Lecture 11

Scaling – case study and details

CS336

Motivation today

What is the best practices for scaling and hparam tuning LMs?

- Does chinchilla's approach to scaling actually work?
- Can we save compute when training and fitting these things?
- Should we be picking particular architectures / parametrizations to scale nicely?

Scaling in practice

The newest model we talked about with scaling details - 2022

DeepMind

Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

*Equal contributions

What about more recently?

cerebras

Cerebras-GPT: Open Compute-Optimal Language Models Trained on the Cerebras Wafer-Scale Cluster

Nolan Dey, Gurpreet Gosal, Zhiming (Charles) Chen, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, Joel Hestness

Cerebras Systems {nolan,joel}@cerebras.net

2023

deepseek

DeepSeek LLM
Scaling Open-Source Language Models with Longtermism

2024

MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies

Shengding Hu[1], Yuge Tu[2], Xu Han[1]*, Chaoqun He[1], Ganqu Cui[1], Xiang Long[2], Zhi Zheng[2], Yewei Fang[2], Yuxiang Huang[1], Weilin Zhao[1], Xinrong Zhang[1], Zheng Leng Thai[1], Kaihuo Zhang[2], Chongyi Wang[2], Yuan Yao[1], Chenyang Zhao[1], Jie Zhou[2], Jie Cai[2], , Zhongwu Zhai[2], Ning Ding[1], Chao Jia[2], Guoyang Zeng[2], Dahai Li[2], Zhiyuan Liu[1]*, Maosong Sun[1]

[1]Department of Computer Science and Technology, Tsinghua University.

[2]Modelbest Inc.

shengdinghu@gmail.com

2024

Meta

The Llama 3 Herd of Models

Llama Team, AI @ Meta[1]

[1]A detailed contributor list can be found in the appendix of this paper.

2024

Hunyuan-Large: An Open-Source MoE Model with 52 Billion Activated Parameters by Tencent

Tencent Hunyuan Team

2024

MINIMAX

MiniMax-01: Scaling Foundation Models with Lightning Attention

MiniMax[1]

2025

Maximum update parametrization – in depth

Recall – the maximum update parametrization makes appealing claims

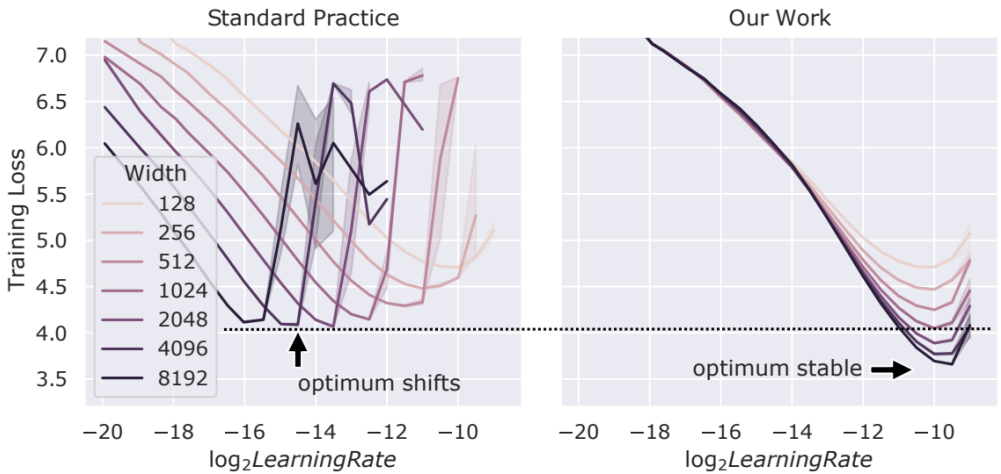


Table 2. μP function for a model M' that is r times the widths of M. If a parameter tensor has 2 dimensions that goes infinite when the model width goes infinite, it is "matrix-like" (*e.g.,* a fully-connected hidden layer); if the number is 1 or 0, it belongs to the "others" class. Note that embedding layers are "others". "Output" means the layer that maps an infinite dimension to a finite dimension, which is the word decoding layer (lm_head) in Transformers. A multiplier is a constant multiplied by a parameter tensor, which has a similar function to softmax temperature.

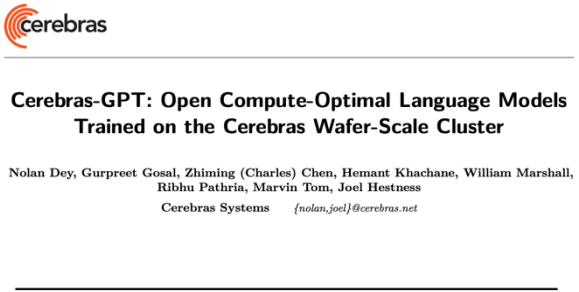
Hyperparameter (weight)	M	$M' \sim r$
AdamW learning rate (matrix-like)	l	l/r
AdamW learning rate (others)	l	l
Initialization variance (matrix-like)	σ	σ/r
Initialization variance (others)	σ	σ
Multiplier (output)	τ	τ/r
Multiplier (others)	τ	τ

Scale-invariant hyperparameter tuning would be very nice.

How does it work, and does it work in practice?

Recent models with detailed, public scaling recipes

1. Cerebras-GPT


cerebras

Cerebras-GPT: Open Compute-Optimal Language Models Trained on the Cerebras Wafer-Scale Cluster

Nolan Dey, Gurpreet Gosal, Zhiming (Charles) Chen, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, Joel Hestness

Cerebras Systems {nolan,joel}@cerebras.net

2. MiniCPM


MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies

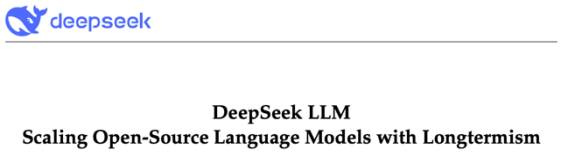
Shengding Hu[1], Yuge Tu[2], Xu Han[1]*, Chaoqun He[1], Ganqu Cui[1], Xiang Long[2], Zhi Zheng[2], Yewei Fang[2], Yuxiang Huang[1], Weilin Zhao[1], Xinrong Zhang[1], Zheng Leng Thai[1], Kaihuo Zhang[2], Chongyi Wang[2], Yuan Yao[1], Chenyang Zhao[1], Jie Zhou[2], Jie Cai[2], , Zhongwu Zhai[2], Ning Ding[1], Chao Jia[2], Guoyang Zeng[2], Dahai Li[2], Zhiyuan Liu[1]*, Maosong Sun[1]

[1]Department of Computer Science and Technology, Tsinghua University.

[2]Modelbest Inc.

shengdinghu@gmail.com

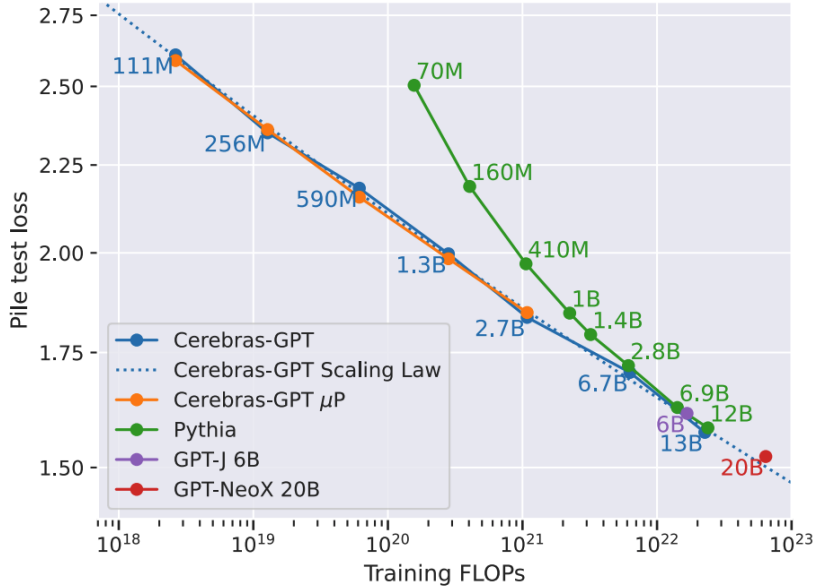
3. DeepSeek


deepseek

DeepSeek LLM
Scaling Open-Source Language Models with Longtermism

CerebrasGPT

CerebrasGPT – 0.1 to 13B models trained with the Chinchilla recipe.



Core finding – using muP parametrization makes scaling more stable

Hyperparam scaling strategy

Cerebras GPT authors find more predictable scaling from muP parametrization

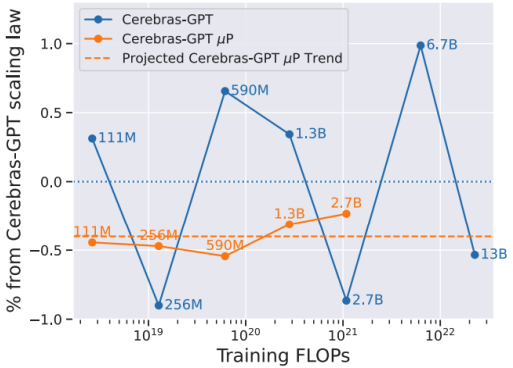


Figure 5: Percentage loss increase relative to Cerebras-GPT scaling law plotted against training FLOPs.

We train a set of Cerebras-GPT models using µP. We follow the µTransfer approach by first tuning hyperparameters for a small, 40M parameter µP model. Then, we transfer the hyperparameters along our µP scaling law up to a 2.7B parameter model. µP requires small changes to our baseline Cerebras-GPT models, including adding element-wise activation tensor scaling, adjusting initializers for affected layers, and adding layer-wise learning rates scaling to certain layers. We discuss the benefits we see with µP in Section 3.3. Refer to Appendix G for our tips to implement µP and our hyperparameter tuning notes.

muP parametrization

Appendix contains a very clear set of differences in parametrizing the model for scaling

	Standard Parameterization (SP)	Maximal Update (μP)
Variables		
W	A multiplicative or fully-connected weights tensor	
b	A bias weights tensor	
X, Y	Activation tensors: layer input, output, respectively	
$d_{\text{model,base}}$	Proxy (base) model's layer width	
d_{model}	Width of each layer	
d_{head}	Size of each attention head	
embed	Combined token + position embedding function	
η_{base}	The base learning rate (LR): Maximum in training schedule	
σ_{base}	The base initialization standard deviation	
m_{width}	—	Layer width multiplier: $d_{\text{model}}/d_{\text{model,base}}$
m_{emb}	—	Emdedding output multiplier

	Standard Parameterization (SP)	Maximal Update (μP)
Empirically Tuned Values		
$d_{\text{model,base}}$	—	256
η_{base}	Must tune for each model size	6e-3
σ_{base}	0.02	0.08
m_{emb}	—	10.0
Formulas		
Embedding initializer	$W_{\text{emb}} \sim N_{\text{trunc}}(0, \sigma^2_{\text{base}})$	
Embedding LR	$\eta_{\text{emb}} = \eta_{\text{base}}$	
Embedding output	$Y_{\text{emb}} = \text{embed}(X)$	$Y_{\text{emb}} = m_{\text{emb}} \cdot \text{embed}(X)$
LN initializer	$W_\gamma \sim 1, b_\beta \sim 0$	
LN LR	$\eta_{\text{LN}} = \eta_{\text{base}}$	
Bias initializer	$b \sim 0$	
Bias LR	$\eta_b = \eta_{\text{base}}$	
MHA equation	$\text{softmax}\left(\frac{Q^T K}{\sqrt{d_{\text{head}}}}\right) V$	$\text{softmax}\left(\frac{Q^T K}{d_{\text{head}}}\right) V$
QKV weights initializer	$W_{\text{qkv}} \sim N_{\text{trunc}}(0, \sigma^2_{\text{base}})$	$W_{\text{qkv}} \sim N_{\text{trunc}}(0, \sigma^2_{\text{base}}/m_{\text{width}})$
QKV weights LR	$\eta_{\text{qkv}} = \eta_{\text{base}}$	$\eta_{\text{qkv}} = \eta_{\text{base}}/m_{\text{width}}$
O weights initializer	$W_{\text{o}} \sim N_{\text{trunc}}(0, \frac{\sigma^2_{\text{base}}}{2 \cdot n_{\text{layers}}})$	$W_{\text{o}} \sim N_{\text{trunc}}(0, \frac{\sigma^2_{\text{base}}}{2 m_{\text{width}} \cdot n_{\text{layers}}})$
O weights LR	$\eta_{\text{o}} = \eta_{\text{base}}$	$\eta_{\text{o}} = \eta_{\text{base}}/m_{\text{width}}$
ffn1 weights initializer	$W_{\text{ffn1}} \sim N_{\text{trunc}}(0, \sigma^2_{\text{base}})$	$W_{\text{ffn1}} \sim N_{\text{trunc}}(0, \sigma^2_{\text{base}}/m_{\text{width}})$
ffn1 weights LR	$\eta_{\text{ffn1}} = \eta_{\text{base}}$	$\eta_{\text{ffn1}} = \eta_{\text{base}}/m_{\text{width}}$
ffn2 weights initializer	$W_{\text{ffn2}} \sim N_{\text{trunc}}(0, \frac{\sigma^2_{\text{base}}}{2 \cdot n_{\text{layers}}})$	$W_{\text{ffn2}} \sim N_{\text{trunc}}(0, \frac{\sigma^2_{\text{base}}}{2 m_{\text{width}} \cdot n_{\text{layers}}})$
ffn2 weights LR	$\eta_{\text{ffn2}} = \eta_{\text{base}}$	$\eta_{\text{ffn2}} = \eta_{\text{base}}/m_{\text{width}}$
Output logits multiplier	$Y_{\text{logits}} = W_{unemb} X$	$Y_{\text{logits}} = W_{unemb} X/m_{\text{width}}$

Setting the empirical values

muP is combined with aggressive scaling for hyperparameter optimization.

We tune µP hyperparameters on a 40M parameter proxy model, and µTransfer those hyperparameters to our models with 111M–2.7B parameters. Figure 13 shows the results of a 200 sample random hyperparameter search on a 40M parameter proxy model ($d_{\text{model}} = d_{\text{model,base}} = 256$, $n_{\text{layers}} = 32$, $d_{\text{head}} = 128$) trained on 600M tokens with a batch size of 131k tokens. From this sweep we obtained the following tuned hyperparameters for our µP models: $\eta_{\text{base}} = 6e\text{-}3$, $\sigma_{\text{base}} = 0.08$, $m_{\text{emb}} = 10$. These hyperparameters also closely match those used by Yang et al. (2021).

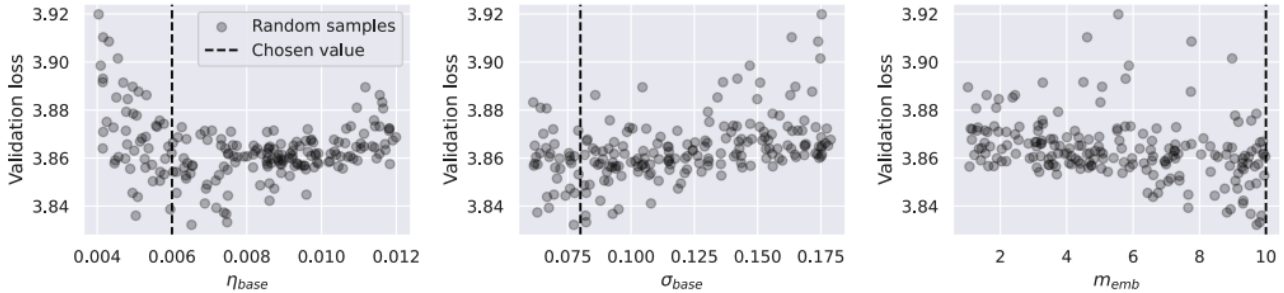


Figure 13: Results from random hyperparameter search on 40M parameter µP proxy model.

Generally stable hyperparameters.

MiniCPM

MiniCPM (2024) – new small, high-perf LM from Tsinghua group.

MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies

Shengding Hu[1], Yuge Tu[2], Xu Han[1]*, Chaoqun He[1], Ganqu Cui[1], Xiang Long[2], Zhi Zheng[2], Yewei Fang[2], Yuxiang Huang[1], Weilin Zhao[1], Xinrong Zhang[1], Zheng Leng Thai[1], Kaihuo Zhang[2], Chongyi Wang[2], Yuan Yao[1], Chenyang Zhao[1], Jie Zhou[2], Jie Cai[2], , Zhongwu Zhai[2], Ning Ding[1], Chao Jia[2], Guoyang Zeng[2], Dahai Li[2], Zhiyuan Liu[1]*, Maosong Sun[1]

[1]Department of Computer Science and Technology, Tsinghua University.

[2]Modelbest Inc.

shengdinghu@gmail.com

Careful, extensive scaling computations + muP to stabilize and simplify scaling

MiniCPM

High performance 1-2.5 B parameter models. These models beat most out 2Bs and match many modern 7B models.

Model	C-Eval	CMMLU	MMLU	HumanEval	MBPP	GSM8K	MATH
Llama2-7B	32.42	31.11	44.32	12.20	27.17	13.57	1.80
Qwen-7B	58.96	60.35	57.65	17.07	42.15	41.24	5.34
Deepseek-7B	42.82	44.45	47.82	20.12	41.45	15.85	1.53
Mistral-7B	46.12	42.96	62.69	27.44	45.20	33.13	5.00
Gemma-7B	42.57	44.20	60.83	38.41	50.12	47.31	6.18
Llama2-13B	37.32	37.06	54.71	17.07	32.55	21.15	2.25
MPT-30B	29.34	32.09	46.56	21.95	35.36	10.31	1.56
Falcon-40B	40.29	41.57	53.53	24.39	36.53	22.44	1.92
TinyLlama-1.1B	25.02	24.03	24.3	6.71	19.91	2.27	0.74
Qwen-1.8B	49.81	45.32	43.37	7.93	17.8	19.26	2.42
Qwen1.5-1.8B	**55.00**	50.85	43.81	5.49	24.82	26.16	3.25
Gemini Nano-3B	-	-	-	-	27.20	22.80	-
StableLM-Zephyr-3B	30.34	30.89	45.90	35.37	31.85	52.54	12.12
Phi-2(2B)	23.37	24.18	52.66	47.56	**55.04**	**57.16**	3.50
Gemma-2B	29.26	28.56	38.49	24.39	29.74	16.83	3.34
MiniCPM-1.2B	49.14	46.81	49.63	44.51	32.75	31.77	10.60
MiniCPM-2.4B	51.13	**51.07**	**53.46**	**50.00**	47.31	53.83	**10.24**

Techique 1: muP to stabilize scaling

Scale_emb = 12, scale_depth = 1.4, init_std = 0.1, lr =0.01

Name	Specific Operation
Embedding Output Scaling	Multiply the output of the embedding by $scale\_emb$
Residual Connection Scaling	Scale the increment at each residual connection in each layer by $scale\_depth/\sqrt{\text{num\_layers}}$
Initialization of Tensors	Set the initialization standard deviation of each two-dimensional tensor parameter to $init\_std/\sqrt{d_m/d_{base}}$, and set other parameters' initialization to 0.1
Learning Rate Scaling of Tensors	Adjust the learning rate of each two-dimensional tensor parameter to $1/(d_m/d_{base})$ times the learning rate of other parts (or the overall learning rate)
LM Head Scaling	Adjust the output logits to $1/(d_m/d_{base})$ times the original value

Table 7: List of operations used when applying tensor program techniques.

c.f. CerebrasGPT – Scale_emb = 10, lr=6e-3, init_base = 0.08

Scaling recipe / strategy

Use muP for initialization, fix the aspect ratio, scale up the overall model size.

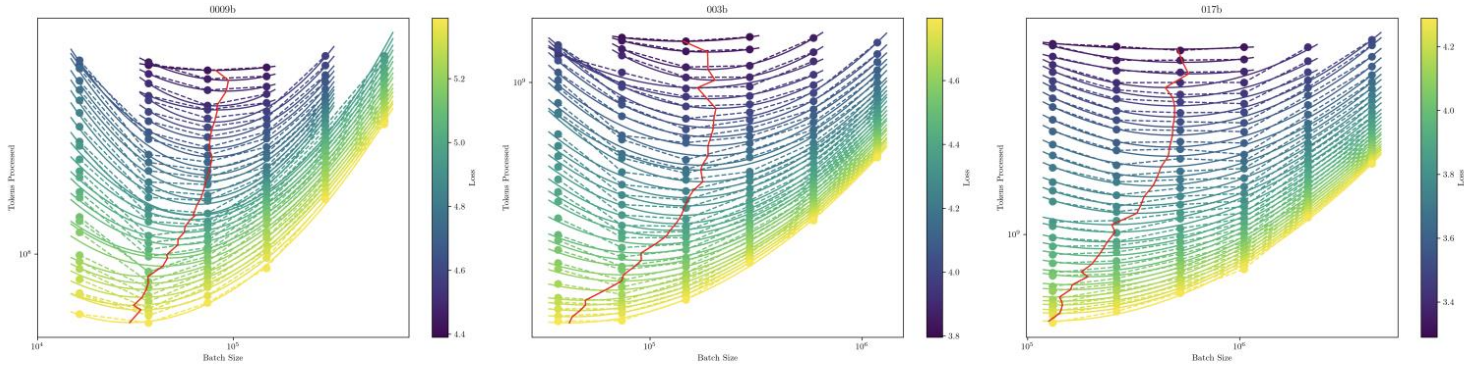
Name	N (B)	d_m	d_{ff}	d_h	n_h	L
9M	0.009	320	800	64	5	8
30M	0.036	512	1280	64	8	12
70M	0.066	640	1600	64	10	14
0.1B	0.109	768	1920	64	12	16
0.17B	0.166	896	2240	64	14	18
0.2B	0.241	1024	2560	64	16	20
0.5B	0.499	1344	3360	64	21	24

Note that the gap between the largest model here and the actual model they train is ~5x

Optimal batch, LR, token-to-size ratios are directly fitted via scaling analysis

Optimal batch

Three model sizes (9m, 30m, 170m) as a function of data size (y), batch (x) and loss (col)



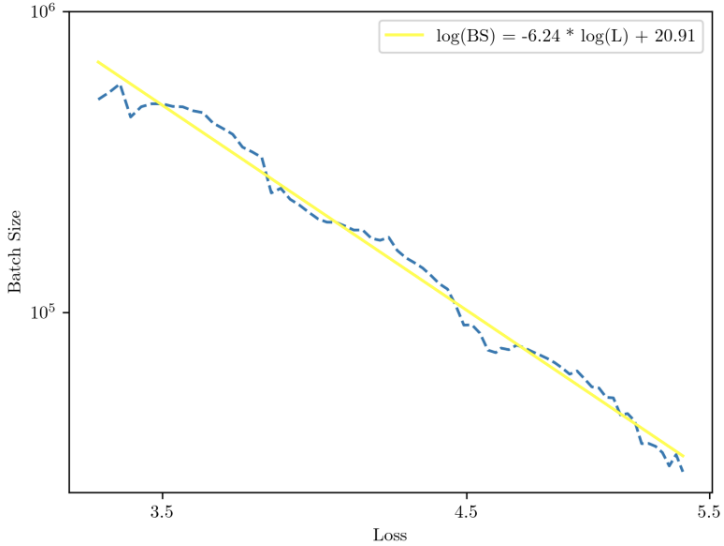
Vertical columns of points represent a single training curve (fixed batch, more points).

Red line attempts to identify minimum loss points for each y-value

– this is the ‘optimal batch size’ for a model size / dataset size combination.

Optimal batch size

We can then follow the Kaplan 2020 analysis and plot optimal batch size vs final loss.



Fairly clean trend – polynomially increase the batch size as loss decreases.

Optimal LR

According to muP – optimal learning rate should be (roughly) stable. Is it?

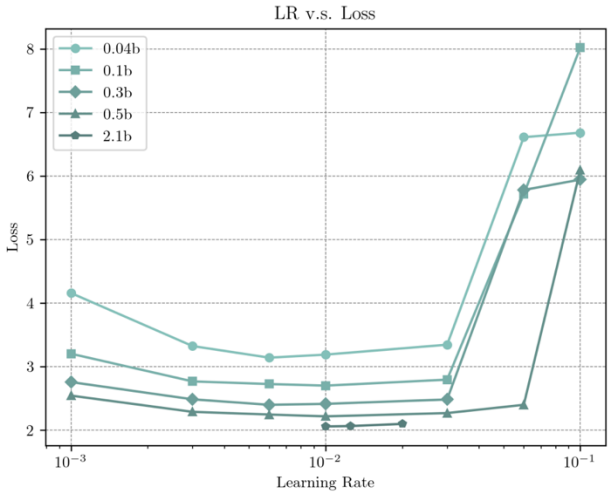
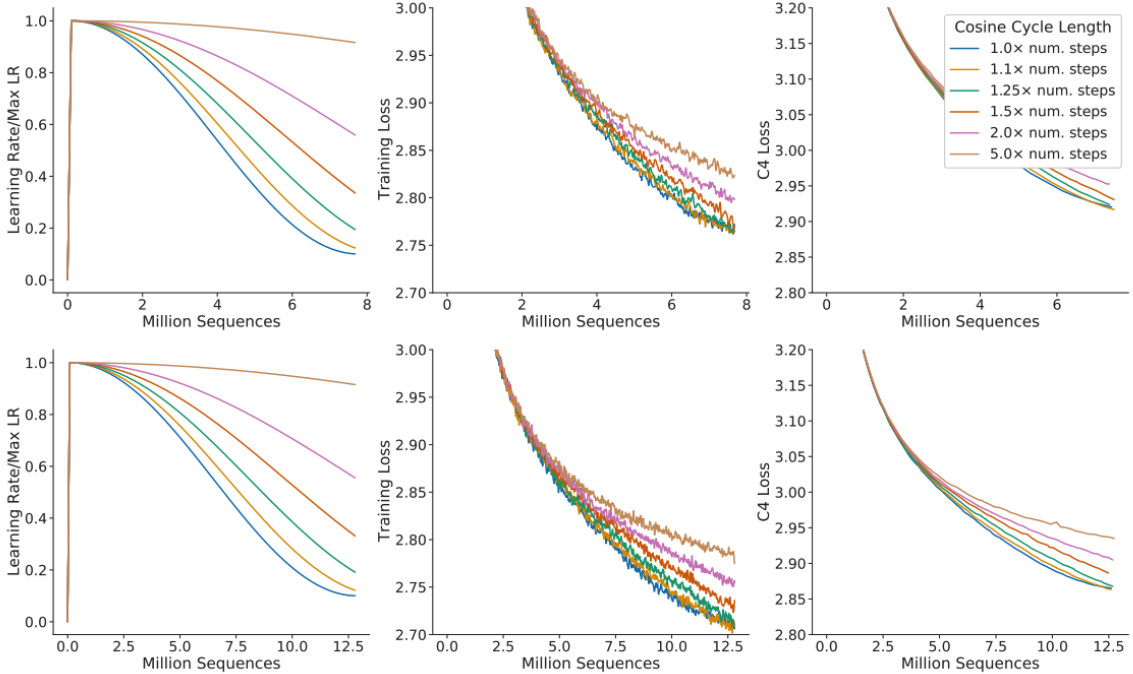


Figure 3: Loss vs Learning Rate. After applying for the Tensor Program, the learning rate shift becomes minimal.

What remains – model size vs data tradeoffs.

From chinchilla – to fit a scaling law, we need to train from scratch, not just early stop



This turns the cost of fitting a scaling law from n to n^2.. Can we avoid this?

(partial) solution in miniCPM – WSD learning rate

Instead of cosine, split learning rate into warmup, stable, and decay phases.

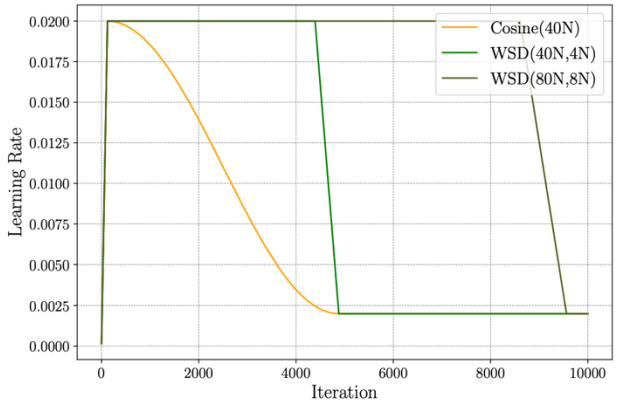


Figure 15: Illustrative comparison between Cosine LRS and WSD LRS. The WSD LRS with different end steps share the same stable training stage.

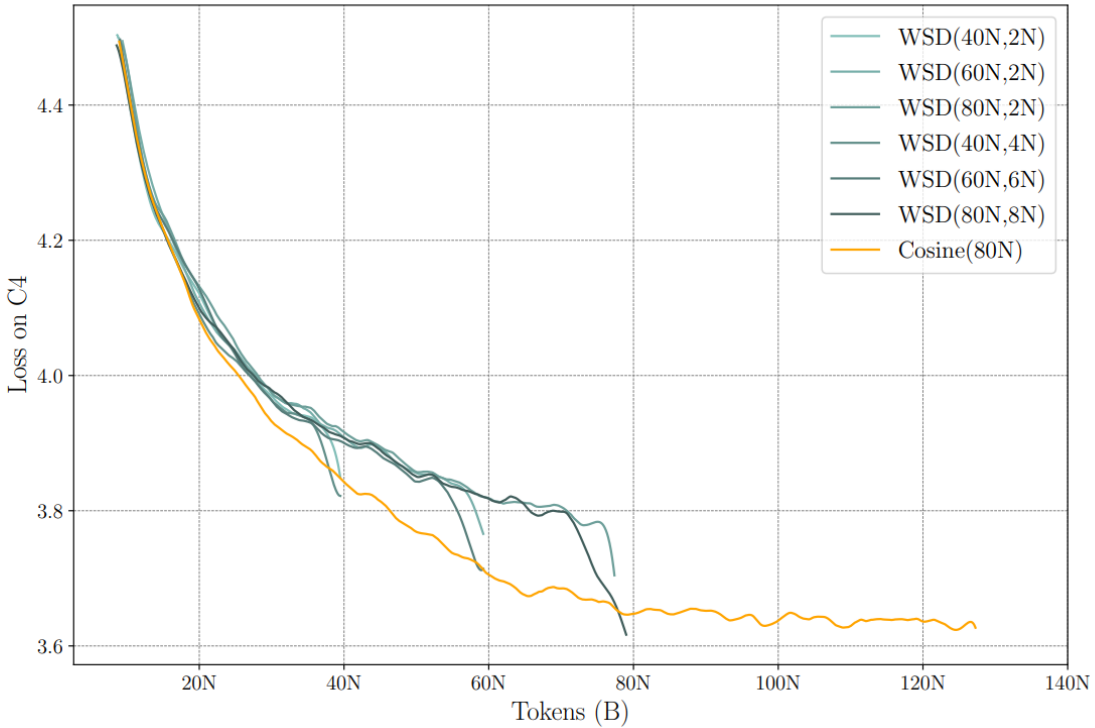
In this section, we introduce the utilization of the WSD scheduler as an effective approach to explore the scaling law with linear cost ($O(mC)$). Since WSD scheduler has the advantage of arriving at the optimal loss of Cosine LRS after decaying from stable stage's checkpoints of any step, we are now able to precisely measure the optimal scaling properties without re-training the models from scratch to different amount of tokens, thus making the scaling law measurement much more efficient along the data axis.

We measure the scaling law along the data and model axes by training SLMs of 6 sizes ranging from 0.04B to 2B, each with 6 decayed model starting from checkpoint of $10N$ to $60N$ data during the stable training stage. The final loss is evaluated on five held-out evaluation dataset. To potentially compare the loss when the model uses different tokenizer, we take the average of loss by number of bytes instead of number of tokens, following Achiam et al. (2023). The final loss of each pair of data size and model size is shown in the blue lines in Figure 17.

For chinchilla-style analysis, can restart the run at the end of the stable phase.

WSD learning rates work well in miniCPM

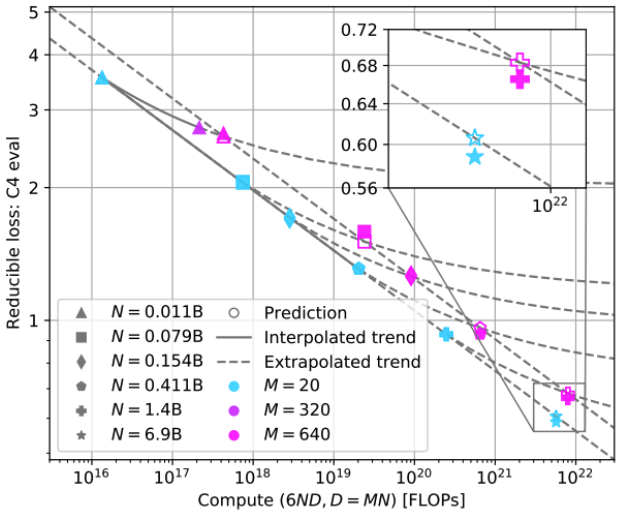
Slower during the stable phase, rapid loss decay during decay phase. Decay ~ 10%.



Side note – other ways of estimating chinchilla curves

Gadre et al propose other, curve-fitting based ways of doing similar things

Core idea – the 'penalty' from overtraining remains stable



$$L(N, D) = E + AN^{-\alpha} + BD^{-\beta}.$$

$$L(C, M) = E + \left(aM^{\alpha_C} + bM^{-\alpha_C}\right) C^{-\alpha_C}, \tag{4}$$

where $\alpha_C = \alpha/2$, $a = A(1/6)^{-\alpha_C}$, $b = B(1/6)^{-\alpha_C}$ gives the relation to Equation (3). For a complete derivation, see Appendix B.

Chinchilla-type analysis

Equipped with the WSD learning rate,

we can now try to find the optimal data-to-model size ratio

Then we fit the losses with model size N and data size D following Hoffmann et al. (2022) using scipy `curvefit` function:

$$L(N, D) = C_N N^{-\alpha} + C_D D^{-\beta} + L_0 \tag{2}$$

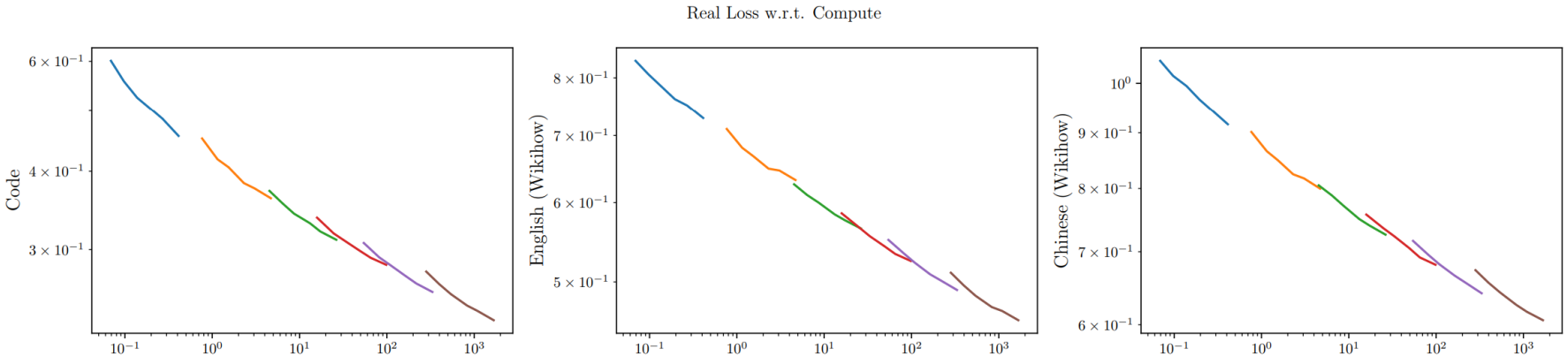
The fitted curve along the data axis for each dataset and each checkpoints are shown in orange lines in Figure 17. Then we have the optimal model size N_{opt}, dataset size D_{opt}, given a fixed amount of compute $C = 6ND$ (Rae et al., 2021) as:

$$\frac{N_{opt}}{D_{opt}} = K^2 \left(\frac{C}{6}\right)^{\eta}, \tag{3}$$

MiniCPM authors choose method 1 (lower envelope) and method 3 (joint fit)

Chinchlla method 1

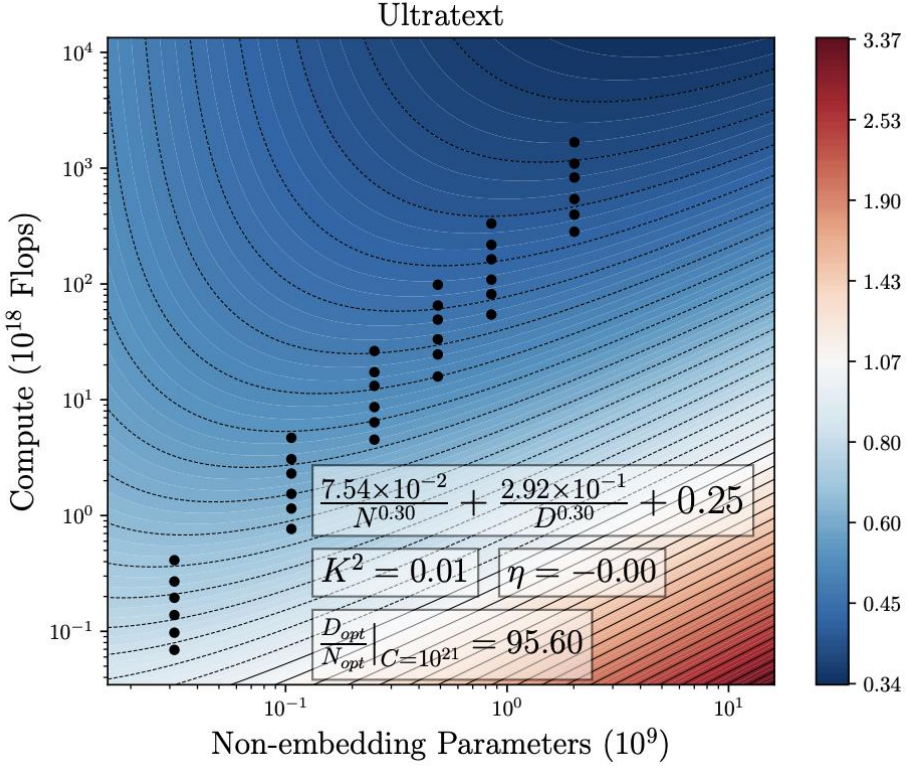
Fairly clear (though maybe not linear?) trends



Different colors indicate different models. Their runs suggest relatively low diminishing returns due to data.

Chinchilla method 3

Their primary scaling approach is the joint fit – they find *very* high data-model ratios.



Tiny models with lots of data

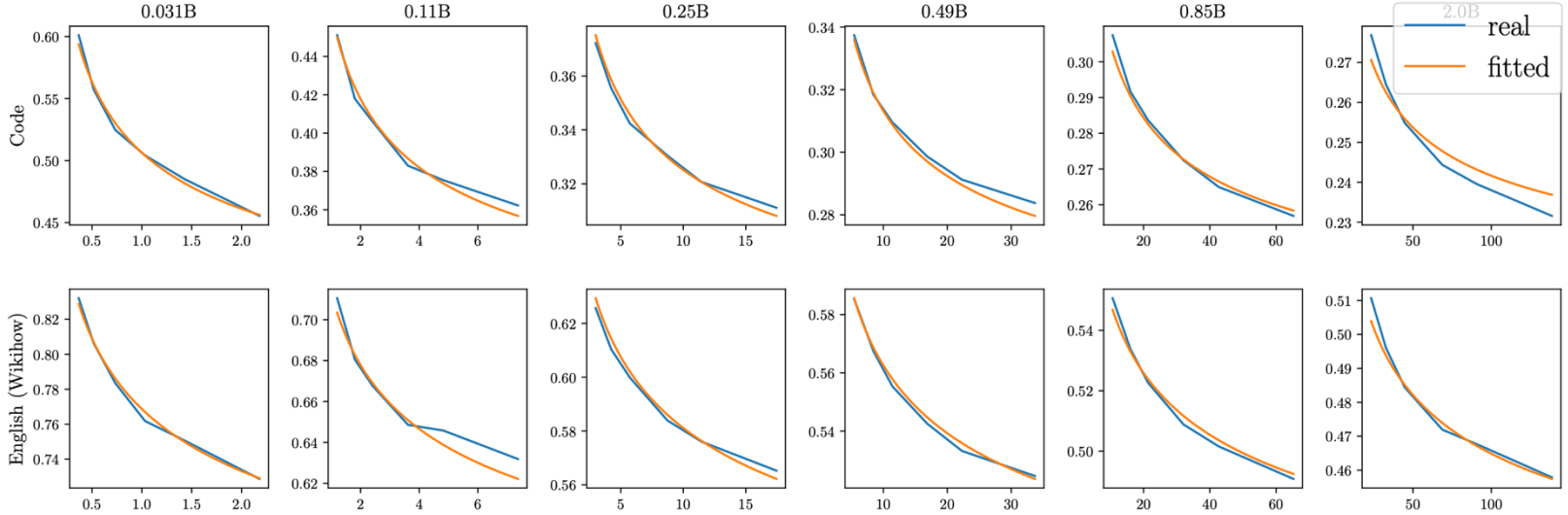
The overall data-to-model ratio is very high (192), though they argue LLaMA architectures should have a higher ratio.

> As for the concrete data-to-model ratio $\frac{D_{opt}}{N_{opt}}$, we notice that there is a huge gap in compute optimal regime of between ours and Hoffmann et al. (2022) despite that the trend of $\frac{D_{opt}}{N_{opt}}$, with compute C is aligned between ours and theirs. Specifically, the data size should be 192 times larger than the model size on average, as opposed to 20 times in Hoffmann et al. (2022). We note that this aligns with the observation in Section 4.3 and Figure 6.

Note that recent models like LLaMA 3 has *significantly* higher data-to-model ratios, suggesting that with more careful optimization, we might be able to go far beyond the 20*model_size rule of thumb.

Scaling curve fits are (generally) good

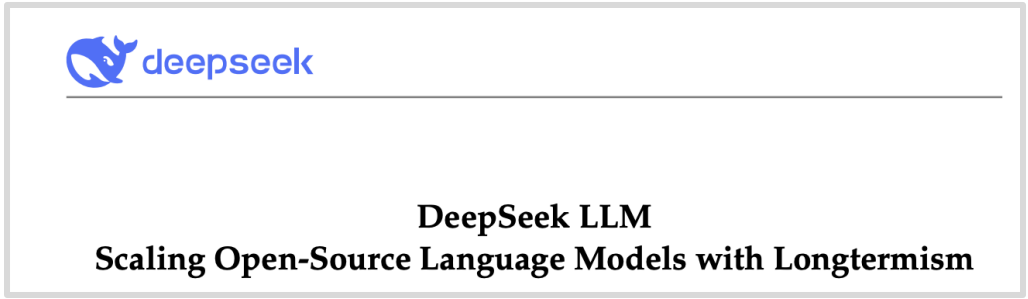
Overall fits and predictions of models across a large range of sizes is fairly good.



X-axis, number of tokens in billions.

DeepSeek

DeepSeek (2024) – another LM with careful scaling analysis



7 and 67B param models – generally high performance compared to other open LM

DeepSeek

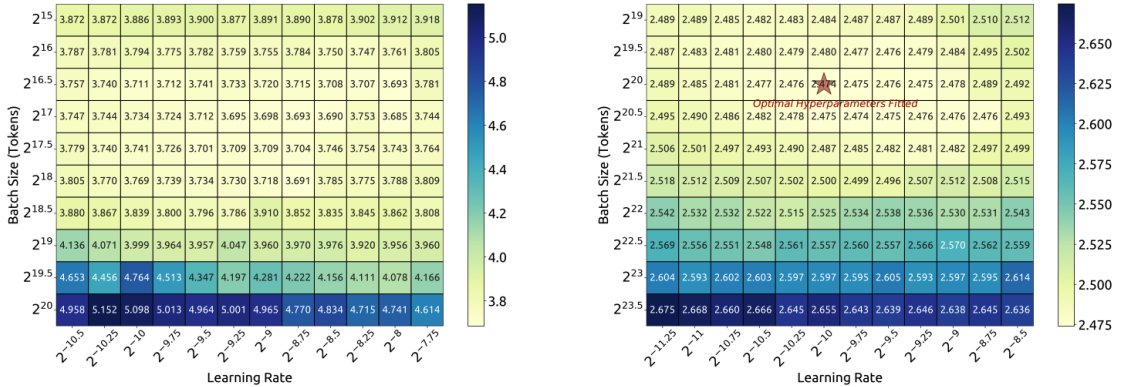
Language	Benchmark	Test-shots	LLaMA2 7B	DeepSeek 7B	LLaMA2 70B	DeepSeek 67B
English	HellaSwag	0-shot	75.6	75.4	**84.0**	**84.0**
	PIQA	0-shot	78.0	79.2	82.0	**83.6**
	WinoGrande	0-shot	69.6	70.5	**80.4**	79.8
	RACE-Middle	5-shot	60.7	63.2	**70.1**	69.9
	RACE-High	5-shot	45.8	46.5	**54.3**	50.7
	TriviaQA	5-shot	63.8	59.7	**79.5**	78.9
	NaturalQuestions	5-shot	25.5	22.2	36.1	**36.6**
	MMLU	5-shot	45.8	48.2	69.0	**71.3**
	ARC-Easy	0-shot	69.1	67.9	76.5	**76.9**
	ARC-Challenge	0-shot	49.0	48.1	**59.5**	59.0
	OpenBookQA	0-shot	57.4	55.8	**60.4**	60.2
	DROP	1-shot	39.8	41.0	**69.2**	67.9
	MATH	4-shot	2.5	6.0	13.5	**18.7**
	GSM8K	8-shot	15.5	17.4	58.4	**63.4**
	HumanEval	0-shot	14.6	26.2	28.7	**42.7**
	MBPP	3-shot	21.8	39.0	45.6	**57.4**
	BBH	3-shot	38.5	39.5	62.9	**68.7**
	AGIEval	0-shot	22.8	26.4	37.2	**41.3**
	Pile-test	-	0.741	0.725	0.649	**0.642**
Chinese	CLUEWSC	5-shot	64.0	73.1	76.5	**81.0**
	CHID	0-shot	37.9	89.3	55.5	**92.1**
	C-Eval	5-shot	33.9	45.0	51.4	**66.1**
	CMMLU	5-shot	32.6	47.2	53.1	**70.8**
	CMath	3-shot	25.1	34.5	53.9	**63.0**
	C3	0-shot	47.4	65.4	61.7	**75.3**
	CCPM	0-shot	60.7	76.9	66.2	**88.5**

Performance - Roughly comparable to LLaMA 2 models of equivalent size.

Scaling strategy – batch + LR

Scaling strategy: don't use any muP, directly estimate optimal batch / LR

We initially conducted a grid search for batch size and learning rate on small-scale experiments with a compute budget of 1e17, and the results of a specific model size (177M FLOPs/token) are illustrated in Figure 2(a). The results demonstrate that the generalization error remains stable across a wide range of choices of batch sizes and learning rates. This indicates that near-optimal performance can be achieved within a relatively wide parameter space.



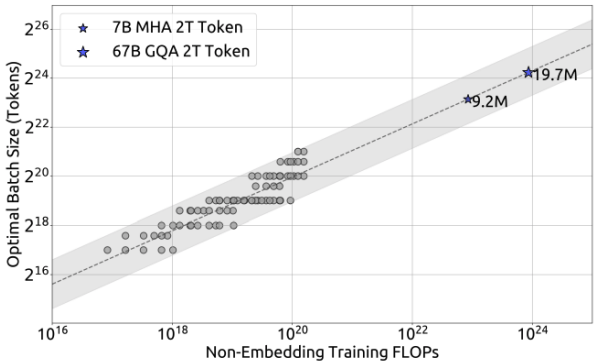
(a) 1e17 FLOPs (177M FLOPs/token)

(b) 1e20 FLOPs (2.94B FLOPs/token)

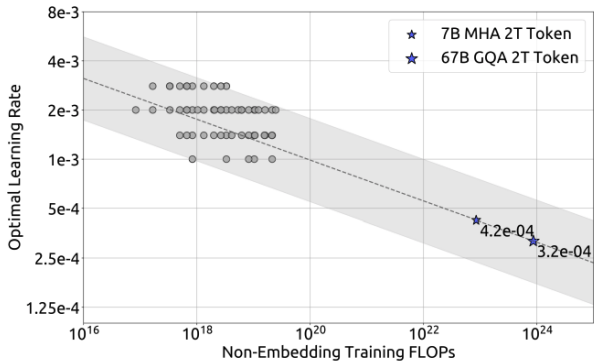
Scaling analysis of learning rates

Small scale runs + collect ‘near optimal’ (within 0.25% of min) models.

$$\eta_{\text{opt}} = 0.3118 \cdot C^{-0.1250}$$
$$B_{\text{opt}} = 0.2920 \cdot C^{0.3271} \quad (1)$$



(a) Batch size scaling curve



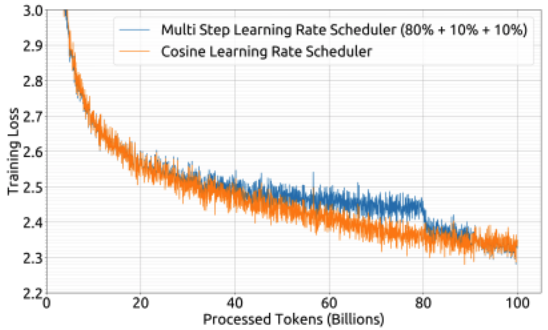
(b) Learning rate scaling curve

Learning rate fit looks a bit questionable..

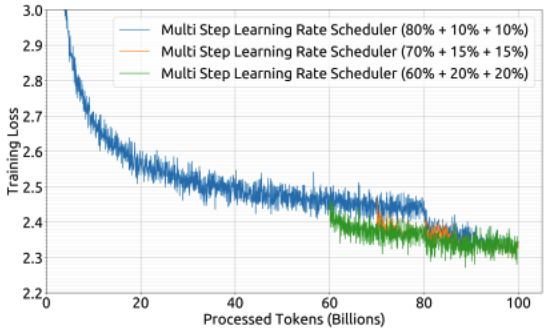
For chinchilla analysis: WSD-style learning rate

Deepseek uses WSD-style learning rate – fast warmup + two decay steps of 10% each.

A multi-step learning rate scheduler is employed during pre-training instead of the typical cosine scheduler. Specifically, the learning rate of the model reaches its maximum value after 2000 warmup steps, and then decreases to 31.6% of the maximum value after processing 80% of the training tokens. It further reduces to 10% of the maximum value after 90% of the tokens. The gradient clipping during the training phase is set to 1.0.



(a) Multi-step v.s. cosine learning rate decay

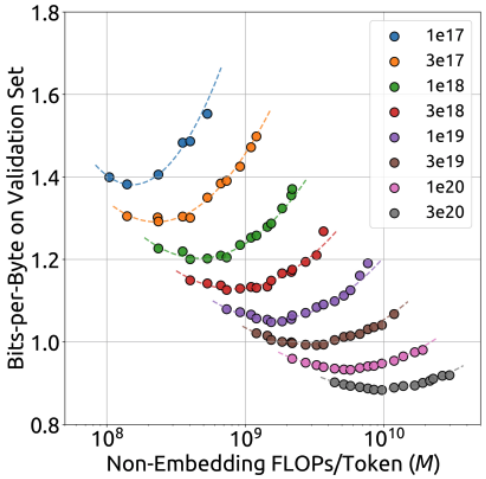


(b) Different proportions of multi-step stages

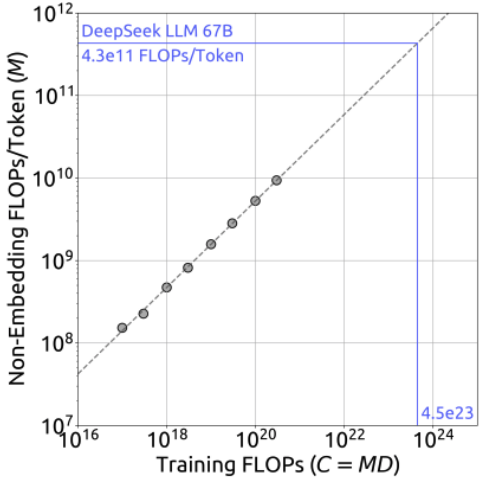
Generally seems to match performance of cosine learning rates.

Data-size tradeoff analysis: Chinchilla method 2

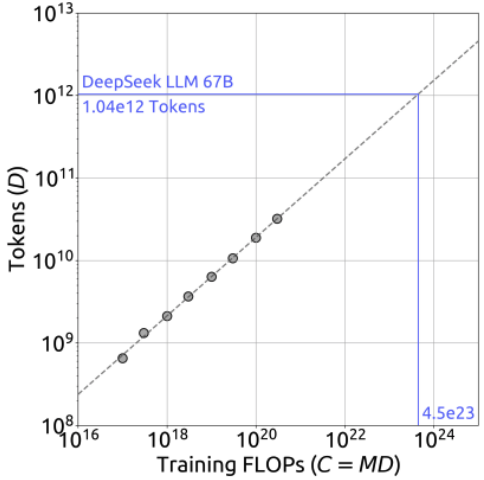
Straightforward isoflop-style analysis for selecting the model size tradeoffs.



(a) IsoFLOP curve



(b) Optimal model scaling



(c) Optimal data scaling

Scaling predicts final model loss

The fitted scaling models (generally) accurately predict the final model losses.

Additionally, we fitted the loss scaling curve according to compute budget C and optimal generalization error, and predicted the generalization error for DeepSeek LLM 7B and 67B, as shown in Figure 5. The results indicate that using small-scale experiments can accurately predict

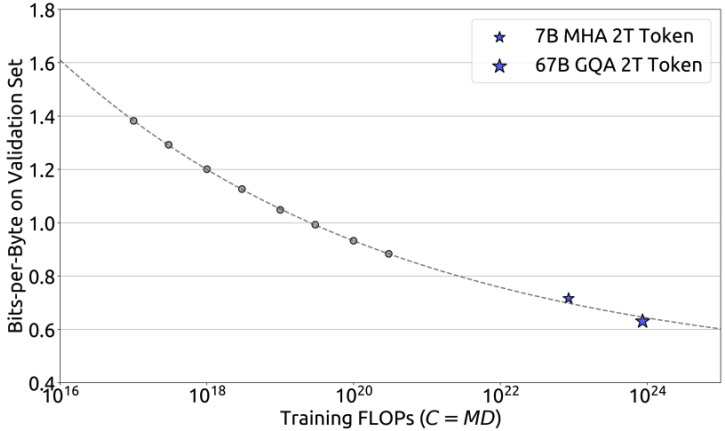


Figure 5 | Performance scaling curve. The metric is the bits-per-byte on the validation set. The dotted line represents the power law fitting the smaller model (grey circles). The blue stars represent DeepSeek LLM 7B and 67B. Their performance is well-predicted by the scaling curve.

LLaMA 3 (2024) Scaling laws

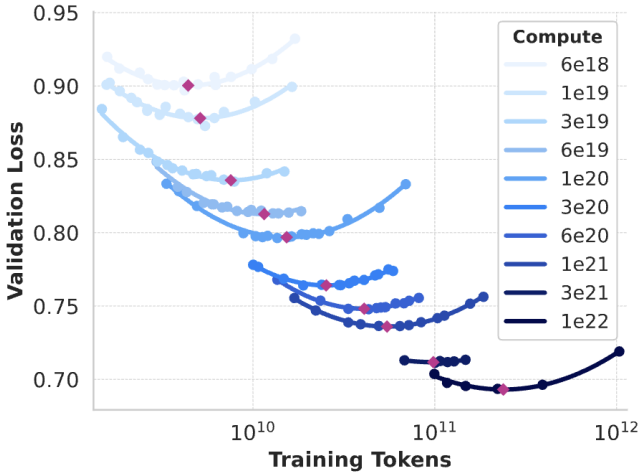
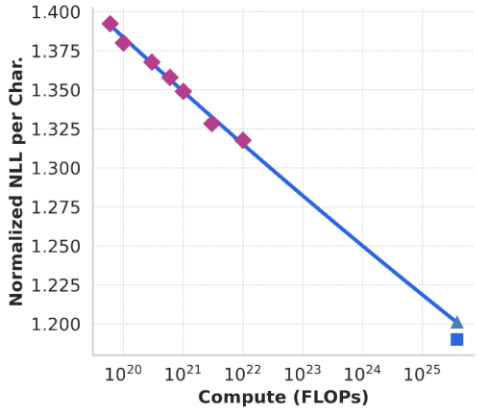
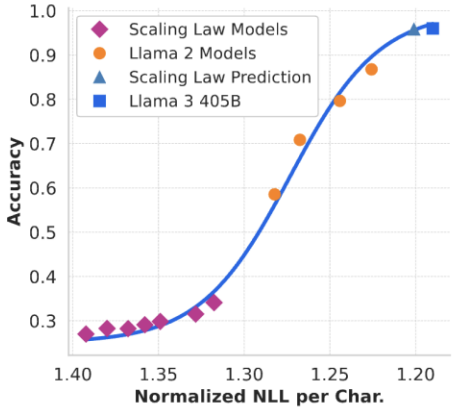


Figure 2 Scaling law IsoFLOPs curves between 6×10^{18} and 10^{22} FLOPs. The loss is the negative log-likelihood on a held-out validation set. We approximate measurements at each compute scale using a second degree polynomial.





Isoflops-style scaling (39-1 ratio)

Compute-to-downstream scaling

Hunyuan-1 (2024) large scaling laws

Yet more isoflops-style scaling (but this time for MoE parameter sizes)

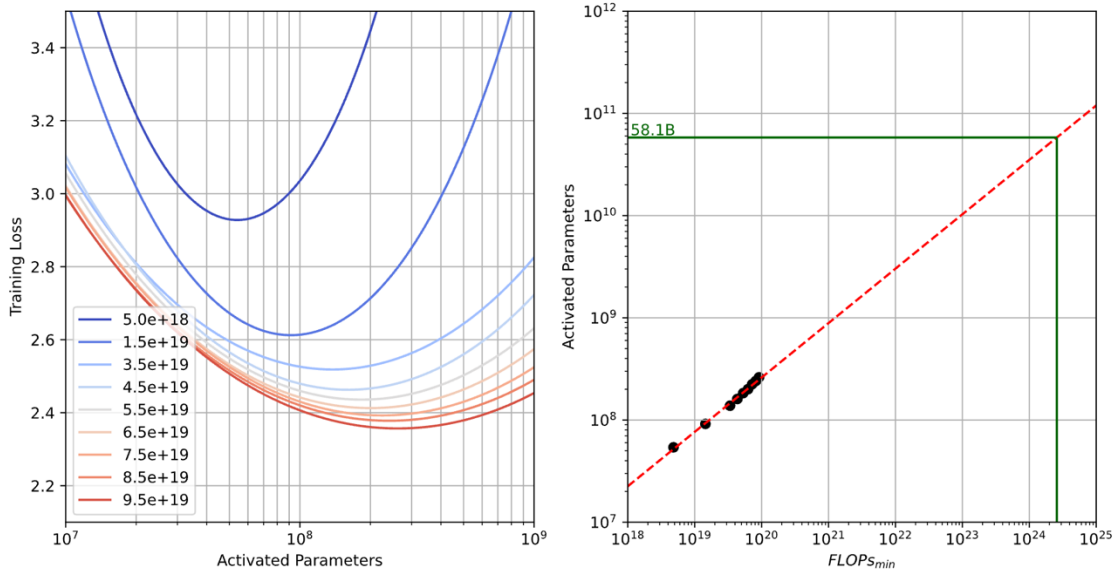


Figure 3: Using quadratic polynomial fitting, we obtain the scaling law of the optimal number of activation parameters under different minimum compute budgets.

Optimal ratio – 96-1 (data to active param)

MiniMax-01 (2025)

Architecture scaling laws + Chinchilla method 1

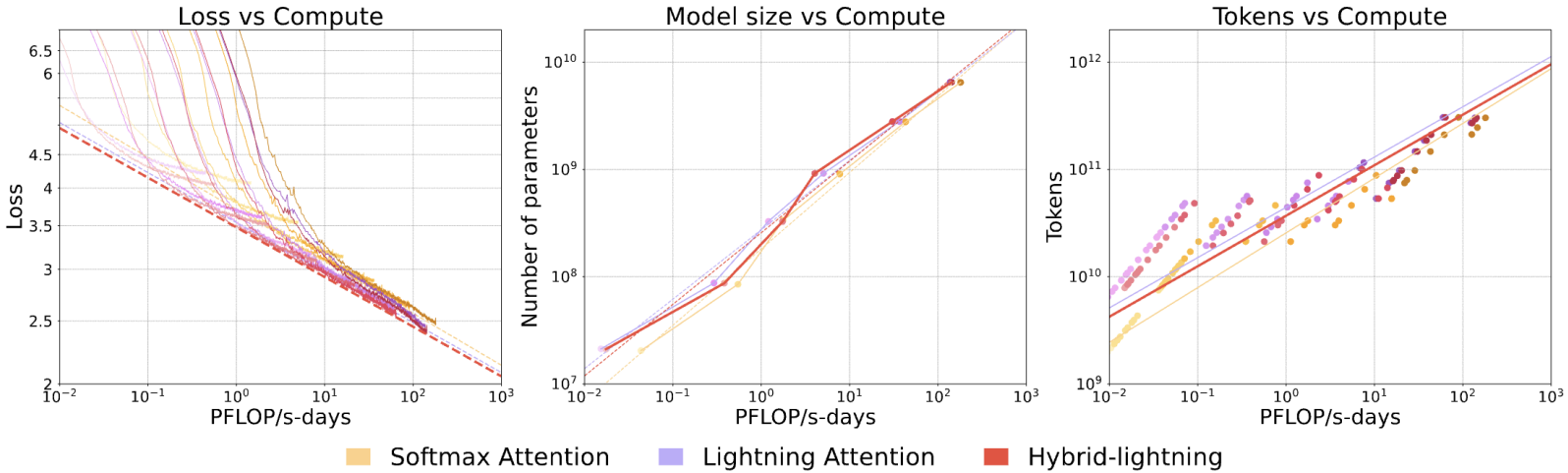


Figure 6 | **Summary of Scaling Laws.** Training curves (left) span models from 70M to 7B parameters. Optimal model size (center) and training tokens (right) are derived based on a specified compute budget estimation.

Recent scaling law recipes

CerebrasGPT

- Use muP to make hyperparams invariant to scale
- Directly use the chinchilla scaling formula

DeepSeek recipe

- Assume most transformer hypers are invariant to scale
- Do a scaling analysis on batch / LR to figure out optimal scaling
- IsoFLOP analysis to figure out model sizing
  - Use a piecewise-linear schedule to make chinchilla scaling cheap.

miniCPM recipe

- Use muP to make transformer + LR invariant to scale
- Use a piecewise linear schedule to get sample for Chinchilla method 3 (curve fitting)

Recent (late 2024+) but less detailed

LLaMA 3 / Hunyuan

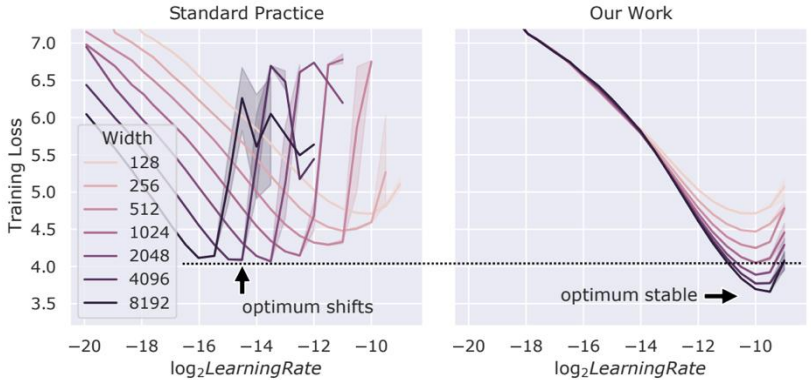
- Just isoflops (no other scaling details)

Minimax

- Architecture choice / decision scaling

Validating and understanding muP

“Scale invariant” hyperparameter tuning would be very useful



CerebrasGPT and miniCPT also use muP – is it actually useful?

Preprint

A Large-Scale Exploration of μ-Transfer

Lucas Dax Lingle
lucasdaxlingle@gmail.com

What is muP, anyway?

(this is a very accessible 'muP for babies' paper)

muP is based off the following assertion. As a function of the width of the network n_l..

A1: The activations at initialization should remain $\Theta(1)$
A2: After one gradient step, the change in activation should be $\Theta(1)$

Note: if individual activations are $\Theta(1)$, then the norm should be $\Theta\left(\sqrt{n_l}\right)$

Deriving muP (condition A1)

Suppose that we have a simple, deep linear network ($h_l = W_l h_{l-1}$) and we init $W_l \sim N\left(0, \sigma I_{n_l \times n_{l-1}}\right)$ then by basic matrix concentration $\|W_l\|_* \to \sigma(\sqrt{n_{l-1}} + \sqrt{n_l})$ and,

$$\|h_l\|_2 \approx \|W_l\|_* \|h_{l-1}\|_2$$

Now let's pick $\sigma = \frac{\sqrt{n_l}}{\sqrt{n_{l-1}}}\left(\sqrt{n_l} + \sqrt{n_{l-1}}\right)^{-1} = \Theta\left(\frac{1}{\sqrt{n_{l-1}}} \min\left(1, \sqrt{\frac{n_l}{n_{l-1}}}\right)\right)$. What happens?

Inductive assumption- $\|h_{l-1}\|_2 = \Theta\left(\sqrt{n_{l-1}}\right)$

Inductive case - $\|W_l\|_* \to \sigma\left(\sqrt{n_{l-1}} + \sqrt{n_l}\right) = \frac{\sqrt{n_l}}{\sqrt{n_{l-1}}}$

$$\|h_l\|_2 = \sqrt{n_l} + o(\sqrt{n_l})$$

[Comments – the ≈ here is a bit hand-wavy, as the min s.v. of a Gaussian is $\sqrt{n_l} - \sqrt{n_{l-1} - 1}$, and is non-uniform, like J.L.]

Deriving muP (condition A2)

Now we need to deal with updates. Suppose we have the update ΔW_l on the weights. For SGD, on a linear layer, this looks like a rank-one loss-activation outer product.

$$\Delta W_l = -\eta_l \nabla_{h_l} \ell \; h_{l-1}^\top$$

Thus, $\|\Delta W_l h_{l-1}\|_2 = \|\Delta W_l\|_* \|h_{l-1}\|_2$. Now note that we have the update

$$\Delta \mathrm{h_l} = \mathrm{W_l} \Delta h_{l-1} + \Delta W_l (h_{l-1} + \Delta h_{l-1})$$

Assuming that the leading order terms don't cancel, we see that

- $\mathrm{W_l} \Delta h_{l-1} = \Theta(\sqrt{n_l})$ from induction assumption + condition A1 argument
- $\Delta W_l h_{l-1} = \|\Delta W_l\|_* \sqrt{n_{l-1}}$ from above, thus $\|\boldsymbol{\Delta W_l}\|_* = \boldsymbol{\Theta}\left(\frac{\sqrt{n_l}}{\sqrt{n_{l-1}}}\right)$
- $\Delta W_l \Delta h_{l-1} = O(\|\Delta W_l\|_* \sqrt{n_{l-1}})$

Deriving muP (condition A2) part 2

Recall – we want all 3 terms of $\Delta h_l = W_l \Delta h_{l-1} + \Delta W_l (h_{l-1} + \Delta h_{l-1})$ to be $\Theta(\sqrt{n_l})$
And the key is to pick LR such that $\|\Delta W_l\|_* \sqrt{n_{l-1}} = \Theta(\sqrt{n_l})$. How can we do that?

Suppose that the loss update *also* scales O(1). Then we can write down..

$$\Delta \ell = \Theta(\langle \Delta W_l, \nabla_{W_l} \ell \rangle) = \Theta\left(\|\Delta W_l\|_F \left\|\nabla_{W_l} \ell\right\|_F \right) = \Theta\left(\|\Delta W_l\|_* \left\|\nabla_{W_l} \ell\right\|_* \right)$$

Where we use the fact that $\Delta W_l = -\eta \nabla_{W_l} \ell$ in standard SGD updates.

Now plug in $\Delta \ell = O(1), \|\Delta W_l\|_* = \Theta\left(\frac{\sqrt{n_l}}{\sqrt{n_{l-1}}} \right)$ to get that $\left\|\nabla_{W_l} \ell\right\|_* = \Theta\left(\frac{\sqrt{n_{l-1}}}{\sqrt{n_l}} \right)$

Finally, from the previous slide, recall that $\Delta W_l = -\eta_l \nabla_{h_l} \ell \; h_{l-1}^\top$ and thus

$$\eta_l = \Theta\left(\frac{n_l}{n_{l-1}} \right)$$

[with Adam, $\|\Delta W_l\|_* \sqrt{n_{l-1}} = \Theta(1)$]

muP mini recap..

So, what is (baby) muP about? Controlling activations (and changes) via W and ΔW

Initialization: Set to $\Theta\left(\frac{1}{\sqrt{n_{l-1}}}\min\left(1, \sqrt{\frac{n_l}{n_{l-1}}}\right)\right)$

Learning rates: Set to $\frac{n_l}{n_{l-1}}$ (for Adam $\frac{1}{n_{l-1}}$)

Compared to 'standard' parametrizations – these set

Initialization: Set to $\frac{1}{\sqrt{n_{l-1}}}$

Learning rates: Set to $\Theta(1)$

Differences – LR changes for Adam, also init diffs when fanout n_l < fanin

Implementation in Cerebras GPT

We now have the background to follow cerebrasGPT

	Standard Parameterization (SP)	Maximal Update (μP)
Variables		
W	A multiplicative or fully-connected weights tensor	
b	A bias weights tensor	
X, Y	Activation tensors: layer input, output, respectively	
$d_{\text{model,base}}$	Proxy (base) model's layer width	
d_{model}	Width of each layer	
d_{head}	Size of each attention head	
embed	Combined token + position embedding function	
η_{base}	The base learning rate (LR): Maximum in training schedule	
σ_{base}	The base initialization standard deviation	
m_{width}	—	Layer width multiplier: $d_{\text{model}}/d_{\text{model,base}}$
m_{emb}	—	Emdedding output multiplier

	Standard Parameterization (SP)	Maximal Update (μP)
Empirically Tuned Values		
$d_{\text{model,base}}$	—	256
η_{base}	Must tune for each model size	$6e$-3
σ_{base}	0.02	0.08
m_{emb}	—	10.0
Formulas		
Embedding initializer	$W_{\text{emb}} \sim N_{\text{trunc}}(0, \sigma^2_{\text{base}})$	
Embedding LR	$\eta_{\text{emb}} = \eta_{\text{base}}$	
Embedding output	$Y_{\text{emb}} = \text{embed}(X)$	$Y_{\text{emb}} = m_{\text{emb}} \cdot \text{embed}(X)$
LN initializer	$W_\gamma \sim 1, b_\beta \sim 0$	
LN LR	$\eta_{\text{LN}} = \eta_{\text{base}}$	
Bias initializer	$b \sim 0$	
Bias LR	$\eta_b = \eta_{\text{base}}$	
MHA equation	$\text{softmax}\left(\frac{Q^T K}{\sqrt{d_{\text{head}}}}\right) V$	$\text{softmax}\left(\frac{Q^T K}{d_{\text{head}}}\right) V$
QKV weights initializer	$W_{\text{qkv}} \sim N_{\text{trunc}}(0, \sigma^2_{\text{base}})$	$W_{\text{qkv}} \sim N_{\text{trunc}}(0, \sigma^2_{\text{base}}/m_{\text{width}})$
QKV weights LR	$\eta_{\text{qkv}} = \eta_{\text{base}}$	$\eta_{\text{qkv}} = \eta_{\text{base}}/m_{\text{width}}$
O weights initializer	$W_{\text{o}} \sim N_{\text{trunc}}(0, \frac{\sigma^2_{\text{base}}}{2 \cdot n_{\text{layers}}})$	$W_{\text{o}} \sim N_{\text{trunc}}(0, \frac{\sigma^2_{\text{base}}}{2 m_{\text{width}} \cdot n_{\text{layers}}})$
O weights LR	$\eta_{\text{o}} = \eta_{\text{base}}$	$\eta_{\text{o}} = \eta_{\text{base}}/m_{\text{width}}$
ffn1 weights initializer	$W_{\text{ffn1}} \sim N_{\text{trunc}}(0, \sigma^2_{\text{base}})$	$W_{\text{ffn1}} \sim N_{\text{trunc}}(0, \sigma^2_{\text{base}}/m_{\text{width}})$
ffn1 weights LR	$\eta_{\text{ffn1}} = \eta_{\text{base}}$	$\eta_{\text{ffn1}} = \eta_{\text{base}}/m_{\text{width}}$
ffn2 weights initializer	$W_{\text{ffn2}} \sim N_{\text{trunc}}(0, \frac{\sigma^2_{\text{base}}}{2 \cdot n_{\text{layers}}})$	$W_{\text{ffn2}} \sim N_{\text{trunc}}(0, \frac{\sigma^2_{\text{base}}}{2 m_{\text{width}} \cdot n_{\text{layers}}})$
ffn2 weights LR	$\eta_{\text{ffn2}} = \eta_{\text{base}}$	$\eta_{\text{ffn2}} = \eta_{\text{base}}/m_{\text{width}}$
Output logits multiplier	$Y_{\text{logits}} = W_{unemb} X$	$Y_{\text{logits}} = W_{unemb} X/m_{\text{width}}$

Preprint

A Large-Scale Exploration of μ-Transfer

Lucas Dax Lingle
lucasdaxlingle@gmail.com

Deeper dive into muP

Recall – muP is a scaling procedure for hyperparams (as a function of width)

Param	Init Variance (Θ)	Adam LR (Θ)	Init Variance (Exact)	Adam LR (Exact)
$\mathbf{W}^{E}$	1	1	1	α
$\mathbf{W}^{AQ}$	$1/M$	$1/M$	$1/M$	$\alpha P/M$
$\mathbf{W}^{AK}$	$1/M$	$1/M$	$1/M$	$\alpha P/M$
$\mathbf{W}^{AV}$	$1/M$	$1/M$	$1/M$	$\alpha P/M$
$\mathbf{W}^{AO}$	$1/(HD)$	$1/(HD)$	$1/M$	$\alpha P/M$
$\mathbf{W}^{FI}$	$1/M$	$1/M$	$1/M$	$\alpha P/M$
$\mathbf{W}^{FO}$	$1/F$	$1/F$	$0.25/M$	$\alpha P/M$
$\mathbf{W}^{U}$	$1/M^2$	$1/M$	$1/M^2$	$\alpha P/M$

Embedding

Attention params

Input/output MLP MM

Softmax linear

Table 2: μP scaling rules for transformers; a rule for attention scale is detailed in text below.

In addition, μP uses an attention scale of $\tau^{-1} = \Theta(1/D)$ instead of the usual $\tau^{-1} = 1/\sqrt{D}$. For simplicity, we use $\tau^{-1} = 1/D$, since in preliminary experiments we observed only a small improvement from using smaller multiples of $1/D$. Note that for D fixed across model widths M, any constant $\tau^{-1} \neq 0$ technically complies with μP (Yang et al., 2021) but in the experiments, τ^{-1} will be shown to have a major impact on performance and transfer.

Scaling protocol in the work

Architecture – mostly similar to what's in the class

Important limitation of the work: *only width scaling*

> We use the following default configuration, deviating from it only if specifically mentioned. The depth is fixed at $L = 24$, and we consider model widths $M \in \{128, 512, 2048\}$, yielding three model sizes ranging from 4.7M to 1.2B non-embedding parameters. The head width is fixed at $D = 128$, the number of heads is $H = M/D$, and MLP hidden width is $F = 4M$. The models use RMS LayerNorm without gains (Zhang & Sennrich, 2019), linear projections without biases (Raffel et al., 2020), RoPE on the queries and keys (Su et al., 2021), and ReLU for the MLP nonlinearity (Vaswani et al., 2017; Raffel et al., 2020).

Replicating muP

Q1: Does muP work as claimed? When we scale widths, is optimal LR constant?

Ablation	Width	Base LR					Transfer
		2^{-10}	2^{-8}	2^{-6}	2^{-4}	2^{-2}	
Baseline μP	128	3.846	3.743	**3.695**	3.884	4.143	✔
	512	3.114	2.993	**2.953**	3.221	3.506	
	2048	2.711	2.553	**2.511**	2.563	3.244	
Projection Biases	128	3.838	3.735	**3.705**	3.911	4.269	✔
	512	3.108	2.986	**2.947**	2.970	3.557	
	2048	2.710	2.552	**2.529**	2.672	3.418	

As shown in Table 1, the learning rates transfer reliably across model sizes under μP. Despite each model being 4x wider (and 16x larger) than the last, the smallest model's optimal base learning rate α directly predicts the optimum in our sweeps for the larger models.

What is muP robust to?

Modern LMs have many components that deviate from muP's theory

- Activations – SwiGLU and squared relu
- Batch sizes – Large / small
- Initialization variations – zero attention, etc.
- RMS norm gains
- Exotic optimizers (Lion)
- Regularizers

Which of these (if any) break muP?

What is muP robust to? Nonlinearities

SwiGLU, Squared ReLU have the same optimal LR (and both provide minor gains)

Ablation	Width	Base LR					Transfer
		2^{-10}	2^{-8}	2^{-6}	2^{-4}	2^{-2}	
Baseline μP	128	3.846	3.743	**3.695**	3.884	4.143	✔
	512	3.114	2.993	**2.953**	3.221	3.506	
	2048	2.711	2.553	**2.511**	2.563	3.244	
SwiGLU Nonlinearity	128	3.800	3.740	**3.715**	4.090	7.024	✔
	512	3.070	2.975	**2.953**	3.175	6.863	
	2048	2.677	2.536	**2.505**	2.553	4.571	
Squared ReLU Nonlinearity	128	3.808	3.735	**3.686**	3.999	4.484	✔
	512	3.071	2.964	**2.929**	3.184	7.299	
	2048	2.666	2.516	**2.482**	2.532	3.259	

What is muP robust to? Batch size

Larger and smaller batches.

The original derivation doesn't handle batch size considerations.

Ablation	Width	Base LR					Transfer
		2^{-10}	2^{-8}	2^{-6}	2^{-4}	2^{-2}	
Baseline μP	128	3.846	3.743	**3.695**	3.884	4.143	✔
	512	3.114	2.993	**2.953**	3.221	3.506	
	2048	2.711	2.553	**2.511**	2.563	3.244	
4x Larger Batch	128	3.844	3.735	**3.697**	3.716	10.380	✔
	512	3.141	2.990	**2.965**	3.305	10.373	
	2048	2.745	2.556	**2.541**	2.697	7.197	
4x Smaller Batch	128	3.855	3.774	**3.736**	3.945	4.104	✔
	512	3.120	3.011	**2.977**	3.024	3.521	
	2048	2.714	2.568	**2.527**	2.549	3.223	

What is muP robust to? initialization

There are new initializations that are sometimes used

- SP Unembedding – This is the pre-softmax linear layer. 1/M (SP) vs 1/M^2 (muP)
- Zero Query – Set the query matrix to zero (so that all items get uniform attention)

SP Unembedding Init	128	3.861	3.765	**3.699**	3.896	4.161	✔
	512	3.119	2.990	**2.951**	3.265	3.582	
	2048	2.716	2.554	**2.509**	2.564	7.471	
Zero Query Init	128	3.836	3.743	**3.694**	3.877	4.167	✔
	512	3.115	2.992	**2.949**	3.135	3.532	
	2048	2.711	2.553	**2.510**	2.551	3.272	

What is muP not robust to? RMSnorm gain

In our arch – RMSNorm has learnable gains. This turns out to break muP

Ablation	Width	Base LR					Transfer
		2^{-10}	2^{-8}	2^{-6}	2^{-4}	2^{-2}	
Baseline μP	128	3.846	3.743	**3.695**	3.884	4.143	✔
	512	3.114	2.993	**2.953**	3.221	3.506	
	2048	2.711	2.553	**2.511**	2.563	3.244	
RMSNorm Gains (Vector)	128	3.842	3.744	3.689	**3.670**	3.681	✗
	512	3.101	2.992	2.951	**2.950**	3.412	
	2048	2.692	**2.553**	2.609	2.605	3.169	
RMSNorm Gains (Scalar)	128	3.843	3.749	3.692	**3.670**	4.471	✗
	512	3.106	3.000	2.961	**2.959**	3.515	
	2048	2.704	2.570	**2.525**	2.542	3.334	

As shown in Table 1, optimal learning rates for these models *do not* reliably transfer when using $\Theta(1)$ learning rate scaling for the gains, despite the fact that the 'coordinate size' of the features before and after RMS Normalization is $\Theta(1)$ w.r.t. width by design. In addition to the lack of transfer in these experiments, we find trainable gains harm the quality of the largest μP models when the base learning rate α is optimal. In Section 4.5, we also find that standard transformers with trainable gains underperform μP transformers without them.

But these gains can be removed with little loss of perf..

What is muP not robust to? Exotic optimizers

There are other, exotic optimizers based on just gradient signs. Do they transfer?

Algorithm 1 AdamW Optimizer

given $\beta_1, \beta_2, \epsilon, \lambda, \eta, f$
initialize $\theta_0, m_0 \leftarrow 0, v_0 \leftarrow 0, t \leftarrow 0$
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
  update EMA of g_t **and** g_t^2
  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  bias correction
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$
  update model parameters
  $\theta_t \leftarrow \theta_{t-1} - \eta_t (\hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon) + \lambda \theta_{t-1})$
end while
return θ_t

Algorithm 2 Lion Optimizer (ours)

given $\beta_1, \beta_2, \lambda, \eta, f$
initialize $\theta_0, m_0 \leftarrow 0$
while θ_t not converged **do**
  $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
  update model parameters
  $c_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  $\theta_t \leftarrow \theta_{t-1} - \eta_t (\text{sign}(c_t) + \lambda \theta_{t-1})$
  update EMA of g_t
  $m_t \leftarrow \beta_2 m_{t-1} + (1 - \beta_2) g_t$
end while
return θ_t

Lion Optimizer	128	**3.708**	3.736	4.057	4.344	10.380	✗
	512	2.952	**2.947**	3.416	3.961	10.285	
	2048	2.519	**2.511**	3.151	10.377	10.377	

What is muP not robust to? – (strong) weight decay

What about strong (0.1) weight decay? – this is maybe the only significant muP failure

Decoupled Weight Decay	128	3.760	**3.679**	3.694	3.741	4.011	✗
	512	3.057	2.963	**2.957**	3.139	3.373	
	2048	2.686	2.535	**2.502**	3.123	6.594	

Algorithm 1 SGD with L_2 regularization and SGD with decoupled weight decay (SGDW), both with momentum

given initial learning rate $\alpha \in \mathbb{R}$, momentum factor $\beta_1 \in \mathbb{R}$, weight decay/L_2 regularization factor $\lambda \in \mathbb{R}$

initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$

repeat

$t \leftarrow t + 1$

$\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient

$\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$

$\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, be used for warm restarts

$\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + \eta_t \alpha \boldsymbol{g}_t$

$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \boldsymbol{m}_t - \eta_t \lambda \boldsymbol{\theta}_{t-1}$

until *stopping criterion is met*

return optimized parameters $\boldsymbol{\theta}_t$

Is muP useful? At least to some extent..

Overall, muP generally seems useful – insofar that SP is quite a bit more unstable.

Width	LR				
	2^{-10}	2^{-8}	2^{-6}	2^{-4}	2^{-2}
128	3.841	3.757	**3.706**	3.879	4.030
512	3.013	**2.967**	2.987	3.383	7.403
2048	**2.738**	2.902	7.247	7.477	7.314

Table 3: Validation losses for SP models.

Params	Width	Base LR		
		2^{-8}	2^{-6}	2^{-4}
2M	128	3.791	**3.766**	3.814
40M	512	3.016	**2.983**	3.004
600M	2048	2.513	**2.459**	2.466
10B	8192	2.238	**2.167**	2.169

Table 4: Validation losses for our large-scale experiment.

Current evidence suggests that muP parametrization / initialization may be easier to tune.

Recap: scaling in the wild

What are challenges in scaling ‘in practice’

1. Setting model arch hyperparameters (width, etc)
2. Setting optimizer hyperparameters (LR, batch)
3. Compute needed to fit the big chinchilla sweep

Some solutions?

1. Assume stability (or use muP)
2. Search for optimal LR / batch in small scale, either keep fixed or predict scaling
3. Use alternative learning schedules (WSD-like)