BNL-224923-2023-COPA

EXARL-PARS: Parallel Augmented Random Search Using Reinforcement Learning at Scale for Applications in Power Systems

H. Sharma, T. Flynn

Submitted to the e-Energy '23: The 14th ACM International Conference on Future Energy Systems Conference
to be held at Orlando FL USA
June 20 - 23, 2023

Computational Science Initiative

Brookhaven National Laboratory

U.S. Department of Energy

USDOE Office of Science (SC), Advanced Scientific Computing Research (SC-21)

DISCLAIMER

EXARL-PARS: Parallel Augmented Random Search Using Reinforcement Learning at Scale for Applications in Power Systems

Himanshu Sharma*
Pacific Northwest National Laboratory
Richland, Washington, USA
himanshu.sharma@pnnl.gov

Joshua Suetterlein
Pacific Northwest National Laboratory
Richland, Washington, USA
joshua.suetterlein@pnnl.gov

Sumathi Lakshmiranganatha
Los Alamos National Laboratory
New Mexico, USA
sumathil@lanl.gov

Thomas Flynn
Brookhaven National Laboratory
Upton, New York, USA
tflynn@bnl.gov

Draguna Vrabie
Pacific Northwest National Laboratory
Richland, Washington, USA
draguna.vrabie@pnnl.gov

Christine Sweeney
Los Alamos National Laboratory
New Mexico, USA
cahrens@lanl.gov

Vinay Ramakrishniah
Los Alamos National Laboratory
New Mexico, USA

ABSTRACT

With recent advances in deep learning and large-scale computing, learning-based controls have become increasingly attractive for complex physical systems. Consequently, developing generalized learning-based control software that takes into account the next generation of computing architectures is paramount. Specifically, for the case of complex control, we present the Easily eXtendable Architecture for Reinforcement Learning (EXARL), which aims to support various scientific applications seeking to leverage reinforcement learning (RL) on exascale computing architectures. We demonstrate the efficacy and performance of the EXARL library for the scientific use case of designing a complex control policy to stabilize a power system after experiencing a fault. We use a parallel augmented random search method developed within EXARL and present its preliminary validation and performance stabilization of a fault for the IEEE 39-bus system.

KEYWORDS

Reinforcement learning, augmented random search, exascale computing, power systems

ACM Reference Format:
Himanshu Sharma, Joshua Suetterlein, Sumathi Lakshmiranganatha, Thomas Flynn, Draguna Vrabie, Christine Sweeney, and Vinay Ramakrishniah. 2018. EXARL-PARS: Parallel Augmented Random Search Using Reinforcement Learning at Scale for Applications in Power Systems. In *Proceedings of ACM International Conference on Future Energy Systems (AMACM e-Energy '23).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

AMACM e-Energy '23, June 21–23,2023, Orlando, FL

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The ongoing developments of the next-generation Exascale Computing Project (ECP) in the United States [17] aim to provide breakthrough capabilities in modern computing hardware for modeling and simulation, data-driven scientific discovery, and addressing challenging problems of national and global interest. Concurrently, machine learning (ML) for science and engineering applications continues to advance [24, 4], prompting the need for scientific ML software capable of leveraging the next generation of computing architectures. This work presents one of these efforts, the Easily eXtendable Architecture for Reinforcement Learning (EXARL) [23], used to develop complex controls for various ECP applications. Reinforcement learning (RL) is an experience-driven learning approach [25]. With the rise of deep learning and the growing computing power, RL has received increased attention from demonstrations of its success in addressing various complex challenges in applications such as autonomous driving [10], robotics [11], energy systems [20], etc.

For many existing applications of RL, the simulations used to collect experiences for RL algorithms (i.e., the environments) are relatively simple (i.e., not time- or compute-intensive). However, this is different for complex scientific and engineering workflows. For example, many modeling and simulation applications leverage parallelization libraries such as OpenMP or MPI running on heterogeneous architectures (e.g., GPU, FPGAs, etc.). Existing RL packages such as RLib [14], Stable-Baseline3 [22], etc. focus on providing various learning algorithms and a basic RL training loop. EXARL differs by allowing scientific workflows to use the full compute

capabilities of high-performance computing systems, including exascale machines such as Frontier, Aurora, and El Capitan. To this end, EXARL implements a parallel RL loop using MPI communicators with the ability to control the number of parallel simulations and how often they generate new experiences with respect to a given learning algorithm. Our strategy allows for the parallelization of training, experience collection, and simulation. Our use of MPI as a parallel programming model, which is widely used and supported by software on HPC platforms, distinguishes us from other solutions such as RLib. We motivate our approach using parallel augmented random search (PARS) [16, 9] within EXARL and using an ECP power grid simulation as an environment.

As part of a nation's critical infrastructure, power systems (i.e., the power grid) must be resilient and reliable to promote a prosperous economy and guarantee national security. While many power grid operators have developed and adopted preventive control measures to ensure adequate security margins against any conceived contingencies (e.g., N-1), these measures are not always effective, with many blackouts occurring in the US, Europe, India, and Brazil over the last two decades [15, 5, 2]. In such situations, emergency control is imperative for real-time operations to avoid the catastrophic damage of widespread blackouts. In such a situation, conventional response control actions include generation re-dispatching, controlled system separation, and dynamic braking [12]. Although many of these responses are automatically triggered, many are still left to the system operator. These decisions are predefined based on offline studies for some forecasts of emergency scenarios. The system operator chooses suitable control actions using a look-up approach. The process is time-consuming and often overwhelming for operators. For example, during the 2011 Southwest blackout in the US, system operators failed to understand the causes and struggled to correct the situation.

The proposed mitigation strategies are classified into three categories: security-constrained alternating current optimal power flow (SC-ACOPF) [18], optimal control [13], and data-driven control (ML/ decision tree/RL) [8, 7]. While the above strategies effectively address the appropriate emergency response, developing their implementations, such as SC-ACOPF or an optimal controller, are resource and computation intensive. However, advances in data-driven approaches provide opportunities to address sequential decision-making problems in real time. Specifically, RL-based methods have the potential to overcome the outstanding issues of speed, adaptiveness, and scalability required to handle large complex power grid systems [26]. In this work, we present our implementation of the PARS algorithm in EXARL and discuss the performance of the resulting control policy used to address the contingency of an IEEE 39-bus system.

The rest of the paper is organized as follows. Section 2 describes the EXARL workflow and the PARS algorithm. Next, Section 3 presents our results. Finally, we conclude with future directions in Section 4.

2 EXARL: REINFORCEMENT LEARNING AT EXASCALE

A typical RL workflow comprises a learner, agent, and environment. An agent explores an environment by taking actions according to an action policy. During training, the agent's action policy is updated by a learner to maximize the reward based on previously observed results. The agent, once trained, can perform actions independently using the learned policy. Comprehensive details on RL theory and its methods can be found in [25].

Training involves many interactions between the agent and the environment to learn an accurate action policy; these interactions are known as episodes. For scientific applications, environments are mostly simulation- or surrogate-based. Each episode involves running the scientific simulation for multiple time steps until the simulation is complete. At every step of the simulation, the agent collects a reward for each action taken to calculate a cumulative reward for the episode. While these episodic interactions can be expensive depending on the simulation, they can be parallelized due to their independence from one another.

The EXARL framework uses the available computing resources to better parallelize the agent's learning process. Fig. 1 shows EXARL's view of the RL process across heterogeneous resources. Agents are represented by one or more coordinated actors. A parallel actor interacts with an environment and performs actions for a fixed number of steps/episodes. The actor collects the resulting observations and rewards and exchanges them with the learner. The learner is then responsible for updating the action policy based on the observations and rewards. The action policy is typically approximated by neural networks, and the updated parameters are then shared with the actors. Fig. 1 shows an EXARL deployment across multiple nodes. In this example, actors are assigned to different (heterogenous) nodes with multiple MPI processes running across a node's compute resources. One or more learner processes are deployed similarly. An EXARL workflow describes the coordination of the RL loop with its parallel actors and learners. The EXARL library provides three workflows: *sync*, *async*, and *rma* implementing sequential and distributed (two-sided and one-sided communication) RL policy training, respectively. Additional details of the EXARL API are available in the EXARL documentation at *https://exarl.readthedocs.io/en/master/*.

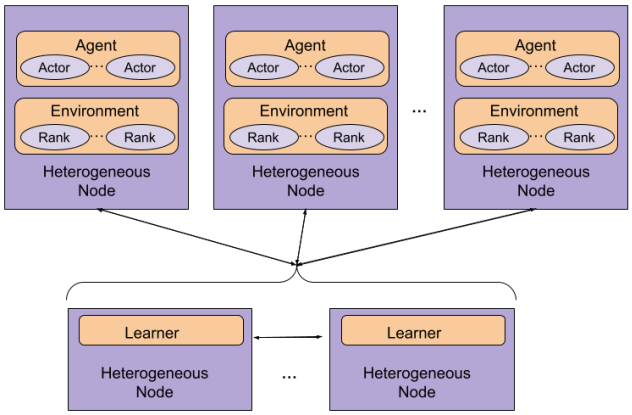

Figure 1: An overview of parallel workflow methodology developed in EXARL.

2.1 PARS:Parallel Augmented Random Search

The augmented random search (ARS) [16] is a derivative-free approach to identifying a control policy for a given environment. This

RL approach uses reward r to optimize policy $\pi(s|\theta)$, where s are the states of the environment, θ is the policy parameter, and γ is the discount factor. To find the optimal policy parameters, we use the expected total discounted reward, $\mathrm{E}\{\sum_{t=0}^{T} \gamma^t r_t(s_t, \pi(s|\theta))\}$.

Most model-free RL algorithms rely on randomly sampling the action space. On the contrary, ARS permits a more structured and efficient exploration of an environment by performing a parameter-space exploration instead. In [21], a parameter-space exploration is shown to return a better policy than an action perturbation. Furthermore, ARS does not require backpropagation to compute the gradients of the policy. Instead, a stochastic perturbation estimates the gradients. In problems with non-smooth reward functions, like power system contingency control, using the stochastic approach avoids the problem of exploding gradients.

Most of the steps in ARS are easily parallelized, and the resulting communication overhead is relatively low compared to other RL methods based on the gradient of policy and the value function. As a result, a parallel ARS can significantly reduce the learning time for computationally expensive environments (i.e., scientific applications). We implemented the parallel ARS (PARS) algorithm [16] within the EXARL framework to perform a large (configurable) number of dynamic power grid simulations (called "environment rollouts"). Each simulation performs a different perturbation at each iteration of the training. The details of PARS implementation in EXARL are presented via pseudocode in Appendix-C as Algo-1.

3 EXPERIMENTS AND RESULTS

This section discusses experiments and results using the EXARL-PARS implementation. We detail the environments that validate EXARL-PARS and our power system contingency scenario.

3.1 EXARL-PARS Validation

We validated our PARS implementation using two common reinforcement learning environments from OpenAI gym: pendulum and lunar lander [3].

Pendulum: The objective is to learn a policy to keep the pendulum upright. The pendulum is represented by a single link fixed at one end and an actuator. Torque is applied at the joint. The pendulum uses the x-y cartesian coordinate system, θ angle of the pendulum in radians and τ as torque in N-m (positive in counter-clockwise). *States*; The x, y, and θ are used as an observation for the agent. *Action*; The agent needs to learn the optimal torque τ to keep the pendulum upright. The action space is a continuous range from $[-2, 2]$. *Reward*: The reward function for the RL agent is defined as $r = -(\theta^2 + 0.1 * \dot{(\theta)}^2 + 0.001 * \tau^2)$. The duration of the episode for the environment is set at 200 steps. For each episode, the pendulum starts from a random position.

Lunar Lander: The lunar lander is a classic 2D optimization problem of the rocket trajectory. The objective of the simulation environment is to land the craft without crashing by firing its engines. The landing coordinates are $[0, 0]$. *States*: The observation space contains eight variables. The x-coordinate of the lander, horizontal (v_x) and vertical velocity (v_y), orientation in space (θ), angular velocity (v_θ), and right-left touching indicator (boolean). Coordinate observations are measured relative to the landing location. *Action*; The environment uses a continuous action space. Two actions are available: a throttle for the main engine and a throttle for the lateral booster. The throttles are turned on or off based on the main and lateral booster values. Additional details are provided in the gym environment [3] documentation page. *Reward*; The agent is awarded 100-140 points if the lander moves from the top of the screen to the landing pad and rests. Moving away from the landing position is rewarded negatively, and if the lander crashes, 100 points are deducted. If the lander comes to rest without crashing, 100 points are awarded. Ten points are awarded for each leg touching the ground. 0.03 is deducted from the total reward for each frame in which the main engine is fired. The episode ends if the lander crashes, leaves the viewport (that is, x coordinate greater than 1), or is not operating according to environmental conditions.

Results: We compare the performance of the PARS EXARL *sync* workflow against the Stable-Baseline3 [22] ARS implementation for both environments described above. We use the same training hyperparameters for both implementations. We evaluate the trained policies by comparing their mean reward and standard deviation for ten trials per environment. Table 1 presents the results. We see that the mean rewards in both cases are very similar. To demonstrate the EXARL's trained policy during the test evaluation, we show the landing sequence of the lunar lander using the EXARL-PARS trained policy in Fig. 2.

	Policy Evaluation: Mean Reward	
	Pendulum-V0	**LunarLanderContinuous-V2**
EXARL-PARS	-912.87 +/- 334	202.65 +/-77.95
StableBaseline3-ARS	-912.106 +/- 552.62	243.31 +/- 71.5

Table 1: Comparison of EXARL-PARS and StableBaseline3-ARS trained policy evaluation for different environments.

3.2 Power Systems Contingency Scenario

Environment: Power grid systems are a complex network of loads, generators, storage sources, and transmission devices. We use the GridPack [19] power grid simulation platform for our RL environment. The GridPack framework provides parallel dynamic power grid simulations capable of running on HPC platforms with high levels of performance and scalability [1]. Although the library supports several different applications related to the power grid, we are interested in contingency control applications for emergency response. The GridPack library is implemented in C++ using MPI. Python bindings are used to integrate with EXARL and call the GridPack simulation an environment for the RL policy training. We use the IEEE 39-bus test system (additional details can be found in [8]) to learn a closed-loop control policy to perform load shedding on buses 4, 7 and 18 to avoid fault-induced delayed voltage recovery and meet the voltage recovery requirements as shown in Fig. 3(b) (black dashed line). For each episode, the environment begins without a fault for 1 second of simulated time before a short-circuit fault is applied. Each episode experiences a different fault duration 0, 0.05, or 0.08 seconds before the fault is self-cleared. *State*: For contingency control, the states observed from the environment include the voltage magnitudes and remaining fractions of loads served at buses 4, 7, 8, and 18.

Actions: Control actions are to shed the load on buses 4, 7, and 18 at each time step. The amount of shedding allowed is a continuous range of [-0.2, 0], where 0 means that no voltage is shed and -0.2 is 20% shedding of the initial load. *Reward*: The reward is designed

to prompt an agent to meet the defined transient voltage recovery criteria. Once the fault is cleared, the standard requires that voltages return to at least 0.8, 0.9, and 0.95 p.u. within 0.33, 0.5, and 1.5 seconds, respectively. To achieve this, the reward r_t at time t is defined as;

$$r_t = \begin{cases} -M, \text{ if } V_i(t) < 0.95, \quad t > T_{pf} + 4 \\ c_1 \sum_i \Delta V_i(t) - c_2 \sum_j \Delta P_j(t) - c_3 u_{ivld}, \text{ otherwise} \end{cases}$$

$$\Delta V_i(t) = \begin{cases} \min\{V_i(t) - 0.7, 0\}, \text{ if } T_{pf} < t < T_{pf} + 0.33 \\ \min\{V_i(t) - 0.8, 0\}, \text{ if } T_{pf} + 0.33 < t < T_{pf} + 0.5 \\ \min\{V_i(t) - 0.9, 0\}, \text{ if } T_{pf} + 0.5 < t < T_{pf} + 1.5 \\ \min\{V_i(t) - 0.95, 0\}, \text{ if } T_{pf} + 1.5 < t \end{cases} \tag{1}$$

T_{pf} is the time for a fault to clear. The reward has three parts: a) the total bus voltage deviation below the standard voltage threshold, where $V_i(t)$ is the bus voltage magnitude; b) the load shedding amount $\Delta P_j(t)$ in p.u. at time t for load bus j; and c) the invalid action penalty u_{ivld} for RL agent for shedding load at the bus which already has been shed to zero or when the system is within a normal operation. There are three weight factors: c_1, c_2, and c_3. The agent is negatively rewarded ($-M$) if the bus voltage is below 0.95 p.u. 4 seconds after the fault is cleared. For the IEEE 39-bus system case study, we set this to -1000.

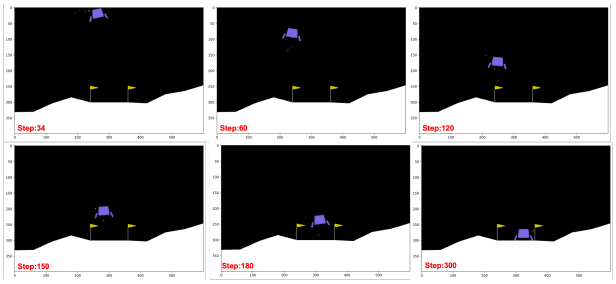

Figure 2: Sequence of lunar lander trajectory for landing using the control policy learning via EXARL-PARS.

Results: Figure 3(a) shows the training reward curve of EXARL-PARS for the IEEE 39-bus system for two different policy network architectures. In [9], authors showed that an LSTM architecture policy is more efficient and robust than a linear and fully connected network policy. Therefore, we directly compare the two different LSTM model architecture each with 2 layers of 32 and 64 neurons, respectively. While both the model architectures show similar learning performance, we observed that the 2 layers of 64 neurons perform better during evaluation. To test the policy, we ran multiple fault scenarios with different fault locations and durations. In Fig. 3(b), we show the result of the trained EXARL-PARS on one such fault case with the fault duration of 0.1 located at Bus-15. The total reward for the case concluding the evaluation was -173.604. We show the voltage recovery (blue curve) of bus 4 after the fault. The EXARL-PARS trained policy quickly restores the system back to normal operation after the fault. Next, we analyze the parallel performance of the EXARL-PARS implementation for the IEEE 39-bus system. We keep the number of episodes, directions, and other hyperparameters constant while increasing the number of actors (i.e., cores) to train the PARS agent policy. Figure 4 shows the parallel scaling performance. We see that our implementation demonstrates excellent performance and scalability. In an earlier work [9] with a Ray library-based implementation of PARS, to achieve the speed-up of 12x, more than 200 cores have been used for the IEEE 39-bus system. However, with the EXARL-PARS implementation, we achieve a similar speed-up with nearly eight cores.

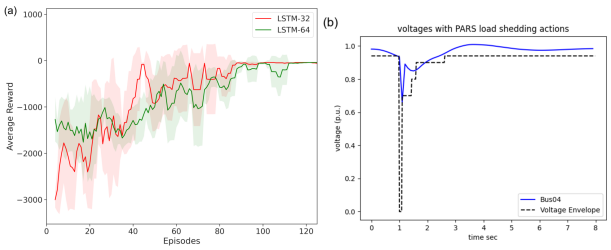

Figure 3: (a) Comparison of training reward of different LSTM networks with two layers and different numbers of hidden layers (32 and 64). (b) Testing the trained LSTM-64 model recovering the fault. Voltage of bus 4 is shown to track (blue) the performance recovery voltage envelope [6] (dashed black).

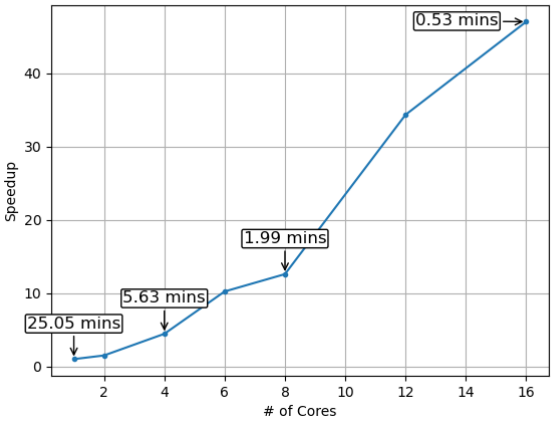

Figure 4: Speed-up ratio of EXARL-PARS for the power systems IEEE 39-Bus system contingency analysis with different number of cores.

4 CONCLUSION AND FUTURE WORK

The paper presents an implementation of parallel augmented random search in EXARL. EXARL's *sync*, *async* and *rma* workflows provide a simple way to scale a (scientific) environment from a single to multiple compute resources. Resource-based workflow customization capabilities for reinforcement learning are essential to utilize compute resources better. This work showcases the EXARL-PARS agent learning controls for power-system contingency. Next, we intend to explore the performance trade-off in parallelization strategies across the various levels (i.e., the simulation, agent, and learner) and additional science applications on ECP test beds and analyze their performance.

ACKNOWLEDGMENT

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

REFERENCES

[1] Shrirang Abhyankar, Renke Huang, Shuangshuang Jin, Bruce Palmer, William Perkins, and Yousu Chen. 2021. Implicit-integration dynamics simulation with the gridpack framework. In *2021 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE, 1–5.

[2] Zeng Bo, Ouyang Shaojie, Zhang Jianhua, Shi Hui, Wu Geng, and Zeng Ming. 2015. An analysis of previous blackouts in the world: lessons for china's power industry. *Renewable and Sustainable Energy Reviews*, 42, 1151–1163.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.

[4] Steven L Brunton and J Nathan Kutz. 2022. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press.

[5] Gerald W Cauley, David N Cook, Holly A Hawkins, and Critical Infrastructure Protection. 2011. North american electric) reliability corporation).

[6] Dept. Valley Forge. 2009. Exelon transmission planning criteria, pjm transm. plan. (2009).

[7] Istemihan Genc, Ruisheng Diao, Vijay Vittal, Sharma Kolluri, and Sujit Mandal. 2010. Decision tree-based preventive and corrective control applications for dynamic security enhancement in power systems. *IEEE Transactions on Power Systems*, 25, 3, 1611–1619.

[8] Qiuhua Huang, Renke Huang, Weituo Hao, Jie Tan, Rui Fan, and Zhenyu Huang. 2019. Adaptive power system emergency control using deep reinforcement learning. *IEEE Transactions on Smart Grid*, 11, 2, 1171–1182.

[9] Renke Huang, Yujiao Chen, Tianzhixi Yin, Xinya Li, Ang Li, Jie Tan, Wenhao Yu, Yuan Liu, and Qiuhua Huang. 2022. Accelerated derivative-free deep reinforcement learning for large-scale grid emergency voltage control. *IEEE Transactions on Power Systems*, 37, 1, 14–25. DOI: 10.1109/TPWRS.2021.3095179.

[10] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. 2021. Deep reinforcement learning for autonomous driving: a survey. *IEEE Transactions on Intelligent Transportation Systems*, 23, 6, 4909–4926.

[11] Jens Kober, J Andrew Bagnell, and Jan Peters. 2013. Reinforcement learning in robotics: a survey. *The International Journal of Robotics Research*, 32, 11, 1238–1274.

[12] P Kundur, GK Morison, and L Wang. 2000. Techniques for on-line transient stability assessment and control. In *2000 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No. 00CH37077)*. Vol. 1. IEEE, 46–51.

[13] Zhihao Li, Guoqiang Yao, Guangchao Geng, and Quanyuan Jiang. 2016. An efficient optimal control method for open-loop transient stability emergency control. *IEEE Transactions on Power Systems*, 32, 4, 2704–2713.

[14] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. Rllib: abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*. PMLR, 3053–3062.

[15] Yuri V Makarov, Viktor I Reshetov, A Stroev, and I Voropai. 2005. Blackout prevention in the united states, europe, and russia. *Proceedings of the IEEE*, 93, 11, 1942–1955.

[16] Horia Mania, Aurelia Guy, and Benjamin Recht. 2018. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*.

[17] Paul Messina. 2017. The exascale computing project. *Computing in Science & Engineering*, 19, 3, 63–67. DOI: 10.1109/MCSE.2017.57.

[18] Sidhant Misra, Line Roald, Marc Vuffray, and Michael Chertkov. 2017. Fast and robust determination of power system emergency control actions. *arXiv preprint arXiv:1707.07105*.

[19] Bruce Palmer et al. 2016. Gridpacktm: a framework for developing power grid simulations on high-performance computing platforms. *The International Journal of High Performance Computing Applications*, 30, 2, 223–240.

[20] ATD Perera and Parameswaran Kamalaruban. 2021. Applications of reinforcement learning in energy systems. *Renewable and Sustainable Energy Reviews*, 137, 110618.

[21] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. 2017. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*.

[22] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-baselines3: reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22, 268, 1–8. http://jmlr.org/papers/v22/20-1364.html.

[23] Vinay Ramakrishnaiah, Malachi Schram, Jamal Mohd-Yusof, Sayan Ghosh, Yunzhi Huang, Ai Kagawa, Christine Sweeney, and Shinjae Yoo. 2020. Easily extendable architecture for reinforcement learning (exarl). https://github.com/exalearn/ExaRL. (2020).

[24] Rick L. Stevens, Valerie E. Taylor, Jeffrey A. Nichols, Arthur B. Maccabe, Katherine A. Yelick, and David Brown. 2020. Ai for science. In.

[25] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[26] Zidong Zhang, Dongxia Zhang, and Robert C Qiu. 2019. Deep reinforcement learning for power system applications: an overview. *CSEE Journal of Power and Energy Systems*, 6, 1, 213–225.

A INTEGRATING GRIDPACK WITH EXARL AND POLICY MODEL DETAILS

The software stack required to implement and run the PARS agent includes GlobalArrays (GA) 5.8.1, the Portable, Extensible Toolkit for Scientific Computation (PETSc) 3.7.6, GridPack 3.4, the GridPack Python wrapper and the power grid environment (powerGridEnv) based on on GridPACK, compatible with OpenAI Gym and thus compatible with EXARL. GridPack is a CPU-only code, however EXARL uses TensorFlow and thus runs on GPU nodes more efficiently than CPU nodes, so we ran our tests on GPU nodes. We did several tests of the GridPack installation via the Power Flow example provided with GridPack. The build and test instructions are made available on our EXARL wiki. All the results generated using the PNNL high performance research computing facility with 2x P100 GPUs node with Dual Intel Broadwell E5-2620 v4 @ 2.10GHz CPUs (16 cores per node) 64 GB 2133Mhz DDR4 memory per node.

B HYPERPARAMETERS FOR RL POLICY TRAINING

Note that for validation of the EXARL-PARS implementation against Stable-Baseline3 implementation we make keep the hyperparameter same. Table 2 presents all the hyperparmeters used for each environment used in the present study with EXARL-PARS.

	Pendulum	**LunarLanderContinuous**	**PowerGrid IEEE-39**
Policy Network Size	Fully Connected [16]	Fully Connected [16]	LSTM [32,32] or [64.64]
Number of Directions	4	4	4
Top Direction	1	1	1
Step Size	0.018	0.018	0.018
Std. Dev	0.1	0.1	0.5
Decay Rate	1	1	1

Table 2: Hyperparameters for EXARL-PARS and Stable-Baseline3

C PSEUDO-CODE FOR PARALLEL AUGMENTED RANDOM SEARCH

The *CallActor* function is defined in EXARL to execute the environment steps until the environment terminates. The function returns the total reward of the environment execution which is the summation of the reward at each step. The algorithm is parallelized by calling the *async* workflow, which uses MPI to exchange the parameters among different actors and learners.

Algorithm 1 Pseudo-Code of Parallel Augmented Random Search Implementation in EXARL

Require: number of perturbation(N), number of top direction (b),step size (η), Initialized policy network (θ), std. deviation (σ), decay rate (ϵ), Number of episodes(N_e)

while $epi \leq N_e$ **do** ▷ Sync and Async Parallel.
  $N \sim \mathcal{N}(0, \sigma^2)$ ▷ Sample N random direc.
  $\Theta \leftarrow [\theta_{epi} + N(i)\ ,\theta_{epi} - N(i)]\ \forall$ i $\in N$ ▷ Concat. ± perturb
  for k = 1 to len(Θ) **do**
    Randomly initialize the state of the environment.
    $[r^+_{epi}, r^-_{epi}] \leftarrow CallActor(\theta_{epi}$(k)) ▷ Reward col. ±
  end for
  Sort max$[r^+_{epi}, r^-_{epi}]$ and select top b direction calculate standard deviation of reward σ^b_r
  Update Policy:

$$\theta_{epi+1} = \theta_{epi} + \frac{\eta}{b * \sigma^b_r} \sum_{i=1}^{b} \left(r^+_{epi} - r^-_{epi}\right) N(i)$$

  Step size η and standard deviation σ decay with rate ϵ: $\eta = \epsilon\eta$, $\sigma = \epsilon\sigma$
  epi += 1 ▷ Increase counter of episode
end while