EMBER: A KILOBYTE-STATE OPTIMIZER FOR TOKEN INTERFACES

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models learn continuous programs over discrete symbols, with the embedding table and LM-head acting as the read/write interface between them. We show that this interface has gradient geometry distinct from dense hidden weights which can be exploited to improve the Pareto frontier across supervised finetuning, RL, and pretraining, while only utilizing kilobytes of optimizer state. We introduce *Ember*, a lightweight optimizer for embedding and LM-head matrices that utilizes $\mathcal{O}(V + D)$ optimizer VRAM, instead of Adam's $\mathcal{O}(2VD)$, and forgoes the need to shard both token table optimizer states. On 50k-step pretraining across two datasets and three paired seeds, Ember lowers val loss by 0.08–0.09 nats relative to tuned AdamW; we also provide empirical evidence that Ember scales across batch size and parameter count. Finally, we provide a distributed Ember implementation that merges cleanly with existing ZeRO/FSDP setups (code to be released).

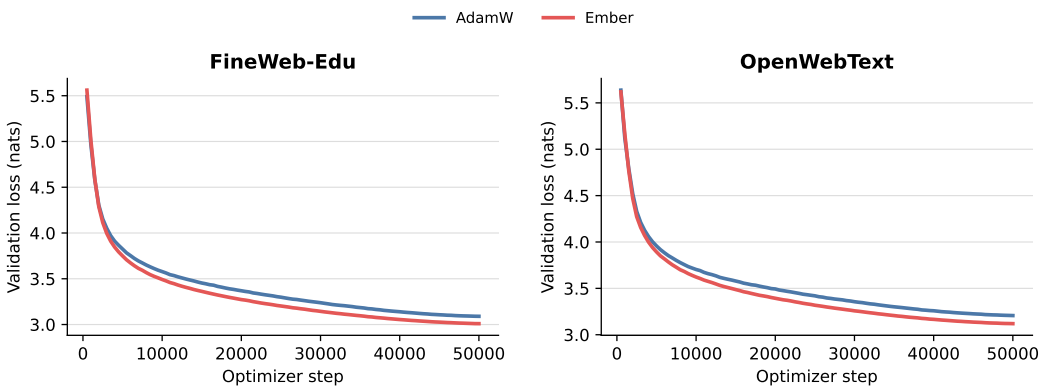


Figure 1: **Long-horizon pretraining.** Validation loss on FineWeb-Edu and OpenWebText over 50,000 optimizer steps; curves are means over three paired seeds. Only the optimizer on the untied input embedding and LM head changes, while the Transformer body uses the same AdamW setup in every run.

1 INTRODUCTION

Embedding and LM-head matrices, or "token interface" parameters, are currently optimized with Adam [2]. This is corroborated by a report from the current open source frontier, DeepSeek V4 [15], which only changed the optimizer for linear layers to Muon [16]. Of note, almost all recent proposed successors of Adam solely apply to linear layers and steer clear of the embedding table. Therefore improvements are pertinent as Adam requires significant distributed training engineering and doesn't fully leverage the geometry of token interfaces. In this paper, we introduce *Ember*, a lightweight optimizer for such Transformer parameters that remedies these and achieves competitive results.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2 METHODS

2.1 DERIVATION

We drop Adam's first moment and approximate its second-moment buffer with an outer product of row and column factors (Alg. 1, Fig. 6). Beyond the learning rate, the only hyperparameter is $\beta_2 = 0.999$, which we lift directly from the original Adam paper. Under matched settings, this update is equivalent to Algorithm 2 in [19], which precedes their final Adafactor recipe. Our contribution is its evaluation and implementation for token interfaces while retaining the body optimizer. We discuss the relationship in Related Work and compare with native Adafactor in Appendix A.6.

We begin from Adam's second-moment buffer: an exponential moving average of squared minibatch gradients. Fisher geometry motivates this form, but this statistic is not generally the model Fisher or the average of per-example gradient squares.

Linear-layer gradients from backpropagation have an outer-product structure. For a single-position forward pass $y = xW$, the weight gradient is $dW = x^\top dy$, an outer product of rank at most one. In an input embedding, x is one-hot, so each position contributes to just one token row. Summing over a batch preserves this row assignment, although the summed gradient need not remain rank one.

Intuitively, backpropagation accumulates gradients every time a token is selected. A token with higher frequency can therefore have a gradient ℓ_2 norm roughly proportional to its frequency, assuming its contributions have comparable magnitudes and point in similar directions. Measured row-gradient energy is strongly associated with token frequency ($r = 0.976$). Ember measures each row's squared gradients directly rather than assuming a frequency law; Appendix A.9 gives the mean and variance terms.

Row-only scaling assigns the same second-moment estimate to every feature in a token row. We add an $\mathcal{O}(D)$ column factor to capture feature-wise differences (Fig. 6, App. 3.4). The default algorithm and headline experiments use both factors; we use the row-only version to explain token frequency.

The product of the two factors has units of squared gradient squared. Dividing by their common grand mean restores the units of a second moment (App. A.10). The geometric-mean expression in Algorithm 1 gives the same normalizer.

Removing the first-moment EMA saves another $V \times D$ state values per token table. Adding it back has no consistent benefit in our two tested batch settings (Table 3); Figure 10 reports the gradient-autocorrelation diagnostic.

2.2 OPTIMIZING SPARSELY ACTIVATED PARAMETERS

When a row's gradient and RMS both scale with token frequency p, unnormalized updates scale with p, inverse-second-moment updates with $1/p$, and inverse-RMS updates are approximately frequency-independent. Ember's row factor uses the last normalization; Appendix A.3 explains this approximation.

3 RESULTS

3.1 PRETRAINING

Long-horizon pretraining. We trained Pythia-410M from scratch for 50,000 steps on FineWeb-Edu and OpenWebText (1.6384 billion tokens per run). For each corpus, AdamW and Ember's token-interface learning rates were selected independently from the same four-point grid before evaluation on three fresh paired seeds per method. At the frozen endpoint — the mean of 11 validation measurements from steps 45,000–50,000 — Ember improved held-out cross-entropy by 0.0802 nats on FineWeb-Edu and 0.0882 nats on OpenWebText, and was lower on every paired seed (Fig. 1). At the shared token-table shape, Ember removes approximately 824 MB of FP32 token-interface optimizer state. Full protocol and paired confidence intervals are in Appendix A.5.

Small-scale pretraining. In a separate 200-step GPT-2-small experiment with Muon on the linear layers and n=3 paired seeds, Ember matches or outperforms AdamW and improves over canonical

Algorithm 1 *Ember* for embedding and LM-head matrices. Row and column second-moment estimates form an outer-product preconditioner. We use $\beta_2 = 0.999$; learning rates are specified for each experiment.

Require: α: step size
Require: $\beta_2 \in [0, 1)$: momentum term for second-moment estimates
Require: $R_t(\theta)$: reward at timestep t (WLOG $R_t = -\mathcal{L}_t$ for loss minimization)
Require: $\theta_0 \in \mathbb{R}^{V \times D}$: initial parameters of the embedding or LM-head matrix
$r_0 \leftarrow \mathbf{0} \in \mathbb{R}^V$ ▷ Initialize row second-moment vector
$c_0 \leftarrow \mathbf{0} \in \mathbb{R}^D$ ▷ Initialize column second-moment vector
$t \leftarrow 0$ ▷ Initialize timestep
while θ_t not converged **do**
  $t \leftarrow t + 1$
  $g_t \leftarrow \nabla_\theta R_t(\theta_{t-1})$ ▷ Compute gradient through backprop
  $r_t \leftarrow \beta_2 r_{t-1} + (1 - \beta_2)\,\mathrm{mean}_j(g_t^2)$ ▷ Update row second moment
  $c_t \leftarrow \beta_2 c_{t-1} + (1 - \beta_2)\,\mathrm{mean}_i(g_t^2)$ ▷ Update column second moment
  $\hat{r}_t \leftarrow r_t / (1 - \beta_2^t)$ ▷ Bias-correct row estimate
  $\hat{c}_t \leftarrow c_t / (1 - \beta_2^t)$ ▷ Bias-correct column estimate
  $s_t \leftarrow \sqrt{\mathrm{mean}(\hat{r}_t)\ \mathrm{mean}(\hat{c}_t)}$ ▷ Geometric-mean normalization
  $\tilde{v}_t \leftarrow \hat{r}_t \hat{c}_t^\top / s_t$ ▷ Form factored preconditioner
  $\theta_t \leftarrow \theta_{t-1} + \alpha\, g_t / (\sqrt{\tilde{v}_t} + 10^{-8})$ ▷ Ascend the reward R_t
end while
return θ_t

Table 1: Supervised fine-tuning validation loss at step 250 across GPT-2 and Pythia model scales, with per-model embedding-optimizer state. Ember is slightly ahead on GPT-2 and within seed noise on Pythia, while reducing token-interface optimizer state from $\mathcal{O}(2VD)$ to $\mathcal{O}(V+D)$. State is FP32 optimizer buffers.

Model	Params (M)	AdamW	Ember	Δ	AdamW state	Ember state
distilgpt2	82	2.3670±.0076	**2.3611±.0072**	**−0.0059±.0004**	309 MB	199 KB
gpt2	124	2.0529±.0030	**2.0450±.0032**	**−0.0079±.0002**	309 MB	199 KB
gpt2-medium	350	1.7483±.0033	**1.7452±.0033**	**−0.0032±.0001**	412 MB	200 KB
gpt2-large	774	1.5937±.0074	**1.5925±.0075**	**−0.0012±.0001**	515 MB	201 KB
pythia-160m	160	2.5306±.0284	2.5175±.0325	−0.0132±.0318	618 MB	399 KB
pythia-1.4b	1400	1.4298±.0040	1.4298±.0039	−0.0001±.0003	1.65 GB	409 KB
pythia-2.8b	2800	1.3530±.0025	1.3525±.0026	−0.0005±.0006	2.06 GB	413 KB

Mean ± sd, n=3 paired seeds; bold: $|\Delta| > 2\sigma$.

Adafactor by 0.095 nats at batch 8 and 0.251 nats at batch 256, while retaining the same $\mathcal{O}(V+D)$ state complexity (Fig. 3, Table 3).

3.2 Model-size robustness

Model size. We compare model sizes using 250-step supervised fine-tuning, rather than long-horizon pretraining. Ember matches AdamW as model size increases while removing the $O(2VD)$ optimizer state on the 2 token tables (Fig. 12). On GPT-2, Ember improves validation loss at every scale. On Pythia, the mean loss difference favors Ember at 160M and is near zero at 1.4B and 2.8B (Table 1).

We visually depict the optimizer state benefit in Fig. 2. Notably, on Pythia-2.8B, AdamW's token-interface state reaches 2 GB, whereas Ember stays at 400 KB. This is a $4{,}900\times$ reduction for the same val loss.

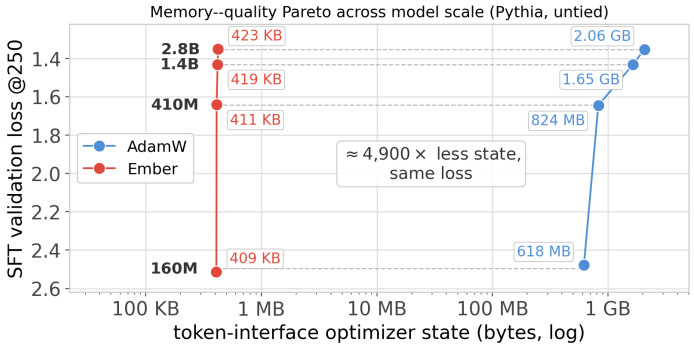


Figure 2: **The memory–quality gap widens with model scale.** The two optimizers reach the same loss at every scale, yet AdamW's state grows as $\mathcal{O}(2VD)$ — crossing 1 GB by 1.4B and reaching $\sim$2 GB at 2.8B — while Ember's $\mathcal{O}(V+D)$ state stays near 400 KB.

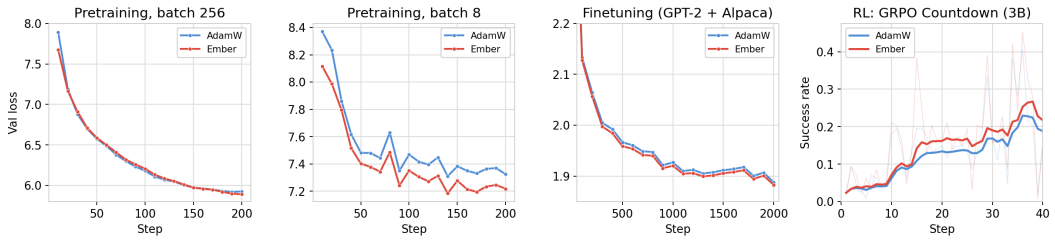


Figure 3: Overview of earlier experiments. Panels A and B compare pretraining at batch sizes 256 and 8, panel C shows supervised fine-tuning, and panel D shows GRPO post-training. Curves show representative runs; replicated endpoint comparisons are reported in the text and tables.

Reinforcement learning. We compare Ember against AdamW on the token tables when post-training Qwen2.5-3B-Instruct with GRPO [6] on the Countdown reasoning task, sweeping the rollout batch size from 64 to 256 (Fig. 4). The curves are comparable across the tested batch sizes; the endpoints at batches 64 and 256 are replicated over three seeds. Ember uses only 600 KB of optimizer state compared to AdamW's 2.5 GB (a 4200$\times$ reduction).

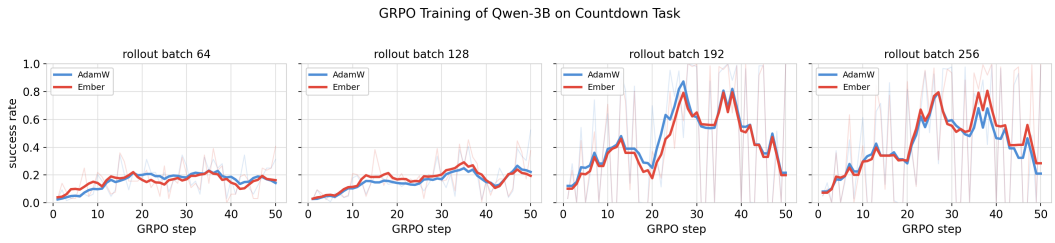


Figure 4: **Ember is competitive in reinforcement learning while removing $\mathcal{O}(2VD)$ optimizer state.** Comparisons at rollout batch sizes 64, 128, 192, and 256. Batches 64 and 256 are replicated over n=3 seeds; the intermediate sizes show representative runs.

3.3 Ablations

Bias correction. Bias correction improves stability and validation loss in the tested configuration (Fig. 5).

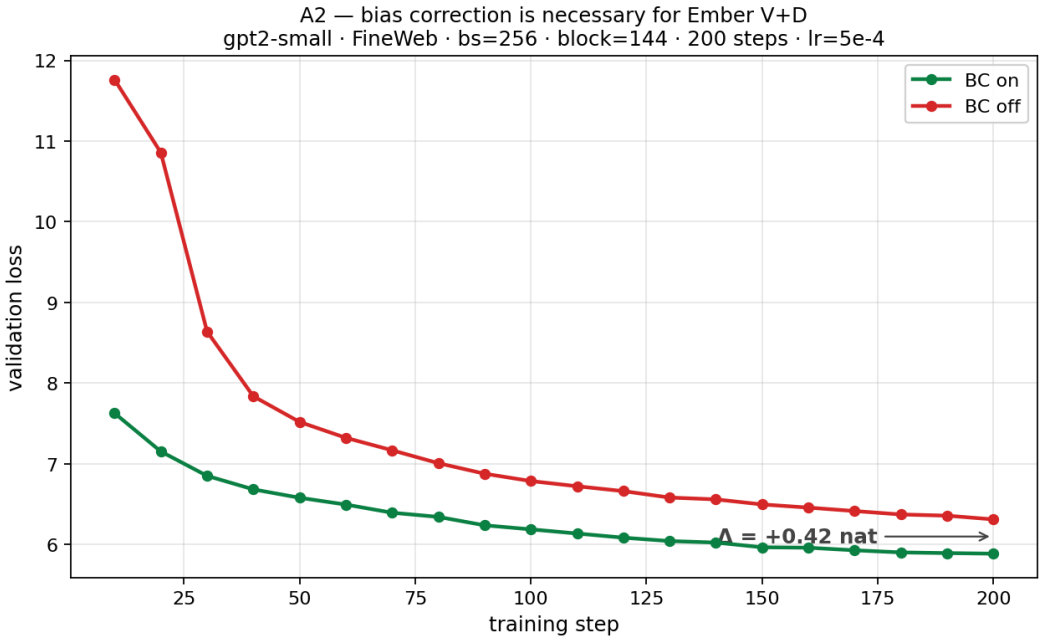


Figure 5: **Ablating bias correction in Ember.** Including the bias correction is crucial to performance and makes training much stabler while helping the optimizer reach lower val loss faster.

3.4 Decomposing the Squared Gradient

For a single token position, write the gradient contribution as

$$Z_{ij} = \delta_i x_j, \qquad Z = \delta x^\top.$$

For an input embedding, δ denotes the one-hot row selector and x the backpropagated feature gradient. For the LM head, δ is the softmax error and x the hidden state.

If the squared factors are independent, the per-position second moment separates:

$$\mathbb{E}[Z_{ij}^2] = \mathbb{E}[\delta_i^2]\mathbb{E}[x_j^2].$$

Without independence, the difference is $\mathrm{Cov}(\delta_i^2, x_j^2)$. This motivates a factored approximation, but does not prove it for the minibatch statistic. A batch gradient $G = \frac{1}{M}\sum_a Z_a$ contains cross-position terms when squared.

Ember approximates the EMA of $G \odot G$ using its row and column means. We test this approximation directly: over all 38M embedding parameters of a 500-step GPT-2-small run on FineWeb, the factored estimate tracks Adam's dense second moment with correlation 0.94 (Fig. 6).

4 Related Work

SGD [17] is the simplest form of gradient descent, and uses the backprop gradient scaled by the learning rate to update parameters. Adam [2] builds on it by dividing this by its RMS, and maintains a bias-corrected EMA of the 1st and 2nd moments which takes $\mathcal{O}(2VD)$ memory per token table.

Lion [18] shows momentum on just the gradient sign is competitive. Muon [16] forces singular values of the gradient matrix for each dense linear layer to be unitary, and is used by frontier models. Shampoo [20; 21] maintains full row and column Gram matrices and preconditions with $L^{-1/4}GR^{-1/4}$. Adafactor [19] factorizes Adam's second moment and removes first-moment state. We arrived at Ember independently, but its written update coincides with their earlier Algorithm 2 under matched settings. Their final recipe additionally uses relative steps, parameter scaling, update clipping, and scheduled decay. They also discuss token-frequency effects in embeddings (Sec. 3.2).

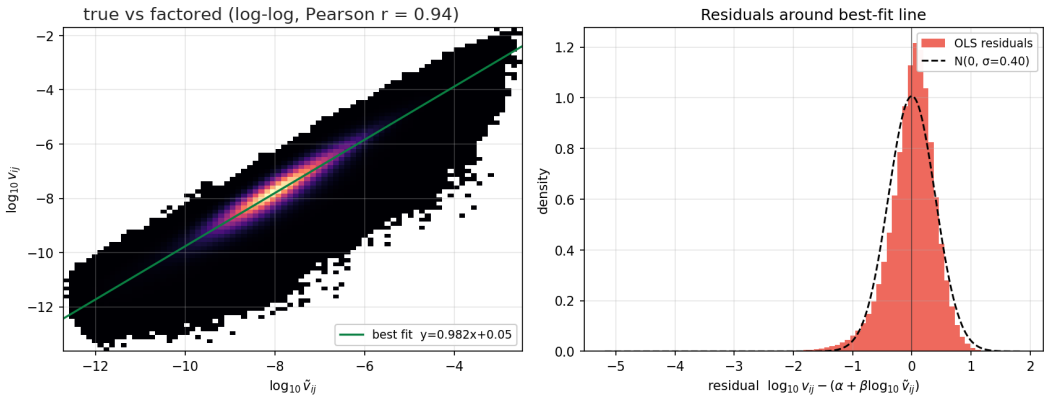


Figure 6: **A factored approximation to Adam's dense second moment on token tables.** *Left*: each point is one embedding parameter (i, j); the factored $\tilde{v}_{ij}$ tracks Adam's dense v_{ij}. Pointwise, the match holds across ten orders of magnitude ($n = 38$M entries) with a correlation of 0.94 and OLS slope 0.98. *Right*: log-space residuals concentrate near zero with no heavy tail, illustrating the approximation error. Analysis in App. 3.4.

Here we evaluate selective use on embedding and LM-head matrices in modern language-model training, with the body optimizer unchanged, and provide a distributed implementation. Our comparison with the final native recipe is in Appendix A.6.

SM3 [22] also uses row/column-sized buffers, but takes a minimum over their estimates rather than an outer product. Coupled Adam [1] studies embedding-specific corrections to Adam's second moment.

Other memory-saving approaches include quantizing optimizer states (8-bit Adam [38]), low-rank gradient projection (GaLore [39]), and sharing second-moment estimates within parameter blocks (Adam-mini [40]). We do not evaluate these methods here; our measured comparisons are against AdamW and the native Adafactor recipe.

[25] claims that linear layers are fundamentally symmetric, so a good optimizer should leverage this rather than update its parameters as if they were a 1D concatenated vector. Modula [5; 14] was one of the first to present a grand unified theory of optimizers, mathematically grounded in norm duality. They derive that row-normalization is optimal for embeddings, corroborating the analysis of this paper.

5 Discussion

We provide a distributed version of Ember compatible with ZeRO [28]; an anonymized code snapshot will accompany the submission as supplementary material. Because Ember's optimizer state is $O(V + D)$, the embedding table optimizer state no longer requires sharding, simplifying distributed training setups.

5.1 Practical implications

Faster iteration. The overwhelming majority of experimentation and finetuning is done on models $\leq$ 7B parameters where Adam's embedding state is a genuine binding constraint. For example, for Qwen2.5-7B, Adam stores roughly $8.72\,\mathrm{GB}$ VRAM for the token interface alone. Ember reduces this to about $1.2\,\mathrm{MB}$. Lower optimizer-state memory can make single-device experiments more practical. In a separate modded-nanoGPT experiment using Ember only on the value-embedding tables, peak allocated VRAM fell by approximately $2.9\,\mathrm{GiB}$ in each of three comparisons. This is a different setup from the input-embedding/LM-head state accounting above. In the FineWeb-Edu Pythia-410M runs, median tokens per second over the final 90% of training differed by less than 0.5% between Ember and AdamW in each of the three seed pairs.

Cleaner engineering. The optimizer state for LayerNorm and the embedding tables are all 1-D and can fit on a single GPU across even the largest models. Only linear layers/MoE require sharding, which can be done more simply as they scale cleanly with model depth and number of experts per layer.

Multiple embedding tables. The savings scale with $V \cdot D$, substantial for large-vocabulary or multimodal models where modalities each carry their own embedding table. Google's recent Gemma 3n models include Per-Layer Embedding [32], where each layer gets its own embedding table; presumably, these would have significant optimizer state and Ember would be even more useful there.

5.2 Conclusion

Ember reduces token-interface optimizer state from $O(2VD)$ to $O(V+D)$. In our 50k-step pretraining experiments, it improves validation loss over tuned AdamW on both datasets. The fine-tuning and RL results provide additional evidence for using this low-state update on token tables while retaining the body optimizer.

Reproducibility Statement

Experimental protocols and additional analyses are documented in the appendix. An anonymized snapshot of the implementation and experiment scripts will accompany the submission as supplementary material. The public repository URL will be added after the review period.

AI Use Statement

Generative AI tools were used to help refine hypotheses, design and debug experiments, implement code, analyze and interpret results, survey related work, produce figures, and organize and edit the manuscript. The author reviewed the AI-assisted work and takes responsibility for the final content, including all text, claims, code, analyses, and artifacts.

REFERENCES

[1] Felix Stollenwerk and Tobias Stollenwerk. Better embeddings with Coupled Adam. In *Proceedings of ACL*, 2025.

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[3] Sham M. Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems 14*, pages 1531–1538. MIT Press, 2001.

[4] Frederik Kunstner, Robin Yadav, Alan Milligan, Mark Schmidt, and Alberto Bietti. Heavy-tailed class imbalance and why Adam outperforms gradient descent on language models. In *Advances in Neural Information Processing Systems*, 2024.

[5] Jeremy Bernstein and Laker Newhouse. Modular duality in deep learning. In *arXiv preprint arXiv:2410.21265*, 2024.

[6] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[7] N. N. Chentsov. *Statistical Decision Rules and Optimal Inference*. Translations of Mathematical Monographs, Vol. 53. American Mathematical Society, Providence, RI, 1982.

[8] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[9] Shun-ichi Amari. *Information Geometry and Its Applications*. Applied Mathematical Sciences, Vol. 194. Springer Japan, Tokyo, 2016.

[10] Alexander Ly, Maarten Marsman, Josine Verhagen, Raoul P. P. P. Grasman, and Eric-Jan Wagenmakers. A tutorial on Fisher information. *Journal of Mathematical Psychology*, 80:40–55, 2017.

[11] Frank Nielsen. An elementary introduction to information geometry. *Entropy*, 22(10):1100, 2020.

[12] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[13] Guillaume Desjardins, Karen Simonyan, Razvan Pascanu, and Koray Kavukcuoglu. Natural neural networks. In *Advances in Neural Information Processing Systems*, 2015.

[14] Jeremy Bernstein. Modular manifolds. *Thinking Machines Lab: Connectionism*, 2025. `https://thinkingmachines.ai/blog/modular-manifolds/`, doi:10.64434/tml.20250926.

[15] DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *Technical Report*, 2026.

[16] Keller Jordan. Muon: An optimizer for LLMs. *Keller Jordan's Blog*, 2024. URL: `https://kellerjordan.github.io/posts/muon/`.

[17] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[18] Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. Symbolic discovery of optimization algorithms. In *Advances in Neural Information Processing Systems*, 2023.

[19] Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning (ICML)*, 2018.

[20] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning (ICML)*, 2018.

[21] Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable second order optimization for deep learning. *arXiv preprint arXiv:2002.09018*, 2020.

[22] Rohan Anil, Vineet Gupta, Tomer Koren, and Yoram Singer. Memory-efficient adaptive optimization. In *Advances in Neural Information Processing Systems*, 2019.

[23] Frederik Kunstner, Lukas Balles, and Philipp Hennig. Limitations of the empirical Fisher approximation for natural gradient descent. In *Advances in Neural Information Processing Systems*, 2019.

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[25] Tim Tsz-Kit Lau and Weijie Su. Symmetry-compatible principle for optimizer design: Embeddings, LM heads, SwiGLU MLPs, and MoE routers. *arXiv preprint arXiv:2605.18106*, 2026.

[26] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning (ICML)*, 2018.

[27] Jorge Nocedal and Stephen J. Wright. Numerical optimization. Springer, 2nd edition, 2006.

[28] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. ZeRO: Memory optimizations toward training trillion parameter models. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC20)*, 2020.

[29] M. S. Bartlett. Approximate confidence intervals. In *Biometrika*, 1953.

[30] Francis Ysidro Edgeworth. The law of error. In *Transactions of the Cambridge Philosophical Society*, 20:33–66, 1905.

[31] Shun-ichi Amari. *Differential-Geometrical Methods in Statistics*. Lecture Notes in Statistics, Volume 28. Springer, 1985.

[32] Google AI. *Gemma 3n model overview*. 2025. `https://ai.google.dev/gemma/docs/gemma-3n`. Accessed: 2026-06-17.

[33] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[35] Ian J. Goodfellow, Oriol Vinyals, and Andrew M. Saxe. Qualitatively characterizing neural network optimization problems. In *International Conference on Learning Representations (ICLR)*, 2015.

[36] Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

[37] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, 2018.

[38] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via blockwise quantization. *arXiv preprint arXiv:2110.02861*, 2021.

[39] Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. GaLore: Memory-efficient LLM training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

[40] Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Diederik P. Kingma, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more. *arXiv preprint arXiv:2406.16793*, 2024.

A SUPPLEMENTARY MATERIAL

We begin with some preliminaries on information geometry:

The Fisher metric. Under regularity assumptions, the expected negative Hessian of the log likelihood equals the score second moment, with both expectations taken under the model:

$$-\mathbb{E}_{p_\theta}\left[\frac{\partial^2}{\partial\theta_i^2}\log p_\theta(x)\right] = \mathbb{E}_{p_\theta}\left[\left(\frac{\partial}{\partial\theta_i}\log p_\theta(x)\right)^2\right].$$

This is an expectation identity, not an equality between derivatives for each example [9; 10; 11] (App. A.1).

Canonical Geometry of Cross-Entropy. The Fisher metric is especially elegant because it is not merely useful as an approximation, but it is the exact second-order term of the Kullback–Leibler divergence:

$$D_{\mathrm{KL}}\left(p_\theta \parallel p_{\theta+d\theta}\right) = \frac{1}{2}d\theta^\top F(\theta)d\theta + \mathcal{O}(\|d\theta\|^3).$$

This local geometry motivates natural-gradient methods [8; 3; 12; 13]; it does not by itself establish which preconditioner works best in training.

Which second moment? The model Fisher averages score outer products under the model distribution. The empirical Fisher averages loss-gradient outer products over observed examples. Adam and Ember instead track an EMA of squared aggregated minibatch gradients. These objects need not coincide [23]. We use Fisher geometry as motivation, not as an identity for Ember's buffer.

A.1 WHEN FISHER AGREES WITH CURVATURE

Under standard regularity conditions, the model Fisher equals the expected negative Hessian of the log likelihood, with the expectation taken under the model. Relating this to the data-loss Hessian additionally requires agreement between the model and data distributions. Stationarity alone,

$$\mathbb{E}_{\mathrm{data}}[\nabla_\theta \ell] = 0, \tag{1}$$

does not establish that agreement. Our experiments do not require a Fisher–Hessian identity.

A.2 SECOND-MOMENT NORMALIZATION

Adam divides its first-moment estimate by the square root of a second-moment estimate. Ember uses the current gradient and a factored second-moment estimate instead. Both normalize gradient scale; neither operation by itself establishes a natural-gradient or trust-region interpretation.

A.3 OPTIMIZING SPARSELY ACTIVATED PARAMETERS

For an input embedding, row i receives contributions only when token i appears. Under the coherent-mean regime described in Appendix A.9, the expected row gradient scales with p_i and its root second moment also scales with p_i. Dividing by the latter can therefore reduce frequency-dependent differences in update scale.

The three update scalings follow directly. If $g_i \propto p_i$ and its second-moment estimate $v_i \propto p_i^2$, then $g_i \propto p_i$, $g_i/v_i \propto 1/p_i$, and $g_i/\sqrt{v_i} \propto p_i^0$. The last ratio is dimensionless, but frequency cancellation alone does not establish isotropic gradient covariance or an identity with the model Fisher.

This argument is conditional: when the mean contribution is zero or the variance term dominates, the same cancellation does not follow. Ember estimates the observed second moment rather than imposing an inverse-frequency correction. The LM head receives dense softmax gradients and needs a separate analysis (App. A.12).

A.4 FURTHER WORK

Future work could test which other parameter classes benefit from low-state second-moment approximations and under what training regimes.

Table 2: Pythia-410M long-horizon pretraining. Losses are frozen endpoint means over three paired seeds; Δ is Ember minus AdamW in nats, with a paired 95% t interval.

Corpus	AdamW LR	Ember LR	AdamW	Ember	Δ [95% CI]
FineWeb-Edu	2×10^{-3}	10^{-3}	3.09699	**3.01677**	$\mathbf{-0.08021}$ $[-0.08646, -0.07396]$
OpenWebText	10^{-3}	2×10^{-3}	3.21497	**3.12680**	$\mathbf{-0.08817}$ $[-0.10238, -0.07395]$

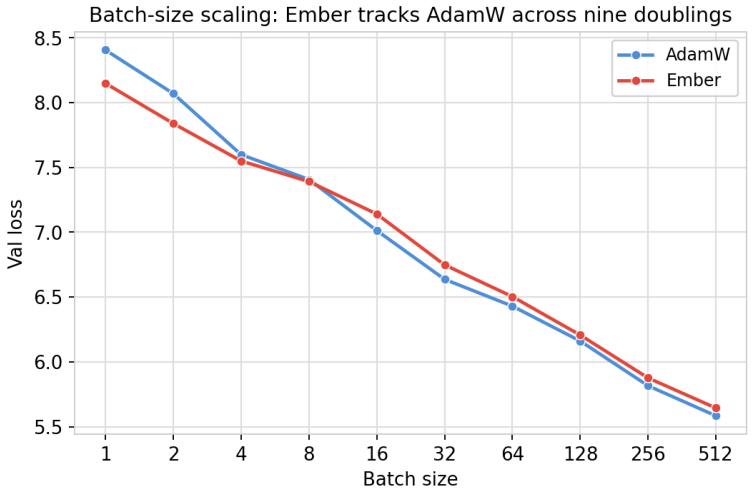


Figure 7: Batch-size scaling across nine doublings. Ember leads at the smallest batches, while AdamW leads at batches 16–256 (n=3 paired seeds; plotted curve is a representative run).

A.5 Experimental details

Long-horizon Pythia-410M pretraining. All runs use global batch size 32, sequence length 1024, and a 1,000-step warmup followed by cosine decay to 10% of the peak learning rate. The Transformer body uses the same AdamW configuration in every run (learning rate 3×10^{-4}, $\beta = (0.9, 0.95)$, and weight decay 0.1), while the token tables use zero weight decay. For each corpus, AdamW and Ember receive the same token-interface learning-rate grid, $\{5\times10^{-4}, 10^{-3}, 2\times10^{-3}, 4\times10^{-3}\}$, in pilot runs before evaluation on three fresh paired seeds. The frozen endpoint is the mean of the 11 validation measurements at steps 45,000, 45,500, ..., 50,000. OpenWebText uses a tuning shard disjoint from its final training shards and an untouched validation shard.

At the shared token-table shape, Ember's two FP32 row-and-column buffers replace 205,942,528 FP32 AdamW state elements, removing 823,770,112 bytes (approximately 824 MB) of token-interface optimizer state.

Scaling parameter count. All model-scaling runs finetune on Alpaca SFT for 250 steps with effective batch size 128, learning rate 2×10^{-5} with cosine decay, and bf16 precision. The transformer body uses AdamW throughout; only the *token-embedding / LM-head* optimizer is varied.

Scaling batch size. We train GPT-2-small ($V = 50257$, block 144) on FineWeb with both token tables untied, sweeping the batch size across nine doublings from 1 to 512 for 300 steps (3 paired seeds). The replicated comparison uses step 195, the same checkpoint as the earlier representative curve in Figure 7; this measures early-training performance, not final convergence.

We find that Ember obeys the same near-linear law in log-batch across nine doublings (Figure 7). Over n=3 paired seeds, Ember performs better at the smallest batches (paired $\Delta = -0.294 \pm 0.040$ at batch 1, -0.159 ± 0.077 at batch 2), while AdamW has lower loss at batches 16–256.

Autoregressive image generation. Ember is competitive in computer vision tasks: on LlamaGen image-AR, using $1500\times$ less optimizer state, Ember is virtually indistinguishable from Adam (Fig. 9). Setup in Appendix A.5.

LlamaGen is a vanilla decoder-only transformer that generates images autoregressively. Instead of predicting over discrete language tokens, it trains its own learnable embedding table to look up within a frozen VQGAN codebook. We compare Adam and Ember at the same fixed learning rate of 1×10^{-4}, rather than comparing independently tuned optima.

A.6 Comparison to Adafactor

The native Adafactor baseline uses the final recipe: relative steps, parameter scaling, RMS update clipping, and scheduled decay. Ember uses a fixed second-moment decay with explicit bias correction and an externally chosen learning rate. Table 3 compares these complete recipes over 200 steps; it does not isolate which difference causes the loss gap. The 50k-step experiments compare Ember with AdamW, not Adafactor.

Relation to Algorithm 2. Let $q_t = 1 - \beta_2^t$ and let m_t be the common grand mean of Ember's uncorrected row and column EMAs. Algorithm 2 in [19] stores sums rather than means, so its buffers satisfy $R_t = Dr_t$ and $C_t = Vc_t$. Hence both reconstructions give

$$\tilde{v}_{t,ij} = \frac{r_{t,i} c_{t,j}}{q_t m_t}.$$

With the same gradient history, zero initialization, decay, learning rate, and epsilon convention, the parameter updates agree. This identity assumes $m_t > 0$; implementations must define handling of an all-zero history.

Table 3: Recipe comparison and component ablations: validation loss at step 200 (GPT-2-small / FineWeb, Muon on linear layers), mean $\pm$ across-seed sample sd over 3 paired seeds; Δ is the paired per-seed difference versus AdamW. Canonical Adafactor uses its native recipe (relative step size, update clipping, β_2 schedule).

	batch 8		batch 256	
Token-table optimizer	val@200	Δ	val@200	Δ
AdamW	7.354$\pm$.030	—	5.829$\pm$.002	—
Canonical Adafactor	7.302$\pm$.030	$-.052 \pm .032$	6.093$\pm$.024	$+.264 \pm .026$
Ember	7.207$\pm$.023	$-.147 \pm .032$	5.842$\pm$.020	$+.013 \pm .021$
Ember + first-moment EMA	7.237$\pm$.031	$-.117 \pm .040$	5.813$\pm$.030	$-.016 \pm .033$
Ember row-only	7.312$\pm$.021	$-.042 \pm .030$	5.890$\pm$.017	$+.061 \pm .016$

Ember improves over the tested native recipe by 0.095 nats at batch 8 and 0.251 nats at batch 256. The separate bias-correction ablation measures its effect within Ember.

A.7 Supplementary figures

The following figures compare the measured principal-component projections with low-degree polynomial fits.

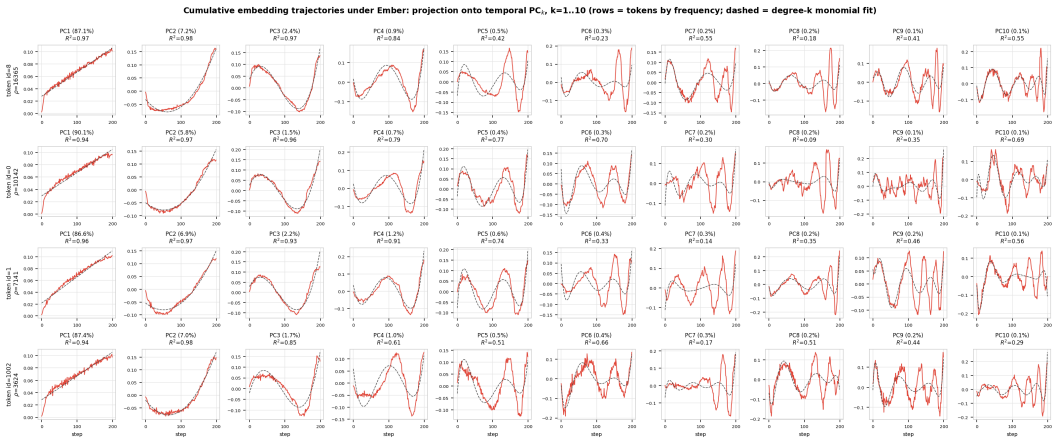


Figure 8: For each of four monitored tokens, we stack its embeddings over training and compute the SVD. Each panel shows a principal-component projection. The observed projections resemble low-degree polynomials, and the leading component explains roughly 90% of trajectory variance.



Figure 9: Autoregressive image generation results. Ember matches Adam while using substantially less optimizer state.

A.8 Ablations

Nesterov look-ahead ablation. We ran a 3×3 grid over $\beta_2 \in \{0.999, 0.95, 0.9\}$ and learning rate $\mathrm{lr} \in \{5 \times 10^{-4}, 10^{-3}, 2 \times 10^{-3}\}$, and consistently found that injecting the current g^2 estimate into the denominator hurts performance across all learning rates. Intuitively, the second moment acts as a smooth scale/curvature estimate and should not react to instantaneous gradients. Canonical $\beta_2 = 0.999$ was already optimal.

A.9 Token frequency and squared batch gradients

Consider an input-embedding row i. For each of M independent positions, let I_a indicate token i and u_a denote its gradient contribution when selected. Write

$$\Pr(I_a = 1) = p_i, \qquad \mu_i = \mathbb{E}[u_a \mid I_a = 1], \qquad S_i = \mathbb{E}[u_a u_a^\top \mid I_a = 1].$$

For the mean-reduced batch gradient $\bar{g}_i = M^{-1} \sum_a I_a u_a$,

$$\mathbb{E}[\bar{g}_i] = p_i \mu_i, \qquad \mathbb{E}[\bar{g}_i \bar{g}_i^\top] = \frac{p_i}{M} S_i + \frac{M-1}{M} p_i^2 \mu_i \mu_i^\top.$$

The first term is linear in frequency. The second is quadratic and requires a nonzero conditional mean. Equivalently, writing $S_i = \mathrm{Cov}(u_a \mid I_a = 1) + \mu_i \mu_i^\top$, the count-variance term is $p_i(1 - p_i)\mu_i \mu_i^\top / M$. For batch-summed gradients, multiply the second-moment expression by M^2.

The quadratic term can dominate for frequent tokens with sufficiently aligned contributions; it vanishes when $\mu_i = 0$. Independence across positions is an approximation for language batches. Our measured association between row-gradient energy and frequency ($r = 0.976$) motivates row-wise scaling, but does not identify an exponent. Ember measures the squared gradients directly and does not assume a global p_i^2 law.

A.10 Outer Product Unit Alignment

Let A_t be the bias-corrected EMA of $G_t \odot G_t$. Ember stores its row means $\hat{r}$ and column means $\hat{c}$. Both grand means equal $\bar{A}_t$:

$$\mathrm{mean}_i \, \hat{r}_i = \mathrm{mean}_j \, \hat{c}_j = \bar{A}_t.$$

Thus the geometric-mean normalizer gives

$$s_t = \sqrt{\mathrm{mean}(\hat{r}) \, \mathrm{mean}(\hat{c})} = \bar{A}_t, \qquad \tilde{v}_{ij} = \frac{\hat{r}_i \hat{c}_j}{\bar{A}_t}.$$

Each factor has units $[g]^2$, so division by $\bar{A}_t$ restores the units of a second moment.

If $A_{t,ij} = a_i b_j$ is exactly rank one with positive grand mean, then $\hat{r}_i = a_i \bar{b}$, $\hat{c}_j = \bar{a} b_j$, and $\bar{A}_t = \bar{a}\bar{b}$, giving $\tilde{v}_{ij} = A_{t,ij}$.

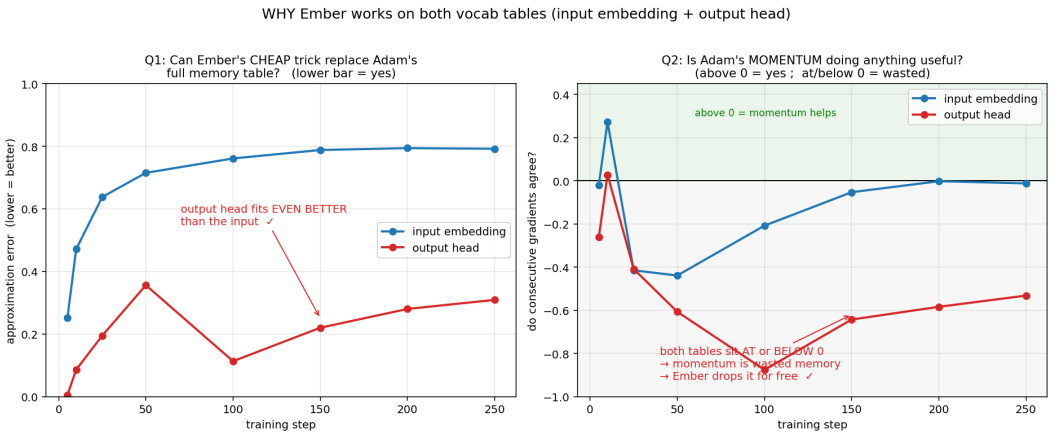


Figure 10: **Gradient diagnostics on token tables.** We instrument gradients of the untied input embedding $E_{\text{in}} \in \mathbb{R}^{V \times d}$ and output head $W_{\text{out}} \in \mathbb{R}^{V \times d}$ during GPT-2 small training on FineWeb. Left: Ember's row-column factored second moment approximates the dense squared-gradient structure, where the y-axis is relative error. Right: temporal gradient autocorrelation is near zero or negative. Table 3 directly evaluates the effect of adding first-moment momentum.

A.11 Related Normalization Methods

A.11.1 Muon: Spectral Normalization

For a matrix H with compact SVD $H = U_r \Sigma_r V_r^\top$ over its nonzero singular values,

$$H(H^\top H)^{\dagger/2} = U_r V_r^\top,$$

where $\dagger/2$ denotes the inverse square root on the nonzero spectrum. This is the polar normalization used in Muon, typically applied to a momentum-adjusted gradient and approximated numerically. It normalizes singular values rather than individual coordinates. The Gram matrix $H^\top H$ is not the full Fisher block on the vectorized parameters.

Figure 11: Earlier embedding-optimizer ablations at two batch sizes for GPT-2 small / FineWeb. The batch-8 row-only run uses learning rate 2×10^{-3}, versus 10^{-3} in the three-seed comparison in Table 3; the two row-only values are not estimates under the same settings. The row-only version is close to Ember at batch 256 but worse at batch 8; column-only is worse at both batches.

Embedding optimizer	Val@200 (bs=8)	Val@200 (bs=256)	State
AdamW	7.325	5.897	$2VD$
Muon	7.354	6.250	VD
Adafactor	7.311	6.137	$V+D$
Ember	**7.219**	**5.890**	$V+D$
Ember row-only	7.665	5.911	V
Ember col-only	10.899	7.958	D

A.11.2 SHAMPOO: TWO-SIDED PRECONDITIONING

Shampoo [20; 21] accumulates row and column Gram matrices,

$$L = \sum_t G_t G_t^\top, \qquad R = \sum_t G_t^\top G_t,$$

and preconditions by $L^{-1/4} G R^{-1/4}$, with regularization for singular matrices. Vectorization gives $(R \otimes L)^{-1/4} \operatorname{vec}(G)$ for symmetric positive-definite factors. This is a structured preconditioner, not in general the same Fisher conditioner as Adam or Ember.

A.12 LM-HEAD GRADIENTS

For hidden state h_a, target y_a, and predicted token probability $q_{a,i}$, the batch-summed gradient of LM-head row i is

$$G_i = \sum_{a=1}^{M} (q_{a,i} - \mathbf{1}[y_a = i])\, h_a.$$

Unlike input embeddings, each row receives a softmax contribution even when its token is not the target. The target and softmax terms can cancel, so a target-count argument alone does not determine the full gradient's frequency scaling.

Under independent positions, let $z_{a,i} = (q_{a,i} - \mathbf{1}[y_a = i])h_a$ have mean ν_i and covariance Σ_i. Then

$$\mathbb{E}[G_i G_i^\top] = M\Sigma_i + M^2 \nu_i \nu_i^\top.$$

This separates variance and coherent-mean contributions, but does not imply rank-one structure across vocabulary and feature dimensions. Using row-column moments for the LM head is an empirical approximation evaluated in our untied-token-table experiments.

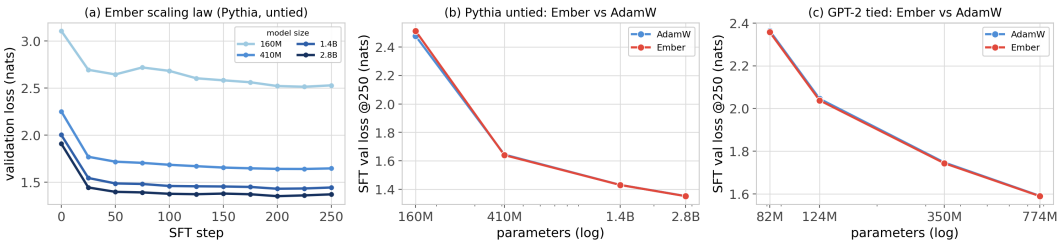


Figure 12: **Model-scaling results for Ember on token tables.** Validation loss after 250 supervised fine-tuning steps. n=3 paired seeds per point. Training setup in Appendix A.5.

A.13 DISTRIBUTED EMBER

Replicate instead of shard. Adam's $\mathcal{O}(2VD)$ fp32 state is roughly $4\times$ larger than the bf16 table it serves, so frameworks like ZeRO/FSDP split it across devices and every checkpoint records that

split, making resumption on a different cluster size a resharding exercise. Ember's $\mathcal{O}(V+D)$ state is a few hundred KB: each device keeps a full copy, and a checkpoint is one small file that loads anywhere.

The table can skip sharding entirely. Since the optimizer state is what usually forces token tables into sharded training, removing it lets the table's weights and gradients train fully replicated at moderate scale.

Persistent state and working memory. Only the $V+D$ factors persist between steps. Materializing the denominator still needs $O(VD)$ temporary memory, so the state bound is not a bound on peak VRAM. Input-embedding gradients can be row-sparse; LM-head gradients are generally dense. The algorithm does not require sparse execution.

A.14 Optimization trajectories

We stack each monitored token's embedding over training into a $D \times T$ matrix and examine its singular vectors (Fig. 13, 8). In these runs, the leading component explains roughly 90% of trajectory variance, and several component projections resemble low-degree polynomials. This describes the measured parameter paths; it does not establish that the loss landscape is one-dimensional or free of local optima.

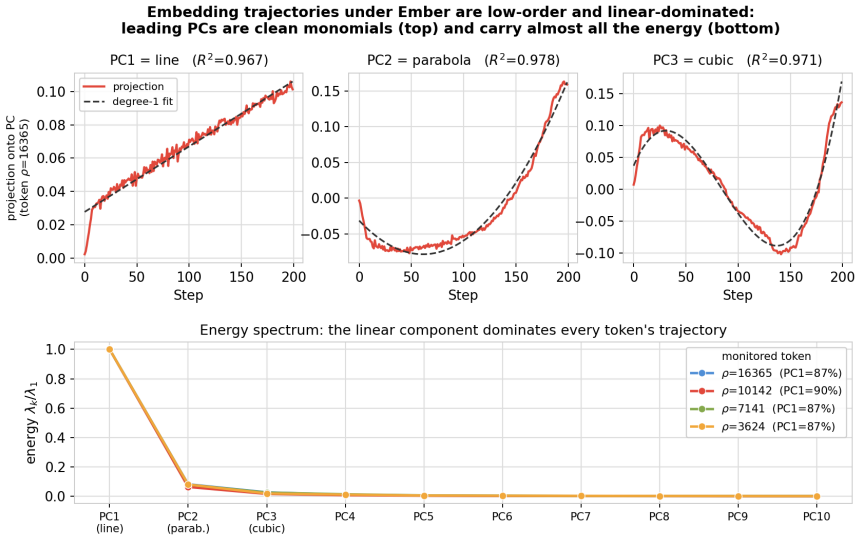


Figure 13: **Polynomial fits to token optimization trajectories.** Fits to the first three principal-component projections and their energy spectrum. The leading component dominates in these runs. More examples are in Figure 8.