

Energy Consumption Analysis of Deep Reinforcement Learning[#]

Zhenghao Yang¹, Xu Wan², Mingyang Sun^{1*}

1 School of Advanced Manufacturing and Robotics, Peking University, Beijing, 100871, China

2 School of Control Science and Engineering, Zhejiang University, Hangzhou, 310058, China

(Corresponding Author: smy@pku.edu.cn)

ABSTRACT

The widespread success of deep reinforcement learning (DRL) is increasingly challenged by its substantial energy footprint, posing a significant obstacle to green artificial intelligence (AI) development. While some research has primarily focused on the energy efficiency of deep neural networks (DNNs), the unique energy consumption characteristics stemming from DRL's interactive learning paradigm have been largely overlooked. This paper bridges this critical gap by introducing *EnergyDRL*, a fine-grained energy analysis framework that enables a precise attribution of energy consumption across the entire DRL workflow by decomposing the training loop into functionally distinct stages. Energy monitoring modules based on *EnergyDRL* are deployed across comprehensive experiments on benchmark tasks, yielding a crucial but frequently overlooked insight: agent-environment interaction, rather than neural network updates, constitutes the primary energy bottleneck. Quantitative investigations further reveal that hyperparameters, model size, and the balance between sampling and updating distinctly influence the energy consumption profile, which provides the empirical foundation for proposed strategies to develop energy-efficient DRL algorithms.

Keywords: deep reinforcement learning, energy consumption analysis, energy efficiency

NONMENCLATURE

Abbreviations

DRL	Deep Reinforcement Learning
AI	Artificial Intelligence
DNN	Deep Neural Network
RL	Reinforcement Learning
DQN	Deep Q-Network
PPO	Proximal Policy Optimization
SAC	Soft Actor-Critic

1. INTRODUCTION

With the rapid development of artificial intelligence (AI), especially the emergence of frontier models such as large language models, carbon emissions from AI training and operation have shown an exponential growth trend, driven by the surging demand for computing power and electricity use [1, 2, 3]. However, this trend stands in stark contrast to the global pursuit of sustainable development, which is reinforced by international initiatives and stringent national policies. This establishes a fundamental tension between the quest for greater computational power and the urgent need for energy conservation. Consequently, enhancing the energy efficiency of AI systems has become a critical imperative for balancing technological progress with environmental responsibility.

The pursuit of environmentally friendly and energy-efficient AI has led to research proposing a series of "green AI" concepts with different connotations [4, 5]. Schwartz et al. [6] point out that the dominant development paradigm in current AI research is a "scale-for-performance" model, leading to massive consumption of computing resources and significant environmental costs. In contrast, green AI advocates for treating computational efficiency as an equally important metric alongside performance. It achieves AI's environmentally friendly and inclusive development by focusing on full-cycle efficiency, innovating evaluation systems, and promoting algorithmic optimization.

Analyses of AI energy consumption often adopt a full lifecycle perspective, as significant energy demands arise not only during model usage but also throughout development and deployment phases [7]. Many studies focus on the training and inference stages of models, proposing numerous optimization methods for specific DNN architectures such as Convolutional Neural Network (CNN) and Transformer [8, 9]. Inspired by

[#] This is a paper for the 17th International Conference on Applied Energy (ICAE2025), December 8-12, 2025, Bangkok, Thailand.

biological neurons, Tavanaei et al. [10] proposed a novel architecture—the Spiking Neural Network (SNN), regarded as the third generation of neural networks—which transmits information via discrete spike signals and achieves higher energy efficiency compared to conventional artificial neural networks.

Regarding the choice of metrics for measuring energy efficiency, electricity consumption is the most direct and accurate indicator of environmental cost. While other metrics such as the number of parameters and Floating Point Operations (FLOPs) have a positive correlation with energy consumption, they are influenced by various factors such as model architecture and tasks [11]. As a result, they do not accurately reflect the inherent characteristics of the algorithm itself.

In the field of AI, DRL exhibits particularly high demands for data and computational resources. However, the energy consumption of DRL remains underexplored. Unlike conventional machine learning methods, DRL's learning paradigm relies on continuous interaction and feedback, where the dynamic energy costs of data sampling and model operations are poorly understood. Therefore, optimizing the energy efficiency of DRL algorithms represents a critical avenue for reconciling the growing demand for computational power with the goals of low-carbon development. Current research lacks systematic characterization and analysis of energy consumption in DRL algorithms. To this end, this study makes the following contributions: (1) it proposes *EnergyDRL*, a fine-grained framework to precisely decompose and analyze energy consumption across the entire DRL workflow; (2) it identifies agent-environment interaction, rather than neural network updates, as the primary energy bottleneck through a series of carefully designed experiments; and (3) it provides empirical evidence to inform the design of future energy-efficient algorithms by quantitatively linking key training factors to the overall energy profile.

2. MATERIAL AND METHODS

2.1 *EnergyDRL* framework

Based on a theoretical analysis of the RL framework and an operational analysis of the Deep Neural Network (DNN) implementation, *EnergyDRL* decomposes the training loop into fine-grained, distinct stages to enable highly granular energy consumption monitoring. As illustrated in Fig. 1, the framework is constructed upon the standard RL loop which comprises three modules: agent, environment, and data. It captures all core computational stages that lead to energy consumption.

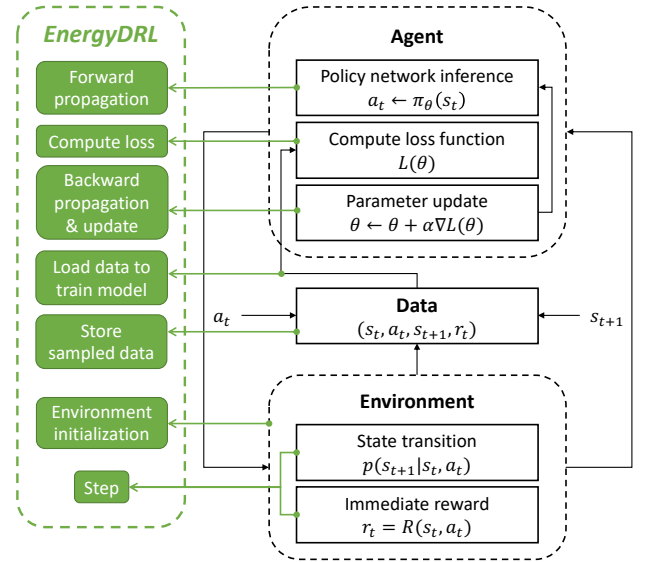


Fig. 1 Framework of *EnergyDRL*

Formally, an RL system typically consists of a closed-loop system composed of an agent, an environment, and the interaction mechanism between them. Mathematically, an RL problem can be formulated as a Markov Decision Process (MDP), which can be described by a quintuple $\langle \mathcal{S}, \mathcal{A}, p, R, \gamma \rangle$, where: \mathcal{S} represents the state space, which defines the set of all possible states in the environment; \mathcal{A} represents the action space, which includes the set of all possible actions the agent can perform; $p: \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, \infty)$ is the state transition probability function, specifically $p(s_{t+1}|s_t, a_t)$ denotes the probability density of transitioning to the next state $s_{t+1} \in \mathcal{S}$ when the agent performs action $a_t \in \mathcal{A}$ in the current state $s_t \in \mathcal{S}$; $R: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the immediate reward function, which quantifies the immediate return of the agent's behavior; $\gamma \in [0, 1]$ is the discount factor, which represents the decay coefficient for future rewards. A policy defines the agent's strategy for acting in a given state. A deterministic policy, $\pi: \mathcal{S} \rightarrow \mathcal{A}$, maps each state $s \in \mathcal{S}$ to a specific action $a \in \mathcal{A}$. In contrast, a stochastic policy, $\pi: \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, gives the probability of selecting action a in state s . The agent's goal is to find a policy π that maximizes the expected return (also known as the value function):

$$V^\pi(s) = \mathbb{E} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, \pi \right] \quad (1)$$

where $r_t = \mathbb{E}_{a \sim \pi} R(s_t, a)$. The data module also plays a crucial role in this loop, acting as the bridge that facilitates information exchange between the agent and the environment.

Leveraging their powerful representation learning capabilities, DNNs are used in DRL models to

parameterize and represent key components of RL algorithms, such as the value functions $V_{\theta}^{\pi}(s)$ and policies π_{θ} . The loss function for training DNNs is also tailored to the specific category of the algorithm. Based on their learning approach, DRL can be divided into two main categories: Value-based methods and Policy Gradient methods. Value-based methods aim to construct an evaluation function from which a policy can be derived. This evaluation function is typically a Q-value function:

$$Q^{\pi}(s, a) = \mathbb{E} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi \right] \quad (2)$$

which quantifies the expected return of taking a specific action in a given state. Policy Gradient methods directly learn a policy that maps states to actions, rather than learning a value function. They use a parameterized policy and update it to maximize the expected return. These methods often employ the Actor-Critic architecture, in which the Actor refers to the policy itself, while the Critic corresponds to an estimator for the value function or Q-value function. Both the Actor and Critic are implemented using DNN approximators.

Based on the source of data used for learning, RL algorithms can be classified into two categories: on-policy and off-policy. On-policy algorithms update themselves using experience gathered from the current policy's execution, which means that the policy being evaluated is the same as the one being updated. In contrast, off-policy algorithms use sampled data generated by one or more behavioral policies that are different from the target policy.

To investigate the differences in energy consumption across various algorithms, this study examines three classes of classic DRL algorithms, as follows:

(1) Deep Q-Network (DQN) [12]: A foundational Value-based algorithm that utilizes DNNs to approximate the optimal Q-value function. Its off-policy nature allows for efficient data reuse from a replay buffer, making it highly effective in tasks with discrete action spaces.

(2) Proximal Policy Optimization (PPO) [13]: A leading on-policy Policy Gradient algorithm that ensures stable training by clipping the policy update objective.

(3) Soft Actor-Critic (SAC) [14]: An advanced off-policy Actor-Critic algorithm that optimizes a trade-off between expected reward and policy entropy. This dual objective enhances sample efficiency through data reuse and encourages robust exploration.

The energy consumption analysis of traditional machine learning models primarily focuses on the computational characteristics of different deep neural network architectures, including operational complexity,

as well as data storage and processing methods. In addition to these aspects, *EnergyDRL* also accounts for the data generation process, including the interaction between the agent and the environment, the computational resources required by the environment itself, and the use of training data. Taking into account both the RL framework and the characteristics of DNNs, *EnergyDRL* propose a detailed division of energy monitoring stages that covers the entire training workflow, as shown in Table 1.

Table 1 DRL energy monitoring stages

Module	Stage	Explanation
Environment	make	Initialize the environment.
	reset	Reset the environment.
	step	Interact with the environment.
Data	ds	Store sampled data.
	dl	Load data to train.
Model (Agent)	fwdex	Forward propagation during exploration.
	fwdtr	Forward propagation during training.
	cpls	Compute the loss function.
	bwd	Backward propagation.
	opt	Update the parameters.
	eval	Evaluate the current model.
	save	Save the current model.

Based on *EnergyDRL*, this study deploys energy monitoring modules in the following experiments. The runtime and power consumption of specific code blocks can be monitored using the corresponding Python package, which facilitates individual measurement of different training stages. The measurements can be analyzed from three distinct perspectives. Firstly, the execution frequency of each stage can be analyzed based on the number of windows created. Secondly, the proportion of energy consumed by different stages can be determined. Finally, the measurements can reveal how energy consumption patterns change during transitions between different stages over a timeline.

2.2 Experiment settings

2.2.1 Algorithm implementation

To select the DRL test cases for this study, three key criteria were considered: algorithm type, application environment, and adherence to a unified measurement standard. Following a thorough investigation, several benchmark environments were selected, spanning domains such as games, robotics, control tasks, and

multi-agent systems. The study employed Ray RLlib [15], an open-source Python library that integrates essential DRL modules. Utilizing Ray RLlib ensures consistent implementation details across algorithms, thereby enhancing the comparability and reproducibility of the experimental results.

2.2.2 Measurement

The open-source Python package Zeus[16] was selected as our energy measurement tool. Its interface allows for querying the GPU's hardware energy consumption information and enables us to specify code blocks to be measured, facilitating the separate energy measurement of each stage in the training process.

The study recorded the start time, duration, and GPU power for all monitoring windows during the training process. Recognizing that power may fluctuate within a given window, the study assumed a linear change and averaged the instantaneous power at the start and end of each window to approximate its average power. Given that the duration of the experimental windows was on the scale of milliseconds, this approximation method does not introduce significant errors.

2.2.3 Hardware

The experiments were run on a single NVIDIA TITAN V GPU, which has 12GB of HBM2 memory, a Thermal Design Power (TDP) of 250W, and an idle power of 26W. Since the experiments directly measured the GPU's power consumption, the study ran all training tasks on a single GPU with only one process running to ensure that the measurement was not affected by other tasks.

2.2.4 RL tasks

In addition to previous introduction, the three algorithms also differ in their compatibility with action spaces: DQN is only suited for discrete environments, SAC for continuous ones, and PPO can be used in both. Three environments were chosen from the Gymnasium library: "CartPole-v1", "Pendulum-v1", and "HalfCheetah-v5". The first two are part of the relatively simple "Classical Control" suite, while the last one belongs to the more complex "MuJoCo" robotics environment. Furthermore, the first environment has a discrete action space, whereas the latter two have continuous action spaces. This study selected six "algorithm-environment" pairs: "PPO-Pendulum", "PPO-Halfcheetah", "PPO-Cartpole", "DQN-Cartpole", "SAC-Pendulum", and "SAC-Halfcheetah".

In DRL algorithm research, the number of timesteps (agent-environment interactions) is commonly used as a

metric to study performance changes during training, which allows researchers to compare the training efficiency of different algorithms. Similarly, this study set a fixed number of timesteps to compare the energy consumption characteristics of different algorithms.

However, a GPU's energy consumption can vary across different training stages. For example, the power at the beginning of training slowly rises from its idle state and stabilizes after some time. Therefore, the selection of timesteps must account for whether the energy consumption characteristics have reached a steady state. To quantify this, the study used the energy proportion of different monitoring windows as a metric. Each experiment was run with stopping timesteps of 100k, 200k, and 500k to compare whether the proportion for each window was sufficiently stable. The results showed that the data from 100k timesteps was already stable, so this dataset was used for subsequent analysis.

The hyperparameter settings are shown in Table 2. Building on Ray RLlib, the number of model parameters was aligned to a similar scale. In Table 2, BS, MBS, and UI are abbreviations for *batch_size*, *minibatch_size*, and *update_interval*, respectively. These hyperparameters define how each algorithm collects data and updates its model. For the on-policy PPO, updates are performed only after a full batch of data with size *batch_size* has been collected. The number of updates is determined by $(batch_size / minibatch_size) * epochs$. For the off-policy DQN and SAC, a certain amount of data is first collected and stored in a replay buffer. The model is then updated by randomly sampling a *batch_size* of data from this buffer after every *update_interval* steps. In the "Experiment" column, the environment in each "algorithm-environment" pair is replaced by its initial.

Table 2 Hyperparameter settings

Experiment	Parameters	BS	MBS	epochs	UI
PPO-P	67587	1024	128	16	/
PPO-H	73741	1024	128	16	/
PPO-C	67843	4000	128	8	/
SAC-P	201988	256	/	/	20
SAC-H	217870	256	/	/	20
DQN-C	133635	32	/	/	5

3. RESULTS

3.1 Verification of Result Stability

A comparison of the experimental results across three different timestep settings (100k, 200k, and 500k) confirms that the energy consumption characteristics

have reached a steady state. As shown in Fig. 2, for the "PPO-Pendulum" experiment, the relative error of the energy consumption proportion for each stage is less than 5% across three conditions. Therefore, subsequent analysis uses only the 100k data to represent the algorithm's typical energy consumption characteristics.

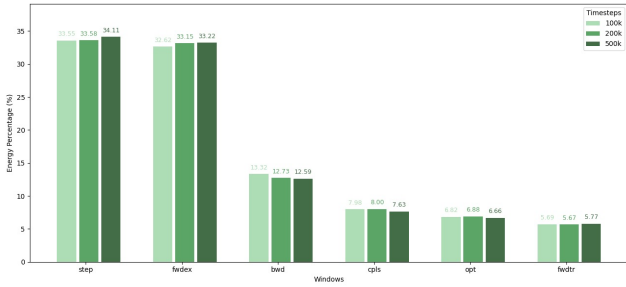


Fig. 2 Result stability in "PPO-Pendulum"

3.2 Statistical data of monitoring windows

This study compiled statistics on the count, duration, and power for all monitoring windows of each stage and visualized this data. This visualization includes both the power-duration distribution and key statistics for different algorithm stages in each experiment. Each subplot in Fig. 4 shows the "power-duration" distribution for each window, based on a random sample of 5% of the data points.

3.3 Energy consumption proportion of each stage

Fig. 3 illustrates the energy consumption breakdown for each stage across all experimental groups. Stages

associated with data sampling, rather than those involving model itself, account for the majority.

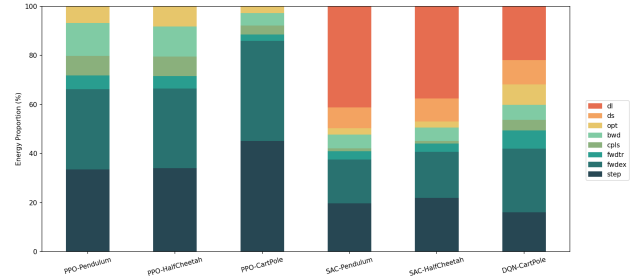


Fig. 3 Energy consumption proportion

4. ANALYSIS AND DISCUSSION

4.1 Algorithm and environment

The results indicate that a single algorithm exhibits similar energy consumption characteristics across different environments. However, for the same environment, the energy performance of different algorithms shows significant variations. Despite the differences in the "power-duration" scatter plot patterns across environments, the relative positions of the mean values remain consistent. For instance, in the PPO algorithm, the stages with the longest average duration are, in order, *bwd*, *cpls*, *opt*, and *fwdtr*. The average power of these four stages is also significantly higher than the rest. Beyond the mean value of each stage, the final total energy consumption proportions exhibit a similar consistency: for a given algorithm, the ranking of

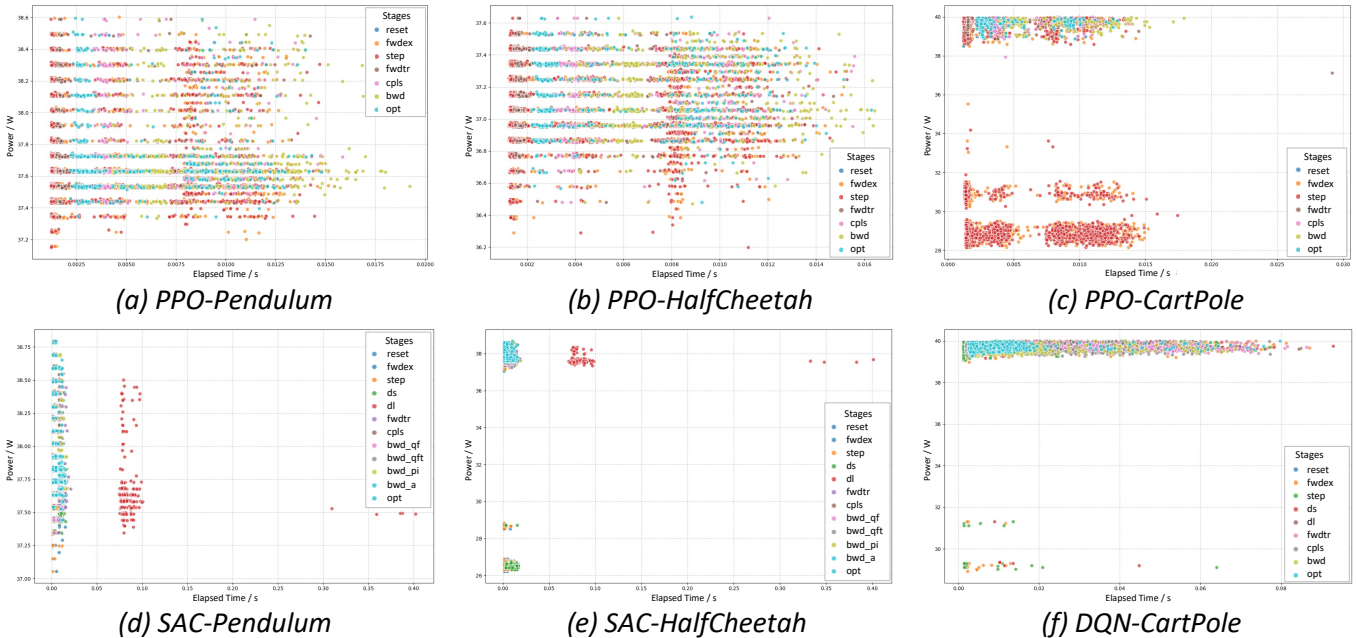


Fig. 4 Power-duration distribution

energy consumption proportion remains consistent across different environments.

On one hand, this similarity in mean values and energy consumption proportions is because a single algorithm has an identical module composition and training workflow, which does not change significantly with different environments. On the other hand, the different patterns observed in the scatter plots stem from variations in the specific model architecture and training hyperparameters, and may also be influenced by the computational hardware. This will be analyzed in a subsequent section.

A key observation is that the stages related to agent-environment interaction—specifically, *fwdex* , *step* , *ds* , and *dl* —account for over 60% of the total energy consumption. This universal finding across all experiments challenges the conventional belief that compute-intensive DNNs are the primary energy bottleneck, highlighting the unique energy consumption profile specific to DRL. Moreover, the proportion of the *dl* and *ds* stages—which are unique to off-policy algorithms—is quite high. This suggests that transferring training data between the replay buffer and the agent (or, at a hardware level, between different memory addresses) is a highly time- and energy-intensive process.

4.2 Number of parameters

The inference energy consumption of a DNN with the same architecture increases with its number of parameters. This directly impacts the *fwdex* and *fwdtr* stages and also increases the computational load—and thus the energy consumption—of the *bwd* stage. Therefore, this study aligned the number of parameters across all experimental groups, striving to use the same DNN architecture and identical hyperparameters, such as the number of hidden layers and neurons.

Comparing the energy consumption and parameter count of each experiment in Fig. 5, a clear relationship where a higher number of parameters correlates with greater energy consumption can be observed. Arguably, the number of parameters primarily influences the computation-intensive stages that are sensitive to model complexity, which are *fwdtr* , *fwdex* , and *bwd* . This influence is exerted through two mechanisms: first, an increase in the average duration of these stages, and second, a rise in the average power during them.

To identify the marginal energy cost of increasing the number of parameters, an experiment was designed. Specifically, for the model in the PPO-HalfCheetah pair, the number of neurons in the hidden layers of the shared encoder was varied, as this part constitutes the largest

portion of the model's parameters. This procedure yielded models with different parameter counts. Results show that for small models, a 12x increase in parameters results in a negligible energy rise of 5%, but for large models, a 3x parameter increase leads to a significant 30% escalation in energy consumption.

This case study suggests that while a positive correlation exists between the number of parameters and model energy consumption, the marginal energy cost of increasing parameters may be increasing and non-linear. At low parameter counts, the training energy consumption includes a fixed overhead, and a moderate increase in parameters adds little to the cost. However, once the parameter count becomes large enough, the additional cost becomes a substantial burden.

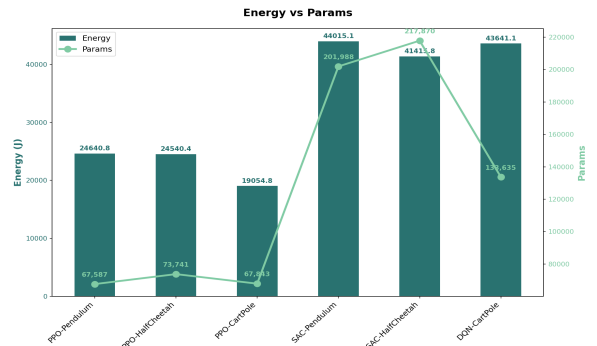


Fig. 5 Parameters-energy relationship

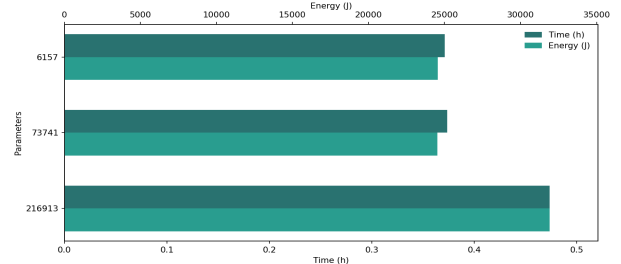
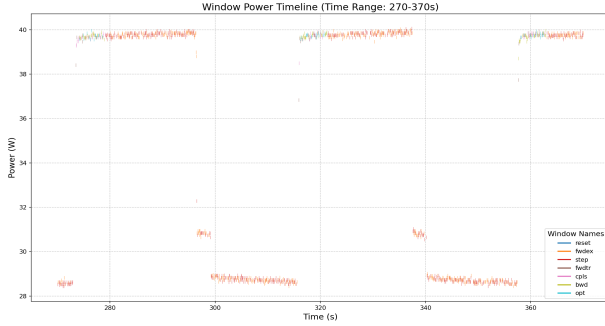


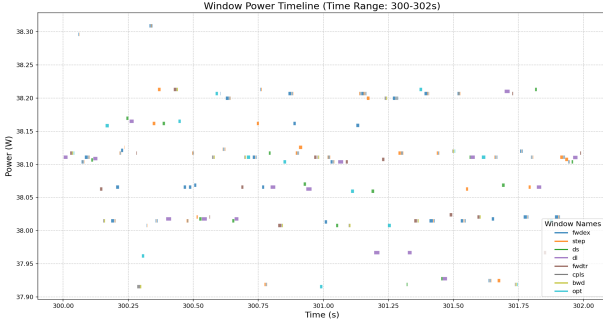
Fig. 6 Marginal energy cost of parameter scaling

4.3 Sampling-updating pattern

The algorithms evaluated in the aforementioned experiments—DQN, SAC, and PPO—employ two distinct "sampling-updating" patterns. The on-policy PPO algorithm first collects a sufficient amount of data and then uses stochastic gradient descent (SGD) to update its network parameters in a centralized manner. After this update, it begins collecting the next batch of data for the subsequent update. In this process, the boundaries between the "sampling" and "updating" phases are clearly defined, with a relatively long interval between two consecutive instances of the same phase. In contrast, the off-policy algorithms (DQN and SAC) can frequently switch between "sampling" and "updating" because they can reuse data from a replay buffer.



(a) PPO-Cartpole



(b) DQN-Cartpole

Fig. 7 Sampling-updating patterns

Experimental results indicate that these two "sampling-updating" patterns, to a certain extent, determine the GPU power consumption patterns during training. The PPO-CartPole experiment serves as a good example. As shown in Fig. 6, a large portion of the data points of *step* and *fwdex* fall into a lower power range (below 32W). The remaining data points are primarily concentrated in a higher level, clustering with data points of *fwd* and *bwd*. Within each training cycle, periods of high power consumption consistently correspond to a continuous sequence of stages: *fwdtr*, *cpls*, *bwd*, and *opt*. In contrast, the low power level is entered during a continuous sequence of the *fwdex* and *step* stages. A transition period of approximately two seconds exists between the two levels.

This change pattern is not difficult to understand. Executing model parameter updates requires extensive parallel computations on the GPU, which causes the GPU to increase its power consumption to handle the computational load. In contrast, during the "sampling" phase, computations occur on the CPU rather than the GPU. As a result, the GPU becomes idle after the "updating" phase concludes. Once this idle state persists for a certain duration, the hardware automatically lowers its power to optimize energy efficiency. This hypothesis regarding the automatic adjustment mechanism is also supported by the GPU-Util information obtained from the command-line utility tool, `nvidia-smi`. This metric indicates the percent of GPU

utilization i.e. percent of the time when kernels were using GPU over the sample period. Across all the experiments, this metric remained at a low level (1%) for most of the time, corresponding to a low power state. The remaining time, which belongs to the "updating" phase, showed high GPU-Util and power consumption.

As previously mentioned, the GPU possesses an automatic power adjustment mechanism to optimize energy consumption: it lowers its power when utilization remains at a low level for an extended period. This triggering condition is difficult to meet for off-policy algorithms, which frequently switch between the "sampling" and "updating" phases. For example, in the DQN-CartPole experiment, the agent updates its parameters after collecting every 5 data samples. The short interval between these updates means the GPU's power consumption remains high consistently, regardless of whether it is actively performing computational tasks.

4.4 Optimizing Energy Consumption in DRL

DRL computations require both CPU and GPU resources, demanding coordinated management for energy-efficient training. Currently, most environment computations are limited to the CPU [17], because GPUs—optimized for parallel tasks—cannot efficiently handle temporally dependent operations. In the agent-environment interaction, where the next state entirely depends on its previous state and action, computations must be performed serially on the CPU. Previous analysis also revealed that different "sampling-updating" patterns influence the GPU's power consumption profile. During the sampling phase, the GPU was identified as having a low utilization rate, indicating that the CPU's sampling speed is insufficient to keep up with the GPU's continuous updates. Therefore, a better distributed reinforcement learning framework and a more rational allocation of computational resources could lead to higher energy efficiency. Besides, adjustment of some hyperparameters can also affect the "sampling-updating" pattern to improve energy efficiency.

On the other hand, since environmental computations are limited to the CPU, a straightforward optimization approach is to offload these computations to the GPU. Liang et al. [18] have already proposed a GPU-accelerated robotics simulation framework to replace traditional CPU-based simulations. This framework leverages the GPU's parallel computing power by loading all robots and task-related objects into a unified simulation environment, allowing for the parallel simulation of multiple robotic agents across

various task scenarios. Because multiple agents can interact within a single simulation, a GPU-based framework is inherently better suited for multi-agent reinforcement learning simulations than the conventional approach of running multiple single-agent simulation processes, and one of the direct benefits of this more efficient use of the GPU is reduced energy consumption.

5. CONCLUSIONS

This paper proposes *EnergyDRL*, a novel analysis framework focusing on the energy consumption of DRL, which decomposes the training process into distinct stages for fine-grained energy monitoring. Based on *EnergyDRL*, the experimental part monitored and collected energy consumption data for three popular DRL algorithms through the training process. The results indicate that the energy consumption characteristics differ significantly between algorithms, primarily due to their distinct training procedures and components. Among all stages, the stage related to agent-environment interaction consumes the most energy. Furthermore, the marginal energy cost of model parameters shows an increasing trend. The "sampling-updating" pattern not only influences the number of executions for each stage but also affects the energy consumption pattern of the computing hardware. These findings provide practical directions for designing more energy-efficient DRL algorithms. The bottleneck limiting the energy efficiency of DRL algorithms exists not only at the algorithmic design level but also in the coordinated optimization of computing resources.

REFERENCE

- [1] Maslej, N., Fattorini, L., Perrault, R., Gil, Y., Parli, V., Kariuki, N., Capstick, E., Reuel, A., Brynjolfsson, E., Etchemendy, J., Ligett, K., Lyons, T., Manyika, J., Niebles, J. C., Shoham, Y., Wald, R., Walsh, T., Hamrah, A., Santarlasci, L., . . . Oak, S. (2025). *The AI index 2025 annual report*. Institute for Human-Centered AI, Stanford University. <https://aiindex.stanford.edu/report/>
- [2] Ding, Z., Wang, J., Song, Y., Zheng, X., He, G., Chen, X., Zhang, T., Lee, W.-J., & Song, J. (2025). Tracking the carbon footprint of global generative artificial intelligence. *The Innovation*, 6(5), 100866. <https://doi.org/10.1016/j.xinn.2025.100866>
- [3] International Energy Agency. (2025, April). *Energy demand from AI*. In *Energy and AI*. <https://www.iea.org/reports/energy-and-ai/energy-demand-from-ai>
- [4] Verdecchia, R., June Sallou, & Luís Cruz. (2023). A systematic review of Green AI. *WIREs Data Mining and Knowledge Discovery*, 13(4), e1507. <https://doi.org/10.1002/widm.1507>
- [5] Bolón-Canedo, V., Morán-Fernández, L., Cancela, B., & Alonso-Betanzos, A. (2024). A review of green artificial intelligence: Towards a more sustainable future. *Neurocomputing*, 599, 128096. <https://doi.org/10.1016/j.neucom.2024.128096>
- [6] Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2020). Green AI. *Communications of the ACM*, 63(12), 54–63. <https://doi.org/10.1145/3381831>
- [7] Jiang, P., Sonne, C., Li, W., You, F., & You, S. (2024). Preventing the Immense Increase in the Life-Cycle Energy and Carbon Footprints of LLM-Powered Intelligent Chatbots. *Engineering*, 40, 202–210. <https://doi.org/10.1016/j.eng.2024.04.002>
- [8] Yang, T.-J., Chen, Y.-H., Emer, J., & Sze, V. (2017). A method to estimate the energy consumption of deep neural networks. *2017 51st Asilomar Conference on Signals, Systems, and Computers*, 1916–1920. <https://doi.org/10.1109/ACSSC.2017.8335698>
- [9] Wang, Y., Qin, Y., Deng, D., Wei, J., Zhou, Y., Fan, Y., Chen, T., Sun, H., Liu, L., Wei, S., & Yin, S. (2023). An Energy-Efficient Transformer Processor Exploiting Dynamic Weak Relevances in Global Attention. *IEEE Journal of Solid-State Circuits*, 58(1), 227–242. *IEEE Journal of Solid-State Circuits*. <https://doi.org/10.1109/JSSC.2022.3213521>
- [10] Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. S. (2019). Deep Learning in Spiking Neural Networks. *Neural Networks*, 111, 47–63. <https://doi.org/10.1016/j.neunet.2018.12.002>
- [11] Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., & Pineau, J. (2022). *Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning* (No. arXiv:2002.05651). arXiv. <https://doi.org/10.48550/arXiv.2002.05651>
- [12] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533. <https://doi.org/10.1038/nature14236>
- [13] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms* (No. arXiv:1707.06347). arXiv. <https://doi.org/10.48550/arXiv.1707.06347>
- [14] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor* (No. arXiv:1801.01290). arXiv. <http://arxiv.org/abs/1801.01290>

- [15] Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., & Stoica, I. (2018). *RLlib: Abstractions for Distributed Reinforcement Learning* (No. arXiv:1712.09381). arXiv. <https://doi.org/10.48550/arXiv.1712.09381>
- [16] You, J., Chung, J.-W., & Chowdhury, M. (2023). Zeus: Understanding and optimizing GPU energy consumption of DNN training. *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 119–139. <https://www.usenix.org/conference/nsdi23/presentation/you>
- [17] Inci, A., Bolotin, E., Fu, Y., Dalal, G., Mannor, S., Nellans, D., & Marculescu, D. (2020). *The Architectural Implications of Distributed Reinforcement Learning on CPU-GPU Systems* (No. arXiv:2012.04210). arXiv. <https://doi.org/10.48550/arXiv.2012.04210>
- [18] Liang, J., Makoviychuk, V., Handa, A., Chentanez, N., Macklin, M., & Fox, D. (2018). GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning. *Proceedings of The 2nd Conference on Robot Learning*, 270–282. <https://proceedings.mlr.press/v87/liang18a.html>