Accelerated Quality-Diversity through Massive Parallelism

Anonymous authors
Paper under double-blind review

Abstract

Quality-Diversity (QD) optimization algorithms are a well-known approach to generate large collections of diverse and high-quality solutions. However, derived from evolutionary computation, QD algorithms are population-based methods which are known to be data-inefficient and require large amounts of computational resources. This makes QD algorithms slow when used in applications where solution evaluations are computationally costly. A common approach to speed up QD algorithms is to evaluate solutions in parallel, for instance by using physical simulators in robotics. Yet, this approach is limited to several dozen of parallel evaluations as most physics simulators can only be parallelized more with a greater number of CPUs. With recent advances in simulators that run on accelerators, thousands of evaluations can now be performed in parallel on single GPU/TPU. In this paper, we present QDax, an accelerated implementation of MAP-Elites which leverages massive parallelism on accelerators to make QD algorithms more accessible. We show that QD algorithms are ideal candidates to take advantage of progress in hardware acceleration. We demonstrate that QD algorithms can scale with massive parallelism to be run at interactive timescales without any significant effect on the performance. Results across standard optimization functions and four neuroevolution benchmark environments show that experiment runtimes are reduced by two factors of magnitudes, turning days of computation into minutes. More surprising, we observe that reducing the number of generations by two orders of magnitude, and thus having significantly shorter lineage does not impact the performance of QD algorithms. These results show that QD can now benefit from hardware acceleration, which contributed significantly to the bloom of deep learning.

1 Introduction

Quality-Diversity (QD) algorithms (Pugh et al., 2016; Cully & Demiris, 2017; Chatzilygeroudis et al., 2021) have recently shown to be an increasingly useful tool across a wide variety of fields such as robotics (Cully et al., 2015; Chatzilygeroudis et al., 2018), reinforcement learning (RL) (Ecoffet et al., 2021), engineering design optimization (Gaier et al., 2018), latent space exploration for image generation (Fontaine & Nikolaidis, 2021) and video game design (Gravina et al., 2019; Fontaine et al., 2020a; Earle et al., 2021). Instead of optimizing for a single solution like in conventional optimization, QD optimization searches for a population of high-performing and diverse solutions.

Adopting the QD optimization framework has many benefits. The diversity of solutions found in QD enables rapid adaptation and robustness to unknown environments (Cully et al., 2015; Chatzilygeroudis et al., 2018; Kaushik et al., 2020). Additionally, QD algorithms are also powerful exploration algorithms. They have been shown to be effective in solving sparse-reward hard-exploration tasks and achieved state-of-the-art results on previously unsolved RL benchmarks (Ecoffet et al., 2021). This is a result of the diversity of solutions present acting as stepping stones (Clune, 2019) during the optimization process. QD algorithms are also useful in the design of more open-ended algorithms (Stanley et al., 2017; Stanley, 2019; Clune, 2019) which endlessly generate their own novel learning opporunities. They have been used in pioneering work for open-ended RL with environment generation (Wang et al., 2019; 2020). Lastly, QD can also be used as effective data generators for RL tasks. The motivation for this arises from the availability of large amounts of data which resulted in the success of modern machine learning. The early breakthroughs in supervised

learning in computer vision have come from the availability of large diverse labelled datasets (Deng et al., 2009; Barbu et al., 2019). The more recent successes in unsupervised learning and pre-training of large models have similarly come from methods that can leverage even larger and more diverse datasets that are unlabelled and can be more easily obtained by scraping the web (Devlin et al., 2018; Brown et al., 2020). As Gu et al. (Gu et al., 2021) highlighted, more efficient data generation strategies and algorithms are needed to obtain similar successes in RL.

Addressing the computational scalability of QD algorithms (focus of this work) offers a hopeful path towards open-ended algorithms that endlessly generates its own challenges and solutions to these challenges. Likewise, they can also play a significant role in this more data-centric view of RL by generating diverse and high-quality datasets of policies and trajectories both with supervision and in an unsupervised setting (Cully, 2019; Paolo et al., 2020).

The main bottleneck faced by QD algorithms are the large number of evaluations required that is on the order of millions. When using QD in the field of Reinforcement Learning (RL) for robotics, this issue is mitigated by performing these evaluations in physical simulators such as Bullet (Coumans & Bai, 2016–2020), DART (Lee et al., 2018), and MuJoCo (Todorov et al., 2012). However, these simulators have mainly been developed to run on CPUs. Methods like MPI can be used to parallelise over multiple machines, but this requires a more sophisticated infrastructure (i.e., multiple machines) and adds some network communication overhead which can add significant runtime to the algorithm. Additionally, the number of simulations that can be performed in parallel can only scale with the number of CPU cores available. Hence, both the lack of scalability coupled with the large number of evaluations required generally make the evaluation process of evolutionary based algorithms like QD, for robotics take days on modern 32-core CPUs. Our work builds on the advances and availability of hardware accelerators, high-performance programming frameworks (Bradbury et al., 2018) and simulators (Freeman et al., 2021) that support these devices to scale QD algorithms.

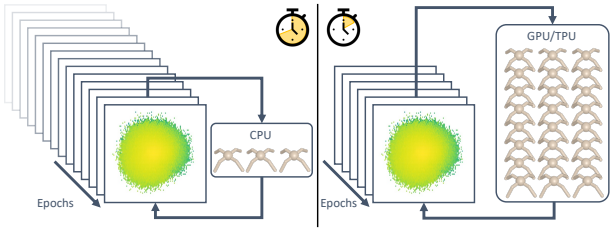

Figure 1: QDax uses massive parallelism on hardware accelerators like GPUs/TPUs to speed up runtime of QD algorithms by orders of magnitude

Historically, significant breakthroughs in algorithms have come from major advances in computing hardware. Most notably, the use of Graphic Processing Units (GPUs) to perform large vector and matrix computations enabled significant order-of-magnitude speedups in training deep neural networks. This brought about modern deep-learning systems which have revolutionized computer vision (Krizhevsky et al., 2012; He et al., 2016; Redmon et al., 2016), natural language processing (Hochreiter & Schmidhuber, 1997; Vaswani et al., 2017) and even biology (Jumper et al., 2021). Even in the current era of deep learning, significant network architectural breakthroughs (Vaswani et al., 2017) were made possible and are shown to scale with larger datasets and more computation (Devlin et al., 2018; Shoeybi et al., 2019; Brown et al., 2020; Smith et al., 2022).

Our goal in this paper is to bring the benefits of advances in compute and hardware acceleration to QD algorithms. The key contributions of this work are: (1) We show that massive parallelization with QD using large batch sizes significantly speeds up run-time of QD algorithms at no loss in final performance, turning hours/days of computation into minutes. (2) We demonstrate that contrary to prior beliefs, the number of iterations (generations) of the QD algorithm is not critical, given a large batch size. We observe this across optimization tasks and a range of QD-RL tasks. (3) We release QDax, an open-source accelerated Python library for Quality-Diversity algorithms which enables massive parallelization on a single machine. This makes QD algorithms more accessible to a wider range of practitioners and researchers. The source code of QDax is available in the Supplementary Materials.

2 Related Work

Quality-Diversity. QD algorithms were derived from interests in divergent search methods (Lehman & Stanley, 2011a) and behavioural diversity (Mouret & Doncieux, 2009; 2012) in evolutionary algorithms and

hybridizing such methods with the notion of fitness and reward (Lehman & Stanley, 2011b). While QD algorithms are promising solutions to robotics and RL, they remain computationally expensive and take a long time to converge due to high sample-complexity. The move towards more complex environments with high-dimensional state and action spaces, coupled with the millions of evaluations required for the algorithm to converge, makes these algorithms even more inaccessible to regular hardware devices. Progress has been made towards lowering the sample complexity of QD algorithms and can generally be categorized into two separate approaches. The first approach is to leverage the efficiency of other optimization methods such as evolution strategies (Colas et al., 2020; Fontaine et al., 2020b; Cully, 2020) and policy-gradients (Nilsson & Cully, 2021; Pierrot et al., 2021). The other line of work known as *model-based quality-diversity* (Gaier et al., 2018; Keller et al., 2020; Lim et al., 2021), reduces the number of evaluations required through the use of surrogate models to provide a prediction of the descriptor and objective.

Our work takes a separate approach orthogonal to sample-efficiency and focuses improvement on the runtime of QD algorithms instead by leveraging the batch size at each iteration. Additionally, despite algorithmic innovations that improve sample-efficiency, most QD implementations still rely on evaluations being distributed over large compute systems. These often give impressive results (Colas et al., 2020; Fontaine et al., 2019) but such resources are mainly inaccessible to most researchers and yet still take significant amount of time to obtain. Our work aims to make QD algorithms more accessible by running quickly on more commonly available accelerators, such as cloud available GPUs.

Hardware Acceleration for Machine Learning. Machine Learning, and more specifically Deep Learning methods, have benefited from specialised hardware accelerators that can parallelize operations. In the mid-2000's researchers started using GPUs to train neural networks (Steinkrau et al., 2005) because of their high degree of parallelism and high memory bandwidth. After the introduction of general purpose GPUs, the use of specialized GPU compatible code for Deep Learning methods (Raina et al., 2009; Ciresan et al., 2012) enabled deep neural networks to be trained a few orders of magnitude quicker than previously on CPUs (Lecun et al., 2015). Very quickly, frameworks such as Torch (Collobert et al., 2011), Tensorflow (Abadi et al., 2016), PyTorch (Paszke et al., 2019) or more recently JAX (Bradbury et al., 2018) were developed to run numerical computations on GPUs or other specialized hardware.

These frameworks have led to tremendous progress in deep learning. In other sub-fields such as deep reinforcement learning (DRL) or robotics, the parallelization has happened on the level of neural networks. However, RL algorithms need a lot of data and require interaction with the environment to obtain it. Such methods suffer from a slow data collection process as the physical simulators used to collect data were mainly developed for CPUs, which results in a lack of scalable parallelism and data transfer overhead between devices. More recently, new rigid body physics simulators that can leverage GPUs and run thousands of simulations in parallel have been developed. Brax (Freeman et al., 2021) and IsaacGym (Makoviychuk et al., 2021) are examples of these new types simulators. Gradient-based DRL methods can benefit from this massive parallelism as this would directly correspond to estimating the gradients more accurately through collection of larger amounts of data in a faster amount of time (Rudin et al., 2021) at each optimization step. Recent work (Gu et al., 2021; Rudin et al., 2021) show that control policies can be trained with DRL algorithms like PPO (Schulman et al., 2017) and DIAYN (Eysenbach et al., 2018) in minutes on a single GPU. However, unlike gradient-based methods, it is unclear until now how evolutionary and population-based approaches like QD would benefit from this massive parallelism since the implications of massive batch sizes has not been studied to the best of our knowledge. Our work studies the impact of massive parallelization on QD algorithms and its limitations.

3 Problem Statement

Quality-Diversity Problem. The Quality-Diversity (QD) problem Pugh et al. (2016); Chatzilygeroudis et al. (2021); Fontaine & Nikolaidis (2021) is an optimization problem which consists of searching for a set of solutions $\mathcal{A}$ that (1) are locally optimal, and (2) exhibit diverse features. QD problems are characterized by two components: (1) an objective function to maximize $f : \Theta \to \mathbb{R}$, and (2) a descriptor function $d : \Theta \to \mathcal{D} \subseteq \mathbb{R}^n$. That descriptor function is used to differentiate between solutions; it takes as input a solution $\boldsymbol{\theta} \in \Theta$, and computes a descriptive low-dimensional feature vector.

The goal of QD algorithms is to return an *archive* of solutions $\mathcal{A}$ satisfying the following condition: for each achievable descriptor $\boldsymbol{c} \in \mathcal{D}$, there exists a solution $\boldsymbol{\theta}_{\mathcal{A},\boldsymbol{c}} \in \mathcal{A}$ such that $d(\boldsymbol{\theta}_{\mathcal{A},\boldsymbol{c}}) = \boldsymbol{c}$ and $f(\boldsymbol{\theta}_{\mathcal{A},\boldsymbol{c}})$ maximizes the objective function with the same descriptor $\{f(\boldsymbol{\theta}) \mid \boldsymbol{\theta} \in \Theta \wedge d(\boldsymbol{\theta}) = \boldsymbol{c}\}$. However, the descriptor space $\mathcal{D}$ is usually continuous. This would require storing an infinite amount of solutions in the set. QD algorithms alleviate this problem by considering a tesselation of the descriptor space into cells (Mouret & Clune, 2015; Cully et al., 2015; Vassiliades et al., 2017): $(\text{cell}_i)_{i\in\mathcal{I}}$ and keeping only a single solution per cell. QD algorithms aim at finding a set of policy parameters $(\boldsymbol{\theta}_j)_{j\in\mathcal{J}}$ maximizing the QD-Score Pugh et al. (2016), defined as follows (where $f(\cdot)$ is assumed non-negative without loss of generality):

$$\underset{(\boldsymbol{\theta}_j)_{j\in\mathcal{J}}}{\text{maximize}} \quad \text{QD-Score} = \sum_{j\in\mathcal{J}} f(\boldsymbol{\theta}_j) \quad \text{such that } \forall j \in \mathcal{J},\ d(\boldsymbol{\theta}_j) \in \text{cell}_j \tag{1}$$

Thus, maximizing the QD-Score is equivalent to maximizing the number of cells containing a policy from the archive, while also maximizing the objective function in each cell.

Quality-Diversity for Neuroevolution-based RL. QD algorithms can also be used on RL problems, modeled as Markov Decision Processes $(\mathcal{S}, \mathcal{A}, p, r)$, where $\mathcal{S}$ is the states set, $\mathcal{A}$ denotes the set of actions, p is a probability transition function, and r is a state-dependent reward function. The return R is defined as the sum of rewards: $R = \sum_t r_t$. In a standard RL setting, the goal is to find a policy $\pi_{\boldsymbol{\theta}}$ maximizing the expected return.

The QD for Reinforcement Learning (QD-RL) problem (Nilsson & Cully, 2021; Tjanaka et al., 2022) is defined as a QD problem in which the goal is to find a set of diverse policy parameters $\mathcal{A} = (\boldsymbol{\theta}_j)_{j\in\mathcal{J}}$ leading to diverse high-performing behaviours. In the QD-RL context, the objective function matches the expected return $f(\boldsymbol{\theta}_j) = \mathrm{E}\left[R^{(\boldsymbol{\theta}_j)}\right]$; the descriptor function $d(\cdot)$ characterizes the state-trajectory of policy $\pi_{\boldsymbol{\theta}_j}$: $d(\boldsymbol{\theta}_j) = \mathrm{E}\left[\widetilde{d}(\boldsymbol{\tau}^{(\boldsymbol{\theta}_j)})\right]$ (with $\boldsymbol{\tau}$ denoting the state-trajectory $\boldsymbol{s}_{1:T}$). The QD-Score to maximize (formula 1) can then be expressed as follows, where all expected returns should be non-negative:

$$\underset{(\boldsymbol{\theta}_j)_{j\in\mathcal{J}}}{\text{maximize}} \quad \text{QD-Score} = \sum_{j\in\mathcal{J}} \mathrm{E}\left[R^{(\boldsymbol{\theta}_j)}\right] \quad \text{such that } \forall j \in \mathcal{J},\ \mathrm{E}\left[\widetilde{d}(\boldsymbol{\tau}^{(\boldsymbol{\theta}_j)})\right] \in \text{cell}_j \tag{2}$$

4 Background: MAP-Elites

MAP-Elites Mouret & Clune (2015) is a well-known QD algorithm which considers a descriptor space discretized into grid cells (Fig. 3 4th column). At the start, an archive $\mathcal{A}$ is created and initialized by evaluating random solutions. Then, at every subsequent iteration, MAP-Elites (i) generates new candidate solutions, (ii) evaluates their return and descriptor and (iii) attempts to add them to the archive $\mathcal{A}$. The iterations are done until we reach a total budget H of evaluations. During step (i), solutions are selected uniformly from the archive $\mathcal{A}$, and undergo variations to obtain a new batch of solutions $\widetilde{\mathcal{B}}$. In all our experiments, we use the iso-line variation operator (Vassiliades & Mouret, 2018) (Appendix Algo. 2). Then (ii), the solutions in the sampled batch $\widetilde{\mathcal{B}} = (\widetilde{\boldsymbol{\theta}}_j)_{j\in[\![1,N_{\mathcal{B}}]\!]}$ are evaluated to obtain their respective returns $(R^{(\widetilde{\boldsymbol{\theta}}_j)})_{j\in[\![1,N_{\mathcal{B}}]\!]}$ and descriptors $(d(\widetilde{\boldsymbol{\theta}}_j))_{j\in[\![1,N_{\mathcal{B}}]\!]}$. Finally (iii), each solution $\widetilde{\boldsymbol{\theta}}_j \in \mathcal{A}$ is placed in its corresponding cell in the behavioural grid according to its descriptor $d(\widetilde{\boldsymbol{\theta}}_j)$. If the cell is empty, the solution is added to the archive. If the cell is already occupied by another solution, the solution with the highest return is kept, while the other is discarded. A pseudo-code for MAP-Elites is presented in Algo. 1.

Algorithm 1 MAP-Elites ($N_{\mathcal{B}}$: Batch size)

```
for iteration ∈ [[1, I]] do
    if first iteration then
        B ← random solutions
    else
        B ← select solutions from archive A
    B̃ = (θ̃_j)_{j∈[[1,N_B]]} ← VARIATION(B)
    ∀j ∈ [[1, N_B]], run episode of π_θ̃j, compute return R^(θ_j)
    and trajectory τ^(θ_j)
    for j ∈ [[1, N_B]] do
        cell ← get grid cell of descriptor d̃(τ^(θ_j))
        θ_cell ← get content of cell
        if θ_cell is None then
            Add θ_j to cell
        else if R^(θ_j) > R^(θ_cell) then
            Replace θ_cell with θ_j in cell
        else
            Discard θ_j
return archive A
```

We use MAP-Elites to study and show how QD algorithms can be scaled through parallelization. We leave other variants and enhancements of QD algorithms which use learned descriptors (Cully, 2019; Paolo et al., 2020) and different optimization strategies such as policy gradients (Nilsson & Cully, 2021; Pierrot et al., 2021; Tjanaka et al., 2022) and evolutionary strategies (Colas et al., 2020; Fontaine et al., 2020b) for future work; we expect these variants to only improve performance on tasks, and benefit from the same contributions and insights of this work.

5 Leveraging Hardware Acceleration for Quality Diversity

In population-based methods, new solutions are the result of older solutions that have undergone variations throughout an iterative process as described in Algo. 1. The number of iterations I of a method commonly depends on the total number of evaluations (i.e. computational budget) H for the optimization algorithm and the batch size $N_{\mathcal{B}}$. For a fixed computation budget, a large batch size $N_{\mathcal{B}}$ would result in a lower number of iterations I and vice versa. At each iteration, a single solution can undergo a variation and each variation is a learning step to converge towards an optimal solution. It follows that the number of iterations I defines the maximum number of learning steps a solution can take. In the extreme case of $N_{\mathcal{B}} = H$, the method simply reduces to a single random variation to a random sample of parameters.

Based on this, an initial thought would be that the performance of population-based methods would be negatively impacted by a heavily reduced number of learning steps and require more iterations to find good performing solutions. The iterations of the algorithm are a sequential operation and cannot be parallelized. However, the evaluation of solutions at each iteration can be massively parallelized by increasing the batch size $N_{\mathcal{B}}$ which suits modern hardware accelerators. We investigate the effect of large $N_{\mathcal{B}}$ on population-based methods (more specifically QD algorithms) by ablating exponentially increasing $N_{\mathcal{B}}$ which was relatively unexplored in the literature.

Conventional QD algorithms parallelize evaluations by utilizing multiple CPU cores, where each CPU separately runs an instance of the simulation to evaluate a solution. For robotics experiments, we utilize Brax (Freeman et al., 2021), a differentiable physics engine in Python which enables massively parallel rigid body simulations. By leveraging a GPU/TPU, utilizing this simulator allows us to massively parallelize the evaluations in the QD loop which is the major bottleneck of QD algorithms. To provide a sense of scale, QD algorithms normally run on the order of several dozens of evaluations in parallel with $N_{\mathcal{B}} \sim 10^2$ due to limited CPUs while Brax can simulate over 10,000 solutions in parallel allowing QDax to have $N_{\mathcal{B}} \sim 10^5$. Brax is built on top of the JAX (Bradbury et al., 2018) programming framework, which provides an API to run accelerated code across any number of hardware acceleration devices such as CPU, GPU or TPU.

Beyond massive parallelization, acceleration of our implementation is also enabled with code compatible with just-in-time (JIT) compilation and fully on-device computation. Another key benefit of JAX is the JIT compilation which allows JAX to make full use of its Accelerated Linear Algebra (XLA).We provide implementation details with static archives that makes QDax compatible with JIT in the Appendix. Lastly, another bottleneck which slowed the algorithm down was the data transfer and marshalling across devices. To address this issue, we carefully consider data structures and place all of them on-device. QDax places the JIT-compiled QD algorithm components on the same device. This enables the entire QD algorithm to be run without interaction with the CPU.

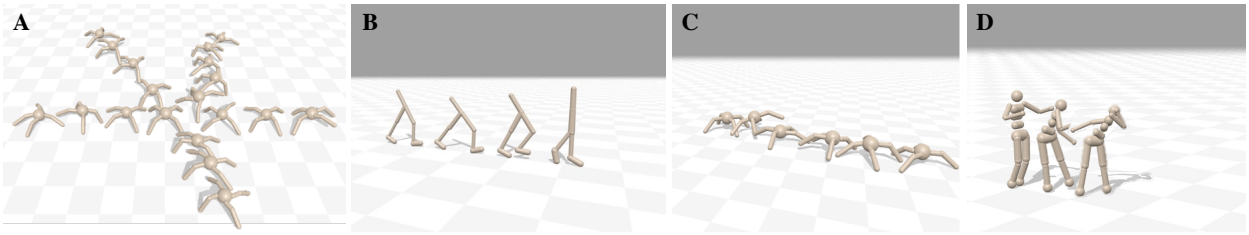

Figure 2: QD algorithms can discover diverse solutions. A: Omni-directional Ant task discovers ways to move in every direction. B,C,D: Uni-directional Walker, Ant and Humanoid task respectively discover diverse gaits for moving forward.

6 Experiments

Our experiments aim to answer the following questions: (1) How does massive parallelization affect performance of Quality-Diversity (MAP-Elites) algorithms? (2) How significant is the number of iterations/learning steps in QD algorithms? (3) What magnitude of speed-up does massive parallelization offer over existing implementations? (4) How does this differ across different hardware accelerators?

6.1 Domains

Rastrigin and Sphere. Rastrigin and Sphere are standard optimization functions commonly used as benchmark domains in optimization (Hansen et al., 2010; 2021) and QD-literature (Fontaine et al., 2020b; Fontaine & Nikolaidis, 2021). We optimize a $n = 100$ dimensional parameter space bounded between 0 and 1. More details regarding the objective function and descriptors are provided in the Appendix. We select these simple domains to demonstrate that the phenomena of large batch sizes applies generally to QD algorithms and not just to certain domains.

Continuous Control QD-RL. We perform experiments on two different QD-RL benchmark tasks (Cully et al., 2015; Nilsson & Cully, 2021; Tjanaka et al., 2022); *omni-directional* robot locomotion and *uni-directional* robot locomotion. In the omni-directional task, the goal is to discover locomotion skills to move efficiently in every direction.The descriptor functions is defined as the final x-y positions of the centre of mass of the robot at the end of the episode while the objective f is defined as a sum of a survival reward and torque cost. In contrast, the goal in the uni-directional tasks is to find a collection of diverse gaits to walk forward as fast as possible. In this task, the descriptor function is defined as the average time over the entire episode that each leg is in contact with the ground. For each foot i, the contact with the ground C_i is logged as a Boolean (1: contact, 0: no-contact) at each time step t. The descriptor function of this task was used to allow robots to recover quickly from mechanical damage (Cully et al., 2015). The objective f of this task is a sum of the forward velocity, survival reward and torque cost. Full equation details can be found in the Appendix B.

We use the Walker2D, Ant and Humanoid gym locomotion environments made available on Brax (Freeman et al., 2021) on these tasks. We run the omni-directional experiments for an episode length T of 100 steps and uni-directional experiments for 300 steps. In total, we report results on a combination of four tasks and environments; Omni-directional Ant and Uni-directional Walker, Ant and Humanoid. Fig. 2 illustrates examples of the types behaviors discovered from these tasks. We use a grid discretizations of shape [100,100], [10,10,10,10], [40,40] and [40.40] respectively for the corresponding tasks. We use fully connected neural network controllers with two hidden layers of size 64 and tanh output activation functions as policies across all environments and tasks.

6.2 Effects of massive parallelization on QD algorithms

To evaluate the effect of the batchsize $N_{\mathcal{B}}$, we run the algorithm for a fixed number of evaluations. We use 5 million evaluations for all QD-RL environments and 20 million evaluations for *rastrigin* and *sphere*. We evaluate the performance of the different batch sizes using the QD-score. The QD-score (Pugh et al., 2016) aims to capture both performance and diversity in a single metric. This metric is computed as the sum of objective values of all solutions in the archive (Eqn. 1). We plot this metric with respect to three separate factors: number of evaluations, number of iterations and total runtime. Other commonly used metrics in QD literature such as the best objective value and coverage can be found in the Appendix. We use a single A100 GPU to perform our experiments.

Fig. 3 shows the performance curves for QD-score and more importantly the differences when plot against the number of evaluations, iterations and total runtime. A key observation in the first column is that the metrics converge to the same final score after the fixed number of evaluations regardless of the batch size used. The Wilcoxon Rank-Sum Test for the final QD score across all the different batches results in p-values p>0.05 after applying the Bonferroni Correction. This shows that we do not observe statistically significant differences between the different batch sizes. Therefore, larger batch sizes and massive parallelism do not negatively impact the final performance of the algorithm. However, an important observation is that the larger batch sizes have a trend to be slower in terms of number of evaluations to converge. This can be

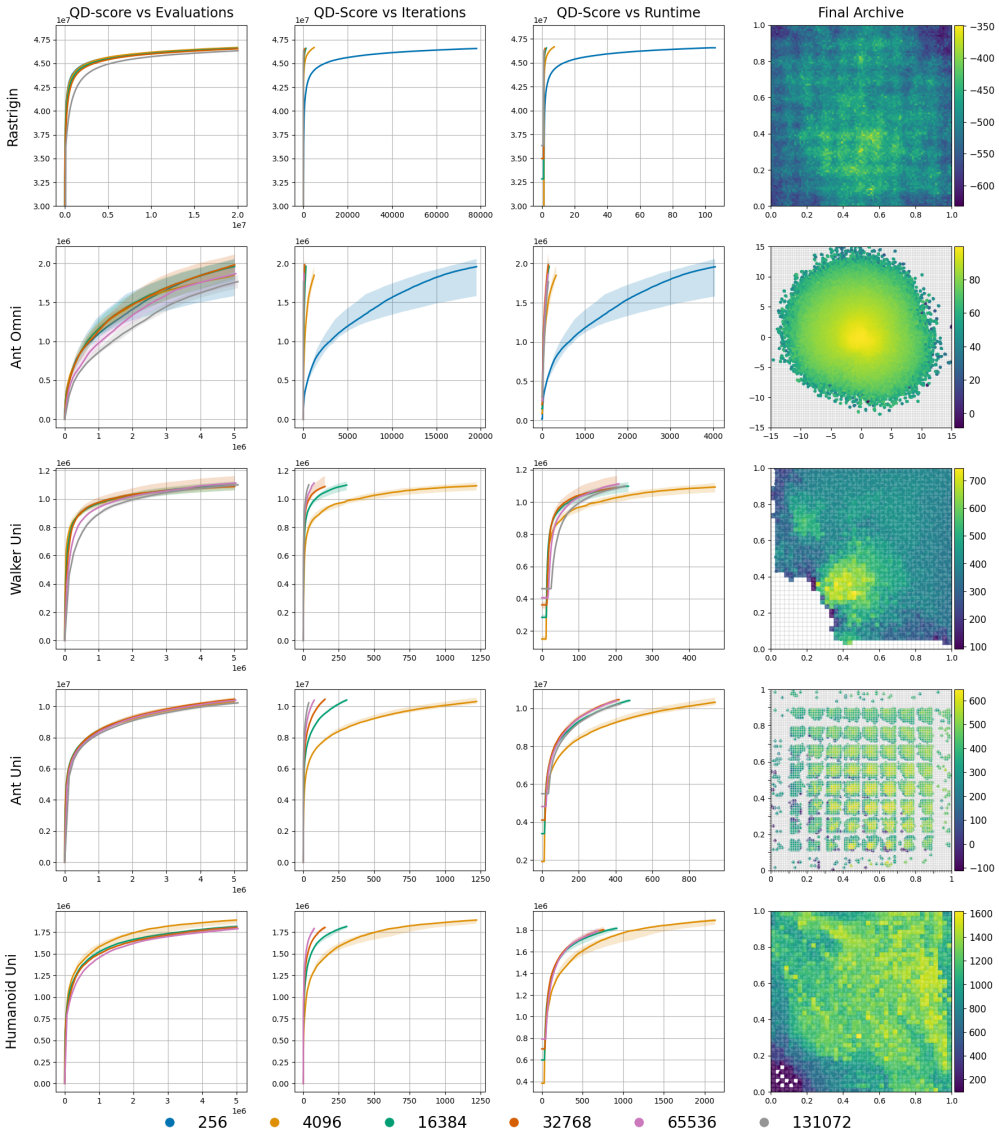

Figure 3: Performance metrics across domains. The plots show the QD-Score against the number of evaluations, iterations and run-times for each batch size. The rightmost column shows the final archives. For Ant Uni, the 4D descriptors are shown in 2D as done in (Cully et al.) Cully et al. (2015). The bold lines and shaded areas represent the median and interquartile range over 10 replications respectively. We leave the $N_{\mathcal{B}} = 256$ out of the Uni environments to increase readability of the large batch sizes.

expected as a larger number of evaluations are performed per iteration at larger batch sizes. Conversely, in some cases (Ant-Uni), a larger batch size can even have a positive impact on the QD-score. Given this result, the third column of Fig. 3 then demonstrates the substantial speed-up in total runtime of the algorithm obtained from using larger batch sizes with massive parallelism while obtaining the same performances. We can obtain similar results but in the order of minutes instead of hours. An expected observation we see when comparing the plots in the second and third column of Fig. 3 is how the total runtime is proportional to the number of iterations. As we are able to increase the evaluation throughput at each iteration through the parallelization, it takes a similar amounts of time to evaluate both smaller and larger batch sizes. The speed-up in total runtime of the algorithm by increasing batch size eventually disappears as we reach the limitations of the hardware. This corresponds to the results presented in the the next section 6.3 (Fig. 4) where we see the number eval/s plateauing.

The most surprising result is that the number of iterations and thus learning steps of the algorithm do not significantly affect the performance of the algorithm when increasing batch sizes $N_{\mathcal{B}}$ are used. In the case of the *QD-RL* domains, we observe that using $N_{\mathcal{B}} = 131{,}072$ which runs for only $I = 39$ provides the same performance as when run with $N_{\mathcal{B}} = 256$ and $I = 19{,}532$. This is true for all the domains presented. Given a fixed number of evaluations, a larger batch size would imply a lower number of iterations. This can be observed in the second column of Fig. 3. Therefore, although our results show that a larger batch size with less iterations has no negative impact on QD algorithms and can significantly speed-up the runtime of QD algorithms, the algorithm still needs to be allowed to run for a sufficient number of iterations to reach the similar final performance scores. The iterations/learning steps remain an important part of QD algorithms as new solutions that have been recently discovered and added to the archive $\mathcal{A}$ from a previous iteration can be selected as stepping stones to form future solutions. Results from our experiments show that iterations are not important as long as the number of evaluations remains identical. This is a critical observation as while iterations cannot be performed in parallel, evaluations can be, which enables the use of massive parallelization.

This is in contrast to other population-based approaches such as evolutionary strategies (Salimans et al., 2017), where we observe that massively large batch sizes can be detrimental when we run similar batch size ablation experiments (see Appendix C). We hypothesize that the reason for this is because in conventional evolutionary strategies (ES), there is a single optimization "thread" going on. The population is mainly used to provide an empirical approximation of the gradient. Increasing the size of the population helps to obtain a better estimate, but after a certain size, there is no benefits. On the contrary, with large batch size, the algorithms will waste evaluations. Conversely, in Quality-Diversity algorithms (MAP-Elites in particular), each of the thousands cells of the container can be seen as an independent optimization "thread". This enables MAP-Elites to greatly benefit from extra-large batch size.

6.3 Evaluation throughput and runtime speed of QD algorithms

We also evaluate the effect that increasing batch sizes have on the evaluation throughput of QD algorithms. We start with a batch size $N_{\mathcal{B}}$ of 64 and double from this value until we reach a plateau and observe a drop in performance in throughput. In practice, a maximum batch size of 131,072 is used.

The number of evaluations per second (eval/s) is used to quantify this throughput. The eval/s metric is computed by running the algorithm for a fixed number of generations (also referred to as iterations) N (100 in our experiments). We divide the corresponding batch size $N_{\mathcal{B}}$ representing the number of evaluations performed in this iteration and divide this value by the time it takes to perform one iteration t_n. We use the final average value of this metric across the entire run: $eval/s = \frac{1}{N}\sum_{n=1}^{N}\frac{N_{\mathcal{B}}}{t_n}$. While the evaluations per second can be an indication of the improvement in throughput from this implementation, we ultimately care about running the entire algorithm faster. To do this, we evaluate the ability to speed up the total runtime of QD algorithms. In this case, we run the algorithm to a fixed number of evaluations (1 million), as usually done in QD literature. Running for a fixed number of iterations would be an unfair comparison as the experiments with smaller batch sizes would have much less evaluations performed in total.

For this experiment, we consider the Ant Omni-directional task. We compare against common implementations of MAP-Elites and open-source simulators which utilize parallelism across CPU threads (see Table 1). All baseline algorithms used simulations with a fixed number of timesteps (100 in our experiments). We compare against both Python and C++ implementations as baselines. Pymapelites (Mouret & Clune, 2015) is a simple reference implementation from the authors of MAP-Elites that was made to be easily transformed for individual purposes. Pyribs (Tjanaka et al., 2021) is a more recent QD optimization library maintained by the authors of CMA-ME (Fontaine et al., 2020b). In both Python implementations, evaluations are parallelised on each core using the multiprocessing Python package. Lastly, $\text{Sferes}_{\text{v2}}$ (Mouret & Doncieux, 2010) is an optimized, multi-core and lightweight C++ framework for evolutionary computation, which includes QD implementations. It relies on template-based programming to achieve optimized execution speeds. The multi-core distribution for parallel evaluations is handled by Intel Threading Building Blocks (TBB) library. For simulators, we use PyBullet (Coumans & Bai, 2016–2020) in our Python baselines and Dynamic Animation and Robotics Toolkit (DART) (Lee et al., 2018) for the C++ baseline. We recognize that this comparison is not perfect and there could exist more optimized combinations of implementations

Implementation	Simulator	Resources	Eval/s	Batch size	Runtime (s)	Batch size
QDax (Ours)	Brax	GPU A100	30,846	65,536	**69**	65,536
QDax (Ours)	Brax	GPU 2080	11,031	8,192	117	8,192
pyribs	PyBullet	32 CPU-cores	184	8,192	7,234	4,096
pymapelites	PyBullet	32 CPU-cores	185	8,192	6,509	16,384
$\text{Sferes}_{\text{v2}}$	DART	32 CPU-cores	1,190	512	1,243	32,768

Table 1: Maximum throughput of evaluations per second and fastest run time obtained and their corresponding batch sizes across implementations. The medians over the 10 replications are reported.

and simulators but believe the baselines selected gives a good representation of what is commonly used in most works Cully & Mouret (2013); Nilsson & Cully (2021); Tjanaka et al. (2022).

We also test our implementation on two different GPU devices, a more accessible RTX2080 local device and a higher-performance A100 on Google Cloud. We only consider a single GPU device at each time. QDax was also tested and can be used to perform experiments across distributed GPU and TPU devices but we omit these results for simplicity.

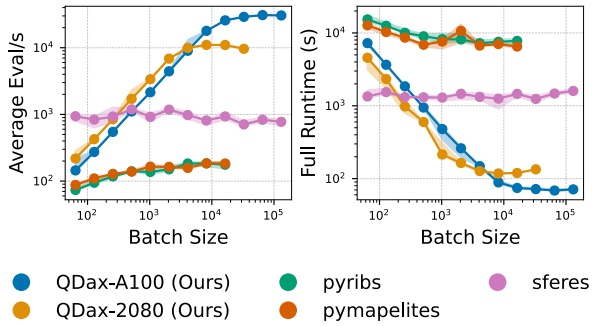

Figure 4: Average number of eval/s and full runtime of algorithm across batch sizes and implementations. Note the log scales on both axes to make distinction between batch sizes clearer.

Fig. 4 (Left) clearly shows that QDax has the ability to scale to much larger batch sizes which results in a higher throughput of evaluations. It is important to note the log scale on both axes to appreciate the magnitude of the differences. For QDax implementations (blue and orange), the number of evaluations per second scales as the batch size used increases. This value eventually plateaus once we reach the limit of the device. On the other hand, all the baseline implementations scales to a significantly lower extent. These results can be expected as evaluations using simulators which run on CPUs are limited as each CPU core runs a separate instance of the simulation. Therefore, given only a fixed number of CPU cores, these baselines would not scale as the batch size is increased. Scaling is only possible by increasing the number of CPUs used in parallel which is only possible with large distributed system with thousands of CPUs in the network. QDax can reach up to a maximum of 30,000 evaluations per second on an A100 GPU compared to maximum of 1,200 (C++) or 200 (Python) evaluations per second in the baselines (see Table 1). This is a 30 to 100 times increase in throughput, turning computation on the order of days to minutes. The negligible differences between the pyribs (green) and pymapelites (red) results show that the major bottleneck is indeed the evaluations and simulator used, as both of these baselines use the PyBullet simulator. The performance of the $\text{Sferes}_{\text{v2}}$ (purple) implementation can be attributed to its highly optimized C++ code. However, the same lack of scalability is also observed when the batchsize is increased. When looking at run time of the algorithm for a fixed number of evaluations on Fig. 4 (Right), we can see the effect of the larger throughput of evaluations at each iteration reflected in the decreasing run-time when larger batch sizes are used. We can run a QD algorithm with 1 million evaluations in just slightly over a minute (See Table 1) when using a batch size of 65,536 compared to over 100 minutes taken by Python baselines.

Our results also show that this scaling through massive parallelism is only limited by the hardware available. The experiments on both the RTX2080 (orange) and A100 (blue) show similar trends and increases in both evaluations per second and total runtime. The 2080 plateaus at a batch size of 8,192 capable of 11,000 eval/s while the higher-end A100 plateaus later at a batch size of 65,536 completing 30,000 eval/s.

7 Limitations and Future Work

In this paper, we presented QDax, an implementation of MAP-Elites that utilizes massive parallelization on accelerators that reduce the runtime of QD algorithms to interactive timescales on the order of minutes

instead of hours or days. We evaluate QDax across a range QD tasks and show that the performance of QD algorithms are maintained despite the significant speed-up that comes with the massive parallelism. Despite the apparent importance of iterations in QD algorithms, we show that when large batch sizes are used, a heavily reduced number of iterations and hence learning steps provides similar results.

Despite reaping the benefits of hardware in order to accelerate the algorithm, there are still some limitations in the future. As the archive stores the parameters among other things, the memory of the device becomes an issue preventing larger networks with more parameters from being used. Similarly, this memory limitation also prevent larger archives with more cells from being used. However, the experiments in this paper do not do use anything less than what is commonly used in literature.

Through this work, we hope to increase accessibility of QDax can help bring ideas from an emerging field of optimization to accelerate progress in machines learning. We also hope to see new algorithmic innovations that could leverage the massive parallelization to improve performance of QD algorithms.

Broader Impact Statement

Accessibility was a key consideration and motivation of this work. Our work turns the execution of QD algorithms from what took days/weeks on large CPU clusters to only minutes on a single easily accessible and free cloud GPU/TPU. We hope that this will prevent limitations of access of these algorithms to wider and more diverse communities of people, particularly from emerging and developing economies and beyond only well-resourced research groups and enterprises. We believe this increase in accessibility has the potential to open up novel applications of QD in new domains. On the other hand, a key scientific takeaway from our work was also that QD algorithms were able to scale with more powerful and modern hardware through massive parallelization. We recognize that this could also open up the possibility of 'buying' results (Schwartz et al., 2020) by utilizing more compute.

We are also aware that algorithms that can scale well with compute come at the risk of carbon footprint cost. Evident from the bloom of deep learning and more recently large language models, computational demands only continue to increase with models that scale. Interestingly, our work shows that given the same machine, we can decrease computation time by two orders of magnitude with no loss in performance of the algorithm which can aid in reducing environmental impact. Nonetheless, we highlight research on the environmental and carbon impact of AI and recommendations to be accountable in minimizing its effects (Dobbe & Whittaker, 2019; Dhar, 2020). A common step towards reducing AI's climate impact is to increase transparency on energy consumption and carbon emissions from computational resources used. We report the estimated emissions used in our experiments in the Appendix. Estimations were conducted using the Machine Learning Impact calculator (Lacoste et al., 2019).

References

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016. URL https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Danny Gutfreund, Joshua Tenenbaum, and Boris Katz. Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. 2019.

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/google/jax.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Konstantinos Chatzilygeroudis, Vassilis Vassiliades, and Jean-Baptiste Mouret. Reset-free trial-and-error learning for robot damage recovery. *Robotics and Autonomous Systems*, 100:236–250, 2018.

Konstantinos Chatzilygeroudis, Antoine Cully, Vassilis Vassiliades, and Jean-Baptiste Mouret. Quality-diversity optimization: a novel branch of stochastic optimization. In *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems*, pp. 109–135. Springer, 2021.

Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012. URL http://arxiv.org/abs/1202.2745.

Jeff Clune. Ai-gas: Ai-generating algorithms, an alternate paradigm for producing general artificial intelligence. *arXiv preprint arXiv:1905.10985*, 2019.

Cédric Colas, Vashisht Madhavan, Joost Huizinga, and Jeff Clune. Scaling map-elites to deep neuroevolution. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 67–75, 2020.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. 2011. URL http://infoscience.epfl.ch/record/192376.

Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2020.

Antoine Cully. Autonomous skill discovery with quality-diversity and unsupervised descriptors. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 81–89, 2019.

Antoine Cully. Multi-emitter map-elites: Improving quality, diversity and convergence speed with heterogeneous sets of emitters. *arXiv preprint arXiv:2007.05352*, 2020.

Antoine Cully and Yiannis Demiris. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation*, 22(2):245–259, 2017.

Antoine Cully and Jean-Baptiste Mouret. Behavioral repertoire learning in robotics. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pp. 175–182, 2013.

Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Payal Dhar. The carbon impact of artificial intelligence. *Nature Machine Intelligence*, 2(8):423–425, 2020.

R. Dobbe and M. Whittaker. Ai and climate change: How they're connected, and what we can do about it. 2019.

Sam Earle, Justin Snider, Matthew C Fontaine, Stefanos Nikolaidis, and Julian Togelius. Illuminating diverse neural cellular automata for level generation. *arXiv preprint arXiv:2109.05489*, 2021.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Matthew Fontaine and Stefanos Nikolaidis. Differentiable quality diversity. *Advances in Neural Information Processing Systems*, 34, 2021.

Matthew C Fontaine, Scott Lee, Lisa B Soros, Fernando de Mesentier Silva, Julian Togelius, and Amy K Hoover. Mapping hearthstone deck spaces through map-elites with sliding boundaries. In *Proceedings of The Genetic and Evolutionary Computation Conference*, pp. 161–169, 2019.

Matthew C Fontaine, Ruilin Liu, Ahmed Khalifa, Jignesh Modi, Julian Togelius, Amy K Hoover, and Stefanos Nikolaidis. Illuminating mario scenes in the latent space of a generative adversarial network. *arXiv preprint arXiv:2007.05674*, 2020a.

Matthew C Fontaine, Julian Togelius, Stefanos Nikolaidis, and Amy K Hoover. Covariance matrix adaptation for the rapid illumination of behavior space. In *Proceedings of the 2020 genetic and evolutionary computation conference*, pp. 94–102, 2020b.

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 http://github.com/google/brax.

Adam Gaier, Alexander Asteroth, and Jean-Baptiste Mouret. Data-efficient design exploration through surrogate-assisted illumination. *Evolutionary computation*, 26(3):381–410, 2018.

Daniele Gravina, Ahmed Khalifa, Antonios Liapis, Julian Togelius, and Georgios N Yannakakis. Procedural content generation through quality diversity. In *2019 IEEE Conference on Games (CoG)*, pp. 1–8. IEEE, 2019.

Shixiang Shane Gu, Manfred Diaz, Daniel C Freeman, Hiroki Furuta, Seyed Kamyar Seyed Ghasemipour, Anton Raichuk, Byron David, Erik Frey, Erwin Coumans, and Olivier Bachem. Braxlines: Fast and interactive toolkit for rl-driven behavior engineering beyond reward maximization. *arXiv preprint arXiv:2110.04686*, 2021.

Nikolaus Hansen, Anne Auger, Raymond Ros, Steffen Finck, and Petr Posik. Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking BBOB-2009. In *ACM-GECCO Genetic and Evolutionary Computation Conference*, Portland, United States, July 2010. URL https://hal.archives-ouvertes.fr/hal-00545727. pp. 1689-1696.

Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. Coco: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andrew J Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021. doi: 10.1038/s41586-021-03819-2.

Rituraj Kaushik, Pierre Desreumaux, and Jean-Baptiste Mouret. Adaptive prior selection for repertoire-based online adaptation in robotics. *Frontiers in Robotics and AI*, 6:151, 2020.

Leon Keller, Daniel Tanneberg, Svenja Stark, and Jan Peters. Model-based quality-diversity search for efficient robot learning. *arXiv preprint arXiv:2008.04589*, 2020.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature 2015 521:7553*, 521:436–444, 5 2015. ISSN 1476-4687. doi: 10.1038/nature14539. URL https://www.nature.com/articles/nature14539.

Jeongseok Lee, Michael X. Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S. Srinivasa, Mike Stilman, and C. Karen Liu. DART: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*, 3(22):500, Feb 2018. doi: 10.21105/joss.00500. URL https://doi.org/10.21105/joss.00500.

Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011a.

Joel Lehman and Kenneth O Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pp. 211–218, 2011b.

Bryan Lim, Luca Grillotti, Lorenzo Bernasconi, and Antoine Cully. Dynamics-aware quality-diversity for efficient learning of skill repertoires. *arXiv preprint arXiv:2109.08522*, 2021.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning. *CoRR*, abs/2108.10470, 2021. URL https://arxiv.org/abs/2108.10470.

J.-B. Mouret and S. Doncieux. SFERESv2: Evolvin' in the multi-core world. In *Proc. of Congress on Evolutionary Computation (CEC)*, pp. 4079–4086, 2010.

J-B Mouret and Stéphane Doncieux. Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary computation*, 20(1):91–133, 2012.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

Jean-Baptiste Mouret and Stéphane Doncieux. Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *2009 IEEE Congress on Evolutionary Computation*, pp. 1161–1168. IEEE, 2009.

Olle Nilsson and Antoine Cully. Policy gradient assisted map-elites. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 866–875, 2021.

Giuseppe Paolo, Alban Laflaquiere, Alexandre Coninx, and Stephane Doncieux. Unsupervised learning and exploration of reachable outcome space. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2379–2385. IEEE, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. URL http://arxiv.org/abs/1912.01703.

Thomas Pierrot, Valentin Macé, Geoffrey Cideron, Karim Beguir, Antoine Cully, Olivier Sigaud, and Nicolas Perrin. Diversity policy gradient for sample efficient quality-diversity optimization. 2021.

Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

Rajat Raina, Anand Madhavan, and Andrew Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 873–880, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553486. URL https://doi.org/10.1145/1553374.1553486.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. *arXiv preprint arXiv:2109.11978*, 2021.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

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

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63 (12):54–63, 2020.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zhang, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model, 2022.

Kenneth O Stanley. Why open-endedness matters. *Artificial life*, 25(3):232–235, 2019.

Kenneth O Stanley, Joel Lehman, and Lisa Soros. Open-endedness: The last grand challenge you've never heard of. *While open-endedness could be a force for discovering intelligence, it could also be a component of AI itself*, 2017.

Dave Steinkrau, Patrice Y. Simard, and Ian Buck. Using gpus for machine learning algorithms. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, ICDAR '05, pp. 1115–1119, USA, 2005. IEEE Computer Society. ISBN 0769524206. doi: 10.1109/ICDAR.2005.251. URL https://doi.org/10.1109/ICDAR.2005.251.

Bryon Tjanaka, Matthew C. Fontaine, Yulun Zhang, Sam Sommerer, Nathan Dennler, and Stefanos Nikolaidis. pyribs: A bare-bones python library for quality diversity optimization. https://github.com/icaros-usc/pyribs, 2021.

Bryon Tjanaka, Matthew C Fontaine, Julian Togelius, and Stefanos Nikolaidis. Approximating gradients for differentiable quality diversity in reinforcement learning. *arXiv preprint arXiv:2202.03666*, 2022.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Vassilis Vassiliades and Jean-Baptiste Mouret. Discovering the Elite Hypervolume by Leveraging Interspecies Correlation. In *GECCO 2018 - Genetic and Evolutionary Computation Conference*, Kyoto, Japan, July 2018. doi: 10.1145/3205455.3205602. URL https://hal.inria.fr/hal-01764739.

Vassilis Vassiliades, Konstantinos Chatzilygeroudis, and Jean-Baptiste Mouret. Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22(4):623–630, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Poet: open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 142–151, 2019.

Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeffrey Clune, and Kenneth Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *International Conference on Machine Learning*, pp. 9940–9951. PMLR, 2020.

Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 3381–3387, 2008. doi: 10.1109/CEC.2008.4631255.