InferCool: Enhancing AI Inference Cooling through Transparent, Non-Intrusive Task Reassignment

Qiangyu Pei
Huazhong University of Science and Technology
Wuhan, China
peiqiangyu@hust.edu.cn

Lin Wang
Paderborn University
Paderborn, Germany
lin.wang@uni-paderborn.de

Dong Zhang
Inspur Data Co., Ltd.
Jinan, China

Bingheng Yan
Inspur Data Co., Ltd.
Jinan, China

Chen Yu
Huazhong University of Science and Technology
Wuhan, China
yuchen@hust.edu.cn

Fangming Liu*
Huazhong University of Science and Technology
Peng Cheng Laboratory
Wuhan, China
fangminghk@gmail.com

ABSTRACT

The increasing power consumption of AI inference in modern datacenters has escalated cooling demands significantly, necessitating the adoption of potent cooling approaches like water cooling. Unlike traditional cloud workloads, AI inference has unique characteristics that create substantial gaps in achieving optimal cooling efficiency. In this work, we present the first comprehensive measurement study of AI inference cooling across various models within an industrial-ready scheduling framework, highlighting significant inefficiencies and their causes. To fill the gap while following the fundamental requirements of cooling systems, we explore a new opportunity presented by modern Multi-Instance GPU-enabled inference serving, where the scheduling dimension is naturally orthogonal to the cooling dimension. Building on this insight, we develop InferCool, a cooling middleware designed to enhance cooling efficiency for inference serving through transparent, non-intrusive task reassignment. It includes a streamlined power and temperature prediction approach and a thermal-aware, adaptive application deployment and request scheduling mechanism. Real-world experiments on a water-cooled testbed and a three-node cluster demonstrate that InferCool can reduce the maximum GPU temperature by 5℃ across eight A100 GPUs, equivalent to cooling energy savings of about 20%. Importantly, InferCool requires no modifications to existing cooling infrastructures and is compatible with existing scheduling systems.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Hardware** → **Thermal issues**; *Enterprise level and data centers power issues.*

KEYWORDS

datacenter cooling, AI inference serving, energy efficiency

ACM Reference Format:
Qiangyu Pei, Lin Wang, Dong Zhang, Bingheng Yan, Chen Yu, and Fangming Liu. 2024. InferCool: Enhancing AI Inference Cooling through Transparent, Non-Intrusive Task Reassignment. In *ACM Symposium on Cloud Computing (SoCC '24), November 20–22, 2024, Redmond, WA, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3698038.3698556

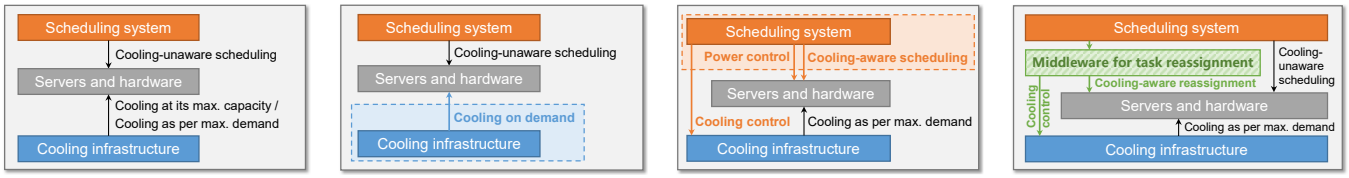
1 INTRODUCTION

Recent years have witnessed a growing demand for powerful datacenters, especially in the era of large language models (LLMs). It was announced by NVIDIA at the end of 2022, that it would collaborate with Microsoft to build a powerful AI supercomputer equipped with tens of thousands of A100 and H100 GPUs for the model training and inference serving [32]. However, more powerful servers and

*F. Liu is the corresponding author. Q. Pei and C. Yu are with National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, Huazhong University of Science and Technology, Wuhan, 430074, China.

SoCC '24, November 20–22, 2024, Redmond, WA, USA

ACM ISBN 979-8-4007-1286-9/24/11…$15.00
https://doi.org/10.1145/3698038.3698556



(a) Conventional non-intrusive but inefficient cooling systems

(b) State-of-the-art systems intrusive to existing cooling infrastructures [19, 39, 46]

(c) State-of-the-art systems intrusive to existing scheduling systems [7, 29, 42, 47, 50]

(d) InferCool is non-intrusive to both existing cooling infrastructures and scheduling systems

Figure 1: Cooling mechanisms of conventional, state-of-the-art, and InferCool-based systems.

hardware also bring higher power demands. In particular, the thermal design power of the A100, H100, and the latest B200 GPUs has reached 400 W, 700 W, and 1 kW, respectively, remarkably higher than previous GPU generations with around 200~300 W consumption. It was reported that a simple ChatGPT query consumes almost 10× the electricity of a usual Google search [44]. Moreover, AI datacenters, which currently account for about 4% of America's power consumption, could see their share soar to 25% by 2030 [44].

Among the energy consumed by datacenters, the cooling infrastructure is a predominant factor of non-IT equipment [19], accounting for up to 30% of total datacenter energy. This proportion can rise to nearly half in edge datacenters that house many high-powered GPUs to support real-time, compute-intensive services like AI inference [39]. Within an inference serving system, a single model typically cannot saturate a GPU's capacity, prompting cloud providers to deploy multiple models on a single GPU using GPU-sharing technologies such as Multi-Process Service (MPS) [35] and Multi-Instance GPU (MIG) [31]. Through fine-grained GPU sharing, the increased utilization and power consumption will intensify the demand for cooling at the same time. Meanwhile, as presented later in Section 3, the unique characteristics of AI inference can cause power levels to fluctuate rapidly, thereby widening the gap to achieving optimal cooling efficiency.

While there have been numerous studies aimed at enhancing the cooling efficiency of datacenters, none have specifically addressed the unique demands of AI workloads. We divide them into two categories: (1) developing new cooling architectures at the infrastructure level, including changing cooling approaches and adding new components [17–19, 39, 46, 47], and (2) applying thermal-aware workload scheduling integrated with power throttling and/or cooling control at the server and platform levels [1, 5, 7, 29, 30, 42, 47, 50]. When applied to AI datacenters, they can exhibit significant limitations in satisfying the two fundamental requirements of the cooling system—*reliability* and *extensibility*—simultaneously in practice, which are especially essential to inference serving systems with strict and various performance requirements. In particular, studies in the second category develop their own schedulers from scratch to realize cooling-aware workload scheduling. Those schedulers not only lack many necessary features like resource scaling and task migration [3], but also cannot be easily integrated into existing widely-used scheduling frameworks. For many legacy and colocation datacenters, it is also costly and even impracticable to adopt newly proposed cooling architectures.

To maintain both reliability and extensibility while achieving high cooling efficiency in AI datacenters for inference serving, we explore a new opportunity presented by MIG-enabled inference serving, a growing trend today [25]. Specifically, previous GPU-sharing methods like MPS tie both scheduling and cooling units to each GPU entity, often causing cooling optimization to contradict scheduling requirements. Our experiments on inference serving indicate that even a load-balancing scheduling strategy can lead to severe thermal imbalances. In contrast, with the modern MIG-based GPU-sharing method, the cooling unit remains at the hardware level while scheduling occurs at the partition level — MIG partitions can be regarded as a series of logically individual, smaller GPUs on top of physical GPU entities. Leveraging this property, the cooling optimization becomes orthogonal to following scheduling requirements in MIG-enabled systems, allowing for task reassignment to balance thermal distribution and improve cooling efficiency without impacting application performance. For example, reassigning a task from a partition on a hot GPU component to another partition with the same size but on a cooler component is unlikely to cause any performance implications due to resource contention, but can achieve better thermal distribution.

To fully exploit the opportunity, we develop a temperature-oriented middleware called InferCool to enhance Inference Cooling through transparent, non-intrusive *task reassignment*. As illustrated in Figure 1, state-of-the-art cooling mechanisms are intrusive to either the underlying cooling infrastructures (impacting reliability) or the top-level scheduling systems (impacting extensibility). In contrast, InferCool integrates seamlessly with widely-adopted, practice-proven cooling architectures and scheduling frameworks, requiring no modifications to existing inference scheduling strategies. During application deployment and request scheduling, InferCool will reassign tasks in a cooling-friendly manner

while satisfying the original deployment and scheduling requirements. We summarize our contributions as follows.

- We conduct a systematic study of AI inference cooling and highlight the challenges of efficiently cooling inference workloads given their unique characteristics.
- Without compromising the requirements of the cooling system, we identify a new opportunity for cooling MIG-enabled co-located inference workloads and propose a temperature-oriented task-reassignment middleware named InferCool, which can operate without affecting application performance or requiring modifications to the underlying infrastructure.
- We model the power and temperature dynamics of GPUs when co-locating multiple inference workloads. Based on empirical studies, we develop a precise and efficient power and temperature prediction approach, which supports the design of a lightweight, thermal-aware application deployment and request scheduling mechanism to optimize cooling efficiency.
- We evaluate InferCool on a water-cooled testbed and a three-node cluster, using a real-world request arrival dataset and a wide range of inference models. The experimental results show that, compared to vanilla Kubernetes, InferCool can reduce cooling energy consumption by up to 20% without being intrusive.

2 BACKGROUND

In this section, we first introduce the latest Multi-instance GPU technology for inference co-location. Then, we present the background of water cooling systems and summarize their key requirements.

2.1 MIG-enabled Inference Serving

The MIG technology was first introduced in NVIDIA's Ampere architecture GPUs in 2020 and has been incorporated into all subsequent generations. Recent AI datacenters from enterprises like Microsoft [32] and Meta [21], have all been equipped with powerful, MIG-supported GPUs like the A100 and H100. As suggested by NVIDIA [28], MIG technology is promising to improve GPU utilization by enabling the colocation of multiple tasks, especially for inference workloads that typically cannot saturate a GPU. Unlike previous software-level GPU-sharing methods, MIG provides full isolation between different slices (also referred to as "partitions"), as it partitions not only compute resources (i.e., streaming multiprocessor (SM)) but also memory resources (e.g., memory, cache, and bandwidth) [31]. Therefore, there is almost no performance interference between partitions, as demonstrated by prior research [25] and our experimental results from the first two rows in Table 1. The cloud platform and applications are now agnostic to GPU entities and instead recognize each GPU partition as an individual, smaller GPU.

Table 1: Processing and energy efficiency of AI inference on different partitions (BERT-large as an example)

Used partition / partitioning plan	Batch size	Avg. inference latency (ms)	Inference throughput	Avg. energy per request (J)*
1 / 7 × 1g.5gb	1	14.86	67	0.426
7 / 7 × 1g.5gb	1	14.98	7 × 67	0.373
1 / 1 × 7g.40gb	1	10.50	95	0.414
1 / 1 × 7g.40gb	17#	13.40	1269	0.106

* The energy values exclude idle energy consumption.
\# The batch size is set to the maximum value that ensures the inference latency does not exceed that on the 1g.5gb partition.

Taking the A100 40GB GPU as an example, there are five available partition sizes, including 1g.5gb, 2g.10gb, 3g.20gb, 4g.20gb, and 7g.40gb. In these notations, the prefix (e.g., 1g, 2g) and suffix (e.g., 5gb, 10gb) denote the amounts of compute and memory resources, respectively. These sizes can be combined to form 19 configurable partitioning plans, with a total size not exceeding 7g.40gb [33], e.g., 1×3g.20gb+ 1×2g.10gb+2×1g.5gb. For AI inference processing, different partition sizes can produce varying processing efficiency (e.g., latency & throughput) and energy efficiency, as evidenced by the first, third, and fourth rows in Table 1. In particular, by allowing for larger batch sizes, larger partitions can significantly reduce inference energy consumption. Moreover, for a specific partitioning plan, when multiple partitions execute inference simultaneously, the energy efficiency generally increases due to higher resource utilization, while the processing efficiency remains unaffected due to full isolation, as evidenced by the first two rows in Table 1.

Through fine-grained sharing of GPU resources when serving compute-intensive AI inference on powerful GPUs, both resource utilization and power consumption increase, which in turn raises thermal intensity, making it difficult for air cooling to dissipate heat efficiently. In a stress test on an A100 GPU in our local workstation placed in a room at around 20℃, when using a customized fan, the GPU encountered automated power capping and its temperature reached over 80℃. By comparison, when we replaced the fan with a cold plate for water cooling, the GPU works well, and its temperature is only around 53℃ when the water temperature is maintained at 30℃. To deliver adequate cooling capacity for high-powered hardware, water cooling offers an appealing solution, which has been adopted by NVIDIA in its latest rack-scale product as well [37].

2.2 Water Cooling System

Figure 2 illustrates a representative water cooling architecture for heterogeneous hardware. Specifically, the cooling water in the secondary loop is pumped from the water tank into every server. Each hardware component in the server

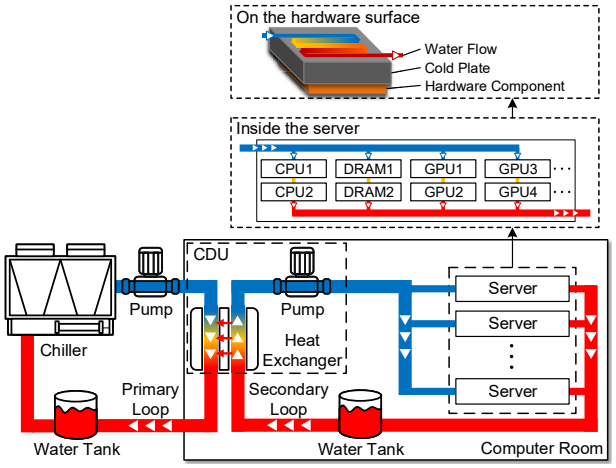


Figure 2: The water cooling architecture in datacenters.

is attached to a cold plate, where the cooling water flows through and absorbs the heat generated by the corresponding component. In some server designs [20], the cooling water passes through multiple components along the same path, causing the water to warm up before reaching subsequent components. When the water exits the servers, it becomes hot and releases heat into the primary loop via a heat exchanger. The water in the primary loop then gets chilled by a chiller and/or cooling tower. To ensure that all hardware components operate below their safe operating temperatures, the required cooling capacity (i.e., the chilled water temperature) in each adaptation period depends on the highest cooling demand of all the components, characterized by their maximum reachable temperature [19, 39].

As the cooling system plays a critical role in maintaining a suitable hardware temperature in datacenters, we emphasize that it needs to satisfy the following requirements.

(1) **Reliability:** A reliable cooling system is essential for maintaining the functionality of the entire datacenter. Lower reliability increases the likelihood of datacenter outages, which can be detrimental to many mission-critical real-time inference applications. According to a study by Vertiv [48], the cost of a cooling-related downtime reaches an average of $554,000.
(2) **Extensibility:** On top of reliability, the cooling system should be able to work seamlessly with any cluster schedulers and accommodate any workloads with specific scheduling requirements. Additionally, it should be capable of adapting to any changing needs, such as hardware upgrades and load intensity variations.
(3) **Efficiency:** As discussed in Section 1, the cooling system contributes significantly to datacenter energy, particularly in supporting power-hungry AI applications. Pursuing optimal cooling efficiency is the ultimate goal on the way to datacenter sustainability.

While water cooling shows promise for supporting AI workloads, significant cooling capacity waste remains due to the unique characteristics of AI inference. In the following, we offer the first detailed analysis of AI inference from a cooling perspective and derive several valuable insights for optimizing cooling efficiency.

3 COOLING PERSPECTIVE OF AI INFERENCE

In this section, we first analyze the unique characteristics of AI inference and their potential impact on cooling efficiency. Then, we conduct a measurement study to reveal the cooling impacts of AI inference. Finally, with a focus on the requirements of cooling systems, we highlight the opportunity to enhance cooling efficiency for MIG-enabled inference serving.

3.1 Key Characteristics of AI Inference

First, we summarize four key characteristics of AI inference applications that significantly impact cooling efficiency.

(1) **Strict performance requirements:** AI inference applications typically demand high end-to-end performance due to their interactive nature. For example, on a popular local life services platform, about 86.2% of the over 600 deployed ML models are expected to deliver responses within 50 ms [49]. According to a recent report, every 100 ms delay in response time can cause a 1% decline in revenue [27].
(2) **Intensive computing demands:** The size of recent AI models has grown significantly. For example, the GPT-2 model contains 1.5B parameters [41] and requires 3400 GFLOPs [10], while the GPT-3 model grows by more than 100× [10]. Such surging computing demand also brings a heavy burden on power supply as well as cooling.
(3) **Stochastic request arrivals:** Recent studies have highlighted the bursty and intermittent nature of inference requests [2, 40, 49]. These requests can vary widely in application type (e.g., image recognition, chat), size (e.g., batch size, input length), and resource demand (e.g., requiring a larger partition size for lower latency), causing significantly different power usage behaviors.
(4) **Short duration:** Given the real-time demands and intermittent request arrivals, AI inference executions are generally short-lived, lasting from tens of milliseconds to several seconds. Once a request is finished and no new requests arrive, the resources can be idle at low power levels.

The above characteristics lead to high while significantly fluctuating hardware power consumption, resulting in severe thermal imbalance both spatially and temporally.

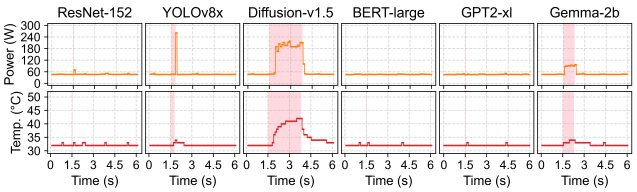
3.2 Cooling Impacts of AI inference

Next, we go deeper into the cooling impacts of AI inference through extensive experiments. In particular, we define three terms to measure power usage patterns and their effects on hardware temperature.

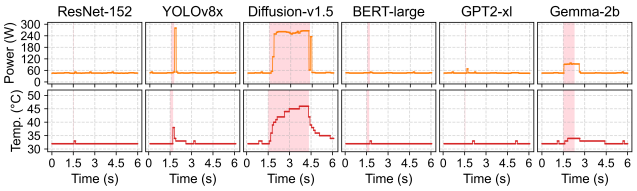
- **Individual intensity:** The average power usage and maximum hardware temperature rise during *a single inference execution*. It depends on the model type, batch size, as well as the hardware partitioning plan[1].
- **Cumulative intensity:** The average power usage and maximum hardware temperature rise over a period of *consecutive inference executions*. This usually happens when the system is overloaded. It depends on the hardware partitioning plan, model type, batch size, and the number of inference executions.
- **Intensity distribution:** The power usage pattern and maximum hardware temperature rise over a period of *intermittent inference executions*. This happens when requests arrive stochastically and the resources are underutilized. It depends on the hardware partitioning plan, model type, batch size, the number of inference executions, and request arrival pattern.

We select ten popular AI models from both the CV and NLP domains, ranging in size from XS to XL, and conduct various measurements on a water-cooled, MIG-enabled A100 GPU[2]. Due to space constraints, we present results for six models in each figure, including ResNet-152 (S), YOLOv8x (L), Diffusion (XL), BERT-large (S), GPT2-xl (L), and Gemma-2b (XL), with results for all ten models available in a supplementary file [38]. There are two exceptions in the experimental settings: (1) For batch sizes of 8 and 16, the Diffusion model uses batch sizes of only 1 and 2, respectively, to maintain acceptable inference latency, and (2) For the partitioning plan of one and seven `1g.5gb` partitions, the Gemma-2b model is placed on one and three `2g.10gb` partitions, respectively, due to its high memory requirements. In all figures, the pink-shaded period denotes the tested phase.

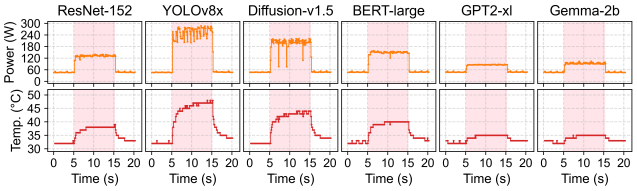
3.2.1 Experiment #1 on general behaviors of the individual and cumulative intensities. First, we evaluate how the individual intensity and cumulative intensity are influenced by the three basic factors: model type, batch size, and the number of inference executions. We fix the partition size to `7g.40gb`. From Figure 3, we can draw three interesting observations: **❶ Despite being power-intensive, a single inference execution can have little influence on GPU temperature.** For example, as shown in Figures 3a and 3b, while YOLOv8x at bs=16 and Diffusion can cause obvious GPU



(a) Running inference *only once (bs = 8)*



(b) Running inference *only once (bs = 16)*



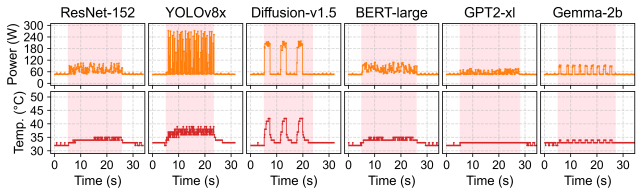
(c) Running inference *continuously for 10 seconds (bs = 8)*

Figure 3: The GPU power and temperature variations under different individual and cumulative intensities.

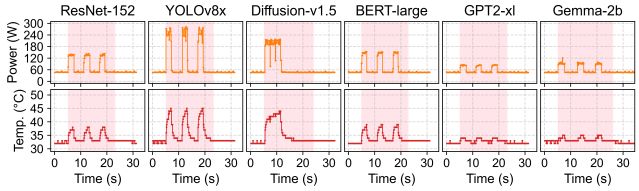
temperature rises, for most models—even LLMs like Gemma-2b with high inference time—the GPU temperature rises by no more than 2℃. **❷ Due to the short-lived nature of inference, the GPU temperature is influenced not only by power consumption but also by the duration of inference executions.** Existing studies [19, 39] assume a one-to-one relationship between the power and temperature, which we find does not hold for inference workloads anymore. For example, as shown in Figure 3b, although Diffusion consumes slightly less average power than YOLOv8x, its final maximum GPU temperature is 8℃ higher; in Figure 3c, when YOLOv8x performs inferences continuously, the GPU temperature keeps growing and stabilizes at 47∼48℃, while a single execution in Figure 3a only causes the temperature to rise to 34℃. **❸ Models with a high individual intensity, however, can exhibit a low cumulative intensity.** For example, in Figures 3a and 3b, Gemma-2b shows a higher individual intensity than ResNet-152 and BERT-large, but in Figure 3c, we can find that ResNet-152 and BERT-large show much higher cumulative intensities, with maximum temperatures 4℃ and 5℃ higher than Gemma-2b, respectively.

3.2.2 Experiment #2 on the intensity distribution under different request arrival patterns. Second, we evaluate how the intensity distribution is influenced by the request arrival pattern. We fix the partition size to `7g.40gb`. For each model, we send the same total number of requests under different arrival patterns. From Figure 4, we can draw the following

[1]We currently do not consider the impact of input length, whose impact on the hardware power and temperature is similar to that of batch size.
[2]Details on the models and experimental setup are provided in Section 6.1.



(a) *Uniform request arrival:* after each inference execution, stopping for twice the inference time (bs = 8)



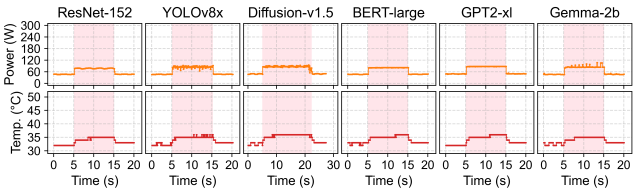
(b) *Bursty request arrival:* for every 2 seconds of inference executions, stopping for 4 seconds (bs = 8)

Figure 4: The GPU power and temperature variations under different intensity distributions.

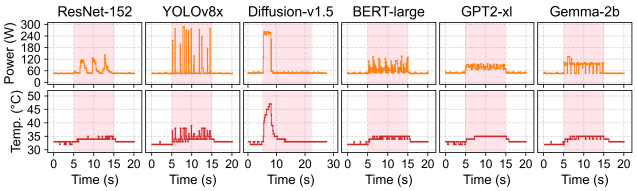
notable observation: ❹ **As fluctuations in request arrivals increase, the intensity distribution becomes more dispersed, significantly raising the maximum GPU temperature even though the average power consumption remains unchanged.** For example, the temperature rise for YOLOv8x under the bursty arrival pattern nearly doubles compared to the uniform arrival pattern—from 7℃ to 13℃.

3.2.3 Experiment #3 on the cumulative intensity and intensity distribution under different GPU partitioning plans. Third, we evaluate how the cumulative intensity and intensity distribution are influenced by the GPU partitioning plan. For each model, we send the same total number of requests under different partitioning plans. From Figure 5, we have two notable observations: ❺ **Different partitions generally exhibit similar cumulative intensities for most models when the request size remains the same.** On the one hand, smaller partitions may show slightly lower energy efficiency, as evidenced in Table 1, resulting in higher average power usage than larger partitions. On the other hand, on larger partitions with more available resources and shorter inference times, the short-lived characteristic can cause more significant power fluctuations and temperature surges. As a result, ❻ **for compute-intensive models, larger partitions can exhibit much higher cumulative intensities than smaller partitions.** As two exceptions—YOLOv8x and Diffusion, when deployed on a `7g.40gb` partition with more available resources, the temperature rise grows from 4℃ to 7℃ and 4℃ to 15℃, respectively.

Accordingly, we can infer that when the load is light and GPUs are largely underutilized (e.g., only reaching the maximum supported throughput of one of the smaller partitions), the intensity distribution under larger partitions

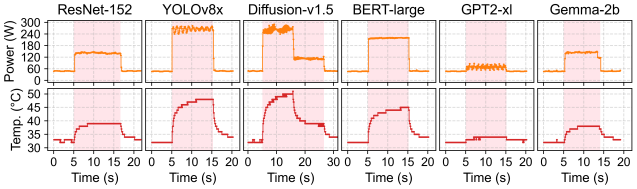


(a) *Smaller partition:* one `1g.5gb` partition (bs = 16)

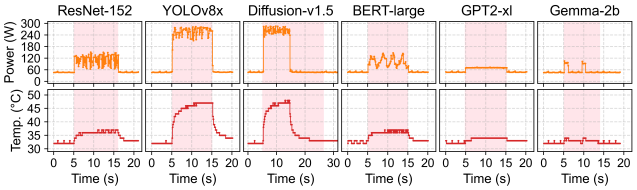


(b) *Larger partition:* one `7g.40gb` partition (bs = 16)

Figure 5: The GPU power and temperature variations under different cumulative intensities influenced by GPU partitioning plans.



(a) *Smaller partition:* bs = 1 on seven `1g.5gb` partitions



(b) *Larger partition: Adaptive bs* on one `7g.40gb` partition

Figure 6: The GPU power and temperature variations under different cumulative intensities influenced by GPU partitioning plans and batch sizes.

would be more sensitive to request arrival patterns. However, since smaller partitions can significantly improve inference throughput, as indicated in Table 1, we also observe that ❼ **when the load is high (e.g., reaching the maximum supported throughput of larger partitions), the intensity distribution under multiple smaller partitions become more sensitive to request arrival patterns, and the cumulative intensity can be much higher due to increased GPU utilization.** The results for running inference on seven `1g.5gb` partitions simultaneously under different arrival patterns and under full load, are available in [38].

As evidenced in Table 1, larger partitions can achieve higher energy efficiency by allowing for larger batch sizes within a fixed time budget. Here, we set the batch size on the `7g.40gb` partition to the maximum value that ensures the inference latency does not exceed that on a `1g.5gb` partition.

Similarly, we send the same total amount of requests under different partitioning plans. For the non-bottleneck partitioning plan, we send requests uniformly. From Figure 6, we can draw a useful observation in comparison to Obs. (5) and (6): **❽ Larger partitions can exhibit lower cumulative intensities by batching requests into larger groups.** This is because batching can improve both processing and energy efficiency. As seen across all the six models, both the average GPU power and maximum GPU temperature decrease, with the latter reducing by an average of 3℃ and up to 8℃.

3.2.4 Experiment #4 on a representative scheduling framework. Finally, to show how existing scheduling frameworks behave in terms of thermal distribution, we conduct a trace-driven experiment on the Kubernetes framework [22]. Specifically, we configure a small cluster with two worker nodes, each equipped with four A100 GPUs. Each GPU is partitioned as $3 \times$ `1g.5gb` $+ 2 \times$ `2g.10gb`. We select four models: SSD300-VGG16, YOLOv8x, Diffusion-v1.5, and Gemma-2b. The latter two models, due to their high memory requirements, are deployed exclusively on each `2g.10gb` partition, while the former two are deployed on each `1g.5gb` partition. We use a trace from Twitter [4] to generate user requests and select a period from 9:00 to 9:30 where the requests per second (RPS) fluctuates in a large range from about 35 to 120, as plotted in Figure 7. Figure 8 shows the power and temperature variations across four GPUs on Node #01 during the 30 minutes. The types and quantities of models deployed on each GPU are indicated below each GPU's power pattern. As we can see, there exhibit significant power and temperature differences among the four GPUs. For example, GPU #03 experiences a maximum temperature of 54℃, while GPU #02 remains no more than 44℃, highlighting the significant thermal imbalance of inference workloads on Kubernetes.

Based on the above measurement study and the cooling mechanism discussed in Section 2.2, we conclude that the thermal imbalance arises from three major aspects as follows.

(1) **Model heterogeneity:** AI models differ in size and computational demands, resulting in different levels of heat generation. The size of GPU partitions that hold these models further affects heat intensity.
(2) **Request heterogeneity:** Batched requests differ in size (e.g., batch size, input length, image size), and a sequence of requests forms distinct arrival patterns, both of which result in varying heat intensity. For different models, the relationship between request size and power usage also varies.
(3) **Cooling non-identity:** As water flows through components along each cooling channel in servers, the downstream component tend to be hotter than the upstream one. This temperature rise depends on the power consumption of the preceding component.

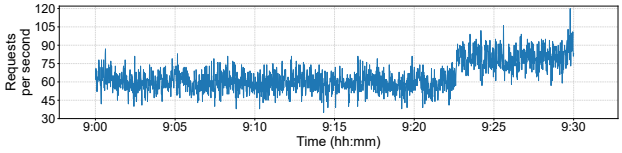


Figure 7: RPS variations from 9:00 to 9:30 of the trace.

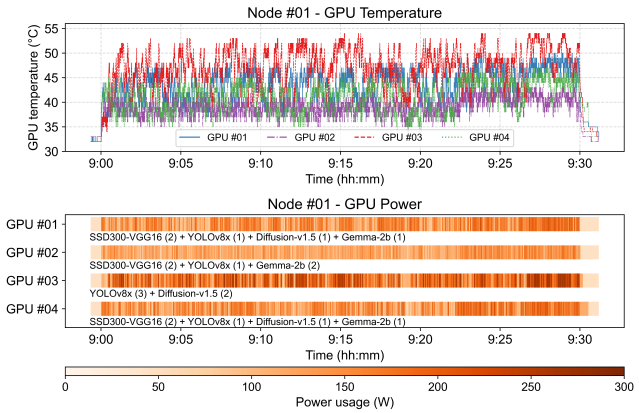
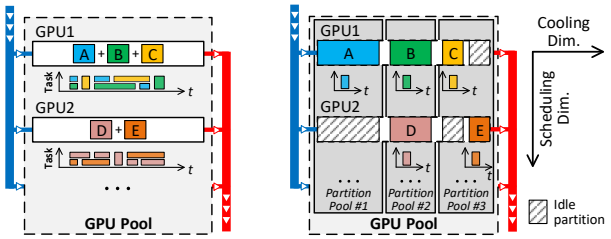


Figure 8: The GPU power and temperature variations when processing inference requests on Kubernetes.

In summary, AI inference workloads make the hardware power and temperature fluctuate severely both spatially and temporally, while existing cluster schedulers are cooling-unaware, leaving a large gap towards optimal cooling efficiency. Moreover, the impact of AI inference on cooling is highly complex—the hardware temperature is significantly influenced by many factors: model type, partition size, batch size, request arrival pattern, and physical layout of GPUs. To optimize AI inference cooling, we must comprehensively account for these key factors and the above three aspects.

3.3 Opportunity for Cooling Awareness

As discussed in Section 1, existing studies on improving cooling efficiency face significant limitations in maintaining both reliability and extensibility. By comparison, we explore a new opportunity for achieving efficient cooling in MIG-enabled AI inference serving. As illustrated in Figure 9a, previous GPU-sharing methods, including spatial and temporal sharing via GPU virtualization and MPS technologies, treat the entire GPU component as the scheduling unit, which is the same as the cooling unit. If the cooling-unaware cluster scheduler makes a decision that leads to thermal imbalance, it is highly challenging to realize thermal balance without affecting scheduling requirements, since any adjustments would inevitably impact application performance due to changes in resource contentions [53]. However, as shown in Figure 9b, with MIG-enabled AI inference, the cooling still occurs at each individual GPU component, but the scheduling unit shifts to the *partitions* on all GPUs, independent of



(a) Prior GPU-sharing. (b) MIG-enabled GPU-sharing.

Figure 9: MIG-enabled GPU-sharing presents a new opportunity to decouple cooling and scheduling.

the physical entity, thus decoupled from the cooling unit. In the MIG-enabled Kubernetes platform we set up, the scheduler interacts only with the MIG partitions instead of the GPU components on each node. In this scenario, when the scheduler makes a decision that causes thermal imbalance, we can reassign application pods between hotter and cooler GPUs within the same partition pool, while preserving the original scheduling requirements and runtime performance.

Despite the potential for achieving thermal balance in a non-intrusive manner, two key challenges must be addressed to fully leverage this opportunity. First, given the unpredictable nature of runtime request arrivals, we need to accurately assess the power and temperature impacts of each application, as well as those of co-located applications and concurrent requests, on each GPU entity in real time. Second, due to the huge search space for application co-location plans, we must make the optimal application deployment and request scheduling decisions with minimal overhead, while ensuring ease of use and compatibility with existing cluster schedulers. To this end, we propose a lightweight cooling middleware designed to tackle these challenges, contributing to greener AI inference serving.

4 MIDDLEWARE ARCHITECTURE

In this section, we formally propose InferCool, a thermal-aware cooling middleware for AI inference serving.

4.1 System Workflow

Figure 10 shows the system overview of InferCool. It consists of three major parts: the bottom cooling system, the middle hardware pool consisting of many water-cooled, partitioned GPUs, and the top scheduling system. InferCool operates between the physical infrastructure and top-level schedulers (i.e., existing cluster schedulers like Kubernetes [22]), as outlined by the green rectangle. It manages thermal distribution and GPU temperatures through three main processes: application deployment and re-deployment at the application level, request scheduling at the request level, and cooling adjustment at the cooling level. Application-level management

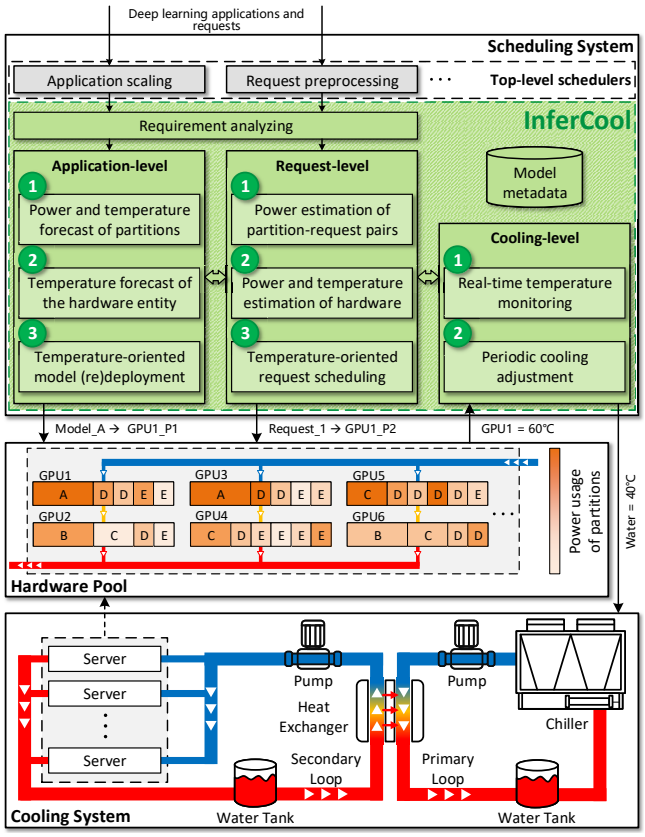


Figure 10: Middleware architecture of InferCool.

plays a fundamental role in thermal balancing, while request-level management enables subsequent dynamic adjustments. Cooling-level management interacts with both application- and request-level management and directly regulates the cooling system. In the following, we detail the workflow of each management level.

Application-level management. When InferCool receives an application deployment command (including initial deployment and runtime scaling in/out) from the top-level scheduler, it first analyzes the deployment requirements (e.g., affinity) specified by the scheduler and generates a list of candidate components that satisfy the requirements. Then, InferCool forecasts the power and temperature effects of each application based on the model metadata and historical request arrival patterns, while the temperature effects on the GPU entity are also influenced by other co-located applications. Finally, InferCool chooses the optimal candidates for all application pods aiming to minimize the maximum GPU temperature in the next adaptation period (e.g., `Model_A` → `GPU1_P1`). Note that only when the top-level scheduler sends a deployment command will InferCool decide a new physical location with thermal awareness. This generally does not impact runtime application performance, as the new decision will only be a partition with the same configuration.

Request-level management. When InferCool receives a request-scheduling command from the top-level scheduler, it first analyzes the scheduling requirements (e.g., hardware type, partition size) and request size and obtains a list of lightly loaded partitions as candidates. Then, InferCool estimates the power and temperature effects of processing the request on each partition based on the model metadata and the request size, while the temperature effects on the GPU entity are influenced by other concurrent requests on the same GPU. Finally, InferCool chooses the optimal partition candidate that minimizes the maximum GPU temperature before the request can complete (e.g., `Request_1` → `GPU1_P2`).

Cooling-level management. InferCool collects real-time temperature data from all GPUs (e.g., `GPU1 = 60℃`) for making application- and request-level decisions. In each adaptation period, based on the forecasted maximum reachable hardware temperature, InferCool adjusts the water temperature (e.g., `Water = 40℃`) to save cooling energy while ensuring that all hardware operates below its predefined safe operating temperature (SOT) [13, 19, 39]. In rare cases where InferCool detects that a hardware component's temperature exceeds the SOT, it will immediately lower the water temperature and enforce thermal-aware request scheduling until the temperature drops below the SOT again.

By enabling different modules, InferCool offers three levels of task reassignment for datacenter operators to choose from. Higher levels offer greater cooling energy savings but may have a slight impact on runtime application performance.

(1) **Level 1: Application deployment only.** The top-level scheduler specifies the number of pods to be deployed for each application, the partition size for each pod, as well as other deployment requirements, such as affinity. InferCool then generates a thermal-aware deployment plan without violating any of these requirements. Notably, this reassignment level does not impact runtime application performance.
(2) **Level 2: Application deployment and re-deployment.** As the system operates, applications may run smoothly without the need for frequent scaling in or out. However, thermal imbalances can arise as the request arrival patterns of different applications gradually change. In such cases, InferCool will perform runtime application re-deployment (i.e., live migration [15]) of some heat-intensive pods to balance the thermal distribution. This process occurs infrequently after thermal-aware application deployment, and workload fluctuations do not directly affect spatial temperature disparities.
(3) **Level 3: Application deployment, re-deployment, and request scheduling.** Typically, top-level schedulers implement their own request scheduling strategies that determine which pod should process each request. However, in certain cases, applications may not have strict performance requirements and prioritize cost reduction (e.g., during off-peak periods or during hot days). InferCool provides this capability by enabling performance- and thermal-aware request scheduling, given that an application typically deploys multiple pods across different physical GPUs.

```
application name: ResNet-152
reassignment level: 1

constraint NodeAntiAffinity {
    Node 005;
    Node 010;
}

constraint PodAffinity {
    Pod YOLOv8x;
}

application name: BERT-large
reassignment level: 2

constraint NodeAffinity {
    Node 005-010;
}

constraint PodAntiAffinity {
    Pod BERT-small;
    Pod GPT2-xl;
}
```

(a) Permanent requirement specs in the global list

```
application name: ResNet-152
reassignment level: 1

constraint NodeAntiAffinity {
    Node 005;
    Node 010;
}

constraint PodAffinity {
    Pod Diffusion-v1.5;
}
```

(b) Temporary requirement spec by each application

```
application name: BERT-large
reassignment level: 3
batch size: 8
input length: 50
```

(c) Temporary requirement spec by each request

Figure 11: Examples of command interfaces.

The top-level scheduler can select one of the above three reassignment levels[3] and specify deployment requirements through a series of command interfaces. In the following, we will detail how the scheduler interacts with InferCool.

4.2 Command Interface

InferCool maintains a global list that records the reassignment level used by each application, along with the deployment requirements — physical deployment constraints. InferCool currently supports four common deployment constraints for MIG-enabled inference serving: `NodeAffinity`, `NodeAntiAffinity`, `PodAffinity`, and `PodAntiAffinity`, which are widely supported by existing schedulers [6, 23]. Figure 11a shows an example of the global list. If no GPU partitions can satisfy all the constraints, InferCool will attempt to satisfy them one by one. By default, the reassignment level is set to 1 to avoid any potential performance impact, and the list can be updated by the top-level scheduler at any time.

Each time the top-level scheduler sends an application deployment command, specifying the number of pods to scale in or out and their partition sizes, it can also specify a temporary reassignment level and deployment constraints that take higher priority than those in the global list, but are

[3] In addition to the three thermal-aware levels, setting the level to zero will completely disable InferCool for specific uncommon scenarios.

valid for this deployment only. Figure 11b shows an example of the deployment command, where the ResNet-152 model adjusts its `PodAffinity` so that it will be scheduled to a GPU where the Diffusion-v1.5 model resides. For a request scheduling command, InferCool generally adheres to the application's original request scheduling strategies, such as percentile latency guarantees and throughput maximization, when the reassignment flag in the global list is set to 1 or 2. InferCool will perform thermal-aware request scheduling in two cases. (a) The reassignment flag is set to 3 in the global list: InferCool will make its own scheduling decisions for all requests. (b) The scheduling command specifies a temporary reassignment level of 3: InferCool will enable thermal-aware scheduling only for that specific request. Figure 11c shows an example.

5 MODULE DESIGN

In this section, we detail the lightweight design of each module, including the power and temperature prediction approach and thermal-aware application (re-)deployment and request scheduling mechanism.

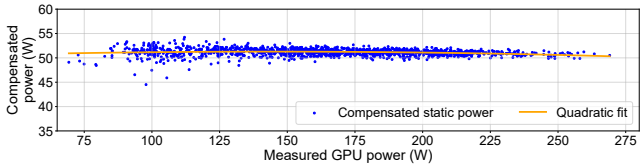
5.1 Power Model

Before making any thermal-aware decisions, InferCool needs to predict the power usage of all partitions on each GPU. Let G_i be the i-th GPU, M_j be the j-th model ($M_j \in G_i$ if M_j running on G_i), and $|G_i|$ be the number of models running on G_i. P^S represents the static power when the GPU is idle which is irrelevant to the workload. $P^D_{j,k,b}$ and $P_{j,k,b}$ represent the average dynamic power and average total power, respectively, for running M_j with a batch size of b on a partition of size k, with all other partitions on the GPU remaining idle. P^{G_i} represents the power consumption of the i-th GPU G_i. Both P^S and $P_{j,k,b}$ are fixed and can be directly measured in advance. Our objective is to estimate P^{G_i} for arbitrary model combinations running on G_i together. According to our measurements, P^{G_i} generally follows the principle of linear superposition:

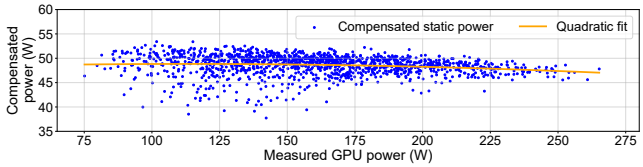
$$\begin{aligned} P^{G_i} &= P^S + \sum_{j, M_j \in G_i} P^D_{j,k,b} \\ &= \sum_{j, M_j \in G_i} P_{j,k,b} - (|G_i| - 1) \cdot P^S, \end{aligned} \tag{1}$$

where $P_{j,k,b} = P^S + P^D_{j,k,b}$.

By measuring P^{G_i} and $P_{j,k,b}$, we can calculate the value of P^S, which we find is slightly different from the ground-truth value of $P^S = 46.7$ W when the GPU is in the idle state. Figure 12 shows the relationship between P^S and P^{G_i} as the number of partitions running inference tasks increases under two different GPU partitioning plans. To address this discrepancy, we replace P^S with the compensated power



(a) The GPU partitioning plan is 7 × 1g.5gb



(b) The GPU partitioning plan is 1 × 1g.5gb + 3 × 2g.10gb

Figure 12: The "static power" is related to the GPU partitioning plan and total GPU power.

P^{S_i}, which is modeled as a function of P^{G_i} under a given partitioning plan:

$$P^{S_i} = f\left(P^{G_i}\right). \tag{2}$$

Table 2 presents the fitting results for P^{S_i} across four selected partitioning plans. These results imply two interesting phenomena: (a) Higher hardware utilization may negatively impact dynamic energy consumption, and the gains in overall energy efficiency from increased utilization diminish, and (b) Finer-grained partitioning plans generally result in higher energy efficiency, even when the hardware utilization and GPU power are comparable to those under coarser-grained partitioning plans. Based on these fitting results, for a collected dataset of 1,500 samples (comprising different numbers of partitions used, various model combinations, and different batch sizes) for each partitioning plan, the average estimation errors of GPU power are 1.06%, 1.37%, 1.62%, and 1.79%, for the four listed partitioning plans, respectively. Notably, when directly using the fixed value of P^S, the estimation errors are as high as 8.58%, 6.97%, 4.81%, and 2.92%, respectively.

Table 2: The fitting results of P^{S_i}

GPU partitioning plan	Quadratic fitting result
7 × 1g.5gb	$P^{S_i} = -0.000061 \cdot (P^{G_i})^2 + 0.0177 \cdot P^{G_i} + 50.01$
5 × 1g.5gb + 1 × 2g.10gb	$P^{S_i} = -0.000030 \cdot (P^{G_i})^2 + 0.0033 \cdot P^{G_i} + 51.02$
3 × 1g.5gb + 2 × 2g.10gb	$P^{S_i} = -0.000034 \cdot (P^{G_i})^2 + 0.0050 \cdot P^{G_i} + 49.99$
1 × 1g.5gb + 3 × 2g.10gb	$P^{S_i} = -0.000078 \cdot (P^{G_i})^2 + 0.0177 \cdot P^{G_i} + 47.83$

5.2 Temperature Model

Then, based on the estimated GPU power, InferCool needs to predict temperature variations over time, as the temperature varies asynchronously with the power.

We first derive the relationship between power and temperature at steady states. Let $P^{G_i}_{\text{steady}}$ be the steady-state power of the i-th GPU G_i, corresponding to a steady-state GPU temperature $T^{G_i}_{\text{steady}}$. The cooling water temperature is maintained

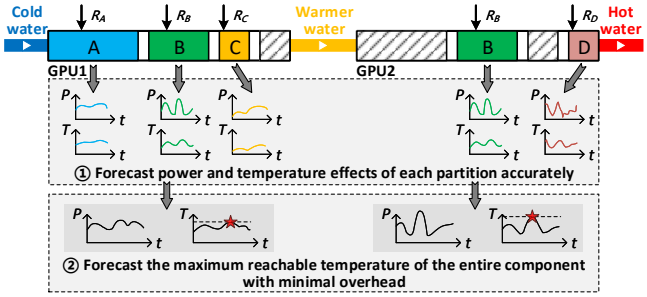


Figure 13: Two key steps for thermal-aware application deployment. The red pentagrams indicate the forecasted maximum reachable temperature of each GPU.

at T_w. Based on Newton's law of cooling, we have:

$$P^{G_i}_{\text{steady}} = hA(T^{G_i}_{\text{steady}} - T_w), \tag{3}$$

where h and A denote the convective heat transfer coefficient and the contact area between the GPU component and the cooling water, respectively, and can be viewed as constants. When the GPU power changes to a new value $P^{G_i}_{\text{steady}'}$, the GPU will eventually reach a new steady-state temperature $T^{G_i}_{\text{steady}'}$, and they follow the same relationship as in Equation 3.

Although the power fluctuates in real-time, the temperature changes comparatively much slower. For short-lived AI inference workloads, however, the GPU temperature typically cannot reach a steady state before its power changes significantly again, as discussed in Section 3.2. Now we derive the temperature change over time $T(t)$ when the power changes to $P^{G_i}_{\text{steady}'}$, which can be described by Newton's law of cooling and the law of conservation of energy:

$$\frac{dQ_{\text{loss}}}{dt} = hA(T(t) - T_w), \tag{4}$$

$$\frac{dQ_{\text{absorb}}}{dt} = mc\frac{dT(t)}{dt}, \tag{5}$$

where m and c are the mass and the specific heat capacity of the GPU, respectively, and can be viewed as constants. $\frac{dQ_{\text{loss}}}{dt}$ and $\frac{dQ_{\text{absorb}}}{dt}$ denotes the rate of heat loss to the water and heat absorbed by the GPU component, respectively, and their sum is equal to the current GPU power $P^{G_i}_{\text{steady}'}$:

$$P^{G_i}_{\text{steady}'} = \frac{dQ_{\text{loss}}}{dt} + \frac{dQ_{\text{absorb}}}{dt}. \tag{6}$$

Combining Equations 4, 5, and 6, we have:

$$\frac{dT(t)}{dt} = \frac{P^{G_i}_{\text{steady}'}}{mc} - \frac{hA}{mc}(T(t) - T_w). \tag{7}$$

By solving this first-order linear differential equation and using the initial condition $T(0) = T_0$ and final condition $T^{G_i}_{\text{steady}'} = T_w + \frac{P^{G_i}_{\text{steady}'}}{hA}$ with Equation 3, we can get:

$$T(t) = (T_0 - T^{G_i}_{\text{steady}'})e^{-\alpha t} + T^{G_i}_{\text{steady}'}, \tag{8}$$

where $\alpha = \frac{hA}{mc}$.

For intermittent inference executions, to get the transient GPU temperature $T(t)$ at any time before the power changes largely again, we need to estimate the value of hA for calculating $T^{G_i}_{\text{steady}'}$ with Equation 3, as well as α. We conduct measurements by setting the GPU at different power levels and tuning it from one to another, and obtain the values of hA and α for the 40GB A100 GPU as 13 and 0.51, respectively.

5.3 Thermal-Aware Application Deployment and Redeployment

Based on the power and temperature models, when the top-level scheduler sends an application deployment command, InferCool will decides on an optimal deployment plan that leads to the lowest maximum GPU temperature across the cluster, thereby achieving thermal balance and allowing for a higher coolant temperature. Figure 13 illustrates the two key steps for generating an optimal deployment plan. Focusing on accurate temperature prediction while maintaining a lightweight design, two main challenges arise, which are described as follows.

First, in Step ①, which involves forecasting the power and temperature effects of each partition, *while the model type and partition size can be determined before deployment, the specific request arrival pattern is unpredictable [2].* Consider two extreme cases: (a) When the RPS is high, e.g., approaching the maximum throughput supported by each pod, the maximum temperature is primarily influenced by the cumulative intensity owing to a high likelihood of receiving many large-sized requests in a short period. (b) When the RPS is very low, the maximum temperature becomes dependent on the individual intensity, with long idle intervals between adjacent requests. However, it is costly to estimate the temperature effects under various possible patterns for each application. Second, in Step ②, despite knowing the temperature effects of every partition from Step ①, forecasting the maximum reachable temperature of the entire GPU component remains challenging, as *the cumulative effects of these partitions on the entire GPU's temperature variations are highly coupled.* One possible solution may be exhaustively estimating the temperature effects of every possible model combination, but its complexity is $O(|M|^K)$ with $|M|$ pods and a maximum of K partitions on a GPU.

We draw two key observations to tackle the above challenges. First, we observe that under a given average request rate and distribution (e.g., Poisson or Gamma), the **temperature distribution** generally remains stable despite variations in specific patterns. Second, we find that, according to Equations 1, 3, and 8, the sum of the **temperature increases** (i.e., temperature values relative to the idle-state GPU temperature) across all partitions is approximately equal to the overall temperature increase of the GPU component (Due

Table 3: Notations

Symbol	Definition
M	The set of application pods to be deployed
N	The set of available GPUs
m_{ji}	Binary variable where $m_{ji} = 1$ if the j-th pod is deployed on the i-th GPU, and $m_{ji} = 0$ otherwise
k_j	Required partition size of the j-th pod
K	Resource capacity of each GPU. For A100, $K = 7$ (i.e., 7g)
$\mathcal{L}(T_j)$	Distribution of the temperature increase for the j-th pod
T_i^G	The temperature rise of the downstream i-th GPU caused by the rise in water temp. flowing from the upstream one

to space constraints, we provide a proof for this in [38]). Hence, at Step ① in Figure 13, we propose using the Monte-Carlo method to obtain the temperature distribution, which integrates various runtime factors while maintaining low complexity, as detailed in Algorithm 1. As the Monte-Carlo simulation can provide a precise forecast of the temperature distribution, InferCool only requires a single simulation for each pod configuration (including application type, hardware type, and partition size).

After obtaining the **distributions of temperature increases** from the Monte-Carlo simulations, we can describe the initial pod deployment process as a mixed-integer linear programming (MILP) problem. Table 3 lists all the variables and their definitions. The goal is equivalently transformed into deploying all $|M|$ pods onto $|N|$ GPUs such that the total resource demands on any GPU do not exceed its capacity, and the expected maximum temperature increase across all GPUs is minimized. We define Z as $\max_i \left(\sum_{j=1}^{|M|} m_{ji} \cdot \mathcal{L}(T_j)\right)$, and the problem can be formulated as follows:

$$\min\ Z \tag{9}$$

$$\text{s.t.} \sum_{i=1}^{|N|} m_{ji} = 1,\ \forall j \in \{1, 2, \ldots, |M|\}, \tag{10}$$

$$m_{ji} \in \{0, 1\},\ \forall j \in \{1, 2, \ldots, |M|\}, \forall i \in \{1, 2, \ldots, |N|\}, \tag{11}$$

$$\sum_{j=1}^{|M|} m_{ji} \cdot k_j \le K,\ \forall i \in \{1, 2, \ldots, |N|\}, \tag{12}$$

$$T_i^G + \sum_{j=1}^{|M|} m_{ji} \cdot \mathcal{L}(T_j) \le Z,\ \forall i \in \{1, 2, \ldots, |N|\}. \tag{13}$$

Equation 10 denotes whether the pod M_j is deployed to G_i or not, while Equation 11 guarantees that each pod is deployed to one and only one partition. Equation 12 ensures that the total required partition size of all pods on a GPU does not exceed its capacity. Equation 13 means that the expected maximum temperature increase across all GPUs is Z. As listed in Table 3, we estimate the value of T_i^G by assuming that this thermal-aware deployment will finally result in a uniform temperature distribution, with the measured relationship between the water temperature increase and GPU power given by $\Delta T_w = 0.0174 \cdot P^G + 0.2425$.

Algorithm 1 Monte-Carlo simulation for each pod

Parameters: Water temperature T_w, idle GPU power P^S, batch size b, inference time t_b, average inference power P_b, historical average arrival interval of requests λ, the constant $\alpha = \frac{hA}{mc}$;
Compute the steady-state temperatures at P^S and P_b with Equation 3: $T_0 \leftarrow T_w + \frac{P^S}{hA}$ and $T_b \leftarrow T_w + \frac{P_b}{hA}$;
for each simulation **do**
  Initialize current time $t \leftarrow 0$, temperature $T_{\text{curr}} \leftarrow T_0$;
  Generate a sequence of request arrival time with exponential distribution: $t_{\text{next}} \leftarrow t_{\text{next}} + \textsc{Exponential}(\lambda)$;
  while $t <$ simulation_time **do**
    if $t < t_{\text{next}}$ **then** *// in idle state*
      $\Delta t \leftarrow t_{\text{next}} - t$, $T_{\text{curr}} \leftarrow T_0 + (T_{\text{curr}} - T_0)e^{-\alpha \Delta t}$, $t \leftarrow t + \Delta t$; *// update temperature and time*
    else *// in running state*
      Set b based on the historical batch size distribution;
      $T_{\text{curr}} \leftarrow T_b + (T_{\text{curr}} - T_b)e^{-\alpha t_b}$, $t \leftarrow t + t_b$;

To convert the distribution of temperature increase $\mathcal{L}(T_j)$ into a calculable upper bound, we notice that the probability of all pods on a GPU reaching their maximum temperatures simultaneously depends on the RPS and reassignment level. Thus, we use the following rules to determine the value: (a) At level 1 or 2, further thermal regulation through request scheduling is not allowed. When the RPS is high, the actual maximum temperature is closer to the estimated median value. Conversely, when the RPS is low, the maximum temperature depends more on the individual intensity and deviates significantly from the median value. Therefore, we choose a percentile value calculated as $\left(1 - \min\left(1, \frac{\text{RPS}}{\text{Throughput}_{\max}}\right)\right) \times 40 + 60$, which ranges in $[60, 100]$ and decreases as RPS increases. (b) At level 3, further thermal balancing is allowed through request scheduling, resulting in smoother temperature variations. Therefore, we choose a lower value, i.e., the 60th percentile of the distribution. Note that to accommodate the deployment constraints listed in Section 4.2, $\mathcal{L}(T_j)$ can be replaced with a GPU-related variable $\mathcal{L}(T_{j,i})$, where setting $\mathcal{L}(T_{j,i})$ to 0 or $+\infty$ indicates `Affinity` or `AntiAffinity` of M_j with G_i, respectively.

InferCool performs application redeployment at runtime if the assignment level is set above 1. Specifically, when InferCool detects that the temperature variance across all GPUs exceeds a predefined threshold, it will migrate some heat-intensive pods, characterized by their converted temperature increase values, from the hottest GPUs to cooler ones, until the variance drops below the threshold. To mitigate migration overhead, InferCool first attempts to move these pods to each idle partition of the same size on cooler GPUs. If no such partition is available, InferCool will swap these heat-intensive pods with other light-loaded pods that have a lower temperature increase value on cooler GPUs.

5.4 Thermal-Aware Request Scheduling

As discussed in Section 3, inference executions are often short-lived, and the GPU temperature varies much more slowly than its power consumption. That is to say, each single inference execution has a limited impact on the future. Therefore, InferCool employs a greedy strategy to schedule requests, while balancing them among partitions serving the same application by avoiding long queuing time on colder GPUs. Algorithm 2 presents the scheduling details. Specifically, InferCool first attempts to find an idle partition to process the request. If no idle partition is available, it sorts non-idle partitions by queue length and filters out the half with the longest queues. Next, for every remaining candidate partition, it predicts the power changes before the request can be completed. To simplify this process, InferCool uses average power over each small time slice (e.g., 500 ms) instead of instantaneous power, as the resulting temperature changes are nearly the same. Then, InferCool estimates the temperature changes in each time slice using Equation 8 until it determines that the request can be completed. Finally, InferCool selects the partition that results in the lowest maximum GPU temperature and assigns the request.

Algorithm 2 Thermal-aware request scheduling

Input: Application index j, partition size k, batch size b, inference time $t_{j,k,b}$, average power consumption $P_{j,k,b}$;
Output: Selected partition $\hat{R}$ for the request;
Initialize $T \leftarrow +\infty$, $\hat{R} \leftarrow$ null;
function EvaluatePartition(R)
    Predict the maximum GPU temperature T^G before the request can be completed using Equations 1, 3, and 8;
    if $T^G < T$ **then**
        $T \leftarrow T^G$, $\hat{R} \leftarrow R$;
for each idle partition R **do** *// Choosing idle partitions first*
    EvaluatePartition(R);
if $\hat{R} \neq$ null **then**
    return $\hat{R}$;
for each non-idle partition R filtered by their queue lengths **do** *// Prioritizing partitions with short queue lengths*
    EvaluatePartition(R);
return $\hat{R}$;

6 EVALUATION

In this section, we conduct extensive experiments to evaluate InferCool. We begin with the evaluation setup and then present the evaluation results. Finally, we discuss the applicability of InferCool in real-world datacenters.

6.1 Evaluation Setup

Experimental environment. We deploy InferCool within Kubernetes on one master node and two worker nodes at CloudLab [12]. The node specifications are listed in Table 4. Since the cloud servers are air-cooled and we cannot control their cooling system, we set up a local water-cooled testbed with one 40GB A100 GPU, as shown in Figure 14, with its specifications also detailed in Table 4. Specifically, we collect inference scheduling data from the cloud servers and replay the scheduling results from each cloud GPU on the local GPU to gather water-cooling data, including GPU power and temperature. As the maximum power consumption of our local PCIe-version GPU is about 260~270 W, to ensure performance consistency between the cloud GPUs and local GPU, we fix their clock frequency at 1350 MHz, slightly lower than the maximum frequency of 1410 MHz. The water temperature and flow rate are maintained at 30℃ and 90 L/h throughout the experiments, with the idle-state GPU temperature at around 33℃.

Implementation. We develop InferCool as a middleware service running within the Kubernetes cluster [22], comprising about 1k lines of Python code. The top-level scheduler can send the application deployment command to InferCool via HTTP requests. InferCool is responsible for parsing and maintaining the global list recording deployment requirements, as well as handling temporary specifications. When the reassignment level is set to 0, the system will automatically bypass all core modules of InferCool except for the temperature monitoring and cooling adjustment modules. When the reassignment level is set to 1 or 2, it only bypasses the request scheduling module. InferCool manages pods through the Kubernetes Python client, and collects MIG information and monitors runtime GPU powers and temperatures with the NVIDIA GPU Operator [34] and DCGM tools [36].

Table 4: Node specifications

Item	Master node	Worker node	Local workstation
CPU	Two Intel(R) Xeon(R) E5-2660 v3 10-core CPUs at 2.60 GHz	Two AMD EPYC 7413 24-core CPUs at 2.65 GHz	One Intel(R) Core(TM) i9-10940X 14-core CPU at 3.30 GHz
Memory	160 GB	512 GB	128 GB
GPU	N/A	Four air-cooled NVIDIA 40GB A100 SXM4 GPUs	One water-cooled NVIDIA 40GB A100 SXM4 to PCIe GPU
System and Software	Ubuntu 20.04, NVIDIA driver 550.54, Kubernetes 1.28, Docker 23.0.1, CUDA 11.6, cuDNN 8.9.7, PyTorch 1.13.1		

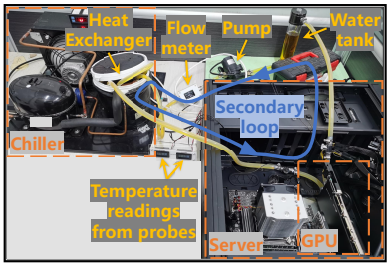
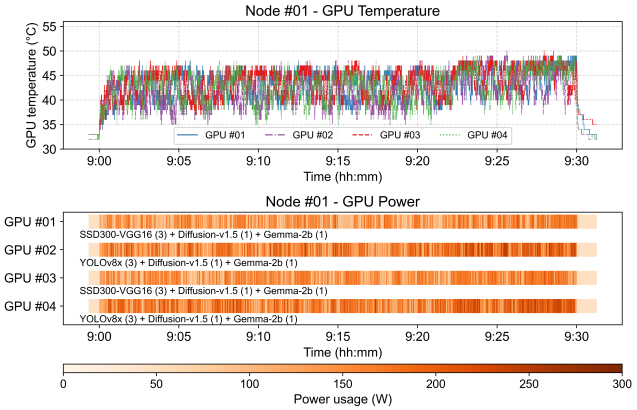


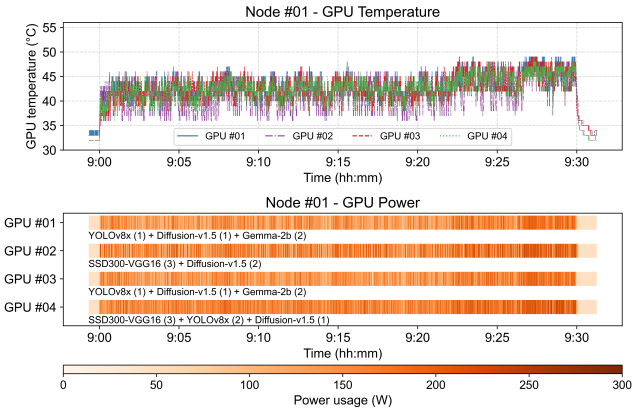
Figure 14: Our water-cooled testbed.

AI inference workloads. We select ten popular open-source AI models from the torchvision package and Hugging Face [16], covering both the CV and NLP domains, with sizes ranging from XS to XL. These models include DenseNet-121 (XS), ResNet-152 (S), SSD300-VGG16 (M), YOLOv8x (L), Diffusion-v1.5 (XL), BERT-base (XS), BERT-large (S), Flan-T5-large (M), GPT2-xl (L), and Gemma-2b (XL). To generate inference requests, we select a real-world dataset from Twitter [4], which is widely used in previous work [2, 40, 52]. This dataset contains arrival interval information of tweets for sentiment analysis. We select a subset spanning the time period from 9:00 to 9:30, which covers a wide range of RPS variations during the day, as plotted earlier in Figure 7. Since the dataset does not contain information on application type and batch size, we randomly assign these attributes to each request (bs $\in [1, 16]$), using the request index as the seed to ensure consistency across multiple tests.

Cooling metric. The cooling energy savings come chiefly from increasing the supply water temperature, while the maximum allowable water temperature is determined by the peak hardware temperature across all servers and components, as discussed in Section 2.2. Therefore, a lower maximum reachable hardware temperature allows for a higher water temperature and thus reduces cooling energy consumption. Previous literature shows that every 1℃ increase in water temperature can save about 4% of cooling energy [13]. Hence, we use the maximum GPU temperature across the cluster in each adaptation period (e.g., 5 minutes) as the key metric for evaluating cooling efficiency.



(a) At reassignment level = 1



(b) At reassignment level = 3

Figure 15: The GPU power and temperature variations with InferCool enabled on Kubernetes.

6.2 Main Performance

First, we evaluate the main performance of InferCool compared to the vanilla Kubernetes framework. We consider a typical water cooling architecture, where GPUs #01 and #03 are positioned downstream of GPUs #02 and #04, respectively. Figure 15 shows the GPU power and temperature variations when InferCool is enabled with the reassignment level set at 1 or 3. Comparing this with the results from vanilla Kubernetes shown earlier in Figure 8, we observe a significant reduction in the maximum GPU temperature. Specifically, as plotted in Figure 15a, with the reassignment level set to 1 (where Kubernetes still manages request scheduling), the maximum GPU temperature drops from 54℃ to 50℃, and the average temperature variance decreases from 18.93 to 6.23. Intuitively, both the YOLOv8x and Diffusion-v1.5 models generally consume more power and generate more heat than SSD300-VGG16 and Gemma-2b, as presented in Section 3.2. Additionally, GPUs #01 and #03 tend to run hotter than GPUs #02 and #04 because GPU #01 (#03) receives water already warmed by GPU #02 (#04). In particular, we find that when the GPU runs at 250 W, the water temperature rise can reach 4.7℃. Furthermore, as plotted in Figure 15b, with the reassignment level set to 3 (where InferCool handles request scheduling), the maximum GPU temperature is reduced further to 49℃ by balancing thermals at runtime, and the average temperature variance drops significantly to 2.46. According to the manual [13], the overall temperature reduction of 5℃ can save about 20% of cooling energy.

Thanks to the lightweight design presented in Section 5, the runtime overhead introduced by InferCool is minimal. First, the Monte-Carlo simulation for each pod configuration is configured to run for 50 ms, for balancing the trade-off between temperature distribution convergence and simulation overhead, which is negligible compared to the subsequent pod-deployment phase. Second, based on the simulated temperature distribution for all pods, generating an optimal initial pod-deployment plan with the MILP solver takes only hundreds of milliseconds to several seconds. Note that this initial pod-deployment process occurs infrequently, such as when applications are first launched or when there are significant workload changes, and subsequent pod-deployment decisions only involves the Monte-Carlo simulation. Third,

Table 5: Adaptability of InferCool to various scenarios

Platform	The max. GPU temperature (℃) in each adaptation period (min.) (The lower, the better)					
	0~5	5~10	10~15	15~20	20~25	25~30
Partitioning plan: 7 × 1g.5gb, Trace: original trace, Models: DenseNet-121 (8) + SSD300-VGG16 (8) + YOLOv8x (8) + Diffusion-v1.5 (8) + BERT-base (8) + BERT-large (8) + GPT2-xl (8)						
Kubernetes	37.8	37.7	37.7	38.2	37.7	38.2
InferCool (1)	37.3	38.2	37.6	37.3	39.0	38.6
InferCool (3)	**36.9**	**37.0**	**37.4**	**36.9**	**37.7**	**37.6**
Partitioning plan: 7 × 1g.5gb, Trace: scaled up by 11.8×, Models: DenseNet-121 (8) + SSD300-VGG16 (8) + YOLOv8x (8) + Diffusion-v1.5 (8) + BERT-base (8) + BERT-large (8) + GPT2-xl (8)						
Kubernetes	49.5	50.3	51.0	49.9	51.6	53.3
InferCool (1)	**47.7**	48.4	**47.8**	**48.3**	50.7	52.4
InferCool (3)	47.8	**48.0**	49.4	48.7	**49.3**	**49.6**
Partitioning plan: 2 × 2g.10gb + 1 × 3g.20gb, Trace: scaled up by 3.3×, Models: Diffusion-v1.5 (4) + BERT-large (8) + Flan-T5-large (8) + Gemma-2b (4)						
Kubernetes	46.9	48.3	48.5	46.3	48.3	48.4
InferCool (1)	45.0	46.0	46.0	44.7	46.3	47.3
InferCool (3)	**44.4**	**44.7**	**43.7**	**43.7**	**45.1**	**45.9**
Partitioning plan: 1 × 1g.5gb + 3 × 2g.10gb, Trace: scaled up by 5.1×, Models: ResNet-152 (4) + YOLOv8x (4) + Diffusion-v1.5 (12) + GPT2-xl (12)						
Kubernetes	52.0	51.4	50.5	51.9	53.1	53.8
InferCool (1)	48.5	47.3	47.6	47.7	48.8	49.8
InferCool (3)	**46.1**	**45.1**	**45.9**	**45.9**	**48.4**	**48.7**

* The idle-state GPU temperature is around 33℃.

the runtime re-deployment process involves live migration of heat-intensive pods. Depending on the threshold for assessing thermal imbalance, this process typically happens infrequently (e.g., every 10,000 requests), and can decrease to a very low frequency if top-level schedulers actively scale pods in and out as workloads fluctuate. With the method proposed in [15], the migration overhead can be reduced to less than 100 ms. Finally, thermal-aware request scheduling involves estimating temperature variations and selecting an optimal pod for each request, which takes only 1.6 ms on average, insignificant compared to the original serving time, which ranges from tens of milliseconds to several seconds.

6.3 Adaptability to Various Scenarios

To evaluate how the application deployment and request scheduling modules adapt to various scenarios, we consider different GPU partitioning plans, model combinations, and request arrival rates by scaling up the original trace to approach the maximum supported throughput of each model. The request ratio for each model is set proportional to its maximum supported throughput. Table 5 presents the maximum GPU temperatures (converted from those under air cooling) under vanilla Kubernetes and InferCool at different reassignment levels. The reassignment level is denoted by the number following the name of InferCool. Each experiment heading includes the partitioning plan, trace details (either original or scaled), and model combinations, with the number of pods for each model shown following the model name. Across the four settings, InferCool reduces the maximum GPU temperature by up to 6.3℃. As we can see, as the arrival rate and the proportion of large models increase, the GPU temperatures are getting higher, leading to more serious thermal imbalances, where InferCool brings more significant temperature reductions. For example, in the last three heat-intensive scenarios, which better represent future trends, InferCool (at reassignment level 3) reduces the maximum temperature increase in each period from 17.3℃ to 13.7℃ on average, translating into cooling energy savings of about 14.4% [13]. Note that InferCool achieves this without sacrificing reliability or extensibility and remains non-intrusive to existing cooling infrastructures and scheduling systems.

6.4 Applicability in Real-World Datacenters

The above evaluation results demonstrate impressive performance of InferCool in improving cooling efficiency. However, two important considerations should be noted for real-world deployment. (1) The GPU temperature reduction results are based on a small-scale testbed. When applied to large-scale, real-world datacenters, additional factors must be considered, e.g., the changeable ambient environment, thermal influence from other jobs in co-located datacenters, and the presence of other kinds of workloads in addition to inference, all of which can increase the complexity of task reassignment. (2) The cooling energy savings are estimated for general cases based on the relationship between the coefficient of performance (COP) of chillers and chilled water temperature, as described in [13]. However, the actual potential for energy savings can vary from datacenter to datacenter, depending on the cooling environment (e.g., ambient conditions, cooling load, and the type and size of the chiller), and can be characterized by how the COP of chillers changes with different water temperature set-points in specific environments.

On the other hand, InferCool may offer even greater benefits in real-world datacenters. In our experimental settings, there are only eight GPUs in total, each with a power limit of just 260~270 W. As the scale of datacenters and the diversity of deployed models grow, the thermal imbalance is likely to become more significant [19], which brings a greater opportunity for thermal-aware task reassignment. Additionally, as modern GPUs adopted in datacenters are getting more power-intensive, e.g., 1 kW per B200 GPU, the thermal issues are also expected to intensify in the near future.

7 RELATED WORK

Co-located inference serving. As the computing power of modern GPUs grows, to fully utilize their capabilities for inference serving, many studies focus on co-locating multiple models using various GPU-sharing methods, including time sharing [8, 43, 45, 51] and space sharing with customized

Table 6: Existing software-based cooling-aware approaches are difficult to integrate with cluster schedulers

Study	Main method	Cooling optimization objective	Placement outcome	Handling of application requirements
MACEEC [7]	Task scheduling + Cooling control (air flow rate and air supply temperature)	Minimizing total energy while avoiding overheating	Physically: a specific server	Satisfying the required amount of resources (soft guarantee)
DeepEE [42]	Task scheduling + Cooling control (airflow rate)	Minimizing PUE while avoiding overheating	Physically: a specific server	Balancing the load
VMT [47]	Job scheduling in a cluster utilizing phase change materials	Reducing peak cooling load	Physically: a particular server in a subset	No consideration
ATAC [50]	Hardware control (power capping) + Cooling control (supply air temperature)	Minimizing cooling energy while avoiding overheating	Physically: a server in a subset with sufficient power quota	No consideration
JETC [5]	CPU scheduling & Memory page migration + Cooling control (fan speed)	Minimizing memory and cooling energy while avoiding overheating	Physically: a specific CPU and memory module	No consideration
Moore et al. [30]	Workload placement (hardware power states)	Minimizing cooling energy	Physically: a server in a subset with a specific power state	No consideration

memory management [51], MPS [8, 9, 11], and MIG [9, 25, 26]. Romero et al. [43] develop an automated inference serving system that supports many key features, including time-sharing of resources. However, with the time-sharing method only, many AI models still cannot saturate the entire GPU, and there can be heavy context-switching overhead. Choi et al. [8] combines both time and space sharing to further boost resource efficiency when serving multiple AI models.

As compared with previous GPU-sharing methods, modern MIG technology provides each application with exclusive access to a GPU slice by partitioning the GPU at the hardware level [31]. Li et al. [24] offer a detailed characterization of performance and energy behaviors of AI inference on MIG slices. They further propose a lightweight, dynamic partitioning technique to improve inference performance [25] and even achieve carbon reduction through adaptively selecting MIG slices and model variants [26]. Although current MIG technology has its limitations, e.g., supporting only fixed partition sizes and having relatively high re-partitioning overhead [25], it achieves a good balance between high resource efficiency and predictable performance, with the latter being crucial for mission-critical inference applications [14]. Orthogonal to existing MIG-based studies on inference performance, we explore the new opportunity of improving cooling efficiency without being intrusive to both existing scheduling systems and cooling infrastructures.

Datacenter cooling. As energy consumption of datacenters grows rapidly and cooling systems account for a significant portion of non-IT energy use, researchers are increasingly focusing on improving cooling efficiency. In addition to developing new cooling architectures [17–19, 39, 46, 47], which are often intrusive to existing infrastructures and involve substantial costs, many software-based approaches have been proposed, including thermal-aware workload scheduling, power throttling, and/or cooling control [1, 5, 7, 29, 30, 42, 47, 50]. Table 6 summarizes some closely related work. Although these approaches can effectively reduce cooling energy consumption, they show significant limitations in compatibility with existing cluster schedulers and in handling original application requirements, as they rely on developing custom schedulers for cooling optimization. Motivated by this practical concern, InferCool digs out the unique opportunity presented by MIG-enabled inference serving, where scheduling and cooling dimensions are orthogonal. Through its lightweight middleware design, InferCool achieves impressive cooling energy reduction without being intrusive.

8 CONCLUSION

Today, with the rapid growth of AI-centric applications like ChatGPT, many companies have heavily invested in building AI datacenters, where inference serving has become a crucial and prominent cloud workload. However, the substantial energy consumption of AI has also raised concerns, particularly as cooling energy can even rival IT energy consumption in managing the significant heat generated by AI workloads. Our measurement study systematically examines the cooling impacts of AI inference workloads, presenting a novel perspective on the challenges of achieving efficient cooling while adhering to the requirements of cooling systems. To address these challenges, we propose InferCool, a flexible cooling middleware designed for MIG-enabled AI inference serving, featuring transparent and non-intrusive task reassignment. We implement InferCool as an easy-to-use Kubernetes service that can also be easily extended to other scheduling frameworks. Extensive experiments on a water-cooled testbed and a three-node cloud platform show that InferCool can lower the maximum GPU temperature by 5℃, resulting in about 20% savings in cooling energy.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd Ryan Marcus for their valuable feedback, as well as the CloudLab team for providing A100 GPU servers. This work was funded in part by National Science Foundation of China under grant 62232012 and in part by National Key Research & Development (R&D) Plan under grant 2022YFB4501703.

REFERENCES

[1] Zahra Abbasi, Georgios Varsamopoulos, and Sandeep KS Gupta. 2012. TACOMA: Server and Workload Management in Internet Data Centers Considering Cooling-Computing Power Trade-Off and Energy Proportionality. *ACM Transactions on Architecture and Code Optimization (TACO)* 9, 2 (2012), 1–37.

[2] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2020. BATCH: Machine Learning Inference Serving on Serverless Platforms with Adaptive Batching. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[3] Georgios Andreadis, Laurens Versluis, Fabian Mastenbroek, and Alexandru Iosup. 2018. A Reference Architecture for Datacenter Scheduling: Design, Validation, and Experiments. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 478–492.

[4] Archive Team. [n. d.]. Archive Team: The Twitter Stream Grab. Retrieved June 29, 2024 from https://archive.org/details/twitterstream

[5] Raid Ayoub, Rajib Nath, and Tajana Rosing. 2012. 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*.

[6] Romil Bhardwaj, Alexey Tumanov, Stephanie Wang, Richard Liaw, Philipp Moritz, Robert Nishihara, and Ion Stoica. 2022. ESCHER: Expressive Scheduling with Ephemeral Resources. In *Proceedings of the 13th Symposium on Cloud Computing*. 47–62.

[7] Ce Chi, Kaixuan Ji, Avinab Marahatta, Penglei Song, Fa Zhang, and Zhiyong Liu. 2020. 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*. 489–495.

[8] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2022. Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 199–216.

[9] Weihao Cui, Han Zhao, Quan Chen, Ningxin Zheng, Jingwen Leng, Jieru Zhao, Zhuo Song, Tao Ma, Yong Yang, Chao Li, and Minyi Guo. 2021. Enable Simultaneous DNN Services Based on Deterministic Operator Overlap and Precise Latency Prediction. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[10] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. 2021. Compute and Energy Consumption Trends in Deep Learning Inference. *arXiv preprint arXiv:2109.05472* (2021).

[11] Aditya Dhakal, Sameer G Kulkarni, and KK Ramakrishnan. 2020. GSLICE: Controlled Spatial Sharing of GPUs for a Scalable Inference Platform. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 492–506.

[12] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 1–14.

[13] Energy Efficiency Manual. 2015. Keep the chilled water supply temperature as high as possible. Retrieved October 15, 2024 from http://energybooks.com/wp-content/uploads/2015/07/264266.pdf

[14] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like Clockwork: Performance Predictability from the Bottom Up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 443–462.

[15] Fan Guo, Yongkun Li, John CS Lui, and Yinlong Xu. 2019. DCUDA: Dynamic GPU Scheduling with Live Migration Support. In *Proceedings of the ACM Symposium on Cloud Computing*. 114–125.

[16] Hugging Face. [n. d.]. Hugging Face. Retrieved October 15, 2024 from https://huggingface.co/

[17] Arman Iranfar, Ali Pahlevan, Marina Zapater, and David Atienza. 2019. Enhancing Two-Phase Cooling Efficiency through Thermal-Aware Workload Mapping for Power-Hungry Servers. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition*.

[18] Majid Jalili, Ioannis Manousakis, Íñigo Goiri, Pulkit A Misra, Ashish Raniwala, Husam Alissa, Bharath Ramakrishnan, Phillip Tuma, Christian Belady, Marcus Fontoura, and Ricardo Bianchini. 2021. Cost-Efficient Overclocking in Immersion-Cooled Datacenters. In *Proceedings of the 48th Annual International Symposium on Computer Architecture*.

[19] Weixiang Jiang, Ziyang Jia, Sirui Feng, Fangming Liu, and Hai Jin. 2019. Fine-grained Warm Water Cooling for Improving Datacenter Economy. In *Proceedings of the 46th Annual International Symposium on Computer Architecture*.

[20] Patrick Kennedy. 2018. Baidu X-MAN Liquid Cooled 8-Way NVIDIA Tesla V100 Shelf. Retrieved October 15, 2024 from https://www.servethehome.com/baidu-x-man-liquid-cooled-8-way-nvidia-tesla-v100-shelf/

[21] Mathew Oldham Kevin Lee, Adi Gangidi. 2024. Building Meta's GenAI Infrastructure. Retrieved October 15, 2024 from https://engineering.fb.com/2024/03/12/data-center-engineering/building-metas-genai-infrastructure/

[22] Kubernetes. [n. d.]. Production-Grade Container Orchestration. Retrieved October 15, 2024 from https://kubernetes.io/

[23] Kubernetes. [n. d.]. Scheduler Configuration. Retrieved October 15, 2024 from https://kubernetes.io/docs/reference/scheduling/config/

[24] Baolin Li, Viiay Gadepally, Siddharth Samsi, and Devesh Tiwari. 2022. Characterizing Multi-Instance GPU for Machine Learning Workloads. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 724–731.

[25] Baolin Li, Tirthak Patel, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2022. MISO: Exploiting Multi-Instance GPU Capability on Multi-Tenant GPU Clusters. In *Proceedings of the 13th Symposium on Cloud Computing*. 173–189.

[26] Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2023. Clover: Toward Sustainable AI with Carbon-Aware Machine Learning Inference Service. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[27] Greg Linden. [n. d.]. Make Data Useful. Retrieved October 15, 2024 from https://www.scribd.com/doc/4970486/

[28] Chetan Tekur Maggie Zhang, James Sohn. 2020. Getting the Most Out of the NVIDIA A100 GPU with Multi-Instance GPU. Retrieved October 15, 2024 from https://developer.nvidia.com/blog/getting-the-most-out-of-the-a100-gpu-with-multi-instance-gpu/

[29] Ioannis Manousakis, Íñigo Goiri, Sriram Sankar, Thu D. Nguyen, and Ricardo Bianchini. 2015. CoolProvision: Underprovisioning Datacenter Cooling. In *Proceedings of the 6th ACM Symposium on Cloud Computing*.

[30] Justin D Moore, Jeffrey S Chase, Parthasarathy Ranganathan, and Ratnesh K Sharma. 2005. Making Scheduling "Cool": Temperature-Aware Workload Placement in Data Centers. In *USENIX Annual Technical Conference, General Track*. 61–75.

[31] NVIDIA. [n. d.]. NVIDIA Multi-Instance GPU. Retrieved October 15, 2024 from https://www.nvidia.com/en-us/technologies/multi-instance-gpu/

[32] NVIDIA. 2022. NVIDIA Teams With Microsoft to Build Massive Cloud AI Computer. Retrieved October 15, 2024 from https://nvidianews.nvidia.com/news/nvidia-microsoft-accelerate-cloud-enterprise-ai

[33] NVIDIA. 2024. A100 MIG Profiles. Retrieved October 15, 2024 from https://docs.nvidia.com/datacenter/tesla/mig-user-guide/#a100-mig-profiles

[34] NVIDIA. 2024. About the NVIDIA GPU Operator. Retrieved October 15, 2024 from https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/index.html

[35] NVIDIA. 2024. Multi-Process Service. Retrieved October 15, 2024 from https://docs.nvidia.com/deploy/mps/index.html

[36] NVIDIA. 2024. NVIDIA DCGM. Retrieved October 15, 2024 from https://developer.nvidia.com/dcgm

[37] NVIDIA. 2024. NVIDIA GB200 NVL72. Retrieved October 15, 2024 from https://www.nvidia.com/en-us/data-center/gb200-nvl72/

[38] Qiangyu Pei. 2024. Supplementary file for InferCool. Retrieved October 15, 2024 from https://qiangyupei.github.io/files/SoCC24_InferCool_supplementary.pdf

[39] Qiangyu Pei, Shutong Chen, Qixia Zhang, Xinhui Zhu, Fangming Liu, Ziyang Jia, Yishuo Wang, and Yongjie Yuan. 2022. CoolEdge: Hotspot-Relievable Warm Water Cooling for Energy-Efficient Edge Datacenters. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 814–829.

[40] Qiangyu Pei, Yongjie Yuan, Haichuan Hu, Qiong Chen, and Fangming Liu. 2023. AsyFunc: A High-Performance and Resource-Efficient Serverless Inference System via Asymmetric Functions. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*. 324–340.

[41] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI blog* 1, 8 (2019), 9.

[42] Yongyi Ran, Han Hu, Xin Zhou, and Yonggang Wen. 2019. DeepEE: Joint Optimization of Job Scheduling and Cooling Control for Data Center Energy Efficiency Using Deep Reinforcement Learning. In *Proceedings of the 39th International Conference on Distributed Computing Systems*.

[43] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-Less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 397–411.

[44] Carol Ryan. 2024. Energy-Guzzling AI Is Also the Future of Energy Savings. Retrieved October 15, 2024 from https://www.wsj.com/business/energy-oil/ai-data-centers-energy-savings-d602296e

[45] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: A GPU Cluster Engine for Accelerating DNN-Based Video Analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 322–337.

[46] Matt Skach, Manish Arora, Chang-Hong Hsu, Qi Li, Dean Tullsen, Lingjia Tang, and Jason Mars. 2015. Thermal Time Shifting: Leveraging Phase Change Materials to Reduce Cooling Costs in Warehouse-Scale Computers. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*.

[47] Matt Skach, Manish Arora, Dean Tullsen, Lingjia Tang, and Jason Mars. 2018. 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*.

[48] Vertiv. [n. d.]. Understanding the Cost of Data Center Downtime: An Analysis of the Financial Impact on Infrastructure Vulnerability. Retrieved October 15, 2024 from https://www.vertiv.com/4a3537/globalassets/images/about-images/news-and-insights/articles/white-papers/understanding-the-cost-of-data-center/datacenter-downtime-wp-en-na-sl-24661_51225_1.pdf

[49] Yanan Yang, Laiping Zhao, Yiming Li, Huanyu Zhang, Jie Li, Mingyang Zhao, Xingzhen Chen, and Keqiu Li. 2022. INFless: A Native Serverless System for Low-Latency, High-Throughput Inference. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 768–781.

[50] Sungkap Yeo, Mohammad M Hossain, Jen-Cheng Huang, and Hsien-Hsin S Lee. 2014. ATAC: Ambient Temperature-Aware Capping for Power Efficient Datacenters. In *Proceedings of the 5th ACM Symposium on Cloud Computing*. 1–14.

[51] Peifeng Yu and Mosharaf Chowdhury. 2020. Salus: Fine-Grained GPU Sharing Primitives for Deep Learning Applications. *MLSys' 20* (2020).

[52] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving. In *2019 USENIX Annual Technical Conference*. 1049–1062.

[53] Wei Zhang, Quan Chen, Kaihua Fu, Ningxin Zheng, Zhiyi Huang, Jingwen Leng, and Minyi Guo. 2022. Astraea: Towards QoS-Aware and Resource-Efficient Multi-stage GPU Services. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 570–582.