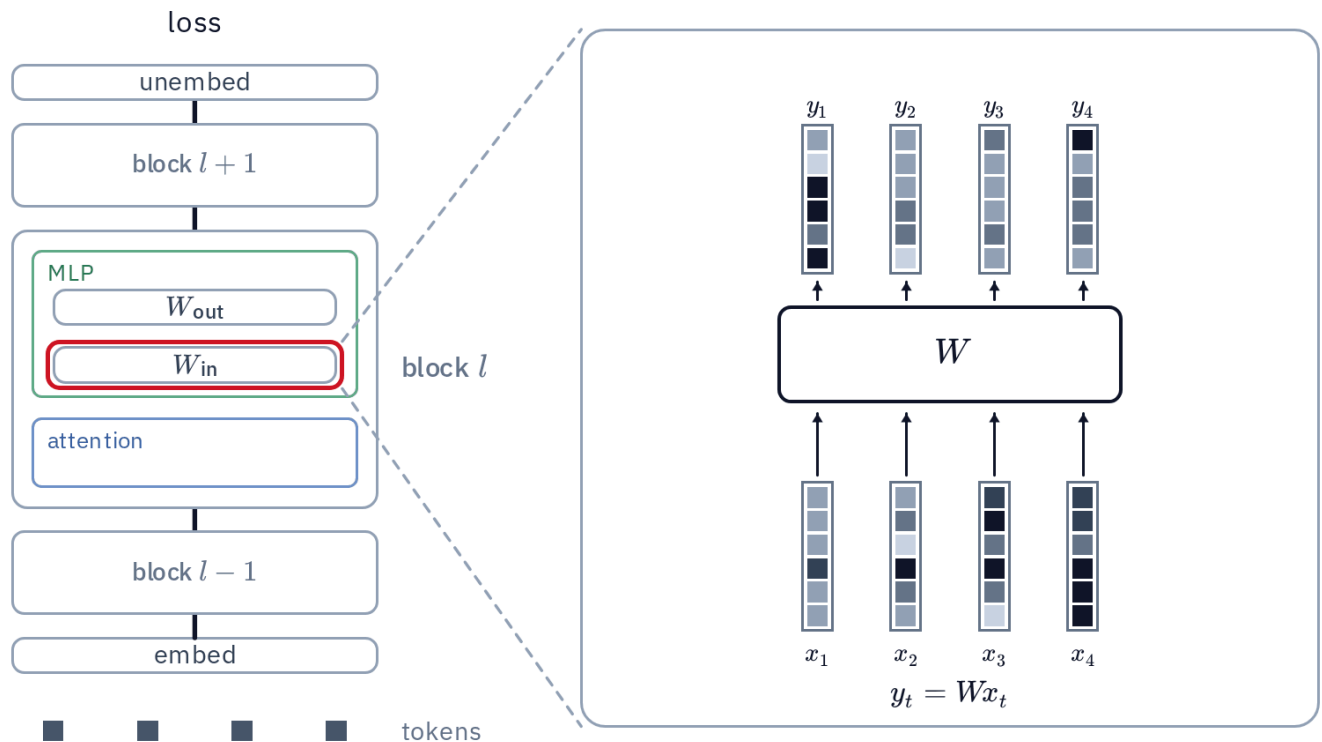


Dust: Pretraining Transformers Without Backpropagation

Samip Dahal, Bishwas Mandal, Serdar Gülbahar, Akshay Vegesna

October 2026 · Correspondence to s@qlabs.sh · [Code](#) · [Cite](#)



TL;DR

- We present the first zeroth-order method that is competitive with backprop at pretraining transformer language models. Dust perturbs activations (node perturbation) independently at every token, so each token is a *virtual* population member and one forward pass evaluates them all in parallel.
- Dust approximates backprop closely at large population (i.e. substantially more compute) and in multiple settings even *exceeds* it. This hints that in a compute-rich regime we might be able to surpass backprop.

- Dust is orders of magnitude more efficient than weight-space ES. From 1M tokens up, Dust is on the order of 10^3 to 10^4 times more efficient than a transformer implementation of EGGROLL, a state-of-the-art ES method, based on our extrapolations.
- Zeroth-order methods are widely believed not to scale to large networks. Strikingly, we find larger models are more population-efficient, not less: a 243M-parameter model outperforms a $120\times$ smaller model at most population sizes.
- Dust’s gradient estimates align better with backprop’s as population grows, and stay well aligned at every scale we test, up to 1B tokens, which is encouraging for scaling.

CONTENTS

TL;DR

1 Introduction

2 Method

2.1 Activation-Space Perturbation

2.2 Credit Assignment

2.3 Interference and Tuning

3 Pretraining Without a Backward Pass

3.1 Setup

3.2 Main Results

3.3 Dust Under Adam

4 Search in High-Dimensional Space

4.1 Overparameterization

4.2 Emergence of Backprop-Like Gradients

5 Conclusion

6 Related Work

References

Appendix

1 Introduction

Deep learning has been built around backprop, the only credit assignment algorithm capable of training modern neural nets, including transformer-based language models. Backprop requires differentiability and produces first-order gradients, and deep learning’s architectures, optimizers, and hardware have co-evolved around this constraint.

However, as the amount of compute available in the world increases, we might prefer more generic and brute-force learning algorithms based on *search* over inductive biases like differentiability, backprop, and approximations of higher-order gradients. The bitter lesson (Sutton, 2019) is that general methods that scale with compute eventually win, and AlphaGo Zero (Silver et al., 2017) is the obvious example. Bootstrapping AlphaGo on human data helped the network learn faster initially, but with a lot of computation the purely self-play network overtook it. Similarly, differentiability and backprop might be good inductive biases in the low-compute regime, where they make learning efficient, but in the high-compute regime they limit the space of architectures that work. Even within an architecture, gradient-based methods fail to explore the loss landscape optimally (Liu et al., 2020). This might also explain why current neural nets require massive amounts of data to generalize. A more flexible credit assignment algorithm based on search is likely an important step towards much better generalization.

In this paper, we aim to replace backprop with a learning algorithm based much more on brute-force computation and much less on analytic structure. We call it Dust. Dust is a zeroth-order optimization algorithm that perturbs activations, rewards each perturbation by how much it lowers the loss, and averages the reward-weighted perturbations over a population to estimate the gradient. Traditional ES methods that perturb weights (Salimans et al., 2017), like EGGROLL (Sarkar et al., 2025), scale with population, but scaling the population is costly because each member must be materialized and evaluated. We remove both costs with the concept of *virtual population*, where we avoid materializing every member by bypassing weight space entirely and instead perturb activations, as in node perturbation (Werfel et al., 2003; Widrow and Lehr, 1990). We do so independently at every token, so each token is a member and one forward pass evaluates them all in parallel.

Activations are a more interesting space to search over than weights. Mechanistic interpretability has shown that reasoning, whether verbalizable or not, lives in the activations (Gurnee et al., 2026; Lindsey et al., 2025), which means this approach could turn training into a search over latent reasoning (Vegesna and Dahal, 2025). We then pair the activation-space perturbation with a very generic credit assignment rule that assigns

different token-level rewards to different layer types in a transformer block. Those two biases, along with a few implementation details and efficiency measures, like avoiding interference between perturbed modules, are the whole algorithm.

We make the following contributions.

- We present the first zeroth-order method that is competitive with backprop at pretraining transformer language models. At large populations Dust exceeds backprop in multiple settings, which suggests that in a compute-rich regime we might be able to surpass backprop.
- Dust is orders of magnitude more efficient than weight-space ES. From 1M tokens up, Dust is on the order of 10^3 to 10^4 times more efficient than a transformer implementation of EGGROLL, based on our extrapolations.
- Contrary to conventional wisdom, larger models are often more population-efficient, not less, and can make use of larger populations. This gives a new view of overparameterization as a larger search space with potentially better geometry.
- Dust’s gradient estimates align better with backprop’s as the population grows, and the alignment holds up at every scale we test, up to 1B tokens, which is encouraging for scaling.

The goal of this paper is to lay the foundations of a search-based credit assignment algorithm that is competitive with backprop on the *hardest task* we could think of: pretraining transformers. We do not attempt to make it compute-efficient enough to replace backprop today. We also do not train the new kinds of neural nets it makes accessible, like nets with an external program in the loop or transformers looped over many steps that backpropagation through time struggles to train. Both are left to future work.

2 Method

Dust works as follows. We add Gaussian noise to the output of each linear layer, independently at every token, run a forward pass, and reward each token’s noise by the change in loss at that token. The reward-weighted noise, averaged over draws, is the estimated error at the layer’s output, and its outer product with the layer’s input is the weight gradient. Attention internals get a variant of it: they are credited through the estimated error at the attention output over current and future tokens, instead of the tokens’

loss directly. The core intuition is that while weight-space ES evaluates one population member per forward pass, we evaluate one per token, in parallel, and a member is materialized by adding noise to a hidden state, which is cheap. On a modern transformer a single forward pass therefore evaluates a population at least *three orders of magnitude* larger than weight-space ES. We describe each component in detail below.

2.1 Activation-Space Perturbation

The bottleneck of evolution strategies is population size. Every member needs its own perturbed copy of the weights and its own forward pass. EGGROLL (Sarkar et al., 2025) makes the copies cheap with low-rank perturbations, but each member is still one sequence element of the batch, so the population is bounded by the forward passes one can afford. We perturb activations instead, independently at every token. At that token the network behaves as if a low-rank perturbation had been applied to the weights of the layer that produced the activations, without the perturbation ever being materialized in the weights. We call this a *virtual population*. A sequence in a transformer has a few thousand tokens, so one forward pass evaluates a few thousand members per sequence instead of one. Every weight in the model is trained this way except the $2L$ residual mixing scalars, which are trained by ordinary weight-space ES.

Adding noise to activations rather than weights is node perturbation (Widrow and Lehr, 1990), and the usual argument for it is dimensionality (Ren et al., 2023; Werfel et al., 2003). A layer’s output has d_{out} entries and its weights have $d_{\text{out}} \times d_{\text{in}}$, so activation noise lives in a much smaller space. Naively, the dimensionality argument doesn’t hold for transformers with large activations across many tokens. The noise on one sequence is a $T \times d_{\text{out}}$ tensor, which has at least as many entries as the weight matrix once $T \geq d_{\text{in}}$. However, with per-token independent perturbations and rewards, what perturbing activations gives is a new, efficient population along the token axis that is *orthogonal* to the batch axis EGGROLL already relies on.

2.2 Credit Assignment

For a linear layer $y_t = Wx_t$ we jitter its output at all tokens, $y_t \rightarrow y_t + \sigma a_t$ with $a_t \sim \mathcal{N}(0, I)$ and σ the noise scale, and run the forward pass. At each token s we calculate the centered loss reduction $c_s = \tilde{\ell}_s - \ell_s$, where ℓ_s is the perturbed loss and $\tilde{\ell}_s$ is the mean perturbed loss at that token across draws evaluated together in the same batched forward.

The reward of the jitter at token t is the loss reduction at t and, with a decay γ , the loss reductions at the tokens after it, which the jitter also reaches through attention,

$$r_t = \sum_{s \geq t} \gamma^{s-t} c_s. \quad (1)$$

With $\gamma = 0$ a jitter is rewarded by its own token alone. We leave it to the tuning of Section 2.3 to decide which layers see the future tokens. One independent jitter of all tokens is a *draw*, and a population is K draws. Averaged over the draws, the reward-weighted noise

$$\hat{g}_t = -\frac{1}{K\sigma} \sum_{i=1}^K r_t^{(i)} a_t^{(i)} \quad (2)$$

is the estimated error at the layer's output, and its outer product with the layer's input, which the forward pass already computed, summed over tokens, is the weight gradient,

$$\hat{G}_W = \sum_t \hat{g}_t x_t^\top. \quad (3)$$

Backprop forms the same outer product with the same input. The only difference is that it gets the output error from the chain rule and we get it from the population. For an embedding layer x_t is one-hot, so the outer product is a scatter-add of $\hat{g}_t$ into the token's row.

With unlimited population the estimator above is the whole method, and every perturbation could go into one forward pass. Each draw's reward-weighted noise is the gradient plus an error with no preferred direction. Over draws the gradient adds up linearly while the errors add up as the square root, so their ratio falls as the population grows and the interference vanishes in the limit.

2.3 Interference and Tuning

At a population we can afford, the main cost is interference, i.e. we jitter many layers at many tokens in the same forward pass, so the loss change that rewards one token's noise also picks up the effect of every other perturbation in that pass. We reduce such interference in three ways. First, different layer types are jittered in separate forward passes, each with its

own noise scale, and each block gets its own passes. These passes are cheaper than full forwards because the clean forward is cached and a draw for block l only reruns blocks l onward. Second, the attention internals (query, key, value, gate, value embedding) are jittered separately. Token losses barely register their jitters, so they are rewarded through the attention output instead, as described below. Third, the language modeling head is jittered directly on the cached logits, re-evaluating only the cross-entropy and only a slab of the vocabulary per draw, which costs a small fraction of a forward pass and lets the head run a much larger population.

For the attention internals we recompute *only* their block’s attention output with the jitter, from the cached clean activations. We score the jitters using the alignment with the estimated attention output gradients,

$$c_s = -\langle \hat{g}_s, \Delta o_s \rangle, \quad (4)$$

where Δo_s is the change the jitter makes in the attention output at token s and $\hat{g}_s$ is that output’s estimated error from Equation 2. The reward is Equation 1 with these scores, computed per head, in place of the loss reductions.

The hyperparameters, the noise scale of each layer type, the credit decay of the attention internals and the share of the population each layer gets, can be tuned in two ways. One is a grid search that trains with each setting on a small token budget and keeps the setting that lowers the loss most. It is reliable but expensive. The other is a grid search that maximizes the cosine between our estimate and the backprop gradient on a single batch, which needs no training at all. A larger cosine on one batch doesn’t always lower the loss after training, so the cosine picks candidates and training decides. Either way the tuning is mostly a one-time cost, since the settings it finds largely generalize across token budgets and population sizes, with one exception. At the largest population at 10M and 20M tokens, a slower credit decay and a shift of draws toward the attention side still pay (Appendix F). Hence, this search recovers general principles of the method rather than settings for one run. As expected, all layers require no credits from future tokens except the keys, values, gates and value embeddings, which are read by the later tokens that attend to them and get a γ close to one.

3 Pretraining Without a Backward Pass

3.1 Setup

We train GPT-style transformers on FineWeb with a 4096-token BPE tokenizer, a batch of 16k tokens (8 sequences of 2048 tokens), one epoch, and SGD with momentum at a constant learning rate. The base model has 8 layers and width 512. Every method gets the same protocol and three seeds per cell, and is tuned separately at every token budget and population. Dust and backprop share one momentum and learning-rate grid; we implement EGGROLL with the same transformer architecture (named EGGROLL-Transformer) and it is tuned over its own grid of step size, momentum, noise scale and fitness shaping. Validation and test are held-out sets of 544 sequences each. We report the test loss at best validation checkpoint.

We count population in draws for Dust and in forward passes of the batch for EGGROLL (Sarkar et al., 2025), our weight-space baseline. A draw is one jitter of activations across all tokens of selected layers, rewarded with the token losses, and the population K is the number of draws per update. A draw is slightly cheaper than a forward pass, because the clean forward is cached and a draw that jitters block l reruns only the blocks from l on. K leaves out the draws of the head and the attention internals, which are a small fraction of the update’s FLOPs (Appendix E). Taken together, from a population of 256 up Dust uses less compute than EGGROLL at the same population, so the comparison is lenient to EGGROLL.

3.2 Main Results

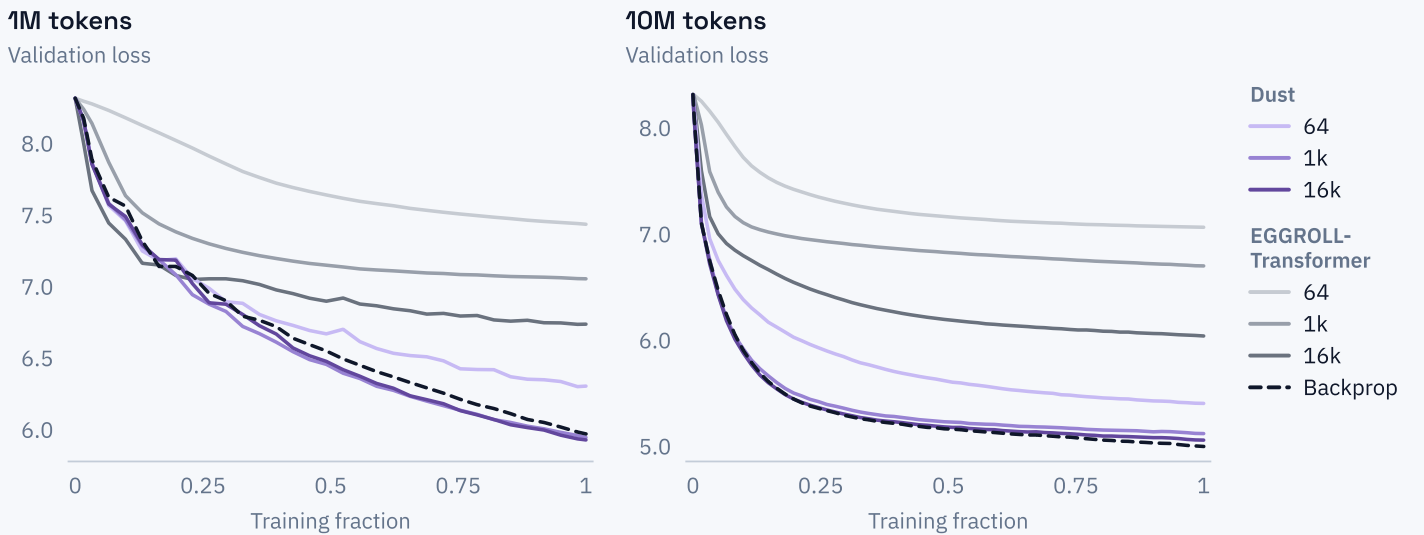


Figure 1: Validation loss over training at 1M and 10M tokens for Dust (violet), EGGROLL at the same populations (gray) and backprop (black, dashed). The x-axis is the fraction of the token budget consumed. Shade is population, 64, 1k and 16k, darker for larger. Curves are means over three seeds.

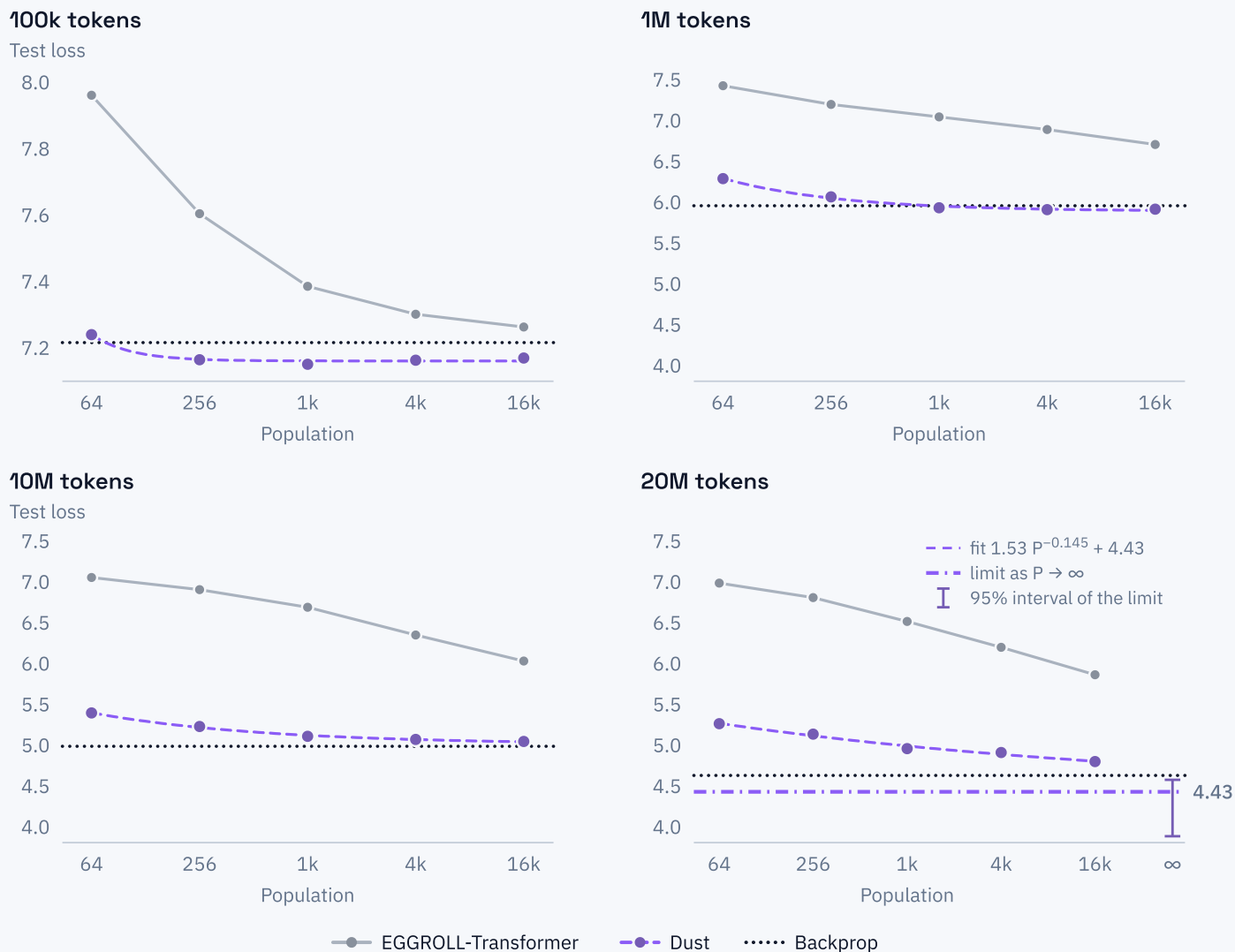


Figure 2: Test loss at the validation-selected checkpoint against population, one panel per token budget, for Dust (violet) and EGGROLL-Transformer (gray). Points are the cells of Table 1, the dotted black line is backprop tuned at the same budget, and the dashed curve is a power law fit through Dust’s five cells. The 100k panel has its own y range; the other three share one. At 20M the dash-dotted violet line is the fit’s limit, 4.43, an extrapolation from a ladder that is still falling, and the bar at ∞ is its 95% bootstrap interval, with the seeds resampled at each population.

We sweep the token budget from 100k to 20M against populations from 64 to 16k, with backprop tuned on the same grid at every budget (Table 1, Figure 2). At 100k and 1M tokens Dust ends below backprop, from a few hundred draws at 100k and from a thousand at 1M. At 10M and 20M tokens, the gap with backprop shrinks with population. At 10M the ladder has flattened and its fitted limit lands just above backprop. At 20M the ladder is still falling at 16k draws. Its power law fit puts the limit at 4.431 (95% interval 3.89 to 4.58), below backprop’s 4.633, but with the ladder still falling the fit is loosely constrained, so we read it as evidence that the gap keeps closing with population rather than as a measured limit.

Weight-space ES is *far less efficient*. With 256 times the population, EGGROLL at 16k still does not reach Dust at 64 draws. It comes within 0.02 at 100k tokens and stays 0.4 to 0.6 above at 1M, 10M and 20M (Figure 2). EGGROLL’s ladder is still falling steeply at 16k, so it would keep improving with more population, but continuing the ladder puts what it needs to match Dust’s smallest population at several thousand to about 10^4 times that population (Appendix D). Figure 1 also shows the gap during training; Dust at 1k and 16k draws stays close to backprop’s curve throughout, while every EGGROLL curve falls behind early and flattens well above Dust at 64 draws.

3.3 Dust Under Adam

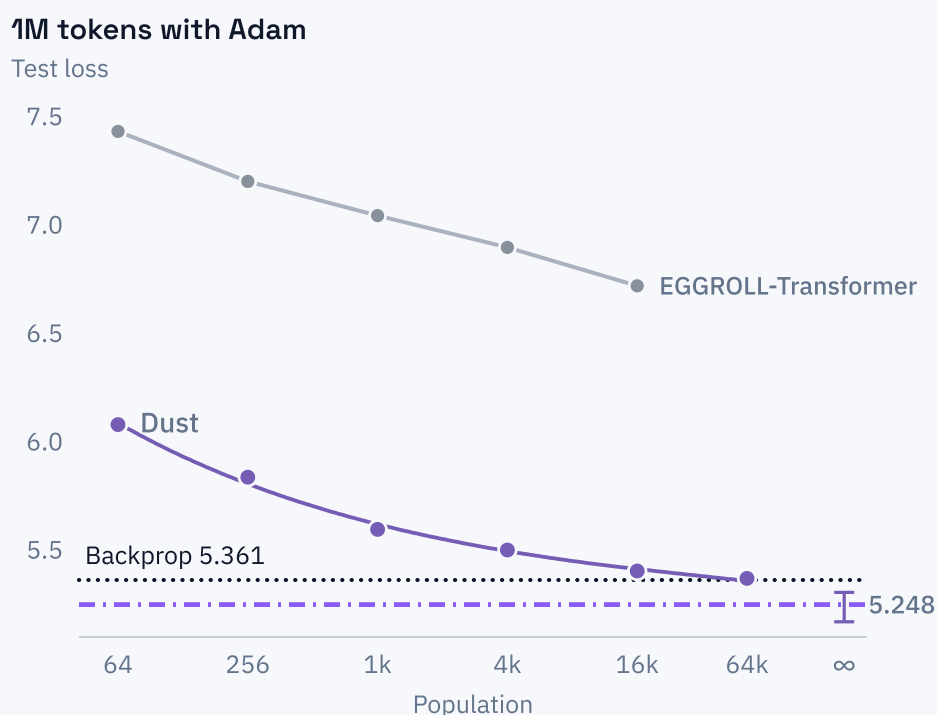


Figure 3: Test loss against population at 1M tokens with Adam, for Dust (violet), EGGROLL (gray) and backprop (black, dotted). The violet curve is a power law fit in population over the populations run, and the dash-dotted violet line is its limit, 5.248, below backprop. The bar at ∞ is the limit’s 95% bootstrap interval, 5.17 to 5.30, with the seeds resampled at each population. Means over five seeds for Dust and backprop and three for EGGROLL; the Adam runs of Dust give the token embedding more draws, about $1.4\times$ the listed population in total.

While we focus primarily on SGD for the rest of the paper, we repeat the 1M token ladder with Adam on all three methods (Figure 3), re-tuning backprop’s learning rates, Dust’s own hyperparameters and EGGROLL’s step size, momentum and fitness shaping at every population. Interestingly, EGGROLL gains almost nothing from Adam and its tuned Adam ladder lands within 0.01 of its SGD ladder in Table 1 at every population. However, Adam

improves both Dust and backprop and leaves the ladder’s shape similar to before. Dust closes on backprop with a large population and the interval on its limit sits below backprop (Figure 3). So Dust’s estimate *already works with modern optimizers*, even though modern optimizers were optimized for backprop gradients. We suspect that coevolution of optimizers with Dust could lead to further gains and leave this to future work.

4 Search in High-Dimensional Space

4.1 Overparameterization

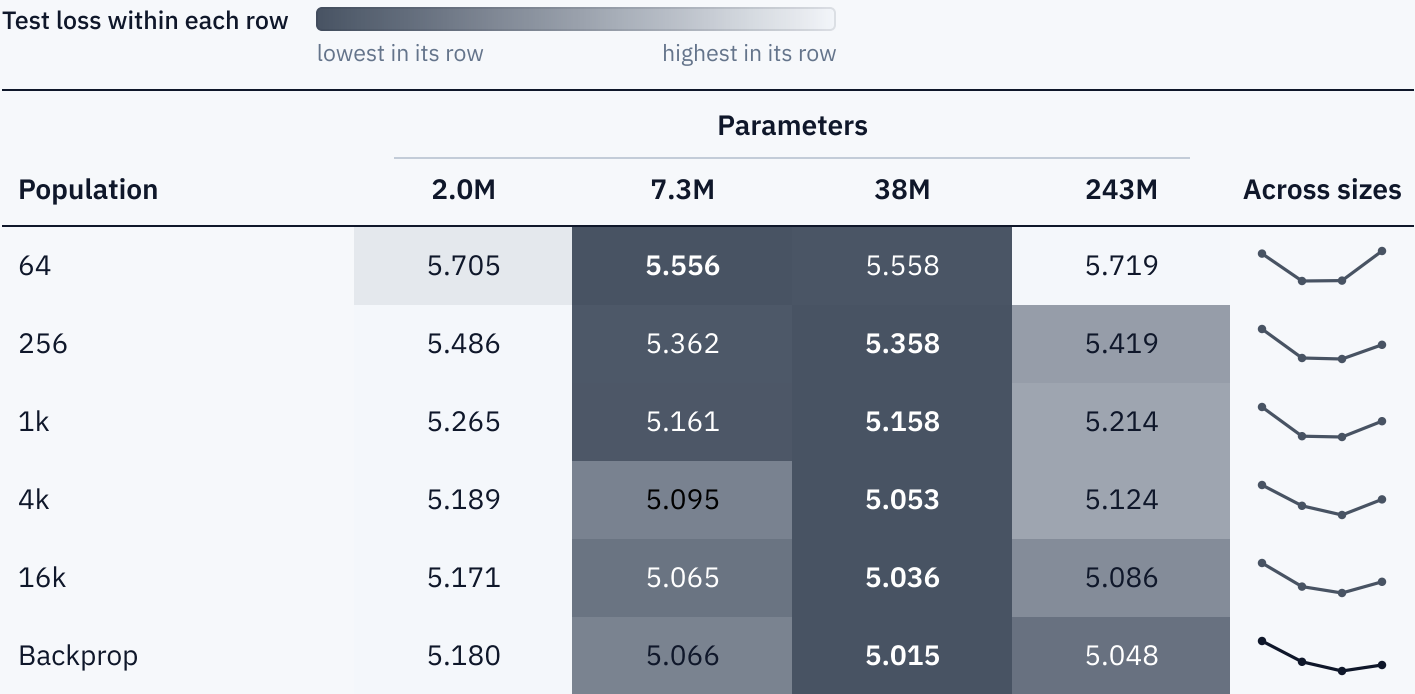


Figure 4: Four model sizes at a fixed 10M tokens, 2M to 243M parameters (L2/d128, L4/d256, L8/d512, L16/d1024), test loss against population. The table has the same cells and backprop per size, with the best size at each population in bold.

The conventional view is that zeroth-order methods cannot train large networks (Lillicrap et al., 2020). A forward pass returns a single scalar, so the variance of the gradient estimate grows with the number of perturbed dimensions, and with it the population needed for a useful update (Nesterov and Spokoiny, 2017; Werfel et al., 2003). We test this directly by training four sizes, 2M, 7M, 38M and 243M parameters (a $120\times$ range in parameter count), at a fixed 10M tokens across the population sweep (Figure 4; see Appendix F for tuning details). We make two striking observations that challenge conventional wisdom.

- *Bigger models are often more population-efficient, not less.* At every population from 256 up the loss falls from 2M to 7M to 38M parameters, and the 243M model is only slightly worse. Even at the smallest population we test, a model $120\times$ larger does about the same, and is better at every other population. Up to a point, a larger model gains more from its size than it loses to variance. What we observe, however, is that the gap to backprop at the same population and model size increases with model size, but barely noticeable, esp at large population sizes.
- *Bigger models keep improving with population where small ones flatten.* Small models saturate sooner, while big models keep improving with large population sizes. Past 1k draws the 38M and 243M models gain about 30% more than the 2M and 7M ones, and no amount of population makes the tiny model competitive.

The right way to think about model size is therefore as the *size and the geometry of the search space*. A larger model has a larger space to search over, which is what lets it put a large population to use, and potentially a better-conditioned loss landscape geometry, which could be why the search is more effective even at small populations.

4.2 Emergence of Backprop-Like Gradients

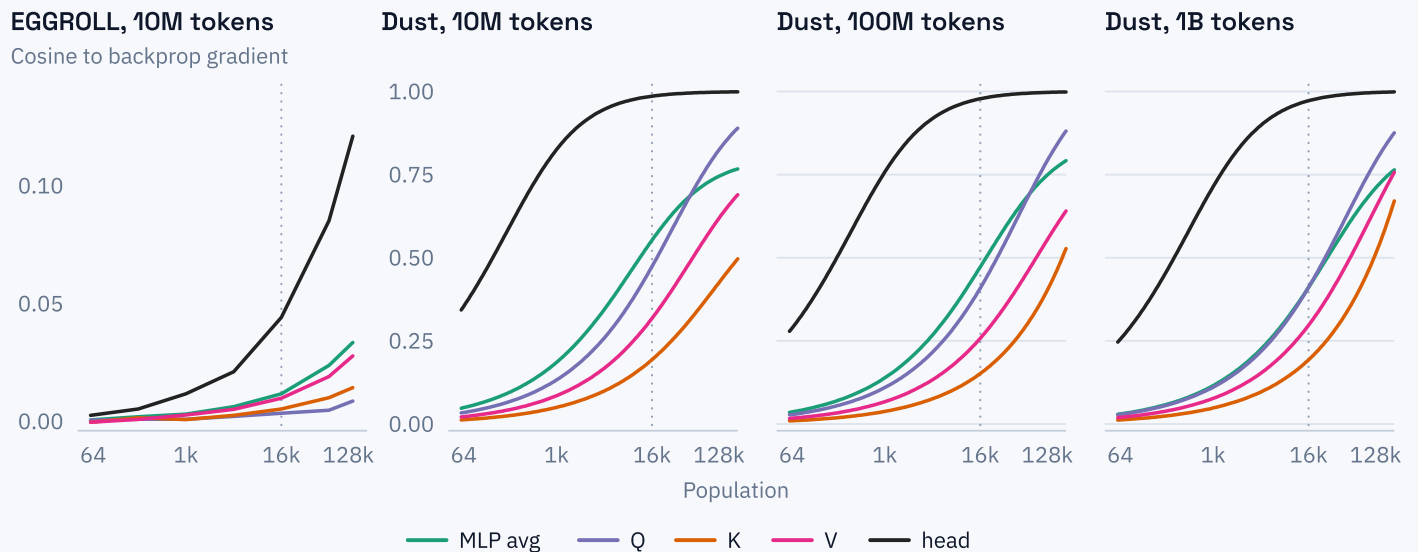


Figure 5: Cosine between the estimated and the backprop gradient on the same batch, per layer type (mean over layers; MLP averages the input and output projections), against population, for Dust on backprop-trained checkpoints at 10M, 100M and 1B tokens and for EGGROLL at 10M tokens at the same compute budget (left, note the axis). For Dust the lines are the fitted law $\cos(K) = c_{\max} / \sqrt{1 + c/K}$ through measurements at populations of 64 to 128k; for EGGROLL they connect the measurements. The dotted vertical line is the 16k population the ladders train at. Initialization and EGGROLL at 100M tokens are in Appendix B.

We measure the cosine between Dust’s estimate and the backprop gradient on the same batch, per layer type and per layer, on backprop-trained checkpoints spanning two orders of magnitude in tokens, i.e. from 10M to 1B tokens, with the population growing from 64 to 128k forward passes per step (Figure 5). The cosine rises with population for every layer type at every stage of training, and a two-parameter law

$$\cos(K) = \frac{c_{\max}}{\sqrt{1 + c/K}} \quad (5)$$

fits each layer type with RMSE below 0.06, with $c_{\max}$ the ceiling and c the population at which the layer reaches $c_{\max}/\sqrt{2}$. Useful gradients *emerge* from the large population alone, with nothing about the chain rule built in, just light tuning of the hyperparameters to maximize the cosine similarity as described in Section 2.3. For the 100M token checkpoint, the fits to layer type mean cosine similarities have RMSEs of 0.0032–0.0367 (see Appendix G for details). Dust’s gradients approach backprop’s in cosine, but how close they get varies by layer type and by layer. EGGROLL’s estimate, measured the same way at the same number of forward passes, is growing with population but is still below 0.05 in cosine at 128k for every type but the head, which might explain why training on it went nowhere in Section 3.2.

Importantly, the cosines hold up across most layers as the token count grows, which is encouraging for scaling. Section 3 shows the population required to match and exceed backprop grows with tokens. However, the cosine stays flat across two orders of magnitude in tokens at large populations, which means that at a large enough population it’s possible that this requirement *no longer grows with more tokens*. Furthermore, the fact that Dust’s gradients approach backprop’s but don’t converge to them exactly is in fact a good property. The estimate points in a similar direction without being backprop’s gradient, which leads to a different optimization trajectory, and in our experiments that trajectory can even be better than backprop’s (Section 3.2).

5 Conclusion

Since Rumelhart et al. (1986), backprop has been the algorithm that trains neural nets, and the architectures, optimizers and hardware we have were all built around it. As compute becomes more abundant, we think much better alternatives are possible. We introduced

Dust, an algorithm that drastically improves upon existing ES algorithms and approximates backprop closely at pretraining transformers, even exceeding it with large amounts of computation.

There are many interesting open questions. The first is whether, and how, Dust can find better directions than backprop’s first-order gradient by *implicitly exploring the loss landscape*, picking up higher-order curvature that pulls the search toward flat regions. We have hints that it can, since at large populations it sometimes ends up below backprop, but the mechanism is not clear. The second is that Dust opens up the search space over architectures, since it does not need the network to be end-to-end differentiable, and it may do better where backprop is known to struggle, like recurrent or looped computation trained by backpropagation through time. The third is compute efficiency, which was not the focus of this paper. We must have orders of magnitude more compute efficiency before Dust becomes a practical alternative to backprop at current levels of compute.

6 Related Work

Early work on zeroth-order pretraining of language models (Allaire et al., 2025) examined the difficulty of training transformers from scratch with weight perturbations. Their later method, KronZO (Allaire et al., 2026), uses compact perturbations with Kronecker structure and selective directional updates to improve pretraining while reducing memory use. EGGROLL (Sarkar et al., 2025) makes large populations of weight perturbations efficient on GPUs through low-rank structure. Dust instead searches over activations and uses rewards for each token to extract more credit from each forward pass.

For fine tuning, MeZO (Malladi et al., 2023) showed that language models can be adapted with forward passes alone and memory use close to inference. Evolution Strategies at Scale (Qiu et al., 2026) demonstrates fine tuning of all parameters with ES in language models with billions of parameters. Neural Thickets (Gan and Isola, 2026) finds useful task experts by randomly perturbing pretrained weights, selecting the best candidates and ensembling their predictions. These results show how much search can achieve around a pretrained model. Our experiments address learning the representations themselves through pretraining from scratch.

Our activation perturbations build on node perturbation (Werfel et al., 2003). GEMINI (Le Cun et al., 1988) injected noise into the first hidden layer and recovered layerwise gradient

estimates through iterative matrix inversion. Zoop (Hu et al., 2025) uses output perturbations for language model fine tuning, converting estimated output gradients into parameter updates with local derivatives. Scaling Forward Gradient with Local Losses (Ren et al., 2023) combines activation perturbations and local losses with forward mode automatic differentiation to reduce estimator variance. Forward gradients with multiple tangents (Flügel et al., 2025) also use forward mode differentiation, combining multiple directional derivatives through orthogonal projection to improve gradient estimates.

A separate line of work replaces the global backward pass with local learning dynamics: Sakana AI’s PC-ALM (Seely and Gould, 2026) propagates credit through local predictive coding dynamics and Lagrange multipliers. It uses local derivatives and is evaluated on image classification tasks. Dust estimates credit from forward perturbations during transformer pretraining, combining rewards for each token with local targets for attention outputs.

REFERENCES

- Nathan Allaire, Mahsa Ghazvini Nejad, Sébastien Le Digabel, and Vahid Partovi Nia. Zeroth order optimization for pretraining language models. In *Proceedings of ICPRAM*, pages 113–121, 2025. doi: [10.5220/0013261100003905](https://doi.org/10.5220/0013261100003905). URL <https://doi.org/10.5220/0013261100003905>.
- Nathan Allaire, Sébastien Le Digabel, Dominique Orban, and Vahid Partovi Nia. Zeroth-order Kronecker optimization for pretraining language models. *SN Computer Science*, 7, 2026. URL <https://www.gerad.ca/en/papers/G-2025-44>. Article 162.
- Katharina Flügel, Daniel Coquelin, Marie Weiel, Charlotte Debus, Achim Streit, and Markus Götz. Beyond backpropagation: Optimization with multi-tangent forward gradients. In *International Joint Conference on Neural Networks*, 2025. URL <https://arxiv.org/pdf/2410.17764v2>.
- Yulu Gan and Phillip Isola. Neural thickets: Diverse task experts are dense around pretrained weights. *arXiv preprint arXiv:2603.12228*, 2026. URL <https://arxiv.org/abs/2603.12228>.
- Wes Gurnee, Nicholas Sofroniew, Adam Pearce, Mateusz Piotrowski, Isaac Kauvar, Runjin Chen, Anna Soligo, Paul Bogdan, Euan Ong, Rowan Wang, Ben Thompson, David Abrahams, Subhash Kantamneni, Emmanuel Ameisen, Joshua Batson, and Jack Lindsey. Verbalizable representations form a global workspace in language models. *arXiv preprint arXiv:2607.15495*, 2026.
- Xixi Hu, Bo Liu, Qiang Liu, Xiaocong Du, Bhargav Bhushanam, Louis Feng, Chengyue Gong, and Kaizhao Liang. Zoop it! Efficient zero-order optimization with output perturbation. In *ICML Workshop on Tiny Titans: The next wave of On-Device Learning for Foundation Models*, 2025. URL <https://openreview.net/forum?id=Tc8vFyRhPO>.
- Yann Le Cun, Conrad C. Galland, and Geoffrey E. Hinton. GEMINI: Gradient estimation through matrix inversion after noise injection. In *Advances in Neural Information Processing Systems*, volume 1, pages 141–148,

1988. URL https://papers.neurips.cc/paper_files/paper/1988/file/a0a080f42e6f13b3a2df133f073095dd-Paper.pdf.
- Timothy P. Lillicrap, Adam Santoro, Luke Marris, Colin J. Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21 (6): 335–346, 2020. doi: [10.1038/s41583-020-0277-3](https://doi.org/10.1038/s41583-020-0277-3).
- Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, et al. On the biology of a large language model. *Transformer Circuits Thread*, 2025.
- Shengchao Liu, Dimitris Papailiopoulos, and Dimitris Achlioptas. Bad global minima exist and SGD can reach them. In *Advances in Neural Information Processing Systems*, volume 33, 2020.
- Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. In *Advances in Neural Information Processing Systems*, volume 36, 2023.
- Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17 (2): 527–566, 2017. doi: [10.1007/s10208-015-9296-2](https://doi.org/10.1007/s10208-015-9296-2).
- Xin Qiu, Yulu Gan, Conor F. Hayes, Qiyao Liang, Yinggan Xu, Roberto Dailey, Elliot Meyerson, Babak Hodjat, and Risto Miikkulainen. Evolution strategies at scale: LLM fine-tuning beyond reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2026. URL <https://arxiv.org/abs/2509.24372>. arXiv:2509.24372.
- Mengye Ren, Simon Kornblith, Renjie Liao, and Geoffrey Hinton. Scaling forward gradient with local losses. In *International Conference on Learning Representations*, 2023.
- David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323: 533–536, 1986.
- Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.
- Bidipta Sarkar, Mattie Fellows, Juan Agustin Duque, Alistair Letcher, Antonio León Villares, Anya Sims, Clarisse Wibault, Dmitry Samsonov, Dylan Cope, Jarek Liesen, Kang Li, Lukas Seier, Theo Wolf, Uljad Berdica, Valentin Mohl, Alexander David Goldie, Aaron Courville, Karin Sevegnani, Shimon Whiteson, and Jakob Nicolaus Foerster. Evolution strategies at the hyperscale. *arXiv preprint arXiv:2511.16652*, 2025.
- Jeffrey Seely and Julian Gould. Augmented Lagrangian predictive coding. *arXiv preprint arXiv:2605.31022*, 2026. URL <https://arxiv.org/abs/2605.31022>.
- David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550 (7676): 354–359, 2017. doi: [10.1038/nature24270](https://doi.org/10.1038/nature24270).
- Richard S. Sutton. The bitter lesson. <http://www.incompleteideas.net/IncIdeas/BitterLesson.html>, 2019. Blog post.
- Akshay Vegesna and Samip Dahal. Decoupling search and learning in neural net training. *arXiv preprint arXiv:2509.10973*, 2025.
- Justin Werfel, Xiaohui Xie, and H. Sebastian Seung. Learning curves for stochastic gradient descent in linear feedforward networks. In *Advances in Neural Information Processing Systems*, volume 16, 2003.

Bernard Widrow and Michael A. Lehr. 30 years of adaptive neural networks: Perceptron, Madaline, and backpropagation. *Proceedings of the IEEE*, 78 (9): 1415–1442, 1990. doi: [10.1109/5.58323](https://doi.org/10.1109/5.58323).

APPENDIX

- A Architecture
- B Cosine Ladder at the Remaining Checkpoints
- C Test Losses by Token Budget and Population
- D EGGROLL’s Extrapolated Population
- E Population Allocation
- F Tuning
- G Cosine-fit parameters

[← Back to Q](#)

Appendix

CONTENTS

- A Architecture
- B Cosine Ladder at the Remaining Checkpoints
- C Test Losses by Token Budget and Population
- D EGGROLL's Extrapolated Population
- E Population Allocation
- F Tuning
- G Cosine-fit parameters
- [← Back to post](#)

A Architecture

The model is a decoder-only transformer of the modded-nanoGPT kind: eight pre-norm blocks of width 512 with eight heads, RMS normalization, rotary position embeddings with normalized queries and keys, a squared-ReLU MLP of width 2048, and no biases. Attention is causal with a sliding window of 1024 tokens in three of every four blocks and the full 2048 in the others. Every other block adds a *value embedding* to its values, a second token embedding scaled per head by a learned gate on the block input. The residual stream entering each block is a learned mix of the previous block's output and the token embedding, and these $2L$ mixing scalars are the parameters trained by weight-space ES. The head is untied, with capped logits. Token and value embeddings are stored in bfloat16 and the forward runs in bfloat16. The base model has 37.7M parameters; the model-size study uses two, four and sixteen blocks of width 128, 256 and 1024, with the same head dimension.

B Cosine Ladder at the Remaining Checkpoints

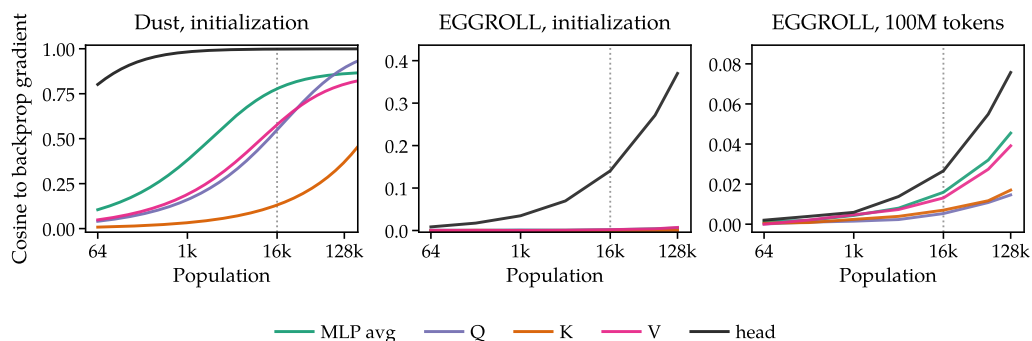


Figure 6: The cosine ladder of Figure 5 at the checkpoints not shown there. Left, Dust at initialization. Middle and right, EGGROLL at initialization and at 100M tokens, each on its own axis.

C Test Losses by Token Budget and Population

Table 1 has the test losses behind Figure 2.

Table 1: Test loss at the validation selected checkpoint averaged over three seeds, across token budgets and populations. Population is draws rewarded with token losses for Dust and forward passes of the batch for EGGROLL (Section 3.1). Backprop is tuned on the same grid at every budget. The ∞ row is the limit of a power law fit in population.

	Population	100k	1M	10M	20M
Dust	64	7.240	6.290	5.397	5.265
	256	7.165	6.067	5.231	5.139
	1k	7.151	5.934	5.111	4.960
	4k	7.164	5.911	5.072	4.912
	16k	7.170	5.916	5.049	4.802
	∞ (fit)	7.161	5.892	5.013	4.431
EGGROLL	64	7.961	7.428	7.056	6.987
	256	7.605	7.200	6.906	6.811
	1k	7.386	7.046	6.692	6.517
	4k	7.302	6.892	6.352	6.200
	16k	7.263	6.708	6.033	5.865
Backprop		7.216	5.959	4.989	4.633

D EGGROLL’s Extrapolated Population

We extrapolate EGGROLL’s ladder at 1M tokens and more to the population at which it would match Dust at 64 draws (Section 3.2). Table 2 gives that population as a multiple of 64 under three fits of EGGROLL’s cells in Table 1. The power law is the fit of Figure 2, $L(P) = AP^{-\alpha} + E$, which we also refit with each population left out in turn. The straight lines in log population are least-squares fits to the top three populations (1k, 4k and 16k) and the top two (4k and 16k). The three fits put the multiple between about 3,000 and 23,000.

Table 2: The population EGGROLL would need to match the test loss of Dust at 64 draws, as a multiple of 64, from fits of its five cells in Table 1. ∞ marks a refit whose floor lies above the loss of Dust.

Fit	1M	10M	20M
Power law, all five populations	23,000×	16,000×	8,100×
Power law, one population left out	11,000×	7,200×	4,700×
Straight line in log population, top three	8,200×	3,700×	3,300×
Straight line in log population, top two	6,000×	4,100×	3,100×

E Population Allocation

K counts the draws rewarded with token losses. Table 3 lists everything an update jitters, with its draws at $K = 16,384$ and its noise scale, and Table 4 splits the three per-block streams over the blocks. Populations from 256 up scale every count by $K/16,384$. At 64 draws the recipe at 1M and 10M is (8, 7, 4, 3, 1, 1, 1, 1) writer draws, (8, 7, 4, 3, 2, 1, 1, 1) MLP draws, one attention-output draw per block, three embedding draws and 1, 5, 2, 3 and 2 draws for q , v , the gate, k and the value embedding. The nominal stream ratio before rounding is (writers + MLP) : attention outputs : embedding = 7 : 0.75 : 0.25 (at 100k tokens 6 : 1.5 : 0.5), with a tilt toward the first blocks that comes from a coordinate descent over the allocation at 2k draws.

Table 3: What one update jitters, at $K = 16,384$. The first four rows are rewarded with token losses and add up to K under SGD and to $11K/8$ in the Adam ladder. The SGD column is the allocation at 1M tokens, and Table 4 has the other budgets. The attention internals are credited through the attention output, and gates and value embeddings exist only on every other block. Each head draw jitters one slab of 256 logits. The attention outputs of the deeper half use 0.4 because their loss changes are weak and poorly resolved at 0.2. An update also runs one clean forward per GPU.

Site	One draw reruns	Draws		
		SGD	Adam	Noise scale
Writer pair	its block onward	7,040	4,096	0.2
MLP hidden layer	its block onward	7,040	4,096	0.2
Attention output	its block onward	1,664	6,144	0.2, deeper half 0.4
Token embedding	the whole network	640	8,192	0.2, Adam 0.4
Query, per block	its block’s attention	128	128	0.05
Value, per block	its block’s attention	1,280	1,280	0.05
Gate, per block	its block’s attention	384	384	0.05
Key, per block	its block’s attention	768	768	0.05
Value embedding, per block	its block’s attention	512	512	0.05
Head	one slab’s cross-entropy	65,536	49,152	0.05
Residual scalars	the whole network, twice	8	8	0.03

Table 4: Draws per block at $K = 16,384$ for the writer pair (the MLP hidden layer gets the same, except block 2 at 100k, which gets 1,664) and for the attention output. The 16k cells at 10M and 20M have allocations of their own; every other cell scales its budget’s column by $K/16,384$. The token embedding gets 640 draws, and 1,152 at 100k. The 10M and 20M allocations at 16k take the same time per update as the one they replace.

Block	Writer pair, MLP hidden layer					Attention output		
	100k	1M, 10M	20M	10M at 16k	20M at 16k	100k	all others	10M at 16k
1	2048	2432	2816	2176	2304	896	384	768
2	1792	2048	2432	1792	2048	768	384	640
3	896	1024	896	896	768	512	256	512
4	640	640	256	512	256	384	128	256
5	256	384	256	384	256	128	128	256
6	256	256	128	256	128	128	128	256
7	128	128	128	128	128	128	128	256
8	128	128	128	128	128	128	128	256
Total	6144	7040	7040	6272	6016	3072	1664	3200

Draws are evaluated n at a time on one GPU, with $n = 1, 2, 8, 8, 16$ at $K = 64, 256, 1,024, 4,096, 16,384$, and split evenly across GPUs. Direct loss estimates and queries use $\gamma = 0$. Keys, values, gates and value embeddings use $\gamma = 0.99$ in the main

comparison’s 10M/16k and 20M/16k runs, and $\gamma = 0.98$ in all other runs. At 20M and 16k draws, k and v get 1,536 and 2,560 draws per block, so that cell has 14,336 draws rewarded with losses at the compute of the other 16k cells. For the other model sizes the tilt is sampled at the same relative depth and the attention-internal counts are scaled so that their totals match the 8-layer model.

F Tuning

Every cell of the main comparison, a token budget and a population for Dust and a token budget for backprop, is tuned the same way. Both methods tune momentum and learning rate only, starting from the grid of Table 5. Every other constant is fixed (Section 3.1, Appendix E), the learning rates of Table 6 included. Seed 42 runs the grid; the best point and every point within 0.02 of it get seeds 43 and 44, as does any point whose single seed beats the current three-seed best. A best point on the edge of the grid has the grid extended one step along its line until the best point is interior, and the two rate half-steps around it are then run under the same rules. A run whose final validation loss exceeds its best by more than 0.25, or that ever exceeds the uniform-prediction loss $\ln V$, or produces a NaN, is rejected together with its configuration. The selected configuration is the lowest mean validation loss over the three seeds, and we report its test loss at each seed’s best-validation checkpoint.

Table 5: The 12-point grid each cell starts from. The 64-draw cells at 10M and 20M need heavy momentum and small rates.

Cells	Momentum	Learning rate
100k tokens	0.2, 0.3, 0.5	0.3, 0.4, 0.5, 0.6
1M, 10M and 20M tokens	0.9, 0.95, 0.98	0.1, 0.15, 0.2, 0.25
10M tokens at 64 draws	0.95, 0.98, 0.99	0.0125, 0.025, 0.05, 0.1
20M tokens at 64 draws	0.95, 0.98, 0.99	0.00625, 0.0125, 0.025, 0.05

Table 6: Learning rates under SGD. Base is the tuned rate of the cell. The model-size study of Section 4.1 predates the protocol of Section 3.1, and its learning rate and momentum are selected per size and population.

Parameters	Main comparison		Model-size study	
	Dust	Backprop	Dust	Backprop
Key projections	base	base	$4\times$ base	$4\times$ base
Value projections	base	base	$8\times$ base	$8\times$ base
Token embedding	1000	1000	0.3	base
Value embeddings	0.3	base	0.3	base
Residual scalars	0.03	base	0.03	base
Everything else	base	base	base	base

The overparameterization experiments were run before we improved the token-embedding learning rate. They use the earlier training settings in Table 6, which explains the difference from the main results.

Dust’s own constants, the noise scales, the credit decay and the draw allocation, are chosen with the cosine diagnostic of Section 2.3, confirmed in training, and held fixed across cells. At the 16k cells of 10M and 20M tokens they are tuned in training together with momentum and rate at fixed compute: each lever, the allocation across layer families, the per-block tilt, the noise scales, the credit decay and the head draws, is run one at a time from the (momentum, rate) winner, seeded in stages on wins paired by seed, and the winning levers are stacked; backprop receives the same wider (momentum, rate) grid at three seeds. The selected constants are those of Appendix E. Not every lever ran at 20M within the campaign, and the other cells keep the constants from the cosine tuning, so at those two budgets the top cell of the ladder has constants of its own, which the 10M fit is read with.

Under Adam, three learning-rate groups, matrices and the head, token and value embeddings, and residual mixing scalars, are tuned for Dust and backprop. Table 7 has the selected settings at 1M tokens.

Table 7: Selected settings under Adam at 1M tokens. Dust’s matrix rate is selected per population. All three methods use $\beta_2 = 0.95$, $\epsilon = 10^{-8}$ and no weight decay, and EGGROLL uses z-scored differences.

	Dust	Backprop	EGGROLL
Matrices and head, at 64 draws	0.00175	0.002	0.003
Matrices and head, from 256 to 4k	0.0015	0.002	0.003
Matrices and head, from 16k up	0.002	0.002	0.003

	Dust	Backprop	EGGROLL
Token and value embeddings	$300\times$ matrices	0.3	0.003
Residual scalars	$100\times$ matrices	0.2	0.003
β_1	0.7	0.8	0.5

The model-size study of Section 4.1 changes two hyperparameters from Section 3 (Table 6). The token embedding stays at its default learning rate, 0.3 for Dust and the base rate for backprop, because at 1000 backprop shows no clear separation between model sizes, which makes overparameterization hard to study. We also keep the key and value learning rates of the earlier protocol and tune them for both Dust and backprop, so the comparison is fair to both.

G Cosine-fit parameters

Table 8 reports fits of $C(K) = c_{\max}/\sqrt{1 + c/K}$ to measured cosine similarities at the final 100M token checkpoint. Each fit uses seven population budgets from 64 to 131072. At each budget, cosine similarity is computed separately for each weight matrix and then averaged across matrices of the same layer type. The fitted parameter c uses the same population-budget units as K , rather than the individual draw count of a particular layer. RMSE is computed across the seven measured points used for fitting.

Table 8: Fitted parameters and errors for cosine similarity as a function of population size. Fits describe layer-type means.

Layer type	$c_{\max}$	c	RMSE
<i>100M-token checkpoint</i>			
MLP out	0.8779	31623	0.0081
MLP in	0.8577	48417	0.0065
Attn out	0.8729	21627	0.0093
Attn Q	1.0000	76736	0.0356
Attn K	1.0000	616595	0.0191
Attn V	0.8776	173780	0.0269
Value gate	0.6657	47863	0.0367
Value emb	1.0000	495450	0.0247

Layer type	$c_{\max}$	c	RMSE
Token emb	0.7318	19498	0.0083
LM head	1.0000	759	0.0032

[← Back to the post](#)