2024 IEEE Real-Time Systems Symposium (RTSS) | 979-8-3315-4026-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/RTSS62706.2024.00019

# DuoJoule: Accurate On-Device Deep Reinforcement Learning for Energy and Timeliness

Soheil Shirvani, Aritra Samanta, Zexin Li, Cong Liu
University of California, Riverside

***Abstract*—Deep Reinforcement Learning (DRL) is critical for autonomous systems to continuously learn and adapt in dynamic environments. However, frequent retraining in DRL leads to high energy consumption, posing significant challenges for mobile and battery-dependent robotic systems. Co-optimizing energy, latency, and algorithm performance is essential for efficient on-device DRL. Current approaches either focus on traditional DNNs like CNNs or target only two out of the three dimensions, rather than addressing all three simultaneously. This paper introduces `DuoJoule`, a comprehensive framework designed to address the unique challenges of DRL workloads by meeting latency deadlines and adhering to energy budgets while maximizing algorithm performance through both application and system-level configurations. `DuoJoule` dynamically coordinates adjustments in DRL algorithm parameters and system frequency settings using Dynamic Voltage and Frequency Scaling (DVFS). A key innovation of `DuoJoule` is its runtime metric tracker, which assesses system status against target budgets and calculates a universal efficiency score. This enables rapid and adaptive tuning at runtime, balancing energy efficiency, latency, and algorithm performance. Extensive evaluation using benchmarks along with a realistic autonomous driving case study demonstrates `DuoJoule`'s versatile cross-platform efficiency, practicality in real-world scenarios, adaptivity to varying constraints, and low runtime overhead evaluated on two widely used autonomous embedded platforms. Empirical results show that `DuoJoule` consistently meets latency and energy targets while maintaining near-optimal performance, showcasing its effectiveness in managing the complex trade-off space of on-device DRL.**

## I. Introduction

Deep Reinforcement Learning (DRL) is becoming increasingly crucial in real-time embedded systems designed for robotic and transportation applications due to its superior capability to handle dynamic and uncertain environments through continuous learning and adaptation [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12]. Despite its promising features, implementing DRL on resource-constrained embedded devices—known as on-device DRL—remains a challenge. This challenge stems from the need for constant adaptation through frequent retraining during inference, which significantly increases computational demands and energy consumption. Additionally, it is compounded by stringent real-time performance and safety requirements. In mobile and battery-powered systems like robots and drones, managing latency is crucial alongside energy efficiency, as both are essential for functionality and sustainability [13]. Unlike conventional Deep Learning (DL) models such as Convolutional Neural Networks (CNNs), which are primarily used for static inference tasks, DRL requires ongoing training and inference cycles. This continuous processing makes real-time performance and energy management critical operational factors [14]. In these environments, energy efficiency is not just a cost concern; it is essential for functionality and sustainability [13]. Additionally, On-device DRL systems face significant latency during training and inference, complicating the balance of energy and efficiency in dynamic environments.

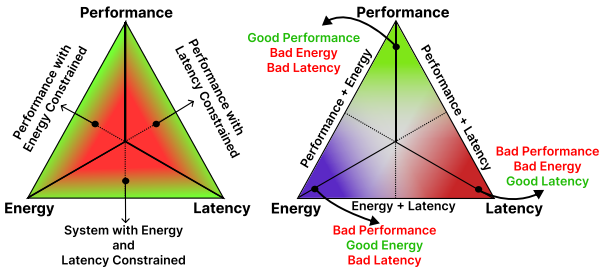



Fig. 1: Trade-offs in DRL system optimization spaces

One critical point that remains unaddressed in the current DRL design space is the provision of energy and latency guarantees while maximizing performance. This approach is particularly beneficial for users who may not prioritize minimal energy use or latency but still require assurances that their systems will not exceed critical thresholds. This is crucial for on-device autonomous systems where breaching energy or latency limits can severely compromise functionality and operational safety. Unfortunately, optimizing the three-dimensional space of energy, latency, and algorithmic performance (i.e., accuracy) of DRL presents inherent conflicts, as these goals often compete with one another, making it challenging to achieve optimal outcomes across all dimensions simultaneously, as demonstrated in Figure 1.

Co-optimizing multiple—often competing—dimensions are clearly critical for modern machine learning-driven systems [15], [16], [17]. While numerous studies [18] have explored optimizing both latency and energy in traditional DNN-driven systems, particularly those using CNNs, these strategies do not address the unique requirements of DRL [19]. DRL necessitates continuous train-inference cycles and frequent retraining to adapt to new environments, introducing diverse computational complexities that present entirely new and challenging problems not encountered in prior efforts [20]. For on-device DRL systems, a couple of recent works have attempted to co-optimize two out of the three critical dimensions. [21] investigates a hardware accelerator-based approach designed for energy-efficient DRL but overlooks latency deadlines and

979-8-3315-4026-5/24/$31.00 ©2024 IEEE
DOI 10.1109/RTSS62706.2024.00019

lacks a comprehensive software solution, while [22] focuses solely on latency and accuracy, neglecting energy considerations. Despite these varied efforts, none of the existing approaches adequately address the simultaneous maintenance of both latency and energy guarantees while ensuring optimal accuracy performance, specifically tailored for DRL.

**Contributions.** In this paper, we introduce DuoJoule, a novel framework designed to address the unique challenges of DRL workloads by meeting latency deadlines and adhering to energy budgets through dynamic configurations at both the application and system levels. DuoJoule stands out by its ability to satisfy energy and latency targets while simultaneously delivering near-optimal accuracy performance, specifically tailored for the continuous train-inference cycles and frequent retraining of DRL. DuoJoule manages the complex multi-dimensional trade-off space by providing a cross-platform and adaptive framework that accommodates constraints on energy and latency, whether they are soft or hard limits. DuoJoule operates by intricately coordinating adjustments at both the application and system levels for efficient operation. It employs a simple and effective runtime MetricTracker to assess the system against targets and calculate a universal score. This score enables dynamic tuning of DRL parameters and system frequency via DVFS. By addressing the unique characteristics of DRL, DuoJoule provides a groundbreaking approach to balance energy efficiency, latency, and accuracy in on-device DRL systems.

We have implemented DuoJoule on two autonomous embedded platforms with heterogeneous computing capabilities: the relatively powerful NVIDIA Jetson AGX Orin and the rather resource-limited NVIDIA Jetson Nano Orin, both featuring GPU-enabled multicore architectures. We evaluated DuoJoule's performance in meeting latency and energy targets, as well as its accuracy, using the Breakout Atari [23] and CartPole classic control [24] environments. These environments are part of the Autonomous Learning Library [25] and the Arcade Learning Environment [23], which are among the most pervasively investigated libraries in the DRL community [26], [27]. Also, we evaluate three prominent DRL algorithms, including DQN [3], DDQN [4], and C51 [5] Additionally, we demonstrated DuoJoule in a Donkey Car [28] case study, a realistic autonomous driving environment, highlighting the practical applicability of our approach. Our results show that DuoJoule achieves:

- **Versatile Overall Effectiveness:** DuoJoule not only meets timing and energy budgets consistently with strong performance across tasks, demonstrating robust resource management, but also ensures efficient operation on diverse hardware, from the powerful NVIDIA Jetson AGX Orin to the resource-limited Jetson Nano. This adaptability highlights its broad applicability in applications from autonomous vehicles to fire detection drones. (Sec. V-B)
- **Practicality:** Through the Donkey Car test case, which closely mimics real-world driving scenarios, DuoJoule has proven its practical applicability and robustness under realistic operational conditions. (Sec. V-D)

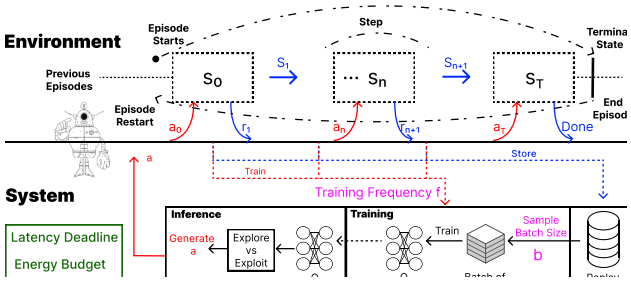



Fig. 2: An overview of Deep Reinforcement Learning (DRL).

- **Adaptivity:** DuoJoule dynamically adapts to varying operational scenarios with different constraints on latency and energy, maintaining flexibility to prioritize resources based on immediate application needs. (Sec. V-C)
- **Low Overhead:** Engineered to impose minimal runtime (as low as 8ms) overhead on existing DRL algorithms, DuoJoule ensures efficient system performance without compromising computational resources. (Sec. V-E)

## II. System Model

In this section, we describe the background of DRL and typical GPU-enabled autonomous embedded hardware platforms.

**DRL Background.** DRL has received significant attention for its ability to learn complex tasks and adapt to dynamic environments, proving invaluable in robotics and autonomous driving [29], [30], [31], [22], [32]. Key components of effective DRL include the agent, environment, policy formulation, reward structure, and episodic execution of steps. As illustrated in Figure 2, an episode in a DRL system involves several steps where each step (or frame in vision-based environments) represents a single iteration in the environment, involve the agent perceiving the current state ($s$), taking action ($a$) based on the state, and receiving feedback in the form of a reward ($r$) which transitions the environment and the agent to the next state ($s'$). This feedback loop continually gathers the past experiences $(s, a, r, s')$ in a buffer known as a replay buffer, allowing sampling from the buffer to update the DNN. In DRL, the Frames Per Second (FPS) metric refers to how quickly the environment's steps or frames are processed and corresponding actions are executed. The performance of DRL algorithms depends on the cumulative (discounted) rewards and the fps as latency, highlighting the dynamic interplay between continuous feedback and systematic learning in advancing autonomous systems.

**DRL Characteristics.** In optimizing DRL for better accuracy, energy, and latency performance within autonomous embedded systems, training frequency $f$ and batch size $b$ are crucial. Unlike CNNs that train on static datasets, DRL continuously adapts to dynamically changing environments influenced by agent interactions. Adjusting the training interval $t$—which determines the number of steps between algorithm updates from the replay buffer—directly affects the update frequency. A lower training interval $t$, meaning fewer frames are skipped,

TABLE I: Step-level average DRL execution energy consumption [mJ] for training and inference on various training frequencies and GPU frequency [MHz].

| GPU Frequency | Training Frequency | Training Energy | Inference Energy |
|---|---|---|---|
| 306 | 1 | 43.1484 | 3.2805 |
| | 4 | 10.8707 | 3.443 |
| 612 | 1 | 43.112 | 3.3359 |
| | 4 | 10.865 | 3.3224 |
| 1020 | 1 | 43.273 | 3.57126 |
| | 4 | 10.874 | 4.2303 |
| 1300 | 1 | 43.69 | 4.225 |
| | 4 | 11.365 | 4.2299 |

results in a higher frequency $f$ of updates, which significantly increases the computational load due to the frame-skipping mechanism. Similarly, batch size impacts the execution, handling multiple data samples from the replay buffer per training session, with the complexity of variable experience quality. Fine-tuning these parameters is vital to balance learning efficiency and resource constraints, ensuring optimal performance within stringent energy and latency budgets. Figure 2 illustrates the roles of these parameters in DRL training.

**Power configurations**. Dynamic Voltage and Frequency Scaling (DVFS) is a power management technique used in GPU-enabled autonomous embedded systems like NVIDIA Jetson Nano and Jetson AGX Orin [33]. By adjusting processor frequency in real-time, DVFS balances energy efficiency and latency. These systems offer multiple power configurations [33], [34], enabling a choice between lower frequencies for improved energy efficiency or higher frequencies for reduced latency, thus optimizing system performance. This trade-off provides opportunities for optimizing system performance across different operational contexts as discussed in previous works such as [35], [36]. Table I lists energy consumed averaged by training and inference steps during DRL execution, under various training frequency and GPU frequency. We can observe two significant patterns from it. (1) A training step consumes much more energy than an inference step, particularly under an intensive training frequency. (2) Different power configurations may have an impact on the energy consumption of DRL execution. Additionally, training energy in I remains stable due to fine-grain cycle calculations and near-maximum GPU utilization during training. Lowering GPU frequency prolongs training and increases total energy consumption while increasing frequency shortens the cycle. This highlights that compared to traditional deep learning techniques that mainly involve inference cycles, supporting on-device DRL can be much more challenging due to the rather energy-intensive nature of training cycles along with inferences.

## III. Motivation

In this section, we conduct several case studies on GPU-enabled autonomous embedded systems which reflect key insights that highlight the primary challenges in developing DRL systems that co-optimize energy, latency, and accuracy.

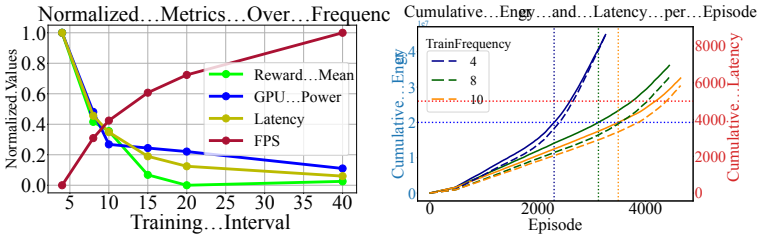
### A. Training Frequency

In this subsection, we present a detailed analysis conducted using the Breakout Atari benchmark [23], utilizing the Autonomous Learning Library (ALL)[25] with PyTorch backend [37] on a Jetson AGX Orin platform. Experiments in this section are all done using C51 [5] algorithm with default parameters of (ALL) and batch size of 64. As illustrated in Figure 3, we delve into the impact of training frequency on three critical optimization metrics: algorithm performance, latency, FPS, and GPU energy consumption. Specifically, Figure 3a examines the trade-offs and trends influenced by decrease in training interval or increases in training frequency. For this experiment, we collect energy, latency, FPS, and reward for each episode across 8 different training frequencies, which reveals a reduction in training frequency or increase in training interval typically results in decreased algorithm performance, better latency, and energy. In contrast, Figure 3b collects cumulative energy and latency data across episodes for three training frequencies, with horizontal lines representing energy and latency budgets. Vertical lines indicate episodes where training exceeds these budgets. This figure demonstrates how imposed budgets on energy or latency limit the number of episodes and frames, thus limiting the accuracy of a DRL system. Further in Figure 3c, we can visualize the impact of the same latency and energy budgets on performance given a cutoff (represented by vertical lines). The figure displays how algorithm performance is capped due to constraints. As the training frequency increases, the cumulative rewards decrease along with the episodes. This suggests training frequency significantly impacts both latency and energy outcomes, potentially preventing the system from achieving its training objectives.

**Observation 1:** Optimal training frequency is crucial in DRL systems to balance achieving higher rewards with managing the constraints of increased latency and energy consumption. Reducing training frequency can decrease algorithm performance but significantly lowers both latency and energy usage. This necessitates careful tuning, particularly under strict latency and energy budgets, to avoid premature training cessation and ensure effective system performance.

### B. Training Batch Size

Following the discussion on training frequency trade-offs, we now turn our attention to another critical parameter: training batch size, as previously introduced. The experimental setup remains consistent with that described in Section III-A with a fixed training frequency of 4. Figure 4 presents a systematic analysis aimed at understanding the impacts of different training batch sizes on algorithm performance, latency, and GPU energy consumption. As shown in Figure 4a, the results delineate trends in DRL algorithm performance, latency, and energy as influenced by varying batch sizes. It features the collection of energy, latency, and accuracy data across nine different batch sizes, comparing the trends of normalized values to delineate how scaling batch size influences efficiency and effectiveness. While an increase in batch size entails

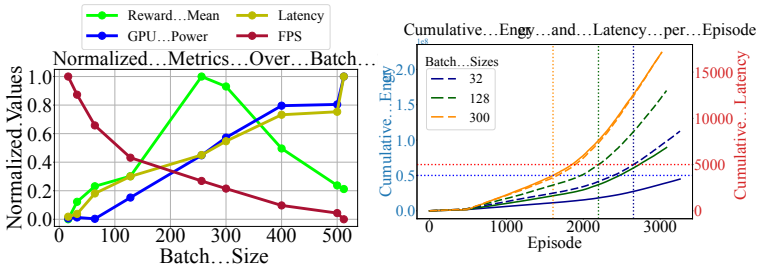



(a) General impacts of training frequency.

(b) Impacts of training frequency under latency or energy budgets.

(c) Algorithm performance for varying training frequency under the same latency or energy budgets in (b).

Fig. 3: Figure (a) illustrates the trend of 8 different training frequencies on the 3 trade-offs among training performance, energy, and latency where the best performance occurs at the highest frequency, unlike latency and energy. Figure (b) presents energy and latency usage per episode under constraints. The horizontal dashed lines denote the imposed constraints, while the vertical dashed lines mark the episode cutoff numbers after reaching those constraints. The solid line indicates the cumulative energy and the dashed line demonstrates the cumulative latency in the experiment. Figure (c) depicts the algorithm's performance cutoff (horizontal line) under the same constraints (vertical lines). Algorithm performance dropped significantly when increasing the training interval from 4 frames to 10 frames per training because the agent trains less frequently, leading to slower learning and adaptation.




(a) Impact of varying batch size.

(b) Impact of batch size under latency or energy budgets.

(c) Algorithm performance for varying batch size under the same latency or energy budgets as in (b).

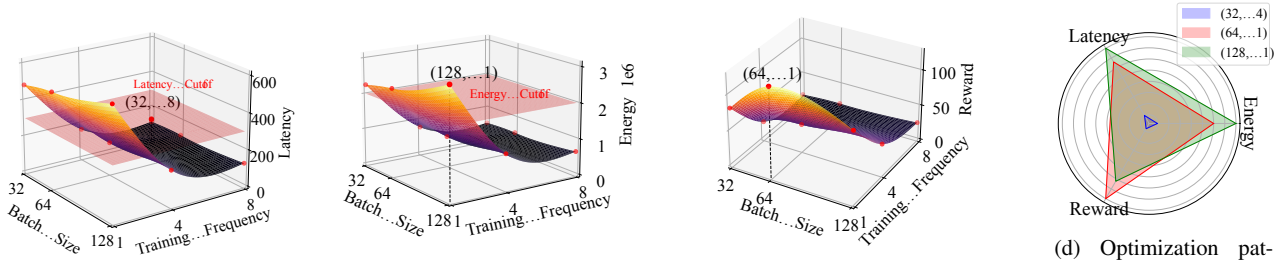
Fig. 4: Figure (a) illustrates the trend of varying 9 training batch sizes on the 3 trade-offs among training performance, energy, and latency where the best performance occurs at 256 batch size, unlike latency and energy. Figure (b) presents energy and latency usage under constraints per episode in each experiment. The horizontal dashed lines denote the imposed constraints, while the vertical dashed lines mark the episode cutoff numbers after reaching those constraints. The solid line indicates the cumulative energy and the dashed line demonstrates the cumulative latency in the experiment. Figure (c) depicts the algorithm's performance cutoff (horizontal line) under the same constraints (vertical lines). Algorithm Performance increased while increasing training batch size from 32 frames to 500.

larger training steps, excessively large batch sizes can lead to training with less informative experiences, potentially causing overfitting and a subsequent decline in performance beyond a certain point. This phenomenon establishes a 'sweet spot' for the batch size, optimizing the trade-off between performance, latency, and energy—a concept also supported by previous studies [38]. Figure 4b illustrates how constraints on latency and energy (indicated by horizontal lines, similar to those used in the training frequency experiments) impact the number of episodes that can be processed. This figure collects cumulative energy and latency data over episodes from experiments of three different settings of batch size, providing a detailed view of operational dynamics. Despite the potential for higher batch sizes to converge to greater rewards, performance differences are minimal due to these constraints. Figure 4c then displays the DRL performance under constraints that terminate training after a certain number of episodes.

**Observation 2:** Increasing training batch sizes in DRL systems typically worsens GPU energy consumption and latency, without clearly improving learning outcomes. A discernible trend shows that beyond a certain 'sweet spot', additional increases in batch size are detrimental. Finding this sweet spot necessitates extensive search through all possible configurations; a fixed value overlooks the system's resource constraints, emphasizing the need for strategic batch size adjustments.

### C. Co-optimizing Training Parameters

Building on our analyses of training frequency and batch size, we delve into the co-tuning of these two control knobs. As depicted in Figure 5, we conduct combinations of experiments using the CartPole environment [24], C51 algorithm [5], and the autonomous learning library [25] on a Jetson AGX Orin platform by varying training batch size and frequency. We changed the environment from Breakout Atari to CartPole to demonstrate the effects of frequency (f) and batch size (b) across different types of environments. By




(a) Configurations for optimal latency cannot meet energy budgets.

(b) Configurations for optimal energy cannot meet latency budgets.

(c) Configurations for algorithm performance yield sub-optimal both latency and energy.

(d) Optimization patterns for different configurations of batch size and training frequency.

Fig. 5: Exploring Trade-offs: A three-dimensional analysis of latency, energy, and reward. Red planes in each plot demonstrate an example of how a budget constrain will cut down latency or energy which results in decrease of performance.

including CartPole, an easier environment, alongside the more challenging Breakout Atari, we show that the trade-offs and the importance of fine-tuning f and b are relevant in both simple and complex scenarios. This highlights that even in easier environments like CartPole, trade-offs exist, underscoring the necessity of optimizing f and b for improved performance. Results highlight an interesting observation where the optimal settings for energy, latency, and algorithm performance occur at different combinations of training frequency and batch size. Unfortunately, this sub-linear relationship complicates the co-optimization of DRL training parameters under energy and latency constraints, primarily due to the interconnect effects where modifying both knobs exacerbate resource consumption nonlinearly. Figure 5a explicitly shows how the batch size and training frequency influence latency, with increases in both parameters, significantly raising latency levels. Figure 5b illustrates the same interconnect effects on energy consumption, where the red plane indicates how a budget on latency and energy can limit viable experiment configurations. Furthermore, Figure 5c identifies the optimum batch size and frequency for training performance; however, budget constraints can potentially alter these ideal settings. Figure 5d shows the radar chart of the three optimum points. Moreover, our analysis demonstrates an interesting insight that *training frequency has a greater impact on latency, while batch size more significantly affects energy consumption, as indicated in Equations 1 and 2*. These insights effectively motivate our system design and implementation, providing a nuanced understanding of the intricate trade-offs in the co-optimization space.

$$\frac{\text{Normalized Batch Size Latency Difference}}{\text{Normalized Frequency Latency Difference}} = 65\% \quad (1)$$

$$\frac{\text{Normalized Batch Size Energy Difference}}{\text{Normalized Frequency Energy Difference}} = 23\% \quad (2)$$

**Observation 3:** Optimizing training frequency and batch size independently can lead to suboptimal performance across different dimensions, exacerbated by their interconnected effects. The challenge becomes more severe under specific energy or latency constraints, where naive adjustments may breach these limits due to both parameters affecting energy and latency. Even though there may be a sweet spot for batch size or training frequency obtained by exhaustive offline search that provides optimal performance, it neglects resource constraints, potentially violating energy or latency budgets, and ignores co-optimization benefits. Our analysis demonstrates that training frequency more significantly impacts latency, while batch size has a greater effect on energy consumption. Additionally, fixed sweet spots for batch size and frequency don't exist due to their interdependency and runtime dynamics, preventing administrators from using offline sweet spots. This highlights the need for strategic parameter adjustments to meet constraints while optimizing for the highest possible reward.

### *D. DRL Specific Energy Patterns*

We conduct a comprehensive analysis of the DRL's latency, energy, and performance by measuring training energy and latency across six GPU frequencies under 15W and 30W power profiles on the Jetson AGX Orin platform. Our experiments reveal an inherent negative correlation between energy consumption and latency: higher frequencies improve latency but increase energy usage, as shown in Figure 6. Another notable observation relates to the characteristics of DRL, which cycles between training and inference. While scaling frequency up during training and down during inference might seem advantageous for saving energy, our findings reveal substantial overhead. Each DVFS adjustment involves an I/O operation of writing frequency on the system file, which can take up to 1-2ms. Given the inference-training cycle of DRL can be under 100ms, this results in up to 20 I/O operations per second (2 I/O per cycle for read and write), making the overhead significant and impractical.

**Observation 4**: Adjusting GPU scaling frequency affects DRL latency and energy, highlighting the importance of using DVFS to dynamically meet energy and latency targets by complementing changes in training frequency and batch size.

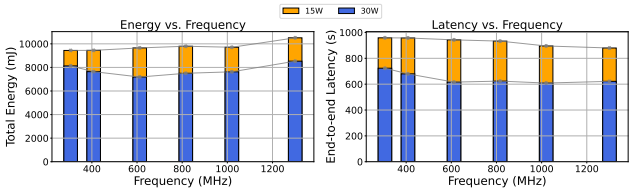



Fig. 6: Comparison of DVFS metrics. *Right:* shows the relationship between end-to-end latency and GPU Frequency. *Left:* shows the relationship between the total energy consumed vs. GPU frequency. In all cases, the GPU frequency was fixed for the entire duration of the experiment.

## IV. Design

In this section, we outline the design of DuoJoule, as shown in Figure 7, to effectively manage system latency and energy consumption within predetermined budgets. DuoJoule employs a Metric Tracker to monitor system status by comparing the expected latency and energy at the end of the experiment to the set budgets. This comparison yields a comprehensive system status score, which serves as a basis for optimizing the balance between energy and latency. The three principal techniques—ParamTuner, DVFS, and EarlyExit—leverage this score to efficiently manage system performance within the constraints. Building on the trade-offs discussed in Section III, we provide detailed explanations of each technique to ensure that the DRL system operates efficiently within the established energy and latency limits.

Latency and energy budgets for DuoJoule are user-defined but adjustable during runtime to reflect changes in task demands or system performance. DuoJoule dynamically adapts to these updates, optimizing performance within the revised constraints. Soft constraints where in systems it usually indicates an easier target and where system can tolerate exceeding budget within a threshold, are set at 75% of maximum system peak to handle temporary fluctuations without impacting overall performance, while hard constraints where systems cannot tolerate above their harder budgets, are fixed at 50% of the peak to ensure safety and prevent overloads. Energy budgets typically stem from battery capacity or system capabilities, whereas latency budgets are shaped by operational requirements such as fast prototyping or production needs [39], [40]. This flexibility allows DuoJoule to maintain optimal performance under varying conditions and constraints.

As a result, DuoJoule dynamically adjusts both algorithm parameters and system configurations in response to the system status score, successfully meeting predefined energy and latency budgets while optimizing performance within these constraints. Additionally, unlike offline approaches like exhaustive search, DuoJoule effectively adopts run-time dynamics such as varying total frames and budgets, making DuoJoule a comprehensive approach to on-device DRL optimization. Algorithm 3 and the following sections describe different DuoJoule and its components.

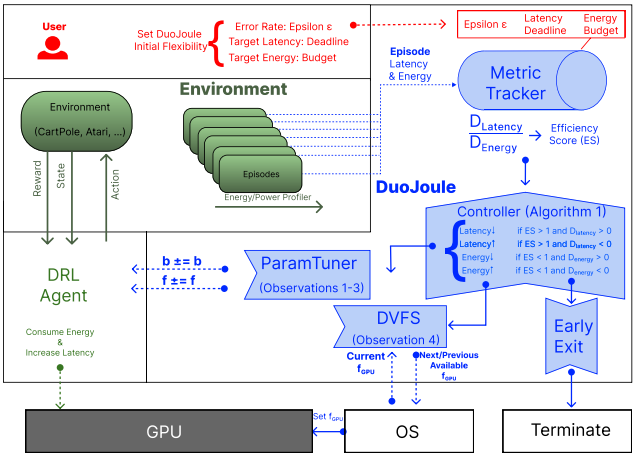



Fig. 7: Design overview of DuoJoule

### A. Metric Tracker

The Metric Tracker is a pivotal component of DuoJoule (Lines 3-4 in Algorithm 3), meticulously monitoring the DRL's performance at each episode by tracking FPS, average GPU power consumption, and session latency. This data enables the Metric Tracker to forecast training energy and latency, allowing it to compare these predictions with preset budgets and calculate a comprehensive system status score. This score informs DuoJoule in making dynamic adjustments to algorithm and system settings, optimizing them to meet the energy and latency budgets. Designed to operate with minimal overhead—as confirmed by our evaluations in Section V-E—the Metric Tracker ensures robust real-time performance without sacrificing efficiency. Furthermore, DuoJoule requires an initial total number of frames for training which might be challenging to determine in real-world DRL applications, as models may converge at different rates, or fail to converge if the total frame count is too low. However, DuoJoule only requires an initial estimate of the total frames. During runtime, if the total frame count changes, DuoJoule dynamically recalculates the expected latency and energy, and adjusts hyperparameters accordingly. This allows DuoJoule to maintain optimal performance and accuracy while respecting the initial constraints, even as the frame count varies.

Before delving into the formal definitions of the Metric Tracker's calculations, it is important to establish the notations used. We represent the latency and energy targets as $L_t$ and $E_t$, respectively. The current latency and energy consumption up to this point in the system are denoted by $L_c$ and $E_c$. In our setup, the total number of frames for training is set at the beginning (flexible to vary during runtime) of the experiment and denoted as $F_t$. The remaining number of frames until the end of the experiment is referred to as $F_r$. $\overline{fps}$ denotes the average fps of last episode window. $P_{EP}$ is defined as episode-level power consumption, $\overline{P_{EP}}$ denotes the average power consumption of the last episode window, $L_{EP}$ is episode-level latency, and $f_{EP}$ is the number of frames (step) in the episode. $\overline{P_{EP}} \times L_{EP}$ calculates the energy consumption during

the episode where dividing it by $f_{EP}$ results in an estimate of energy consumption per frame. Based on this information, we can calculate the expected latency and expected energy consumption as follows:

$$L_{et} = L_c + \frac{F_r}{fps} \tag{3}$$

$$E_{et} = E_c + \overline{\left[\frac{\overline{P_{EP}} \times L_{EP}}{f_{EP}}\right]} \times F_r \tag{4}$$

then the deviation of the calculated expected values from the respective latency and energy budgets can be calculated based on Eq. 4 as follows:

$$D_{\text{latency}} = \frac{L_{et} - L_t}{L_t} \tag{5}$$

$$D_{\text{energy}} = \frac{E_{et} - E_t}{E_t} \tag{6}$$

**Definition 1: Efficiency Score (ES).**

$$\text{Efficiency\_Score (ES)} = \frac{|D_{\text{latency}}|}{\alpha \times |D_{\text{energy}}|} \tag{7}$$

The *Efficiency_Score* (ES) metric, derived by comparing the absolute values of $D_{\text{latency}}$ and $D_{\text{energy}}$ as calculated in Eq. 6, assists in identifying which aspect-latency or energy-requires more immediate attention. The sign of $D_{\text{latency}}$ and $D_{\text{energy}}$ indicates the direction of deviation: a positive sign signifies the value exceeds its budget, while a negative sign indicates compliance with the budget. Moreover, the weight $\alpha$ quantifies the relative importance of energy compared to latency. For instance, an $\alpha$ of 2 signifies that meeting the energy target is twice as important as meeting the latency target.

The absolute values of $D_{\text{latency}}$ and $D_{\text{energy}}$, representing the normalized differences from performance targets, can be treated as the error rate $\epsilon$. These error rates quantify how much the actual performance deviates from the budgets. When the absolute value of $D_*$, where $* \in \{\text{latency}, \text{energy}\}$, is less than a predefined threshold $\epsilon$, the system is considered stable, indicating that it is within an acceptable range of its budgetary goals. This threshold, $\epsilon$ which is defined by the system initially, thus not only signifies the system's tolerance for deviations but also determines its operational flexibility, allowing `DuoJoule` to adjust its strategy based on the magnitude of $D_*$. A smaller $\epsilon$ means tighter control, aiming for precision in meeting targets, while a larger $\epsilon$ offers more leeway, reducing the frequency of adjustments but potentially allowing larger deviations.

The development of the ES metric responds directly to the co-optimization challenges highlighted in Observation 3 (Section III-C). ES simplifies the simultaneous management of latency and energy by merging their deviations into a single actionable score. This metric, as implemented in the controller part of `DuoJoule` (Algorithm 3), optimally adjusts training batch size, frequency, and DVFS settings based on the stability criterion—keeping the absolute value of deviation $D_*$ below the error rate $\epsilon$. The algorithm also limits exploration (See sections IV-B and IV-C) once the system stabilizes, ensuring minimal computational overhead and eventual system stability. Its effectiveness and simplicity directly address the issues presented in Section III, making it a practical tool for optimizing real-time embedded systems and navigating the intricate balance between latency and energy.

**Algorithm 1** ParamTuner component of `DuoJoule`.

**Require:** Minibatch size $b$, training frequency $f$, Last Updated Parameter $p$, Minibatch size modification unit$b^*$

**function** PARAMTUNER(mode, f, b, constrain)
  **if** $constrain ==$ "energy" **or** "explore" **then**
    **if** $mode ==$ "increase" **or** "explore" **then**
      **if** $p$ **or** $f \leq 1$ **then**
        $b \leftarrow b + b^*$
      **else**
        $f \leftarrow f - 1$
      **end if**
    **else if** $mode ==$ "decrease" **then**
      **if not** $p$ **or** $b < b^*$ **then**
        $f \leftarrow f + 1$
      **else**
        $b \leftarrow b - b^*$
      **end if**
    **end if**
  **else if** $constrain ==$ "latency" **or** "explore" **then**
    **if** $mode ==$ "increase" **or** "explore" **then**
      **if** $p$ **or** $b < b^*$ **then**
        $f \leftarrow f + 1$
      **else**
        $b \leftarrow b - b^*$
      **end if**
    **else if** $mode ==$ "decrease" **then**
      **if not** $p$ **or** $f \leq 1$ **then**
        $b \leftarrow b + b^*$
      **else**
        $f \leftarrow f - 1$
      **end if**
    **end if**
  **end if**
  $p \leftarrow$ **not** $p$
  **return** $b, f$
**end function**

### B. ParamTuner

As previously discussed, the ES assesses deviations from latency and energy budgets by evaluating the sign and magnitude of $D_*$. In response, ParamTuner, the main component of `DuoJoule`, adjusts key parameters, batch size ($b$), and training frequency ($f$), to meet budget constraints. Utilizing observations from Section III, batch size influences energy consumption, while training frequency primarily impacts latency. Depending on the ES, adjustments are made accordingly: if $D_{\text{energy}}$ is positive, $b$ is decreased to reduce energy usage; if negative, $b$ is increased by a constant value $b^*$ which

often is small and depends on the environment and algorithm (in our setup we used 8 for CartPole and 16 for Breakout). Similarly, if $D_{\text{latency}}$ is positive, $f$ is decreased to improve latency; if negative, $f$ is increased by 1. These adjustments continue until the system stabilizes, ensuring compliance with the set budgets. Algorithm 1 shows the ParamTuner algorithm. However, this naive ParamTuner faces several challenges.

ParamTuner (Lines 5-14 in Algorithm 3) navigates a complex dynamic where both batch size ($b$) and training frequency ($f$) simultaneously influence latency and energy consumption, as discussed in Section III-C. The interdependence of these parameters can cause compounding effects and system instability when adjusted concurrently. To mitigate this, ParamTuner employs a sequential adjustment strategy based on the ES, modifying one parameter at a time followed by a stabilization period within a defined change window $w_p$. This approach allows the system to adapt and reflect the new settings' impacts before further adjustments, ensuring controlled, measurable changes. Despite fostering stability, this method can restrict the exploration of potentially superior configurations. If `DuoJoule` shows that performance is within target limits (deviation from target less than error), further adjustments cease, potentially overlooking optimal settings. To address this, ParamTuner intermittently experiments with increasing $b$ or $f$ after periods of stability, assessing the impact on ES within a toggle strategy framework (Line 3,17 in 1). This exploration is limited to several attempts and strikes a balance between achieving system stability and discovering enhanced configurations.

ParamTuner, the core component of the `DuoJoule` system, effectively balances energy and latency to meet predefined budgets while optimizing for the best performance configuration. Enhancements such as DVFS and EarlyExit further improve `DuoJoule`, ensuring compliance with budgets. Algorithm 3 illustrates the `DuoJoule` framework. Inputs include latency and energy budgets, and an error rate $\epsilon$, which should be specified beforehand. The energy budget is typically set based on the embedded system's workload or battery capacity, while target latency varies with the operational context—whether a prototype run, production, or test—and these values are user-defined, similar to the concept described in [41]. Although these values are initially fixed, they can be adjusted during experiments, allowing `DuoJoule` to adapt. The total number of training frames in DRL is generally predefined, akin to epochs in deep learning [42]. Crucial hyperparameters for the system are the ParamTuner and DVFS adjustment windows, denoted as $w_p$ and $w_d$ respectively.

Target latency and energy define system performance flexibility, while adjustment windows dictate parameter tuning leeway. On devices like the Jetson Orin, GPU power updates take about 5-10 seconds after algorithm adjustments, whereas DVFS changes take just a second. Consequently, the DVFS window ($w_d$) is smaller than the ParamTuner window ($w_p$). Given the variability in response times across systems, establishing suitable windows is crucial. $w_p$ and $w_d$ depend on the platform as different platforms take varying times to reflect system changes. However, `DuoJoule` employs a learning method that can quickly determine these values at runtime by updating f, b, and GPU frequency, and setting the number of frames required to reflect changes in fps and GPU power as windows values. Note that small windows reduce stability and hinder optimal convergence, while large windows decrease responsiveness and extend convergence time.

### C. Dynamic Voltage and Scaling Frequency (DVFS)

DVFS (Lines 15-24 in Algorithm 3), and Algorithm 2, as discussed in Sec. II, is a pivotal technique in managing energy efficiency for GPU-intensive tasks, such as DRL, as it dynamically adjusts GPU frequencies to optimize latency or energy consumption [43], [44]. Unlike ParamTuner, which can significantly alter training performance (see Section III), DVFS adjusts system parameters with no impact on performance, making it a more favorable trade-off. However, DVFS's effect on latency and energy is subsequently smaller than ParamTuner which allows DVFS to provide finer-grained control over system constraints, particularly useful when deviations are marginally outside the acceptable error bounds.

In `DuoJoule`, ParamTuner and DVFS serve complementary roles in managing performance and system efficiency. However, a notable challenge with DVFS is that there is no clear "sweet spot" for GPU frequency, as frequency adjustments often have inverse effects on latency and energy. To address this, `DuoJoule` dynamically selects from the available GPU frequencies at runtime, such as the 11 options provided by systems like Jetson Orin. `DuoJoule` uses the ES in the same way as ParamTuner, increasing (or decreasing) the GPU frequency if the ES is positive (or negative) to prioritize latency (or energy) (2). The DVFS change window $w_d$ is set to be much smaller than that of ParamTuner (see ParamTuner IV-B, allowing DVFS to tune the frequency to stabilize the system after a change in parameters. Similar to ParamTuner, DVFS explores higher or lower frequencies even after stabilization of the system to favor more latency or energy constrained systems.

**Algorithm 2** DVFS component of `DuoJoule`.

**Require:** GPU frequency $f$, List of Available GPU frequency $L_f$

```
1: function DVFS(mode, f)
2:     if mode == "increase" or "explore" then
3:         f ← L_f[next]
4:     else if mode == "decrease" then
5:         f ← L_f[previous]
6:     end if
7:     return f
8: end function
```

### D. EarlyExit

Following our discussion of ParamTuner and DVFS, we introduce EarlyExit (Lines 26-28 in Algorithm 3), a strategy that effectively enhances system efficiency, potentially trading

off some algorithmic performance for improvements in energy and latency. DuoJoule adopts EarlyExit to halt operations when the algorithm converges and performance no longer improves, or when energy and latency budgets are exceeded. This strategy is critical in systems with hard constraints on latency or energy, where adjusting configurations might overly compromise performance. By using an exit window—defined by episodes or frames—DuoJoule can effectively satisfy these targets, terminating the process when further adjustments would lead to unacceptable trade-offs.

Considering system safety is paramount in our EarlyExit mechanism. In the first scenario, where the algorithm has converged, there is no system safety issue, as it has already reached its maximum performance, allowing us to save latency and energy. In the second scenario, where constraints are exceeded, exiting the training improves safety by preventing overuse. DuoJoule's adaptability and flexibility permit dynamic adjustments to constraints during runtime, allowing administrators to increase budgets if the algorithm is running out before convergence. If this is not feasible, exiting training conserves critical battery life for safety or backup systems, with the option to continue training from a checkpoint later. Integrating EarlyExit with ParamTuner and DVFS, DuoJoule efficiently meets stringent operational constraints, ensuring optimal performance within predefined limits.

**Algorithm 3** Controller component of DuoJoule.

**Require:** Latency deadline $L_t$, Energy Budget $E_t$, Total frames $F$, ParamTuner change window $w_p$, DVFS change window $w_d$, error rate $\epsilon$

**function** CONTROLLER(inputs)
  /* Metric Tracker */
  $D_{latency}, D_{energy} \leftarrow$ Calculate by Eq. 6
  $ES \leftarrow$ Calculate $ES$ by Eq. 7
  /* ParamTuner */
  **if** $w_p$ AND ($|D_{latency}| > \epsilon$) OR $|D_{latency}| > \epsilon(\times\alpha)$) **then**
    **if** $|ES| \geq 1$ **then**
      $f, b \leftarrow ParamTuner(signD_{lat.}, f, b, latency)$
    **else if** $|ES| \leq 1$ **then**
      $f, b \leftarrow ParamTuner(signD_{ene.}, f, b, energy)$
    **else**
      $f, b \leftarrow ParamTuner(explore, f, b)$
    **end if**
  **end if**
  /* DVFS */
  **if** $w_d$ AND ($|D_{lat.}| > \epsilon$) OR $|D_{lat.}| > \epsilon(\times\alpha)$) **then**
    **if** $|ES| \geq 1$ **then**
      $freq \leftarrow DVFS(signD_{lat.}, freq)$
    **else if** $|ES| \leq 1$ **then**
      $freq \leftarrow DVFS(signD_{ene.}, freq)$
    **else**
      $freq \leftarrow DVFS(explore, freq)$
    **end if**
  **end if**
  /* EarlyExit */
  **if** $L > L_t$ OR $E > E_t$ OR $Converged$ **then**
    EarlyExit()
  **end if**
  **return** $freq, b, f$
**end function**

## V. Evaluation

### A. Experiment Setup

**Testbeds.** We evaluate DuoJoule on two GPU-enabled NVIDIA platforms, Jetson AGX Xavier and Jetson Nano Orin as testbeds, which are widely used in autonomous driving [45], [46] and robotics [47], [48], [49], [50], where reliability and performance of energy and latency are critical [51].

**Benchmarks and DRL algorithms.** We evaluate our solution on two representative widely used DRL benchmarks, including Classic Control [24] and Atari [52], which are integrated with the widely used Autonomous Learning Library (ALL) [25]. These two benchmarks represent two different-sized DRL scenarios, where Classic Control is a small benchmark and Atari is a large benchmark. Our experiments evaluate the performance of three prominent DRL algorithms: DQN [3], DDQN [4], and C51 [5]. For DuoJoule, we started all our experiments from the default parameters of ALL. The framework then controls the system and hyperparameters for best reward.

**Robotic Case Study.** To showcase the robustness of our solution in realistic settings, we conduct a practical case study involving robotic autonomous navigation scenarios, utilizing a high-resolution simulator DonkeyCar [28].

**Metrics.** Our evaluation uses three main types of metrics: latency predictability, assessed by end-to-end latency; algorithm performance, measured by maximal cumulative rewards; and GPU energy consumption, profiled using the Tegrastats profiler [53] to measure system-level power consumption and calculate total energy consumption. We had at least 3 runs for each experiment to make sure our framework and results were consistent. However, we only showed one of the runs as an evaluation and in our plots.

**Baselines.** We compare DuoJoule against four baselines:

- **Default**: This approach utilizes preset training parameters from the Autonomous Learning Library (ALL), representing a balanced configuration for general user preferences.
- **MAX-A**: MAX-A optimizes training parameters for best algorithm performance.
- **MIN-E**: MIN-E optimizes training parameters for achieving best energy efficiency.
- **R**$^3$ [22]: This state-of-the-art solution considers auto-balancing timing and algorithm performance under hard memory constraints.

**Implementation Details.** Experiments on the Breakout environment utilize 1,000,000 frames with an error rate of 5%. CartPole experiments utilize 50,000 frames with an error rate of 5%. We apply both hard and soft constraints on energy and latency for DuoJoule. All evaluations in this section

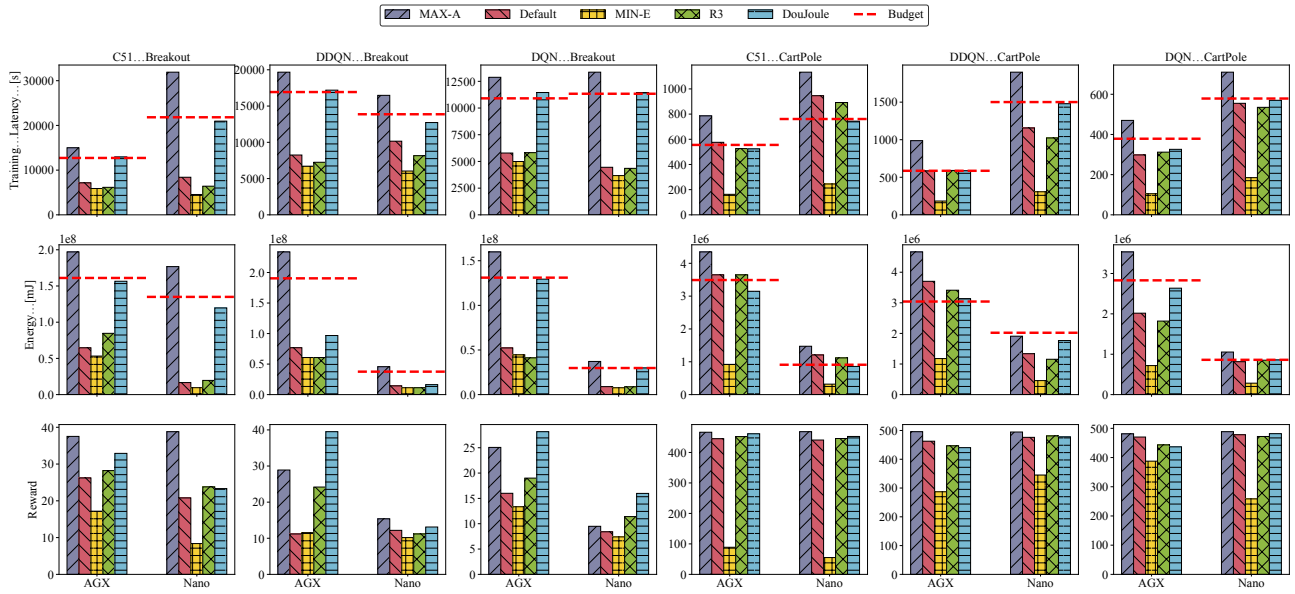



Fig. 8: Overall effectiveness of DuoJoule.

are easily reproducible if one follows the implementation of the algorithms in ALL [25] and obtains the metrics discussed. Results should be consistent with the plots in the paper.

### B. Versatile Overall Effectiveness

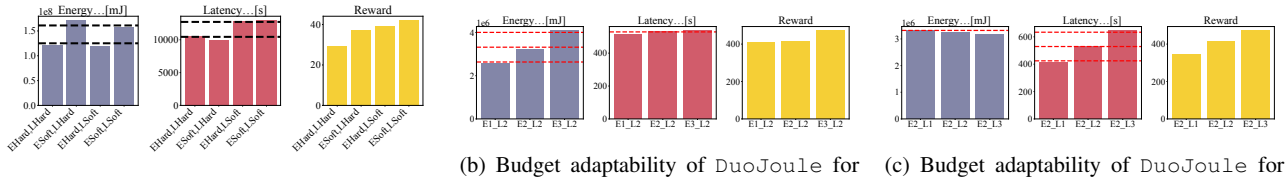
In this subsection, we evaluate the overall effectiveness of DuoJoule on two platforms on both environments with DRL c51 algorithm. We set the constraints of DuoJoule to soft energy and soft latency in all experiments in this subsection. For a clear comparison and demonstration, we separately discuss the performance of DuoJoule's comparison across all baselines under three metrics. The main goal of DuoJoule is to achieve the highest possible algorithm performance while consistently meeting systems' energy and latency budgets.

**Latency Predictability.** As displayed in the first row of Fig. 8, on the small benchmark CartPole, DuoJoule consistently outperforms MAX-A by an average of 22.13%. Additionally, DuoJoule always meets the latency budgets on the small benchmark, but MAX-A, Default, and $R^3$ run out of latency budgets by 6, 1, and 2 times, out of 6 experiments respectively. On the large benchmark Breakout, Default and $R^3$ achieve lower training latency than DuoJoule by 52.31% and 63.01%, since they all consider timing as an important metric but energy-agnostic. However, MAX-A underperformed DuoJoule by an average 25.28% higher latency. Note that considering latency budgets, DuoJoule can still almost use up them while Default and $R^3$ overshoot to optimize latency, which may lead to lower algorithm performance.

**Energy Efficiency.** As displayed in the second row of Fig. 8, the small benchmark CartPole, DuoJoule consistently outperforms MAX-A on average by 30%. Meanwhile, DuoJoule consumes more energy than MIN-E by 112%. It is since MIN-E aims to optimize energy efficiency without considering algorithm performance. We observe that DuoJoule and MIN-E always meet energy budgets on the small benchmark (DuoJoule may end before reaching budgets), but Default, MAX-A, and $R^3$ run out of energy budgets by 4, 6, and 1 times, respectively. On the large benchmark, MIN-E achieves lower energy consumption than DuoJoule by 52.85%. While Default and $R^3$ overshoot in optimizing the budgets (use approximately on average only 71%, 68% of budget respectively, while DuoJoule uses 98%). Note that, considering energy budgets, DuoJoule uses nearly the entire budget to achieve the highest possible performance, while $R^3$ optimizes memory usage and latency and MIN-E only optimizes energy.

**Algorithm Performance.** As demonstrated in the third row of Fig. 8, on the small benchmark CartPole, DuoJoule achieves near-optimal algorithm performance. DuoJoule trades off by only 4.24% on average compared to MAX-A which tries to optimize only performance while DuoJoule outperforms MAX-A in energy by 31.33% and latency by 34.71%. DuoJoule outperforms MIN-E on average of algorithm performance by 98.7%. On the large benchmark Breakout, DuoJoule consistently outperforms all baselines except MAX-A in terms of algorithm performance. However, MAX-A cannot meet latency and energy budgets under all experiments on the large benchmark. This indicates that DuoJoule could strike a good balance in achieving good algorithm performance, particularly in large benchmarks under system constraints.

**Versatile overall effectiveness**: DuoJoule auto-balances latency, energy, and algorithm performance under-budgeted DRL training across different platforms.




(a) DuoJoule Adaptability to constraints.

(b) Budget adaptability of DuoJoule for same latency with different energy budgets

(c) Budget adaptability of DuoJoule for same energy with different latency budgets

Fig. 9: DuoJoule can adapt to various constraints scenarios and dynamic budgets.

### C. Adaptability.

We evaluate the adaptability of DuoJoule from two angles. First, we showcase adaptability across different constraints, i.e., soft and hard constraints, on latency and energy, respectively. Second, we demonstrate DuoJoule's adaptability across dynamic budgeting across latency and energy.

**Constrains Adaptability.** In this subsection, we evaluate the adaptability of DuoJoule across four combinations of latency and energy constraints. As depicted in Fig. 9a, we evaluate the differences across these four constraint settings. Generally, ESoft_LSoft demonstrates that sometimes systems can run out of budgets within an acceptable error threshold. However, it still achieves the highest algorithm performance. EHard_LSoft focuses energy as the hard constraint, decreasing running out of energy budgets compared to ESoft_LSoft, but acceptably trade-off algorithm performance by an average of 5.06%. On the other hand, ESoft_LHard focuses latency as the hard constraint, decreasing running out of latency budgets compared to ESoft_LSoft, but trade-off algorithm performance by an average of 7.23%. EHard_LHard reduces almost all running of budget cases on latency and energy budgets, respectively. However, it trades off most algorithm performance, by an average of 35.48% compared to the ESoft_LSoft.

**Dynamic Budgets Adaptability.** In this subsection, we evaluate the adaptability of DuoJoule to dynamic budgets. All experiments for dynamic budgets are based on the C51 algorithm on the CartPole environment on the Xavier platform. First, we evaluate DuoJoule under different energy budgets within the same latency budget as shown in Fig. 9b, as well as, three distinct latency budgets with the same energy budget Fig. 9c. The results demonstrate that DuoJoule can consistently avoid running out of budget while achieving acceptable performance.

> **Adaptability**: DuoJoule consistently maintains effectiveness across various constraint settings and dynamic budgets, ensuring robustness and resilience in different user scenarios.

### D. A Practical Case Study: Autonomous Navigation via DRL

To evaluate the effectiveness of the proposed approach in a real-world scenario, we conduct a realistic case study based on an autonomous navigation context. We utilize the DonkeyCar [28] for our experiments as shown in Fig. 10. DonkeyCar processes high-resolution RGB images at 60 FPS which is significantly challenging for resource-constrained devices, particularly under latency and energy budgets. Specifically, we conduct the case study using a desktop with one GPU, and we choose the DDQN algorithm for this autonomous navigation task. For conducting this experiment, we imposed (ESoft_LSoft) constraints on the budgets.

TABLE II: Quantitative results on DonkeyCar simulator on an x86 desktop. Latency describes the end-to-end latency of a DRL on-device training session, Budget refers to the energy consumption of the DRL algorithm on the device as a percentage. Reward$_{max}$ implies maximal episode rewards, and Reward$_{avg}$ implies average episode rewards. Arrow directions indicate better performance.

| Solution | Latency(↓) | Budget(↑) | R$_{max}$(↑) | R$_{avg}$(↑) |
|---|---|---|---|---|
| MAX-A | 1090.8 | 100.0% | 3980.7 | 272.0 |
| Default | 1082.7 | 95.4% | 3768.6 | 259.2 |
| MIN-E | 1002.4 | 78.2% | 254.1 | 37.1 |
| R3 | 1063.4 | 94.2% | 4187.4 | 359.2 |
| DuoJoule | 924.6 | 88.2% | 4263.7 | 373.7 |

(a) Warehouse Track

(b) Generated Road Track

(c) Waveshare Track

(d) Monaco Track

Fig. 10: Autonomous Navigation Environments in DonkeyCar.

As shown in Table II, DuoJoule can attain on average 7.1% and 1577.96% better maximal episode rewards, and 37.38% and 907.27% better average episode rewards than MAX-A and MIN-E, respectively. Although R3 and Default do not focus on energy optimization, they underperform DuoJoule in latency by 15% and 18%, respectively. R3 achieves nearly the same performance as DuoJoule but uses 6% more budget. This indicates better algorithm performance of DuoJoule under practical latency and energy-constrained scenarios. Also, we observe that MAX-A runs out of its budget, which is dangerous under hard constraints, particularly in safety-critical robotic applications. DuoJoule maximizes the usage of budgets without running out of them, demonstrating its practical usability in intelligent robotic scenarios.

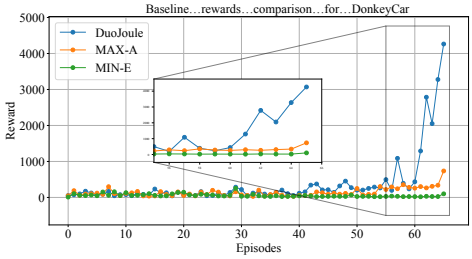



Fig. 11: Qualitative results of DuoJoule on the DonkeyCar environment showing a better algorithm performance than MAX-A and MIN-E with same number of episodes.

TABLE III: Execution overhead [ms] of DuoJoule.

| Platform | Env | ParamTuner | DVFS | EarlyExit |
|---|---|---|---|---|
| Orin | Atari | 8.220 | 2.466 | 0.004 |
| | Classic Control | 1.260 | 4.400 | 0.002 |
| Nano | Atari | 7.430 | 0.065 | 0.003 |
| | Classic Control | 1.210 | 0.050 | 0.003 |

Furthermore, we conduct a detailed examination of the algorithm performance during the training episode by a curve of cumulative rewards as shown in Fig. 11. Qualitatively, DuoJoule's holistic efficient design offers significantly higher algorithm performance quantitatively under practical robotic scenarios by safely handling both the latency and energy budgets.

**Practical Usability**: DuoJoule's effectiveness in a practical autonomous navigation scenario demonstrates its pragmatic nature in intelligent robotic scenarios.

### *E. Overhead Analysis*

In this subsection, we provide a breakdown analysis of DuoJoule to analyze the module-level execution overhead. Table III showcases the execution overhead of the different components of our algorithm, including ParamTuner, DVFS, and EarlyExit. Across two testbeds in our evaluations, the execution overhead remains under 10.690 milliseconds of the end-to-end execution time, underscoring DuoJoule's overall light overhead. Specifically, ParamTuner contributes the most to this overhead, as it computes the ES according to Sec. IV, yet it still takes less than 8.220 milliseconds, which is negligible during DRL runtime. EarlyExit consistently exhibits a very low execution overhead, always lower than 0.004 milliseconds across the NVIDIA Jetson Orin Nano and NVIDIA Jetson AGX Orin, since it will be executed at most only once during the whole DRL execution.

**Low overhead**: DuoJoule's efficient implementation introduces low overhead, making it a lightweight deployment solution for optimizing on-device DRL training.

## VI. Related Work and Discussion

**Related Work.** DRL has become increasingly efficient in tasks such as autonomous driving [54], [55], [29], [56], robotic manipulation [57], health care [58], [59], federated learning [60], [61], and interactive gaming [42], [52]. Advanced algorithms like SAC [9], TRPO [8], and A3C [62] have made significant strides. Efforts are underway to generalize DRL to solve multiple tasks [63] and to develop agents that require less experience [64]. However, these works primarily focus on optimizing algorithms without addressing latency and energy efficiency. A few recent works have attempted to co-optimize two of the three critical dimensions. [21] uses accelerators for energy-efficient DRL but ignores latency and lacks a full software solution, while [22] prioritizes latency and accuracy, neglecting the energy aspect. However, optimizing all three dimensions in on-device DRL remains challenging due to their conflicting and complex interactions. Furthermore, previous works have explored quantization [65], [66] and approximate computing [67], [68] to reduce latency and energy in on-device DRL. Although quantization, including quantization-aware training [69], improves efficiency, it often compromises accuracy. In contrast, DuoJoule optimizes batch size and training intervals to meet budget constraints without sacrificing precision, while still allowing the application of quantization if needed. This is the first work to co-optimize latency, energy, and performance for on-device DRL under budget constraints, addressing their complex interdependencies.

**Limitations of the Proposed Method.** DuoJoule offers a pure software-level solution, while some works focus on hardware optimization for energy efficiency [21]. It provides an energy-aware optimization strategy for replay-based DRL, but its performance with other DRL methods remains unexplored. Also, DuoJoule does not consider energy consumption beyond SoC, i.e., motor or control modules, which could also make an impact on energy efficiency in the intelligent robots context. Additionally, DuoJoule does not account for mixed workload scenarios, which are common in advanced commercial autonomous systems, such as Tesla vehicles, where multiple components may run concurrently. Future work will investigate these areas further.

## VII. Conclusion

DuoJoule offers a practical framework that addresses the critical challenge of co-optimizing energy, latency, and algorithm performance in on-device DRL workloads. Extensive evaluations on two autonomous embedded platforms have demonstrated DuoJoule's versatile cross-platform efficiency, practicality, adaptability, and low runtime overhead. It consistently meets latency deadlines and adheres to energy budgets while maintaining near-optimal performance. By leveraging a metric tracker and dynamically tuning DRL algorithm parameters and system frequency settings, DuoJoule provides a flexible and adaptive solution. This approach not only enhances the efficiency and applicability of DRL in real-world autonomous systems but also ensures that these systems can operate effectively under varying conditions. The ability to co-optimize across multiple performance dimensions under resource constraints underscores the practical value of DuoJoule in the field of on-device DRL.

## REFERENCES


[1] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[2] S. Shirvani, Y. Baseri, and A. Ghorbani, "Evaluation framework for electric vehicle security risk assessment," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[4] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[5] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *International conference on machine learning*. PMLR, 2017, pp. 449–458.

[6] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[8] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[10] S. Nikkhoo, Z. Li, A. Samanta, Y. Li, and C. Liu, "Pimbot: Policy and incentive manipulation for multi-robot reinforcement learning in social dilemmas," *arXiv preprint arXiv:2307.15944*, 2023.

[11] S. He, S. Han, and F. Miao, "Robust electric vehicle balancing of autonomous mobility-on-demand system: A multi-agent reinforcement learning approach," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 5471–5478.

[12] S. He, Y. Wang, S. Han, S. Zou, and F. Miao, "A robust and constrained multi-agent reinforcement learning electric vehicle rebalancing method in amod systems," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 5637–5644.

[13] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3826–3839, 2020.

[14] M. Q. Mohammed, K. L. Chung, and C. S. Chyi, "Review of deep reinforcement learning-based object grasping: Techniques, open challenges, and recommendations," *IEEE Access*, vol. 8, pp. 178 450–178 481, 2020.

[15] C. Wan, M. Santriaji, E. Rogers, H. Hoffmann, M. Maire, and S. Lu, "{ALERT}: Accurate learning for energy and timeliness," in *2020 USENIX annual technical conference (USENIX ATC 20)*, 2020, pp. 353–369.

[16] S. Wang, H. Hoffmann, and S. Lu, "Agilectrl: a self-adaptive framework for configuration tuning," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 459–471.

[17] A. Ghandehari, S. Shirvani, and H. Moradi, "Evaluating the impact of traveling on covid-19 prevalence and predicting the new confirmed cases according to the travel rate using machine learning: A case study in iran," in *2021 11th International Conference on Computer Engineering and Knowledge (ICCKE)*. IEEE, 2021, pp. 290–295.

[18] N. Mishra, H. Zhang, J. D. Lafferty, and H. Hoffmann, "A probabilistic graphical model-based approach for minimizing energy under performance constraints," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1, pp. 267–281, 2015.

[19] A. Heidari, N. J. Navimipour, M. A. J. Jamali, and S. Akbarpour, "A hybrid approach for latency and battery lifetime optimization in iot devices through offloading and cnn learning," *Sustainable Computing: Informatics and Systems*, vol. 39, p. 100899, 2023.

[20] I. Budhiraja, N. Kumar, H. Sharma, M. Elhoseny, Y. Lakys, and J. J. Rodrigues, "Latency-energy tradeoff in connected autonomous vehicles: A deep reinforcement learning scheme," *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[21] J. Lee, C. Kim, D. Han, S. Kim, S. Kim, and H.-J. Yoo, "Energy-efficient deep reinforcement learning accelerator designs for mobile autonomous systems," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2021, pp. 1–4.

[22] Z. Li, A. Samanta, Y. Li, A. Soltoggio, H. Kim, and C. Liu, "R3: On-device real-time deep reinforcement learning for autonomous robotics," in *2023 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2023, pp. 131–144.

[23] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[24] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.

[25] C. Nota, "The autonomous learning library," https://github.com/cpnota/autonomous-learning-library, 2020.

[26] D. Patel, H. Hazan, D. J. Saunders, H. T. Siegelmann, and R. Kozma, "Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to atari breakout game," *Neural Networks*, vol. 120, pp. 108–115, 2019.

[27] S. Nagendra, N. Podila, R. Ugarakhod, and K. George, "Comparison of reinforcement learning algorithms applied to the cart-pole problem," in *2017 international conference on advances in computing, communications and informatics (ICACCI)*. IEEE, 2017, pp. 26–32.

[28] T. Kramer, "Openai gym environments for donkey car," 2023.

[29] S. He, S. Han, and F. Miao, "Robust electric vehicle balancing of autonomous mobility-on-demand system: A multi-agent reinforcement learning approach," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 5471–5478.

[30] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017.

[31] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[32] H. Chae, C. M. Kang, B. Kim, J. Kim, C. C. Chung, and J. W. Choi, "Autonomous braking system via deep reinforcement learning," in *2017 IEEE 20th International conference on intelligent transportation systems (ITSC)*. IEEE, 2017, pp. 1–6.

[33] "Jetson orin nx series and jetson agx orin series," https://docs.nvidia.com/jetson/archives, accessed: 2024-04-24.

[34] L. S. Karumbunathan, "Nvidia jetson agx orin series," https://www.nvidia.com/content/dam/en-zz/Solutions/gtcf21/jetson-orin/nvidia-jetson-agx-orin-technical-brief.pdf.

[35] C. Lin, K. Wang, Z. Li, and Y. Pu, "A workload-aware dvfs robust to concurrent tasks for mobile devices," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, 2023, pp. 1–16.

[36] H. Huang, M. Lin, L. T. Yang, and Q. Zhang, "Autonomous power management with double-q reinforcement learning method," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 1938–1946, 2019.

[37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[38] M. Choi, "An empirical study on the optimal batch size for the deep q-network," in *Robot Intelligence Technology and Applications 5: Results from the 5th International Conference on Robot Intelligence Technology and Applications 5*. Springer, 2019, pp. 73–81.

[39] H. Hoffmann, "Jouleguard: Energy guarantees for approximate applications," in *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015, pp. 198–214.

[40] N. Mishra, C. Imes, J. D. Lafferty, and H. Hoffmann, "Caloree: Learning control for predictable latency and low energy," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 184–198, 2018.

[41] M. Li, E. Yumer, and D. Ramanan, "Budgeted training: Rethinking deep neural network training under resource constraints," *arXiv preprint arXiv:1905.04753*, 2019.

[42] G. Lample and D. S. Chaplot, "Playing fps games with deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[43] S. Bateni and C. Liu, "{NeuOS}: A {Latency-Predictable}{Multi-Dimensional} optimization framework for {DNN-driven} autonomous

systems," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 371–385.
[44] S. Bateni, H. Zhou, Y. Zhu, and C. Liu, "Predjoule: A timing-predictable energy optimization framework for deep neural networks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 107–118.
[45] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 287–296.
[46] B. Kisačanin, "Deep learning for autonomous vehicles," in *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, 2017, pp. 142–142.
[47] A. Popov, P. Gebhardt, K. Chen, R. Oldja, H. Lee, S. Murray, R. Bhargava, and N. Smolyanskiy, "Nvradarnet: Real-time radar obstacle and free space detection for autonomous driving," *arXiv preprint arXiv:2209.14499*, 2022.
[48] NVIDIA, "Duckiebot (db-j)," https://get.duckietown.com/products/duckiebot-db21, 2022.
[49] ——, "Sparkfun jetbot ai kit," https://www.sparkfun.com/products/18486, 2022.
[50] ——, "Waveshare jetbot ai kit," https://www.amazon.com/Waveshare-JetBot-AI-Kit-Accessories/dp/B07V8JL4TF/, 2022.
[51] A. A. Süzen, B. Duman, and B. Şen, "Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. IEEE, 2020, pp. 1–5.
[52] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
[53] NVIDIA, "tegrastats utility," https://docs.nvidia.com/drive/drive_os_5.1.6.1L/nvvib_docs/index.html#page/DRIVE_OS_Linux_SDK_Development_Guide/Utilities/util_tegrastats.html, 2022.
[54] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2021.
[55] S. He, Y. Wang, S. Han, S. Zou, and F. Miao, "A robust and constrained multi-agent reinforcement learning framework for electric vehicle amod systems," *arXiv preprint arXiv:2209.08230*, 2022.
[56] ——, "A robust and constrained multi-agent reinforcement learning electric vehicle rebalancing method in amod systems," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 5637–5644.
[57] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations," in *Proceedings of Robotics: Science and Systems (RSS)*, 2018.
[58] H. Ma, D. Zeng, and Y. Liu, "Learning optimal group-structured individualized treatment rules with many treatments," *Journal of Machine Learning Research*, vol. 24, no. 102, pp. 1–48, 2023.
[59] ——, "Learning individualized treatment rules with many treatments: A supervised clustering approach using adaptive fusion," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 956–15 969, 2022.
[60] G. Lan, D.-J. Han, A. Hashemi, V. Aggarwal, and C. G. Brinton, "Asynchronous federated reinforcement learning with policy gradient updates: Algorithm design and convergence analysis," *arXiv preprint arXiv:2404.08003*, 2024.
[61] G. Lan, H. Wang, J. Anderson, C. Brinton, and V. Aggarwal, "Improved communication efficiency in federated natural policy gradient via admm-based gradient updates," *Advances in Neural Information Processing Systems*, vol. 36, 2024.
[62] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.
[63] M. Laskin, L. Wang, J. Oh, E. Parisotto, S. Spencer, R. Steigerwald, D. Strouse, S. Hansen, A. Filos, E. Brooks *et al.*, "In-context reinforcement learning with algorithm distillation," *arXiv preprint arXiv:2210.14215*, 2022.
[64] S. Kapturowski, V. Campos, R. Jiang, N. Rakićević, H. van Hasselt, C. Blundell, and A. P. Badia, "Human-level atari 200x faster," *arXiv preprint arXiv:2209.07550*, 2022.
[65] S. Krishnan, S. Chitlangia, M. Lam, Z. Wan, A. Faust, and V. J. Reddi, "Quantized reinforcement learning (quarl)," *arXiv preprint arXiv:1910.01055*, 2019.
[66] S. Krishnan, M. Lam, S. Chitlangia, Z. Wan, G. Barth-Maron, A. Faust, and V. J. Reddi, "Quarl: Quantization for fast and environmentally sustainable reinforcement learning," *arXiv preprint arXiv:1910.01055*, 2019.
[67] X. Wang, Y. Gu, Y. Cheng, A. Liu, and C. P. Chen, "Approximate policy-based accelerated deep reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 6, pp. 1820–1830, 2019.
[68] J. Henkel, H. Li, A. Raghunathan, M. B. Tahoori, S. Venkataramani, X. Yang, and G. Zervakis, "Approximate computing and the efficient machine learning expedition," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.
[69] J. Yang, S. Hong, and J.-Y. Kim, "Fixar: A fixed-point deep reinforcement learning platform with quantization-aware training and adaptive parallelism," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 259–264.