

To the Graduate Council:

I am submitting herewith a thesis written by Tokey Tahmid entitled
“Energy-Efficient Computing for Scalable and Sustainable AI.” I have examined the
final paper copy of this thesis for form and content and recommend that it be
accepted in partial fulfillment of the requirements for the degree of Master of
Science, with a major in Major.

Catherine Schuman, Major

Professor

We have read this thesis
and recommend its acceptance:

Catherine Schuman

Piotr Luszczek

James Plank

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

To the Graduate Council:

I am submitting herewith a thesis written by Tokey Tahmid entitled
“Energy-Efficient Computing for Scalable and Sustainable AI.” I have examined the
final electronic copy of this thesis for form and content and recommend that it be
accepted in partial fulfillment of the requirements for the degree of Master of
Science, with a major in Major.

Catherine Schuman, Major Professor

We have read this thesis
and recommend its acceptance:

Catherine Schuman

Piotr Luszczek

James Plank

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Energy-Efficient Computing for Scalable and Sustainable AI

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Tokey Tahmid

December 2024

© by Tokey Tahmid, 2024
All Rights Reserved.

Acknowledgements

I would like to express my sincere gratitude to my advisors, Dr. Catherine Schuman, Dr. Piotr Luszczyk, and Dr. Mark Gates, for their invaluable guidance, support, and expertise throughout the development of this thesis. I am also thankful to my mentor, Dr. Wesley Da Silva Pereira, for his exceptional mentoring during my Summer 2024 internship at the National Renewable Energy Laboratory (NREL), which significantly contributed to the successful completion of this research. Funds that supported this research are listed as follows:

1. This work was supported by the U.S. Department of Energy, Office of Science, ASCR under Award Number DE-SC0021419.
2. This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering.
3. This material is based upon work supported by the U.S. Department of Energy, National Renewable Energy Laboratory under contract number DE-AC36-08GO28308.

Abstract

In this modern era of AI revolution, there have been massive and rapid investments in data-driven, large-scale AI systems. However, the high-performance computing techniques that enable the computation for these rapidly growing AI systems, consume staggering amounts of energy and resources. This proliferation of AI brings new optimization challenges for sustainability without compromising scalability and performance. This thesis research aims to tackle these challenges and provide a way forward for scalable and sustainable AI. Among the energy-efficient alternatives to the traditional Von Neumann architecture, neuromorphic computing and its Spiking Neural Networks (SNNs) are a promising choice due to their inherent energy efficiency. However, in real-world application scenarios, such as complex continuous control tasks, SNNs often lack the performance optimizations of traditional artificial neural networks. To address this issue, researchers have combined SNNs with Deep Reinforcement Learning (DeepRL) algorithms to utilize their optimization techniques. Although this integration enables the completion of these complex tasks, the question of scalability still remains unexplored. Hence, this thesis presents a novel model called SpikeRL, which is a scalable and efficient framework for DeepRL-based SNNs for complex continuous control tasks. The SpikeRL framework consists of three major components. First, a DeepRL-based SNN model utilizing population encoding. Second, distributed computing across models and environments is implemented using PyTorch’s distributed package with both Message Passing Interface (MPI) and NVIDIA Collective Communications Library

(NCCL) backends. Lastly, further optimization for model training is achieved by using mixed-precision techniques for parameter updates. Comparison analyses with the state-of-the-art SNN methods demonstrated that the SpikeRL model achieves an overall performance increase of 40%, an energy efficiency of 39%, and a carbon emission reduction of 28%. The SpikeRL model was also tested on a neuromorphic hardware at TENNLab using the Reduced Instruction Spiking Processor (RISP) simulation to run the inference of the model. Although the deployment of SpikeRL on a neuromorphic hardware is still a work in progress, the research findings presented in this thesis demonstrate the scalability and energy efficiency of SpikeRL for the training of complex continuous control agents, contributing to advancements in scalable and sustainable AI.

Table of Contents

1	Introduction	1
1.1	Energy Efficiency with the SpikeRL Framework	1
1.2	Research Contribution and Structure	2
2	Related Work	4
2.1	Neuromorphic Computing	4
2.1.1	Characteristics of Neuromorphic Computing	5
2.1.2	Key Features of Neuromorphic Computers	5
2.2	Advancements in Spiking Neural Networks for Robotics	6
2.3	Integration of SNNs with Deep Reinforcement Learning	7
2.4	Contribution of SpikeRL to Energy-Efficient AI Systems	12
2.4.1	Distributed Training with MPI and NCCL	13
2.4.2	Mixed Precision	14
3	Methodology	15
3.1	SpikeRL Framework Architecure	15
3.1.1	Population Encoding and Decoding	17
3.1.2	DeepRL-SNN Integration	21
3.2	Distributed Training with MPI and NCCL	23
3.2.1	Custom MPI Implementation Using mpi4py	23
3.2.2	Final Implementation with MPI and NCCL	25
3.3	Optimization with Mixed Precision	27

3.4	Energy Measurements	29
3.4.1	Energy Measurement Tools and Their Comparison	29
3.4.2	Metrics for Measuring Energy Efficiency	30
4	Results	33
4.1	Experimental Setup	33
4.1.1	Cluster Configuration	33
4.1.2	Simulation Environment	33
4.1.3	Training Details	35
4.2	Experimental Results	36
4.2.1	Preliminary Results	36
4.2.2	Final Performance Benchmarks	36
4.2.3	Final Energy Efficiency Benchmarks Summary	42
4.2.4	Greenup And Carbon Footprint Analysis	46
4.3	Discussion and Analyses	53
5	Conclusion	54
	Bibliography	56
	Appendix	64
A	SpikeRL Code	64
Vita		64

List of Tables

4.1	Energy efficiency benchmark for OpenAI gym Ant-v4	43
4.2	Energy efficiency benchmark for OpenAI gym Hopper-v4	44
4.3	Energy efficiency benchmark for OpenAI gym HalfCheetah-v4	45

List of Figures

3.1	System architecture of the SpikeRL framework.	16
3.2	Distributed training with MPI using <i>mpi4py</i>	24
3.3	Distributed training with MPI/NCCL backend using PyTorch distributed package.	26
3.4	Automatic mixed-precision training with BFLOAT16.	28
4.1	OpenAI gym environments from the MuJoCo physics simulator. . . .	34
4.2	Distributed training performance results of SpikeRL on Walker2d-v4 environment.	37
4.3	SpikeRL performance benchmark of 100k training steps against PopSAN and DeepRL methods on Walker2d-v4.	38
4.4	Ant-v4 performance benchmark of SpikeRL over PopSAN and DSRL (mean of 5 runs with shaded area as standard deviation).	39
4.5	Hopper-v4 performance benchmark of SpikeRL over PopSAN and DSRL (mean of 5 runs with shaded area as standard deviation). . . .	40
4.6	HalfCheetah-v4 performance benchmark of SpikeRL over PopSAN and DSRL (mean of 5 runs with shaded area as standard deviation). . . .	41
4.7	Greenup, powerup, and speedup (GPS-UP) quadrant graph for SpikeRL (NCCL and MPI) over PopSAN in the Ant-v4 environment. . . .	47
4.8	Greenup, powerup, and speedup (GPS-UP) quadrant graph for SpikeRL (NCCL and MPI) over PopSAN in the Hopper-v4 environment. .	48

4.9	Greenup, powerup, and speedup (GPS-UP) quadrant graph for SpikeRL (NCCL and MPI) over PopSAN in the HalfCheetah-v4 environment.	49
4.10	Ant-v4 carbon footprint analysis of SpikeRL over PopSAN.	50
4.11	Hopper-v4 carbon footprint analysis of SpikeRL over PopSAN.	51
4.12	HalfCheetah-v4 carbon footprint analysis of SpikeRL over PopSAN. .	52

Nomenclature

AI	Artificial Intelligence - The simulation of human intelligence in machines that are programmed to think and learn like humans.
AMP	Automatic Mixed Precision - A technique that uses both half-precision (FP16) and single-precision (FP32) computations to optimize speed and energy efficiency.
ANN	Artificial Neural Network - A computing system inspired by the biological neural networks of animal brains, used in machine learning and artificial intelligence.
DDPG	Deep Deterministic Policy Gradient - An algorithm for continuous action spaces in reinforcement learning.
DeepRL	Deep Reinforcement Learning - A subset of machine learning that combines deep learning and reinforcement learning principles.
EDP	Energy-Delay Product - A metric used to evaluate the efficiency of computing systems by considering both energy consumption and computational delay.
FP16	Half Precision - A 16-bit floating-point format used to speed up computation and reduce energy consumption.
FP32	Single Precision - A 32-bit floating-point format used for standard precision in computation.

LIF	Leaky Integrate-and-Fire - A type of spiking neuron model used to simulate neuronal activity in Spiking Neural Networks.
MPI	Message Passing Interface - A standardized and portable message-passing system designed to function on parallel computing architectures.
PopSAN	Population Spiking Neural Network - A model that uses population coding techniques to enhance the data representation capabilities of Spiking Neural Networks.
PPO	Proximal Policy Optimization - A reinforcement learning algorithm that aims to improve policy optimization performance while ensuring stability.
PUE	Power Usage Effectiveness - A metric that measures how efficiently a computer data center uses energy; specifically, the ratio of total facility energy consumption to energy used by the IT equipment.
RISP	Reduced Instruction Spiking Processor - A neuromorphic hardware used for simulating Spiking Neural Networks.
SAC	Soft Actor-Critic - A reinforcement learning algorithm that maximizes entropy in addition to expected reward to ensure more exploration and robustness.
SNN	Spiking Neural Network - An artificial neural network using spiking neurons that mimics the neural structure and functioning of biological neurons.
T_e	Total Active Time - The total time during which a component or system is actively consuming energy.

TD3	Twin Delayed Deep Deterministic Policy Gradient - A variant of the DDPG algorithm that addresses some of its function approximation errors.
TDP	Thermal Design Power - The maximum amount of heat a computer chip, such as a CPU or GPU, is expected to generate under normal computing workloads.

Disclosure Statement

This thesis is based on the works presented in the following papers,

(1) Towards Scalable and Efficient Spiking Reinforcement Learning for Continuous Control Tasks.

(2) Low Precision for Lower Energy Consumption.

(3) Low Precision and Efficient Programming Languages for Sustainable AI: Final Report for the Summer Project of 2024.

This thesis builds on the original work from publication (1) and integrates content from (2) and (3), offering a comprehensive literature review, new experimental results, in-depth theoretical exploration, performance and efficiency benchmarks against state-of-the-art methods, and discussions on future research directions. As the first author of publication (1), I developed the SpikeRL framework, conducted experiments and analyses, and wrote the initial manuscript detailing the model and experimental results. Co-authors Dr. Mark Gates, Dr. Piotr Luszczek, and Dr. Catherine Schuman contributed by editing and finalizing the text, figures, tables, and overall structure. As a co-author of (2) and (3), I explored mixed-precision techniques for AI applications across various hardware and software frameworks in collaboration with Joao Vitor De Oliveira Silva and mentor Wesley da Silva Pereira. These mixed-precision techniques are also incorporated into this thesis.

Note: Publication (1) is soon to appear in IEEE proceedings.

Chapter 1

Introduction

The rapid advancements in Artificial Intelligence (AI) bring about significant energy costs that pose sustainability challenges. For instance, OpenAI's GPT-3, a language model with 175 billion parameters, was estimated to consume approximately 1287 MWh for its training and deployment phase (4). This staggering amount of energy underscores the growing concern as AI systems become more complex and widespread. Future projections are even more alarming, with estimates suggesting that the energy consumption of Nvidia's new AI servers could surpass the annual energy usage of countries like Argentina and Sweden (4). These figures highlight an urgent need for innovations in energy-efficient computing to mitigate the environmental impact of such technologies while continuing to harness their capabilities.

1.1 Energy Efficiency with the SpikeRL Framework

Spiking Neural Networks (SNNs) are non-von Neumann architecture that is inspired by the human brain and mimics the neural structure and functioning of the human brain. SNNs offer a compelling energy efficient alternative to traditional high energy consuming architectures. These networks inherently supports energy efficiency and parallelism, making them ideal for developing energy efficient AI applications (5).

Despite their promise, SNNs have historically struggled to perform at levels comparable to conventional Artificial Neural Networks (ANNs) in real world applications such as complex continuous control tasks, primarily due to their lack of advanced optimization techniques (6).

Recognizing the untapped potential of SNNs, recent research has focused on their integration with Deep Reinforcement Learning (DeepRL) algorithms (7; 8; 9; 10; 11; 12; 13; 14; 15; 16; 17). This strategic amalgamation aims to harness the biological fidelity and energy efficiency of SNNs alongside the powerful optimization capabilities of DeepRL. Such integration enhances the performance of SNNs in complex control scenarios, where adaptability and precision are paramount.

Building on these concepts, this thesis introduces a novel framework named SpikeRL. This framework is specifically designed to leverage the synergies between SNNs and DeepRL, focusing on scalability and efficiency in both training and deployment. It incorporates a DeepRL based SNN with population encoding (18), utilizes the Message Passing Interface (MPI) (19), and NVIDIA Collective Communications Library (NCCL) (20) for distributed training across multiple GPUs. Additionally, it adopts mixed-precision training techniques (21) to further optimize computational resource usage.

1.2 Research Contribution and Structure

The contribution of this thesis with the SpikeRL framework, in the domain of scalable and sustainable AI, is three-fold. I) A custom DeepRL based SNN algorithm utilizing population encoding II) Scaling up the training process using distributed training with MPI and NCCL; III) Optimizing for energy efficiency using mixed-precision training. One of the key future work entails running inference on specialized simulated neuromorphic hardware such as the RISP from TennLab UTK.

The following chapters of this thesis include: Chapter 2 with related work, Chapter 3 with the implementation of SpikeRL, Chapter 4 with experimental results and analyses, and finally Chapter 5 with discussion, future work, and conclusion.

Chapter 2

Related Work

This chapter reviews the body of literature on SNNs and their integration with DeepRL, emphasizing the enhancements in energy efficiency and performance in robotic applications. It lays the groundwork for understanding the novel contributions of the SpikeRL framework utilizing neuromorphic computing in addressing the challenges of scalability and computational efficiency in AI systems.

2.1 Neuromorphic Computing

Neuromorphic computing is emerging as a promising energy-efficient alternative to traditional Von-Neumann architecture (6). This new paradigm seeks to achieve continued performance improvements in computing by drawing inspiration from the structure and functioning of biological brains. Initially coined by Carver Mead in the late 1980s (22; 23), the term neuromorphic originally referred to mixed analogue–digital systems designed for brain-inspired computation. Over time, and with the advent of substantial research initiatives like DARPA’s Synapse project and the European Union’s Human Brain Project, the definition of neuromorphic computing has broadened to encompass a wide array of hardware implementations that mimic various aspects of neural systems (6).

2.1.1 Characteristics of Neuromorphic Computing

Neuromorphic computers are non-von Neumann systems composed of artificial neurons and synapses, where the architecture and operational principles are inspired by biological neural networks. Unlike traditional von Neumann computers, which separate processing units (CPUs) from memory units, neuromorphic systems integrate computation and memory. In these systems, neurons process information and synapses store and transmit data, blurring the line between processing and memory elements. The computation in neuromorphic systems is defined by the structure of the neural network and the parameters of these neurons and synapses, rather than explicit instruction sets. Additionally, while conventional computers represent information in binary form, neuromorphic computers use spikes (discrete events) over time to encode information (6).

2.1.2 Key Features of Neuromorphic Computers

Neuromorphic systems differ fundamentally from traditional computing architectures in several ways as detailed in (6). These systems leverage massive parallelism as each neuron and synapse can operate concurrently. Although each neuron and synapse performs relatively simple computations, the parallel nature leads to significant computational capabilities. By mimicking the brain’s structure, neuromorphic systems collocate processing and memory within the network of neurons and synapses. This integration helps mitigate the von Neumann bottleneck, reduces energy consumption, and improves data throughput by eliminating the need for extensive data movement (24). Neuromorphic architectures can scale naturally by adding more neurons and synapses, often through integrating multiple neuromorphic chips to create larger, more complex neural networks. Projects such as SpiNNaker (25; 26) and Intel’s Loihi (27) have demonstrated scalability by successfully combining numerous neuromorphic cores to model large-scale neural systems. Computations in neuromorphic systems are triggered by events (spikes)

rather than being performed continuously. This event-driven nature leads to efficient processing, as neurons and synapses consume energy only when they are active, aligning with temporally sparse activity patterns observed in biological neural systems (28; 29). Neuromorphic systems can incorporate randomness in their computations, such as in the firing behavior of neurons, which can help them be more robust and able to handle a range of computational tasks in the presence of noise.

Among these features, the extremely low power consumption of neuromorphic computers stands out as particularly attractive. SNNs in neuromorphic architectures can operate at orders of magnitude lower power than conventional computing systems, making them suitable for energy-constrained applications like edge computing. The energy efficiency of SNNs is a direct consequence of their event-driven and massively parallel architectures, where only a fraction of the system is active at any moment (6).

2.2 Advancements in Spiking Neural Networks for Robotics

Research in the field of robotics has demonstrated the utility of SNNs in various complex tasks. (30) and (31) successfully employed SNN-based Central Pattern Generators (sCPG) on the SpiNNaker platform (26) to manage locomotion tasks such as walking, trotting, or running in hexapod robots. These implementations showcase the potential of SNNs in generating rhythmic patterns without continuous sensory feedback, highlighting their efficiency and autonomy in control systems. Further extending the capabilities of SNNs, (32) and (33) demonstrated their application in biomimetic robots, such as a lamprey-like model, to dynamically control movements. For spatial awareness and navigation, projects such as Spiking

RatSLAM (34) and Gridbot (35) utilized SNNs for effective environment mapping, proving their efficiency in real-world and simulated environments (36).

2.3 Integration of SNNs with Deep Reinforcement Learning

The integration of SNNs with DeepRL has been a pivotal advancement, addressing the limitations of SNNs in terms of optimization, data representation, and learning efficiency. The following studies comprise of the state-of-the-art methods in this research area:

One of the studies (7) explore integrating SNNs into DeepRL to address challenges such as sensitivity to noisy, incomplete, or misleading input data. While DeepRL, leveraging multi-layer neural networks, achieves remarkable success in tasks requiring trained policies, its robustness remains a limitation. Inspired by biological neural systems, the approach builds on prior work in supervised learning, where standard neural networks with the Rectified Linear Unit (ReLU) activation functions were effectively converted to SNNs with minimal performance loss. Extending this concept, the method adapts Q-learning networks trained with RL to SNNs, demonstrating improved resilience to input disturbances like image occlusion. This study also successfully applies the conversion to full-scale Deep Q-networks, setting the stage for future advancements in robust DeepRL systems.

Population-coded spiking actor network (PopSAN) (8) addresses the growing need for energy-efficient control mechanisms in mobile robots, particularly as real-world tasks demand handling high-dimensional observation and action spaces within the constraints of limited on-board resources. It focuses on SNNs executed on neuromorphic processors, which offer a promising alternative to traditional controllers for low-dimensional control tasks due to their energy efficiency and robustness. However, scaling SNNs for real-world applications is challenging due to

limitations in their information representation and training algorithms. To overcome these barriers, the proposed method introduces PopSAN trained alongside a deep critic network using DeepRL. Inspired by neural population coding in the brain, this approach enhances the network’s representation capacity while combining the training benefits of deep networks with the energy-efficient inference of SNNs. The method integrates seamlessly with various policy-gradient DeepRL algorithms, supporting both on-policy and off-policy methods. Benchmarked on OpenAI Gym tasks, the PopSAN was deployed on Intel’s Loihi neuromorphic chip, achieving performance comparable to standard deep actor networks while consuming 140 times less energy per inference.

Another study (9) focuses on improving energy-efficient motion in robotics using DeepRL, addressing limitations in standard methods that penalize energy expenditure through action penalties in the reward function. While effective for its simplicity, this approach faces challenges such as excessive penalties leading to local minima with non-moving solutions, while insufficient penalties failing to ensure energy-efficient behavior. Tuning these hyperparameters is computationally expensive, necessitating a more robust learning method. To address this, the study leverages SNNs, known for their computational efficiency and suitability for neuromorphic architectures. Gait experiments with a hexapod agent were conducted in a simulated environment, varying energy penalty settings to evaluate the impact on energy-efficient motion, measured through Cost of Transport (CoT), a standard gait efficiency metric. Integrating SNNs with state-of-the-art DeepRL algorithms, the study demonstrated improved energy efficiency. Their findings indicated that combining SNNs with DeepRL enables the learning of energy-efficient gaits while maintaining robustness across a wider range of penalty settings.

Multiscale dynamic coding spiking actor network (MDC-SAN) (10) similarly to PopSAN integrates population coding at the network level. However, this new approach utilizes dynamic neuron coding, featuring second-order neuronal dynamics at the individual neuron level. The work demonstrates that this design provides a

rich spatial-temporal state representation for decision-making. Their experimental results demonstrate that MDC-SAN outperforms conventional deep actor networks (DNN-based) on four continuous control tasks from OpenAI Gym.

The Deep Spiking Reinforcement Learning (DSRL) (15) method demonstrates how SNNs can be integrated into DeepRL algorithms such as Deep Q-Network (DQN) for discrete action spaces and Twin-Delayed Deep Deterministic Policy Gradient (TD3) for continuous control tasks. The study highlights the sensitivity of SNNs to hyperparameters like current and voltage decay factors and firing thresholds, requiring extensive tuning. However, techniques like increasing simulation time, applying two-neuron encoding for input observations, and randomizing membrane parameters significantly improve training stability and robustness. The findings show that SNNs can effectively handle complex continuous control problems with state-of-the-art DeepRL algorithms, and while training complexity may increase, the trained SNNs can leverage the energy efficiency of neuromorphic hardware for deployment.

Deep spiking Q-network (DSQN) (11) focuses on advancing the DSRL (15) method. DSRL development faces challenges due to the binary and nondifferentiable nature of spiking functions. To overcome these limitations, the study proposes DSQN built on leaky integrate-and-fire (LIF) neurons and DQN architecture. A direct spiking learning algorithm is adapted for DSQN, and the theoretical benefits of LIF neurons within this framework are analyzed. Extensive evaluations across 17 top-performing Atari games demonstrate the method’s superior performance, stability, generalization, and energy efficiency compared to state-of-the-art conversion techniques. Notably, this is the first directly trained SNN approach to achieve state-of-the-art results on multiple Atari games.

Another work (12) addresses the growing interest in integrating SNNs with DeepRL architectures to develop effective training methods. To maintain the asynchronous nature of SNNs, the study builds on recent advancements in temporal coding, adapting it for RL tasks. To overcome challenges such as the high sparsity of spikes,

the method introduces a self-incremental variable that encourages spiking neurons to fire, ensuring the network remains fully differentiable. Additionally, a novel encoding strategy is proposed to mitigate information loss in temporally coded inputs, preserving the fidelity of the data. Experimental evaluations demonstrate that SNNs trained using this approach achieve performance on par with state-of-the-art artificial neural networks in RL benchmark tasks, showcasing the potential of temporal coding for advancing SNN-based DeepRL.

One of the studies (13) explore the potential of memristor-based SNNs for efficient in-memory computation, addressing challenges associated with traditional learning methods like backpropagation, which are infeasible for such hardware. SNNs are particularly suited for this application, as their weights can be updated locally and self-organized without requiring high-precision global calculations. The study focuses on DeepRL tasks, where stochasticity in model initialization, training, or decision-making poses significant challenges. It proposes an online RL algorithm that updates connection weights incrementally after processing each environmental state during interaction-driven data generation. The algorithm is tailored for SNNs utilizing memristor-based synaptic plasticity rules inspired by spike-timing-dependent plasticity (STDP). These plasticity functions were derived from experimentally developed memristors made of poly-p-xylylene and CoFeB-LiNbO nanocomposite. The SNN, composed of leaky integrate-and-fire neurons, encodes environmental states as spike timings and decodes control actions through the timing of output spikes. Successfully solving the Cart-Pole benchmark task, the approach demonstrates a step toward real-time agent learning in continuous-time environments, paving the way for neuromorphic systems powered by memristive synapses.

In another study (14) SNNs are leveraged to achieve energy-efficient DeepRL for real-time obstacle avoidance in Unmanned Aerial Vehicles (UAVs) using event-based input from a Dynamic Vision Sensor (DVS). Unlike hybrid SNN implementations that rely on indirect training, this study directly trains SNNs, improving

performance and efficiency through an adapted Spatio-Temporal Backpropagation (STBP) algorithm tailored for RL tasks. Using a photorealistic simulation pipeline in AirSim, the SNN is benchmarked against a state-of-the-art Convolutional Neural Network (CNN) designed for the same obstacle avoidance task. The trained networks are also optimized for deployment on advanced neuromorphic hardware, with three embedded SNN versions evaluated for latency and throughput.

Experimental results demonstrate that the SNN achieves superior obstacle avoidance performance while consuming six times less energy than the CNN. Additionally, the study provides a detailed analysis of SNN hardware implementations, highlighting their energy efficiency and spiking activity, positioning SNNs as a robust and power-efficient alternative for embedded AI applications in robotics.

The fully spiking actor network with intralayer connections (ILC-SAN) (16) advances the integration of SNNs with DeepRL to enable energy-efficient AI for real-world control tasks requiring multidimensional deterministic policies. While recent surrogate gradient methods have enabled SNNs to achieve performance on par with deep networks, most existing spike-based DeepRL methods use firing rates to represent actions, converting them into continuous action spaces through fully connected (FC) layers. However, this approach relies on floating-point matrix operations, hindering deployment on neuromorphic hardware. To address this, the study proposes a fully spiking actor network (SAN) that eliminates floating-point operations by using the membrane voltage of nonspiking neurons to represent actions. Inspired by biological systems, multiple population neurons are introduced to decode different action dimensions, with intralayer connections enhancing spatial and temporal representation capacities. This design, termed ILC-SAN, mimics mechanisms observed in animal neural networks and significantly boosts representation efficiency. Experimental evaluations on OpenAI Gym tasks show that ILC-SAN outperforms state-of-the-art methods in continuous control tasks. The brain Q-network (BrainQN) (17) introduces , a spiking neural network (SNN)-based approach designed to address the limitations of low robustness and

high power consumption in DeepRL. By replacing traditional neurons in the classic DQN algorithm with biologically inspired SNN neurons, BrainQN leverages the energy efficiency and robustness inherent to third-generation neural networks. It utilizes both surrogate gradient learning (SGL) and ANN-to-SNN conversion methods for training. Robustness evaluations reveal BrainQN’s superiority under noisy conditions, achieving 82.14% higher rewards in low-noise scenarios and 71.74% in high-noise scenarios compared to DQN. Notably, SGL-trained BrainQN exhibits greater robustness than ANN-to-SNN conversion, attributed to differences in network output correlations and training methodologies. BrainQN’s effectiveness is further demonstrated by its successful transition from a simulated Pong environment to a real-world ball-catching robot equipped with dynamic vision sensors (DVS). On the neuromorphic chip PAICORE, BrainQN showcases significant improvements in latency and power efficiency compared to Jetson Xavier NX.

2.4 Contribution of SpikeRL to Energy-Efficient AI Systems

The reviewed works demonstrate various strategies employed to reduce the energy demands of AI systems, which is crucial for sustainable development and deployment, especially in resource-limited environments like mobile robotics. Building on these foundations, the SpikeRL framework incorporates the energy-efficient features of SNNs with the robust learning capabilities of DeepRL. By integrating MPI and NCCL for scalable distributed training and employing mixed precision techniques, SpikeRL aims to set new benchmarks for energy efficiency without sacrificing performance.

2.4.1 Distributed Training with MPI and NCCL

MPI (19) is a standardized library interface specification designed to support message-passing parallel programming models. In MPI, data is transferred from the address space of one process to another through cooperative operations, facilitating efficient communication in distributed memory systems. MPI extends the classical message-passing model by including features like collective operations, remote-memory access, dynamic process creation, and parallel I/O. Importantly, MPI is a specification rather than an implementation, meaning multiple implementations exist that adhere to this standard. It is not a programming language; instead, MPI operations are expressed as functions, subroutines, or methods, with standard language bindings provided for C and Fortran. Developed through a collaborative effort among parallel computing vendors, computer scientists, and application developers, MPI aims to establish a practical, portable, efficient, and flexible standard for writing message-passing programs. Key goals of MPI include designing an application programming interface that allows for efficient communication—minimizing memory copying and enabling overlap of computation and communication as well as supporting heterogeneous computing environments. The interface assumes a reliable communication subsystem, freeing the user from handling communication failures, and is designed to be thread-safe and language-independent. By providing a clear set of routines that can be efficiently implemented and potentially supported by hardware, MPI enhances scalability and portability across various vendor platforms without necessitating significant changes to underlying communication and system software.

NCCL (pronounced "Nickel") (20) is a library developed to accelerate inter-GPU communication by providing topology-aware collective communication and point-to-point send/receive primitives. It focuses on efficient communication by implementing operations like AllReduce, Broadcast, Reduce, AllGather, and ReduceScatter within single kernels that handle both computation and

communication tasks. This approach enables fast synchronization and optimal bandwidth utilization across multiple GPUs, both within and across nodes. NCCL supports various interconnect technologies such as PCIe, NVLINK, InfiniBand Verbs, and IP sockets, and is compatible with different multi-GPU parallelization models, including single-threaded, multi-threaded, and multi-process configurations. By offering a simple C API that aligns closely with the MPI collectives API and integrating seamlessly with the CUDA programming model through a "stream" argument, NCCL simplifies development and eliminates the need for machine-specific optimizations. It is particularly beneficial in deep learning frameworks for tasks like neural network training, where efficient AllReduce operations are crucial.

2.4.2 Mixed Precision

The approach of mixed precision training, elaborated by (21; 2; 3), represents a crucial strategy in reducing computational demands and enhancing the speed of operations. By employing a half-precision floating-point format (FP16) alongside a single-precision floating-point format (FP32), this technique significantly reduces memory bandwidth requirements and accelerates computations, particularly on platforms equipped with NVIDIA Tensor Cores (3). This approach not only conserves energy but also maintains model accuracy through strategic loss-scaling and precision management (3).

SpikeRL directly addresses the dual challenges of scalability and efficiency highlighted in previous studies by utilizing distributed training and mixed-precision training, making it a valuable contribution towards the advancement of sustainable AI.

Chapter 3

Methodology

This chapter outlines the detailed methodology employed in the development of the SpikeRL framework, which integrates SNNs with DeepRL to create an energy-efficient approach to AI computing. The framework, first introduced in (1), consists of three main components: DeepRL based SNN with population encoding, distributed training with MPI and NCCL, and optimization through mixed-precision training. Each component is designed to enhance the efficiency and scalability of the model, crucial for sustainable AI development.

3.1 SpikeRL Framework Architecture

The SpikeRL system architecture is illustrated in Figure 3.1. The SpikeRL framework consists of three major components. First, a DeepRL-based SNN model utilizing population encoding and decoding. Second, distributed training with both MPI and NCCL backends is implemented through the PyTorch Distributed package. Lastly, further optimization for model training is achieved by using mixed-precision training with BFLOAT16 data type for parameter updates.

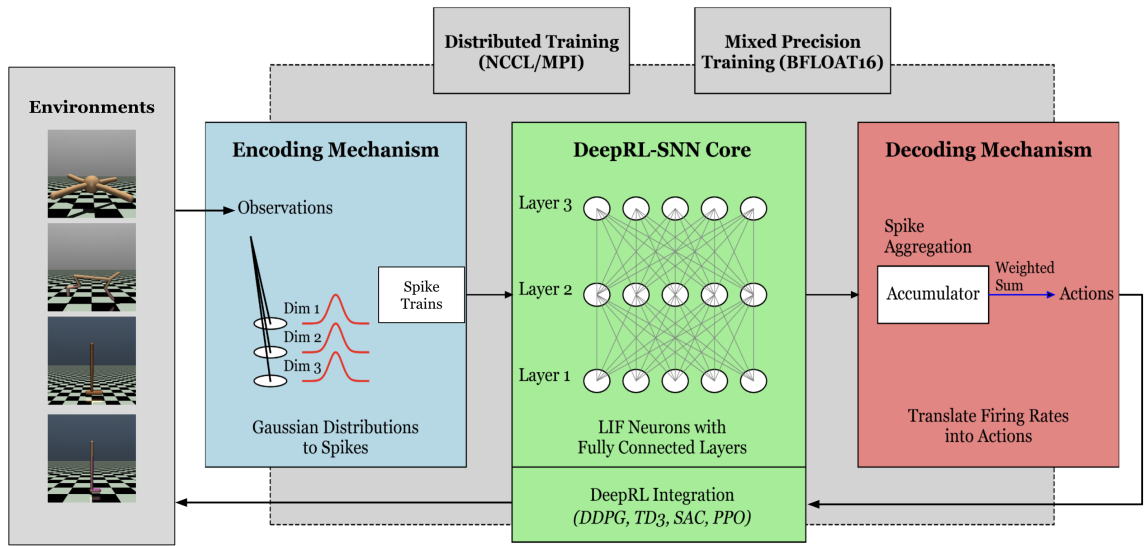


Figure 3.1: System architecture of the SpikeRL framework.

3.1.1 Population Encoding and Decoding

The SpikeRL framework employs population encoding and decoding strategies to transform continuous observations into spike trains suitable for the SNN and to map spiking network outputs back to continuous actions, respectively. This approach captures the temporal dynamics of spiking neurons and leverages their ability to process information in a manner akin to biological neural systems. Population encoding in SpikeRL converts continuous observations from the environment into spike trains that can be processed by the SNN actor network. This encoding involves several steps:

Gaussian Receptive Field Encoding

Each dimension of the observation vector $\mathbf{o} \in \mathbb{R}^{n_{\text{obs}}}$ is represented by a population of neurons, each with a Gaussian receptive field. The activation a_i of the i -th neuron for a given observation $\mathbf{o}$ is computed using a Gaussian function:

$$\text{pop_act}_i(\mathbf{o}) = \exp\left(-\frac{1}{2} \times \frac{(\mathbf{o} - \boldsymbol{\mu}_i)^2}{\sigma_i^2}\right) \quad (3.1)$$

Where, $\boldsymbol{\mu}_i$ is the preferred stimulus (mean) of the i -th neuron. σ_i is the standard deviation (width) of the Gaussian receptive field for the i -th neuron. $\text{pop_act}_i(\mathbf{o})$ gives the continuous activation level of the i -th neuron, representing how strongly this neuron responds to the observation vector $\mathbf{o}$. The resulting activation values pop_act lie within the range $[0, 1]$ and form a population code representing the observation space.

Spike Generation

Based on these activation values, each neuron’s output spike behavior is determined. SpikeRL provides two types of encoders for generating spikes from activation levels: I) Regular Encoder and II) Poisson Encoder.

I) Regular Encoder: Uses a deterministic approach where membrane potential V_t accumulates activation over timesteps. At each timestep t , the membrane potential V_t is updated, and a spike S_t is generated if the membrane potential exceeds a threshold V_{th} as show in the following equations.

$$V_t = V_{t-1} + \text{pop_act} \quad (3.2)$$

$$S_t = \begin{cases} 1 & \text{if } V_t \geq V_{th} \\ 0 & \text{otherwise} \end{cases} \quad (3.3)$$

$$V_t = V_t - S_t \times V_{th} \quad (3.4)$$

Here, the membrane potential integrates the input activation, and a spike is emitted whenever V_t crosses the threshold, after which the voltage is reset.

II) Poisson Encoder: Generates spikes probabilistically at each timestep based on the neuron's activation value $p = \text{pop_act}$. Spike generation follows a Bernoulli process shown in the following equation.

$$S_t \sim \text{Bernoulli}(p) \quad (3.5)$$

The probability of spiking at each timestep t is equal to the neuron's activation value p . This introduces stochasticity in the spike generation, aligning with the probabilistic nature of neuronal firing observed in biological systems.

Population Code Output

Both encoders output spike trains $\mathbf{S}_{\text{enc}} \in \{0, 1\}^{n_{\text{batch}} \times n_{\text{enc}} \times T_s}$, where, n_{batch} is the batch size, n_{enc} is the total number of encoder neurons (observation dimensions multiplied by population dimensions), T_s is the number of spike timesteps. These spike trains provide a temporal, distributed representation of the observations to the SNN actor network.

Population Decoding

After the SNN actor network processes the encoded spike trains, it produces spike activities corresponding to action values. The population decoding phase converts these spike activities into continuous actions that can be executed in the environment. The SNN’s output layer consists of neurons representing each action dimension. Over multiple timesteps, these neurons generate spike trains. The output spike activities $\mathbf{S}_{\text{out}}$ for each action dimension are integrated or averaged over the simulation period as denoted by the following equation.

$$\text{out_pop_act} = \frac{1}{T_s} \sum_{t=1}^{T_s} S_{\text{out},t} \quad (3.6)$$

Where, $S_{\text{out},t}$ is the spike output of the SNN’s output layer at timestep t , T_s is the number of timesteps.

The decoded output out_pop_act represents the firing rates or accumulated spike activities of each neuron in the output population. The decoder layer maps these activities to continuous action values with the following steps.

I) Convolutional or Linear Transformation: The decoder layer (e.g., a 1D convolutional layer for group-wise operation) transforms spike activities into raw action values.

$$\text{raw_act} = W_{\text{dec}} \times \text{out_pop_act} + b_{\text{dec}} \quad (3.7)$$

where W_{dec} and b_{dec} are the decoder’s weights and biases.

II) Non-Linear Activation: An activation function (e.g., the hyperbolic tangent, $\tanh$) is applied to ensure that actions remain within valid ranges.

$$\text{Actions} = \text{act_limit} \cdot \tanh(\text{raw_act}) \quad (3.8)$$

Here, act_limit scales the output to match the environment’s action bounds and $\tanh(\cdot)$ ensures the action values lie between -1 and 1 if $\text{act_limit} = 1$. This results

in a continuous action vector $\mathbf{a} \in \mathbb{R}^{n_{\text{act}}}$, where n_{act} is the dimension of the action space. The final decoded continuous actions, computed from the SNN output spikes, are used to interact with the environment as shown in the following equation.

$$\mathbf{a} = \text{act_limit} \cdot \tanh \left(\text{Decoder} \left(\frac{1}{T_s} \sum_{t=1}^{T_s} S_{\text{out},t} \right) \right) \quad (3.9)$$

Significance of Population coding in the SpikeRL Framework

The encoding and decoding processes allow the SNN to capture and exploit the temporal aspects of information, essential for tasks with temporal dependencies (8).

Exploration vs. Exploitation: The Poisson encoder’s stochastic spiking can enhance exploration in certain environments, while the regular encoder’s deterministic approach can provide more stable inputs in others.

Biological Plausibility: Using population encoding and decoding draws inspiration from how sensory and motor information is represented and processed in biological neural systems, potentially leveraging these biological properties for improved learning performance (8).

Integration with Twin Delayed Deep Deterministic Policy Gradient

(TD3) (37): By providing a spike-based representation of states and actions, population encoding and decoding smoothly integrates with the TD3 algorithm’s actor and critic networks, enabling an SNN-based policy representation and evaluation.

In summary, the population encoding and decoding steps in the SpikeRL framework transform continuous data between the environment and the spiking domain of the SNN actor network. These transformations, illustrated through Equations 3.1 to 3.9, enable the SNN to receive meaningful spike-based inputs and produce spike-based outputs that can be decoded into continuous actions, thereby bridging the gap between spiking network computations and continuous control tasks in reinforcement learning.

3.1.2 DeepRL-SNN Integration

The SpikeRL framework integrates the SNN with a DeepRL algorithm, specifically the TD3 algorithm. This integration involves two main steps. First, processing encoded spike trains through the SNN actor network to compute continuous actions. Second, integrating the SNN actor with the TD3 framework, which uses two critic networks to evaluate actions and update both the actor and critics.

Processing Spike Trains Through the SNN Actor Network

After the continuous observations from the environment are encoded into spike trains by the population encoder, these spike trains are fed into the SNN actor network. The SNN actor processes these inputs over multiple timesteps and outputs spiking activity that is then decoded into continuous actions.

The SNN actor network is composed of an input layer receiving spike trains from the encoder, hidden layers of spiking neurons, and an output layer that produces spike activities to represent action values. The hidden layers utilize the Leaky Integrate-and-Fire (LIF) (38) model. At each timestep $t \in \{1, \dots, T_s\}$, the spike trains $S_{\text{enc},t}$ from the encoder are fed into the input layer. For each spiking neuron in the SNN, the synaptic current I_t and membrane voltage V_t are updated.

$$I_t = \alpha_I \cdot I_{t-1} + W \cdot S_{\text{pre},t} \quad (3.10)$$

$$V_t = \alpha_V \cdot V_{t-1} \cdot (1 - S_{t-1}) + I_t \quad (3.11)$$

Here, α_I and α_V are decay factors for current and voltage, W are the synaptic weights, $S_{\text{pre},t}$ are the spikes from pre-synaptic neurons at timestep t . A spike S_t is generated if the membrane voltage V_t exceeds a threshold V_{th} as shown in equation 3.4. The output spiking activity is accumulated over timesteps to represent the neuron's firing rate.

$$S_{\text{out,accum}} = \frac{1}{T_s} \sum_{t=1}^{T_s} S_{\text{out},t} \quad (3.12)$$

Here, $S_{\text{out},t}$ are the spikes from the output layer at timestep t . The averaged output activity $S_{\text{out,accum}}$ is used to represent the continuous action value. The SNN’s output spiking activity for each action dimension is decoded into a continuous action vector as described by equation 3.9 in population coding.

Integration with the TD3 Algorithm

The output actions from the actor network (SNN) are evaluated and updated using the TD3 algorithm. TD3 addresses function approximation errors in the deep deterministic policy gradient (DDPG) (39) by introducing clipped double Q-learning, delayed policy updates, and target policy smoothing (37).

The TD3 algorithm maintains an actor network π_θ (in SpikeRL, this is the SNN actor) that selects actions based on states, two critic networks Q_{ϕ_1} and Q_{ϕ_2} (regular Multilayer Perceptrons(MLPs) in SpikeRL) that estimate Q-values, target networks $\pi_{\theta_{\text{targ}}}$, $Q_{\phi_{1,\text{targ}}}$, and $Q_{\phi_{2,\text{targ}}}$ for stable target calculations (37).

The SNN actor outputs continuous actions $\mathbf{a}$ after processing spike trains and decoding them. These actions, along with the observations $\mathbf{s}$, are fed into the critic networks Q_{ϕ_1} and Q_{ϕ_2} to compute Q-values.

$$Q_{\phi_i}(\mathbf{s}, \mathbf{a}) = \text{MLP}_{\phi_i}([s, a]) \quad (3.13)$$

where $[\mathbf{s}, \mathbf{a}]$ denotes the concatenation of observation and action vectors. The actor network (SNN) is updated using surrogate gradients as shown in the following equation.

$$\nabla_\theta J = \mathbb{E} [\nabla_a Q_{\phi_1}(s, a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)] \quad (3.14)$$

Since spiking neurons are non-differentiable, surrogate gradient methods are used to approximate gradients of the spike activation function. In SpikeRL, once every d iterations, the actor network is updated by maximizing the Q-value Q_{ϕ_1} .

3.2 Distributed Training with MPI and NCCL

3.2.1 Custom MPI Implementation Using mpi4py

As shown in the previous work in (1), SpikeRL initially utilized the MPI interface through `mpi4py` (40) to achieve scalability in the training process across multiple computational nodes. In this implementation as shown in Figure 3.2, MPI is first set up using `MPI.COMM_WORLD`, retrieving the total number of processes (`size`) and the rank of each individual process (`rank`). Each process is assigned a unique random seed that depends on its rank, ensuring diversity in experience sampling across processes. Each MPI process is assigned to a specific GPU device based on its rank. All processes initialize identical actor-critic models, and the model parameters are synchronized across processes at the start using a custom `sync_params` function, which broadcasts parameters from the root process to all others.

During training, each process interacts with its own instance of the environment, collecting transitions independently. These experiences are stored in a shared replay buffer in each process. For parameter updates, every process samples a batch of experiences from its replay buffer and computes gradients locally. To ensure consistent model updates across all processes, SpikeRL uses the Allreduce MPI operation in the custom `avg_grads` function to average the computed gradients over all processes. Each parameter’s gradient is summed across all processes and then divided by the total number of processes, ensuring that the learning update is the result of the combined experiences of all parallel environments. After gradient averaging, each process applies the update to the model parameters, keeping models synchronized. Also, processes periodically synchronize the target networks to maintain consistent target parameters. Although this custom setup allowed SpikeRL to harness parallel GPU resources and distributed training, the limitations of this setup was load imbalance causing overhead of MPI communication as discussed in (1).

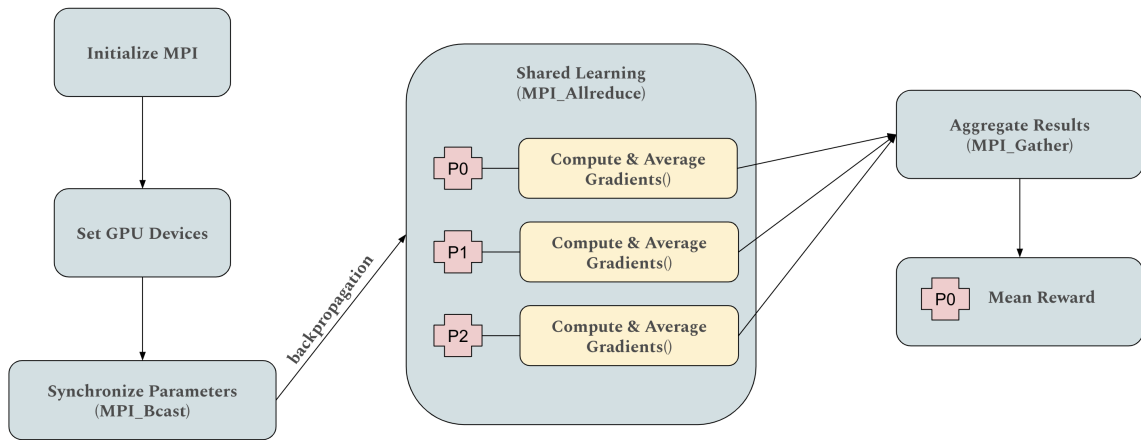


Figure 3.2: Distributed training with MPI using *mpi4py*.

3.2.2 Final Implementation with MPI and NCCL

The limitations in the initial setup (1) motivated further exploration of distributed training implementations. The final setup as illustrated in Figure 3.3, shows a more efficient implementation of distributed training using the distributed package from PyTorch.

In this setup, each process is initialized in a distributed setting with either the NCCL or MPI backend (chosen by the user) and assigned to a specific GPU device based on its local rank, ensuring parallelization of computation across multiple GPUs. For the MPI backend support, PyTorch is compiled from source using the OpenMPI installation. The actor-critic model and target networks are instantiated within each process to maintain identical versions of the model. To parallelize training, PyTorch’s DistributedDataParallel (DDP) wrapper is applied to these models. The DDP wrapper synchronizes gradients across all processes using the Allreduce collective operation, ensuring that after each backward pass, gradients are averaged, and model parameters are updated consistently across all processes. Each process independently interacts with its own copy of the environment, collecting distinct experiences and storing them in its local replay buffer. During training updates, each process samples a batch of data from its local buffer, performs forward and backward passes, and DDP automatically handles the communication of gradients between processes. Consequently, all processes contribute their gradient updates, which are averaged and applied across all replicated models in sync. When updating the target networks, each process computes the updates locally, and DDP ensures that the model parameters remain synchronized across all processes after each update. For evaluation, Allreduce is used to aggregate the test rewards from each process. The root process calculates the mean reward and saves the model. This distributed setup using PyTorch’s DDP framework optimally utilizes available GPUs to accelerate training and avoids the limitations of the previous implementation (1).

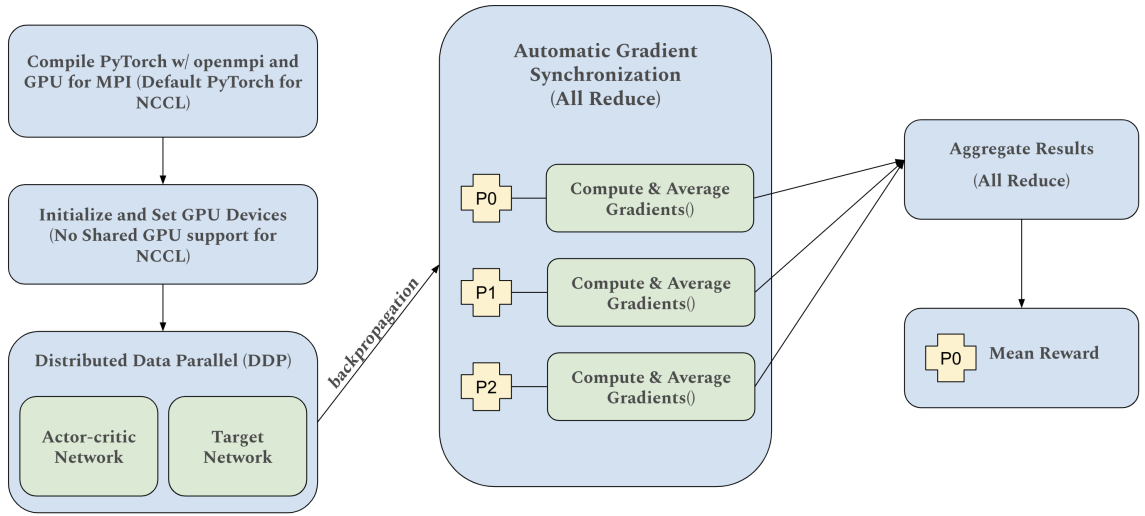


Figure 3.3: Distributed training with MPI/NCCL backend using PyTorch distributed package.

3.3 Optimization with Mixed Precision

The final component of the SpikeRL framework is the optimization of the training process through mixed-precision techniques implemented by using PyTorch’s Automatic Mixed Precision (AMP) that utilize NVIDIA’s AMP toolkit (41) to accelerate computations and reduce memory usage while maintaining training stability. Specifically, SpikeRL implements AMP using the `autocast` context manager and the `GradScaler` class.

Autocast Context Manager: During each training iteration, the forward and backward passes for both the Q-network update and the actor network (Spiking Actor-Critic) update, are enclosed within the autocast context. As shown in Figure 3.4, the context manager automatically casts operations to a specified lower precision (BFLOAT16 in this case) where it is beneficial, while keeping operations that are more sensitive in full precision (FLOAT32). The matrix multiplication operations of weights and inputs in the forward pass occur in BF16 for faster computation, while loss computation occurs in FP32 to maintain stability. The convolution operations in the backward pass again occurs in BF16, while weight updates are done in FP32 to maintain accuracy.

GradScaler: The `GradScaler` function dynamically adjusts the scale of the loss to prevent gradients from underflowing to zero when working with low precision (BF16). As shown in Figure 3.4, after computing the forward passes and the losses, the `GradScaler` is used to scale the loss values before the backward pass. This scaling step ensures gradients remain in a good numerical range to prevent issues like underflow that can occur with very small gradient values. Subsequently, the `GradScaler` method is used to update the parameters, and the loss is descaled before the updates. It also adjusts the scaling factor for the next iteration based on whether gradients overflowed or not. This dynamic scaling ensures stable and efficient training by adjusting the scale to an appropriate value in each iteration.

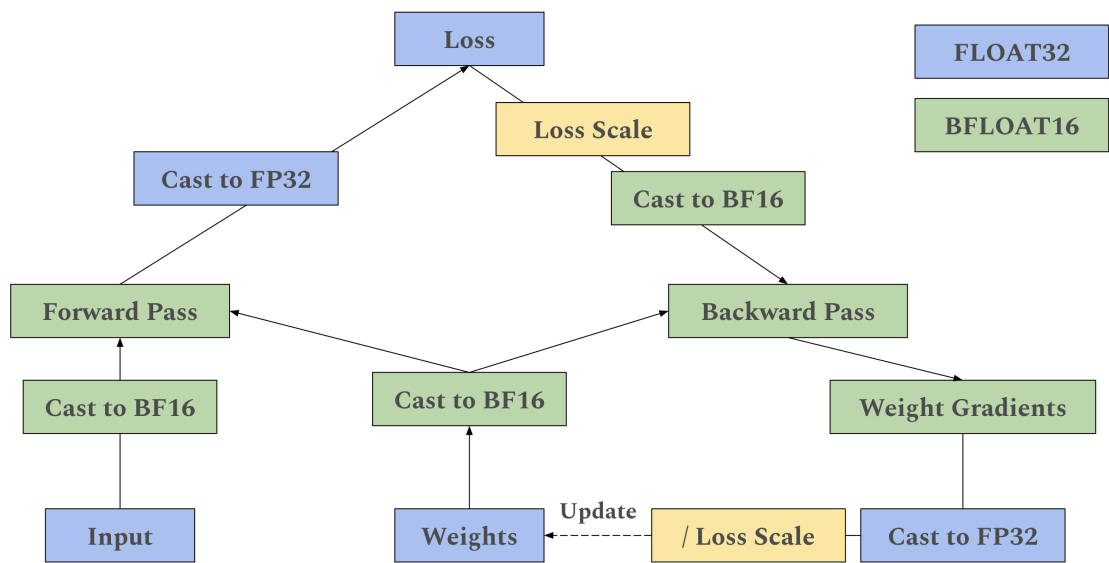


Figure 3.4: Automatic mixed-precision training with BFLOAT16.

3.4 Energy Measurements

The evaluation of energy efficiency in computational applications is multifaceted, encompassing various methodologies that offer insights into the utilization of energy resources by both the application and its underlying hardware.

3.4.1 Energy Measurement Tools and Their Comparison

As studied in (3), several tools are available for measuring the energy consumption and carbon footprint of training AI models. This section provides an overview of these tools.

- **CodeCarbon (CC)** (42): An embedded package that uses Intel’s Running Average Power Limit (RAPL) (43) for CPU and NVIDIA’s Management Library (NVML) (44) with pynvml for GPU energy measurements. It is known for its accuracy in direct measurement scenarios but requires root access for full functionality.
- **Green-Algorithms (GA)** (45): An online calculator and server-side tool that estimates energy consumption based on the TDP values from a database or manual input, suitable for quick assessments but may overestimate energy use.
- **Eco2AI (E2)** (46): Uses CPU/GPU models to fetch TDP values and psutil for memory usage measurements, offering flexibility with higher overhead.
- **CarbonTracker (CT)** (47): Supports real-time fetching of carbon intensity data for specific regions, using RAPL and pynvml, best for applications where regional carbon intensity data is crucial.
- **Experiment-Impact-Tracker (EIT)** (48): Provides detailed energy usage factors using RAPL and nvidia-smi, suitable for detailed analysis but with higher overhead.

- **MLCO2 (49)**: An online calculator that assumes maximum load for GPUs, providing a simplified energy estimation approach.
- **Cumulator (CMLTRs) (50)**: Measures energy for either CPU or GPU at a time, with simple setup but limited detailed tracking.

After much testing, the tool utilized for the energy measurements of SpikeRL, is CodeCarbon due its detailed reports on energy consumption and carbon emissions of different hardware elements. The degrees of detail and accuracy in estimating energy consumption and carbon emissions provided by CodeCarbon is crucial for assessing the energy efficiency of the framework and its deployment in different computational environments.

3.4.2 Metrics for Measuring Energy Efficiency

The quantification of energy consumption of a system typically involves measuring the power (in watts) used by the different components of the system over time. The most accurate method to measure energy involves recording the entire history of power usage during the application’s runtime and computing the area under the power-time curve (3). This approach is ideally implemented using tools like RAPL and NVML. An alternative, more simplified calculation is given by the formula in (3).

$$E = PUE \times \sum_{e \in E} \sum_{d \in D} P_{avg,d,e} T_e \quad (3.15)$$

where PUE is the Power Usage Effectiveness, E represents different components, D denotes different time intervals, $P_{avg,d,e}$ is the average power consumed by component e during interval d , and T_e is the total active time for component e . Additionally, to assess the environmental impact of an application, metrics such as Power Usage Effectiveness (PUE) (51) and carbon dioxide equivalent (CO2) emissions are considered. PUE, for instance, is a ratio that measures how efficiently

a data center uses energy, specifically comparing total facility energy use to the energy used by the IT equipment alone. A lower PUE indicates higher energy efficiency “with the ideal value being 1 or 100% supplied power used by the equipment and 0% wasted”.

Using the multiplication of total energy consumption and total execution time, another useful energy metric is computed called the Energy Delay Product (EDP) ($\text{EDP} = \text{Energy} \times \text{Run-time}$). EDP provides a metric that balances both performance (through delay) and energy consumption. Lower EDP values indicate a more efficient system in terms of both speed and energy usage (3).

While the EDP is a widely used metric that considers both energy consumption and run-time, it is not always adequate for evaluating software energy efficiency. This is because EDP can yield the same value for optimizations that have significantly different impacts on performance and energy consumption. For instance, one optimization might drastically reduce execution time but increase power consumption, while another might moderately improve both performance and energy usage. Despite these differences, both could have the same EDP, making it difficult to distinguish which optimization is more energy-efficient. This limitation highlights the necessity for the Greenup metric (52), which assesses the ratio of energy consumption before and after optimization ($\text{Greenup} = \text{Original Energy} / \text{Optimized Energy}$). The Greenup energy efficiency metric is calculated as,

$$\text{Greenup} = \frac{\text{Speedup}(\frac{T\phi}{T_o})}{\text{Powerup}(\frac{P_o}{P\phi})} \quad (3.16)$$

where, $T\phi$ is optimized execution time, T_o is base execution time, $P\phi$ is optimized powerup and P_o is base powerup. This metric can be demonstrated by the Greenup, Powerup (change in power consumption) and Speedup (performance improvement) (GPS-UP) quadrant graph that shows four energy saving zones as green zones with four red zones that waste energy (see the graph in (52)). When used alongside Speedup and Powerup, Greenup provides a more nuanced

understanding of how an optimization affects both energy efficiency and performance. This trio of metrics allows for a comprehensive evaluation, effectively categorizing optimizations based on whether they save energy, improve performance, or do both, thus offering a clearer and more informative assessment than EDP alone. The Greenup metric provides a more comprehensive insight into how the energy efficiency is gained or how energy is being wasted by the application.

Chapter 4

Results

This chapter presents the results for the SpikeRL framework, focusing on the impact of the distributed training and mixed-precision training on the performance and efficiency of reinforcement learning tasks. The results are compared with two state-of-the-art methods, PopSAN and DSRL, which do not utilize distributed training or mixed-precision techniques.

4.1 Experimental Setup

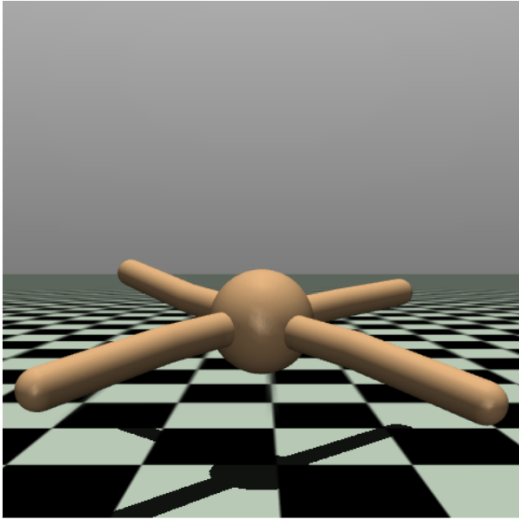
4.1.1 Cluster Configuration

For preliminary results presented in (1), the experiments were conducted on the ISAAC-NG High Performance & Scientific Computing (HPSC) cluster (53), which is equipped with Tesla V100S-PCIE-32GB (54) GPUs. Final performance and energy efficiency benchmarks were conducted on the Guyot node of the Innovative Computing Lab’s (ICL) cluster, equipped with 8 Nvidia A100 80GB GPUs.

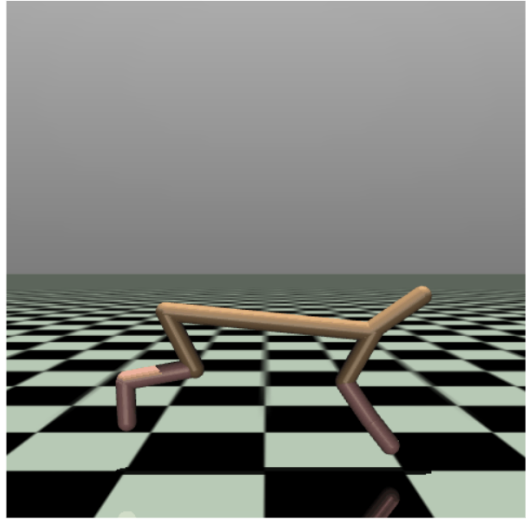
4.1.2 Simulation Environment

Figure 4.1 shows the gymnasium (55) Mujoco (56) environments that were chosen for the continuous control experiments. The environments are as follows:

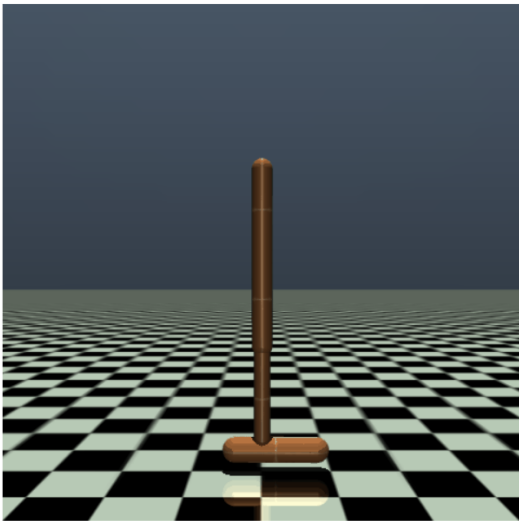
Ant-v4



HalfCheetah-v4



Hopper-v4



Walker2d-v4

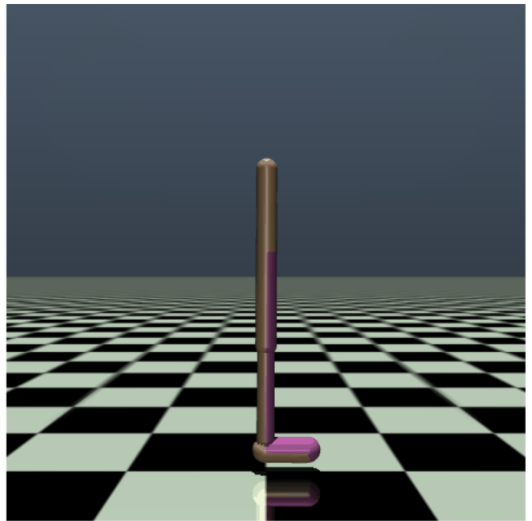


Figure 4.1: OpenAI gym environments from the MuJoCo physics simulator.

Ant-v4: A 3D quadruped robot with 8 hinges. It can take actions in the environment by applying torque to its 8 hinges. It has 105 observation space with position, velocity, and mass values.

HalfCheetah-v4: A 2D robot with 6 hinges. It can take actions in the environment by applying torque to its 6 hinges to move forward as fast as possible. It has 17 observation space with position and velocity values.

Hopper-v4: A 2D one-legged robot with 3 hinges. It can take actions in the environment to hop forward by applying torque to its 3 hinges. It has 11 observation space with position and velocity values.

Walker2D-v4: A 2D two-legged robot with 6 hinges. It can take actions in the environment by applying torque to its 6 hinges to move forward. It has 17 observation space with position and velocity values.

These continuous control problems serve as standard benchmarks within the reinforcement learning community.

4.1.3 Training Details

In preliminary experiments in (1), the SpikeRL model was trained on the Walker2D-v4 environment for over 10 epochs with 10k steps, employing various numbers of MPI processes (2, 4, 8, 16, and 32) to assess the scalability and efficiency of the distributed training. These tests were not repeated after the original publication because additional tests on Ant-v4, Hopper-v4, and HalfCheetah-v4 environments were done for comparing SpikeRL with the state-of-the-art SNN methods such as PopSAN and DSRL. These three environments were common for the two baseline methods.

4.2 Experimental Results

4.2.1 Preliminary Results

The preliminary results presented in the previous work (1), are also illustrated here in Figure 4.2 and Figure 4.3. Figure 4.2 shows that an increase in MPI ranks leads to improvements in both the highest and mean test rewards. This trend indicates effective shared learning and scalability of the model across MPI processes.

Figure 4.3 illustrates that the SpikeRL model outperforms PopSAN and DeepRL methods by almost 1k average rewards after 100k training steps. However, the limitation of this initial implementation is load imbalance and MPI communication overhead. With the increasing number of MPI ranks, specifically at 16 and 32 ranks, the communication overhead grew despite maintaining a fixed computational load of 100k training steps (see details in (1)).

4.2.2 Final Performance Benchmarks

The final performance benchmarks are done with the SpikeRL version utilizing PyTorch’s distributed package for distributed training with both MPI and NCCL. Figures 4.4, 4.5, and 4.6 present the SpikeRL performance benchmarks across 5 runs in the Ant, Hopper, and HalfCheetah environments, compared to the PopSAN and DSRL methods. For Ant environment, SpikeRL achieves 3.14% higher mean reward compared to PopSAN and achieves 65.62% higher mean reward compared to DSRL. In the Hopper environment, SpikeRL achieves 2.01% higher mean reward compared to PopSAN and achieves 98.26% higher mean reward compared to DSRL. For HalfCheetah, SpikeRL achieves 1.51% higher mean reward compared to PopSAN and achieves 70.91% higher mean reward compared to DSRL. Overall, SpikeRL outperformed both PopSAN and DSRL in all three continuous control environments.

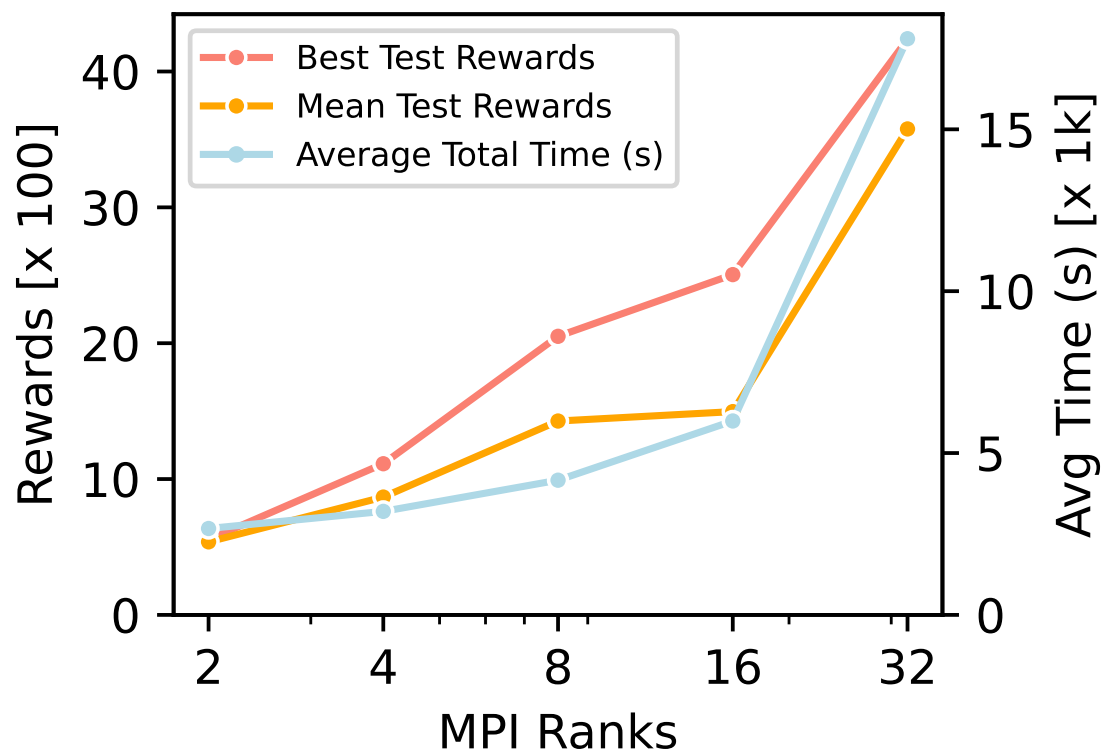


Figure 4.2: Distributed training performance results of SpikeRL on Walker2d-v4 environment.

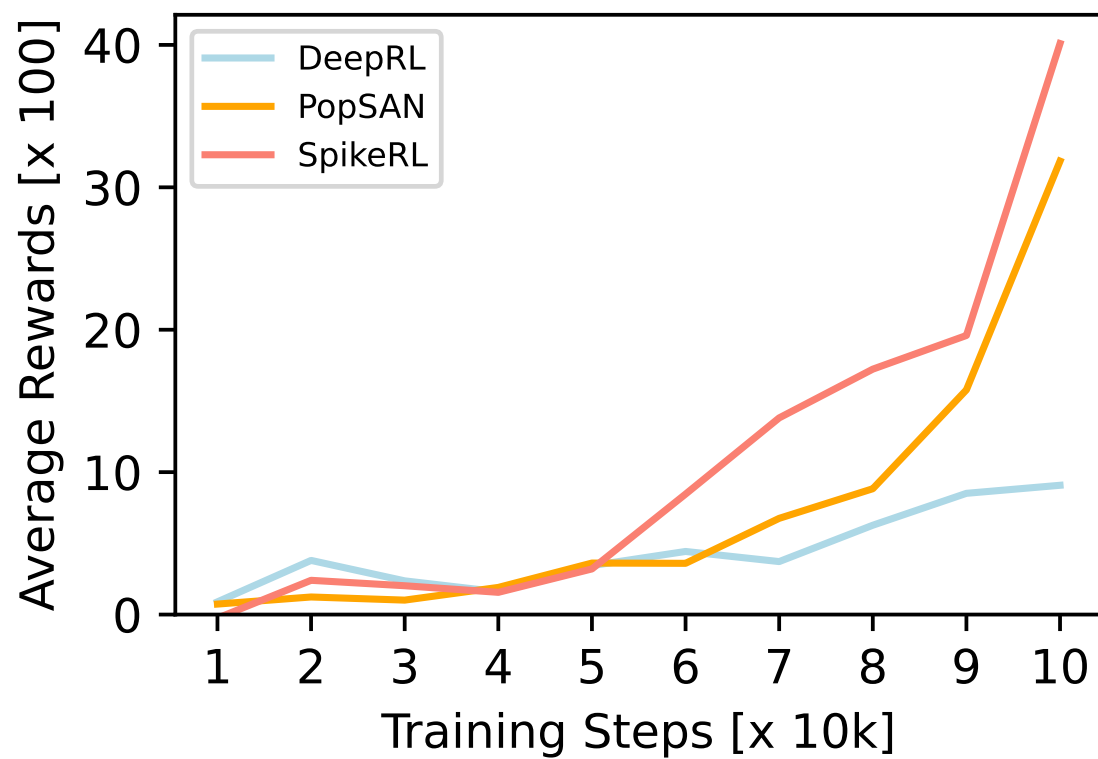


Figure 4.3: SpikeRL performance benchmark of 100k training steps against PopSAN and DeepRL methods on Walker2d-v4.

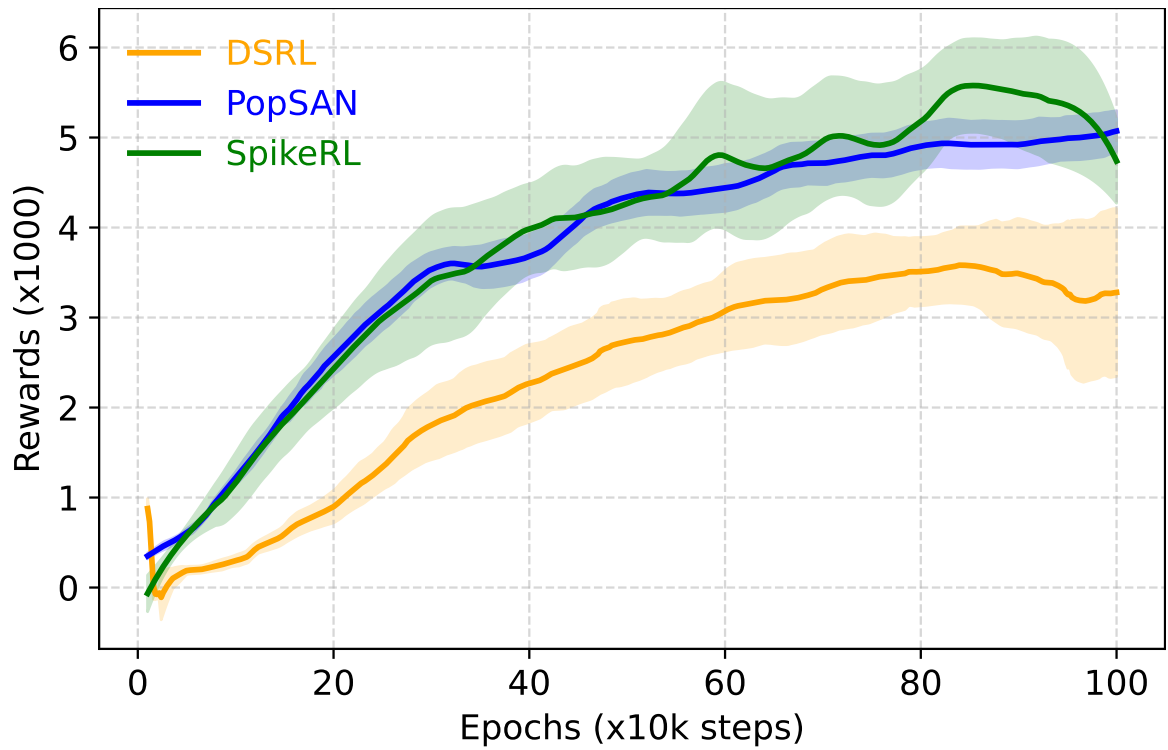


Figure 4.4: Ant-v4 performance benchmark of SpikeRL over PopSAN and DSRL (mean of 5 runs with shaded area as standard deviation).

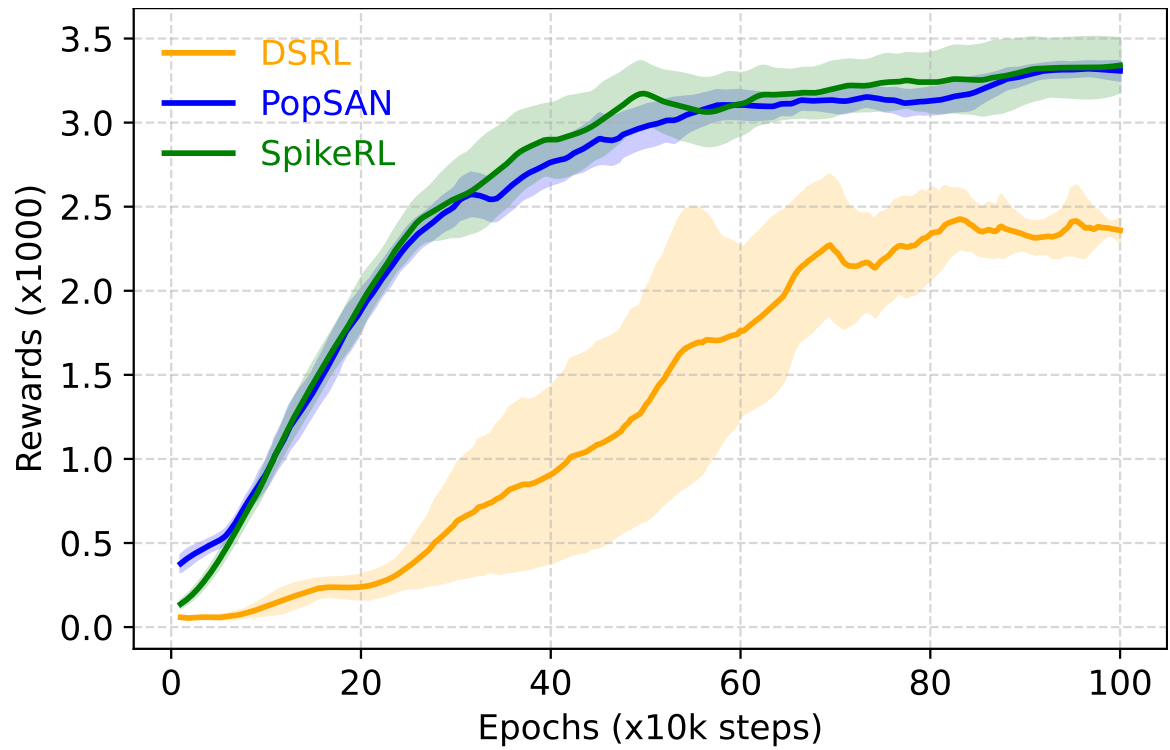


Figure 4.5: Hopper-v4 performance benchmark of SpikeRL over PopSAN and DSRL (mean of 5 runs with shaded area as standard deviation).

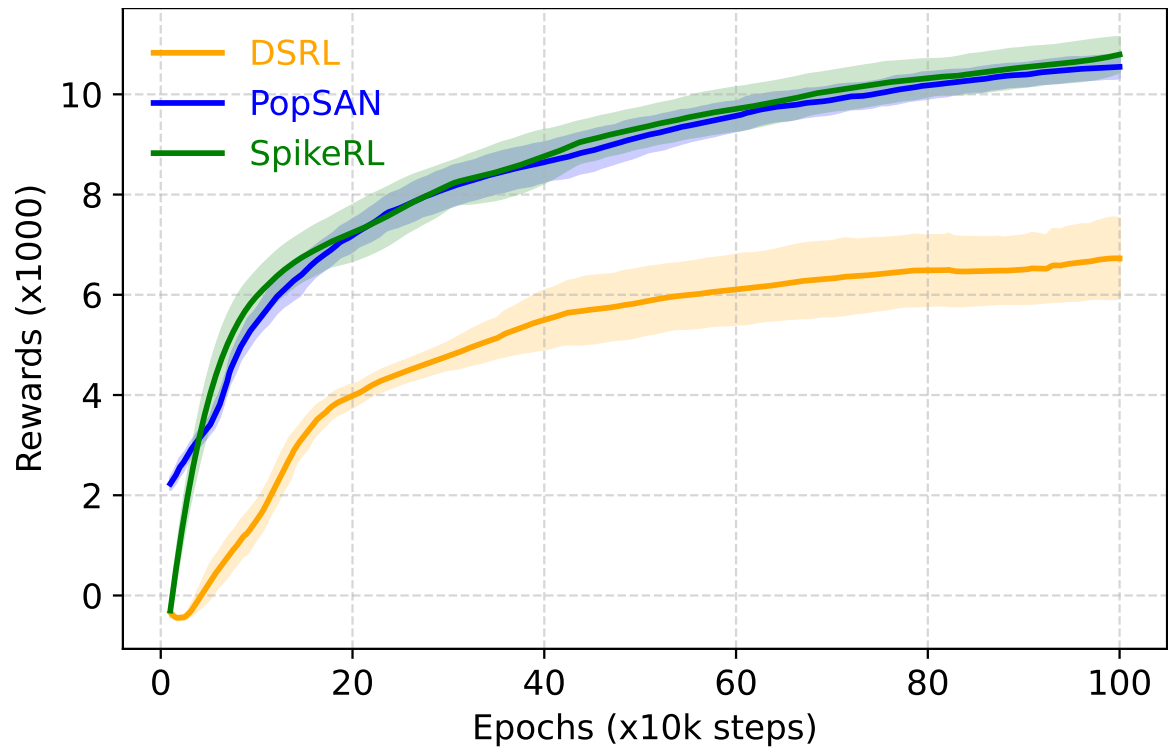


Figure 4.6: HalfCheetah-v4 performance benchmark of SpikeRL over PopSAN and DSRL (mean of 5 runs with shaded area as standard deviation).

4.2.3 Final Energy Efficiency Benchmarks Summary

The results of the energy efficiency benchmarks for SpikeRL (NCCL) and SpikeRL (MPI) over PopSAN in the OpenAI Gym environments, Ant-v4, Hopper-v4, and HalfCheetah-v4 are summarized in Tables 4.1, 4.2, and 4.3 respectively. Each benchmark includes metrics for execution time, energy consumption, and average power consumption, with further GPS-UP metric calculations for speedup, powerup, and greenup ratios to compare the methods.

For the Ant-v4 environment, SpikeRL (NCCL) demonstrates a substantial improvement over PopSAN, achieving a speedup of $1.99\times$. Though the powerup also goes up to $1.53\times$ suggesting more power consumption, the trade-off results in a greenup of $1.30\times$. SpikeRL (MPI) also shows efficiency gains over PopSAN with a speedup of $1.84\times$, a powerup of $1.49\times$, and a greenup of $1.24\times$. When comparing SpikeRL (NCCL) to SpikeRL (MPI), the NCCL backend exhibits a modest advantage with a speedup of $1.08\times$ and a greenup of $1.05\times$. So, SpikeRL (NCCL) was slightly more energy efficient than SpikeRL (MPI).

In the Hopper-v4 environment, SpikeRL (NCCL) improves over PopSAN with a speedup of $1.30\times$. In this case, it also consumes less power than PopSAN with its powerup being slightly below 1 at $0.93\times$. This combined efficiency results in a higher greenup than the Ant-v4 environment with its value reaching at $1.39\times$.

SpikeRL (MPI) displays a slight edge in performance with a speedup of $1.37\times$, a powerup of $0.97\times$, and a greenup of $1.41\times$. Comparing the two SpikeRL methods, the MPI backend had a marginal energy efficiency gain over the NCCL backend.

For HalfCheetah-v4, SpikeRL (NCCL) demonstrates a speedup of $1.48\times$, a powerup of $0.98\times$ and a greenup of $1.52\times$ over PopSAN. SpikeRL (MPI) shows similar gains over PopSAN, with a speedup of $1.41\times$, powerup of $0.96\times$, and greenup of $1.47\times$. Between the two SpikeRL models, NCCL is slightly more energy efficient in this case, with a speedup of $1.05\times$, powerup of $1.01\times$, and greenup of $1.03\times$.

Table 4.1: Energy efficiency benchmark for OpenAI gym Ant-v4

Metric	PopSAN	SpikeRL (NCCL)	SpikeRL (MPI)
Time (hours)	11.92	6.00	6.46
Energy Consumed (kWh)	18.70	14.40	15.11
Avg Power Consumption (kW)	1.57	2.40	2.34
GPS-UP Metric Calculations			
Comparison	Speedup	Powerup	Greenup
SpikeRL (NCCL) over PopSAN	1.99×	1.53×	1.30×
SpikeRL (MPI) over PopSAN	1.84×	1.49×	1.24×
SpikeRL (NCCL) over SpikeRL (MPI)	1.08×	1.03×	1.05×

Table 4.2: Energy efficiency benchmark for OpenAI gym Hopper-v4

Metric	PopSAN	SpikeRL (NCCL)	SpikeRL (MPI)
Time (hours)	8.52	6.57	6.22
Energy Consumed (kWh)	18.93	13.63	13.45
Avg Power Consumption (kW)	2.22	2.07	2.16
GPS-UP Metric Calculations			
Comparison	Speedup	Powerup	Greenup
SpikeRL (NCCL) over PopSAN	1.30×	0.93×	1.39×
SpikeRL (MPI) over PopSAN	1.37×	0.97×	1.41×
SpikeRL (NCCL) over SpikeRL (MPI)	0.95×	0.96×	0.99×

Table 4.3: Energy efficiency benchmark for OpenAI gym HalfCheetah-v4

Metric	PopSAN	SpikeRL (NCCL)	SpikeRL (MPI)
Time (hours)	9.09	6.14	6.43
Energy Consumed (kWh)	19.73	12.99	13.44
Avg Power Consumption (kW)	2.17	2.12	2.09
GPS-UP Metric Calculations			
Comparison	Speedup	Powerup	Greenup
SpikeRL (NCCL) over PopSAN	1.48×	0.98×	1.52×
SpikeRL (MPI) over PopSAN	1.41×	0.96×	1.47×
SpikeRL (NCCL) over SpikeRL (MPI)	1.05×	1.01×	1.03×

4.2.4 Greenup And Carbon Footprint Analysis

The GPS-UP quadrant graphs visually represent the greenup metric with powerup versus speedup across the three environments Ant-v4, Hopper-v4, and HalfCheetah-v4. Figure 4.7 illustrates the Ant-v4 GPS-UP quadrant graph, where SpikeRL (NCCL) and SpikeRL (MPI) are positioned in the green zone at the top right quadrant indicating that the energy efficiency is largely being attributed to their speedup over PopSAN rather than their powerup. In the case of Hopper-v4 GPS-UP quadrant graph in Figure 4.8, the greenup of both MPI and NCCL are in the all green zone of bottom right quadrant. This zone signifies that the energy efficiency in this case is a result of both speedup gain and from less power consumption. Figure 4.9 shows the GPS-UP quadrant graph of HalfCheetah-v4, where similarly to the Hopper-v4 environment, both SpikeRL models are in the all green zone with an even higher greenup value indicating more energy efficiency over PopSAN compared to the case of Ant-v4 and Hopper-v4 environments.

These energy efficiency gains are reflected in the carbon footprint analysis which highlights the environmental benefits of using SpikeRL over PopSAN across these environments. Figure 4.10 demonstrates carbon footprint analysis in the Ant-v4 environment where where SpikeRL reduces carbon emission by 23.5% compared to PopSAN which is equivalent to 23.8% less weekly household emissions, 23.8% less miles driven of a gasoline car, and 25% less tv hours watched. Figure 4.11 illustrates Hopper-v4 carbon footprint, where the reductions of SpikeRL are slightly more pronounced, with decreases around 28-29% for carbon and household emissions. Carbon footprint for HalfCheetah-v4 in Figure 4.12 showcases the most significant reductions, with a 34.8% decrease in carbon emission, 34.2% decrease in weekly household emissions, 36.4% less miles driven, and a similar 25% TV hours watched. These results illustrate the substantial environmental impact of using SpikeRL over PopSAN.

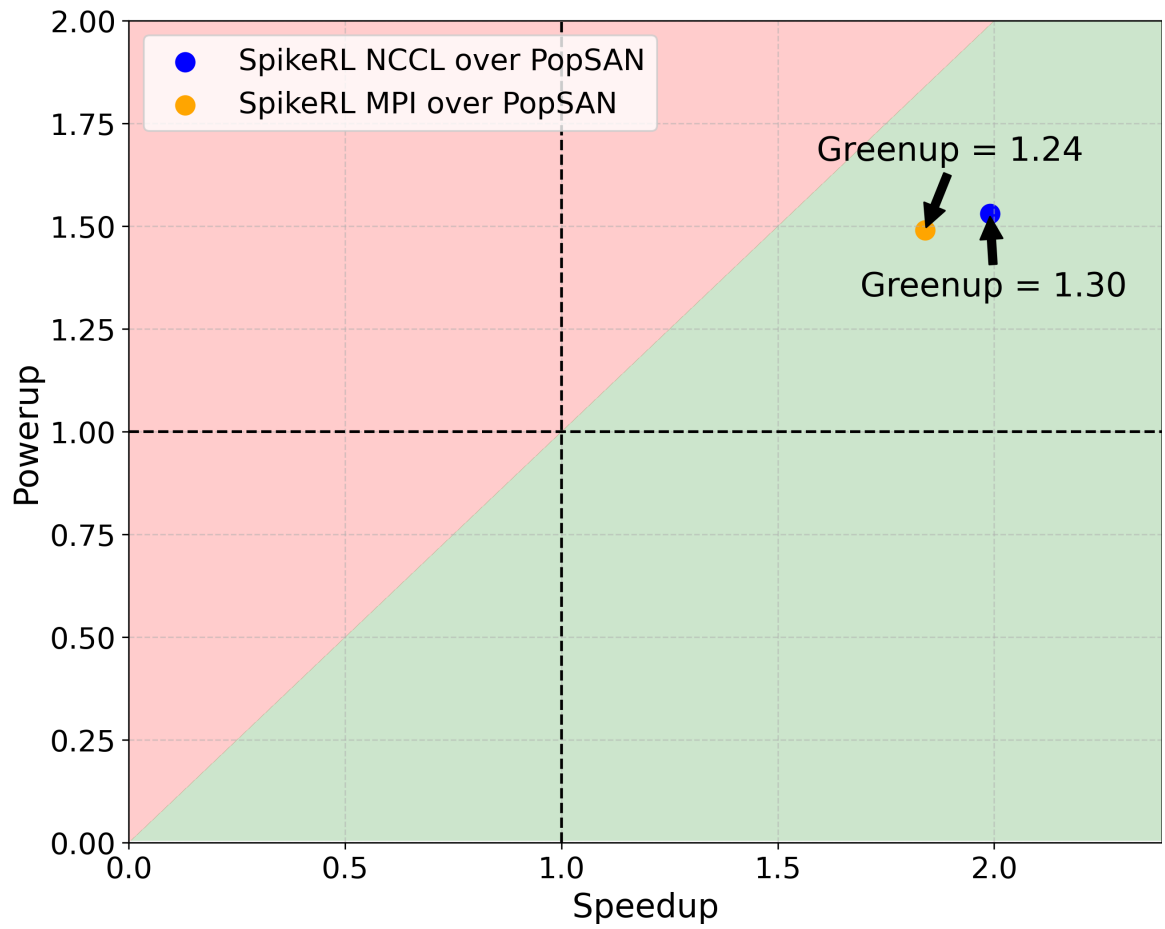


Figure 4.7: Greenup, powerup, and speedup (GPS-UP) quadrant graph for SpikeRL (NCCL and MPI) over PopSAN in the Ant-v4 environment.

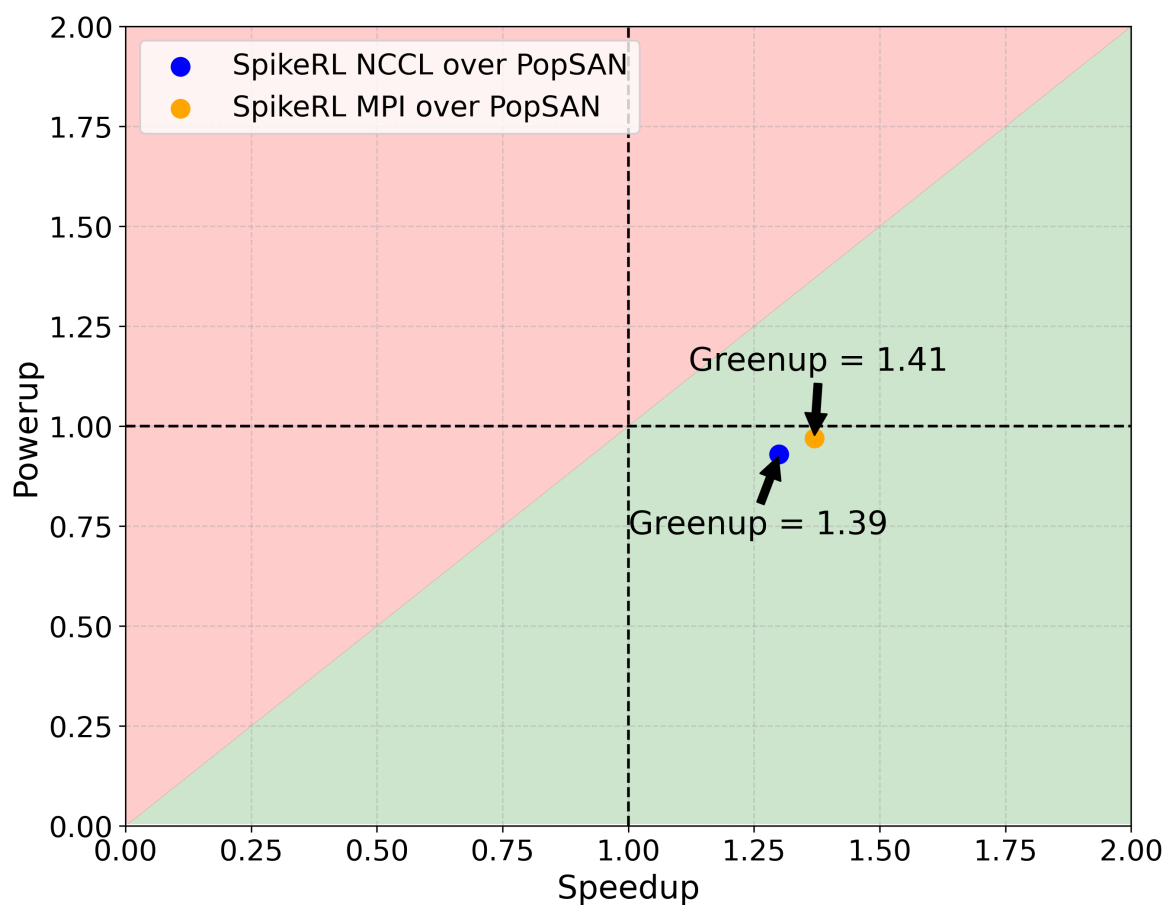


Figure 4.8: Greenup, powerup, and speedup (GPS-UP) quadrant graph for SpikeRL (NCCL and MPI) over PopSAN in the Hopper-v4 environment.

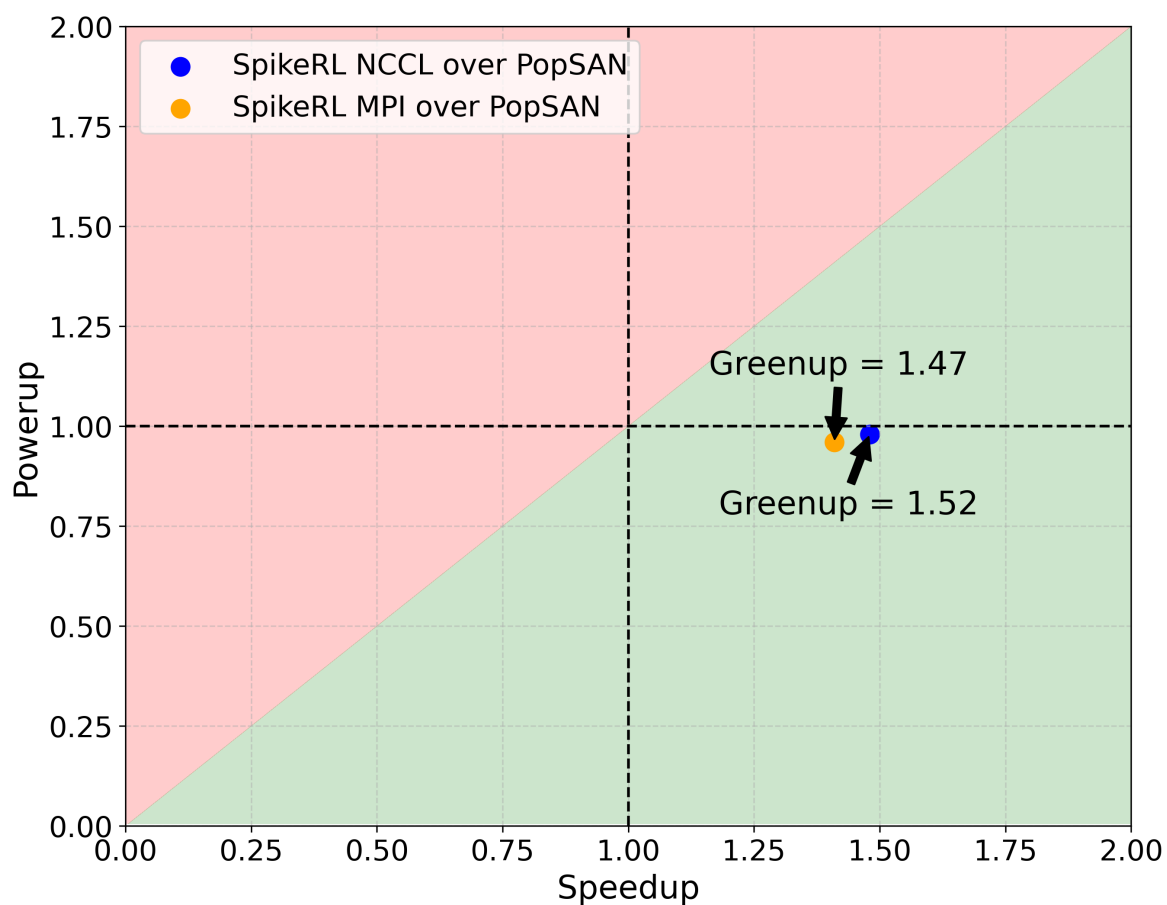


Figure 4.9: Greenup, powerup, and speedup (GPS-UP) quadrant graph for SpikeRL (NCCL and MPI) over PopSAN in the HalfCheetah-v4 environment.

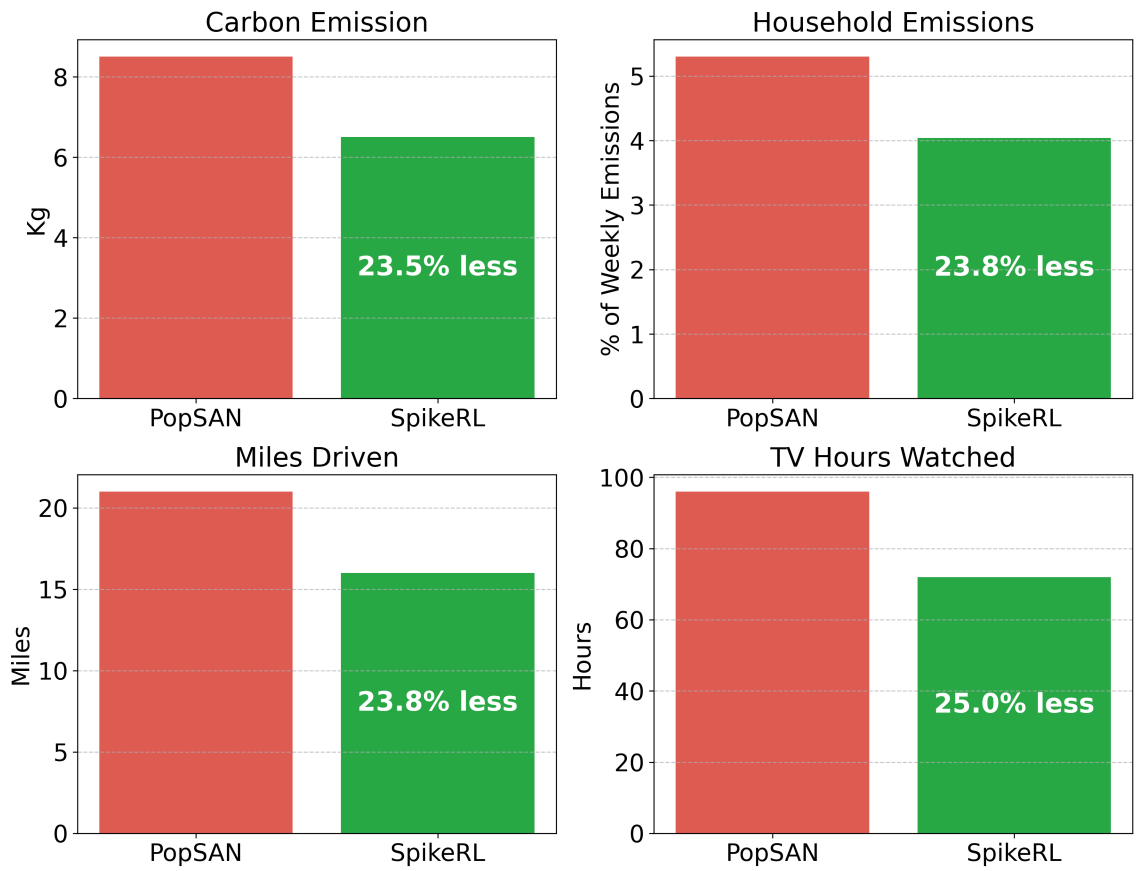


Figure 4.10: Ant-v4 carbon footprint analysis of SpikeRL over PopSAN.

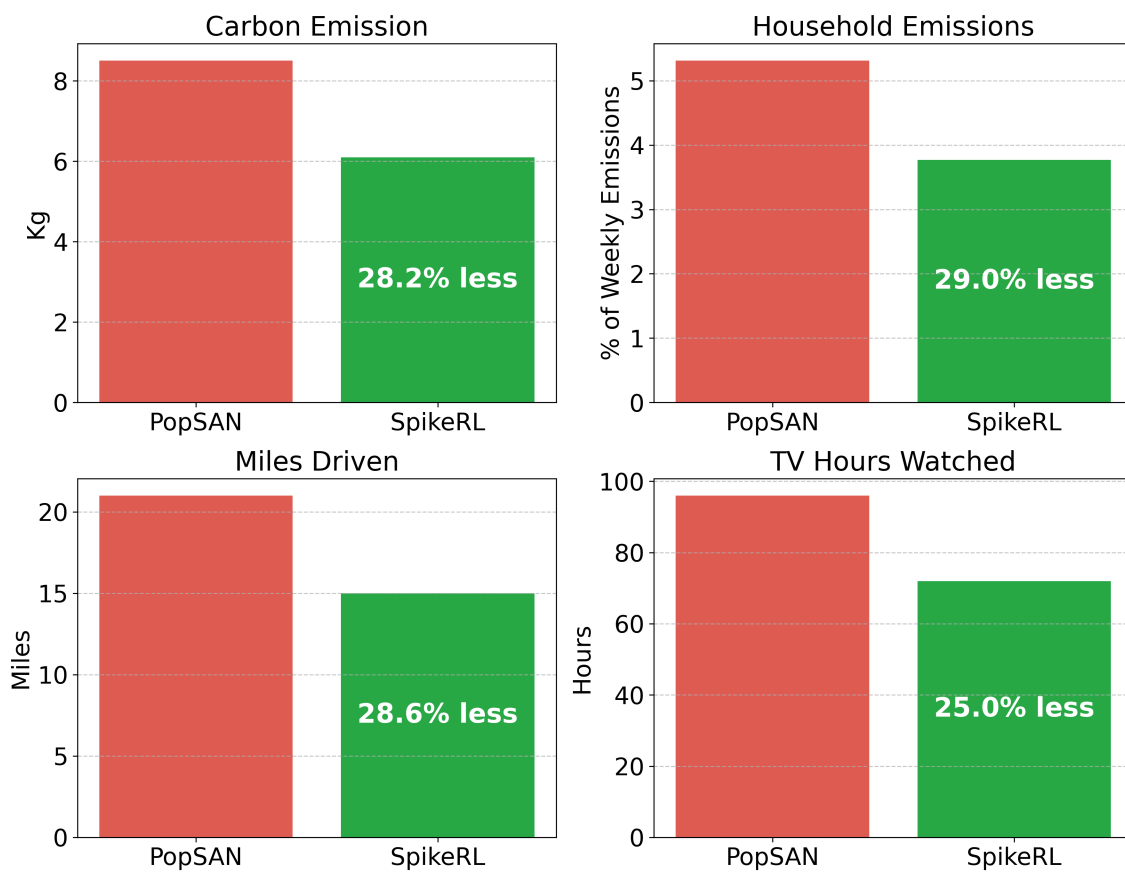


Figure 4.11: Hopper-v4 carbon footprint analysis of SpikeRL over PopSAN.

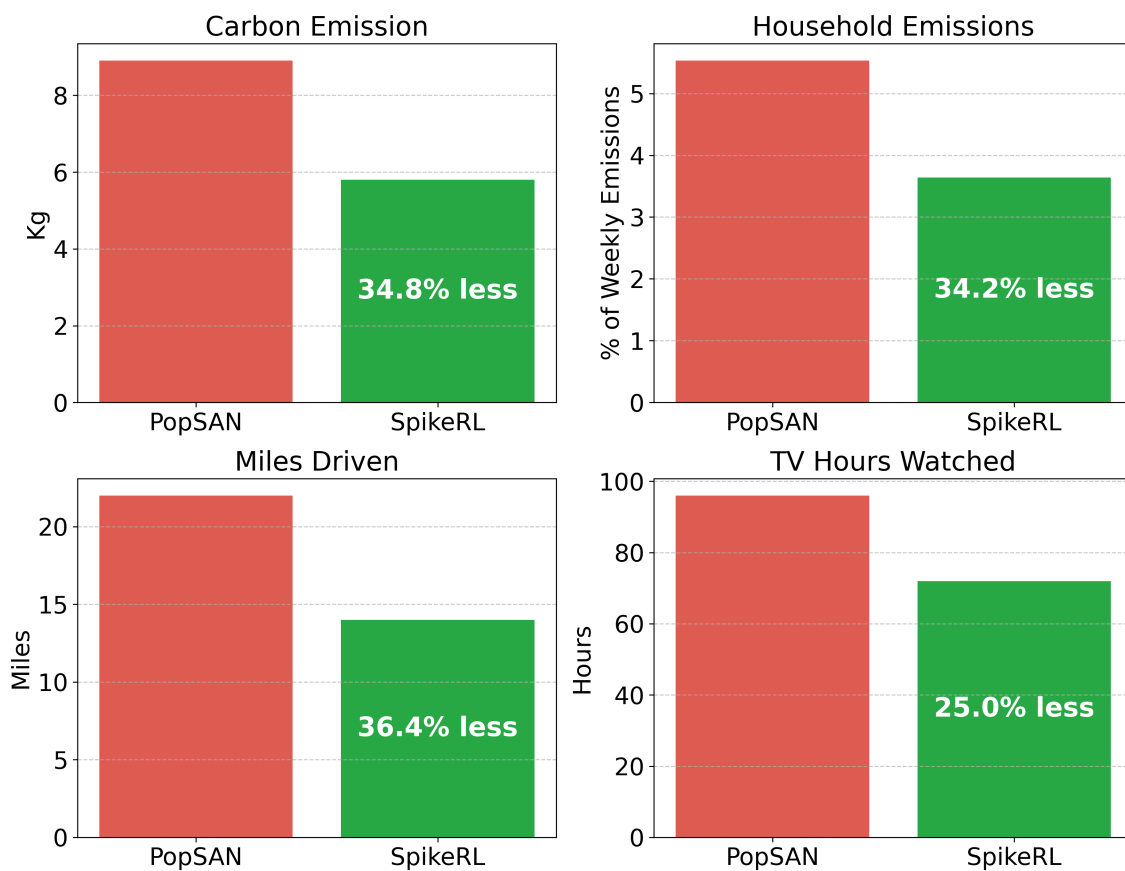


Figure 4.12: HalfCheetah-v4 carbon footprint analysis of SpikeRL over PopSAN.

4.3 Discussion and Analyses

The preliminary experimental results of SpikeRL demonstrated a scalability and efficiency potential of SpikeRL as shown in (1), while the final benchmarks presented in this thesis demonstrated a clear advantage in performance and a superior advantage in energy efficiency over the compared methods.

In terms of performance benchmarks, the results presented demonstrate a significant amount of performance boost over the DSRL model with an average of 78.23% more performance. Comparison with the PopSAN method showed only a moderate amount of performance improvement with an average of 2.22% improved performance. This similarity in performance to PopSAN can be attributed to the population encoding and decoding mechanism which is present in both SpikeRL and PopSAN. Moreover, the performance benchmarks were done by comparing with the best results published in the respective papers of the compared methods.

Hyper-parameter tuning was not performed to an extensive level for SpikeRL to achieve the best results possible and was left for future work.

Although SpikeRL performed somewhat similarly to PopSAN in accumulating rewards, energy efficiency benchmarks demonstrated a superior energy efficiency of SpikeRL over PopSAN. With an average of 1.39x more energy efficiency than PopSAN, SpikeRL managed to reduce the carbon emissions by up to 34.8%. The benchmarks presented in this thesis highlight the energy efficiency and environmental benefits of the SpikeRL framework, positioning it as an energy-efficient solution that maintains scalability and performance while training continuous control agents across various OpenAI Gym environments.

Chapter 5

Conclusion

In this thesis, the SpikeRL framework is introduced, presenting a novel approach that integrates SNNs with DeepRL to create a scalable and energy-efficient alternative to solving complex continuous control problems. This research contributes to the ongoing efforts in the field of sustainable AI by addressing two critical challenges: the high energy consumption associated with large-scale or complex AI models and the scalability limitations of SNNs in continuous control problems.

The SpikeRL framework achieves its objectives through three core components: a DeepRL-based SNN utilizing population encoding, distributed training facilitated by the MPI and NCCL backend in PyTorch distributed package, and optimization via mixed-precision training with BFLOAT16. Each of these components plays a vital role in enhancing the overall performance and energy efficiency of the framework. Preliminary experimental results (1) demonstrated that the SpikeRL framework showed scalability potential compared to traditional DeepRL models and SNNs. This distributed training approach using MPI shows a scalability of up to 8 MPI ranks based on the speedup measurements. Although efficiency decreases for all MPI ranks due to load imbalance and increased communication overhead, the SpikeRL framework still maintains a competitive edge over PopSAN and traditional

DeepRL with improved performance in average rewards. The limitations of this preliminary implementation was addressed in this thesis.

The final performance benchmarks, compared the performance of the SpikeRL framework with the state-of-the-art methods PopSAN and DSRL. In these performance benchmarks, SpikeRL performed on average 40% better than the baseline methods. Moreover, in the final energy-efficiency benchmarks, the SpikeRL framework with both MPI and NCCL backend were compared with the PopSAN method. Since, the PopSAN method demonstrated energy efficiency results compared to other methods in their publication (8), SpikeRL was only compared with PopSAN for the energy efficiency benchmarks. In this case, SpikeRL achieved an average energy efficiency improvement of 39% compared to the state-of-the-art PopSAN method, resulting in a corresponding 28% reduction in carbon emissions. These results validate the efficacy of the mixed precision training techniques and distributed learning strategies implemented within the framework.

The deployment of the SpikeRL model on neuromorphic hardware, specifically the RISP (Reduced Instruction Spiking Processor) at TENNLab was a work in progress. Even though the implementation works, the inference results were not satisfactory to be published in the thesis. Hence, the tuning of this inference implementation along with the hyper-parameter tuning of SpikeRL for maximum performance, is a necessary future work. Additional future work entails performance and energy efficiency benchmarks on more complex continuous control problems such as the Humanoid environment in OpenAI gym. Lastly, the deployment of SpikeRL on a neuromorphic hardware would further solidify the sustainability of the framework in the domain.

In conclusion, the SpikeRL framework sets a new standard for performance and sustainability in AI systems, particularly in scenarios requiring complex continuous control. Ultimately, this research marks a significant step towards achieving scalable and sustainable AI, aligning with global efforts to reduce the environmental impact of the technology.

Bibliography

- [1] T. Tahmid, M. Gates, P. Luszczek, and C. D. Schuman, “Towards scalable and efficient spiking reinforcement learning for continuous control tasks,” in *Proceedings of the International Conference on Neuromorphic Systems (To appear in)*. IEEE, 2024. [xiv](#), [15](#), [23](#), [25](#), [33](#), [35](#), [36](#), [53](#), [54](#)
- [2] W. da Silva Pereira, J. Silva, and T. Tahmid, in *Low Precision for Lower Energy Consumption: Preprint*, 2024, aSCR Workshop on Energy-Efficient Computing for Science ; Conference date: 09-09-2024 Through 12-09-2024. [xiv](#), [14](#)
- [3] J. V. de Oliveira Silva, T. Tahmid, and W. da Silva Pereira, “Low precision and efficient programming languages for sustainable ai: Final report for the summer project of 2024,” National Renewable Energy Laboratory (NREL), Golden, CO (United States), Tech. Rep., 2024. [xiv](#), [14](#), [29](#), [30](#), [31](#)
- [4] Y. I. Alzoubi and A. Mishra, “Green artificial intelligence initiatives: Potentials and challenges,” *Journal of Cleaner Production*, p. 143090, 2024. [1](#)
- [5] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, “A survey of neuromorphic computing and neural networks in hardware,” *arXiv preprint arXiv:1705.06963*, 2017. [1](#)
- [6] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, B. Kay *et al.*, “Opportunities for neuromorphic computing algorithms and applications,” *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, 2022. [2](#), [4](#), [5](#), [6](#)

- [7] 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. [2](#), [7](#)
- [8] G. Tang, N. Kumar, R. Yoo, and K. Michmizos, “Deep reinforcement learning with population-coded spiking neural network for continuous control,” in *Conference on Robot Learning*. PMLR, 2021, pp. 2016–2029. [2](#), [7](#), [20](#), [55](#)
- [9] K. Naya, K. Kutsuzawa, D. Owaki, and M. Hayashibe, “Spiking neural network discovers energy-efficient hexapod motion in deep reinforcement learning,” *IEEE Access*, vol. 9, pp. 150 345–150 354, 2021. [2](#), [8](#)
- [10] D. Zhang, T. Zhang, S. Jia, and B. Xu, “Multi-sacle dynamic coding improved spiking actor network for reinforcement learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 1, 2022, pp. 59–67. [2](#), [8](#)
- [11] G. Liu, W. Deng, X. Xie, L. Huang, and H. Tang, “Human-level control through directly trained deep spiking q-networks,” *IEEE transactions on cybernetics*, vol. 53, no. 11, pp. 7187–7198, 2022. [2](#), [9](#)
- [12] G. Wu, D. Liang, S. Luan, and J. Wang, “Training spiking neural networks for reinforcement learning tasks with temporal coding method,” *Frontiers in Neuroscience*, vol. 16, p. 877701, 2022. [2](#), [9](#)
- [13] D. Vlasov, A. Minnekhanov, R. Rybka, Y. Davydov, A. Sboev, A. Serenko, A. Ilyasov, and V. Demin, “Memristor-based spiking neural network with online reinforcement learning,” *Neural Networks*, vol. 166, pp. 512–523, 2023. [2](#), [10](#)
- [14] L. Zanatta, A. Di Mauro, F. Barchi, A. Bartolini, L. Benini, and A. Acquaviva, “Directly-trained spiking neural networks for deep reinforcement learning: Energy efficient implementation of event-based obstacle avoidance on a neuromorphic accelerator,” *Neurocomputing*, vol. 562, p. 126885, 2023. [2](#), [10](#)

- [15] M. Akl, D. Ergene, F. Walter, and A. Knoll, “Toward robust and scalable deep spiking reinforcement learning,” *Frontiers in Neurorobotics*, vol. 16, p. 1075647, 2023. [2](#), [9](#)
- [16] D. Chen, P. Peng, T. Huang, and Y. Tian, “Fully spiking actor network with intralayer connections for reinforcement learning,” *IEEE Transactions on Neural Networks and Learning Systems*, 2024. [2](#), [11](#)
- [17] S. Feng, J. Cao, Z. Ou, G. Chen, Y. Zhong, Z. Wang, J. Yan, J. Chen, B. Wang, C. Zou *et al.*, “Brainqn: Enhancing the robustness of deep reinforcement learning with spiking neural networks,” *Advanced Intelligent Systems*, vol. 6, no. 9, p. 2400075, 2024. [2](#), [11](#)
- [18] A. P. Georgopoulos, A. B. Schwartz, and R. E. Kettner, “Neuronal population coding of movement direction,” *Science*, vol. 233, no. 4771, pp. 1416–1419, 1986. [2](#)
- [19] D. W. Walker and J. J. Dongarra, “Mpi: a standard message passing interface,” *Supercomputer*, vol. 12, pp. 56–68, 1996. [2](#), [13](#)
- [20] N. NVIDIA, “Nvidia collective communications library (nccl),” 2017. [2](#), [13](#)
- [21] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh *et al.*, “Mixed precision training,” *arXiv preprint arXiv:1710.03740*, 2017. [2](#), [14](#)
- [22] C. Mead, “Neuromorphic electronic systems,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, 1990. [4](#)
- [23] —, “How we created neuromorphic engineering,” *Nature Electronics*, vol. 3, no. 7, pp. 434–435, 2020. [4](#)

- [24] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, “Hardware for machine learning: Challenges and opportunities,” in *2017 IEEE custom integrated circuits conference (CICC)*. IEEE, 2017, pp. 1–8. [5](#)
- [25] C. Mayr, S. Hoepfner, and S. Furber, “Spinnaker 2: a 10 million core processor system for brain simulation and machine learning. arxiv,” *arXiv preprint arXiv:1911.02385*, vol. 10, 2019. [5](#)
- [26] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, “The spinnaker project,” *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014. [5](#), [6](#)
- [27] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, “Loihi: A neuromorphic manycore processor with on-chip learning,” *Ieee Micro*, vol. 38, no. 1, pp. 82–99, 2018. [5](#)
- [28] H. Mostafa, L. K. Müller, and G. Indiveri, “An event-based architecture for solving constraint satisfaction problems,” *Nature communications*, vol. 6, no. 1, p. 8941, 2015. [6](#)
- [29] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza *et al.*, “A low power, fully event-based gesture recognition system,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7243–7252. [6](#)
- [30] B. Cuevas-Arteaga, J. P. Dominguez-Morales, H. Rostro-Gonzalez, A. Espinal, A. F. Jimenez-Fernandez, F. Gomez-Rodriguez, and A. Linares-Barranco, “A spinnaker application: design, implementation and validation of scpgs,” in *Advances in Computational Intelligence: 14th International Work-Conference on Artificial Neural Networks, IWANN 2017, Cadiz, Spain, June 14-16, 2017, Proceedings, Part I 14*. Springer, 2017, pp. 548–559. [6](#)

- [31] B. Strohmer, P. Manoonpong, and L. B. Larsen, “Flexible spiking cpgs for online manipulation during hexapod walking,” *Frontiers in neurorobotics*, vol. 14, p. 41, 2020. [6](#)
- [32] E. Donati, F. Corradi, C. Stefanini, and G. Indiveri, “A spiking implementation of the lamprey’s central pattern generator in neuromorphic vlsi,” in *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*. IEEE, 2014, pp. 512–515. [6](#)
- [33] E. Angelidis, E. Buchholz, J. Arreguit, A. Rougé, T. Stewart, A. von Arnim, A. Knoll, and A. Ijspeert, “A spiking central pattern generator for the control of a simulated lamprey robot running on spinnaker and loihi neuromorphic boards,” *Neuromorphic Computing and Engineering*, vol. 1, no. 1, p. 014005, 2021. [6](#)
- [34] F. Galluppi, J. Conradt, T. Stewart, C. Eliasmith, T. Horiuchi, J. Tapson, B. Tripp, S. Furber, and R. Etienne-Cummings, “Live demo: Spiking ratslam: Rat hippocampus cells in spiking neural hardware,” in *2012 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2012, pp. 91–91. [7](#)
- [35] G. Tang and K. P. Michmizos, “Gridbot: An autonomous robot controlled by a spiking neural network mimicking the brain’s navigational system,” in *Proceedings of the International Conference on Neuromorphic Systems*, 2018, pp. 1–8. [7](#)
- [36] Z. Bing, I. Baumann, Z. Jiang, K. Huang, C. Cai, and A. Knoll, “Supervised learning in snn via reward-modulated spike-timing-dependent plasticity for a target reaching vehicle,” *Frontiers in neurorobotics*, vol. 13, p. 18, 2019. [7](#)
- [37] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596. [20](#), [22](#)

- [38] M. A. Nahmias, B. J. Shastri, A. N. Tait, and P. R. Prucnal, “A leaky integrate-and-fire laser neuron for ultrafast cognitive computing,” *IEEE journal of selected topics in quantum electronics*, vol. 19, no. 5, pp. 1–12, 2013. [21](#)
- [39] T. Lillicrap, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015. [22](#)
- [40] L. Dalcin and Y.-L. L. Fang, “mpi4py: Status update after 12 years of development,” *Computing in Science & Engineering*, vol. 23, no. 4, pp. 47–54, 2021. [23](#)
- [41] N. Corporation, “Mixed precision training,” <https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html>, 2024, accessed: September 15, 2024. [27](#)
- [42] B. Courty, V. Schmidt, S. Luccioni, Goyal-Kamal, MarionCoutarel, B. Feld, J. Lecourt, LiamConnell, A. Saboni, Inimaz, supatomic, M. Léval, L. Blanche, A. Cruveiller, ouminasara, F. Zhao, A. Joshi, A. Bogroff, H. de Lavoreille, N. Laskaris, E. Abati, D. Blank, Z. Wang, A. Catovic, M. Alencon, M. Stęchły, C. Bauer, L. O. N. de Araújo, JPW, and MinervaBooks, “mlco2/codecarbon: v2.4.1,” May 2024. [Online]. Available: <https://doi.org/10.5281/zenodo.11171501> [29](#)
- [43] I. Corporation, “Running average power limit energy reporting,” <https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html>, 2024, accessed: September 15, 2024. [29](#)
- [44] N. Corporation, “Nvidia management library (nvml),” <https://developer.nvidia.com/management-library-nvml>, 2024, accessed: September 15, 2024. [29](#)

- [45] L. Lannelongue, J. Grealey, and M. Inouye, “Green algorithms: quantifying the carbon footprint of computation,” *Advanced science*, vol. 8, no. 12, p. 2100707, 2021. [29](#)
- [46] S. A. Budenny, V. D. Lazarev, N. N. Zakharenko, A. N. Korovin, O. Plosskaya, D. V. Dimitrov, V. Akhripkin, I. Pavlov, I. V. Oseledets, I. S. Barsola *et al.*, “Eco2ai: carbon emissions tracking of machine learning models as the first step towards sustainable ai,” in *Doklady Mathematics*, vol. 106, no. Suppl 1. Springer, 2022, pp. S118–S128. [29](#)
- [47] L. F. W. Anthony, B. Kanding, and R. Selvan, “Carbontracker: Tracking and predicting the carbon footprint of training deep learning models,” *arXiv preprint arXiv:2007.03051*, 2020. [29](#)
- [48] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, “Towards the systematic reporting of the energy and carbon footprints of machine learning,” 2020. [29](#)
- [49] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, “Quantifying the carbon emissions of machine learning,” *arXiv preprint arXiv:1910.09700*, 2019. [30](#)
- [50] T. Trébaol, “Cumulator—a tool to quantify and report the carbon footprint of machine learning computations and communication in academia and healthcare,” 2020. [30](#)
- [51] E. Jauregui, “Pue: The green grid metric for evaluating the energy efficiency in dc (data center). measurement method using the power demand,” in *2011 IEEE 33rd International Telecommunications Energy Conference (INTELEC)*. IEEE, 2011, pp. 1–8. [30](#)
- [52] S. Abdulsalam, Z. Zong, Q. Gu, and M. Qiu, “Using the greenup, powerup, and speedup metrics to evaluate software energy efficiency,” in *2015 Sixth*

- International Green and Sustainable Computing Conference (IGSC)*. IEEE, 2015, pp. 1–8. [31](#)
- [53] K. University of Tennessee, “System overview: Cluster at kpb,” <https://oit.utk.edu/hpsc/isaac-open-enclave-new-kpb/system-overview-cluster-at-kpb/>, 2024, accessed: September 15, 2024. [33](#)
- [54] N. Corporation, “Nvidia tesla v100,” <https://www.nvidia.com/en-gb/data-center/tesla-v100/>, 2024, accessed: September 15, 2024. [33](#)
- [55] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG *et al.*, “Gymnasium: A standard interface for reinforcement learning environments,” *arXiv preprint arXiv:2407.17032*, 2024. [33](#)
- [56] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 5026–5033. [33](#)

Appendix

Appendix A

SpikeRL Code

The source code and additional supplementary material can be found at this
GitHub: <https://github.com/icl-utk-edu/spike-rl>

Vita

Tokey Tahmid is an MsC student in Computer Science at the University of Tennessee, Knoxville. He completed Bachelor's degree in computer science at BRAC University, Bangladesh in December 2021. In January 2023, he started working as a Graduate Research Assistant (GRA) at the Innovative Computing Laboratory (ICL) with the linear algebra group under the supervision of Dr. Piotr Luszczek. In May 2023, started working with TENNLab, UTK, Dr. Catherine Schuman being the primary thesis advisor. His primary research focus is in Artificial Intelligence (AI) and High-performance Computing (HPC).