Enabling Energy-Efficient Simultaneous Multi-Task Reinforcement Learning through Spiking Neural Networks with Active Dendrites for Bio-inspired Generalist Agents

Anonymous authors
Paper under double-blind review

Abstract

Reinforcement learning (RL) has demonstrated remarkable capabilities in training agents to solve complex tasks autonomously, such as mobile robots, UAVs/UGVs, and game-playing agents). However, scaling RL to master multiple tasks simultaneously (i.e., so-called *multi-task RL*) remains a significant challenge. Such a multi-task RL capability especially is important for agents to adapt to changes in real-world operational environments. State-of-the-art works show that, training agents with neural networks and shared structures across tasks promises improved generalization in simultaneous multi-task RL. However, they still suffer from *task interference* and incur high energy consumption due to intensive computation. To address this, we propose ***MTSpark***, a novel methodology that enables energy-efficient simultaneous multi-task RL using spiking neural networks (SNNs) equipped with active dendrites for bio-inspired generalist agents. Specifically, MTSpark enhances a Deep Spiking Q-Network (DSQN) with active dendrites, a dueling structure, and task-specific context signals to dynamically form specialized sub-networks for individual tasks, while exploiting sparse operations for energy-efficient network processing. Experimental results demonstrate that MTSpark achieves higher performance and efficiency compared to state-of-the-art by obtaining high scores across three Atari games (i.e., Pong: -5.4, Breakout: 0.6, and Enduro: 371.2), approaching human-level performance (i.e., Pong: -3, Breakout: 31, Enduro: 368), while incurring similar memory and about 2x lower energy than state-of-the-art. These results show that our MTSpark potentially advances the frontiers toward energy-efficient generalist agents by combining RL and SNNs.

1 Introduction

In recent years, reinforcement learning (RL) has emerged as a powerful approach for enabling agents to autonomously acquire complex behaviors. While RL methods have demonstrated exceptional success in single-task environments, such as mastering Chess, Go, and Atari games (Mnih et al., 2013; 2015; Silver et al., 2017), scaling these techniques to simultaneous multi-task settings remains an open challenge (Zhang & Yang, 2018). The ability to train an agent on multiple tasks simultaneously is essential for dynamic, real-world applications that require robust generalization and adaptation (Sodhani et al., 2021b). One of the key challenges is to manage and effectively use the increased amount of data from different tasks and extended training time (Espeholt et al., 2018; Hessel et al., 2019).

A natural approach to multi-task RL is to train agents jointly on all tasks, leveraging shared structure and paramaters across tasks to improve sample efficiency and performance, as compared to solving each task independently (Sodhani et al., 2021a; D'Eramo et al., 2020). However, this approach still suffers from *task interference* challenges, where conflicting task objectives disrupt the learning process, resulting in suboptimal representations and leading to degraded performance (Parisotto et al., 2015; Rusu et al., 2015). These limitations are shown by labels-① and ② in Figure 1(a), where the foundation RL-based methods in both Deep Neural Network (DNN) and Spiking Neural Network (SNN) domains struggle to effectively learn multiple tasks (i.e., three Atari games: Pong, Breakout, and Enduro). Therefore, Deep

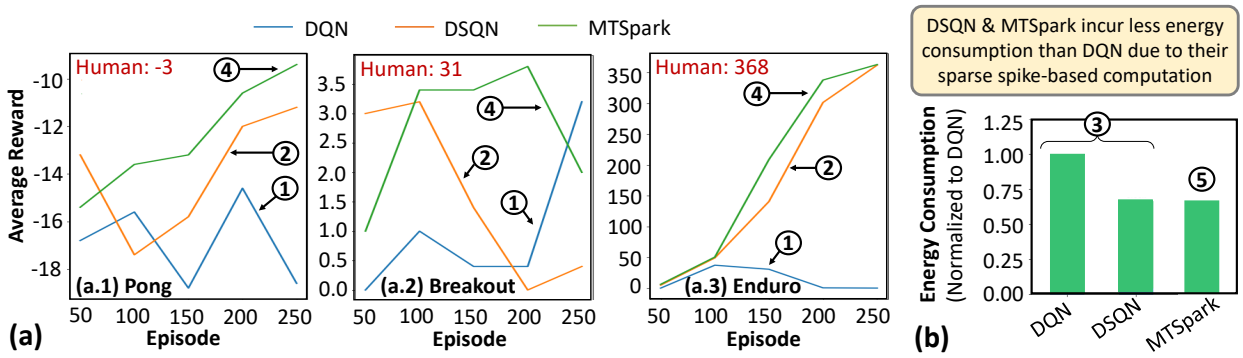

Figure 1: **(a)** Multi-task learning performance of the state-of-the-art RL-based methods for DNN (i.e., DQN) and SNN (i.e., DSQN), and our MTSpark, across three Atari games (i.e., Pong, Breakout, and Enduro). Our MTSpark achieves high performance in all tasks, while the state-of-the-art only perform well in certain tasks. **(b)** Estimated energy consumption of different RL-based methods: DQN, DSQN, and our MTSpark.

Q-Network (DQN) (Mnih et al., 2013) and Deep Spiking Q-Network (DSQN) (Chen et al., 2022) can only perform well in certain games. Furthermore, this approach also suffers from *high energy consumption* due to its intensive computation, as shown in Figure 1(b). These limitations severely impact the practicality of RL-based agents in real-world applications, where energy-efficient solutions with capabilities of adaptability and generalization across tasks are critically required (e.g., autonomous robots that can adapt to diverse environments) (Sodhani et al., 2021b; Putra & Shafique, 2021; 2022; Minhas et al., 2025a). Therefore, the **targeted research problem** is *how can we develop energy-efficient RL process that can learn multiple tasks simultaneously without significant performance degradation for generalist agents?* An efficient solution to this problem is a step forward in realizing general artificial intelligence (AI).

1.1 State-of-the-Art and Their Limitations

DNN Domain: Several strategies have been proposed, each striving to balance the shared knowledge with the task-specific information. *Replay-based methods* aim to strengthen the knowledge of previously learned tasks by replaying the corresponding data (Lopez-Paz & Ranzato, 2017; Minhas et al., 2025a;b; Riemer et al.). However, they come with significant memory and storage demands to keep large amounts of previous data, thus limiting their scalability. *Architectural-based methods* employ modular architectures to separate task-specific components from shared representations. However, they also face scalability issues, since adding new components lead to uncontrolled growth in network size and compute cost (Minhas et al., 2025a). Prominent works in RL-based DNNs introduce DQN architecture (Mnih et al., 2013), which performs exceptionally well in single-task settings but struggles in multi-task settings; see ① of Figure 1(a). Specifically, DQN achieves notable performance improvement in Breakout after 200 training episodes[1], while struggling to master other games. This shows the difficulty of applying RL models to simultaneous multi-task settings. Recently, IMPALA performs multi-task learning by employing *actors* to generate actions in environments, and *a learner* to train a central model on the collected data (Espeholt et al., 2018). Meanwhile, PopArt complements IMPALA by stabilizing the reward scale changes across tasks (Hessel et al., 2019). These works (Espeholt et al., 2018; Hessel et al., 2019) have demonstrated improvements in multi-task RL but they still rely on intensive processing, which incurs high energy consumption, thereby making it unsuitable for resource-constrained autonomous agents.

SNN Domain: Bio-inspired SNNs have gained traction as promising alternative energy-efficient machine learning (ML) algorithms, since they show substantial potential in diverse applications (Pes et al., 2024; Putra & Shafique, 2020; Cordone et al., 2022; Luo et al., 2020; Bartolozzi et al., 2022; Putra & Shafique, 2026). The inherent ability of SNNs to process temporal information efficiently makes them particularly suitable for data stream processing, a key feature for enabling multi-task learning. For instance, TM-SNN

[1]An episode is a full game playthrough where the agent interacts with the environment until the game ends.

employs threshold modulated SNNs for learning multiple tasks, but these tasks are based on a small NMNIST dataset (Cachi et al., 2023). Hence, its performance has not been evaluated for more complex tasks, such as Atari games. Recently, DSQN (Chen et al., 2022), the SNN equivalent of DQN, has higher performance than DQN in both Pong and Enduro after 150 training episodes, while incurring lower energy consumption than DQN, as shown by labels-② and ③ in Figure 1. These results highlight the potentials of employing SNNs to exploit temporal information and sparse computation for improving performance and efficiency of multi-task RL as well as enabling on-device learning. However, *the potential of SNNs in on-device multi-task RL remains largely unexplored, hindering their applicability in real-world use-cases where generalization is crucial*. Moreover, many multi-task RL methods are not suitable as solutions, as they rely on separate task training and testing. Meanwhile, *the targeted problem is simultaneous multi-task RL where different tasks are learned at the same time, directly addressing multiple tasks with only one network model*.

1.2 Associated Research Challenges

Developing an RL-based SNN architecture that can support multi-task learning is non-trivial. The associated **key research challenges** include the following.

- The network should employ an effective yet simple function to quantify the reward and punishment values for training process, enabling an efficient RL mechanism.
- The network should leverage task-specific context information (e.g., task identity) to effectively guide the learning process.
- The network needs to understand the importance of different actions and their consequences after each state transition, since the agent should identify the most appropriate action to take based on the state and the task context, optimizing performance for each individual task while maintaining efficiency across different task

1.3 Our Novel Contributions

To address the research challenges, we propose **MTSpark**, *a novel methodology to enable energy-efficient simultaneous Multi-Task reinforcement learning using Spiking neural networks for realizing bio-inspired generalist agents*; see the overview in Figure 2. Moreover, it is also the first work that enables simultaneous multi-task RL through SNNs, marking a notable step toward efficient, bio-inspired generalist agents. Its key novel contributions are summarized as follows.

- We critically examine the limitations of prominent RL-based network models (i.e., DQN from Mnih et al. (2013) and DSQN from Chen et al. (2022)) in simultaneous multi-task settings through an experimental case study, thereby highlighting the need for a better solution.
- We introduce the MTSpark methodology, whose key novelty is leveraging bio-inspired spiking operations and architectural enhancements (i.e., active dendrites) for improving multi-task RL quality through task-specific context augmentation. Its innovations include the following key points.
  - **Development of spiking neuron with *active dendrites* (Section 3.1):** It aims to enhance task specialization and mitigates interference when learning multiple tasks simultaneously. Active dendrite is a component that performs modulation for input signals based on the task-specific context, thereby regulating task-specific spiking activities and minimizing task interferences, enabling the model to better distinguish the task and decide the appropriate action to take for improving the overall performance.
  - **Network architecture design (Section 3.2):** It targets to accommodate sub-networks specialization for learning individual tasks through the integration of active dendrites. We also propose two variants of active dendrite-equipped spiking architectures: *network with active dendrites* (MTSpark_AD), and *network with active dendrites & dueling structure* (MTSpark_ADD).
  - **Multi-task training strategy (Section 3.3):** It facilitates the network to effectively learn multiple tasks and minimize task interference by combining the replay mechanism, linear exploration rate, and reward-driven target function.

- We perform a comprehensive evaluation of MTSpark using multiple tasks, i.e., three Atari games (Pong, Breakout, and Enduro). In general, MTSpark achieves high scores across three Atari games, having better overall performance than state-of-the-art.

Key Results: We evaluate the MTSpark using a Python implementation considering three Atari games (i.e., Pong, Breakout, and Enduro), and run it on the Nvidia RTX A6000 multi-GPU machines. Remarkably, MTSpark achieves scores of -5.4 for Pong, 0.6 for Breakout, and 371.2 for Enduro, approaching human-level performance (i.e., -3 for Pong, 31 for Breakout, and 368 for Enduro). It is also a clear improvement over the state-of-the-art (DQN: -18.6 for Pong, 3.2 for Breakout, and 0 for Enduro; DSQN: -11.2 for Pong, 0.4 for Breakout, and 362.2 for Enduro), incurring similar memory and about 2× lower energy consumption; see labels-④ and ⑤ in Figure 1.

2 Related Works

Deep Q-Network (DQN): DQN combines Q-learning with DNNs to enable agents to learn optimal policies in high-dimensional environments (Mnih et al., 2013). In DQN, a neural network (NN) approximates the Q-value function, which predicts the expected future reward for each state-action pair, helping the agent decide the best actions to take. DQN stabilizes training by using a replay buffer, storing past experiences and updating the model based on mini-batches of data drawn from this memory, which helps avoid correlations in the data that would hinder learning. The DQN architecture consists of three convolutional (CONV) layers and two fully connected (FC) layers. These CONV layers extract useful features from raw input data (e.g., game frames), while the FC layers compute the Q-values for each possible action in the given state.

Task Interference Mitigation: Several methods have been proposed to mitigate task interference and improve learning efficiency across multiple tasks (Minhas et al., 2025a), as discussed below.

- *Architectural methods* add task-specific components to the network model, such as extra layers or dedicated modules for each new task, allowing the model to specialize certain parts for specific tasks while sharing common components for generalized learning. However, they face scalability issues as the number of tasks grows due to new layers/modules for each task (Lee et al., 2017; Rusu et al., 2016), which are not feasible in large-scale real-world applications.

- *Replay methods* periodically reintroduce previously learned tasks during training. They work by either generating synthetic samples or using actual past samples of old tasks, which are then replayed alongside new tasks (Riemer et al.; Lopez-Paz & Ranzato, 2017; Shin et al., 2017). However, these methods require revisiting a large number of stored samples to maintain performance. Hence, they are compute- and memory-intensive, demanding significant resources to store and process replayed samples.

- *Regularization methods* constrain the learning process by stabilizing important parameters critical to previous tasks. For instance, Elastic Weight Consolidation (EWC) calculates the significance of each weight for previous tasks and protects these weights during new task learning by adding a regularization term to the loss function, discouraging large changes to important weights (Kirkpatrick et al., 2017), hence retaining knowledge from earlier tasks while adapting to new ones.

3 The MTSpark Methodology

Our proposed MTSpark methodology addresses energy-efficient simultaneous multi-task RL through network architectural enhancements (i.e., active dendrites) and effective training strategy, as shown in Figure 2 and described in the subsequent sub-sections.

3.1 Spiking Neuron with Active Dendrites

This step aims to enhance task specialization and mitigate interference when learning multiple tasks. To achieve this, *our key novelty in this step is equipping the spiking neurons with active dendrites, which modulate*

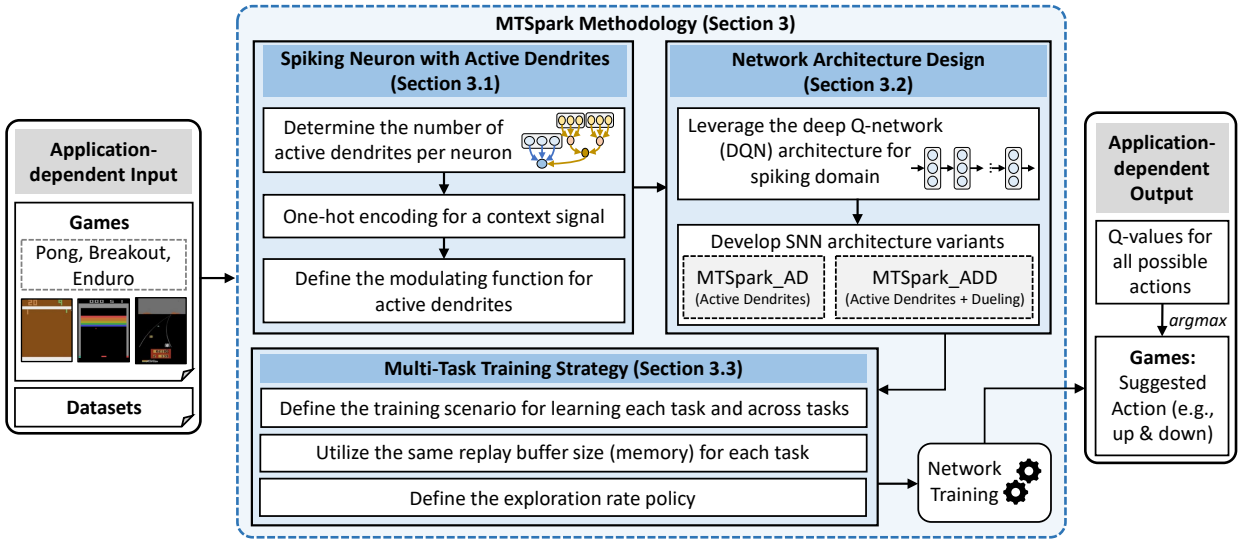

Figure 2: The overview of our MTSpark methodology with key steps highlighted in blue: spiking neuron development with active dendrites, network architecture design with two variants (i.e., MTSpark_AD and MTSpark_ADD), and multi-task training strategy.

the spiking activity of the integrate-and-fire (IF) neurons based on task-specific context signals, as shown in Figure 3. The context signal (c) is provided as input to all of the dendrites on each IF neuron. After introducing active dendrites to the IF neurons, the incoming spikes contribute to the membrane potential of the respective neuron, whose behavior can be stated as Equation 1. Here, d_j are the dendritic weights associated with the jth dendritic segment, and $f(.)$ is a modulating function. Meanwhile, $V(t)$ is the neuron's membrane potential at time t, and $s_i(t)$ is the incoming spike from the ith presynaptic neuron. If $V(t)$ crosses a predefined threshold (V_{th}), the neuron generates a spike.

$$
\begin{aligned}
V(t) &= V(t-\Delta t) + f\left(\sum_i s_i(t), \max_j(d_j^T c)\right) \\
\text{with} \quad s(t) &= \begin{cases} 1 & \text{if } V(t) \geq V_{th}, \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{1}
$$

Active Dendritic Segment: Each IF neuron can be equipped with any number of active dendrites in each dendritic segment. To ensure efficient implementation, we utilize the same number of dendrites as the number of tasks. To enable task differentiation, we represent each task (T_i) with a unique one-hot-encoded vector of context signal (c). This context signal functions as a task identifier, modulating the neuron activations in order to bias the network toward task-relevant pathways and minimize interference across tasks. We define the modulation function $f(.)$ for active dendrites as Equation 2. The sum of spikes from all presynaptic neurons are weighted by $\sigma(\max_j(d_j^T c))$.

$$
f\left(\sum_i s_i(t), \max_j(d_j^T c)\right) = \sum_i s_i(t) \cdot \sigma(\max_j(d_j^T c)) \tag{2}
$$

Task-Specific Spiking Activity: Each spiking IF neuron with active dendrites can selectively adjust its response to a context signal based on the modulation provided by maximal dendritic activation. This modulation provides the network with a form of "gating" that prevents interference between tasks and minimizes the catastrophic forgetting effects. Different tasks exhibit different dendritic activations, and as a result, they contribute differently to the membrane potential of each neuron, leading to different spiking patterns and "sub-networks" optimized for each task.

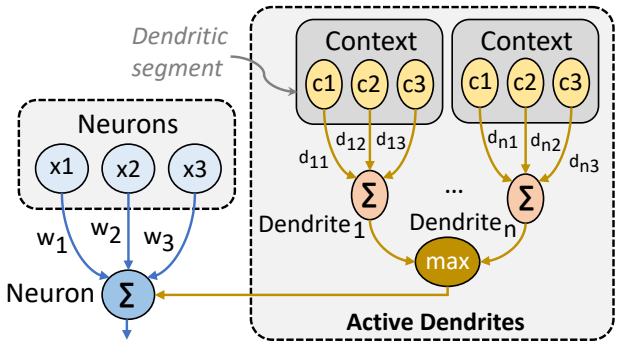

Figure 3: The proposed IF neuron model is enhanced using active dendrites with context signals.

3.2 Network Architecture Design

This design step aims to accommodate sub-network specialization when learning individual tasks. To attain this, *our key novelty in this step is leveraging the state-of-the-art DSQN architecture (Chen et al., 2022) as the foundation, then enhancing it with active dendrites to develop the MTSpark network architectures.* Here, active dendrites have the responsibility of modulating input signals to spiking neurons based on task-specific context, thereby regulating the corresponding spiking activity and establishing task-specific sub-network pathways. In this manner, the network model will be able to distinguish the task and select the appropriate sub-network pathways for generating a proper output/action.

MTSpark Network Architectures: In general, our MTSpark network architecture designs feature three CONV layers with kernel sizes of 8, 4, and 3 and strides of 4, 2, and 1, respectively; each followed by batch normalization and spiking IF neuron layer. Input observations are processed through these layers, producing feature maps that are flattened and passed to an FC layer with 512 units. Afterward, a spiking neuron layer equipped with active dendrites (AD) and context signals, followed by a final FC layer producing the Q-value over all possible actions in the environment. To provide trade-offs between performance and network size, we propose two variants of MTSpark network architectures, namely *MTSpark_AD* and *MTSpark_ADD*; whose details are described in the following.

- **MTSpark_AD** employs active dendrites at the FC layer right before the last layer, as shown in Figure 4(a) and presented in Table 1. This structure targets to provide context signals effectively without incurring significant additional resource overheads by leveraging high-level features that have been extracted by previous CONV layers.
- **MTSpark_ADD** employs the *dueling structure* on top of the MTSpark_AD to enhance the performance, thereby forming the MTSpark_ADD; see Figure 4(b) and Table 2. The dueling structure employs two distinct estimators: *the state value function* and *the action-specific advantage function.* This function separation enhances the ability to generalize learning across actions without modifying the RL algorithm (Wang et al., 2016). The state value function gives information regarding how good the current state is regardless of the action, while the advantage function gives information regarding how good the possible actions are, hence helping the agent to improve learning in states where many actions have similar advantages. Therefore, MTSpark_ADD typically provides better performance than MTSpark_AD at a higher cost (e.g., larger model size).

3.3 Multi-task Training Strategy

This step aims to facilitate the network to effectively learn multiple tasks and minimize task interference. To attain this, *our key novelty in this step is combining an efficient replay mechanism, a linear exploration rate, and a reward-driven target function.* Details of the strategy are described below.

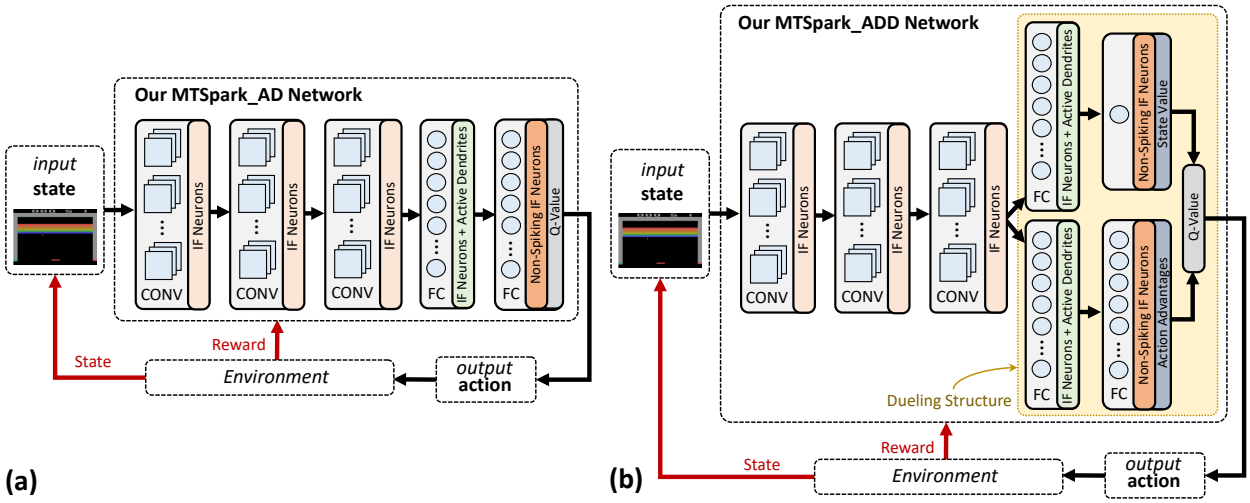

Figure 4: Our proposed MTSpark architectures: **(a)** a network with active dendrites (i.e., MTSpark_AD), and **(b)** a network with active dendrites and dueling structure (i.e., MTSpark_ADD).

Table 1: MTSpark_AD network architecture.

Layer	Input Dimension	Output Dimension
8×8-Convolution	84×84×4	20×20×32
BatchNorm	20×20×32	20×20×32
IF Neuron	20×20×32	20×20×32
4×4-Convolution	20×20×32	9×9×64
BatchNorm	9×9×64	9×9×64
IF Neuron	9×9×64	9×9×64
3×3-Convolution	9×9×64	7×7×64
BatchNorm	7×7×64	7×7×64
IF Neuron	7×7×64	7×7×64
— *Flatten* —		
FC	3136×1	512×1
IF Neuron + AD	512×1	512×1
FC	512×1	18×1
Non-Spiking IF Neuron	18×1	18×1

Table 2: MTSpark_ADD network architecture.

Layer	Input Dimension	Output Dimension
8×8-Convolution	84×84×4	20×20×32
BatchNorm	20×20×32	20×20×32
IF Neuron	20×20×32	20×20×32
4×4-Convolution	20×20×32	9×9×64
BatchNorm	9×9×64	9×9×64
IF Neuron	9×9×64	9×9×64
3×3-Convolution	9×9×64	7×7×64
BatchNorm	7×7×64	7×7×64
IF Neuron	7×7×64	7×7×64
— *Flatten* —		
FC	3136×1	512×1
IF Neuron + AD	512×1	512×1
— *Value* —		
FC	512×1	1×1
Non-Spiking Neuron	1×1	1×1
— *Advantage* —		
FC	512×1	18×1
Non-Spiking Neuron	512×1	18×1

- **Training Scenario and Replay Buffer:** We train an MTSpark network (i.e., either MTSpark_AD or MTSpark_ADD) on each environment (E) for P episodes before switching to the next environment in a round-robin scheduling to enforce the simultaneous multi-task learning scenario. To facilitate this, we maintain a fixed-size replay buffer (N) in MTSpark to efficiently store transitions for each environment during the training process.

- **Exploration Rate (ϵ):** In line with the widely-used RL training settings (Mnih et al., 2013; 2015), we linearly anneal the exploration rate ϵ from an initially high rate to a low rate over the first T frames, and keep it fixed at the low rate thereafter. This allows the agent to freely explore all possible actions in earlier frames and gradually steer the agent toward more effective actions.

- **Target Function:** We define our target (y_n) as Equation 3. Here, r_n denotes the reward received after taking an action in the state ϕ_n, while ϕ_{n+1} denotes the state seen immediately after taking the action, Q denotes our network, and Q^* denotes the target network.

$$y_n = \begin{cases} r_n & \text{if } \phi_{n+1} \text{ is terminal} \\ r_n + k & \text{if otherwise} \end{cases} \\ \text{with } \; k = \gamma Q^*(\phi_{n+1}, \arg\max_a Q(\phi_{n+1}, a)) \tag{3}$$

To systematically perform the above techniques, we propose a novel training strategy in Algorithm 1.

Algorithm 1 Proposed Training Strategy

Initialization:
  Network Q and target network Q^* with random weights;
  List of environments E;
  Environment index $i \leftarrow 0$;
  Replay memory D to capacity N for each environment;
  Episode counter episode_count $\leftarrow 0$;
  State ϕ;
Process:
for timestep $t = 1$ to $T_{\max}$ **do**
  With probability ϵ, select a random action a;
    otherwise, $a = \arg\max_a Q(\phi, a)$;
  Perform action a, observe reward r and next state ϕ';
  Store transition (ϕ, a, r, ϕ') in memory D_i;
  for environment j in E **do**
    Sample transitions $(\phi_n, a_n, r_n, \phi_{n+1})$ from D_j;
    Set target y_n as defined in Equation 3;
    Perform gradient descent on $(y_n - Q(\phi_n, a_n))^2$;
  end for
  if ϕ' is terminal **then**
    Reset state: $\phi \leftarrow$ initial state;
    episode_count $\leftarrow$ episode_count $+ 1$;
  else
    Update state: $\phi \leftarrow \phi'$;
  end if
  if episode_count $= 25$ **then**
    $i \leftarrow (i + 1) \mod$ number_of_environments;
    episode_count $\leftarrow 0$;
  end if
  if $t \mod$ target_update $= 0$ **then**
    Update target network: $Q^* \leftarrow Q$;
  end if
end for

4 Evaluation Methodology

4.1 Experimental Setup

We evaluate the MTSpark methodology using Python implementation, and then run it on the Nvidia RTX A6000 48GB multi-GPU machine. The experimental setup choices (e.g., games/environments and network hyper-parameters) are selected based on the typical settings used in the literature for training an agent with Atari games under RL scenarios Mnih et al., 2013; 2015; Chen et al., 2022, as described below.

Tasks/Environments: We consider three Atari games (i.e., Pong, Breakout, and Enduro) as the targeted tasks (environments), and utilize an action space of 18 possible actions, maintaining consistency in the action

space across all environments. We focus on a subset of representative Atari games to allow deeper and more controlled evaluation of our methodology, particularly regarding task interference and training dynamics, while providing a range of task diversity and difficulty. Pong expects a quick response for hitting a ball past the opponent through up/down movements; hence, it has relatively short action-to-reward correlation. Breakout expects the agent to destroy all bricks with a ball; hence, it typically has sparse rewards and relatively long action-to-reward correlation. Meanwhile, Enduro expects the agent to find a path to overtake other cars while avoiding collisions through right/left movements; hence, it has relatively short action-to-reward correlation. Furthermore, we also employ 210×160×3 RGB images as inputs, identical to those presented to human players.

Comparison Partners: We consider the prominent DQN (Mnih et al., 2013) and the state-of-the-art DSQN (Chen et al., 2022) as comparison partners. For completeness of the study, we also implement their dueling variants, i.e., DQN with dueling structure (DQN_D) and DSQN with dueling structure (DSQN_D). For all spiking-based networks, we employ timestep $T = 4$.

Training Settings: Details of the hyperparameter settings for the training process in our experiments are summarized in Table 3. All models are trained with the Adam optimizer, a static learning rate of 1e-4, and a batch size of 64 over 4 million frames in each environment. Following best practices, we apply a discount factor γ of 0.99 to stabilize learning, use a replay buffer of size 2^{20} for each environment to store transitions for experience replay, and decay exploration from ϵ=1.00 to ϵ=0.10 over 1 million total frames. We collect information on 25 episodes during training in each environment before switching to the next one.

Table 3: Hyperparameter settings for the experiments.

Variable	**Value**
Replay Buffer Size (N)	2^{20}
Batch Size	64
Epsilon Start	1.00
Epsilon End	0.10
Decay Epsilon Over	1,000,000 frames
Update Target Network	Every 10,000 frames
Learning Rate	1e-4
Switch Environments	Every 25 episodes

4.2 Energy Consumption Model

To estimate the energy consumption of NN processing, we follow the analytical model proposed in the recent paper from Lemaire et al. (2022). The total energy consumption (E_{Total}) is defined as the sum of energy consumption for memory accesses (E_{Mem}) and computation (E_{Com}), as expressed in Equation 4.

$$E_{Total} = E_{Mem} + E_{Com} \tag{4}$$

Memory Access Energy: E_{Mem} in Equation 4 can be estimated as the sum of memory read energy (E_{MemRd}) and memory write energy (E_{MemWr}); see Equation 5. For DNNs, E_{MemRd} is obtained by multiplying the total number of memory read for input (Rd_{inp}^{DNN}) and weight parameters (Rd_{par}^{DNN}) with the RAM energy-per-read ($E_{RAM_{Rd}}$), while E_{MemWr} is obtained by multiplying the total number of memory write for output (Wr_{out}^{DNN}) with the RAM energy-per-write ($E_{RAM_{Wr}}$); see Equation 6. For SNNs, E_{MemRd} is obtained by multiplying the total number of memory read for input (Rd_{inp}^{SNN}), weight parameters (Rd_{par}^{SNN}), and membrane potentials (Rd_{pot}) with the $E_{RAM_{Rd}}$, while E_{MemWr} is obtained by multiplying the total number of memory write for output (Wr_{out}^{SNN}) and membrane potentials (Wr_{pot}) with the $E_{RAM_{Wr}}$; see Equation 6. For simplicity, we consider that $E_{RAM_{Rd}} = E_{RAM_{Wr}}$.

$$E_{Mem} = E_{MemRd} + E_{MemWr} \tag{5}$$

$$\begin{aligned} E_{Mem}^{DNN} &= (Rd_{inp}^{DNN} + Rd_{par}^{DNN}) \cdot E_{RAM_{Rd}} + Wr_{out}^{DNN} \cdot E_{RAM_{Wr}} \\ E_{Mem}^{SNN} &= (Rd_{inp}^{SNN} + Rd_{par}^{SNN} + Rd_{pot}) \cdot E_{RAM_{Rd}} + (Wr_{out}^{DNN} + Wr_{pot}) \cdot E_{RAM_{Wr}} \end{aligned} \tag{6}$$

To further define the above-mentioned variables, several related operations are considered, including (1) read operations for inputs in DNNs and SNNs, (2) read operations for weight parameters in DNNs and SNNs, (3) read and write operations for membrane parameters in SNNs, and (4) write operations for outputs in DNNs and SNNs; see further details in Lemaire et al. (2022).

Computation Energy: E_{Com} from Equation 4 can be estimated as the sum of multiplication-and-accumulation (MAC) energy (E_{MAC}) and bias accumulation energy (E_{ACC}). For both DNNs and SNNs, E_{MAC} is obtained by multiplying the energy of addition (E_{add}) and multiplication (E_{mul}) with the number of MAC (MAC_{Com}), while E_{ACC} is obtained by multiplying the E_{add} with the number of accumulation (ACC_{Com}); see Equation 7.

$$\begin{aligned} E_{Com} &= E_{MAC} + E_{ACC} \\ &= (E_{add} + E_{mul}) \cdot MAC_{Com} + E_{add} \cdot ACC_{Com} \end{aligned} \tag{7}$$

Computation cost encompasses two parts: *elementary operations (ops)*, such as MAC and accumulation; as well as *addressing (addr)* to identify which data to process. Therefore, MAC_{Com} and ACC_{Com} in Equation 7 can be extracted further into Equation 8. Here, MAC_{ops} and ACC_{ops} denote the number of operations for MAC and bias accumulation, respectively. Meanwhile, MAC_{addr} and ACC_{addr} denote the addressing counts during MAC and bias accumulation, respectively.

$$MAC_{Com} = MAC_{ops} + MAC_{addr} \quad \text{and} \quad ACC_{Com} = ACC_{ops} + ACC_{addr} \tag{8}$$

To further define the above-mentioned variables, we consider layer-wise computation, comprising components from CONV and FC layers of DNNs and SNNs; see further details in Lemaire et al. (2022).

Operation-wise Energy Cost: To calculate the energy consumption, we leverage the energy consumption for each operation (i.e., an addition, a multiplication, and a memory access) shown in Table 4 under 45nm CMOS technology; based on studies in Jouppi et al. (2021).

Table 4: Energy consumption values for operations considered in the experiments.

Operation [32-bit]	**Energy Consumption** $[pJ]$
Addition	0.1
Multiplication	3.1
SRAM Access:	
8 kB	10
32 kB	20
1 MB	100

5 Results and Discussion

5.1 Performance on Task Functionality

Figure 5(a) and Table 5 summarize the performance of various network models across three Atari games: Pong, Enduro, and Breakout. The results highlight distinct performance patterns between models with and without spiking components, as well as the competitive advantage of our MTSpark models. Specifically, network models without spiking components (such as DQN and DQN_D) exhibit notable performance degradation after certain points during training. For instance, their performance drops significantly after point-❶ in Pong and point-❷ in Enduro. These declines reflect the challenges in managing task interference and encoding long-term action dependencies. In contrast, network models with spiking components (such as DSQN, DSQN_D, and our MTSpark variants) demonstrate more stable performance, as spiking-based operations leverage temporal dynamics for encoding sequential dependencies, thereby mitigating interference between tasks more effectively than non-spiking models. Among these, our MTSpark_AD and MTSpark_ADD

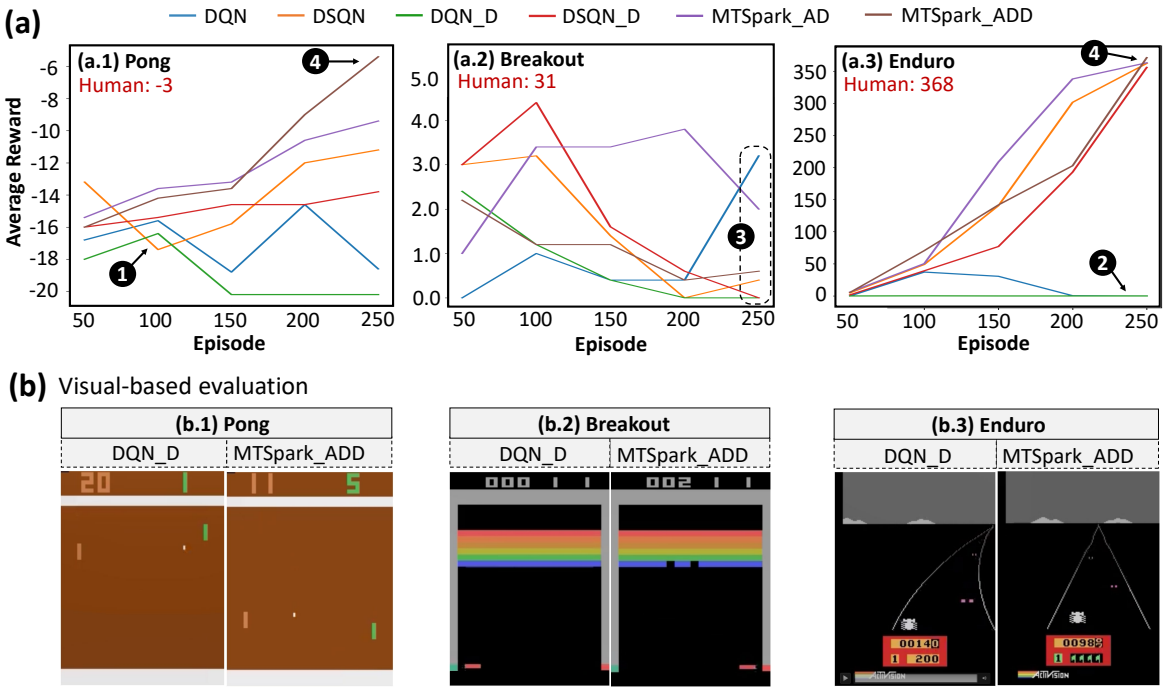

Figure 5: **(a)** Performance of different models (i.e., DQN, DSQN, DQN_D, DSQN_D, MTSpark_AD, and MTSpark_ADD) when trained for 250 episodes across three environments/games. **(b)** Visual-based evaluation of the performance between DQN_D and MTSpark_ADD across three Atari games. Game scores show that our MTSpark_ADD outperforms DQN_D in all games.

Table 5: Performance scores of different network models across three environments (i.e., Pong, Breakout, and Enduro) that are obtained from our experiments under the same simultaneous multi-task RL settings. Bold text indicates the highest achieved score among all trained models. Human scores are identical to those reported in Mnih et al. (2013).

Environment	DQN	DSQN	DQN_D	DSQN_D	MTSpark_AD	MTSpark_ADD	Human
Pong	-18.6	-11.2	-20.2	-13.8	-9.4	**-5.4**	-3
Breakout	**3.2**	0.4	0	0	2	0.6	31
Enduro	0	362.2	0	356	363.2	**371.2**	368

stand out, achieving promising results across all tasks as compared to the comparison partners, and some visual-based evaluations are also presented in Figure 5(b). Following are the key observations.

- In Pong, our MTSpark_ADD achieves the highest score among all network models with a score of -5.4, approaching human-level performance (with a score of -3). Meanwhile, our MTSpark_AD also performs strongly with a score of -9.4. In comparison, other spiking models, such as DSQN and DSQN_D, achieve lower scores, ranging from -11.2 to -20.2, showing their relative struggle to match the adaptive learning efficiency of MTSpark.
- In Enduro, MTSpark_ADD achieves a score of 371.2, slightly surpassing human performance (with a score of 368). MTSpark_AD follows closely with a score of 363.2. Other spiking models, such as DSQN, show competitive but inferior results, while non-spiking models fall short. This highlights the ability of MTSpark's active dendrite mechanism and dueling structure to dynamically route task-specific information and adapt during training.
- Breakout presents a greater challenge for all models, as shown by ❸. None of the models achieves scores near human-level performance (with a score of 31). DQN achieves the highest score (with a score of

3.2), followed by MTSpark_AD (with a score of 2). The challenges posed by Breakout likely stem from its sparse rewards and the need for effective long-term action planning, which may not be adequately addressed by the current architectures.

The superior performance of our MTSpark models in Pong and Enduro (see ❹) can be attributed to their architectural innovations, such as active dendrites and the dueling structure. These features allow MTSpark to effectively mitigate task interference by dynamically identifying and routing task-specific sub-networks during the learning process. In general, spiking components enhance temporal representation and task-specific adaptation, and the additional enhancements in our MTSpark (i.e., active dendrites and/or dueling structure) give it an edge over other spiking-based network models.

5.2 Network Model Sizes

Experimental results in Table 6 show that nearly identical parameter counts are observed among DQN, DSQN and MTSpark_AD models, as well as among DQN_D, DSQN_D and MTSpark_ADD models. These results indicate that equipping the IF neurons with active dendrites does not incur significant parameter overhead. Moreover, these results even suggest that the improved performance achieved by MTSpark is not derived merely from an increase in model complexity. Unlike conventional multi-task RL methods that often rely on separate, task-specific network modules or require extensive off-line data storage, MTSpark utilizes a unified parameter set across all tasks, relying only on the context signals to differentiate tasks and exhibit varying network behavior. MTSpark models also leverage context signals and dendritic modulation to adapt to multiple tasks simultaneously, unlike the approaches that freeze certain parts of the network and/or add task-specific modules or extensive replay. These highlight that MTSpark models handle multiple tasks without significantly expanding model sizes.

5.3 Energy Consumption

Energy consumption results are shown in Figure 6. In general, spiking networks incur lower energy consumption than their conventional counterparts, mainly due to their less memory accesses and sparse spike-based computation. For instance, DSQN consumes 34% less energy than DQN, and DSQN_D consumes 47% less energy than DQN_D. We also observe that, MTSpark_AD has similar energy consumption to DSQN, as they have similar memory access energy and computation, hence leading to about 34% energy saving from DQN, 66% energy saving from DQN_D, and 39% energy saving from DSQN_D. Meanwhile, MTSpark_ADD has similar energy consumption to DSQN_D, as they also have similar memory access energy and computation requirements, thereby leading to negligible 5% energy overhead over DQN and 47% (about 2×) energy saving over DQN_D.

Table 6: Number of trainable parameters of different network models, thereby reflecting the model sizes.

Model	# Trainable Parameters
DQN	1,693,682
DSQN	1,693,682
DQN_D	3,300,339
DSQN_D	3,300,339
MTSpark_AD	1,693,691
MTSpark_ADD	3,300,357

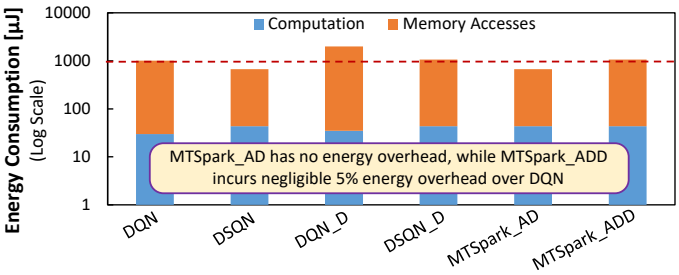

Figure 6: Energy consumption of different network models: DQN, DSQN, DQN_D, DSQN_D, MTSpark_AD, and MTSpark_ADD.

5.4 Ablation Study

5.4.1 Impact of Spiking Neurons

To isolate the effect of spiking neurons, we compare DQN vs. DSQN and DQN_D vs. DSQN_D. From experimental results in Table 5, we observe that *spiking-based* DSQN achieves overall better performance than *non-spiking-based* DQN in Pong and Enduro, with no memory overhead and lower energy consumption. Similar trends are also observed for DSQN_D and DQN_D, as *spiking-based* DSQN_D also achieves overall better performance than *non-spiking-based* DQN_D in Pong and Enduro, with no memory overhead and lower energy. In Pong, DQN scores -18.6 and DSQN scores -11.2, while DQN_D scores -20.2 and DSQN_D scores -13.8. In Enduro, DQN scores 0 and DSQN scores 362.2, while DQN_D scores 0 and DSQN_D scores 356. In Breakout, the benefits are less clear as DSQN (0.4) performs slightly worse than DQN (3.2), and DSQN_D (0) is on par with DQN_D (0), suggesting that its reward mechanism poses challenges for both conventional and spiking networks. Performance improvements achieved by spiking networks are attributed to their capabilities to learn long-term dependence through temporal information across frames. Meanwhile, energy-efficiency improvements achieved by spiking networks come from their reduced memory accesses and sparse spike-based operations.

5.4.2 Impact of Dueling Structure

To evaluate the impact of dueling structure, we compare DQN vs. DQN_D, DSQN vs. DSQN_D, and MTSpark_AD vs. MTSpark_ADD based on the experimental results in Table 5. In Pong, DQN_D (-20.2) performs slightly worse than DQN (-18.6), and similarly, DSQN_D (-13.8) performs slightly worse than DSQN (-11.2). However, when active dendrites are included, the dueling structure outperforms the non-dueling one (MTSpark_ADD: -5.4 vs. MTSpark_AD: -9.4), suggesting that the benefits of dueling architecture may emerge when paired with more sophisticated neuronal dynamics (e.g., active dendrites). In Enduro, results show mixed effects. DSQN (362.2) and DSQN_D (356) have comparable performance, while MTSpark_ADD (371.2) slightly outperforms MTSpark_AD (363.2), implying that the dueling structure can yield some gains when used in conjunction with both spiking neurons and active dendrites. In Breakout, the dueling structure results in plateaus or decreased performance (DQN: 3.2 vs. DQN_D: 0, DSQN: 0.4 vs. DSQN_D: 0, and MTSpark_AD: 2 vs. MTSpark_ADD: 0.6), implying that the dueling structure still struggles to contribute positively in Breakout. These results indicate that relying solely on the dueling structure is insufficient to master all tasks, as it cannot effectively distinguish different tasks. However, the addition of spiking neurons and active dendrites on top of the dueling structure helps the model to better perform task identification, thereby improving the overall performance. Furthermore, the dueling structure increases model size and energy consumption by about 2× as compared to non-dueling model counterparts.

5.4.3 Impact of Active Dendrites

To evaluate the impact of active dendrites, we compare DSQN vs. MTSpark_AD and DSQN_D vs. MTSpark_ADD based on the experimental results in Table 5. In general, MTSpark_AD improves upon DSQN and MTSpark_ADD improves upon DSQN_D. In Pong, MTSpark_AD scores -9.4 and DSQN scores -11.2, while MTSpark_ADD scores -5.4 and DSQN_D scores -13.8. In Breakout, MTSpark_AD scores 2 and DSQN scores 0.4, while MTSpark_ADD scores 0.6 and DSQN_D scores 0. In Enduro, MTSpark_AD scores 363.2 and DSQN scores 362.2, while MTSpark_ADD scores 371.2 and DSQN_D scores 356. These results suggest a consistent positive impact of active dendrites in defining effective sub-network pathways for targeted tasks, thereby leading to better task-specific responses and better long-term planning for learning tasks with complex input or delayed rewards.

5.5 Applicability of MTSpark for Different Multi-Task Applications

To evaluate the versatility of MTSpark for different multi-task applications, we assess its performance on image classification tasks based on different datasets. In multi-task image classification, one task is represented with a specific dataset. Here, we consider MNIST (Deng, 2012), Fashion MNIST (Xiao et al., 2017), CIFAR-10 (Krizhevsky et al., 2014), and Imagenette (10-class subset of ImageNet) datasets since they have

been widely used for evaluating SNNs (Diehl & Cook, 2015; Putra & Shafique, 2020; Putra & Shafique, 2023), and they have the same number of output space (i.e., 10 classes).

- MNIST represents a simple dataset, which has 60,000 training images and 10,000 testing images, each having a dimension of 28x28 pixels and 1 channel (grayscale).
- Fashion MNIST also represents a simple dataset, that has 60,000 training images and 10,000 testing images, each having a dimension of 28x28 pixels and 1 channel (grayscale).
- CIFAR-10 represents a medium complexity dataset, that has 50,000 training images and 10,000 testing images, each having 32×32 pixels and 3 channels (RGB).
- Imagenette represents a more complex dataset with 10-class subset of ImageNet, that has 9,469 training images and 3,925 testing images, each having 160×160 pixels and 3 channels (RGB).

Network Architectures: We compare MTSpark against: (1) a baseline DNN, and (2) an EWC-based DNN, since EWC is a prominent and widely-used technique for mitigating task interference (Minhas et al., 2025a). The baseline DNN architecture is compact and tailored for standard image classification tasks. It comprises three CONV layers with 32, 64, and 64 filters, using filter sizes of 8×8, 4×4, and 3×3, respectively, and followed by batch normalization and ReLU activation. The extracted features are then flattened and passed through an FC layer with 128 units before mapping to the 10 output classes. MTSpark adopts the same architecture but replaces the ReLU activations with IF neurons equipped with active dendrites.

Comparison with the Baseline DNN: Figure 7 shows the experimental results after performing simultaneous multi-task learning, i.e., the network is trained with different tasks/datasets at once. For MNIST, the baseline DNN achieves only modest performance, plateauing at around 57% accuracy after 50 epochs; see the blue line of ❺. In contrast, MTSpark demonstrates a steep learning curve, surpassing 92% accuracy by epoch 10 and achieving approximately 98% by the end of training; see the blue line of ❻. A similar trend is observed for Fashion MNIST. The baseline DNN performs poorly, reaching only 29% accuracy by the final epoch; see the orange line of ❺. Meanwhile, MTSpark rapidly achieves 84% accuracy within the first 10 epochs and concludes training with an accuracy of 88%; see the orange line of ❻. Performance on CIFAR-10, which is relatively more complicated than both MNIST and Fashion MNIST, also follows similar trends. While the baseline DNN struggles, achieving a maximum accuracy of only 32%, MTSpark begins with an initial accuracy of 37% and steadily improves to approximately 57% by the end of training; see green line of ❻. These results underscore MTSpark's superiority across diverse image classification tasks, as its ability to leverage active dendrites for task-specific context processing enables faster, more effective learning than the conventional DNN.

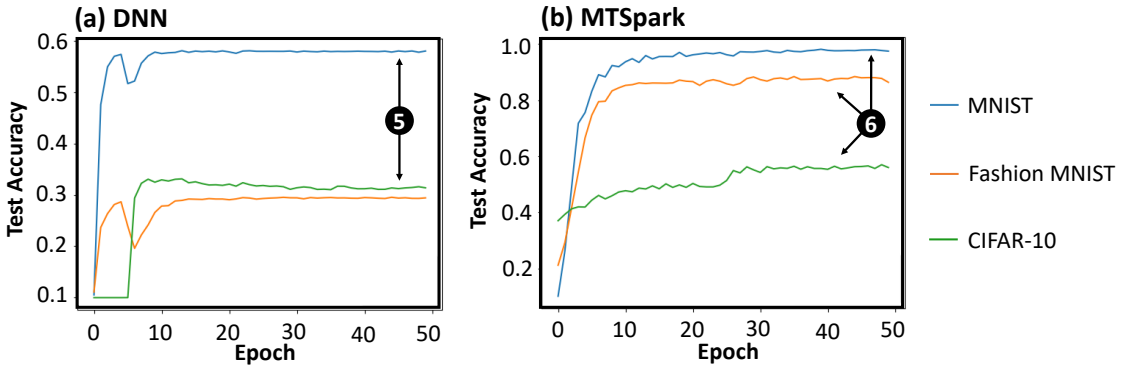

Figure 7: Test accuracy of different network models: **(a)** DNN and **(b)** our MTSpark, on three different image classification tasks: MNIST, Fashion MNIST, and CIFAR-10.

Comparison with the EWC-based DNN: To benchmark our MTSpark against the EWC-based DNN, we subsequently train the same baseline DNN on MNIST, Fashion MNIST, and CIFAR-10. EWC is applied

to the DNN with four different regularization weights (i.e., 1e2, 1e4, 1e6, and 1e8) to assess its ability to balance knowledge retention and adaptability. The experimental results are summarized in Table 7. These results show that, the EWC-based DNN struggles to deliver competitive performance across all datasets. Even at its best, accuracy remains low, with values of 33.5% for MNIST, 48.2% for Fashion MNIST, and 21.7% for CIFAR-10. This highlights the challenges posed by multi-task learning. In contrast, our MTSpark achieves significantly higher accuracies: 97.5% for MNIST, 86.4% for Fashion MNIST, and 56.0% for CIFAR-10. This performance illustrates the ability of MTSpark to learn and retain knowledge across tasks more effectively than EWC.

Table 7: Performance comparison between the EWC-based DNN and the MTSpark-based model at the end of training on image classification tasks.

Model	MNIST	Fashion MNIST	CIFAR-10
EWC_1e2	10.5%	13.8%	20.3%
EWC_1e4	12.1%	16.8%	21.7%
EWC_1e6	33.5%	39.4%	18.0%
EWC_1e8	30.0%	48.2%	12.8%
MTSpark	97.5%	86.4%	56.0%

Different Multi-Task Scenarios: We also conduct an ablation study to evaluate the impact of conventional DNN and MTSpark architectures on various multi-task learning scenarios. Specifically, we perform experimental case studies for two scenarios: (1) *two-task learning*, using the MNIST and Fashion MNIST datasets; and (2) *three-task learning*, using the MNIST, Fashion MNIST, and Imagenette (10-class subset of ImageNet) datasets. Experimental results are provided in Figure 8 and Figure 9 for two-task and three-task learning, respectively. Following are the key observations.

- *Two-task learning scenario*: The DNN achieves high accuracy on a single task (MNIST), but struggles with the second task (Fashion MNIST), as shown in Figure 8(a). This is attributed to the DNN architecture, which is optimized for learning specific tasks, and consequently, it exhibits frequent fluctuations in its learning curve when exposed to a new task. Conversely, MTSpark-based model demonstrates superior performance by achieving high accuracy on both tasks. Its learning curves are smoother and converge earlier in the training process, as shown in Figure 8(b). This performance improvement is due to MTSpark's active dendrite mechanisms, which effectively modulate spiking activity based on contextual signals, enabling efficient multi-task learning.

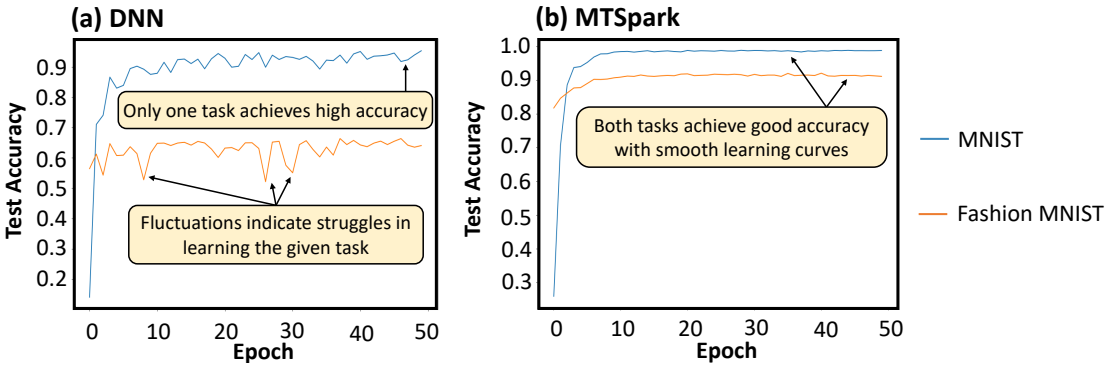

Figure 8: Test accuracy in the two-class learning scenario for **(a)** DNN and **(b)** our MTSpark, considering two tasks (i.e., MNIST and Fashion MNIST).

- *Three-task learning scenario*: In this case, we observe that the limitations of the DNN are further amplified. The DNN achieves high accuracy on MNIST, moderate accuracy on Fashion MNIST, and very low accuracy on Imagenette, as shown Figure 9(a). Adding additional tasks leads to significant accuracy

degradation on previously learned tasks and a failure to generalize to new ones, reflecting inherent limitations of the DNN in multi-task learning. In contrast, MTSpark-based model achieves high accuracy on MNIST and Fashion MNIST while delivering better accuracy on Imagenette compared to the DNN. Its learning curves remain smooth, as shown in Figure 9(b), highlighting its ability to learn multiple tasks more effectively due to its active dendrites.

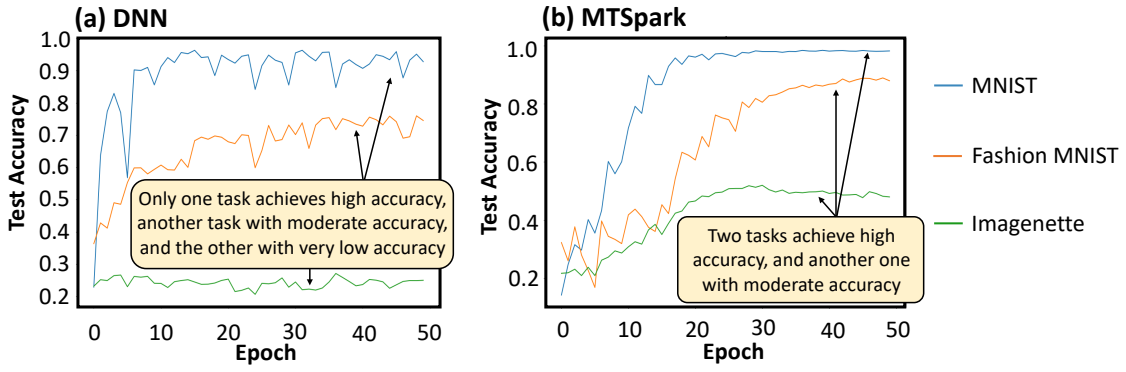

Figure 9: Test accuracy in the three-class learning scenario **(a)** DNN and **(b)** our MTSPark, considering three tasks (i.e., MNIST, Fashion MNIST, and Imagenette).

6 Conclusion

We present a novel MTSpark methodology to enables simultaneous multi-task RL using bio-inspired spiking networks. MTSpark exploits sparse spike-based operations to achieve energy-efficient computation, and leverages temporal information within spike trains to learn long-term dependencies. MTSpark leverages active dendrites, dueling structure, and task-specific context signals for enabling dynamic and context-sensitive task differentiation, thereby regulating task-specific spiking activities and minimizing task interferences. As a result, MTSpark models can better distinguish the task and decide the appropriate action to take for improving the overall performance. Experimental results demonstrate the effectiveness of MTSpark (i.e., Pong: -5.4, Breakout: 0.6, and Enduro: 371.2), surpassing state-of-the-art, while incurring similar memory footprint and 2× lower energy consumption. These results also show the potential of our MTSpark in advancing multi-task RL toward enabling energy-efficient bio-inspired generalist agents.

References

Chiara Bartolozzi, Giacomo Indiveri, and Elisa Donati. Embodied neuromorphic intelligence. *Nature communications*, 13(1):1024, 2022.

Paolo G Cachi, Sebastián Ventura Soto, and Krzysztof J Cios. Tm-snn: threshold modulated spiking neural network for multi-task learning. In *International Work-Conference on Artificial Neural Networks*, pp. 653–663. Springer, 2023.

Ding Chen, Peixi Peng, Tiejun Huang, and Yonghong Tian. Deep reinforcement learning with spiking q-learning. *arXiv preprint arXiv:2201.09754*, 2022.

Loic Cordone, Benoît Miramond, and Phillipe Thierion. Object detection with spiking neural networks on automotive event data. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, July 2022.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. doi: 10.1109/MSP.2012.2211477.

Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and J. Peters. Sharing knowledge in multi-task deep reinforcement learning. *ArXiv*, abs/2401.09561, 2020. URL `https://api.semanticscholar.org/CorpusId:209479151`.

Peter Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*, 9:99, 2015. ISSN 1662-5188. doi: 10.3389/fncom.2015.00099. URL `https://www.frontiersin.org/article/10.3389/fncom.2015.00099`.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado Van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3796–3803, 2019.

Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson. Ten lessons from three generations shaped google's tpuv4i : Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–14, 2021. doi: 10.1109/ISCA52012.2021.00010.

James Kirkpatrick et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences (PNAS)*, 114(13):3521–3526, 2017.

Alex Krizhevsky, Vinod Nair, Geoffrey Hinton, et al. The cifar-10 dataset. *online: http://www.cs.toronto.edu/kriz/cifar.html*, 55(5):2, 2014.

Jeongtae Lee et al. Lifelong learning with dynamically expandable networks. *CoRR*, abs/1708.01547, 2017. URL `http://arxiv.org/abs/1708.01547`.

Edgar Lemaire, Loïc Cordone, Andrea Castagnetti, Pierre-Emmanuel Novac, Jonathan Courtois, and Benoît Miramond. An analytical estimation of spiking neural networks energy efficiency. In *International Conference on Neural Information Processing (ICONIP)*, pp. 574–587. Springer, 2022.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in Neural Information Processing Systems (NIPS)*, 30, 2017.

Yuling Luo, Qiang Fu, Juntao Xie, Yunbai Qin, Guopei Wu, Junxiu Liu, Frank Jiang, Yi Cao, and Xuemei Ding. Eeg-based emotion classification using spiking neural networks. *IEEE Access*, 8:46007–46016, 2020. doi: 10.1109/ACCESS.2020.2978163.

Mishal Fatima Minhas, Rachmad Vidya Wicaksana Putra, Falah Awwad, Osman Hasan, and Muhammad Shafique. Continual learning with neuromorphic computing: Foundations, methods, and emerging applications. *IEEE Access*, 13:124824–124873, 2025a. doi: 10.1109/ACCESS.2025.3588665.

Mishal Fatima Minhas, Rachmad Vidya Wicaksana Putra, Falah Awwad, Osman Hasan, and Muhammad Shafique. Replay4ncl: An efficient memory replay-based methodology for neuromorphic continual learning in embedded ai systems. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7, 2025b. doi: 10.1109/DAC63849.2025.11132839.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

Lorenzo Pes, Rick Luiken, Federico Corradi, and Charlotte Frenkel. Active dendrites enable efficient continual learning in time-to-first-spike neural networks. In *IEEE 6th International Conference on AI Circuits and Systems (AICAS)*, pp. 41–45. IEEE, 2024.

Rachmad Vidya Wicaksana Putra and Muhammad Shafique. FSpiNN: An optimization framework for memory-efficient and energy-efficient spiking neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 39(11):3601–3613, 2020. doi: 10.1109/TCAD.2020.3013049.

Rachmad Vidya Wicaksana Putra and Muhammad Shafique. Spikedyn: A framework for energy-efficient spiking neural networks with continual and unsupervised learning capabilities in dynamic environments. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1057–1062. IEEE, 2021.

Rachmad Vidya Wicaksana Putra and Muhammad Shafique. lpspikecon: Enabling low-precision spiking neural network processing for efficient unsupervised continual learning on autonomous agents. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2022.

Rachmad Vidya Wicaksana Putra and Muhammad Shafique. Topspark: A timestep optimization methodology for energy-efficient spiking neural networks on autonomous mobile agents. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3561–3567, 2023. doi: 10.1109/IROS55552.2023.10342499.

Rachmad Vidya Wicaksana Putra and Muhammad Shafique. Spikenas: A fast memory-aware neural architecture search framework for spiking neural network-based embedded ai systems. *IEEE Transactions on Artificial Intelligence*, 7(2):947–959, 2026. doi: 10.1109/TAI.2025.3586238.

Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. Learning to learn without forgetting by maximizing transfer and minimizing interference. In *International Conference on Learning Representations (ICLR)*.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

Andrei A. Rusu et al. Progressive neural networks. *CoRR*, abs/1606.04671, 2016. URL `http://arxiv.org/abs/1606.04671`.

Hanul Shin et al. Continual learning with deep generative replay. *Advances in Neural Information Processing Systems (NIPS)*, 30, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, 2021a. URL `https://api.semanticscholar.org/CorpusId:231879645`.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning (ICML)*, pp. 9767–9779. PMLR, 2021b.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 1995–2003. PMLR, 2016.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Yu Zhang and Qiang Yang. An overview of multi-task learning. *National Science Review*, 5:30–43, 2018.