Working Smarter Not Harder: Hybrid Cooling for Deep Learning in Edge Datacenters

Qiangyu Pei, Yongjie Yuan, Haichuan Hu, Lin Wang, *Senior Member, IEEE*, Dong Zhang, Bingheng Yan, Chen Yu, *Member, IEEE*, Fangming Liu*, *Senior Member, IEEE*

***Abstract*—The proliferation of deep-learning-based mobile and IoT applications has driven the increasing deployment of edge datacenters equipped with domain-specific accelerators. The unprecedented computing power offered by these accelerators puts a heavy burden on the cooling system, motivating more potent cooling techniques like cold water cooling. However, we observe that cold water cooling results in significant energy waste in edge datacenters due to the fluctuating resource utilization both spatially and temporally. To tackle this issue, we propose the concept of "working smarter" by slowing down accelerators deliberately whenever possible and enabling warm water cooling during these times to achieve cooling efficiency. Based on this concept, we develop Hyco—a hybrid water cooling system tailored for edge datacenters running deep learning workloads. First, Hyco features a zone-based cooling architecture enabling dynamic switching between cold water and warm water cooling. Then, based on a lightweight latency estimation method, Hyco incorporates a learning-based scheduling scheme to determine "which" accelerator workers and "when" to slow down through an adaptive and intelligent power-latency trade-off for deep learning models. The simulation with real-world traces shows that Hyco reduces the cooling energy consumption by up to 34.74× while satisfying latency constraints more than 99% of the time for deep-learning-based applications.**

***Index Terms*—Edge datacenters, water cooling, deep neural network, deep reinforcement learning, energy efficiency.**

I. Introduction

As a critical infrastructure for edge computing, edge datacenters are rapidly emerging in urban areas to support real-time services. Deloitte predicts that the global market for the intelligent edge is growing at a compound annual growth rate of around 35% [1]. As one of the representative edge workloads, deep neural network-based (DNN-based) services have been progressively deployed in the edge datacenters [2], such as edge-assisted augmented reality (AR) [3] and autonomous driving [4]. The recent advancement of large language models like ChatGPT [5] has further accelerated this trend. However, their widespread proliferation also prompts a significant increase in housing high-powered heterogeneous hardware at the edge [6], [7] from graphics processing units (GPUs) to application-specific integrated circuits (ASICs). Such an ever-increasing demand for computing power, joined by the end of Dennard scaling and the slowing of Moore's law, has escalated power consumption and carbon footprints of edge datacenters to unprecedented levels at the same time. Figure 1 shows the computing performance and thermal design power (TDP)[1] of representative Nvidia GPUs launched from 2008 to 2023. It is worth noting that the TDP remained stable in the past but started surging in 2018 as the performance grew.

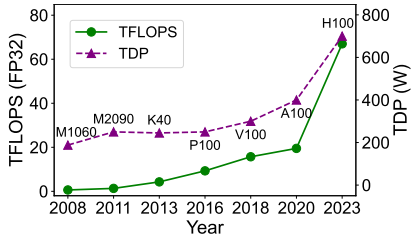


Fig. 1. The performance (TFLOPS) and power demand (TDP) of Nvidia GPUs. The text labels denote the GPU models.

Facing the rapidly growing power density, edge datacenters are getting increasingly hotter, which puts great pressure on efficient cooling. Extremely potent cooling techniques like cold water cooling are indispensable so as to timely take away the heat from hotter hardware though at the expense of high cooling energy. According to a recent report [8], the power usage effectiveness (PUE)[2] of a typical edge datacenter can reach as high as 2, much higher than that of cloud datacenters [10], e.g., 1.1 averagely achieved by Google in 2021 [11]. Figure 2 illustrates the cold water cooling architecture in detail. We can see that after absorbing heat from each hardware component through a cold plate, the water flows back to the centralized

This work was supported in part by the National Key Research & Development (R&D) Plan under grant 2022YFB4501703, and in part by The Major Key Project of PCL (PCL2022A05). Lin Wang was supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 210487104 - SFB 1053. (Corresponding author: Fangming Liu)

Q. Pei, Y. Yuan, H. Hu, and C. Yu are with the National Engineering Research Center for Big Data Technology and System, the Services Computing Technology and System Lab, Cluster and Grid Computing Lab in the School of Computer Science and Technology, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan 430074, China. E-mail: {peiqiangyu, huhc, yuchen}@hust.edu.cn, jayayuan@outlook.com

L. Wang is with Paderborn University and TU Darmstadt, Germany. E-mail: lin.wang@uni-paderborn.de

Dong Zhang and Bingheng Yan are with Jinan Inspur Data Co.,Ltd. E-mail: {zhangdong, yanbh}@inspur.com

F. Liu is with Huazhong University of Science and Technology, and Peng Cheng Laboratory, China. E-mail: fangminghk@gmail.com

Manuscript received xxxx xx, 2024; revised xxxx xx, 2024.

[1]TDP refers to the maximum amount of heat in watts that a processor can generate under the maximum load.

[2]PUE is an indicator used to reflect the energy efficiency of a datacenter and is defined as the ratio of the total facility energy and the IT equipment energy [9]. Therefore, a value closer to one means higher efficiency.

0000–0000/00$00.00 © 2024 IEEE

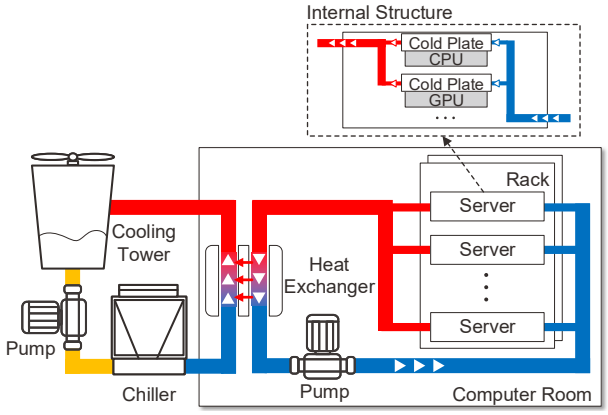


Fig. 2. The water cooling architecture in an edge datacenter.

cooling tower and/or the chiller to exchange heat and finally returns to the computer room with a much lower temperature. This temperature is usually set extremely low, e.g., 7∼10°C, to satisfy the cooling demands of all hardware components. To achieve high cooling efficiency, researchers have put forward *warm water cooling* by increasing the temperature of the chilled water to about 40∼45°C [12]. However, because of its limited cooling capacity, warm water cooling could be no more safe for hardware components running at a high power level like 1 kW of the latest Nvidia B200 GPU.

We observe that in edge datacenters, hardware components reach their maximum power only occasionally. Unlike cloud workloads, edge workloads are highly dependent on user behaviors: there would be diverse latency requirements as well as varying hardware utilization levels depending on user scenarios. For example, AR-enhanced head-up display for cars (AR-HUD for short) [13] imposes tight latency constraints to ensure driving safety, while AR-based indoor navigation (AR-NAV for short) can generally tolerate higher latency. Hardware utilization levels also fluctuate significantly and asynchronously: a high demand for transportation could lead to a high utilization level of the AR-HUD workload during rush hours, while the utilization level of the AR-NAV workload can grow high on weekends; both workloads will own very low utilization at night. A recent study on edge datacenters [14] confirms this phenomenon: there exists significant fluctuation in hardware utilization both spatially and temporally in edge datacenters due to the daily behaviors of end-users [15].

Besides these *diurnal* utilization fluctuations, there also exist *second-level* and even *sub-second-level* fluctuations, owing to different amounts of computations within and among DNN models when performing inference. For example, the FLOPs of each layer in the YOLOv3 model [16] varies between 262,144 and 303,300,608 (the difference is over 1,000×), and its power draw fluctuates between less than 80 W and 250 W during an inference process on an Nvidia Titan Xp GPU. Since the layer composition of a model varies a lot, not to mention the difference in power demands of different models. Recent literature develops an adaptive batching technique for inference serving [17], [18], while different batch sizes further increase the power fluctuation. For example, the average power draw of ResNetV1-101 grows from around 133 W to 211 W when the batch size increases from 1 to 5. Such characteristics of edge workloads result in a severe *power-cooling mismatch issue* both *spatially* and *temporally*. However, in view of the coarse-grained cooling architecture and unavoidable cooling delay, existing water cooling systems cannot solve the issue by adjusting the water temperature dynamically according to the runtime power levels of each hardware component. Recent studies on fine-grained cooling architectures [12], [19] also face limitations, such as limited scalability for high-powered accelerators [12] and a lack of support for managing temporal power-cooling mismatches during second-level and sub-second-level power fluctuations in inference workloads [19].

To improve the PUE by employing warm water cooling at non-peak spaces and times, we propose the concept of **"working smarter not harder"** to reach a **"sweet point"** revealed in Section II-C, by slowing down accelerators for a portion of workloads deliberately, based on a fact that limited performance drop is acceptable during non-peak periods as long as their latency requirements are satisfied. As shown later in Figure 5, by imposing a power limit on hardware components, the cooling efficiency can be improved by a significant margin by using cooling tower-based warm water cooling. However, we need to carefully decide *which* accelerator workers and *when* these workers can be slowed down, in order not to affect the quality-of-service (QoS) of edge workloads, i.e., inference latency. To address these challenges, we propose Hyco, a HYbrid water COoling system that improves the PUE by making smart use of warm water cooling in edge datacenters. Specifically, we make the following four contributions:

- We identify critical challenges imposed by edge inference workloads on efficient cooling which behave much differently from traditional cloud workloads.
- We propose a concept of "working smarter not harder" in edge datacenters to reach a sweet point. Based on that, we design a valve-controlled zone-based water cooling architecture to make smart use of cooling tower-based warm water cooling together with chiller-based cold water cooling.
- To determine which workers and when to slow down, we develop an inference latency estimation method and design a lightweight learning-based scheduling scheme. The scheduler evaluates the energy demand and timing pattern of edge workloads, and chooses the best candidate worker cooled by either warm or cold water with performance-aware power capping.
- We build a hardware prototype to collect thermal data under various cooling conditions and evaluate Hyco through simulations using real-world traces. The evaluation results show that Hyco achieves at most 34.74× cooling energy reduction, with a latency violation rate of only 0.67%. The estimation for 2,000 small-scale edge datacenters indicates that Hyco can reduce carbon footprints by about 4.8 kt CO_2 and monetary costs by $4,874,000 annually, compared to the conventional water cooling system.

The rest of the paper is organized as follows. In Section II, we present the background of edge datacenters and analyze

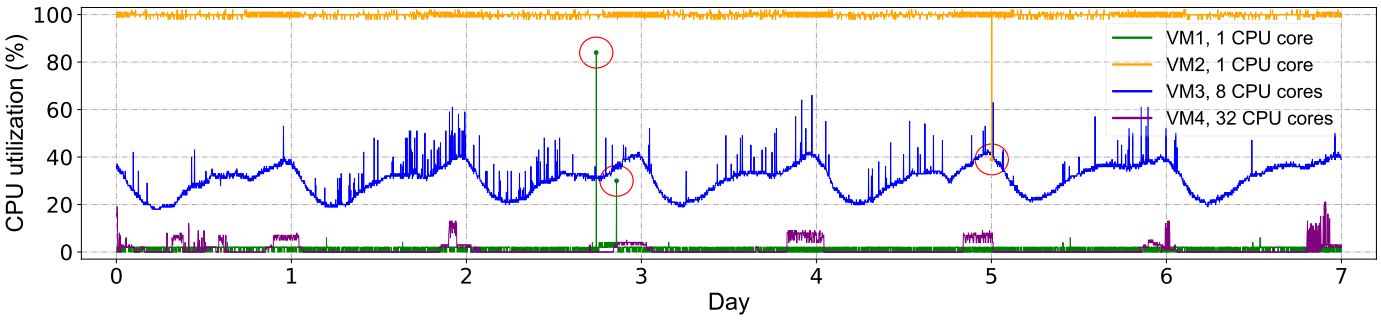


Fig. 3. CPU utilization variations of four representative VMs during seven days in edge datacenters (every two minutes) [14]. The red circles indicate two utilization spikes and one plummet. Note that there are actually more spikes and plummets since we sample the trace for readability.

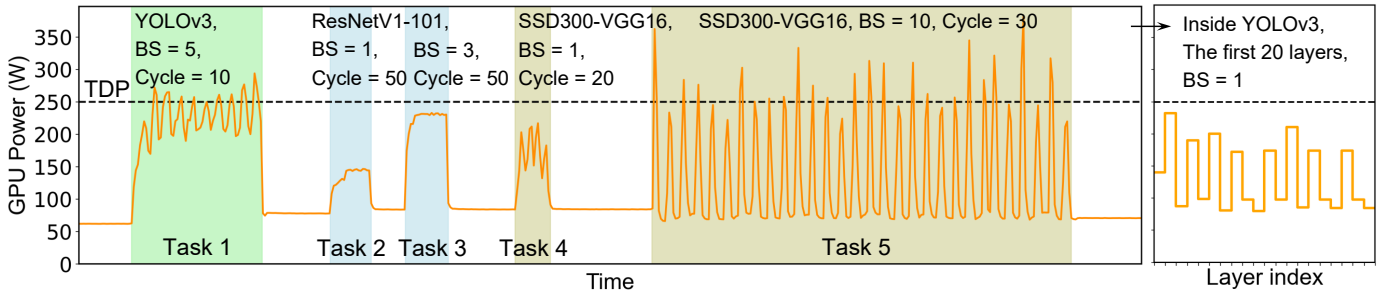


Fig. 4. Power fluctuations of five different inference tasks. The colored and non-colored periods indicate an inference task execution and idle states waiting for new tasks, respectively. "BS" is short for the batch size, and "Cycle" denotes inference times.

critical challenges in cooling. To address the challenges, in Section III, we propose a hybrid water cooling system tailored to edge datacenters named Hyco, present a DNN inference latency estimation method, and provide the problem formulation of energy-efficient DNN inference scheduling. Then, in Section IV, we design a learning-based scheme to solve this problem in an intelligent way. The evaluation results are presented in Section V. At last, we discuss the related work in Section VI and conclude our work in Section VII.

II. Background and Motivation

In this section, we first describe three characteristics of edge datacenters. Then, we analyze representative cooling techniques as well as their limitations for edge datacenters. Finally, we motivate the idea of making smart use of warm water cooling to address these limitations based on new observations.

A. Characteristics of Edge Datacenters

Extending the cloud, edge datacenters are distributed at the network edge to provide latency-critical computing services, especially AI-centric intelligent services, to end-users. Hence, the characteristics of edge datacenters are highly dependent on user behaviors [15]. Comparing them with cloud datacenters, we summarize three key distinct characteristics of edge datacenters revealed by a recent measurement study [14] and our further measurements.

Higher performance requirements. As compared to cloud applications, edge applications tend to be deployed on slightly more virtual machines (VMs) and each VM is allocated with much more hardware resources due in large part to the higher performance requirements, such as low delay and high reliability [14]. For example, about 10% of the edge applications use no fewer than 50 VMs but this number is only 6.1% for cloud applications. Moreover, the median of CPU cores allocated to a VM is eight and one in the studied edge and cloud datacenters, respectively, and over 60% VMs in the edge datacenters require more than four CPU cores, while the number is only 10% in the cloud.

Lower utilization levels. Although the amount of resources allocated to each edge application is much higher than that for cloud applications, the overall edge hardware utilization levels are lower. For example, the average hardware utilization of as many as 74% VMs is less than 10% in the edge datacenters, while it is only 47% in the cloud. Despite higher capital and operating costs, edge service providers usually over-provision hardware resources possibly due to the higher performance requirements of edge applications and a lack of understanding of their specific resource demands [14].

Larger utilization fluctuation. The significant impact of user behaviors makes the hardware utilization fluctuate significantly in edge datacenters. For instance, the median number of the variation coefficient of hardware utilization in edge datacenters is twice that in cloud datacenters. Based on the measurement results in [14], we plot the CPU utilization variations of four representative VMs running in the edge datacenters in Figure 3. For most VMs, the CPU utilization remains at a relatively low level generally and grows to a higher level periodically, e.g., from 23:00 to 24:00 every day when users tend to play on their mobile phones. For VM2, however, the CPU utilization remains high all the time. There

are also many unexpected spikes or plummets. For example, as indicated by the circles in Figure 3, the utilization of VM1 increases abruptly to more than 80% at the end of the third day, and the utilization of VM2 suddenly decreases to less than 40% at the beginning of the sixth day. In addition to the above *diurnal* utilization fluctuations, we notice that there are significant *second-level* and even *sub-second-level* fluctuations of edge inference workloads. We conduct an experiment using three representative DNN models on our local server equipped with an Nvidia Titan Xp GPU and monitor GPU power variation during inference executions with the `py3nvml` tool. As plotted in Figure 4, different models and internal layers, as well as batch sizes, will all lead to very different power demands that fluctuate over time.

The above three characteristics place a heavy burden on efficient cooling for edge datacenters. Specifically, the high-performance demands are driving edge datacenters to deploy more domain-specific accelerators like GPUs and other ASICs, and their astonishing TDPs increase the cooling demands. The large utilization fluctuation and low utilization levels further cause a severe power-cooling mismatch issue, which will be discussed in the following.

B. Cooling for Edge Datacenters

The most commonly-used cooling techniques in cloud datacenters include *free cooling*, *air cooling*, *water cooling*, and *immersion cooling*. However, the former two techniques are no more suitable for edge datacenters [19]. First, free cooling relies on the natural environment to provide free cooling sources, such as cold air and lake water, which cannot be easily accessed by edge datacenters located in the urban. Second, air cooling adopts hot aisle and cold aisle containment to realize air circulation, which is no more suitable for edge datacenters with ever-increasing power density due to its low cooling capacity [20]. Recently, water cooling and immersion cooling have gradually taken the place of air cooling due to their higher cooling efficiency and larger cooling capacity. However, recent studies [21]–[23] also highlight significant challenges in server reliability and maintenance under immersion cooling. Thus, we argue that water cooling, as a more mature technique, is currently a promising solution for edge datacenters. As shown in Figure 2, in a water-cooled edge datacenter, the chilled cooling water directly flows across each hardware component to absorb its generated heat through an attached cold plate. Then, the heated water flows back to the cooling tower and/or chiller and gets chilled again.

The use of the cooling tower and/or chiller depends on the environment and water temperature. When the environment temperature is low, the cooling tower alone can cool down the water by evaporating a small volume of water. Then, the chiller can be turned off if the required cooling water temperature is comparable to the environment temperature. On the contrary, when the environment temperature exceeds the required water temperature, e.g., during hot days or when requiring excessively cold cooling water, the chiller is indispensable in removing the remaining heat with a significantly higher energy footprint. Therefore, researchers have recently

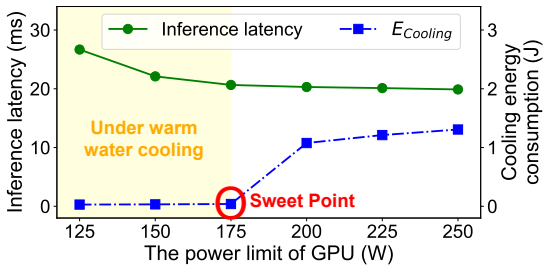


Fig. 5. The impact of GPU power limits on the inference latency and cooling energy.

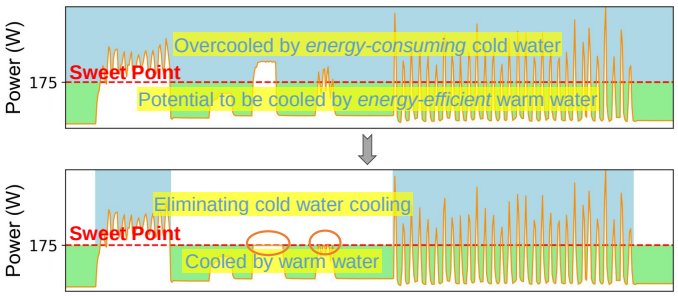


Fig. 6. Capping the power to cross the "sweet point". The orange circles denote performance-aware power capping, resulting in only slightly higher latency.

proposed warm water cooling to improve PUE by reducing the use of the chiller [12], [24]. However, warm water cooling not only aggravates the hardware safety-performance trade-off but also has a hard time satisfying the cooling demand of emerging high-powered accelerators housed by the edge datacenters.

As presented in Section II-A, the hardware utilization varies considerably both spatially and temporally, and thus, a large number of accelerators, especially when underutilized, are overcooled under cold water cooling. To deal with this power-cooling mismatch issue, Jiang et al. [12] propose a thermoelectric cooler-based (TEC-based) warm water cooling solution, i.e., using warm water for global cooling and TECs to cool down some overutilized processors. However, this solution cannot be easily extended to high-powered accelerators due to the installation problem and limited cooling capacity of TECs [19]. Recently, Pei et al. [19] advance a fine-grained water cooling architecture for heterogeneous hardware, such as CPUs and GPUs, in edge datacenters, which addresses the power-cooling mismatch issue by providing customized cooling water through valves according to the cooling demand of each hardware component. However, it is somewhat costly to install control valves for every hardware component, and it still cannot handle temporal power-cooling mismatches under the second-level and sub-second-level power fluctuations of inference workloads as illustrated in Figure 4.

In addition to water cooling, there are numerous studies on solving the power-cooling mismatch issue in air-cooled [25]–[31] and immersion-cooled [21], [22] systems through thermal-aware workload scheduling, power throttling, and/or cooling control. However, as illustrated in Figure 2, the cooling mechanism of water cooling differs significantly from that of air or immersion cooling, making these approaches

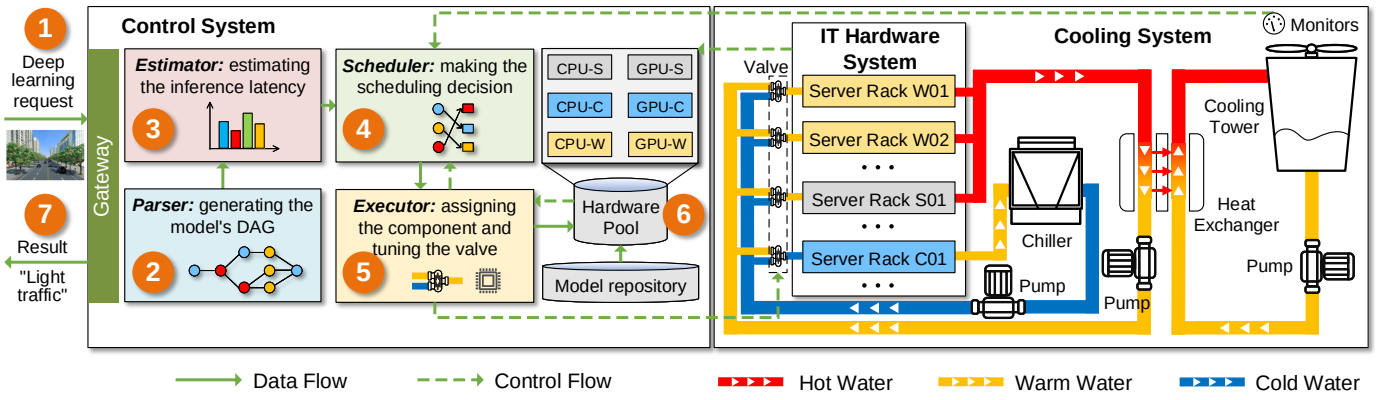


Fig. 7. System overview of Hyco in an edge datacenter.

inapplicable to water-cooled edge datacenters. Moreover, these studies primarily focus on conventional cloud workloads and cannot effectively manage user-facing inference workloads that have stringent performance requirements and exhibit significant power fluctuations at second and sub-second levels. This highlights the need for a tailored approach that accounts for the specific characteristics of inference workloads to improve cooling efficiency.

C. Working Smarter Not Harder

DNN inference has become a core component in various intelligent edge applications [32], [33]. While existing literature claims to achieve extreme hardware performance when executing these inference tasks at the edge, we observe that although many deep learning applications impose strict performance constraints and demand high computing power as much as possible, others only report relaxed requirements [34]. Such characteristics can be exploited to save computing energy by adjusting hardware performance based on the specific needs of each application. However, the energy saving is generally trivial and marginal considering the near-linear relationship between hardware performance and energy consumption [27].

Besides hardware energy, we observe a significant cooling energy reduction by switching from chiller-based cold water cooling to cooling tower-based warm water cooling under a lower power limit of computing hardware. As shown in Figure 5, for the Unet model running on an Nvidia GPU, when the hardware power limit decreases from 250 W to 125 W, the cooling energy decreases by up to 97.8% by switching to warm water cooling, while the inference latency increases from 20 ms to 27 ms (i.e., 35% higher) somewhat undesirably. Interestingly though, as observed, the maximum benefit arises primarily from crossing the **"sweet point"**, which is defined as the transition point from relying on energy-consuming cold water cooling to allowing energy-efficient warm water cooling, achieved by tuning the hardware power limit appropriately. In the example given in Figure 5, just allowing warm water cooling can bring remarkably 96.4% cooling energy reduction while incurring only 1.7% latency increase (i.e., 0.35 ms). After crossing the sweet point, reducing the power limit aggressively only brings marginal benefits but incurs relatively high performance degradation.

To this end, rather than setting the default power limit (i.e., TDP) all the time or aggressively reducing the power limit which sacrifices the performance severely, we propose to impose a power limit just below the "sweet point" with minor performance degradation for applications with relaxed requirements. As illustrated in Figure 6 (using the same trace as in Figure 4), after crossing the sweet point, the *energy-consuming* chiller-based cold water [35] can be eliminated for Tasks 2, 3, and 4, and the *energy-efficient* cooling tower-based warm water [36] is adequate to preserve desired performance for them. By contrast, in the case that Task 5 owns relatively high power demands and Task 1 imposes a tight latency requirement, the two corresponding components should still be cooled by the cold water.

To achieve adaptive while cost-effective switching between cold water and warm water cooling, we develop a zone-based cooling architecture and a performance-aware scheduling scheme that handle both spatial and temporal cooling mismatches. Now, Tasks 2, 3, and 4 in Figure 6 may be scheduled to the warm water cooling zone. The scheduler is responsible for making such decisions, i.e., which accelerator workers and when to slow down under the concept of "working smarter not harder".

III. Hyco: A Hybrid Water Cooling System

In this section, we present Hyco, a hybrid water cooling system for deep learning in edge datacenters. We begin with the system overview, and then elaborate on a DNN inference latency estimation scheme. Finally, we formulate the energy-efficient DNN inference scheduling problem.

A. System Overview

Figure 7 shows the system overview of Hyco. There are three core parts involved, i.e., the cooling system, IT hardware system, and control system. Based on the observations presented in Section II-C, we design a zone-based water cooling architecture and formulate a DNN inference scheduling problem with the goal of improving energy efficiency.

(1) **Cooling system.** A zone-based water cooling architecture is developed to make smart use of both cold water cooling and warm water cooling as shown in the right block of

Figure 7. At the inlet of each server rack, an L-type three-way ball valve [37] is installed to dynamically select either warm water or cold water to flow into the water pipes of the rack. Provided with cold water chilled by the chiller, all components are able to run at their maximum power, i.e., TDP, without the risk of overheating. On the contrary, for other components cooled by warm water without active chilling, there must be a power limit to guarantee the hardware safety with a slight performance drop. The power limit is set according to the sweet point.

(2) **IT hardware system.** In a small-scale edge datacenter, there are usually only a limited number of server racks, and each rack contains a number of high-powered servers with heterogeneous hardware components, including CPUs, GPUs, and other accelerators. Based on the cooling water type and server state, we divide the server racks into three types logically, i.e., cold-water-active (C), warm-water-active (W), and warm-water-sleeping (S), and the server racks are numbered with the letter W, C, or S at the beginning. It is worth noting that the sleeping state can reduce idle power to near zero as compared to the active state [38]. Moreover, based on the server type and hardware type, the hardware pool can be divided into six groups as shown in Figure 7, i.e., CPU or GPU followed by the letter of either C or W or S.

(3) **Control system.** As shown in Figure 7, once receiving an inference request, Hyco needs to estimate the inference latency on different components before scheduling to ensure the latency constraint. First, the *parser* parses the model's architecture information to generate a `.cfg` file that contains each layer's type and parameters, and connections among layers in the computational directed acyclic graph (DAG). Next, the *estimator* estimates the model's inference latency based on the estimated latency of all its layers and their connections. Then, the *scheduler* dispatches the inference task through performance-aware power capping to make smart use of warm water cooling, based on the predicted inference latency on each type of hardware, the predefined latency requirement, the remaining idle hardware resources, and the current "sweet point" value that is influenced by the ambient environment and determined based on outdoor temperature monitors. Finally, the *executor* executes the task on the assigned hardware component and tunes the valve as needed. In the following, we will dig into the key design of the *estimator* and *scheduler*.

B. Estimator: Lightweight Latency Estimation

There are already several schemes to get the inference latency of DNN layers and models, including *profiling offline* [39], *profiling at run time* [40], and *leveraging linear regression models to make estimations* [41]. While profiling offline beforehand and profiling at run time can yield accurate inference latency values, they come with significant drawbacks. Specifically, profiling models offline and storing all the model profiles in a database can lead to substantial storage costs, high updating costs, and long searching time as the number of DNN models grows to thousands or even tens of thousands [42]. For profiling models at run time when a new request arrives, there would be significant system delays and heavy computation burden by executing the inference for repeated times (e.g., 20) on each hardware type. In comparison, Neurosurgeon [41] proposes an estimation method with linear regression models. Although there is negligible runtime overhead, this method can incur high estimation errors according to our experimental results. For instance, our trials show that the relative error for just a single convolutional layer on GPU already reaches as high as 70.3%, making this method completely unworkable in our scenarios requiring strict latency guarantees.

To estimate DNN layers' and models' inference latency in real time and accurately, we first make extensive measurements of common layer types under different parameters to study their runtime characteristics. We notice that the relationships between the inference latency and layer parameters are intricate and not simply linear or polynomial in a large range. To effectively learn such intricate relations, we apply a series of lightweight, piecewise two-layer DNN models to make the estimations. Specifically, we leverage the Profiler tool [43] in TensorFlow [44] to collect the inference latency information of each layer type with different parameters. Then, we divide the measurement results into several statistic regions and analyze the empirical relationships between the inference latency and the layer parameters in each region separately. For regions with similar relationships, we merge them together to reduce the number of stored estimation models. Finally, for each merged statistic region, we apply a two-layer lightweight DNN model to learn such a relationship, which achieves relatively high accuracy. After estimating the inference latency of each layer, we leverage the *critical path analysis* to get the model's inference latency from the first layer to the end, based on the computation DAG of the DNN model generated by the *parser*.

C. Scheduler: Making Smart Use of Warm Water Cooling

In an edge datacenter supporting real-time DNN inference, there is a sequence of user requests $U = \{u_1, u_2, ..., u_i, ..., u_{|U|}\}$ arriving successively during a time period T, and each of the user requests needs to be scheduled to a non-sleeping component $h \in H$, including CPU-W, CPU-C, GPU-W, and GPU-C. To avoid performance interference among tasks, we assume that each component only conducts one task at a time as prior work [45] does. In the case that there is no available non-sleeping component to serve requests, another server rack in the sleeping state needs to be switched on. As shown in Figure 7, since all the servers in the same rack share the same power strategy (i.e., switch on or off) and cooling strategy (i.e., warm water or cold water cooling), even if only one server within a rack needs to be switched on and becomes active, all the rest will wake up and consume idle energy and corresponding cooling energy. We use $b_{h,t}$ to denote the number of each type of non-sleeping components $h \in H$ including in either the active or the idle state in each time slot $t \in T$, and M_h to denote the number of hardware components of type $h \in H$ in each rack, where $b_{h,t}$ must be a multiple of M_h.

For each user request $u_i \in U$, there are starting time g_i, finishing time f_i, and latency requirement q_i of a single DNN inference execution. The arrival rate of request $u_i \in U$ can

TABLE I
NOTATIONS

Symbol	Description
U	The set of user requests, where $u_i \in U$ is the i-th incoming request
T	The total time span of all the user requests U, $t \in T$ and $\|T\| = \max\{f_i\} - 1$
H	The set of heterogeneous IT hardware type, $h \in H$ where $\|H\|$ is an even number. An odd and even h refers to hardware in the warm water and cold water cooling zones, respectively. For example, $h = 1$, 2, 3, and 4 refer to CPU-W, CPU-C, GPU-W, and GPU-C, respectively, when $\|H\| = 4$
M_h	# of hardware components of type $h \in H$ in a rack
g_i	The starting time of the task $u_i \in U$
f_i	The finishing time of the task $u_i \in U$
$c_{i,h}$	The inference latency of a single DNN inference execution when the task $u_i \in U$ is placed on the component of type $h \in H$
$e_{i,h}$	The energy consumption of a single DNN inference execution when the task $u_i \in U$ is placed on the component of type $h \in H$
q_i	The latency requirement of the task $u_i \in U$
$x_{i,h}$	$x = 1$ if the task $u_i \in U$ is placed on the component of type $h \in H$, $x = 0$ otherwise
$b_{h,t}$	The number of each type of non-sleeping components $h \in H$ in each time slot $t \in T$
p_h^{idle}	The idle power of the component of type $h \in H$
η_h	The cooling coefficient of the component of type $h \in H$

be expressed by $1/q_i$ in general, and the total number of DNN inference executions from this request can be calculated as $\lfloor (f_i - g_i)/q_i \rfloor$, where $\lfloor \cdot \rfloor$ is the floor operation. We use $c_{i,h}$ and $e_{i,h}$ to indicate the inference latency and energy consumption of one inference execution on the component of type $h \in H$. To avoid latency violations, $c_{i,h}$ should be no more than q_i when $u_i \in U$ is placed on the component of type $h \in H$. Then, we use $x_{i,h}$ to represent whether $u_i \in U$ is placed on the component of type $h \in H$, and $b_{h,t}$ should be larger than the number of all allocated tasks $\sum_{i:g_i \le t < f_i} x_{i,h}$ in each time slot $t \in T$. To dissipate all the heat generated by the component, different proportions of cooling energy to IT hardware energy will be consumed, which is expressed by the cooling coefficient $\eta_h = \frac{\text{Cooling Energy}}{\text{IT Hardware Energy}} = \text{pPUE} - 1$[3] for the component of type $h \in H$, and $\eta_1 = \eta_3$ and $\eta_2 = \eta_4$. We use p_h^{idle} to represent the idle power of CPUs and GPUs, where $p_1^{idle} = p_2^{idle}$ and $p_3^{idle} = p_4^{idle}$. The notations are listed in Table I.

The optimization objective is to minimize the overall energy consumption of both IT hardware and cooling equipment with latency guarantees. Note that the IT hardware components in the active state and idle state all consume energy. We use $E_{IT_{active}}$, $E_{IT_{idle}}$, and $E_{Cooling}$ to denote the energy consumption of IT hardware in active states, IT hardware in idle states, and cooling equipment, respectively, all of which are highly related to the sequence of the user requests U and their placement $x_{i,h}$. To calculate $E_{IT_{idle}}(x_{i,h})$, we have to quantify the running tasks on each type of heterogeneous hardware $h \in H$ in each time slot $t \in T$. In other words, the values of both g_i and f_i of each request u_i are used to compute $E_{IT_{idle}}(x_{i,h})$. Here, we apply timing series analysis to calculate $E_{IT_{idle}}(x_{i,h})$. On the contrary, $E_{IT_{active}}(x_{i,h})$ is only related to the service time $(f_i - g_i)$ of each request u_i instead of the specific values of g_i and f_i. The derivation procedure is presented as follows:

$$
\begin{aligned}
&E_{IT} + E_{Cooling} \\
=&E_{IT_{active}}(x_{i,h}) + E_{IT_{idle}}(x_{i,h}) + E_{Cooling}(x_{i,h}) \\
=&(E_{IT_{active}}(x_{i,h}) + E_{IT_{idle}}(x_{i,h}))(1+\eta_h) \\
=&\sum_{i=1}^{n}\sum_{h=1}^{|H|} x_{i,h}\Big((e_{i,h} + p_h^{idle}(q_i - c_{i,h}))\lfloor (f_i - g_i)/q_i \rfloor \\
&+ p_h^{idle}(f_i - g_i - q_i\lfloor (f_i - g_i)/q_i \rfloor)\Big)(1+\eta_h) \\
&+ \sum_{t=0}^{|T|}\sum_{i:g_i \le t < f_i}\sum_{h=1}^{|H|} p_h^{idle}(b_{h,t} - x_{i,h})(1+\eta_h). \qquad (1)
\end{aligned}
$$

Thus, the DNN inference scheduling problem can be formulated as follows:

$$\min\ E_{IT} + E_{Cooling} \qquad (2)$$

$$s.t.\ \ x_{i,h}(q_i - c_{i,h}) \ge 0, \forall i = 1, ..., n, h = 1, ..., |H|, \qquad (3)$$

$$b_{h,t} = \max_{h'=2j-h \bmod 2}\{\lceil \sum_{i:g_i \le t < f_i} x_{i,h'}/M_{h'} \rceil M_{h'}\},$$
$$\forall t = 0, ..., |T|, h = 1, ..., |H|, j = 1, ..., |H|/2, \qquad (4)$$

$$\sum_{h=1}^{|H|} x_{i,h} = 1, \forall i = 1, ..., n, \qquad (5)$$

$$x_{i,h} \in \{0, 1\}, \forall i = 1, ..., n, h = 1, ..., |H|. \qquad (6)$$

Equation (3) means that the real inference latency should be lower than the latency requirement. Equation (4) indicates the number of each type of non-sleeping components in each server rack in $t \in T$. Equations (5) and (6) together guarantee that one and only one hardware component executes each of the tasks.

Because of unknown future user requests and their mutual influence caused by activating all the components in the same rack, it would be impossible to search for an optimal solution at run time. One possible solution may be the greedy heuristic algorithm that is executed each time a new request arrives. However, if we just handle each user request one by one in a sequence greedily, the impact of the current decision on future decisions is neglected, bringing high idle energy and cooling energy consumption. In addition, the inference energy cannot be accurately accessed in advance by the greedy algorithm when making a scheduling decision, because of the closed-source GPU drivers and excessively low prediction accuracy if we just consider available layer parameters. As a result, it is vital for the scheduler to make each decision in real time with regard to its influence on future decisions and learn the true energy consumption without manual effort.

[3]The partial Power Usage Effectiveness (pPUE) is defined as (IT Energy + Cooling Energy) / IT Energy [12].

IV. Learning-based Scheduling Scheme

In this section, we design a learning-based agent as the scheduler. We first explain the rationale for preferring the learning-based scheme, and then introduce its five key concepts. In the end, we present the design details.

A. Motivation for Deep Reinforcement Learning

The scheduler is responsible for deciding the most efficient hardware component for each incoming inference request in real-time. Firstly, since it is almost impossible to estimate the inference energy accurately according to our trials, the scheduler needs to *extrapolate the potential inference energy on each hardware type based on the input model's configuration information*. Secondly, given the additional energy consumed by idle components when activating a rack, the scheduler needs to *evaluate the impacts of current scheduling decisions on the future by learning from historical workload patterns*. However, it is generally impractical to accurately know future requests at run time to make a global optimization indicated by Equations (2) to (6). Thirdly, conventional *rule-based schemes* basically rely on expert domain knowledge, which raises heavy manual efforts and may even be problematic if making many inaccurate assumptions [46]–[48], leading to inefficiency for diverse edge datacenters and mission-critical edge workloads.

As a *learning-based scheme*, reinforcement learning (RL) is promising in addressing the above challenges. It leverages an agent to learn and make decisions through trial and error. Deep RL (DRL) is a combination of RL and deep learning and uses a DNN for the decision-making process. The agent observes the state and then takes one of the actions with the highest Q value. The Q value comprises the reward R, and the possible maximum Q value of the next state s' and action a', i.e., $Q = R + \gamma \max_{a'} Q(s', a')$, where γ is the discount factor.

As one of the commonly used RL algorithms, Q-learning has been adopted widely for decision-making [49]. However, it can only handle discrete state space since it requires a table to store Q values of each state-action pair. In Hyco, to satisfy the latency requirement, estimation results of inference latency should be regarded as states. However, this continuous value would make Q-learning infeasible. In contrast, by replacing the table with the DNN, DRL is able to handle various unstructured inputs. As a representative, the deep Q-network (DQN) can handle continuous values of the state easily by adopting a DNN to process the state [50]. In this work, we leverage the double DQN (DDQN), a variant of the DQN, for the scheduling problem to accelerate the training process and avoid overfitting [51]. In the following, we map the scheduling problem in Section III-C into the DDQN algorithm to realize a lightweight while efficient agent.

B. Key Concepts

In DRL, there are five key concepts: agent, environment, state, action, and reward. Based on the system architecture introduced in Section III-A and the scheduling process by the scheduler presented in Section III-C, their implications are presented as follows:

TABLE II
Considered states of the DRL agent

State		Description
DNN-related state s_d	T_{CPU-W}	Estimated inference latency on CPU-W
	T_{CPU-C}	Estimated inference latency on CPU-C
	T_{GPU-W}	Estimated inference latency on GPU-W
	T_{GPU-C}	Estimated inference latency on GPU-C
	N_{CONV}	# of convolutional layers
	N_{POOL}	# of pooling layers
	N_{FC}	# of fully-connected layers
	N_{NORM}	# of normalization layers
	N_{AC}	# of activation layers
	N_{RC}	# of recurrent layers
User-related state s_u	q	The latency requirement imposed by the user
Hardware-related state s_h	N_{CPU-W}	# of remaining idle CPU-W
	N_{CPU-C}	# of remaining idle CPU-C
	N_{GPU-W}	# of remaining idle GPU-W
	N_{GPU-C}	# of remaining idle GPU-C

Agent. The agent is a decision-maker on the best-suitable hardware component for each user request.

Environment. The environment here refers to an edge datacenter providing DNN-based services, where there are a series of server racks containing heterogeneous computing hardware.

State (*S*). The state s includes DNN-related state s_d, user-related state s_u, and hardware-related state s_h. All the details are listed in Table II.

Action (*A*). The action a indicates where to process the user request, i.e., $h \in H$.

Reward (*R*). The reward r comprises the energy consumption (E) and whether the latency requirement q is satisfied (Q). Note that the energy consumption comes from not only the IT hardware $E_{self-IT}$ and cooling equipment $E_{self-cooling}$ for running the task itself, but also the IT hardware $E_{other-IT}$ and cooling equipment $E_{other-cooling}$ from the rest of servers within the same rack due to the need to switch on the whole server rack when there are no available candidate components. Thus, when u_i is placed on the hardware of type $h \in H$, i.e., $x_{i,h} = 1$, their expressions are given as follows:

$$\begin{aligned} E_{self_{IT}} =& (e_{i,h} + p_h^{idle}(q_i - c_{i,h}))\lfloor (f_i - g_i)/q_i \rfloor \\ &+ p_h^{idle}(f_i - g_i - q_i\lfloor (f_i - g_i)/q_i \rfloor), \quad (7) \\ E_{self_{cooling}} =& \eta_h E_{self_{IT}}, \quad (8) \\ E_{other_{IT}} =& \sum_{h':|h'-h| \bmod 2=0} p_h^{idle}(M'_h - x_{i,h'}) \\ &\cdot \max\{0, (f_i - \max_j\{f_j\})\}, \quad (9) \\ E_{other_{cooling}} =& \eta_h E_{other_{IT}}, \quad (10) \\ Q =& \ln(1 + \exp(100(c_{i,h} - q_i)/q_i)) \\ &\cdot c_{i,h}\lfloor (f_i - g_i)/q_i \rfloor, \quad (11) \end{aligned}$$

Algorithm 1 Training the DDQN algorithm for Hyco

1: Initialize: the training Q-network $Q(S,a;\theta)$ randomly with weight θ, target Q-network $\hat{Q}(S,a;\theta')$ with $\theta'=\theta$, learning rate λ, discounting factor γ, exploration probability ϵ, replay butter D, and minibatch size B.
2: **for** episode in $1,2,\ldots,E$ **do**
3: Initialize: extract user requests U from a training dataset and set $s_{h,0}=0$;
4: **for** $t=0,1,\ldots,|T|$ **do**
5: For all tasks finishing at t, update $s_{h,t}$ and $K_{l,m,t}$;
6: **while** a user request arrives at t **do**
7: Update $s_{d,t}$ and $s_{u,t}$;
8: With probability ϵ, select a_t from A randomly, otherwise select $a_t=\max_a Q(s_t,a;\theta)$;
9: Execute a_t, update $K_{l,m,t}$, calculate r_t via (12), and observe s_{t+1};
10: Store transition (s_t,a_t,r_t,s_{t+1}) into D;
11: Sample a random minibatch of B transitions $\{s_j,a_j,r_j,s_{j+1}\}$, $1\leqslant j\leqslant|D|$;
12: Set $y_j=\begin{cases} r_j, & \text{for terminal } j+1,\\ r_j+\gamma\hat{Q}\left(s_{j+1},\underset{a_{j+1}}{\operatorname{argmax}}\,Q(s_{j+1},a_{j+1};\theta);\theta'\right), & \text{otherwise;}\end{cases}$
13: Compute the stochastic gradient $\nabla_\theta L_t^\theta(\theta)$ on $L_t^\theta(\theta)=(y_j-Q(s_j,a_j;\theta))^2$;
14: Every C steps update $\hat{Q}=Q$;
15: Update $s_t=s_{t+1}$;
16: **end while**
17: Update s_{t+1};
18: **end for**
19: **end for**

where j denotes all the tasks u_j running in the same server rack. If $j\in\varnothing$, $\max_j\{f_j\}=g_i$. Note that in the expression of Q, we apply a "softplus" operation instead of $\max\{0,100(c_{i,h}-q_i)/q_i\}$, since it is differentiable and more effective for the training. Thus, the reward r for each scheduling decision of a user request u_i can be written as follows:

$$R=\ln\big(1/(\alpha(E_{self_{IT}}+E_{self_{cooling}}+E_{other_{IT}}+E_{other_{cooling}})+\beta Q)\big), \tag{12}$$

where α is the energy consumption weight and β is the latency violation weight. Their values depend on the importance of saving energy and satisfying latency requirements. To compute the reward, we also need to record all servers' states (active, idle, or sleeping) within each rack at each time slot $t\in T$, which is denoted as $K_{l,m,t}$, where l is the rack number, and m is the server number in the rack l.

C. Design Details with the DRL agent

There are three kinds of events that change the state s: request arriving, request scheduled, and request finishing. Algorithm 1 shows the training phase of the DDQN algorithm. At each time slot t, it firstly checks whether any tasks finish and updates the hardware-related state $s_{h,t}$ (Line 5). Then, for each arriving user request, it updates the DNN-related state $s_{d,t}$ and user-related state $s_{u,t}$ based on the DNN model structure, estimated inference latency, and latency requirement (Lines 6-7). Next, it makes the decision of a_t with the epsilon-greedy policy [52] (Line 8), calculates the reward r_t with Equation (12) and observe the next state s_{t+1} (Line 9), and stores the transition (s_t,a_t,r_t,s_{t+1}) for experience replay [53] (Line 10). For sampled transitions, it calculates the target denoted by y_j and performs a gradient descent operation that updates the training network (Lines 11-13). It then updates the target network every C steps (Line 14). Finally, the state s_t is updated to s_{t+1}(Line 15), and if no more requests arrive at t, s_{t+1} is updated instead (Line 17).

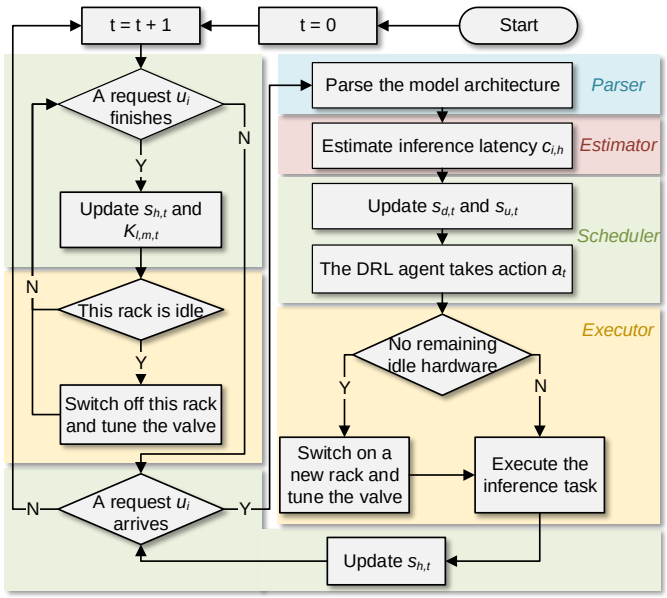


Fig. 8. Execution flow at run time.

Figure 8 shows the execution flow in the running phase. At each time slot t, Hyco handles all finishing tasks first, including updating the states and other system parameters and switching off racks if all internal servers become idle. After that, Hyco handles each request on arrival by parsing the model architecture, estimating the inference latency, and updating the related states. Next, the DRL agent takes an action of the best-suitable component. Note that there is only the trained target network in the running phase. Finally, Hyco dispatches the task to the candidate component, and switch on or off racks and tune the valve if necessary.

V. Evaluation

In this section, we conduct extensive simulations to evaluate Hyco with real-world traces. We first introduce the evaluation setup and then present the evaluation results compared to five baselines. Finally, we discuss the flexible trade-off provided by Hyco and quantify the scheduling and training overhead.

A. Evaluation Setup

IT hardware system. There are four types of heterogeneous hardware (i.e., $|H|=4$), including CPU-W, CPU-C, GPU-W, and GPU-C. We consider that there are two Intel Xeon E5-2697 v4 CPUs and two Nvidia Titan Xp GPUs in each server and ten servers in total in each rack. Each inference task will be placed on either two CPUs or one GPU within the same server, thus M_1, M_2, M_3, $M_4=10,10,20,20$,

respectively. The TDPs of the CPU and GPU are 145 W and 250 W, respectively. We set p_1^{idle} and p_2^{idle} for the two CPUs as 26 W, and p_3^{idle} and p_4^{idle} for the GPU as 10 W based on our measurements.

Cooling system. The cooling efficiency η_2 and η_4 for the cold water cooling are set as 0.26 [35] by using the chiller, and η_1 and η_3 for the warm water cooling are set as an excessively low value of 0.01 [36] when operating with only the cooling tower. The cold and warm water temperatures are set as 20°C and 45°C, respectively [12]. Raising the water temperature can reduce cooling energy consumption, but it also necessitates imposing stricter power limits on the hardware to prevent overheating and ensure safety. Therefore, the TDPs of the CPU and GPU under warm water cooling are limited to 40 W and 125 W (i.e., the sweet points), respectively, based on our measurement results from real hardware [12], [19].

Control system and deep learning models. The four hyperparameters of DRL, α, β, λ, and γ, are set as 1.0e^{-5}, 1.0e^{-5}, 1.0e^{-5}, and 0.9, respectively, based on a previous study on workload scheduling with DRL [29] and fine-tuning through small incremental trials to ensure rapid convergence. To achieve a better balance between exploration and exploitation during the DRL agent's learning process, the exploration probability ϵ is initialized at 0.3, and decreased by 1.07e^{-6} each time after choosing an action, which we find improves the convergence of the training process. The DRL agent employs a lightweight DNN model with two fully-connected layers to take an action for a state input. Considering the relatively small number of states and actions in our setup, the number of neurons in the first and second layers are configured as small values of 128 and 32, respectively [29], to avoid overfitting and minimize system overhead. We select seven representative DNN models for the inference tasks covering image classification (including ResNetV1-50, EfficientNet-B1, and EfficientNet-B3), semantic segmentation (including Unet), object detection (including YOLOv3-spp and YOLOv3-tiny), and named entity recognition (including BiLSTM). We use the measured values and estimated values of the inference latency for the DRL training and testing (i.e., running) phases, respectively.

Workload. We use deep learning traces collected from Alibaba Platform for Artificial Intelligence [54] to generate user requests. The trace contains information like the job name, starting time, ending time, and instance number of each deep learning task on over 6,500 GPUs from July to August in 2020. We randomly select 20 training datasets and 5 testing datasets each of which contains 4% of the data items (i.e., 36459 items each). We take the instance number per task as the batch size and scale it to 1∼10. We set the latency requirement q based on the Gaussian distribution in the range of $1.1 \cdot \min\{c_{i,h}\}$ to $1.5 \cdot \max\{c_{i,h}\}$, $\forall h = 1, ..., |H|$, to ensure that at least one hardware component can process the request with QoS guarantee.

Baselines. To evaluate Hyco in terms of both the zone-based water cooling architecture and the DRL-based scheduler, we choose five baselines as follows:

- **Conventional cold water cooling system (*Cold*):** In a conventional coarse-grained cold water cooling system, all servers are directly cooled by cold water uniformly. The scheduler only chooses the hardware type with the least IT hardware energy while ensuring the latency requirement.
- **Fine-grained warm water cooling system proposed in [19] (*CoolEdge*):** This solution targets heterogeneous edge datacenters, and adopts fine-grained warm water cooling that customizes the water temperature for each hardware component by mixing certain amounts of hot and cold water. We use the average power demand of an inference process to decide the best cooling water temperature.
- **Greedy algorithm (*Greedy*):** We implement the greedy heuristic algorithm to evaluate the DRL agent. The scheduler always prioritizes QoS satisfaction and chooses the hardware type with the least overall energy consumption. For a fair comparison, it only knows the average energy consumption on the four types of hardware, as the accurate energy consumption cannot be accessed in advance.
- **Partially-ideal oracle algorithm (*Oracle*):** On top of the greedy algorithm, the scheduler has full knowledge of the real inference latency on four types of hardware.
- **Fully-ideal oracle algorithm (*Oracle*$^+$):** On top of the greedy algorithm, the scheduler has full knowledge of both the real inference latency and inference energy on four types of hardware.

TABLE III
INFERENCE LATENCY ESTIMATION ERRORS OF SEVEN MODELS

Model	# of layers	Error on CPU and GPU
ResNetV1-50	172	8.8%, -13.4%
EfficientNet-B1	433	0.7%, 9.0%
EfficientNet-B3	490	8.3%, -8.4%
UNet	60	-13.0%, 9.6%
YOLOv3-spp	258	-5.1%, 4.8%
YOLOv3-tiny	43	-0.7%, 12.8%
BiLSTM	4	1.0%, 0.0%

B. Overall Performance

We first present the inference latency estimation results of the *estimator* and then make a comparison with baselines in the aspect of energy consumption, QoS satisfaction, carbon footprints, and cost savings.

Inference latency estimation. Table III summarizes the relative error of the inference latency estimation (i.e., $\frac{\text{Estimated Value}-\text{Measured Value}}{\text{Measured Value}} \times 100\%$). The average estimation error is only around 6.8%, exerting a negligible influence on subsequent task scheduling. Moreover, thanks to the learning capability of the DRL agent that can mitigate the effects of the estimation errors, the latency violation rate of *Hyco* is much lower than other methods, as evidenced by the following results on QoS satisfaction.

Energy consumption. Figure 9 plots the energy consumption and pPUE. As compared with the *Cold*, *CoolEdge*, and *Greedy* baselines, *Hyco* reduces the overall energy consump-

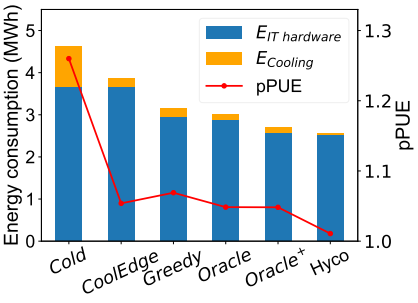


Fig. 9. Energy consumption and pPUE.

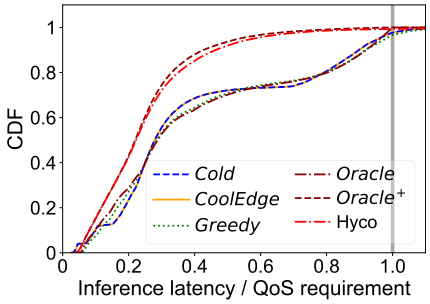


Fig. 10. QoS satisfaction.

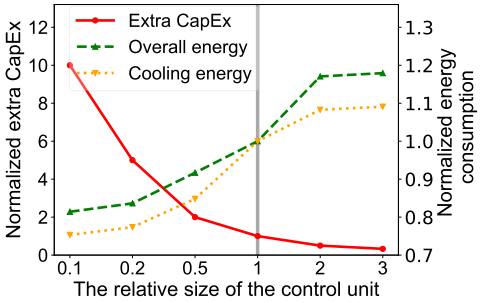


Fig. 11. The energy-CapEx trade-off.

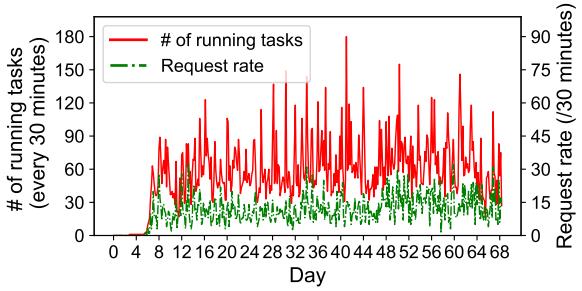


Fig. 12. User request pattern.

tion by 44.5%, 33.6%, and 19.0%, respectively. Owing to the mix of cold water and warm water, the *CoolEdge* baseline reduces the cooling energy by 4.83× than the *Cold* baseline, but *Hyco* further lowers the energy by 7.20× than *CoolEdge* (i.e., 34.74× than *Cold*) thanks to the more use of warm water cooling. The reduced IT hardware energy of the *Greedy* baseline can be attributed to the smart selection between CPU and GPU that leads to different energy efficiency for each DNN model, as well as the power capping policy that leads to higher energy efficiency. By comparison, *Hyco* further reduces the IT hardware energy since the DRL agent can speculate future requests on arrival and their energy consumption information to some extent, and dares to make more aggressive visionary decisions such as switching on a new rack even if there are available alternative components. That is why *Hyco* still lowers the overall energy consumption by 14.8% and 5.3%, respectively, as compared with the *Oracle* and *Oracle*$^+$ baselines. Figure 12 shows the request arrival rate and the number of running tasks every 30 minutes, and Figure 13 plots the energy consumption patterns from one of the testing datasets. As shown, the deep learning workload fluctuates severely in a periodic and bursty manner, and both the IT hardware and cooling energy consumption fluctuate synchronously in general. By comparison, *Hyco* fluctuates only slightly, especially for the cooling energy, which potentially helps save the capital expenditures of the cooling equipment [19].

QoS satisfaction. To show the effectiveness of *Hyco* in satisfying latency requirements, we calculate the actual inference latency of each user request to its latency requirement from one aforementioned testing dataset and plot the cumulative distribution function (CDF) in Figure 10. As we can see, although *Hyco* targets minimizing the energy consumption by tuning down the power limit of a number of components, the latency violation rate is as low as 0.67% thanks to the DRL agent that learns to reduce the estimation error of inference latency adaptively and does not count on high estimation accuracy. In contrast, the *Cold*, *CoolEdge*, and *Greedy* baselines behave poorly in satisfying the latency requirements whose latency violation rates are 2.33%, 2.33%, and 3.41%, respectively. Although the *Oracle* and *Oracle*$^+$ baselines can keep the latency violation rate as low as zero, they are impractical for real-world deployment. These baselines rely on the assumption of having prior knowledge of the real inference latency for all DNN models, which cannot be accurately predicted, as demonstrated by our trials and prior studies [40]. On the contrary, *Hyco* relies mainly on the model architecture information, such as the percentage of each type of layer, which can be easily obtained at run time.

Carbon footprints. To estimate the carbon emissions from the IT and cooling equipment of *Hyco*, as well as the *Cold* and *CoolEdge* baselines, we refer to the simulated energy consumption data presented in Figure 9 and global carbon intensity data from reputable sources [55]. Specifically, the results in Figure 9 indicate that the energy consumption of *Cold*, *CoolEdge*, and *Hyco* is about 4.616 MWh, 3.861 MWh, and 2.562 MWh, respectively, over a 68.5-day period for a single edge datacenter. Given that the average global carbon intensity in 2022 is about 0.22 kg/kWh [55], the corresponding carbon emissions are estimated to be 1015.52 kg CO_2, 849.42 kg CO_2, and 563.64 kg CO_2. Consequently, for 2,000 small-scale edge datacenters located within a city, *Hyco* can cut down carbon footprints by about 4.8 kt CO_2 and 3.0 kt CO_2 every year, as compared to the conventional *Cold* baseline and the state-of-the-art *CoolEdge* baseline, respectively.

Cost savings. The benefits of Hyco arise from reduced energy consumption in both IT and cooling systems, while the primary cost involves installing additional control valves on each server rack. Specifically, for operational cost savings related to energy usage, the results in Figure 9 indicate that the energy consumption of *Cold* and *Hyco* is about 4.616 MWh and 2.562 MWh, respectively, over a 68.5-day period for a single edge datacenter with 50 servers. With the EU average electricity price for non-household consumers being around 22.5 cents/kWh in 2023 [56], the associated operational costs are 110.68 \$/(server×year) for *Cold* and 61.43 \$/(server×year) for *Hyco*, leading to savings of 49.25 \$/(server×year). The additional capital cost is estimated at 0.51 \$/(server×year) over a 10-year operation, considering that each well-developed

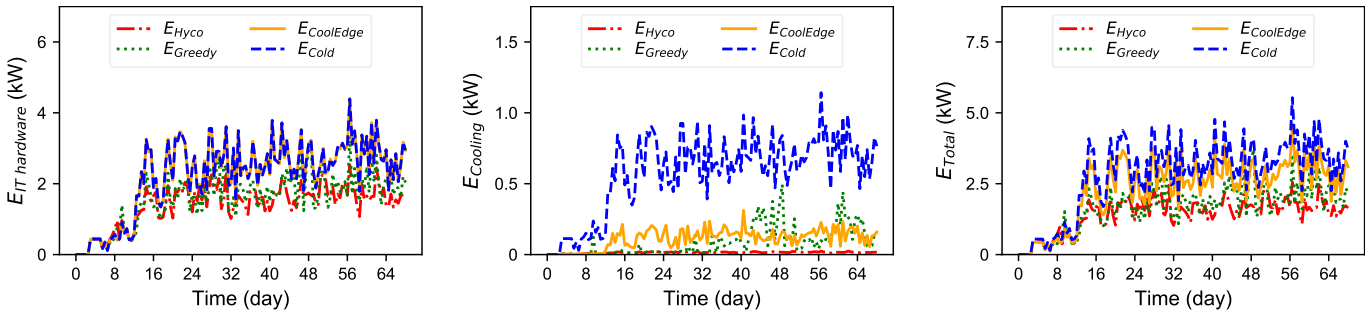


Fig. 13. Energy consumption patterns of the IT hardware, the cooling equipment, and the entire infrastructure.

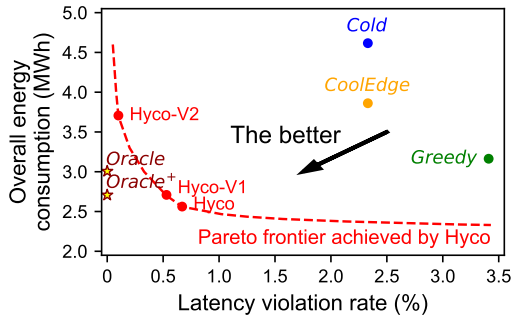


Fig. 14. The trade-off between energy consumption and the latency violation rate. *Hyco* and its variants form a Pareto frontier.

three-way ball valve costs $51 with a lifespan of 100,000 cycles [57]. Finally, the net cost savings with *Hyco* are estimated as 48.74 $/(server×year). For 2,000 small-scale edge datacenters (each equipped with 50 servers) within a city, *Hyco* is projected to save $4,874,000 annually, highlighting its significant potential for widespread adoption.

In summary, by combining the flexible zone-based water cooling architecture with the intelligent DRL agent, *Hyco* achieves significant energy and cost savings as well as carbon reductions in edge datacenters, particularly for the cooling equipment, while maintaining a low QoS violation rate.

C. Flexibility and Practicability for Datacenter Economy

We first discuss the flexible trade-offs achieved by *Hyco* and its variants and then present its small training and runtime scheduling overhead, both of which make *Hyco* practical in the production environment.

Flexible trade-offs through choosing different configurations. Figure 11 shows the trade-off between extra capital expenditures (CapEx) and energy consumption under different sizes of the power and cooling control unit, i.e., the number of server racks that share a common valve and the same power and cooling strategy. The values are normalized to the default setting of size 1, and the value of 0.1 means that every server is individually controlled. A smaller size achieves more fine-grained control and thus less IT hardware and cooling energy. However, the extra CapEx increases sharply because of more valves and pipes installed as well as other supports. We conclude that it is key to select an appropriate size of the control unit for each edge datacenter considering the location, the services, electricity costs, carbon intensity and allowances, maintainability, and so on. For instance, to support sustained compute-intensive edge workloads, saving the operational expenditures (OpEx) for electricity may play a major consideration, while the CapEx may become the first priority if the edge datacenter has abundant access to renewable energy. *Hyco* supports such flexibility to achieve minimum costs (relevant to CapEx plus OpEx) for datacenter operators. Figure 14 shows the trade-off between energy consumption and the latency violation rate by setting different hyperparameters α/β. Compared to baselines, *Hyco* and its variants form a Pareto frontier [58] with different trade-offs, where we cannot improve energy efficiency (QoS satisfaction) further without sacrificing QoS satisfaction (energy efficiency). *Hyco* supports such flexibility to achieve maximum profits (relevant to revenue minus costs that are influenced by QoS satisfaction and energy efficiency respectively) for service providers. Moreover, a top-level scheduler can be further incorporated into *Hyco* to choose between these variants at run time according to real-time requirements.

Runtime scheduling and training overhead. As *Hyco* only considers 15 states in total (e.g., just the numbers of remaining idle components as hardware-related states), the DRL-based scheduler is lightweight and consumes only 0.14 ms on average to take an action and update the state with one CPU core. In addition, it takes less than 3 ms to estimate the inference latency of each DNN model by batching the estimation process of the same layers together. Such computational overhead would be negligible for end-users, as a DNN inference execution usually takes tens to hundreds of milliseconds. As for training, it takes no more than 2 hours with 10 cores of the Intel Core i9-10940X CPU. For an edge datacenter, the DRL model only needs to be trained once at the start of the datacenter's operation, and the trained model remains effective throughout the operational period unless significant changes occur in hardware types, workload patterns, or the ambient environment. In such cases, the DRL model can be retrained in a fast manner through transfer learning [59]. To avoid high latency violation rates and poor service quality, the DRL agent does not make scheduling decisions during the training phase. Instead, all requests are scheduled to the cold water cooling zone directly (i.e., following the *Cold* baseline). While this approach may result in some energy inefficiency, we consider it acceptable since the training time is negligible compared to the typical service period of an edge datacenter, which can span months or even years.

VI. Related Work

Energy management in datacenters. It is widely recognized that datacenters consume substantial amounts of energy and generate significant carbon emissions with the fast growing of cloud computing and edge computing [60]. Many works have concentrated on reducing the energy consumption of either IT hardware [61], [62] or cooling equipment [12], [19], [25], [63]–[66] or both [26], [29], [30], [67]–[69]. Wang et al. [61] propose a workload scheduling solution to reduce IT hardware energy while satisfying QoS requirements. Instead, Liu et al. [67] and Ji et al. [69] notice that saving computing energy alone may increase cooling energy usage and thus they develop a holistic approach separately. Jiang et al. [12] focus on a workload-agnostic cooling solution to deal with the power-cooling mismatch issue in datacenters and advance a TEC-based fine-grained cooling architecture. Rather than conventional cloud datacenters, Pei et al. [19] focus on power-cooling mismatches in edge datacenters and propose a component-level cooling control solution CoolEdge. It achieves high cooling efficiency by customizing water temperatures but incurs relatively high CapEx and increases complexity. On the other hand, compared to our work, CoolEdge cannot handle temporal cooling mismatches as it makes cooling decisions based on only one power value over a period, while the common inference workload shows significant power fluctuations. Motivated by the "sweet point" phenomenon, we propose a hardware-software co-design to address this issue both spatially and temporally in a cost-effective way.

Deep learning deployment. Generally, there are three common ways to execute a DNN inference: locally [70]–[73], cloud/edge-only [45], [74], [75], and collaboratively [40], [41], [76], [77]. Wang et al. [70] notice the inefficiency in deploying inference tasks on modern asymmetric multiprocessors and propose an asymmetry-aware scheduling scheme combined with model partition and hardware frequency setting. However, local execution could be restricted on many resource-limited and energy-constrained mobile devices. Therefore, cloud or edge datacenters equipped with abundant heterogeneous hardware are promising to support compute-intensive deep-learning-based applications. Le et al. [74] propose a scheduling algorithm to allocate heterogeneous hardware resources to users in a fair manner. Due to the long transmission latency and fluctuating network connection, some literature considers executing the inference tasks in a collaborative way. Kang et al. [41] notice that the output size of some intermediate layers is comparatively smaller, which can help reduce both transmission latency and device energy for sending data. Therefore, they advance a layer-level computation partitioning strategy enabling collaborative execution between mobile devices and the cloud. By contrast, our work focuses on the cooling aspect of edge datacenters that deploy these deep-learning-based applications whose power consumption fluctuates severely.

Workload scheduling with RL. There are many state-of-the-art works using the RL technique for workload scheduling [46], [47], [49], [78], [79]. Mao et al. [47] propose an RL-based scheduling algorithm that achieves automatic adaptation to specific workloads. Zou et al. [78] notice the device heterogeneity and task complexity at the edge, and propose a DRL-based algorithm to offload tasks from mobile devices to different edge servers. Similarly, Kim et al. [49] develop an RL-based method to select the best candidate of mobile devices or nearby servers to execute inference tasks by considering DNN models' characteristics and the stochastic nature. Different from them, this work studies the relationship between inference performance and cooling efficiency, and proposes the concept of "working smarter not harder" to achieve overall benefits.

VII. Conclusion

In this paper, we propose Hyco, a hybrid water cooling system to optimize the cooling efficiency of edge datacenters. We analyze three characteristics of edge datacenters and the inefficiency in cooling to deal with power-cooling mismatches from deep learning workloads. Motivated by the "sweet point" phenomenon, we design a zone-based water cooling architecture and a learning-based scheduling scheme with performance-aware power capping under the concept of "working smarter not harder". The trace-driven simulation results demonstrate the effectiveness of Hyco which reduces the cooling energy by up to 34.74× as compared with conventional and state-of-the-art baselines. Our estimation for 2,000 small-scale edge datacenters suggests that Hyco can reduce carbon footprints by approximately 4.8 kt CO_2 and cut monetary costs by about \$4,874,000 every year. The substantial decrease in carbon footprints shows its potential to significantly mitigate the environmental impact of datacenters. As we currently evaluate Hyco through datacenter-level simulations with thermal data collected from a water-cooled testbed, we are also interested in studying the performance of Hyco in real-world scenarios in future work.

References

[1] Deloitte, "Gaining an intelligent edge: Edge computing and intelligence could propel tech and telecom growth," https://www2.deloitte.com/global/en/insights/industry/technology/technology-media-and-telecom-predictions/2021/edge-intelligence-fourth-industrial-revolution.html[Online Accessed, 19-October-2022], 2020.

[2] L. Zhang, Z. Fu, B. Shi, X. Li, R. Lai, C. Chen, A. Zhou, X. Ma, S. Wang, and M. Xu, "SoC-cluster as an edge server: an application-driven measurement study," *arXiv preprint arXiv:2212.12842*, 2022.

[3] J. Meng, Z. Kong, Q. Xu, and Y. C. Hu, "Do larger (more accurate) deep neural network models help in edge-assisted augmented reality?" in *Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration*, 2021, pp. 47–52.

[4] I. Gog, S. Kalra, P. Schafhalter, J. E. Gonzalez, and I. Stoica, "D3: a dynamic deadline-driven approach for building autonomous vehicles," in *Proceedings of the 17th European Conference on Computer Systems*, 2022, pp. 453–471.

[5] OpenAI, "Chatgpt," https://openai.com/chatgpt[Online Accessed, 10-April-2024], 2023.

[6] K. Bittner, "Liquid cooling vs air cooling in data centers," https://www.exittechnologies.com/blog/data-center/liquid-cooling-vs-air-cooling-in-data-centers/[Online Accessed, 12-April-2024], 2020.

[7] LF Edge, "State of the edge 2021," https://www.lfedge.org/2021/03/12/state-of-the-edge-2021-report/[Online Accessed, 19-October-2022].

[8] Schneider Electric, "Digital economy and climate impact," https://download.schneider-electric.com/files?p_File_Name=998-21202519.pdf[Online Accessed, 12-April-2024].

[9] NREL, "High-performance computing data center power usage effectiveness," https://www.nrel.gov/computational-science/measuring-efficiency-pue.html[Online Accessed, 10-April-2024].

[10] A. Lawrence, "Which regions have the most energy efficient data centers?" https://journal.uptimeinstitute.com/datacenter-energy-efficiency-by-region/[Online Accessed, 12-April-2024], 2020.

[11] Google, "Efficiency," https://www.google.com/about/datacenters/efficiency/[Online Accessed, 12-April-2024].

[12] W. Jiang, Z. Jia, S. Feng, F. Liu, and H. Jin, "Fine-grained warm water cooling for improving datacenter economy," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 474–486.

[13] M. Firth, "Introduction to automotive augmented reality head-up displays using TI DLP technology," https://www.ti.com/lit/wp/dlpy009/dlpy009.pdf[Online Accessed, 12-April-2024], 2019.

[14] M. Xu, Z. Fu, X. Ma, L. Zhang, Y. Li, F. Qian, S. Wang, K. Li, J. Yang, and X. Liu, "From cloud to edge: a first look at public edge platforms," in *Proceedings of the 21st ACM Internet Measurement Conference*, 2021.

[15] A. Ali-Eldin, B. Wang, and P. Shenoy, "The hidden cost of the edge: a performance comparison of edge and cloud latencies," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–12.

[16] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[17] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 613–627.

[18] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "Batch: Machine learning inference serving on serverless platforms with adaptive batching," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–15.

[19] Q. Pei, S. Chen, Q. Zhang, X. Zhu, F. Liu, Z. Jia, Y. Wang, and Y. Yuan, "CoolEdge: hotspot-relievable warm water cooling for energy-efficient edge datacenters," in *27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 814–829.

[20] Asetek, "Liquid cooling for data centers," https://www.asetek.com/data-center/technology-for-data-centers[Online Accessed, 12-April-2024].

[21] M. Jalili, I. Manousakis, Í. Goiri, P. A. Misra, A. Raniwala, H. Alissa, B. Ramakrishnan, P. Tuma, C. Belady, M. Fontoura, and R. Bianchini, "Cost-efficient overclocking in immersion-cooled datacenters," in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, 2021.

[22] D. Min, I. Byun, G.-h. Lee, and J. Kim, "CoolDC: A cost-effective immersion-cooled datacenter with workload-aware temperature scaling," *ACM Transactions on Architecture and Code Optimization*, 2024.

[23] K. Haghshenas, B. Setz, Y. Blosch, and M. Aiello, "Enough hot air: the role of immersion cooling," *Energy Informatics*, vol. 6, no. 1, p. 14, 2023.

[24] S. Zimmermann, I. Meijer, M. K. Tiwari, S. Paredes, B. Michel, and D. Poulikakos, "Aquasar: A hot water cooled data center with direct energy reuse," *Energy*, vol. 43, no. 1, pp. 237–245, 2012.

[25] J. D. Moore, J. S. Chase, P. Ranganathan, and R. K. Sharma, "Making scheduling "cool": Temperature-aware workload placement in data centers," in *USENIX Annual Technical Conference, General Track*, 2005, pp. 61–75.

[26] R. Ayoub, R. Nath, and T. Rosing, "JETC: Joint energy thermal and cooling management for memory and CPU subsystems in servers," in *Proceedings of the 18th International Symposium on High Performance Computer Architecture*, 2012.

[27] S. Yeo, M. M. Hossain, J.-C. Huang, and H.-H. S. Lee, "ATAC: Ambient temperature-aware capping for power efficient datacenters," in *Proceedings of the ACM Symposium on Cloud Computing*, 2014, pp. 1–14.

[28] M. Skach, M. Arora, D. Tullsen, L. Tang, and J. Mars, "Virtual melting temperature: Managing server load to minimize cooling overhead with phase change materials," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, 2018.

[29] Y. Ran, H. Hu, X. Zhou, and Y. Wen, "DeepEE: Joint optimization of job scheduling and cooling control for data center energy efficiency using deep reinforcement learning," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 645–655.

[30] C. Chi, K. Ji, A. Marahatta, P. Song, F. Zhang, and Z. Liu, "Jointly optimizing the IT and cooling systems for data center energy efficiency based on multi-agent deep reinforcement learning," in *Proceedings of the eleventh ACM international conference on future energy systems*, 2020, pp. 489–495.

[31] J. Li, Y. Deng, Y. Zhou, Z. Zhang, G. Min, and X. Qin, "Towards thermal-aware workload distribution in cloud data centers based on failure models," *IEEE Transactions on Computers*, vol. 72, no. 2, pp. 586–599, 2023.

[32] J. Chen and X. Ran, "Deep learning with edge computing: A review." *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[33] Q. Pei, Y. Yuan, H. Hu, Q. Chen, and F. Liu, "AsyFunc: A high-performance and resource-efficient serverless inference system via asymmetric functions," in *Proceedings of the 2023 ACM Symposium on Cloud Computing*, 2023, pp. 324–340.

[34] Y. Xiang and H. Kim, "Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 392–405.

[35] F. Yu, K. Chan, R. Sit, and J. Yang, "Review of standards for energy performance of chiller systems serving commercial buildings," *Energy Procedia*, vol. 61, pp. 2778–2782, 2014.

[36] Kelvion, "Modular & sustainable solutions," https://www.kelvion.com/products/[Online Accessed, 19-October-2022].

[37] S. Williams, "Three-way ball valve flow patterns," https://www.industrialspec.com/about-us/blog/detail/three-way-ball-valve-t-port-l-port-flow-patterns[Online Accessed, 19-October-2022], 2018.

[38] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: eliminating server idle power," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2009.

[39] A. E. Eshratifar and M. Pedram, "Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, 2018, pp. 111–116.

[40] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, 2019.

[41] 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 22nd International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017.

[42] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Automated model-less inference serving," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 397–411.

[43] TensorFlow, "Optimize TensorFlow performance using the Profiler," https://www.tensorflow.org/guide/profiler[Online Accessed, 12-April-2024].

[44] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation*, 2016, pp. 265–283.

[45] A. Gujarati, R. Karimi, S. Alzayat, W. Hao, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving DNNs like Clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation*, 2020, pp. 443–462.

[46] Z. Wang, S. Zhu, J. Li, W. Jiang, K. Ramakrishnan, Y. Zheng, M. Yan, X. Zhang, and A. X. Liu, "DeepScaling: microservices autoscaling for stable CPU utilization in large scale cloud systems," in *Proceedings of the 13th Symposium on Cloud Computing*, 2022, pp. 16–30.

[47] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 270–288.

[48] H. Qiu, W. Mao, A. Patke, C. Wang, H. Franke, Z. T. Kalbarczyk, T. Başar, and R. K. Iyer, "SIMPPO: a scalable and incremental online learning framework for serverless resource management," in *Proceedings of the 13th Symposium on Cloud Computing*, 2022, pp. 306–322.

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

[50] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

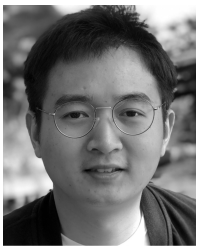
[51] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[52] E. Even-Dar, S. Mannor, Y. Mansour, and S. Mahadevan, "Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems." *Journal of Machine Learning Research*, vol. 7, no. 6, 2006.
[53] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3-4, pp. 293–321, 1992.
[54] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 945–960.
[55] Global Carbon Budget (2023); U.S. Energy Information Administration (2023); Energy Institute - Statistical Review of World Energy (2023) – with major processing by Our World in Data, "Carbon intensity of energy production – GCB," https://ourworldindata.org/grapher/co2-per-unit-energy[Online Accessed, 12-April-2024], 2023.
[56] Eurostat, "Electricity prices for non-household consumers - bi-annual data (from 2007 onwards)," https://ec.europa.eu/eurostat/databrowser/view/nrg_pc_205/default/table?lang=en[Online Accessed, 15-Aug-2024], 2024.
[57] Alibaba, "Price of the 3-way motorized control ball valve," https://www.alibaba.com/product-detail/3-way-DC12V-AC110-230V-1_60442576676.html[Online Accessed, 15-Aug-2024], 2024.
[58] Y. Censor, "Pareto optimality in multiobjective problems," *Applied Mathematics and Optimization*, vol. 4, no. 1, pp. 41–59, 1977.
[59] Z. Zhu, K. Lin, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *arXiv preprint arXiv:2009.07888*, 2020.
[60] N. Bashir, D. Irwin, P. Shenoy, and A. Souza, "Sustainable computing - without the hot air," *ACM SIGENERGY Energy Informatics Review*, vol. 3, no. 3, pp. 47–52, 2023.
[61] S. Wang, Y. Liang, and W. Zhang, "Poly: Efficient heterogeneous system and application management for interactive applications," in *25th IEEE International Symposium on High Performance Computer Architecture*, 2019.
[62] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *49th Annual IEEE/ACM International Symposium on Microarchitecture*, 2016, pp. 1–13.
[63] L. Ramos and R. Bianchini, "C-Oracle: Predictive thermal management for data centers," in *Proceedings of the 14th International Symposium on High Performance Computer Architecture*, 2008.
[64] T. D. Nguyen and R. Bianchini, "CoolAir: Temperature- and variation-aware management for free-cooled datacenters," in *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015.
[65] I. Manousakis, Í. Goiri, S. Sankar, T. D. Nguyen, and R. Bianchini, "CoolProvision: Underprovisioning datacenter cooling," in *Proceedings of the 6th ACM Symposium on Cloud Computing*, 2015, pp. 356–367.
[66] D. Van Le, Y. Liu, R. Wang, R. Tan, and L. H. Ngoh, "Air free-cooled tropical data center: Design, evaluation, and learned lessons," *IEEE Transactions on Sustainable Computing*, vol. 7, no. 3, pp. 579–594, 2021.
[67] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser, "Renewable and cooling aware workload management for sustainable data centers," in *ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, 2012, pp. 175–186.
[68] A. Khosravi, L. L. Andrew, and R. Buyya, "Dynamic VM placement method for minimizing energy and carbon cost in geographically distributed cloud data centers," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 183–196, 2017.
[69] K. Ji, C. Chi, A. Marahatta, F. Zhang, and Z. Liu, "Energy efficient scheduling based on marginal cost and task grouping in data centers," in *The Eleventh ACM International Conference on Future Energy Systems*, 2020, pp. 482–488.
[70] M. Wang, S. Ding, T. Cao, Y. Liu, and F. Xu, "AsyMo: scalable and efficient deep-learning inference on asymmetric mobile CPUs," in *The 27th Annual International Conference on Mobile Computing and Networking*, 2021.
[71] J. Wang, B. Cao, P. Yu, L. Sun, W. Bao, and X. Zhu, "Deep learning towards mobile applications," in *38th IEEE International Conference on Distributed Computing Systems*, 2018, pp. 1385–1393.
[72] L. Zhou, M. H. Samavatian, A. Bacha, S. Majumdar, and R. Teodorescu, "Adaptive parallel execution of deep neural networks on heterogeneous edge devices," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 195–208.
[73] S. Bateni and C. Liu, "NeuOS: A latency-predictable multi-dimensional optimization framework for DNN-driven autonomous systems," in *2020 USENIX Annual Technical Conference*, 2020, pp. 371–385.
[74] T. N. Le, X. Sun, M. Chowdhury, and Z. Liu, "Allox: compute allocation in hybrid clusters," in *15th European Conference on Computer Systems*, 2020, pp. 1–16.
[75] J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang, "DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015.
[76] 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*, 2018, pp. 401–411.
[77] J. Wu, L. Wang, Q. Pei, X. Cui, F. Liu, and T. Yang, "HiTDL: High-throughput deep learning inference at the hybrid mobile edge," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4499–4514, 2022.
[78] J. Zou, T. Hao, C. Yu, and H. Jin, "A3C-DO: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE Transactions on Computers*, 2020.
[79] Y. G. Kim and C.-J. Wu, "AutoFL: Enabling heterogeneity-aware energy efficient federated learning," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 183–198.

Qiangyu Pei received the BS degree in physics from Huazhong University of Science and Technology, China, in 2019. He is currently working toward the PhD degree in the School of Computer Science and Technology, Huazhong University of Science and Technology. His research interests include edge computing, green computing, and deep learning.

Yongjie Yuan received the B.Eng. degree from the School of Computer Science and Technology, Huazhong University of Science and Technology, China, in 2021, where he is currently pursuing the M.Eng. degree. His research interests include edge computing and serverless computing.

Haichuan Hu received his B.Eng. degree in computer science from Nanchang University, China, in 2022. He is currently an M.Eng. student in the School of Computer Science and Technology, Huazhong University of Science and Technology, China. His research interests include edge computing, parallel computing, and serverless computing.



Lin Wang is currently a Full Professor and Head of the Chair of Computer Networks in the Department of Computer Science at Paderborn Univesity. He received his Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences in 2015. Previously, he held positions at Vrije Universiteit Amsterdam, TU Darmstadt, SnT Luxembourg, and IMDEA Networks Institute. He is broadly interested in networked systems, with a focus on in-network computing, machine learning systems, and intermittently-powered IoT systems. He has received a Google Research Scholar Award, an Outstanding Paper Award of RTSS 2022, Best Paper Awards of IPCCC 2023 and HotPNS 2016, and an Athene Young Investigator Award of TU Darmstadt. He is currently a Senior Member of IEEE.

Zhang Dong is a professorate senior engineer of Jinan Inspur Data Technology Co., Ltd. He has led the development of the world's highest computing and storage density rack server, the first China UNIX operating system. He has made creative contributions in areas such as converged architecture and high-end system software, earning one national award, and eleven provincial-level awards.

Yan Bingheng is a senior engineer of Jinan Inspur Data Technology Co., Ltd. He received his Ph.D. at Xi'an JiaoTong University in 2010, and broadly interested in the area of OS, virtualization, and cloud computing. He has led a wide range of virtualization research projects, and the development of Inspur's virtualization product InCloud Sphere, which break the global record of SpecVirt.

Chen Yu received the BS degree in mathematics and the MS degree in computer science from Wuhan University, Wuhan, China, in 1998 and 2002, respectively, and the PhD degree in information science from Tohoku University, Sendai, Japan, in 2005. From 2005 to 2006, he was a Japan Science and Technology Agency postdoctoral researcher with the Japan Advanced Institute of Science and Technology In 2006, he was at Japan Society for the Promotion of Science postdoctoral fellow with the Japan Advanced Institute of Science and Technology. Since 2008, he has been with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, where he is currently a professor and a special research fellow, working in the areas of ubiquitous computing, edge computing and industrial Internet.

Fangming Liu (S'08, M'11, SM'16) received the B.Eng. degree from the Tsinghua University, Beijing, and the Ph.D. degree from the Hong Kong University of Science and Technology, Hong Kong. He is currently a Full Professor with the Huazhong University of Science and Technology, Wuhan, China. His research interests include cloud computing and edge computing, datacenter and green computing, SDN/NFV/5G and applied ML/AI. He received the National Natural Science Fund (NSFC) for Excellent Young Scholars, and the National Program Special Support for Top-Notch Young Professionals. He is a recipient of the Best Paper Award of IEEE/ACM IWQoS 2019, ACM e-Energy 2018 and IEEE GLOBECOM 2011, the First Class Prize of Natural Science of Ministry of Education in China, as well as the Second Class Prize of National Natural Science Award in China.