# SERAPH: A Performance-Cost Aware Tuner for Training Reinforcement Learning Model on Serverless Computing


Jinbo Han
Institute of Parallel and Distributed Systems, SEIEE, Shanghai Jiao Tong University
Shanghai, China

Xingda Wei*
Institute of Parallel and Distributed Systems, SEIEE, Shanghai Jiao Tong University
Shanghai, China

Rong Chen
Institute of Parallel and Distributed Systems, SEIEE, Shanghai Jiao Tong University
Shanghai, China

Haibo Chen
Institute of Parallel and Distributed Systems, SEIEE, Shanghai Jiao Tong University
Shanghai, China



## Abstract

Training a reinforcement learning model is critical for various AI tasks. However, determining the hardware resources required for training RL models is challenging due to the interaction between the CPU and GPU, and the variability that exists in the training. The problem becomes more challenging when deploying an RL training job on the cloud with serverless computing, as we should consider both the performance and cost of training RL models. Existing tuners, like Ray Tune, require users to provide a search space. It is both error-prone and unable to search the setup with the desired cost. We present SERAPH, the first tuner for RL training that finds the hardware configuration with the best performance within a user-given cost boundary. SERAPH explicitly models the performance of training by decomposing RL training and using a stochastic model to harness variability. Compared to Ray Tune, it finds the optimal with 71% tuning time reduction.



## CCS Concepts

• **Computer systems organization** → *Cloud computing*.



*Corresponding author: Xingda Wei, wxdwfc@sjtu.edu.cn





*APSys '24, September 4–5, 2024, Kyoto, Japan*





ACM ISBN 979-8-4007-1105-3/24/09
https://doi.org/10.1145/3678015.3680479

## Keywords

Systems for machine learning, Reinforcement Learning, Serverless Computing

**ACM Reference Format:**
Jinbo Han, Xingda Wei, Rong Chen, and Haibo Chen. 2024. SERAPH: A Performance-Cost Aware Tuner for Training Reinforcement Learning Model on Serverless Computing. In *ACM SIGOPS Asia-Pacific Workshop on Systems (APSys '24), September 4–5, 2024, Kyoto, Japan.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3678015.3680479


## 1 Introduction

Reinforcement learning (RL) [17] has shown emerging capabilities in various tasks like gaming and control. Unlike supervised training, the training of RL is more complex, as its training data is generated by sampling with the currently trained model. For example, at each iteration (round), the RL training system requires a set of CPU workers (actors) to generate new training data. The training data is then fed to the GPUs for training the model.

It is challenging for developers to determine the right hardware configurations for training a model with RL: during training, the CPUs interact extensively with the GPUs, and GPU efficiency may depend on the CPU. If an insufficient number of CPUs are provisioned, the GPU will have to wait for the CPU during the sampling process. On the other hand, if an insufficient number of GPUs are deployed, the CPU will be idle while waiting for the model to train. Worse even, variability exists in model training and sampling—the sample time changes when the model is updated, making it difficult for us to capture the relationship between hardware resources and performance. Finally, the developers must consider the cost when running on the cloud. If too many CPUs/GPUs are provisioned, then they must pay for this.

Existing tuners for RL like Ray tune don't model the capital cost, and it requires developers to define the search space, which is error-prone. For example, with a half search space, the configuration generated by the Ray tune can result in

1.6× more training time compared to the optimal. Moreover, it can generate a configuration with 1.3× cost compared to minimal resource configuration each round.

The challenge of resource configuration intensifies when considering serverless, a new cloud paradigm suitable for training RL models [19, 20]. Specifically, in serverless computing, the CPU can be provisioned on-demand to save costs when the job is idle, such as when waiting for the GPU to finish. However, serverless introduces factors such as cold starts, making performance modeling and configuration harder.

In this paper, we present Seraph, a tuner for tuning hardware configurations for training models with RL. Instead of requiring the user to provide the search space, we directly model the performance of RL training, inspired by recent works on hardware configurations for the cloud applications [8, 21]. The modeling is based on two observations. First, RL training algorithms have a phase-to-phase pattern, which we can clearly decompose the interaction. Second, we can model the variability via stochastic variables through Monte Carlo sampling. Based on the model, we can add the cost constraint to solve the model's Pareto front [3]: the hardware configuration within the cost bound that has the best performance. The problem can be quickly solved by leveraging its convexity.

We have implemented Seraph on Ray. Compared to the Ray Tune, the default hardware configuration tuner in Ray, we can find the optimal performance hardware configuration 71% faster, without requiring developers to provide a search space. Meanwhile, we can find a configuration within the cost boundary, while the Ray Tune may generate a setup that exceeds the budget.

In summary, our paper made the following contributions:

- We present the first systematic tuner for training RL.
- We first model the performance of training RL that captures the relationship between hardware configurations and the training performance.
- We implement Seraph on Ray with extensive evaluations showing the accuracy of our model and the effectiveness of the tuner.

## 2 Background and Motivation

**RL training with serverless.** Training an RL model differs from training traditional ML models. Data is generated by sampling, e.g., playing a game with the trained model, and then using the game result as the training data [15]. On-policy reinforcement algorithms like PPO [14] have low CPU utilization and waste user's money, actors who sample the training data have to wait for the learners to update policy. The key pushing hands of running RL training with serverless is two-fold. First, the number of actors needed changes dynamically during the training process [4, 7, 12, 16, 20], so

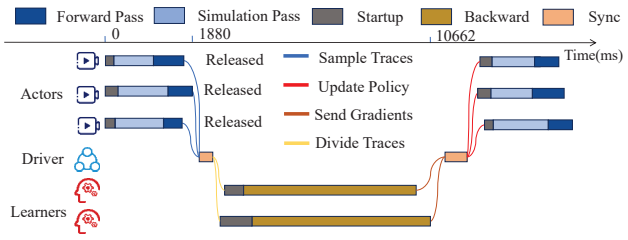



**Figure 1: An illustration of distributed reinforcement learning (RL) with serverless.**

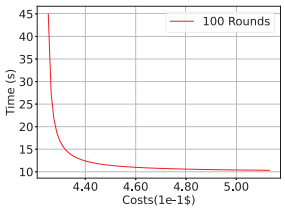



**Figure 2: Pareto front of Cost and performance**

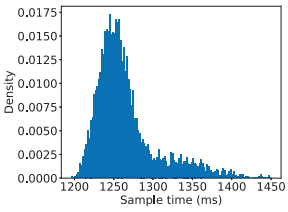



**Figure 3: Time distribution of sample phase**

serverless can quickly adjust the resources according to the variance. Second, during training, the sampler's CPU is free when the GPU is running (Figure 1), so running the sampler with serverless can save cost thanks to its pay-as-you-go feature [19, 20].

Figure 1 presents the architecture of the training RL models with serverless, where we term the GPU that trains the model the learner (The learners (GPUs) are provisioned in a serverful way). The train is carried in iterations (rounds). Before starting a round, the system will spawn actors to sample the data given the current trained model. Afterward, these actors will be released to save costs.

**Resource configuration for cost-efficient RL training.** Configuring the number of resources (e.g., GPUs, CPUs) is critical in training RL models [5, 9]. For example, allocating more GPUs or CPUs can improve the training efficiency. However, more resources also increase the operational cost when deploying applications on the cloud. For instance, training a model with 100 rounds with one actor to sample requires 4500 seconds and 0.42$, while using 41 actors requires about 1,000 seconds and 0.46$ [1].

**Resource configuration and Pareto optimal.** To enable cost-efficient training, it is desirable to find the configuration (e.g., the number of GPUs required, or the number of vCPUs, see Table 1) that achieves the Pareto optimal [3] of training. Specifically, the Pareto optimal is the minimal training time given the cost boundary, as shown in Figure 2. Finding the

[1] Measured on Figure 2.

configuration for Pareto optimal in training RL, however, is challenging even without serverless. Specifically, it is hard to model the configuration-to-performance relationship. First, as Figure 3 shows, the sample time has variance due to the randomness of the actor's action. Second, the communication time between actors and the server is relevant to hardware configuration. Integrating RL with serverless further makes the problem more complex because (1) The serverless infrastructure will add additional costs like cold start that needs to be modeled and (2) the billing policy is more complex.

**Existing works.** The closest to our work is Ray Tune [11], a tuning toolkit that uses a measurement-based search method to find the best configuration given a (developer-provided) search space. Specifically, Ray Tune requires developers to provide a set of configurations (e.g., number of GPUs) to search, and it will measure the performance of each configuration to find the best one. Such a design has two limitations. First, it cannot find the Pareto optimal of cost and performance if the user passes an incomplete set of configurations, which is the common case. Second, the search time is long because it has to run each configuration to find the best one.

## 3 The design of SERAPH

**Design goals.** To facilitate developers finding the right resource configuration for training RL models in a serverless environment, we design SERAPH with the following goals:

- **Automatic configuration generation**: Unlike Ray tune, we don't require users to specify the search space. SERAPH can automatically find the optimal configurations for training RL models on serverless.
- **Finding the resource configuration for Pareto optimal**: We aim to find the configuration that considers both the training efficiency and cost, i.e., Pareto optimal.

**Approach overview.** SERAPH finds the optimal resource configuration by solving the optimal parameters of the the performance model of the RL training. Specifically, we derive a performance model that models the training time concerning the resources used. To use SERAPH to find the resource configuration (the number of GPUs and CPUs to use), the developers only need to pass the cost constraint to the system, and we solve the model to find the resource configuration within the cost boundary, and with the best performance. Compared with Ray Tune, developers don't need to specify the search space, and we can consider the cost by explicitly modeling the cost in the model.

**Challenge and solution.** Our model needs to precisely capture the relationship between the resources and the RL training time. The challenge here is the training time is a complex function of the resources since the CPU and GPU interact in a complex way. Meanwhile, the execution has variances due to factors like cold start and communication time. We address this using stochastic variables to represent the relationship between time and hardware configuration.

Second, we need to quickly find the optimal resource configuration within the cost boundary. Finding the optimal resource configuration in a naive performance model is time-consuming. To this end, we simplify the model as a convex optimization problem and use gradient descent to solve it quickly.

**Table 1: Constant and variables to configure in training**

| Symbol | Explanation | Attribute |
|---|---|---|
| $v$ | vCPU number | Solved |
| $u$ | GPU number | Solved |
| $m$ | Training batch size | Constant |
| $\alpha$ | Cost for renting one vCPU/s | Constant |
| $\beta$ | Cost for renting one vGPU/s | Constant |
| $\gamma$ | Cost for renting one GB/s | Constant |
| $\epsilon$ | Cost for one serverless invoke | Constant |
| $T_{inits}$ | Actor initialization time | Variable |
| $T_t$ | Transmission time | Variable |
| $T_s$ | Weight synchronization time | Variable |
| $A_i, B_i$ | Fitting parameter | Parameter |

### 3.1 The performance model

Observing that the RL training executes in phases sequentially (Figure 1), our performance model characterizes the performance of three sub-phases in one round of RL training with respective to the resource: the *sampling*, *training* and *update* phase. Besides, we also need to model the phase for *serverless initialization*. We use $T_{inits}$ to denote the serverless function startup cost. Table 1 shows a full list of constants and parameters solved with SERAPH, The parameter is a linear model's parameters to profile sample time and training time.

**Model the sampling phase.** The sampling phase includes two parts: (1): Actor initialization. (2): Actor sampling. For the initialization phase, we use a stochastic variable $T_{inits}$ to represent the initialization time of the serverless function and reinforcement environment. For the sampling phase, assuming each actor is corresponding to one vCPU, batch size is $m$, each actor will have $\frac{m}{v}$ length data to sample. We use a linear model to profile the relationship between sampling time and data length $x$.

$$T_r(x, i) = A_1 * x + B_1 \tag{1}$$

In Equation 1, $A_1$, $B_1$ are random variables, we can fit them using historical data.

The total sample time $T_r$ will be the maximum time of these actors:

$$T_r = \max\{T_{inits}(i) + T_r(\frac{m}{v}, i)\}, i = 1 \dots v \tag{2}$$

**Model the training phase.** The training phase includes two parts: (1) data transmission (2) learner training. We use a stochastic variable $T_t$ to represent the data transmission time. Training data will be divided into $u$ learners equally, each learner is corresponding to one GPU. Let $T_u(y, i)$ be the training time of learner $i$. The data length processed by learner $i$ is $y$, we still use linear model:

$$T_u(y, i) = A_2 * y + B_2 \tag{3}$$

Similarly, $A_2$, $B_2$ are stochastic variables, fitting with historical data. The training time $T_u$ will be the maximum time of these learners:

$$T_u = \max\{T_u(\frac{m}{u}, i)\}, i = 1 \dots u \tag{4}$$

**Model the sychnoziation phase.** After all the learners have trained their data, the weights will be gathered and sent to every actor, ready for the next iterations' sampling. The synchronization time $T_s$ is still a stochastic variable. The end-to-end time $T_{tot}$ will be the sum of these times.

$$T_{tot} = T_r + T_t + T_u + T_s \tag{5}$$

**Model the memory constraint.** A serverless function typically has a memory limit. Assuming the memory limit corresponding to each core is $M_v$, and the GPU memory bound is $M_u$. The memory usage of each actor and learner is $M_a, M_l$. We need to ensure the memory usage does not exceed the limit, i.e.,$M_a \le M_v, M_l \le M_u$. Luckily, most workloads use few memory. For example, the memory usage of the sampling SpaceInvader environment is about 1 GB, which is far less than the 2 GB minimal function memory in the Alibaba Cloud Environment.

**The cost model.** Finally, we need to consider the cost in our model as a constraint, which is determined by the number of CPUs and GPUs rented. Assume the average cost of renting one vCPU is denoted by $\alpha$ per second, renting one GPU server costs $\beta$ per second, renting one GB of memory in serverless costs $\gamma$ per second, and invoking a function incurs a cost of $\epsilon$ per invocation. $\alpha, \beta, \gamma, \epsilon$ are platform-specific and hardware-specific, which remain fixed within a particular cloud platform context. The total cost of the training task $O_{tot}$ will then be:

$$\begin{aligned} O_{tot} = (\Sigma_{i=1}^{v} T_r(\frac{m}{v}, i) * \alpha + \Sigma_{i=1}^{v} T_r(\frac{m}{v}, i) * \gamma * M_a * v \\ + v * \epsilon) + T_{tot} * \beta \end{aligned} \tag{6}$$

## 3.2 Performance model simplification and formulation

All stochastic variables $T_s, T_t$ can be determined after sampling. Through multiple samplings in specific hardware configurations, we use the average value of these variables to replace these stochastic variables. For linear-regression parameters $A_1$, $B_1$, $A_2$, and $B_2$, we can fit them based on historical data. With these simplifications, the problem transforms into a deterministic one:

$$\begin{aligned} &Minimize \quad T_{tot} \\ &s.t. \quad O_{tot} <= \Omega, M_a \le M_v, M_l \le M_u \end{aligned} \tag{7}$$

$\Omega$ is a cost budget provided by the user/developer.

## 3.3 Convex optimizer

We observe that the optimization problem Equation 7 is convex. Convex optimization problem has an important advantage, local extrema coincides with global extrema. We can use a gradient descent algorithm to find the optimal solution.

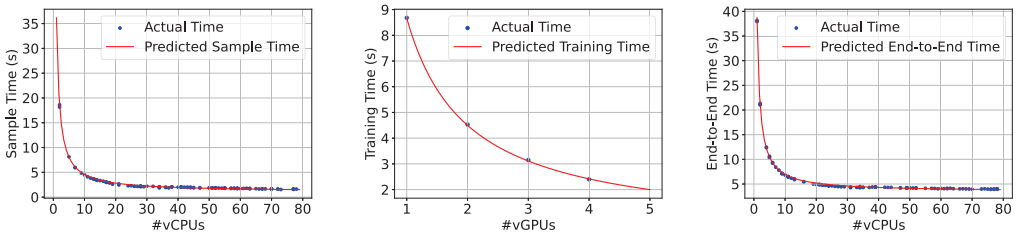
**Convex analysis.** The convex optimization problem can be quickly resolved using a gradient descent algorithm. Luckily, $T_{tot}$ and $O_{tot}$ are both convex functions in our setup. Due to space limitations, we skip the detailed proof. At a high level, because the maximum or sum of some convex functions is still convex, we only need to prove $T_r(\frac{m}{v}, i)$ and $T_u(\frac{m}{u}, i)$ are convex. Ignore the $i$ notation because it is irrelevant to the whole convexity. Then we need to prove $T_r(v)$ and $T_u(u)$ are convex.

$T_r(v)$ and $T_u(u)$ are convex because the Hessian matrix are positive-definite. $T_t$ and $T_s$ are fixed after sampling. So $T_{tot}$ is convex. The polynomial item $f(v) = v * \epsilon$ is convex trivially, the multiplier $\alpha$ and $\beta$ are both positive, so $O_{tot}$ is also convex.

**Solving the problem.** Since the optimization problem is convex, we employ a gradient descent algorithm for its solution. More specifically, we first apply the gradient descent algorithm within the solution space until convergence is achieved. The problem here is that the solved variables $v$ and $u$ may be real, but they must be integer as they determine the resources needed for the RL training. Therefore, we further enumerate the rounded solutions around the optimal solution given by gradient descent. We select some integer hardware configurations around the optimal, calculate the corresponding cost and performance and find the configuration with the lowest training time under the cost boundary.

# 4 Evaluation

**Implementaiton.** We integrate our tuner into Ray 2.8.0 with 5547 LoC. It can serve as a drop-in replacement of the Ray Tune. The integration mainly includes samplers to measure




(a) Sample Stage (b) Training Stage (c) End-to-End Time

**Figure 4: Effectiveness of Seraph's performance model for SpaceInvader**

**Table 2: Billing policy**

| Item | Price ($) per second |
|---|---|
| 1 vCPU core in SAE | 7.095e-6 |
| 1 GB in SAE | 1.773e-6 |
| 1 T4 GPU Server with 4 vCPUs | 4.49e-4 |
| 1 T4 GPU Server with 40 vCPUs | 5.65e-4 |

**Table 3: Mean average percentage error of prediction**

| | SpaceInvader | Breakout | Beamrider | Qbert |
|---|---|---|---|---|
| Sample | 4.55% | 3.93% | 5.25% | 3.59% |
| Training | 0.66% | 1.13% | 0.72% | 1.07% |
| End-to-End | 2.51% | 1.81% | 2.06% | 2.21% |

the time taken to sample and train given the number of CPUs and GPUs. Besides, it also includes a solver for finding the configurations.

**Evaluation setup.** To show the effectiveness of our approach, we evaluate Seraph on two testbeds on Elastic Compute Service (ECS) and Serverless Application Engine (SAE) of Alibaba Cloud—a major cloud vendor that supports serverless. For evaluating the effectiveness of our performance model, our testbed have a server with 4 × Nvidia Tesla V100-SXM2-16GB GPUs. For evaluating the cost effectiveness, we use a T4 GPU[2].

**Evaluated applications.** We evaluate Seraph by training four typical RL applications (SpaceInvaders-v4, Qbert-v4, BeamRider-v4, and Breakout-v4) from ALE [1]. We employ the Proximal Policy Optimization (PPO) [14] algorithm—the state-of-the-art method for training RL models. We use the default hyperparameters of PPO in Ray [10].

## 4.1 Effectiveness of our performance model

To show the effectiveness of Seraph's performance model, we gather data on the time taken for the sampling stage, training stage, and end-to-end stage, then compare these metrics with the predictions provided by Seraph. Figure 4 illustrates the actual time and Seraph's prediction about SpaceInvader. The results of the other three applications are not because of the space limit, whose results are similar. The number of vCPUs are set to the number of actors. We can see that our performance model has remarkable accuracy in predicting the actual time. Specifically, Table 3 presents the mean absolute percentage error (MAPE) across these four environments. The end-to-end stage MAPE are 2.51%, 1.81%, 2.06%, 2.21% for these applications. BeamRider exhibits slightly higher variance compared to the other three environments in sample stage, since it has the highest action space dimension, which introduces more randomness.

## 4.2 Comparison with Ray Tune

**Performance.** We compare the cost and training speed under different tuned configurations of Seraph and Ray Tune. Figure 6a illustrates the cost and performance Pareto front and prediction of Seraph and Ray Tune. We set the cost boundary as [0.399$, 0.40$, 0.44$] respectively, covering a range of low and high cost. We also set the search space of Ray Tune as vCPUs [5, 15, 30, 45, 60, 70] and GPUs [1, 2, 3, 4]. We can see that Ray tune can search the best performance configuration in this space but it neglects the cost boundary, while Seraph finds the configurations for the Pareto optimal under each cost boundary.

**Tuning time.** Figure 6b further shows the execution time for tuning. Seraph find the same performance optimal with 71% time reduction than Ray Tune, thanks to the more efficient model-based sampling technique. The majority of execution time of Seraph is sampling phase, while the optimization program solving time is about 10ms which can be neglected.

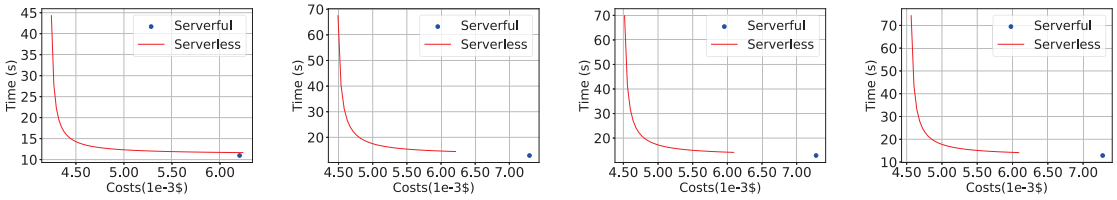
## 4.3 Cost-Performance pareto front

We calculate the cost and time of two strategies. First, we train the model using one server with 40 CPUs. Second, we train the model using one server with 4 CPUs and an additional 40 vCPUs of serverless application engine (SAE). Table 2 shows the billing policy. In the second strategy, the startup, release time of SAE are free according to the policy.

[2]We were unable to rent the V100 GPUs on the Alibaba Cloud during these evaluations.

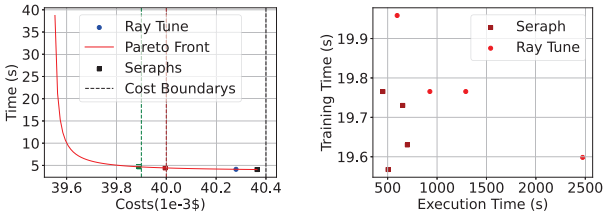
**Table 4: Cost and performance of serverful and serverless in one round**

| Environment | Serverful time (s) | Serverful cost($) | Serverless time(s) | Serverless cost($) |
|---|---|---|---|---|
| SpaceInvader | 10.97 | 0.0062 | 12.61 | 0.0048 |
| Qbert | 12.87 | 0.0073 | 14.77 | 0.0056 |
| Breakout | 12.90 | 0.0073 | 14.81 | 0.0058 |
| BeamRider | 12.88 | 0.0073 | 14.81 | 0.0059 |




**(a) SpaceInvaders-v4** **(b) Breakout-v4** **(c) BeamRider-v4** **(d) Qbert-v4**

**Figure 5: Cost and performance pareto front predicted by Seraph in two strategies**




**(a) Cost and time** **(b) Sample and training time**

**Figure 6: Comparison between Seraph and Ray Tune**

Figure 5 illustrates that training a reinforcement learning model with a single high number of CPUs server is faster but more expensive than training with a single low number of CPUs server while utilizing serverless functions for sampling. Mixed use can reduce the cost by at most 21.9% with the training time increased by only 1.6s. This is because actors in serverless functions will be released immediately and not count for cost while incurring additional initial time and network communication time. Thus, opting for a low number of CPUs server and leveraging serverless functions for sampling is advisable when developers have strict cost constraints but more flexibility in time. The detail cost and time are shown in Table 4.

## 5 Discussion

**Generality of the work.** All on-policy algorithms (like PPO, A2C, A3C [13]) follow same phase-to-phase pattern. Actors interact with learners synchronously in each round. Our performance model and solving method can be directly applied to any of them after setting hyperparameters and re-profiling. For some off-policy algorithms like IMPALA [6], the performance model should be modified since the training phase is not dependent on the sampling phase but overlaps. While the number of actors and learners still needed to be configured wisely to get the most cost-efficient training, we can develop a simulator to simulate the training time and cost in this scenario. We leave these as our future work.

**Handling memory-intensive workloads.** ALE [1], Box2D [2] and MuJoCo [18] environments consume a small amount of memory which does not exceed the memory constraint of a core in our measurement. For other memory-intensive workloads, we can rent the minimal memory instance to run an environment as an actor and scale the number of actors by replica. The performance model can be constructed as normal.

## 6 Conclusion

We present Seraph, the first tuner that finds the hardware configuration for training RL models with both high performance and low cost. We demonstrate that configuration tuning can be made feasible in RL training by modeling the relationship between hardware and performance using stochastic performance models. Moreover, solving for the optimal parameters of such a model is tractable due to its convexity. Compared to existing search-based tuners, our model-based tuner is more accurate and can find the configuration more quickly.

## Acknowledgments

We sincerely thank the APSys reviewers for the insightful comments. This work was supported in part by the National Key Research & Development Program of China (No. 2022YFB4500700), the National Natural Science Foundation of China (No. 62202291), the Fundamental Research Funds for the Central Universities, and the HighTech Support Program from STCSM (No. 22511106200).

## References


[1] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *J. Artif. Intell. Res.* 47 (2013), 253–279. https://doi.org/10.1613/JAIR.3912

[2] Erin Catto. 2020. *Box2D 2.4.1 A 2D physics engine for games.* Retrieved May 28, 2024 from https://box2d.org/documentation/

[3] Yair Censor. 1977. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization* 4, 1 (1977), 41–59.

[4] Aditya Devarakonda, Maxim Naumov, and Michael Garland. 2017. AdaBatch: Adaptive Batch Sizes for Training Deep Neural Networks. *CoRR* abs/1712.02029 (2017). arXiv:1712.02029 http://arxiv.org/abs/1712.02029

[5] Theresa Eimer, Marius Lindauer, and Roberta Raileanu. 2023. Hyperparameters in Reinforcement Learning and How To Tune Them. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 9104–9149. https://proceedings.mlr.press/v202/eimer23a.html

[6] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 1406–1415. http://proceedings.mlr.press/v80/espeholt18a.html

[7] Andrew Ilyas, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. 2020. A Closer Look at Deep Policy Gradients. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=ryxdEkHtPS

[8] Chao Jin, Zili Zhang, Xingyu Xiang, Songyun Zou, Gang Huang, Xuanzhe Liu, and Xin Jin. 2023. Ditto: Efficient Serverless Analytics with Elastic Parallelism. In *Proceedings of the ACM SIGCOMM 2023 Conference, ACM SIGCOMM 2023, New York, NY, USA, 10-14 September 2023*, Henning Schulzrinne, Vishal Misra, Eddie Kohler, and David A. Maltz (Eds.). ACM, 406–419. https://doi.org/10.1145/3603269.3604816

[9] Mariam Kiran and Buse Melis Özyildirim. 2022. Hyperparameter Tuning for Deep Reinforcement Learning Applications. *CoRR* abs/2201.11182 (2022). arXiv:2201.11182 https://arxiv.org/abs/2201.11182

[10] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael I. Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 3059–3068. http://proceedings.mlr.press/v80/liang18b.html

[11] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. 2018. Tune: A Research Platform for Distributed Model Selection and Training. arXiv:1807.05118 [cs.LG] https://arxiv.org/abs/1807.05118

[12] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. 2018. An Empirical Model of Large-Batch Training. *CoRR* abs/1812.06162 (2018). arXiv:1812.06162 http://arxiv.org/abs/1812.06162

[13] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings, Vol. 48)*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.). JMLR.org, 1928–1937. http://proceedings.mlr.press/v48/mniha16.html

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347

[15] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nat.* 529, 7587 (2016), 484–489. https://doi.org/10.1038/NATURE16961

[16] Ryan Sullivan, Jordan K. Terry, Benjamin Black, and John P. Dickerson. 2022. Cliff Diving: Exploring Reward Surfaces in Reinforcement Learning Environments. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.). PMLR, 20744–20776. https://proceedings.mlr.press/v162/sullivan22a.html

[17] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement learning - an introduction.* MIT Press. https://www.worldcat.org/oclc/37293240

[18] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*. IEEE, 5026–5033. https://doi.org/10.1109/IROS.2012.6386109

[19] Hao Wang, Di Niu, and Baochun Li. 2019. Distributed Machine Learning with a Serverless Architecture. In *2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29 - May 2, 2019*. IEEE, 1288–1296. https://doi.org/10.1109/INFOCOM.2019.8737391

[20] Hanfei Yu, Jian Li, Yang Hua, Xu Yuan, and Hao Wang. 2024. Cheaper and Faster: Distributed Deep Reinforcement Learning with Serverless Computing. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (Eds.). AAAI Press, 16539–16547. https://doi.org/10.1609/AAAI.V38I15.29592

[21] Zili Zhang, Chao Jin, and Xin Jin. 2024. Jolteon: Unleashing the Promise of Serverless for Serverless Workflows. In *21st USENIX Symposium on Networked Systems Design and Implementation, NSDI 2024, Santa Clara, CA, April 15-17, 2024*, Laurent Vanbever and Irene Zhang (Eds.). USENIX Association, 167–183. https://www.usenix.org/conference/nsdi24/presentation/zhang-zili-jolteon