LLM Pretraining

short presentation

Andrei Semenov

MLO Group Meeting, 07.05.2025

Intro

Optimizers..., hyperparameters...

MUON IS SCALABLE FOR LLM TRAINING

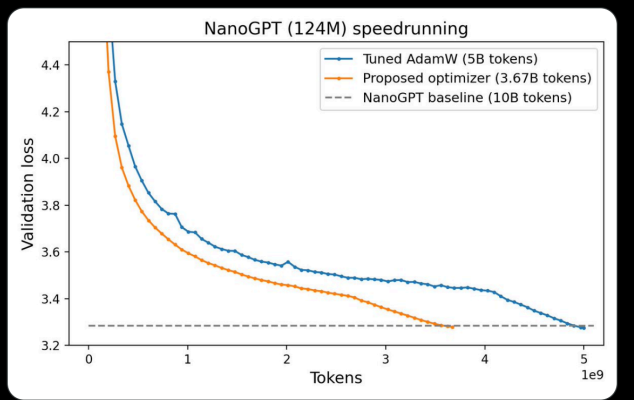
TECHNICAL REPORT

Post

Keller Jordan
@kellerjordan0

New training speed record for @karpathy's 124M-parameter NanoGPT setup: 3.28 Fineweb validation loss in 3.7B training tokens

Previous record: 5B tokens
Changelog: new optimizer
1/8

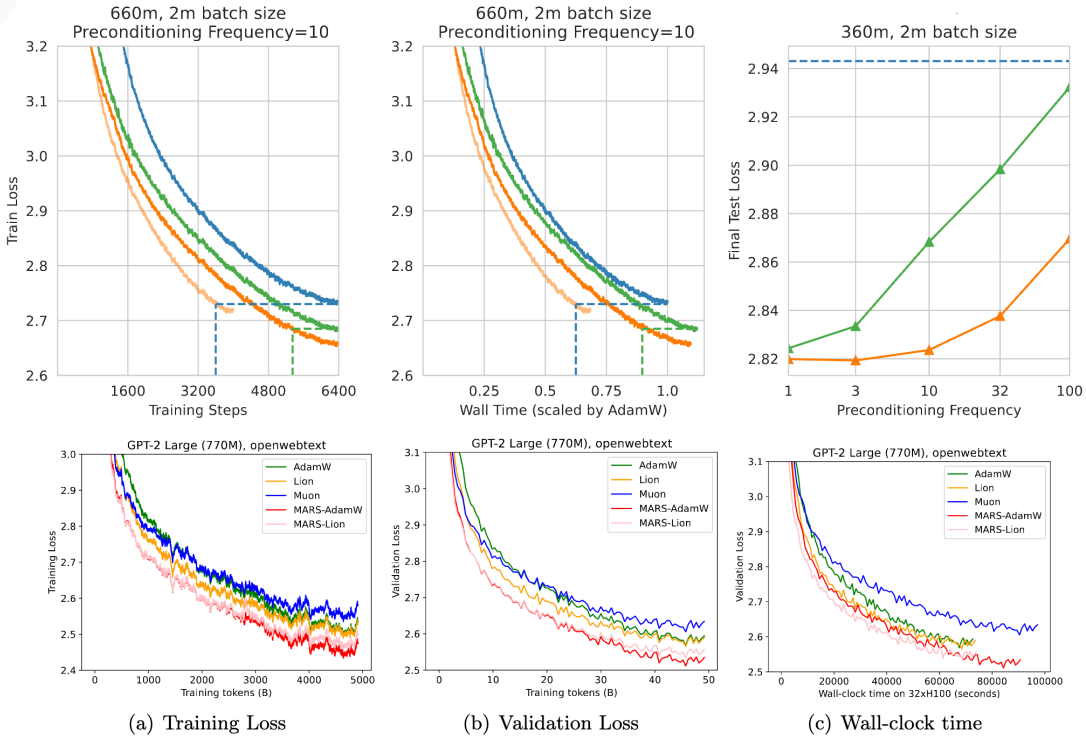


10:29 PM · Oct 4, 2024 · **252.6K** Views

Dion: A Communication-Efficient Optimizer for Large Models

Kwangjun Ahn Byron Xu
Microsoft Research

BETTER, FASTER, OLDER



(a) Training Loss (b) Validation Loss (c) Wall-clock time

Figure 1: The training and validation loss curves, plotted against both training tokens and wall-clock time on GPT-2 large model (770M).

ADOPT: Modified Adam Can Converge with Any β_2 with the Optimal Rate

So we benchmarked

Algorithm 1 `AdamW`

Input: Initial parameter
Initialize: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$, $\boldsymbol{v}_0$
for $t \in [T]$ **do**
  $\boldsymbol{g}_t \leftarrow \nabla \mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1$
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 -$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$,
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \gamma_t \left(\frac{r}{\sqrt{\hat{v}}} \right.$
end for
Return: $\boldsymbol{x}_T$

Algorithm 2 `ADOPT`

Input: Initial parameters $\boldsymbol{x}_0$, nu
Initialize: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$, $\boldsymbol{v}_0 \leftarrow \nabla \mathcal{L}($
for $t \in [T]$ **do**
  $\boldsymbol{g}_t \leftarrow \nabla \mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$
  $c_t \leftarrow t^{1/4}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)$cl
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2) \boldsymbol{g}_t \odot$
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \gamma_t \left(\boldsymbol{m}_t + \lambda \boldsymbol{x}_t \right)$
end for
Return: $\boldsymbol{x}_T$

Algorithm 3 `AdEMAMix`

Input: Initial parameters $\boldsymbol{x}_0$, number of iterations T, learning rate γ_t, weight decay λ, β_1, β_2, β_3, β_{start}, α, `beta_scheduler`, `alpha_scheduler`, warmup parameters T_{β_3} and T_α, ε.
Initialize: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$, $\boldsymbol{m}_0^{\text{slow}} \leftarrow \boldsymbol{0}$, $\boldsymbol{v}_0 \leftarrow \boldsymbol{0}$
for $t \in [T]$ **do**
  $\beta_3(t) \leftarrow \texttt{beta\_scheduler}(t, \beta_3, \beta_{\text{start}}, T_{\beta_3})$, $\alpha(t) \leftarrow \texttt{alpha\_scheduler}(t, \alpha, T_\alpha)$ ▷ Update β_3 and α schedulers
  $\boldsymbol{g}_t \leftarrow \nabla \mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \boldsymbol{g}_t$
  $\boldsymbol{m}_t^{\text{slow}} \leftarrow \beta_3(t) \boldsymbol{m}_{t-1}^{\text{slow}} + (1 - \beta_3(t)) \boldsymbol{g}_t$ ▷ Update slow EMA
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2) \boldsymbol{g}_t \odot \boldsymbol{g}_t$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$, $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \gamma_t \left(\frac{\hat{\boldsymbol{m}}_t + \alpha(t) \boldsymbol{m}_t^{\text{slow}}}{\sqrt{\hat{\boldsymbol{v}}_t} + \varepsilon} + \lambda \boldsymbol{x}_t \right)$
end for
Return: $\boldsymbol{x}_T$

Algorithm 4 `Lion`

Algorithm 6 `Signum` (our PyTorch variant)

Input: Initial parameters $\boldsymbol{x}_0$, number of iterations T, learning rate γ_t, weight decay λ, momentum β.
Initialize: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$
for $t \in [T]$ **do**
  $\boldsymbol{g}_t \leftarrow \mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$
  $\boldsymbol{m}_t \leftarrow \beta \boldsymbol{m}_{t-1} + \boldsymbol{g}_t$
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \gamma_t \left(\texttt{sign}\left(\beta \boldsymbol{m}_t + \boldsymbol{g}_t \right) + \lambda \boldsymbol{x}_t \right)$
end for
Return: $\boldsymbol{x}_T$

Algorithm 10 `SOAP`

Algorithm 11 `Sophia`

Algorithm 12 `SF-AdamW`

Algorithm 8 `Muon`

Algorithm 13 `Prodigy`

Algorithm 14 `MARS (MARS-AdamW)`

Optimizers: Adam-like

Algorithm 3 `AdEMAMix`

Input: Initial parameters $\boldsymbol{x}_0$, number of iterations T, learning rate γ_t, weight decay λ, β_1, β_2, β_3, β_{start}, α, `beta_scheduler`, `alpha_scheduler`, warmup parameters T_{β_3} and T_α, ε.
Initialize: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$, $\boldsymbol{m}_0^{\text{slow}} \leftarrow \boldsymbol{0}$, $\boldsymbol{v}_0 \leftarrow \boldsymbol{0}$
for $t \in [T]$ **do**
  $\beta_3(t) \leftarrow \texttt{beta\_scheduler}(t, \beta_3, \beta_{\text{start}}, T_{\beta_3})$, $\alpha(t) \leftarrow \texttt{alpha\_scheduler}(t, \alpha, T_\alpha)$ ▷ Update β_3 and α schedulers
  $\boldsymbol{g}_t \leftarrow \nabla \mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$
  $\boldsymbol{m}_t^{\text{slow}} \leftarrow \beta_3(t)\boldsymbol{m}_{t-1}^{\text{slow}} + (1-\beta_3(t))\boldsymbol{g}_t$ ▷ Update slow EMA
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1-\beta_1^t)$, $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1-\beta_2^t)$
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \gamma_t \left(\frac{\hat{\boldsymbol{m}}_t + \alpha(t)\boldsymbol{m}_t^{\text{slow}}}{\sqrt{\hat{\boldsymbol{v}}_t} + \varepsilon} + \lambda \boldsymbol{x}_t \right)$
end for
Return: $\boldsymbol{x}_T$

Algorithm 2 `ADOPT`

Input: Initial parameters $\boldsymbol{x}_0$, number of iterations T, learning rate γ_t, weight decay λ, β_1, β_2, ε.
Initialize: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$, $\boldsymbol{v}_0 \leftarrow \nabla \mathcal{L}(\boldsymbol{x}_0, \boldsymbol{\xi}_0) \odot \nabla \mathcal{L}(\boldsymbol{x}_0, \boldsymbol{\xi}_0)$
for $t \in [T]$ **do**
  $\boldsymbol{g}_t \leftarrow \nabla \mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$
  $c_t \leftarrow t^{1/4}$ ▷ Update clipping value schedule
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\texttt{clamp}\left(\frac{\boldsymbol{g}_t}{\max\{\sqrt{\boldsymbol{v}_{t-1}}, \varepsilon\}}, c_t \right)$
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t$
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \gamma_t (\boldsymbol{m}_t + \lambda \boldsymbol{x}_t)$ ▷ Update without $\boldsymbol{v}_t$
end for
Return: $\boldsymbol{x}_T$

Optimizers: "Sign-based"

Algorithm 4 `Lion`

Input: Initial parameters $\boldsymbol{x}_0$, number of iterations T, learning rate γ_t, weight decay λ, β_1, β_2.
Initialize: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$
for $t \in [T]$ **do**
  $\boldsymbol{g}_t \leftarrow \nabla \mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$
  $\boldsymbol{m}_t \leftarrow \beta_2 \boldsymbol{m}_{t-1} + (1-\beta_2)\boldsymbol{g}_t$ ▷ Update EMA of $\boldsymbol{g}_t$
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \gamma_t \left(\texttt{sign}\left(\beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t\right) + \lambda \boldsymbol{x}_t\right)$
end for
Return: $\boldsymbol{x}_T$

Algorithm 5 `Signum` (basic)

Input: Initial parameters $\boldsymbol{x}_0$, number of iterations T, learning rate γ_t, weight decay λ, momentum β.
Initialize: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$
for $t \in [T]$ **do**
  $\boldsymbol{g}_t \leftarrow \nabla \mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$
  $\boldsymbol{m}_t \leftarrow \beta \boldsymbol{m}_{t-1} + (1-\beta)\boldsymbol{g}_t$
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \gamma_t \left(\texttt{sign}\left(\boldsymbol{m}_t\right) + \lambda \boldsymbol{x}_t\right)$
end for
Return: $\boldsymbol{x}_T$

Algorithm 6 `Signum` (our PyTorch variant)

Input: Initial parameters $\boldsymbol{x}_0$, number of iterations T, learning rate γ_t, weight decay λ, momentum β.
Initialize: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$
for $t \in [T]$ **do**
  $\boldsymbol{g}_t \leftarrow \mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$
  $\boldsymbol{m}_t \leftarrow \beta \boldsymbol{m}_{t-1} + \boldsymbol{g}_t$
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \gamma_t \left(\texttt{sign}\left(\beta \boldsymbol{m}_t + \boldsymbol{g}_t\right) + \lambda \boldsymbol{x}_t\right)$
end for
Return: $\boldsymbol{x}_T$

Optimizers: Using matrix information

Algorithm 7 `MuonNon1D` (for non-1D parameters)

Input: Initial non-1D parameters $\boldsymbol{x}_0$, number of iterations T, learning rate γ_t, momentum β, number of Newton-Schulz iterations T_{NS}, a, b, c coefficients.
Initialize: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$
for $t \in [T]$ **do**
  $\boldsymbol{g}_t \leftarrow \nabla \mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$
  $\boldsymbol{m}_t \leftarrow \beta \boldsymbol{m}_{t-1} + \boldsymbol{g}_t$
  $\boldsymbol{g}_t \leftarrow \beta \boldsymbol{m}_t + \boldsymbol{g}_t$ ▷ Practical implementation of Nesterov momentum
  Set: $\boldsymbol{w}_0 \leftarrow \boldsymbol{g}_t / \|\boldsymbol{g}_t\|_F$
  for $n \in [T_{\text{NS}}]$ **do**
    $\boldsymbol{w}_{n+1} \leftarrow a\boldsymbol{w}_n + b\boldsymbol{w}_n\boldsymbol{w}_n^\top + c\left(\boldsymbol{w}_n\boldsymbol{w}_n^\top\right)^2 \boldsymbol{w}_n$ ▷ Newton-Schulz iteration
  end for
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \gamma_t \boldsymbol{w}_{T_{\text{NS}}}$
end for
Return: $\boldsymbol{x}_T$

Algorithm 8 `Muon` (general scheme)

Input: Initial parameters $\boldsymbol{x}_0$, number of iterations T. `Muon`'s parameters: learning rate $\gamma_t^{\texttt{M}}$, momentum β, number of Newton-Schulz iterations T_{NS}, a, b, c coefficients. `AdamW`'s parameters: learning rate $\gamma_t^{\texttt{A}}$, weight decay λ, β_1, β_2, ε.
for $t \in [T]$ **do**
  if $\boldsymbol{x}_t \in \{\texttt{embeds}, \texttt{scalar\_params}, \texttt{lm\_head}\}$ **then**
    $\boldsymbol{x}_t^{\texttt{A}} \leftarrow \boldsymbol{x}_t$
    $\boldsymbol{x}_{t+1}^{\texttt{A}} \leftarrow \texttt{AdamW}\left(\boldsymbol{x}_t^{\texttt{A}}, \gamma_t^{\texttt{A}}, \lambda, \beta_1, \beta_2, \varepsilon, T=1\right)$ ▷ One iteration of `AdamW`
  else
    $\boldsymbol{x}_t^{\texttt{M}} \leftarrow \boldsymbol{x}_t$
    $\boldsymbol{x}_{t+1}^{\texttt{M}} \leftarrow \texttt{MuonNon1D}\left(\boldsymbol{x}_t^{\texttt{M}}, \gamma_t^{\texttt{M}}, T_{\text{NS}}, \beta, a, b, c, T=1\right)$ ▷ One iteration of `MuonNon1D`
  end if
end for
Return: $\boldsymbol{x}_T^{\texttt{A}}, \boldsymbol{x}_T^{\texttt{M}}$

Algorithm 9 `SOAPNon1D` (for non-1D parameters)

Input: Initial parameters $\boldsymbol{x}_0$, number of iterations T, learning rate γ_t, weight decay λ, β_1, β_2, preconditioning frequency ϕ, ε.
Initialize: $\boldsymbol{m}_0 \leftarrow \boldsymbol{0}$, $\boldsymbol{v}_0 \leftarrow \boldsymbol{0}$
Initialize preconditioners: $\boldsymbol{q}_l, \boldsymbol{q}_r \leftarrow \texttt{eigenbasis}\left(\nabla\mathcal{L}(\boldsymbol{x}_0, \boldsymbol{\xi}_0)\nabla\mathcal{L}(\boldsymbol{x}_0, \boldsymbol{\xi}_0)^\top\right)$
for $t \in [T]$ **do**
  $\boldsymbol{g}_t \leftarrow \nabla\mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$
  $\boldsymbol{g}_t' \leftarrow \boldsymbol{q}_l^\top \boldsymbol{g}_t \boldsymbol{q}_r$ ▷ Rotate $\boldsymbol{g}_t$
  $\boldsymbol{m} \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$
  $\boldsymbol{m}_t' \leftarrow \boldsymbol{q}_l^\top \boldsymbol{m}_t \boldsymbol{q}_r$ ▷ Compute `Adam`'s statistics in rotational space
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t' \odot \boldsymbol{g}_t'$
  $\gamma_t \leftarrow \gamma_t \frac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}$ ▷ Optional: use bias correction
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \gamma_t \left(\boldsymbol{q}_l \frac{\boldsymbol{m}_t'}{\sqrt{\boldsymbol{v}_t}+\varepsilon} \boldsymbol{q}_r^\top + \lambda \boldsymbol{x}_t\right)$ ▷ Perform update in original space
  $\boldsymbol{l}_t \leftarrow \beta_2 \boldsymbol{l}_{t-1} + (1-\beta_2)\boldsymbol{g}_t\boldsymbol{g}_t^\top$ ▷ Update preconditioners
  $\boldsymbol{r}_t \leftarrow \beta_2 \boldsymbol{r}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^\top\boldsymbol{g}_t$
  if $t \equiv 1 \pmod{\phi}$ **then**
    $\boldsymbol{q}_l \leftarrow \texttt{QR}\left(\boldsymbol{l}_t \boldsymbol{q}_l\right)$
    $\boldsymbol{q}_r \leftarrow \texttt{QR}\left(\boldsymbol{r}_t \boldsymbol{q}_r\right)$
  end if
end for
Return: $\boldsymbol{x}_T$

Optimizers: Parameter-free effort

Algorithm 13 `Prodigy`

Input: Initial parameters $\boldsymbol{x}_0$, number of iterations T, learning rate γ, weight decay λ, β_1, β_2, ε.
Initialize: $d_0 \leftarrow 10^{-6}$, $\gamma \leftarrow 1$, $\boldsymbol{m}_0 \leftarrow \mathbf{0}$, $\boldsymbol{v}_0 \leftarrow \mathbf{0}$, $r_0 \leftarrow 0$, $\boldsymbol{s}_0 \leftarrow \mathbf{0}$ ▷ Optional: use scheduler on γ
for $t \in [T]$ **do**
  $\boldsymbol{g}_t \leftarrow \nabla \mathcal{L}(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1) d_t \boldsymbol{g}_t$
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2) d_t^2 \boldsymbol{g}_t \odot \boldsymbol{g}_t$
  $\gamma_t \leftarrow \gamma \sqrt{1-\beta_2^t}/(1-\beta_1^t)$ ▷ Optional: use bias correction
  $r_t \leftarrow \sqrt{\beta_2} r_{t-1} + (1-\sqrt{\beta_2}) \gamma_t d_t^2 \langle \boldsymbol{g}_t, \boldsymbol{x}_0 - \boldsymbol{x}_t \rangle$
  $\boldsymbol{s}_t \leftarrow \sqrt{\beta_2} \boldsymbol{s}_{t-1} + (1-\sqrt{\beta_2}) \gamma_t d_t^2 \boldsymbol{g}_t$
  $d_{t+1} \leftarrow \max\{d_t, \frac{r_t}{\|\boldsymbol{s}_t\|_1}\}$
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \gamma_t d_t \left(\boldsymbol{m}_t / \left(\sqrt{\boldsymbol{v}_t} + d_t \varepsilon \right) + \lambda \boldsymbol{x}_t \right)$
end for
Return: $\boldsymbol{x}_T$

Algorithm 12 `SF-AdamW`

Input: Initial parameters $\boldsymbol{x}_0$, number of iterations T, learning rate γ, weight decay λ, β_1, β_2, warmup iterations T_{warmup}, ε.
Initialize: $\boldsymbol{z}_0 \leftarrow \boldsymbol{x}_0$, $\boldsymbol{v}_0 \leftarrow \mathbf{0}$
for $t \in [T]$ **do**
  $\boldsymbol{y}_t \leftarrow (1-\beta_1) \boldsymbol{z}_t + \beta_1 \boldsymbol{x}_t$
  $\boldsymbol{g}_t \leftarrow \nabla \mathcal{L}(\boldsymbol{y}_t, \boldsymbol{\xi}_t)$
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2) \boldsymbol{g}_t \odot \boldsymbol{g}_t$
  $\gamma_t \leftarrow \gamma \sqrt{1-\beta_2^t} \min\{1, t/T_{\text{warmup}}\}$
  $\boldsymbol{z}_{t+1} \leftarrow \boldsymbol{z}_t - \gamma_t \left(\boldsymbol{g}_t / (\sqrt{\boldsymbol{v}_t} + \varepsilon) + \lambda \boldsymbol{y}_t \right)$
  $c_{t+1} \leftarrow \frac{\gamma_t^2}{\sum_{i=0}^t \gamma_i^2}$
  $\boldsymbol{x}_{t+1} \leftarrow (1-c_{t+1}) \boldsymbol{x}_t + c_{t+1} \boldsymbol{z}_{t+1}$
end for
Return: $\boldsymbol{x}_T$

Setup

Number of steps? batch size? scheduler? hyperparameters?

Setup

Number of steps? batch size? scheduler? hyperparameters?

DeepMind

Training Compute-Optimal Large Language Models

databricks

All / Mosaic Research / How Long Should You Train Your Language Model?

How Long Should You Train Your Language Model?

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

Predictable Scale: Part I — Optimal Hyperparameter Scaling Law in Large Language Model Pretraining

November 22, 2024

How Does Critical Batch Size Scale in Pre-training? (Decoupling Data and Model Size)

Optimal Linear Decay Learning Rate Schedules and Further Refinements

Scaling Laws and Compute-Optimal Training Beyond Fixed Training Durations

The Surprising Agreement Between Convex Optimization Theory and Learning-Rate Scheduling for Large Model Training

Iterations & batch size

Table 2. Lengths of training for **Small batch settings** (32×512).

# Parameters	Tokens (Iterations)						Chinchilla Tokens
124M	1B (64k)	2.1B (128k)	4.2B (256k)	6.3B (384k)	8.4B (512k)	16.8B (1024k)	2.5B
210M	1B (64k)	2.1B (128k)	4.2B (256k)	6.3B (384k)	8.4B (512k)	16.8B (1024k)	4.2B

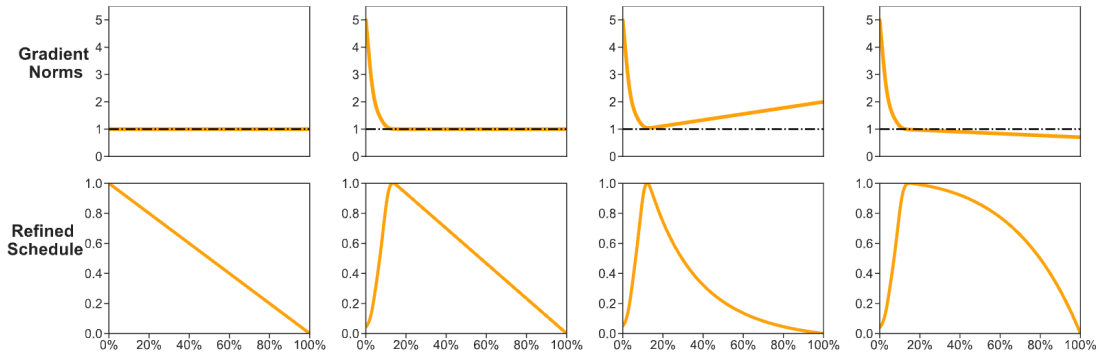
Table 3. Lengths of training for **Large batch settings** (256×512).

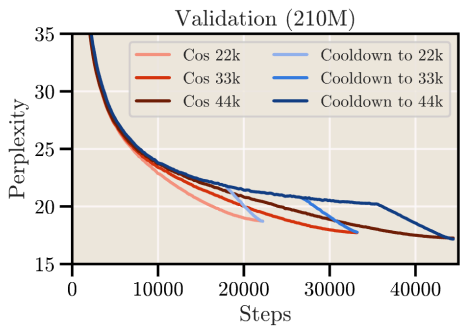
# Parameters	Tokens (Iterations)						Chinchilla Tokens
124M	1B (8k)	2.1B (16k)	4.2B (32k)	6.3B (48k)	8.4B (64k)	16.8B (128k)	2.5B
210M	1B (8k)	2.1B (16k)	4.2B (32k)	6.3B (48k)	8.4B (64k)	16.8B (128k)	4.2B

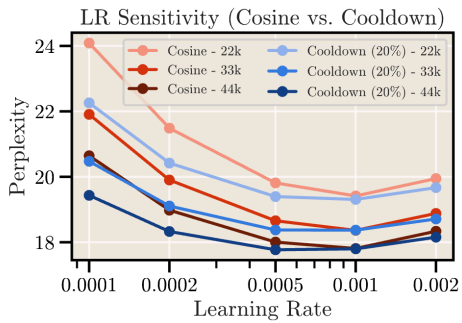
Table 4. Lengths of training for **X-Large batch settings** (1984×512).

# Parameters	Tokens (Iterations)			Chinchilla Tokens
720M	8B (8k)	16B (16k)	48B (48k)	14.4B

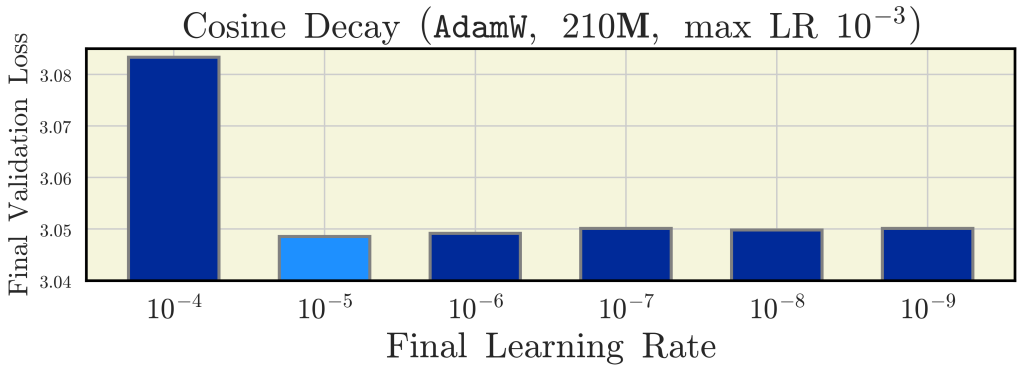
Scheduler

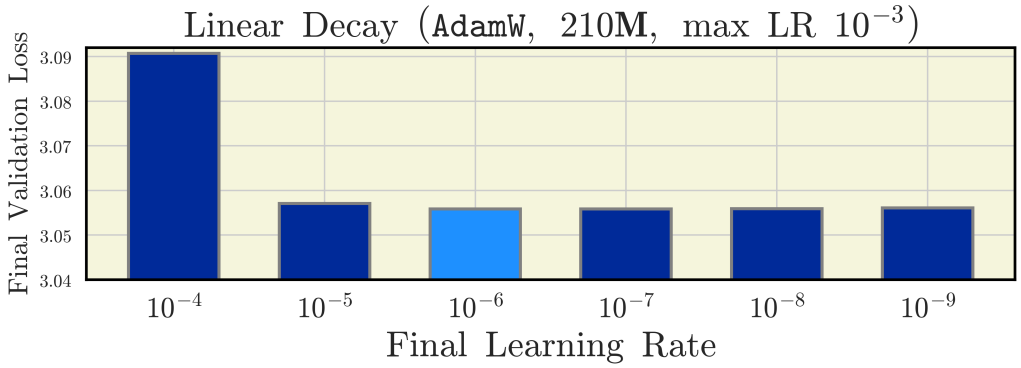

Gradient Norms
Refined Schedule
0%
20%
40%
60%
80%
100%

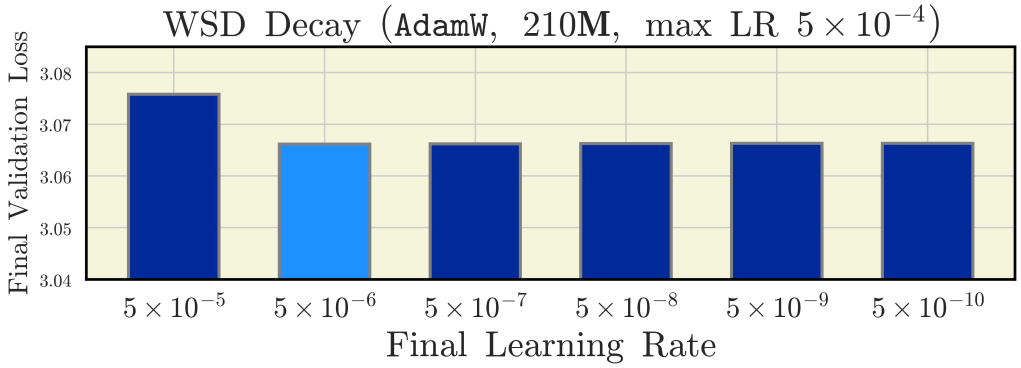

Validation (210M)
Cos 22k
Cos 33k
Cos 44k
Cooldown to 22k
Cooldown to 33k
Cooldown to 44k
Perplexity
Steps


LR Sensitivity (Cosine vs. Cooldown)
Cosine - 22k
Cosine - 33k
Cosine - 44k
Cooldown (20%) - 22k
Cooldown (20%) - 33k
Cooldown (20%) - 44k
Perplexity
Learning Rate

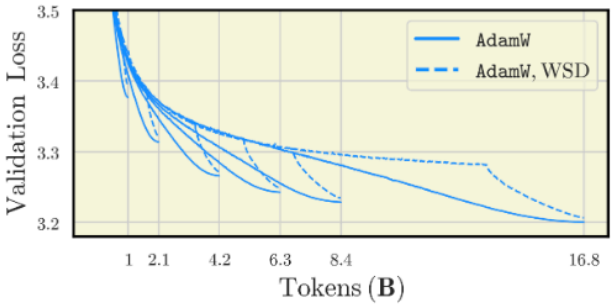
Learning rate decay: down to zero

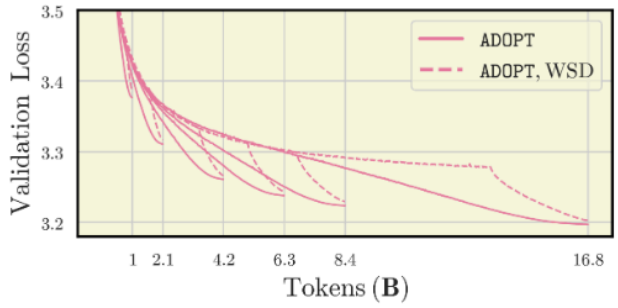


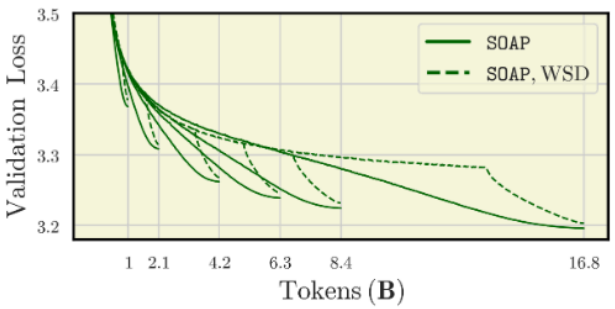


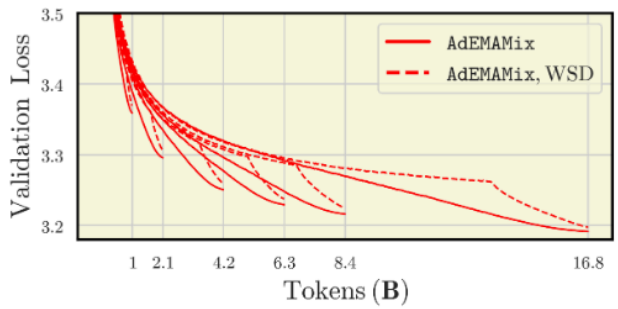


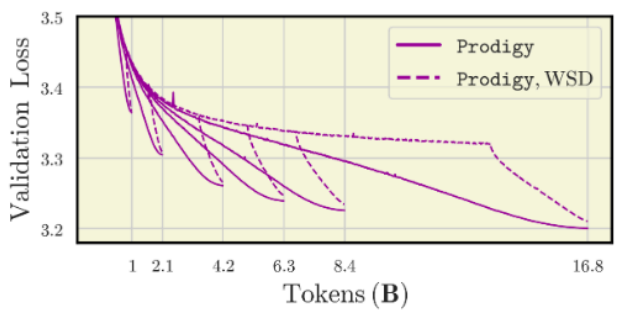
Benchmarking schedulers for different optimizers

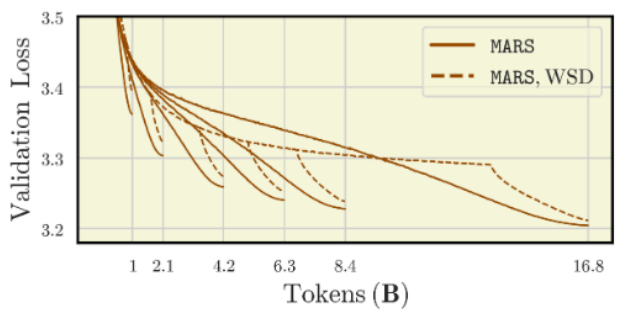




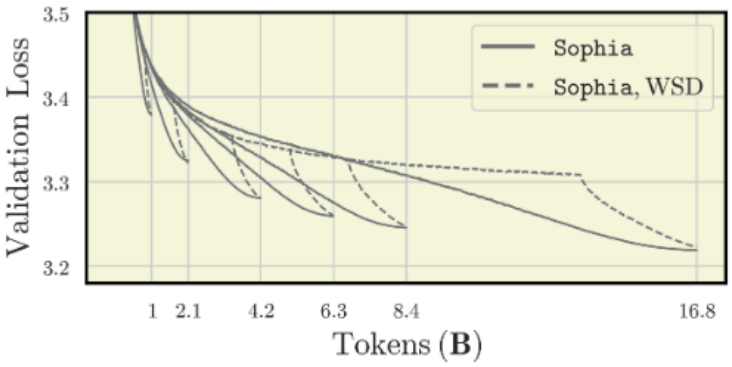


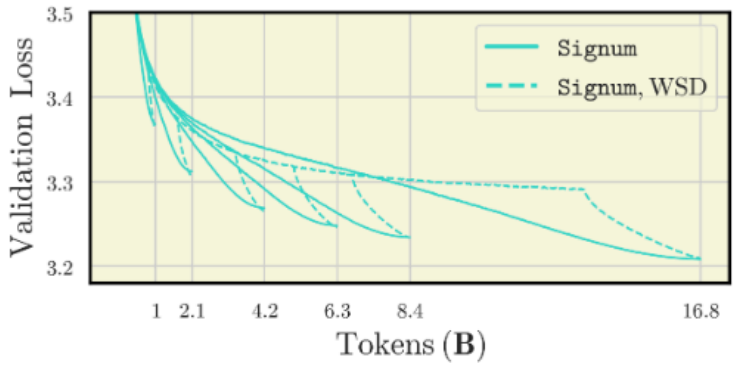


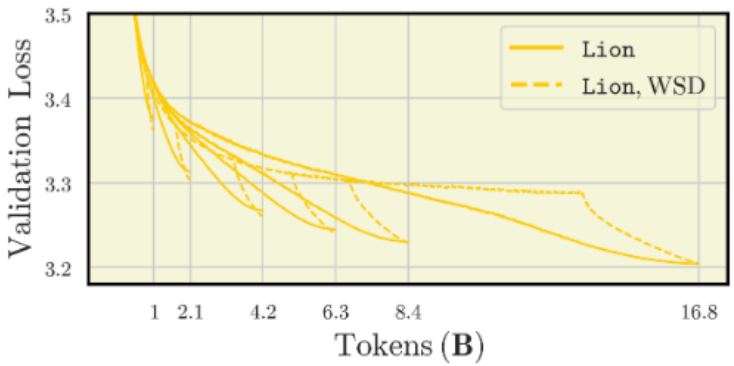


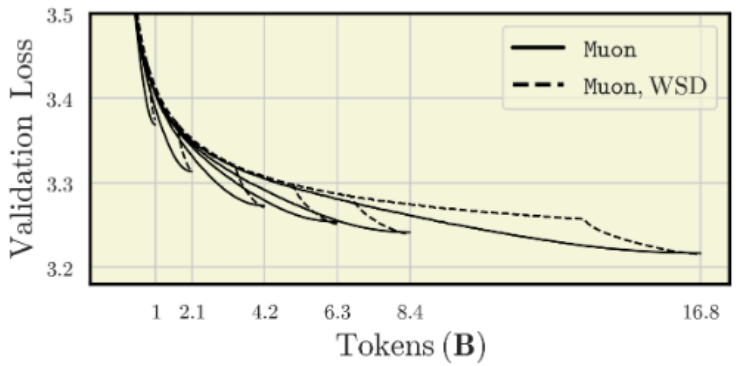


Benchmarking schedulers for different optimizers

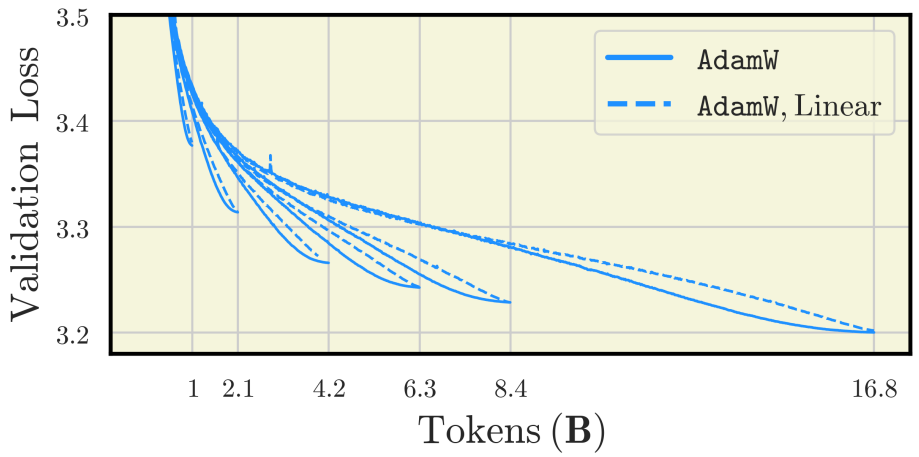


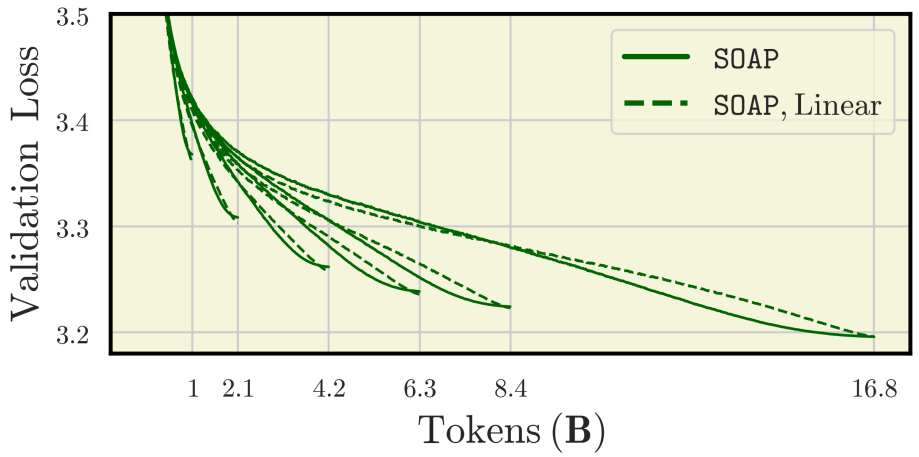




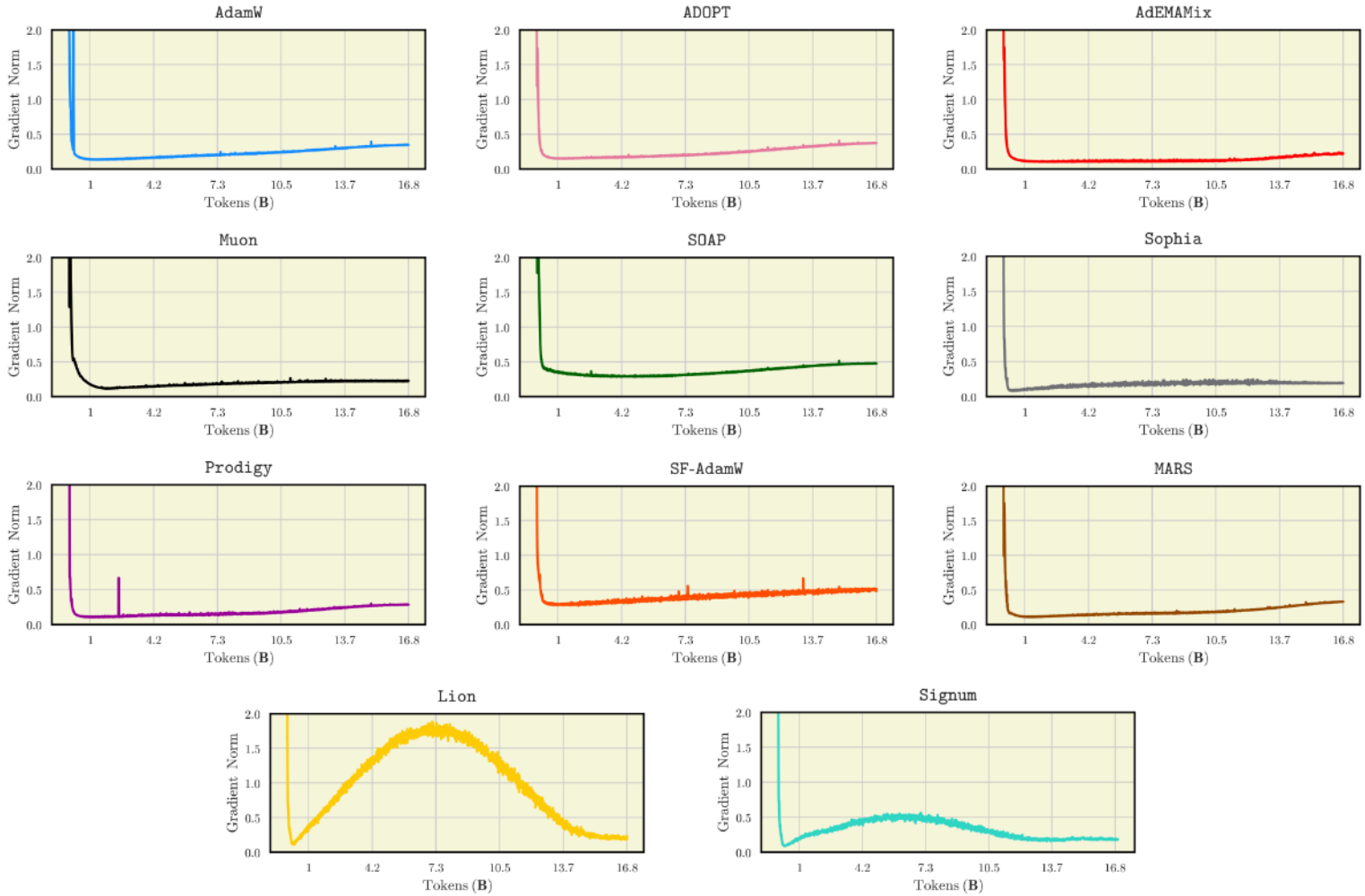


Benchmarking schedulers for different optimizers

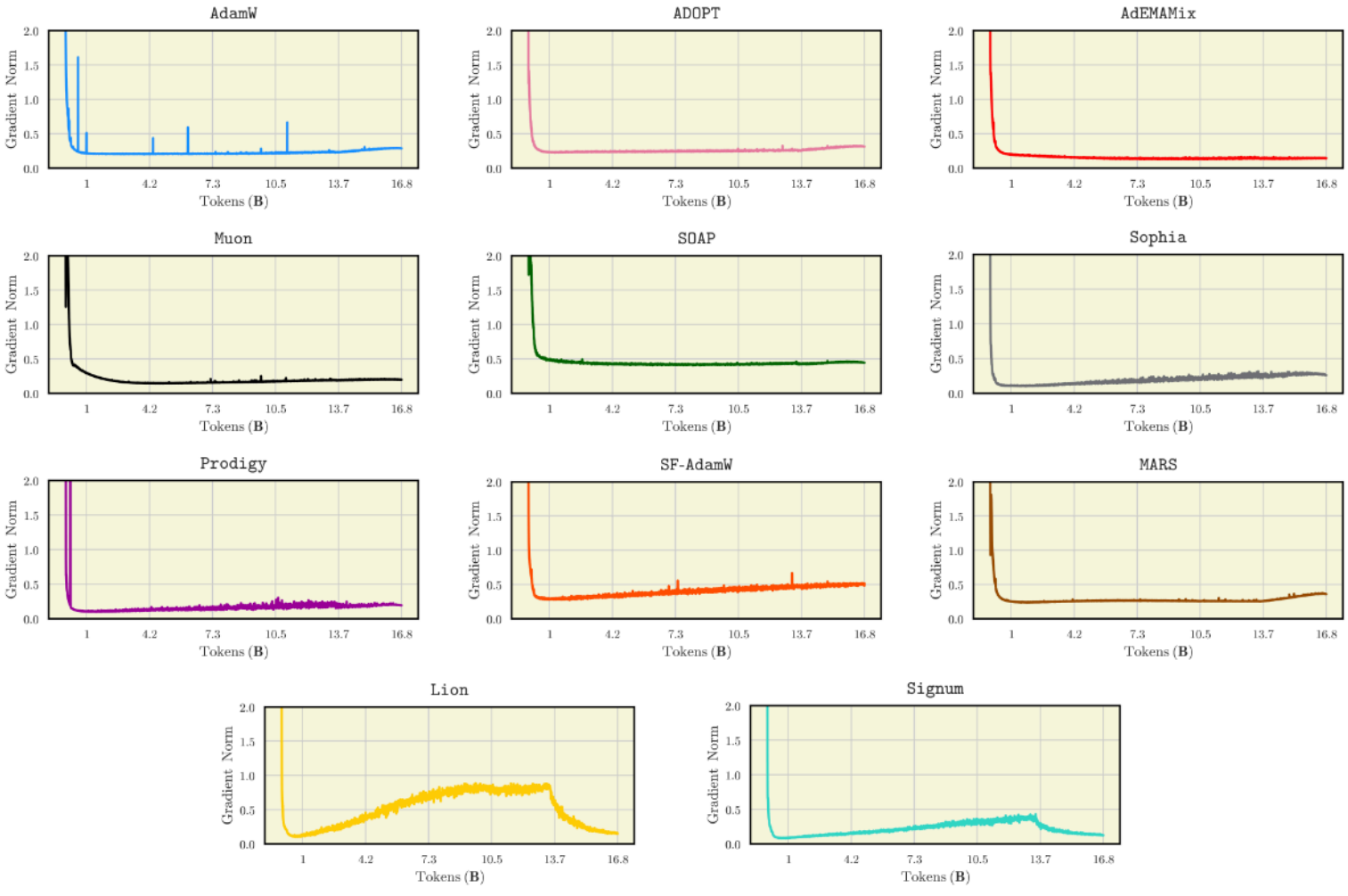




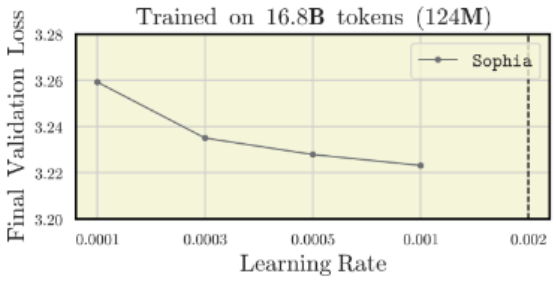
Scheduler: gradient norms perspective

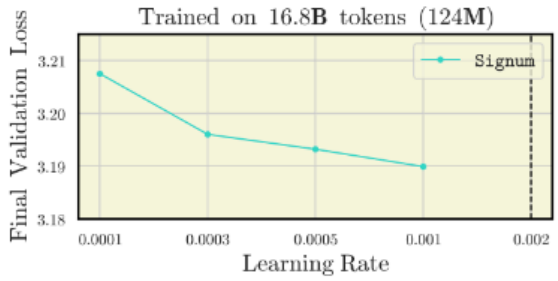


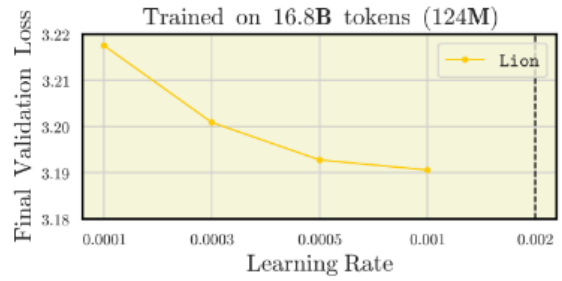
Scheduler: gradient norms perspective

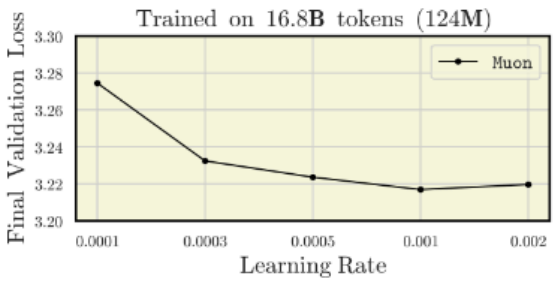


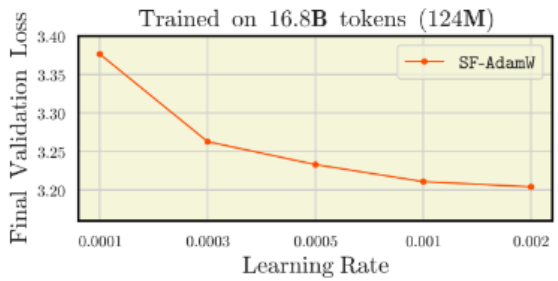
Hyperparameters: learning rate

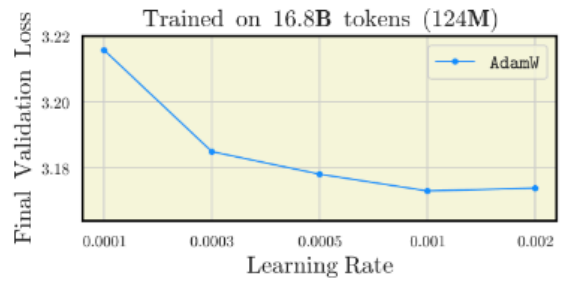


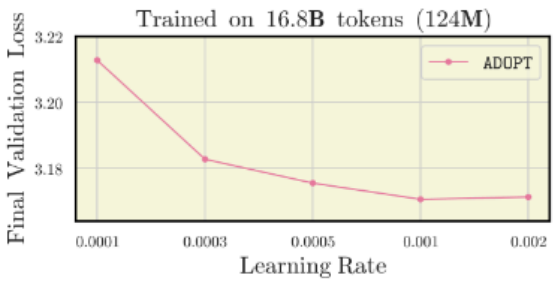


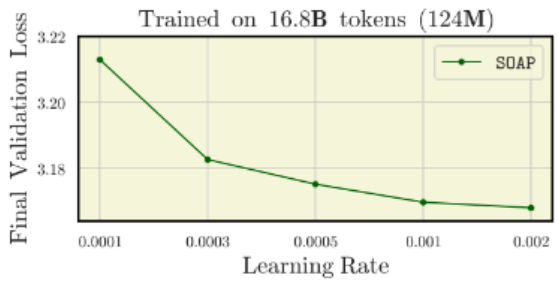


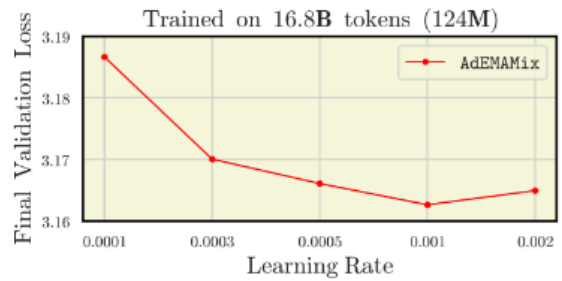




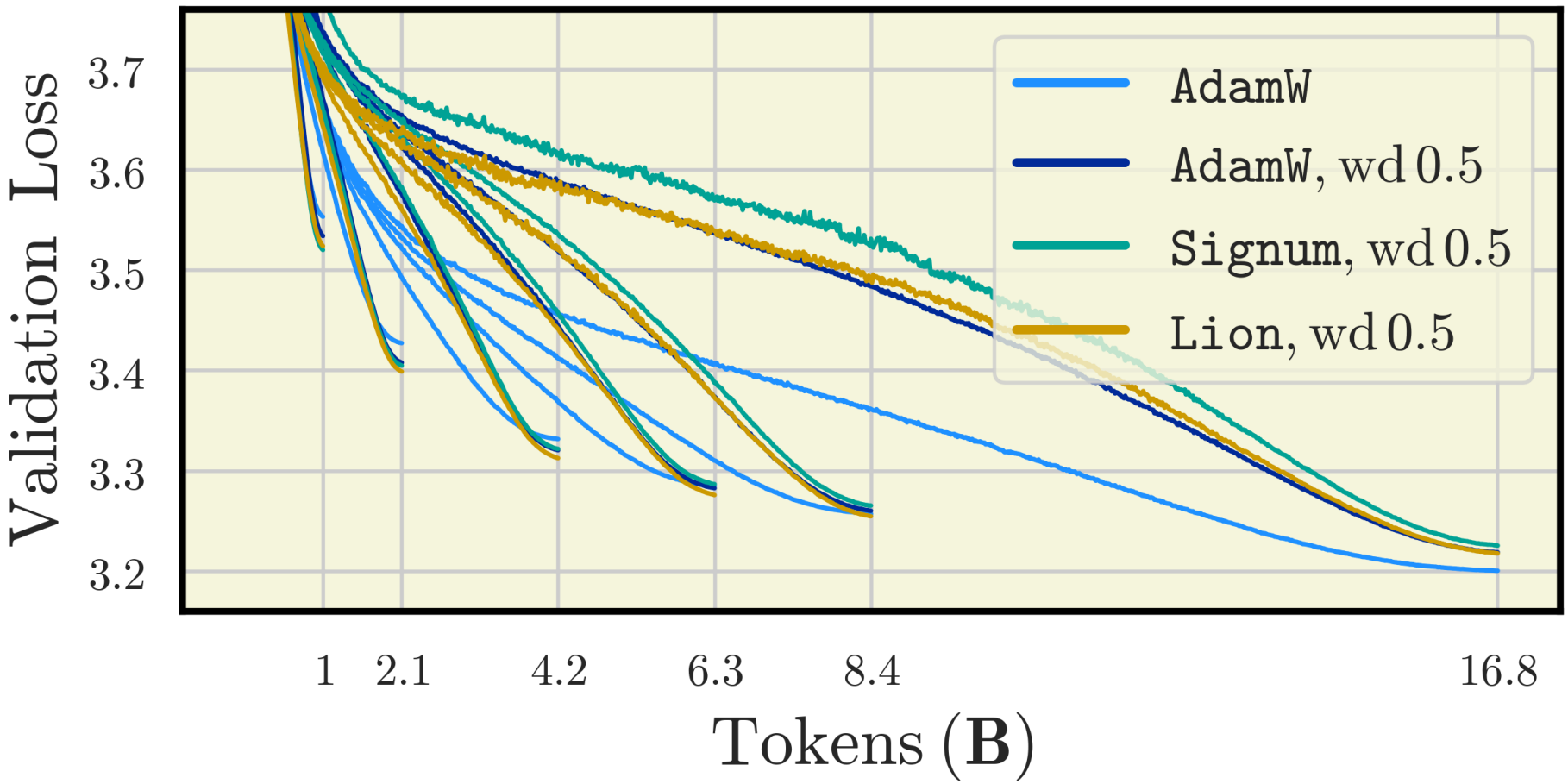




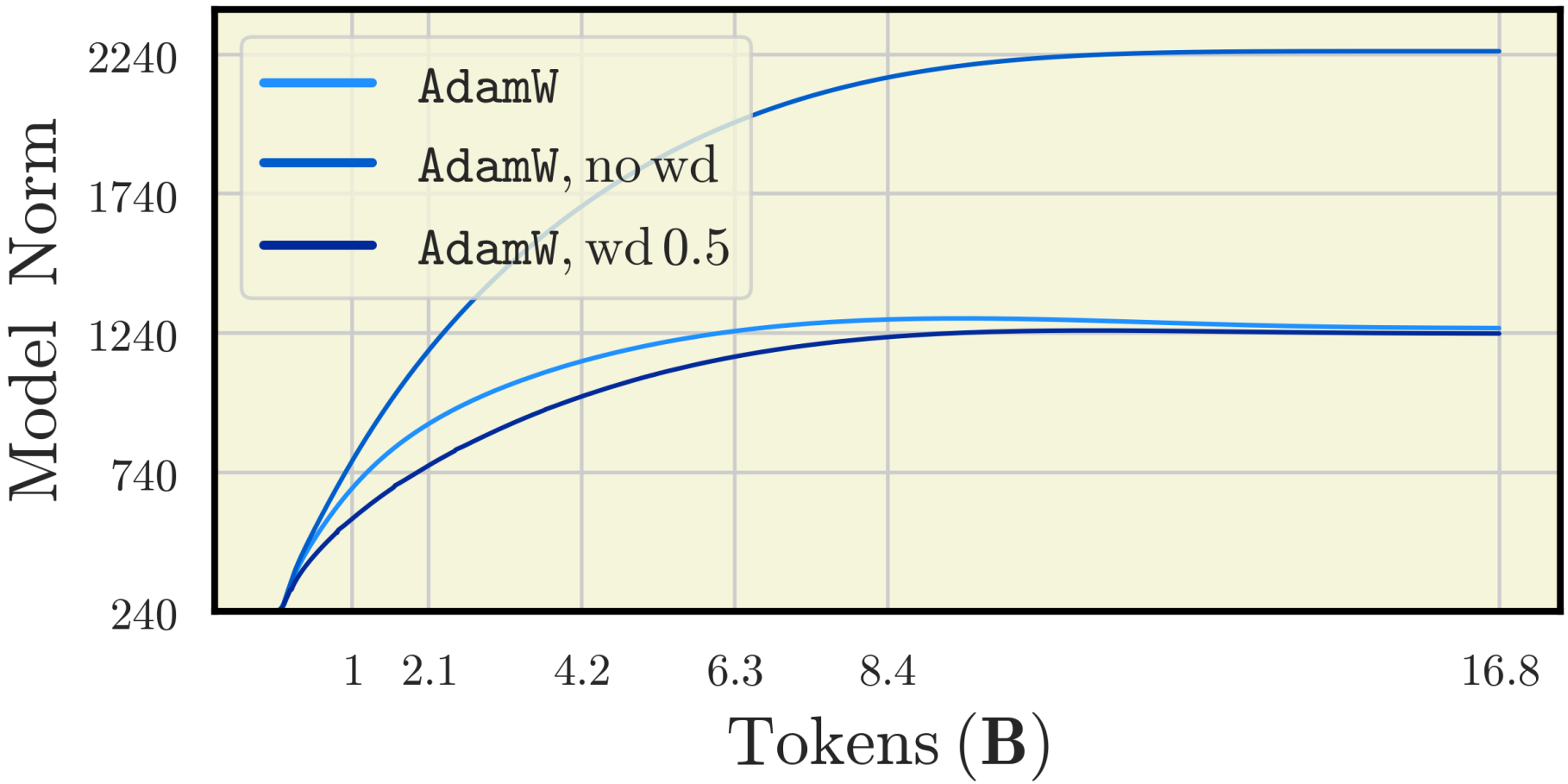




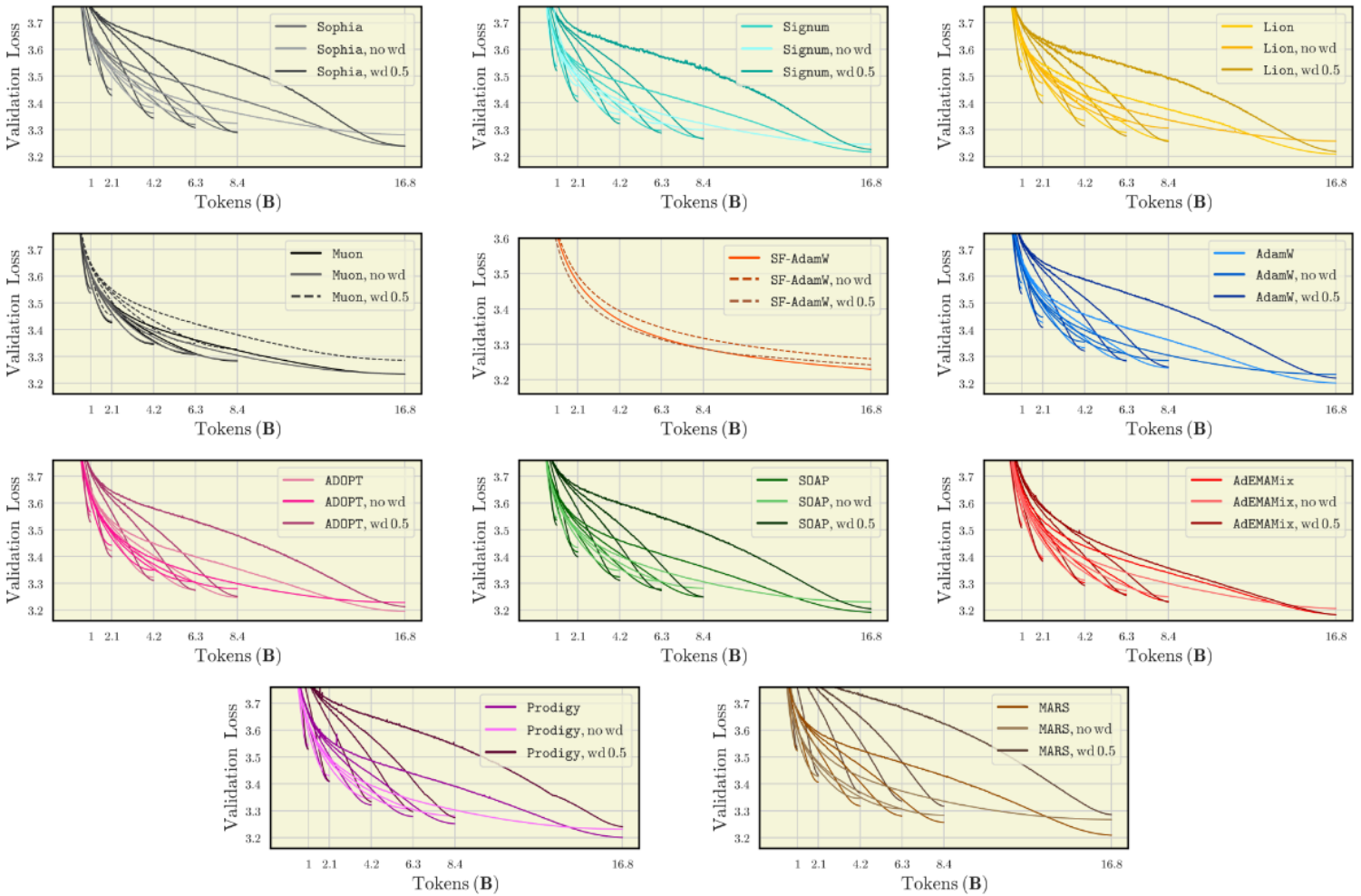
Hyperparameters: weight decay



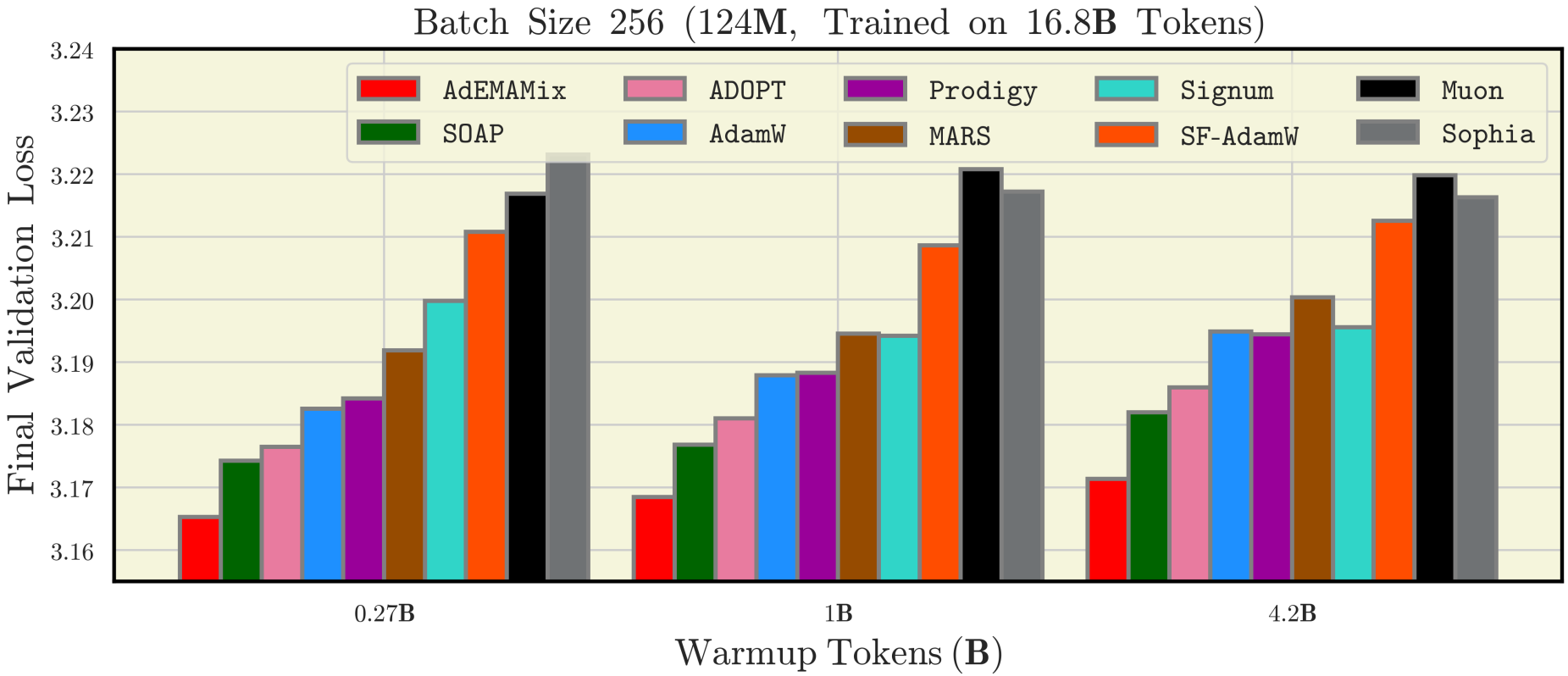
Hyperparameters: weight decay



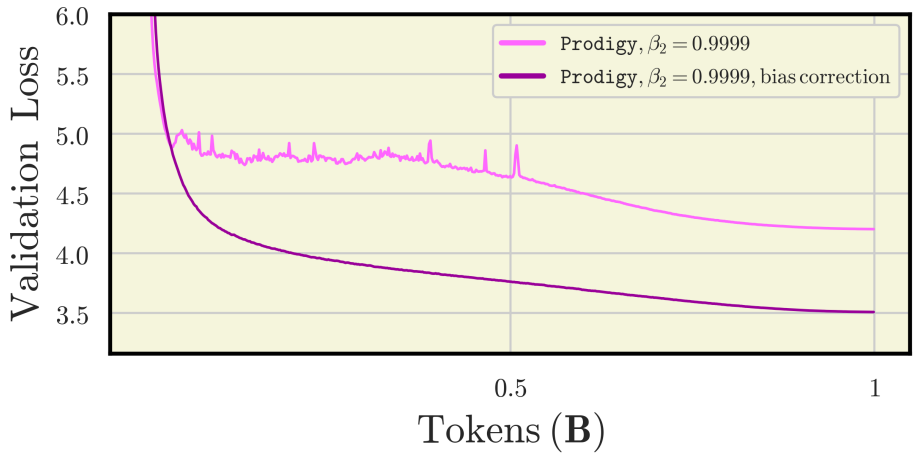
Hyperparameters: weight decay



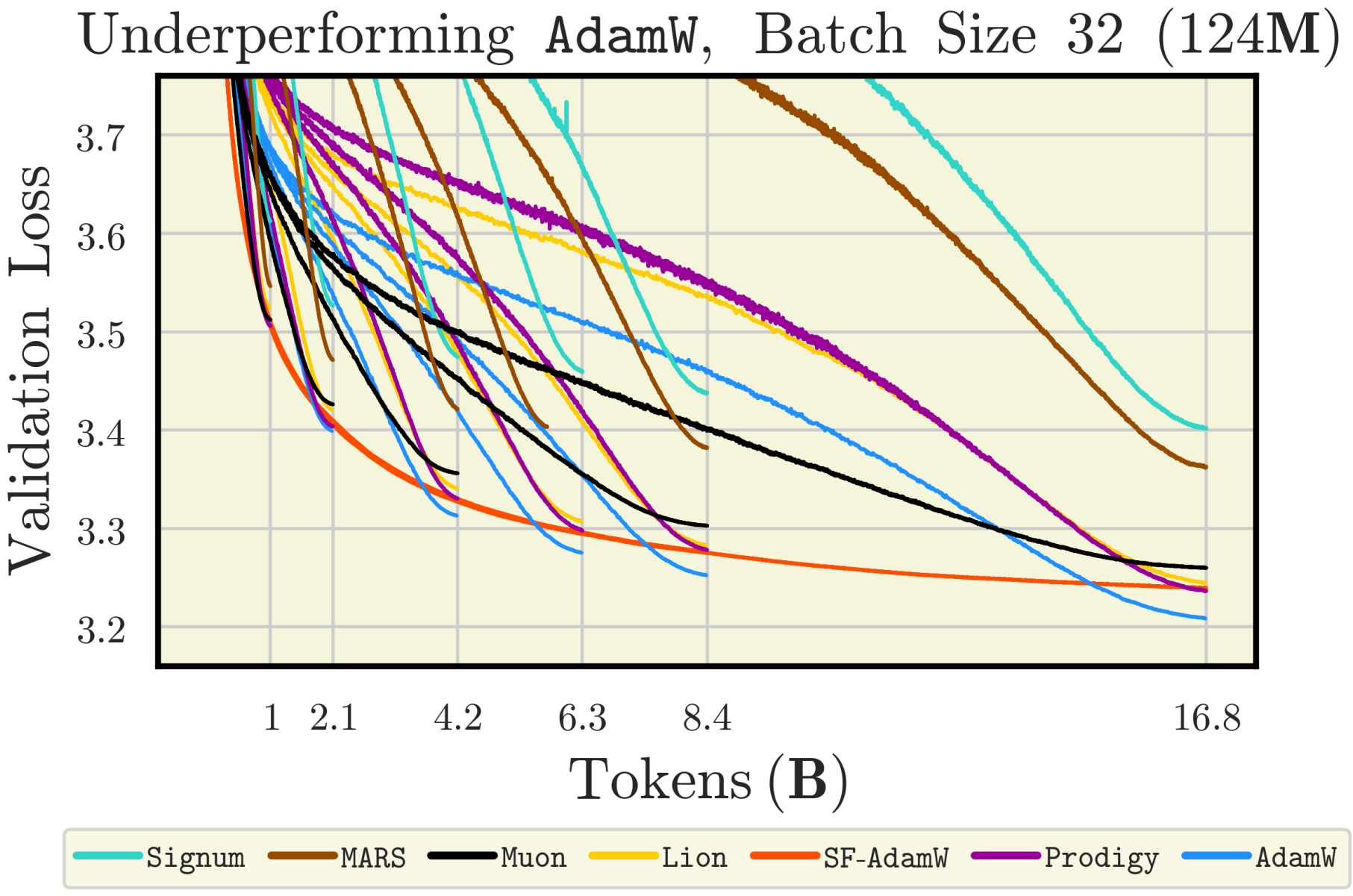
Hyperparameters: warmup



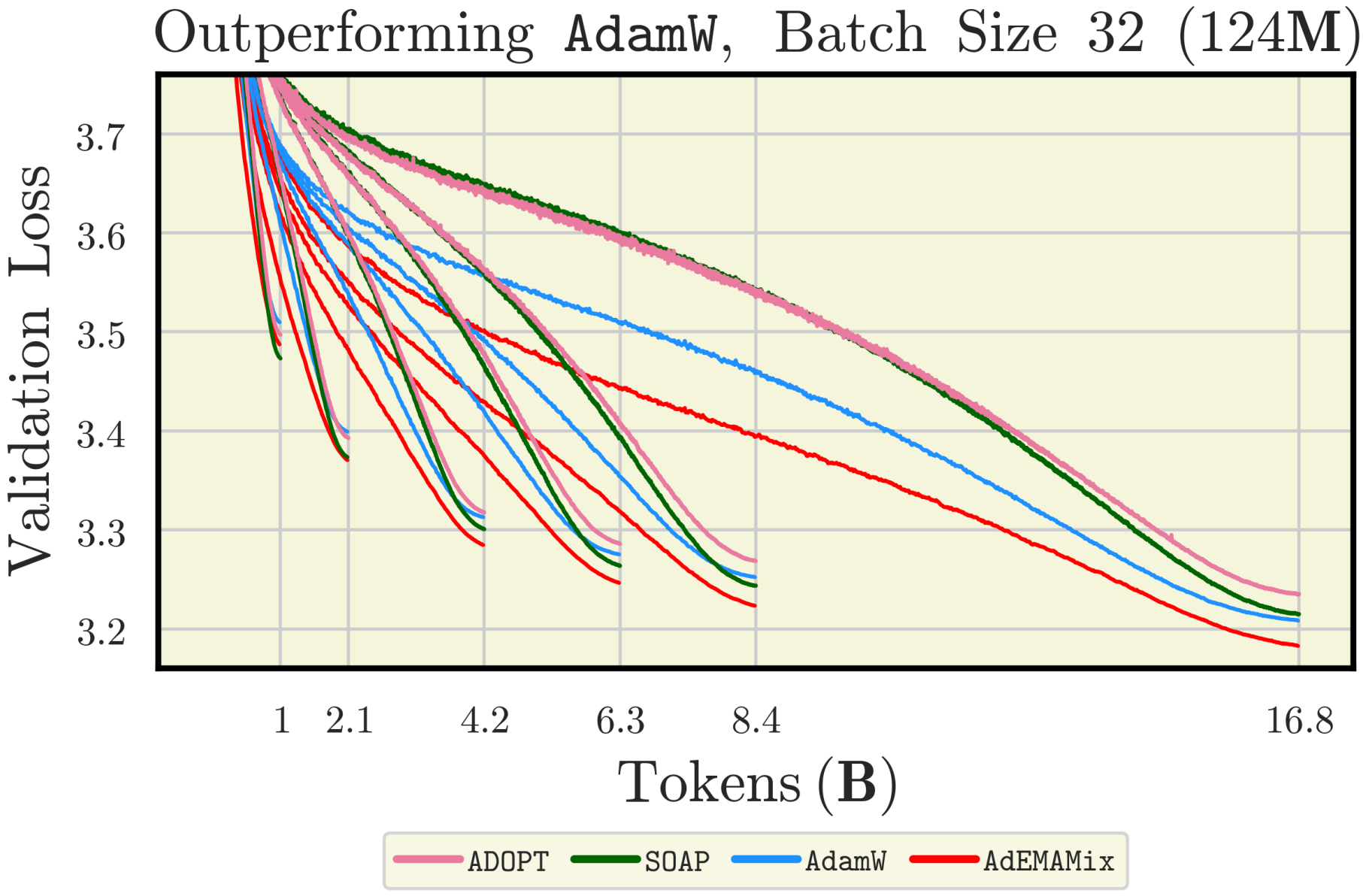
Hyperparameters: betas



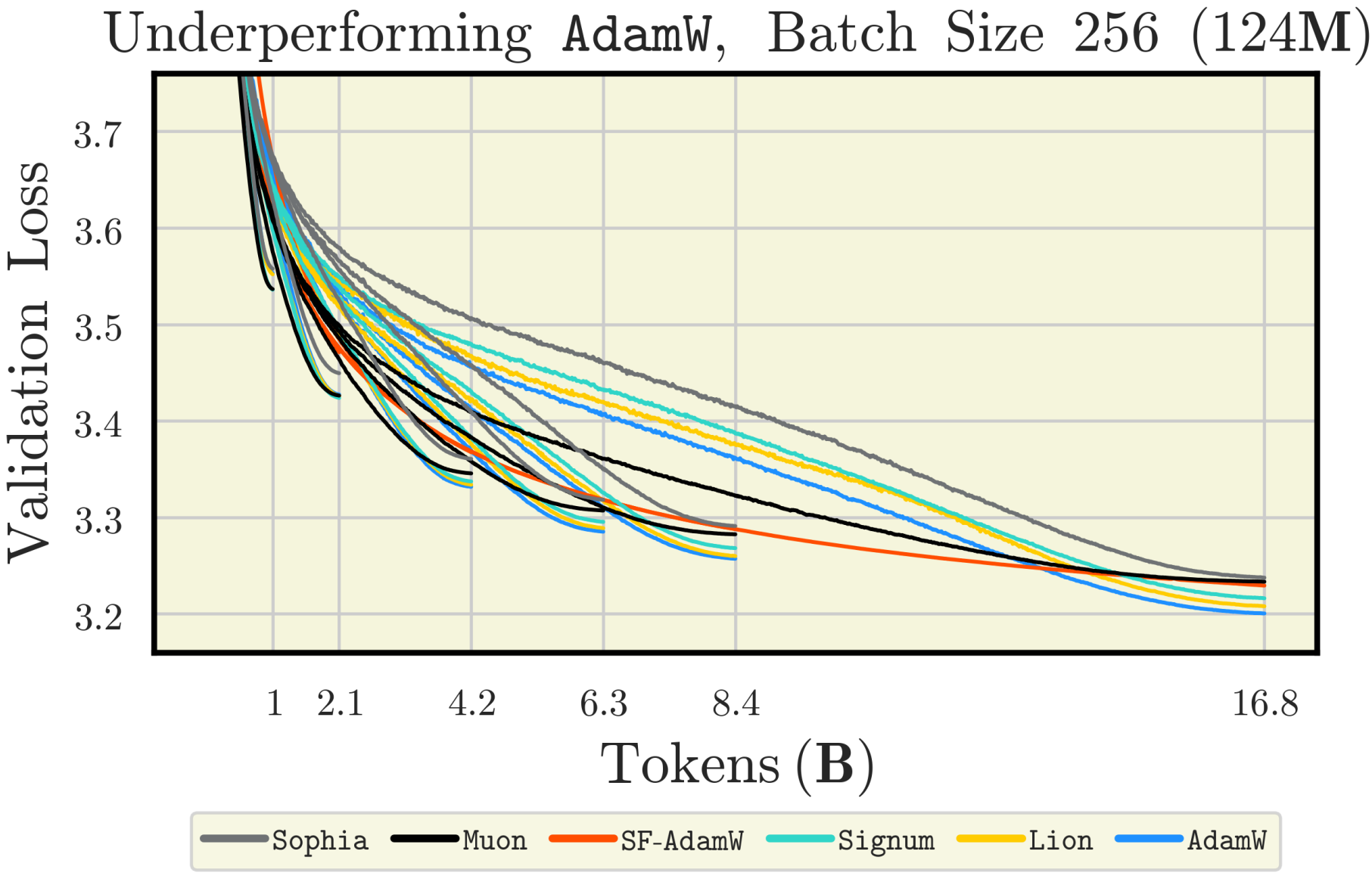
Benchmarking at small scale



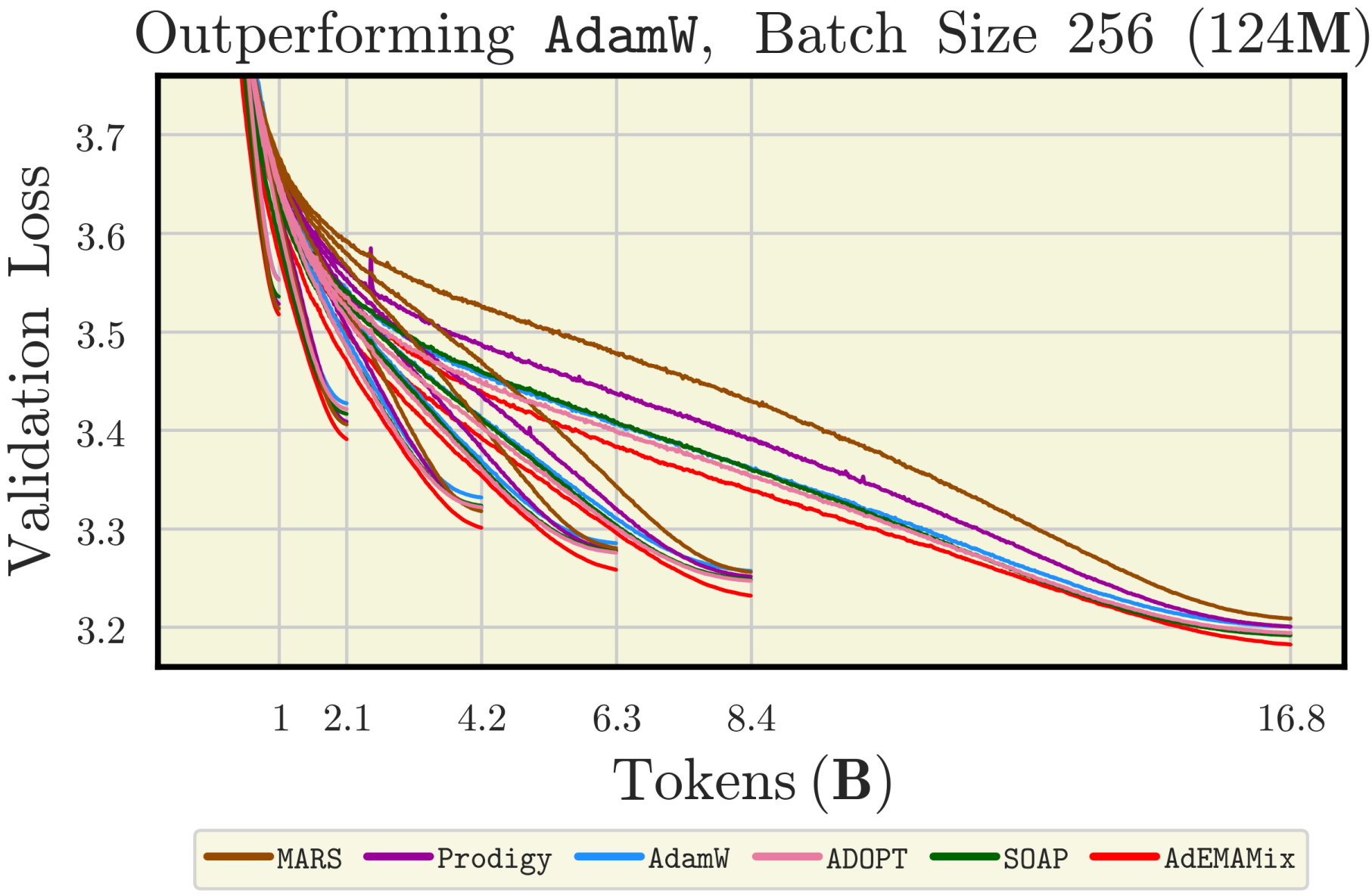
Benchmarking at small scale



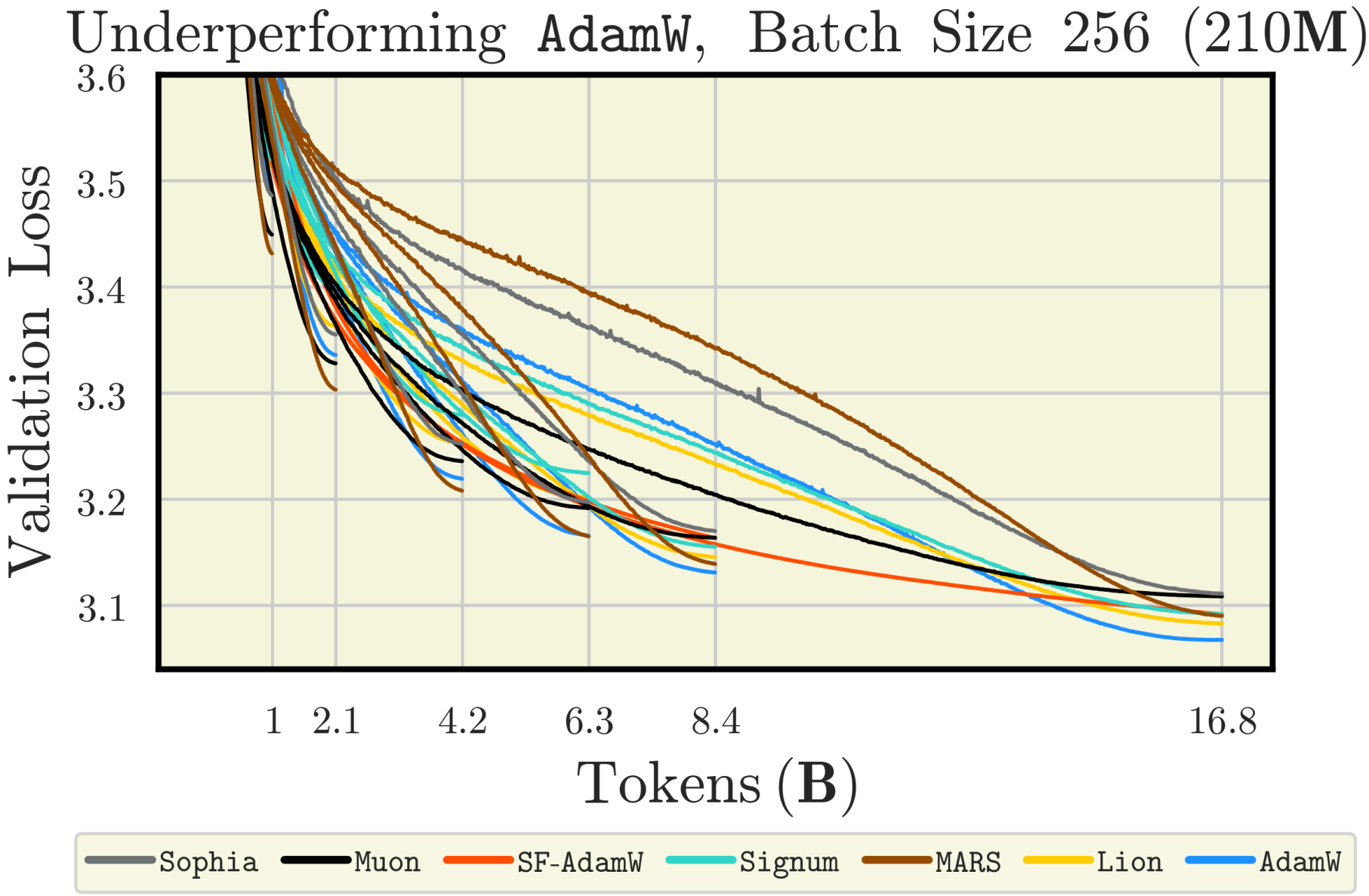
Benchmarking at small scale



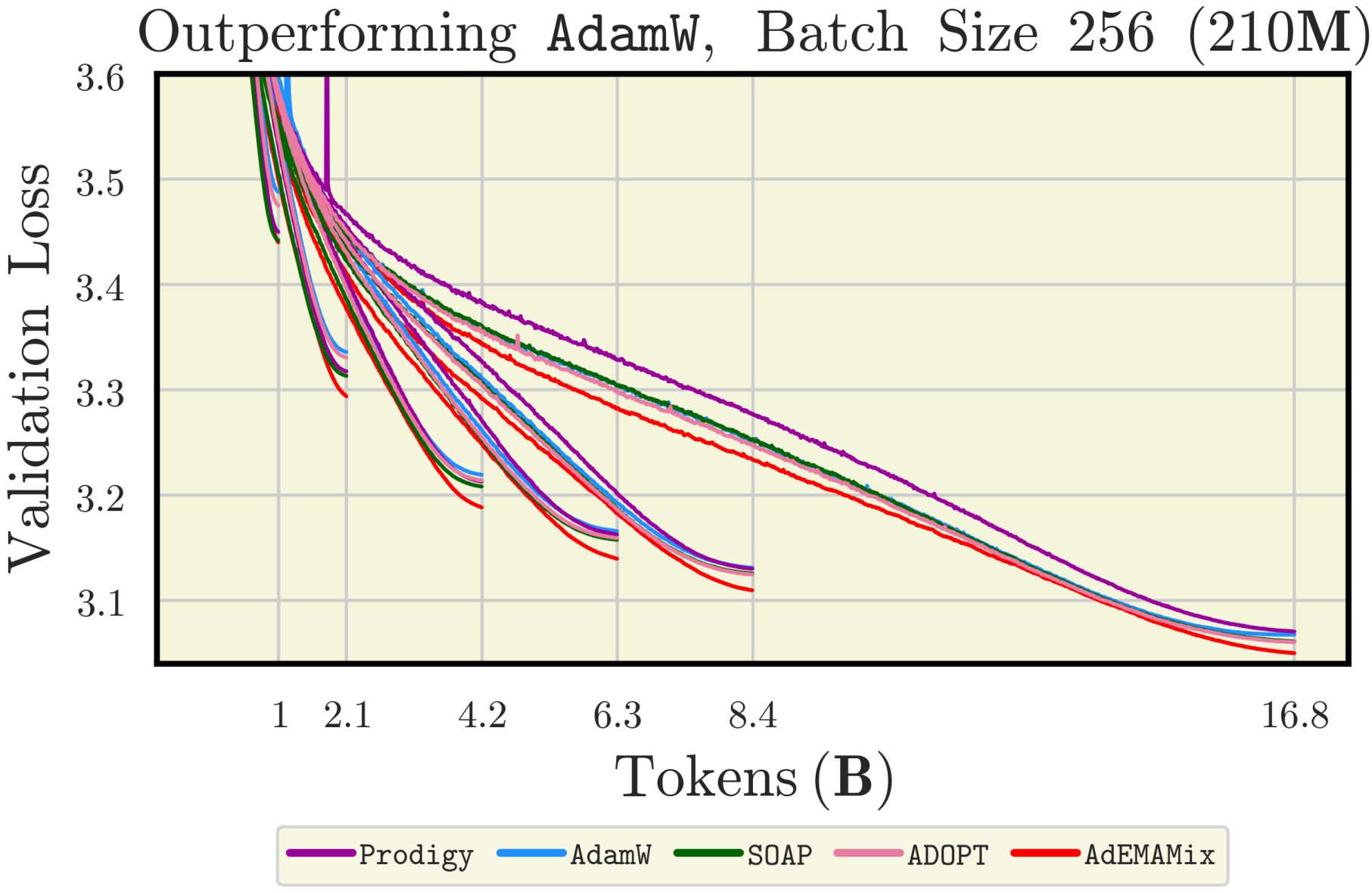
Benchmarking at small scale



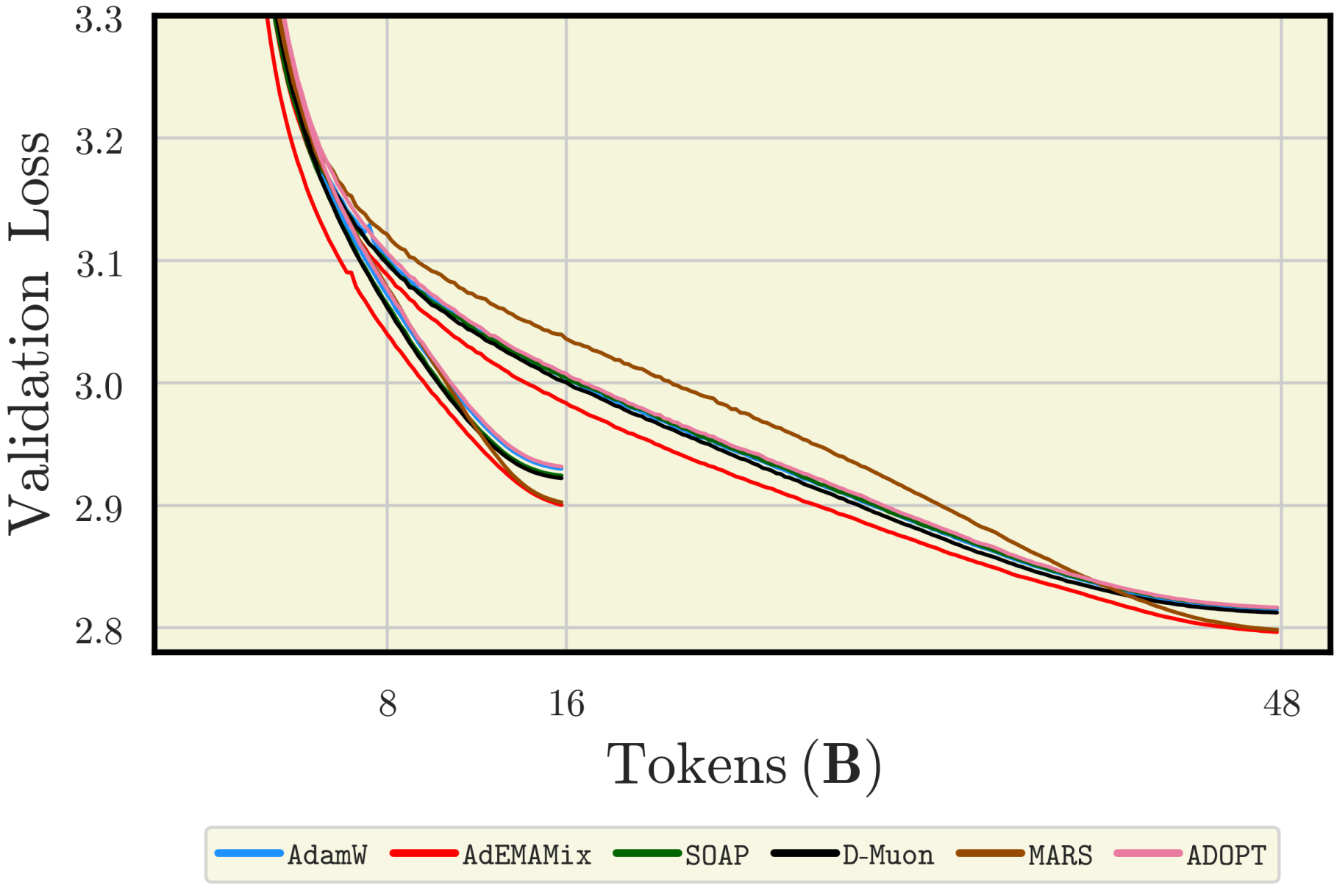
Benchmarking at small scale



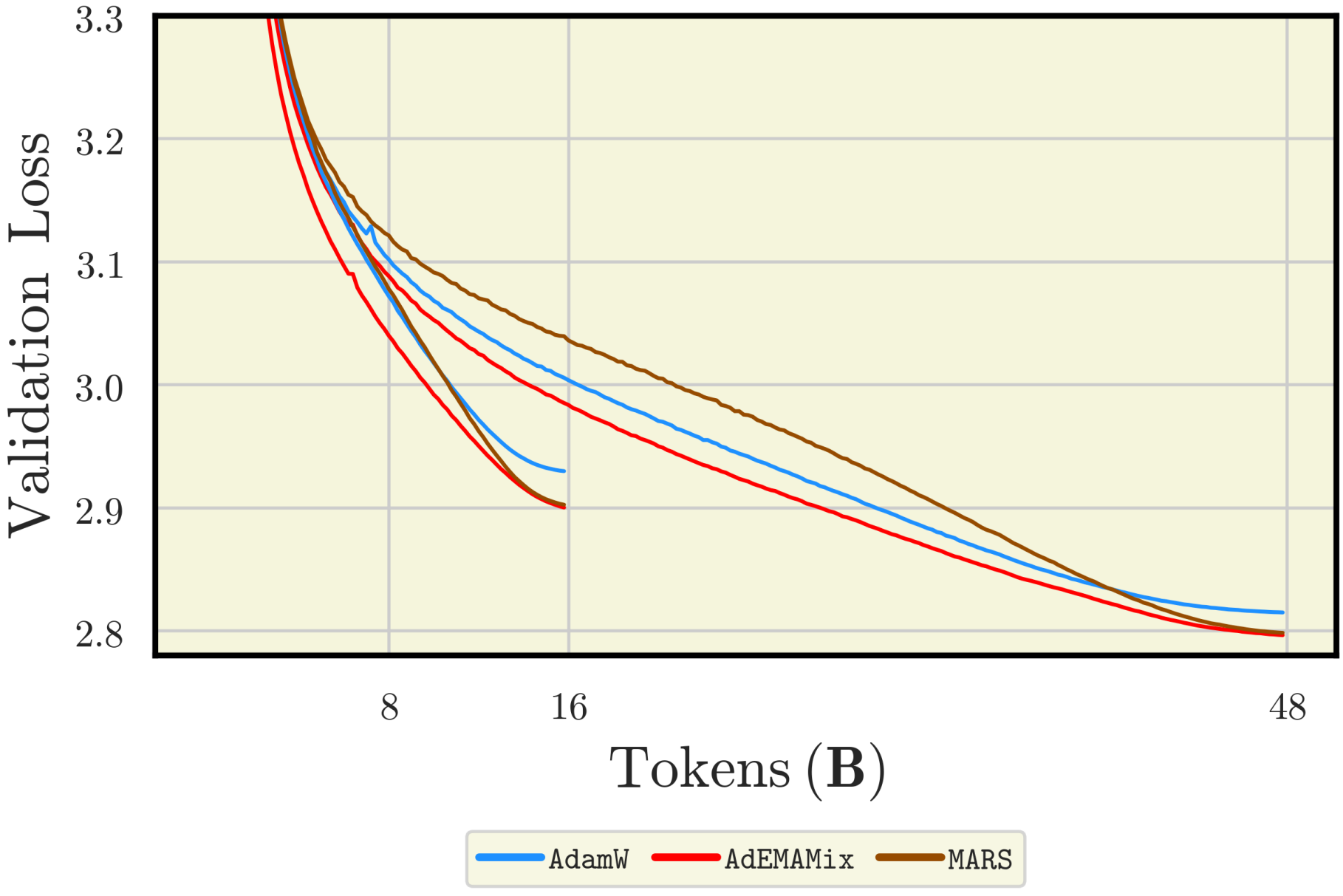
Benchmarking at small scale



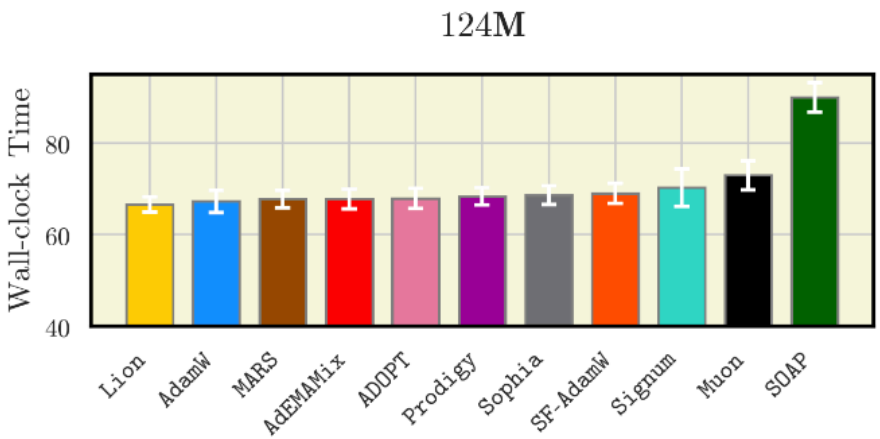
Benchmarking: Scaling Up

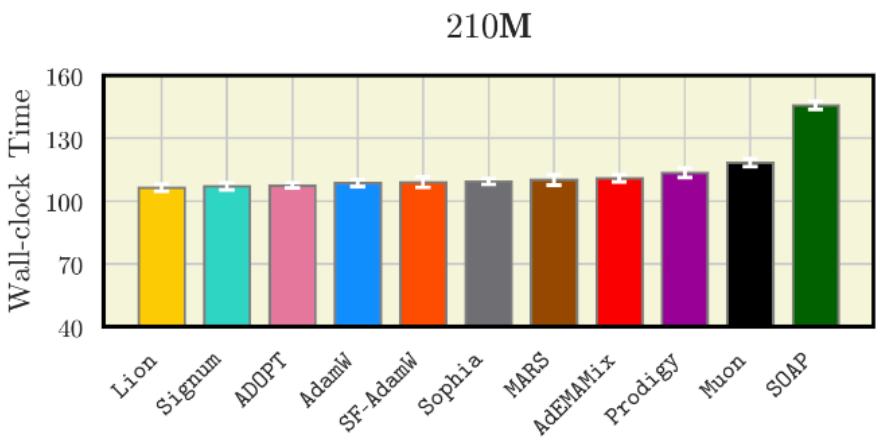


Benchmarking: Scaling Up



Benchmarking: Wall-Time





The End?

Thanks!