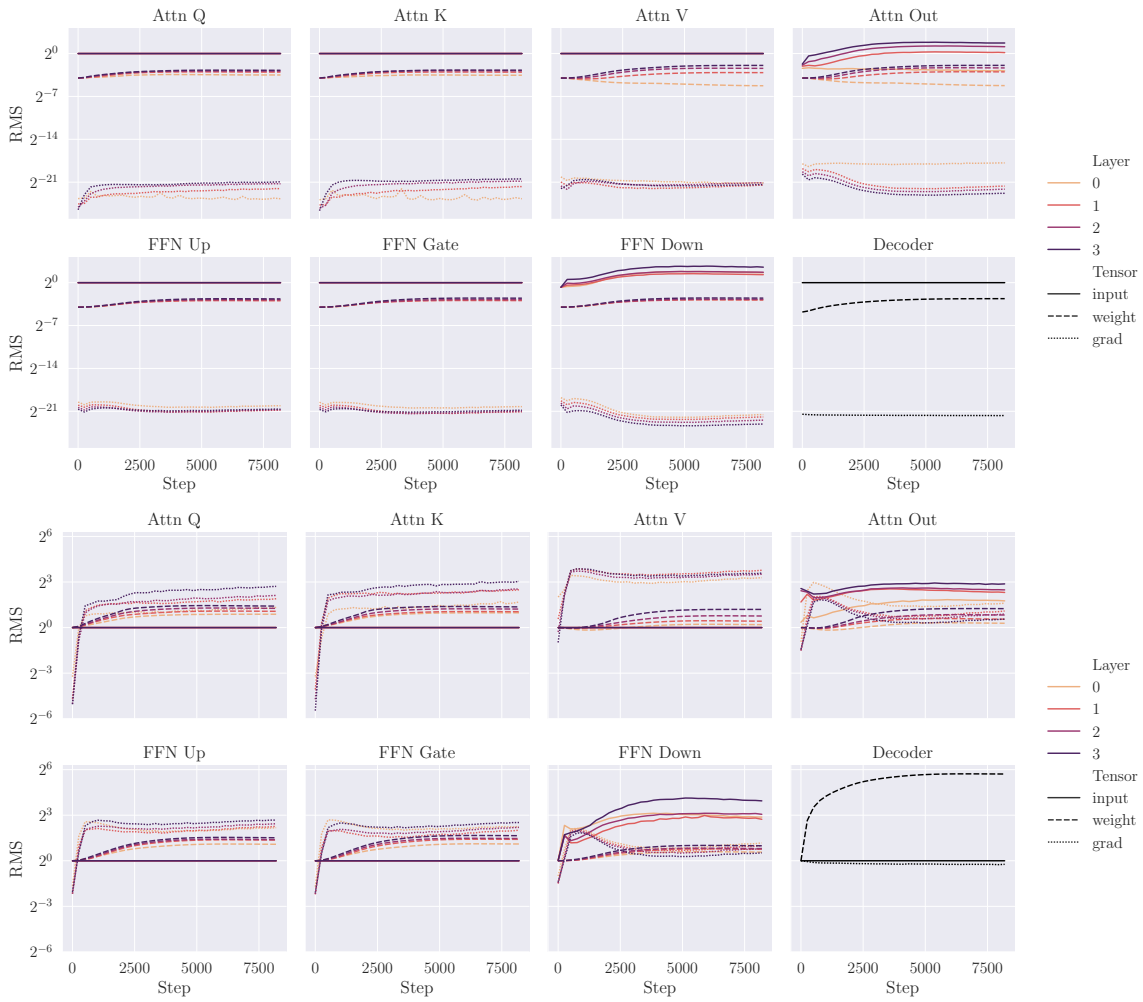

Figure 9: RMS during training, for all parametrized matmul inputs, for μP (top) and u-μP (bottom). Model width 256, default hyperparameters, $\eta = (2^1, 2^{-8})$ for (u-μP, μP).

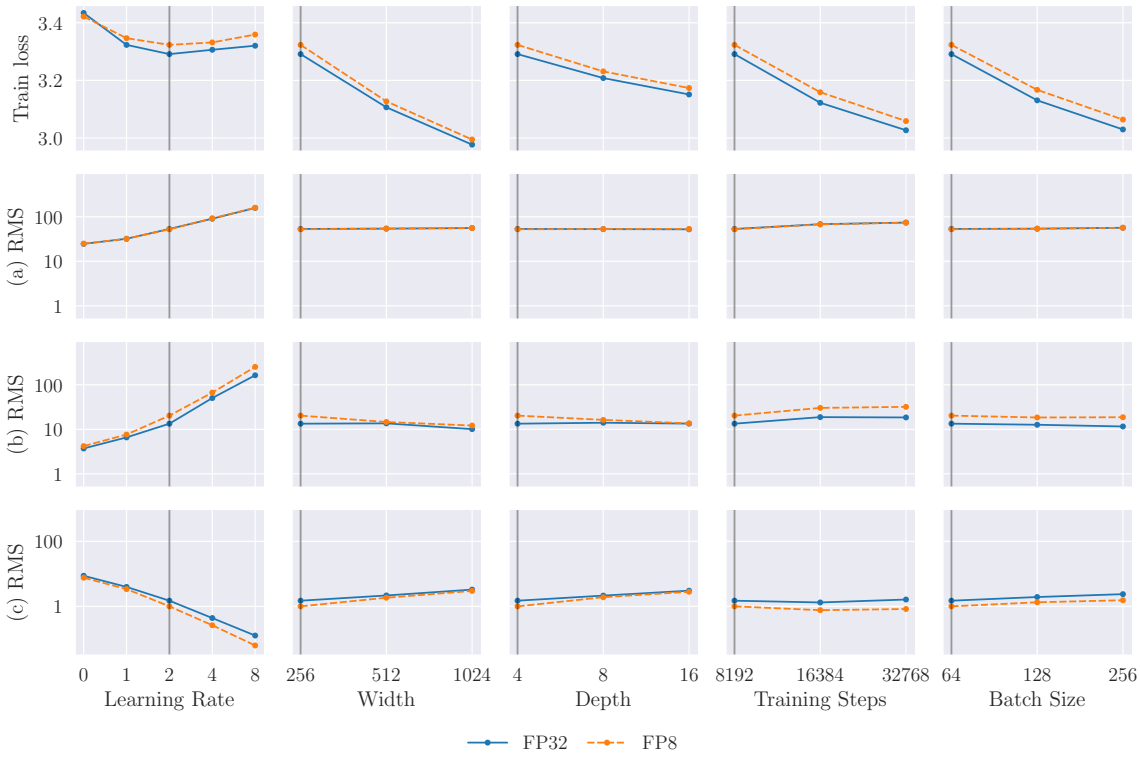

Figure 10: The effect of hyperparameters on FP8 training loss and on the end-training RMS of various tensors: (a) decoder weight, (b) last-layer FFN down-projection input and (c) last-layer FFN down-projection output gradient. Only learning rate has a substantial effect on the end-training RMS. Vertical lines show the default setting of that hyperparameter, as used for all other plots.

Appendix B. Unit-scaled op definitions

Table 3: Implementations of unit-scaled ops, building on Table A.2. from the Unit Scaling paper [2]. These are considered part of u-μP and should be used in the place of standard operations.

Op	Unit Scaling factors
$\text{matmul}(x, w) = xw$	$\alpha = \frac{1}{\sqrt{\text{fan-in}}}, \beta_x = \frac{1}{\sqrt{\text{fan-out}}}, \beta_w = \frac{1}{\sqrt{\text{batch-size}}}$
$\text{attention}(q, k, v) =$ $\quad \text{softmax}\left(\alpha_{\text{attn}}\, d_{\text{head}}^{-1}\, (qk^\top) \odot c_{\text{mask}}\right) v$	$\alpha = \beta_q = \beta_k = \beta_v =$ $\quad 1/\,\text{log\_interpolate}\left(\frac{1}{1+\frac{4d_{\text{head}}}{\alpha_{\text{attn}}^2}}, 1, \sqrt{\frac{\log(s)}{s}}\right)$
$\text{gated\_silu}(x_{\text{in}}, x_{\text{gate}}) =$ $\quad x_{\text{in}} \odot x_{\text{gate}} \odot \text{sigmoid}(\alpha_{\text{ffn-act}}\, x_{\text{gate}})$	$\alpha = \beta_{x_{\text{in}}} = \beta_{x_{\text{gate}}} =$ $\quad 1/\,\text{log\_interpolate}\left(\frac{1}{1+\frac{1}{\alpha_{\text{ffn-act}}^2}}, \frac{1}{\sqrt{2}}, \frac{1}{2}\right)$
$\text{residual\_add}(x_{\text{resid.}}, x_{\text{skip}}) =$ $\quad a\, x_{\text{resid.}} + b\, x_{\text{skip}}$	$a = \frac{\tau}{\sqrt{\tau^2+1}},\ b = \frac{1}{\sqrt{\tau^2+1}}$ (see below for full details, inc. values for τ, which depends on α_{res} and $\alpha_{\text{res-attn-ratio}}$.)
$\text{softmax\_xent}(x, t) =$ $\quad \text{log\_softmax}(\alpha_{\text{loss-softmax}}\, \text{x})_t$	$\alpha = 1,\ \beta = s/\sqrt{s-1}$
$\text{RoPE}(x)$	$\alpha = \beta = 1$ (i.e. no scaling)
$\text{RMSNorm}(x)$ (non-trainable, see [12])	$\alpha = \beta = 1$ (i.e. no scaling)

The Unit Scaling paper provides scaling factors for various ops, in order to make them unit-scaled. However, these ops do not cover every case required for the Llama architecture used in our experiments, nor do they cover our updated residual layer implementation. To address this, in this section we outline a series of new unit-scaled ops for each of our required architectural features, as well as existing unit-scaled ops, given in Table 3.

The presentation here is derived from that of the Unit Scaling Compendium given in [2, Table A.2]. 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 (see Appendix D.3 for an example).

The attention and gated-silu operations are sufficiently complex that we found an empirical model of their scale to be more accurate than any mathematically-derived rule.

For these cases we make use of the function

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

which empirically allows us to model the scaling behavior of these ops closely.

Pre-norm residual layers Our implementation of residual layers for u-μP is more complex than other operations, as adjustments are required to make pre-norm residual networks support Unit Scaling and satisfy our requirements for principled HPs (see Appendix E).

We consider the following definition of a pre-norm transformer:

$$R_0(x) = cx, \tag{1}$$
$$R_l(x) = a_l f_l(R_{l-1}(x)) + b_l R_{l-1}(x), \quad l = 1, .., L \tag{2}$$

where a_l, b_l and c are scalar multipliers, and the f_l alternate between self-attention and feed-forward layers.

A standard (depth-)μP model implements residual layers using: $a_l = \alpha_{\text{residual}}/\sqrt{L/2},\ b_l = 1,\ c = \alpha_{\text{emb}}$. For u-μP we drop $(\alpha_{\text{residual}}, \alpha_{\text{emb}})$ and introduce $(\alpha_{\text{res}}, \alpha_{\text{res-attn-ratio}})$, and implement the network as follows:

$$a_l^2 = \frac{\tau_l^2}{\tau_l^2 + 1} \tag{3}$$
$$b_l^2 = \frac{1}{\tau_l^2 + 1} \tag{4}$$
$$c = 1, \tag{5}$$
$$\tag{6}$$
$$\tau_l^2 = \begin{cases} \dfrac{\hat{\alpha}_a^2}{\frac{L}{2} + \ell\hat{\alpha}_a^2 + \ell\hat{\alpha}_f^2} & l \text{ is odd} \\ \dfrac{\hat{\alpha}_f^2}{\frac{L}{2} + (\ell+1)\hat{\alpha}_a^2 + \ell\hat{\alpha}_f^2} & l \text{ is even} \end{cases}, \quad \ell = \left\lfloor \frac{l-1}{2} \right\rfloor \tag{7}$$
$$\hat{\alpha}_a^2 = \alpha_{\text{res-attn-ratio}}^2 \, \hat{\alpha}_f^2 \tag{8}$$
$$\hat{\alpha}_f^2 = \frac{2}{\alpha_{\text{res-attn-ratio}}^2 + 1} \, \alpha_{\text{res}}^2 \, . \tag{9}$$

This scheme is slightly more complex to implement, but satisfies two key properties. Firstly, the variance at initialization of each $R_l(x)$ is always 1 (i.e. we have Unit Scaling throughout), without having reduced the expressivity of the network (assuming a final normalization layer at the end of the residual-stack).

The second is that, unlike in the original model, our $(\alpha_{\text{res}}, \alpha_{\text{res-attn-ratio}})$ HPs are designed specifically to determine key dynamics in the network at initialization: $\sigma(\alpha_{\text{res}}) = \sigma\left(\sum_{l=1}^{L} R_l\right)/\sigma(R_0)$ (the ratio of the average scale of the residuals' contributions to those of the embedding) and

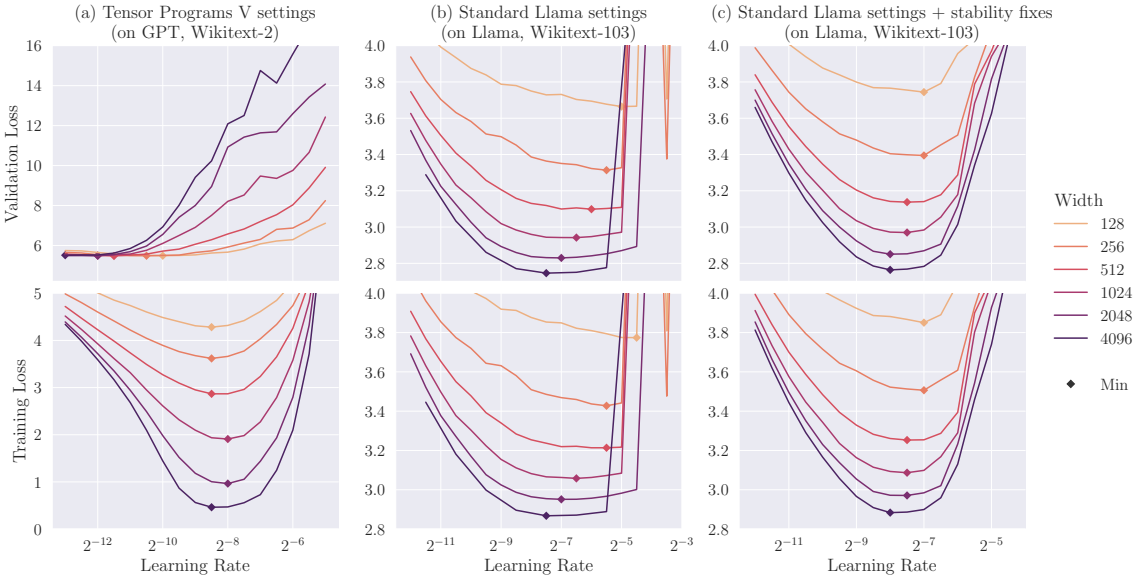

Figure 11: Effective μTransfer does not hold across all training setups. **(a)** We show strong transfer for the unrealistic setup used in Tensor Programs V (too many epochs; constant LR). **(b)** Moving to a more standard Llama training setup, transfer breaks down. **(c)** This is restored by the introduction of two stability fixes: non-parametric norms and independent weight decay.

$\sigma(\alpha_{\text{res-attn-ratio}}) = \sigma\left(\sum_{l=1}^{L/2} R_{2l-1}\right) / \sigma\left(\sum_{l=1}^{L/2} R_{2l}\right)$ (the ratio of the scale of the attention-residuals' contributions to those of the feed-forward-residuals). By defining our residual HPs in this way their effects are more separable, and the optimal values we find have a natural interpretation.

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 [2], 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.

Appendix C. The challenges with μP in practice

Lingle [12] shows that directly applying μP to a decoder LM fails to provide LR transfer across width. Given that the primary use of μP in the literature has been LM training of this kind, this result suggests a significant limitation. How do we reconcile this with the strong LR transfer across width shown for language models in Tensor Programs V?

We answer this Figure 11. The first training setup (a) is aligned with that used in Tensor Programs V (their Figure 4). There are several atypical aspects to their training setup, primarily the use of a constant LR schedule and a high number of epochs. This overfitting regime makes validation loss unusable, and transfer misleadingly good. When we remove these and shift to a standard Llama training setup (b), optimal HPs begin to drift with width. This confirms Lingle's findings that standard μP is in fact a poor fit for modern LM training. We fix this (c) by the removal of parameters from LayerNorms/RMSNorms, as suggested by Lingle, and the introduction of *independent* weight decay

for AdamW, as suggested by Wortsman et al. [27] [1] (see [26] for further analysis). With these changes adopted, we recover the strong transfer shown in Tensor Programs V's experiments.

Appendix D. Additional background material

D.1. The Maximal Update Parametrization

Theoretical background We do not cover the theory underpinning μP in this paper, presenting only its resulting scaling rules (Table 4). For readers interested in this theory, the extensive Tensor Programs series [28–30, 32, 33] builds up a framework from which μP is derived [31]. For those requiring a more accessible introduction, [35] show that μP can be derived in a simpler and more general way by placing a spectral scaling condition on the norm of weights and their updates.

ABC-parametrizations μP, SP, and the Neural Tangent Kernel (NTK) [10] are all instances of abc-parametrizations. This assumes a model under training where weights are defined as:

$$\begin{aligned} w_0 &\sim \mathcal{N}(0, B_W^2), \\ W_t &= A_W \cdot w_t, \\ w_{t+1} &= w_t + C_W \cdot \Phi_t(\nabla \mathcal{L}_0, ..., \nabla \mathcal{L}_t), \end{aligned} \tag{10}$$

with t a time-step and $\Phi_t(\nabla \mathcal{L}_0, ..., \nabla \mathcal{L}_t)$ the weight update based on previous loss gradients.

A parametrization scheme such as μP is then defined specifying how scalars A_W, B_W, C_W change with model width. This can be expressed in terms of width-dependent factors a_W, b_W, c_W, such that $A_W \propto a_W$, $B_W \propto b_W$, $C_W \propto c_W$. The values these factors take are what characterize a particular scheme. For μP these are given in Table 4. For depth a similar result has been proved using depth-μP [36], albeit in a restricted setting. When we refer to μP in the paper we assume the depth-μP scaling rules (Table 1, 'Residual' column).

Transferable HPs μP focuses on the subset of HPs whose optimal values we expect to *transfer across* axes such as width and depth. We term these μTransferable HPs. All μTransferable HPs function as multipliers and can be split into three kinds, which contribute to the three (non-HP) multipliers given by the abc-parametrization: $\alpha_W, \sigma_W, \eta_W$ where $A_W \propto \alpha_W, B_W \propto \sigma_W, C_W \propto \eta_W$. The difference between these multipliers and the ones that define a parametrization is that they are specified by the user, rather than being a function of width.

Table 4: The scaling rules defining μP. The type of a weight is determined by whether fan-in & fan-out both depend on width (hidden), only fan-out does (input), or only fan-in (output). Hence fan-in is always a multiple of width here.

	ABC-multiplier		Weight (W) Type		
			Input	Hidden	Output
μP	parameter	(a_W)	1	1	$1/\text{fan-in}(W)$
	initialization	(b_W)	1	$1/\sqrt{\text{fan-in}(W)}$	1
	Adam LR	(c_W)	1	$1/\text{fan-in}(W)$	1

[1] Lingle suggests independent weight decay is unstable, but we find it to be more so than Adam or standard AdamW.

Base shape Two additional (non-µTransferable) HPs introduced by µP are the base-width and base-depth. This refers to a mechanism where a user specifies a particular shape for the model, where its behavior under µP and SP are the same. The µP model still *scales* according to the abc-rules, so for all other shapes the two models will be different. This is implemented by dividing the µP scaling rules for the given model by those of a fixed-shape model at the base-width and base-depth.

Putting this together with our abc-parametrization given in Equation (10), and the µTransferable HPs outlined above, we now derive our final, absolute expressions for A_W, B_W, C_W:

$$A_W \leftarrow \alpha_W \frac{a_W}{a_{W_{\text{base}}}}, \quad B_W \leftarrow \sigma_W \frac{b_W}{b_{W_{\text{base}}}}, \quad C_W \leftarrow \eta_W \frac{c_W}{c_{W_{\text{base}}}} \tag{11}$$

Though base shapes are necessary for µP, they are not typically swept. Rather, they are considered a preference of the user, who may wish to retain the behavior of an existing SP model at a given shape.

Choosing HPs to sweep In theory, the search space of µTransferable HPs includes $\alpha_W, \sigma_W, \eta_W$ for every parameter tensor W in the model. In practice far fewer HPs are swept, with global grouping often used for σ_W and η_W, and many α_Ws dropped or grouped across layers.

Table 5 outlines the ways in which users of µP in the literature have approached HP sweeping. These all follow the approach used in Tensor Programs V of a random sweep, sampling combinations from the joint space of all HPs. The authors of Tensor Programs V note that other more complex methods may be more efficient, but these are considered beyond the scope of their work and have not been used widely. A Bayesian search method was used for the development of MiniCPM [8], but the authors give no further details—as they use 400 runs in their sweep it is not clear that this approach makes HP search easier.

Table 5: Sweeping configurations used for a selection of µP models from the literature. The sweeping process is similar across models, the only differences being the choice of discrete or continuous distributions and their ranges.

Model	proxy/target tokens used	proxy/target model size	sweep size	base width	HPs swept
T.P.V WMT14 [34]	100%	7.1%	64		$\eta, \alpha_{\text{out}}, \alpha_{\text{attn}}$
T.P.V $\text{BERT}_{\text{large}}$ [34]	10%	3.7%	256	?	$\eta, \eta_{\text{emb}}, \alpha_{\text{out}}, \alpha_{\text{attn}}, \alpha_{\text{LN}}, \alpha_{\text{bias}}$
T.P.V GPT-3 [34]	1.3%	0.6%	350		$\eta, \sigma, \alpha_{\text{emb}}, \alpha_{\text{out}}, \alpha_{\text{attn}}, \alpha_{\text{pos}}$
MiniCPM [8]	0.008%	0.45%	400	256	$\eta, \sigma, \alpha_{\text{emb}}, \alpha_{\text{residual}}$
Cerebras-GPT [5]	1.1%	1.5%	200	256	$\eta, \sigma, \alpha_{\text{emb}}$
SµPar [6]	6.6%	6.4%	350	256	$\eta, \sigma, \alpha_{\text{emb}}$

D.2. Low-precision training

All the major potential bottlenecks of model training—compute, communication and storage—see roughly linear improvements as the bit-width of their number format is reduced. In modern LLM training, the compute cost of large matrix multiplications (matmuls) means that substantial gains are available if these can be done in low-precision (< 32 bit) formats. With the ending of Dennard scaling and Moore's law [7, 23], the use of low-precision formats represents one of the most promising avenues towards increased efficiency in deep learning.

The standard numerical representations used in deep learning are the set of formats defined by the IEEE 754 floating-point standard [9]. IEEE floats comprise three elements: a sign bit, exponent bits, and mantissa bits. The number of exponent bits determines the *range* of a format, while the mantissa determines the *precision*. The default format used for training is the single-precision floating-point format, commonly known as FP32. The 16-bit FP16 and BF16 formats were later introduced, and more recently the FP8 E5 & E4 formats [17, 18, 22]. The use of multiple formats is known as *mixed precision* [16].

Recent AI hardware offers substantial acceleration for the 8-bit FP8 E4 and E5 formats. However the reduced range of these formats means that they cannot directly represent some values generated during training. [16] address this by introducing a fixed global *loss-scale* HP, which multiplies the loss value in the backward pass, artificially up-scaling gradients to lie within FP16 range. *Automatic loss scaling* [11] builds upon this idea, making the loss-scale a dynamic value that is tuned during training. The scaling within Transformer Engine [19] introduces per-tensor dynamic re-scaling, but this comes at the cost of added complexity and potential overheads.

D.3. Unit Scaling

An example: the unit-scaled matmul op Here we outline the procedure for calculating the scaling factor of a matmul op, which practitioners can use as a guide for scaling new ops that we do not cover in this paper (see Appendix B).

There are two potential approaches here. The first is to derive scaling factors from an analysis of an op's dynamics. Specifically, given the assumption of unit-scaled inputs, the appropriate scaling factor is the reciprocal of the expected output scale. For a basic matrix-matrix matmul we have,

$$\mathrm{matmul}(X, W) = XW, \qquad X \in \mathbb{R}^{d_{\text{batch}} \times d_{\text{fan-in}}},\ W \in \mathbb{R}^{d_{\text{fan-in}} \times d_{\text{fan-out}}},$$

where weights and activations are sampled i.i.d. from a centered Gaussian:

$$X_{ij} \sim \mathcal{N}(0, \sigma_X^2),\ W_{jk} \sim \mathcal{N}(0, \sigma_W^2).$$

From this we can derive the expected output scale (i.e. $\sigma(\mathrm{matmul})$):

$$\mathrm{matmul}(X, W)_{ik} = \sum_{j=1}^{d_{\text{fan-in}}} X_{ij} W_{jk},$$
$$\sigma\left(\mathrm{matmul}(X, W)_{ik}\right) = \sqrt{d_{\text{fan-in}}}\, \sigma_W\, \sigma_X.$$

Under Unit Scaling we have $\sigma_W = \sigma_X = 1$, and hence the scaling factor required to ensure a unit-scaled output is $1/\sqrt{d_{\text{fan-in}}}$. This gives our final unit-scaled matmul:

$$\text{u-matmul}(X, W) = \mathrm{matmul}(X, W)/\sqrt{d_{\text{fan-in}}}$$

Applying unit scaling To apply Unit Scaling to a model and train in low-precision, the following steps are required:

1. Scale parameter initializations to have zero-mean and unit variance.
2. Replace operations with their unit-scaled equivalents (including and especially the loss, matmuls and residual-adds).

3. *Constrain* the scales of operations which are required to have the same forward and backward factors.

4. Place a simple `.to(fp8)` cast on the inputs to matmuls.

Step 3 relates to the problem of conflicting scales in the forward and backward passes. A single linear layer in a differentiated model requires 3 matmul ops in the forward and backward passes, each requiring a different scaling factor ($\frac{1}{\sqrt{d_{\text{fan-in}}}}, \frac{1}{\sqrt{d_{\text{fan-out}}}}, \frac{1}{\sqrt{d_{\text{batch-size}}}}$). However, using these directly would give invalid gradients. The compromise here is that the activations and activation gradients have their scaling factors *constrained* such that they are equal (the Unit Scaling paper recommends taking the geometric mean; we modify this for u-µP in Appendix B to simply use the forward scale everywhere). Weight gradients can still be given their own scaling factor.

Appendix E. Justifying the u-µP hyperparameter scheme

E.1. A principled approach to hyperparameters

The problem of selecting HPs to sweep can be framed as choosing a subset of the per-tensor $\alpha_W, \sigma_W, \eta_W$ HPs outlined in Section 2.1, and grouping across/within layers. As shown in Table 5, µTransfer experiments in the literature have done this in a variety ways. Practitioners have not justified these choices, appearing to rely on a mixture of precedent and intuition. We outline two major downsides to the lack of a principled approach.

Firstly, not all groupings of HPs are suitable. Consider the commonly-used global σ_{init} HP. At initialization the activations going into the FFN swish function have $\text{std}(x_{\text{swish}}) \propto \sigma_{W_{\text{gate}}}$, whereas the self-attention softmax activations have $\text{std}(x_{\text{attn}}) \propto \sigma_{W_{\text{Q}}} \sigma_{W_{\text{K}}}$. A global σ HP thus has a linear effect on the FFN and a quadratic effect on attention, suggesting that this grouping may be unideal.

Secondly, not all HPs are independent of one another. The key example of this is the interaction between σ_W and η_W. The relative size of a weight update is determined by the ratio η_W / σ_W, not by either HP individually. Because of this, the optimal values for σ and η depend on each other, which we demonstrate empirically in Appendix A.2. This can make the problem of HP search much harder, and may be why hundreds of random-search runs have been required for sweeps in the literature.

To this end, we propose the following ideal criteria:

1. **Minimal cardinality**: the use of as few HPs as possible.

2. **Maximal expressivity**: the ability to still express any model defined using the per-tensor $\alpha_W, \sigma_W, \eta_W$ HPs outlined in Section 2.1 (in practice, we relax this slightly).

3. **Minimal interdependency**: the optimal value of each HP should not depend on the value of other HPs, simplifying the search space.

4. **Interpretability**: there should be a clear explanation for what an HP's value 'means' in the context of the model.

The u-µP HPs given in Table 6 are designed to satisfy these criteria, to the fullest extent possible. The placement of these HPs in the model is given in Table 3.

Cardinality & expressivity We arrive at our set of HPs in three steps, starting with the full $\alpha_W, \sigma_W, \eta_W$ for each weight tensor W. Firstly, we can choose to 'drop' any one of these three HPs by permuting under abc-symmetry, such that one HP $= 1$. As we want our weights to begin with unit scale, we choose σ_W, leaving just α_W, η_W.

Secondly, we observe that several of the α_W HPs combine linearly with other α_W HPs, providing an opportunity to re-parametrize with a single HP. We thus associate α HPs with operations instead of weights. This applies to all operations, unless they are unary and k-homogeneous for $k \geq 0$, in which case they propagate scale and don't require an HP. This results in the set of HPs shown, with their placement in the model given in Table 3.

Thirdly, we use a single global η and group α HPs across layers. This breaks our expressivity criterion, but we argue represents the best trade-off between expressivity and cardinality.

Table 6: Typical transformer HPs used under different schemes. *Basic* HPs in **bold** are considered most impactful and are commonly swept. *Extended* HPs in non-bold are not always swept, often set heuristically or dropped.

SP	μP	u-μP
$\boldsymbol{\eta}$	$\boldsymbol{\eta}$	$\boldsymbol{\eta}$
σ-scheme	$\boldsymbol{\sigma_{\text{init}}}$	
	$\boldsymbol{\alpha_{\text{emb}}} \vert \boldsymbol{\eta_{\text{emb}}}$	$\alpha_{\text{ffn-act}}$
	α_{attn}	$\alpha_{\text{attn-softmax}}$
	α_{out}	α_{res}
	base-width	$\alpha_{\text{res-attn-ratio}}$
	base-depth	$\alpha_{\text{loss-softmax}}$

Interdependency The second stage above, moving α HPs from weights into subsequent operations, not only reduces the number of HPs, but also minimizes the interdependence between those that remain. Interactions between HPs are complex and unlikely to be entirely separable, but we find that u-μP's optimal HP values depend less on each other than under μP (see Appendix A.2).

Interpretability The combination of unit scale and reduced dependencies between HPs means that each α can be interpreted as determining some fundamental property of the model at initialization. For example, the $\alpha_{\text{loss-softmax}}$ HP defines the (inverse of) the softmax's *temperature* for a unit-scaled input. Finally, we choose not to include base shape HPs in u-μP. They do not add to expressivity, lack a clear interpretation (besides alignment to a base model at a particular shape), break the interpretations of other HPs (as given above), and complicate implementation.