Is your batch size the problem? Revisiting the Adam-SGD gap in language modeling

Teodora Srećković TEODORASREC@GMAIL.COM
Jonas Geiping JONAS@TUE.ELLIS.EU
Antonio Orvieto ANTONIO@TUE.ELLIS.EU
Max Planck Institute for Intelligent Systems, ELLIS Institute Tübingen, Tübingen AI Center

Abstract

Adam is known to perform significantly better than Stochastic Gradient Descent in language models, a phenomenon for which a number of explanations have been proposed. In this work, we revisit this "optimizer gap" through a series of comprehensively tuned baseline training runs for language modeling with transformers. We exhaustively study how momentum, gradient clipping, and batch size affect the gap between SGD and Adam. Our empirical findings show that SGD with momentum can actually perform similarly to Adam in small-batch settings, if tuned correctly. We revisit existing explanations for Adam's advantage, including heavy-tailed class imbalance, directional sharpness, and Hessian heterogeneity, which struggle to explain these findings. Finally, by analyzing our transformer training runs and a simple quadratic setting, we provide new insights into what makes SGD perform poorly - showing that batch size has to be a necessary component of any explanation of the optimizer gap.

1. Introduction

The Adam optimizer [15] is used pervasively in deep learning, especially when training large language models (LMs) [4, 10, 19] and vision transformers [16]. Industrial practice relies on the success of Adam, and thousands of GPU hours every day are spent at large companies using Adam to train their next-generation large language models.

Even in new sophisticated optimization pipelines looking to dethrone Adam, such as Muon [13], most current implementations [20, 32] rely on plain Adam with weight decay (AdamW, Loshchilov and Hutter [23]) for critical subsets of parameters, such as normalization layers, text embeddings and prediction heads. This new world is still a bit surprising. Up until around the year 2018, the Adam optimizer was in occasional use, but stochastic gradient descent (SGD) with momentum was known to lead to neural networks with better accuracy on unseen data [40], relegating Adam to speed runs and quick comparisons [9]. Yet, from the start, language modeling with Transformers required Adam. In fact, Transformer LMs have been reportedly untrainable with SGD [41], especially due to the critical parameters listed above.

Over the years, researchers have offered a number of compelling explanations regarding the remarkable performance of Adam compared to SGD in language modeling, attributing it either to the peculiar noisy nature of text data [44, 45] or the heterogeneous structure [27, 46] of the Transformer architecture [38] — comprising semantically and structurally dissimilar layers. While most hypotheses regarding the Adam-SGD gap can help guide our understanding [2], a particularly crucial insight was recently brought to light by Kunstner et al. [17]: the Adam-SGD gap is also

observable in full-batch training, and is hence clear that the stochastic and potentially heavy-tailed nature of stochastic gradients may not be the challenge Adam is able to tackle. Inspired by the latter discussion, we take an **orthogonal approach**:

Instead of asking why Adam often outperforms SGD, we wonder:
In which Transformer-based language model training setting, if any, does SGD work?

In other words, while most recent works try to maximize the gap between SGD an Adam in order to explain it more easily, we here try to minimize it. We believe such view is novel in the literature, and can provide many valuable insights on the Adam-SGD gap, especially to discover settings that falsify existing hypotheses, and to enumerate necessary criteria that explanations have to fulfill.
Our contributions are as follows:

- Despite our own surprise, we show that LMs can be trained with SGD as effectively as Adam at the same token budget, as long as the batch size is chosen small enough, and hyperparameters, such as clipping and momentum are chosen correctly. We found this holding even at a scale of 1B parameters.
- We carefully revisit prior explanations — such as heavy-tailed class imbalance [18], directional sharpness [29], and Hessian heterogeneity [46] — in our setup. While our experiments confirm that these explanations can shed light and are useful to describe settings where Adam outperforms SGD, we find that no prior work can directly explain why SGD can outperform Adam at low batch sizes, while achieving satisfactory performance. Notably, in stark contrast with works attributing the gap to heavy-tail noise, we observe that increased stochasticity actually reduces the Adam–SGD gap.
- We enhance our intuition by further studying how gradient clipping and learning rate grafting [1] affect performance.
- We cross-check our findings in the toy quadratic setup of Zhang et al. [46], and further study why adaptive optimization may have a different batch size sensitivity compared to SGD, inspired by recent works on SDE models [7, 24].

Taken together these findings paint a new picture of the optimizer gap, provide practical hints for practitioners in small-scale settings where small batch sizes are the norm, and optimizer memory usage is critical, and constrains future theoretical investigations.

2. Adam vs. SGD: Effects of hyperparameters and training regimes

To systematically investigate the performance gap between Adam and SGD, we conduct a series of experiments in language modeling using a Transformer architecture. Our goal is to understand how this gap evolves under various training regimes and hyperparameter configurations.

2.1. Experimental setup

We conduct all experiments on the SlimPajama [34] dataset using a 160M-parameter 12-layers transformer, trained with Adam and SGD with momentum. Full model and training details are provided in Appendix B.

2.2. Effect of batch size on the Adam-SGD gap

We first study how the gap between Adam and SGD changes with batch size under a fixed compute budget, when momentum and learning rate are tuned.

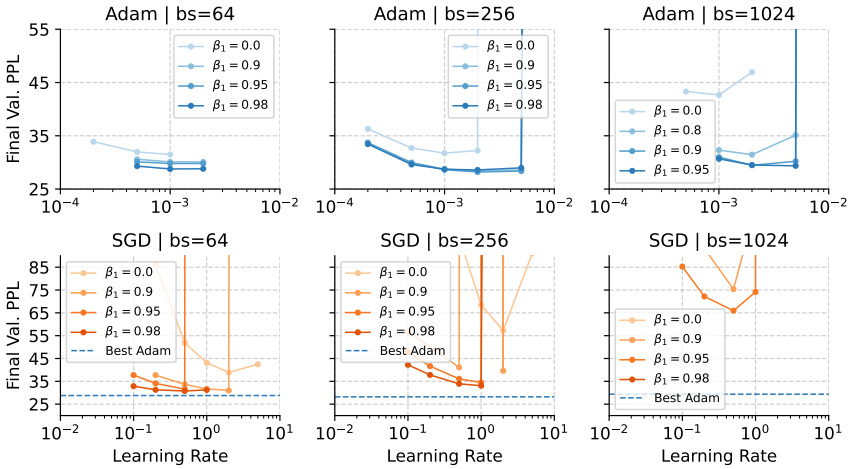

Figure 1: Learning rate and momentum sweep for SGD and Adam across batch sizes under a fixed compute budget of 1.3B tokens. Perplexities are measured on 100M held-out tokens.

Setup. All experiments use 512-token sequences, a 1.3 B-token budget, and a cosine learning-rate schedule [22] with 10 % warm-up. We test batch sizes of 64, 256, and 1024, tuning the learning rate and momentum β_1 for each, as detailed in the in the Appendix C. Results are reported as the final validation perplexity evaluated on 100M held-out tokens and are shown in Figure 1. When very large learning rates cause instability, we report the best run at the largest stable rate.

Results. Adam shows similar performance across different batch sizes under a fixed token budget, as shown in Figure 1. Surprisingly, *SGD can match Adam when training with small batch sizes*, but the gap increases as the batch size grows. For both SGD and Adam, momentum becomes crucial once the batch size is increased, as noted also by Kunstner et al. [17] and Zhao et al. [47].

We also find that using a relatively small sequence length of 512 is not a crucial factor in these dynamics. As we show in the next section, qualitatively the same behavior emerges at sequence length 2048 – as long as the number of tokens per iteration is held constant. This suggests that performance differences can be attributed to the *effective batch size (in tokens)* at a given sequence length, rather than sequence length alone. Further analysis is provided in Appendix C.

2.3. Are large batch sizes the problem, or is it the number of steps?

Our previous experiments show that SGD can match Adam in small-batch settings when both optimizers are carefully tuned. Crucially, note that in Figure 1 all methods see a total of 1.3B tokens. This implies that, e.g., at batch size 1024, methods perform 1/16 of the steps compared to batch size 64. This observation raises an important question: does SGD truly break at large batch sizes, or is it simply slower to converge, compared to Adam, at higher batch sizes? In other words, *can SGD reach Adam-level performance even at higher batch sizes, if given more training steps?*

To investigate this, we compare performance across batch sizes under two training regimes: (1) a fixed token budget and (2) a fixed number of steps. This comparison allows us to disentangle the effects of slow convergence from actual poor optimization performance.

Setup. The experimental setting is the same as in the previous section, except with a sequence length of 2048 – which we increased to ablate on this factor for the second experiment. We train models across a range of batch sizes, from 8 to 1024, and run for different numbers of steps, ranging

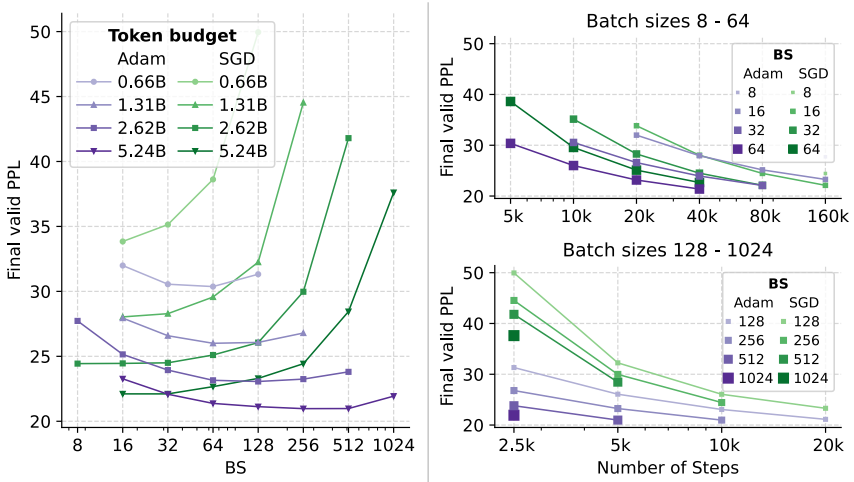

Figure 2: SGD (green) and Adam (purple) performance across batch sizes. Left: fixed token budget (darker colors – more tokens); the gap increases with batch size across all token budgets. Right: fixed number of steps (darker colors – larger batch sizes); the gap decreases with the number of steps. SGD improves with longer training and can match Adam, given a sufficiently small batch size.

from 2.5k to 160k, so larger batches are trained for fewer steps and smaller batches for more. We switch from the cosine scheduler used previously to a WSD scheduler [11], in order to better compare runs before learning rate decay begins. SGD and Adam hyper-parameters follow the tuned configurations in Figure 1. Full details are provided in the Appendix D.

Results. The left panel of Figure 2 clearly shows that Adam improves with larger batch sizes at a fixed token budget, while SGD shows a drastically opposite trend – performance consistently degrades as batch size increases. Under a fixed token budget, matching performance between Adam and SGD is conditional on using very small batch sizes, leading to significantly longer training and poor memory usage. This result highlights a key limitation of SGD: it is highly inefficient in realistic large-scale language model training, where large batches are required for practical efficiency. In the right panel on Figure 2, we show performance after training with various numbers of steps. At the same step count, the gap between Adam and SGD grows with batch size, but SGD improves significantly with more steps and can eventually match or even outperform Adam with long enough training. This illustrates that SGD is not necessarily bad, just very slow to converge in large-batch settings. More results are reported in Appendix D.

Scaling experiments. To test whether our findings persist at scale and across datasets, we experiment with larger models (250M, 410M, and 1B parameters) and include additional training on the FineWeb dataset [30] . We repeat the same experiments, varying token budgets and number of training steps, for the 160M model on SlimPajama and 250M model on FineWeb. Results and setup details are reported in Appendix B and Appendix E, showing that our core claims hold when scaling up the model and switching to a different dataset.

To further test whether SGD can outperform Adam at scale, we train:

- **410M** model on SlimPajama (sequence length 2048, batch size 8, 500k steps);
- **1B** model on FineWeb (sequence length 1024, batch size 16, steps 850k).

Full training details and learning rate tuning plots are provided in Appendix B. Trajectories for both models are shown in Figure 5, demonstrating that **SGD can outperform Adam even at a 410M**

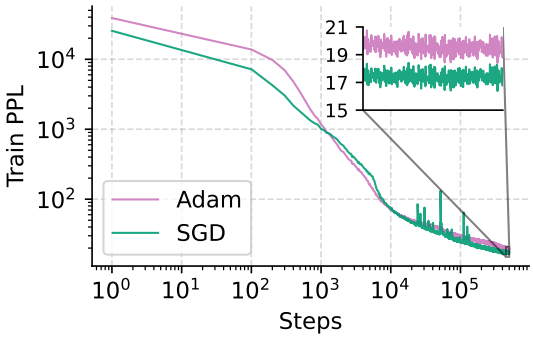

Figure 3: 410M model on SlimPajama (seq. length 2048, batch size 8, 500k steps) – 1.5 days of training.

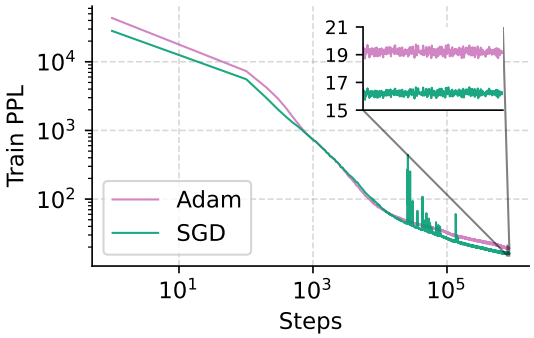

Figure 4: 1B model on FineWeb (seq. length 1024, batch size 16, 850k steps) – 5 days of training.

Figure 5: SGD can outperform Adam even at 410M and 1B scales in small-batch regimes.

and 1B scale. For these experiments, we choose the largest batch size that can fit in our NVIDIA A100 80GB card, and do not use gradient accumulation.

3. Revisiting and extending explanations for the Adam–SGD gap

Several recent works have proposed explanations for Adam's advantage over SGD through the lens of data or architectural properties (see Appendix A). All these explanations improve our understanding of the performance gap, yet most are restricted to specific scenarios where the gap between Adam and SGD is pronounced. In contrast, we ask: *do these explanations also account for SGD's strong performance in small-batch settings?*

In this section, we revisit the heavy-tailed class imbalance hypothesis proposed by Kunstner et al. [18], while we present analyses of directional sharpness and Hessian heterogeneity in Appendix F, and show that they struggle to explain good SGD performance. We give theoretical insight based on an SDE-based model in Appendix H.

Complementing this perspective, to understand what limits SGD in large-batch settings, we analyze the roles of update direction and magnitude, using grafting and adaptive clipping. We find that direction is the key issue as we show in Appendix G.

3.1. Heavy-tailed class imbalance

Prior work by Kunstner et al. [17] attributes Adam's advantage over SGD to heavy-tailed class imbalance in token distributions, showing that SGD has difficulty optimizing rare (least common) tokens. We follow their methodology and group all tokens from the training set into 10 frequency groups, from the first group, which contains the 10% least frequent tokens, to the last group, which contains the 10% most frequent ones.

We apply this analysis to the setting from subsection 2.2, comparing batch sizes 64 and 1024, where SGD performs drastically differently, using runs with the optimal combination of β_1 and learning rate for each case. We find that class imbalance exists in both cases: the persistence of low- and high-frequency tokens is similar, as shown in Figure 7. However, this does not appear to cause

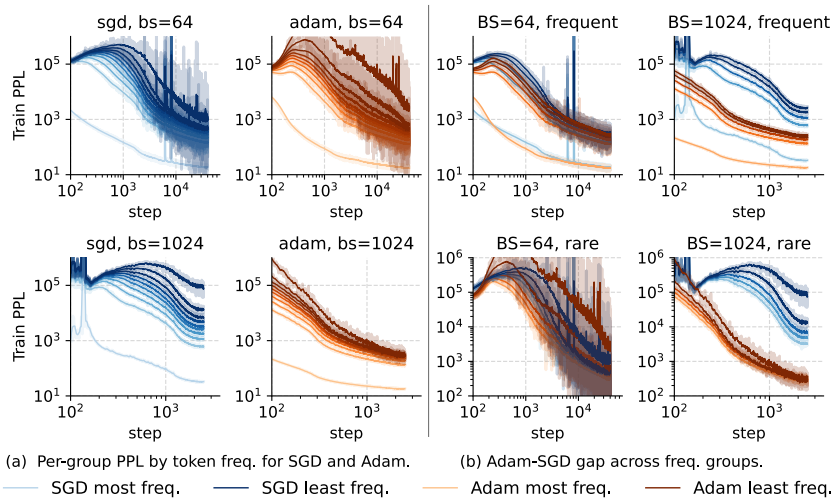

Figure 6: **(a)** Perplexity during training by frequency group, in small- and large-batch settings. **(b)** Adam–SGD gap per group: in large batches, SGD lags behind across all groups–especially on rare tokens. This effect is absent in small batches.

problems for small-batch SGD, suggesting that *class imbalance alone does not imply an Adam-SGD gap across all training regimes*.

We further compute perplexity separately for each frequency group and report it over training. From Figure 6 (a), we observe that both optimizers make faster progress on more frequent tokens in all settings, as expected. The relative difference in perplexity between frequency groups is more significant for SGD in the large-batch setting than for the small, while the opposite holds for Adam.

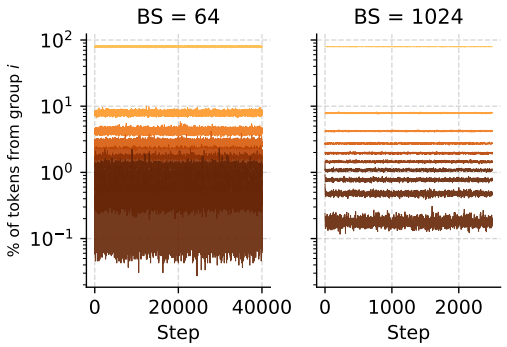

Figure 7: Batch token distribution for batch sizes 64 and 1024. Lighter colors – less frequent tokens. Token statistics at lower batch sizes are nosier but of similar magnitude.

Comparing Adam and SGD across frequency groups in Figure 7, we observe that in the large-batch setting, SGD underperforms Adam across all groups, as shown in Figure 6 (b). However, the gap is notably more significant for less frequent tokens, which aligns well with findings from Kunstner et al. [18], suggesting that rare tokens could be more challenging for SGD in imbalanced settings. In contrast, this effect is not observed for the small-batch regime in our setting, as also clear from the results in section 2. We would expect this problem of SGD to hold, independent of batch size, but in the setting where SGD works well, the issue disappears.

4. Discussion

Is it impossible to train language models with SGD? We show that in small-batch settings, with tuned momentum and clipping, SGD can be competitive, even for a 1B model. This challenges common explanations for the Adam–SGD gap. Revisiting prior theories, we find they fall short, and argue instead that gradient noise, amplified by batch size, plays a central role, motivating a stronger explanation based on SDEs.

References

[1] Naman Agarwal, Rohan Anil, Elad Hazan, Tomer Koren, and Cyril Zhang. Disentangling adaptive gradient methods from learning rates. *arXiv preprint arXiv:2002.11803*, 2020.

[2] Kwangjun Ahn, Xiang Cheng, Minhak Song, Chulhee Yun, Ali Jadbabaie, and Suvrit Sra. Linear attention is (maybe) all you need (to understand transformer optimization), March 2024. URL http://arxiv.org/abs/2310.01082. arXiv:2310.01082 [cs, math].

[3] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signSGD: Compressed Optimisation for Non-Convex Problems, August 2018. URL http://arxiv.org/abs/1802.04434. arXiv:1802.04434 [cs, math].

[4] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *ICML*, 2023.

[5] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. *arXiv preprint arXiv:2204.06745*, 2022.

[6] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240): 1–113, 2023.

[7] Enea Monzio Compagnoni, Tianlin Liu, Rustem Islamov, Frank Norbert Proske, Antonio Orvieto, and Aurelien Lucchi. Adaptive methods through the lens of SDEs: Theoretical insights on the role of noise. In *The Thirteenth International Conference on Learning Representations*, 2025.

[8] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35, 2022.

[9] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[10] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[11] Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro Von Werra, and Martin Jaggi. Scaling Laws and Compute-Optimal Training Beyond Fixed Training Durations, October 2024. URL http://arxiv.org/abs/2405.18392. arXiv:2405.18392 [cs].

[12] Kaiqi Jiang, Dhruv Malik, and Yuanzhi Li. How Does Adaptive Optimization Impact Local Neural Network Geometry?, November 2022. URL http://arxiv.org/abs/2211.02254. arXiv:2211.02254 [cs].

[13] Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

[14] Andrej Karpathy. Nanogpt, 2022.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Ananya Kumar, Ruoqi Shen, Sébastien Bubeck, and Suriya Gunasekar. How to fine-tune vision models with sgd. *arXiv preprint arXiv:2211.09359*, 2022.

[17] Frederik Kunstner, Jacques Chen, Jonathan Wilder Lavington, and Mark Schmidt. Noise is not the main factor behind the gap between sgd and adam on transformers, but sign descent might be. In *ICLR*, 2023.

[18] Frederik Kunstner, Robin Yadav, Alan Milligan, Mark Schmidt, and Alberto Bietti. Heavy-Tailed Class Imbalance and Why Adam Outperforms Gradient Descent on Language Models, July 2024. URL http://arxiv.org/abs/2402.19449. arXiv:2402.19449 [cs, math, stat].

[19] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[20] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for LLM training. *arXiv preprint arXiv:2502.16982*, 2025.

[21] Tianyi Liu, Zhehui Chen, Enlu Zhou, and Tuo Zhao. A diffusion approximation theory of momentum stochastic gradient descent in nonconvex optimization. *Stochastic Systems*, 2021.

[22] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[23] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[24] Sadhika Malladi, Kaifeng Lyu, Abhishek Panigrahi, and Sanjeev Arora. On the SDEs and Scaling Rules for Adaptive Gradient Algorithms, February 2023. URL http://arxiv.org/abs/2205.10287. arXiv:2205.10287 [cs].

[25] GN Mil'shtein. Weak approximation of solutions of systems of stochastic differential equations. *Theory of Probability & Its Applications*, 30(4):750–766, 1986.

[26] Toan Q Nguyen and Julian Salazar. Transformers without tears: Improving the normalization of self-attention. *arXiv preprint arXiv:1910.05895*, 2019.

[27] Lorenzo Noci, Sotiris Anagnostidis, Luca Biggio, Antonio Orvieto, Sidak Pal Singh, and Aurelien Lucchi. Signal propagation in transformers: Theoretical perspectives and the role of rank collapse. *Advances in Neural Information Processing Systems*, 2022.

[28] Antonio Orvieto and Robert Gower. In search of adam's secret sauce. *arXiv preprint arXiv:2505.21829*, 2025.

[29] Yan Pan and Yuanzhi Li. Toward Understanding Why Adam Converges Faster Than SGD for Transformers, May 2023. URL http://arxiv.org/abs/2306.00204. arXiv:2306.00204 [cs].

[30] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

[31] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[32] Ishaan Shah, Anthony M Polloreno, Karl Stratos, Philip Monk, Adarsh Chaluvaraju, Andrew Hojel, Andrew Ma, Anil Thomas, Ashish Tanwer, Darsh J Shah, et al. Practical efficiency of muon for pretraining. *arXiv preprint arXiv:2505.02222*, 2025.

[33] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[34] Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama. https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama, 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

[35] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[36] Akiyoshi Tomihari and Issei Sato. Understanding Why Adam Outperforms SGD: Gradient Heterogeneity in Transformers, January 2025. URL http://arxiv.org/abs/2502.00213. arXiv:2502.00213 [cs].

[37] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[39] Ben Wang and Aran Komatsuzaki. Gpt-j-6b: A 6 billion parameter autoregressive language model. 2021. *URL https://github. com/kingoflolz/mesh-transformer-jax*, page 8, 2022.

[40] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. *Advances in neural information processing systems*, 30, 2017.

[41] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pages 10524–10533. PMLR, 2020.

[42] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

[43] Hanlin Zhang, Depen Morwani, Nikhil Vyas, Jingfeng Wu, Difan Zou, Udaya Ghai, Dean Foster, and Sham Kakade. How Does Critical Batch Size Scale in Pre-training?, February 2025. URL http://arxiv.org/abs/2410.21676. arXiv:2410.21676 [cs].

[44] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity, February 2020. arXiv:1905.11881 [cs, math].

[45] Jingzhao Zhang, Sai Praneeth Karimireddy, Andreas Veit, Seungyeon Kim, Sashank J. Reddi, Sanjiv Kumar, and Suvrit Sra. Why are Adaptive Methods Good for Attention Models?, October 2020. URL http://arxiv.org/abs/1912.03194. arXiv:1912.03194 [cs, math].

[46] Yushun Zhang, Congliang Chen, Tian Ding, Ziniu Li, Ruoyu Sun, and Zhi-Quan Luo. Why Transformers Need Adam: A Hessian Perspective, June 2024. URL http://arxiv.org/abs/2402.16788. arXiv:2402.16788 [cs].

[47] Rosie Zhao, Depen Morwani, David Brandfonbrener, Nikhil Vyas, and Sham Kakade. Deconstructing What Makes a Good Optimizer for Language Models, July 2024. URL http://arxiv.org/abs/2407.07972. arXiv:2407.07972 [cs].

Appendix A. Related work

Class imbalance. Kunstner et al. [18] explains the advantage of Adam over SGD on language tasks through the heavy-tailed class imbalance in token distributions. They show that the loss for rare tokens decreases significantly more slowly when training with SGD than with Adam, making SGD training inefficient in such settings. In contrast, Adam makes steady progress even on these low-frequency tokens. Their empirical findings show that this explanation is robust to different architectures and settings, suggesting that the performance gap is primarily driven by class imbalance. This explanation is not limited to text data and Transformers: the authors show that an Adam-SGD gap consistently appears in class-imbalanced scenarios, but vanishes when the data is balanced.

Transformer architecture. Another line of work focuses on the specific characteristics of Transformer architectures. Zhang et al. [46] provide a Hessian-based perspective, showing that Transformers have a block-heterogeneous Hessian spectrum, meaning the Hessian spectrum varies significantly across parameter blocks. In such settings, Adam outperforms SGD by a large margin, while both optimizers perform similarly on architectures with a more homogeneous Hessian. They empirically show that this finding holds across different data modalities and architectures, and that Adam outperforms SGD even on ViT models, differing from the findings of Kunstner et al. [18].
In contrast, Tomihari and Sato [36] focus on gradient heterogeneity, explaining Hessian heterogeneity as a consequence of the correlation between gradients and the Hessian. They observe that, in Transformers, a large disparity in gradient norms across parameters leads to optimization challenges for SGD, which Adam's adaptivity can address.
Finally, through empirical studies, Zhao et al. [47] find that adaptive optimizers have stable performance over a wide range of hyperparameter settings, while SGD is highly sensitive and often requires careful tuning. They also confirm that full Adam-style adaptivity is not always necessary, showing that their proposed method, which applies adaptivity only to normalization layers and the final layer, can close most of the gap compared to Adam in their setting.

Heavy-tailed gradient noise. Earlier work by Zhang et al. [45] asks whether the nature of stochastic gradient noise explains why the Adam-SGD gap exists in Transformer models but not in other architectures. They show that Transformer models produce gradient noise with heavy-tailed distributions, in contrast to nearly Gaussian noise in CNNs. They argue that heavy-tailed gradient noise degrades the performance of SGD, while Adam demonstrates greater robustness.
However, evidence from Kunstner et al. [18] shows that noise alone is not the primary cause of Adam's superiority, since the gap exists even in the full-batch setting. They find that the performance gap persists, and even that Adam's advantage grows, as stochastic noise vanishes.

Optimization trajectories. Several researchers have investigated how Adam differs from SGD by characterizing the path taken during training. Jiang et al. [12] analyze local geometry along training paths and define a statistic that measures the uniformity of the diagonal of the Hessian. On LMs, they find that Adam's trajectory consistently moves through regions where this measure is significantly smaller than the values found along the trajectory of SGD with momentum.
Similarly, Pan and Li [29] introduce directional sharpness as a metric to explain Adam's faster convergence in Transformers. Rather than examining the entire Hessian, they look at the sharpness along the update direction at each step, showing that Adam makes updates in directions with much smaller sharpness than SGD. Although these measures help characterize the training dynamics of SGD and Adam, they appear to correlate with the performance gap rather than fully explain it.

Evidence from simplified settings. Recent work shows that the Adam-SGD gap persists even in simplified Transformer architectures. Ahn et al. [2] demonstrate that the characteristic optimization

challenges mentioned above also appear in shallow linear Transformers, models without nonlinear activations, on a linear regression task.

Appendix B. Further Experiments and Experimental Details

We conduct most of our experiments on the SlimPajama [34] dataset using a nanoGPT [14], augmenting it with Rotational Positional Embedding [35], RMSNorm [42], and SwiGLU [33]. We do not adopt QK normalization or z-loss, as those modifications are quite recent. All our models have a vocabulary size of 50280 and make use of GPT-Neox tokenizer [5]. We adopt an enhanced training recipe, made popular by large language models such as LLaMa [37]. These modifications include: training in bfloat16; employing a linear learning rate warm-up for 10% of the training steps (unless specified otherwise), followed by either cosine annealing to $1e-5$ of WSD [11]. Global norm clipping is used (unless specified or ablated upon) for gradients with norms above 1 (on the raw gradient, as a first step). We have no weight tying between the embedding and the last linear layer. Validation perplexity always refers to a separate subset consisting of 100M tokens.

We do not apply weight decay in any of our experiments to eliminate potential side-effects. For both SGD and Adam, we perform global gradient norm clipping on raw gradients (e.g., before applying momentum) unless otherwise stated. The β_2 parameter for Adam is fixed at 0.95 throughout all experiments, as common in the literature [4]. SGD always refers to SGD with momentum unless explicitly stated otherwise. Other details regarding sequence length, batch size, training budget, and hyperparameter grids are reported directly in the respective sections.

B.1. Experimental Setup

Computational Resources. All experiments use a single NVIDIA A100-SXM4-80GB.

Code. All our runs use the repository `https://github.com/Niccolo-Ajroldi/plainLM`.

Datasets. We test our claims on both the SlimPajama [34] and Fineweb [30] datasets.

Model settings (12 Layers, 160M). We use the same configuration as [4]: `https://github.com/EleutherAI/pythia/blob/main/models/160M/pythia-160m.yml`

- *Layers:* 12 Transformer [38] layers
- *Attention heads:* 12
- *Hidden size:* 768
- *Attention implementation:* Flashattention [8].
- *MLP type:* SwiGLU [33] with expansion factor 8/3.
- *Backbone:* PreLN Transformer [41] with skip connections.
- *Normalization:* RMSnorm [42] for both Attention and MLP.
- *Position embeddings:* Rotary embeddings (RoPE) to 25% of dimensions ([35])
- *Initialization:* the MLP and Attention output weights are initialized with variance $0.02/\sqrt{2\#\text{layers}}$ (scaling also similar to [31]). All other weights (comprising embeddings) are initialized with a standard deviation of 0.02 (Nguyen and Salazar [26], Wang and Komatsuzaki [39], Sec. 2.2). Biases are always initialized at zero.
- *Precision:* Mixed precision FP16 enabled.
- *Dropout:* Disabled for both hidden and attention layers (see also Chowdhery et al. [6]).

Model settings (250 M, 24 layers). We keep it identical to the setting above, and just increase the number of layers to 24.

Model settings (410 M). We use the same setting as [4], configuration can be found here: https://github.com/EleutherAI/pythia/blob/main/models/410M/pythia-410m-deduped.yml
- *Layers:* 24 Transformer layers
- *Attention heads:* 16
- *Hidden size:* 1024
- Other settings as 160M parameters.

Model settings (1B). We use the same setting as [4], configuration can be found here: https://github.com/EleutherAI/pythia/blob/main/models/1B/pythia-1b-deduped.yml
- *Layers:* 16 Transformer layers
- *Attention heads:* 8
- *Hidden size:* 2048
- Other settings as 160M parameters.

B.2. Hyperparameter Tuning for Section 2.3

Combined, the experiments in this section account for full training (at different token budgets) of more than 250 language models at different scales and batch sizes. Every reported result is relative to the best learning rate in our grid, defined for each setup.

Small-scale experiments. We consider SGD with $\beta = 0.98$ and global clipping before applying momentum. For Adam, we use the setting $\beta_1 = \beta_2 = 0.95$. Both settings are suggested by the sweep in Figure 1 and recent literature [28, 43, 47].
- For Figure 2 and Figure 12 (SlimPajama, 160M), we choose a sequence length of 2048. Inspired by the careful tuning of Figure 1, we consider the learning rate grid $[0.25, 0.5, 1.0]$ for SGD and $[0.001, 0.002, 0.004]$ for Adam.
- For Figure 14 (Fineweb, 160M), we choose a sequence length of 1024. Our learning rate grid here is the same as for SlimPajama (previous point). As a sequence length of $160k$, given our lack of experience with extremely low batch sizes (shorter sequence length), we operate on a slightly larger grid: $[0.0001, 0.0003, 0.001, 0.003]$ for Adam and $[0.03, 0.1, 0.3, 1]$ for SGD.
- For Figure 15 (SlimPajama, 250M - 24 layers), we choose a sequence length of 2048 and we also operate on a larger grid: $[0.0001, 0.0003, 0.001, 0.003]$ for Adam and $[0.03, 0.1, 0.3, 1]$ for SGD.

Medium scale experiments. For all SGD runs, we use $\beta = 0.98$. For Adam, we use the standard choice (0.9, 0.95) [4]. All our runs use global norm clipping and no weight decay.
- **410M model** (Figure 3): We train with sequence length 2048, for $500k$ steps on SlimPajama. Learning rate grid is $[1.25e-4, 2.5e-4, 5.0e-4, 1.0e-3]$ for Adam and $[0.125, 0.25, 0.5, 1]$ for SGD. The sweep results are presented in Figure 8.
- **1B model** (Figure 4): We train with sequence length 1024, for 850k steps on Fineweb. Learning rate sweep, shown in Figure 9 uses $[6.25e-5, 1.25e-4, 2.5e-4, 5.0e-4, 1.0e-3]$ for Adam and $[0.0625, 0.125, 0.25, 0.5, 1]$ for SGD.

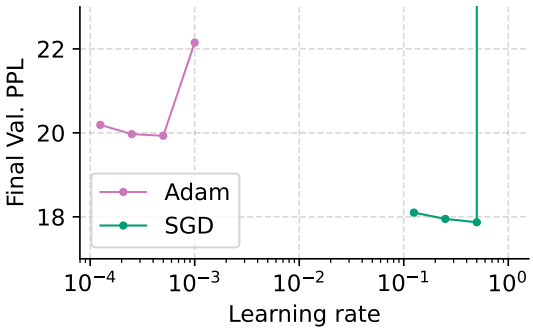

Figure 8: 410M model on SlimPajama (seq. length 2048, batch size 8, 500k steps)

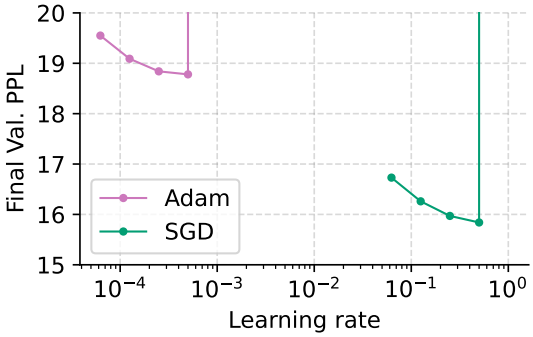

Figure 9: 1B model on FineWeb (seq. length 1024, batch size 16, 850k steps)

Figure 10: Learning rate sweep for 410M and 1B models. Trajectories for the optimal learning rate are shown in Figure 5.

Appendix C. Effect of batch size on the Adam–SGD Gap

Setup. All experiments use a sequence length of 512, a fixed token budget of 1.3B tokens, and a cosine learning rate scheduler [22] with 10% warmup. We compare three batch sizes: 64, 256, and 1024. The learning rate and momentum values are tuned for both optimizers at a batch size of 256. A sweep is performed over 5 learning rates and momentum values of 0.9, 0.95, and 0.98, including runs without momentum. High momentum values are motivated by findings from Zhao et al. [47], where SGD performs best with momentum 0.98. Based on the optimal learning rate found at batch size 256, we scale down the learning rate grid for batch size 64 and scale it up for batch size 1024, sweeping over 3 values in each case. Results are reported as the final validation perplexity evaluated on 100M held-out tokens and are shown in Figure 1. Some settings become unstable at very large learning rates, where one run may succeed, even if the median run diverges. In those settings, we report runs at the largest stable learning rate as optimal.

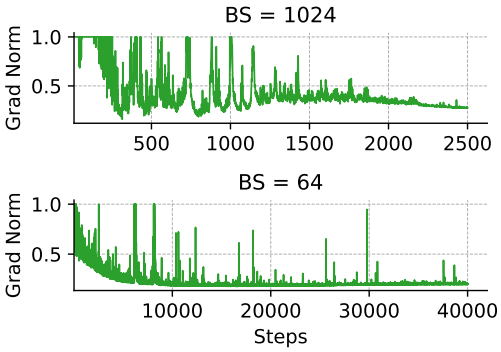

Figure 11: Gradient norm after clipping (threshold 1.0) shows that clipping is more frequent in large-batch training.

Clipping acts differently at different batch sizes. We observe that gradients are clipped more frequently when training with SGD at large batch sizes, as shown in Figure 11. Additionally, at small batch sizes, SGD performs equally well even without clipping; instead, at large batch sizes, training diverges if clipping is not employed.

Warmup length is not a confounder. We also verify that warmup length is not a confounding effect, sweeping 5–20% warmup schedules in our cosine-with-warmup scheduler.

Appendix D. Training under fixed token budget and number of steps

Setup. We train models across a range of batch sizes, from 8 to 1024, and run for different numbers of steps, ranging from 2.5k to 160k. All runs use a fixed warmup of 2000 steps. We switch from the cosine scheduler used previously to a WSD scheduler [11], to better compare runs before learning rate decay begins. We limit the total token budget between approximately 650M and 5.2B tokens: models using larger batches are not trained for the largest number of steps, while models using smallest batches are trained only for a large number of steps. For SGD, we choose a momentum $\beta = 0.98$ and run three distinct learning rates: $1, 0.5$ and 0.25. For Adam, we choose $\beta_1 = \beta_2 = 0.95$ as suggested by modern literature [43, 47] and report performance for the best performing learning rate in the grid $[1e-3, 2e-3, 4e-3]$. Both the SGD and Adam configurations are suggested from our more careful tuning performed in Figure 1.

Results. In addition to Figure 2, we report the perplexity during training for SGD and Adam with batch sizes 16 and 128 in Figure 12. For both optimizers, the gap decreases as training progresses. In the small-batch setting, SGD even outperforms Adam at the maximum number of steps. We show the same plots for other batch sizes in Figure 13.

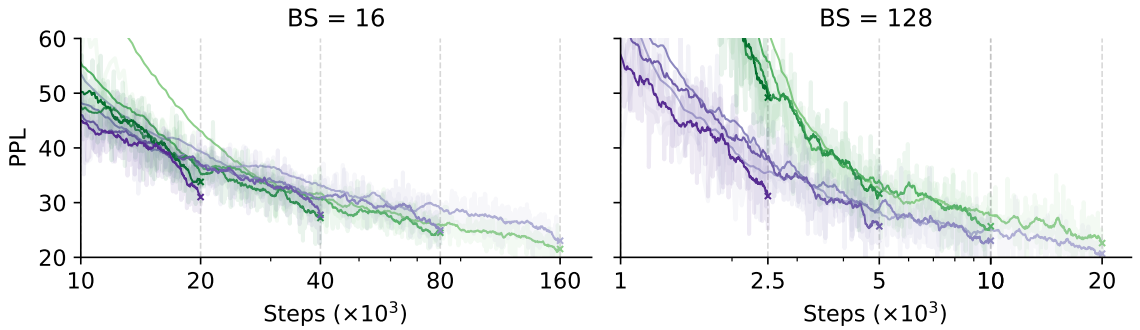

Figure 12: Perplexity during training for SGD (green) and Adam (purple) across different training lengths in small- and large-batch settings. For both, the gap decreases the longer we train. For small batch, at the max number of steps, SGD even performs better than Adam.

Appendix E. Additional results

We report the validation perplexity for the best-performing β_1 and learning rate combination for both Adam and SGD across batch sizes in Table 1. The experimental setting is described in subsection 2.2, and the results correspond to the sweep shown in Figure 1.

In addition to Figure 12, we report the training perplexity for all other batch sizes in Figure 12. We repeat the experiments from subsection 2.3 to verify that our findings generalize to a different dataset and a deeper model.

E.1. Scaling experiments across model sizes and datasets

First, we train the same 12-layer Transformer on the Fineweb dataset using SGD with momentum and Adam, tuning the learning rate as explained in Appendix B. Batch sizes vary from 4 to 512, and we use 3 different run lengths (i.e., different token budgets). From Figure 14, we observe that, at a fixed number of steps, the performance gap increases with batch size, and that with smaller batches and sufficiently long training, SGD outperforms Adam, consistent with the findings reported earlier.

Table 1: Best validation perplexities and corresponding hyperparameters for Adam and SGD across batch sizes. Results correspond to the sweep shown in Figure 1.

Batch Size	Optimizer	PPL	Hyperparameters
64	Adam	**28.77**	$\beta_1 = 0.98$, lr $= 1e{-3}$
	SGD	**30.76**	$\beta_1 = 0.98$, lr $= 5e{-1}$
256	Adam	**28.20**	$\beta_1 = 0.95$, lr $= 2e{-3}$
	SGD	**33.08**	$\beta_1 = 0.98$, lr $= 1e{+0}$
1024	Adam	**29.36**	$\beta_1 = 0.95$, lr $= 5e{-3}$
	SGD	**65.94**	$\beta_1 = 0.95$, lr $= 5e{-1}$

In a second experiment, we increase the model depth to 24 layers while keeping all other settings identical to subsection 2.3. We vary batch sizes from 4 to 64 and training lengths, and tune the learning rate as explained in Appendix B. As shown in Figure Figure 15, the same pattern holds for a deeper model.

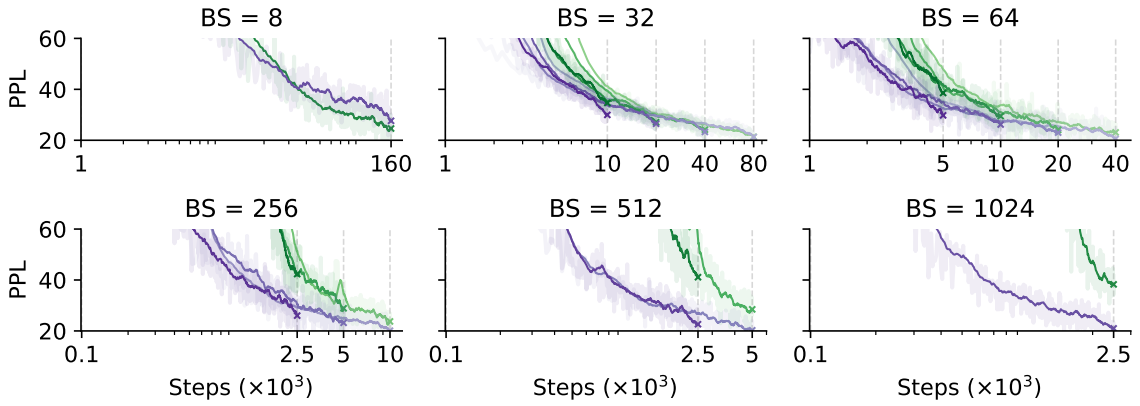

Figure 13: Perplexity during training for SGD (green) and Adam (purple) across different training lengths for all other batch sizes not shown in Figure 12. Solid lines show the rolling mean of PPL values; lighter lines show the raw values. As before, the gap decreases the longer we train, and SGD can eventually outperform Adam.

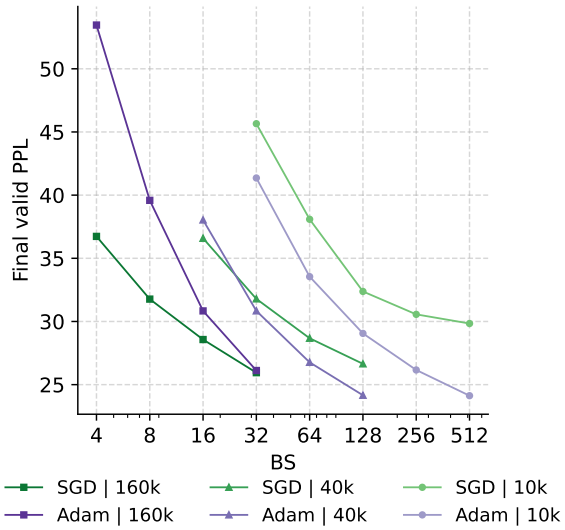

Figure 14: Fineweb dataset, sequence length 2048, 12 layers Transformer.

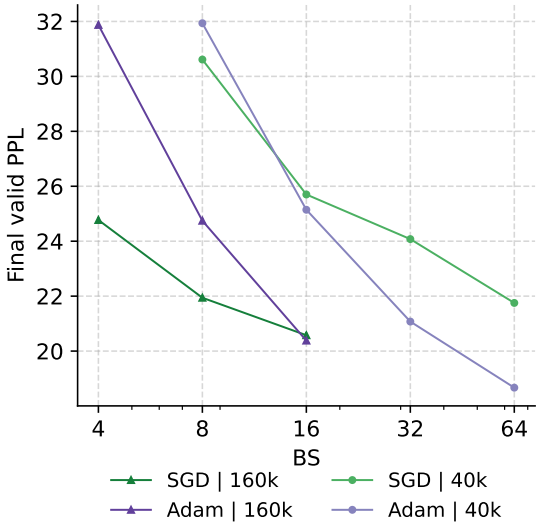

Figure 15: SlimPajama dataset, sequence length 2048, 24 layers Transformer.

Appendix F. Revisiting prior explanations

F.1. Directional sharpness

Pan and Li [29] introduce directional sharpness to explain the optimizer gap by studying a second-order Taylor expansion of the loss along the update direction. In this view, the first-order term (gradient correlation) measures how well the update aligns with the negative gradient, while the second-order term (directional sharpness) measures curvature along that direction. Making optimiza-

tion progress requires a strong negative gradient correlation and low directional sharpness. Let f be a generic loss to optimize and x_k denote the model parameters at iteration k, then

$$f(x_{k+1}) = f(x_k) + \underbrace{\nabla f(x_k)^\top (x_{k+1} - x_k)}_{\text{gradient correlation}} + \frac{1}{2} \underbrace{(x_{k+1} - x_k)^\top \nabla^2 f(x_k)(x_{k+1} - x_k)}_{\text{directional sharpness}} + O(\eta^3) \quad (1)$$

In Figure 16, we visualize gradient correlation, directional sharpness, and their sum — a second-order approximation of loss change, to indicate progress. As in our previous analysis, we compare two settings with drastic performance differences: batch sizes 64 and 1024 from subsection 2.2. In the large-batch setting, SGD shows low gradient correlation and high directional sharpness, resulting in weak or even positive total loss change, as reflected in the sum. In contrast, Adam has higher gradient alignment and lower directional sharpness throughout training. When SGD succeeds, its gradient correlation and directional sharpness closely match Adam's, producing a negative loss change in the sum. While these metrics align with SGD's success or failure, they do not directly explain why Adam outperforms SGD, nor why SGD performs well in small-batch regimes.

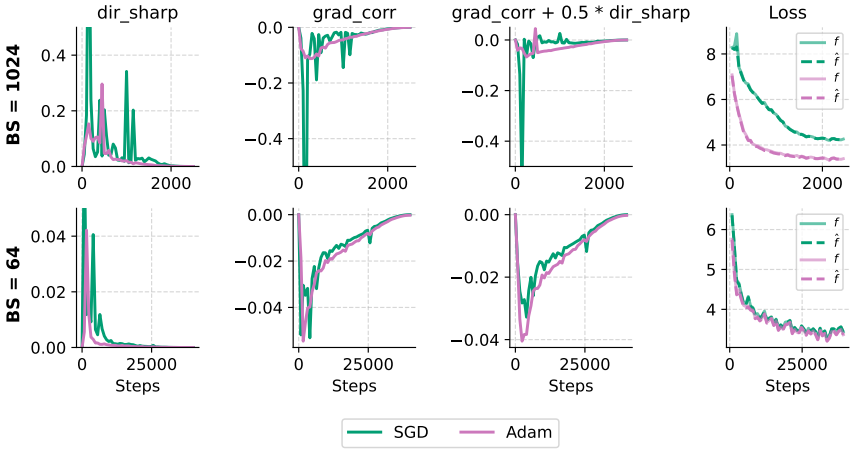

Figure 16: Gradient correlation, directional sharpness, their sum and second-order loss approximation during training, under small- and large-batch settings. Irrespective of the optimizer, good training trajectories have strong negative gradient correlation and low directional sharpness.

F.2. Hessian heterogeneity

From the line of work focusing on the architectural properties of Transformers, Zhang et al. [46] argue that the block-wise heterogeneity of the Hessian spectrum is a key factor behind Adam's strong performance and the weakness of SGD. They propose that, based on the Hessian structure at initialization, it is possible to predict whether SGD will perform well, **offering an explanation that is invariant to batch size**. To further explore the effect of batch size on heterogeneous problems, we revisit the simplified quadratic setting from their work and extend it by including batch size variation. We compare optimization on problems with homogeneous (CNN-like) and heterogeneous (Transformer-like) Hessians, where both share the same eigenvalue spectrum. We train with SGD and Adam using a cosine learning rate schedule and no clipping.

We observe the following in Figure 17 (details in Appendix I):

- Across batch sizes, the largest Adam-SGD gap is observed in the heterogeneous setting – this is the result by Zhang et al. [46]. As noted by [17], a similar pattern can be observed for signed momentum [3]. We develop on this in Appendix H.
- Adam benefits from increasing batch size, in both homogeneous and heterogeneous Hessian problems. SGD does not seem to profit from an increased batch size. We motivate this theoretically in Appendix H: early progress in SGD performance is driven by number of iterations, while it is batch size dependent for signed gradient methods.

To summarize, a performance **boost can be observed** at higher batches for both SignSGD and Adam, **regardless of heterogeneity**. While we confirm that heterogeneity amplifies the gap between SGD and adaptive methods, this result showcases that the phenomenon we study in this paper may not be limited to the heterogeneous setting. This is a key insight, bringing the discussion back to a statistical level where the landscape structure plays a less crucial role. We develop on this in Appendix H.

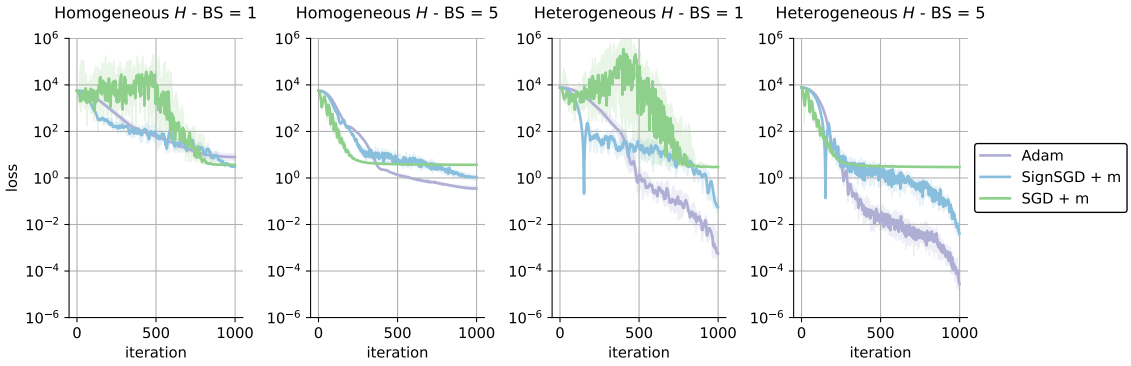

Figure 17: The gap between Adam and SGD relative to batch size also appears when studying only noisy quadratic models in [46]. The Heterogeneous setting is inspired by the Transformer structure, while the Homogeneous by the CNN structure. See the appendix for details. We note that the advantage of Adam of a big batch is also noticeable in the Homogeneous setting, yet much more drastically in the Heterogeneous setting. Learning rates are tuned so to give similar performance at Homogeneous batch size 1. Shown is mean and 2-sigma standard dev. for 10 runs.

Appendix G. Understanding what limits SGD performance

We saw that while prior work sheds light on the Adam-SGD gap in the large-batch regime (Appendix F), it remains unclear how batch size itself factors into these explanations.

Towards gaining more insights, we proceed as follows:

- In G.1 and G.2 we approach this from the SGD angle: *What goes wrong (see e.g. Figure 11) for SGD in large-batch settings that does not appear at small batch sizes?* To investigate this, we attempt to separate which component of the SGD update is more problematic in settings where it fails — is it its direction or magnitude?. We focus on the setting from subsection 2.2, using batch size 1024 and the optimal combination of β_1 and learning rate.
- In Appendix H we take a different approach, one based on noise statistics and adaptivity in a setup which is non-specific to the Transformer architecture. This analysis is inspired by the results in Figure 17, showing how adaptive methods may profit from large batch sizes regardless of the

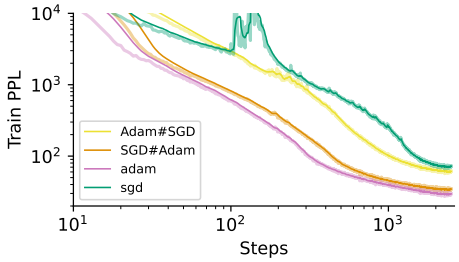

Figure 18: Grafting in large-batch training: using Adam's direction results in performance closer to Adam, while SGD direction leads to results closer to SGD.

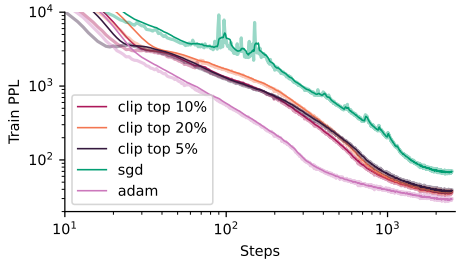

Figure 19: Adaptive clipping with different percentages of clipped coordinates in large-batch training. It improves SGD but still does not fully match Adam.

Hessian structure. Using theoretical tools, we prove here that while SGD performance in early training is dominated by number of iterations, the dynamics of signed momentum methods (cf. Figure 17) showcase a strong dependency on batch size from the very first iterations.

G.1. Insights from Grafting

To isolate the role of update direction and magnitude, we use the grafting technique proposed by Agarwal et al. [1], which applies the update direction of one optimizer with the magnitude of another. We train the model in the large batch setting, using both combinations: 1) SGD magnitude with Adam direction (SGD#Adam), and 2) Adam magnitude with SGD direction (Adam#SGD). We use the optimal β_1 from subsection 2.2, and sweep the learning rate for the grafted update. We report the training perplexity using the optimal learning rate for both grafting combinations in Figure 18. As shown, using SGD magnitude with Adam's direction performs comparably to Adam, while the reverse combination behaves similarly to SGD. This suggests that the *update direction* is the more problematic component of the SGD update in large-batch training.

G.2. Insights from Adaptive Clipping

The perspective that direction is the core problem aligns with the observation that global norm clipping does not help much in large-batch training with SGD. If the direction is the main issue, simply rescaling the gradient norm does not lead to better updates.

To investigate this further, we experiment with adaptive clipping, motivated by Pan and Li [29]. As shown in algorithm 1, we clip the top $p\%$ of the largest momentum coordinates at each step. We test several values of p (5, 10, and 20 %). For each value, we keep the optimal β_1 from the previous setting and tune the learning rate. Clipping with $p = 10\%$ performs best, but we observe that performance does not vary much across different values of p. This method helps reduce the gap between SGD and Adam, as shown in Figure 19. This suggests that a subset of larger update coordinates consistently contributes to poor update directions and slows down SGD in large-batch training. In contrast, small-batch training does not present the same problematic coordinates.

Further, we ask whether certain groups of parameters are more likely to produce problematic coordinates. To explore this, we inspect which layers the clipped momentum coordinates come from, using the best-performing setting with $p = 10\%$. In Figure 20, we show the fraction of parameters within each layer that are clipped, relative to the total number of parameters in that layer. We find

Algorithm 1 SGD with Adaptive Momentum Clipping

Input: Initial point x_0, learning rate η, momentum β, clipping fraction $p \in (0, 1)$
for $t = 1$ **to** T **do**
  $g_t \leftarrow \nabla f(x_t)$
  $m_t \leftarrow \beta m_{t-1} + g_t$
  Set clipping threshold τ_t as the $(1-p)$-quantile of $|m_t|$
  $\hat{m}_t \leftarrow \text{sign}(m_t) \cdot \min(|m_t|, \tau_t)$
  $x_{t+1} \leftarrow x_t - \eta \hat{m}_t$
end

that normalization layers are clipped the most, which aligns with findings from Zhao et al. [47] and Tomihari and Sato [36]. However, this does not imply that only normalization layers are problematic. As we observe significant clipping across other layers as well, this suggests that large coordinates persist across all parameters, though they are most pronounced in normalization layers.

Appendix H. Theoretical insights.

Towards explaining the phenomena observed in this paper, we provide a preliminary yet insightful analysis based on [7], based on the observed similarity between behaviors of Adam and SignSGD in Figure 17, as well as recent literature on their relation [13, 17].

Let X denote the model parameters and γ denote a batch of size B. We denote the stochastic gradient as $\nabla f_\gamma(x) := \frac{1}{B}\sum_{i\in\gamma}\nabla(f_i(x))$ and by $\Sigma(x)$ the noise covariance at batch size 1. The stochastic differential equation (SDE) approximation of SGD reads [21, 25]

$$dX_t = -\nabla f(X_t)dt + \sqrt{\frac{\eta\Sigma}{B}}dW_t, \qquad (2)$$

We now state a recent result showing that the drift of signed updates – driving performance in early training – has an extra dependency on the batch size.

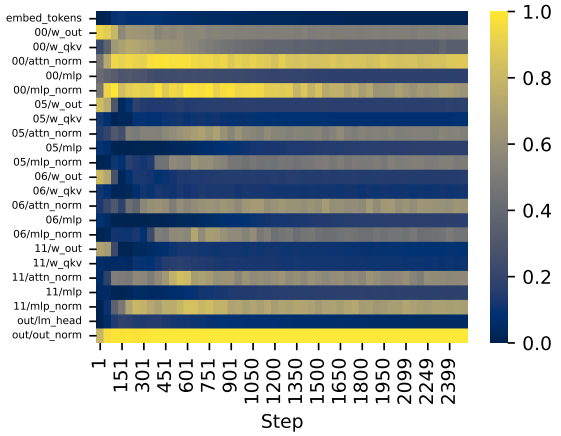

Figure 20: Fraction of clipped momentum coordinates per layer during training, using $p = 10\%$ adaptive clipping. Only a subset of blocks is shown for clarity, as similar patterns are observed across all blocks. Clipping is present across all parameters, but most pronounced in normalization layers.

theorem[7]] Under the assumption of i.i.d. Gaussian noise, the following SDE provides a 1-weak approximation [25] of the discrete update of SignSGD

$$dX_t = -\operatorname{erf}\left(\sqrt{\frac{B}{2}}\Sigma^{-\frac{1}{2}}\nabla f(X_t)\right)dt + \sqrt{\eta}\sqrt{I_d - \operatorname{Diag}\left(\operatorname{erf}\left(\frac{\sqrt{B}\Sigma^{-\frac{1}{2}}\nabla f(X_t)}{\sqrt{2}}\right)\right)^2}dW_t, \qquad (3)$$

where the error function $\operatorname{erf}(x) := \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$ and the square are applied component-wise.

While Compagnoni et al. [7] provide a similar result for the Heavy-tail setting, the Gaussian case already highlights a crucial distinction between signed gradient methods and classical SGD:

recall that erf (similar in shape to a tanh) is linear on a large interval around zero. The local update of parameters is then driven by $-\operatorname{erf}\left(\sqrt{\frac{B}{2}}\Sigma^{-\frac{1}{2}}\nabla f(X_t)\right)$ in the signSGD case, while in the SGD setting, this term is simply $-\nabla f(X_t)dt$. This analysis provides strong evidence for our results: using large batch sizes accelerates convergence (larger drift) in signSGD, while the performance of SGD in early training is batch-size agnostic and hence driven by the number of iterations.

Appendix I. Toy Quadratic Example

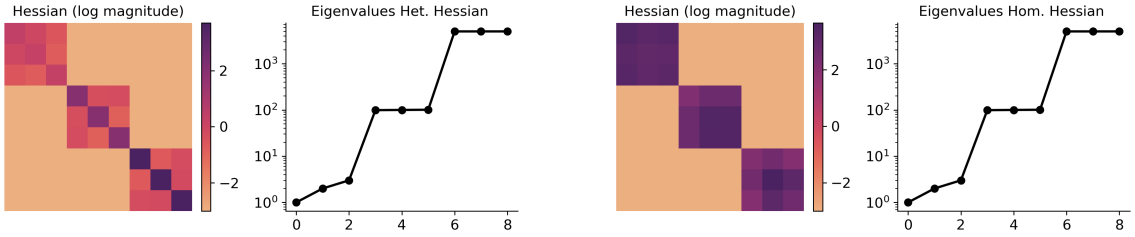

Figure 21: *(left) Heterogeneous and (right) Homogeneous Hessian considered in Figure 17.*

Our setup is inspired from the results and discussions in Zhang et al. [46], and uses the codebase of Orvieto and Gower [28]. We consider the loss

$$L(w) = \frac{1}{2} w^\top H w$$

where we construct the Homogeneous and Heterogeneous Hessians using the following procedure:

- We fix the eigenvalues, equal in both cases, to

$$\text{eig}(H_{\text{hom}}) = \text{eig}(H_{\text{het}}) = \{1, 2, 3, 99, 100, 101, 4998, 4999, 5000\}.$$

- We choose both Hessians to be block-diagonal, with blocks of size 3×3. The homogeneous Hessian has eigenvalues of different magnitude in each block, while the Heterogeneous keeps similar magnitudes in each block.

```
H_details_het = [[1,2,3],[99,100,101],[4998,4999,5000]]
H_details_hom = [[1,99,4998],[2,100,4999],[3,101,5000]]
```

- For each block, we apply a random rotation to the diagonal matrix of eigenvalues, specific to each block. Each rotation is sampled from the Haar measure by decomposition of a random 3×3 positive semidefinite matrix $AA^\top$, where $A \in \mathbb{R}^{3\times 3}$ has i.i.d. Gaussian entries.

The result is shown in Figure 21. Leraning rates for each method are tuned.

Next, to introduce stochasticity in this setting, we simply take the square root of the Hessian to define a 9×9 design matrix X:

$$H = X^\top X, \qquad X = H^{\frac{1}{2}},$$

and subsample a number (the batchsize) of rows of X at each iteration.

Additional learning rates for Figure 21 are reported in Figure 22.

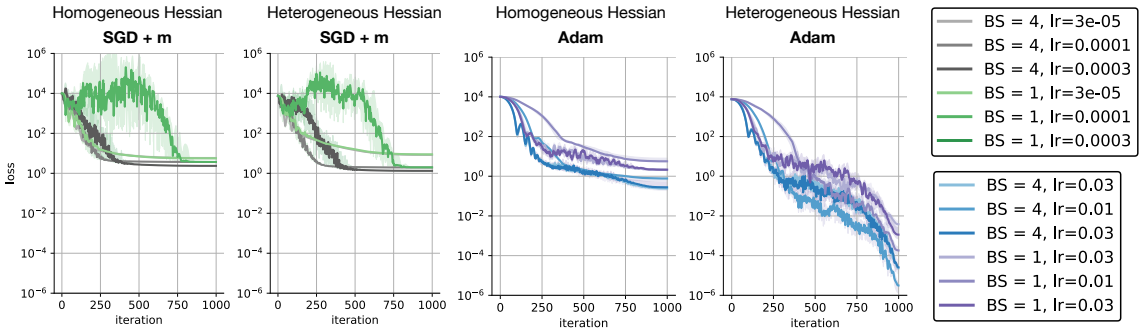

Figure 22: Complement to Figure 17.