Cheaper and Faster: Distributed Deep Reinforcement Learning with Serverless Computing

Anonymous submission

Abstract

Deep reinforcement learning (DRL) has gained immense success in many applications, including gaming AI, robotics, and system scheduling. Distributed algorithms and architectures have been vastly proposed (*e.g.*, actor-learner architecture) to accelerate DRL training with large-scale server-based clusters. However, training on-policy algorithms with the actor-learner architecture unavoidably induces resource wasting due to synchronization between learners and actors, thus resulting in significantly extra billing. As a promising alternative, serverless computing naturally fits on-policy synchronization and alleviates resource wasting in distributed DRL training with pay-as-you-go pricing. Yet, none has leveraged serverless computing to facilitate DRL training. This paper proposes MINIONSRL, the first serverless distributed DRL training framework that aims to accelerate DRL training- and cost-efficiency with dynamic actor scaling. We prototype MINIONSRL on top of Microsoft Azure Container Instances and evaluate it with popular DRL tasks from OpenAI Gym. Extensive experiments show that MINIONSRL reduces total training time by up to 52% and training cost by 86% compared to latest solutions.

Introduction

The success of AlphaGo (Silver et al. 2016) inspires various deep reinforcement learning (DRL) applications, such as gaming AI (Vinyals et al. 2019; Berner et al. 2019), robotics (Ji et al. 2022; Thumm and Althoff 2022), system scheduling (Mao et al. 2022; Qiu et al. 2023), bioinformatics (Jumper et al. 2021), and large language model training (OpenAI 2023). DRL training is *expensive*, which takes numerous trials and errors, consuming countless computing resources and time. Thus, a few distributed DRL algorithms are proposed to parallelize and accelerate the training with multiple servers (Luo et al. 2019; Wijmans et al. 2019; Espeholt et al. 2018; Horgan et al. 2018; Kapturowski et al. 2018; Hessel et al. 2018; Espeholt et al. 2020).

The actor-learner architecture represents one of the most efficient distributed DRL training paradigms available (Luo et al. 2019; Espeholt et al. 2018, 2020). This approach decouples the DRL agent's responsibilities into two distinct roles: *actors* for data sampling and *learners* for policy updates. On-policy algorithms (Schulman et al. 2017; Achiam et al. 2017; Wijmans et al. 2019) have emerged as a prominent DRL algorithm family, fully leveraging the actor-learner training architecture with distributed computing clusters (Gu et al. 2017).

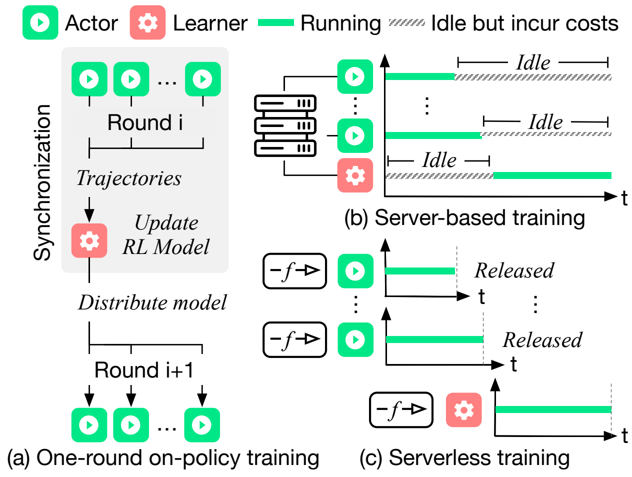


Figure 1: Server-based *v.s.* serverless architectures.

To facilitate efficient learning in a distributed environment with consistent DRL policies, on-policy algorithms enforce a synchronization process between learners and actors after every training round, as Fig. 1(a) shows. Due to the stochastic environment dynamics (*e.g.*, game environments), some actors might have episodes that end sooner, leading them to finish rounds earlier and wait in idle for other actors. Additionally, all actors remain idle during the policy update by the learner as Fig. 1(b) shows. However, these idle actors significantly waste computing resources, amplifying training costs with *server-based* clusters.

The dilemma of server-based DRL training. The state-of-the-art *server-based* approaches reserve a fix number of workers (*e.g.*, physical or cloud servers) for distributed DRL training. These methods face two primary challenges: 1) their coarse-grained resource management (*e.g.*, server-level instead of CPU core-level) leaves idle actors' resource unreleased; and 2) the prolonged server startup process (minute-level) prevents efficient mitigation of DRL actors' idle time through frequent server toggling. Thus, we propose to enable *cheaper* and *faster* distributed DRL with *serverless computing*.

Serverless computing and how it fits distributed DRL? *Serverless Computing*, also known as Function-as-a-Service

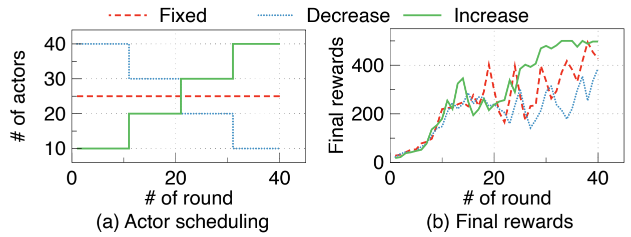


Figure 2: Adjusting the number of actors when training OpenAI Gym CartPole-v1 (Brockman et al. 2016) with Proximal Policy Optimization (PPO) (OpenAI 2017).

(FaaS), is a new cloud computing model that uses lightweight containers as execution units. Unlike physical clusters and traditional cloud computing that require tedious configuration, serverless computing packages and executes tasks (*e.g.*, DRL actors and learner) as *functions* with instant toggling (*i.e.*, sub-second level) and autoscaling. Thus, serverless computing has been widely deployed to serve computation-intensive applications, such as deep learning (Ali et al. 2020; Carreira et al. 2019; Wang, Niu, and Li 2019; Yu et al. 2021) and scientific computing (Chard et al. 2020; Roy et al. 2022). Fig. 1(c) shows how serverless functions naturally accommodates on-policy training process with on-and-off DRL actors and learners, which mitigates idle resources.

Leveraging serverless computing's fine-grained resource provisioning and instant execution, a fundamental question arises—how to achieve faster and cheaper DRL training with an appropriate number of concurrent actors in each round?

To answer this question, we propose MINIONSRL, the first serverless DRL training framework, which dynamically adjusts the number of actors according to the DRL training progress. As the training proceeds, it takes varying volumes of training data to advance neural network model quality in each round (Devarakonda, Naumov, and Garland 2017; McCandlish et al. 2018). In the actor-learner architecture, the number of actors in each round determines the volume of sampled training data, thus impacting the policy network quality. This intuition leads us to design an intelligent scheduler that learns to perform *dynamic actor scaling* for each training round to optimize the DRL policy quality with minimal training time and costs. Our main contributions are as follows:

- We propose MINIONSRL, the first distributed DRL training framework based on serverless computing.
- We design an intelligent scheduler that learns to scale out actors dynamically and accelerate distributed DRL training with minimal costs.
- We evaluated MINIONSRL on an off-the-shelf serverless testbed (*i.e.*, Microsoft Azure). Experiments with OpenAI Gym show that MINIONSRL reduces up to 52% total training time and 86% costs, respectively.

Preliminaries

Actor-learner architecture

The actor-learner architecture is one of the most performant and efficient approaches that attempt to scale and accelerate DRL training. A3C (Mnih et al. 2016) first introduced a simple actor-leaner prototype. IMPALA (Espeholt et al. 2018) proposed a standard actor-learner architecture with V-trace correction for off-policy training. IMPACT (Luo et al. 2019) added a surrogate target network to the actor-learner architecture for stabilizing training performance. SEED RL (Espeholt et al. 2020) aimed to accelerate actor-learner architecture by centralizing actor inferences to GPUs.

Server-based *v.s.* Serverless DRL Training

Server-based training platforms provide users with an entire server with coarse-grained resources packed together. For example, the cheapest Azure cloud server equipped with a V100 GPU is `Standard_NC6s_v3`, bundled with 6 CPU cores and 112GB memory. Instead, serverless computing executes tasks with lightweight containers, thus allowing fine-grained resource provisioning with instant function launch/release, which charges users by the amount of resources (*e.g.*, CPU/GPU and memory) only in actual execution (*e.g.*, second). Due to the unique features, serverless computing is particularly appealing for tasks that require elasticity and high concurrency, such as scientific computing (Chard et al. 2020; Roy et al. 2022) and distributed training (Wang, Niu, and Li 2019; Guo et al. 2022; Thorpe et al. 2021; Yu et al. 2021).

Motivating Dynamic Actor Scaling for DRL

One of the fundamental differences between DRL and supervised learning is the training data. In supervised learning tasks, training data is collected offline before the training starts, whereas DRL tasks sample the training data *online* during the rollout of the current policy with actors. As the training proceeds, neural networks tend to demand varying volumes of training data in each round (Devarakonda, Naumov, and Garland 2017; McCandlish et al. 2018). Hence, the number of DRL actors dictates the amount of training data sampled in each round, potentially influencing the efficiency of DRL training and the quality of the policy.

Fig. 2 uses a real-world experiment to show the potential impact on policy quality when adjusting the number of actors during DRL training. Fig. 2(a) shows the three actor dynamic scaling strategies: 1) **Fixed**, which uses a fixed number of actors, 2) **Decrease**, which decreases ten actors every ten training rounds, and 3) **Increase**, which increases ten actors every ten rounds. Note that the three strategies are under the same actor budget (*i.e.*, the cumulative number of total used actors is the same). Fig. 2(b) shows the different final rewards achieved by the three actor scaling strategies, raising a fundamental question—given the flexibility and scalability of serverless computing, how to dynamically scale out actors for faster and cheaper DRL training?

MINIONSRL's Design

Overview

To answer this questions, we propose MINIONSRL, which refactors the actor-learner DRL architecture into independent serverless functions with fine-grained resource management. MINIONSRL aims to instantly launch necessary

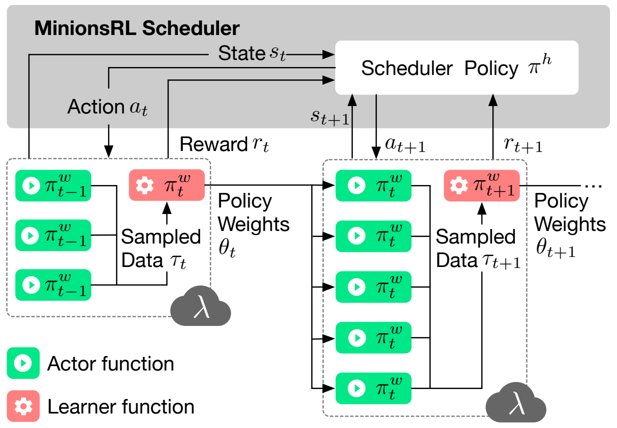


Figure 3: MINIONSRL's architecture.

number of actors for faster DRL training, while promptly releasing idle actor and learner functions to optimize cost-efficiency. Serverless computing's pay-as-you-run nature frees MINIONSRL from expenses on stopped functions, thus reducing unnecessary monetary costs throughout the training process. Serverless computing provides agile scalability so that MINIONSRL can dynamically scale the number of actors in real-time as needed. Specifically, MINIONSRL aims to address two primary challenges:

Incorporate characteristics of DRL tasks. DRL training significantly differs from other ML training, for example, the recurrent interaction and online data sampling. To achieve high performance and low cost, it's necessary to incorporate MINIONSRL's scheduling with awareness of DRL workload characteristics. However, existing machine learning schedulers are not designed for distributed DRL training (Guo et al. 2022; Wang, Niu, and Li 2019; Carreira et al. 2019), thus their tricks are not directly applicable.

Solution: The training process of MINIONSRL is designed to be DRL objective and constraint-aware. To capture unique characteristics of DRL workloads, we embed critical features into the states of MINIONSRL's agent, such as the average final rewards and Kullback–Leibler (KL) divergence. The reward function of MINIONSRL's agent is also crafted with awareness of the momentary budget and workload actor performance, guiding MINIONSRL to search for optimal scheduling decisions through training.

Trade-off between training performance and cost. It's ambiguous to determine how many actors should be launched in each round to hit a sweet spot between training performance and cost. Moreover, it's difficult to infer the complicated dependency between actor scheduling and policy updates, further escalating the challenge.

Solution: We formulate the actor scheduling of distributed DRL training as a sequential decision problem and analyze the complexity. Based on the analysis, we devise a DRL-based scheduler to dynamically scale actors by learning from experiences.

Problem Formulation

We consider a general RL training setting—an agent continuously interacts with the environment to learn a policy that maximizes cumulative rewards. The training proceeds in the actor-learner fashion as shown in Fig. 3. The training is terminated when the agent achieves target final rewards J or runs out of a monetary budget B. Let $f_{k,i}$ denote the actor function i scheduled for sampling trajectories in round k, where $i \in \{1, \dots, I_k\}$, $k \in \{1, \dots, K\}$ that I_k and K denote the total number of training rounds to reach final reward J and the total number of actor functions in round k, respectively. When all actor functions are terminated after sampling, one learner function is launched to learn and update the policy based on the sampled data. At the end of round k, the cumulative reward achieved by the agent is represented as j_k. Let $P^a_{k,i}$ and P^l_k denote the execution time of the i^{th} actor and learner function in round k, where each actor and learner function is allocated with d^a and d^l resources, respectively. We use c to represent the unit price of executing a function with a unit resource for one second. Thus, the duration P_k and cost C_k of round k in on-policy training is given by

$$P_k := P^l_k + \max_i \{P^a_{k,i}\}, \tag{1}$$

$$C_k := c\big(P^l_k d^l + \sum_{i=1}^{I_k} P^a_{k,i} d^a\big). \tag{2}$$

The goal is to minimize the total training duration $\sum_{k=1}^{K} P_k$ via Eq. 1 while the total cost $\sum_{k=1}^{K} C_k$ via Eq. 2 subjects to a monetary budget B, by deciding I_k in each round:

$$\min_{I_k} \sum_{k=1}^{K} \Big(P^l_k + \max_i \{P^a_{k,i}\}\Big), \tag{3}$$

$$\text{s.t.} \;\; K \geq 1, \; j_k \geq J,$$

$$c\big(P^l_k d^l + \sum_{i=1}^{I_k} P^a_{k,i} d^a\big) \leq B. \tag{4}$$

The optimization problem is a challenging sequential decision problem with an exponential complexity of $\mathcal{O}(I^K)$ for searching optima. Exhaustively enumerating the optimal solution is unrealistic due to the need for countless retraining. What's more, the complex correlation between actor scheduling and policy update further escalates the difficulty of solving the problem. Therefore, we resort to DRL itself—using a DRL agent to learn how to optimally schedule actors for distributed DRL training workloads.

DRL-based Actor Scheduler

Fig. 3 depicts the architecture of MINIONSRL with a two-fold workflow: 1) the DRL workload that trains in actor-learner fashion, and 2) the DRL-based actor scheduler that manages the DRL workload training. At the beginning of each round k, the scheduler takes an *action* on deciding how many actors I_k should be launched for evaluation and data sampling, based on the *state* collected from the leaner update. The action made by the scheduler is judged by a per-round *reward* from the actors. We describe the design of states, actions, and rewards in our actor scheduler as follows:

Table 1: Hyperparameters of PPO used in the training workloads and the search ranges of the scheduler.

Parameter	Workload	Scheduler
Learning rate	0.00005	[0.001, **0.005**, 0.01]
Discount factor (γ)	0.99	0.99
Mini-batch size	256	[**1**, 2, 4]
Clip parameter	0.3	[0.1, **0.2**, 0.3]
KL coefficient	0.2	0.0
KL target	0.01	[0.005, **0.01**, 0.015]
Entropy coefficient	0.0	[0.005, **0.01**, 0.015]
Value function coefficient	1.0	[0.1, 0.3, **0.5**, 0.7]

State. The state is represented by a flat vector $s_k = (k, L_{k-1}, \bar{R}_{k-1}, D_k^{\text{KL}}, P_{k-1}^l, \bar{P}_{k-1}^a, P_{k-1}, b_k)$. Specifically, L_{k-1} and $\bar{R}_{k-1}$ are the loss value of the learner and *average* final rewards of actors evaluated from the previous round. $D_k^{\text{KL}} := \sum_a \pi_k(a|s) \log\left(\frac{\pi_k(a|s)}{\pi_{k-1}(a|s)}\right)$ denotes the KL divergence of two consecutive workload policies π_k^w and π_{k-1}^w, which is commonly employed to measure the difference between two policies (Achiam et al. 2017; Schulman et al. 2017, 2015). We include L_{k-1}, $\bar{R}_{k-1}$, and D_k^{KL} in the state to provide the scheduler insights about how the learner policy updates. Recall that P_{k-1}^l and P_{k-1} represent the execution time of the learner and the total duration of training round $k-1$, respectively. Additionally, $\bar{P}_{k-1}^a$ represents the execution time averaged over actors from the previous round, and b_k represents the budget remaining after training of the current round. The scheduler leverages the above metrics to adjust the decisions during the scheduling process.

Action. At the beginning of round k, the scheduler outputs an action $a_k := I_k$, a scalar value selected within $[1, I_{max}] \in \mathbb{Z}^+$, where I_{max} is the maximum number of actors that we can allocate per round. The scheduler chooses action a_k under the guidance of its policy $\pi^h(\theta)$.

Reward. The reward returned at the end of round k is defined as $r_k := -\beta P_k$, where $\beta \in (0, 1)$ is a reward coefficient. The cumulative rewards through K rounds is given by $-\sum_{k=1}^{K} \gamma^t \beta P_k$, where $\gamma \in (0, 1)$. Intuitively, the longer the workload takes to finish training (either actor evaluation reaching target final reward J or running out of budget B), the more we will penalize the scheduler. Additionally, we define the reward of the end round K as

$$r_K := \begin{cases} -\beta P_K & \bar{R}_K \geq J \text{ and } b_k \geq 0, \\ -\beta P_K + (\max_k \bar{R}_k - J) & \text{otherwise.} \end{cases}$$

We add an additional term to the reward at the end round to judge the overall performance of MINIONSRL's scheduler. If the scheduler fails, *i.e.*, actor evaluation always fails to reach target final reward J and runs out of the budget B, the term $(\max_k \bar{R}_k - J < 0)$ penalizes the scheduler with negative returns. What's more, lower actor evaluation performance gets more penalties. Thus, we guide MINIONSRL's scheduler to overcome failures by minimizing the gap $(\max_k \bar{R}_k - J)$ while aiming to speed up the workload training.

Note that both training time and cost are considered in the problem formulation, where training time is our direct optimization objective (Eq. 3), and training cost is a hard constraint for MINIONSRL (Eq. 4). MINIONSRL supports minimizing cost by slightly changing the reward function. We assume the common practice is to optimize training performance under a given monetary budget.

Table 2: Total training time and costs for six tasks.

Environment	Baseline	Time (s)	Cost ($)
Hopper	MINIONSRL	241 ± 24	$\mathbf{1.2 \pm 0.2}$
	Azure ML	277 ± 21	4.5 ± 0.5
	IMPACT	291 ± 26	4.6 ± 0.8
	MINIONSRL-Adapt	403 ± 45	4.4 ± 0.6
	MINIONSRL-Max	$\mathbf{232 \pm 19}$	1.7 ± 0.3
Humanoid	MINIONSRL	$\mathbf{334 \pm 42}$	$\mathbf{1.3 \pm 0.4}$
	Azure ML	464 ± 56	9.3 ± 1.2
	IMPACT	436 ± 57	2.9 ± 0.7
HalfCheetah	MINIONSRL	220 ± 29	$\mathbf{1.1 \pm 0.3}$
	Azure ML	458 ± 49	2.9 ± 0.6
	IMPACT	$\mathbf{193 \pm 12}$	3.0 ± 0.8
Gravitar	MINIONSRL	$\mathbf{2295 \pm 337}$	$\mathbf{6.6 \pm 0.9}$
	Azure ML	2902 ± 481	11.4 ± 1.5
	IMPACT	3375 ± 714	12.0 ± 1.7
SpaceInvaders	MINIONSRL	$\mathbf{1787 \pm 229}$	$\mathbf{7.8 \pm 1.2}$
	Azure ML	2260 ± 343	26.9 ± 3.1
	IMPACT	2628 ± 402	25.8 ± 2.4
Qbert	MINIONSRL	506 ± 59	2.3 ± 0.8
	Azure ML	872 ± 68	6.0 ± 1.0
	IMPACT	768 ± 66	6.4 ± 1.3
	MINIONSRL-Adapt	750 ± 61	$\mathbf{2.0 \pm 0.7}$
	MINIONSRL-Max	$\mathbf{484 \pm 33}$	5.1 ± 1.2

Training MINIONSRL's Scheduler

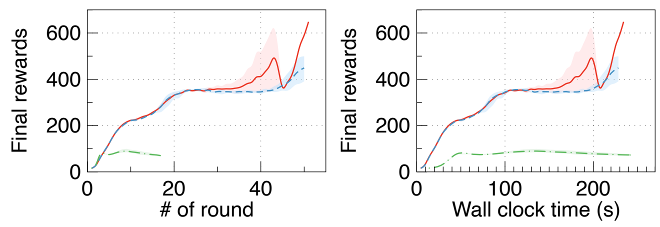
We employ the famous PPO algorithm (Schulman et al. 2017) to train MINIONSRL's scheduler. Table 1 characterizes the hyperparameters and search ranges of PPO used in MINIONSRL. We employed Ray-Tune (Liaw et al. 2018) to efficiently search for optimal hyperparameters within the ranges. Table 3 in the Supplementary Material describes the architecture of neural networks used in MINIONSRL. The lightweight policy and critic networks in MINIONSRL's scheduler are constructed by two fully-connected layers of 64 hidden units with `Tanh` activation. We follow existing DRL-driven scheduling works (Mao et al. 2019; Qiu et al. 2020, 2022, 2023) to use Tanh for simple neural architectures. MinioinsRL also supports other activation units. We update the parameters of the scheduler policy using the Adam optimizer (Kingma and Ba 2014) with a learning rate of 0.005. MINIONSRL is trained with 100 episodes per task.

Evaluation

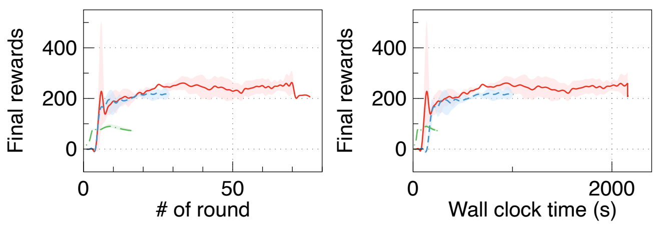
We prototype MINIONSRL on top of ACI (Azure Container Instances 2022). The implementation and code details are available in the Supplementary Material.

Experimental Setup

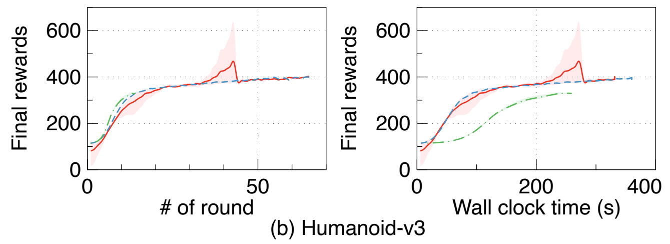
Testbeds. We deploy all server-based baselines to a cluster of Azure VMs: one `Standard_NC6s_v3` virtual machine (VM) and four `Standard_E16-8s_v5` VMs. The cluster contains one NVIDIA V100 GPU and four 8-core Intel Xeon Platinum CPUs (in total 32 cores) for training DRL



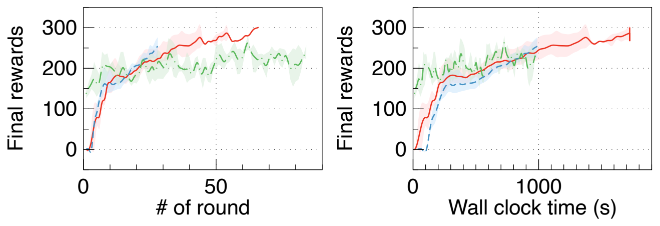
(a) Hopper-v3



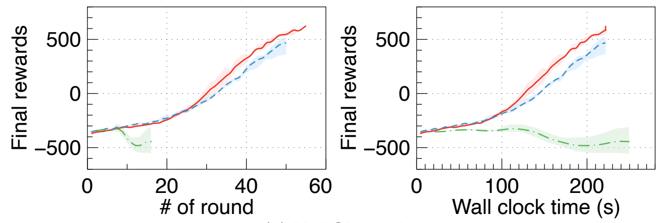
(d) GravitarNoFrameskip-v4



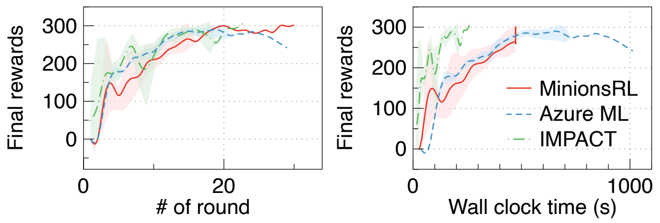
(b) Humanoid-v3



(e) SpaceInvadersNoFrameskip-v4



(c) HalfCheetah-v3



(f) QbertNoFrameskip-v4

Figure 4: MINIONSRL outperforms baselines on statistical and time efficiency for continuous (Hopper-v3, Humanoid-v3, and HalfCheetah-v3) and discrete control tasks (GravitarNoFrameskip-v4, SpaceInvadersNoFrameskip-v4, and QbertNoFrameskip-v4).

workloads. MINIONSRL is prototyped on Azure Container Instances (ACI) (Azure Container Instances 2022). When training DRL workloads with MINIONSRL, according to our workload profiling, each learner container is configured with one V100 GPU and each actor container is with one CPU core, respectively. Table 4 and 5 in the Supplementary Material characterize the unit price per hour of Azure VM and ACI instances in region `us-west-2` used by server-based baselines and MINIONSRL, respectively. We limit the actor allocation range of MINIONSRL within $[1, 32]$ during every training round.

Workloads. Six popular environments from OpenAI Gym are used to evaluate MINIONSRL and other baselines, including three continuous-action MuJoCo environments (Hopper-v3, Humanoid-v3, and HalfCheetah-v3) and three discrete-action Atari environments (SpaceInvadersNoFrameskip-v4, QbertNoFrameskip-v4, and GravitarNoFrameskip-v4). Table 3 in the Supplementary Material characterizes the policy networks used in our evaluation. For three MuJoCo environments, the policy network consists of two fully-connected layers of 256 hidden units with `Tanh` activation. For three Atari environments, the policy network consists of three convolutional layers of 8×8, 4×4, and 11×11 kernel sizes with `ReLU` activation, respectively. The input sampled from Atari games is a stack of three 84×84 images. In both cases, the critic networks share the same architecture as the policy networks. Due to superior performance and popularity (OpenAI 2017), we use PPO as the learner policy optimizer (shown in Fig. 3) for the above six workloads in the evaluation. Table 1 describes the hyperparameter settings of PPO used in training workloads. We used the default hyperparameters from Ray-RLlib (Liang et al. 2018) for Mujoco and Atari tasks. It's fair to compare baselines as long as using the same tasks. While evaluating on the six tasks, our solution is broadly applicable to DRL workloads with any reinforcement learning (RL) training algorithms and environments.

Comparisons with Baselines

We compare MINIONSRL with two server-based baselines: 1) **Azure ML** (Azure Machine Learning 2022) is a state-of-the-practice, ML-as-a-Service platform that provides rapid model deployment and training. Despite waiving the deployment and startup costs of DRL workloads, users are still charged with resource idle time during workload training, as demonstrated in Fig. 1. We implement the distributed PPO training method using the testbed cluster on Azure ML. 2) **IMPACT** (Luo et al. 2019) is a state-of-the-art actor-learner training architecture. IMPACT itself builds on a long list of improvements over PPO and combines various tricks for asynchronous training, such as V-trace importance sampling (Espeholt et al. 2018) and the surrogate target network (Lillicrap et al. 2015). We included IMPACT in our evaluation to investigate how MinionsRL compares with off-policy architectures.

Final rewards. Fig. 4 shows the final rewards averaged over five times of repeated experiments, each with a different ran-

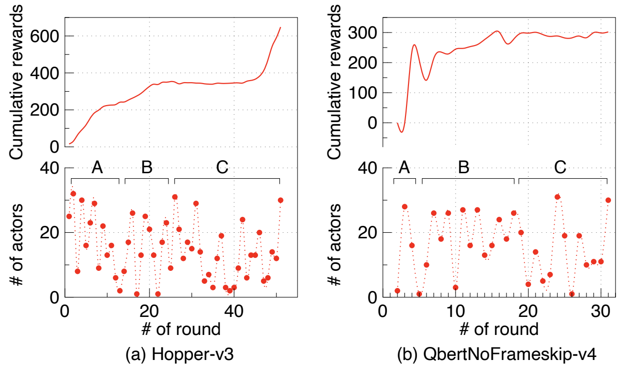


Figure 5: MINIONSRL's actor scheduling decisions on Hopper-v3 and QbertNoFrameskip-v4. MINIONSRL dynamically schedules actors to strike a balance between training performance and cost.

dom seed, for three continuous and three discrete control tasks, respectively. MINIONSRL and baselines are stopped if reaching the same target final reward or running out of the same budget. The performance variation is subtle from the beginning and gradually increases as training proceeds. The variation drops at the final parts because some of the five experiments have ended earlier (either reaching desired rewards or running out of budget). Thus, only one or two experiments proceed to further rounds/timestamps, leaving less variation—zero variation at the end if only one experiment remains. The results show that MINIONSRL is more efficient in transforming the monetary budget into training time. Under the same budget, MINIONSRL trains much faster than Azure ML and IMPACT in statistical efficiency and wall clock time while achieving similar or better performance.

Training cost. Table 2 reports the total training time and costs for six tasks when baselines reached the same final rewards. Compared to Azure ML and IMPACT, MINIONSRL reduces training time and costs at most by 52% and 86%, respectively.

Actor Scheduling

We record and report how MINIONSRL makes actor scheduling decisions to investigate the rationale behind the performance gain compared with the baselines. Fig. 5 depicts the number of actors MINIONSRL schedules and final rewards per round on Hopper-v3 and QbertNoFrameSkip-v4, respectively. We use A, B, and C for convenience when referring to the three phases of decisions made by MINIONSRL in Fig. 5. For Hopper-v3, MINIONSRL launches more actors at the beginning of Phase A to boost training and gradually decreases the number of actors to save cost when performance steadies in Phase B and C. More actors are launched by MINIONSRL at the end of Phase C to explore optimal performance. We observe similar results on QbertNoFrameSkip-v4, where MINIONSRL boosts training with more actors in Phase A and B, and reduces actors in steady Phase C to save cost.

In contrast to two baselines (*i.e.*, Azure ML and IMPACT) that launch a fixed number of actors for every round, MINIONSRL dynamically schedules actors throughout the training process to strike a balance between training performance and cost, thus completing training tasks cheaper and faster.

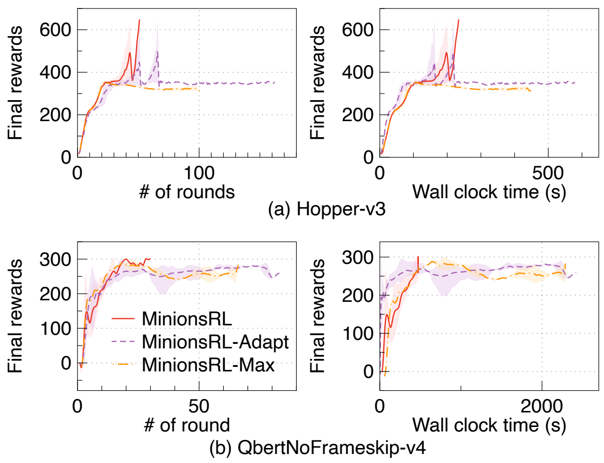


Figure 6: Ablation study of MINIONSRL with its two variants: MINIONSRL-Adapt and MINIONSRL-Max.

Ablation Study

To verify the effectiveness of two key components: serverless functions and DRL-based scheduler, we compare MINIONSRL with two variants of itself: 1) **MINIONSRL-Max** statically launches all 32 actors in every training round, and 2) **MINIONSRL-Adapt** schedules actors with a naive, reward ratio-based scheduler. Let J denote the target final reward and I_{max} denote the maximum number of available actors per round. Let $\hat{J}_k$ denote approximated final reward that the learner policy can achieve at round k, which is computed using a moving window averaged over the last n rounds given by $\hat{J}_k := \sum_{x=k-n-1}^{k-1} J_x$. MINIONSRL-Adapt schedules a set of actor functions I_k proportional to the ratio of reward $\hat{J}$ and J, which is given by $I_k := \text{clip}(1, \frac{\hat{J}_k}{J} I_{max}, I_{max})$. This naive scheduler follows the intuition that a better policy may produce better data, so we proportionally allocate more actors when the policy quality is higher. We set the moving window size $n = 5$ in the evaluation.

Final rewards. Fig. 6 shows the final rewards averaged over five times of repeated experiments for Hopper-v3 and QbertNoFrameskip-v4, respectively. By comparing MINIONSRL with MINIONSRL-Max, we can observe that MINIONSRL's DRL-based scheduler can preserve similar or better training efficiency while saving actor costs. Note that MINIONSRL-Max also runs the same DRL tasks with serverless functions. When comparing MINIONSRL with MINIONSRL-Adapt, the results demonstrate that MINIONSRL's DRL-guided scheduler makes better decisions on actor scheduling than the naive ratio-based scheduler.

Training cost. Table 2 shows MINIONSRL's the total training time and costs and the two variants when reaching the same final rewards. Compared to MINIONSRL-Max, MINIONSRL significantly reduces training cost by up to 44% while completing training with a similar duration.

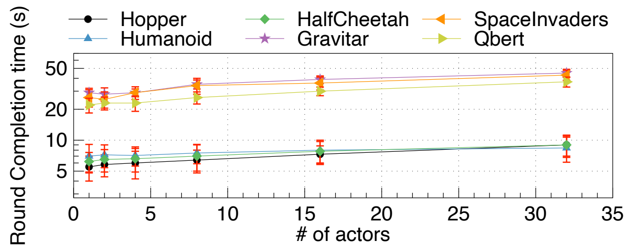


Figure 7: Scalability of MINIONSRL with respect to the number of actors in six environments.

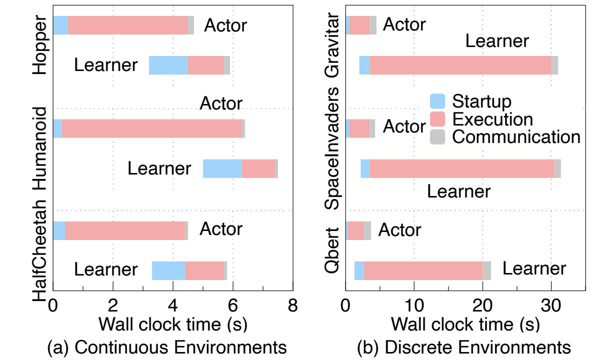


Figure 8: Latency breakdown of interaction between actor and learner function in MINIONSRL's one-round training.

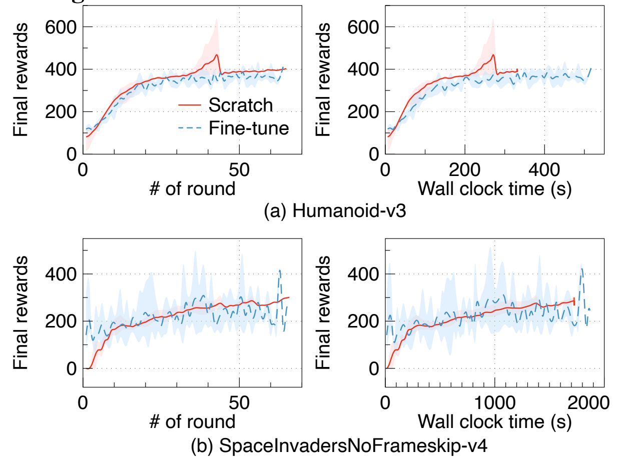


Figure 9: Training the scheduler from scratch *v.s.* fine-tuning the scheduler trained from a different task.

Scalability

Fig. 7 illustrates MINIONSRL's scalability using the same testbed. The total completion time of one-round DRL training increases as the number of actors increases. Training time for Atari environments (*i.e.*, GravitarNoFrameskip-v4, SpaceInvadersNoFrameskip-v4, and QbertNoFrameskip-v4) has a larger increase rate than Mujoco environments (*i.e.*, Hopper-v3, Humanoid-v3, HalfCheetah-v3), because processing stacked frames brings significantly more computation load to the learner.

Latency Breakdown

Latency breakdown. Fig. 8(a) and (b) characterize the latency breakdown of interaction between actor and learner function in MINIONSRL's one-round training. Launching an actor and learner function takes around 300 and 1500 ms (attaching GPUs to the learner container takes more time), respectively. We further eliminate the startup overhead by function pre-warming.

Communication overheads. MINIONSRL uses the efficient gRPC library to enable lightweight communication between actor and learner functions . Fig. 8(a) and (b) show the communication overhead between actor and learner functions. For (continuous) Mujoco environments, transferring 65,536 timesteps between actor and learner function incurs less than 100 ms communication overhead. For (discrete) Atari environments, the overhead is less than 800 ms for 6,144 stacked frames. The communication overheads are trivial compared to the end-to-end training time per round.

Scheduler Training Overhead Mitigation

MINIONSRL trains the scheduler for each DRL task, which may lead to high overheads. For example, training a scheduler for Humanoid-v3/SpaceInvadersNoFrameskip-v4 from scratch took around 10/50 hours. We further investigate mitigating such overheads by fine-tuning a trained scheduler of one task to other tasks. Fine-tuning MINIONSRL is feasible since different DRL tasks have the same observation and action shapes as input and output sizes so that scheduler networks can adapt to similar tasks. Fig. 9 presents the performance of training MINIONSRL from scratch and fine-tuning from another task. We fine-tune trained schedulers of Hopper-v3 and QbertNoFrameskip-v4 to Humanoid-v3 and SpaceInvadersNoFrameskip-v4, respectively. Fine-tuning each scheduler took ten episodes while achieving similar or better performance than training from scratch. More importantly, fine-tuning drastically reduces scheduler training time and cost. It only took around one/four hours to fine-tune a scheduler for Humanoid-v3/SpaceInvadersNoFrameskip-v4, reducing the scheduler training time and cost by 90%. Despite training from scratch and fine-tuning incurring hour-long overheads, this one-time training overhead is acceptable since the trained scheduler can be directly reused at no training overhead for accelerating the same (or similar) environments with varying target rewards or monetary budgets for long-term benefits.

Conclusion

We proposed MINIONSRL, the first distributed DRL training framework based on serverless computing. By leveraging serverless computing, MINIONSRL enables agile autoscaling and fine-grained resource provisioning to extensively mitigate resource wasting during distributed DRL training. To accelerate training- and cost-efficiency, we designed a DRL-driven scheduler to seek the optimal number of actors by learning the fundamental trade-off between training performance and cost. We evaluated MINIONSRL on realistic clusters with popular tasks from OpenAI Gym. Experimental results show that MINIONSRL outperforms state-of-the-art and state-of-the-practice solutions by reducing up to 52% total training time and 86% training cost.

References

Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained Policy Optimization. In *International Conference on Machine Learning (ICML)*.

Ali, A.; Pinciroli, R.; Yan, F.; and Smirni, E. 2020. BATCH: Machine Learning Inference Serving on Serverless Platforms with Adaptive Batching. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE.

Azure Container Instances. 2022. Azure Container Instances. https://azure.microsoft.com/en-us/products/container-instances/. [Online; accessed 1-Jan-2022].

Azure Machine Learning. 2022. Azure Machine Learning. https://azure.microsoft.com/en-us/products/machine-learning/. [Online; accessed 1-Jan-2022].

Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv preprint arXiv:1912.06680*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540*.

Carreira, J.; Fonseca, P.; Tumanov, A.; Zhang, A.; and Katz, R. 2019. Cirrus: a Serverless Framework for End-to-end ML Workflows. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*.

Chard, R.; Babuji, Y.; Li, Z.; Skluzacek, T.; Woodard, A.; Blaiszik, B.; Foster, I.; and Chard, K. 2020. FuncX: A Federated Function Serving Fabric for Science. In *Proc. of the 29th International Symposium on High-performance Parallel and Distributed Computing (HPDC)*, 65–76.

Devarakonda, A.; Naumov, M.; and Garland, M. 2017. AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks. *arXiv preprint arXiv:1712.02029*.

Espeholt, L.; Marinier, R.; Stanczyk, P.; Wang, K.; and Michalski, M. 2020. Seed RL: Scalable and Efficient Deep-RL with Accelerated Central Inference. In *International Conference on Learning Representations (ICLR)*.

Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *International Conference on Machine Learning (ICML)*.

Gu, S. S.; Lillicrap, T.; Turner, R. E.; Ghahramani, Z.; Schölkopf, B.; and Levine, S. 2017. Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation for Deep Reinforcement Learning. *Advances in Neural Information Processing Systems (NIPS)*.

Guo, R.; Guo, V.; Kim, A.; Hildred, J.; and Daudjee, K. 2022. Hydrozoa: Dynamic Hybrid-Parallel DNN Training on Serverless Containers. *Proceedings of Machine Learning and Systems (MLSys)*.

Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2018. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *Thirty-second AAAI conference on artificial intelligence (AAAI)*.

Horgan, D.; Quan, J.; Budden, D.; Barth-Maron, G.; Hessel, M.; Van Hasselt, H.; and Silver, D. 2018. Distributed Prioritized Experience Replay. *arXiv preprint arXiv:1803.00933*.

Ji, Y.; Li, Z.; Sun, Y.; Peng, X. B.; Levine, S.; Berseth, G.; and Sreenath, K. 2022. Hierarchical Reinforcement Learning for Precise Soccer Shooting Skills using a Quadrupedal Robot. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Jumper, J.; Evans, R.; Pritzel, A.; Green, T.; Figurnov, M.; Ronneberger, O.; Tunyasuvunakool, K.; Bates, R.; Žídek, A.; Potapenko, A.; et al. 2021. Highly Accurate Protein Structure Prediction with AlphaFold. *Nature*.

Kapturowski, S.; Ostrovski, G.; Quan, J.; Munos, R.; and Dabney, W. 2018. Recurrent Experience Replay in Distributed Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*.

Kingma, D. P.; and Ba, J. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.

Liang, E.; Liaw, R.; Nishihara, R.; Moritz, P.; Fox, R.; Goldberg, K.; Gonzalez, J.; Jordan, M.; and Stoica, I. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In *International Conference on Machine Learning (ICML)*.

Liaw, R.; Liang, E.; Nishihara, R.; Moritz, P.; Gonzalez, J. E.; and Stoica, I. 2018. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint arXiv:1807.05118*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous Control with Deep Reinforcement Learning. *arXiv preprint arXiv:1509.02971*.

Luo, M.; Yao, J.; Liaw, R.; Liang, E.; and Stoica, I. 2019. IMPACT: Importance Weighted Asynchronous Architectures with Clipped Target Networks. *arXiv preprint arXiv:1912.00167*.

Mao, H.; Schwarzkopf, M.; Venkatakrishnan, S. B.; Meng, Z.; and Alizadeh, M. 2019. Learning Scheduling Algorithms for Data Processing Clusters. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

Mao, W.; Qiu, H.; Wang, C.; Franke, H.; Kalbarczyk, Z.; Iyer, R.; and Basar, T. 2022. A Mean-field Game Approach to Cloud Resource Management with Function Approximation. *Advances in Neural Information Processing Systems (NIPS)*.

McCandlish, S.; Kaplan, J.; Amodei, D.; and Team, O. D. 2018. An Empirical Model of Large-batch Training. *arXiv preprint arXiv:1812.06162*.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *International Conference on Machine Learning (ICML)*.

Moritz, P.; Nishihara, R.; Wang, S.; Tumanov, A.; Liaw, R.; Liang, E.; Elibol, M.; Yang, Z.; Paul, W.; Jordan, M. I.; et al.

2018. Ray: A Distributed Framework for Emerging AI Applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

OpenAI. 2017. Proximal Policy Optimization. https://openai.com/blog/openai-baselines-ppo/. [Online; accessed 1-Jan-2022].

OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774.

Qiu, H.; Banerjee, S. S.; Jha, S.; Kalbarczyk, Z. T.; and Iyer, R. K. 2020. FIRM: An Intelligent Fine-grained Resource Management Framework for SLO-Oriented Microservices. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

Qiu, H.; Mao, W.; Patke, A.; Wang, C.; Franke, H.; Kalbarczyk, Z. T.; Başar, T.; and Iyer, R. K. 2022. SIMPPO: A Scalable and Incremental Online Learning Framework for Serverless Resource Management. In *Proceedings of the 13th Symposium on Cloud Computing*.

Qiu, H.; Mao, W.; Wang, C.; Franke, H.; Youssef, A.; Kalbarczyk, Z. T.; Basar, T.; and Iyer, R. K. 2023. AWARE: Automate Workload Autoscaling with Reinforcement Learning in Production Cloud Systems. In *2023 USENIX Annual Technical Conference (USENIX ATC)*.

Roy, R. B.; Patel, T.; Gadepally, V.; and Tiwari, D. 2022. Mashup: Making Serverless Computing Useful for HPC Workflows via Hybrid Execution. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 46–60.

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust Region Policy Optimization. In *International Conference on Machine Learning (ICML)*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*.

Thorpe, J.; Qiao, Y.; Eyolfson, J.; Teng, S.; Hu, G.; Jia, Z.; Wei, J.; Vora, K.; Netravali, R.; Kim, M.; et al. 2021. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

Thumm, J.; and Althoff, M. 2022. Provably Safe Deep Reinforcement Learning for Robotic Manipulation in Human Environments. In *2022 International Conference on Robotics and Automation (ICRA)*.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster Level in StarCraft II Using Multi-agent Reinforcement Learning. *Nature*.

Wang, H.; Niu, D.; and Li, B. 2019. Distributed Machine Learning with a Serverless Architecture. In *IEEE 2019 Conference on Computer Communications (INFOCOM)*.

Wijmans, E.; Kadian, A.; Morcos, A.; Lee, S.; Essa, I.; Parikh, D.; Savva, M.; and Batra, D. 2019. DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames. *arXiv preprint arXiv:1911.00357*.

Yu, M.; Jiang, Z.; Ng, H. C.; Wang, W.; Chen, R.; and Li, B. 2021. Gillis: Serving Large Neural Networks in Serverless Functions with Automatic Model Partitioning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE.