HERTI: a Reinforcement Learning-Augmented System for Efficient Real-Time Inference on Heterogeneous Embedded Systems

Myeonggyun Han
Department of CSE, UNIST
hmg0228@unist.ac.kr

Woongki Baek
Department of CSE and Graduate School of AI, UNIST
wbaek@unist.ac.kr

Abstract—**Real-time inference is the key technology that enables a variety of latency-critical intelligent services such as autonomous driving and augmented reality. Heterogeneous embedded systems that consist of various computing devices with widely-different architectural and system-level characteristics are emerging as a promising solution for real-time inference. Despite extensive prior works, it still remains unexplored to design and implement a practical system that enables efficient real-time inference on heterogeneous embedded systems.**

To bridge this gap, we propose HERTI, a reinforcement learning-augmented system for efficient real-time inference on heterogeneous embedded systems. HERTI efficiently explores the state space and robustly finds an efficient state that significantly improves the efficiency of the target inference workload while satisfying its deadline constraint through reinforcement learning. Our quantitative evaluation conducted on a real heterogeneous embedded system demonstrates the effectiveness of HERTI in that HERTI achieves high inference efficiency in multiple metrics (i.e., energy and energy-delay product) with a strong deadline guarantee in contrast to the state-of-the-art techniques, delivers larger gains as the inference deadline and the system heterogeneity increase, provides strong generality for hyper-parameter tuning, and significantly reduces the training time through its estimation-based approach across all the evaluated inference workloads and scenarios.

Index Terms—**Real-Time Inference; Heterogeneous Embedded Systems; Reinforcement Learning;**

I. Introduction

The need for real-time deep-learning inference is ever increasing to enable latency-critical intelligent services such as autonomous driving, interactive image editing, and augmented reality. For such services, it is highly crucial to meet the deadline of real-time inference workloads to prevent any dangerous and disastrous situations from happening and deliver the best possible user experiences. Real-time inference workloads are often executed within a single system without relying on cloud services. This approach provides various advantages such as low latency and enhanced security and privacy.

Heterogeneous embedded systems have emerged as a promising solution for efficient real-time inference in a variety of computing domains ranging from mobile to high-performance computing. Heterogeneous embedded systems consist of various computing devices (e.g., big core cluster, little core cluster, GPU, and neural processing unit (NPU)), each of which exhibits widely-different architectural and system-level characteristics in terms of performance, power consumption, functionality (e.g., supported mathematical operations), memory capacity, and communication overheads.

Inference workloads also exhibit widely-different characteristics on heterogeneous computing devices in terms of performance, power efficiency, communication overheads, and constraints. For example, some parts of inference workloads may efficiently be executed on hardware accelerators such as NPUs, whereas other parts of inference workloads need to be executed on CPUs to avoid excessive communication overheads. If inference workloads are scheduled across heterogeneous computing devices in a suboptimal manner, they may fail to meet their deadlines and/or achieve low efficiency.

Despite extensive prior works on enhancing the efficiency of inference workloads on various systems, little work has been done to investigate the design and implementation of a practical system that enables efficient real-time inference on heterogeneous embedded systems. The main challenge for developing such a system is the high design complexity, which is caused by various characteristics of inference workloads and heterogeneous computing devices and critical constraints such as the inference deadline and the functional and capacity constraints of heterogeneous computing devices.

To bridge this gap, this work proposes *HERTI*, a reinforcement learning (RL)-augmented system for efficient real-time inference on heterogeneous embedded systems. HERTI employs the accurate and efficient execution and communication cost estimators for inference workloads, which significantly reduce the training time. HERTI efficiently explores the state space with heterogeneity and constraint awareness and robustly finds the efficient state that executes the target inference workload across the heterogeneous computing devices while satisfying the deadline constraint through RL.

Specifically, this paper makes the following contributions:

- We propose HERTI, an RL-augmented system for efficient real-time inference on heterogeneous embedded systems. HERTI builds on the accurate and lightweight execution and communication cost estimators to significantly accelerate the training process. HERTI generates the efficient model slicing and execution plan for the target real-time inference workload with a strong deadline

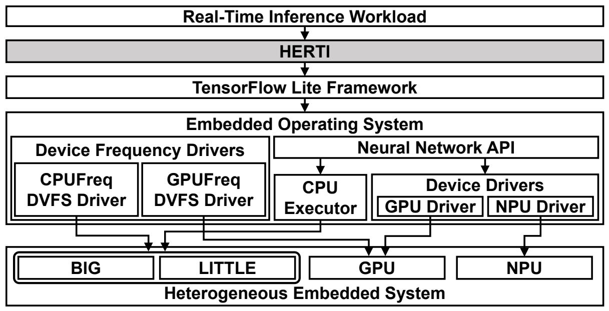

Fig. 1: Hardware and software stacks for deep-learning inference on heterogeneous embedded systems.

guarantee based on RL.

- Since the model slicing and execution planning problem for efficient real-time inference on heterogeneous embedded systems is a variant of the NP-hard multiple-choice quadratic knapsack problem [27] and has the Markov property [49], we formulate it as an RL problem. To overcome the state space explosion problem, we employ the state-of-the-art RL algorithm (i.e., deep Q-network (DQN) [35]) for solving the model slicing and execution planning problem.
- We design and implement the prototype of HERTI as a user-level runtime system based on the TensorFlow Lite programming framework [8] for deep-learning inference on the Android OS. HERTI significantly improves the efficiency of the target inference workload with a strong deadline guarantee by executing its slices across computing devices on the underlying heterogeneous embedded system in a heterogeneity- and constraint-aware manner.
- We quantify the effectiveness of HERTI using widely-used inference workloads on a real heterogeneous embedded system that consists of the big core cluster, little core cluster, GPU, and NPU. Our quantitative evaluation demonstrates the effectiveness of HERTI in that it achieves high efficiency and outperforms the deadline-conscious state-of-the-art technique (i.e., AutoScale [29]) in multiple metrics (i.e., energy (i.e., 28.4%) and energy-delay product (i.e., 29.2%)) while satisfying the deadline constraint, consistently meets inference deadlines in contrast to the deadline-oblivious state-of-the-art technique (i.e., MOSAIC [19]), effectively translates increased inference deadlines and system heterogeneity into larger efficiency gains, exhibits the strong generality in that the RL hyper-parameters tuned for a specific configuration can effectively be used in other configurations, and significantly reduces the training time through its estimation-based approach across all the inference workloads and scenarios.

II. Background

A. Heterogeneous Embedded Systems and Inference

Heterogeneous embedded systems consist of various computing devices (e.g., big core cluster, little core cluster, GPU, and NPU). Each heterogeneous computing device exhibits widely-different characteristics in terms of performance, power efficiency, communication overheads, functionality, and memory capacity [1]–[5].

Widely-used deep-learning (DL) frameworks (e.g., TensorFlow Lite [8], PyTorch [39]) simplify the inference workload programming on heterogeneous embedded systems. With such frameworks, parts of inference workloads are offloaded to computing devices through the invocations of the API functions provided by the frameworks, which eliminate the need for implementing device-specific code.

Figure 1 illustrates the hardware and software stacks for DL inference on heterogeneous embedded systems. The system (i.e., HERTI) proposed in this work is shown in grey color.

Inference workloads comprise *layers*. A layer is defined as a set of associated mathematical operations such as convolution and rectifier. Each layer takes a set of input tensors (i.e., multidimensional arrays), performs computations specified by its mathematical operations, and generates a set of output tensors.

We define a *model slice* as a set of consecutive layers, which are executed on the same computing device in the underlying heterogeneous embedded system. There exist communication overheads between adjacent slices as they need to communicate through input and output tensors.

Due to memory or functionality constraints, it may be infeasible to execute a model slice on certain heterogeneous computing devices. For example, if a model slice includes mathematical operations unsupported by a computing device or has a memory footprint larger than the memory capacity of a computing device, the model slice cannot be executed on the device.

While the importance of efficient real-time inference is ever increasing, it is highly challenging to maximize the efficiency of real-time inference workloads on heterogeneous embedded systems. As shown in prior work [19], each layer exhibits widely-different performance and power consumption characteristics on each heterogeneous computing device and different model slicing plans incur different execution and communication costs. In addition, each heterogeneous computing device imposes different constraints. Therefore, the design complexity and the state space that an optimizing system needs to cover drastically increase.

B. Deep Q-Network

Reinforcement learning (RL) is a machine learning technique that aims to learn a sequence of decisions that maximizes the cumulative reward [49]. In RL, there are two main components – the agent and the environment. The agent interacts with the environment in discrete time steps. The state of the environment at step t is denoted as s_t. The agent chooses an action a_t from the available action set according to a policy and performs it. The state of the environment is then transitioned to s_{t+1} and the agent receives the reward r_t. The goal of the agent is to learn the best policy that maximizes the cumulative reward.

Q-learning is one of the widely-used RL algorithms because of its effectiveness and simplicity [49]. Q-learning maintains a matrix called the Q-table whose row and column counts are equal to the number of feasible states of the environment and the number of actions that can be performed by the agent, respectively. Each element in the Q-table encodes the expected value (i.e., the Q-value) of its associated action when the environment is in its associated state. At each time step t, the agent performs the action that has the maximum value in the current state. The corresponding Q-table entry is then updated using Equation 1, where α and γ denote the learning rate ($0 < \alpha < 1$) and the discount factor ($0 < \gamma < 1$), respectively. The learning rate controls how fast Q-values change based on new observations. The discount factor determines the relative importance of the future rewards over the immediate rewards.

$$Q(s_t,a_t) \leftarrow (1-\alpha)\cdot Q(s_t,a_t) + \alpha\cdot(r_t + \max_{a} \gamma\cdot Q(s_{t+1},a)) \quad (1)$$

Despite its effectiveness and simplicity, Q-learning has a major drawback in that its memory usage for storing the Q-table becomes infeasible with modern computer systems when it is applied to solve complex problems with large numbers of states and/or actions. Unfortunately, the problem that this work aims to address belongs to such a category. For instance, the numbers of states and actions associated with the deadline-conscious energy efficiency optimization of the RN inference workload evaluated in this work are 1.1×10^{90} and 328, respectively (more details in Section III-C). This indicates that the memory usage for solving the energy optimization problem for RN using Q-learning is 1.4×10^{93} bytes (assuming each Q-value requires 4 bytes), which is infeasible on any kind of modern computer systems.

To address this limitation, recent work has proposed a variant of Q-learning called deep Q-network (DQN) [35]. The main idea of DQN is to replace the aforementioned memory matrix required for storing accurate the Q-values with a deep neural network called the Q-network, which is used to approximate the Q-table. Since the reasonably-sized Q-network can effectively approximate the Q-table, it eliminates the excessive memory usage issue of the original Q-learning algorithm. To mitigate the instability issues caused by the correlations in the sequence of observations and the correlations between the Q and target values, Mnih et al. use the experience replay and iterative update techniques [35]. Because of its effectiveness and practicality, we have made a design decision to employ DQN for HERTI.

III. Design and Implementation

HERTI is a software-based system that determines the efficient model slicing and execution plan for the target real-time inference workload on the underlying heterogeneous embedded system. Figure 2 shows its overall architecture, which mainly comprises the profiler, the execution and communication cost estimators, the model slicing and execution planner, and the runtime system.

A. Profiler

The profiler collects the total cost data (i.e., execution time and energy consumption) for executing each layer in the target inference workload on each computing device in the underlying heterogeneous embedded system. The collected total cost data is used to construct the execution and communication cost estimators.

If a layer can be executed on a computing device and the computing device supports DVFS, the total cost data for executing the layer is collected at the minimum and maximum available frequencies of the computing device as multiple data points are needed to construct the DVFS-aware performance estimator. If a layer cannot be executed on a computing device due to a constraint (e.g., memory or functional constraint), the total costs for executing the layer on the computing device are set to an infinite value. The profiler also collects the input and output tensor sizes of each layer, which are used to estimate the communication costs associated with the layer.

B. Execution and Communication Cost Estimators

The total cost data of each layer collected by the profiler includes both the execution and communication costs. Similarly to prior work [19], HERTI employs the execution and communication cost estimators. Specifically, they estimate the execution and communication costs of the layer on each heterogeneous computing device based on the total cost data and the input and output tensor sizes of the layer.

1) Communication Cost Estimator

Our experimental results with a microbenchmark that performs communications with various tensor sizes show that the communication cost is linearly proportional to the transferred tensor size. Based on this observation, we design the communication cost estimator to estimate the communication costs associated with the input and output tensors of the layer l using Equations 2 and 3. Specifically, $\tau_{i,l}$, $\tau_{o,l}$, β_{d,f_d}, and δ_{d,f_d} are the total sizes of the input and output tensors of the layer l and the regression coefficients for a computing device d at frequency f_d.

$$C_{i,l,d,f_d} = \beta_{d,f_d} \cdot \tau_{i,l} + \delta_{d,f_d} \quad (2)$$

$$C_{o,l,d,f_d} = \beta_{d,f_d} \cdot \tau_{o,l} + \delta_{d,f_d} \quad (3)$$

2) Execution Cost Estimator

The execution cost estimator is to estimate the cost for executing each layer on each computing device without including the communication costs. Specifically, the execution cost estimator first estimates the total execution cost of each layer on a computing device d running at frequency f_d using the profile data collected at the maximum and minimum frequencies of d. The execution cost estimator then subtracts the estimated communication costs of the layer (generated by the communication cost estimator) from the estimated total cost of the layer. The execution cost estimator comprises the performance and power estimators.

Performance Estimator: The performance estimator estimates the latency for executing a layer l on a computing device d at frequency f_d. The performance estimator employs a linear model shown in Equation 4, where T_{l,d,f_d}, $\theta_{l,d}$, and $\rho_{l,d}$ indicate the estimated latency and coefficients, respectively. $\theta_{l,d}$ and $\rho_{l,d}$ are computed using the data collected by the profiler at the maximum and minimum frequencies of d.

$$T_{l,d,f_d} = \theta_{l,d} \cdot \frac{1}{f_d} + \rho_{l,d} \quad (4)$$

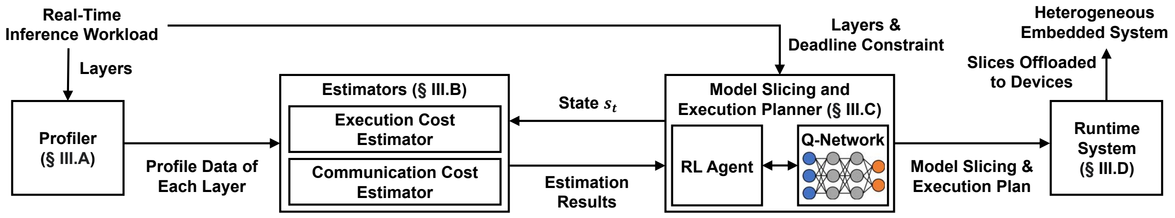
Real-Time Inference Workload
Layers
Profiler (§ III.A)
Profile Data of Each Layer
Estimators (§ III.B)
Execution Cost Estimator
Communication Cost Estimator
State s_t
Estimation Results
Layers & Deadline Constraint
Model Slicing and Execution Planner (§ III.C)
RL Agent
Q-Network
Model Slicing & Execution Plan
Heterogeneous Embedded System
Slices Offloaded to Devices
Runtime System (§ III.D)
Fig. 2: Overall architecture of HERTI

TABLE I: Voltage and frequency levels

Device	Voltage and frequency levels (V, MHz)
Big core cluster	(0.7, 682), (0.8, 1018), (0.8, 1210), (0.8, 1364), (0.9, 1498), (0.9, 1652), (0.9, 1863), (1.0, 2093), (1.1, 2362)
Little core cluster	(0.7, 509), (0.8, 1018), (0.9, 1210), (0.9, 1402), (1.0, 1556), (1.0, 1690), (1.1, 1844)
GPU	(0.6, 104), (0.7, 151), (0.7, 237), (0.7, 332), (0.8, 415), (0.8, 550), (0.9, 667), (1.0, 767)

Power Estimator: The power estimator estimates the power consumption for executing a layer l on a computing device d at frequency f_d using Equation 5. Specifically, the power estimator decomposes the total power consumption into dynamic (i.e., $P_{\text{dynamic},l,d,f_d}$) and static (i.e., P_{static,d,f_d}) power consumption. Static power consumption is the inherent property of a computing device and independent of the characteristics of the target inference workload. Therefore, it can be profiled only once for each computing device without the need for per-workload profiling.

$$P_{l,d,f_d} = P_{\text{dynamic},l,d,f_d} + P_{\text{static},d,f_d} \tag{5}$$

Since dynamic power consumption is dependent on the characteristics of the computing device (and its frequency) and the characteristics of the layer, the power estimator estimates dynamic power consumption in order to eliminate the need for extensive offline profiling. Specifically, the power estimator uses Equation 6 to estimate the dynamic power consumption (i.e., $P_{\text{dynamic},l,d,f_d}$) of layer l on computing device d at frequency f_d (and the corresponding voltage level V_{f_d}). In Equation 6, $f_{d,\max}$, $V_{f_{d,\max}}$, and $P_{\text{dynamic},l,d,f_{d,\max}}$ indicate the maximum frequency of d, the voltage level at $f_{d,\max}$, and the dynamic power consumption at $f_{d,\max}$.

$$P_{\text{dynamic},l,d,f_d} = \frac{V_{f_d}^2 \cdot f_d}{V_{f_{d,\max}}^2 \cdot f_{d,\max}} \cdot P_{\text{dynamic},l,d,f_{d,\max}} \tag{6}$$

The voltage and frequency levels of computing devices can be found in their specification documents or directly measured using instruments. Table I summarizes the voltage and frequency levels of the big core cluster, little core cluster, and GPU on the evaluated heterogeneous embedded system, which we found in their device tree source (DTS) files.[1] Dynamic power consumption can be estimated by substituting voltage and frequency levels into Equation 6.

Our experimental results show that the performance and power estimators achieve high accuracy. Specifically, the average performance and energy consumption estimation errors across all the evaluated inference workloads on the target heterogeneous embedded system are 3.0% and 4.3%, which are small.

[1]The evaluated NPU runs at 960 MHz and lacks a DVFS capability.

The main reason for adopting the estimation-based approach in this work is to significantly accelerate the training process. The deep Q-network model used in HERTI could be trained by repeatedly executing the target inference workload on the underlying heterogeneous embedded system in a large number of different configurations. However, it will significantly increase the training time due to the time spent for repeatedly executing the target inference workload on the heterogeneous embedded system.

To significantly reduce the training time, we have made a design decision to estimate the total costs for executing the target inference workload in each configuration. For instance, estimating the execution cost of the RN inference workload (Section IV) based on the estimators may be up to 14,313 times faster than executing it on the heterogeneous embedded system. Since the estimators achieve high estimation accuracy, the quality of the trained model based on the estimators is high and effectively guides to efficient states.

C. Model Slicing and Execution Planner

The main goal of the model slicing and execution planner (MSEP) of HERTI is to generate an efficient model slicing and execution plan for the target inference workload while satisfying its deadline constraint. Specifically, MSEP generates an efficient model slicing and execution plan, which specifies the number of slices, the specific layers that belong to each slice, and the specific computing device and frequency for each slice to efficiently execute the target inference workload with a strong deadline guarantee. MSEP can perform optimizations based on the user-defined metric (e.g., energy consumption, energy-delay product).

The model slicing and execution planning problem to maximize the inference efficiency on heterogeneous embedded systems with the deadline constraint is a variant of the multiple-choice quadratic knapsack problem, which is NP-hard [27]. In addition, it has the Markov property (i.e., the memoryless property) in that the next state is determined solely by the current state and the action without any dependency on the previous states or actions [49]. Therefore, we formulate the model slicing and execution planning problem as a reinforcement learning (RL) problem because RL is capable of effectively solving problems with the Markov property.

In our RL formulation, the environment is the target inference workload and the underlying heterogeneous embedded system and the agent is MSEP. The state of the environment is the current model slicing and execution plan. Equation 7 shows the state at step t. The state is expressed as a vector whose length is equal to the number of layers (i.e., Λ in Equation 7)

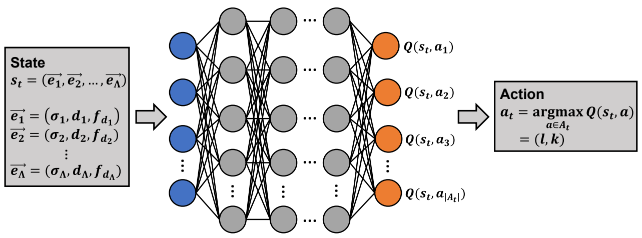
State
$s_t = (\overrightarrow{e_1}, \overrightarrow{e_2}, \ldots, \overrightarrow{e_\Lambda})$
$\overrightarrow{e_1} = (\sigma_1, d_1, f_{d_1})$
$\overrightarrow{e_2} = (\sigma_2, d_2, f_{d_2})$
$\overrightarrow{e_\Lambda} = (\sigma_\Lambda, d_\Lambda, f_{d_\Lambda})$
$Q(s_t, a_1)$
$Q(s_t, a_2)$
$Q(s_t, a_3)$
$Q(s_t, a_{|A_t|})$
Action
$a_t = \underset{a \in A_t}{\mathrm{argmax}}\, Q(s_t, a) = (l, k)$

Fig. 3: DQN architecture of MSEP

in the target inference workload.

$$s_t = (\vec{e}_1, \vec{e}_2, \cdots, \vec{e}_\Lambda) \tag{7}$$

Each element of the state vector is also a vector, which is shown in Equation 8. l ($1 \leq l \leq \Lambda$) denotes the layer index. σ ($\sigma \in \{0, 1\}$) specifies whether the corresponding layer is the first layer of a slice (i.e., $\sigma = 1$) or not (i.e., $\sigma = 0$). d ($d \in D$) denotes the computing device where the layer is scheduled to execute. In case of the heterogeneous embedded system evaluated in this work, the device set D is equal to $\{\text{big}, \text{little}, \text{GPU}, \text{NPU}\}$. f_d ($f_d \in F_d$) denotes the frequency of device d and F_d indicates the available frequencies of d.

$$\vec{e}_l = (\sigma, d, f_d) \tag{8}$$

At each step t, the agent (i.e., MSEP) interacts with the environment (i.e., the target inference workload and the underlying heterogeneous embedded system) by performing an action. We design the agent to choose a single layer (instead of multiple layers) to perform an action. This is to ensure that the state space is gradually explored. Equation 9 shows the action performed by the agent at step t.

$$a_t = (l, k) \tag{9}$$

Specifically, l ($1 \leq l \leq \Lambda$) indicates the layer index selected to perform the action. k ($k \in K$) denotes the specific change associated with the action. For each layer, the agent can apply one of the changes (i.e., K) defined in Equation 10 – creating a new slice starting from the layer (i.e., k_{split}), merging the slice where the layer belongs with the previous slice (i.e., k_{merge}), changing the computing device by traversing the device list in either direction (i.e., $k_{d,\text{next}}$ or $k_{d,\text{prev}}$), or increasing (i.e., $k_{f_d,\text{up}}$) or decreasing (i.e., $k_{f_d,\text{down}}$) the computing device frequency by one level.

$$K = \{k_{\text{split}}, k_{\text{merge}}, k_{d,\text{next}}, k_{d,\text{prev}}, k_{f_d,\text{up}}, k_{f_d,\text{down}}\} \tag{10}$$

MSEP employs the state-of-the-art RL algorithm (i.e., deep Q-network (DQN) [35]) to solve the model slicing and execution planning problem. Figure 3 shows the DQN architecture of MSEP, which mainly comprises a fully connected layer and ReLU functions.

Algorithm 1 shows the pseudocode for MSEP. After initializing key variables, MSEP iterates the main loop to train the DQN model (Lines 7–23). Without loss of generality, we discuss the main logic of MSEP in the context of step t.

At step t, MSEP chooses the best feasible action (i.e., a_t) to perform based on the Q-network with a probability of $1 - \epsilon(t)$ (Line 12 in Algorithm 1). To prevent the algorithm from converging to a local optimum, MSEP randomly (instead of consulting the Q-network) chooses a feasible action with a

Algorithm 1 The findEfficientSlicingAndExecPlan function

```
 1: procedure FINDEFFICIENTSLICINGANDEXECPLAN($\Lambda$)
 2:     $Q \leftarrow$ initializeQNetwork($\Lambda$, depth, width)
 3:     $M_r \leftarrow$ initializeReplayMemory()
 4:     $s_1 \leftarrow s_{\text{init}}$
 5:     $s_{\text{best}} \leftarrow s_{\text{init}}$
 6:     $r_{\text{best}} \leftarrow -\Phi$
 7:     for $t \leftarrow 1$ to $t_{\max}$ do
 8:         $A_t \leftarrow$ getValidActions($s_t$)
 9:         if generateRandomNumber() $< \epsilon$(t) then
10:             $a_t \leftarrow$ getRandomAction($A_t$)
11:         else
12:             $a_t \leftarrow \text{argmax}_{a \in A_t} Q(s_t, a)$
13:         end if
14:         $r_t \leftarrow$ calculateImmediateReward($s_t$, $a_t$)
15:         $s_{t+1} \leftarrow$ applyAction($s_t$, $a_t$)
16:         if $r_t > r_{\text{best}}$ then
17:             $r_{\text{best}} \leftarrow r_t$
18:             $s_{\text{best}} \leftarrow s_{t+1}$
19:         end if
20:         $M_r \leftarrow M_r \cup \{(s_t, a_t, r_t, s_{t+1})\}$
21:         $B \leftarrow$ sampleMiniBatch($M_r$)
22:         trainAndUpdateQNetwork($Q$, $B$, $t$)
23:     end for
24:     return $s_{\text{best}}$
25: end procedure
```

Algorithm 2 The calculateImmediateReward function

```
 1: procedure CALCULATEIMMEDIATEREWARD($s$, $a$)
 2:     $s' \leftarrow$ applyAction($s$, $a$)
 3:     $C_{s'} \leftarrow$ estimateCommunicationCost($s'$)
 4:     $T_{s'} \leftarrow$ estimatePerformance($s'$)
 5:     $P_{s'} \leftarrow$ estimatePowerConsumption($s'$)
 6:     $\Gamma_{s'} \leftarrow$ calculateCost($C_{s'}$, $T_{s'}$, $P_{s'}$)
 7:     if $T_{s'} \leq T_{\text{deadline}}$ then
 8:         $r_{s'} \leftarrow -\sqrt{\Gamma_{s'}}$
 9:     else
10:         $r_{s'} \leftarrow -\Phi$
11:     end if
12:     return $r_{s'}$
13: end procedure
```

probability of $\epsilon(t)$ (Line 10). As t increases, $\epsilon(t)$ decreases to facilitate choosing the action based on the trained Q-network.

MSEP then computes the immediate reward (i.e., r_t) associated with the state transition caused by performing the action by invoking the `calculateImmediateReward` function shown in Algorithm 2 (Line 14 in Algorithm 1). The `calculateImmediateReward` function uses the execution and communication cost estimators discussed in Section III-B to compute the immediate reward (Lines 3–11 in Algorithm 2).

The reward computation equation is shown in Equation 11, where Γ_s denotes the cost (e.g., energy consumption, energy-delay product) associated with state s and Φ indicates the penalty, which is set to 100, for states that violate the deadline constraint. If the deadline constraint of the target inference

TABLE II: Tunable hyper-parameters

Hyper-parameter	Candidate values	Tuned value
Learning rate	10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}	10^{-3}
Discount factor	0.9, 0.99	0.9
Q-network depth	2, 4, 8, 16	4
Q-network width	32, 64, 128, 256, 512, 1024	512
Batch size	32, 64	32
Epsilon in Adam optimizer	10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}	10^{-5}

workload is satisfied in state s, the reward is set to $-\sqrt{\Gamma_s}$, which increases as the cost decreases. Otherwise, the reward is set to Φ to exclude states in which the deadline constraint is violated.

$$r_s = \begin{cases} -\sqrt{\Gamma_s} & \text{if } T_s \leq T_{\text{deadline}} \\ -\Phi & \text{otherwise} \end{cases} \tag{11}$$

MSEP determines the next state (i.e., s_{t+1}) and memorizes it if the associated immediate reward is higher than the highest reward that has been discovered so far (Lines 15–19 in Algorithm 1). MSEP then inserts the tuple of (s_t, a_t, r_t, s_{t+1}) into the replay memory (Line 20). The replay memory is the key data structure to enable the experience replay mechanism of DQN [35]. Specifically, the replay memory contains the previous observations (i.e., the immediate rewards received by previous state transitions), which are used to train the Q-network.

MSEP generates a batch of training data by randomly choosing samples from the replay memory (Line 21 in Algorithm 1). MSEP then trains the Q-network using the batch of training data (Line 22). Since the Q-network is continuously trained using the previous observations, it can help MSEP to perform more effective actions in subsequent steps. MSEP repeats this process until a predefined iteration count is reached and returns the best state with the highest reward (Lines 7–24).

MSEP employs 6 tunable hyper-parameters. We use the Hyperband algorithm [31], [32], which is one of the most widely-used and effective algorithms, to automatically tune the hyper-parameters. Table II summarizes the six hyper-parameters and their candidate values explored by the tuner. Table II also shows the values of the hyper-parameters, which have been tuned using the MN-1.3 inference workload and its medium deadline (Table III). We have made a design decision to use only a single inference workload and a single deadline for hyper-parameter tuning in order to eliminate the need for per-workload and/or per-deadline hyper-parameter tuning. Section V-D quantifies the generality of this approach.

D. Runtime System

The runtime system of HERTI is a user-level process that executes the model slices of the target inference workload across the heterogeneous computing devices based on the model slicing and execution plan generated by the model slicing and execution planner. The runtime system is implemented in the C++ programming language based on the TensorFlow Lite framework [8] on the Android OS because it supports various heterogeneous computing devices and provides a well-established programming environment. However, since HERTI employs a framework-agnostic approach for slicing and executing the target inference workload, we believe that HERTI can widely be applied to other representative deep-learning frameworks such as PyTorch [39].

IV. Experimental Methodology

Inference Workloads: Table III summarizes the evaluated inference workloads – Inception V4 (IN) [50], MnasNet with the width parameters (p_W) of 1.0 (MN-1.0) and 1.3 (MN-1.3) [51], MobileNet V2 with $p_W = 1.3$ (MO-1.3) and $p_W = 1.4$ (MO-1.4) [43], ResNet V2 (RN) [21], and VGG (VGG) [45]. The evaluated inference workloads achieve high accuracy (e.g., the Top-1 accuracy with the ImageNet dataset [42] in Table III) and exhibit widely-different characteristics such as the model size (i.e., the memory usage reported by TensorFlow Lite), the layer count, and the inference latency.

MobileNet V2 and MnasNet provide a mechanism to tune their inference accuracy and latency using the width parameter (i.e., p_W), which controls the number of channels. When it is set to a larger value, their inference accuracy and latency tend to increase.

The deadline of each workload is determined using Equation 12, where w, ζ, $T_{w,\text{best}}$, and $T_{w,\text{energy\_optimized}}$ denote the inference workload, a scale factor, and the latencies that incur the highest performance (i.e., the best-case execution time) and the lowest energy consumption that are determined by the technique proposed in [19] (more details in Section V-A), respectively. As shown in Table III, we use three deadlines (i.e., `short` ($\zeta = 0.25$), `medium` ($\zeta = 0.5$), and `long` ($\zeta = 1.0$)) for each inference workload to investigate the sensitivity of HERTI to the inference deadline.

$$\text{Deadline}_w = T_{w,\text{best}} + \zeta \cdot (T_{w,\text{energy\_optimized}} - T_{w,\text{best}}) \tag{12}$$

Heterogeneous Embedded System: To investigate the effectiveness of HERTI, we use a heterogeneous embedded system, the HiKey 970 development board [6]. The evaluated system is equipped with the Kirin 970 mobile processor [1] that consists of a CPU with four Cortex-A73 (big) cores, four Cortex-A53 (little) cores, a Mali-G72 GPU, and an NPU. Table I summarizes their available voltage and frequency levels.

Due to the memory constraint, the NPU cannot execute a model slice larger than 100 MB [6]. The exact memory constraints of the big core cluster, little core cluster, and GPU are unknown. However, they can successfully execute all the evaluated inference workloads without any memory capacity issues. Therefore, we assume that their memory constraints are sufficiently large for the evaluated inference workloads.

As for the system software stack, Android 8.1 is installed in the evaluated heterogeneous embedded system. In addition, HERTI and the evaluated inference workloads are implemented in the TensorFlow Lite 1.11.0 [8].

We measure the inference latency through the `high_resolution_clock` C++ standard library function. We collect the inference energy consumption data using an external power monitor [7]. The power monitor periodically samples the

TABLE III: Evaluated inference workloads

Workload	Accuracy (%)	Size (MB)	Layers	`short`/`medium`/`long` deadlines in milliseconds
Inception V4 (IN) [50]	80.1	183.7	20	190/210/250
MnasNet with $p_W = 1.0$ (MN-1.0) [51]	74.1	321.9	20	85/100/130
MnasNet with $p_W = 1.3$ (MN-1.3) [51]	75.2	511.6	20	105/120/150
MobileNet V2 with $p_W = 1.3$ (MO-1.3) [43]	74.4	187.8	18	75/90/120
MobileNet V2 with $p_W = 1.4$ (MO-1.4) [43]	75.0	214.7	18	85/100/130
ResNet V2 (RN) [21]	77.8	260.4	53	155/170/200
VGG (VGG) [45]	71.5	407.4	16	135/150/180

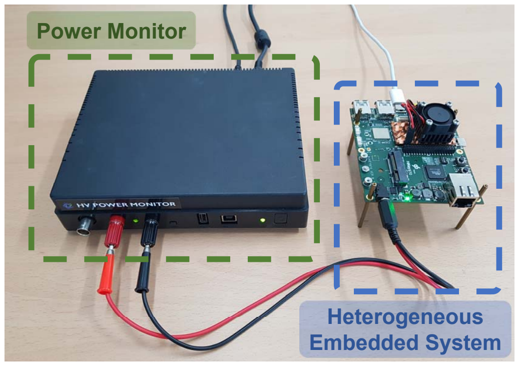

Fig. 4: Evaluated heterogeneous embedded system and power monitor

voltage and current applied to the evaluated heterogeneous embedded system at the rate of 5000 samples per second. Figure 4 shows the evaluated heterogeneous embedded system and the power monitor.

Training Server System: To train the DQN model of HERTI, we use a server system equipped with two Intel Xeon E5-2640 16-core CPUs running at 2.6 GHz and 32 GB memory. The server system is installed with Ubuntu 18.04 and TensorFlow 2.3.1 [9].

V. Evaluation

A. Overview

This section quantifies the effectiveness of HERTI. Specifically, we aim to investigate (1) inference latency, (2) inference energy, (3) the sensitivity of the efficiency gains to the inference deadline and the system heterogeneity, (4) generality, (5) energy-delay product (EDP) efficiency, and (6) training time reduction through the estimation-based approach.

For each inference workload, we evaluate 13 versions – the big core cluster-preferred (`TFL-BIG-P`), little core cluster-preferred (`TFL-LITTLE-P`), GPU-preferred (`TFL-GPU-P`), NPU-preferred (`TFL-NPU-P`) with the performance governor of the Android OS, the big core cluster-preferred (`TFL-BIG-O`), little core cluster-preferred (`TFL-LITTLE-O`), GPU-preferred (`TFL-GPU-O`), NPU-preferred (`TFL-NPU-O`) with the on-demand governor of the Android OS, MOSAIC configured to conduct performance optimization (`MOSAIC-P`), MOSAIC configured to perform energy optimization (`MOSAIC-E`), AutoScale, exhaustive, and HERTI versions.

With the big core cluster-, little core cluster-, GPU-, and NPU-preferred versions, the slices of each inference workload are executed on the corresponding preferred computing device. To generate the big core cluster-, little core cluster-, GPU-, and NPU-preferred versions, we include as many consecutive layers as possible in each slice if they satisfy the memory and functionality constraints. If a layer cannot be included in the current slice due to a memory constraint, a new slice is created and the layer is added to the new slice, which is executed on the preferred device. If a layer cannot be executed on the preferred device due to a memory or functionality constraint, a separate slice is created for the layer and executed on a feasible device among NPU, GPU, and big core cluster (in the preference order).

The performance governor tends to deliver higher performance and lower energy efficiency than the on-demand governor because the performance governor always executes the target inference workload at the maximum frequency of the underlying computing device. In contrast, since the on-demand governor, which is the default governor of the Android OS, performs DVFS based on the dynamic load of the target inference workload, it typically exhibits lower performance and higher energy efficiency than the performance governor.

MOSAIC is a state-of-the-art technique that generates the optimal model slicing and execution plan of the target inference workload on heterogeneous embedded system in terms of the user-specified metric [19]. However, it is deadline-oblivious and provides no deadline guarantee for real-time inference workloads. MOSAIC can perform optimizations based on various user-specified metrics. For the `MOSAIC-P` and `MOSAIC-E` versions, MOSAIC is configured to generate the optimal model slicing and execution plan in terms of inference latency and energy efficiency, respectively. While the `MOSAIC-E` version lacks a deadline guarantee, its energy efficiency is always optimal. As quantified in this work, when a deadline is configured in a way that the `MOSAIC-E` version also happens to meet the deadline, HERTI achieves the same efficiency (i.e., the optimal energy efficiency) as the `MOSAIC-E` version, demonstrating its effectiveness.

AutoScale is a state-of-the-art technique that executes the target inference workload in a deadline-conscious manner and employs DVFS to enhance inference efficiency based on machine learning [29]. However, it lacks the consideration of the heterogeneity- and constraint-aware model slicing and execution for the target inference workload and executes the entire inference workload on a single device. In other words, AutoScale cannot exploit the optimization opportunities for slicing and executing the model at the layer level.

The exhaustive version empirically determines the model slicing and execution plan that exhibits the highest inference efficiency through exhaustive search. Note that exhaustive search is highly time- and resource-consuming and impractical. For example, it is estimated to take 6.1×10^{11} years to empirically find the best model slicing and execution plan for MN-1.3 through exhaustive search even using the lightweight execution and communication cost estimators. The experimental data for the exhaustive version of each inference workload has been collected for 10 days, covering up to 3.7×10^9 states. Finally, the HERTI version uses HERTI to generate the efficient model slicing and execution plan.

B. Inference Latency and Energy Efficiency

In this section, we evaluate the effectiveness of HERTI when it is configured to optimize the energy efficiency of the inference workload while satisfying its deadline constraint. The deadline for each inference workload is set to the medium one in Table III.

We first investigate the effectiveness of HERTI in terms of inference latency. Figure 5 shows the inference latency of each version of the inference workload, normalized to the deadline.[2] If the normalized inference latency of a certain version is 1 or lower, the version meets the deadline. Otherwise, it fails to satisfy the deadline constraint.

First, HERTI robustly satisfies the deadline constraint across all the evaluated inference workloads, which empirically demonstrates that HERTI provides a strong deadline guarantee for real-time inference workloads. Interestingly, HERTI chooses states that lead to the inference latency lower than the deadline (i.e., the normalized latency is lower than 1). This is mainly because such states result in higher energy efficiency than the other states that incur inference latency close to the deadline because such states facilitate the use of the evaluated NPU that lacks DVFS support (but still more efficient than other devices in many cases) and reduce static energy consumption.

Second, aside from HERTI, only the `TFL-NPU-P`, `MOSAIC-P`, and AutoScale versions meet the deadlines across the evaluated inference workloads. Since the NPU is effective for reducing the inference latency and the performance governor of the Android OS is optimized for performance, the `TFL-NPU-P` version meets the inference deadline. As the `MOSAIC-P` version conducts performance optimizations without considering the deadline of each inference workload, it incurs the latency significantly lower than the other versions at the cost of reduced inference efficiency. In addition, the AutoScale version meets the inference deadline by finding a single device and its DVFS setting and executing the entire inference workload (i.e., no model slicing) on the device.

Third, all the other versions fail to meet the deadlines of the inference workloads. The big core cluster-, little core cluster-, and GPU-preferred and `TFL-NPU-O` versions fail to meet the inference deadline because they lack the capability of exploiting heterogeneity in the underlying system and the Android OS is deadline-oblivious. The `MOSAIC-E` version fails to meet the inference deadline as it performs optimizations solely to reduce the inference energy consumption without considering the deadline constraint. Despite the extensive search conducted for each workload for 10 days, the exhaustive version still fails to meet the deadline. Due to the excessively large state space, the optimal state still remains undiscovered even after the extensive search.

[2]All the reported experimental data is the average of 100 experiments.

TABLE IV: Model slicing and execution plans

Workload	Model slicing and execution plan
IN	N_1^{960}, N_{11}^{960}, L_{20}^{1690}
MN-1.0	N_1^{960}, L_{14}^{1690}, N_{18}^{960}, L_{20}^{1402}
MN-1.3	N_1^{960}, B_9^{1210}, G_{12}^{332}, B_{14}^{1210}, G_{18}^{332}, B_{20}^{1210}
MO-1.3	N_1^{960}, G_8^{550}, L_{15}^{1402}, N_{18}^{960}
MO-1.4	N_1^{960}, G_9^{667}, L_{15}^{1690}, N_{18}^{960}
RN	B_1^{1863}, N_2^{960}, L_4^{1690}, N_5^{960}, B_{12}^{1364}, N_{13}^{960}, N_{30}^{960}, B_{48}^{1364}, N_{49}^{960}, B_{53}^{1863}
VGG	N_1^{960}, G_{14}^{332}, G_{15}^{237}, B_{16}^{682}

We now investigate the effectiveness of HERTI in terms of energy efficiency. Figure 6 shows the energy consumption of each version, normalized to that of the `MOSAIC-P` version. First, we observe that HERTI significantly outperforms other versions in terms of energy efficiency. For instance, HERTI consumes 28.0% and 28.4% lower energy than the `MOSAIC-P` and AutoScale versions, respectively. Our experimental results provide empirical evidence that HERTI robustly finds the optimal state that minimizes the energy consumption of each inference workload with a strong deadline guarantee through reinforcement learning.

Second, the `MOSAIC-E` version exhibits slightly lower energy consumption than the HERTI version when the inference deadline is set to the medium one. This is mainly because the `MOSAIC-E` version aggressively applies DVFS by solely focusing on minimizing the energy consumption without considering the inference deadline. Hence, as discussed above (Figure 5), the `MOSAIC-E` version fails to meet the deadlines of all the inference workloads.

In contrast, since HERTI performs inference energy efficiency optimizations in a deadline-conscious manner, it achieves the energy efficiency comparable to that of the `MOSAIC-E` version while robustly satisfying the deadline constraint for all the evaluated inference workloads. Further, as quantified in Section V-C, our experimental results show that HERTI achieves the same energy efficiency as that of the `MOSAIC-E` version when it is given a sufficiently long deadline with which the `MOSAIC-E` version happens to meet the inference deadline.

Table IV summarizes the model slicing and execution plans generated by HERTI for deadline-conscious energy optimization. Each letter (i.e., big core cluster (B), little core cluster (L), GPU (G), and NPU (N)) denotes a slice and its associated device. The subscript and the superscript to each letter indicate the ID of the first layer of the corresponding slice and the frequency of the computing device in MHz, respectively. HERTI tends to execute slices with relatively-low computation intensity and large communication overheads on the big or little cluster, computation-intensive slices with functional/memory constraints on the GPU, and computation-intensive slices without any constraints on the NPU.

C. Sensitivity

In this section, we analyze the sensitivity of the efficiency gains of HERTI to the inference deadline and the degree of

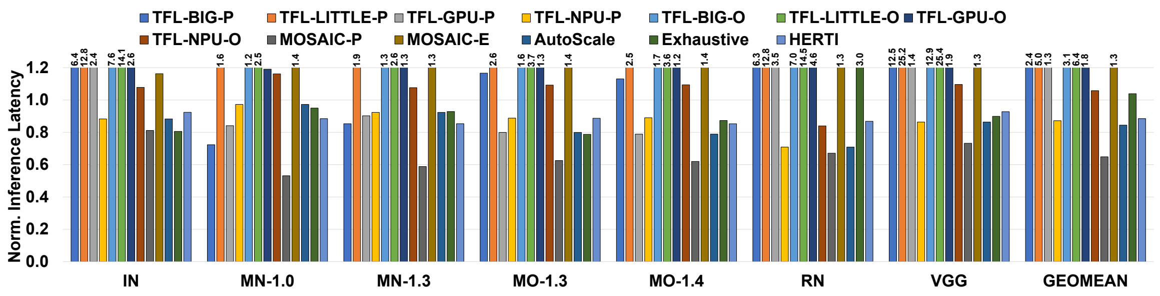
Fig. 5: Inference latency

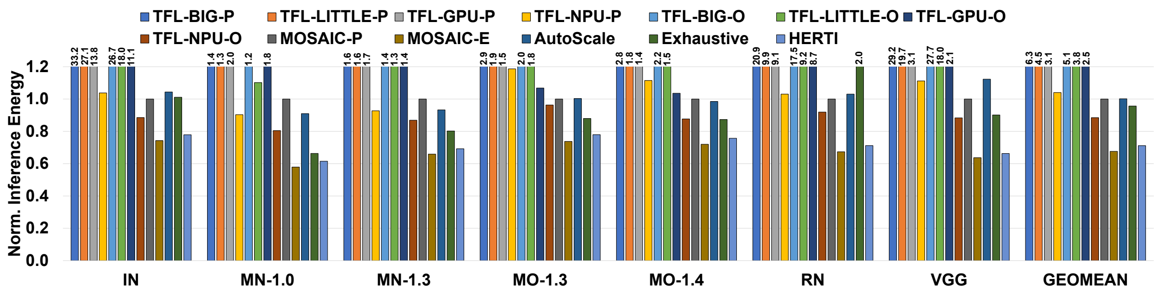
Fig. 6: Inference energy

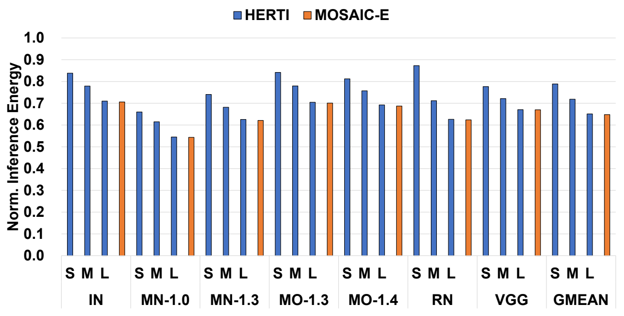
Fig. 7: Sensitivity to the inference deadline

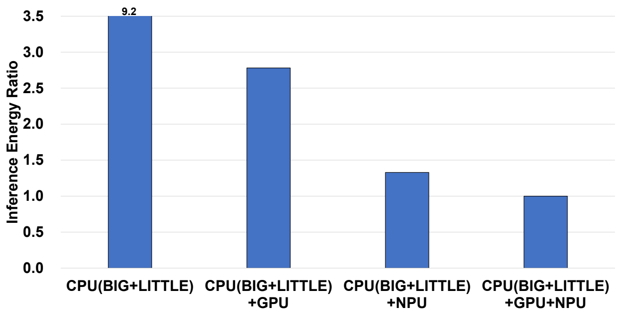
Fig. 8: Sensitivity to the system heterogeneity

the heterogeneity of the underlying system. Figure 7 shows the energy consumption of HERTI for each inference workload, normalized to that of the `MOSAIC-P` version. Each bar represents the energy consumption when the inference deadline is configured to the `short` (S), `medium` (M), or `long` (L) deadline in Table III. Figure 7 also shows the normalized energy efficiency of the `MOSAIC-E` version. Since the `MOSAIC-E` version is deadline-oblivious, we only show one bar for the `MOSAIC-E` version with each inference workload.

First, as the inference deadline increases, the energy efficiency gains of HERTI increase across all the inference workloads. For instance, the average energy efficiency gain increases by 14.1% as the inference deadline increases from the `short` deadline to the `long` deadline. This is mainly because the more relaxed constraint with a longer inference deadline facilitates HERTI to explore and find states that result in higher energy efficiency but still meet the inference deadline.

Second, with the `long` deadline, HERTI achieves the same energy efficiency as the `MOSAIC-E` version across all the inference workloads. While the `MOSAIC-E` version is deadline oblivious and lacks a deadline guarantee, the `MOSAIC-E` version happens to meet the inference deadline when the inference deadline for each inference workload is set to the `long` deadline. In this case, HERTI robustly generates the same model slicing and execution plan as that of the `MOSAIC-E` version, which is optimal.

We investigate the sensitivity of the efficiency gains of HERTI to the degree of the heterogeneity of the underlying system. To this end, we have created synthetic versions of HERTI, which only employ the CPU (i.e., the big and little core clusters), the CPU and another device (e.g., GPU or NPU), and all the computing devices (i.e., the full version of HERTI). Figure 8 shows the energy efficiency gain of each version, normalized to that of the full version of HERTI.[3]

We observe that the HERTI continues to achieve higher

[3]While omitted for conciseness, our experimental results show that HERTI outperforms MOSAIC-P and AutoScale by 20.6% and 11.8% when only CPU and GPU are used and by 21.7% and 13.3% when only CPU and NPU are used in terms of energy efficiency.

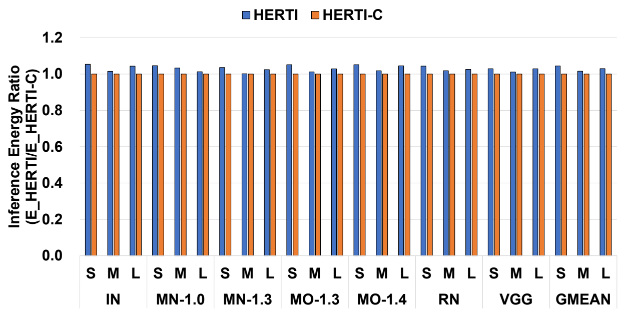

Fig. 9: Generality of HERTI

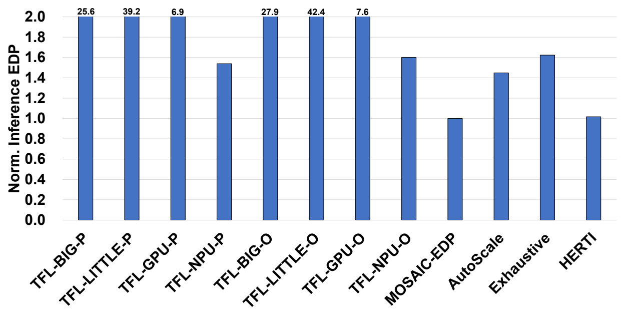

Fig. 10: Energy-delay product

energy efficiency with more heterogeneous computing devices by effectively utilizing all the available devices with a strong deadline guarantee. Our experimental results also demonstrate the importance of heterogeneity-aware systems for maximizing the efficiency of real-time inference workloads on heterogeneous embedded systems.

D. Generality

The hyper-parameters of the DQN model used in HERTI have been tuned using the MN-1.3 inference workload and its medium deadline. This design decision has been made to eliminate the need for per-workload and/or per-deadline hyper-parameter tuning. To quantify the generality of this approach, we compare the energy efficiency of the default version of HERTI with a highly customized version (i.e., `HERTI-C`) of HERTI whose hyper-parameters have been tuned for each pair of the inference workload and its target deadline. Figure 9 reports the energy efficiency of the two versions, normalized to the `HERTI-C` version.

We observe that the default version of HERTI achieves the energy efficiency similar to that of the `HERTI-C` version. For instance, the average energy consumption of the `HERTI-C` version is 2.9% lower than the default version of HERTI. Our experimental results demonstrate that HERTI exhibits strong generality in that it achieves high efficiency across all the evaluated inference workloads and scenarios without requiring per-workload or per-deadline hyper-parameter tuning. HERTI inherits this generality property from RL, which can widely be applied to various tasks with a set of hyper-parameters tuned for a single task.

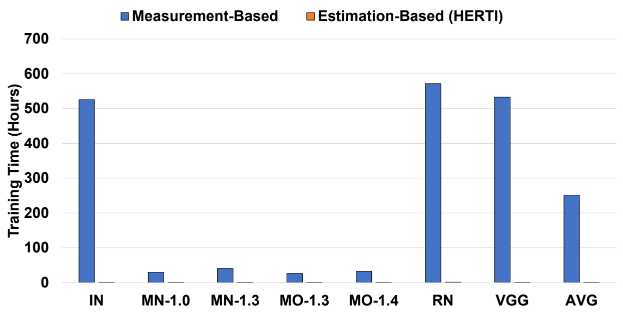

Fig. 11: Training time comparison

E. Energy-Delay Product Efficiency

So far, we have evaluated the effectiveness of HERTI in terms of energy efficiency. HERTI is versatile in that it can perform optimizations in various user-specified metrics. To evaluate the versatility of HERTI, we configure HERTI to perform optimizations based on the energy-delay product (EDP) metric while satisfying the deadline constraint. For conciseness, Figure 10 shows the EDP of each version, which is averaged (i.e., geometric mean) across all the inference workloads and normalized to that of the `MOSAIC-EDP` version. While MOSAIC is deadline-oblivious, the EDP of the `MOSAIC-EDP` version is optimal.

We observe that HERTI achieves the optimal EDP efficiency (i.e., same as the `MOSAIC-EDP` version) and significantly outperforms other versions. While the inference latency data is omitted for brevity, our experimental results demonstrate that HERTI robustly satisfies the deadline constraint across all the inference workloads when it is configured to perform EDP optimizations.

F. Training Time

HERTI employs the estimation-based approach to significantly reduce the time spent for training the DQN model. To quantify the effectiveness of this approach, we compare the training time of the estimation-based approach with that of the measurement-based approach. Since the measurement-based approach incurs excessively long training time for each inference workload, we were unable to measure its training time. To estimate the training time of the measurement-based approach, we first conduct training based on the estimation-based approach. We then compute the estimated training time with the measurement-based approach by summing the estimated execution times of all the explored states, which are estimated by the execution and communication cost estimators (Section III-B).

Figure 11 shows the training time of the estimation-based and measurement-based approaches for deadline-conscious energy optimization. First, the estimation-based approach significantly reduces the training time in comparison with the measurement-based approach. For instance, the estimation-based approach (i.e., 48 minutes) incurs 314 times shorter training time than the measurement-based approach (i.e., 15,088 minutes (251 hours)) on average across the evaluated inference workloads. Since executing the inference workload on the heterogeneous embedded system takes drastically longer time than estimating its execution and communication costs using the lightweight estimators, the estimation-based approach significantly reduces the training time.

Second, the estimation-based approach leads to more significant reduction in training time with some inference workloads such as IN and RN. This is mainly because such inference workloads are more complex than the others, requiring longer execution time on the underlying heterogeneous embedded system with the measurement-based approach.

Overall, our experimental results clearly demonstrate the effectiveness of HERTI in that it achieves high inference efficiency in terms of both energy and EDP metrics while satisfying the deadline constraint, effectively translates longer inference deadlines and a higher degree of heterogeneity of the underlying system into higher efficiency, exhibits strong generality with respect to hyper-parameter tuning, and significantly reduces the training time through its estimation-based approach across all the evaluated inference workloads and scenarios.

VI. Related Work

Prior works have extensively investigated deep-learning model analysis and optimization techniques to improve the inference latency and/or address the constraints of inference workloads [10], [12], [13], [19], [22], [24], [26], [29], [38], [40], [41], [47], [52]. While insightful, most of the prior works do not consider the efficiency heterogeneity and the constraints of inference workloads and emerging computing devices in an integrated manner [10], [13], [22], [24], [26], [38], [40], [41], [47], [52].

While the system (i.e., MOSAIC) proposed in [19] considers the efficiency heterogeneity and constraints for efficient inference, it lacks the consideration of the deadline constraint for real-time inference, which is the key technology that enables latency-critical intelligent services. As quantified in this work, MOSAIC fails to satisfy the deadline constraint and/or exhibits suboptimal efficiency (e.g., the `MOSAIC-P` version exhibits suboptimal energy efficiency as it is configured to minimize the inference latency) when executing inference workloads. In contrast, our work is significantly different in that it thoroughly addresses the efficient model slicing and execution planning problem with a strong deadline guarantee for real-time inference workloads through reinforcement learning (RL).

The systems (e.g., AutoScale) proposed in [12], [29] consider the real-time property of inference workloads and employ DVFS to enhance energy efficiency. However, they lack the consideration of the heterogeneity- and constraint-aware model slicing and execution for real-time inference workloads and emerging computing devices (e.g., NPU), which are crucial for achieving the best possible efficiency on heterogeneous embedded systems. For instance, as quantified in this work, the heterogeneity-oblivious inference workload execution employed by the aforementioned systems significantly degrades the efficiency. Our work significantly differs as it proposes an RL-augmented system that robustly addresses the model slicing and execution planning problem for real-time inference workloads in a heterogeneity- and constraint-aware manner and quantifies the effectiveness of HERTI on a real heterogeneous embedded system with a highly-optimized NPU.

Prior works have proposed techniques to apply machine learning to optimize the efficiency of computer systems in various contexts such as cloud resource management [15], [28], [30], [36], [37], memory system design and management [17], [44], and compiler analysis and optimizations [33], [34]. Their common theme is that machine learning can effectively be used to address problems whose optimal solutions are unknown or exact solutions are computationally too expensive due to their high complexity. In line with the prior works, our work demonstrates that RL can effectively be used to solve the model slicing and execution planning problem for real-time inference workloads in order to maximize their efficiency while meeting their deadlines.

Prior works have explored the design and implementation of hardware accelerators for deep learning (DL) [11], [14], [16], [18], [20], [23], [25], [46], [48]. As hardware accelerators for DL become more widely-used and diverse, we believe that systems like HERTI will become even more crucial to effectively utilize all the heterogeneous computing devices in the underlying system with heterogeneity and constraint awareness for efficient real-time inference.

VII. Conclusions

This paper presents HERTI, a reinforcement learning (RL)-augmented system for efficient real-time inference on heterogeneous embedded systems. HERTI employs the accurate and efficient execution and communication cost estimators to significantly accelerate the training process. HERTI builds on the state-of-the-art RL algorithm (i.e., deep Q-network) to eliminate the state space explosion problem. HERTI efficiently explores the state space with heterogeneity and constraint awareness and robustly generates the efficient state for the target inference workload with a strong deadline guarantee through RL. Our quantitative evaluation based on the widely-used inference workloads and heterogeneous embedded system demonstrates the effectiveness of HERTI in the sense that it achieves high efficiency and significantly outperforms the deadline-conscious state-of-the-art technique (i.e., AutoScale) in multiple metrics (i.e., energy, energy-delay product) with a strong deadline guarantee, robustly satisfies the deadline constraint in contrast to the deadline-oblivious state-of-the-art technique (i.e., MOSAIC), delivers larger efficiency gains as the inference deadline and the system heterogeneity increase, exhibits the strong generality for hyper-parameter tuning, and significantly reduces the training time based on its estimation-based approach across all the evaluated inference workloads and scenarios.

Acknowledgements

This research was partly supported by NRF (NRF-2021R1A2C1011482) and IITP (No.2018-0-00769, No.2021-0-01817). Woongki Baek is the corresponding author.

References

[1] https://www.hisilicon.com/en/products/Kirin, accessed: 2021-08-10.

[2] http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html, accessed: 2021-08-10.

[3] https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-2100/, accessed: 2021-08-10.

[4] https://www.qualcomm.com/products/snapdragon-888-5g-mobile-platform, accessed: 2021-08-10.

[5] https://www.apple.com/newsroom/2020/09/apple-unveils-all-new-ipad-air-with-a14-bionic-apples-most-advanced-chip/, accessed: 2021-08-10.

[6] https://www.96boards.org/product/hikey970/, accessed: 2021-08-10.

[7] "High voltage power monitor," https://www.msoon.com/, accessed: 2021-08-10.

[8] "Tensorflow lite," https://www.tensorflow.org/lite/, accessed: 2021-08-10.

[9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, ser. OSDI '16. GA: USENIX Association, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[10] A. Anderson and D. Gregg, "Optimal dnn primitive selection with partitioned boolean quadratic programming," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, ser. CGO 2018. New York, NY, USA: ACM, 2018, pp. 340–351. [Online]. Available: http://doi.acm.org/10.1145/3168805

[11] S. Angizi, Z. He, and D. Fan, "Dima: A depthwise cnn in-memory accelerator," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18. New York, NY, USA: ACM, 2018, pp. 122:1–122:8. [Online]. Available: http://doi.acm.org/10.1145/3240765.3240799

[12] S. Bateni, H. Zhou, Y. Zhu, and C. Liu, "Predjoule: A timing-predictable energy optimization framework for deep neural networks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, Dec 2018, pp. 107–118.

[13] X. Chen, D. Z. Chen, and X. S. Hu, "modnn: Memory optimal dnn training on gpus," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, ser. DATE'18, March 2018, pp. 13–18.

[14] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 367–379. [Online]. Available: https://doi.org/10.1109/ISCA.2016.40

[15] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and qos-aware cluster management," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 127–144. [Online]. Available: http://doi.acm.org/10.1145/2541940.2541941

[16] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "Dracc: A dram based accelerator for accurate cnn inference," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: ACM, 2018, pp. 168:1–168:6. [Online]. Available: http://doi.acm.org/10.1145/3195970.3196029

[17] T. D. Doudali, S. Blagodurov, A. Vishnu, S. Gurumurthi, and A. Gavrilovska, "Kleio: A hybrid memory page scheduler with machine intelligence," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 37–48. [Online]. Available: https://doi.org/10.1145/3307681.3325398

[18] S. Ghodrati, H. Sharma, S. Kinzer, A. Yazdanbakhsh, J. Park, N. S. Kim, D. Burger, and H. Esmaeilzadeh, "Mixed-signal charge-domain acceleration of deep neural networks through interleaved bit-partitioned arithmetic," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 399–411. [Online]. Available: https://doi.org/10.1145/3410463.3414634

[19] M. Han, J. Hyun, S. Park, J. Park, and W. Baek, "Mosaic: Heterogeneity-, communication-, and constraint-aware model slicing and execution for accurate and efficient inference," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2019, pp. 165–177.

[20] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 243–254. [Online]. Available: https://doi.org/10.1109/ISCA.2016.30

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision – ECCV 2016*, ser. ECCV'16, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 630–645.

[22] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '17. New York, NY, USA: ACM, 2017, pp. 82–95. [Online]. Available: http://doi.acm.org/10.1145/3081333.3081360

[23] R. Hwang, T. Kim, Y. Kwon, and M. Rhu, "Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations," in *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ser. ISCA '20. IEEE Press, 2020, p. 968–981. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00083

[24] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon, "Ionn: Incremental offloading of neural network computations from mobile devices to edge servers," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '18. New York, NY, USA: ACM, 2018, pp. 401–411. [Online]. Available: http://doi.acm.org/10.1145/3267809.3267828

[25] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: ACM, 2017, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080246

[26] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. New York, NY, USA: ACM, 2017, pp. 615–629. [Online]. Available: http://doi.acm.org/10.1145/3037697.3037698

[27] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack problems*. Springer, 2004. [Online]. Available: https://doi.org/10.1007/978-3-540-24777-7

[28] S. Kim, N. Pham, W. Baek, and Y.-r. Choi, "Holistic vm placement for distributed parallel applications in heterogeneous clusters," *IEEE Transactions on Services Computing*, pp. 1–1, 2018.

[29] Y. G. Kim and C. J. Wu, "Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1082–1096.

[30] A. Klimovic, H. Litz, and C. Kozyrakis, "Selecta: Heterogeneous cloud storage configuration for data analytics," in *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '18. USA: USENIX Association, 2018, p. 759–773.

[31] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: Bandit-based configuration evaluation for hyperparameter optimization," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=ry18Ww5ee

[32] C.-J. M. Liang, H. Xue, M. Yang, L. Zhou, L. Zhu, Z. L. Li, Z. Wang, Q. Chen, Q. Zhang, C. Liu, and W. Dai, "Autosys: The design and operation of learning-augmented systems," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Jul. 2020, pp. 323–336. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/liang-mike

[33] C. Mendis, A. Renda, S. Amarasinghe, and M. Carbin, "Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks," in *Proceedings of the 36th International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, Jun 2019, pp. 4505–4515. [Online]. Available: http://groups.csail.mit.edu/commit/papers/19/ithemal-icml.pdf

[34] C. Mendis, C. Yang, Y. Pu, D. Amarasinghe, and M. Carbin, "Compiler auto-vectorization with imitation learning," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019, pp. 14 625–14 635. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/d1d5923fc822531bbfd9d87d4760914b-Paper.pdf

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[36] R. Nishtala, P. Carpenter, V. Petrucci, and X. Martorell, "Hipster: Hybrid task manager for latency-critical cloud workloads," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, ser. HPCA '17, Feb 2017, pp. 409–420.

[37] R. Nishtala, V. Petrucci, P. Carpenter, and M. Sjalander, "Twig: Multi-agent task management for colocated latency-critical cloud services," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 167–179.

[38] Y. H. Oh, Q. Quan, D. Kim, S. Kim, J. Heo, S. Jung, J. Jang, and J. W. Lee, "A portable, automatic data qantizer for deep neural networks," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '18. New York, NY, USA: ACM, 2018, pp. 17:1–17:14. [Online]. Available: http://doi.acm.org/10.1145/3243176.3243180

[39] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[40] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," in *ICLR*, 2017.

[41] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-49. Piscataway, NJ, USA: IEEE Press, 2016, pp. 18:1–18:13. [Online]. Available: http://dl.acm.org/citation.cfm?id=3195638.3195660

[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vision*, vol. 115, no. 3, pp. 211–252, Dec. 2015. [Online]. Available: http://dx.doi.org/10.1007/s11263-015-0816-y

[43] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, ser. CVPR'18, June 2018.

[44] S. Sen and N. Imam, "Machine learning based design space exploration for hybrid main-memory design," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 480–489. [Online]. Available: https://doi.org/10.1145/3357526.3357544

[45] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[46] L. Song, Y. Wang, Y. Han, X. Zhao, B. Liu, and X. Li, "C-brain: A deep learning accelerator that tames the diversity of cnns through adaptive data-level parallelization," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 123:1–123:6. [Online]. Available: http://doi.acm.org/10.1145/2897937.2897995

[47] M. Song, Y. Hu, H. Chen, and T. Li, "Towards pervasive and user satisfactory cnn across gpu microarchitectures," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2017, pp. 1–12.

[48] Z. Song, F. Wu, X. Liu, J. Ke, N. Jing, and X. Liang, "Vr-dann: Real-time video recognition via decoder-assisted neural network acceleration," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 698–710.

[49] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[50] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI*, ser. AAAI'17. AAAI Press, 2017, pp. 4278–4284.

[51] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," *ArXiv e-prints*, Jul. 2018.

[52] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li, S. L. Song, Z. Xu, and T. Kraska, "Superneurons: Dynamic gpu memory management for training deep neural networks," in *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '18. New York, NY, USA: ACM, 2018, pp. 41–53. [Online]. Available: http://doi.acm.org/10.1145/3178487.3178491