Evolution Strategies at the Hyperscale

Bidipta Sarkar[*1,2], **Mattie Fellows**[*1], **Juan Agustin Duque**[*2,3],
Alistair Letcher[†1], **Antonio León Villares**[†1], **Anya Sims**[†1], **Clarisse Wibault**[†1],
Dmitry Samsonov[†6], **Dylan Cope**[†1], **Jarek Liesen**[†1], **Kang Li**[†1], **Lukas Seier**[†1], **Theo Wolf**[†1],
Uljad Berdica[†1], **Valentin Mohl**[†1],
Alexander David Goldie[1,2], **Aaron Courville**[3,5], **Karin Sevegnani**[4],
Shimon Whiteson[‡2], **Jakob Nicolaus Foerster**[‡1].
[1] FLAIR - University of Oxford, [2] WhiRL - University of Oxford, [3] MILA– Québec AI Institute
[4] NVIDIA AI Technology Center, [5] CIFAR AI Chair, [6] NormaCore.dev
{bidipta.sarkar,matthew.fellows,jakob.foerster}@eng.ox.ac.uk
juan.duque@mila.quebec, shimon.whiteson@cs.ox.ac.uk

Abstract

Evolution Strategies (ES) is a class of powerful black-box optimisation methods that are highly parallelisable and can handle non-differentiable and noisy objectives. However, naïve ES becomes prohibitively expensive at scale on GPUs due to the low arithmetic intensity of batched matrix multiplications with unstructured random perturbations. We introduce Evolution Guided GeneRal Optimisation via Low-rank Learning (EGGROLL), which improves arithmetic intensity by structuring individual perturbations as rank-r matrices, resulting in a hundredfold increase in training speed for billion-parameter models at large population sizes, achieving up to 91% of the throughput of pure batch inference. We provide a rigorous theoretical analysis of Gaussian ES for high-dimensional parameter objectives, investigating conditions needed for ES updates to converge in high dimensions. Our results reveal a linearising effect, and proving consistency between EGGROLL and ES as parameter dimension increases. Our experiments show that EGGROLL: (1) enables the stable pretraining of nonlinear recurrent language models that operate purely in integer datatypes, (2) is competitive with GRPO for post-training LLMs on reasoning tasks, and (3) does not compromise performance compared to ES in tabula rasa RL settings, despite being faster. Our code is available at https://eshyperscale.github.io/.

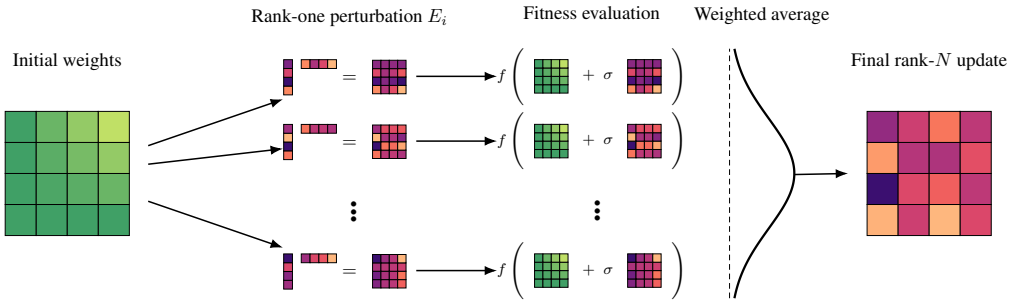


Figure 1: Schematic visualisation of EGGROLL using N workers.

1 Introduction

Evolution Strategies (ES) (Rechenberg, 1978; Beyer, 1995; Beyer & Schwefel, 2002) is an attractive alternative to first-order methods based on gradient backpropagation for several reasons. First, ES does not require differentiability; it can optimise a broader class of models, like those with discrete parametrisations (cellular

[*]Equal Contribution [†] Core Contributor, sorted by alphabetical order in first names [‡]Equal Senior Authors

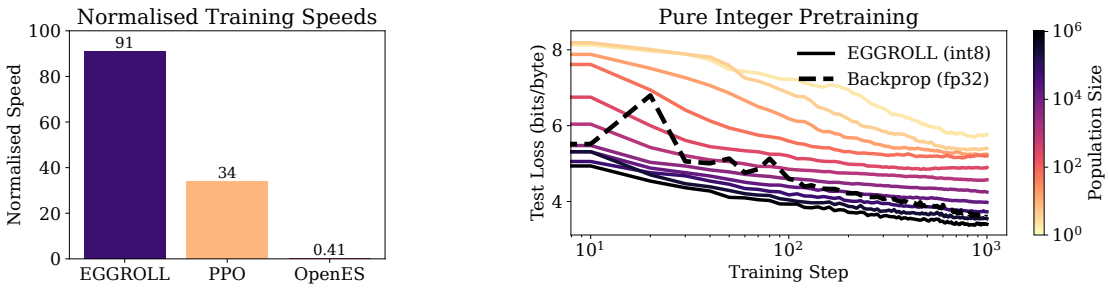


Figure 2: (a) Relative speed of our method, EGGROLL, in terms of experience throughput versus prior methods, where 100 is the maximum batch inference throughput. See Appendix E for more details. (b) We use EGGROLL to train an int8 RNN language model from scratch, scaling population size from 2 to 1,048,576 with a fixed data batch size of 16. The dotted line is a fp32 Transformer trained with backprop SGD. EGGROLL's test next-token cross-entropy of 3.40 bits/byte while backprop only gets 3.58 bits/byte.

automata) or objectives for which gradients are unavailable or noisy, such as outcome-only rewards in LLM fine-tuning (Qiu et al., 2025). Second, ES can be more robust to noisy and ill-conditioned optimisation landscapes (Wierstra et al., 2011; Xue et al., 2021). Population-based exploration smooths irregularities (Salimans et al., 2017), tolerates discontinuities, and mitigates issues like ill-conditioned curvature or vanishing and exploding gradients in long-range or recurrent settings (Hansen, 2023). Third, ES is highly amenable to parallel scaling, since fitness evaluations are independent across population members and require only the communication of scalar fitnesses, which maps cleanly onto modern inference infrastructure and yields near-linear speedups on large clusters (Salimans et al., 2017). By contrast, backpropagation requires communicating and aggregating gradients across devices, yielding updates with high memory and computational costs. Furthermore, backpropagation requires special care when training models with low-precision datatypes (Fishman et al., 2025), whereas ES can directly optimise any model with the same datatypes used at inference time. Together, these properties position ES as a potentially powerful tool for training large, discrete, or hybrid architectures, and end-to-end systems with non-differentiable components, including LLMs (Brown et al., 2020; Chowdhery et al., 2023; Du et al., 2022; Fedus et al., 2022).

However, there are currently practical obstacles to employing ES at scale. In deep learning architectures (Goodfellow et al., 2016), the majority of trainable parameters form linear mappings represented by matrices (Rosenblatt, 1962; Hochreiter & Schmidhuber, 1996; Bengio et al., 2000; Krizhevsky et al., 2012; Goodfellow et al., 2014; Kingma & Welling, 2014; Vaswani et al., 2017). Naïvely adapting ES therefore requires generating full-rank matrix perturbations that replicate the entire parameter set for every population member. This inflates memory costs and forces frequent movement of large weight tensors. Evaluating these perturbations then requires a separate sequence of matrix multiplications per member, so the total compute and wall-clock time scale roughly with the population size and sequence length since batched matrix multiplication has a low arithmetic intensity, i.e., the ratio of arithmetic operations to memory traffic (Williams, 2008). In billion-parameter regimes, these two costs dominate, limiting ES to small models and small populations (Qiu et al., 2025; Korotyshova et al., 2025).

To mitigate both memory and computational bottlenecks, we introduce Evolution Guided GeneRal Optimisation via Low-rank Learning (EGGROLL), an ES algorithm that allows for the efficient training of neural network architectures with billions of parameters. Analogous to LoRA's low-rank adapters in gradient-based training (Hu et al., 2022), EGGROLL generates *low-rank* parameter-space perturbations for ES; instead of sampling a full-rank matrix $E \in \mathbb{R}^{m\times n}$, we sample $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{n\times r}$ with $r \ll \min(m, n)$ and form $E = \frac{1}{\sqrt{r}}AB^\top$. This reduces auxiliary perturbation matrix storage from mn to $(m + n)r$ per layer, and proportionally reduces tensor movement.

Moreover, we use a counter-based deterministic random number generator (RNG) (Salmon et al., 2011; Bradbury et al., 2018) to reconstruct noise on demand, so matrix perturbations need not persist in memory. When evaluating the fitness of members of multiple perturbations in parallel, EGGROLL batches a population of low-rank adapters and shares the base activations, enabling a single forward pass that applies all $AB^\top$ updates via specialised batched matrix multiplications with significantly higher arithmetic intensity, resulting

in over a hundredfold increase in training throughput for large neural networks at large population sizes, as shown in Fig. 2a. Crucially, EGGROLL does not restrict updates to be low-rank, as the overall update is a weighted average of rank r matrices across the population, making the matrix parameter update rank $\min(Nr, m, n)$.

To understand ES when applied to large parameter models, we analyse the convergence properties of general Gaussian ES in high dimensions, showing there exists a critical noise scaling $\sigma_d = o(d^{-1/2})$ under which the update provably linearises and converges to the first-order derivative for a broad class of (possibly discontinuous) objectives. We identify three distinct regimes—linearisation, critical, and divergence—and establish provably tight conditions for stable ES optimisation in large models. Building on this, we extend the analysis to EGGROLL and prove that even fixed low-rank updates (including rank-1) converge to the true ES gradient as dimension grows, despite heavier-tailed perturbations. Our results explain the empirical success of EGGROLL in high-dimensional neural networks and connect its behaviour to neural tangent kernel-style linearisation (Jacot et al., 2018), yielding explicit convergence rates under standard overparameterised regimes. We also provide a rigorous theoretical analysis of the low-rank approximation accuracy, proving that EGGROLL updates converge to the full-rank Gaussian ES updates at a fast $\mathcal{O}(r^{-1})$ rate.

Furthermore, in our extensive empirical evaluation, we test this hypothesis across a wide range of domains. In tabula rasa and multi-agent RL (MARL) settings, we show that EGGROLL does not compromise performance compared to naïve ES despite being faster. We demonstrate the scalability of EGGROLL for LLM fine-tuning with experiments on pretrained RWKV7 (Peng et al., 2025) models, modern recurrent language models that enable large batch inference due to their constant state size. Finally, we develop a nonlinear RNN language model that operates purely in integer datatypes, and demonstrate that EGGROLL can stably pretrain this language model, a feat which is only feasible due to the large population sizes enabled by EGGROLL.

2 Preliminaries

2.1 Low-Rank Matrix Approximations

When adapting high-dimensional foundation models for specific tasks, updating the parameters using gradient-based methods has high memory requirements. LoRA (Hu et al., 2022) applies low-rank approximations to the matrix multiplications to reduce these costs. For each matrix $M_i \in \mathbb{R}^{m\times n}$ in the model, a low-rank approximation can be made by decomposing each matrix:

$$M_i \approx M_i^0 + A_i B_i^\top,$$

where $M_i^0 \coloneqq \text{StopGrad}(M_i)$ is the imported matrix from the foundation model with frozen parameters and $A_i \in \mathbb{R}^{m\times r}$ and $B_i \in \mathbb{R}^{n\times r}$ are low-width column matrices (i.e., $r \ll \min(m, n)$) whose parameters are updated through gradient-based optimisation during task-specific adaptation. This reduces the number of optimisation parameters for each matrix from mn to $r(m + n)$. EGGROLL uses a similar low-rank approximation for evolutionary strategies.

2.2 Evolution Strategies

Evolution strategies (ES) (Rechenberg, 1978; Beyer, 1995; Beyer & Schwefel, 2002) is a set of black-box optimisation methods that has emerged as a useful alternative to first-order gradient-based methods like stochastic gradient descent (SGD), particularly for noisy or non-differentiable systems. Let $f : \mathbb{R}^d \to \mathbb{R}$ denote an objective to be optimised, known as the *fitness*, where the goal is to find an optimising set of parameters $x^\star \in \arg\max_{x\in\mathbb{R}^d} f(x)$. Each set of parameters is collected into a d-dimensional vector known as a genotype. We denote the derivative of the fitness $\nabla_x f(x)|_{x=a}$ evaluated at $x = a$ as $\nabla f(a)$. Unlike first-order gradient-based methods, which query derivatives $\nabla f(x)$ to update the vector of parameters x, evolutionary methods update a parametric population distribution over the fitness parameter space $\pi(x|\theta)$, which is smoothly parametrised by a separate set of parameters $\theta \in \Theta$. The population distribution generates perturbations $x \sim \pi(x|\theta)$ known as mutations. The problem of optimising the fitness $f(x)$ for x reduces to optimising the parameters of the population distribution θ. This is achieved by solving a *secondary* optimisation problem to maximise the expected fitness under $\pi(x|\theta)$ for θ:

$$J(\theta) = \mathbb{E}_{x\sim\pi(x|\theta)}\left[f(x)\right].$$

Introducing a population distribution *smooths* the fitness landscape; since $\pi(x|\theta)$ is smooth in θ, the resulting objective $J(\theta)$ is also smooth in θ, provided $f(x)$ is measurable and integrable but not necessarily differentiable. Evolution strategies can therefore optimise black-box problems that may be non-differentiable as the derivatives of $J(\theta)$ exist for fitness functions that are discontinuous, yielding a gradient with respect to θ:

$$\nabla_\theta J(\theta) = \mathbb{E}_{x\sim\pi(x|\theta)}\left[\nabla_\theta \log \pi(x|\theta) f(x)\right],$$

where $\nabla_\theta \log \pi(x|\theta)$ is known as the score function. A Monte Carlo estimate is formed by sampling N search mutations $x_i \sim \pi(x_i|\theta)$ and computing an average of the score-weighted fitnesses:

$$\hat{\nabla}_\theta J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi(x_i|\theta) f(x_i), \tag{1}$$

with which we update θ via stochastic gradient ascent with a suitable stepsize α_t:

$$\theta_{t+1} \leftarrow \theta_t + \alpha_t \hat{\nabla}_\theta J(\theta_t).$$

ES does not require taking derivatives directly through the fitness function; instead the Monte Carlo update in Eq. (1) only requires evaluation of $f(x_i)$ for each mutation x_i to estimate $\nabla_\theta J(\theta)$. As ES only queries $f(x)$ and not $\nabla f(\mu)$, it is a *zeroth-order* optimisation method.

In this paper, we study ES using Gaussian population distributions: $\pi(x|\theta) = \mathcal{N}(\mu, I_d\sigma^2)$. In addition to its mathematical convenience, the central limit theorem means that the Gaussian distribution emerges naturally from the EGGROLL low-rank approximation as rank increases, even if the matrices A and B are themselves non-Gaussian. Moreover, most widely-used ES algorithms assume Gaussian population distributions (Rechenberg, 1978; Schwefel, 1995; Hansen & Ostermeier, 2001a; Beyer & Schwefel, 2002; Auger & Hansen, 2011; Wierstra et al., 2011; Salimans et al., 2017). In our setting, ES optimises over the population mean $\mu \in \mathbb{R}^d$, which acts as a proxy for the true maximum of the fitness function, and the variance parameter $\sigma^2 \geq 0$ is treated as a hyperparameter to be tuned.

For the Gaussian population distribution we study in this paper, the ES update can be written using an expectation under a standard normal distribution by making a transformation of variables $v = \frac{x-\mu}{\sigma}$ (Wierstra et al., 2011; Salimans et al., 2017):

$$\begin{aligned}\nabla_\mu J(\theta) &= -\frac{1}{\sigma}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\nabla_v \log p(v)\cdot f(\mu+\sigma v)\right], \\ &= \frac{1}{\sigma}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[v\cdot f(\mu+\sigma v)\right],\end{aligned} \tag{2}$$

where $v \sim P(v) = \mathcal{N}(0, I_d)$ and $p(v)$ denotes the density of $P(v)$. In this form, Eq. (2) shows that Gaussian ES methods optimise the fitness by generating search vectors from a standard normal distribution $\mathcal{N}(0, I_d)$ around the mean parameter μ.

2.3 Evolution Strategies for Matrix Parameters

A key focus of this paper is to develop efficient methods for evolution strategies that target *matrix parameters*. When working in matrix space, it is convenient to use the matrix Gaussian distribution (Dawid, 1981), which is defined directly over matrices $X \in \mathbb{R}^{m\times n}$:

$$\mathcal{N}(M, U, V) = \frac{1}{(2\pi)^{\frac{mn}{2}}\det(U)^{\frac{n}{2}}\det(V)^{\frac{m}{2}}}\exp\left(-\frac{1}{2}\text{tr}\left(V^{-1}(X-M)^\top U^{-1}(X-M)\right)\right),$$

where $M \in \mathbb{R}^{m\times n}$ is the mean matrix, $U \in \mathbb{R}^{m\times m}$ is the row covariance matrix and $V \in \mathbb{R}^{n\times n}$ is the column covariance matrix. We use $\text{vec}(\cdot)$ to denote the vectorisation operator:

$$\text{vec}(X) \coloneqq [x_{1,1}, \ldots x_{m,1}, x_{1,2}, \ldots x_{m,n}]^\top.$$

The matrix Gaussian distribution is a generalisation of the multivariate Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ defined over vector space. Sampling a matrix $X \sim \mathcal{N}(M, U, V)$ from a matrix Gaussian distribution is equivalent

to sampling a vector $\text{vec}(X) \sim \mathcal{N}(\mu, \Sigma)$ from a multivariate Gaussian distribution with mean $\mu = \text{vec}(M)$ and covariance matrix $\Sigma = V \otimes U$ where $\otimes$ denotes the Kronecker product. For isotropic matrix Gaussian distributions with covariance matrices $U = \sigma I_m$ and $V = \sigma I_n$, the equivalent multivariate Gaussian distribution is also isotropic with $\Sigma = \sigma^2 I_{mn}$. We denote the ℓ^2 vector norm as $\|\cdot\|$ and to measure distance between matrices, we use the Frobenius norm:

$$\|M\|_F \coloneqq \sqrt{\sum_{i,j} {m_{i,j}}^2} = \|\text{vec}(M)\|\,,$$

which provides an upper bound on the matrix 2-norm (Petersen & Pedersen, 2012). Let $W \in \mathbb{R}^{m \times n}$ be a set of matrix parameters where $\text{vec}(W)$ forms a subset of the full parameter vector x, typically parametrising the weights of a linear layer in a neural network. As we derive in Section B, the Gaussian ES update associated with the matrix W is:

$$\begin{aligned}\nabla_M J(\theta) &= -\frac{1}{\sigma}\mathbb{E}_{E\sim P(E)}\left[\nabla_E \log p(E) \cdot f(W = M + \sigma E)\right], \\ &= \frac{1}{\sigma}\mathbb{E}_{E\sim P(E)}\left[E \cdot f(W = M + \sigma E)\right], \end{aligned} \tag{3}$$

where M is the mean matrix associated with W, i.e. $\text{vec}(M)$ forms a subset of μ, and $P(E)$ is a zero-mean standard normal matrix distribution: $p(E) = \mathcal{N}(0, I_m, I_n)$. The gradient in Eq. (3) is estimated using the Monte Carlo estimate:

$$\hat{\nabla}_M J(\theta) = \frac{1}{\sigma N}\sum_{i=1}^{N} E_i \cdot f(W = M + \sigma E_i),$$

by sampling N search matrices $E_i \sim P(E_i)$ from a standard matrix normal distribution $\mathcal{N}(0, I_m, I_n)$ around the mean parameter matrix M, which is updated via stochastic gradient ascent:

$$M_{t+1} \leftarrow M_t + \alpha_t \hat{\nabla}_M J(\theta_t).$$

3 Related Work

3.1 Evolutionary Algorithms

Evolutionary algorithms have long been a compelling alternative to backpropagation-based training methods (e.g., genetic algorithms (Such et al., 2018) or symbolic evolution (Koza, 1994)). Much research in evolution has focused on developing algorithms for deep learning that scale well to distributed parallel computation (Jaderberg et al., 2017; Hansen & Ostermeier, 2001b; Salimans et al., 2017). These approaches have increased in popularity following the application of ES to policy learning in deep RL environments (Salimans et al., 2017). Since then, evolution has been widely applied in other domains, such as meta-learning (e.g., (Lu et al., 2022; Metz et al., 2022; Lange et al., 2023; Goldie et al., 2024; 2025)), hyperparameter tuning (e.g., (Parker-Holder et al., 2021; Tani et al., 2021; Vincent & Jidesh, 2023)), and drug discovery (Towers et al., 2025). ES has also enabled the development of neural network architectures that are unsuitable for backpropagation, such as activation-free models that exploit floating point rounding error as an implicit nonlinearity (Foerster, 2017). Here, we consider how to apply ES at a scale beyond the small networks and population sizes of prior work. For example, Salimans et al. (2017) use a maximum population size of 1440, whereas we use over a million.

While low-rank structures have been used in prior evolutionary algorithms, they have been applied to different ends, with different trade-offs, relative to EGGROLL. Choromanski et al. (2019) use a low-rank search space found via principal component analysis, which provides a better search direction to more efficiently use small populations. Garbus & Pollack (2025) optimise a low-rank factorisation instead of the full dense matrix with neuroevolution, achieving similar computational gains to EGGROLL but is limited to the low-rank structure regardless of population size.

3.2 Evolution Strategies for LLMs

Although gradient backpropagation is typically used for LLM training and fine-tuning, prior work explores ES variants for fine-tuning. In particular, Zhang et al. (2024)'s two-point zeroth-order gradient estimator, which can be viewed as an ES-inspired method using a single perturbation direction and two function queries per update, is used by Malladi et al. (2023) for memory-efficient LLM fine-tuning. Yu et al. (2025) extend this approach by projecting perturbations to a low-rank subspace, improving convergence. Jin et al. (2024) perform ES directly on LoRA matrices. These works focus on supervised fine-tuning and report performance comparable to full fine-tuning, but do not address whether pretraining is possible with two-point zeroth-order methods; we find that large population sizes are necessary for pretraining, indicating such methods are unsuitable here.

Recent work also explores ES in the context of LLM reasoning. Korotyshova et al. (2025) first train LoRA adapters using supervised fine-tuning (SFT) before decomposing them into fixed SVD bases alongside singular values that are trained using CMA-ES. They achieve comparable performance to GRPO (Shao et al., 2024) in significantly less wall-clock time on maths reasoning benchmarks. Qiu et al. (2025) directly use ES to optimise all LLM parameters for reasoning, with stronger performance than GRPO on the countdown reasoning task. However, both of these approaches use relatively small population sizes, on the order of a hundred unique perturbations per update, and instead collect hundreds of rollouts per perturbation to efficiently use GPUs. By contrast, our approach allows all generations to use different perturbations, such that our maximum population size per update is orders of magnitude larger (equal to the maximum inference batch size), without compromising token generation throughput.

4 EGGROLL

We now introduce EGGROLL (Algorithm 1). A practical issue with using a low-rank matrix approximation is that its distribution and score function have no analytic solution except for degenerate cases, so in Section 4.1 we derive the EGGROLL approximate score function from the limiting high-rank Gaussian. Section 4.2 describes how to efficiently implement EGGROLL on modern hardware.

4.1 Low-Rank Evolution Strategies

Recall the Gaussian matrix ES update from Eq. (3). Our goal is to introduce a tractable approximation to generating full-rank matrices by using low-rank matrices $AB^\top$ as our search matrices instead. Let $p(A)$ and $p(B)$ denote the distribution of $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{n\times r}$.

Assumption 1 (I.I.D. Sampling)**.** *Assume all elements $a_{i,j} \in A$ and $b_{i,j} \in B$ are continuous, identically and independently distributed random variables according to some zero-mean, symmetric, absolutely continuous distribution $p_0(\cdot)$ with finite fourth-order moments and unit variance.*

Algorithm 1 EGGROLL$(r, \alpha, \sigma, T_{\text{max}}, N_{\text{workers}})$

initialise M and workers with known random seeds ς
for T_{max} timesteps **do**
  for each worker $i \in \{1, \dots N_{\text{workers}}\}$ in parallel **do**
    $A_i \sim p(A_i), B_i \sim p(B_i)$
    $E_i \leftarrow \frac{1}{\sqrt{r}} A_i B_i^\top$
    $f_i \leftarrow f(W = M + \sigma E_i)$
  end for
  workers share scalar fitness f_i with other workers
  for each worker $i \in \{1, \dots N_{\text{workers}}\}$ in parallel **do**
    reconstruct E_j for $j \in \{1, \dots N_{\text{workers}}\}$ from ς
    $M \leftarrow M + \alpha \frac{1}{N_{\text{Workers}}} \sum_{j=1}^{N_{\text{Workers}}} E_j f_j$
  end for
end for

This assumption is easily satisfied for most perturbation distributions used by ES, including members from the set of generalised Gaussian distributions like Laplace, normal, and uniform distributions. We then form a low-rank search matrix: $E = \frac{1}{\sqrt{r}} AB^\top$. The $\frac{1}{\sqrt{r}}$ scaling ensures the variance of E remains bounded for all r. We denote the induced distribution of E as $P(E)$. $E = \frac{1}{\sqrt{r}} AB^\top$ maps to the manifold $\mathbb{M}^r \subset \mathbb{R}^{m\times n}$ of rank-r matrices. Hence, the density $p(E)$ is defined with respect to a unit volume on the manifold and cannot be defined with respect to the standard unit volume in Euclidean space. For the corresponding score function, gradients with respect to $\log p(E)$ are not defined over the usual Euclidean space. Instead, we use an approximation $\hat{S}(E) : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$ for the score function, yielding our low-rank update:

$$\hat{g}_{\text{LR}} = -\frac{1}{\sigma} \mathbb{E}_{E\sim p(E)} \left[\hat{S}(E) \cdot f(W = M + \sigma E) \right]. \tag{4}$$

In our experiments, analysis and Algorithm 1, we use a Gaussian approximate score function:

$$\hat{S}(E) = -E, \tag{5}$$

which is the score function for the Gaussian distribution $\mathcal{N}(0, I_m, I_n)$. This choice is motivated by two theoretical insights from Section 5. The matrix $AB^\top$ can be decomposed as a sum of independent, zero-mean vector outer products. Under Assumption 1, the central limit theorem applies to this sum of variables, proving that $P(E)$ converges in distribution to a Gaussian $\mathcal{N}(0, I_m, I_n)$ as rank r increases, recovering the approximate Gaussian score in the limit. Secondly, we investigate the convergence of ES and EGGROLL as the number of parameters grows, proving both updates converge to a linearised form that is consistent with the EGGROLL update using the Gaussian approximate score function.

EGGROLL is not wedded to any particular score function approximator and we derive and explore a set of mean-field approximators in Appendix D.1 as alternatives. However, our experiments show that the Gaussian approximator has the best overall performance on the tasks we consider. To optimise the ES objective using the EGGROLL update, we adapt the parallelised evolutionary strategies algorithm from Salimans et al. (2017). We make a Monte Carlo estimate of the expectation in Eq. (4) with N_{workers} samples to optimise the mean matrix parameters M using (approximate) stochastic gradient ascent. This yields the Gaussian EGGROLL update:

EGGROLL UPDATE: For each worker i (in parallel), sample $A_{i,t} \sim p(A_{i,t}), B_{i,t} \sim p(B_{i,t})$ and form a low-rank perturbation $E_{i,t} = \frac{1}{\sqrt{r}} A_{i,t} B_{i,t}^\top$. Update matrix parameters using:

$$M_{t+1} \leftarrow M_t + \frac{\alpha_t}{N_{\text{workers}}} \sum_{i=1}^{N_{\text{workers}}} E_{i,t} f(W = M_t + \sigma E_{i,t}). \tag{6}$$

Here we absorb the constant $\frac{1}{\sigma}$ into the tunable learning rate α_t. As each random matrix $E_{i,t}$ in Eq. (6) has rank r almost surely and the matrix is updated using a sum of N_{worker} such matrices, the overall EGGROLL matrix parameter update has rank $\min(Nr, m, n)$ almost surely, i.e., the overall parameter update is not restricted to be low-rank. For all experiments in Section 6, $Nr > \min(m, n)$, i.e., EGGROLL parameter updates are full-rank.

4.2 Hardware-Efficient Implementation

A key reason to use EGGROLL over standard ES is that large populations can be simulated in parallel on a GPU thanks to the low-rank perturbations. For the sake of exposition, we write equations from the perspective of a single worker, i, and explain in text how this corresponds to batched GPU operations. Consider the task of computing a batched forward pass over inputs $u_i \in \mathbb{R}^{d_{in}}$ for a linear layer with mean parameter $M \in \mathbb{R}^{d_{out} \times d_{in}}$. The standard forward pass is just a regular matrix multiplication, $u_i M^T$, since M is constant across all threads. In contrast, naïvely applying ES by trying to compute $u_i(M + \sigma E_i)^T$ becomes a batched matrix multiplication, which is inefficient on GPUs since every element of $M + \sigma E_i$ is only used in a single multiplication, yielding poor arithmetic intensity.

However, with EGGROLL we know that $u_i(M + \sigma E_i)^T = u_i M^T + \frac{\sigma}{\sqrt{r}}(u_i B_i) A_i^T$, which improves arithmetic intensity since it preserves the efficient general matrix multiplication used in batched inference while adding some additional cheap work per perturbation. In this context, the bulk of compute is spent on the efficient calculation of $u_i M^T$ using regular matrix multiplication. Meanwhile, when $r = 1$, $u_i B_i$ simply becomes an inexpensive batch of N vector-vector dot products of length d_{in} to get a batch of N scalars, which is then processed by a batched scalar-vector multiplication when multiplying by A_i^T. This decomposition is key to efficient batched LoRA inference, such as those used by vLLM (Kwon et al., 2023), which is why EGGROLL achieves the same speeds as batched LoRA inference systems. The batched LoRA inference enables high arithmetic intensity, enabling us to saturate compute with many unique perturbations per input. Note that this is impossible with naïve ES because each perturbation requires a separate matrix-vector multiplication, setting an upper bound of 1 for arithmetic intensity regardless of population size; see Appendix F for a full derivation. We additionally optimise the update by not explicitly materialising the individual E_i in the computation of $\sum_{i=1}^N E_i f_i$, the key term in the Gaussian approximate score function. In particular, when the rank is 1, we reconstruct $A \in \mathbb{R}^{N \times d_{out}}$ and $B \in \mathbb{R}^{N \times d_{in}}$ and calculate the expression as $(\text{diag}(f)A)^T B$, a simple matrix multiplication.

5 Analysis

Proofs for all theorems can be found in Appendices A to D.

In this section, we investigate the theoretical properties of the ES and EGGROLL updates. In Section 5.1, we study the convergence properties of the general Gaussian ES update as the parameter dimension $d \to \infty$, obtaining the conditions required for convergence to a linearised form. We then extend this analysis to the EGGROLL update in Section 5.2. Finally, in Section 5.3 we provide an analysis investigating the effect that increasing the rank of the EGGROLL approximation, proving convergence to the true ES update in the limit.

5.1 High-Dimensional Gaussian ES

We first analyse the general ES update under Gaussian perturbations from Eq. (2):

$$\nabla_\mu J(\theta) = \frac{1}{\sigma_d}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[v \cdot f(\mu + \sigma_d v)\right],$$

where $v \in \mathbb{R}^d$. In high dimensions, the Gaussian annulus theorem (Vershynin, 2018; Wegner, 2024) proves that the probability mass of standard Gaussian distributions concentrates in thin shells of radius $\sqrt{d}$, which place probability mass further from the origin as dimension d increases. To counter this, we let σ_d depend on d and analyse the *critical decay rate* of σ_d that yields convergence of the ES updates. We make the following mild regularity assumptions:

Assumption 2 (Locally Continuous Fitness)**.** *With probability 1 with respect to the random initialisation of μ, assume there exists a ball $B_\rho(\mu) \coloneqq \{x' | \|x' - \mu\| < \rho\}$ of fixed radius $\rho > 0$ where $f(x)$ is C^1-continuous for all $x \in B_\rho(\mu)$. Within this ball, let $\nabla f(x)$ be α-Hölder continuous, i.e., $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|^\alpha$ for all $x, y \in B_\rho(\mu)$, $\alpha \in (0, 1]$ and $L = \mathcal{O}(1)$.*

Assumption 2 *does not restrict the fitness to be globally continuous*; with probability one with respect to the initialisation distribution there must exist an arbitrarily small C^1-continuous ball around μ. In particular, discontinuities, kinks, and non-differentiable regions may exist in the domain, provided they are not encountered with nonzero probability in the local region explored by the algorithm. α-Hölder is the weakest simple, dimension-robust assumption that guarantees vanishing local gradient variation under Gaussian perturbations; it is weaker than Lipschitz continuity, which is recovered with $\alpha = 1$.

Assumption 3 (Global Polynomial Growth)**.** *Assume that there exists some constant $0 < C < \infty$ that is $\mathcal{O}(1)$ in d and finite polynomial degree $p \geq 0$ such that $|f(\mu + \sigma_d v)| \leq C(1 + \|\mu + \sigma_d v\|^p)$ and $\|\nabla f(\mu + \sigma_d v)\| \leq C(1 + \|\mu + \sigma_d v\|^p)$ almost surely under $v \sim \mathcal{N}(0, I_d)$.*

Unlike Assumption 2, this is a *global* assumption. Again, discontinuities can exist. The assumption is weaker than boundedness, is satisfied by essentially all fitness functions used in ES, and ensures that both the objective and its gradient are integrable under Gaussian perturbations; objectives violating this condition typically exhibit super-polynomial growth and derivative growth, which leads to ill-defined or highly unstable ES updates. Moreover, if the condition is not satisfied almost surely, then the function and its gradients are undefined in regions that have nonzero Gaussian measure.

Assumption 4 (Bounded Derivative)**.** *With probability 1 with respect to the random initialisation of μ, assume that $\|\mu\| = \mathcal{O}(1)$ and $\|\nabla f(\mu)\| = \mathcal{O}(1)$, i.e. $\|\mu\|$ and $\|\nabla f(\mu)\|$ do not grow with increasing d.*

This assumption is standard in high-dimensional analysis proving convergence to linearity, as proving convergence to $\nabla f(\mu)$ becomes meaningless if $\|\nabla f(\mu)\| \to \infty$. Moreover, the ES update as a whole can diverge if Assumption 4 is not satisfied. It can be ensured by scaling, typically by scaling networks parameters by $d^{-\frac{1}{2}}$ or using an appropriate scaled initialisation, commonly Gaussian initialisation $\mu \sim \mathcal{N}\left(0, \frac{1}{d}I_d\right)$. This is precisely the scaling employed in the neural tangent kernel (NTK) regime (Jacot et al., 2018; Lee et al., 2019; Chizat et al., 2019), where it guarantees dimension-independent gradients and stable training dynamics.

These assumptions encompass essentially all objectives encountered in modern machine learning, including networks with finitely many ReLU activations, max- and hinge-based losses, and other piecewise-smooth or discontinuous models. Our first theorem proves convergence of a Gaussian ES update to a linearised form, that is to the local first-order derivative $\nabla f(\mu)$, with a tight convergence rate for any function satisfying these assumptions:

Theorem 1 (Convergence to Linearity)**.** *Let Assumptions 2, 3, and 4 hold and* $\sigma_d = o\left(d^{-\frac{1}{2}}\right)$*. Then:* $\|\nabla_\mu J(\theta) - \nabla f(\mu)\| = \Theta\left(\left(\sigma_d\sqrt{d}\right)^\alpha\right) = o(1)$*, almost surely with respect to the distribution over* μ*.*

To understand the effect that breaching the $\sigma_d = o\left(d^{-\frac{1}{2}}\right)$ rate has on the convergence of Gaussian ES, we study the space of functions that can be represented by cubic polynomials of the form:

$$f(x) = a^\top x + \frac{1}{2}x^\top Bx + \frac{1}{6}C(x,x,x), \tag{7}$$

where $a \in \mathbb{R}^d$, $B \in \mathbb{R}^{d\times d}$ is a symmetric matrix and $C(x,x,x) = \sum_{i,j,k} c_{i,j,k}x_ix_jx_k$ denotes a symmetric 3-linear map represented by the symmetric 3-tensor $C \in \mathbb{R}^{d\times d\times d}$, which generalises cubic equations of the form $f(x) = ax + bx^2 + cx^3$ to vector-valued x. These are non-pathological, well-behaved, analytic C^∞-continuous functions, and include a rich subclass of convex optimisation problems, for instance, cubic perturbations of strictly convex quadratics. Moreover, any convex C^3-continuous objective admits a local third-order Taylor expansion of this form around a minimiser.

Theorem 2 (Exact Divergence for Cubic Objectives)**.** *Let* $f(x)$ *denote the cubic polynomial in Eq. (7). Assume* $\|a\| = \mathcal{O}(1), \|B\| = \mathcal{O}(1)$*,* $\|C\| = \mathcal{O}(1)$ *where* $\|\cdot\|$ *denotes operator norm for i-tensor* $T(x_1, \dots x_i)$*:* $\|T\| \coloneqq \sup_{\|x_1\|=\dots=\|x_i\|=1}|T(x_1, \dots x_i)|$*. Let Assumption 4 hold, then:*

$$\nabla_\mu J(\theta) = \nabla f(\mu) + \frac{\sigma_d^2}{2}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v,v,\cdot)\right].$$

Moreover:

$$\left\|\frac{\sigma_d^2}{2}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v,v,\cdot)\right]\right\| = \Theta(\sigma_d^2 d), \tag{8}$$

$$\|\nabla_\mu J(\theta) - \nabla f(\mu)\| = \Theta(\sigma_d^2 d). \tag{9}$$

Together, Theorems 1 and 2 prove Gaussian ES has a *critical convergence rate* of $\sigma_d = o\left(d^{-\frac{1}{2}}\right)$ in high dimensions, and operates in three regimes:

Regime I (Convergence to Linearity): For $\sigma_d = o\left(d^{-\frac{1}{2}}\right)$, ES converges to a linearised form, recovering a local first-order gradient update $\nabla f(\mu)$. This result is *analogous to neural tangent kernel* (NTK) type theorems, which prove that neural networks linearise in high dimensions (Jacot et al., 2018) and results from the concentration of the population distribution as $d \to \infty$, but applies to a more general set of objectives including discontinuous architectures. Moreover, Theorem 1 proves that the $(\sigma_d\sqrt{d})^\alpha$ rate at which Gaussian ES converges is tight and cannot in general be improved upon without strengthening continuity or introducing specific structure into the objective to ensure the Hölder constant L decays with d; for the class of cubic functions we consider in Theorem 2, the faster $\sigma_d^2 d$ convergence rate found in Eq. (9) is possible due to the C^∞-continuity of this function class, which means the converge rate is governed by third order derivative terms.

Regime II (Critical): For $\sigma_d \asymp d^{-\frac{1}{2}}$, Gaussian ES converges to a nonlinear limiting update that may retain higher-order derivative terms when they exist; for our cubic example, Eq. (8) proves that at this critical rate, the second-order term associated with the matrix B vanishes due to symmetry and the third-order term associated with the tensor C remains:

$$\left\|\frac{\sigma_d^2}{2}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v,v,\cdot)\right]\right\| = \Theta(1).$$

As the polynomial form is representative of general Taylor expansions, this implies that the limiting high dimensional update retains third-order derivatives (and higher order odd derivatives) as $d \to \infty$.

Regime III (Divergence): For $d^{-\frac{1}{2}} = o(\sigma_d)$, Theorem 2 shows that there exist smooth cubic objectives with bounded coefficients for which:

$$\|\nabla_\mu J(\theta)\| = \Theta(\sigma_d^2 d) \to \infty.$$

In particular, divergence occurs whenever the cubic tensor has a non-vanishing Gaussian contraction (equivalently, non-zero partial trace), i.e. in non-degenerate cases; only in the exceptional trace-free case does the cubic contribution vanish.

In practice, σ_d is often absorbed into the ES update stepsize, and its scale is adjusted automatically as part of the hyperparameter regime to ensure stability.

5.2 High-Dimensional EGGROLL

We now extend our high-dimensional analysis to study the EGGROLL update using the Gaussian approximate score function $\hat{g}_{\text{LR}}$ from Eq. (5). Taking r as fixed, we consider the Gaussian matrix ES setting outlined in Section 2.3. We take $x = \text{Vec}(W)$ where $W \in \mathbb{R}^{m\times n}$ and analyse the effect of increasing the total number of matrix parameters $d = mn$. Recall the true ES Gaussian matrix update is:

$$\nabla_M J(\theta) = \frac{1}{\sigma}\mathbb{E}_{E\sim P(E)}\left[E \cdot f(W = M + \sigma E)\right],$$

where M is the set of mean matrix parameters associated with the matrix W and $P(E)$ is a zero-mean standard normal $p(E) = \mathcal{N}(0, I_m, I_n)$.

Two key differences between full-rank Gaussian ES and EGGROLL are that $\hat{g}_{\text{LR}}$ is an approximation to a true gradient and $P(E)$ may have heavier tails than a Gaussian. To account for these differences, we require a slightly stricter local continuity control assumption:

Assumption 5 (EGGROLL Locally Continuous Fitness)**.** *With probability 1 with respect to the random initialisation of μ, assume there exists a ball $B_\rho(\mu) := \{x' | \|x' - \mu\| < \rho\}$ of fixed radius $\rho > 0$ where $f(x)$ is C^2-continuous for all $x \in B_\rho(\mu)$ and $\|\nabla^2 f(\mu)\|$ be polynomial bounded in d. Within this ball, let $\nabla^2 f(x)$ be Lipschitz continuous, i.e. $\|\nabla^2 f(x) - \nabla^2 f(y)\| \le L_d\|x - y\|$ for all $x, y \in B_\rho(\mu)$.*

This assumption still permits discontinuous objectives. We also assume that $p_0(\cdot)$ generates sub-Gaussian elements with uniform tail control:

Assumption 6 (Sub-Gaussian Tails)**.** *In addition to Assumption 1, assume that $p_0(\cdot)$ generates variables that have sub-Gaussian tails, i.e. for $x_i \sim p_0(x_i)$:*

$$\mathbb{P}(|x_i| > t) \le 2\exp(-Ct^2),$$

for some $0 \le C < \infty$ that does not depend on d.

We discuss sub-Gaussian variables and their properties in Section C.3 The assumption is trivially satisfied for Gaussian distributions $a \sim \mathcal{N}(0, I_m)$ and $b \sim \mathcal{N}(0, I_n)$, and holds more generally, for example for bounded distributions, uniform distributions and generalised Gaussian distributions with shape parameter greater than two. This flexibility is particularly relevant for the models in Section 6.1, where heavier-shouldered distributions may be preferred over the Gaussian.

Theorem 3 (EGGROLL Convergence to Linearity)**.** *Let $W \in \mathbb{R}^{m\times n}$, $d = mn$ and $x = \text{Vec}(W)$. Let Assumptions 3, 4, 5 and 6 hold, $\sigma_d = o(d^{-1/2})$, and $L_d(\sigma_d d)^2 = o(1)$. Then there exists some $K > 0$ such that:*

$$\|\hat{g}_{LR} - \nabla_W f(W = M)\|_F = \mathcal{O}\left(L_d(\sigma_d d)^2\right) + \mathcal{O}\left(\frac{\sqrt{d}}{\sigma_d^2}\exp\left(-K\frac{\rho}{\sqrt{d}\sigma_d}\right)\right) = o(1), \tag{10}$$

and

$$\|\hat{g}_{LR} - \nabla_M J(\theta)\|_F = \mathcal{O}\left(\sigma_d\sqrt{d}\cdot\left(1 + L_d\sigma_d d^{\frac{3}{2}}\right)\right) = o(1). \tag{11}$$

almost surely with respect to the distribution over μ.

Our theory explains the success of EGGROLL in high dimensions with rank as small as $r = 1$; Eq. (11) proves EGGROLL converges to the true update matrix ES update $\nabla_M J(\theta)$ as $d \to \infty$ regardless of r. In addition, Eq. (10) proves that under the same conditions, the EGGROLL update also linearises like the true Gaussian ES update analysed in Section 5.1, recovering a local first-order derivative as $d \to \infty$. For high-dimensional neural networks, standard parametrisations place training in the NTK regime, in which the network behaves approximately linearly in its parameters and gradient descent converges to a global minimum (Jacot et al., 2018; Lee et al., 2019; Chizat et al., 2019). Recent results show that the spectral norm of the Hessian decays polynomially with width, and that higher-order derivatives governing the variation of the Hessian also vanish (Liu et al., 2020). Consequently, the Lipschitz constant $L_d = o(1)$, typically at rate $d^{-\frac{1}{2}}$ or d^{-1} depending on the network architecture. Substituting these rates into our upper bound in Eq. (10) yields convergence rates of $\mathcal{O}(\sigma_d^2 d^{\frac{3}{2}})$ or $\mathcal{O}(\sigma_d^2 d)$ respectively.

5.3 Rank Analysis

We now analyse how fast the low-rank update from Eq. (4) with Gaussian score approximation converges to the true Gaussian ES matrix gradient in Eq. (3) as the rank of the update r increases. We make notation explicit in r in this subsection, for example writing $E^r = \frac{1}{\sqrt{r}} A^r B^{r\top}$. We introduce the following formal regularity assumption for the fitness function:

Assumption 7 (Bounded Fitness)**.** *Assume that $f(W)$ is bounded, that is $\sup_W |f(W)| < \infty$.*

Our key theoretical result characterises the error rate between the Gaussian score approximator in the low-rank update $\hat{g}^r_{\text{LR}}$ from Eq. (4) and the true gradient using the matrix Frobenius norm:

Theorem 4 (EGGROLL Rank Convergence)**.** *Let Assumptions 1 and 7 hold, then:*

$$\|\hat{g}^r_{\text{LR}} - \nabla_\mu J(\theta)\|_F = \mathcal{O}\left(r^{-1}\right). \tag{12}$$

The convergence rate in Eq. (12) is faster than the typical $\mathcal{O}\left(r^{-\frac{1}{2}}\right)$ rate dictated by the general parametric central limit theorem. Our analysis shows that this is due to the symmetry in our problem under Assumption 1. To obtain our results, we make an Edgeworth expansion (Bhattacharya & Ranga Rao, 1976) of the distribution $P(E^r)$, which expands $P(E^r)$ as the limiting Gaussian distribution plus a sum of decaying terms that are controlled by the 3rd order and higher cumulants of $P(E^r)$. Each ith order cumulant term is multiplied by a factor that decays at rate $\mathcal{O}\left(r^{-\frac{i-2}{2}}\right)$. For symmetric zero-mean distributions, all odd cumulants are zero (for the same reason that all odd moments of a symmetric distribution are zero). Hence, the rate of convergence to the limiting distribution is controlled by the 4th order term, which has rate $\mathcal{O}\left(r^{-1}\right)$.

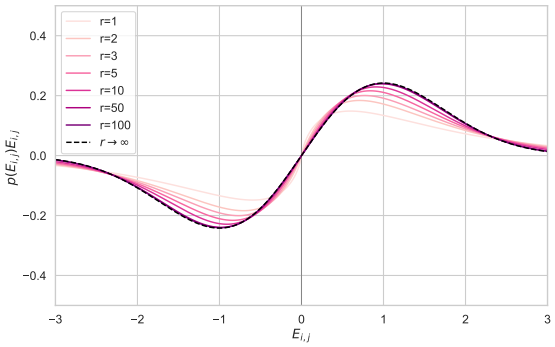


Figure 3: Plot of Marginal Score Multiplied by Density for Increasing r

Although the full distribution $P(E^r)$ has no general closed-form solution, the distribution over marginals $P(E_{i,j})$ is more amenable to analysis. We derive the density of the marginal distribution $P(E_{i,j})$ for generalised Gaussian distributed $a_{i,j}$ and $b_{i,j}$ in Section D.1. To illustrate the fast convergence rate, we plot the negative density $\times$ score function $p(E_{i,j})E_{i,j}$ for the marginal density $p(E_{i,j})$ in Fig. 3 using Gaussian distributed $a_{i,j}$ and $b_{i,j}$ (see Theorem 6 for a derivation). The figure shows that $p(E_{i,j})E_{i,j}$ quickly converges to the limiting function $\frac{E_{i,j}}{\sqrt{2\pi}} \exp\left(-\frac{{E_{i,j}}^2}{2}\right)$, recovering the Gaussian form from the true Gaussian ES update. Even at $r = 1$, the function is not a poor approximation. After $r = 10$, the function has nearly converged and after $r = 50$, the function is visually indistinguishable from the limit, providing evidence for the hypothesis that the low-rank approximation is accurate even for very low-rank regimes $r \ll \min(m, n)$.

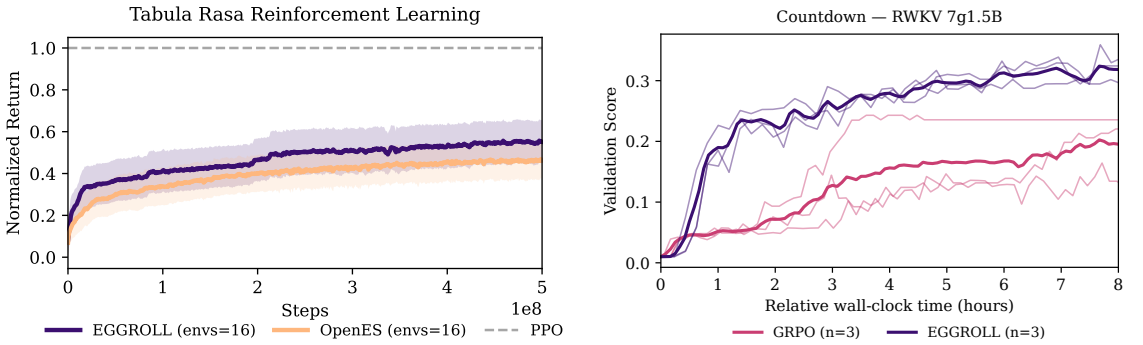


Figure 4: (a) Comparison of reinforcement learning returns normalised by PPO performance across 16 environments for 10 seeds. The shaded region is the standard error of the mean.(b) Validation score of 3 seeds of EGGROLL v.s. 3 seeds of GRPO in countdown task with an RWKV 7g1.5B model on a single GPU. EGGROLL allows 1024 parallel generations per GPU (618 updates) whereas GRPO only 64 (915 updates).

6 Experiments

In the following section we showcase the effectiveness of EGGROLL in a variety of tasks that position it as a strong alternative to back-propagation for the end-to-end training of foundation models.

6.1 Pure Integer Language Model Pretraining

To demonstrate the potential of EGGROLL as a general optimisation method, we apply it to language model pretraining. Since EGGROLL does not rely on gradients, we explicitly design a language model architecture to be efficient and hardware-friendly at inference time. To highlight EGGROLL's flexibility, we train a nonlinear recurrent neural network (RNN) in pure integer datatypes with no explicit activation functions, relying only on the implicit nonlinearity of clipping in int8 operations. We call the resulting language model EGG, the Evolved Generative GRU, an EGGROLL-friendly architecture with all weights in int8. See Appendix G for more details on the architecture and motivation behind EGG.

We train an EGG model with 6 layers and hidden dimension 256 (6L-256D) to do character-level prediction on the minipile dataset (Kaddour, 2023). We update parameters after 100 tokens for each population member, applying truncated ES by keeping the hidden state and only resetting at document boundaries. We plot the test loss in Fig. 2b over training steps across a range of population sizes with a fixed data batch size of 16 sequences per step, where the best test loss is 3.40 bits/byte. With a sufficiently large population size, EGG outperforms a dense 6L-256D Transformer trained with backprop SGD using the same data batch size. Note that larger population sizes require more parallel compute for the same amount of data; our largest population size of $2^{20} = 1048576$ requires around 180 times more GPU-hours than the backprop baseline, demonstrating the potential for compute-only scaling in limited data regimes using EGGROLL.

Moreover, our largest population size of 2^{20} is three orders of magnitude larger than the largest experiment done by Salimans et al. (2017) while only requiring a single GPU to train, highlighting EGGROLL's computational efficiency. We note that large population sizes are critical for pretraining; a population size of 2, analogous to MeZO (Malladi et al., 2023), significantly underperforms larger population sizes despite having access to the same data batch. We conduct more ablations in Appendix I, analysing the tradeoff between population size and data batch size.

6.2 Reinforcement Learning Tasks

To verify that low-rank perturbations do not change the optimisation behavior of ES in standard control settings, we benchmark EGGROLL against OpenES (Salimans et al., 2017) across 16 tabula rasa environments spanning Navix, Craftax, Brax, Kinetix, and Jumanji. We use a fixed 3-layer MLP policy (256 hidden units) and perform per-environment hyperparameter optimisation for each method before evaluating the selected configuration over 10 random seeds, reporting mean performance (normalised by PPO) and uncertainty. Overall, EGGROLL is competitive with OpenES on 7/16 environments, underperforms on 2/16, and outperforms on 7/16, while often delivering substantial wall-clock improvements due to its batched low-rank structure (full environment

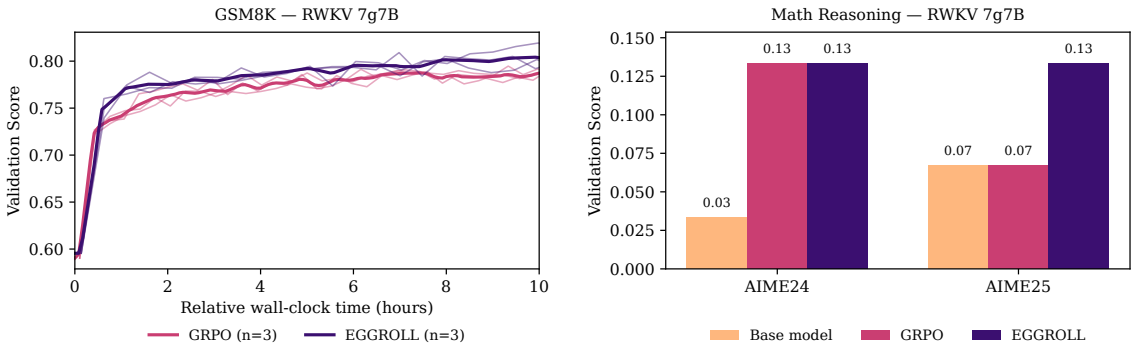


Figure 5: (a) Comparison of the validation score of 3 seeds of EGGROLL v.s. 3 seeds of GRPO in GSM8K task with an RWKV 7g7B model on 8 GPUs. EGGROLL allows 8192 parallel generations (1024 per GPU with 260 updates) whereas GRPO only 256 (32 per GPU with 340 updates). (b) Performance of our finetuned RWKV 7G 7 billion model on hard reasoning tasks using 128 GPUs for 12 hours. The model was trained using the DeepScaleR dataset and the best checkpoint was chosen by evaluating on AIME24.

list, learning curves, timing comparisons, and complete HPO ranges/settings are provided in Appendix N.4). Figure 4a shows the averaged normalised return across the 16 environments with 10 seeds per environment. We additionally report MARL results in Section N.1.

6.3 Foundation Model Fine-tuning

We apply EGGROLL to finetune an RWKV-7 (Peng et al., 2025) LLM on two reasoning tasks: countdown (Gandhi et al., 2024) and GSM8K (Cobbe et al., 2021). RWKV is a recurrent model that is better suited to parallelisation than transformers because any memory otherwise spent on the KV cache is used to evaluate population members. Figure 4b shows that EGGROLL fine-tuning on an RWKV-7 1.5B model converges to a higher validation accuracy of 35% (vs. 23%) given the same hardware and wall-clock time in the countdown task. Similarly, Figure 5a shows that EGGROLL outperforms GRPO on GSM8K fine-tuning. Our scoring function draws parallels to the group relative advantage of GRPO. In particular, to score a set of noise directions, $E \equiv \{E_1, \ldots, E_n\}$, we first compute their accuracies, $\{s_{1,q_i}, \ldots, s_{n,q_i}\}$, on $|q| = m$ questions, creating a matrix of scores $S \in \mathbb{R}^{m \times n}$. We then compute the normalised score per question, with the main difference that we use the global variance $\bar{\sigma}$, and average over all the questions to compute a score for the noise direction E_i:

$$\bar{s}_i = \frac{1}{m} \sum_{j=1}^{m} z_{i,q_j} = \frac{1}{m} \sum_{j=1}^{m} \frac{s_{i,j} - \mu_{q_j}}{\bar{\sigma}}.$$

This scoring function weights all questions within the same batch the same across population members. We use this recipe to train a 14 billion parameter RWKV 7 model on the DeepScaleR dataset and evaluate in more challenging maths reasoning tasks. In this regime, GRPO is infeasible due to the extra memory used by the Adam optimiser Kingma & Ba (2014). Using a thinking budget of 5000 tokens for training and evaluation, our fine-tuned 14B model improves from 13% to 30% accuracy on AIME24, from 7% to 33% accuracy on AIME25 and from 11% to 13% accuracy on HMMT25 after training on 32 GPUs for 12 hours (Figure 13b). On 7B models, we outperform GRPO using 128 GPUs for 24 hours (Figure 5b).

In Section L, we achieve similar performance to GRPO when fine-tuning Qwen Transformer models, and additionally demonstrate that EGGROLL can directly optimise for pass@k, a known limitation of GRPO (Yue et al., 2025). Beyond language models, we also fine-tune a finance world model into an agent for high-frequency trading that directly optimises for PnL; see Section M for more details.

6.4 Fine-tuning Integer Quantised LLMs

We follow the same procedure as Jacob et al. (2017) to quantise the RWKV-7 family of models by dividing by the maximum *per-channel* value on each weight matrix and mapping into the int8 range of $[-127, 127]$. We then apply EGGROLL with Adam to do model distillation from the original, non-quantised RWKV-7, into the resulting int8 quantised model using examples from GSM8K. See Appendix K for full details about the

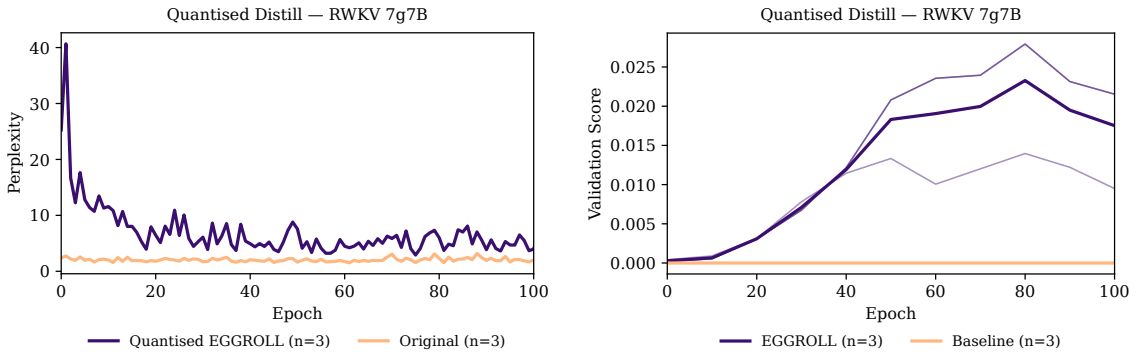


Figure 6: (a) Average per token perplexity (during training) of 3 seeds of a quantised (int8) RWKV 7G 7 billion parameter model on distillation from the non quantised model using examples from GSM8K. (b) Validation score on unseen examples of GSM8K of 3 seeds of a quantised RWKV 7G 7 billion parameter model. Initially the model is unable to solve any problems, but progressively it is capable of solving more problems. The baseline here indicates the validation score of a quantised model without any further training.

specifics of quantisation and fine-tuning. The distillation is done by matching the distributions between the quantised and non-quantised models on teacher forced examples (with solutions) from the GSM8K dataset. More specifically, the fitness for a given set of parameters, μ_i, is computed as follows:

$$f_{\mu_i}(x_{1:T}) = \sum_{t=1}^{T} \mathrm{KL}\left(p_t || q_t(\cdot; \mu_i)\right),$$

where $x_{1:T}$ is a subsequence of tokens taken from the solutions of GSM8K and $\mathrm{KL}\left(p_t || q_t(\cdot; \mu_i)\right)$ is the Kullback-Leibler divergence between the distribution of the non-quantised model, p_t, and the distribution of the quantised model q_t over the vocabulary at token t. Figure 6a shows the average per token perplexity of 3 seeds of a quantised RWKV 7G 7 billion parameter model compared to that of the original non-quantised model over the same sequence, as a baseline. Progressively, the quantised model recovers the capability to solve a subset of the GSM8K dataset (Figure 6b).

7 Conclusion

We introduce EGGROLL, a powerful method for black-box optimisation that scales evolutionary strategies to billion-parameter models and beyond using low-rank search matrices. Our experiments demonstrate that EGGROLL is effective with a rank of 1, giving substantial computational and memory savings for negligible decrease in performance when compared to the full-rank perturbations. Empirically, EGGROLL delivers large speedups over naïve ES in tabula rasa and multi-agent RL, and can power end-to-end training pipelines for foundation models. Our theoretical analysis shows that the EGGROLL update converges towards the Gaussian ES update with increasing rank r and parameter dimension $d = mn$, and we provide a rigorous study of general ES at high dimensions, deriving necessary and sufficient conditions for convergence and linearisation.

Looking forward, we can use EGGROLL for other problems beyond the reach of modern first-order gradient-based techniques. In particular, EGGROLL can enable the training of large scale end-to-end neurosymbolic systems (Sarker et al., 2021) with non-differentiable components. For instance, we can train neural networks that interface with symbolic modules for specific functions, like memory or calculations. We can also optimise end-to-end systems of language models, training them to be aware of inference-time harnesses and interactions with other agents in complex systems.

Acknowledgements

Compute for this project is graciously provided by the Isambard-AI National AI Research Resource, under the projects "FLAIR 2025 Moonshot Projects" and "Robustness via Self-Play RL." Some experiments also used compute generously given by JASMIN, the UK's collaborative data analysis environment (https://www.jasmin.ac.uk).

Bidipta Sarkar is supported by the Clarendon Fund Scholarship in partnership with a Department of Engineering Science Studentship for his Oxford DPhil. Mattie Fellows is funded by a generous grant from the UKRI Engineering and Physical Sciences Research Council EP/Y028481/1. Juan Agustin Duque is supported by the St-Pierre-Larochelle Scholarship at the University of Montreal and by Aaron Courville's CIFAR AI Chair in Representations that Generalize Systematically. Jarek Liesen and Theo Wolf are supported by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines & Systems EP/Y035070/1. Jarek Liesen is also supported by Sony Interactive Entertainment Europe Ltd. Uljad Berdica is supported by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines & Systems EP/S024050/1 and the Rhodes Scholarship. Lukas Seier is supported by the Intelligent Earth CDT with funding from the UKRI grant number EP/Y030907/1. Alexander D. Goldie is funded by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems EP/S024050/1. Jakob Nicolaus Foerster is partially funded by the UKRI grant EP/Y028481/1 (originally selected for funding by the ERC). Jakob Nicolaus Foerster is also supported by the JPMC Research Award and the Amazon Research Award.

We thank Andreas Kirsch for discovering an emergent log-linear scaling law for EGG loss with respect to int8 OPs in this tweet along with other community members for their comments and recommendations during the first arXiv release of this work.

References

Agentica Organization, Michael Luo, Sijun Tan, and Justin Wong. Deepscaler-preview-dataset. https://huggingface.co/datasets/agentica-org/DeepScaleR-Preview-Dataset, 2025. Accessed: 2025-01-14.

R. Askey and R. (eds.) Roy. Nist digital library of mathematical functions, chapter 5: Gamma function. Online: https://dlmf.nist.gov/5, 2020-2026. Section 5.11 (Stirling / asymptotic expansions), release 1.1.16.

Anne Auger and Nikolaus Hansen. Theory of evolution strategies: A new perspective. In Anne Auger and Benjamin Doerr (eds.), *Theory of Randomized Search Heuristics: Foundations and Recent Developments*, pp. 289–325. World Scientific, Singapore, 2011.

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. Matharena: Evaluating llms on uncontaminated math competitions, 2026. URL https://arxiv.org/abs/2505.23281.

A. B. Basset. *A Treatise on Hydrodynamics: with numerous examples*, volume 2. Deighton, Bell, and Co., Cambridge, UK, 1888.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In T. Leen, T. Dietterich, and V. Tresp (eds.), *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000. URL https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf.

Hans-Georg Beyer. Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation*, 3: 311–347, 1995. URL https://api.semanticscholar.org/CorpusID:17416734.

Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies –a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

R. N. Bhattacharya and R. Ranga Rao. *Normal approximation and asymptotic expansions*. Wiley series in probability and mathematical statistics. Wiley, New York, 1976. ISBN 047107201X.

Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence I. Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries P. Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn N. Waters, Mohamed A. Mimouni, Ulrich A. Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in jax, 2024. URL https://arxiv.org/abs/2306.09884.

Jean-Philippe Bouchaud, Julius Bonart, Jonathan Donier, and Martin Gould. *Trades, quotes and prices: financial markets under the microscope*. Cambridge University Press, 2018.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Lénaïc Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/ae614c557843b1df326cb29c57225459-Paper.pdf.

Krzysztof M Choromanski, Aldo Pacchiano, Jack Parker-Holder, Yunhao Tang, and Vikas Sindhwani. From complexity to simplicity: Adaptive es-active subspaces for blackbox optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/88bade49e98db8790df275fcebb37a13-Paper.pdf.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sashank Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24(1144), 2023. URL https://jmlr.org/papers/volume24/22-1144/22-1144.pdf.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

A. P. Dawid. Some matrix-variate distribution theory: Notational considerations and a bayesian application. *Biometrika*, 68(1):265–274, 1981. ISSN 0006-3444.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P. Bosma, Zongwei Zhou, Tao Wang, Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. Glam: Efficient scaling of language models with mixture-of-experts. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5547–5569, Jul 2022. URL https://proceedings.mlr.press/v162/du22c.html.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1):1–39, January 2022. ISSN 1532-4435. URL https://jmlr.org/papers/volume23/21-0998/21-0998.pdf.

Maxim Fishman, Brian Chmiel, Ron Banner, and Daniel Soudry. Scaling fp8 training to trillion-token llms, 2025. URL https://arxiv.org/abs/2409.12517.

Jakob Nicolaus Foerster. Nonlinear computation in deep linear networks, sep 2017. URL https://blog.openai.com/nonlinear-computation-in-linear-networks/. Accessed: 2025-11-20.

Gerald B. Folland. *Real Analysis: Modern Techniques and Their Applications*. John Wiley & Sons, New York, 2nd edition, 1999. See Theorem 8.22 (Riemann–Lebesgue Lemma).

Catherine Forbes, Merran Evans, Nicholas Hastings, and Brian Peacock. *Statistical Distributions*. Wiley Series in Probability and Statistics. John Wiley & Sons, Hoboken, NJ, USA, 4th edition, 2011. ISBN 9780470390634.

C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax – a differentiable physics engine for large scale rigid body simulation, 2021. URL https://arxiv.org/abs/2106.13281.

Sascha Yves Frey, Kang Li, Peer Nagy, Silvia Sapora, Christopher Lu, Stefan Zohren, Jakob Foerster, and Anisoara Calinescu. Jax-lob: A gpu-accelerated limit order book simulator to unlock large scale reinforcement learning for trading. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, pp. 583–591, 2023.

Kevin Galim, Wonjun Kang, Yuchen Zeng, Hyung Il Koo, and Kangwook Lee. Parameter-efficient fine-tuning of state space models, 2025. URL https://arxiv.org/abs/2410.09016.

Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. Simplifying deep temporal difference learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=7IzeL0kflu.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (sos): Learning to search in language, 2024. URL https://arxiv.org/abs/2404.03683.

Jack Garbus and Jordan Pollack. Low rank factorizations are indirect encodings for deep neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '25 Companion, pp. 2371–2379, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714641. doi: 10.1145/3712255.3734297. URL https://doi.org/10.1145/3712255.3734297.

Alexander D. Goldie, Chris Lu, Matthew T. Jackson, Shimon Whiteson, and Jakob N. Foerster. Can Learned Optimization Make Reinforcement Learning Less Difficult? In *Advances in Neural Information Processing Systems*, volume 37, pp. 5454–5497, 2024.

Alexander David Goldie, Zilin Wang, Jaron Cohen, Jakob Nicolaus Foerster, and Shimon Whiteson. How Should We Meta-Learn Reinforcement Learning Algorithms? May 2025. URL https://openreview.net/forum?id=jKzQ6af2DU.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/f033ed80deb0234979a61f95710dbe25-Paper.pdf.

Martin D Gould, Mason A Porter, Stacy Williams, Mark McDonald, Daniel J Fenn, and Sam D Howison. Limit order books. *Quantitative Finance*, 13(11):1709–1742, 2013.

I. S. (Izrail Solomonovich) Gradshteĭn, I. M. (Iosif Moiseevich) Ryzhik, Daniel Zwillinger, Victor Moll, and Inc Scripta Technica. *Table of integrals, series, and products*. Academic Press, San Diego ; Tokyo, 8 edition, 2015. ISBN 0123849330.

G R Grimmett and D R Stirzaker. Probability and random processes. *Journal of the Royal Statistical Society. Series A, Statistics in society*, 156(3):503–503, 1993. ISSN 0964-1998.

Peter Hall. *The bootstrap and Edgeworth expansion*. Springer series in statistics. Springer-Verlag, New York, 1992. ISBN 9780387945088.

Nikolaus Hansen. The cma evolution strategy: A tutorial, 2023. URL https://arxiv.org/abs/1604.00772.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001a.

Nikolaus Hansen and Andreas Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, June 2001b. ISSN 1063-6560. doi: 10.1162/106365601750190398. URL https://ieeexplore.ieee.org/document/6790628.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems, 2024. URL https://arxiv.org/abs/2402.14008.

Joel Heck and Fathi M. Salem. Simplified minimal gated unit variations for recurrent neural networks, 2017. URL https://arxiv.org/abs/1701.03452.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. In M.C. Mozer, M. Jordan, and T. Petsche (eds.), *Advances in Neural Information Processing Systems*, volume 9. MIT Press, 1996. URL https://proceedings.neurips.cc/paper_files/paper/1996/file/a4d2f0d23dcc84ce983ff9157f8b7f88-Paper.pdf.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, 2014. doi: 10.1109/ISSCC.2014.6757323.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *ICLR*. OpenReview.net, 2022.

Ruihong Huang and Tomas Polak. LOBSTER: Limit order book reconstruction system. *Available at SSRN 1977207*, 2011.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017. URL https://arxiv.org/abs/1712.05877.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population Based Training of Neural Networks, November 2017. URL http://arxiv.org/abs/1711.09846. arXiv:1711.09846 [cs].

Feihu Jin, Yifan Liu, and Ying Tan. Derivative-free optimization for low-rank adaptation in large language models. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 32:4607–4616, October 2024. ISSN 2329-9290. doi: 10.1109/TASLP.2024.3477330. URL https://doi.org/10.1109/TASLP.2024.3477330.

Jean Kaddour. The minipile challenge for data-efficient language models. *arXiv preprint arXiv:2304.08442*, 2023.

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

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6114.

Daria Korotyshova, Boris Shaposhnikov, Alexey Malakhov, Alexey Khokhulin, Nikita Surnachev, Kirill Ovcharenko, George Bredis, Alexey Gorbatovski, Viacheslav Sinii, and Daniil Gavrilov. Essa: Evolutionary strategies for scalable alignment, 2025. URL https://arxiv.org/abs/2507.04453.

John R. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2):87–112, June 1994. ISSN 1573-1375. doi: 10.1007/BF00175355. URL https://doi.org/10.1007/BF00175355.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Robert Tjarko Lange, Tom Schaul, Yutian Chen, Tom Zahavy, Valentin Dallibard, Chris Lu, Satinder Singh, and Sebastian Flennerhag. Discovering Evolution Strategies via Meta-Black-Box Optimization, March 2023. URL http://arxiv.org/abs/2211.11260. arXiv:2211.11260 [cs].

Pierre-Simon Laplace. Mémoire sur les intégrales définies et leur application aux probabilités, et spécialement à la recherche du milieu qu'il faut choisir entre les résultats des observations. *Mémoires de la Classe des Sciences Mathématiques et Physiques de l'Institut Impérial de France,* 1re série, 11(1re partie):297–347, 1811.

Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models, 2022. URL https://arxiv.org/abs/2206.14858.

Junjie Li, Yang Liu, Weiqing Liu, Shikai Fang, Lewen Wang, Chang Xu, and Jiang Bian. Mars: a financial market simulation engine powered by generative foundation model. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=Yqk7EyT52H.

Oscar Li, James Harrison, Jascha Sohl-Dickstein, Virginia Smith, and Luke Metz. Variance-reduced gradient estimation via noise-reuse in online evolution strategies. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Elliott H. Lieb and Michael Loss. *Analysis*. Graduate studies in mathematics ; volume 14. American Mathematical Society, Providence, Rhode Island, 2nd ed. edition, 2010 - 2010. ISBN 1-4704-1143-1.

Jarek Liesen, Chris Lu, and Robert Lange. rejax, 2024. URL https://github.com/keraJLi/rejax.

Chaoyue Liu, Libin Zhu, and Mikhail Belkin. On the linearity of large non-linear models: when and why the tangent kernel is constant. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NeurIPS 2020, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Jun S. Liu. Siegel's formula via stein's identities. *Statistics and probability letters*, 21(3):247–251, 1994. ISSN 0167-7152.

Zichen Liu, Anya Sims, Keyu Duan, Changyu Chen, Simon Yu, Xiangxin Zhou, Haotian Xu, Shaopan Xiong, Bo Liu, Chenmien Tan, Chuen Yang Beh, Weixun Wang, Hao Zhu, Weiyan Shi, Diyi Yang, Michael Shieh, Yee Whye Teh, Wee Sun Lee, and Min Lin. Gem: A gym for agentic llms, 2025. URL https://arxiv.org/abs/2510.01051.

Ryan Lowe, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35:16455–16468, 2022.

H. M. Macdonald. Zeroes of the bessel functions. *Proceedings of the London Mathematical Society*, 30: 165–179, 1899. doi: 10.1112/plms/s1-30.1.165.

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 *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. *arXiv preprint arXiv:2402.16801*, 2024.

Michael T. Matthews, Michael Beukman, Chris Lu, and Jakob Nicolaus Foerster. Kinetix: Investigating the training of general agents through open-ended physics-based control tasks. In *ICLR*, 2025. URL https://openreview.net/forum?id=zCxGCdzreM.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 35492–35506. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/merrill24a.html.

Luke Metz, James Harrison, C. Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, and Jascha Sohl-Dickstein. VeLO: Training Versatile Learned Optimizers by Scaling Up, November 2022. URL http://arxiv.org/abs/2211.09760. arXiv:2211.09760 [cs, math, stat].

Valentin Mohl, Sascha Frey, Reuben Leyland, Kang Li, George Nigmatulin, Mihai Cucuringu, Stefan Zohren, Jakob Foerster, and Anisoara Calinescu. Jaxmarl-hft: Gpu-accelerated large-scale multi-agent reinforcement learning for high-frequency trading. In *Proceedings of the 6th ACM International Conference on AI in Finance*, pp. 18–26, 2025. URL https://doi.org/10.1145/3768292.3770416.

Peer Nagy, Sascha Frey, Silvia Sapora, Kang Li, Anisoara Calinescu, Stefan Zohren, and Jakob Foerster. Generative ai for end-to-end limit order book modelling: A token-level autoregressive generative model of message flow using a deep state space network. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, ICAIF '23, pp. 91–99, 2023.

Peer Nagy, Sascha Yves Frey, Kang Li, Bidipta Sarkar, Svitlana Vyetrenko, Stefan Zohren, Ani Calinescu, and Jakob Nicolaus Foerster. LOB-bench: Benchmarking generative AI for finance - an application to limit order book data. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=CXPpYJpYXQ.

Brian Ning, Franco Ho Ting Lin, and Sebastian Jaimungal. Double deep q-learning for optimal execution. *Applied Mathematical Finance*, 28(4):361–380, 2021.

Jack Parker-Holder, Vu Nguyen, and Stephen Roberts. Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits, June 2021. URL http://arxiv.org/abs/2002.02518. arXiv:2002.02518 [cs].

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, Nathan Wilce, Johan S. Wind, Tianyi Wu, Daniel Wuttke, and Christian Zhou-Zheng. Rwkv-7 "goose" with expressive dynamic state evolution, 2025. URL https://arxiv.org/abs/2503.14456.

K. B. Petersen and M. S. Pedersen. The matrix cookbook, nov 2012. URL http://localhost/pubdb/p.php?3274. Version 20121115.

Eduardo Pignatelli, Jarek Liesen, Robert Tjarko Lange, Chris Lu, Pablo Samuel Castro, and Laura Toni. Navix: Scaling minigrid environments with jax, 2024. URL https://arxiv.org/abs/2407.19396.

Xin Qiu, Yulu Gan, Conor F. Hayes, Qiyao Liang, Elliot Meyerson, Babak Hodjat, and Risto Miikkulainen. Evolution strategies at scale: Llm fine-tuning beyond reinforcement learning, 2025. URL https://arxiv.org/abs/2509.24372.

I. Rechenberg. Evolutionsstrategien. In Berthold Schneider and Ulrich Ranft (eds.), *Simulationsmethoden in der Medizin und Biologie*, pp. 83–114, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg. ISBN 978-3-642-81283-5.

V. K. Rohatgi. *An introduction to probability theory and mathematical statistics*. Wiley series in probability and mathematical statistics. Wiley, New York, 1976. ISBN 0471731358.

Frank. Rosenblatt. *Principles of neurodynamics : perceptrons and the theory of brain mechanisms.* Spartan Books, Washington, 1962.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Garðar Ingvarsson, Timon Willi, Ravi Hammond, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, Robert Tjarko Lange, Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktäschel, Chris Lu, and Jakob Nicolaus Foerster. JaxMARL: Multi-agent RL environments and algorithms in JAX. In *The Thirty-eighth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017. URL https://arxiv.org/abs/1703.03864.

John K. Salmon, Mark A. Moraes, Ron O. Dror, and David E. Shaw. Parallel random numbers: As easy as 1, 2, 3. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2011. doi: 10.1145/2063384.2063405.

Md Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. Neuro-symbolic artificial intelligence: Current trends, 2021. URL https://arxiv.org/abs/2105.05330.

Hans-Paul Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, New York, 1995.

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, 2024. URL https://arxiv.org/abs/2402.03300.

Zhihong Shao, Yuxiang Luo, Chengda Lu, Z. Z. Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. Deepseekmath-v2: Towards self-verifiable mathematical reasoning, 2025. URL https://arxiv.org/abs/2511.22570.

Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. In *International Conference on Learning Representations*, 2023.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms, 2012. URL https://arxiv.org/abs/1206.2944.

Charles Stein. A bound for the error in the normal approximation to the distribution of a sum of dependent random variables. In *Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability*, volume 2, pp. 583–602, Berkeley, CA, 1972. University of California Press.

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning, April 2018. URL http://arxiv.org/abs/1712.06567. arXiv:1712.06567 [cs].

Laurits Tani, Diana Rand, Christian Veelken, and Mario Kadastik. Evolutionary algorithms for hyperparameter optimization in machine learning for application in high energy physics. *The European Physical Journal C*, 81(2):170, February 2021. ISSN 1434-6044, 1434-6052. doi: 10.1140/epjc/s10052-021-08950-y. URL http://arxiv.org/abs/2011.04434. arXiv:2011.04434 [hep-ex].

Nico M Temme. *Bessel Functions*, chapter 9, pp. 219–255. John Wiley and Sons, Ltd, 1996. ISBN 9781118032572. doi: https://doi.org/10.1002/9781118032572.ch9. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118032572.ch9.

Sebastian Towers, Aleksandra Kalisz, Philippe A. Robert, Alicia Higueruelo, Francesca Vianello, Ming-Han Chloe Tsai, Harrison Steel, and Jakob N. Foerster. ADIOS: Antibody Development via Opponent Shaping, June 2025. URL http://arxiv.org/abs/2409.10588. arXiv:2409.10588 [q-bio].

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Roman Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge, UK, 2018. ISBN 9781108415194. Foundational text covering concentration of norms and high-dimensional Gaussian phenomena.

Amala Mary Vincent and P. Jidesh. An improved hyperparameter optimization framework for AutoML systems using evolutionary algorithms. *Scientific Reports*, 13(1):4737, March 2023. ISSN 2045-2322. doi: 10.1038/s41598-023-32027-3. URL https://doi.org/10.1038/s41598-023-32027-3.

Martin J. Wainwright. *Basic tail and concentration bounds*, pp. 21–57. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2019.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. Bitnet: Scaling 1-bit transformers for large language models, 2023. URL https://arxiv.org/abs/2310.11453.

G. N. Watson. *A Treatise on the Theory of Bessel Functions*. Cambridge University Press, Cambridge, 2 edition, 1944. Reprinted with corrections, various later printings.

Sven A. Wegner. Gaussian random vectors in high dimensions. In *Mathematical Introduction to Data Science*, pp. 139–149. Springer, Berlin, Heidelberg, 2024. doi: 10.1007/978-3-662-69426-8_10. Chapter proving and discussing the Gaussian annulus theorem.

G. B. Whitham. *Linear and nonlinear waves*. Pure and applied mathematics. Wiley-Interscience, New York, 1999. ISBN 9786613306241.

Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, and Jürgen Schmidhuber. Natural evolution strategies, 2011. URL https://arxiv.org/abs/1106.4487.

Samuel Webb Williams. *Auto-tuning performance on multicore computers*. PhD thesis, USA, 2008. AAI3353349.

C.S. Withers. A simple expression for the multivariate hermite polynomials. *Statistics and Probability Letters*, 47(2):165–169, 2000. ISSN 0167-7152. doi: https://doi.org/10.1016/S0167-7152(99)00153-4. URL https://www.sciencedirect.com/science/article/pii/S0167715299001534.

Ke Xue, Chao Qian, Ling Xu, and Xudong Fei. Evolutionary gradient descent for non-convex optimization. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 3221–3227. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/443. URL https://doi.org/10.24963/ijcai.2021/443. Main Track.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Ziming Yu, Pan Zhou, Sike Wang, Jia Li, Mi Tian, and Hua Huang. Zeroth-order fine-tuning of llms in random subspaces. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4475–4485, October 2025.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

Yihua Zhang, Pingzhi Li, Junyuan Hong, Jiaxiang Li, Yimeng Zhang, Wenqing Zheng, Pin-Yu Chen, Jason D. Lee, Wotao Yin, Mingyi Hong, Zhangyang Wang, Sijia Liu, and Tianlong Chen. Revisiting zeroth-order optimization for memory-efficient llm fine-tuning: A benchmark, 2024.

Appendix

A Notation

In our proofs, we use the integral notation $\int$ to denote the integral over the corresponding $\mathbb{R}^d$ space, for example, for a matrix $E \in \mathbb{R}^{m \times n}$, $\int f(E)dE = \int_{\mathbb{R}^{m \times n}} f(E)dE$ and for a vector $E \in \mathbb{R}^{mn}$, $\int f(v)dv = \int_{\mathbb{R}^{mn}} f(v)dv$. For $f : \mathbb{R}^d \to \mathbb{R}$, we use $\nabla f(x)$ to denote the derivative of $f(\cdot)$ evaluated at x. For a vector $v \in \mathbb{R}^{mn}$, we define the mat operator as:

$$\text{mat}(v) = \begin{bmatrix} v_1 & v_{m+1} & \cdots & v_{(n-1)m+1} \\ v_2 & v_{m+2} & \cdots & v_{(n-1)m+2} \\ \vdots & \vdots & \ddots & \vdots \\ v_m & v_{2m} & \cdots & v_{mn} \end{bmatrix},$$

so $\text{mat}(\text{vec}(M)) = M$. We will use the fact that the Frobenius norm becomes the ℓ_2 norm in vector space:

$$\|M\|_F = \sqrt{\sum_{i,j} {m_{i,j}}^2} = \sqrt{\sum_k \text{vec}(M)_k{}^2} = \|\text{vec}(M)\|. \tag{13}$$

Our proofs make use of Fourier analysis. For a vector-valued function $f(v) : \mathbb{R}^d \to \mathbb{R}$, we define the Fourier transform as:

$$\tilde{f}(\omega) = \mathcal{F}[f](\omega) \coloneqq \int f(v) \exp(-i\omega^\top v)dv,$$

and the inverse Fourier transform as:

$$f(v) = \mathcal{F}^{-1}[\tilde{f}](v) \coloneqq \frac{1}{(2\pi)^d} \int \tilde{f}(\omega) \exp(i\omega^\top v)d\omega,$$

B ES Matrix Gradient Deviations

Let $\mu_M = \text{vec}(M) \in \mathbb{R}^{mn}$ be the vector of mean parameters associated with the matrix M. Let $v_M \in \mathbb{R}^{mn}$ denote the corresponding search vector associated with μ_M. As each element of v is generated independently from a standard normal $\mathcal{N}(0, 1)$, the search vector v_M is generated from the standard multivariate norm: $v_M \sim \mathcal{N}(0, I_{mn})$. From Eq. (2), the update for μ_M is:

$$\begin{aligned} \sigma \nabla_{\mu_M} J(\theta) &= \mathbb{E}_{v_M \sim \mathcal{N}(0, I_{mn})} \left[v_M \cdot f(W = \text{mat}(\mu_M) + \sigma \text{mat}(v_M)) \right], \\ &= \mathbb{E}_{v_M \sim \mathcal{N}(0, I_{mn})} \left[\text{vec}(\text{mat}(v_M)) \cdot f(W = \text{mat}(\mu_M) + \sigma \text{mat}(v_M)) \right], \\ &= \mathbb{E}_{E \sim \mathcal{N}(0, I_m, I_n)} \left[\text{vec}(E) \cdot f(W = M + \sigma E) \right], \end{aligned}$$

where $E = \text{mat}(v_M)$ and we have used the fact that sampling $v_M \sim \mathcal{N}(0, I_{mn})$ is equivalent to sampling $E \sim \mathcal{N}(0, I_m, I_n)$ and applying $v_M = \text{vec}(E)$. Now

$$\begin{aligned} \nabla_M J(\theta) &= \text{mat}(\nabla_{\mu_M} J(\theta)), \\ &= \frac{1}{\sigma} \mathbb{E}_{E \sim \mathcal{N}(0, I_m, I_n)} \left[\text{mat}(\text{vec}(E)) \cdot f(W = M + \sigma E) \right], \\ &= \frac{1}{\sigma} \mathbb{E}_{E \sim \mathcal{N}(0, I_m, I_n)} \left[E \cdot f(W = M + \sigma E) \right], \\ &= -\frac{1}{\sigma} \mathbb{E}_{E \sim \mathcal{N}(0, I_m, I_n)} \left[\nabla_E \log p(E) \cdot f(W = M + \sigma E) \right]. \end{aligned}$$

C High-Dimensional Analysis

C.1 High-Dimensional Gaussian ES and Convergence

We use insights from the Gaussian annulus theorem when investigating the convergence properties of high-dimensional ES: our proof relies on the fact that all probability mass converges to the interior of the ball $B_\epsilon(\mu) \coloneqq \{x' | \|x' - \mu\| < \epsilon\}$ where $\epsilon = \frac{\rho}{2}$ in the limit $d \to \infty$, where ρ is the radius of the local ball from Assumption 2, meaning we only need to consider the smooth region around μ in this limit. Our first result proves that the mass outside of the ball for any polynomially bounded function tends to zero at an exponential rate.

Lemma 1 (Polynomial Tail Bounds)**.** *Let $g(x)$ be polynomial bounded as:*

$$\|g(\mu + \sigma_d v)\| \leq C\|v\|^q(1 + \|\mu + \sigma_d v\|^p),$$

for some finite polynomial of orders p and q and constant $C > 0$. Let $A_d \coloneqq \{\|\sigma_d v\| \geq \epsilon\}$ denote the event that a mutation lies outside the a local ball of radius ϵ around μ. Assume $\sigma_d = o(d^{-1/2})$. Then for some constant $K > 0$:

$$\left\|\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[g(\mu + \sigma_d v)\mathbb{1}(A_d)\right]\right\| = \mathcal{O}\left(d^{\frac{q}{2}} \exp\left(-K\left(\frac{\epsilon}{\sigma_d}\right)^2\right)\right),$$

and in particular the right-hand side is $o(1)$ as $d \to \infty$.

Proof. We start by bounding the integrand using the polynomial bound. Denote $\mathbb{P}(A_d) \coloneqq \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}[\mathbb{1}(A_d)]$. Then, by Jensen's inequality in the first line, polynomial boundedness in the second and $\|a + b\|^p \leq 2^{p-1}(\|a\|^p + \|b\|^p)$ in the third:

$$\begin{aligned}
\left\|\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[g(\mu + \sigma_d v)\mathbb{1}(A_d)\right]\right\| &\leq \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|g(\mu + \sigma_d v)\|\mathbb{1}(A_d)\right], \\
&\leq C\,\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|^q(1 + \|\mu + \sigma_d v\|^p)\mathbb{1}(A_d)\right], \\
&\leq C\,\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|^q(1 + 2^{p-1}\|\mu\|^p)\mathbb{1}(A_d) + 2^{p-1}\sigma_d^p\|v\|^{p+q}\mathbb{1}(A_d)\right], \\
&= C'\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|^q\mathbb{1}(A_d)\right] + C''\sigma_d^p\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|^{p+q}\mathbb{1}(A_d)\right].
\end{aligned}$$

where $C' = C(1 + 2^{p-1}\|\mu\|^p)$ and $C'' = C2^{p-1}$ are constants independent of d. Applying the Cauchy–Schwarz inequality to the second expectation gives:

$$\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}[\|v\|^{p+q}\mathbb{1}(A_d)] \leq \sqrt{\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}[\|v\|^{2(p+q)}]} \cdot \sqrt{\mathbb{P}(A_d)}.$$

Now, the variable $\|v\|$ is χ_d-distributed. Using the formula for the i-th central moment of $\|v\|$ about the origin (Forbes et al., 2011, Chapter 11.3) yields:

$$\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|^i\right] = 2^{\frac{i}{2}}\frac{\Gamma\left(\frac{1}{2}(d+i)\right)}{\Gamma\left(\frac{1}{2}d\right)}.$$

Applying the identity $\frac{\Gamma(z+a)}{\Gamma(z+b)} \sim z^{a-b}$ (Askey & Roy, 2020-2026, Eq. 5.11.12):

$$\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|^i\right] \sim 2^{\frac{i}{2}}\left(\frac{d}{2}\right)^{\frac{i}{2}} = d^{\frac{i}{2}}, \tag{14}$$

where $\sim$ denotes asymptotic equivalence. For $i = 2(p + q)$, this yields the bound:

$$\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}[\|v\|^{2(p+q)}] = \mathcal{O}(d^{p+q}),$$

hence:

$$\begin{aligned}
\left\|\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[g(\mu + \sigma_d v)\mathbb{1}(A_d)\right]\right\| &\leq C'd^{\frac{q}{2}}\sqrt{\mathbb{P}(A_d)} + C''\sigma_d^p d^{\frac{p+q}{2}}\sqrt{\mathbb{P}(A_d)}, \\
&= (C' + C''\sigma_d^p d^{\frac{p}{2}})d^{\frac{q}{2}}\sqrt{\mathbb{P}(A_d)},
\end{aligned} \tag{15}$$

We use the Gaussian concentration inequality for the Euclidean norm (Vershynin, 2018, Theorem 3.1.1), which states that for $x \sim \mathcal{N}(0, I_d)$ there exists an absolute constant $K > 0$ such that for all $t \geq 0$,

$$\mathbb{P}\left(\left|\|x\| - \sqrt{d}\right| \geq t\right) \leq 2\exp(-Kt^2).$$

In our setting, we need to bound:

$$\mathbb{P}(A_d) = \mathbb{P}(\|\sigma_d v\| \geq \epsilon) = \mathbb{P}\left(\|v\| \geq \frac{\epsilon}{\sigma_d}\right) = \mathbb{P}\left(\|v\| - \sqrt{d} \geq \frac{\epsilon}{\sigma_d} - \sqrt{d}\right).$$

Setting $t = \frac{\epsilon}{\sigma_d} - \sqrt{d}$, the assumption $\sqrt{d}\sigma_d = o(1)$ implies for sufficiently large d that $\sqrt{d}\sigma_d \leq \epsilon$ and therefore $t \geq 0$, so we can apply the concentration bound to obtain:

$$\begin{aligned}\mathbb{P}(A_d) &= \mathbb{P}\left(\|v\| - \sqrt{d} \geq t\right) \leq \mathbb{P}\left(\left|\|v\| - \sqrt{d}\right| \geq t\right), \\ &= \mathcal{O}\left(\exp\left(-K\left(\frac{\epsilon}{\sigma_d} - \sqrt{d}\right)^2\right)\right) = \mathcal{O}\left(\exp\left(-K\left(\frac{\epsilon}{\sigma_d}\right)^2\left(1 - \frac{\sigma_d\sqrt{d}}{\epsilon}\right)^2\right)\right).\end{aligned} \tag{16}$$

Now, as $\sqrt{d}\sigma_d = o(1)$, it follows $\frac{\sigma_d\sqrt{d}}{\epsilon} = o(1)$, yielding:

$$\begin{aligned}\mathbb{P}(A_d) &= \mathcal{O}\left(\exp\left(-K\left(\frac{\epsilon}{\sigma_d}\right)^2\right)\right), \\ \implies \sqrt{\mathbb{P}(A_d)} &= \mathcal{O}\left(\exp\left(-\frac{K}{2}\left(\frac{\epsilon}{\sigma_d}\right)^2\right)\right), \\ \implies d^{\frac{q}{2}}\sqrt{\mathbb{P}(A_d)} &= \mathcal{O}\left(d^{\frac{q}{2}}\exp\left(-\frac{K}{2}\left(\frac{\epsilon}{\sigma_d}\right)^2\right)\right),\end{aligned}$$

Applying these results to Eq. (15) , along with $\sigma_d^p d^{\frac{p}{2}} = \mathcal{O}(d^{\frac{-p}{2}})d^{\frac{p}{2}} = \mathcal{O}(1)$, yields our desired result:

$$\begin{aligned}\left\|\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[g(\mu + \sigma_d v)\mathbb{1}(A_d)\right]\right\| \leq& C'\mathcal{O}\left(d^{\frac{q}{2}}\exp\left(-\frac{K}{2}\left(\frac{\epsilon}{\sigma_d}\right)^2\right)\right) \\ &+ C''\mathcal{O}(1)\mathcal{O}\left(d^{\frac{q}{2}}\exp\left(-\frac{K}{2}\left(\frac{\epsilon}{\sigma_d}\right)^2\right)\right), \\ =& \mathcal{O}\left(d^{\frac{q}{2}}\exp\left(-K\left(\frac{\epsilon}{\sigma_d}\right)^2\right)\right).\end{aligned}$$

where we have absorbed the factor of $\frac{1}{2}$ into the constant K. □

Our proof in Lemma 1 reveals the necessity of the condition $\sigma_d\sqrt{d} = o(1)$ for convergence as we can only apply the Gaussian concentration inequality in Eq. (16) for $\sigma_d\sqrt{d} = o(1)$; this is a direct consequence of the Gaussian annulus theorem, as for slower rates $1 = o(\sigma_d\sqrt{d})$, the Gaussian probability mass will exit any local ball around μ and flood the tail, meaning that the tail probability will grow with increasing d. Having bounded the tail, convergence to linearity follows by proving convergence within the ball, which allows us to exploit the local C^1 smoothness of $f(x)$:

Theorem 1 (Convergence to Linearity)**.** *Let Assumptions 2, 3 and 4 hold and $\sigma_d = o\left(d^{-\frac{1}{2}}\right)$. Then:*

$$\|\nabla_\mu J(\theta) - \nabla f(\mu)\| = \Theta\left(\left(\sigma_d\sqrt{d}\right)^\alpha\right) = o(1),$$

almost surely with respect to the distribution over μ.

Proof. We start with the definition of the ES update:

$$\nabla_\mu J(\theta) = \frac{1}{\sigma_d}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[v \cdot f(\mu + \sigma_d v)\right].$$

Now let $\epsilon = \frac{\rho}{2}$ where ρ is the radius of the ball from Assumption 2. Consider the hinge function:

$$\phi(x) = \begin{cases} 1, & \|x\| \le \epsilon, \\ 2 - \frac{\|x\|}{\epsilon}, & \epsilon < \|x\| < 2\epsilon, \\ 0, & \|x\| \ge 2\epsilon, \end{cases}$$

which interpolates between 1 and 0 in the region $\epsilon < \|x\| < 2\epsilon$. Our first goal is to use $\phi(x)$ to generate a function $\tilde{f}(x)$ that is absolutely continuous and has integrable derivatives outside of $B_\rho(\mu)$ to allow us to apply Stein's lemma (Stein, 1972). We define $\tilde{f}(x)$ as:

$$\tilde{f}(x) = f(x) \cdot \phi(x - \mu)$$

Consider the closed ball $B_\epsilon(\mu) \coloneqq \{x' | \|x' - \mu\| \le \epsilon\}$. We note that within the ball $f(\mu + \sigma_d v)$ remains unchanged:

$$\tilde{f}(\mu + \sigma_d v) = \begin{cases} f(\mu + \sigma_d v), & \|\sigma_d v\| \le \epsilon, \\ f(\mu + \sigma_d v) \cdot \left(2 - \frac{\|\sigma_d v\|}{\epsilon}\right), & \epsilon < \|\sigma_d v\| < 2\epsilon, \\ 0, & \|\sigma_d v\| \ge 2\epsilon. \end{cases} \tag{17}$$

The derivative of the function with respect to v is:

$$\nabla_v \tilde{f}(\mu + \sigma_d v) = \begin{cases} \sigma_d \nabla f(\mu + \sigma_d v), & \|\sigma_d v\| \le \epsilon, \\ \sigma_d \nabla f(\mu + \sigma_d v) \cdot \left(2 - \frac{\|\sigma_d v\|}{\epsilon}\right) - \frac{\sigma_d v}{\epsilon \|v\|} \cdot f(\mu + \sigma_d v), & \epsilon < \|\sigma_d v\| < 2\epsilon, \\ 0, & \|\sigma_d v\| \ge 2\epsilon. \end{cases} \tag{18}$$

where the gradient fails to exist only on the sets $\|\sigma_d v\| \in \{\epsilon, 2\epsilon\}$, which have Lebesgue measure zero. We start by using this function to decompose $J(\mu)$ into a smoothed part and a remainder:

$$\nabla_\mu J(\theta) = \underbrace{\frac{1}{\sigma_d}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[v \cdot \tilde{f}(\mu + \sigma_d v)\right]}_{\coloneqq \nabla_\mu \tilde{J}(\mu)} + \underbrace{\frac{1}{\sigma_d}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[v \cdot (f(\mu + \sigma_d v) - \tilde{f}(\mu + \sigma_d v))\right]}_{\coloneqq \Delta(\mu)},$$

Hence:

$$\|\nabla_\mu J(\theta) - \nabla f(\mu)\| \le \left\|\nabla_\mu \tilde{J}(\mu) - \nabla f(\mu)\right\| + \|\Delta(\mu)\|. \tag{19}$$

Consider the smoothed part:

$$\nabla_\mu \tilde{J}(\mu) \coloneqq \frac{1}{\sigma_d}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[v \cdot \tilde{f}(\mu + \sigma_d v)\right].$$

Our goal is to apply Stein's lemma (Stein, 1972) in its multivariate form (Liu, 1994, Lemma 1). The assumptions of (Liu, 1994, Lemma 1) require that the partial derivatives $\partial_{v_i} \tilde{f}(\mu + \sigma_d v)$ are absolutely continuous almost everywhere and:

$$\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[|\partial_{v_i} \tilde{f}(\mu + \sigma_d v)|\right] < \infty.$$

These two conditions are satisfied by construction. Indeed, under Assumption 2, $f(\cdot)$ is C^1 continuous on $B_\rho(\mu)$, hence from Eq. (17), $\tilde{f}(\cdot)$ coincides with a compactly supported, piecewise C^1 function whose gradient (Eq. (18)) exists almost everywhere. Moreover, under Assumption 3. both $f(\mu + \sigma_d v)$ and $\nabla f(\mu + \sigma_d v)$ are polynomially bounded, and since $\nabla \tilde{f}(\mu + \sigma_d v)$ is supported on $\|\sigma_d v\| \le 2\epsilon$, it follows that:

$$\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|\nabla \tilde{f}(\mu + \sigma_d v)\|\right] < \infty.$$

Applying (Liu, 1994, Lemma 1):

$$\begin{aligned}\frac{1}{\sigma_d}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[v\cdot f(\mu+\sigma_d v)\right] &= \frac{1}{\sigma_d}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\nabla_v \tilde{f}(\mu+\sigma_d v)\right], \\ &= \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\nabla \tilde{f}(\mu+\sigma_d v)\right], \\ \implies \|\nabla_\mu \tilde{J}(\mu) - \nabla f(\mu)\| &= \left\|\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\nabla \tilde{f}(\mu+\sigma_d v) - \nabla f(\mu)\right]\right\| \\ &\leq \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\left\|\nabla \tilde{f}(\mu+\sigma_d v) - \nabla f(\mu)\right\|\right].\end{aligned} \tag{20}$$

Let $\{\mu+\sigma_d v \in B_\epsilon(\mu)\} = \{\|\sigma_d v\| \leq \epsilon\}$ denote the event that a mutation lies within the ball $B_\epsilon(\mu)$. We now split the integral into two regions, the first within the ball and the second outside:

$$\begin{aligned}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|\nabla f(\mu+\sigma_d v) - \nabla f(\mu)\|\right] =& \underbrace{\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\left\|\nabla \tilde{f}(\mu+\sigma_d v) - \nabla f(\mu)\right\| \mathbb{1}(\|\sigma_d v\| \leq \epsilon)\right]}_{:=I_{\text{loc}}} \\ &+ \underbrace{\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\left\|\nabla \tilde{f}(\mu+\sigma_d v) - \nabla f(\mu)\right\| \mathbb{1}(\|\sigma_d v\| > \epsilon)\right]}_{:=I_{\text{tail}}}.\end{aligned}$$

Consider the region inside the ball, I_{loc}. From Eq. (18), $\nabla \tilde{f}(\mu+\sigma_d v) = \nabla f(\mu+\sigma_d v)$ within this region. Using the local α-Hölder continuity from Assumption 2:

$$\begin{aligned}I_{\text{loc}} =& \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|\nabla f(\mu+\sigma_d v) - \nabla f(\mu)\| \mathbb{1}(\|\sigma_d v\| \leq \epsilon)\right], \\ \leq& L\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|\sigma_d v\|^\alpha \mathbb{1}(\|\sigma_d v\| \leq \epsilon)\right], \\ \leq& {\sigma_d}^\alpha L\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|^\alpha\right].\end{aligned}$$

Now, applying the identity $\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|^i\right] \sim d^{\frac{i}{2}}$, from Eq. (14):

$$I_{\text{loc}} = \mathcal{O}\left(\left(\sigma_d\sqrt{d}\right)^\alpha\right).$$

We now bound the tail region outside the ball:

$$\begin{aligned}I_{\text{tail}} &= \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\left\|\nabla \tilde{f}(\mu+\sigma_d v) - \nabla f(\mu)\right\| \mathbb{1}(\|\sigma_d v\| > \epsilon)\right], \\ &\leq \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\left\|\nabla \tilde{f}(\mu+\sigma_d v) - \nabla f(\mu)\right\| \mathbb{1}(\|\sigma_d v\| \geq \epsilon)\right].\end{aligned}$$

Now, as $\|\nabla f(\mu)\| = \mathcal{O}(1)$ from Assumption 4 and we have established that $\|\nabla \tilde{f}(\mu+\sigma_d v)\|$ is polynomial bounded under Assumption 3 when applying Stein's lemma, it follows that $\left\|\nabla \tilde{f}(\mu+\sigma_d v) - \nabla f(\mu)\right\|$ is also polynomial bounded, that is there exists some constant $C > 0$ and finite polynomial order p such that:

$$\left\|\nabla \tilde{f}(\mu+\sigma_d v) - \nabla f(\mu)\right\| \leq C(1 + \|\mu+\sigma_d v\|^p).$$

Applying Lemma 1, it follows:

$$I_{\text{tail}} = \mathcal{O}\left(\exp\left(-K\left(\frac{\epsilon}{\sigma_d}\right)^2\right)\right),$$

for some constant $K > 0$. Together, this yields:

$$\begin{aligned}\|\nabla_\mu \tilde{J}(\mu) - \nabla f(\mu)\| &= I_{\text{loc}} + I_{\text{tail}}, \\ &= \mathcal{O}\left(\left(\sigma_d\sqrt{d}\right)^\alpha\right) + \mathcal{O}\left(\exp\left(-K\left(\frac{\epsilon}{\sigma_d}\right)^2\right)\right).\end{aligned}$$

As $\exp(-x) = o\left(x^{-a}\right)$ for any $a > 0$, we take $a = \alpha/2$ to obtain a weakened bound matching the first term:

$$\exp\left(-K\left(\frac{\epsilon}{\sigma_d}\right)^2\right) = o\left(\left(\frac{\sigma_d}{\epsilon}\right)^\alpha\right) = o\left(\left(\sigma_d\sqrt{d}\right)^\alpha\right).$$

This yields the upper bound:

$$\|\nabla_\mu \tilde{J}(\mu) - \nabla f(\mu)\| = \mathcal{O}\left(\left(\sigma_d\sqrt{d}\right)^\alpha\right). \tag{21}$$

Returning to Eq. (19), we must bound the remainder term:

$$\begin{aligned}\|\Delta(\mu)\| &= \left\|\frac{1}{\sigma_d}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[v\cdot\left(f(\mu+\sigma_d v) - \tilde{f}(\mu+\sigma_d v)\right)\right]\right\|, \\ &\leq \frac{1}{\sigma_d}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|\cdot\left|\left(f(\mu+\sigma_d v) - \tilde{f}(\mu+\sigma_d v)\right)\right|\right].\end{aligned}$$

Again, from Assumption 3, it follows that $\left|\left(f(\mu+\sigma_d v) - \tilde{f}(\mu+\sigma_d v)\right)\right|$ is polynomially bounded, that is there exists some constant $C' > 0$ and finite polynomial order p' such that:

$$\left|\left(f(\mu+\sigma_d v) - \tilde{f}(\mu+\sigma_d v)\right)\right| \leq C'(1+\|\mu+\sigma_d v\|^p).$$

Applying Lemma 1 with $q = 1$:

$$\|\Delta(\mu)\| = \mathcal{O}\left(\frac{d^{\frac{1}{2}}}{\sigma_d}\exp\left(-K\left(\frac{\epsilon}{\sigma_d}\right)^2\right)\right).$$

Now, as $\exp(-x) = o\left(x^{-1}\right)$ for $x\to\infty$, it follows:

$$\begin{aligned}\exp\left(-K\left(\frac{\epsilon}{\sigma_d}\right)^2\right) &= o\left(\sigma_d^2\right), \\ \implies \|\Delta(\mu)\| &= \mathcal{O}\left(\frac{d^{\frac{1}{2}}}{\sigma_d}\exp\left(-K\left(\frac{\epsilon}{\sigma_d}\right)^2\right)\right), \\ &= o(\sigma_d\sqrt{d}), \\ &= o\left((\sigma_d\sqrt{d})^\alpha\right),\end{aligned} \tag{22}$$

where the final line follows from the fact $\sqrt{d}\sigma_d = o(1)$. Assembling our bounds using Ineq. 19 yields our desired result:

$$\|\nabla_\mu J(\theta) - \nabla f(\mu)\| \leq \underbrace{\left\|\nabla_\mu \tilde{J}(\mu) - \nabla f(\mu)\right\|}_{=O\left((\sigma_d\sqrt{d})^\alpha\right),\text{ Eq. 21}} + \underbrace{\|\Delta(\mu)\|}_{=o\left((\sigma_d\sqrt{d})^\alpha\right),\text{ Eq. 22}} = O\left((\sigma_d\sqrt{d})^\alpha\right).$$

We now show that the bound is tight. Consider the function $f(x) = \frac{L}{2}\sum_{i=1}^d x_i|x_i| + a^\top x$ where $\|a\| = \mathcal{O}(1)$. Taking partial derivatives:

$$\partial_i f(x) = L|x_i| + a_i, \tag{23}$$

hence:

$$\|\nabla f(x) - \nabla f(y)\| = L\sqrt{\sum_{i=1}^d (|x_i| - |y_i|)^2}$$

Applying the reverse triangle inequality $||x_i| - |y_i|| \leq |x_i - y_i| \implies (|x_i| - |y_i|)^2 \leq (x_i - y_i)^2$:

$$\|\nabla f(x) - \nabla f(y)\| \leq L\sqrt{\sum_{i=1}^d (x_i - y_i)^2} = L\|x - y\|.$$

We have thus shown that $f(x)$ is C^1-continuous and its gradient has Lipschitz constant L, i.e. $\alpha = 1$ with Hölder constant L. It is also bounded by a polynomial of order 2. Without loss of generality, we take a deterministic initialisation $\mu = 0$ to simplify algebra, yielding;

$$\nabla f(\mu) = a \implies \|\nabla f(\mu)\| = \|a\| = \mathcal{O}(1).$$

$f(x)$ thus satisfies Assumptions 2, 3 and 4. Using $f(x)$ as the fitness:

$$\begin{aligned}\nabla_\mu J(\theta) - \underbrace{\nabla f(\mu)}_{=a} &= \frac{1}{\sigma_d}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[v \cdot f(\sigma_d v)\right] - a, \\ &= \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\nabla f(\sigma_d v)\right] - a;\end{aligned}$$

Taking expectations element-wise and using Eq. (23):

$$\begin{aligned}[\nabla_\mu J(\theta) - \nabla f(\mu)]_i &= \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\partial_i f(\sigma_d v)\right] - a_i, \\ &= \sigma_d L \mathbb{E}_{v_i\sim\mathcal{N}(0,1)}[|v_i|].\end{aligned}$$

Applying Eq. (14):

$$\mathbb{E}_{v_i\sim\mathcal{N}(0,1)}[|v_i|] = \sqrt{2}\frac{\Gamma(1)}{\Gamma(\frac{1}{2})} = \sqrt{\frac{2}{\pi}},$$

Hence:

$$\|\nabla_\mu J(\theta) - \nabla f(\mu)\| = \sigma_d\sqrt{d}\cdot L\sqrt{\frac{2}{\pi}},$$

thereby attaining the upper bound rate of $\sigma_d\sqrt{d}$. □

C.2 Critical Convergence Rate

To show that our rate is critical, we investigate the space of functions that can be represented by cubic polynomials of the form:

$$f(x) = a^\top x + \frac{1}{2}x^\top B x + \frac{1}{6}C[x,x,x], \tag{24}$$

where $a \in \mathbb{R}^d$, $B \in \mathbb{R}^{d\times d}$ is a symmetric matrix and $C[x,x,x] = \sum_{i,j,k} c_{i,j,k}x_i x_j x_k$ denotes a symmetric 3-linear map represented by the 3-tensor $C \in \mathbb{R}^{d\times d\times d}$.

Since our theory depends on analysing the local stability of a smooth ball for a fitness function, stability over this class is necessary for convergence on more general objectives. We show that once σ_d decays slower than the critical rate, divergence already occurs within this subclass, establishing the sharpness of the rate.

Theorem 2 (Exact divergence for cubic objectives)**.** *Let $f(x)$ denote the cubic polynomial in Eq. (24). Assume $\|a\| = \mathcal{O}(1), \|B\| = \mathcal{O}(1)$, $\|C\| = \mathcal{O}(1)$ where $\|\cdot\|$ denotes operator norm for i-tensor $T(x_1, \dots x_i)$: $\|T\| \coloneqq \sup_{\|x_1\|=\cdots=\|x_i\|=1}|T(x_1,\dots x_i)|$. Let Assumption 4 hold, then:*

$$\nabla_\mu J(\theta) = \nabla f(\mu) + \frac{\sigma_d^2}{2}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v,v,\cdot)\right].$$

Moreover:

$$\begin{aligned}\left\|\frac{\sigma_d^2}{2}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v,v,\cdot)\right]\right\| &= \Theta(\sigma_d^2 d), \\ \|\nabla_\mu J(\theta) - \nabla f(\mu)\| &= \Theta(\sigma_d^2 d).\end{aligned}$$

Proof. We start by taking derivatives of $f(x)$:

$$\nabla f(x) = a + Bx + \frac{1}{2}C(x, x, \cdot).$$

Substituting this into the definition of $\nabla_\mu J(\theta)$ and using Eq. (20):

$$\begin{aligned}
\nabla_\mu J(\theta) &= \frac{1}{\sigma_d}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[v f(\mu + \sigma_d v)\right], \\
&= \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\nabla f(\mu + \sigma_d v)\right], \\
&= \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[a + B(\mu + \sigma_d v) + \frac{1}{2}C(\mu + \sigma_d v, \mu + \sigma_d v, \cdot)\right], \\
&= a + B\mu + \sigma_d B\underbrace{\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}[v]}_{=0} + \frac{1}{2}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(\mu + \sigma_d v, \mu + \sigma_d v, \cdot)\right], \\
&= a + B\mu + \frac{1}{2}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(\mu, \mu, \cdot) + \sigma_d C(v, \mu, \cdot) + \sigma_d C(\mu, v, \cdot) + \sigma_d^2 C(v, v, \cdot)\right], \\
&= \underbrace{a + B\mu + \frac{1}{2}C(\mu, \mu, \cdot)}_{=\nabla f(\mu)} + \frac{1}{2}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[2\sigma_d C(v, \mu, \cdot) + \sigma_d^2 C(v, v, \cdot)\right], \\
&= \nabla f(\mu) + \sigma_d \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v, \mu, \cdot)\right] + \frac{\sigma_d^2}{2}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v, v, \cdot)\right],
\end{aligned}$$

where we have used the fact $C(v, \mu, \cdot) = C(\mu, v, \cdot)$ by definition of the symmetry of C. As $C(v, \mu, \cdot)$ is linear in v, its expectation under zero-mean $\mathcal{N}(0, I_d)$ is zero, hence:

$$\nabla_\mu J(\theta) = \nabla f(\mu) + \frac{\sigma_d^2}{2}\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v, v, \cdot)\right],$$

proving our first result. Now, it follows that $\|C(v, v, \cdot)\| \leq \|C\|\|v\|^2$ and as $\|C\| = \mathcal{O}(1)$:

$$\begin{aligned}
\|\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v, v, \cdot)\right]\| \leq& \|C\|\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|^2\right], \\
=& \mathcal{O}(\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|^2\right])
\end{aligned}$$

Now as v is unit Gaussian: $\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|^2\right] = d$, hence:

$$\|\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v, v, \cdot)\right]\| = \mathcal{O}(d).$$

We now show that the bound is tight. Consider the function $f(x) = u^\top x\|x\|^2$ for $u^\top = \frac{1}{\sqrt{d}}[1, \dots 1]$. The factor of $\frac{1}{\sqrt{d}}$ ensures that the gradient of the function $\nabla_x f(x) = \mathcal{O}(1)$. We can write $\|x\|^2$ as the tensor contraction:

$$\|x\|^2 = I_d(x, x),$$

where I_d is the identity matrix and:

$$u^\top(x) = u(x),$$

hence we write $f(x)$ as a tensor contraction as:

$$f(x) = C(x, x, x),$$

where $C \coloneqq \mathrm{Sym}(u \otimes I_d)$. Using this function:

$$\begin{aligned}
C(v, v, \cdot) &= \nabla_v(u^\top v\|v\|^2) \\
&= u\|v\|^2 + 2vu^\top v, \\
\implies \mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v, v, \cdot)\right] &= u\underbrace{\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[\|v\|^2\right]}_{=d} + 2\underbrace{\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[vv^\top\right]}_{=I_d}u, \\
&= u(d + 2),
\end{aligned}$$

hence $\|\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v,v,\cdot)\right]\| = d+2$, achieving the upper bound rate of $\mathcal{O}(d)$ which implies:

$$\|\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v,v,\cdot)\right]\| = \Theta(d).$$

Our final result follows immediately:

$$\|\nabla_\mu J(\theta) - \nabla f(\mu)\| = \frac{(\sigma_d)^2}{2}\|\mathbb{E}_{v\sim\mathcal{N}(0,I_d)}\left[C(v,v,\cdot)\right]\| = \Theta(\sigma_d^2 d). \quad \square$$

C.3 EGGROLL Linearisation

We now study the effect of EGGROLL in high dimensions. We introduce the notation $v = \text{vec}(E)$ to denote the vectorisation of the low-rank matrix perturbation $E = \frac{1}{\sqrt{r}}AB^\top$ and work in vector space. The EGGROLL vector update v can thus be written as sum of independent variables:

$$v = \sum_{i=1}^{r} \frac{1}{\sqrt{r}} u_i$$

with:

$$u_i = \text{vec}\left(a_i b_i^\top\right),$$

where recall a_i and b_i are the ith column vectors of A and B. We write $\mu = \text{vec}(M)$. Using Eq. (13), we can convert between results in vector space and matrix space as:

$$\begin{aligned}\|\text{vec}(\hat{g}_{\text{LR}}) - \nabla f(\mu)\| &= \|\hat{g}_{\text{LR}} - \nabla_W f(W=M)\|_F, \\ \|\text{vec}(\hat{g}_{\text{LR}}) - \nabla_\mu J(\theta)\| &= \|\hat{g}_{\text{LR}} - \nabla_M J(\theta)\|_F.\end{aligned}$$

To extend our analysis, we need to ensure that all polynomial moments of $P(v)$ are finite and grow at most polynomially in the dimension $d = mn$. In particular, such tail bounds are sufficient to dominate polynomial error terms in our analysis. To introduce sub-Gaussian variables, we follow the exposition of Vershynin (2018) and results therein. A random variable $x_i \in \mathbb{R}$ is sub-Gaussian if there exists some finite constant $C > 0$ such that for all $t > 0$:

$$\mathbb{P}(|x_i| > t) \leq 2\exp(-Ct^2),$$

meaning their their tails decay like Gaussians. This is equivalent to any of the following three properties holding (Vershynin, 2018, 2.6.1): There exist constants $C_1, C_2, C_3 > 0$ that differ at most by an absolute constant factor such that:

$$\begin{aligned}(\mathbb{E}[|x_i|^p])^{\frac{1}{p}} &\leq C_1\sqrt{p}, \quad \forall p \geq 1, \\ \mathbb{E}\left[\exp\left(\frac{x_i^2}{C_2^2}\right)\right] &\leq 2,\end{aligned}$$

and if $\mathbb{E}[x_i] = 0$:

$$\mathbb{E}\left[\exp(\lambda x_i)\right] \leq \exp(C_3^2\lambda^2), \quad \forall \lambda \in \mathbb{R}.$$

A random vector $x \in \mathbb{R}^d$ is sub-Gaussian if all one-dimensional marginals of x are sub-Gaussian, i.e. $x^\top u$ is sub-Gaussian for all $u \in \mathbb{R}^d$. The sub-Gaussian norm is defined as:

$$\|x\|_{\psi_2} := \inf_K \left\{ K \middle| \mathbb{E}\left[\exp\left(u^\top(x - \mathbb{E}[x])\right)\right] \leq \exp\left(\frac{K^2\|u\|^2}{2}\right), \quad \forall u \in \mathbb{R}^d \right\}.$$

which returns the smallest universal sub-Gaussian constant for all marginals.

A key property of sub-Gaussian vectors that we use in our proofs is the sub-Gaussian concentration inequality for the Euclidean norm (Vershynin, 2018, Theorem 3.1.1), which states that for if x is a sub-Gaussian vector with $\mathbb{E}[x_i^2] = 1$ and $K = \|x\|_{\psi_2}$, there exists an absolute constant $C > 0$ such that for all $t \geq 0$,

$$\mathbb{P}\left(\left|\|x\| - \sqrt{m}\right| \geq t\right) \leq 2\exp\left(-\frac{Ct^2}{K^4}\right). \tag{25}$$

We also use a weaker form of control, that replaces the Gaussian-like tail decay with an exponential decay, but all other properties are defined similarly. In this paper, we use the definition that a variable x is known as sub-exponential if there exists a $K > 0$ such that for all $t \geq 0$:

$$\mathbb{P}\left(|x| \geq t\right) \leq 2\exp\left(-\frac{t}{K}\right).$$

Our first result derives a bound on the expected value of the norms $\|a\|^i$ and $\|b\|^i$:

Lemma 2. *Let Assumption 6 hold. Let $P(a)$ denote the distribution over columns of A and $P(b)$ denote the distribution over columns of B. Then:*

$$\mathbb{E}_{a\sim P(a)}[\|a\|^i] = \mathcal{O}(m^{\frac{i}{2}}), \quad \mathbb{E}_{b\sim P(b)}[\|b\|^i] = \mathcal{O}(n^{\frac{i}{2}}).$$

Proof. It suffices to prove $\mathbb{E}_{a\sim P(a)}[\|a\|^i] = \mathcal{O}(m^{\frac{i}{2}})$ as $\mathbb{E}_{b\sim P(b)}[\|b\|^i] = \mathcal{O}(n^{\frac{i}{2}})$ follows automatically from the same assumptions. We start by using the 'layer cake' representation of the expectation Lieb & Loss (2010 - 2010, Theorem 1.13):

$$\mathbb{E}_{a\sim P(a)}[\|a\|^i] = i\int_0^\infty t^{i-1}\mathbb{P}(\|a\| > t)dt.$$

Let $t_m = C\sqrt{m}$ for any $C > 1$. We split the integral into two regions:

$$i\int_0^\infty t^{i-1}\mathbb{P}(\|a\| > t)dt = \int_0^{t_m} it^{i-1}\mathbb{P}(\|a\| > t)dt + \int_{t_m}^\infty it^{i-1}\mathbb{P}(\|a\| > t)dt.$$

For the first integral:

$$\begin{aligned}\int_0^{t_m} it^{i-1}\mathbb{P}(\|a\| > t)dt &\leq \int_0^{t_m} it^{i-1}dt,\\ &= (t_m)^i,\\ &= C^i m^{\frac{i}{2}}.\end{aligned}$$

For the second integral, we wish to bound $\mathbb{P}(\|a\| > t)$ for the region $t \geq t_m = C\sqrt{m}$. Setting $t' = t - \sqrt{m} > 0$, the assumption $C > 1$ implies $t' \geq 0$ in this region, hence

$$\mathbb{P}(\|a\| > t) = \mathbb{P}(\|a\| - \sqrt{m} > t') \leq \mathbb{P}(|\|a\| - \sqrt{m}| > t').$$

We bound this using the sub-Gaussian concentration inequality from Eq. (25). Under Assumption 6, a is a sub-Gaussian vector with $\|x\|_{\psi_2} \leq \infty$, hence there exists an absolute constant $C' > 0$ such that for all $t' \geq 0$,

$$\mathbb{P}\left(\left|\|a\| - \sqrt{m}\right| \geq t'\right) \leq 2\exp\left(-C't'^2\right).$$

This implies:

$$\mathbb{P}\left(\|a\| \geq t\right) \leq 2\exp\left(-C'(t - \sqrt{m})^2\right),$$

for all $t \geq t_m$. Substituting yields:

$$\int_{t_m}^\infty it^{i-1}\mathbb{P}(\|a\| > t)dt \leq \int_{t_m}^\infty it^{i-1}\exp\left(-C'(t - \sqrt{m})^2\right)dt.$$

Let $x = t - \sqrt{m} \implies dt = dx$:

$$\int_{t_m}^{\infty} it^{i-1}\mathbb{P}(\|a\| > t)dt \leq \int_{\sqrt{m}(C-1)}^{\infty} i(x+\sqrt{m})^{i-1}\exp\left(-C'x^2\right)dx.$$

Now, $\sqrt{m} \leq \frac{x}{C-1}$ for all $x \geq \sqrt{m}(C-1)$, hence:

$$\begin{aligned}
\int_{t_m}^{\infty} it^{i-1}\mathbb{P}(\|a\| > t)dt &\leq \int_{\sqrt{m}(C-1)}^{\infty} ix^{i-1}\left(1+\frac{1}{C-1}\right)^{i-1}\exp\left(-C'x^2\right)dx, \\
&= i\left(1+\frac{1}{C-1}\right)^{i-1}\int_{\sqrt{m}(C-1)}^{\infty} x^{i-1}\exp\left(-C'x^2\right)dx, \\
&\leq i\left(1+\frac{1}{C-1}\right)^{i-1}\int_{0}^{\infty} x^{i-1}\exp\left(-C'x^2\right)dx, \\
&\leq i\left(1+\frac{1}{C-1}\right)^{i-1}\frac{1}{2}(C')^{-i/2}\Gamma\left(\frac{i}{2}\right), \\
&= \mathcal{O}(1).
\end{aligned}$$

Combining the two bounds yields:

$$\mathbb{E}_{a\sim P(a)}[\|a\|^i] = \mathcal{O}(m^{\frac{i}{2}}),$$

as required. □

Using this result, we now bound the whole vector $v = \frac{1}{\sqrt{r}}\sum_{i=1}^{r}\text{vec}(a_i b_i^\top)$

Lemma 3. *Let $i \geq 1$. Under Assumption 6:*

$$\mathbb{E}_{v\sim P(v)}\left[\|v\|^i\right] = \mathcal{O}\left((rmn)^{\frac{i}{2}}\right)$$

Proof. For any vectors a, b:

$$\begin{aligned}
\|\text{vec}(ab^\top)\| &= \sqrt{\sum_{j=1}^{m}\sum_{k=1}^{n}(a_j b_k)^2} = \sqrt{\sum_{j=1}^{m} {a_j}^2 \sum_{k=1}^{n} {b_k}^2} = \|a\|\|b\|, \\
\implies \|\text{vec}(ab^\top)\|^i &= \|a\|^i\|b\|^i.
\end{aligned}$$

Applying Lemma 2 under Assumption 6 for each summand of $v = \frac{1}{\sqrt{r}}\sum_{l=1}^{r}\text{vec}(a_l b_l^\top)$:

$$\begin{aligned}
\mathbb{E}_{v\sim P(v)}\left[\|\text{vec}(a_l b_l^\top)\|^i\right] &= \mathbb{E}_{a_l\sim P(a_l)}\left[\|a_l\|^i\right]\mathbb{E}_{b_l\sim p(b_l)}\left[\|b_l\|^i\right], \\
&= \mathcal{O}\left((mn)^{\frac{i}{2}}\right).
\end{aligned}$$

Applying the triangle inequality:

$$\begin{aligned}
\mathbb{E}_{v\sim P(v)}\left[\|v\|^i\right] &= \mathbb{E}_{v\sim P(v)}\left[\left\|\frac{1}{\sqrt{r}}\sum_{l=1}^{r}\text{vec}(a_l b_l^\top)\right\|^i\right], \\
&\leq \mathbb{E}_{v\sim P(v)}\left[\left(\frac{1}{\sqrt{r}}\sum_{l=1}^{r}\left\|\text{vec}(a_l b_l^\top)\right\|\right)^i\right], \\
&= r^{\frac{i}{2}}\mathbb{E}_{v\sim P(v)}\left[\left(\frac{1}{r}\sum_{l=1}^{r}\left\|\text{vec}(a_l b_l^\top)\right\|\right)^i\right].
\end{aligned}$$

Now, as $i \geq 1$, we can apply Jensen's inequality:

$$\left(\frac{1}{r}\sum_{l=1}^{r}\left\|\text{vec}(a_l b_l^\top)\right\|\right)^i \leq \frac{1}{r}\sum_{l=1}^{r}\|\text{vec}(a_l b_l^\top)\|^i,$$

yielding:

$$\mathbb{E}_{v\sim P(v)}\left[\|v\|^i\right] \leq r^{\left(\frac{i}{2}-1\right)}\sum_{l=1}^{r}\mathbb{E}_{v\sim P(v)}\left[\|\text{vec}(a_l b_l^\top)\|^i\right] = r^{\frac{i}{2}}\mathcal{O}\left((mn)^{\frac{i}{2}}\right) = \mathcal{O}\left((rmn)^{\frac{i}{2}}\right). \qquad \square$$

Our proof borrows techniques used to prove linearisation of the ES update in Section C.1 by bounding the tail probability of any polynomial under the low-rank distribution outside of the ball $B_\rho(\mu)$. To apply the concentration inequality that would generalise Lemma 1, we show that v has an exponentially decaying tail:

Lemma 4 (Exponential Tail Bound)**.** *Let $r < \infty$ and Assumption 6 hold. Then all elements of v are sub-exponential and for $\sqrt{d}\sigma_d = o(1)$ there exists some constant $C > 0$ such that:*

$$\mathbb{P}(\|\sigma_d v\| \geq \rho) \leq 2d\exp\left(-C\frac{\rho}{\sqrt{d}\sigma_d}\right).$$

Proof. In matrix form:

$$E = \frac{1}{\sqrt{r}}\sum_{i=1}^{r} a_i b_i^\top.$$

The elements of E are thus:

$$E_{j,k} = \frac{1}{\sqrt{r}}\sum_{i=1}^{r} a_{ij} b_{ik}.$$

As a_{ij} and b_{ik} are independent sub-Gaussian random variables with zero mean, it follows from Vershynin (2018, Lemma 2.8.6) that their product $a_{ij}b_{ik}$ is a zero-mean sub-exponential variable with a uniform norm $\|a_{ij}b_{ik}\|_{\psi_1} < \infty$. Finally, a finite sum of sub-exponential variables is sub-exponential (Wainwright, 2019, Eq. (2.18)) with a uniform norm, so all elements of E and hence $v = \text{vec}(E)$ are sub-exponential and zero-mean with a uniform ψ_1-norm $K < \infty$.

We now bound $\mathbb{P}(\|\sigma_d v\| \geq \rho) = \mathbb{P}(\|v\| \geq \frac{\rho}{\sigma_d})$. For the vector v, it follows for $t \geq 0$:

$$\|v\| \geq t \implies \max_j |v_j| \geq \frac{t}{\sqrt{d}}.$$

This is easily proven via the contrapositive: if $\max_j |v_j| < \frac{t}{\sqrt{d}}$ then

$$\|v\|^2 = \sum_{j=1}^{d} v_j^2 < d\frac{t^2}{d} = t^2,$$

implying $\|v\| < t$. This means for $t \geq 0$:

$$\begin{aligned}\mathbb{P}(\|v\| \geq t) &\leq \mathbb{P}\left(\max_j |v_j| \geq \frac{t}{\sqrt{d}}\right), \\ &\leq \sum_{j=1}^{d}\mathbb{P}\left(|v_j| \geq \frac{t}{\sqrt{d}}\right). \end{aligned} \tag{26}$$

As v_j is a sub-exponential variable with finite uniform sub-exponential norm, by definition (Vershynin, 2018, Proposition 2.8.1) there exists a finite K such that for all j:

$$\mathbb{P}\left(|v_j| \geq \frac{t}{\sqrt{d}}\right) \leq 2\exp\left(-\frac{t}{\sqrt{d}K}\right).$$

Applying to Eq. (26) yields:

$$\mathbb{P}(\|v\| \geq t) \leq 2d\exp\left(-\frac{t}{\sqrt{d}K}\right).$$

Now, using $t = \frac{\rho}{\sigma_d}$ and $C = \frac{1}{K}$ yields:

$$\mathbb{P}(\|\sigma_d v\| \geq \rho) \leq 2d\exp\left(-C\frac{\rho}{\sqrt{d}\sigma_d}\right). \qquad \square$$

We now use these results to assemble into our key polynomial tail bound:

Lemma 5 (EGGROLL Polynomial Tail Bounds)**.** *Let Assumption 6 hold. Let $g(x)$ be polynomial bounded as:*

$$\|g(x)\| \leq C(1 + \|x\|^p),$$

for some finite polynomial of order p and constant $C > 0$. Consider the ball $B_\rho(\mu) \coloneqq \{x' | \|x' - \mu\| < \rho\}$. Let $\{\mu + \sigma_d v \in B_\rho(\mu)\} = \{\|\sigma_d v\| < \rho\}$ denote the event that a mutation lies outside the ball. Assume $\sigma_d = o(d^{-1/2})$. Then for some constant $K > 0$ independent of d:

$$\left\|\mathbb{E}_{v\sim P(v)}\left[g(\mu + \sigma_d v)\mathbb{1}(A_d)\right]\right\| = \mathcal{O}\left(\sqrt{d}\exp\left(-K\frac{\rho}{\sqrt{d}\sigma_d}\right)\right),$$

and in particular the right-hand side is $o(1)$ as $d \to \infty$.

Proof. Let

$$A_d \coloneqq \{\mu + \sigma_d v \in B_\rho(\mu)\}$$

and denote $\mathbb{P}(A_d) \coloneqq \mathbb{E}_{v\sim P(v)}[\mathbb{1}(A_d)]$. Our proof proceeds as in Lemma 1 to obtain:

$$\left\|\mathbb{E}_{v\sim P(v)}\left[g(\mu + \sigma_d v)\mathbb{1}(A_d)\right]\right\| \leq C'\mathbb{P}(A_d) + C''\sigma_d^p\mathbb{E}_{v\sim P(v)}\left[\|v\|^p\mathbb{1}(A_d)\right].$$

where $C' = C(1 + 2^{p-1}\|\mu\|^p)$ and $C'' = C2^{p-1}$ are constant in d. Applying the Cauchy–Schwarz inequality to the second expectation gives:

$$\mathbb{E}_{v\sim P(v)}[\|v\|^p\mathbb{1}(A_d)] \leq \sqrt{\mathbb{E}_{v\sim P(v)}[\|v\|^{2p}]} \cdot \sqrt{\mathbb{P}(A_d)}.$$

Applying Lemma 3 with fixed r and $d = mn$:

$$\sqrt{\mathbb{E}_{v\sim P(v)}[\|v\|^{2p}]} = \mathcal{O}\left(d^{\frac{p}{2}}\right).$$

Now, $\mathbb{P}(A_d) = \mathbb{P}(\|\sigma_d v\| \geq \rho)$. From Lemma 4, there exists some $K > 0$ such that:

$$\mathbb{P}(\|\sigma_d v\| \geq \rho) \leq 2d\exp\left(-K\frac{\rho}{\sqrt{d}\sigma_d}\right),$$
$$\implies \sqrt{\mathbb{P}(A_d)} = \mathcal{O}\left(\sqrt{d}\exp\left(-K\frac{\rho}{\sqrt{d}\sigma_d}\right)\right),$$

where we have absorbed the factor of $\frac{1}{2}$ into K, hence:

$$\mathbb{E}_{v\sim P(v)}[\|v\|^p\mathbb{1}(A_d)] = \mathcal{O}\left(d^{\frac{p+1}{2}}\exp\left(-K\frac{\rho}{\sqrt{d}\sigma_d}\right)\right).$$

Now, as $\sqrt{d}\sigma_d = o(1)$, $\sigma_d^p d^{\frac{p}{2}} = o(1)$, hence:

$$\sigma_d^p \mathbb{E}_{v\sim P(v)}\left[\|v\|^p \mathbb{1}(A_d)\right] = \mathcal{O}\left(\sqrt{d}\exp\left(-K\frac{\rho}{\sqrt{d}\sigma_d}\right)\right).$$

Applying our bounds yields our desired result:

$$\left\|\mathbb{E}_{v\sim P(v)}\left[g(\mu+\sigma_d v)\mathbb{1}(A_d)\right]\right\| = \mathcal{O}\left(\sqrt{d}\exp\left(-K\frac{\rho}{\sqrt{d}\sigma_d}\right)\right) = o(1).$$

where the $o(1)$ bound follows from the fact that the exponential factor dominates $\sqrt{d}$ and $\sqrt{d}\sigma_d = o(1)$. □

Theorem 3 (EGGROLL Convergence to Linearity)**.** *Let Assumptions 3, 4, 5 and 6 hold and $\sigma_d = o(d^{-1/2})$ and $L_d(\sigma_d d)^2 = o(1)$. Then there exists some $K > 0$ such that:*

$$\|\text{vec}(\hat{g}_{LR}) - \nabla f(\mu)\| = \mathcal{O}\left(L_d(\sigma_d d)^2\right) + \mathcal{O}\left(\frac{\sqrt{d}}{\sigma_d^2}\exp\left(-K\frac{\rho}{\sqrt{d}\sigma_d}\right)\right) = o(1),$$
$$\|\text{vec}(\hat{g}_{LR}) - \nabla_\mu J(\theta)\| = \mathcal{O}\left(\sigma_d\sqrt{d}\cdot\left(1 + L_d\sigma_d d^{\frac{3}{2}}\right)\right) = o(1).$$

almost surely with respect to the distribution over μ.

Proof. We start with the definition of the vectorised EGGROLL update:

$$\begin{aligned}
\text{vec}(\hat{g}_{\text{LR}}) - \nabla f(\mu) &= \frac{1}{\sigma_d}\mathbb{E}_{v\sim P(v)}\left[v\cdot f(\mu+\sigma_d v)\right] - \nabla f(\mu),\\
&= \frac{1}{\sigma_d}\mathbb{E}_{v\sim P(v)}\left[v\cdot f(\mu+\sigma_d v)\right] - \frac{1}{\sigma_d}\underbrace{\mathbb{E}_{v\sim P(v)}[v]}_{=0}\cdot f(\mu) - \underbrace{\mathbb{E}_{v\sim P(v)}[vv^\top]}_{I_d}\nabla f(\mu)\\
&\quad + \frac{1}{2\sigma_d}\underbrace{\mathbb{E}_{v\sim P(v)}[{\sigma_d}^2 vv^\top\nabla^2 f(\mu)v]}_{=0},\\
&= \frac{1}{\sigma_d}\mathbb{E}_{v\sim P(v)}\left[v\cdot\left(\underbrace{f(\mu+\sigma_d v) - f(\mu) - \sigma_d v^\top\nabla f(\mu) + \frac{{\sigma_d}^2}{2}v^\top\nabla^2 f(\mu)v}_{:=T_d(v)}\right)\right],\\
&= \frac{1}{\sigma_d}\mathbb{E}_{v\sim P(v)}\left[v\cdot T_d(v)\right],
\end{aligned}$$

where we have used the fact that the expectation of an odd function under a symmetric, zero mean distribution is always zero, and $P(v)$ satisfies this under Assumption 6, hence $\mathbb{E}_{v\sim P(v)}[vv^\top\nabla^2 f(\mu)v] = 0$, and $\mathbb{E}_{v\sim P(v)}[vv^\top] = I_d$ from Lemma 6. Consider the ball $B_\rho(\mu) \coloneqq \{x' | \|x'-\mu\| < \rho\}$. We now split the integral into two regions, the first within the ball and the second outside:

$$\begin{aligned}
\frac{1}{\sigma_d}\mathbb{E}_{v\sim P(v)}\left[v\cdot(f(\mu+\sigma_d v) - f(\mu))\right] &= \underbrace{\frac{1}{\sigma_d}\mathbb{E}_{v\sim P(v)}\left[v\cdot T_d(v)\mathbb{1}(\|\sigma_d v\| < \rho)\right]}_{:=I_{\text{loc}}}\\
&\quad + \underbrace{\frac{1}{\sigma_d}\mathbb{E}_{v\sim P(v)}\left[v\cdot T_d(v)\mathbb{1}(\|\sigma_d v\| \geq \rho)\right]}_{:=I_{\text{tail}}}.
\end{aligned}$$

Consider the region inside the ball:

$$\begin{aligned}
\|I_{\text{loc}}\| &= \frac{1}{\sigma_d}\left\|\mathbb{E}_{v\sim P(v)}\left[v\cdot T_d(v)\mathbb{1}(\|\sigma_d v\| < \rho)\right]\right\|,\\
&\leq \frac{1}{\sigma_d}\mathbb{E}_{v\sim P(v)}\left[\|v\|\,|T_d(v)|\,\mathbb{1}(\|\sigma_d v\| < \rho)\right].
\end{aligned} \tag{27}$$

Within this region, $f(\mu + \sigma_d v)$ is C^2 continuous under Assumption 5. We can thus write $f(\mu + \sigma_d v)$ using a first-order Taylor expansion about μ with a Hessian (second order derivative) remainder within the ball:

$$\begin{aligned} f(\mu + \sigma_d v) &= f(\mu) + \sigma_d \nabla f(\mu)^\top v + {\sigma_d}^2 v^\top \left(\int_0^1 (t-1) \nabla^2 f(\mu + t\sigma_d v) dt \right) v, \\ \implies T_d(v) &= {\sigma_d}^2 v^\top \left(\int_0^1 (t-1) \nabla^2 f(\mu + t\sigma_d v) dt \right) v + \frac{{\sigma_d}^2}{2} v^\top \nabla^2 f(\mu) v, \\ &= {\sigma_d}^2 v^\top \left(\int_0^1 (t-1) (\nabla^2 f(\mu + t\sigma_d v) - \nabla^2 f(\mu)) dt \right) v. \end{aligned}$$

Applying the Lipschitz bound on the Hessian from Assumption 5:

$$\begin{aligned} |T_d(v)| &\leq {\sigma_d}^2 \|v\|^2 \left\| \int_0^1 (t-1) (\nabla^2 f(\mu + t\sigma_d v) - \nabla^2 f(\mu)) dt \right\|, \\ &\leq {\sigma_d}^2 \|v\|^2 \int_0^1 (t-1) \left\| \nabla^2 f(\mu + t\sigma_d v) - \nabla^2 f(\mu) \right\| dt, \\ &\leq {\sigma_d}^2 \|v\|^2 \int_0^1 (t-1) L_d \|t\sigma_d v\| dt, \\ &= {\sigma_d}^3 \|v\|^3 L_d \left| \int_0^1 (t-1) t dt \right|, \\ &= \frac{L_d}{6} {\sigma_d}^3 \|v\|^3. \end{aligned}$$

Using this to bound Eq. (27):

$$\begin{aligned} \|I_{\text{loc}}\| &\leq \frac{L_d}{6} {\sigma_d}^2 \mathbb{E}_{v \sim P(v)} \left[\|v\|^4 \mathbb{1}(\|\sigma_d v\| < \rho) \right], \\ &\leq \frac{L_d}{6} {\sigma_d}^2 \mathbb{E}_{v \sim P(v)} \left[\|v\|^4 \right]. \end{aligned}$$

Now, (for fixed r) we apply the identity $\mathbb{E}_{v \sim P(v)} \left[\|v\|^4 \right] = \mathcal{O}\left((mn)^2 \right)$ with $mn = d$ from Lemma 3:

$$\|I_{\text{loc}}\| = \mathcal{O}(L_d (\sigma_d d)^2).$$

We now bound the tail region outside the ball:

$$\begin{aligned} I_{\text{tail}} &= \frac{1}{\sigma_d} \mathbb{E}_{v \sim P(v)} \left[v \cdot T_d(v) \mathbb{1}(\|\sigma_d v\| \geq \rho) \right], \\ &\leq \frac{1}{\sigma_d} \mathbb{E}_{v \sim P(v)} \left[\|v\| |T_d(v)| \mathbb{1}(\|\sigma_d v\| \geq \rho) \right], \\ &= \frac{1}{\sigma_d^2} \mathbb{E}_{v \sim P(v)} \left[\|\sigma_d v\| |T_d(v)| \mathbb{1}(\|\sigma_d v\| \geq \rho) \right]. \end{aligned}$$

Now under Assumptions 3, 4 and 5, $f(\mu + \sigma_d v)$ is polynomial bounded, $\|\nabla f(\mu)\| = \mathcal{O}(1)$ and $\|\nabla^2 f(\mu)\|$ is polynomial bounded hence there exists some finite constant $C > 0$ and finite polynomial order p such that:

$$\|\sigma_d v\| |T_d(v)| \leq C(1 + \|\mu + \sigma_d v\|^p).$$

We thus apply Lemma 5:

$$\begin{aligned} \frac{1}{\sigma_d^2} \mathbb{E}_{v \sim P(v)} \left[\|\sigma_d v\| |T_d(v)| \mathbb{1}(\|\sigma_d v\| \geq \rho) \right] &= \mathcal{O}\left(\frac{\sqrt{d}}{\sigma_d^2} \exp\left(-K \frac{\rho}{\sqrt{d}\sigma_d} \right) \right), \\ &= \mathcal{O}\left(\frac{d\sqrt{d}}{d\sigma_d^2} \exp\left(-K \frac{\rho}{\sqrt{d}\sigma_d} \right) \right). \end{aligned}$$

Now, as $\sigma_d\sqrt{d} = o(1)$, the exponential term dominates the prefactor $\frac{d\sqrt{d}}{d\sigma_d^2}$, we conclude:

$$\frac{1}{\sigma_d^2}\mathbb{E}_{v\sim P(v)}\left[\|\sigma_d v\||T_d(v)|\mathbb{1}(\|\sigma_d v\| \geq \rho)\right] = o(1)$$

Our final result follows from:

$$\begin{aligned}\|\text{vec}(\hat{g}_{\text{LR}}) - \nabla_\mu J(\theta)\| &= \|\text{vec}(\hat{g}_{\text{LR}}) - \nabla f(\mu) + \nabla f(\mu) - \nabla_\mu J(\theta)\|, \\ &\leq \|\text{vec}(\hat{g}_{\text{LR}}) - \nabla f(\mu)\| + \|\nabla f(\mu) - \nabla_\mu J(\theta)\|.\end{aligned}$$

We have already shown $\|\text{vec}(\hat{g}_{\text{LR}}) - \nabla f(\mu)\| = o(1)$ and under the assumptions for this theorem, Theorem 1 holds and so $\|\nabla f(\mu) - \nabla_\mu J(\theta)\| = o(1)$. □

D Asymptotic Rank Analysis

For convenience, we work with random vectors in our analysis. We analyse the vector $v^r = \text{vec}(E^r)$, which is the vectorisation of the low-rank matrix E^r. We denote $v = \text{vec}(E)$, which is the vectorisation of the full rank matrix E. Note $v \sim \mathcal{N}(0, I_d)$ which we denote as $P(v)$. We write v^r as a standardised sum of r independent, zero-mean random vectors. Let

$$u_i = \text{vec}\left(a_i b_i^\top\right), \tag{28}$$

where recall a_i and b_i are the ith column vectors of A and B so:

$$v^r = \frac{1}{\sqrt{r}} \sum_{i=1}^{r} u_i.$$

Denoting the covariance matrix of $p(u)$ as Σ_u, the central limit theorem proves that the distribution of v^r converges in distribution to a zero-mean Gaussian $\mathcal{N}(0, \Sigma_r)$. In Lemma 6, we derive the covariance matrix for Σ_u, which we prove is the identity. Our analysis uses an Edgeworth expansion (Bhattacharya & Ranga Rao, 1976) to characterise precisely the rate at which $P(v^r)$ converges to the limiting Gaussian distribution. In Lemma 7, we make an Edgeworth expansion of $P(v^r)$ to show that it is dominated by $\mathcal{O}\left(r^{-1}\right)$ terms and higher. These are then used to prove Lemma 8, which allows us to bound the integral of the remainder of the Edgeworth expansion, thereby characterising how fast $P(v^r)$ converges to the limiting Gaussian distribution.

Lemma 6. *Let Assumption 1 hold and u_i be defined in Eq. (28). Then the variable u_i has identity covariance matrix:*

$$\Sigma_u \coloneqq \mathbb{E}_{u_i \sim p(u_i)}[u_i u_i^\top] = I_d,$$

has finite 4th-order absolute moments:

$$\mathbb{E}_{u_i \sim p(u_i)}\left[\|u_i\|^4\right] < \infty,$$

and the vector $v^r = vec(E^r)$ is zero-mean and has identity covariance matrix:

$$\Sigma_v \coloneqq \mathbb{E}_{v^r \sim P(v^r)}[v^r {v^r}^\top] = I_d$$

Proof. Under the vec operator, the vector u_i can be written element wise as:

$$u_i = [a_1 b_1, a_2 b_1, \ldots a_m b_1, a_1 b_2, \ldots a_m b_n]^\top.$$

We note that all elements in the vector u_i have zero mean, and so the covariance matrix is the expectation of the outer product:

$$\Sigma_u = \mathbb{E}_{u_i \sim p(u_i)}\left[u_i {u_i}^\top\right].$$

The diagonal elements of Σ_u are:

$$\mathbb{E}_{a_i, b_j}\left[(a_i b_j)^2\right] = \mathbb{E}_{a_i}\left[a_i^2\right] \mathbb{E}_{b_j}\left[b_j^2\right] = 1. \tag{29}$$

As all elements of a, b and ϵ are zero-mean, off-diagonal elements are zero:

$$\mathbb{E}_{a_i, b_j, a_k, b_l}\left[a_i b_j a_k b_l\right] = 0 \quad i \neq k \text{ or } j \neq l. \tag{30}$$

Using Eqs. (29) and (30), our first result follows:

$$\Sigma_u = I_d.$$

Now, as u_i is a vector of elements which are sums and products of variables which all have finite 4th order moments from Assumption 1, it immediately follows that u has finite 4th order absolute moments.

For our final result, we can write v^r as sum of independent variables:

$$v^r = \sum_{i=1}^{r} \left(r^{-\frac{1}{2}} u_i \right) = \sum_{i=1}^{r} x_i,$$

where $x_i \coloneqq \frac{1}{\sqrt{r}} u_i$. As v^r is a sum of zero-mean vectors, it is also zero-mean. We use the fact that the covariance of r i.i.d. random variables is equal to the sum of the individual covariances, hence

$$\begin{aligned}
\mathbb{E}_{v^r}[v^r v^r] =& r\mathbb{E}_{x_i}[x_i x_i^\top], \\
=& r\mathbb{E}_{u_i}\left[\frac{1}{r} u_i u_i^\top\right], \\
=& \mathbb{E}_{u_i}\left[u_i u_i^\top\right], \\
=& I_d,
\end{aligned}$$

as required. □

Using Lemma 6, we see the asymptotic Gaussian density of v^r is a standard normal:

$$g(v^r) = \frac{1}{\sqrt{(2\pi)^d}} \exp\left(-\frac{\|v^r\|^2}{2}\right). \tag{31}$$

which is the density of $P(v)$, where recall $v = \text{vec}(E)$, is the vectorisation of the full rank matrix E.

Although $P(v^r)$ does not have a density in the usual sense for low-rank r, we can still approximate it with a distribution $\hat{p}(v^r)$ by making a Taylor series expansion of its characteristic function, which always exists regardless of whether $P(v^r)$ has a well-defined density or not. We now derive the 4th order Edgeworth expansion for $P(v^r)$. Our proof reveals that 3rd order cumulants control all terms in the expansion that decay at rate $\mathcal{O}\left(r^{-\frac{1}{2}}\right)$. As 3rd order cumulants are all zero due to symmetry in Assumption 1, the overall decay rate is controlled by $\mathcal{O}\left(r^{-1}\right)$ terms associated with 4th order cumulants. It is for this reason that we obtain a faster convergence rate than the standard central limit theorem.

Lemma 7. *Let Assumption 1 hold and let $v^r = vec(E^r)$ and u_i be defined in Eq. (28). Let $g(v^r)$ denote the limiting Gaussian density in Eq. (31). Then, the 2nd order Edgeworth expansion of v^r is a distribution $\hat{P}(v^r)$ defined by the approximate density:*

$$\hat{p}(v^r) = g(v^r) + \frac{1}{4!r} g(v^r) \sum_{i,j,k,l} \kappa^4_{i,j,k,l} H_{i,j,k,l}(v^r),$$

where:

$$H_{i,j,k,l}(v^r) \coloneqq \exp\left(\frac{\|v^r\|^2}{2}\right) \frac{\partial^4}{\partial v^r_i \partial v^r_j \partial v^r_k \partial v^r_l} \exp\left(-\frac{\|v^r\|^2}{2}\right)$$

is a 4th order Hermite polynomial associated with $g(v^r)$ (Laplace, 1811; Hall, 1992; Withers, 2000).

Proof. We denote the characteristic function of $P(u_i)$ as:

$$\varphi_U(\omega) = \int \exp\left(-i\omega^\top u\right) dP(u),$$

and the characteristic function of $P(v^r)$ as:

$$\varphi_r(\omega) = \int \exp\left(-i\omega^\top u\right) dP(v^r).$$

Recall $v^r = \frac{1}{\sqrt{r}} \sum_{i=1}^{r} u_i$ is the sum of r i.i.d. copies of $\frac{1}{\sqrt{r}} u_i$. Using the scaling property of the Fourier transform, the characteristic function of $\frac{1}{\sqrt{r}} u_i$ is $\varphi_U\left(\frac{\omega}{\sqrt{r}}\right)$. The distribution of a sum of r independent random

variables is given by the r-fold convolution of the individual distributions. As convolution in the spatial domain corresponds to multiplication in the frequency domain, the characteristic function of v^r is (Bhattacharya & Ranga Rao, 1976):

$$\varphi_r(\omega) = \left(\varphi_U\left(\frac{\omega}{\sqrt{r}}\right)\right)^r.$$

Taking logarithms yields the log-characteristic function:

$$\begin{aligned}\log \varphi_r(\omega) &= r \log\left(\varphi_U\left(\frac{\omega}{\sqrt{r}}\right)\right), \\ &= r K_U\left(\frac{\omega}{\sqrt{r}}\right),\end{aligned}$$

where $K_U(\omega) \coloneqq \log \varphi_U(\omega)$. The cumulants are defined by

$$\kappa^{(n)}_{i_1,\ldots,i_n} \coloneqq i^{-n} \left.\frac{\partial^n K_U(\omega)}{\partial\omega_{i_1}\cdots\partial\omega_{i_n}}\right|_{\omega=0}.$$

The Edgeworth expansion proceeds by a Taylor expansion of $rK_U\left(\frac{\omega}{\sqrt{r}}\right)$ about $\omega = 0$. A 4th order expansion yields:

$$\begin{aligned}rK_U\left(\frac{\omega}{\sqrt{r}}\right) \approx\ & rK_U(0) + \sqrt{r}\sum_i \omega_i \kappa^1_i + \frac{1}{2!}\sum_{i,j}\omega_i\omega_j\kappa^2_{i,j} \\ &+ \frac{1}{3!\sqrt{r}}\sum_{i,j,k}\omega_i\omega_j\omega_k\kappa^3_{i,j,k} + \frac{1}{4!r}\sum_{i,j,k,l}\omega_i\omega_j\omega_k\omega_l\kappa^4_{i,j,k,l},\end{aligned}$$

where $K_U(0) = 0$. Under Assumption 8, u_i is symmetric, hence all odd-order cumulants vanish: $\kappa^1 = \kappa^3 = 0$. The second-order cumulant satisfies

$$\sum_{i,j}\omega_i\omega_j\kappa^2_{i,j} = -\omega^\top \Sigma_u \omega,$$

and from Lemma 6 we have $\Sigma_u = I$. Substituting yields:

$$rK_U\left(\frac{\omega}{\sqrt{r}}\right) \approx -\frac{\|\omega\|^2}{2} + \frac{1}{4!r}\sum_{i,j,k,l}\omega_i\omega_j\omega_k\omega_l\kappa^4_{i,j,k,l}.$$

Exponentiating and expanding the exponential to first-order in $1/r$ gives:

$$\begin{aligned}\varphi_r(\omega) &= \exp\left(rK_U\left(\frac{\omega}{\sqrt{r}}\right)\right), \\ &\approx \exp\left(-\frac{\|\omega\|^2}{2}\right)\left(1 + \frac{1}{4!r}\sum_{i,j,k,l}\omega_i\omega_j\omega_k\omega_l\kappa^4_{i,j,k,l}\right).\end{aligned}$$

Taking the inverse Fourier transform (with the convention $\mathcal{F}^{-1}(f)(v) = (2\pi)^{-d}\int e^{i\omega^\top v} f(\omega)d\omega$) yields:

$$\hat{p}(v^r) = g(v^r) + \frac{1}{4!r}\sum_{i,j,k,l}\kappa^4_{i,j,k,l}\frac{\partial^4}{\partial v^r_i \partial v^r_j \partial v^r_k \partial v^r_l} g(v^r),$$

and using the identity $H_{i,j,k,l}(v^r) = g(v^r)^{-1}\frac{\partial^4}{\partial v^r_i \partial v^r_j \partial v^r_k \partial v^r_l} g(v^r)$, we recover the stated Edgeworth density. □

We now apply key results from Bhattacharya & Ranga Rao (1976) to bound the difference in expectation between the low-rank distribution and the Edgeworth approximation as well as the difference in expectation between the true ES Gaussian distribution and the Edgeworth approximation.

Lemma 8. *Let $f(v) \coloneqq f(M = \mu + \sigma mat(v))$, let $P(v) = \mathcal{N}(0, I_d)$, $P(v^r)$ be the distribution of v^r and $\hat{P}(v^r)$ be the 2nd order Edgeworth expansion of $P(v^r)$. Let Assumptions 1 and 7 hold and let $v^r = vec(E^r)$ and u_i be defined in Eq. (28). Then:*

$$\left\| \mathbb{E}_{v^r \sim P(v^r)} \left[v^r \cdot f(v^r) \right] - \mathbb{E}_{v^r \sim \hat{P}(v^r)} \left[v^r \cdot f(v^r) \right] \right\| = \mathcal{O}\left(r^{-1}\right),$$
$$\left\| \mathbb{E}_{v \sim P(v)} \left[v \cdot f(v) \right] - \mathbb{E}_{v \sim \hat{P}(v)} \left[v \cdot f(v) \right] \right\| = \mathcal{O}\left(r^{-1}\right).$$

Proof. From Lemma 7, we have shown that the Edgeworth expansion for $P(v^r)$ is controlled by 4th order cumulants and higher, that is;

$$\hat{p}(v^r) = g(v^r) + \frac{1}{4!r} g(v^r) \sum_{i,j,k,l} \kappa^4_{i,j,k,l} H_{i,j,k,l}(v^r). \tag{32}$$

We show that the three assumptions needed to apply Bhattacharya & Ranga Rao (1976, Theorem 20.1) to obtain our result using Eq. (32) hold. Firstly, the boundedness assumption of the integrand holds:

$$\sup_{v^r} \frac{\|f(v^r)v^r\|}{1+\|v^r\|} \leq \sup_{v^r} |f(v^r)| < \infty.$$

Secondly, the sampling regularity assumption that u_i (as defined in Eq. (28)) is zero-mean i.i.d. (satisfied under Assumption 1) with finite 4th order moments (satisfied from Lemma 6) holds. Let $\varphi_U(\omega)$ denote the characteristic function of $p(u)$, then the final assumption we need to verify is the Cramer condition: $\limsup_{\|\omega\| \to \infty} \varphi_U(\omega) < 1$, which is satisfied from the Riemann-Lebesgue lemma Folland (1999, Theorem 8.22) because $p_0(\cdot)$ is absolutely continuous under Assumption 1 and hence $|\varphi_U(\omega)| \to 0$ as $\|\omega\| \to 0$. Our first result thus follows from applying Bhattacharya & Ranga Rao (1976, Theorem 20.1):

$$\left\| \mathbb{E}_{v^r \sim P(v^r)} \left[v^r \cdot f(v^r) \right] - \mathbb{E}_{v^r \sim \hat{P}(v^r)} \left[v^r \cdot f(v^r) \right] \right\| = \mathcal{O}\left(r^{-1}\right).$$

We now derive our second result.

$$\begin{aligned}
\mathbb{E}_{v \sim \hat{P}(v)} \left[v \cdot f(v) \right] &= \int v \cdot f(v) g(v) \left(1 + \frac{1}{4!r} \sum_{i,j,k,l} \kappa^4_{i,j,k,l} H_{i,j,k,l}(v) \right) dv, \\
&= \mathbb{E}_{v \sim P(v)} \left[v \cdot f(v) \right] - \int v \cdot f(v) g(v) \frac{1}{4!r} \sum_{i,j,k,l} \kappa^4_{i,j,k,l} H_{i,j,k,l}(v) dv,
\end{aligned}$$

hence

$$\begin{aligned}
\| \mathbb{E}_{v \sim P(v)} \left[v \cdot f(v) \right] - \mathbb{E}_{v \sim \hat{P}(v)} \left[v \cdot f(v) \right] \| &= \frac{1}{r} \left\| \int v \cdot f(v) \frac{1}{4!} \sum_{i,j,k,l} \kappa^4_{i,j,k,l} H_{i,j,k,l}(v) g(v) dv \right\|, \\
&\leq \frac{1}{r} \int \|v\| \cdot |f(v)| \frac{1}{4!} \sum_{i,j,k,l} |\kappa^4_{i,j,k,l} H_{i,j,k,l}(v)| g(v) dv.
\end{aligned}$$

Now by definition, $H_{i,j,k,l}(v)$ is a 4th order Hermite polynomial and under Assumption 7, $|f(v)|$ is bounded, hence $\|v\| \cdot |f(v)| \frac{1}{4!r} \sum_{i,j,k,l} |\kappa^4_{i,j,k,l} H_{i,j,k,l}(v)|$ has polynomial growth of order 5 and is bounded by:

$$\|v\| \cdot |f(v)| \frac{1}{4!} \sum_{i,j,k,l} |\kappa^4_{i,j,k,l} H_{i,j,k,l}(v)| \leq C(1 + \|v\|^5)$$

for some finite $C > 0$. As the expectation of a finite order polynomial under $\mathcal{N}(0, I_d)$ is bounded, it thus follows:

$$\|\mathbb{E}_{v\sim P(v)}\left[v \cdot f(v)\right] - \mathbb{E}_{v\sim \hat{P}(v)}\left[v \cdot f(v)\right]\| \leq \frac{1}{r}\int C(1+\|v\|^5)g(v)dv = \mathcal{O}\left(r^{-1}\right),$$

as required.

□

Using Lemma 8, we have all ingredients needed derive our main about the convergence result, which follows after some simple algebra on the norm:

Theorem 4. *Let Assumptions 1 and 7 hold, then:*

$$\|\nabla_\mu J(\theta) - \hat{g}^r_{\text{LR}}\|_F = \mathcal{O}\left(r^{-1}\right).$$

Proof. We start by converting the Frobenius norm to vector form using Eq. (13):

$$\begin{aligned}
\|\nabla_\mu J(\theta) - g^r_{\text{LR}}\|_F &= \left\|\frac{1}{\sigma}(\text{vec}\left(\mathbb{E}_E\left[E \cdot f(W = M + \sigma E)\right]\right) - \text{vec}\left(\mathbb{E}_{E^r}\left[E^r \cdot f(W = M + \sigma E^r)\right]\right))\right\|, \\
&= \left\|\frac{1}{\sigma}(\mathbb{E}_E\left[\text{vec}(E) f(W = M + \sigma E)\right] - \mathbb{E}_{E^r}\left[\text{vec}(E^r) f(W = M + \sigma E^r)\right])\right\|, \\
&= \left\|\frac{1}{\sigma}(\mathbb{E}_v\left[v f(v)\right] - \mathbb{E}_{v^r}\left[v^r f(v^r)\right])\right\|,
\end{aligned}$$

where $f(v) \coloneqq f(M = \mu + \sigma\text{mat}(v))$ and $v = \text{vec}(E)$ is the vectorisation of variable E, which is distributed as $v \sim P(v) \coloneqq \mathcal{N}(0, I_d)$. Let $\hat{P}(v)$ be the distribution for the 2nd order Edgeworth expansion, which we derived in Lemma 7. Since $\hat{P}(v^r)$ and $\hat{P}(v)$ are identified as the same Edgeworth-expanded distribution on $\mathbb{R}^d$, we may equivalently write:

$$\mathbb{E}_{v^r\sim\hat{P}(v^r)}\left[v^r f(v^r)\right] = \mathbb{E}_{v\sim\hat{P}(v)}\left[v^r f(v)\right],$$

hence:

$$\begin{aligned}
&\mathbb{E}_v\left[v f(v)\right] - \mathbb{E}_{v^r}\left[v^r f(v^r)\right] = \mathbb{E}_v\left[v f(v)\right] - \mathbb{E}_{v\sim\hat{P}(v)}\left[v f(v)\right] + \mathbb{E}_{v^r\sim\hat{P}(v^r)}\left[v^r f(v^r)\right] - \mathbb{E}_{v^r}\left[v^r f(v^r)\right], \\
&\implies \|\nabla_\mu J(\theta) - \hat{g}^r_{\text{LR}}\|_F \leq \frac{1}{\sigma}\left\|\mathbb{E}_v\left[v f(v)\right] - \mathbb{E}_{v\sim\hat{P}(v)}\left[v f(v)\right]\right\| \\
&\qquad + \frac{1}{\sigma}\left\|\mathbb{E}_{v^r\sim\hat{P}(v^r)}\left[v^r f(v^r)\right] - \mathbb{E}_{v^r}\left[v^r f(v^r)\right]\right\|.
\end{aligned}$$

Applying Lemma 8 to each bound yields our desired result:

$$\|\nabla_\mu J(\theta) - \hat{g}^r_{\text{LR}}\|_F = \mathcal{O}\left(r^{-1}\right).$$

□

D.1 Mean Field Score Function Approximator

We will use nth order Bessel functions of the second kind $K_n(z)$ (Basset, 1888; Macdonald, 1899; Watson, 1944), which are conveniently represented by the integral equations:

$$K_n(z) = \int_0^\infty \exp(-z\cosh\theta)\cosh(n\theta)d\theta.$$

Bessel functions are the solutions to systems of differential equations that occur naturally in phenomena where there is strong radial symmetry, typically involving the propagation of spherical waves from points like the ripples formed from water droplets (Whitham, 1999). For our setting, Bessel functions describe the

probability density of the product of rotationally invariant random variables, whose solution is analogous to the interference pattern of two spherical wave propagators.

Using the representation, we find the derivative of the zeroth order function takes the recursive form:

$$\frac{dK_0(z)}{dz} = -\int_0^\infty \exp(-z\cosh\theta)\cosh(\theta)d\theta = -K_1(z). \tag{33}$$

More generally, the derivative of the nth order Bessel function is Watson (1944, Section 3.71, Eq. 4):

$$\frac{dK_n(z)}{dz} = \frac{n}{z}K_n(z) - K_{n+1}(z). \tag{34}$$

D.2 Derivation of Mean-field Approximators

To derive a mean-field approximation, we assume that the elements of A and B are drawn independently from the set of generalised Gaussian distributions (GGDs):

Assumption 8. *Assume each element $a_{i,j} \sim \mathcal{GG}(s,p)$ and $b_{i,j} \sim \mathcal{GG}(s,p)$ of A and B is independently distributed according to the zero-mean generalised Gaussian distribution $\mathcal{GG}(s,p)$ with density:*

$$\mathcal{GG}(x|s,p) = \frac{p}{2s\Gamma\left(\frac{1}{p}\right)}\exp\left(-\left|\frac{x}{s}\right|^p\right),$$

where $0 < s < \infty$ is the scale parameter, $p > 0$ the shape parameter and $\Gamma(\cdot)$ is the gamma function.

We observe common distributions emerge from the set of GGDs including the Laplace for $p = 1$, the Gaussian for $p = 2$ and the uniform over $[-s, +s]$ in the limit $p \to \infty$.

If we make the assumption that all elements of E are independent (this is true as r grows) then we can write $p(E) \approx \hat{p}(E) \coloneqq \prod_{i=1}^m \prod_{j=1}^n p(E_{i,j})$ as the product of the marginal distributions. Under this approximation, the score function can be defined element-wise as:

$$[\nabla_E \log p(E)]_{i,j} \approx \hat{S}(E_{i,j}) \coloneqq \partial_{E_{i,j}} \log p(E_{i,j}).$$

Using this approximation we apply the score function $\hat{S}(\cdot)$ element-wise to the matrix E:

$$g_{\text{LR}} \approx \hat{g}_{\text{MF}} \coloneqq -\frac{1}{\sigma}\mathbb{E}_{E\sim p(E)}\left[f(W = M + \sigma E)\cdot \hat{S}\odot(E)\right].$$

For $r = 1$, $\hat{S}(\cdot)$ has a convenient analytic form for all members of the set of GGDs:

Theorem 5. *Let Assumption 8 hold and $r = 1$. Then the distribution over marginals $p(E_{i,j})$ is:*

$$p(E_{i,j}) = \frac{p}{\left(s\Gamma\left(\frac{1}{p}\right)\right)^2}K_0\left(\frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}\right), \tag{35}$$

where $K_0(\cdot)$ is the zeroth-order modified Bessel function of the second kind and the marginal score function is defined element-wise as:

$$\hat{S}(E_{i,j}) = -\frac{K_1\left(\frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}\right)}{K_0\left(\frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}\right)}\cdot\frac{p|E_{i,j}|^{\frac{p}{2}-1}\text{sign}(E_{i,j})}{s^p}.$$

Proof. For $r = 1$, we denote the elements of vector A as a_i and elements of vector B as b_j, then the elements of matrix $E = AB^\top$ are: $E_{i,j} = a_i b_j$. We now derive the distribution of the unnormalised variables: $E_{i,j}$

using the formula for the distribution of the product of two independent random variables (Rohatgi, 1976; Grimmett & Stirzaker, 1993):

$$\begin{aligned}
p(E_{i,j}) &= \int_{-\infty}^{\infty} p(a_i) p\left(b_j = \frac{E_{i,j}}{a_i}\right) \frac{1}{|a_i|} da_i, \\
&= \left(\frac{p}{2s\Gamma\left(\frac{1}{p}\right)}\right)^2 \int_{-\infty}^{\infty} \exp\left(-\left|\frac{a_i}{s}\right|^p\right) \exp\left(-\left|\frac{E_{i,j}}{a_i s}\right|^p\right) \frac{1}{|a_i|} da_i, \\
&= 2\left(\frac{p}{2s\Gamma\left(\frac{1}{p}\right)}\right)^2 \int_{0}^{\infty} \exp\left(-\left|\frac{a_i}{s}\right|^p\right) \exp\left(-\left|\frac{E_{i,j}}{a_i s}\right|^p\right) \frac{1}{|a_i|} da_i,
\end{aligned}$$

where we have used symmetry of the integrand about 0 to derive the final line. Now, making the substitution $x = \left(\frac{a_i}{s}\right)^p$, we have:

$$\frac{da_i}{dx} = \frac{s x^{\frac{1}{p}-1}}{p}, \quad a_i = s x^{\frac{1}{p}}$$

hence:

$$p(E_{i,j}) = \frac{p}{\left(s\Gamma\left(\frac{1}{p}\right)\right)^2} \frac{1}{2} \int_0^{\infty} \exp\left(-x - \frac{1}{x}\frac{|E_{i,j}|^p}{s^{2p}}\right) \frac{1}{x} dx.$$

Now, we use the identity (Temme, 1996, Theorem 9.42):

$$K_0(z) = \frac{1}{2} \int_0^{\infty} \exp\left(-x - \frac{z^2}{4x}\right) \frac{1}{x} dx,$$

with $z = \frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}$ to yield:

$$p(E_{i,j}) = \frac{p}{\left(s\Gamma\left(\frac{1}{p}\right)\right)^2} K_0\left(\frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}\right),$$

as required for Eq. (35). Now we derive the marginal score function by applying the chain rule:

$$\begin{aligned}
\partial_{E_{i,j}} \log p(E_{i,j}) &= \partial_{E_{i,j}} \log K_0\left(\frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}\right), \\
&= \partial_z \log K_0\left(z = \frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}\right) \partial_{E_{i,j}} \frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}, \\
&= \partial_z \log K_0\left(z = \frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}\right) \frac{p|E_{i,j}|^{\frac{p}{2}-1}\text{sign}(E_{i,j})}{s^p}, \\
&= \frac{\partial_z K_0\left(z = \frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}\right)}{K_0\left(z = \frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}\right)} \cdot \frac{p|E_{i,j}|^{\frac{p}{2}-1}\text{sign}(E_{i,j})}{s^p}, \\
&= -\frac{K_1\left(\frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}\right)}{K_0\left(\frac{2|E_{i,j}|^{\frac{p}{2}}}{s^p}\right)} \cdot \frac{p|E_{i,j}|^{\frac{p}{2}-1}\text{sign}(E_{i,j})}{s^p},
\end{aligned}$$

where we have used the identity $\partial_z K_0(x) = -K_1(x)$ from Eq. (33). □

For $r > 1$ we can derive $\hat{S}(\cdot)$ for the Gaussian sampling case:

Theorem 6. *Let Assumption 8 hold and $p = 2$. Then the distribution over marginals $p(E_{i,j})$ is:*

$$p(E_{i,j}) = \frac{2\sqrt{r}|\sqrt{r}E_{i,j}|^{\frac{r-1}{2}}}{s^{r+1}\sqrt{\pi}\Gamma\left(\frac{r}{2}\right)} \cdot K_{\frac{r-1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right).$$

and the score function is (for $E_{i,j} \neq 0$):

$$\hat{S}(E_{i,j}) = \frac{r-1}{E_{i,j}} - \frac{2\sqrt{r}\text{sign}(E_{i,j})}{s^2} \frac{K_{\frac{r+1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right)}{K_{\frac{r-1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right)}.$$

Proof. Each element $E_{i,j}$ is the sum of r independent variables $u_{i,j,l} \coloneqq a_{i,l}b_{j,l}$ distributed according to Eq. (35) with $p = 2$:

$$E_{i,j} = \frac{1}{\sqrt{r}}\sum_{l=1}^{r} a_{i,l}b_{j,l} = \frac{1}{\sqrt{r}}\sum_{l=1}^{r} u_{i,j,l}.$$

Let $Z_{i,j} = \sqrt{r}E_{i,j}$, hence:

$$Z_{i,j} = \sum_{l=1}^{r} u_{i,j,l}.$$

We first find the density $p(Z_{i,j})$. From Eq. (35), the distribution of each $u_{i,j,l}$ is:

$$p(u_{i,j,l}) = \frac{2}{s^2\pi}K_0\left(\frac{2|u_{i,j,l}|}{s^2}\right)$$

We use the fact that the PDF of a sum of r independent random variables (i.e. $Z_{i,j}$) is given by the r-fold convolution of the individual PDFs. As convolution in the spatial domain is equal to multiplication in the frequency domain, the PDF $p(Z_{i,j})$ follows by taking Fourier transform of $p(u_{i,j,l})$, taking the power r and then taking the inverse Fourier transform:

$$p(Z_{i,j}) = \left(\frac{2}{s^2\pi}\right)^r \mathcal{F}^{-1}\left[\mathcal{F}\left[K_0\left(\frac{2|\cdot|}{s^2}\right)\right]^r\right](Z_{i,j}),$$

where recall from Section A with $d = 1$, $\mathcal{F}[f](\omega) \coloneqq \int f(x)\exp(-i\omega x)dx$ denotes the Fourier transform and $\mathcal{F}^{-1}[\tilde{f}](x) \coloneqq \frac{1}{2\pi}\int \tilde{f}(\omega)\exp(i\omega x)d\omega$, the inverse Fourier transform. Taking the Fourier transform of the Bessel function:

$$\begin{aligned}
\mathcal{F}\left[K_0\left(\frac{2|\cdot|}{s^2}\right)\right](\omega) &= \int \exp(-i\omega x)K_0\left(\frac{2|x|}{s^2}\right)dx, \\
&= \int \cos(\omega x)K_0\left(\frac{2|x|}{s^2}\right)dx - i\int \sin(\omega x)K_0\left(\frac{2|x|}{s^2}\right)dx, \\
&= \int \cos(\omega x)K_0\left(\frac{2|x|}{s^2}\right)dx, \\
&= 2\int_0^\infty \cos(\omega x)K_0\left(\frac{2x}{s^2}\right)dx, \qquad (36)
\end{aligned}$$

where we have used the fact that $K_0\left(\frac{2|x|}{s^2}\right)$ is an even function of x and so its integral with $\sin(\omega x)$ in the second line is zero. Using a standard result, we can evaluate the integral in Eq. (36) Gradshteĭn et al. (2015, 6.671 Integral 14):

$$\mathcal{F}\left[K_0\left(\frac{2|\cdot|}{s^2}\right)\right](\omega) = \frac{\pi}{\sqrt{\omega^2 + \left(\frac{2}{s^2}\right)^2}},$$

hence:

$$
\begin{aligned}
p(Z_{i,j}) &= \left(\frac{2}{s^2\pi}\right)^r \mathcal{F}^{-1}\left[\frac{\pi^r}{\left(\omega^2+\left(\frac{2}{s^2}\right)^2\right)^{\frac{r}{2}}}\right](Z_{i,j}),\\
&= \left(\frac{2}{s^2}\right)^r \mathcal{F}^{-1}\left[\left(\omega^2+\left(\frac{2}{s^2}\right)^2\right)^{-\frac{r}{2}}\right](Z_{i,j}),\\
&= \left(\frac{2}{s^2}\right)^r \frac{1}{2\pi}\int \exp(i\omega Z_{i,j})\left(\omega^2+\left(\frac{2}{s^2}\right)^2\right)^{-\frac{r}{2}} d\omega,\\
&= \left(\frac{2}{s^2}\right)^r \frac{1}{2\pi}\Bigg(\int \cos(\omega Z_{i,j})\left(\omega^2+\left(\frac{2}{s^2}\right)^2\right)^{-\frac{r}{2}} d\omega\\
&\qquad + i\int \sin(\omega Z_{i,j})\left(\omega^2+\left(\frac{2}{s^2}\right)^2\right)^{-\frac{r}{2}} d\omega\Bigg),\\
&= \left(\frac{2}{s^2}\right)^r \frac{1}{2\pi}\int \cos(\omega Z_{i,j})\left(\omega^2+\left(\frac{2}{s^2}\right)^2\right)^{-\frac{r}{2}} d\omega,\\
&= \left(\frac{2}{s^2}\right)^r \frac{1}{\pi}\int_0^\infty \cos(\omega Z_{i,j})\left(\omega^2+\left(\frac{2}{s^2}\right)^2\right)^{-\frac{r}{2}} d\omega, \qquad (37)
\end{aligned}
$$

where we have used the fact that the integrand is an even function and so its integral with $\sin(\omega Z_{i,j})$ is zero to derive the penultimate line. To evaluate the integral in Eq. (37) we apply Gradshteĭn et al. (2015, 3.771 Integral 2):

$$
\begin{aligned}
p(Z_{i,j}) &= \left(\frac{2}{s^2}\right)^r \cdot \frac{1}{\sqrt{\pi}\Gamma\left(\frac{r}{2}\right)}\left(\frac{s^2|Z_{i,j}|}{4}\right)^{\frac{r-1}{2}} \cdot K_{\frac{r-1}{2}}\left(\frac{2|Z_{i,j}|}{s^2}\right),\\
&= \frac{2|Z_{i,j}|^{\frac{r-1}{2}}}{s^{r+1}\sqrt{\pi}\Gamma\left(\frac{r}{2}\right)} \cdot K_{\frac{r-1}{2}}\left(\frac{2|Z_{i,j}|}{s^2}\right).
\end{aligned}
$$

Using the transformation of variables $E_{i,j} = \frac{1}{\sqrt{r}}Z_{i,j}$ yields our desired results:

$$
\begin{aligned}
p(E_{i,j}) &= \sqrt{r}p(Z_{i,j} = \sqrt{r}E_{i,j}),\\
&= \frac{2\sqrt{r}|\sqrt{r}E_{i,j}|^{\frac{r-1}{2}}}{s^{r+1}\sqrt{\pi}\Gamma\left(\frac{r}{2}\right)} \cdot K_{\frac{r-1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right).
\end{aligned}
$$

Now, we derive the score function:

$$
\begin{aligned}
\partial_{E_{i,j}} \log p(E_{i,j}) &= \frac{r-1}{2}\cdot\partial_{E_{i,j}}\log|\sqrt{r}E_{i,j}| + \partial_{E_{i,j}}\log K_{\frac{r-1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right),\\
&= \frac{r-1}{2E_{i,j}} + \frac{2\partial_{E_{i,j}}|\sqrt{r}E_{i,j}|}{s^2}\frac{\partial_x K_{\frac{r-1}{2}}\left(x=\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right)}{K_{\frac{r-1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right)},\\
&= \frac{r-1}{2E_{i,j}} + \frac{2\sqrt{r}\text{sign}(E_{i,j})}{s^2}\frac{\partial_x K_{\frac{r-1}{2}}\left(x=\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right)}{K_{\frac{r-1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right)},
\end{aligned}
$$

Now, from Eq. (34) for $E_{i,j} \neq 0$:

$$\begin{aligned}
\frac{\partial_x K_{\frac{r-1}{2}}(x)}{K_{\frac{r-1}{2}}(x)} &= \frac{\frac{r-1}{2x} K_{\frac{r-1}{2}}(x) - K_{\frac{r+1}{2}}(x)}{K_{\frac{r-1}{2}}(x)}, \\
&= \frac{r-1}{2x} - \frac{K_{\frac{r+1}{2}}(x)}{K_{\frac{r-1}{2}}(x)}, \\
\implies \partial_{E_{i,j}} \log p(E_{i,j}) &= \frac{r-1}{2E_{i,j}} + \frac{(r-1)\text{sign}(E_{i,j})}{2|E_{i,j}|} - \frac{2\sqrt{r}\text{sign}(E_{i,j})}{s^2} \frac{K_{\frac{r+1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right)}{K_{\frac{r-1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right)}, \\
&= \frac{r-1}{2E_{i,j}} + \frac{(r-1)}{2E_{i,j}} - \frac{2\sqrt{r}\text{sign}(E_{i,j})}{s^2} \frac{K_{\frac{r+1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right)}{K_{\frac{r-1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right)}, \\
&= \frac{r-1}{E_{i,j}} - \frac{2\sqrt{r}\text{sign}(E_{i,j})}{s^2} \frac{K_{\frac{r+1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right)}{K_{\frac{r-1}{2}}\left(\frac{2|\sqrt{r}E_{i,j}|}{s^2}\right)},
\end{aligned}$$

as required. □

E EGGROLL Speed

All timings were done on a single GPU on a GH200 (equivalent to a single H100) for a linear model with dimension 8192 in bfloat16, allowing a maximum batch size of 1024. For the graph in Fig. 2a, we pre-generate the noises instead of integrating the noise generation into the forward pass.

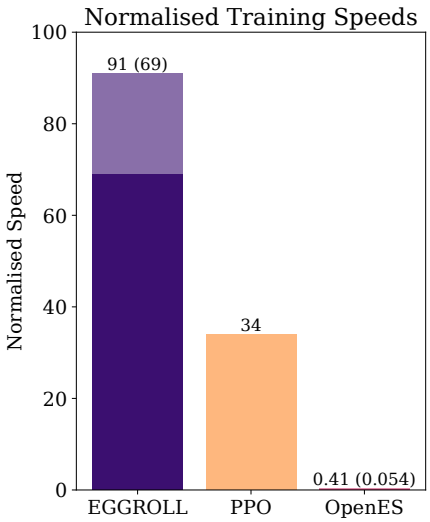


Figure 7: Relative speed of EGGROLL, when including jax noise regeneration.

In Fig. 7, we consider the impact of regenerating noises on-the-fly using jax PRNG. The darker area and value in parenthesis for EGGROLL and OpenES indicate the speed when regenerating noises on-the-fly, while the full bar indicates the speed when the noises are already generated.

We regenerate noises on the fly in our primary jax codebase, but pre-generating the EGGROLL perturbations beforehand is also a practical possibility since low-rank perturbations only require a small amount of memory, proportional to the square root of the size of the original parameter matrices.

F Arithmetic Intensity Analysis

In this section, we derive the arithmetic intensity of standard batched inference, Gaussian matrix ES, and EGGROLL. We calculate arithmetic intensity as the number of operations divided by the total number of bytes read from or written to. For context, for the (b)float16 datatype on an H100 GPU, there are approximately 1000 teraFLOPS of compute (without sparsity) and 3.35 TB/s of GPU memory bandwidth, meaning that the roofline threshold is approximately 300 ops/byte, defined as the minimum for computation needed for it to be the bottleneck instead of memory movement.

In the following subsections, we are considering a single linear layer with mean parameter $M \in \mathbb{R}^{d_{out} \times d_{in}}$ and a batch of inputs $u \in \mathbb{R}^{B \times d_{in}}$. All operations occur with a precision of s bytes per element.

F.1 Arithmetic Intensity of Standard Batched Inference

In standard batched inference, we wish to simply calculate uM^T. The total bytes read as input are $B \times d_{in} \times s$ (for u) and $d_{out} \times d_{in} \times s$ (for M), and the total bytes written as output are $B \times d_{out} \times s$. The total number of operations are $B \times d_{in} \times d_{out} \times 2$ since matrix multiplication requires both multiplications and additions for each element of u across all of d_{out}. Therefore, the arithmetic intensity is:

$$\frac{B \times d_{in} \times d_{out} \times 2}{B \times d_{in} \times s + B \times d_{out} \times s + d_{out} \times d_{in} \times s}.$$

When $s = 2$ (for (b)float16) and $d_{out} = d_{in} = m$, the arithmetic intensity simplifies to

$$\frac{Bm}{2B + m}.$$

The batch size needed to achieve a desired arithmetic intensity of A is derived as follows:

$$\begin{aligned} Bm &= 2AB + Am \\ Bm - 2AB &= Am \\ B &= \frac{Am}{m - 2A} \end{aligned}$$

Therefore, achieving an arithmetic intensity of 300 ops/byte with $m = 8192$ requires a minimum batch size of 324.

F.2 Arithmetic Intensity of Gaussian Matrix ES

In Gaussian matrix ES, we assume access to pre-generated perturbations of shape $\mathbb{R}^{B \times d_{out} \times d_{in}}$. The total bytes read as input are $B \times d_{in} \times s$ (for u) and $B \times d_{out} \times d_{in} \times s$ (for M), and the total bytes written as output are $B \times d_{out} \times s$. Otherwise, the total number of operations is identical to standard batched inference, giving us an arithmetic intensity of

$$\frac{B \times d_{in} \times d_{out} \times 2}{B \times d_{in} \times s + B \times d_{out} \times s + B \times d_{out} \times d_{in} \times s} = \frac{d_{in} \times d_{out} \times 2}{d_{in} \times s + d_{out} \times s + d_{out} \times d_{in} \times s}.$$

When $s = 2$ (for (b)float16) and $d_{out} = d_{in} = m$, the arithmetic intensity simplifies to

$$\frac{m}{2 + m}.$$

This means that arithmetic intensity is always strictly less than 1, regardless of batch size or dimensionality. The common way to increase arithmetic intensity is to bring it closer to standard batched inference, reusing the same perturbation across multiple inputs. For instance, when $m = 8192$, achieving an arithmetic intensity of 300 ops/byte requires that each perturbation is reused at least 324 times, and smaller values of m need to be reused even more often.

F.3 Arithmetic Intensity of EGGROLL

For EGGROLL, we assume access to the pre-generated decomposed perturbations $A \in \mathbb{R}^{B\times d_{out}\times r}$ and $B \in \mathbb{R}^{B\times d_{in}\times r}$. Therefore, the bytes read as pure input are $B\times d_{in}\times s+B\times(d_{in}+d_{out})\times r\times s+d_{out}\times d_{in}\times s$ and the bytes written as pure output are $B \times d_{out} \times s$. However, the efficient low-rank perturbation calculation requires writing and reading an intermediate matrix of shape $B \times r$, so the total bytes read are

$$(B \times d_{in} + B \times (d_{in} + d_{out} + 2) \times r + d_{out} \times d_{in} + B \times d_{out}) \times s.$$

The total number of operations includes the amount for standard batch inference, $B \times d_{in} \times d_{out} \times 2$, along with the rank-$r$ perturbations, $B \times (d_{in} + d_{out}) \times r \times 2$, and the final sum between the main calculation and perturbation $B \times d_{out}$. Therefore, the arithmetic intensity is

$$\frac{B \times d_{in} \times d_{out} \times 2 + B \times (d_{in} + d_{out}) \times r \times 2 + B \times d_{out}}{(B \times d_{in} + B \times (d_{in} + d_{out} + 2) \times r + d_{out} \times d_{in} + B \times d_{out}) \times s}.$$

When $s = 2$ (for (b)float16) and $d_{out} = d_{in} = m$, the arithmetic intensity simplifies to

$$\begin{aligned}&\frac{Bm + 2Br + \frac{B}{2}}{B + Br(2 + \frac{2}{m}) + m + B}\\ &= \frac{m + 2r + \frac{1}{2}}{2 + r(2 + \frac{2}{m}) + \frac{m}{B}}.\end{aligned}$$

The batch size needed to achieve a desired arithmetic intensity of A is derived as follows:

$$\begin{aligned}2A + rA(2 + \frac{2}{m}) + \frac{Am}{B} &= m + 2r + \frac{1}{2}\\ \frac{Am}{B} &= m + 2r + \frac{1}{2} - 2A - rA(2 + \frac{2}{m})\\ B &= \frac{Am}{m - 2A + 2r + \frac{1}{2} - rA(2 + \frac{2}{m})}\end{aligned}$$

Note that the only difference with the critical batch size of standard batched inference is the additional $2r + \frac{1}{2} - rA(2 + \frac{2}{m})$ in the denominator. Therefore, achieving an arithmetic intensity of 300 ops/byte with $m = 8192$ and $r = 1$ requires a minimum batch size of 352, compared to 324 for standard batched inference. This means that EGGROLL can saturate compute with unique perturbations per input, unlike Gaussian matrix ES.

Note that there is an overhead of $Bm(4r + 1)$ flops relative to standard batched inference, resulting in an additional compute rate of $\frac{Bm(4r+1)}{2Bm^2} = \frac{4r+1}{2m}$, which is effectively negligible for large enough matrices.

G EGG Architecture

In the following section, we detail the design of our EGG model, which follows the high-level structure of modern pre-layernorm decoder-only language models, but replaces self-attention with a modified minGRU and standard layernorms with a custom variant to enable pure integer training. See Algorithm 2 for an overview of the forward pass of the EGG architecture.

Algorithm 2 EGG forward pass

Require: Input token $t \in \mathbb{U}_8$, input state $s \in \mathbb{I}_8^{l \times D}$, network parameters θ
Ensure: Output vector $y \in \mathbb{I}_8^D$ and output state $s' \in \mathbb{I}_8^{l \times D}$
  $s' \leftarrow \mathbb{I}_8^{l \times D}$ initialised to 0
  $y \leftarrow \text{EMBED}(\theta_{\text{emb}}, t)$
  for $i \in \{0, \ldots, l-1\}$ **do**
    $y', s'_i \leftarrow \text{GRU}(\theta_{\text{gru},i}, \text{LN}(\theta_{\text{ln1},i}, y), s_i)$
    $y \leftarrow \mathbb{I}_8(\mathbb{I}_{32}(y') + \mathbb{I}_{32}(y))$
    $y' \leftarrow \text{MLP}(\theta_{\text{mlp},i}, \text{LN}(\theta_{\text{ln2},i}, y))$
    $y \leftarrow \mathbb{I}_8(\mathbb{I}_{32}(y') + \mathbb{I}_{32}(y))$
  end for
  $y \leftarrow \text{LN}(\theta_{\text{lnout},i}, y) @ \theta_{\text{head}}^T$

G.1 Motivation

Since EGGROLL does not rely on gradients, we can explicitly design a language model architecture to be efficient and hardware-friendly at inference time. In particular, we design EGG under the following constraints to emphasise the flexibility of EGGROLL:

Pure Integer Training: On H100 systems, int8 is the fastest datatype and int8 matrix multiplication with int32 accumulation is the fastest tensor core operation. Furthermore, integer datatypes are much simpler to implement in hardware, providing massive energy savings for high-throughput systems (Horowitz, 2014). Therefore, we keep all weights in int8 and all activations in integer formats, *never* casting to floating point at any point during training. This stands in contrast to the standard approach for language model quantisation through "quantisation aware training" with backpropagation, where floating point activations are still necessary (Wang et al., 2023).

Nonlinear RNN: Modern language models use sequence-parallel architectures like Transformers and SSMs, since they enable stable gradients without backpropagation through time. However, most of these architectures cannot handle simple state tracking (Merrill et al., 2024), whereas classic recurrent networks like LSTMs and GRUs can do so with a single layer. Since EGGROLL does not require backpropagation through time, we can train on unbounded sequence lengths (Li et al., 2023) with nonlinear RNNs of broader complexity classes. Specifically, we develop a variant of the minGRU model (Heck & Salem, 2017) that performs all operations in integer formats.

Removal of all Activation Functions: Inspired by Foerster (2017), we remove all activation functions, like the rectified linear unit and hyperbolic tangent, due to the nonlinearity present in the int8 datatype. Specifically, the saturated addition of int8 values provides sufficient nonlinearity due to the implicit clipping of values to the int8 dynamic range, which evolution strategies can exploit.

G.2 Notation and Operations

We use the constant $l \in \mathbb{Z}^+$ to denote the number of layers of the model and $D = 4^d$ as the hidden dimension of the model, where $d \in \mathbb{Z}^+$.

We use $\mathbb{I}_n$ to denote an n-bit signed integer and $\mathbb{U}_n$ to denote an n-bit unsigned integer. We denote casting vector $\vec{u}$ to format $\mathbb{I}_n$ as $\mathbb{I}_n(\vec{u})$, which implicitly includes clipping to the bounds of the datatype. To ensure symmetry between positive and negative values of each datatype, we consider the value -2^{n-1} to be invalid for datatype $\mathbb{I}_n$; for instance, for 8-bit signed integers we only allows value from -127 to 127.

We use the following operations:

- $\vec{u}@M$ indicating scaled vector-matrix multiplication of $\mathbb{I}_8^n \times \mathbb{I}_8^{n,m} \to \mathbb{I}_8^m$, corresponding to int8 tensor core multiplication with int32 accumulation and scaling. The details of this operation are described in Section G.4.
- $a \cdot b$ indicates dot product with int32 accumulation, $\mathbb{I}_8^n \times \mathbb{I}_8^n \to \mathbb{I}_{32}$, and $a \odot b$ indicates the Hadamard (elementwise) product.
- Standard integer operations: $+$ for addition, $-$ for subtraction, and $\odot$ for element-wise multiplication.
- $|u|$ indicates taking the element-wise absolute value of u, $\mathbb{I}^n \to \mathbb{I}^n$.
- $\text{sign}(u)$ indicates taking the element-wise sign of u, giving 1 for positive values, -1 for negative values, and 0 for zero.
- $\text{sum}(u)$ indicates taking the sum of all elements in u (casting to $\mathbb{I}_{32}$ to prevent overflow): $\mathbb{I}^n \to \mathbb{I}_{32}$.
- $u \gg n$ indicates an elementwise bitwise right shift by n, which is typically equivalent to $2^{-n}u$. Similarly, $u \ll n$ indicates a bitwise left shift by n, which is typically equivalent to $2^n u$.
- Square-bracket indexing. For instance $M[i, j]$ extracts the element at index i in axis 0 and index j in axis 1, following the zero-based indexing convention.

G.3 Parameter Initialisation

The standard initialisation for matrix parameters in our model is rounding 16 times a sample from the standard normal, and casting to $\mathbb{I}_8$. This can be precomputed on a CPU since this is only done once at the start of training.

The egg model has the following parameters (where an additional subscript of i indicates that there is a version of this parameter for each layer of the model):

- $\theta_{\text{emb}} \in \mathbb{I}_8^{256 \times D}$, following standard initialisation.
- $\theta_{\text{head}} \in \mathbb{I}_8^{256 \times D}$, following standard initialisation.
- $\theta_{\text{lnout}} \in \mathbb{I}_8^D$, initialised to 16 for each element.
- $\theta_{\text{ln1},i}, \theta_{\text{ln2},i} \in \mathbb{I}_8^D$, initialised to 16 for each element
- $\theta_{\text{mlp},i,1} \in \mathbb{I}_8^{4D \times D}$ and $\theta_{\text{mlp},i,2} \in \mathbb{I}_8^{D \times 4D}$, following standard initialisation.
- $\theta_{\text{GRU},i,\text{[Wf,Uf,Wh,Uh]}} \in \mathbb{I}_8^{D \times D}$, following standard initialisation.
- $\theta_{\text{GRU},i,\text{[bfm bh]}} \in \mathbb{I}_8^D$, initialised to 0 for each element.

In total there are $513D + l(4D + 12D^2)$ parameters in the model.

G.4 Matrix Multiplication

Tensor cores in GPUs are able to calculate fast vector-matrix multiplications with int32 accumulation as $uM \in \mathbb{I}_{32}^m$ where $u \in \mathbb{I}_8^n$ and $M \in \mathbb{I}_8^{n \times m}$. For our purposes, we define $u@M$ as a scaled multiplication:

$$u@M := \mathbb{I}_8 \left(\frac{uM}{16\sqrt{n}} \right).$$

Note that when $n = 4^d$, the division operation just becomes a right-shift by $4 + d$, which is fast to calculate.

We choose this scaled matrix multiplication because we initialise M to 16 times standard normal samples for each element, so dividing by $16\sqrt{n}$ preserves the magnitude of u for the output. In particular, if all elements of u and M are drawn from independently from the standard normal distribution multiplied by 16, the central limit theorem tells us that the expected value per element of the output will be $256\sqrt{n}$, so dividing by $16\sqrt{n}$ preserves the standard deviation of 16.

G.5 Embedding

Our embedding function takes as input an embedding matrix $\theta_{\text{emb}} \in \mathbb{I}_8^{256\times D}$ and an input token $t \in \mathbb{U}_8$, and simply outputs the vector corresponding to that token: $\theta_{\text{emb}}[t] \in \mathbb{I}_8^D$.

G.6 Layer Normalisation (LN)

Our layer normalisation operation involves multiplying our input $u \in \mathbb{I}_8^D$ with a weight $\theta_{\text{ln}} \in \mathbb{I}_8^D$ before dividing by the mean absolute value of u.

We decide to divide by the mean absolute value of the input instead of the more common root-mean-squared since square roots are expensive on integers. Note that the $L1$ norm after dividing the input by the mean absolute value (when using real numbers) is D instead of 1, which we intentionally choose to preserve more bits of information given the limited range of $\mathbb{I}_8$.

We calculate the mean absolute value of input u as:

$$u_{\text{mav}} = \mathbb{I}_8(\text{sum}(|u|) \gg (2d)),$$

Note that we can safely cast the mean absolute value to an $\mathbb{I}_8$ without overflow given the properties of the mean of a set, though we lose precision due to truncating the fractional component.

The output of layernorm is calculated as:

$$\text{DIVIDE}(\mathbb{I}_{16}(u) \odot \mathbb{I}_{16}(\theta_{\text{ln}}), u_{\text{mav}}).$$

Since division is an expensive operation, we precompute it using a lookup table. Note that the product of two $\mathbb{I}_8$ values will always remain in the dynamic range of $\mathbb{I}_{16}$, so our lookup table will be of shape $2^{16} \times 2^8$.

G.7 MLP

Each MLP block consists of two weight parameters: $\theta_1 \in \mathbb{I}_8^{4D\times D}$ and $\theta_2 \in \mathbb{I}_8^{D\times 4D}$. Given an input $u \in \mathbb{I}_8^D$, we calculate the output as:

$$(u@\theta_1^T)@\theta_2^T.$$

Note that we do not use an activation function, because the @ operation is already nonlinear due to the saturated conversion from $\mathbb{I}_{32}$ to $\mathbb{I}_8$

G.8 GRU

Each GRU block accepts an input vector and state $u, s \in \mathbb{I}_8^D$ consists of 6 weight parameters: $\theta_{\text{Wf}}, \theta_{\text{Uf}}, \theta_{\text{Wh}}, \theta_{\text{Uh}} \in \mathbb{I}_8^{D\times D}$ and $\theta_{\text{bf}}, \theta_{\text{bh}} \in \mathbb{I}_8^D$.

Using these weight matrices, we calculate the following vectors:

$$\begin{aligned}
f &= \sigma(\mathbb{I}_8(\mathbb{I}_{32}(u@\theta_{\text{Wf}}^T) + \mathbb{I}_{32}(s@\theta_{\text{Uf}}^T) + \mathbb{I}_{32}(\theta_{\text{bf}}))),\\
\hat{f} &= \mathbb{I}_8(((\mathbb{I}_{32}(f) + 127) \odot \mathbb{I}_{32}(s)) \gg 8),\\
\hat{h} &= \phi(\mathbb{I}_8(\mathbb{I}_{32}(u@\theta_{\text{Wh}}^T) + \mathbb{I}_{32}(\hat{f}@\theta_{\text{Uh}}^T) + \mathbb{I}_{32}(\theta_{\text{bh}}))),\\
h &= s + \mathbb{I}_8(((\mathbb{I}_{32}(f) + 127) \odot (\mathbb{I}_{32}(\hat{h}) - \mathbb{I}_{32}(s))) \gg 8),
\end{aligned}$$

where h is the output and the new hidden state. In the typical GRU, σ stands for the sigmoid function while ϕ stands for the hyperbolic tangent, but we find that setting these as identity operations is sufficient due to the nonlinearity already present in the clipped addition. One can view this clipped addition operation as scaled and shifted version of the "hard" tanh and sigmoid operators.

To explain why we perform these operations, we can analyse this relative to the original GRU. The f vector for the standard GRU has all elements between 0 and 1 due to the sigmoid, but our elements are between -127 and 127. Therefore, to calculate $\hat{f}$ (which is typically just $f \odot s$), we first add 127 to f, getting the range between 0 and 254 before multiplying by s before bit-shifting right by 8 again to bring our values back to the $\mathbb{I}_8$ dynamic range. We apply similar logic to calculate the final h, which is typically just $h = s + f \odot (\hat{h} - s)$ but needs to be rescaled to keep the int8 dynamic range.

G.9 Fitness Calculation in Integer Types

The "fitness" used in language model pretraining is the log-likelihood of correctly generating the next token, treating the outputs of the language model as logits (unnormalised log probabilities). If $t' \in \mathbb{U}_8$ is the next token to predict and $y \in \mathbb{I}_8^{256}$ are the logits, we can calculate the log likelihood as follows:

$$\begin{aligned} y' &= \mathbb{I}_{32}(y) + 128, \\ o &= y'[t'] - \text{LOG2}[\text{sum}(\text{EXP2}[y'])], \end{aligned}$$

where o is the loss for one token. We implement EXP2 and LOG2 as lookup tables, where

$$\begin{aligned} \text{EXP2}[i] &= 16 \times 2^{i/16}, \\ \text{LOG2}[i] &= 16 \times \log_2(i/16). \end{aligned}$$

Note that each element in EXP2 for any $\mathbb{U}_8$ input requires at most 20 bits, so the sum of exponents across all possible choices is at most 28 bits, meaning we have to precompute LOG2 for 2^{28} values.

H EGG Pretraining with Integer EGGROLL

The core ideas of EGGROLL still apply in this integer-based training setting, but we have to make some modifications to ensure it only uses integer operations.

H.1 Adding EGGROLL Perturbations

For parameter $\theta \in \mathbb{I}_8^{m \times n}$ that represents a matrix multiplication, we first sample rank-1 perturbation vectors for each index in the batch: $A \in \mathbb{I}_8^m$ and $B \in \mathbb{I}_8^n$. We sample these vectors from the standard random normal multiplied by 16 and rounded to the nearest $\mathbb{I}_8$ (clipping if necessary). To prevent the use of floating-point arithmetic on the accelerator, we pre-generate a large matrix of these random values, randomly indexing into it to get the perturbation vectors.

Given an input $u \in \mathbb{I}_8^n$, instead of calculating $u @ \theta^T$, we calculate

$$\mathbb{I}_8 \left(\frac{u\theta^T + ((u \cdot B)\mathbb{I}_{32}(A) \gg (4 + \hat{\sigma}))}{16\sqrt{n}} \right).$$

The value of $\hat{\sigma}$ is a hyperparameter, related to the σ in the main paper as $\sigma = 2^{-\hat{\sigma}}$. Note that the batched forward pass remains efficient since it still simply performs a batched vector-vector dot product in int8 (with int32 accumulate) and a batched vector-scalar product in int32.

We apply this same logic to the embedding matrix, since we can interpret $\theta[t]$ as one_hot$(t)\theta$ and still apply our rank-1 updates in that context. In practice, this means replacing $u \cdot B$ with $B[t]$.

H.2 Fitness Shaping

We employ a simple fitness shaping scheme based on antithetical pairs. Specifically, given raw fitnesses s^+, s^-, for the positive and negative sample of the antithetical pair respectively, the transformed fitness for the noise is:

$$\text{sign}(s^+ - s^-),$$

Note that the only possible values for the fitness after shaping are $\{-1, 0, 1\}$.

H.3 Parameter Update

For parameter $\theta \in \mathbb{I}_8^{m \times n}$ that represents a matrix multiplication (or embedding vector), suppose the sampled batch of rank-1 perturbation vectors are $A \in \mathbb{I}_8^{N \times m}$ and $B \in \mathbb{I}_8^{N \times n}$, and let the fitnesses after shaping be $F \in \mathbb{I}_8^N$. Then we calculate an intermediate value $E \in \mathbb{I}_{32}^{m \times n}$ as:

$$E = (\text{diag}(F)A)^T B.$$

We use E to determine if each element of θ should be increased or decreased. In particular, when the absolute value of E is above a pre-specified threshold we move θ by one discrete bin in the direction of the sign of E. Since there are only 255 unique values for each element in $\mathbb{I}_8$, restricting updates to single bins improves stability without compromising the ability for a parameter to get to any other value with relatively few updates. In particular, we have a real-valued hyperparameter, $\alpha \in (0, 1)$ such that the threshold equals

$$\mathbb{I}_{32}\left(16 \times \Phi^{-1}\left(\frac{1-\alpha}{2}\right)\right) \times 16\sqrt{N},$$

where Φ is the normal cumulative distribution function. Note that this threshold can be precalculated on a CPU. We observe that α approximately equals the fraction of parameters that are updated at each step.

We currently do not incorporate any momentum or other optimiser states, but this remains critical future work to improve the speed of convergence for pure integer training.

Across model sizes and population size, we find that setting $\hat{\sigma}$ to 4 and letting α decay over training steps as $\frac{1}{.015t+1}$ gives consistently strong results.

I EGG Ablations

In our main experiments, we use a fixed data batch size of 16 sequences for population sizes 2 and powers of 4 ranging from 4 to $4^{10} = 1048576$. In this section, we vary the batch size by powers of 4, ranging from 4 to $4^5 = 1024$, while varying population size by powers of 4 from 16 to 1048576. When the batch size, b is greater than half of the population size, N, we give each antithetical pair $\frac{2b}{N}$ sequences, functionally giving a cleaner fitness signal to each member of the population. This also means that the number of parallel "inferences" required is $\max(2b, N)$.

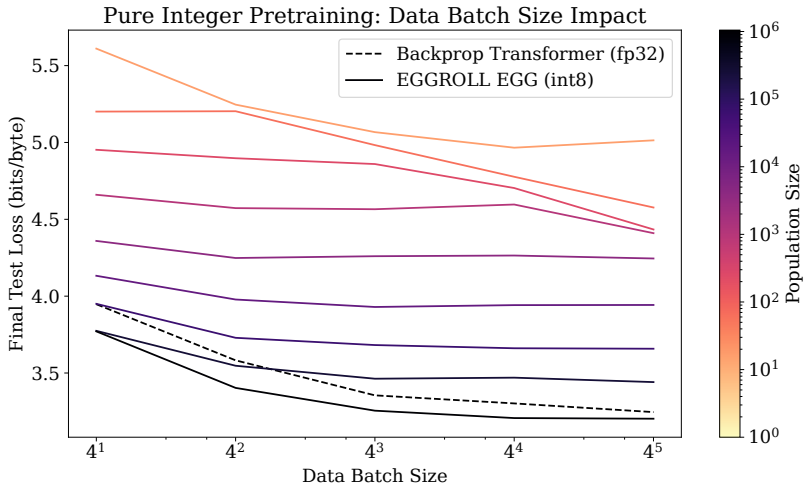


Figure 8: Test loss curves when varying data batch size and population size.

In Fig. 8, we observe that the final test loss for each population size is relatively constant beyond a specific data batch size threshold. At the top right of the figure, we observe a decrease in loss for small population sizes after $b > \frac{N}{2}$, which is an artifact of the increased compute usage necessary to use the full data batch. Ignoring this artifact, the minimum batch size for near-optimal performance at a given population size N appears to be $\frac{N}{4^6}$. We see that large population sizes need larger data batches for improved performance, since a batch size of 4 results in nearly identical performance for population sizes $4^9 = 262144$ and $4^{10} = 1048576$, but this diverges as data batch size increases.

J Distributed EGGROLL Framework

To facilitate the large-scale experiments, where we scale population sizes beyond 1M, we develop a lightweight distributed training framework designed to minimise network overhead.

J.1 Base-3 Fitness Packing and Bandwidth Efficiency

A key bottleneck in distributed training is the communication of gradients or results. We address this via a custom base-3 packing scheme for fitness vectors. Since workers evaluate perturbations in antithetic pairs, the raw signal is discretised into ternary values $\{+1, 0, -1\}$. These are mapped to $\{0, 1, 2\}$ and packed five at a time into a single byte:

$$byte = \sum_{i=0}^{4} v_i \cdot 3^i$$

This yields an effective bitrate of 1.6 bits per value (near the $\log_2 3 \approx 1.585$ theoretical limit). Consequently, the network payload per chunk is approximately $52 + \text{chunk\_size}/10$ bytes, rendering bandwidth usage independent of model size.

J.2 System Architecture

The system employs a Coordinator-Worker topology. The Coordinator maintains the global state and assigns population chunks to Workers. Workers calculate fitness on GPU, apply signal shaping (chunk mean filtering, adaptive thresholding), and return only the packed ternary fitness, minimising traffic significantly compared to standard gradient transmission.

K Fine-tuning of Integer Quantised Models

K.1 Quantisation Procedure

To maximise population throughput and reduce device memory during EGGROLL fine-tuning, we represent the large matrix-multiplication parameters of RWKV in an int8 weight format while keeping non-matmul parameters (e.g., small biases / bookkeeping tensors) in floating point, bf16. Following Jacob et al. (2017), for each weight matrix $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, we use symmetric per-channel int8 quantisation with an absmax scale. For each output channel we first compute:

$$s_i = \max\left(\frac{\max_j |W_{i,j}|}{127}, \epsilon\right),$$

where ϵ is some small scalar. Then, we store each s_i in bf16, and quantise weights as

$$Q_{i,j} = \text{clip}\left(\text{round}\left(\frac{W_{i,j}}{s_i}\right), -127, 127\right) \in \mathbb{I}_8.$$

Every matrix parameter is stored as a dictionary containing the quantised weight matrix Q, the scale parameters per channel $\{s_i\} \forall i \in 1, \ldots, d_{\text{out}}$ and an input scale factor s_x in bf16 precision. At runtime, the forward pass is computed by scaling the input vector by s_x and the quantised matrix Q with the scales per channel, $[s_1, \ldots, s_{d_{\text{out}}}]$,

$$x_{n+1} = (x_n \odot s_x)^T (W \odot [s_1, \ldots, s_{d_{\text{out}}}]).$$

K.2 Integrating integer-quantised EGGROLL with Adam

EGGROLL performs black-box (ES) optimisation directly over the parameter representation used in the forward pass, including integer quantised weights. We integrate this with the Adam optimiser (Kingma & Ba, 2014) by maintaining Adam's moment estimates in bf16, while enforcing that all quantised tensors remain on the int8 lattice.

ES gradients. EGGROLL estimates gradients via antithetic ES perturbations and score-weighted averaging. This yields a bf16 gradient estimate for: (i) floating-point parameters (when present), (ii) quantised matrix parameters via a low-rank perturbation pathway, and (iii) scale parameters $\{s_i\}\forall i \in 1, \ldots, d_{\text{out}}$ and s_x via explicit scale perturbations. We then pass these gradients to Adam (Optax), which produces an update tensor u for each parameter leaf.

Adam updates for int8 tensors (discretised). For integer parameters (notably int8), Adam produces a real-valued proposal u (stored in bf16). Since the parameter itself must remain int8, we convert this proposal into a sparse unit-step update using a normalised thresholding rule. Let $Q \in \mathbb{Z}_8^{m\times n}$ be an int8 tensor and $u \in \mathbb{R}^{m\times n}$ be Adam's proposed update. We compute a per-tensor z-score normalisation

$$z = \frac{u - \mu(u)}{\sigma(u) + 10^{-8}},$$

then apply a threshold τ to form the integer step

$$\Delta = \text{sign}(z) \cdot \mathbb{1}\{|z| \geq \tau\} \in \{-1, 0, +1\}^{m\times n}.$$

Finally we update by unit increments and clip to the valid int8 range:

$$Q \leftarrow \text{clip}(Q + \Delta,\ -127, 127).$$

Intuitively, Adam supplies a magnitude- and history-aware proposal, while the discretisation enforces the integer constraint and yields a stable, sparse update pattern (only entries with sufficiently large normalised updates are modified).

Memory considerations. We store Adam's optimiser state (moments) in bf16 for all array-valued leaves to reduce memory footprint, while keeping scalar bookkeeping in full precision. This keeps the dominant memory cost of optimisation close to that of the parameters themselves, which is particularly important when fine-tuning large models with large ES populations.

Model distillation. We distil a non-quantised model into the quantised RWKV-7 model by matching the two distributions in teacher forced examples from GSM8k. More specifically, the fitness for a given set of parameters, μ_i, is computed as follows:

$$f_{\mu_i}(x_{1:T}) = \sum_{t=1}^{T} \text{KL}\left(p_t || q_t(\cdot; \mu_i)\right),$$

where $x_{1:T}$ is a subsequence of tokens taken from the solutions of GSM8K and $\text{KL}\left(p_t || q_t(\cdot; \mu_i)\right)$ is the Kullback-Leibler divergence between the distribution of the non-quantised model, p_t, and the distribution of the quantised model q_t over the vocabulary at token t.

L Fine-tuning Pretrained Transformer LLMs with Verifiable Rewards

This section describes compares EGGROLL to standard RL from Verifiable Rewards (RLVR). We first describe our experimental results, before including details of the infrastructure used to run these experiments.

L.1 Results

Here we demonstrate that EGGROLL can be used to fine-tune pre-trained LLMs on verifiable rewards. We use the vLLM library Kwon et al. (2023) for efficient inference. More infrastructure detail is given in Section L.2.

We first fine-tune the Qwen3-4B-Base model Yang et al. (2025) on the DeepScaleR Agentica Organization et al. (2025), a dataset of 40k maths questions. As in standard RLVR, the model generates a chain-of-thought (CoT) followed by a final answer. Fitness is then simply calculated by extracting the final answer and comparing it to the ground truth answer Shao et al. (2025). We evaluate performance on MATH500 Hendrycks et al. (2021), OlympiadBench He et al. (2024), AIME24 Balunović et al. (2026), AMC, and MinervaMath Lewkowycz et al. (2022). Training curves are shown in Figure 9. Here we see that fine-tuning with EGGROLL significantly

improves performance over the base model. In Section L.1 we show final accuracies with EGGROLL and with the equivalent RL experiment. The RL values are taken from Liu et al. (2025), and we match all the relevant shared hyperparameters and setup, such as maximum response length and prompt phrasing. We see that EGGROLL is able to match the RL optimisation with very minimal hyperparameter tuning, a LoRA rank of 1 and a moderately small population size of 2048. Full hyperparameter details are given in Table 3.

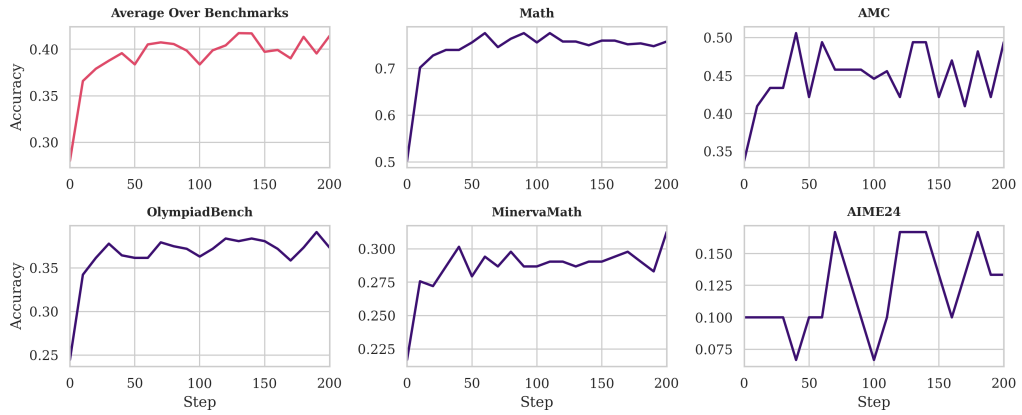


Figure 9: Training curves for fine-tuning Qwen3-4B-Base on the DeepScaleR math dataset. Similar to RL from Verifiable Rewards (RLVR), we see that optimising with EGGROLL is able to improve chain-of-thought reasoning performance on a range of math benchmarks.

	MATH500	OlympiadBench	AIME24	AMC	MinervaMath	**Average**
Qwen3-4B-Base	50.2	24.4	10.0	33.7	21.7	28.0
+EGGROLL	**75.8**	**37.3**	13.3	**49.4**	31.3	**41.4**
+RL	67.4	33.5	**16.7**	**49.4**	**40.1**	**41.4**

Table 1: Final test accuracies when training on the DeepScaleR dataset to optimise verifiable rewards with EGGROLL and RL. We see that EGGROLL significantly boosts performance from the base model and is able to match the equivalent RL experiment.

Since EGGROLL can be used to optimise non-differentiable objectives we next try optimising for pass@k. While zero-shot (pass@1) is differentiable, the pass@k objective is not as it depends on multiple samples from the model. This means it cannot be optimised easily with RL. In Figure 10 we fine-tune the Qwen3-1.7B model on the DeepScaleR dataset with a population size of 256, LoRA rank 1, and $K = 4$. We see that EGGROLL successfully optimises both the pass@1 (differentiable) and pass@k (non-differentiable) objectives. In Figure 10 *(right)* we plot the number of distinct answers in 4 samples from the model. We see then when optimising for pass@k the answer diversity sampled by the model increases over training, whereas when optimising for zero-shot (pass@1) the model collapses towards a single final answer.

L.2 Training Infrastructure for Large-Scale Transformer LLMs

EGGROLL facilitates the fine-tuning of transformer-based LLMs at scale. We achieve this by repurposing the `vLLM` inference engine, leveraging its high-throughput kernel implementations and native support for multi-LoRA serving. The system utilises vLLM's native Tensor Parallelism (TP) to shard the model weights across the GPUs within a node, while cross-node parallelisation is employed for the concurrent evaluation of the LoRA population.

To render ES-based optimisation feasible and efficient across a wide range of model sizes, we implement several critical systems-level optimisations:

Custom `WorkerExtension` and Sharding-Aware Updates By implementing a custom `WorkerExtension`, we effectively convert the vLLM inference engine into a training-capable runtime. This extension allows the optimisation logic to reside within the GPU process space, enabling direct,

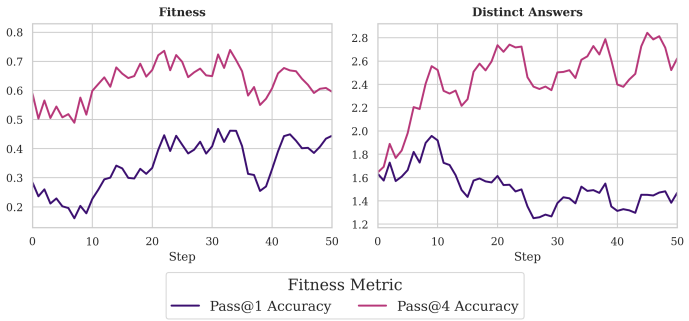


Figure 10: Using EGGROLL to optimise non-differentiable objectives. *Left*: Fitness curves comparing training with pass@1 (differentiable) versus pass@k (non-differentiable), where $K = 4$. *Right*: The mean number of unique final answers generated per 4-sample set. We observe that when optimizing for pass@k increases answer diversity, whereas optimizing for zero-shot accuracy (pass@1) reduces it.

in-place manipulation of the model's weights. A significant complexity of this integration is vLLM's internal tensor parallelism, which frequently fuses weights (e.g. combining `q_proj`, `k_proj`, and `v_proj` into a single `qkv_proj` tensor). Our update mechanism is explicitly "sharding-aware"; it constructs a dictionary which maps individual LoRA updates to the specific fused slices held by each local GPU rank. This ensures that the global ES update is mathematically consistent across all distributed shards.

Layer-wise Memory Management To prevent out-of-memory (OOM) errors during the update phase, the `WorkerExtension` performs the ES weight application in a streaming, layer-wise fashion. By processing one layer at a time and clearing temporary buffers, the memory overhead of the update remains independent of the total model depth. This allows for the fine-tuning of models of very different sizes with a VRAM footprint barely exceeding that of standard inference.

Direct GPU-to-GPU Weight Synchronization After computing the ES update on the primary rank, we broadcast the updated parameters to all model instances using NCCL via `PyNcclCommunicator`. This approach bypasses CPU-based communication and instead uses hardware interconnects to transfer weights directly between GPUs, preventing synchronization from becoming a bottleneck when scaling to more nodes.

Meta-Device Blueprinting To initialise models that exceed the physical RAM of the control node, we employ Meta-Device Initialisation. Using `accelerate`'s `init_empty_weights`, we instantiate a "meta" version of the model to derive the weight shapes and sharding requirements for the LoRA adapters. This allows the system to generate a complete parameter blueprint for models of arbitrary size without ever allocating the full weight tensors in system memory.

`vLLM` Engine Settings Throughout the different experiments with `vLLM`, we use the following engine settings. These generally allow for high throughput across model sizes (e.g. at least 800 tokens/second), but we haven't performed hyperparameter sweeps, so potentially faster, more memory-efficient settings may be used for improved results.

Parameter	Value
Tensor parallel size	2,4
Data type	auto
Enable prefix caching	True
Enforce eager execution	True
Enable LoRA	True
Max LoRAs	$\lceil$population_size/num_engines$\rceil$
GPU memory utilisation	0.90
Max number of sequences	384
Max model length	$\max(1024, 512 + \text{max\_tokens})$
Max batched tokens	prompt_batch_size $\times$ 1024
Load format	auto

Table 2: `vLLM` engine configuration parameters to allow for high throughput EGGROLL training on large-scale transformer LLMs.

Parameter	Value
Population size	256, 2048
Sigma	0.001
Learning Rate	0.001
Max Response Length	4096
Temperature	0.0, 0.7
Samples Per Prompt	1, 4
Pass at K	True, False
LoRA Rank	1
LoRA Reuse Steps	4

Table 3: Hyperparameters for the verifiable reward transformer fine-tuning experiments in Section L.1.

M Fine-tuning Time Series Foundation Model: High-Frequency Trading

The preceding experiments demonstrate the effectiveness of EGGROLL on natural language reasoning tasks. We now investigate whether EGGROLL can effectively fine-tune pretrained foundation models on a fundamentally different data modality: structured time series. We focus on high-frequency trading (HFT) for two reasons. First, HFT generates data at an unprecedented scale. The S&P 500 constituents alone produced approximately 3.8 trillion tokens of order flow data between 2016 and 2021, comparable to the largest natural language corpora. Second, the domain presents a well-defined downstream task (order execution) with a natural reward signal: the realised profit and loss, also known as PnL, making it amenable to fine-tuning via evolution strategies.

Order execution takes place in limit order books (LOBs), which are the mechanism upon which modern financial exchanges operate (Gould et al., 2013; Bouchaud et al., 2018). They allow market participants to submit limit orders that specify the details of intended transactions. Specifically, each limit order contains the order type, direction, price, and quantity. The continuous stream of these orders is known as the order flow. LOBs aggregate the limit orders that have not been matched yet. Unlike natural language, where tokens are purely symbolic, order flow messages comprise both categorical values (e.g., order type, direction) and numerical values (e.g., price, quantity) in which magnitude carries semantic meaning. This structure provides a distinct test of EGGROLL's ability to fine-tune foundation models on time series sequential data.

A central objective in this context is order execution, which consists of buying or selling a specified quantity of an asset within a given time window. The goal is to maximise profit by transacting at favourable prices. In prior reinforcement learning approaches to this problem, the action space is usually simplified (Frey et al., 2023; Mohl et al., 2025; Ning et al., 2021). In contrast, we aim to give the model full flexibility in choosing

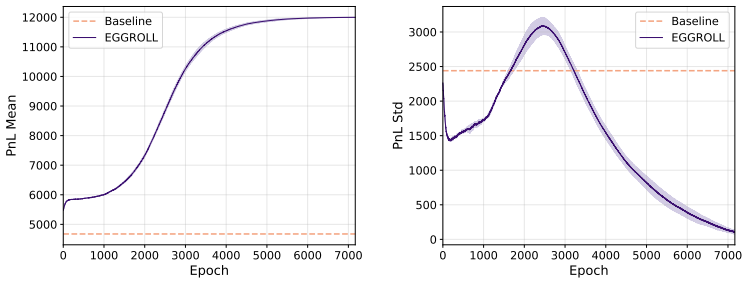


Figure 11: Training curves for order execution with EGGROLL. **Left**: Mean PnL over training epochs for the baseline ($\sigma = 0$, orange dashed) and EGGROLL ($\sigma = 0.01$, blue solid). **Right**: PnL standard deviation over training epochs. Shaded regions indicate the interquartile range across runs.

limit orders, i.e., to freely choose the order type, direction, price, and quantity. We achieve this by tokenising the limit order book messages and providing the model with a token-level action space.

Foundation models have recently been used to generate synthetic order flow (Nagy et al., 2023; Li et al., 2025) and have been shown to replicate realistic market behaviour (Nagy et al., 2025) through next-token prediction. We therefore first pretrain a foundation model on tokenised limit order book messages, and then fine-tune it using EGGROLL for the order execution task. The pretraining follows the approach of Nagy et al. (2023): we employ an S5 model architecture (Smith et al., 2023) that generates next-token probabilities, with cross-entropy as the training loss. The pretraining is conducted on the LOBSTER data set (Huang & Polak, 2011) for the Google stock (GOOG) in 2022, which contains around 25B tokens.

Subsequently, we fine-tune the model using EGGROLL. The training parameters are summarised in Table 4. The task is to execute a sell order of $Q = 30$ shares within a horizon of $T = 10$ steps. In each episode, the LOB is initialised based on a LOB snapshot followed by 10 warm-up background messages. In each step, the population members generate their messages, which are then followed by 50 real background data messages. The orders are executed using the Jax-LOB (Frey et al., 2023) simulator. We perform the fine-tuning on a fixed time window for GOOG in January 2023. Following (Galim et al., 2025), we apply LoRA with rank 4 on all projection matrices while freezing SSM parameters and layer norms. Performance is evaluated using PnL based on the executed prices and the initial mid price. Specifically, for a sell task of total quantity Q, the PnL is computed as

$$\sum_{i=1}^{N} q_i p_i - Q\, P_{\text{mid}}^{\text{init}},$$

where q_i and p_i denote the quantity and price of the i-th executed trade and $P_{\text{mid}}^{\text{init}}$ is the mid-price at the beginning of the execution window. If the agent does not execute the entire quantity by the end of the episode, an automatic market order is submitted selling the remaining quantity. To improve robustness to outliers, fitness is defined as a rank-based transformation of the PnL. Specifically, for a population of size M, the PnL values

$$\mathcal{P} = \{\text{PnL}_1, \ldots, \text{PnL}_M\},$$

are mapped to the interval $[-0.5, 0.5]$, where $\text{rank}(\text{PnL}_i) \in \{0, \ldots, M-1\}$ denotes the rank of the i-th individual's PnL:

$$F_i = \frac{1}{2} - \frac{\text{rank}(\text{PnL}_i)}{M-1},$$

Training curves over 6,500 epochs are shown in Figure 11. The baseline policy ($\sigma = 0$), corresponding to the pretrained model, achieves a mean PnL of approximately 4,700. In contrast, EGGROLL fine-tuning ($\sigma = 0.01$) improves the mean PnL to around 12,000, corresponding to a roughly 155% improvement over the baseline. The right panel of Figure 11 depicts the PnL standard deviation during fine-tuning: it initially increases to around 3,100 during the first 2,500 epochs, which corresponds to an exploration phase where the population tries out diverse strategies, before decreasing to approximately 400 by the end of training, indicating that the population concentrates around a high-performing policy.

Hyperparameter	Value
Model	LOBS5-360M
Parallel generations per GPU	2,048
Total parallel generations	65,536
LoRA rank	4
Sigma	0.01
Learning rate η	0.001
Epochs	6,500

Table 4: Model and EGGROLL fine-tuning settings for high-frequency trading.

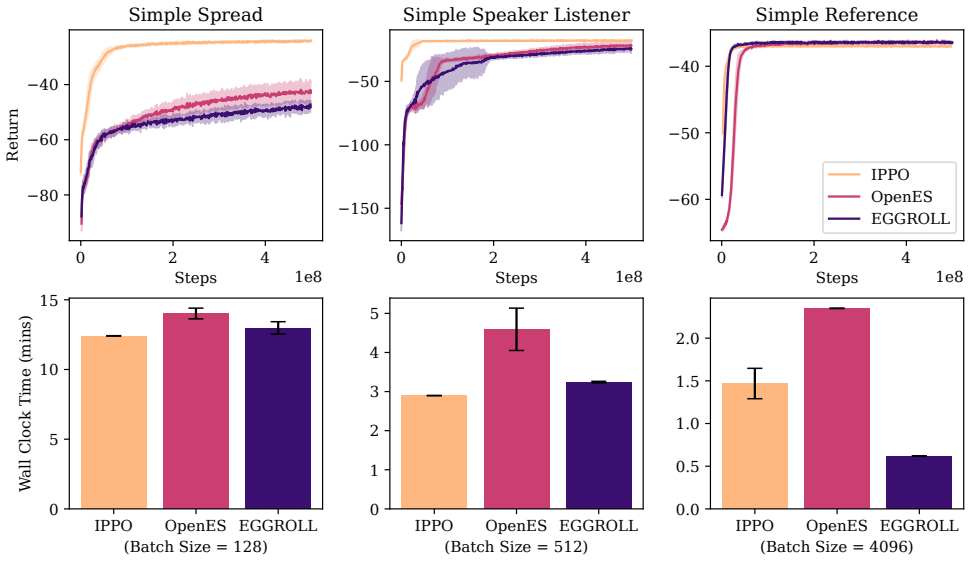


Figure 12: Training curves and wall clock times for cooperative Multi Particle Environments. Hyperparameter optimisation yielded equal batch sizes for all algorithms on the same environment. All EGGROLL runs used rank 1 perturbations. Shaded regions are standard errors of mean values.

N Experimental Details

N.1 Multi Agent Reinforcement Learning Experiments

Table 5: Hyperparameter Ranges Used in MPE Sweeps for EGGROLL and OpenES

Hyperparameter	Values
activation	pqn, tanh
pop_size	128, 512, 1024, 2048, 4096
learning_rate	0.01, 0.05, 0.1, 0.5
lr_decay	0.3, 0.7, 1.0
sigma	0.1, 0.2, 0.3, 0.4, 0.5
rank_transform	true, false

Table 6: Hyperparameter Ranges Used in MPE Sweeps for IPPO

Hyperparameter	Values
activation	relu, tanh
pop_size	128, 512, 1024, 2048, 4096
learning_rate	5e-5, 1e-4, 2.5e-4, 1e-3
entropy_coef	0.001, 0.005, 0.01

Table 7: MPE Simple Spread v3

Hyperparameter	eggroll	open_es	ippo
activation	tanh	tanh	tanh
deterministic_policy	true	true	false
learning_rate	0.01	0.01	0.001
lr_decay	0.7	0.7	linear
layer_size	64	64	64
n_layers	3	3	3
pop_size	128	128	128
optimizer	adamw	adamw	adam
rank	1	1	-
rank_transform	false	false	-
sigma	0.5	0.5	-
n_minibatches	-	-	4
update_epochs	-	-	4
gamma	-	-	0.99
gae_lambda	-	-	0.95
epsilon_clip	-	-	0.2
entropy_coef	-	-	0.01
value_coef	-	-	0.5
max_grad_norm	-	-	0.5

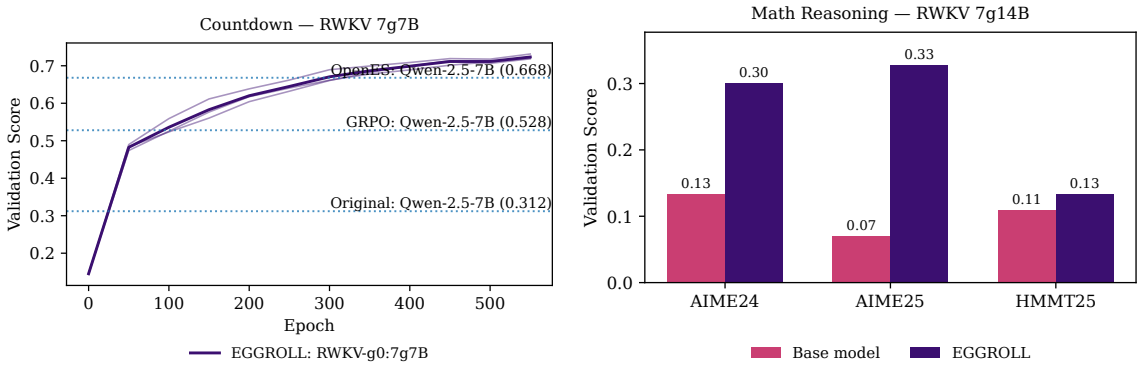
Table 8: MPE Simple Speaker Listener v4

Hyperparameter	eggroll	open_es	ippo
activation	tanh	tanh	relu
deterministic_policy	true	true	false
learning_rate	0.01	0.01	0.001
lr_decay	0.7	0.3	linear
layer_size	64	64	64
n_layers	3	3	64
pop_size	512	512	512
optimizer	adamw	adamw	adam
rank	1	1	-
rank_transform	true	true	-
sigma	0.5	0.5	-
n_minibatches	-	-	4
update_epochs	-	-	4
gamma	-	-	0.99
gae_lambda	-	-	0.95
epsilon_clip	-	-	0.2
entropy_coef	-	-	0.005
value_coef	-	-	0.5
max_grad_norm	-	-	0.5

Table 9: MPE Simple Reference v3

Hyperparameter	eggroll	open_es	ippo
activation	pqn	tanh	relu
deterministic_policy	true	true	false
learning_rate	0.01	0.01	0.001
lr_decay	0.3	0.3	linear
layer_size	64	64	64
n_layers	3	3	3
pop_size	4096	4096	4096
optimizer	adamw	adamw	adam
rank	1	1	-
rank_transform	false	true	-
sigma	0.1	0.3	-
n_minibatches	-	-	4
update_epochs	-	-	4
gamma	-	-	0.99
gae_lambda	-	-	0.95
epsilon_clip	-	-	0.2
entropy_coef	-	-	0.01
value_coef	-	-	0.5
max_grad_norm	-	-	0.5

We train on three cooperative Multi Particle Environments (MPEs) (Lowe et al., 2017) implemented in JaxMARL (Rutherford et al., 2024) with feed-forward networks of width 64 and depth 3, performing Bayesian hyperparameter optimisation for each environment and algorithm. All runs were executed on NVIDIA A100-SXM4-40GB GPUs. We find that the optimal batch size is consistent across algorithms on the same environment. Figure 12 shows that EGGROLL with rank 1 trains up to 2.4 times faster than OpenES for large batch sizes while staying competitive in performance.



(a) (b)

Figure 13: (a) Comparison of our finetuned RWKV 7G 7 billion parameter model using 8 GPUS with the results reported by Qiu et al. (2025) on similarly sized Qwen models. (b) Performance of our finetuned RWKV 7G 14 billion parameter model on hard reasoning tasks using 32 GPUs for 12 hours. The model was trained using the DeepScaleR dataset and the best checkpoint was chosen by evaluating on AIME24. Due to the size of the model we were not able to run similar baseline experiments using GRPO.

N.2 Reasoning Fine-tuning Experiments: Countdown

We ran a Bayesian hyper-parameter sweep (Snoek et al., 2012) for both GRPO and EGGROLL and used the best set found to run the experiments in figure 4b. For GRPO we swept over sampling temperature and learning rate, whereas for EGGROLL we swept over the standard deviation of the ES sampling (σ) and the learning rate scale. The best hyper-parameters found are detailed on tables 10 (EGGROLL) and 11 (GRPO). All of the experiments run in 8 hours on a NVIDIA H200 GPU.

Hyperparameter	Value
Model	RWKV 7g1.5B
Optimiser	Gradient descent
ES standard deviation σ	7×10^{-4}
Rank r	1
Learning-rate scale η_{scale}	0.125
Population size	256
Parallel generations per GPU	1536
Prompts per epoch	6
Generation / thinking length	1000 tokens
Train / val temperature	0 / 0
Parallel validations	128

Table 10: Key hyperparameters for EGGROLL training on Countdown with FastRWKV-7g1.5B.

We also run an experiment where we increase the number of GPUs to 8 and use a bigger model, RWKV 7g7B, on the Countdown task, allowing for stronger final performance. Notably, we compare to the results reported by Qiu et al. (2025) on Countdown. Figure 13a shows that starting from our significantly weaker model (RWKV 7g7B v.s. Qwen 2.5-7B), we are able to train to a higher validation accuracy (72.9%), v.s. the ones reported for training with GRPO (52.8%) and Open ES (66.8%). Qiu et al. (2025) do not report the wall clock time or the hardware used for their experiments which makes it difficult to establish a fair comparison.

Hyperparameter	Value
Model	RWKV 7g1.5B
Optimiser	Radam
Learning rate η	3×10^{-6}
Generations per prompt G	8
Parallel generations per GPU	64
Prompts per epoch	8
Generation length	1000 tokens
Number of minibatches	4
PPO clip parameter ϵ_{clip}	0.2
Train / val temperature	1 / 0
Parallel validations	128

Table 11: Key hyperparameters for GRPO training on Countdown with AssociativeScanRWKV-7g1.5B.

N.3 Reasoning Fine-tuning Experiments: GSM8K

We used the hyper-parameters found for Countdown as a starting point and reduced the learning rates for both GRPO and EGGROLL using linear search until we found the best performing one on the validation set. Our experiments for GSM8K run on 8 NVIDIA H200 GPUS for 8 hours each. We also increase the standard deviation, σ, parameter for ES (from 7×10^{-4} to 2×10^{-3}) as the significantly bigger population sizes (8096 v.s. 512) allow for much more stable training and aggressive exploration.

Hyperparameter	Value
Model	RWKV 7g7B
ES standard deviation σ	2×10^{-3}
Rank r	1
Learning-rate scale η_{scale}	0.06
Generations per prompt G	512
Parallel generations per GPU	1024
Total parallel generations	8192
Prompts per epoch	16
Generation length	1000 tokens
Noise reuse factor	1
Freeze non-LoRA params	True
Train / val temperature	0 / 0
Parallel validations	128

Table 12: Key hyperparameters for multi-GPU EGGROLL training on GSM8K with FastRWKV-7g7B.

N.4 Reinforcement Learning Experiments

Next, we compare the performance of EGGROLL against standard OpenES as implemented in Salimans et al. (2017) on reinforcement learning tasks. Given the small network sizes, we can use OpenES at this scale, but as network sizes increase, the use of vanilla OpenES becomes computationally infeasible. We use the standard formulation of simply optimising for the final return in the environment. For both EGGROLL and OpenES, we perform hyperparameter optimisation (HPO) separately for each environment. For each algorithm–environment pair, we define plausible ranges for all key hyperparameters based on prior work and preliminary experiments. We then perform 20 random search trials, where each trial corresponds to a single training run with a randomly sampled hyperparameter configuration. Each configuration is evaluated based on the final return achieved by the mean policy parameters at the end of training. After all trials, we select the configuration that yields the highest final return. Using this best configuration, we then run 10 independent seeds to evaluate performance and report the mean and standard error of the mean across these seeds.

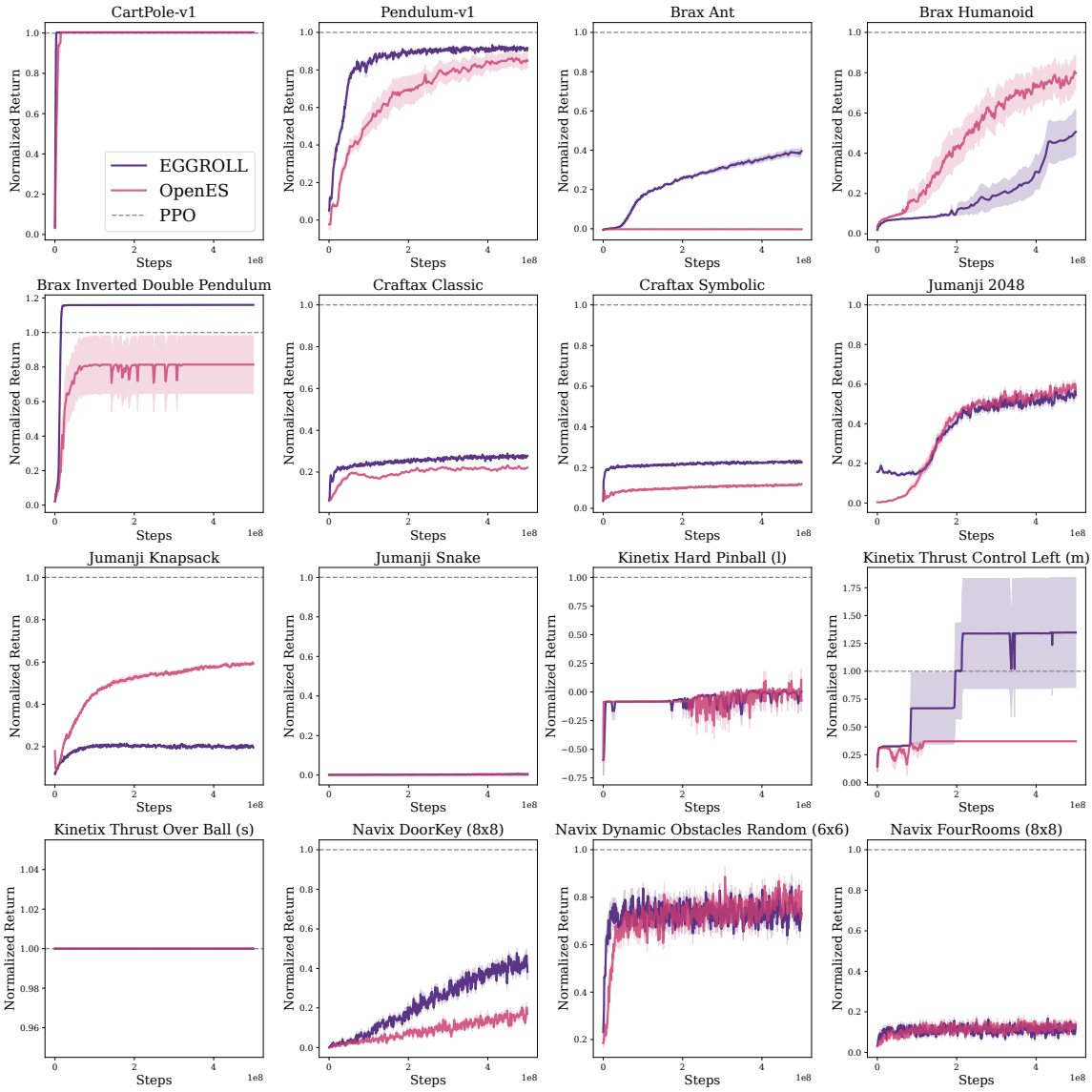


Figure 14: Comparison of reinforcement learning results: Mean returns for each environment and algorithm across 10 random seeds. The returns are evaluated using the mean of the parameters. HPO was conducted for each algorithm/environment pair. The shaded region is the standard error of the mean.

Hyperparameter	Value
Model	RWKV 7g7B
Learning rate η	1×10^{-6}
Generations per prompt G	8
Parallel generations per GPU	32
Total parallel generations	256
Prompts per epoch	32
Generation length	1000 tokens
Number of minibatches	16
Number of workers (processes)	8
PPO clip parameter ϵ_{clip}	0.2
Train / val temperature	1 / 0
Parallel validations	128

Table 13: Key hyperparameters for multi-GPU GRPO training on GSM8K with AssociativeScanRWKV-7g7B.

Hyperparameter	Value
Model	RWKV 7g7B
Optimiser	EGGROLL (Quantised))
ES standard deviation σ	0.4
Rank r	1
Learning-rate scale η_{scale}	3×10^{-7}
Population size	8192
Parallel generations per GPU	256
Prompts per epoch	1
Generation / thinking length	256 tokens
Train / val temperature	0 / 0
Parallel validations	128

Table 14: Key hyperparameters for quantised EGGROLL training on GSM8K (teacher-forced) with RWKV-`7g7B`.

We use policy networks with 3 layers of 256 neurons and a range of environments that demonstrate different capabilities. We evaluate across the Navix (Pignatelli et al., 2024), Craftax (Matthews et al., 2024), Brax (Freeman et al., 2021), Kinetix (Matthews et al., 2025), and Jumanji (Bonnet et al., 2024) suites of environments. We evaluate 16 environments in total. We choose environments that are not trivial or impossible for PPO to solve, according to the original papers. We also choose environments that belong to different categories (e.g., environment size in Kinetix or categories in Jumanji).

We show a subsample of the evaluated environments in Fig. 4a with the remaining results and hyperparameter details in Appendix N.4. Our findings show that EGGROLL is competitive with Open ES on 7/16 environments, underperforms on 2/16, and outperforms on 7/16. This does not take into account the speed-ups when compared to using OpenES with full-rank updates (see Figure 15). We postulate that the reason for this performance increase is that the large networks are difficult to optimise for OpenES and lend themselves well to low-rank updates.

We present here the hyperparameter ranges we used for hyperparameter optimisation, as well as all hyperparameter settings for all the experiments. All RL experiments were run on an NVIDIA L40S GPU. For PPO, we use the same methodology to tune the hyperparameters as we did for OpenES and EGGROLL as described in Section 6.2. We report the ranges and the final hyperparameters here. We train PPO agents using Rejax (Liesen et al., 2024). We use the activation function from Gallici et al. (2025) in our experiments, which we refer to as the "pqn" activation function in our hyperparameter tables.

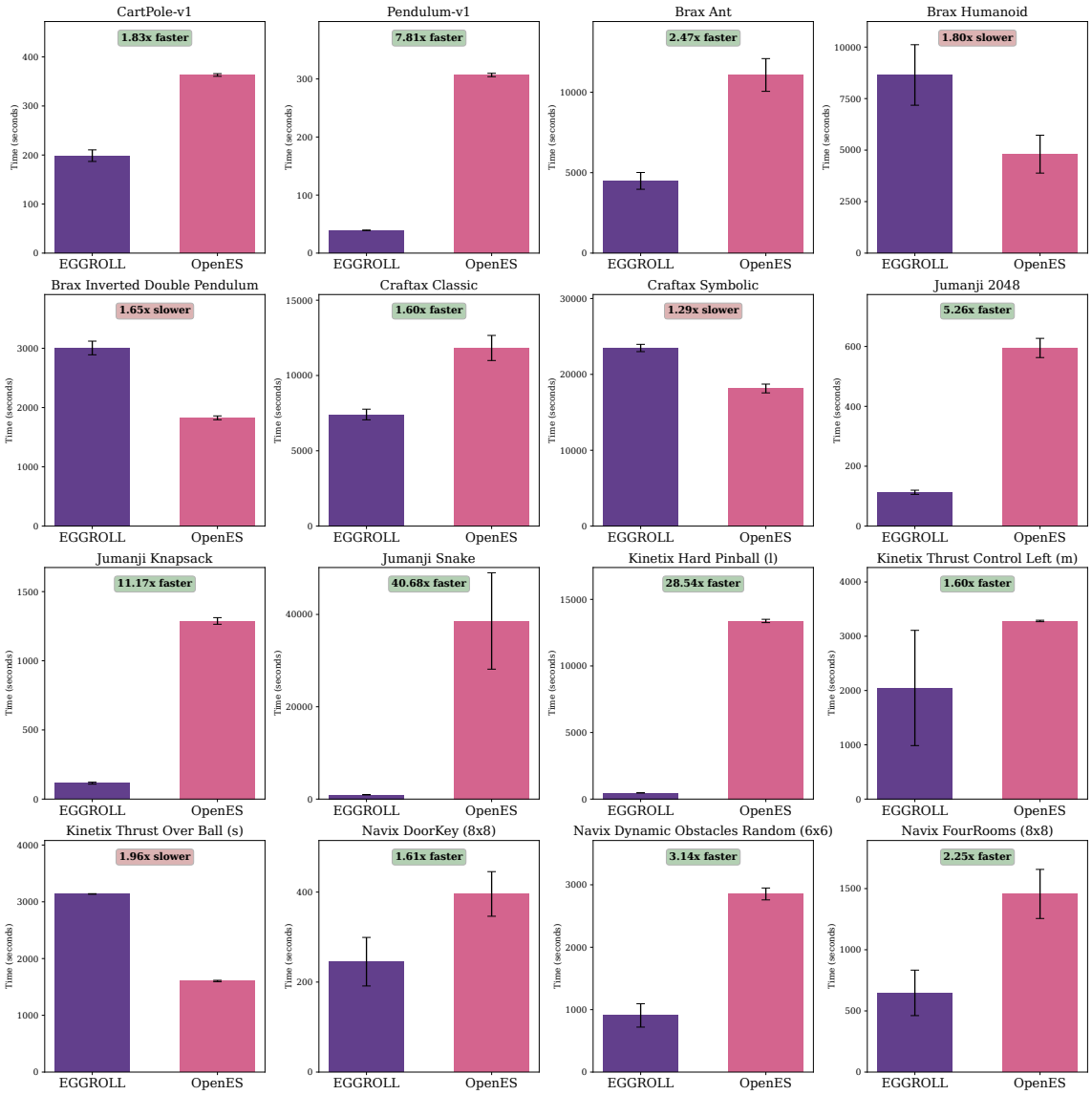


Figure 15: Comparison of reinforcement learning results: Mean and standard deviation of training time. Note that some of the timing difference is due to the differences in episode lengths, which is why the total time for EGGROLL sometimes appears longer than OpenES despite EGGROLL being faster on a per-timestep basis.

Table 15: Hyperparameter Ranges for EGGROLL and OpenES

Hyperparameter	**Values**
pop_size	512, 1024, 2048, 4096
n_parallel_evaluations	1, 4, 8
rank	1, 2, 4
optimizer	adamw, sgd, adam
learning_rate	1e-3, 1e-2, 1e-1
lr_decay	0.995, 0.999, 0.9995, 1.0
sigma	0.05, 0.2, 0.5
sigma_decay	0.995, 0.999, 0.9995, 1.0
rank_transform	true, false
deterministic_policy	true, false

Table 16: Hyperparameter Ranges for PPO

Hyperparameter	**Values**
clip_eps	0.1, 0.2, 0.3
ent_coef	0, 0.0001, 0.001
gae_lambda	0.9, 0.95, 0.98
gamma	0.95, 0.99, 0.995, 0.999
learning_rate	0.0001, 0.0003, 0.001
max_grad_norm	0.5, 1, 2
layer_size	256
n_layers	3
normalize_observations	true
normalize_rewards	false
num_envs	64, 128, 256
num_epochs	4, 8, 16
num_minibatches	16, 32, 64
num_steps	64, 128, 256
reward_normalization_discount	0.99
skip_initial_evaluation	false
vf_coef	0.5, 0.75, 1

Table 17: CartPole-v1

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	true
learning_rate	0.1	0.1
lr_decay	0.9995	0.9995
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	1	4
pop_size	2048	512
optimizer	sgd	adamw
rank	4	/
rank_transform	false	true
sigma	0.2	0.5
sigma_decay	0.999	0.9995

Table 18: Pendulum-v1

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	true
learning_rate	0.01	0.01
lr_decay	0.995	0.995
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	1	4
pop_size	4096	4096
optimizer	adam	adamw
rank	4	/
rank_transform	false	false
sigma	0.05	0.05
sigma_decay	0.995	1

Table 19: brax/ant

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	false
learning_rate	0.01	0.1
lr_decay	0.9995	0.995
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	1	8
pop_size	2048	512
optimizer	adam	adam
rank	1	/
rank_transform	false	false
sigma	0.05	0.05
sigma_decay	0.9995	0.9995

Table 20: brax/humanoid

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	true	false
learning_rate	0.1	0.1
lr_decay	1	0.995
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	8	8
pop_size	4096	1024
optimizer	adam	sgd
rank	1	/
rank_transform	true	true
sigma	0.2	0.2
sigma_decay	0.9995	0.995

Table 21: brax/inverted_double_pendulum

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	true	true
learning_rate	0.1	0.1
lr_decay	1	0.995
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	1	1
pop_size	2048	4096
optimizer	adam	adam
rank	2	/
rank_transform	true	true
sigma	0.5	0.05
sigma_decay	0.995	1

Table 22: craftax/Craftax-Classic-Symbolic-AutoReset-v1

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	false
learning_rate	0.01	0.001
lr_decay	0.995	0.995
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	4	8
pop_size	2048	4096
optimizer	sgd	adamw
rank	1	/
rank_transform	false	false
sigma	0.05	0.05
sigma_decay	1	0.995

Table 23: craftax/Craftax-Symbolic-AutoReset-v1

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	false
learning_rate	0.01	0.1
lr_decay	0.999	0.995
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	1	4
pop_size	512	1024
optimizer	sgd	adam
rank	4	/
rank_transform	true	false
sigma	0.05	0.5
sigma_decay	0.999	1

Table 24: jumanji/Game2048-v1

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	true
learning_rate	0.1	0.01
lr_decay	1	0.999
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	4	4
pop_size	1024	1024
optimizer	adamw	adamw
rank	1	/
rank_transform	false	true
sigma	0.5	0.05
sigma_decay	0.9995	0.9995

Table 25: jumanji/Knapsack-v1

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	false
learning_rate	0.1	0.01
lr_decay	0.999	1
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	4	1
pop_size	1024	2048
optimizer	sgd	adamw
rank	4	/
rank_transform	true	true
sigma	0.05	0.5
sigma_decay	1	0.995

Table 26: jumanji/Snake-v1

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	false
learning_rate	0.001	0.001
lr_decay	0.9995	1
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	8	1
pop_size	4096	2048
optimizer	adam	sgd
rank	1	/
rank_transform	true	false
sigma	0.05	0.2
sigma_decay	0.9995	1

Table 27: kinetix/l/hard_pinball

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	true	true
learning_rate	0.01	0.01
lr_decay	0.995	1
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	8	1
pop_size	2048	512
optimizer	sgd	sgd
rank	4	/
rank_transform	true	true
sigma	0.05	0.5
sigma_decay	0.999	0.9995

Table 28: kinetix/m/h17_thrustcontrol_left

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	false
learning_rate	0.1	0.001
lr_decay	0.9995	1
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	4	1
pop_size	512	1024
optimizer	sgd	adam
rank	4	/
rank_transform	true	true
sigma	0.5	0.5
sigma_decay	1	0.999

Table 29: kinetix/s/h1_thrust_over_ball

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	false
learning_rate	0.1	0.01
lr_decay	0.995	0.995
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	1	1
pop_size	512	2048
optimizer	adamw	sgd
rank	1	/
rank_transform	true	true
sigma	0.5	0.05
sigma_decay	0.9995	1

Table 30: navix/Navix-DoorKey-8x8-v0

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	false
learning_rate	0.01	0.01
lr_decay	0.9995	1
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	1	8
pop_size	1024	2048
optimizer	adamw	adam
rank	1	/
rank_transform	false	true
sigma	0.05	0.05
sigma_decay	1	1

Table 31: navix/Navix-Dynamic-Obstacles-6x6-Random-v0

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	false
learning_rate	0.01	0.01
lr_decay	0.999	1
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	4	1
pop_size	512	4096
optimizer	adam	adam
rank	2	/
rank_transform	false	false
sigma	0.05	0.2
sigma_decay	1	0.995

Table 32: navix/Navix-FourRooms-v0

Hyperparameter	eggroll	open_es
activation	pqn	pqn
deterministic_policy	false	false
learning_rate	0.01	0.001
lr_decay	0.999	0.9995
layer_size	256	256
n_layers	3	3
n_parallel_evaluations	4	4
pop_size	2048	2048
optimizer	sgd	adam
rank	4	/
rank_transform	true	false
sigma	0.05	0.05
sigma_decay	0.9995	0.9995

Table 33: PPO Hyperparameters (Set 1)

Hyperparameter	CartPole	Pendulum	Ant	Humanoid	IDP	CraftaxClassic	CraftaxSymbolic	Game2048
activation	pqn	pqn	pqn	pqn	pqn	pqn	pqn	pqn
clip_eps	0.2	0.1	0.2	0.3	0.1	0.2	0.2	0.3
ent_coef	0.0001	0.001	0	0.0001	0.0001	0.0001	0	0.001
gae_lambda	0.9	0.95	0.95	0.9	0.98	0.98	0.9	0.9
gamma	0.995	0.999	0.995	0.95	0.99	0.95	0.95	0.99
learning_rate	0.0003	0.0003	0.0003	0.0001	0.001	0.001	0.0003	0.0003
max_grad_norm	0.5	1	0.5	2	2	2	2	2
layer_size	256	256	256	256	256	256	256	256
n_layers	3	3	3	3	3	3	3	3
normalize_obs	true	true	true	true	true	true	true	true
normalize_rew	false	false	false	false	false	false	false	false
num_envs	256	256	64	256	64	128	256	64
num_epochs	4	16	8	4	4	4	4	8
num_minibatches	32	16	32	64	64	32	32	16
num_steps	128	256	128	64	128	128	64	64
rew_norm_discount	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
skip_initial_eval	false	false	false	false	false	false	false	false
vf_coef	0.5	1	1	0.75	1	0.5	0.75	0.75

Table 34: PPO Hyperparameters (Set 2)

Hyperparameter	Knapsack	Snake	HardPinball	ThrustLeft	ThrustBall	DoorKey	DynamicObs	FourRooms
activation	pqn	pqn	pqn	pqn	pqn	pqn	pqn	pqn
clip_eps	0.1	0.3	0.1	0.2	0.2	0.1	0.1	0.1
ent_coef	0.0001	0.001	0.0001	0.0001	0.0001	0.0001	0.001	0.001
gae_lambda	0.9	0.95	0.9	0.9	0.95	0.98	0.98	0.9
gamma	0.99	0.999	0.99	0.995	0.999	0.95	0.999	0.99
learning_rate	0.0001	0.0001	0.0001	0.0001	0.0001	0.0003	0.001	0.001
max_grad_norm	0.5	0.5	1	2	0.5	0.5	1	1
layer_size	256	256	256	256	256	256	256	256
n_layers	3	3	3	3	3	3	3	3
normalize_obs	true	true	true	true	true	true	true	true
normalize_rew	false	false	false	false	false	false	false	false
num_envs	256	128	256	256	64	64	128	256
num_epochs	4	4	16	16	16	16	4	8
num_minibatches	64	16	16	32	16	64	16	32
num_steps	128	128	64	128	64	256	128	256
rew_norm_discount	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
skip_initial_eval	false	false	false	false	false	false	false	false
vf_coef	0.75	0.75	0.5	0.5	0.5	0.75	0.5	0.75