Efficient Exploration in Edge-Friendly Hyperdimensional Reinforcement Learning

Yang Ni*
yni3@uci.edu
University of California, Irvine
Irvine, California, USA

William Youngwoo Chung*
chungwy1@uci.edu
University of California, Irvine
Irvine, California, USA

Samuel Cho
samuelc7@uci.edu
University of California, Irvine
Irvine, California, USA

Zhuowen Zou
zhuowez1@uci.edu
University of California, Irvine
Irvine, California, USA

Mohsen Imani
m.imani@uci.edu
University of California, Irvine
Irvine, California, USA

ABSTRACT

Integrating deep learning with Reinforcement Learning (RL) results in algorithms that achieve human-like learning in complex yet unknown environments via a process of trial and error. Despite the advancements, the computational costs associated with deep learning become a major drawback. This paper proposes a revamped Q-learning algorithm powered by Hyperdimensional Computing (HDC), targeting more efficient and adaptive exploration. We introduce a solution leveraging model uncertainty to navigate agent exploration. Our evaluation shows that the proposed algorithm is a significant enhancement in learning quality and efficiency compared to previous HDC-based algorithms, achieving more than 330 more rewards with small overheads in computation. In addition, it maintains an edge over DNN-based alternatives by ensuring reduced runtime costs and improved policy learning, achieving up to 6.9× faster learning.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Intelligent agents**; • **Computer systems organization** → *Embedded systems.*

KEYWORDS

Reinforcement Learning, Agent Exploration, Hyperdimensional Computing, Brain-inspired Computing

ACM Reference Format:
Yang Ni*, William Youngwoo Chung*, Samuel Cho, Zhuowen Zou, and Mohsen Imani. 2024. Efficient Exploration in Edge-Friendly Hyperdimensional Reinforcement Learning. In *Great Lakes Symposium on VLSI 2024 (GLSVLSI '24), June 12–14, 2024, Clearwater, FL, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3649476.3658760

GLSVLSI '24, June 12–14, 2024, Clearwater, FL, USA

ACM ISBN 979-8-4007-0605-9/24/06
https://doi.org/10.1145/3649476.3658760

1 INTRODUCTION

Smart systems on edge operate in constantly changing environments with diverse needs, where intelligent algorithms are required to ensure optimal decision-making without explicit prior knowledge [14]. RL is one such algorithm that focuses on solving sequential decision-making problems, where the agent learns through trial and error, similar to human beings. RL agents utilize the feedback from the environment to improve their policies and maximize the accumulated returned rewards. The combination of deep learning and RL has led to a wide range of algorithms capable of learning in environments with complex action and state spaces. Take the Deep-Q-Network (DQN) as an example, the unscalable Q-table in traditional algorithms [43] is replaced with a Deep Neural Network (DNN) [24]. However, utilizing DNN results in high computational costs that are less friendly to resource-limited devices. In fact, RL agents are frequently deployed in the edge environment for tasks like smart transportation and healthcare [11, 19].

Recent work has proposed a more efficient and hardware-friendly HDC-based RL algorithm [27]. HDC is motivated by how brains represent information using neural activities in large dimensions [17]. At the functional level, it achieves human-like memorization and reasoning via operations on high-dimensional vectors in the dimension of several thousand, i.e., hypervectors [15]. More specifically, HDC-based Q-learning algorithms represent the state and Q-function using hypervectors, and computing the Q-value for a state-action pair becomes a lightweight similarity check between hypervectors. Compared to DNN, HDC-based RL algorithms achieve significantly faster learning, higher rewards, and better efficiency when implemented with a limited computing budget [27].

However, as learning and interaction are highly intertwined, it is vital for RL algorithms to properly handle the exploration-exploitation dilemma [3]. During the interaction, the RL agent can either follow current knowledge to maximize the immediate reward or explore unknown states and actions that are temporarily sub-optimal but possibly informative to reach a better solution. Prior HDC-based Q-learning applies a common technique called ϵ-greedy to encourage exploration, which occasionally forces the agent to take random actions [27]. However, such a random and naive exploration is undirected and will inevitably compromise the learning efficiency.

In this paper, we redesign the HDC-based Q-learning algorithm for more efficient and adaptive exploration, which is capable of

*These authors contributed equally to this work.

reaching a better decision-making policy with fewer interactions given the same environment. The main contributions of the paper are listed as follows:

- We highlight the challenge faced by agents with prior hyperdimensional RL algorithms. They lack the stochasticity for effective exploration when dealing with an environment that either gives highly sparse feedback or hides the optimal state with long trajectories. Even with dithering techniques, we find that prior HDC-based Q-learning can easily get stuck in suboptimal policies if not completely lost in the sparse reward space.
- The proposed algorithm leverages the uncertainty from the model distribution to guide agent exploration. Instead of going through the bulky computation of the model posterior, our algorithm is composed of several HDC sub-models, i.e., forming an ensemble. This amounts to a stochastic model that encourages deep exploration in RL.
- We specially designed the algorithm to ensure sub-model diversity during training. Compared to prior HDC algorithms with zeroed initial models, we propose to use random and standalone prior hypervectors coupled with random model initialization to help diversify sub-models. This is effectively having HDC sub-models with different prior knowledge about the environment. Therefore, HDC sub-models learn different aspects of the space and naturally show uncertainty when facing unseen samples.
- Our algorithms are optimized towards low-power hardware platforms while achieving significantly better learning quality and efficiency, thanks to the proposed exploration mechanism. Due to its lightweight HDC backbone, our algorithm also outperforms the DNN-based counterparts with various exploration techniques, showing notable improvements in the runtime costs (up to 6.9× faster) and learned policy (over 800 higher rewards for CartPole).

2 HYPERDIMENSIONAL Q-LEARNING

2.1 Overview of Reinforcement Learning

The RL algorithm essentially tells the agent which actions to take given the current state of the environment and its forecast of future rewards. Therefore, we can view the interaction in RL as a loop-like structure: (1) Observing the state $\vec{s}_t$ at time step t, the agent takes an action a_t guided by the RL policy. (2) The environment updates its state to $\vec{s}_{t+1}$ after the agent acts. (3) The agent receives a feedback reward r_t from the environment and possibly updates its policy. (4) Loop until the end of the current trajectory. We summarize each step of the interaction by an experience tuple $(\vec{s}_t, a_t, r_t, \vec{s}_{t+1})$, which is then used for training RL agents.

In this paper, our focus is on value-based Q-learning, which learns a Q-function $Q(\vec{s}_t, a_t)$ to evaluate how profitable is to take action a_t in the state of $\vec{s}_t$. In practice, the value of this function predicts the expected accumulated future rewards after time step t, so that a greedy policy can be applied to select an action with the largest Q-value. Note that the RL policy can also be directly parameterized and learned, however, it is well known that value-based methods are intrinsically more sample-efficient than policy-based RL methods. This is mainly because the Q-function can be trained using off-policy samples, that is, any experience tuples from past trajectories.

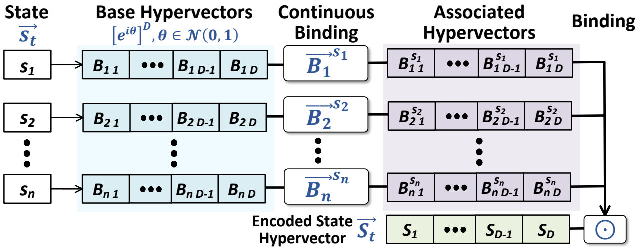
State
$\vec{s_t}$
Base Hypervectors
$[e^{i\theta}]^D, \theta \in \mathcal{N}(0,1)$
Continuous Binding
Associated Hypervectors
Binding
s_1
s_2
s_n
$B_{1\,1}$ ••• $B_{1\,D-1}$ $B_{1\,D}$
$B_{2\,1}$ ••• $B_{2\,D-1}$ $B_{2\,D}$
$B_{n\,1}$ ••• $B_{n\,D-1}$ $B_{n\,D}$
$\vec{B_1}^{s_1}$
$\vec{B_2}^{s_2}$
$\vec{B_n}^{s_n}$
$B_{1\,1}^{s_1}$ ••• $B_{1\,D-1}^{s_1}$ $B_{1\,D}^{s_1}$
$B_{2\,1}^{s_2}$ ••• $B_{2\,D-1}^{s_2}$ $B_{2\,D}^{s_2}$
$B_{n\,1}^{s_n}$ ••• $B_{n\,D-1}^{s_n}$ $B_{n\,D}^{s_n}$
Encoded State Hypervector $\vec{S_t}$
S_1 ••• S_{D-1} S_D
$\odot$

Figure 1: Overview of HDC encoder with continuous binding.

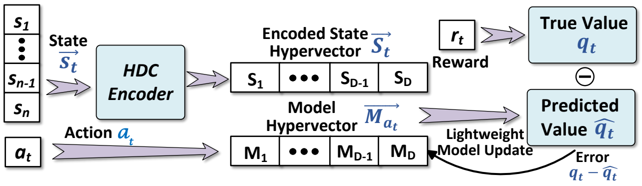
s_1
s_{n-1}
s_n
State
$\vec{s_t}$
HDC Encoder
Encoded State Hypervector $\vec{S_t}$
S_1 ••• S_{D-1} S_D
r_t
Reward
True Value
q_t
$\ominus$
Predicted Value $\widehat{q_t}$
Model Hypervector $\vec{M_{a_t}}$
a_t
Action a_t
M_1 ••• M_{D-1} M_D
Lightweight Model Update
Error
$q_t - \widehat{q_t}$

Figure 2: Model hypervector training in HDC-based RL.

2.2 Hypervector Encoding

In HDC, the encoder projects original inputs to a high-dimensional space, or hyperspace. This allows information to be stored holistically and distributed evenly in each element of the hypervector, giving HDC robustness against hardware noise. As shown in Figure 1, we assume that the input to the encoder is an agent state vector $\vec{s}_t \in \mathbb{R}^n$ at a specific time step t. The HDC encoder is composed of n base hypervectors $\{\vec{B}_1, \vec{B}_2, \ldots, \vec{B}_n\}$, and the encoding is carried out independently for each element of $\vec{s}_t$. These base hypervectors are generated by randomly sampling from a distribution. It is not hard to find that they are near-orthogonal to each other: $\vec{B}_1 \cdot \vec{B}_2 \approx 0$. In prior HDC works, a wide range of random distributions have been selected for encoding [4, 4, 17]. For HDC-based Q-learning, the base hypervectors are comprised of random phasors following prior work [27]: $\vec{B} \in [e^{i\theta}]^D$, where $\theta \in \mathcal{N}(0,1)$ and the hypervector dimensionality $D \gg n$.

Figure 1 is an overview of the encoding process. To encode the state vector $\vec{s}_t$, we apply ***Continuous Binding*** to represent continuous element values of $\vec{s}_t : \{s_1, s_2, \ldots, s_n\}$ in the hyperspace, i.e., element-wise exponential of base hypervectors with the exponent being the value to associate. More specifically, we compute the encoded state hypervector: $\vec{S}_t = \vec{B}_1^{s_1} \odot \vec{B}_2^{s_2} \odot \cdots \odot \vec{B}_n^{s_n}$. Here the encoder uses a hypervector operation called ***Binding*** ($\odot$), i.e., element-wise multiplication, to form a single hypervector that represents the whole state vector.

2.3 Reinforcement Learning with HDC

In both DQN and HDC-based Q-learning, we will maintain an experience replay buffer to save experience tuples generated during the interaction. The major difference between these two algorithms is that the Q-function is abstracted using hypervectors instead of DNN, leading to changes in the agent decision-making process and model training. For example, the DQN training process and parameter update are based on back-propagation, whereas QHD utilizes lightweight hypervector memorization.

In HDC-based Q-learning, the model is comprised of several hypervectors $Q = \{\vec{M}_1, \vec{M}_2, \dots, \vec{M}_m\}$, each corresponds to one of the m possible actions and has the same dimensionality D as the encoded state. For example, to select an action with a greedy policy, we predict the Q-value through the dot-product similarity between the encoded state hypervector and the model hypervector corresponding to the selected action:

$$a_t = \operatorname*{argmax}_{a \in \mathcal{A}} Q(\vec{S}_t, a) = \operatorname*{argmax}_{a \in \mathcal{A}} \mathrm{real}(\vec{S}_t \cdot \vec{M}_a^{\dagger} / D) \tag{1}$$

where $\mathcal{A}$ is the action space, D is the normalization factor, $Q(\cdot)$ represent the Q-function, and $\vec{M}_a^{\dagger}$ is the conjugate vector. In HDC, computing the dot product of two hypervectors is known as the ***Similarity Check***. HDC-based Q-learning leverages the similarity values between state and model hypervectors to predict Q-values. Note that since two concerning vectors are complex-valued, we take the real part of the result.

In Figure 2, we provide the outline of HDC-based Q-learning. The training begins by randomly sampling an experience tuple from the experience replay buffer, e.g., $(\vec{s}_t, a_t, r_t, \vec{s}_{t+1})$. The algorithm first encodes the state of the current time step and the next one to state hypervectors $\vec{S}_t$ and $\vec{S}_{t+1}$, respectively. Then we can predict the Q-value for the state-action pair $\{\vec{s}_t, a_t\}$:

$$\hat{q_t} = Q(\vec{S}_t, a_t) = \mathrm{real}(\vec{S}_t \cdot \vec{M}_{a_t}^{\dagger} / D) \tag{2}$$

The target Q-value q_t is derived from the Bellman optimality equation as the following:

$$q_t = \begin{cases} r_t & t \text{ is the last step} \\ r_t + \gamma \max\limits_a \bar{Q}(\vec{S}_{t+1}, a) & t \text{ is not the last step} \end{cases} \tag{3}$$

It is computed as the sum of immediate reward r_t and the greedy prediction of the future accumulated rewards. As proposed in Double Q-learning [40], the Q-value for the next time step is provided by a delayed model $\bar{Q}$, which will be updated less frequently by copying the model hypervectors from the model Q every few training steps. This helps stabilize the training and reduce the overestimation of Q-values when taking the maximum. γ is the reward discount factor that decides how short-sighted is the agent. The model update is based on the off-policy TD error, i.e., $q_t - \hat{q}_t$. The model hypervector corresponding to the chosen action a_t will be updated:

$$\vec{M}_{a_{t+1}} = \vec{M}_{a_t} + \eta(q_t - \hat{q_t})\vec{S}_t \tag{4}$$

where η is the learning rate. Depending on the sign of the TD error, a weighted hypervector $\vec{S}_t$ is added to or subtracted from the model hypervector, so a higher error results in a more aggressive update.

As shown in Equation 4, the model hypervectors $\vec{M}_a$ are learned to be a weighted combination of encoded state hypervectors, which is usually referred to as hypervector ***Bundling*** in HDC. This allows HDC to distinguish itself from DNNs used in DQN. Models in HDC-based algorithms serve as memorization components that explicitly record past experienced states $\vec{S}$ and their rewards q. Therefore, similarities in Equation 2 naturally gives Q-value predictions.

After training on a batch of prior experience samples, the updated agent continues exploring the environment and collects new samples for the experience replay buffer.

3 BALANCE BETWEEN EXPLORATION AND EXPLOITATION IN HDC-BASED RL

If we directly apply the HDC-based Q-learning introduced in the previous section, it will not return satisfying results in most cases. The main problem, also the focus of this work, lies in the Equation 1. This equation stands for a purely greedy policy that always exploits the current knowledge of the agent to choose the best possible action. However, in the early phases of RL, the agent has very limited knowledge about the environment, which cannot effectively support a high-quality policy. As we mentioned in Section 1, ϵ-greedy mitigates this issue by letting the agent randomly explore the space with a decaying probability. This is based on a simple notion that the need for exploration is greater in the beginning.

However, in our algorithm, we argue that the HDC-based agent should choose to explore only when the agent is uncertain about the current situation; this effectively de-correlates the exploration from the number of training iterations, making it more flexible on tasks with different levels of complexity. In order to find out the confidence of the agent, we apply ensemble learning based on HDC-based Q-learning.

3.1 Stochastic Policy via HDC Sub-models

The greedy policy in Q-learning is deterministic, yet it is desired to obtain a stochastic policy for exploration. A possible solution is to have a posterior distribution of the Q-function. By randomly sampling a Q-value from the distribution, the policy becomes stochastic. As the posterior is closely connected to the confidence of the agent, the agent is prone to exploration when it faces uncertain states due to a wider Q-value distribution, and it automatically prioritizes exploitation when the confidence is high. However, directly evaluating the posterior in practice is highly inefficient.

With several HDC sub-models, our algorithm approximates the posterior distribution by randomly choosing one sub-model to guide the agent. Assume we have k different sub-models $\{Q_1, Q_2, \dots, Q_k\}$. For efficiency purposes, we only use one of them to actually guide the agent interaction, different from typical ensemble learning. Periodically, we perform a random selection from k HDC sub-models such that the agent follows a stochastic policy. When the agent is unfamiliar with the surrounding environment, this uncertainty is reflected through disagreements between HDC sub-models. Therefore, sampling from sub-models effectively encourages agent exploration for less frequently visited states, as each sub-model will be likely to choose a different action.

The frequency of sub-model re-selection can be tuned depending on the depth of exploration needed. When the agent re-selects a sub-model for every step of interaction, our algorithm is an efficient approximation of Thompson sampling without explicit posterior. Thompson sampling has been shown to greatly boost agent exploration in the multi-armed bandit problem [1]. It is suitable for tasks that need shallow or local exploration. In contrast, when a sub-model is re-sampled every episode, it is capable of deep exploration. In this setting, the agent proactively looks for long-shot rewards and informative states even if it means taking many sub-optimal actions. We will show more details in Section 4.2.

Figure 3 shows the overall structure of our proposed design. All HDC sub-models share the same hyperdimensional encoder, leading

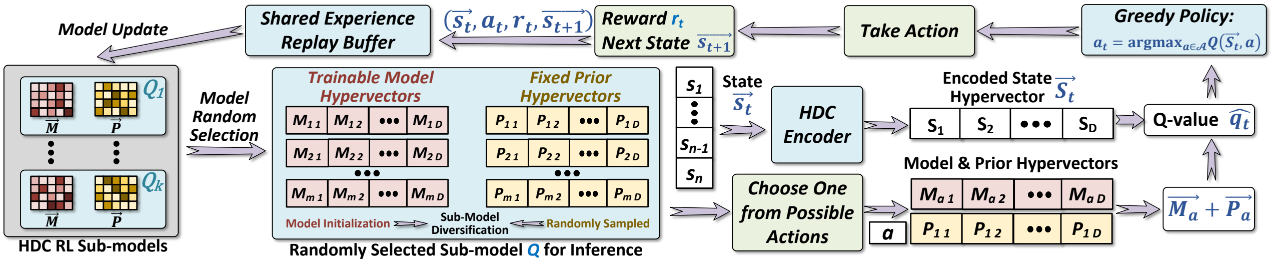
Model Update
Shared Experience Replay Buffer
$(\vec{s_t}, a_t, r_t, \vec{s_{t+1}})$
Reward r_t
Next State $\vec{s_{t+1}}$
Take Action
Greedy Policy:
$a_t = \text{argmax}_{a\in\mathcal{A}} Q(\vec{S_t}, a)$
Q_1
$\vec{M}$
$\vec{P}$
Q_k
$\vec{M}$
$\vec{P}$
HDC RL Sub-models
Model Random Selection
Trainable Model Hypervectors
Fixed Prior Hypervectors
M_{11} M_{12} ••• M_{1D}
M_{21} M_{22} ••• M_{2D}
M_{m1} M_{m2} ••• M_{mD}
P_{11} P_{12} ••• P_{1D}
P_{21} P_{22} ••• P_{2D}
P_{m1} P_{m2} ••• P_{mD}
Model Initialization
Sub-Model Diversification
Randomly Sampled
Randomly Selected Sub-model Q for Inference
s_1
s_{n-1}
s_n
State $\vec{S_t}$
HDC Encoder
Encoded State Hypervector $\vec{S_t}$
S_1 S_2 ••• S_D
Q-value $\hat{q_t}$
Model & Prior Hypervectors
Choose One from Possible Actions
a
M_{a1} M_{a2} ••• M_{aD}
P_{11} P_{12} ••• P_{1D}
$\vec{M_a} + \vec{P_a}$

Figure 3: HDC-based Q-learning Exploration with Sub-model Ensemble.

to an efficient implementation. Unifying the encoding process will not compromise the stochastic policy, since the HDC encoder does not contain trainable parameters. During training, HDC sub-models will sample past experience tuples from a shared experience replay buffer. Thus, tuples collected by individual sub-models are shared with each other to improve the utilization of training samples. This ensures any informative or highly rewarding state will be used to train all sub-models.

3.2 Diversification in HDC Ensemble Learning

In Figure 3, we have extra hypervectors $\vec{P}$ in each sub-model Q, besides the regular model hypervectors $\vec{M}$. As a crucial part of our design, they help keep sub-models different from each other. To encourage exploration, these sub-models should obtain a diverse understanding of the environment. Highly similar models will defeat the purpose of an ensemble and lead to deterministic policies.

However, training sub-models via hypervector memorization (Equation 4) leads to the high similarity between those models, given that the training samples are shared among them. In prior works, HDC-based models generally show much faster convergence than other ML models even though they usually require no model initialization. For sub-model diversification, however, a fast model convergence and zero initialization are not ideal.

Motivated by the Bayesian inference, we include the prior distribution to the HDC learning. Similar to how we approximate the posterior via multiple HDC sub-models, the prior is represented by a series of randomly sampled hypervectors loaded before any training. Now in each sub-model Q, there are additional m hypervectors $\{\vec{P}_1, \vec{P}_2, \ldots, \vec{P}_m\}$. These hypervectors are not updated during training and are isolated from the model hypervectors during Q-value computation.

We sample the prior hypervectors with dimensionality D from a zero-mean Gaussian distribution: $\vec{P}_1, \vec{P}_2, \ldots, \vec{P}_m \in [\mathcal{N}(0, \sigma^2)]^D$, where σ is the standard deviation and a hyperparameter in our algorithm. We will then modify the inference process (Equation 1) and $Q(\cdot)$ accordingly:

$$a_t = \underset{a\in\mathcal{A}}{\text{argmax}}\, Q(\vec{S}_t, a) = \underset{a\in\mathcal{A}}{\text{argmax}}\, \frac{\text{real}\left(\vec{S}_t \cdot (\vec{M}_a^\dagger + \vec{P}_a^\dagger)\right)}{D} \qquad (5)$$

In this equation, we first bundle the model hypervector and the prior hypervector of the same action, and then the dot product similarity is calculated between the encoded state and the bundled hypervector. Q-value computation in Equation 2 and 3 are also changed accordingly. As for training, we still use Equation 4 to update only the model hypervectors and leave the prior untouched.

Table 1: Runtime speedup on embedded CPU with different proposed optimization techniques.

Proposed Optimization	Pre-Encoding	Batching	Mask	Stacking	Full Optimization
Speedup Over Non-optimized	1.31×	1.44×	1.82×	4.56×	5.95×

Adding prior distribution will not introduce bias in Q-value prediction because sufficient training will eventually cancel out the fixed prior hypervector. More importantly, when the agent reaches a novel state, the lack of training means that prior hypervectors will play a significant role in the Q-value prediction. Suppose in the extreme case that two sub-models Q_1 and Q_2 are never trained and will be tested on a state-action pair $\{\vec{S}, a\}$: the model hypervectors can be ignored if they are initialized with zero, and the prior hypervectors $\vec{P}_a^1$ and $\vec{P}_a^2$ are randomly generated. The first sub-model will predict $\hat{q}_1 = \text{real}(\vec{S} \cdot \vec{P}_a^{1\dagger}/D)$, and for the second one will give $\hat{q}_2 = \text{real}(\vec{S} \cdot \vec{P}_a^{2\dagger}/D)$. It is not hard to observe that they have diverse predictions on Q-values due to different prior knowledge, which naturally leads to exploration in unseen states.

To further encourage sub-model diversity, we also apply two similar techniques as in the prior work [32]: random initialization for model hypervectors and bootstrapped sampling for HDC sub-model training. For the first technique, we randomly sample model hypervectors using $\mathcal{N}(0, 1)$, such that the differences in sub-models Q are from both $\vec{M}$ and $\vec{P}$. As for the second trick, we give a binary mask of length k for each experience tuple in the experience buffer: $\vec{m} \in \{0, 1\}^k$. For example, $m_3 = 1$ means that this tuple will be used to train the third sub-model. As the mask is not one-hot, a particular tuple can be used for multiple sub-models. The mask elements will be sampled independently from a Bernoulli distribution with probability p. A probability of less than 1 makes sub-models train on slightly different samples.

3.3 Edge-Friendly HDC-based RL

Various techniques have been applied to our model for efficient learning on embedded systems. By leveraging unique properties of HDC, we can initialize a single tensor to efficiently represent a group of model hypervectors. This allows us to train the models using the same number of operations as training a single QHD model, significantly decreasing the computational overhead of training multiple models. We refer to this technique as "Stacking" in Table 1 and report a 4.56x speedup.

Prior work [32] experiments with using a mask to implement an online bootstrap by masking each episode of data. It was reported to show little to no benefits to performance yet required more iterations to converge. However, we adjust the bootstrap to mask which models should be trained per time step t instead of per episode. This allows models to be trained during any given episode, reducing the possibility of the selected exploration model not being trained for an entire episode, which prohibited the benefit of deep exploration. We note that the result of our masking method not only reduces the runtime by 1.82×, but also increases the amount of rewards obtained in Figure 8 (a).

Other optimizations we introduce are pre-encoding and batching. Prior HDC-based RL work [16, 27, 29] encodes the state into high-dimensional vectors for both inference and training. Since the state encoding matrix remains fixed, this allows us to re-use the same state hypervectors for inference and training and to disregard the need for encoding the state twice. Training samples can also be efficiently batched to minimize the number of complex high-dimensional encoding to one per training step. This has no effect on the performance of the algorithm as we only batch the encoding processes of the state vectors and not the training directly. These approaches can be easily applied to existing HDC-based RL methods for accelerated learning on embedded systems. We notice a 1.33× and 1.44× speedup respectively.

Additionally, we combine all optimization methods and make slight modifications to the QHD framework to make it more memory efficient and compatible with learning on the edge and combine all optimizations, resulting in 5.95× speedup over a naive implementation of training on embedded CPUs.

4 EXPERIMENTAL RESULT

4.1 Experiment Settings

Our implementation is based on the low-power laptop CPU Intel Core-i5-8259U and an embedded ARM CPU on the Raspberry Pi platform (with 6w TDP) using Python and the Pytorch framework. The RL environments used are based on OpenAI Gym [6]. We increase the task difficulty by making the rewards sparser (assign reward only after reaching the goal) and each episode shorter (half the number of steps in LunarLander and Acrobot). We compare the proposed design against several state-of-the-art baselines including QHD [27], Double DQN (DDQN) [40], Double DQN with Thompson Sampling (TDQN) [1], and Bootstrapped DQN (BDQN) [32]. By default, we use hypervectors with $D = 6000$ for QHD as described in the original paper. However, for our design with ensemble sub-models, we set $D = 2000$, as we find that it balances between learning quality and runtime cost. The backbone of all DQN-based methods is a neural network with two hidden layers. The first layer has 128 neurons, and the second one has 256; this gives a similar computation to a single QHD model at $D = 6000$ during inference. We record multiple trials when evaluating our design and other baselines, and we provide the moving average values of 20 episodes for reward and runtime. For several aforementioned hyperparameters, we set the default probability p for the bootstrapping mask to 1, and $\sigma = 5$ for sampling prior hypervectors. Our ablation study in Section 4.4 explores the effect of these hyperparameters. In the following sections, for HDC-based algorithms, TQHD stands for QHD with Thompson sampling (i.e., the HDC sub-model is re-selected for each interaction), and BQHD is for the setting where the same sub-model is used until another trajectory starts.

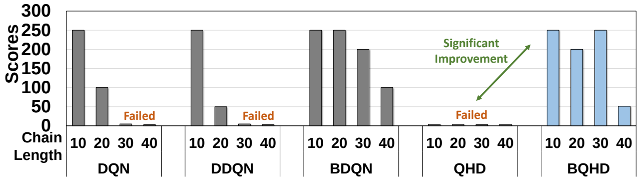

Figure 4: Accumulated scores/rewards in Chain environment.

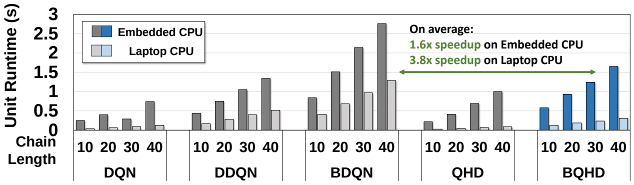

Figure 5: Unit runtime in Chain environment.

4.2 Deep Exploration with HDC-based Q-learning

Figure 4 and Figure 5 illustrate the rewards and unit runtimes of various RL algorithms in a Chain environment, which is designed to highlight the importance of thorough, deep exploration. These experiments involve simulated environments with chains of length $N \geq 10$ and each episode lasting $N + 9$ discrete steps. Additionally, this environment has also been modified such that the reward is 0.001 when the agent is in the left-most state but reward is 1 when the agent is in the right-most state. The simulation is finished if the agent obtains the optimal +10 rewards 50 times or runs over 2000 episodes. There is a fundamental trade-off between employing a well-established, moderately successful strategy and experimenting an unfamiliar yet potentially more lucrative approach, favoring algorithms that drive deep exploration. As shown in Figure 4, bootstrapping drives efficient exploration and is evident through the differences in solved chain lengths and cumulative rewards compared to non-bootstrapped architectures. BQHD significantly improves upon QHD in deep exploration as QHD is not able to learn the optimal policy at any given chain length. Moreover, Figure 5 shows that BQHD performs similar exploration as BDQN while significantly reducing the runtime per episode.

4.3 Learning Efficiency and Quality Comparison of Various RL Algorithms

Figure 6 shows the learning curves in the Cartpole task of different RL algorithms with or without adaptive exploration. Consistent with the prior work [27], we observe that HDC-based algorithms outperform the DQN-based ones in terms of achieved rewards

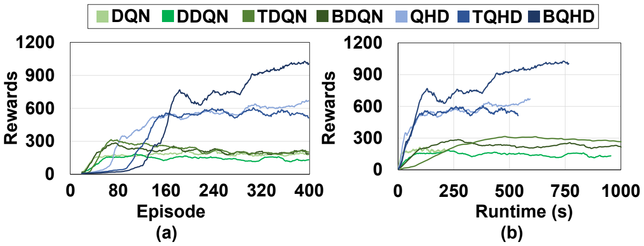

Figure 6: Cartpole learning curves comparison in terms of (a) episodes and (b) runtime. Note that the runtime figure is clipped at 1000 seconds.

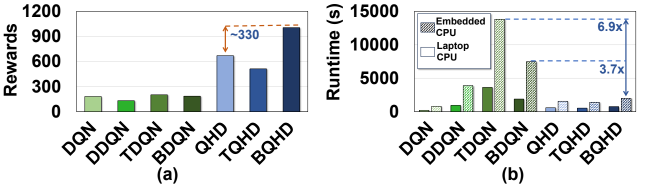

Figure 7: In Cartpole, compare RL algorithms at 400 learning episodes in terms of: (a) achieved average rewards and (b) required total runtime.

also runtime efficiency. More importantly, the proposed method BQHD is able to further enhance that benefit by a notable margin, achieving over 1000 rewards at 400 episodes. The result indicates that adaptive exploration guided by the model uncertainty helps the agent discover a much more optimal policy. Figure 7(a) shows that our proposed BQHD is able to achieve 330 rewards higher than the baseline QHD. Thompson sampling is not performing as well mainly due to its inability of deep exploration. In Figure 7(b), we compare the required total runtime for 400 episodes in both laptop and embedded CPU platforms. The results show that TDQN and BDQN, due to having multiple DNNs, require a much longer learning runtime (e.g., TDQN needs nearly 3800 seconds on the laptop CPU and 13800 seconds on the embedded CPU); they are less desirable for deployments in edge or resource-limited devices. On the other hand, our BQHD is about 6.9× faster than TDQN and 3.7× faster than BDQN. In Table 2, we further evaluate our algorithm on the LunarLander and Acrobot task, where the BQHD achieves much higher rewards than BDQN while providing runtime speedups, e.g., 5.5× (2.7×) on embedded CPU for Acrobot (LunarLander).

4.4 Ablation Study

Figure 8(a) highlights the difference between different mask probabilities. The probability determines which models are trained on the specific sample, which effectively allows the shared buffer to act as separate smaller buffers for each sub-model. Prior work [32] suggests that training with multiple mini buffers increases the run time but allows for more diversity within the models. According to our results, using a p between 0.5 and 0.75 is a fair compromise between average rewards and run time.

As stated in Section 4.1, the dimensionality of each sub-model is correlated with the quality of learning. Figure 8(b) showcases different dimensionalities, D, for each sub-model and compares the average reward for 200 episodes. We notice that increasing the dimensionality increases the learning runtime for the benefit of higher average rewards. $D = 2000$ is chosen as it has comparable convergence speed to $D = 1000$, yet still achieves higher rewards than vanilla QHD at 200 episodes.

Table 2: Learning quality and efficiency comparison

Task	Alg.	Reward	Laptop Runtime (s)	RPi Runtime (s)
CartPole	BDQN	186	1896	7489
	BQHD	1005	766	2006
Acrobot	BDQN	17	6066	38507
	BQHD	70	906	7045
LunarLander	BDQN	27	13977	71480
	BQHD	65	4097	27180

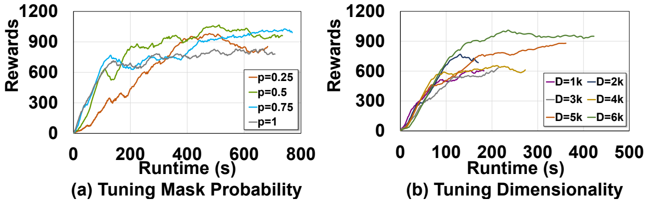

Figure 8: Explore the changes in the Cartpole learning curve when tuning (a) Bernoulli probability p and (b) hypervector dimensionality D. We only record 200 episodes for (b).

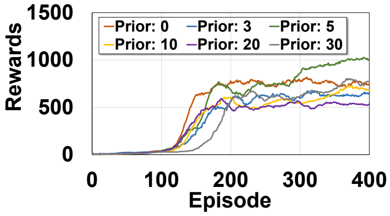

Figure 9: Ablation study on the scale of prior hypervectors.

The scale, σ, for prior hypervectors heavily influences the exploration of the agent. The goal of the randomized prior tuning is to find an optimal scale such that it helps the sub-models predict different actions at any given new state, but not too large such that the agent fails to learn the optimal action. Figure 9 suggests that using $\sigma = 5$ gives just enough motivation for the agent to explore efficiently and optimally in terms of runtime and average rewards.

5 RELATED WORK

Reinforcement Learning: By integrating deep learning, modern RL algorithms play an increasingly important role in fields like wireless communication [22], resource sharing in smart cities [2, 26], computer games [24], and intelligent transportation optimization [19]. Deep RL algorithms leverage DNN to abstract better policy for agents. However, they are computationally intensive due to frequent agent learning, rendering poor applicability at the edge. **Exploration in RL:** In RL, there is a dilemma between exploration and exploitation [3]. In many applications, RL agent exploration is empirically maintained through the ϵ decay rate in ϵ-greedy [23, 24]. Alternatively, the Boltzmann exploration assigns probabilities to each action proportionally to their scores, avoiding blind random actions [12, 37]. More advanced exploration methods are usually designed in conjunction with RL models. Examples include random value functions [33], bootstrapped DQN [32], Thompson sampling [1, 5], and Bayesian inference [10]. They share a similar motivation to our algorithm, i.e., following the "optimism in the face of uncertainty" principle [21]. However, the overhead of these exploration techniques further inhibits the deployments in practice.

Learning with Hyperdimensional Computing: As an alternative lightweight machine learning method, HDC has been implemented to solve various learning and cognitive reasoning tasks, such as graph reasoning [18, 35, 44], biosignal processing [25, 31, 34, 36, 38], human activity classification [20, 30, 46], genomic sequencing [7, 45], regression [9, 13, 28], outlier detection [41, 42], and text recognition [39]. These works show that HDC can outperform machine learning solutions, e.g., support vector machines and DNNs. As for RL tasks, HDC has been utilized to achieve faster convergence and lower the computational cost for both value-based and policy-based RL algorithms [8, 27, 29]. The proposed algorithm in work [27] requires a significantly shorter runtime in learning and obtains a better policy than DQN. As a downstream application, work in [16] combines HDC-based RL and environment modeling for resource optimization in an end-edge-cloud system. However, prior works do not systematically handle RL exploration; instead, they apply naive random exploration guided by ϵ-greedy [27] or annealing variance [29], which gives rise to suboptimal sample efficiency. In comparison, via uncertainty estimation, our proposed method adaptively encourages exploration when facing poorly understood samples.

6 CONCLUSION

We propose an HDC-based lightweight RL algorithm that adaptively encourages agent exploration. Our algorithm discovers possibly informative states through uncertainty estimation of HDC models. It enables the self-learning agent to interact with and learn in challenging environments more efficiently. Our evaluation shows a significant improvement in the sample and runtime efficiency when compared with prior HDC-based and DNN-based RL algorithms.

ACKNOWLEDGMENTS

This work was supported in part by DARPA Young Faculty Award, National Science Foundation #2127780, #2319198, #2321840, #2312517, and #2235472, Semiconductor Research Corporation (SRC), Office of Naval Research through the Young Investigator Program Award, and grants #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force Office of Scientific Research, grants #FA9550-22-1-0253, and generous gifts from Cisco.

REFERENCES

[1] Shipra Agrawal and Navin Goyal. 2012. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on learning theory*. JMLR Workshop and Conference Proceedings, 39–1.

[2] Aseel AlOrbani and Michael Bauer. 2021. Load balancing and resource allocation in smart cities using reinforcement learning. In *2021 IEEE International Smart Cities Conference (ISC2)*. IEEE, 1–7.

[3] Kai Arulkumaran, Marc Peter Deisenroth, et al. 2017. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine* 34, 6 (2017), 26–38.

[4] Sercan Aygun, Mehran Shoushtari Moghadam, et al. 2023. Learning from Hypervectors: A Survey on Hypervector Encoding. *arXiv preprint arXiv:2308.00685* (2023).

[5] Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. 2018. Efficient exploration through bayesian deep q-networks. In *2018 Information Theory and Applications Workshop (ITA)*. IEEE, 1–9.

[6] Greg Brockman et al. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[7] Hanning Chen and Mohsen Imani. 2022. Density-aware parallel hyperdimensional genome sequence matching. In *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 1–4.

[8] Hanning Chen, Mariam Issa, et al. 2022. Darl: Distributed reconfigurable accelerator for hyperdimensional reinforcement learning. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[9] Hanning Chen, M Hassan Najafi, et al. 2022. Full stack parallel online hyperdimensional regression on fpga. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 517–524.

[10] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*. PMLR, 1050–1059.

[11] Florin-Cristian Ghesu and Bogdan others Georgescu. 2017. Multi-scale deep reinforcement learning for real-time 3D-landmark detection in CT scans. *IEEE transactions on pattern analysis and machine intelligence* 41, 1 (2017), 176–189.

[12] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. 2017. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*. PMLR, 1352–1361.

[13] Alejandro Hernández-Cano, Yang Ni, Zhuowen Zou, Ali Zakeri, and Mohsen Imani. 2024. Hyperdimensional computing with holographic and adaptive encoder. *Frontiers in Artificial Intelligence* 7 (2024). https://doi.org/10.3389/frai.2024.1371988

[14] Wenjun Huang, Arghavan Rezvani, Hanning Chen, Yang Ni, Sanggeon Yun, Sungheon Jeong, and Mohsen Imani. 2024. A Plug-in Tiny AI Module for Intelligent and Selective Sensor Data Transmission. *arXiv preprint arXiv:2402.02043* (2024).

[15] Mohsen Imani, Zhuowen Zou, Samuel Bosch, Sanjay Anantha Rao, Sahand Salamat, Venkatesh Kumar, Yeseong Kim, and Tajana Rosing. 2021. Revisiting hyperdimensional learning for fpga and low-power architectures. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 221–234.

[16] Mariam Issa, Sina Shahhosseini, et al. 2022. Hyperdimensional hybrid learning on end-edge-cloud networks. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 652–655.

[17] Pentti Kanerva. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation* 1, 2 (2009), 139–159.

[18] Jaeyoung Kang, Minxuan Zhou, et al. 2022. RelHD: A Graph-based Learning on FeFET with Hyperdimensional Computing. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*. IEEE, 553–560.

[19] Jintao Ke et al. 2019. Optimizing online matching for ride-sourcing services with multi-agent deep reinforcement learning. *arXiv preprint arXiv:1902.06228* (2019).

[20] Yeseong Kim, Mohsen Imani, and Tajana S Rosing. 2018. Efficient human activity recognition using hyperdimensional computing. In *Proceedings of the 8th International Conference on the Internet of Things*. 1–6.

[21] Tze Leung Lai, Herbert Robbins, et al. 1985. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics* 6, 1 (1985), 4–22.

[22] Le Liang, Hao Ye, and Geoffrey Ye Li. 2019. Spectrum sharing in vehicular networks based on multi-agent reinforcement learning. *IEEE Journal on Selected Areas in Communications* 37, 10 (2019), 2282–2292.

[23] Gabriel Maicas, Gustavo Carneiro, et al. 2017. Deep reinforcement learning for active breast lesion detection from DCE-MRI. In *International conference on medical image computing and computer-assisted intervention*. Springer, 665–673.

[24] Volodymyr Mnih, Koray Kavukcuoglu, et al. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[25] Ali Moin, Andy Zhou, et al. 2021. A wearable biosensing system with in-sensor adaptive machine learning for hand gesture recognition. *Nature Electronics* 4, 1 (2021), 54–63.

[26] Almuthanna Nassar and Yasin Yilmaz. 2021. Deep reinforcement learning for adaptive network slicing in 5G for intelligent vehicular systems and smart cities. *IEEE Internet of Things Journal* 9, 1 (2021), 222–235.

[27] Yang Ni, Danny Abraham, et al. 2023. Efficient Off-Policy Reinforcement Learning via Brain-Inspired Computing. In *Proceedings of the Great Lakes Symposium on VLSI 2023*. 449–453.

[28] Yang Ni, Hanning Chen, et al. 2023. Brain-Inspired Trustworthy Hyperdimensional Computing with Efficient Uncertainty Quantification. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 01–09.

[29] Yang Ni, Mariam Issa, et al. 2022. Hdpg: hyperdimensional policy-based reinforcement learning for continuous control. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 1141–1146.

[30] Yang Ni, Yeseong Kim, et al. 2022. Algorithm-hardware co-design for efficient brain-inspired hyperdimensional learning on edge. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 292–297.

[31] Yang Ni, Nicholas Lesica, Fan-Gang Zeng, and Mohsen Imani. 2022. Neurally-inspired hyperdimensional classification for efficient and robust biosignal processing. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[32] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. 2016. Deep exploration via bootstrapped DQN. *Advances in neural information processing systems* 29 (2016).

[33] Ian Osband, Benjamin Van Roy, and Zheng Wen. 2016. Generalization and exploration via randomized value functions. In *International Conference on Machine Learning*. PMLR, 2377–2386.

[34] Una Pale, Tomas Teijeiro, and David Atienza. 2022. Multi-centroid hyperdimensional computing approach for epileptic seizure detection. *Frontiers in Neurology* 13 (2022), 816294.

[35] Prathyush Poduval, Haleh Alimohamadi, et al. 2022. Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning. *Frontiers in Neuroscience* 16 (2022), 757125.

[36] Abbas Rahimi, Artiom Tchouprina, et al. 2020. Hyperdimensional computing for blind and one-shot classification of EEG error-related potentials. *Mobile Networks and Applications* 25 (2020), 1958–1969.

[37] Brian Sallans and Geoffrey E Hinton. 2004. Reinforcement learning with factored states and actions. *The Journal of Machine Learning Research* 5 (2004), 1063–1088.

[38] Sina Shahhosseini, Yang Ni, et al. 2022. Flexible and personalized learning for wearable health applications using hyperdimensional computing. In *Proceedings of the Great Lakes Symposium on VLSI 2022*. 357–360.

[39] Kumar Shridhar and Harshil others Jain. 2020. End to end binarized neural networks for text classification. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*. 29–34.

[40] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30.

[41] Ruixuan Wang, Xun Jiao, and X Sharon Hu. 2022. Odhd: one-class brain-inspired hyperdimensional computing for outlier detection. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 43–48.

[42] Ruixuan Wang, Sabrina Hassan Moon, X Sharon Hu, Xun Jiao, and Dayane Reis. 2024. A Computing-in-Memory-based One-Class Hyperdimensional Computing Model for Outlier Detection. *IEEE Trans. Comput.* (2024).

[43] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[44] Ali Zakeri, Zhuowen Zou, et al. 2024. Conjunctive block coding for hyperdimensional graph representation. *Intelligent Systems with Applications* (2024), 200353.

[45] Zhuowen Zou, Hanning Chen, et al. 2022. Biohd: an efficient genome sequence search platform using hyperdimensional memorization. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 656–669.

[46] Zhuowen Zou, Yeseong Kim, et al. 2021. Scalable edge-based hyperdimensional learning system with brain-like neural adaptation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.