Optimal Scaling

Oleg Filatov, Jiangtao Wang, Jan Ebert

Jülich Supercomputing Center

JÜLICH Forschungszentrum

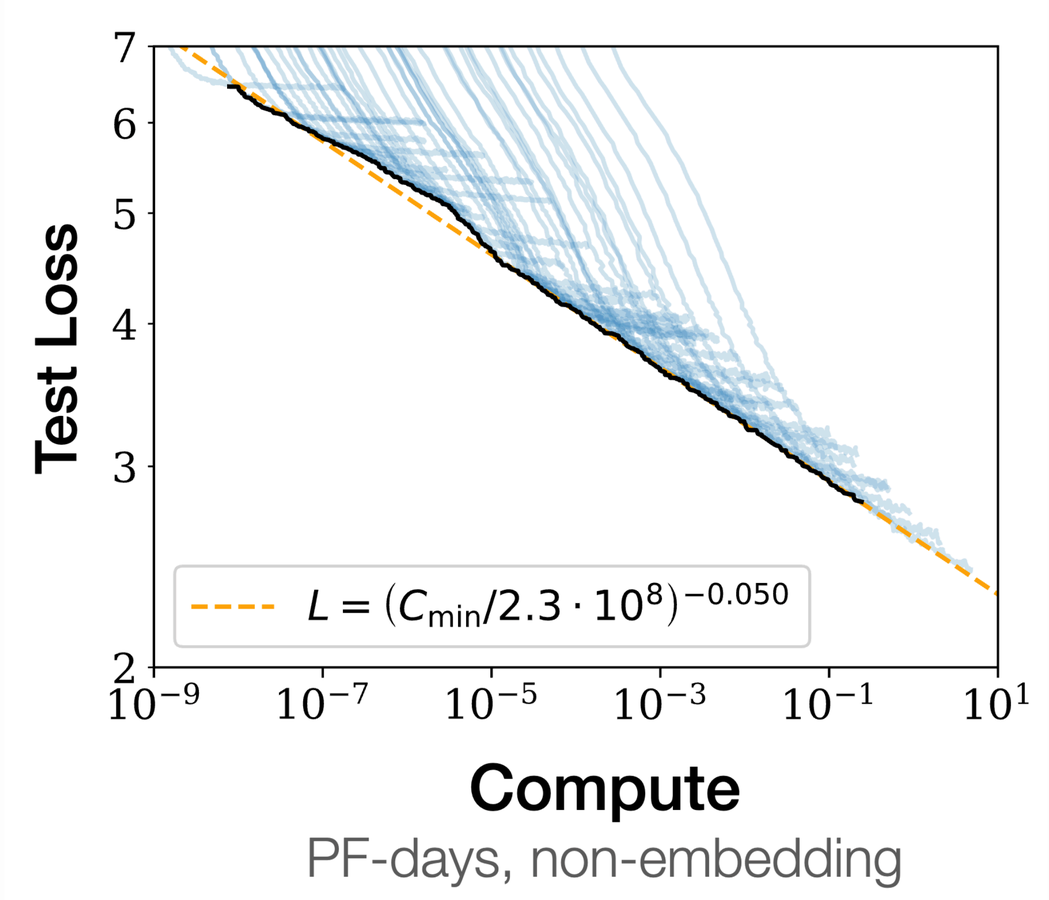
HAICON Prologue Day
02.06.2025

HELMHOLTZ AI

Prologue

Scale is all you need

Training Compute-Optimal Large Language Models



What is scale?

Where you have min ↔ max

- **Model**
  - Width, depth, memory
- **Data**
  - Amount, diversity, quality
- **Context**
  - Sequence length, resolution, receptive field
- **Compute & Time**
  - (GPU/Brain) Throughput, inference speed, utilisation

- **Performance**
  - Evals, evals, evals
- **Adoption**
  - Users, applications
- **Scientific progress**
  - AGI, ASI, cyberpunk
- **Harm & Ethics**
  - Destroy vs. save

Is it needed?

- **Do think about it before doing it**
  - What is important for you?
  - Which scale? Scale up or down?
  - Name → Prioritise → Measure → Optimise
- **It’s a trade-off**
  - You are always constrained
  - No free lunch

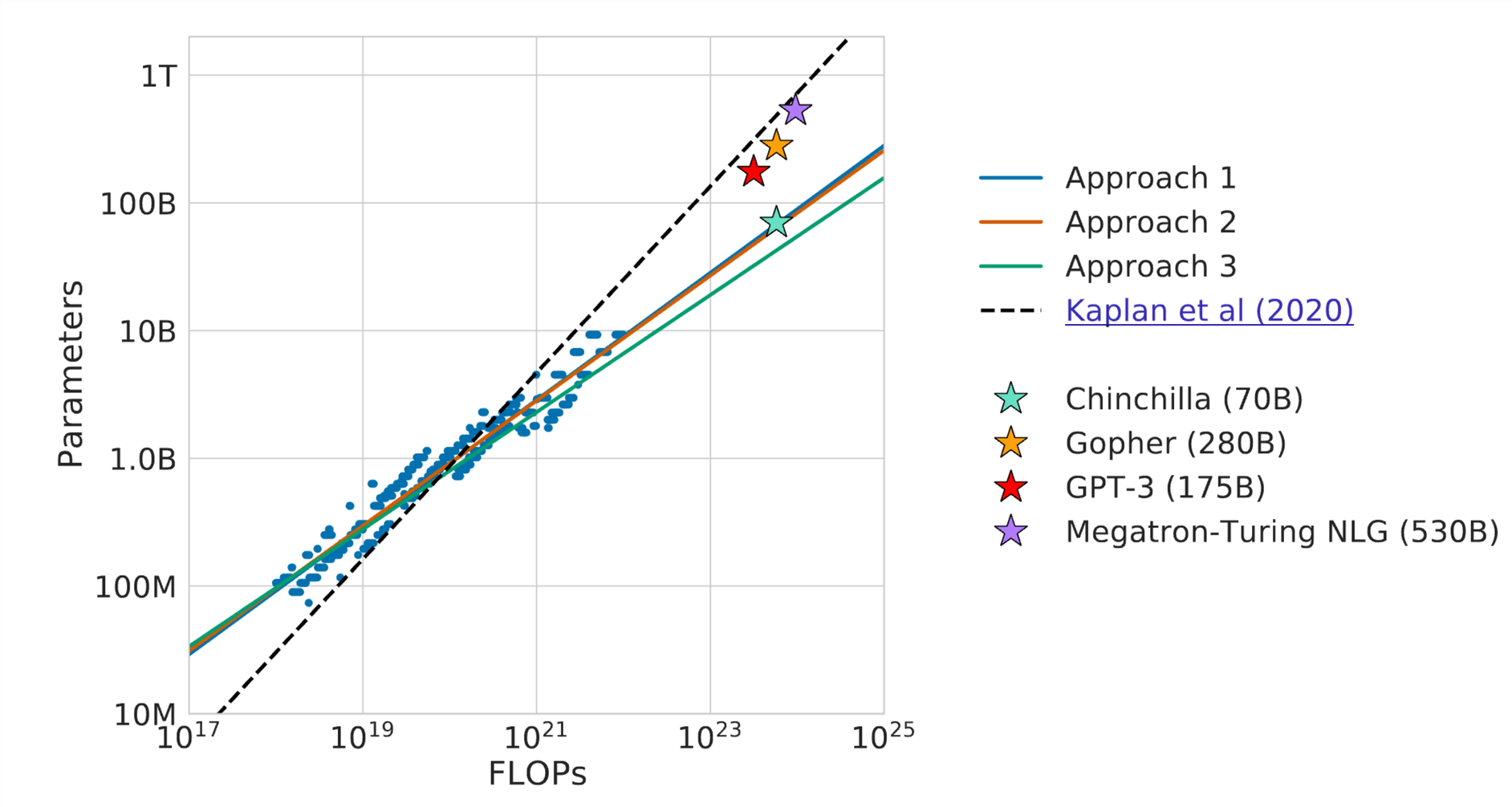
In this lecture

- **Will talk about ☺️**
  - Methods to scale with model & data
  - Scaling laws as guidance
  - *In general:* applicable to ~any domain
- **Won't talk about 😩**
  - Secret sauce (ask your local döner dealer)
  - Long context (maybe next time, lmk)
  - Compute/hardware optimisation (heavy stuff)

Model scaling

Is it optimal?

Training Compute-Optimal Large Language Models



What is optimal?

- **Getting the "best" model**
  - Assuming fixed scales (e.g. compute, data)
  - Model A > Model B > Model C > .... > divergence
  - You define what ">" means (e.g. accuracy on your benchmark)
- **How to get there?**
  - Need to "control" the model family
  - To optimise & vary it
  - Control → Parametrisation → Parameters

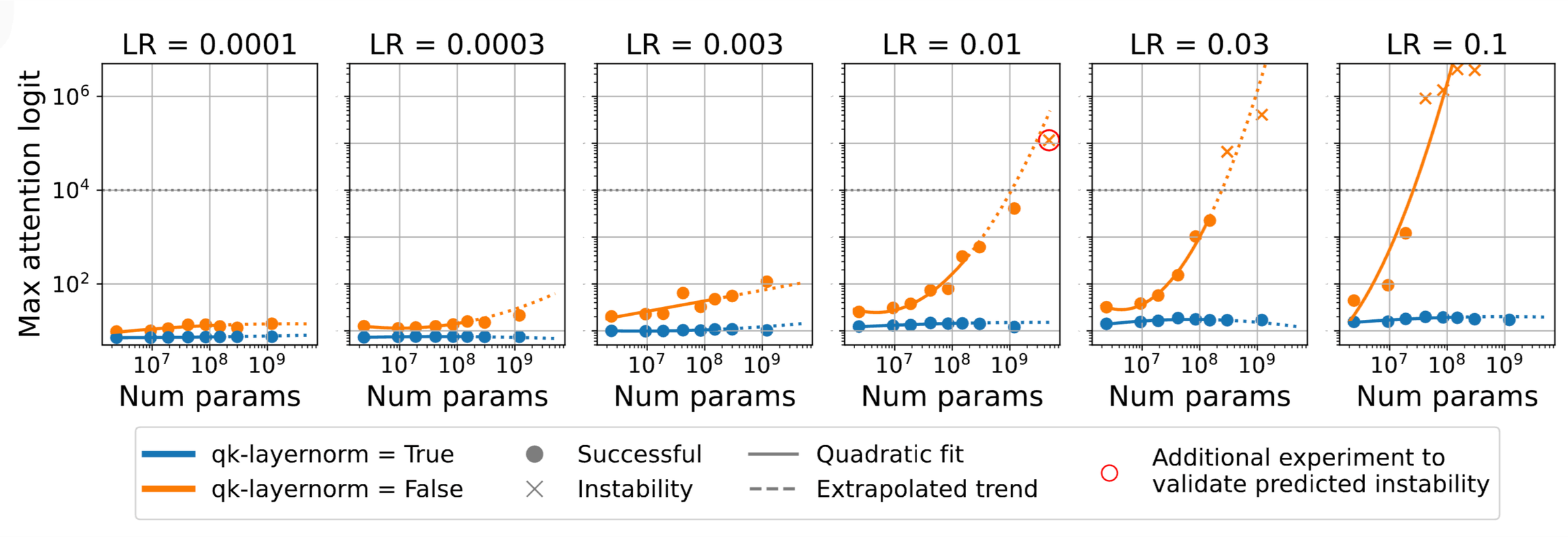
Hyperparameters

😩🤬😤

- **Optimisation**
  - Learning rate, momentum
  - Regularisation
- **Architecture**
  - Weight initialisation
  - Layer choice
- **Scale**
  - Width, depth → model scale
  - Batch size, # iterations → data scale

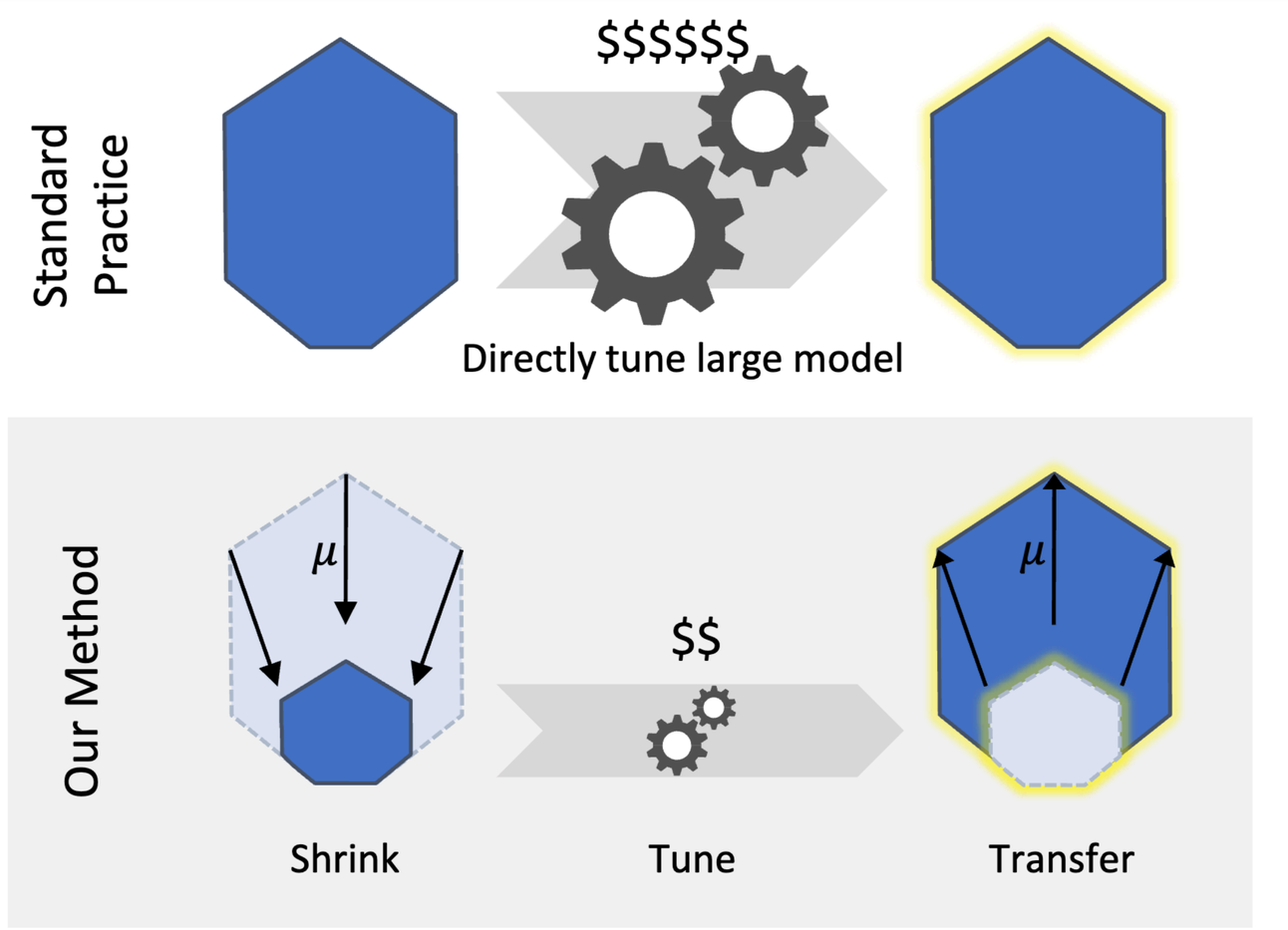
Can mess it up

Small-scale proxies for large-scale Transformer training instabilities



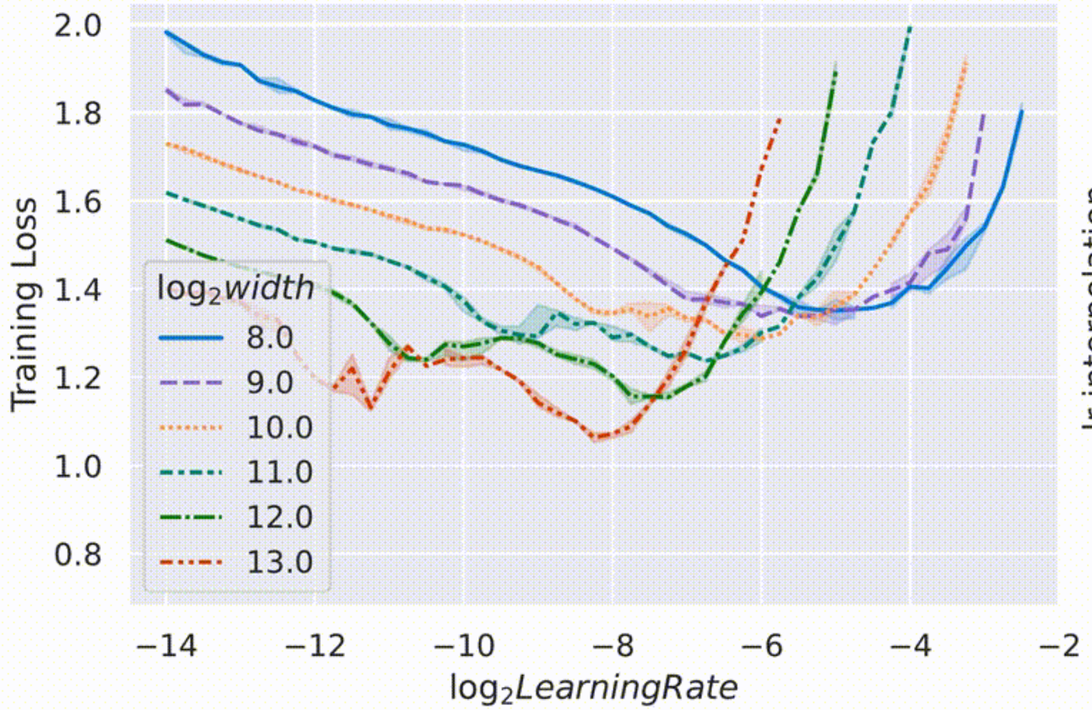
Tuning is expensive

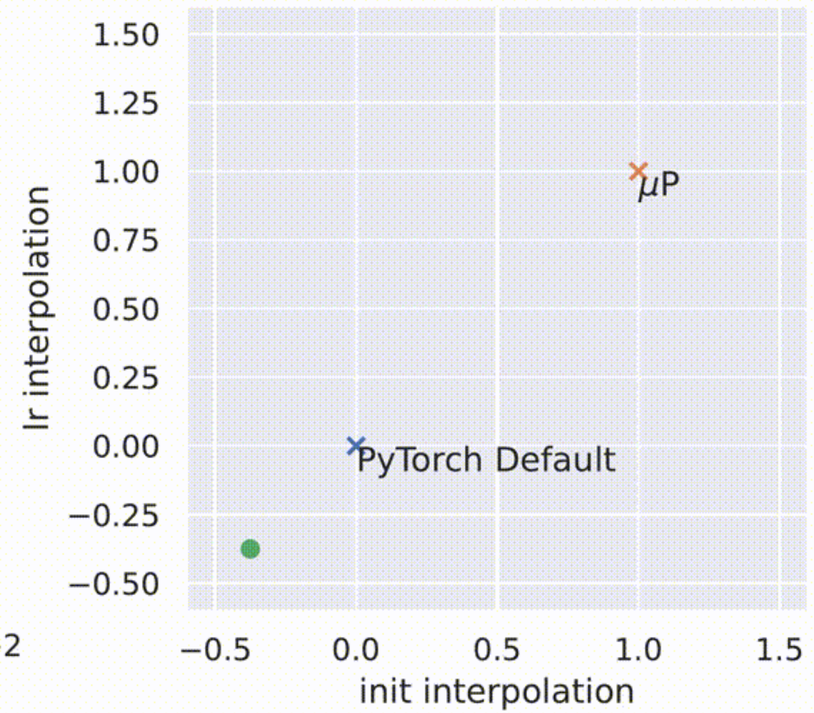
Tensor Programs V



μP

Edward Hu's blog





μTransferable	Not μTransferable	μTransferred *Across*
optimization related, init, parameter multipliers, etc	regularization (dropout, weight decay, etc)	width, depth*, batch size*, training time*, seq length*

This is actually **not** entirely true

μP = Maximal Update Parametrisation

μP

The parameterization for each layer l is specified by three values $\{a_l, b_l, c_l\}$, where:

- the parameter multiplier is n^{-a_l},
- the parameter initialization is $W_l \sim \mathcal{N}(0, n^{-2b_l})$, and
- the learning rate $\eta_l \propto n^{-c_l}$ with width-independent constant of proportionality that we omit here.

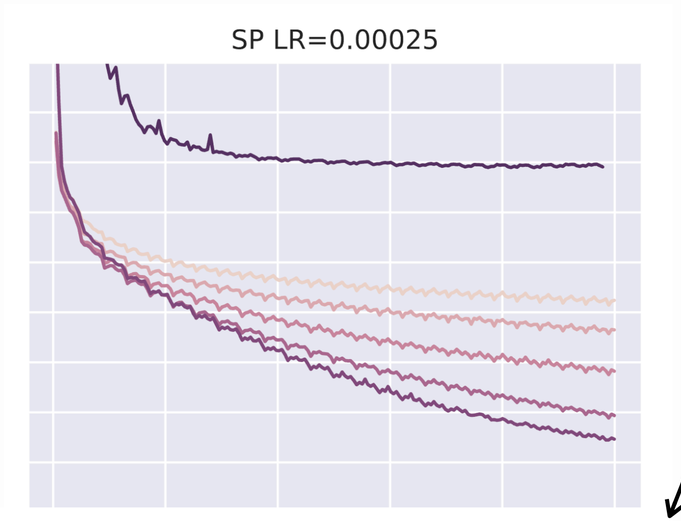
- $W_1 \in \mathbb{R}^{n \times d}$ for the embedding layer
- $W_2, \ldots W_L \in \mathbb{R}^{n \times n}$ for the hidden layers
- $W_{L+1} \in \mathbb{R}^{d \times n}$ for the readout layer.

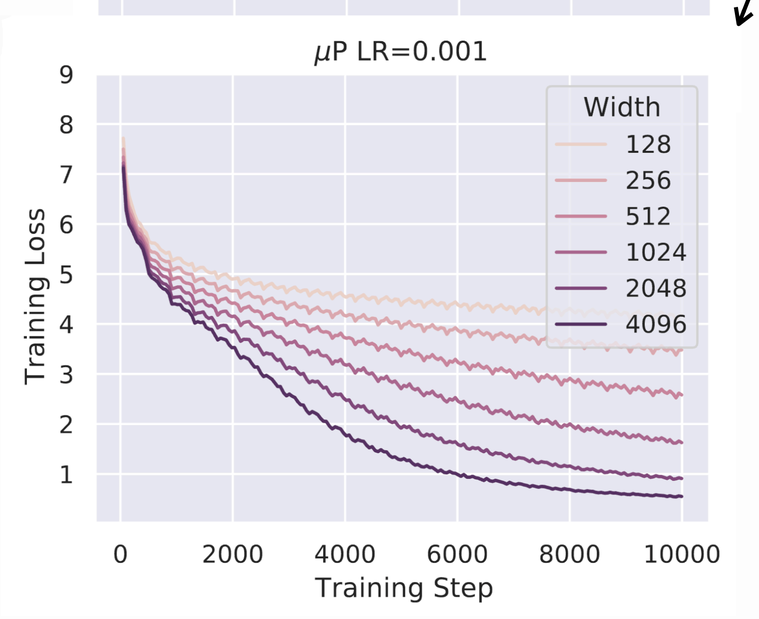
	Input weights & all biases	Output weights	Hidden weights
Init. Var.	$1/\text{fan\_in}$	$1/\text{fan\_in}^2$ $(1/\text{fan\_in})$	$1/\text{fan\_in}$
SGD LR	fan_out (1)	$1/\text{fan\_in}$ (1)	1
Adam LR	1	$1/\text{fan\_in}$ (1)	$1/\text{fan\_in}$ (1)

Note: they also provide scaling rules for attention & conv layers

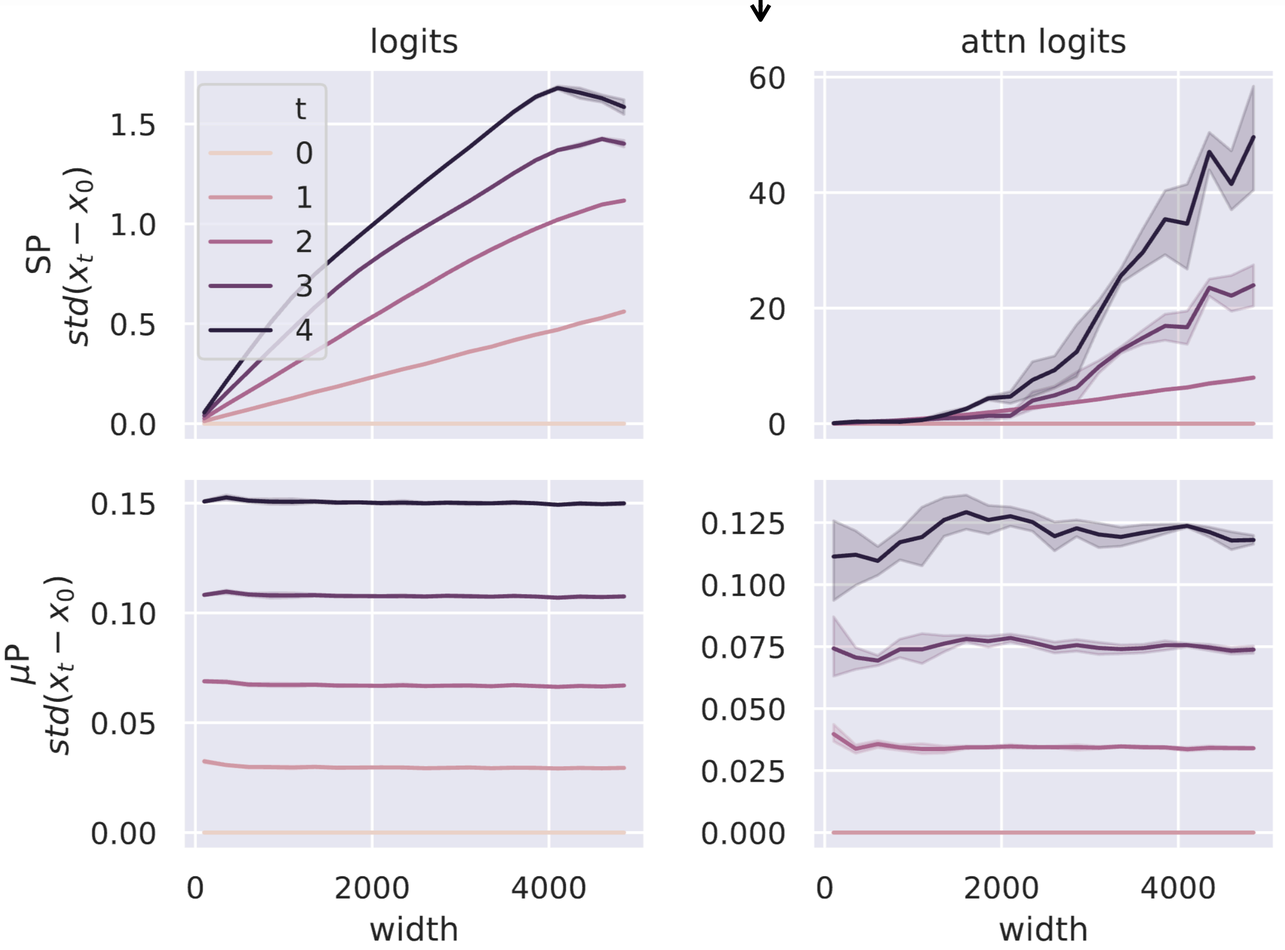
μP

Wider is always better in training loss





Coordinate check stable



μP

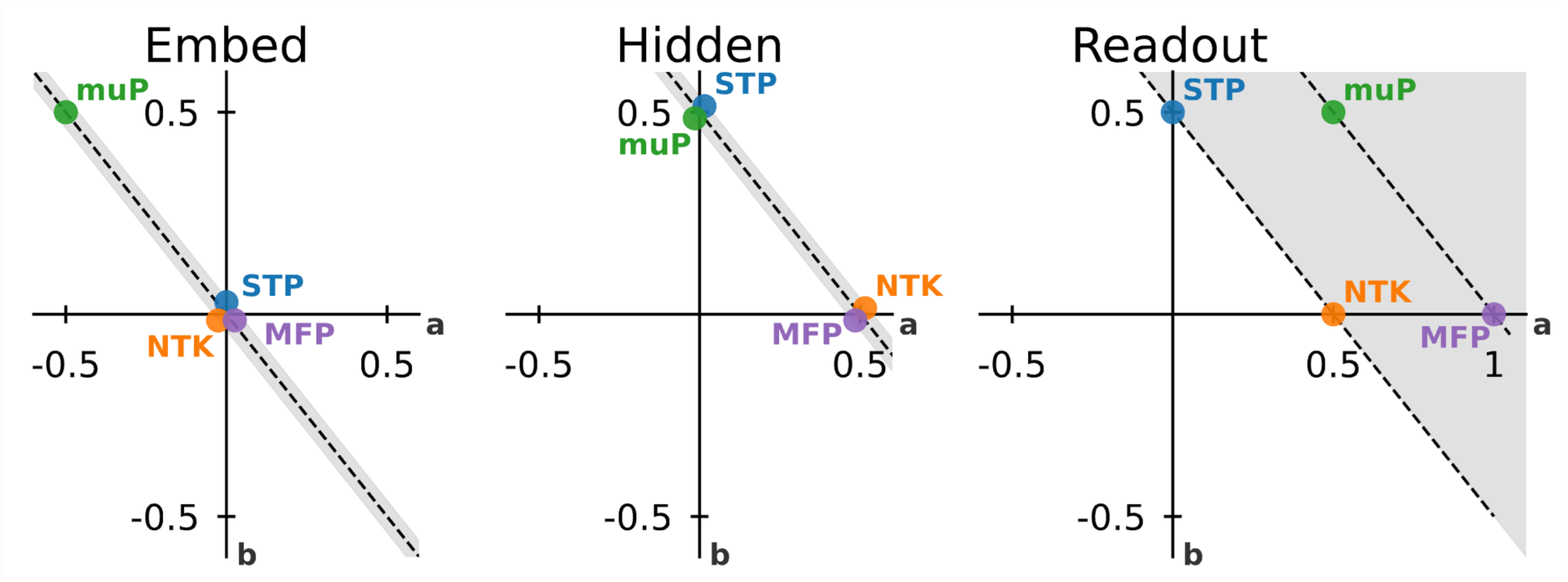
- **Part of larger Tensor Programs series**
- **Depth Scaling:**
  - TP VI, CompleteP, Bordelon et al.
- **Code, tutorials:**
  - EleutherAI tutorial
  - Original repo, Nanotron, nanogpt, NeMo
- **Examples:**
  - Video generation, diffusion transformers, SSM (with modification)
  - OpenAI, xAI, Llama 4, Apple, Cerebras, DeepMind, IBM, Cohere, AlephAlpha, Falcon family, Databricks ...

But likely you won't need this by the end of the lecture 🙃

Parametrisations

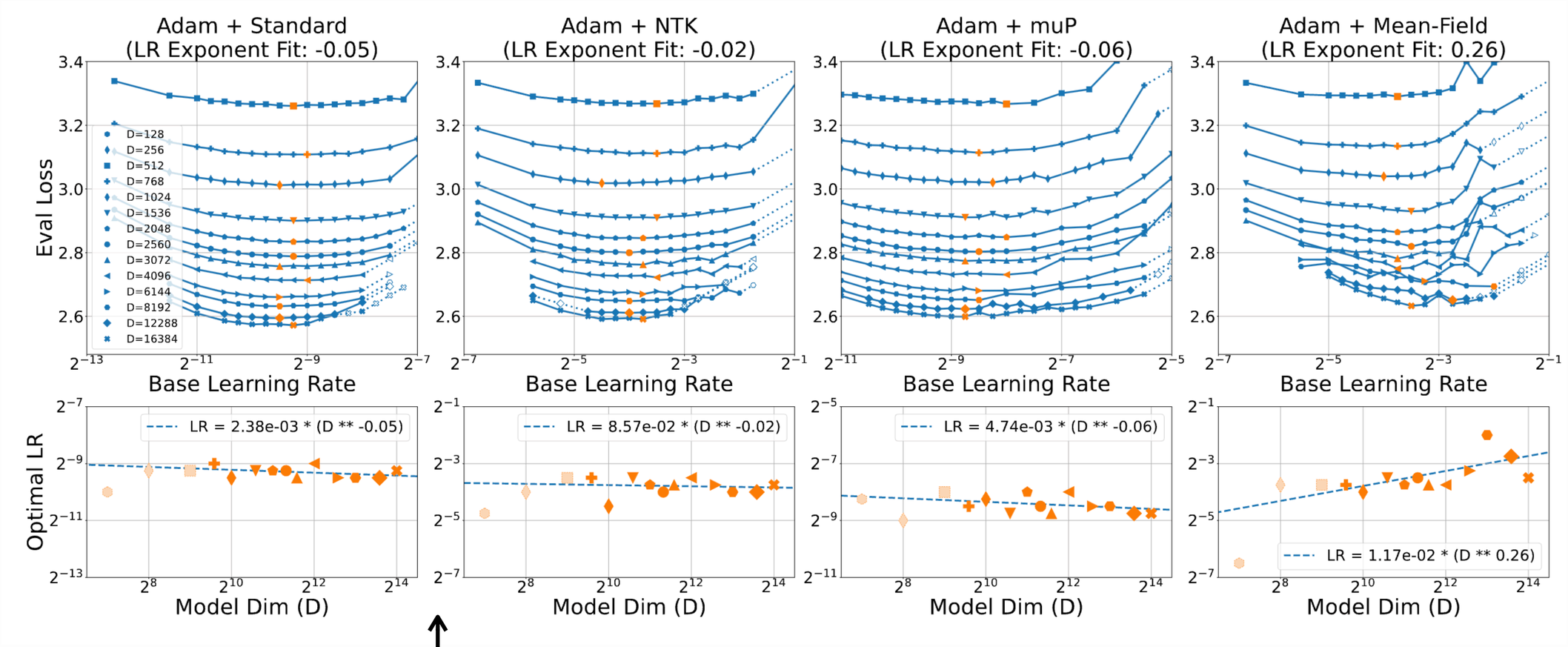
Scaling Exponents Across Parameterizations and Optimizers

Oh yes there's more



μP not all you need

Scaling Exponents Across Parameterizations and Optimizers



"Alignment-corrected" parametrisations ⟶

Definition 3.1. We define the *log alignment ratio* as

$$A_l^t = \log_{\text{fan-in}} \frac{\left\|W_l^t z_{l-1}^t\right\|_{RMS}}{\left\|W_l^t\right\|_{RMS}\left\|z_{l-1}^t\right\|_{RMS}} \in \mathbb{R},$$

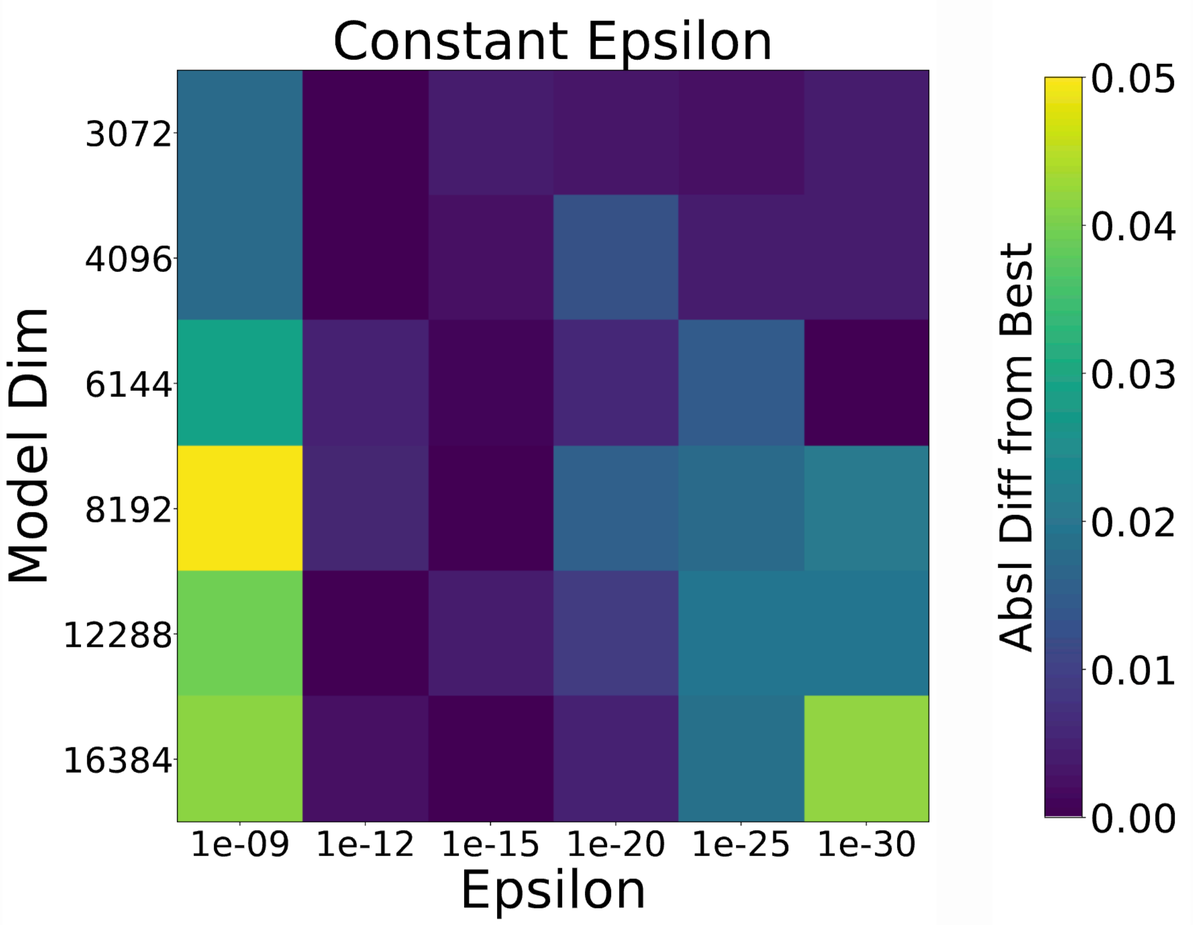
Sidenote

Scaling Exponents Across Parameterizations and Optimizers

Tune your ϵ

$$\theta_t \leftarrow \theta_t - \gamma \widehat{m_t} / \left(\sqrt{\widehat{v_t}} + \epsilon \right)$$

- **Like really**
- **No kidding, at large scale it matters**
- **Heard it messed up 2 LLM trainings**



Spectral condition

A Spectral Condition for Feature Learning

Is all you need

Definition 1 (Spectral norm)**.** The spectral norm of a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ is given by

$$\|\boldsymbol{A}\|_* := \max_{\boldsymbol{v} \in \mathbb{R}^n \setminus \{\boldsymbol{0}\}} \frac{\|\boldsymbol{A}\boldsymbol{v}\|_2}{\|\boldsymbol{v}\|_2}.$$

Condition 1 (Spectral scaling)**.** Consider applying a gradient update $\Delta \boldsymbol{W}_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ to the ℓth weight matrix $\boldsymbol{W}_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$. The spectral norms of these matrices should satisfy:

$$\|\boldsymbol{W}_\ell\|_* = \Theta\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right) \quad \text{and} \quad \|\Delta \boldsymbol{W}_\ell\|_* = \Theta\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right), \quad \text{at layers } \ell = 1, ..., L.$$

$f(n) = \Theta(g(n))$ means that $f(n)$ “scales like” or “is order” $g(n)$

Spectral condition

Is all you need

A Spectral Condition for Feature Learning

5.1 Comparison with "maximal update parametrization"

Maximal update parametrization (μP) was recently proposed as a scaling rule that retains feature learning even at infinite width. μP as given in Table 3 of Yang et al. (2021) may be recovered from our Parametrization 1 by setting $n_0 = n_L = 1$ and $n_1 = n_2 = \ldots = n_{L-1}$. Our Parametrization 1 actually streamlines and generalizes μP: we provide a unifying treatment for any rectangular matrix, rather than treating input, hidden and output layers separately. In other words, from a spectral point of view, no layer is special. Our parametrization also includes scaling with respect to the input dimension n_0 and output dimension n_L (as opposed to neglecting them as agnostic $\Theta(1)$ quantities) and treats hidden widths of unequal dimension ($n_{\ell-1} \neq n_\ell$).

	Input weights & all biases	Output weights	Hidden weights
Init. Var.	$1/\text{fan\_in}$	$1/\text{fan\_in}^2$ $(1/\text{fan\_in})$	$1/\text{fan\_in}$
SGD LR	fan_out (1)	$1/\text{fan\_in}$ (1)	1
Adam LR	1	$1/\text{fan\_in}$ (1)	$1/\text{fan\_in}$ (1)

$$\|\boldsymbol{W}_\ell\|_* = \Theta\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right)$$

$$\|\Delta\boldsymbol{W}_\ell\|_* = \Theta\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right)$$

Remember me?

Remember again, I'll be back

Definition 3.1. We define the *log alignment ratio* as

$$A_l^t = \log_{\text{fan-in}} \frac{\left\| W_l^t z_{l-1}^t \right\|_{RMS}}{\left\| W_l^t \right\|_{RMS} \left\| z_{l-1}^t \right\|_{RMS}} \in \mathbb{R},$$

$$\|\boldsymbol{A}\|_* := \max_{\boldsymbol{v} \in \mathbb{R}^n \setminus \{\mathbf{0}\}} \frac{\|\boldsymbol{A}\boldsymbol{v}\|_2}{\|\boldsymbol{v}\|_2}.$$

Condition for what?

It’s deeper than HP transfer

- **In the infinite width limit, any time, any layer**
- **We want:**
  - **Stability**
    - (Pre)activations neither blown up, nor vanish
  - **Non-triviality**
    - Model function (=output) doesn’t stagnate
  - **Feature learning**
    - Parameters are updated maximally

Natural norm

A Spectral Condition for Feature Learning

Is all you need

$$\|\boldsymbol{W}_\ell\|_* = \Theta\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right) \quad \text{and} \quad \|\Delta\boldsymbol{W}_\ell\|_* = \Theta\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right)$$

↓

Condition 2 (Spectral scaling, natural norms)**.** Consider applying a gradient update $\Delta\boldsymbol{W}_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ to the ℓth weight matrix $\boldsymbol{W}_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$. The spectral norms of these matrices should satisfy:

$$\|\boldsymbol{W}_\ell\|_{\tilde{*}} = \Theta(1) \quad \text{and} \quad \|\Delta\boldsymbol{W}_\ell\|_{\tilde{*}} = \Theta(1), \quad \text{at layers } \ell = 1, \ldots, L.$$

In summary, using these rescaled norms (that we call "natural norms"), our problem is nondimensionalized: feature vectors, feature vector updates, weight matrices, and weight matrix updates are $\Theta(1)$ in norm. These natural norms provide a universal framework that covers specific cases, such as one-hot embeddings in language models, that are not handled directly by Condition 1.

All good stuff is in Appendix, trust me

What is natural?

Modular Duality in Deep Learning

Modular dual norm

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

$$\|\boldsymbol{M}\|_{\alpha \to \beta} = \max_{\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}} \frac{\|\boldsymbol{M}\boldsymbol{x}\|_\beta}{\|\boldsymbol{x}\|_\alpha}. \tag{8}$$

hello

Module	Weight Space $\mathcal{W}$	Module.norm
Linear	$\mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$	$\boldsymbol{W} \mapsto \|\boldsymbol{W}\|_{\text{RMS} \to \text{RMS}}$
Embed	$\mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$	$\boldsymbol{W} \mapsto \|\boldsymbol{W}\|_{\ell_1 \to \text{RMS}}$
Conv2D	$\mathbb{R}^{d_{\text{out}} \times d_{\text{in}} \times k \times k}$	$\boldsymbol{W} \mapsto k^2 \max_{i,j=1}^{k} \|\boldsymbol{W}_{\cdot\cdot ij}\|_{\text{RMS} \to \text{RMS}}$

Not gonna explain modular part but it's really cool

???

$$\|\boldsymbol{W}_\ell\|_{\tilde{*}} = \Theta(1)$$

$$\|\Delta\boldsymbol{W}_\ell\|_{\tilde{*}} = \Theta(1)$$

Dualise 💅

Definition 1 (Dual norm). *Given a norm* $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$*, the dual norm* $\|\cdot\|^\dagger$ *of a vector* $\boldsymbol{g} \in \mathbb{R}^n$ *is given by:*

$$\|\boldsymbol{g}\|^\dagger := \max_{\boldsymbol{t} \in \mathbb{R}^n : \|\boldsymbol{t}\| = 1} \boldsymbol{g}^\top \boldsymbol{t}. \tag{5}$$

Definition 2 (Duality map based on a norm). *Given a norm* $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$*, we consider the duality map:*

$$\text{dualize}_{\|\cdot\|}\, \boldsymbol{g} := \operatorname*{arg\,max}_{\boldsymbol{t} \in \mathbb{R}^n : \|\boldsymbol{t}\| = 1} \boldsymbol{g}^\top \boldsymbol{t}, \tag{6}$$

where, if the arg max *is not unique,* $\text{dualize}_{\|\cdot\|}$ *returns any maximizer.*

Module	Weight Space $\mathcal{W}$	Module.norm	Module.dualize
Linear	$\mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$	$\boldsymbol{W} \mapsto \|\boldsymbol{W}\|_{\text{RMS} \to \text{RMS}}$	$\boldsymbol{G} \mapsto \sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}} \times \boldsymbol{U}\boldsymbol{V}^\top$
Embed	$\mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$	$\boldsymbol{W} \mapsto \|\boldsymbol{W}\|_{\ell_1 \to \text{RMS}}$	$\text{col}_j(\boldsymbol{G}) \mapsto \frac{\text{col}_j(\boldsymbol{G})}{\|\text{col}_j(\boldsymbol{G})\|_{\text{RMS}}}$
Conv2D	$\mathbb{R}^{d_{\text{out}} \times d_{\text{in}} \times k \times k}$	$\boldsymbol{W} \mapsto k^2 \max_{i,j=1}^{k} \|\boldsymbol{W}_{\cdot\cdot ij}\|_{\text{RMS} \to \text{RMS}}$	$\boldsymbol{G}_{\cdot\cdot ij} \mapsto \frac{1}{k^2} \sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}} \times \boldsymbol{U}_{ij}\boldsymbol{V}_{ij}^\top$

Table 1: Duality maps for three atomic modules: Linear, Embed, and Conv2D. These atomic modules are sufficient to build convolutional neural networks and transformers. In Linear.dualize, we let $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ denote the reduced SVD of the gradient matrix $\boldsymbol{G}$. In Conv2D.dualize, we let $\boldsymbol{U}_{ij}\boldsymbol{\Sigma}_{ij}\boldsymbol{V}_{ij}^\top$ denote the reduced SVD of the slice of the gradient tensor $\boldsymbol{G}_{\cdot\cdot ij}$ at kernel indices i and j. Section 5 provides GPU-friendly algorithms for computing these duality maps based on a family of Newton-Schulz iterations.

Dual ~ steepest

Old Optimizer, New Norm: An Anthology

Suppose that we have found a *norm* $\|\cdot\| : \mathcal{W} \to \mathbb{R}$ and a *sharpness parameter* $\lambda > 0$ that serve as a good model of the higher-order terms in the Taylor expansion of the loss function given in Equation (3):

$$\mathcal{L}(\boldsymbol{w} + \Delta\boldsymbol{w}) \approx \mathcal{L}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w})^\top \Delta\boldsymbol{w} + \frac{\lambda}{2} \cdot \|\Delta\boldsymbol{w}\|^2. \tag{4}$$

Proposition 1 (Steepest descent under a norm)**.** *For any* $\boldsymbol{g} \in \mathbb{R}^n$ *thought of as "the gradient", any* $\lambda \geqslant 0$ *thought of as "the sharpness", and any norm* $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ *with dual norm* $\|\cdot\|^\dagger$ *and duality map* $\text{dualize}_{\|\cdot\|}$*:*

$$\arg\min_{\Delta\boldsymbol{w} \in \mathbb{R}^n} \left[\boldsymbol{g}^\top \Delta\boldsymbol{w} + \frac{\lambda}{2} \|\Delta\boldsymbol{w}\|^2 \right] = -\frac{\|\boldsymbol{g}\|^\dagger}{\lambda} \times \text{dualize}_{\|\cdot\|}\, \boldsymbol{g}. \tag{7}$$

Story I. Adam as Steepest Descent under the Max-of-Max Norm

ADAM is a widely used deep learning optimizer: the original paper of Kingma and Ba (2015) now has well over 100,000 citations. Adam has been motivated in various ways, including through convex analysis (Kingma and Ba, 2015) and as an approximate second-order method (Sun and Spall, 2021). However, there have been efforts to

Story II. Shampoo as Steepest Descent under the Spectral Norm

NOW, dear reader, we turn our attention to Shampoo (Gupta et al., 2017, 2018). A variant of the Shampoo optimizer won the external tuning track of the *2024 AlgoPerf: Training Algorithms* competition (Dahl et al., 2023). While the method was originally motivated as a generalization of the AdaGrad convex optimizer (Duchi et al., 2011) to tensor spaces, more recent work casts Shampoo as an approximate second-order

Normalise everything

Aka put RMSNorms everywhere

Scalable Optimization in the Modular Norm

See also nGPT and Small-scale proxies

Writing $\boldsymbol{f} = \mathsf{Linear}(d_{\text{out}}, d_{\text{in}})$.forward, its derivative and second derivative at $(\boldsymbol{W}, \boldsymbol{x})$ are given by:

$$\nabla \boldsymbol{f} \diamond (\Delta \boldsymbol{W}, \Delta \boldsymbol{x}) = \sqrt{d_{\text{out}}/d_{\text{in}}} \left((\Delta \boldsymbol{W})\, \boldsymbol{x} + \boldsymbol{W}\, (\Delta \boldsymbol{x}) \right), \tag{B.1}$$

$$(\Delta \boldsymbol{W}, \Delta \boldsymbol{x}) \diamond \nabla^2 \boldsymbol{f} \diamond (\Delta \widetilde{\boldsymbol{W}}, \Delta \widetilde{\boldsymbol{x}}) = \sqrt{d_{\text{out}}/d_{\text{in}}} \left((\Delta \boldsymbol{W})(\Delta \widetilde{\boldsymbol{x}}) + (\Delta \widetilde{\boldsymbol{W}})(\Delta \boldsymbol{x}) \right). \tag{B.2}$$

from which we conclude that $\mathsf{Linear}(d_{\text{out}}, d_{\text{in}})$ is well-normed, using the RMS norms on its input and output, so long as its arguments satisfy:

$$\|\boldsymbol{W}\|_*, \|\boldsymbol{x}\|_{\mathsf{RMS}} \leq 1. \tag{B.3}$$

These conditions will be automatically satisfied for many neural networks under *orthogonal initialization* of the weights, and especially if a linear module is immediately preceded by something like a LayerNorm module. Moreover, orthogonal initialization guarantees that the well-normed inequality

$$\|\nabla \boldsymbol{f} \diamond \Delta \boldsymbol{x}\|_{\mathsf{RMS}} \leq \|\boldsymbol{x}\|_{\mathsf{RMS}} \tag{B.4}$$

holds tightly in nearly-square matrices at initialization, which is important for getting good signal propagation through the whole network.

1. The optimizer journey

Others ?
- Google
- xAI?
- OpenAI??
- Anthorpic???

Adam
- Llama1/2/3
- Olmo
- Qwen
- Deepseek
- …
- ~Apple

Muon

Keller Jordan's blog

Algorithm 2 Muon

Require: Learning rate η, momentum μ

1: Initialize $B_0 \leftarrow 0$
2: **for** $t = 1, \ldots$ **do**
3: Compute gradient $G_t \leftarrow \nabla_\theta \mathcal{L}_t(\theta_{t-1})$
4: $B_t \leftarrow \mu B_{t-1} + G_t$
5: $O_t \leftarrow \text{NewtonSchulz5}(B_t)$
6: Update parameters $\theta_t \leftarrow \theta_{t-1} - \eta O_t$
7: **end for**
8: **return** θ_t

The function of the NS iteration is to approximately *orthogonalize* the update matrix, *i.e.*, to apply the following operation:

$$\text{Ortho}(G) = \arg\min_O \{\|O - G\|_F : \text{either } O^\top O = I \text{ or } OO^\top = I\}$$

In other words, the NS iteration effectively replaces SGD-momentum's update matrix with the nearest semi-orthogonal matrix to it. This is equivalent to replacing the update by $UV^\top$, where $USV^\top$ is its singular value decomposition (SVD).

Note 1: input/output layers are still learned with AdamW!

Note 2: Low-rank distributed optimised version: Dion

Note 3: See Gluon for deeper optimisation perspective

Scion

Training Deep Learning Models with Norm-Constrained LMOs

Algorithm 1 Unconstrained SCG (uSCG)

Input: Horizon n, initialization $x^1 \in \mathcal{X}$, $d^0 = 0$, momentum $\alpha_k \in (0, 1]$, and stepsize $\gamma_k \in (0, 1)$

1: **for** $k = 1, \ldots, n$ **do**
2: Sample $\xi_k \sim \mathcal{P}$
3: $d^k \leftarrow \alpha_k \nabla f(x^k, \xi_k) + (1 - \alpha_k) d^{k-1}$
4: $x^{k+1} \leftarrow x^k + \gamma_k \operatorname{lmo}(d^k)$
5: Choose $\bar{x}^n$ uniformly at random from $\{x^1, \ldots, x^n\}$

Return $\bar{x}^n$

(i) image domains: Spectral → Spectral → Sign

(ii) 1-hot input: ColNorm → Spectral → Sign

(iii) weight sharing: Sign → Spectral → Sign

$$\|\boldsymbol{W}_\ell\|_{\tilde{*}} = \Theta(1)$$
$$\|\Delta \boldsymbol{W}_\ell\|_{\tilde{*}} = \Theta(1)$$

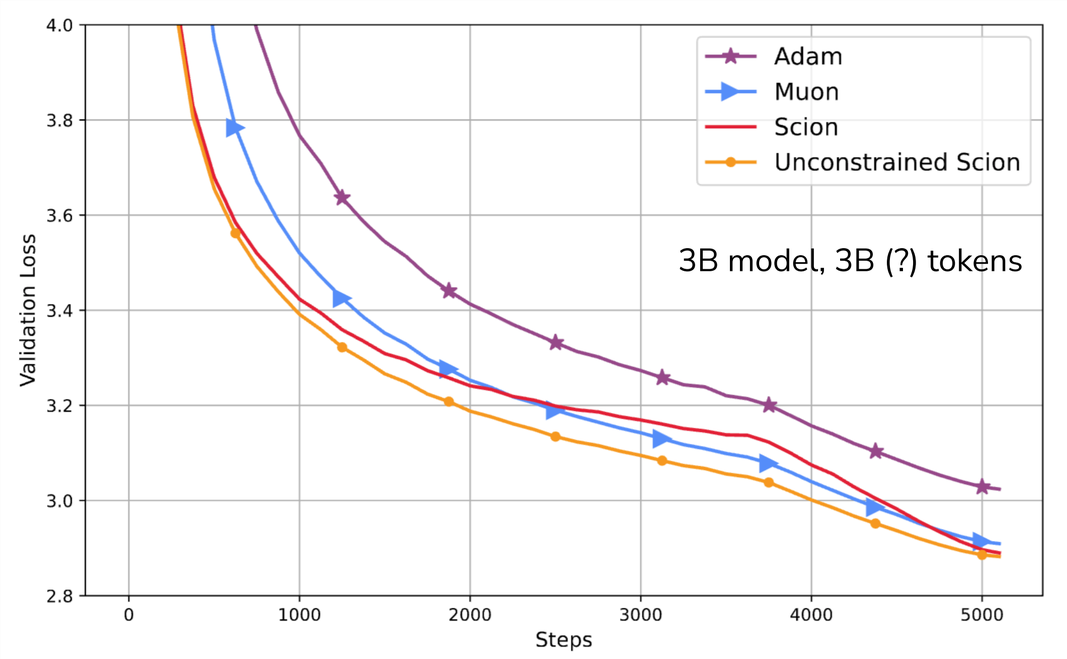
↓

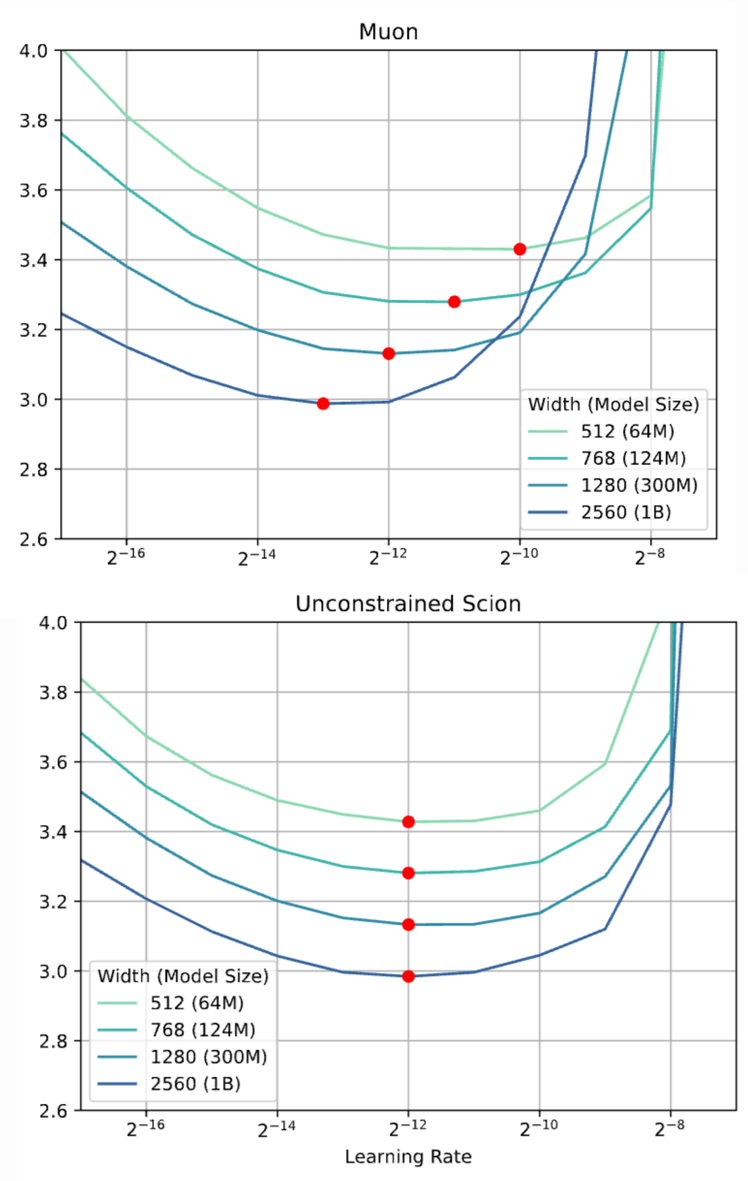
Table 3. The choice of lmo can be different between layers and can depend on the assumptions on the input. For simplicity we overload notation and write the reduced SVD as $W_\ell = U \operatorname{diag}(\sigma) V^\top \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ for all $\ell \in [L]$.

Parameter	W_1 (image domain)	$\{W_\ell\}_{\ell \in [2,\ldots,L-1]}$	W_L			b_ℓ
Norm	RMS → RMS	RMS → RMS	RMS → RMS	RMS → ∞	1 → ∞	RMS
LMO	$\max(1, \sqrt{d_{\text{out}}/d_{\text{in}}}) UV^\top$	$\sqrt{d_{\text{out}}/d_{\text{in}}} UV^\top$	$\sqrt{d_{\text{out}}/d_{\text{in}}} UV^\top$	$\operatorname{row}_j(W_L) \mapsto \frac{1}{\sqrt{d_{\text{in}}}} \frac{\operatorname{row}_j(W_L)}{\|\operatorname{row}_j(W_L)\|_2}$	$\frac{1}{d_{\text{in}}} \operatorname{sign}(W_L)$	$\frac{b_\ell}{\|b_\ell\|_{\text{RMS}}}$
Init.	Semi-orthogonal	Semi-orthogonal	Semi-orthogonal	Row-wise normalized Gaussian	Random sign	0

Scion

Training Deep Learning Models with Norm-Constrained LMOs





- **HP transfer *and* "steepest" learning**
- **Literally SGD w/ momentum**
  - **No warmup, betas, init.std**
  - **Almost stateless optimisation (ScionLight)**
  - **NB: remove grad. clipping**

Summary

- μP 🤔
- Aligned parametrisations 🤯
- Spectral condition 😉
- Modular norm 🤯🤯
- Muon 🤌
- (Un)constrained LMOs / Scion 🤯🤯🤯

Data scaling

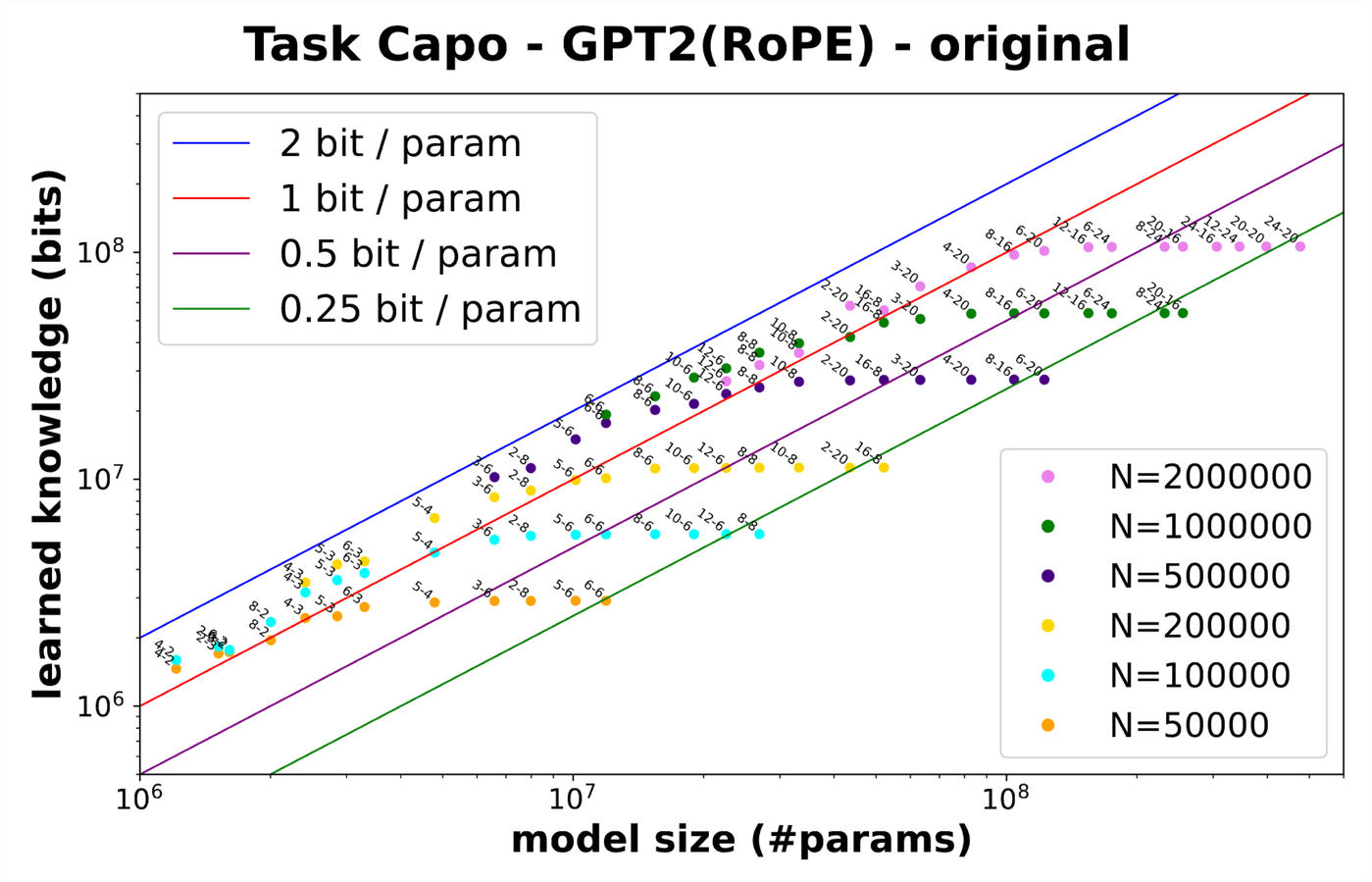
Disclaimer

- **Won’t discuss quality / diversity**
  - Affects amount ↔ performance Pareto
  - Definitely take time to sit with it
- **Will discuss scaling "i.i.d" style:**
  - Same distribution, # data points → ∞
  - No curriculum (also affects a. ↔ perf.)

Why more data?

Memory perspective

Physics of LMs, Part 4.1



Why more data?

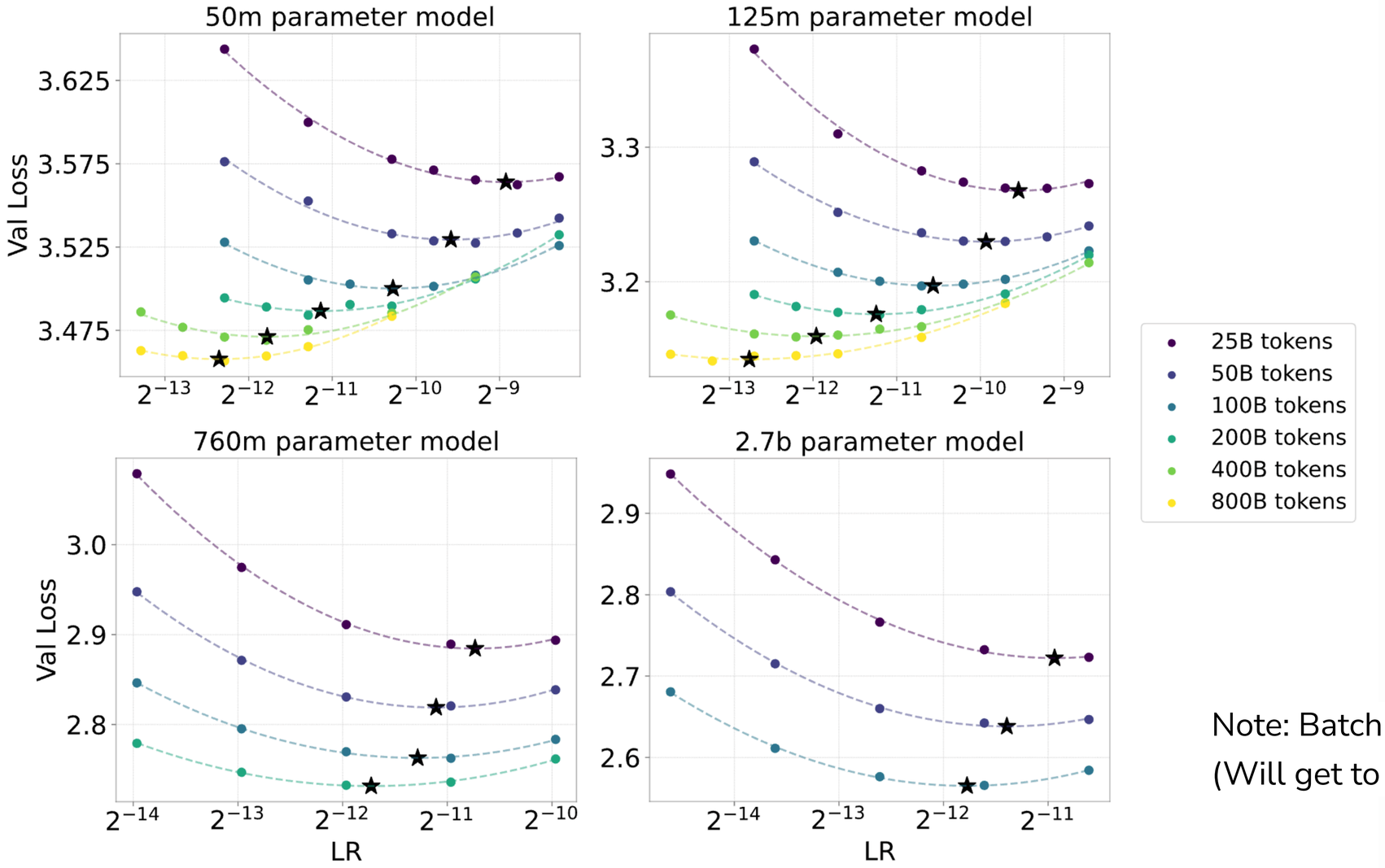
Reasoning perspective

(i guess it helps)

LR optimal?

No 😒

Scaling Optimal LR Across Token Horizons

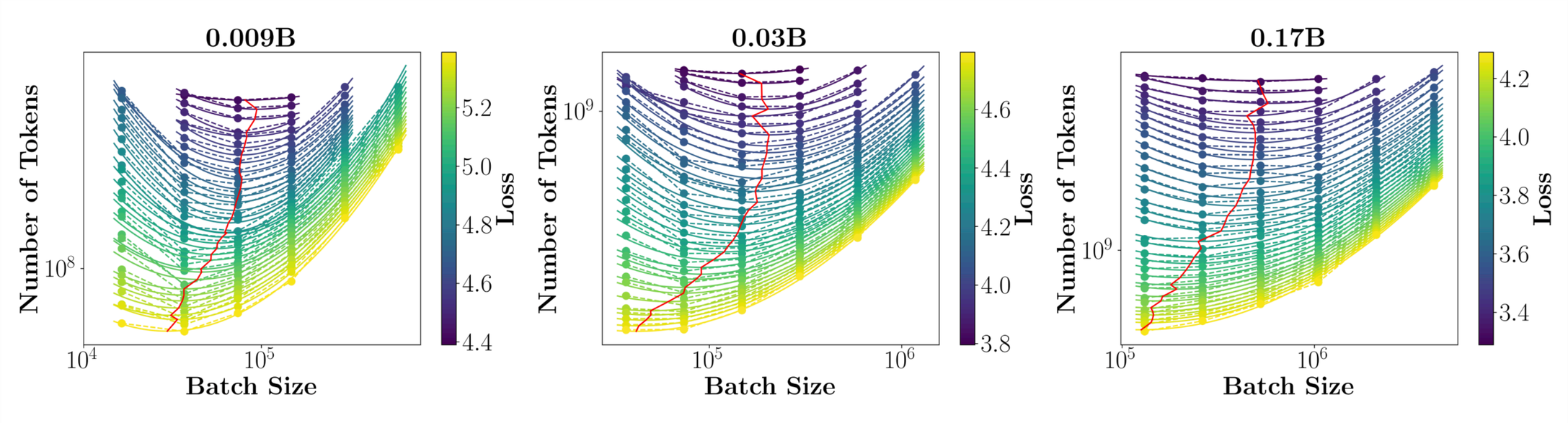


Note: Batch size fixed!
(Will get to that later)

Batch size?

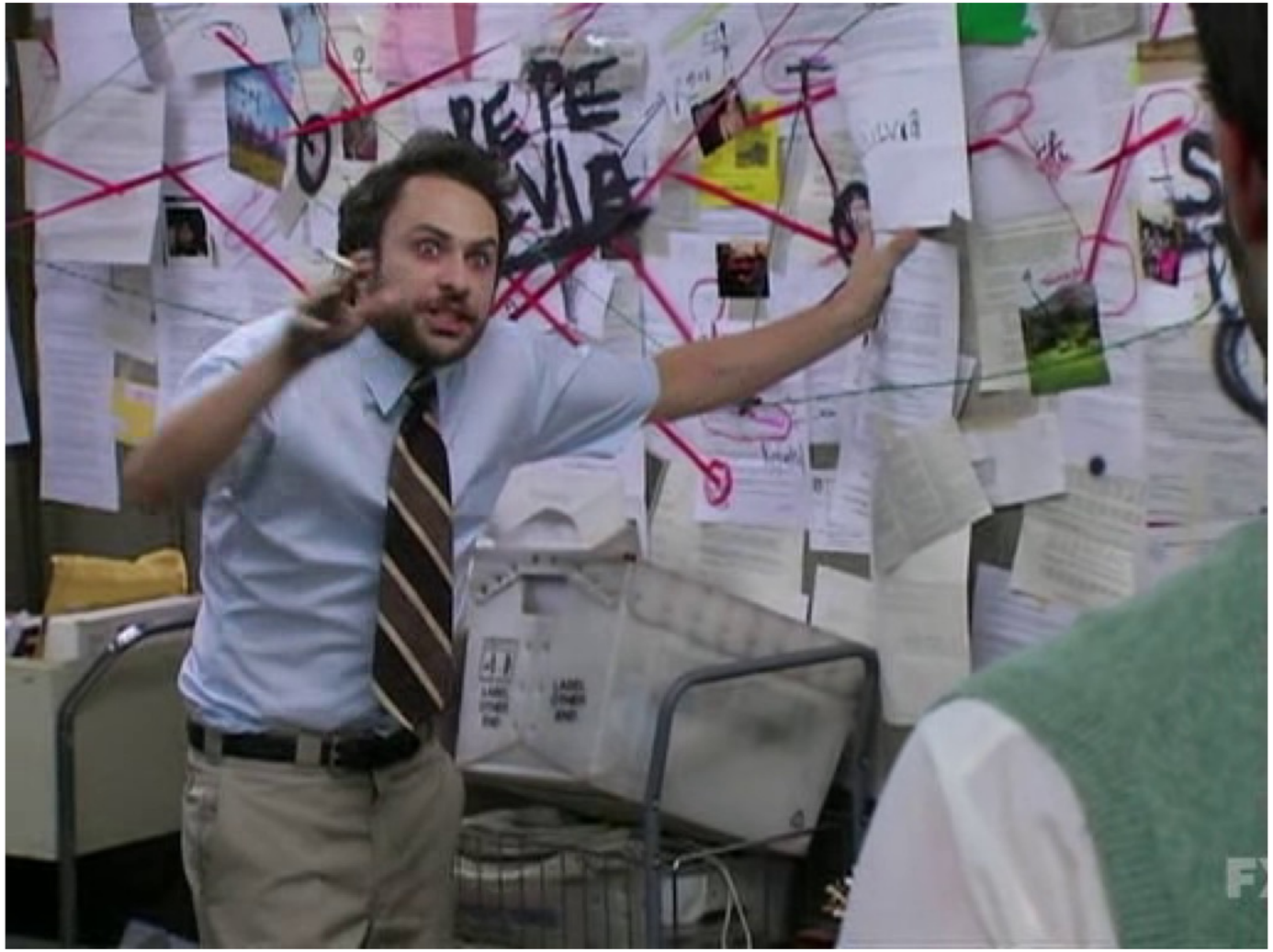
MiniCPM

No 😒



Note: optimiser HPs fixed!

(Will get to that later)



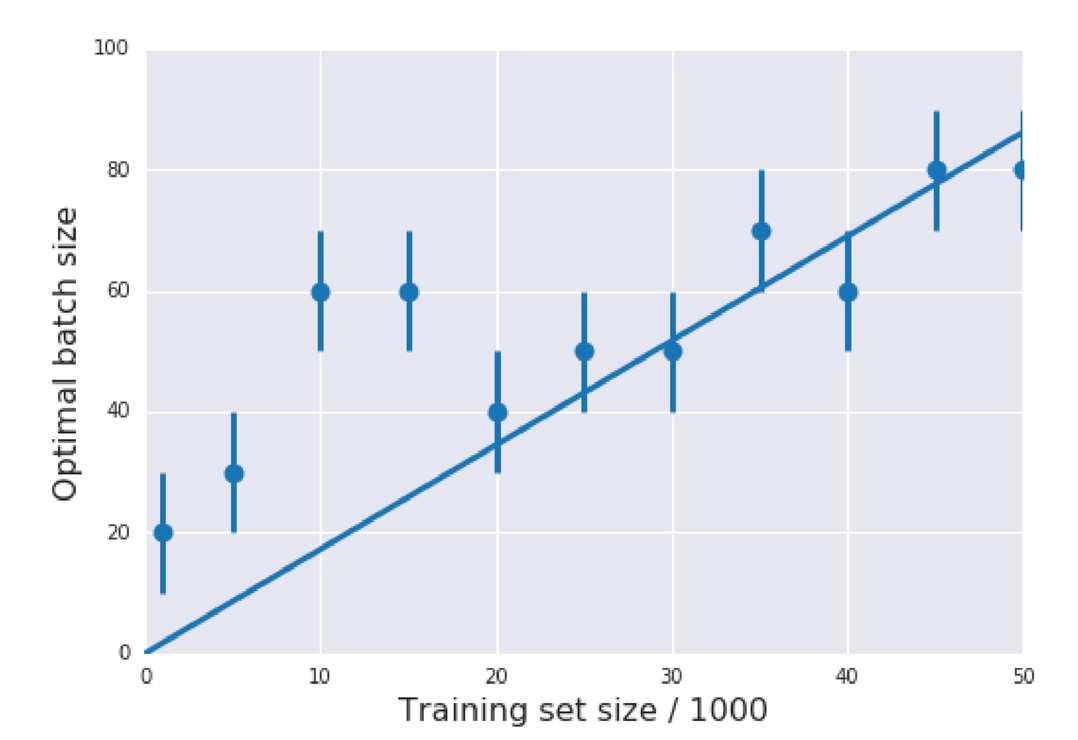
Brace yourself

A Bayesian Perspective on Generalisation and SGD

Noise scale

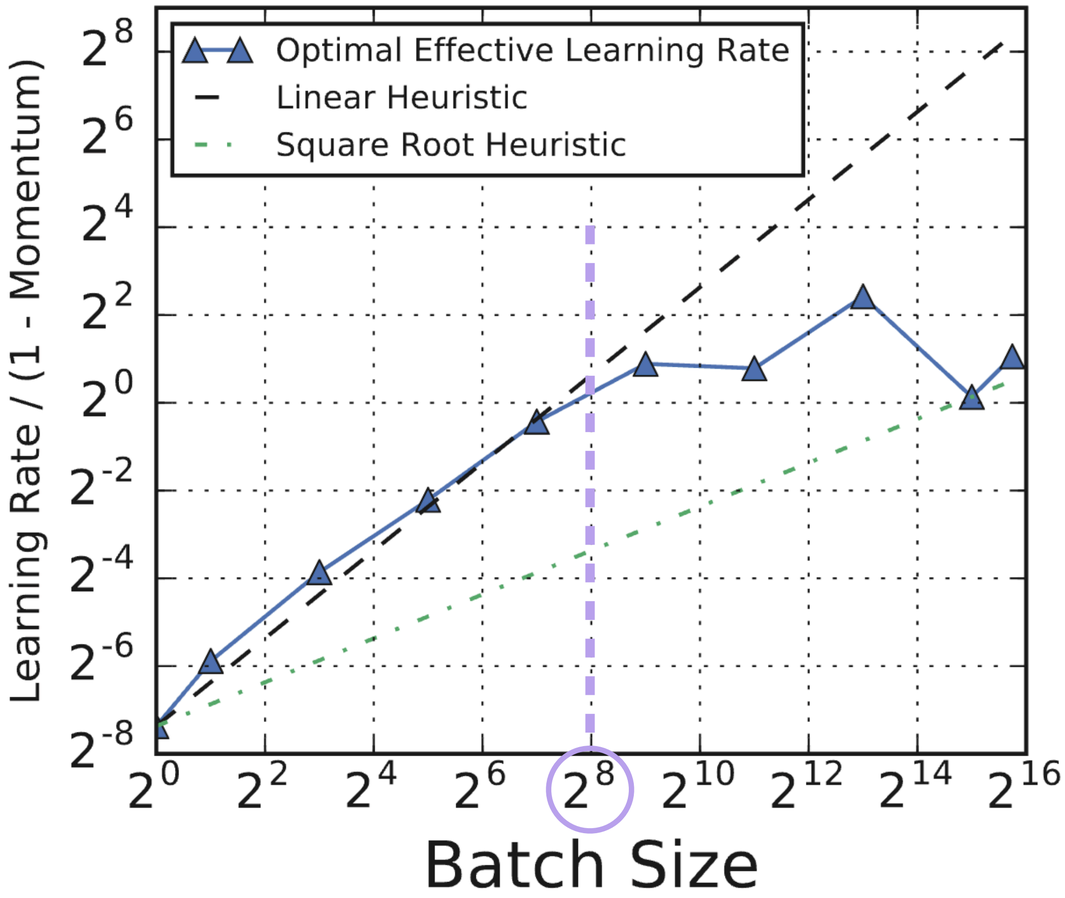
$$B_{noise}^{SDE} \approx \frac{\eta D}{B(1-m)}$$

- **Invariant → scaling rules**
  - e.g. $\eta \propto B, \eta \propto 1/D$
- **Breaks when SDE approx. not valid**
  - continuous time discretisation breaks
- **Still, simple rule of thumb**
  - NB: above for SGD, see here for Adam



Brace yourself

Critical batch size



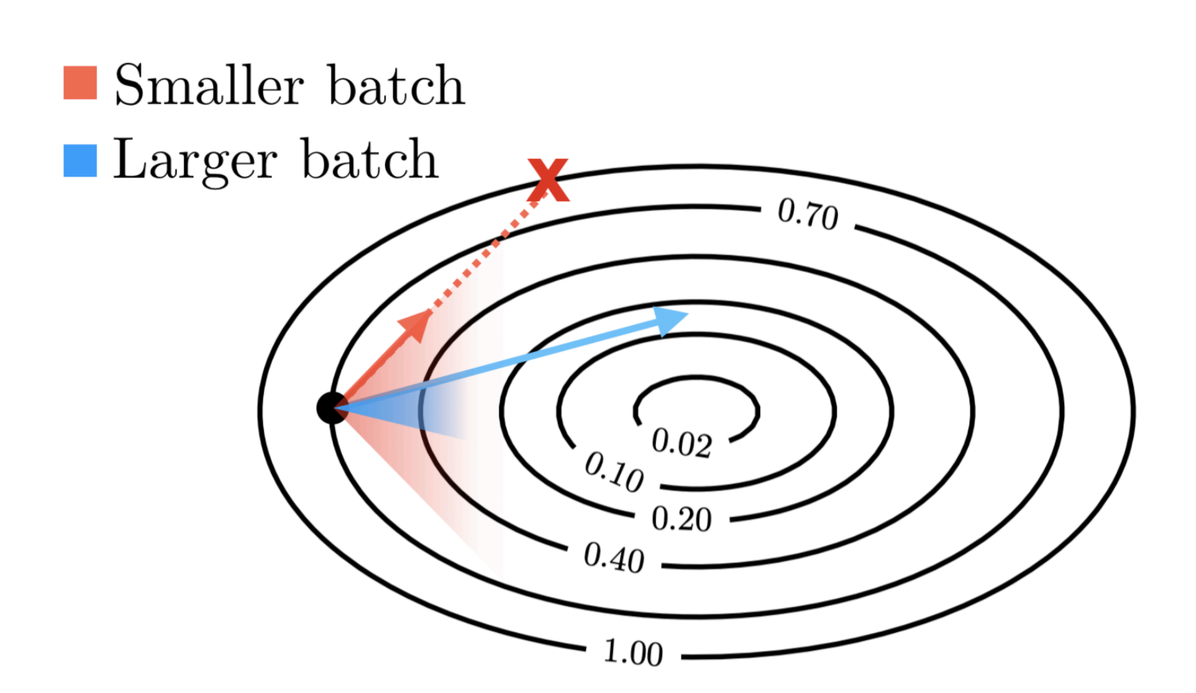
$$\eta^* = \frac{\eta_{crit}}{1 + \frac{B_{crit}}{B}}$$

Critical bsz = linear (sqrt for Adam) scaling rule "breaks"

Why B_{crit}?

An Empirical Model of Large-Batch Training

Gradient becomes sufficiently "noiseless"

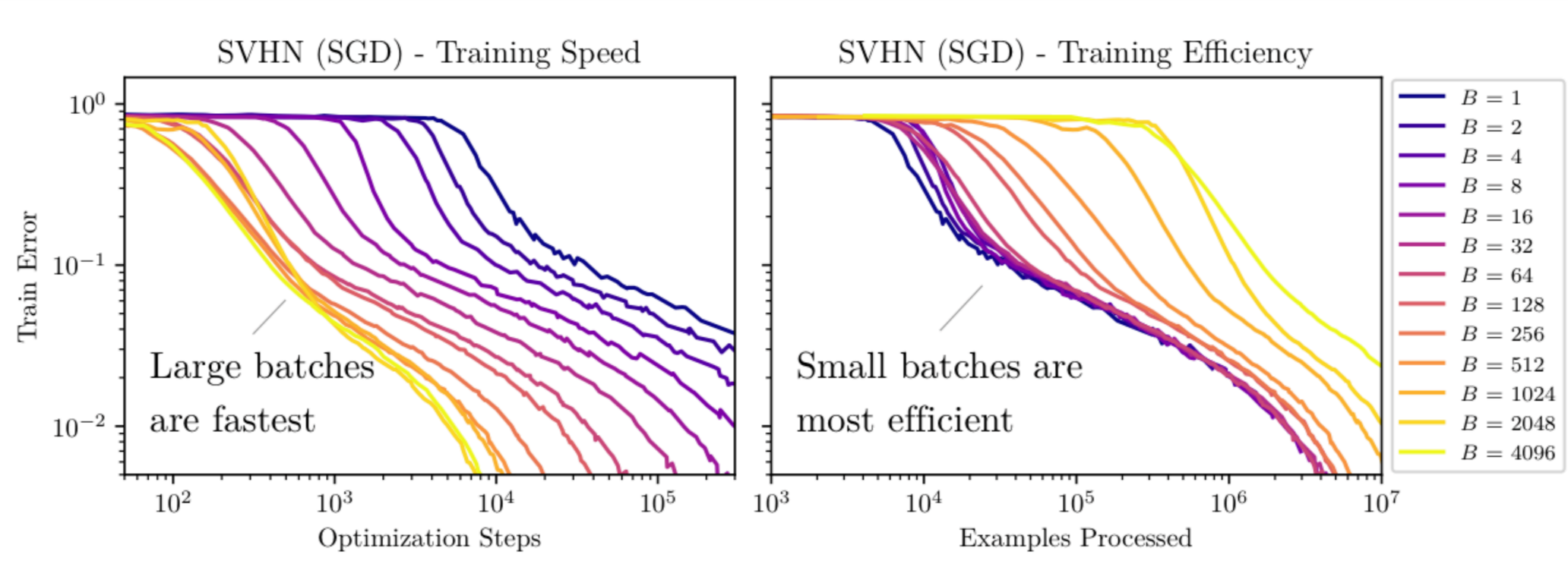


No noise ↔ SDE discretisation breaks

Why B_{crit}?

An Empirical Model of Large-Batch Training

It is steps vs. data Pareto optimal



No free acceleration above cbsz (need "more data")

No free acceleration from adding more GPUs (DDP)

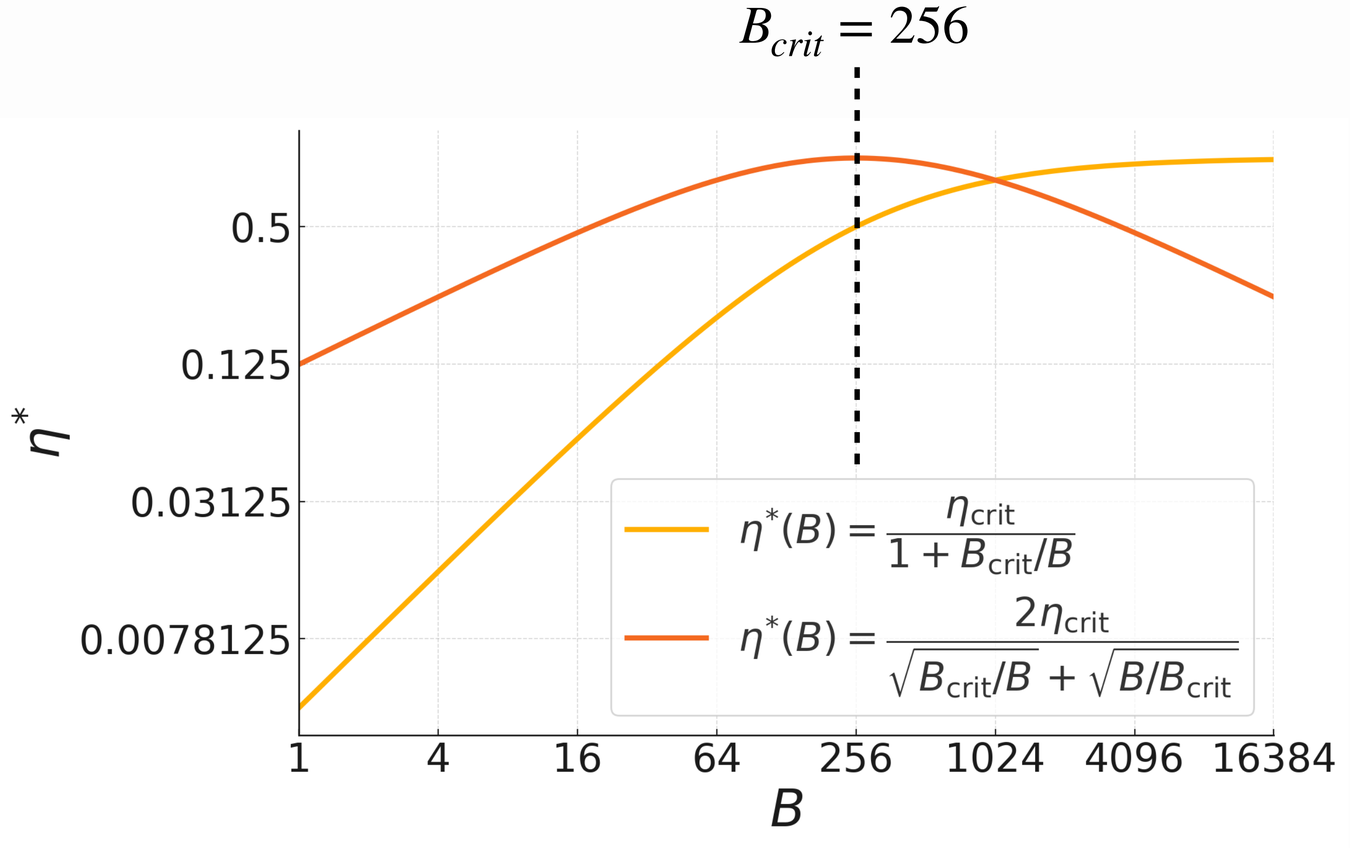
What about Adam?

Surge Phenomenon in Optimal Learning Rate and Batch Size Scaling

Oops

SGD $$\eta^* = \frac{\eta_{crit}}{1 + \frac{B_{crit}}{B}}$$

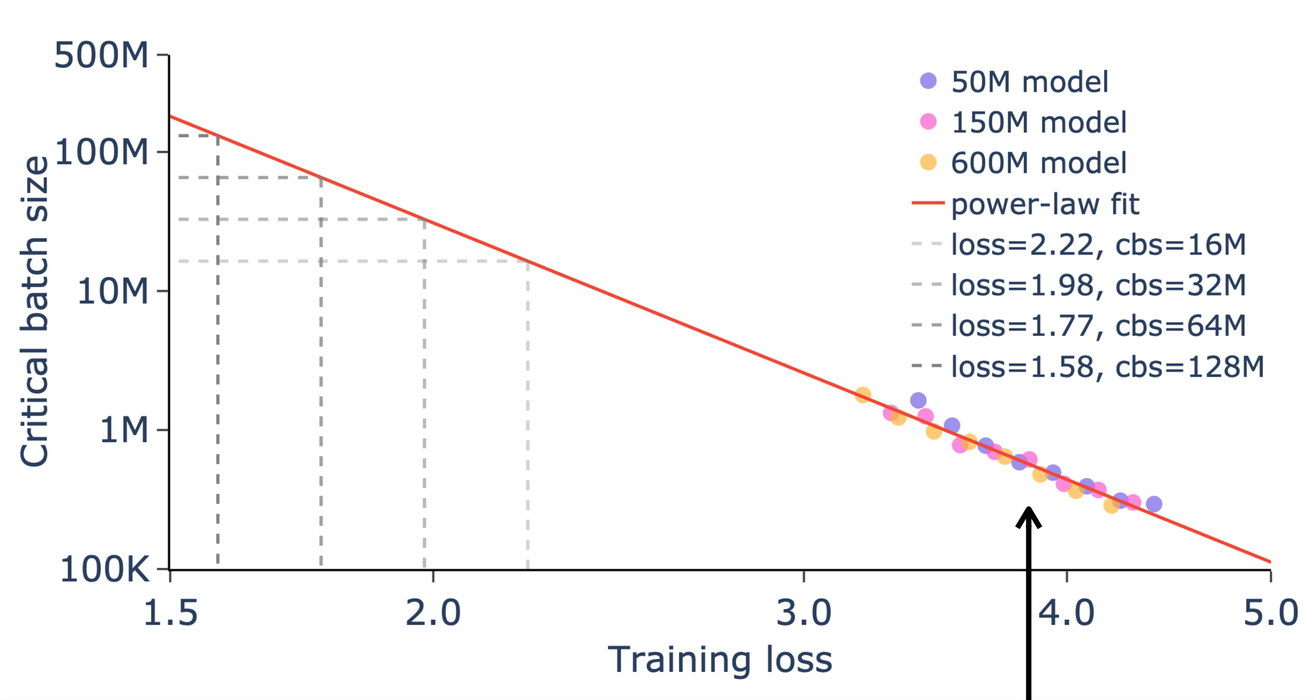
Adam $$\eta^* = \frac{\eta_{crit}}{\sqrt{\frac{B}{B_{crit}}} + \sqrt{\frac{B_{crit}}{B}}}$$



Is B_{crit} constant?

Minimax-01

No 🥲

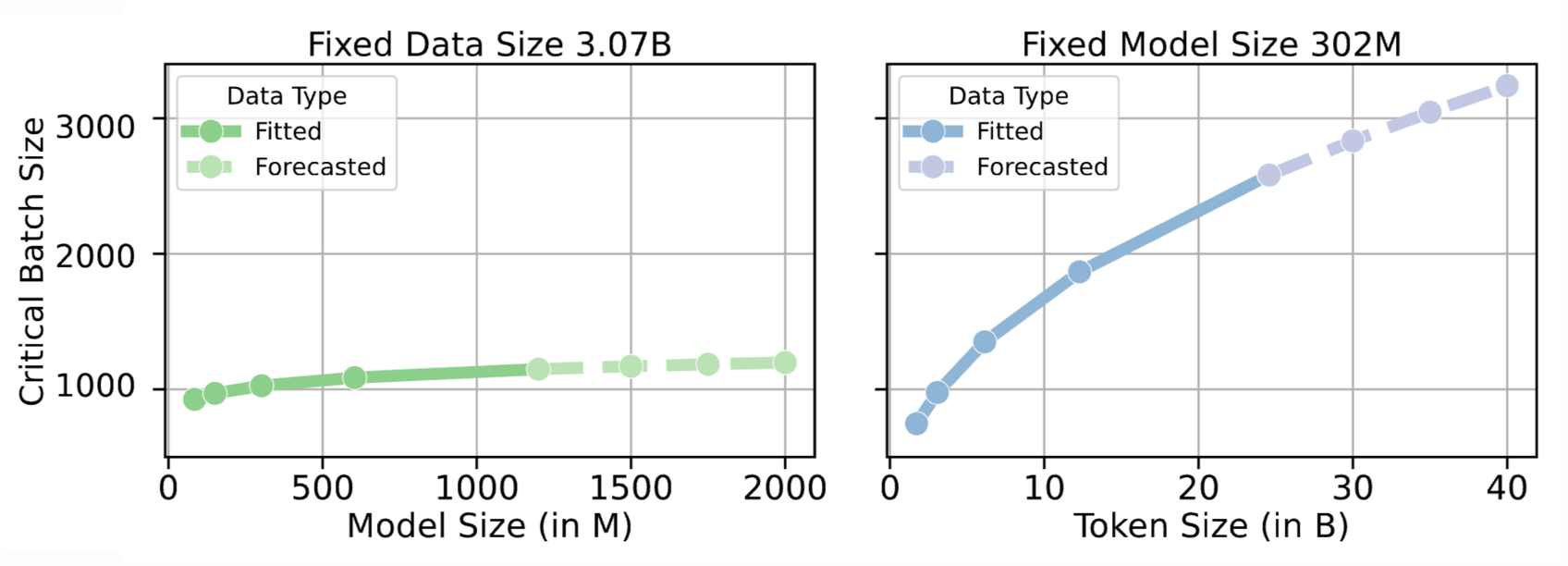


Doesn’t depend on model size

Is B_{crit} constant?

How Does Critical Batch Size Scale in Pre-training?

No 🥲 🥲



Takeaway on the theory of batch size scaling when scaling up data and model size:

- As we scale up model size while keeping the data size fixed, μP suggests that critical batch size does not scale with model width beyond a point.

Takeaway on scaling laws for critical batch sizes: Based on the scaling laws and controlled comparisons, we conclude that the increase in CBS in Chinchilla-optimal training is more strongly attributed to extended data size or training durations rather than the increase of model size.

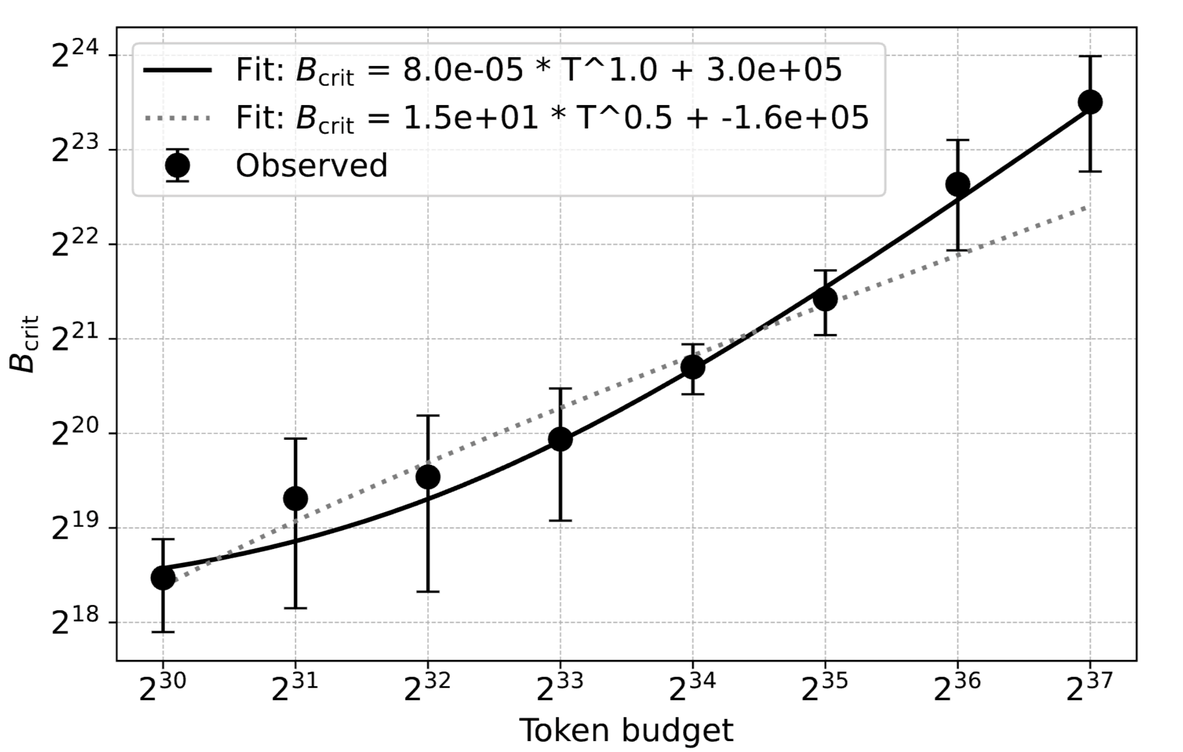
Is B_{crit} constant?

Time Transfer

No 🥲 🥲 🥲

$$\eta^* = \frac{\eta_{crit}}{\sqrt{\frac{B}{B_{crit}}} + \sqrt{\frac{B_{crit}}{B}}}$$

$$\eta^*(T) = \frac{\eta_{crit}(T)}{\sqrt{\frac{B}{B_{crit}(T)}} + \sqrt{\frac{B_{crit}(T)}{B}}}$$

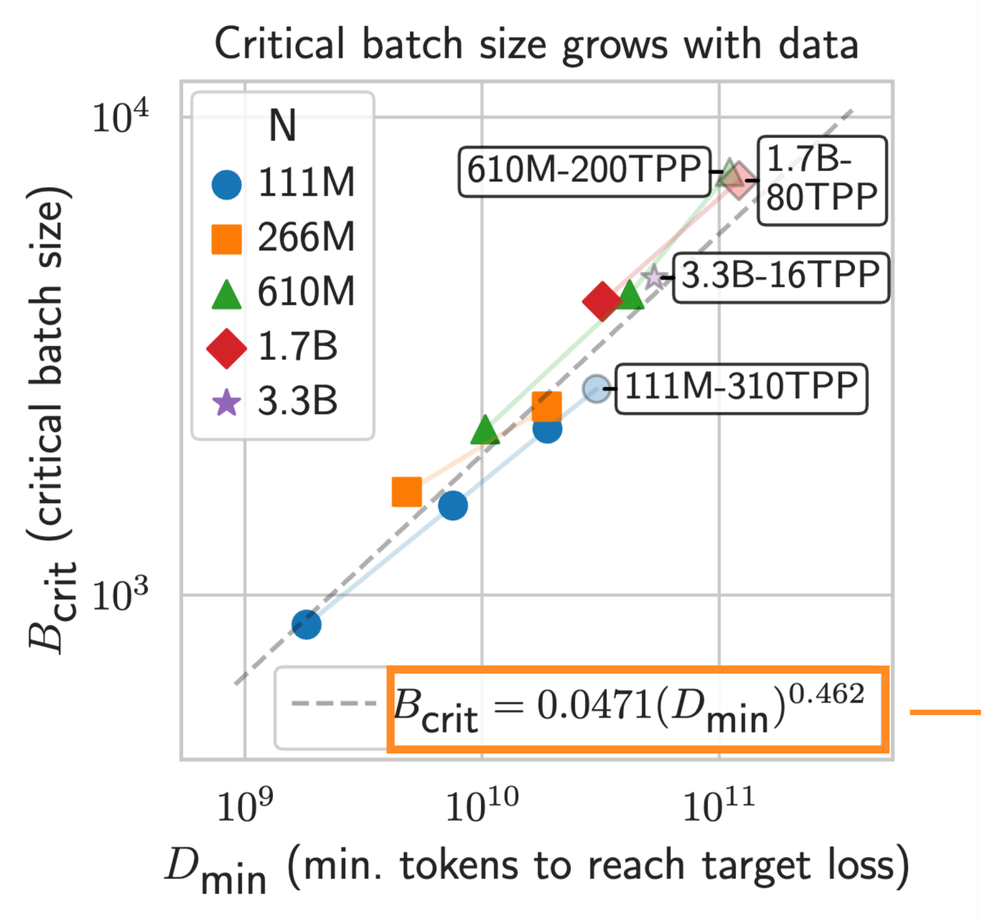


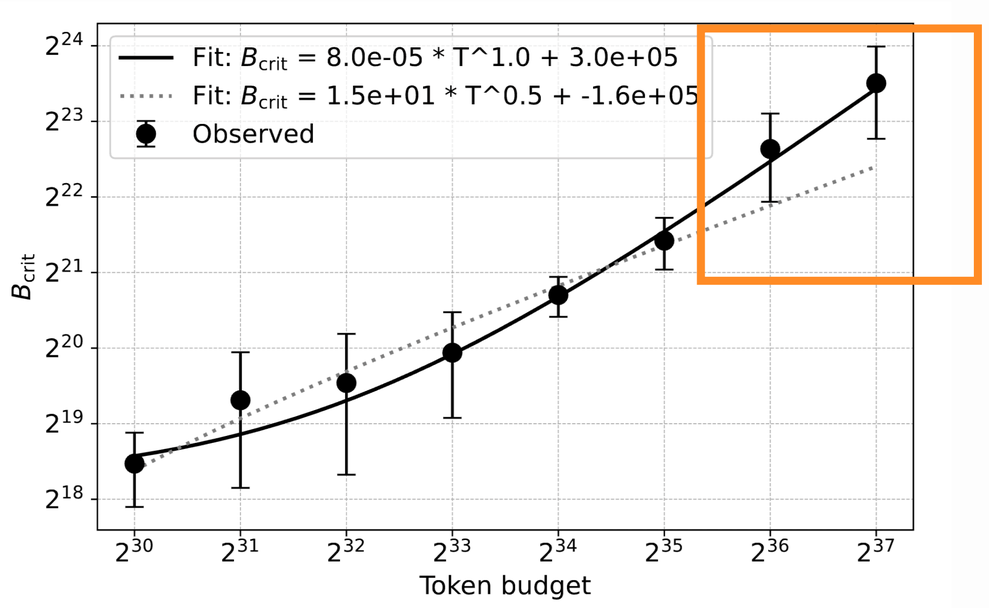
- **Critical batch size:** we experimentally find B_{crit} (see definition in Sec. 2.1) to evolve in time with $B_{\text{crit}} \propto T^{\alpha_B}$ and $\alpha_B = 1.0 \pm 0.2$ (Fig. 2, left). This dynamic affects optimal η scaling via Eq. 3.1 and drives the transition between various scaling behaviors (Sec. 3.2). Surprisingly, we show evidence that B_{crit} is not exclusively defined by the value of the loss function (Eq. 8) as suggested by McCandlish et al. (2018): models within μP share the same B_{crit} region while having different performance in terms of loss.

$B_{crit} \propto \sqrt{D}$?

Power Lines

Maybe, need better precision





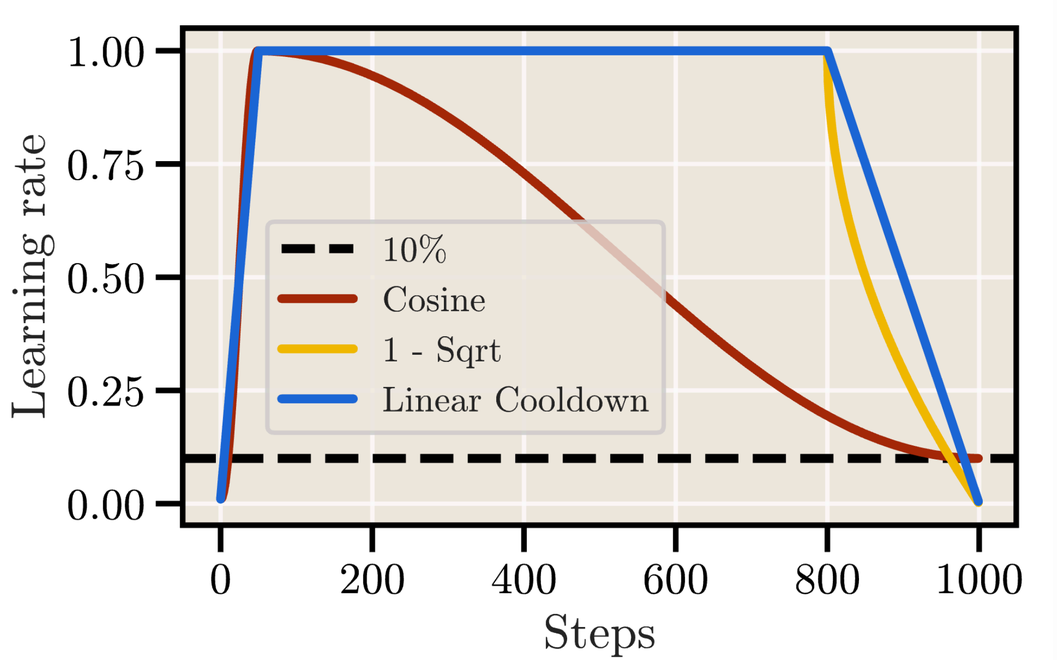
Strikingly, our fitted B_{crit} power law exponent is very close to that from [12]: 0.47 vs. 0.462. Given the many differences in approach (including context length, architecture, dataset, parameterization, use of weight decay, LR schedule, HP tuning strategy, etc., see Appendix D.3), this agreement in exponents indicates the fundamental power law relationship of B_{crit} with D persists across such differences. Moreover, given the significant implications of B_{crit}'s fundamental scaling behavior, it is valuable, scientifically and practically, that different approaches independently measure a similar scaling relationship.

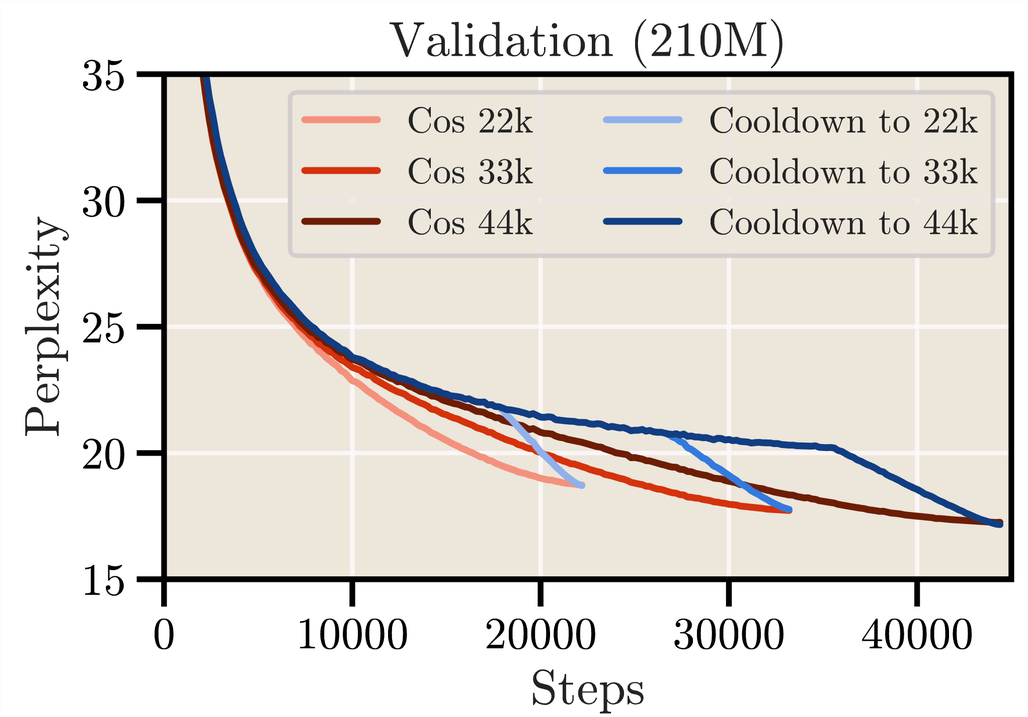
What to do?

Scaling Laws and Compute-Optimal Training
Beyond Fixed Training Durations

LR schedule

Fixed beforehand (**cosine**) Flexible (**WSD**)





Tldr; probably doesn't matter [click, click]

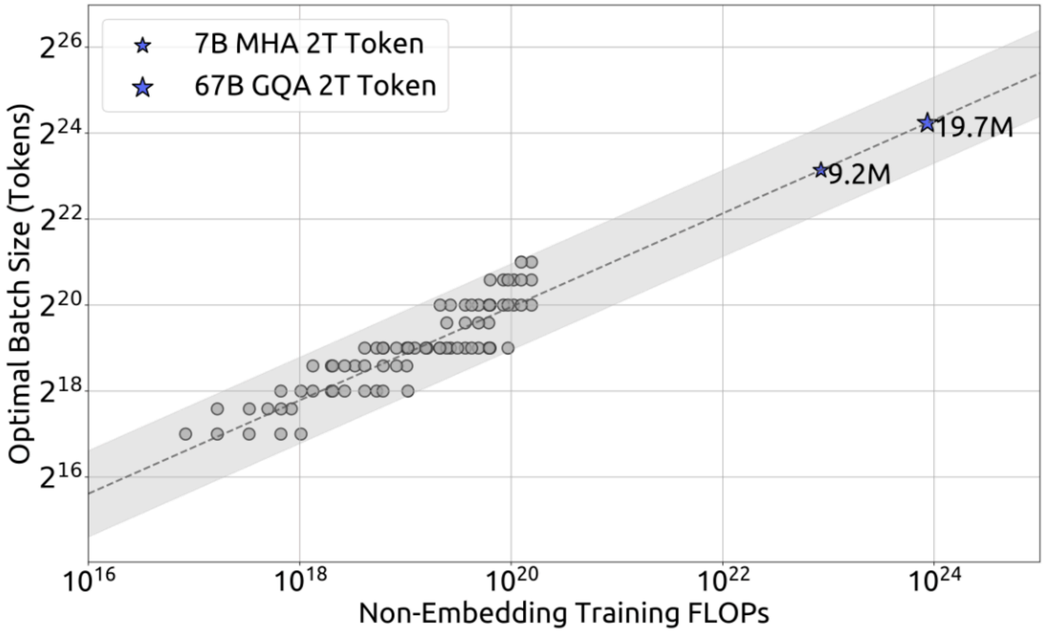
People say WSD better for continual pretraining [click, click],

Beware of re-warmup, decay to 0 [click]

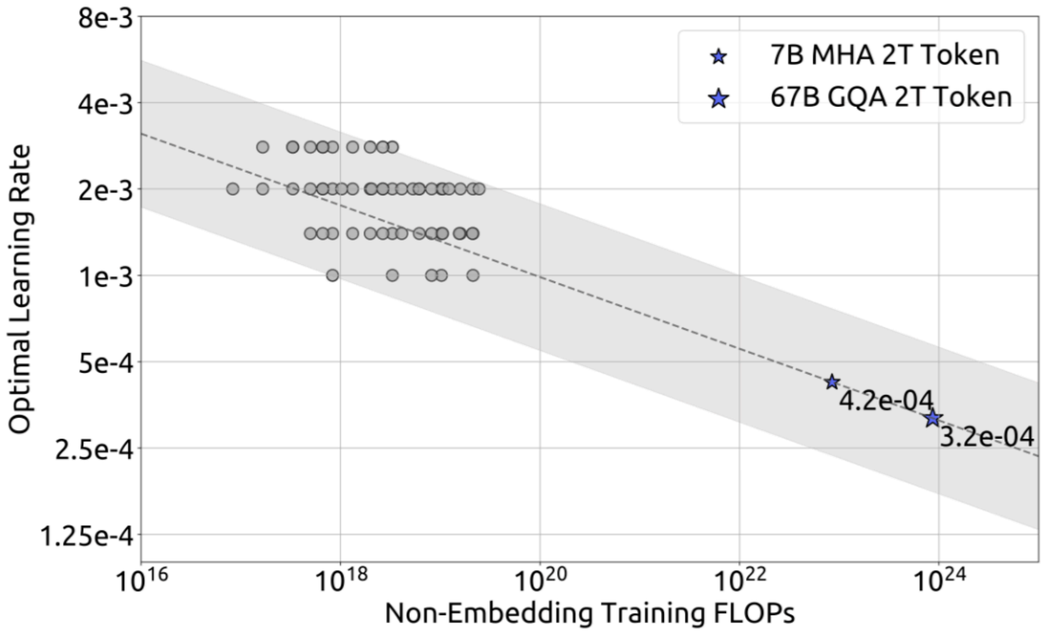
What to do?

DeepSeek LLM

Scaling laws [1]



(a) Batch size scaling curve



(b) Learning rate scaling curve

What to do?

Predictable Scale, Part I

Scaling laws [2]

Name	Data Recipe	Model Sparsity	LR	BS	Relative Error
OpenAI Law (Kaplan et al., 2020)	✘	✘	$3.239 * 10^{-3} + -1.395 * 10^{-4} log(N)$	$2e18\mathcal{L}^{-4.76190}$	9.51‰
Microsoft Law (Bjorck et al., 2024)	✘	✘	$1.3192e^{-5} N^{-0.23} D^{-0.32}$	-	9.25‰
DeepSeek Law (DeepSeek-AI et al., 2024a)	✘	✘	$0.3188C^{-0.1250}$	$0.2920C^{0.3271}$	9.26‰
Porian Law (Porian et al., 2024)	✘	✘	$3.7N^{-0.36}$	$0.7576N^{0.703}$	3.71‰
MiniCPM Law (Hu et al., 2024)	✘	✘	-	$\frac{2e18}{L^{6.24}}$	-
MeiTuan Law (Wang et al., 2024)	✘	✔	$\lambda\mathcal{L}^{-\alpha}$	$\lambda_B\mathcal{L}^{-\alpha_B^{-1}}$	-
Ours (Step Law)	✔	✔	$1.79N^{-0.713}D^{0.307}$	$0.58D^{0.571}$	**0.94‰**

Also show topological, sparsity and data-distribution independence

Model in SP, can do smarter?

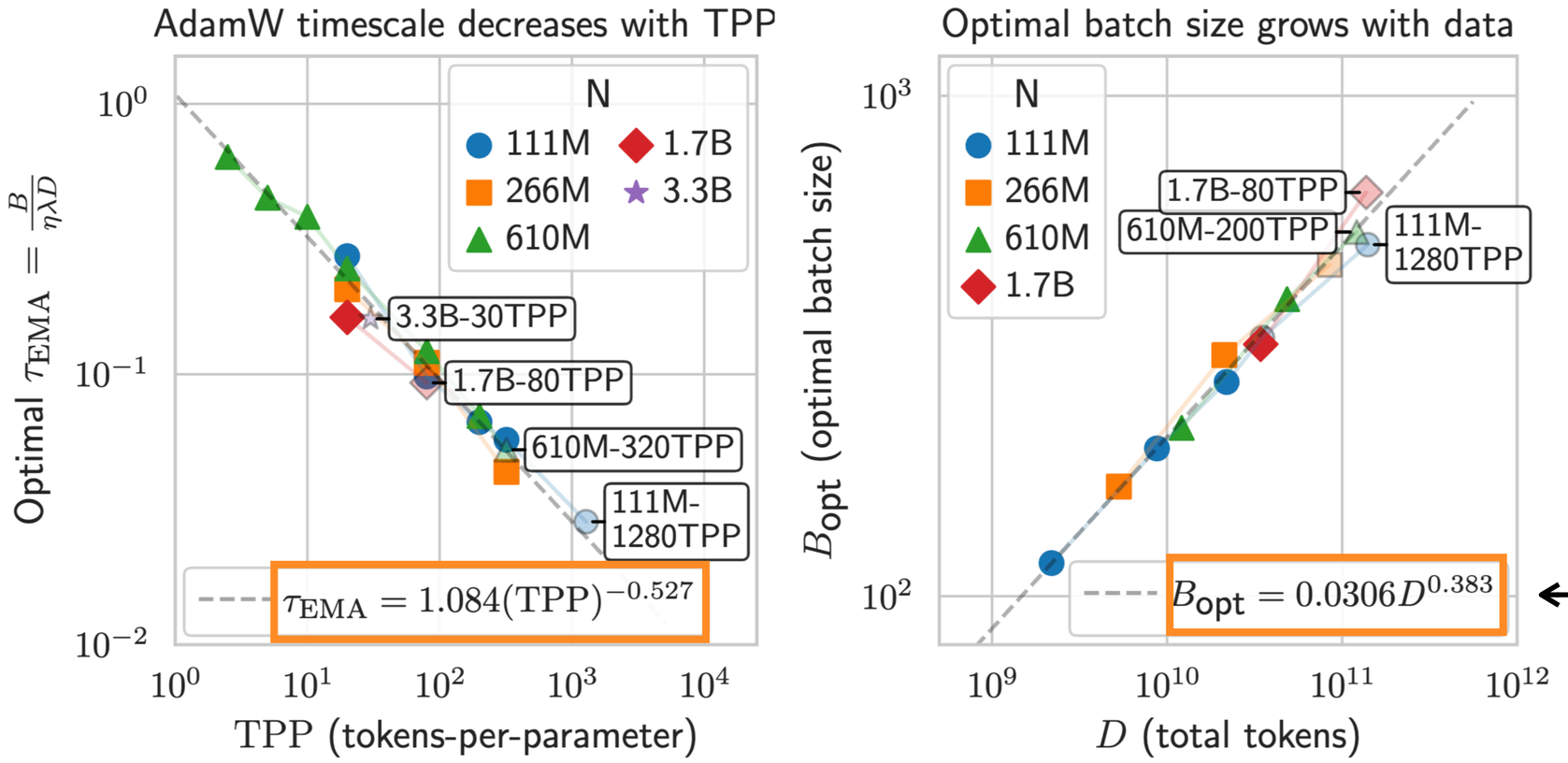
Doesn't depend on model size (recall same for cbsz)

What to do?

Scaling laws [3]

All models are in μP
$TPP = D/N$, λ = weight decay
$\tau_{EMA} = B/(\eta\lambda D)$ // recall noise scale

Power Lines



$B^* \sim \sqrt{D}$
(kind of ?)

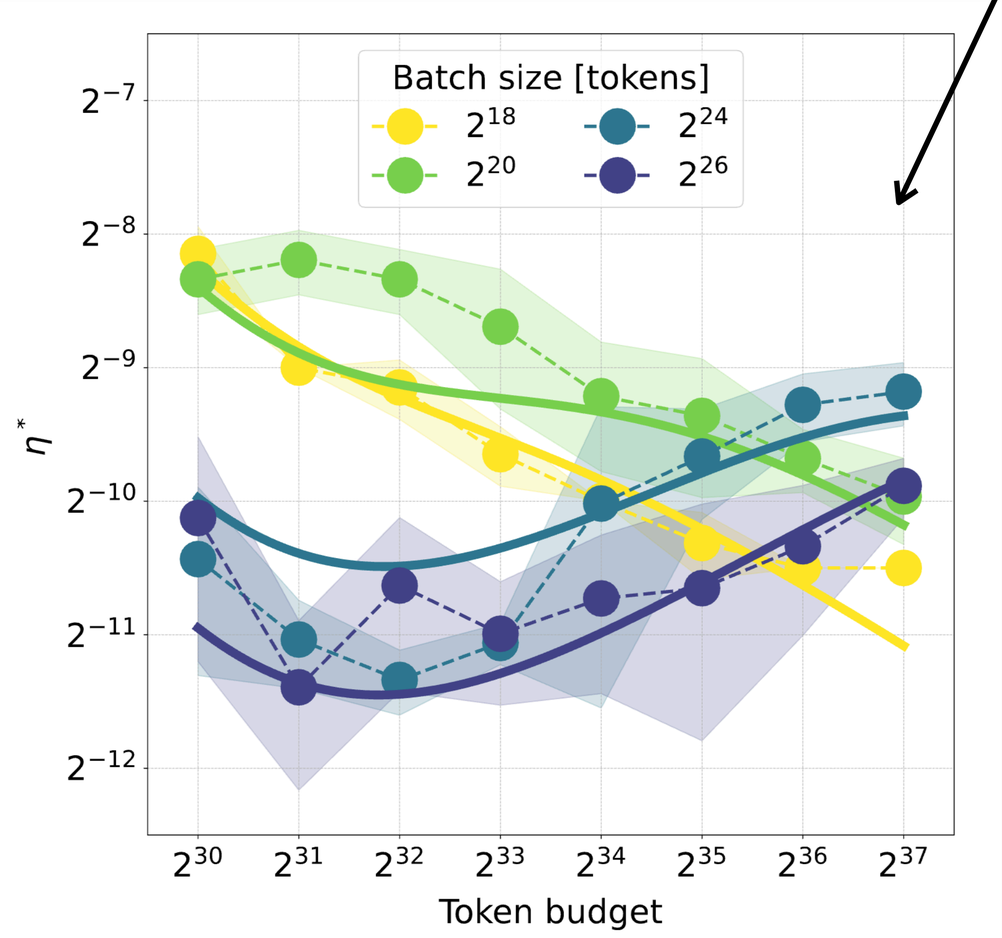
Note: this is in tension with [1, 2] and in agreement with [3]🤷🏻

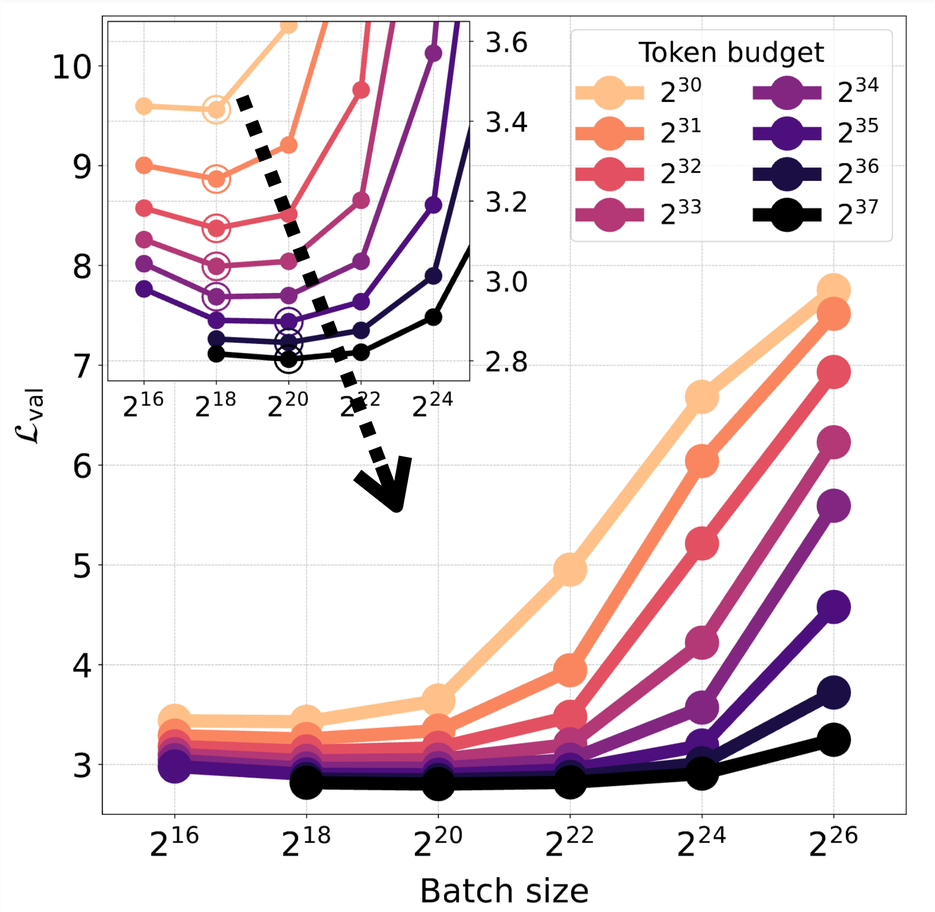
What to do?

Fit with $\eta^*(D) = \dfrac{\eta_{crit}(D)}{\sqrt{\dfrac{B}{B_{crit}(D)}} + \sqrt{\dfrac{B_{crit}(D)}{B}}}$

Time Transfer

Joint (B, η) look, beyond B_{crit}





Scaling regime depends on B

$B^* \sim \sqrt{D}$ (kind of ?)

What to do?

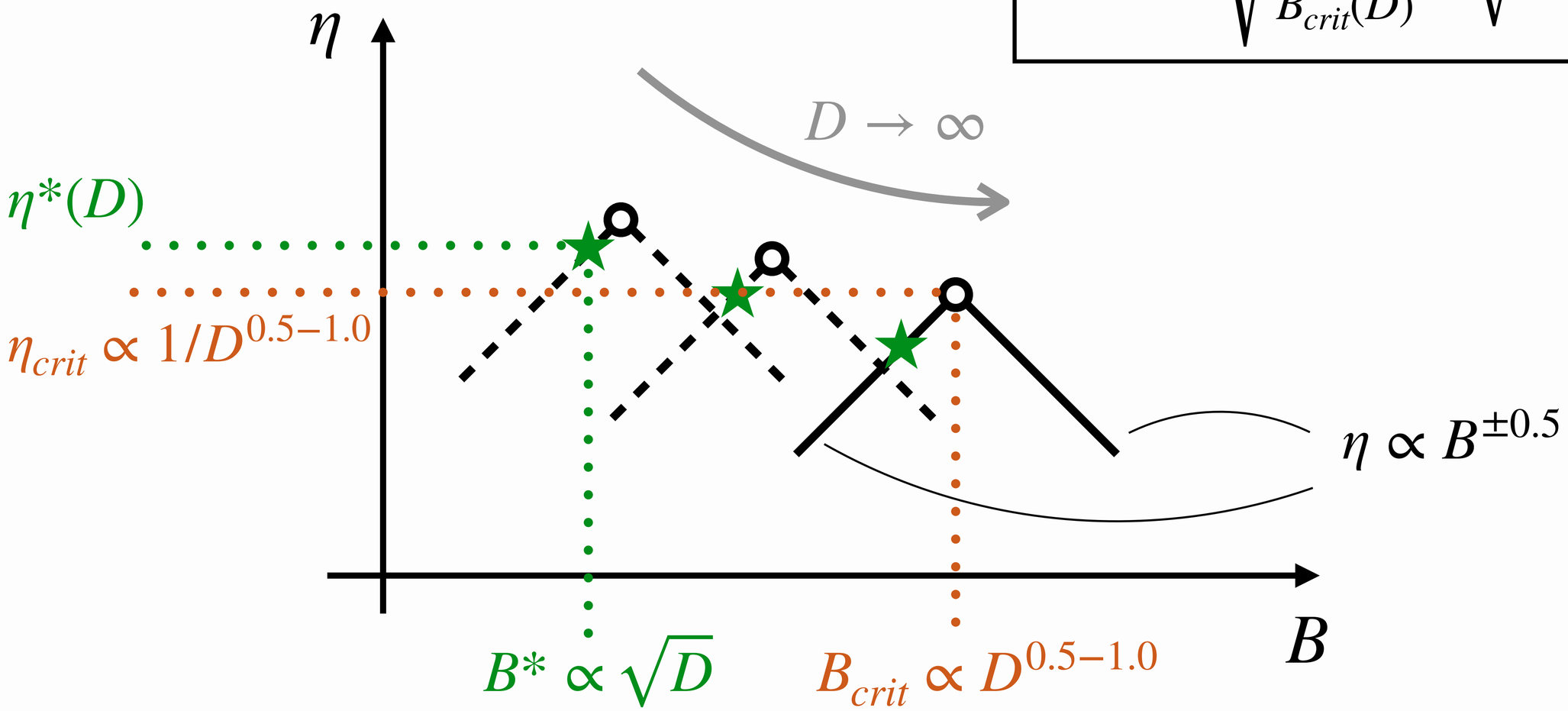
- **Measure it yourself** 😪
- **For example:**
  - Take small proxy model (in μP / Scion)
  - Run a grid search over (D, B, η) until compute allows
  - Make sure having B large enough to go above B_{crit}
  - Derive your scaling rules for η^*, B^*, B_{crit}
  - Share them with me!

What to do?

(My thoughts)

★ HP optimum

$$\eta^*(D) = \frac{\eta_{crit}(D)}{\sqrt{\frac{B}{B_{crit}(D)}} + \sqrt{\frac{B_{crit}(D)}{B}}}$$



Summary

- **It LOOKS like**
  - *Independently of model size (at least in μP)...*
  - *... and with no direct dependency on loss/compute*
  - *... for Adam(W)*
- $B_{crit} \propto D^{0.5-1.0}$
- $B^* \propto \sqrt{D}$ **, *assuming*** $\eta = \eta^*(D)$
- η^* **unclear**
  - μP makes scaling independent of model size
  - [?] Data scaling regime depends on B and B_{crit}

- **Open research questions**
  - Scaling with normed optimisers
  - D-scaling exponents + universality
  - Joint (η, B, D) modelling
  - Unified theory with model N-scaling

Scaling laws

Intro

Why scaling laws?

- **Predictability**
  - What will I get?
- **Optimality**
  - Do I invest resources right?
- **Confidence**
  - Will I get screwed?

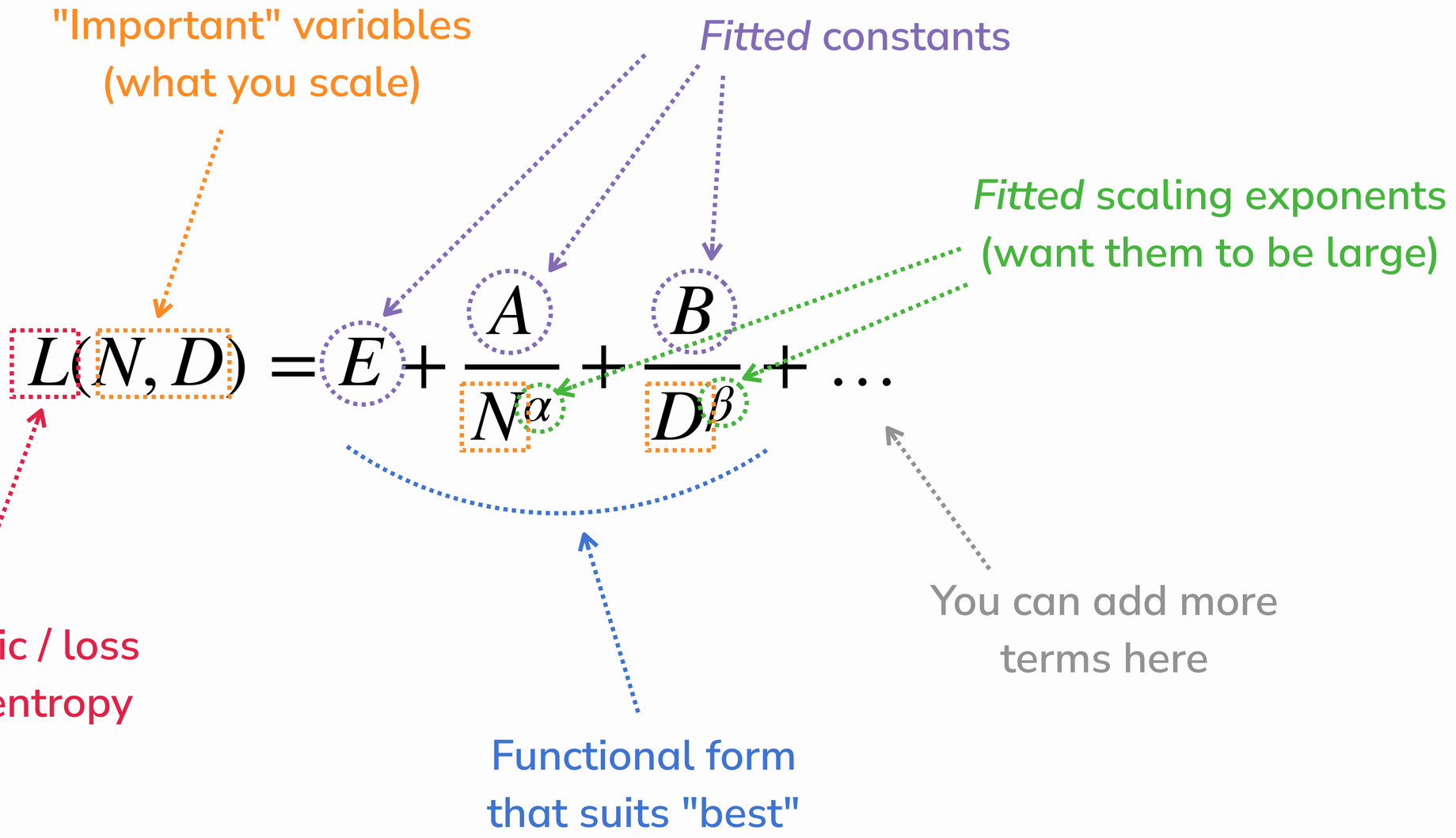
Intro

Most common form

$$L(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}} + \dots$$

Intro

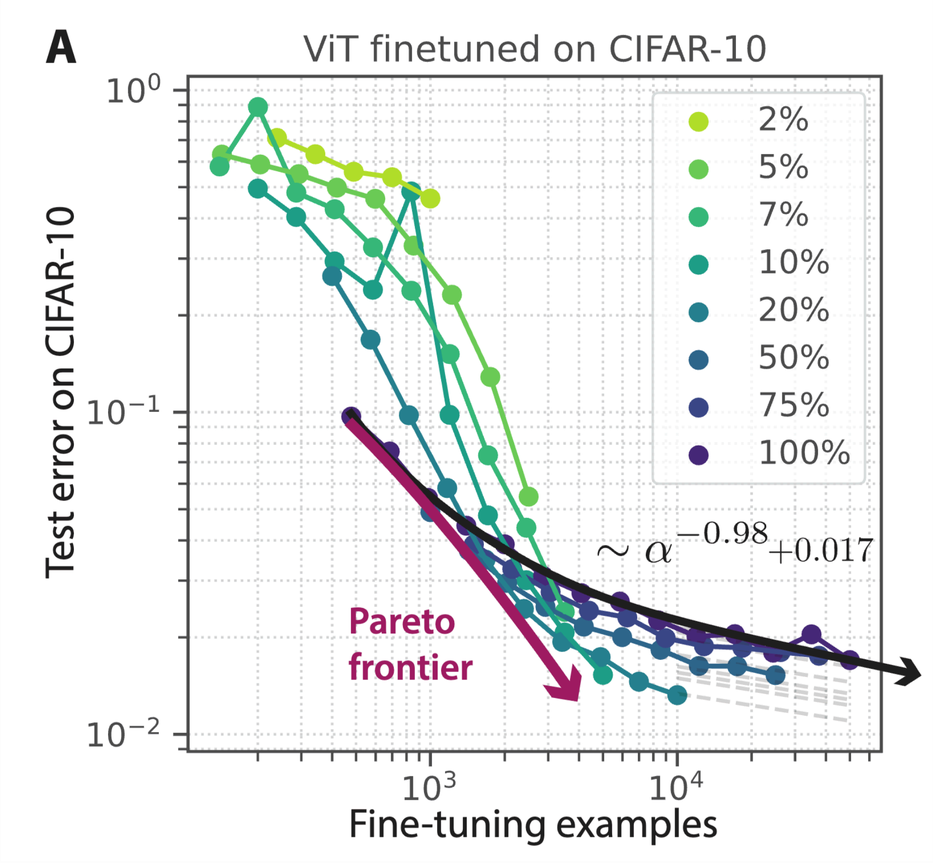
Most common form



Intro

Katie Everett's thread

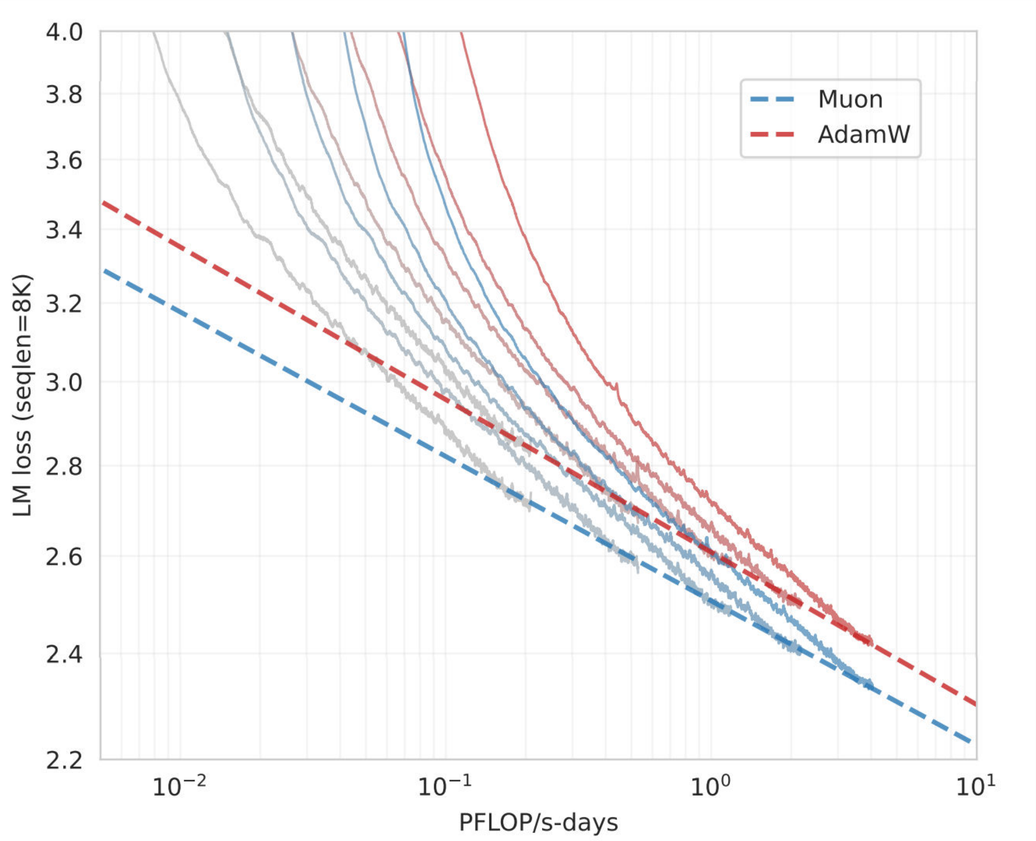
Example: what scales better?



Data pruning strategy?

Table 3: Fitted parameters of the scaling law curves

	Muon	AdamW
LM loss (seqlen=8K)	$2.506 \times C^{-0.052}$	$2.608 \times C^{-0.054}$

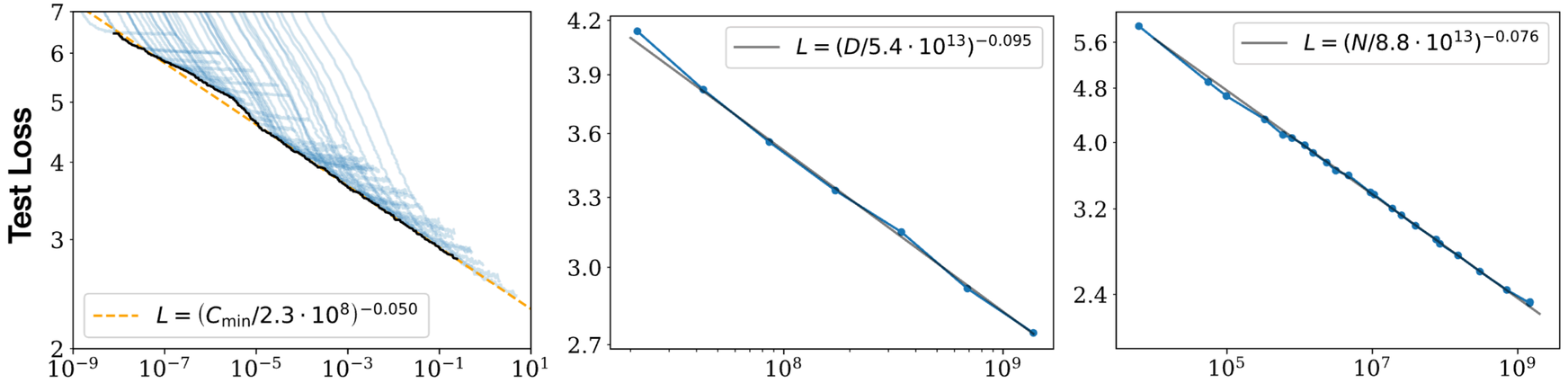


Optimizer choice?

Base

Kaplan

Scaling Laws for Neural Language Models



Parameters	Data	Compute	Batch Size	Equation
N	∞	∞	Fixed	$L(N) = (N_c/N)^{\alpha_N}$
∞	D	Early Stop	Fixed	$L(D) = (D_c/D)^{\alpha_D}$
Optimal	∞	C	Fixed	$L(C) = (C_c/C)^{\alpha_C}$ (naive)
N_{opt}	D_{opt}	C_{min}	$B \ll B_{\text{crit}}$	$L(C_{\text{min}}) = \left(C_c^{\text{min}}/C_{\text{min}}\right)^{\alpha_C^{\text{min}}}$
N	D	Early Stop	Fixed	$L(N,D) = \left[\left(\frac{N_c}{N}\right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D}\right]^{\alpha_D}$
N	∞	S steps	B	$L(N,S) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{\text{min}}(S,B)}\right)^{\alpha_S}$

There's way more in the paper

Base

Chinchilla

Training Compute-Optimal Large Language Models

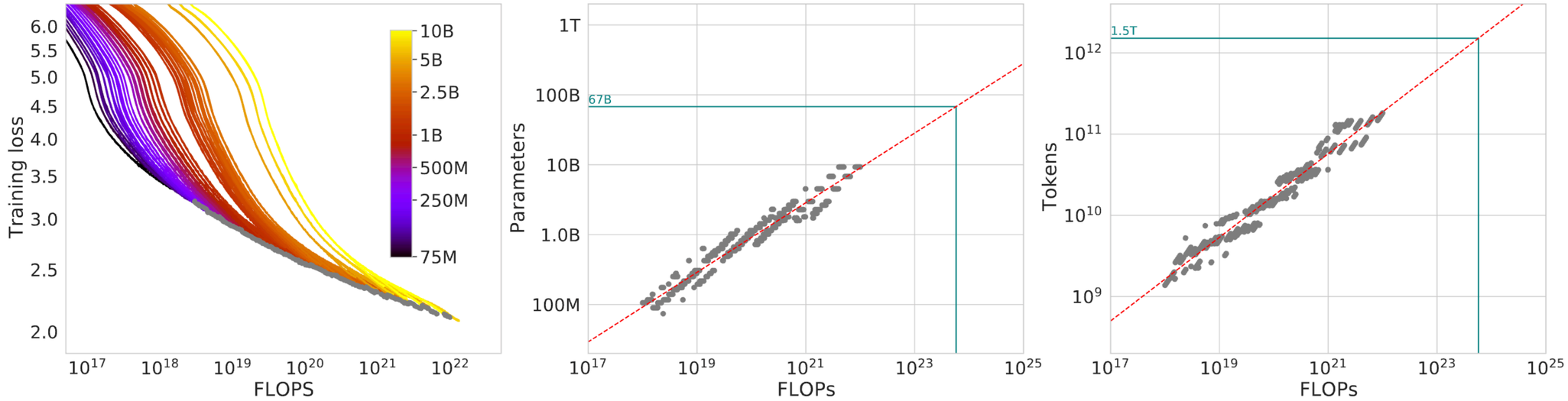


Table 2 | **Estimated parameter and data scaling with increased training compute.** The listed values are the exponents, a and b, on the relationship $N_{opt} \propto C^a$ and $D_{opt} \propto C^b$. Our analysis suggests a near equal scaling in parameters and data with increasing compute which is in clear contrast to previous work on the scaling of large models. The 10^{th} and 90^{th} percentiles are estimated via bootstrapping data (80% of the dataset is sampled 100 times) and are shown in parenthesis.

Approach	Coeff. a where $N_{opt} \propto C^a$	Coeff. b where $D_{opt} \propto C^b$
1. Minimum over training curves	0.50 (0.488, 0.502)	0.50 (0.501, 0.512)
2. IsoFLOP profiles	0.49 (0.462, 0.534)	0.51 (0.483, 0.529)
3. Parametric modelling of the loss	0.46 (0.454, 0.455)	0.54 (0.542, 0.543)
Kaplan et al. (2020)	0.73	0.27

???

3. Parametric modelling of the loss	0.46 (0.454, 0.455)	0.54 (0.542, 0.543)
Kaplan et al. (2020)	0.73	0.27

Methodology

How to (scaling laws)

(Mis)Fitting: A Survey of Scaling Laws

SCALING LAW REPRODUCIBILITY CHECKLIST

Scaling Law Hypothesis (§3)
- Form
- Variables (Input)
- Parameters
- Derivation and Motivation
- Assumptions

Training Setup (§4)
- # of models
- Model size range
- Dataset source & size
- Parameter/ FLOP Count Calculation
- Hyperparameter Choice
- Other Settings
- Code Open Sourcing

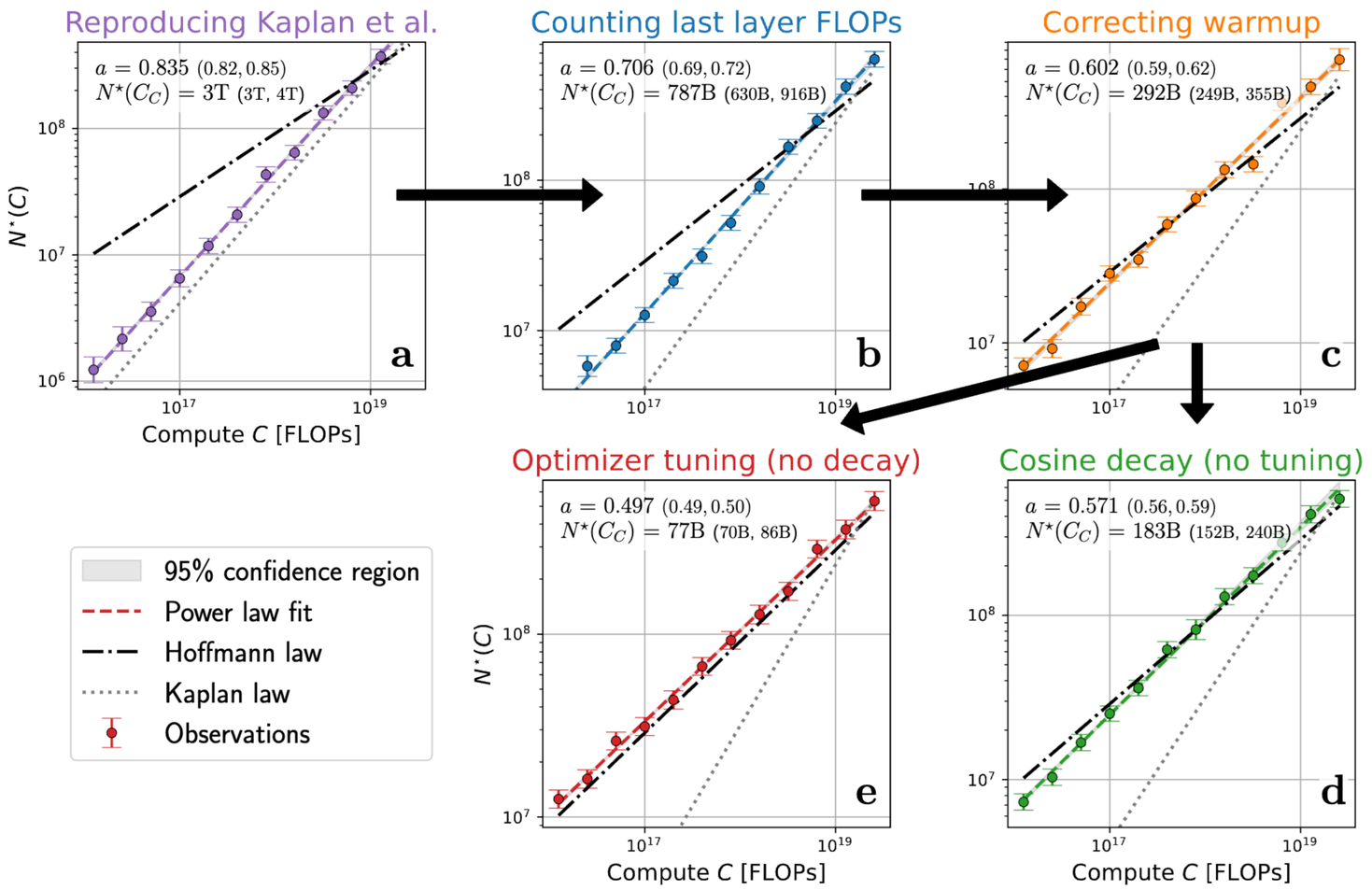
Data Collection(§5)
- Checkpoint Open Sourcing
- # Checkpoints per Power Law
- Evaluation Dataset & Metric
- Metric Modification & Code

Fitting Algorithm (§6)
- Objective (Loss)
- Algorithm
- Optimization Hyperparameters
- Optimization Initialization
- Data Usage Coverage
- Validation of Law(s)

Methodology

Differences to reconcile

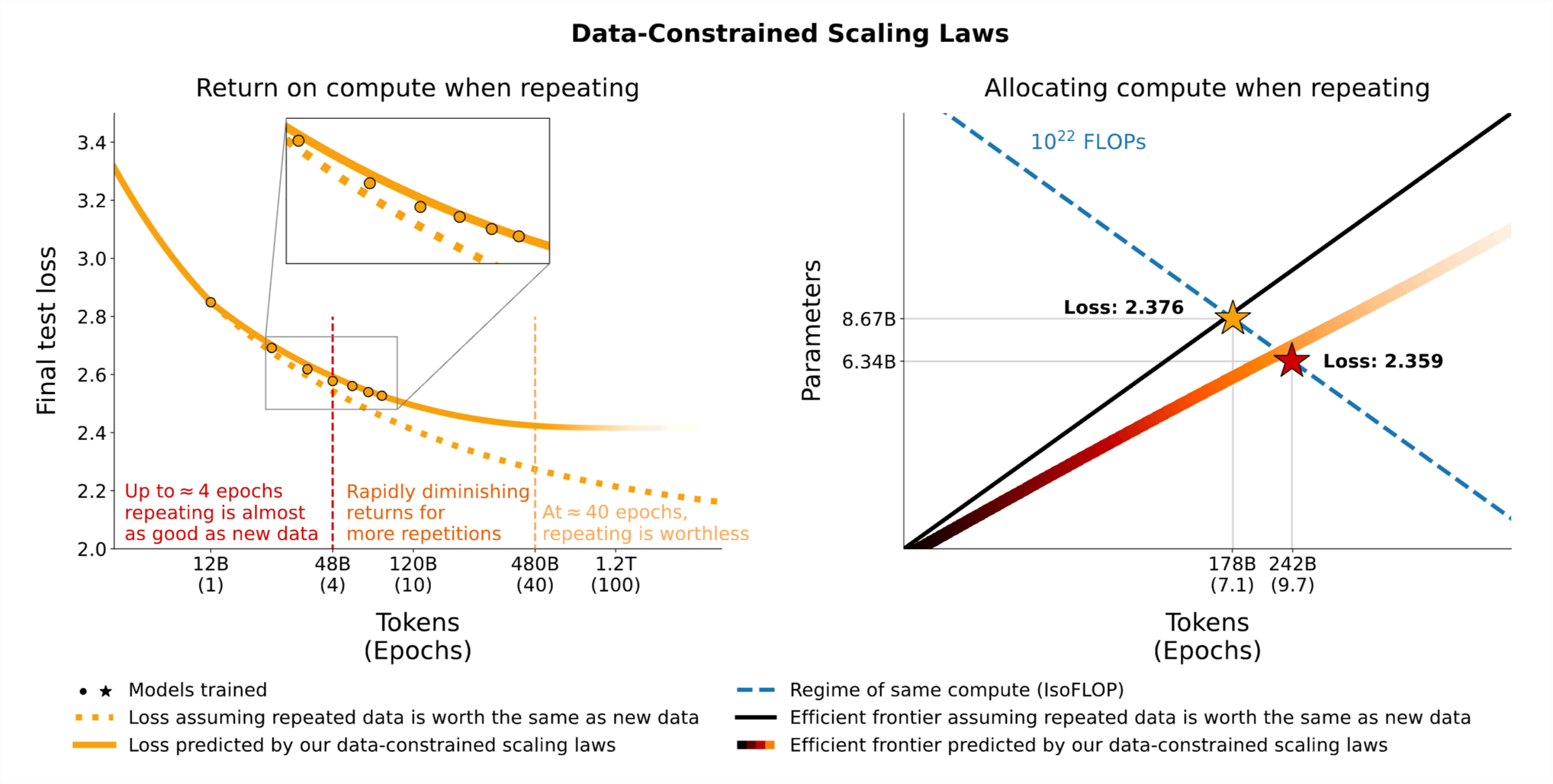
Resolving Discrepancies in Compute-Optimal Scaling of Language Models



Data repetition?

Scaling Data-Constrained Language Models

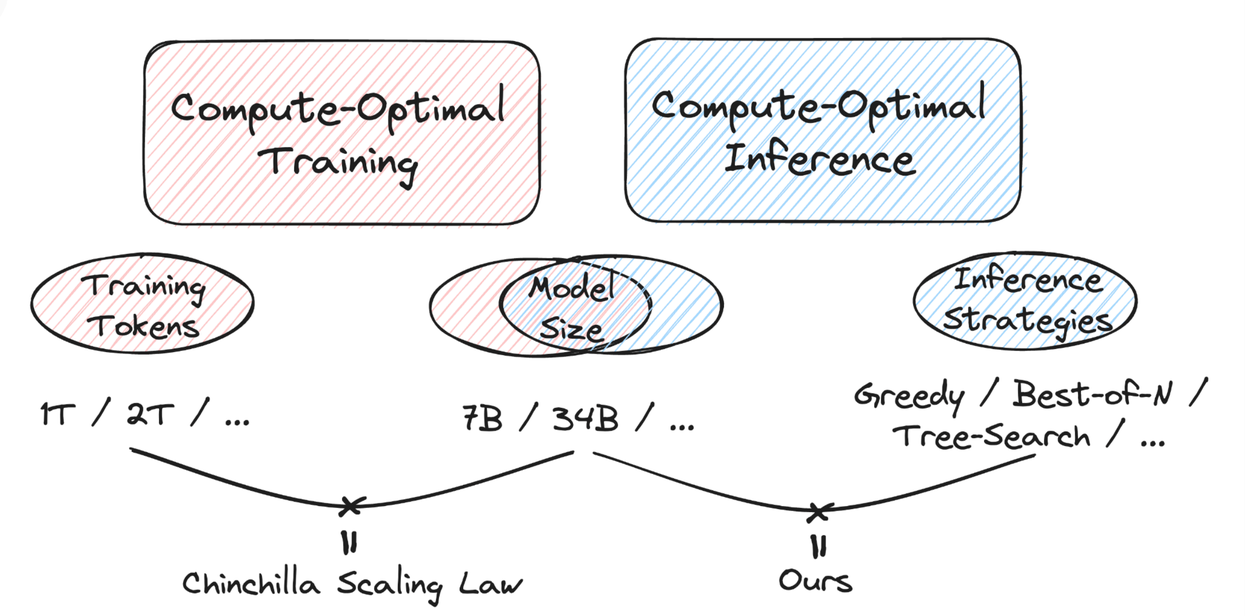
Yes

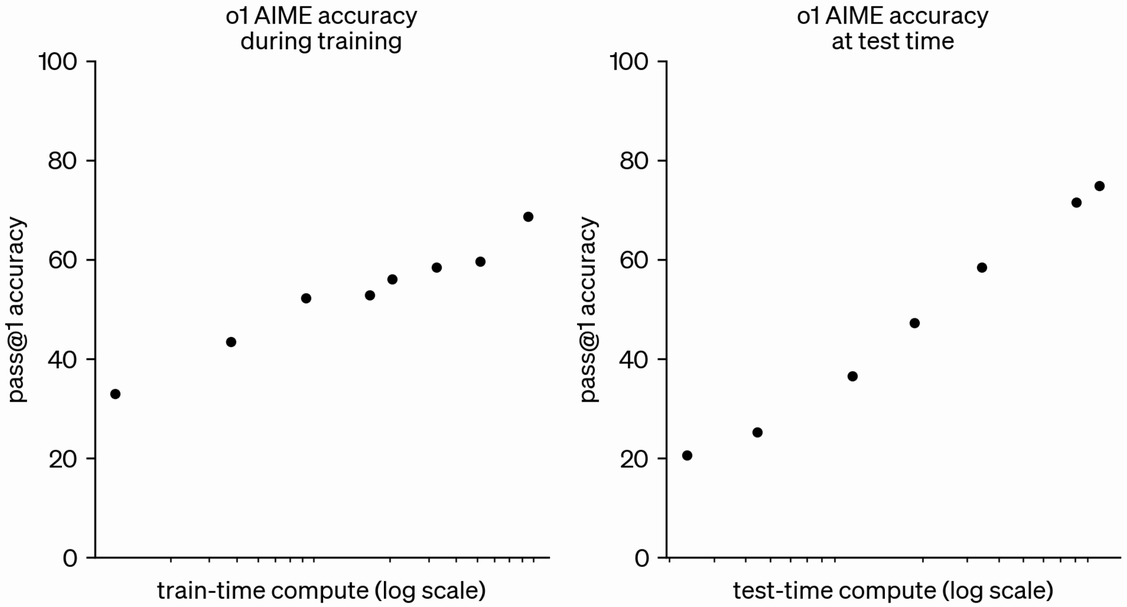


Beyond training

Test-time scaling

o1 release post
Inference scaling laws
HF blogpost





Beyond training

Inference (exists)

Beyond Chinchilla-Optimal: Accounting for Inference in Language Model Scaling Laws

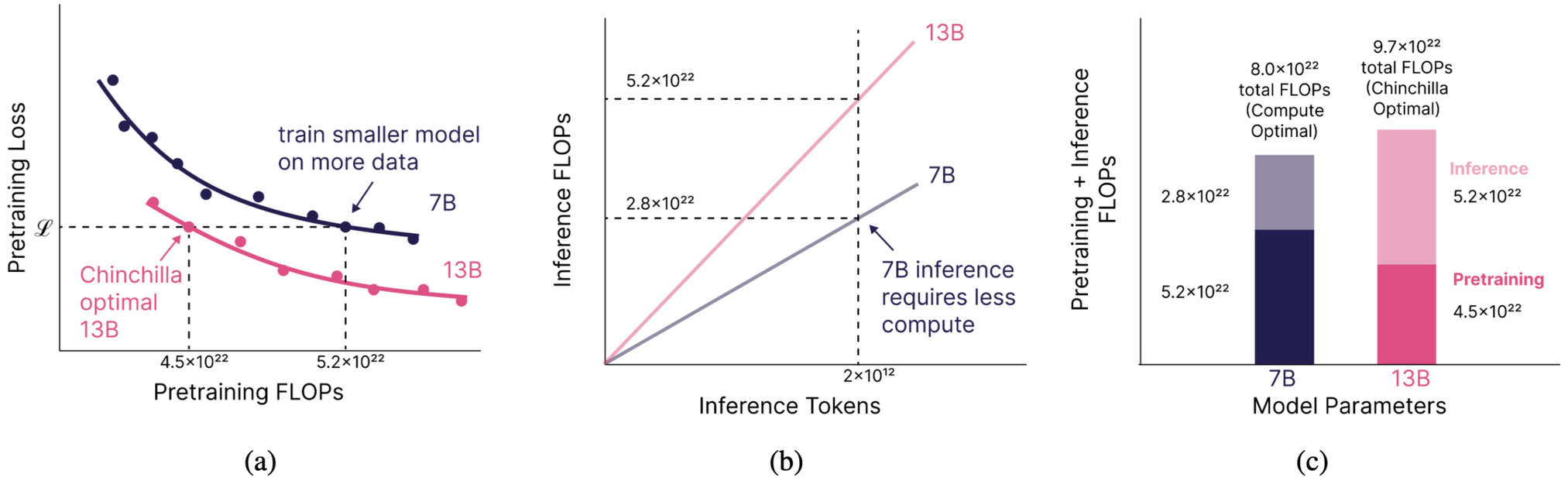
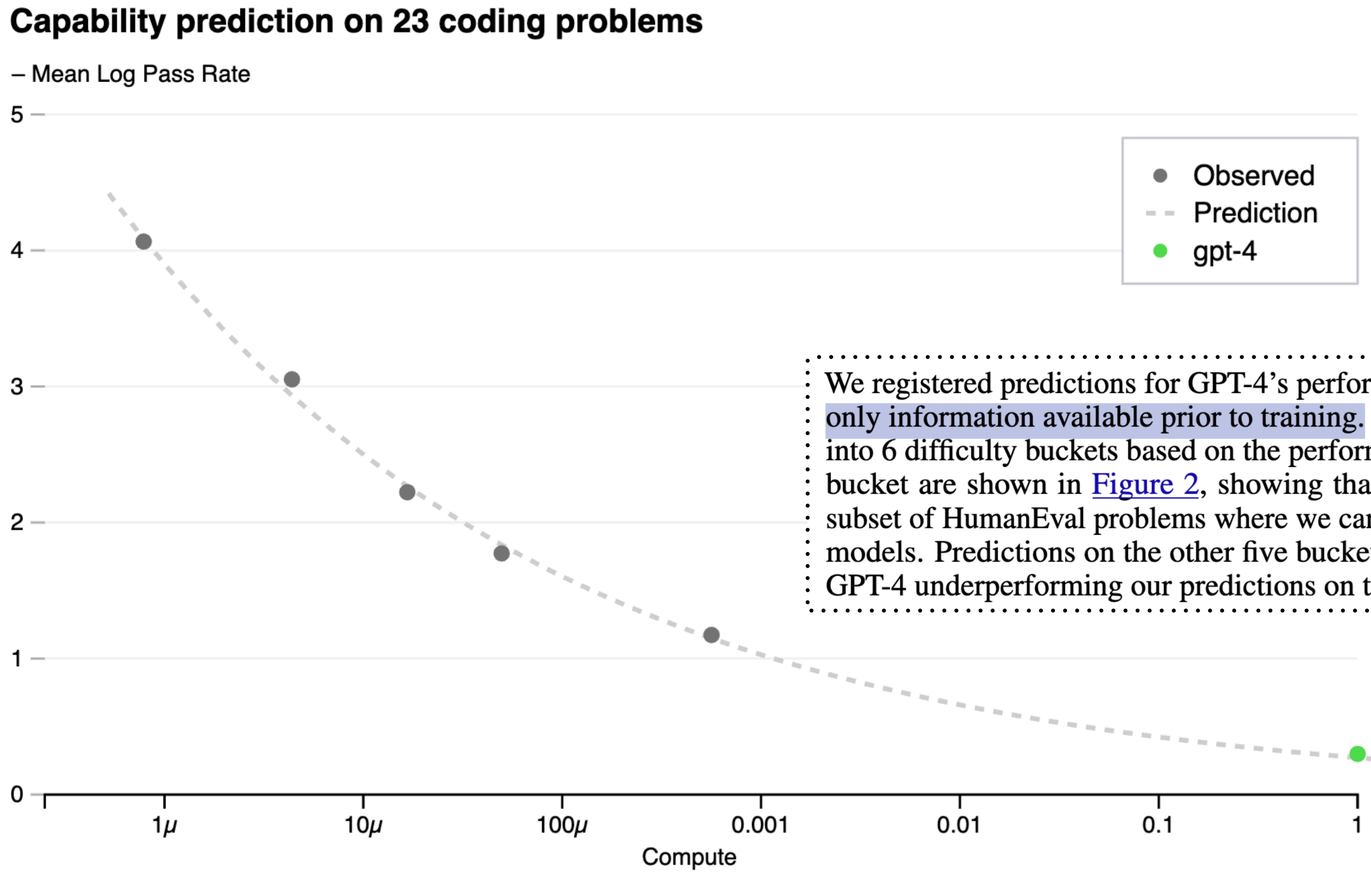


Figure 1: Schematic of compute savings achieved via our approach. An LLM developer seeking to train a 13B model who expects 2 trillion tokens of inference demand during the model's lifetime can reduce their total compute by 1.7×10^{22} FLOPs (17%) by instead training a 7B model on more data (a). The extra compute required to train the 7B model beyond its Chinchilla-optimal point to match the 13B's quality is made up for during inference (b), (c). Our method quantifies this training-inference trade-off, producing models that are optimal over their total lifetime.

Beyond loss prediction

GPT 4 report

GPT 4



We registered predictions for GPT-4's performance on HumanEval before training completed, using only information available prior to training. All but the 15 hardest HumanEval problems were split into 6 difficulty buckets based on the performance of smaller models. The results on the 3rd easiest bucket are shown in Figure 2, showing that the resulting predictions were very accurate for this subset of HumanEval problems where we can accurately estimate $\log(\text{pass\_rate})$ for several smaller models. Predictions on the other five buckets performed almost as well, the main exception being GPT-4 underperforming our predictions on the easiest bucket.

Figure 2. Performance of GPT-4 and smaller models. The metric is mean log pass rate on a subset of the HumanEval dataset. A power law fit to the smaller models (excluding GPT-4) is shown as the dotted line; this fit accurately predicts GPT-4's performance. The x-axis is training compute normalized so that GPT-4 is 1.

Beyond loss prediction

Llama 3 report

Llama 3

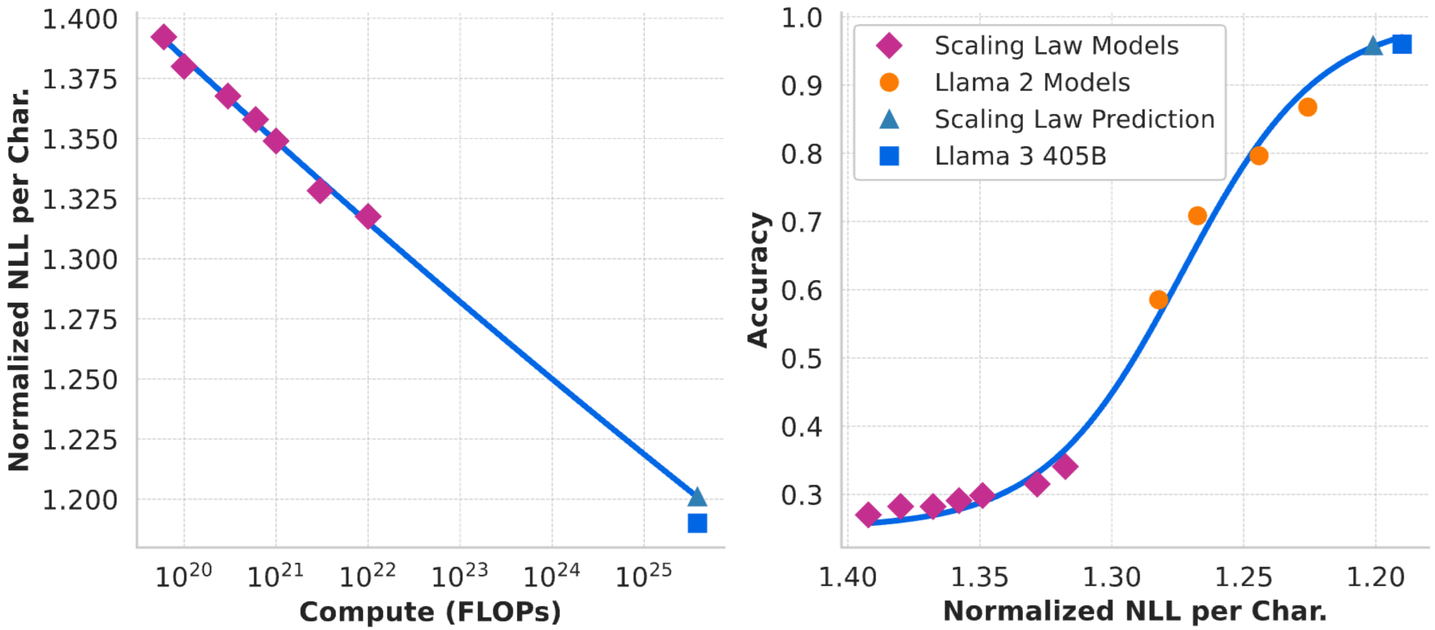


Figure 4 Scaling law forecast for ARC Challenge. *Left:* Normalized negative log-likelihood of the correct answer on the ARC Challenge benchmark as a function of pre-training FLOPs. *Right:* ARC Challenge benchmark accuracy as a function of the normalized negative log-likelihood of the correct answer. This analysis enables us to predict model performance on the ARC Challenge benchmark before pre-training commences. See text for details.

Beyond loss prediction

PassUntil

Predicting Emergent Abilities with Infinite Resolution Evaluation

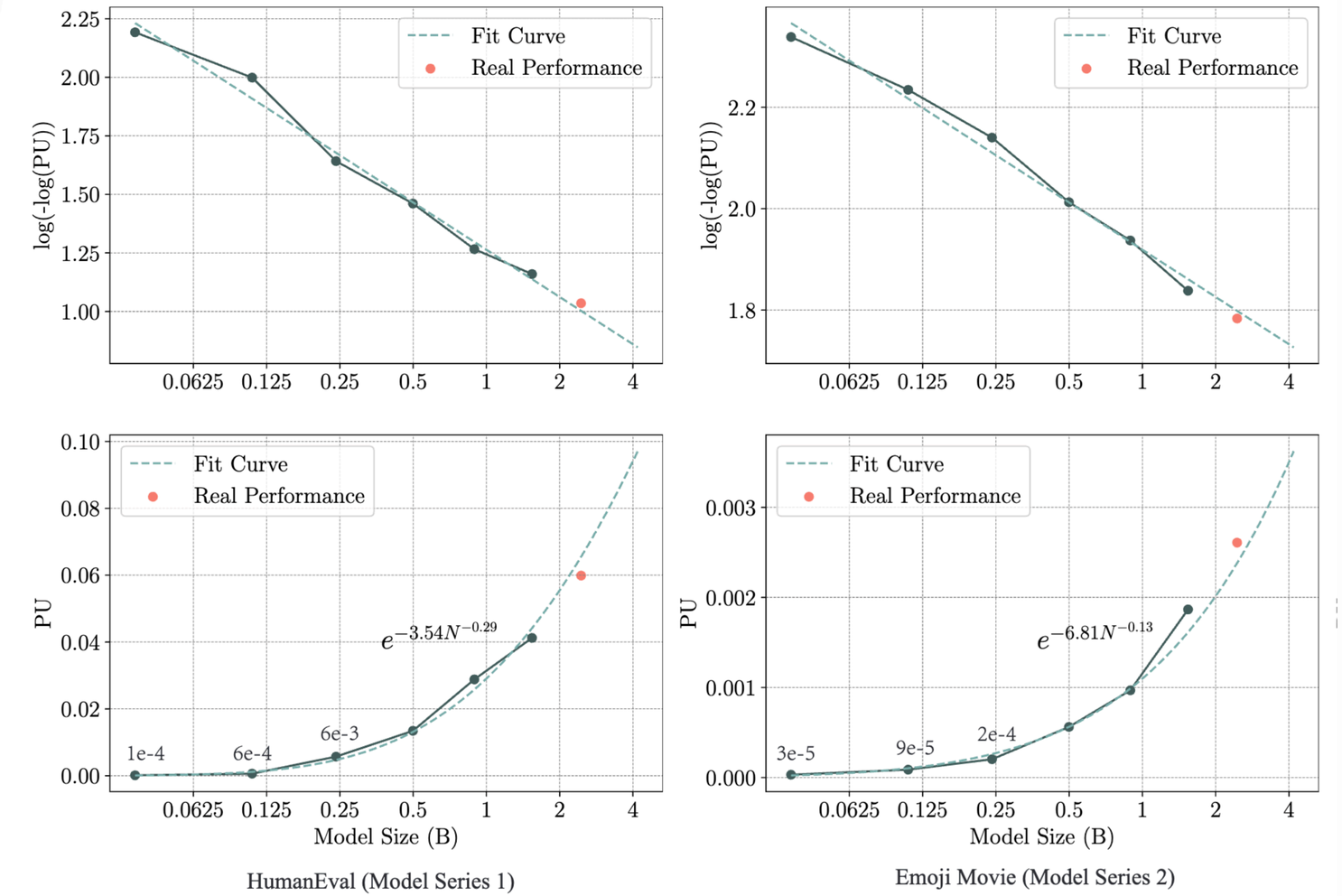
See also How Do Large Language Monkeys Get Their Power (Laws)?

Probability estimate of model passing a task

Number of passes we want

$$PU = \frac{r}{K}$$

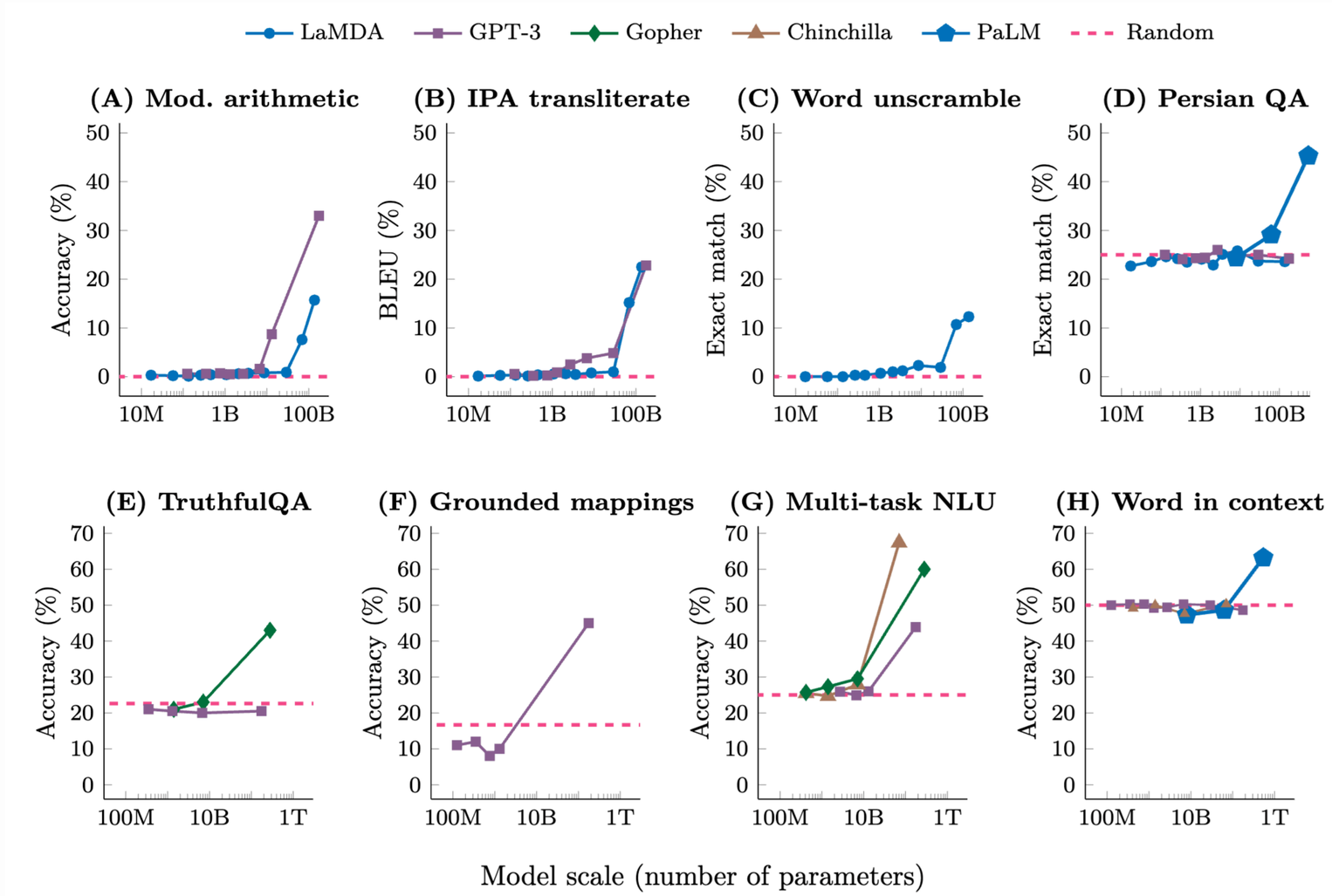
Number of samples generated until r is passed



Can be unexpectedly good

Emergent Abilities of Large Language Models

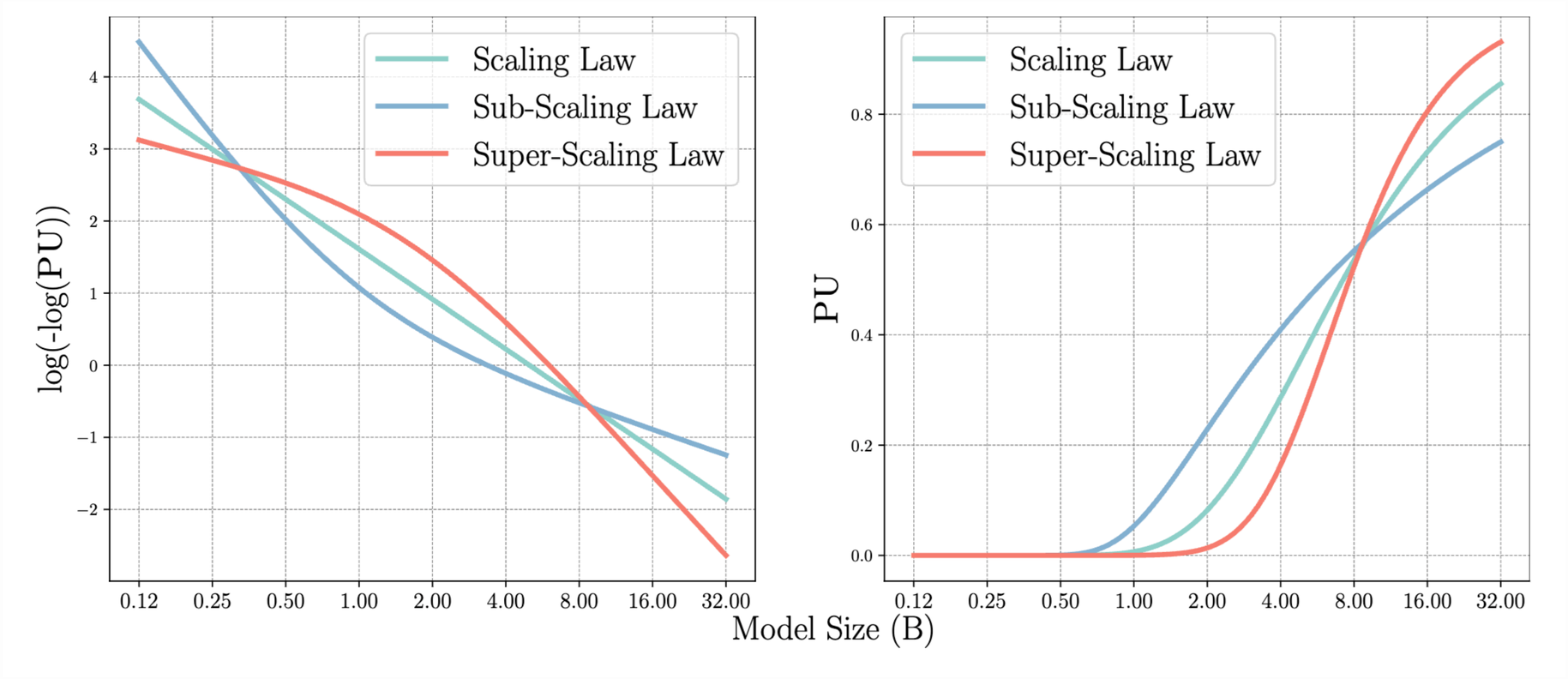
Emergent abilities



Can be unexpectedly good

Predicting Emergent Abilities with Infinite Resolution Evaluation

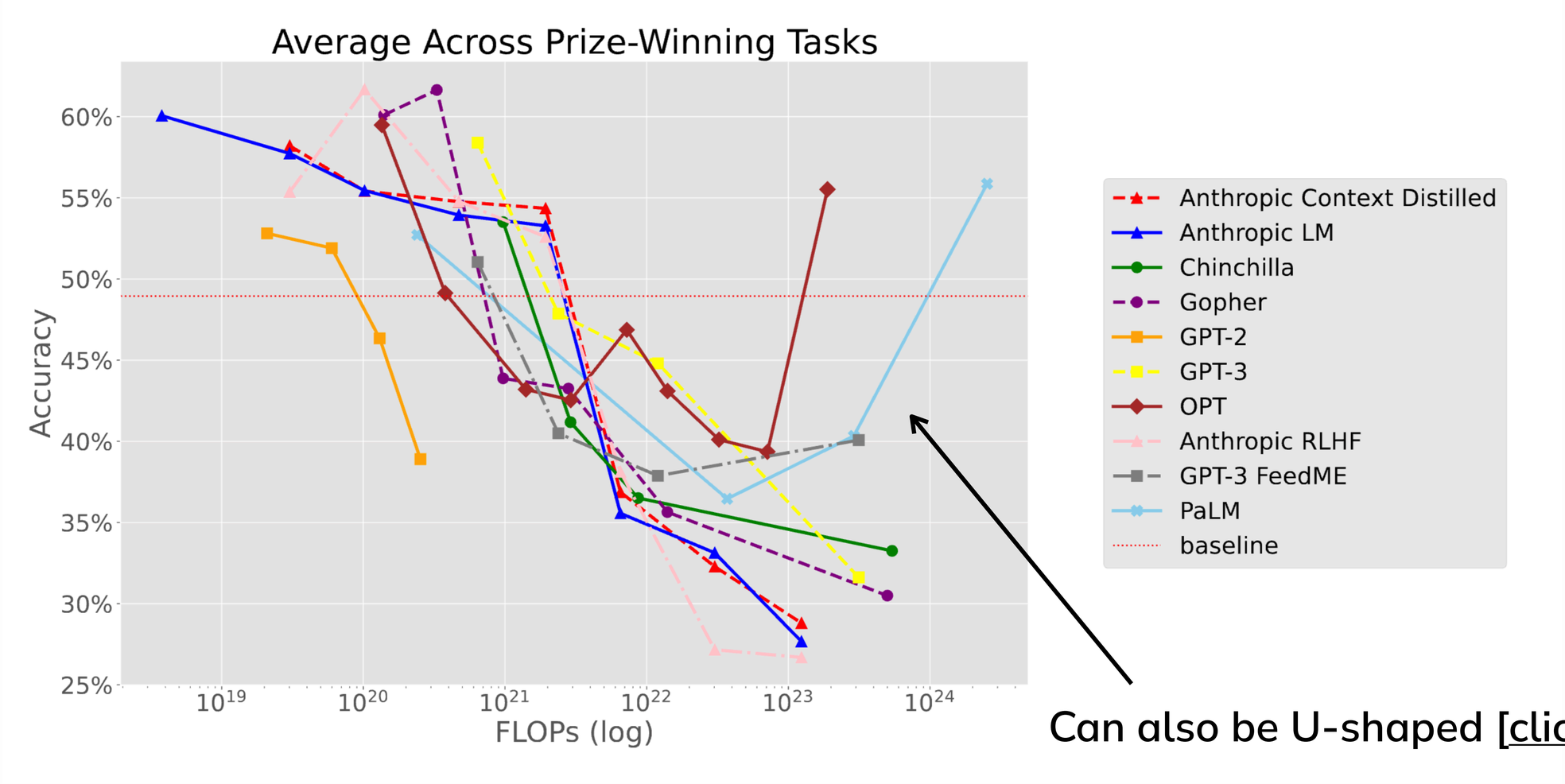
Emergent abilities



Also unexpectedly bad

Inverse Scaling: When Bigger Isn't Better

Inverse scaling



Can also be U-shaped [click]

Some theory to indulge

- A Dynamical Model of Neural Scaling Laws
- Explaining Neural Scaling Laws
- A Neural Scaling Law from the Dimension of the Data Manifold
- How Feature Learning Can Improve Neural Scaling Laws

Summary

- **Understand why you need it**
  - Predictability, optimality, confidence, … ?
- **Develop good methodology**
  - Will your results be useful for others?
- **Don’t draw immediate conclusions**
  - Scaling laws are not always "linear"

Epilogue

Big summary

- **Model scaling**
  - Capacity to learn
- **Data scaling**
  - Knowledge & skills to be learnt
- **Scaling laws**
  - Guidance for learning

Should I scale?

- **Do it**
  - Right
  - Explore your limits
- **Doubt it**
  - DYOR
  - Wisdom is a moving target
- **Don’t overthink it**
  - It’s ok to be suboptimal
  - 80/20

Confused?

- **Me too!**
  - (and that’s ok)
- **If you…**
  - consider scaling
  - or question the model "optimality"
  - or don’t want to tune HPs
  - or something made you interested
- **Just let me know** 😌
  - o.filatov@fz-juelich.de

Something more

Newton-Schulz iterations

Bernstein's blog

Step 4: Dualizing fast via Newton-Schulz

The final step in the derivation is to work out a fast means of performing the gradient orthogonalization step:

$$\nabla_W \mathcal{L} = U\Sigma V^\top \mapsto UV^\top.$$

We proposed performing this step via a so-called "Newton-Schulz" iteration. The main idea underlying the technique is that odd matrix polynomials $p(X)$ "commute" with the SVD. In other words, for an odd polynomial

$$p(X) := a \cdot X + b \cdot XX^\top X + c \cdot XX^\top XX^\top X + \cdots,$$

applying p to the matrix X is equivalent to applying p directly to the singular values of X and then putting the singular vectors back together:

$$p(X) = p(U\Sigma V^\top) = Up(\Sigma)V^\top.$$

To orthogonalize a matrix, we would like to set all the singular values to one. Since all the singular values are positive, we can think of this as applying the *sign function* to the singular values. While the sign function is an odd function, it is unfortunately not a polynomial. However, we can produce low-order odd polynomials that converge to the sign function when iterated. A simple example (Kovarik, 1970) is the cubic polynomial:

$$p_3(\Sigma) := \frac{3}{2} \cdot \Sigma - \frac{1}{2} \cdot \Sigma\Sigma^\top\Sigma.$$

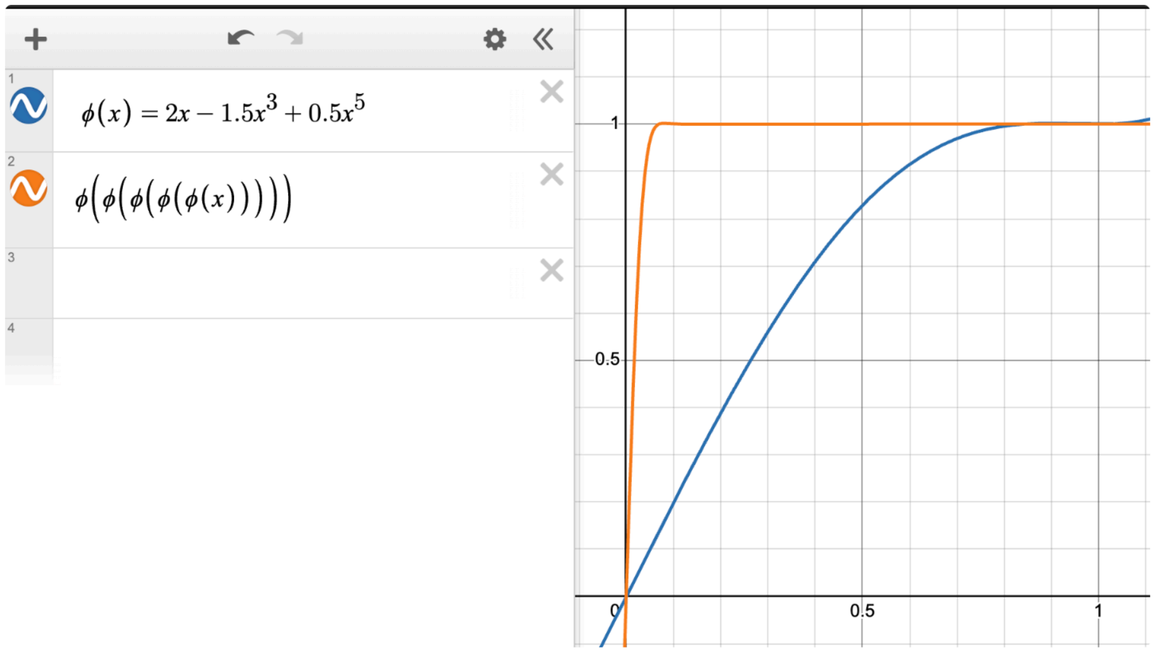


Figure 3. Baseline coefficients for Newton-Schulz iteration.