2022 IEEE 40th International Conference on Computer Design (ICCD) | 978-1-6654-6186-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCD56317.2022.00100

Hyperdimensional Hybrid Learning on End-Edge-Cloud Networks

Mariam Issa, Sina Shahhosseini, Yang Ni, Tianyi Hu, Danny Abraham,
Amir M. Rahmani, Nikil Dutt, and Mohsen Imani
University of California, Irvine
{mariamai, sshahhos, yni3, tianyih7, dannya1, a.rahmani, amirdutt, m.imani}@uci.edu

***Abstract*—In this paper, we present Hyperdimensional Hybrid Learning (HDHL), which combines model-free and model-based Reinforcement Learning, to effectively reduce the computational cost and environment interaction for optimizing an intelligent cloud service. We first show that Hyperdimensional Q-Learning (QHD), the state-of-the-art Hyperdimensional Computing value-based Reinforcement Learning algorithm, is computationally faster than the Deep Q-Network (DQN) for this task. In addition, we demonstrate how HDHL reduces the number of environment interactions by $4.8\times$ to learn the near optimal configuration. Our evaluation shows that HDHL is computationally more efficient than both Q-Learning algorithms, with the total time being reduced by $21.0\times$ compared to DQN and $16.5\times$ compared to QHD.**

I. Introduction

Resource management is a multi-dimensional optimization problem since it requires finding the appropriate device to offload computation while minimizing latency in aberrant network conditions [1]. Configuring this system is extremely complex and requires machine learning algorithms' run-time decision-making capabilities. Reinforcement Learning (RL) algorithms have shown great potential in solving dynamic resource management and scheduling problems [2], [3]. RL methods are developed to interact with the environment by trying different actions and reinforcing those that provide higher rewards [4].

Hybrid learning is an approach that utilizes model-based and model-free RL algorithms to exploit their strengths and lessen their respective disadvantages, e.g., costly real-time exploration and model-bias. To mitigate the trial-and-error phase of model-free learning, a model-based RL system model is incorporated to aid the model-free agent by producing data samples from the simulated environment. As a result, this reduces the number of direct interactions with the system to enable faster real-time learning. However, previously proposed Hybrid Learning algorithms utilize deep neural networks [5], which have the following computational challenges: (1) demand unreasonable resources to train since neural networks rely on costly back-propagation and gradient-based methods, (2) are extremely sensitive to noise in data, network, or underlying hardware, and (3) lack human-like cognitive support for long-term memorization and transparency.

To address the above listed challenges, recent intelligent algorithms aim to model the human brain more closely. Brain-inspired Hyperdimensional Computing (HDC) is gaining traction for its impressive learning ability, lightweight hardware implementation, and computational efficiency [6]–[9]. The origin of the field stems from neuroscience research on how the human brain processes information in high dimensions due to the brain's extensive circuitry [10], [11]. HDC abstractly extends this concept as a learning paradigm by modeling and operating on high-dimensional data representations [10], [12].

This class of algorithms is able to accomplish both classification and regression machine learning tasks [13]–[17] and has achieved comparable results to neural networks [6]. HDC has also been used for popular model-free RL algorithms, including PPO and Q-learning [18], [19], all while enabling a higher quality of learning (e.g., faster convergence, learning speed) and computational efficiency.

We employ an HDC version of the Deep Dyna-Q model to a hybrid learning algorithm for orchestrating an end-edge-cloud architecture for DL inference tasks [20], [21]. As a result, our algorithm significantly speeds up training by reducing the number of real-time interactions with the system and enabling computationally efficient learning.

In this paper, we propose HDHL, a novel hyperdimensional hybrid learning model to optimize resource management applications. Our contributions in this paper are listed below:

- To the best of our knowledge, HDHL is the first hyperdimensional hybrid learning algorithm to successfully and efficiently orchestrate an end-edge-cloud system.
- Compared to DQN, HDHL is a computational. more efficient algorithm with an improved quality of learning

We show that the novel HDHL algorithm is computationally more efficient than DQN by $21.0\times$ for three devices and by $9.5\times$ for four devices. In addition, we show HDHL's improved learning quality and significant reduction to the number of direct interactions with the system environment by $4.8\times$ for three devices and by $5.7\times$ for four devices.

II. Network Architecture Overview

Our network consists of a cloud node that hosts the main RL agent i.e., the intelligent orchestrator, where the system's offloading decision-making occurs, an edge device, and multiple end-devices. The metric that we aim to optimize is the comprehensive time of deciding which device to offload to, the service request, and the handling of the results. In addition, we define the Quality of Service (QoS) Goal as the service's accuracy constraint that needs to be satisfied.

To learn the optimal network configuration given the network conditions, we employ an RL agent to interact with the environment to collect experience, which is then used to formulate a policy. The ***State Space*** is the system dynamics at a given time step, which are the available processing cores, network stability, and memory availability. The measures of each of these components impact the optimal offloading decisions.

Furthermore, the ***Action Space*** is comprised of the offloading decisions made at each time step. The size of the

2576-6996/22/$31.00 ©2022 IEEE
DOI 10.1109/ICCD56317.2022.00100

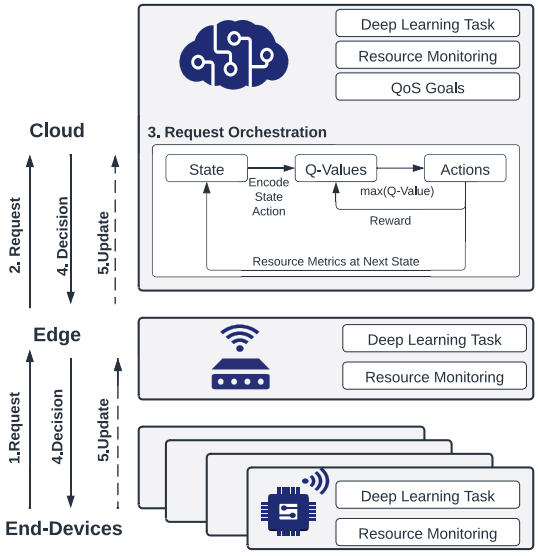

Fig. 1: Multi-user DL inference orchestration framework.

action space depends on the number of end-devices and the number of deep learning models stored in an end-node. Each end-device either offloads the intelligent task to a higher computational capacity node or executes the task locally, which requires a selection between the l models. To simplify the action space, we restrict the cloud and edge devices to store one model. We define the ***Reward Function*** as the observed average response time of the agent's consequent offloading decisions given the network conditions at each time step. This is the feedback from the environment used for optimizing the system's orchestration. To incorporate the accuracy threshold constraint, we penalize the agent if this constraint is violated; otherwise, the reward remains the average response time.

The end-edge-cloud architecture is presented in Figure 1 and shows the sequence of steps in this protocol. Each end-device include two software components: (i) the resource monitoring service which periodically collects the device system's parameters and broadcasts these metrics to the cloud and edge nodes; and (ii) the collection of deep learning models, which are trained for image classification. The cloud and edge nodes also contain these two software components. In addition, the cloud hosts the orchestrator, which is responsible for offloading decisions and local model selection if applicable.

III. Hyperdimensional Computing Hybrid Learning

To further improve the QHD algorithm, we incorporate model-based reinforcement learning to learn a system model and include a planning phase to leverage faster training. This approach is sectioned into three phases (1) the exploratory phase used for data collection, (2) the system model training phase, and (3) the planning phase which involves training the Q-Learning agent.

A. Hybrid Learning Algorithm

An issue that arises in applying the Q-Learning algorithm to this problem is working with an action space that is extremely large. This is a challenge since a large state space requires extensive exploration to learn the best configuration of actions. As described in the previous section, each end-device has $l+2$ choices at any time point: it either selects one of the l locally stored models or it offloads the task to either of the edge or cloud devices. Given the number of end devices, the action space quickly grows e.g., for a network configuration of n end-devices and l local models, the number of actions is $(l+2)^n$. This is difficult for any RL algorithm to efficiently tackle since the length of exploration consumes much of the time and cost to train the agent.

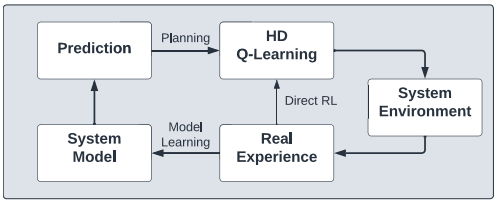

Fig. 2: Hybrid Learning Architecture.

A hybrid approach to this problem is to re-introduce the problem to the agent with the inclusion of a planning phase that utilizes a system model. To mitigate the expensive exploration phase, we use a system model to simulate the environment by training it to learn the next state and response times given a state-action pair. This is a computationally efficient alternative to leverage since relying solely on direct interactions with the environment for training is expensive. However, this approach still includes the Q-Learning exploration phase since it collects the initial data samples necessary for training the system model. The system model includes two models: the first is the *Time Model*, an HDC regression model used to predict the response time of taking an action at a given state, and the second is the *State Model*, which is implemented using a collection of HDC classification models to predict the next observation space.

In the first phase of this hybrid learning approach, the QHD model is deployed to the environment to collect data to populate the replay buffer, buffer$_{direct}$, for the subsequent phase. The algorithm used includes the Epsilon-Greedy Algorithm, a random neighbor action selection, and the calculation of Q-Values using HDC. This first phase is used solely for data collection; thus, no updates are made to either of the target or policy models. The second phase utilizes buffer$_{direct}$ to train the system model in predicting the network behaviors, which is then used in the planning phase. This third phase uses the system model's simulated data to further train the QHD agent.

B. System Model Design

As mentioned, we train a system model in order to learn the network behavior and simulate the environment in order to accumulate more data samples. The first component of the system model, the *Time Model*, predicts the reward e.g., response time, of selecting action A_t at state S_t. Since this is a regression model, it is quite similar to the QHD model in terms of its implementation; however, instead of using the regression model to output the Q-value of a state-action pair, it outputs the predicted response time of taking action A_t at a given state S_t. This is achieved by encoding the state S_t into hyperdimensional space and associating it to each of the model's class hypervectors $\{\vec{\mathcal{M}}_0, \vec{\mathcal{M}}_1, ...\vec{\mathcal{M}}_{N_{actions}}\}$, which outputs the predicted reward for selecting each action respectively. We sample tuples, $\{S_t, A_t, R_t, S_{t+1}\}$, from the

buffer$_{direct}$ to enable a supervised training of the model by updating the model by the error in predicted response time, R'_t, as shown below

$$M_{A_t} = M_{A_t} + \alpha(R_t - R'_t) \times \vec{\mathcal{S}}_t \quad (1)$$

where α is the learning rate, M_{A_t} is the action's corresponding class hypervector, and $\vec{\mathcal{S}}_t$ is the state's encoded hypervector. We train this model to be leveraged in the planning phase of this approach.

The system model's second component is the *State Model*, which is trained to predict the next state S_{t+1} of the environment given action A_t at state S_t. This involves predicting each of the elements in the observation space, which includes each device's CPU utilization, available memory, and available bandwidth. An issue that arises in implementing this classification task is due to the large number of possible network configurations. To calculate the number of classes for classification involves exhaustively enumerating all combinations of each resource metric. Accounting for all these possible network states and translating this into a classification task would require an HDC model that contains $2^{2n} \cdot 10^2 \cdot 2^2$ class hypervectors for n end-devices. This implementation would be infeasible since each class hypervector contains thousands of elements; hence, it can be assumed that training such a model would perform far worse than the QHD model, making this hybrid learning approach entirely futile.

Instead, we break the classification into sub-tasks by training multiple HDC models, where each model is trained to only predict one of the observations in the state space. Although there is a trade-off in having to train $4 + 2 \cdot n_{devices}$ HDC models, this is a far better alternative since all, except the cloud and edge devices' CPU utilization levels, would require binary classification tasks. The input for each of these models are the concatenated state and action pairs, which are encoded using the Radial Basis Function kernel. The resulting hypervector is then used to predict its observation feature using the cosine similarly between the encoded vector and each of the model's class hypervectors. Once each of these models output their respective predicted observation, the predictions are then concatenated to reconstruct the predicted state at the next time step, S_{t+1}. To update the HDC model, we implement an adaptive iterative training approach, which updates the class hypervectors based on the cosine similarity of the encoded state hypervector of the same class in order to increase their similarity in hyperspace. Conversely, if the class is predicted incorrectly, we update the incorrectly predicted class hypervector in the opposite direction to decrease the similarity:

$$M_{correct} = M_{correct} + \alpha\delta(\vec{\mathcal{H}}_{S_tA_t}, M_{correct}) \times \vec{\mathcal{H}}_{S_tA_t}$$
$$M_{wrong} = M_{wrong} - \alpha\delta(\vec{\mathcal{H}}_{S_tA_t}, M_{wrong}) \times \vec{\mathcal{H}}_{S_tA_t}$$

where α is the learning rate, δ is the cosine similarity, $\vec{\mathcal{H}}_{S_tA_t}$ is the encoded state-action hypervector, and $M_{correct}$ and M_{wrong} are the correctly and incorrectly predicted class hypervectors respectively. By training each of these HDC classification models separately, we are able to predict with high accuracy the overall next state of the network.

C. Planning Phase

The third phase is broken down into two parts. The first component involves populating a replay buffer, buffer$_{recom}$, with the best recommended state-action pairs, which is then used for its subsequent counterpart of training and updating the QHD agent. To populate the buffer$_{recom}$, the trained system model from the previous phase is used to predict the next state and reward given any state-action pair. For a given state, we utilize the *Time Model* to predict the action that yields the highest reward and use this selected action to predict the next state of the environment via the *State Model*. We use this fabricated data to populate a prioritized replay buffer, buffer$_{plan}$, to be used later in this phase. To tackle the challenge of exploring a large space, we also select a random neighboring action for each of the top action's adjacent action configurations to populate the buffer$_{plan}$, which enables a strategic exploration of the large state-action space. Furthermore, the buffer stores exactly one state-action pair for each respective state. In the scenario where a state exists in the buffer$_{plan}$ when the same state is populated with a new action: the buffer will replace the previously matched action with the new one, along with its corresponding predicted value. By employing these two models together, we collect additional data samples without needing to directly interact with the environment, saving time and computational overhead.

The next step in planning trains the QHD agent by sampling from the buffer$_{plan}$. By using the simulated data, only the states that are most likely to yield the highest rewards are strategically used to train the agent. It uses the QHD model to select the best action for each state in the sampled buffer and updates the QHD model with the correct and incorrect action values. After iterating through this step a number of times, the target model is updated at the end of the epoch. Additionally, this step includes updating the policy QHD model with the Q-value errors. This final phase of the hybrid learning concludes and returns to the exploratory first phase.

IV. Evaluation

A. Experimental Setup

For the DL orchestration framework, we first consider the MobileNet image classification application as the benchmark [22]. We consider eight different MobileNet models [22] with varying levels of accuracy and performance, with each model having different response times and accuracy levels. Our framework supports up to four end-device nodes, networked with edge and cloud layers.

Each end-user device is connected to a single edge device, and can request a DL inference service to the cloud layer. The cloud layer hosts the orchestrator that contains the RL agent, which handles the inference orchestration. Upon each service request, the RL agent is invoked to determine: (i) where the request should be processed and (ii) what DL model should be executed for the corresponding request. The platform consists of five AWS a1.medium instances with single ARM-core (as the end-device nodes), connected to an AWS a1.large instance with two CPUs (as edge device), and an AWS a1.xlarge instance with four CPUs (as the cloud node). In this work, we conduct experiments under four unique scenarios with varying network conditions.

V. Experimental Results

A. Performance

We demonstrate the proposed agent's performance in finding the optimal orchestration configuration. At design time, we

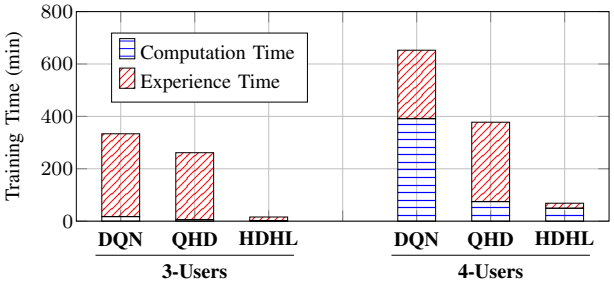

Fig. 3: Comparison of training times between DQN [23], QHD, and HDHL.

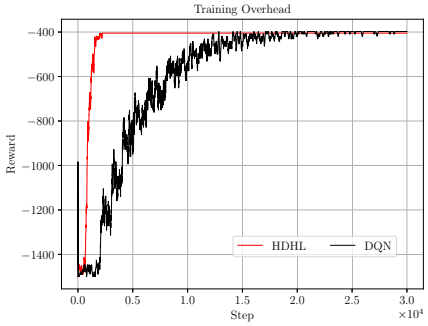

Fig. 4: Learning Overhead for DQN and HDHL

determine the true optimal configuration for orchestrating a DL task under any given condition of workload, and number of active users using a brute-force search. This is used for comparing the orchestration decision made by our agent to the true optimal decisions. Our proposed algorithm yields a 100% prediction accuracy in comparison with the true optimal decision. For each end-user node within each experimental scenario, we present the orchestration decision made by the proposed agent. The decision consists of the choice of execution node and inference model. Our proposed agent explores the Pareto-optimal space of offloading choice and model selection to minimize the response time within the accuracy requirement. For instance in the experimental scenario A, maintaining Max accuracy threshold results in an average response time of 410.35ms, by: setting the models to $d0$ on device $S1$-$S4$ and offloading choices to L (local device), E (Edge device), and C (Cloud device). The agent can improve the response time by sacrificing the accuracy within a predetermined tolerable level. For example, by lowering the accuracy threshold to 85%, the average response time can be reduced by 62%.

B. Results

Both Figure 3 and Table I shows the computational efficiency between the baseline DQN and the two HDC-based RL models. The QHD algorithm realizes an overall $1.3\times$ speed up to the total time. This result is enhanced when deployed to a system with four end-devices with the total time seeing a speed up of $1.5\times$. Meanwhile, the HDHL realizes an acceleration to the total time by $21.0\times$ for 3 devices and by $9.5\times$ for four devices. In addition, we compare the learning quality between the DQN and the HDHL models and demonstrate the latter's surpassing performance in Figure 4. It can be observed that HDHL converges at a significantly faster rate than the baseline DQN algorithm in $4.8\times$ fewer interactions with the environment, attaining a near optimal reward, which is 4.6×10^3 milliseconds slower than the optimal configuration.

VI. Conclusion

In this paper we present Hyperdimensional Hybrid Learning, the first HDC based hybrid learning algorithm for orchestrating deep learning tasks in an end-edge-cloud architecture. We demonstrate its computational efficiency and effective learning quality.

TABLE I: Training overhead (presented in number of steps) for different number of users compared with the state-of-the-art [23].

# of Users	DQN	QHD	HDHL
Three	1.2×10^4	5×10^4	0.2×10^4
Four	4.0×10^4	4.5×10^4	0.8×10^4

Acknowledgment

This work was supported in part by National Science Foundation #2127780, Semiconductor Research Corporation (SRC) AIHW and HW Security, Department of the Navy, Office of Naval Research, grants #N00014-21-1-2225 and #N00014-22-1-2067, the Air Force Office of Scientific Research under award #FA9550-22-1-0253, and a generous gift from Cisco.

References

[1] B. A. Mudassar, J. H. Ko, and S. Mukhopadhyay, "Edge-cloud collaborative processing for intelligent internet of things: A case study on smart surveillance," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2018.
[2] S. a. Yue, "Reinforcement learning based dynamic power management with a hybrid power supply," in *2012 IEEE 30th International Conference on Computer Design (ICCD)*, 2012.
[3] Y. Xu, H. G. Lee, Y. Tan, Y. Wu, X. Chen, L. Liang, L. Qiao, and D. Liu, "Tumbler: Energy efficient task scheduling for dual-channel solar-powered sensor nodes," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2019.
[4] François-Lavet *et al.*, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.
[5] S. Shahhosseini and othersl, "Hybrid learning for orchestrating deep learning inference in multi-user edge-cloud networks," 2022.
[6] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.
[7] A. Rahimi, P. Kanerva, and J. M. Rabaey, "A robust and energy-efficient classifier using brain-inspired hyperdimensional computing," in *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, ISLPED '16, (New York, NY, USA), p. 64–69, Association for Computing Machinery, 2016.
[8] A. Hernandez-Cane, N. Matsumoto, E. Ping, and M. Imani, "Onlinehd: Robust, efficient, and single-pass online learning using hyperdimensional system," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*.
[9] Z. Zou *et al.*, "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*, pp. 1–15, 2021.
[10] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation*, vol. 1, 2009.
[11] P. Poduval *et al.*, "Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning," *Frontiers in Neuroscience*, p. 5, 2022.
[12] A. Hernández-Cano *et al.*, "Prid: Model inversion privacy attacks in hyperdimensional learning systems," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 553–558, IEEE, 2021.
[13] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 445–456, IEEE, 2017.
[14] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.
[15] Y. Ni, Y. Kim, T. Rosing, and M. Imani, "Algorithm-hardware co-design for efficient brain-inspired hyperdimensional learning on edge," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 292–297, IEEE, 2022.
[16] P. Poduval *et al.*, "Cognitive correlative encoding for genome sequence matching in hyperdimensional system," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*.
[17] Z. Zou *et al.*, "Eventhd: Robust and efficient hyperdimensional learning with neuromorphic sensor," *Frontiers in Neuroscience*, vol. 16, 2022.
[18] Y. Ni, D. Abraham, M. Issa, Y. Kim, P. Mercati, and M. Imani, "Qhd: A brain-inspired hyperdimensional reinforcement learning algorithm," 2022.
[19] Y. Ni, M. Issa, D. Abraham, M. Imani, X. Yin, and M. Imani, "Hdpg: Hyperdimensional policy-based reinforcement learning for continuous control," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, (New York, NY, USA), p. 1141–1146, Association for Computing Machinery, 2022.
[20] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Machine Learning Proceedings 1990* (B. Porter and R. Mooney, eds.), pp. 216–224, San Francisco (CA): Morgan Kaufmann, 1990.
[21] B. Peng, X. Li, J. Gao, J. Liu, and K. Wong, "Integrating planning for task-completion dialogue policy learning," *CoRR*, vol. abs/1801.06176, 2018.
[22] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.
[23] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," pp. 389–400, 06 2018.