SMORE: Enhancing GPU Utilization in Deep Learning Clusters by Serverless-based Co-location Scheduling

Junhan Liu*, Zinuo Cai*, Yumou Liu, Hao Li, Zongpu Zhang, Ruhui Ma†, *Member, IEEE*, Rajkumar Buyya, *Fellow, IEEE*

***Abstract*—Deep learning (DL) clusters allow machine learning practitioners to submit their computation-intensive tasks, with GPUs accelerating their execution process. However, GPUs in current deep learning clusters are often under-utilized, which hampers the job performance and overall cluster throughput. It is urgent to improve GPU utilization, but existing works lack research on fine-grained allocation for GPU resources, as it typically allocates GPUs as indivisible units. Serverless computing reveals an opportunity to optimize utilization with fine-grained resource allocation methods, but it requires addressing three main challenges: co-location performance degradation, service level objectives guarantee of serverless functions, and cold start overhead. We propose SMORE, a framework based on serverless computing to optimize GPU resource utilization of DL clusters. SMORE dynamically predicts the possible co-location performance degradation and leverages a degradation-aware scheduling algorithm to ensure that the co-location decisions do not impact workload performance. It also dynamically preloads or offloads DL models by predicting the request numbers of the subsequent period to address the cold start issue. Through actual trace testing on the prototype of SMORE, we find that the average GPU utilization can be increased by 34% with degradation being controlled effectively.**

***Index Terms*—serverless computing, GPU cluster, workload co-location, cold start**

I. INTRODUCTION

Deep learning (DL) has deeply involved in daily applications in recent years [1] and achieved significant performance breakthroughs in several real-world application domains, such as natural language processing [2], speech recognition [3], and recommendation systems [4]. To meet the computational requirements of deep learning models, large research organizations and enterprises set up GPU clusters to deploy deep learning workloads [5], [6]. GPU clusters promote the effective processing of large-scale data sets and the training of complex models by significantly improving computing efficiency. Scaling up GPU cluster deployment is an inevitable trend in the future development of deep learning. As clusters grow larger, service providers encounter escalating costs.

Junhan Liu, Zinuo Cai, Yumou Liu, Hao Li, Zongpu Zhang, Ruhui Ma are with the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China (e-mail: {liujunhan, kingczn1314, liiyumou, lihao-1011, zhangz-z-p, ruhuima}@sjtu.edu.cn)

Rajkumar Buyya is with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Australia (e-mail: rbuyya@unimelb.edu.au)

[1]Both authors contributed equally to this work.

[2]Corresponding author is Ruhui Ma (ruhuima@sjtu.edu.cn).

However, existing multi-tenant DL clusters suffer from low GPU utilization, which hampers overall system performance and diminishes the throughput of deep learning models [7]. Based on existing works [5], [6], [8] and open-source GPU traces [9], [10], we attribute the low GPU utilization in DL clusters to two aspects, *the allocation methodology of GPU resources* and *the resource utilization characteristics of distributed DL tasks*. From the perspective of the GPU allocation method, most deep learning tasks fail to fully saturate the GPU Streaming Multiprocessors (SM), and approximately 90% of the GPUs exhibit GPU utilization rates below 80% [8]. In production clusters, GPUs do not natively support sharing and are typically treated as indivisible units [6], [11], leading to under-utilization and wasted computational capacity. Although current technologies offer partitionable GPU instances, like MIG, the granularity of this partitioning is inherently coarse. In addition, from the perspective of resource utilization characteristics of distributed DL tasks, communication and synchronization overhead inherent in distributed training can create bottlenecks in GPU usage. Typically, around 90% of the time is wasted waiting on the network during cloud-based training [12]. Moreover, certain inefficient task-scheduling strategies such as gang scheduling [6] also contribute to GPU under-utilization. Thus, inefficient GPU allocation methods and distributed DL task training schemes lead to low utilization of cluster resources.

Existing works have made efforts to address the problem of GPU under-utilization. Some works are from a task-oriented perspective. Gandiva [13] proposes time-slicing, migration, and packing methods to allow GPU sharing. To address the fairness problem of time partitioning for workloads of multiple scales and sizes, Gandiva$_{fair}$ [14] proposes an automated trading mechanism. AntMan [8] introduces opportunistic DL workloads as low-priority tasks to maximize the utilization of GPU cycles. Others are from a cluster-oriented perspective. AlloX [15] formulates the scheduling problem as a minimum-cost two-point matching problem to schedule DL workloads among exchangeable resources. However, all of these works either consider GPUs as indivisible units [14], [15] or only support certain types of workloads [8], [13], such as training or inference workloads.

We recognize that serverless computing has advantages in fine-grained resource allocation and are the first to propose its adoption in optimizing GPU resource utilization in DL clusters. Serverless computing is a popular cloud computing

paradigm where developers focus on the application's logic, leaving server configuration, maintenance, and scalability to the provider [16]. It allows users to submit tasks, known as functions, which are executed by the cloud provider on its servers [17]. It can dynamically allocate resources on demand and scale up or down to meet workload requirements. Given the diverse properties of serverless computing and the multitasking capabilities offered by GPUs, deploying serverless function services on under-utilized DL clusters provides an opportunity to improve resource utilization.

Existing works [9], [18]–[20] have attempted to harvest CPU and memory resources with serverless computing. However, these methods fail to harvest GPU resources. The gap arises from immature isolation mechanisms for GPUs. We conduct experiments and identify the challenges of optimizing GPU cluster resource utilization by co-locating serverless and serverful workloads. The **first** challenge is that different workloads compete for GPU and other computing resources, leading to workload performance degradation. Our experiments in §II find that unreasonably co-locating serverless functions and serverful workloads results in over 10% performance degradation. As the number of co-located workloads increases, performance degradation will rapidly escalate. The **second** challenge is meeting the Service Level Objectives (SLOs) of serverless functions under workload co-location. Serverless functions are subject to finite deadlines, and performance degradation may result in their inability to complete execution before their deadlines. The **third** challenge comes from cold start issues, and cold start times exceed the function execution time, significantly increasing the overall latency of serverless functions. Our experiments find that some tasks exhibit cold start overheads nearly 100 times greater than their warm start overheads.

Therefore, we propose SMORE to optimize GPU resource utilization of DL clusters by allowing serverless functions to utilize idle GPU resources. SMORE is designed with the objective of enabling the selective co-location of appropriate serverless function workloads within an ongoing serverful workload DL cluster to improve GPU resource utilization, while also striving to meet SLO requirements and minimize performance impact. It can make admission decisions for incoming serverless function requests on clusters with serverful tasks, predicting their potential degradation to control workload degradation. Besides, it can effectively address cold start issues, balancing cold start rates and resource wastage time.

To be specific, we first construct a performance degradation predictor to better quantify the performance degradation caused by co-location. The predictor comprises pair-wise and multi-way sub-predictors, divided into offline training and online update phases. In the offline phase, we effectively gather training samples. After training and evaluating multiple distinct regression models, the most accurate regression model is selected as the core model of the pair-wise sub-predictor. The multi-way sub-predictor integrates the results of the pair-wise predictor to handle scenarios where multiple functions are co-located. In the online phase, we track the real execution results and update the dataset to retrain the predictor.

Secondly, we design a degradation-aware scheduler to allocate resources and execute serverless functions with the goal of managing performance degradation and maximizing the number of function executions meeting their SLOs. The scheduler assesses incoming serverless function requests based on the predicted performance degradation and the current workload status obtained from real-time monitoring of clusters. If this function can be executed, the scheduler will determine the most suitable GPU for its execution, aiming to increase GPU utilization while managing performance degradation.

Furthermore, we present a prewarmer module that dynamically preloads the DL model to alleviate cold starts and offloads the DL model to reduce resource wastage in idle containers. Its core is a Long-Short LSTM (LS-LSTM) model to predict the number of requests in the next period, effectively capturing the patterns of requests in both the long term and short term. By observing the patterns of long and short cycles of past request arrivals, the prewarmer can effectively predict the patterns of incoming requests. Based on the prediction results, the prewarmer can preload functions in advance to reduce the cold start overhead or terminate function execution to reduce resource waste.

We implement a prototype of SMORE in 3000+ LOC of Python. We conduct experiments on our self-built GPU cluster relying on the function traces provided by Azure [9]. We analyze the performance loss, GPU utilization, and function request reception. The results indicate that our system can increase GPU utilization by up to 34% while controlling degradation within a certain range.

Our contribution can be summarized as follows.

- We propose SMORE, the first serverless-based framework to optimize the GPU utilization of DL clusters.
- We construct a predictor for performance degradation due to workload co-location, and it can achieve high-accuracy results with a small amount of training data.
- We design a dynamically-aware scheduling algorithm to improve resource utilization and control the degree of performance degradation.
- We present a novel Long-Short LSTM to predict the situation of upcoming function requests and control the start of function instances ahead of time.
- We implement a system prototype of SMORE. Experiments on real clusters show that our approach improves the GPU utilization of clusters by up to 34%.

II. WORKLOAD CO-LOCATION CHARACTERIZATION

Co-location testing, where multiple workloads share the same hardware resources, is a critical aspect of resource management and performance analysis. In CPU-based systems, resource isolation is a mature field, with established techniques like virtualization and containerization providing relatively strong isolation between co-located workloads. However, GPU colocation presents inherent challenges regarding isolation. Despite the introduction of technologies like NVIDIA Multi-Process Service (MPS) and Multi-Instance GPU (MIG) aimed at improving GPU sharing and isolation, the level of isolation achieved is generally weaker compared to CPUs.

TABLE I: Typical deep learning workloads.

Model	Type	Dataset
VGG-16 [26]	CV	CIFAR100 [27]
MobileNet [28]	CV	CIFAR100
DeepViT [29]	CV	ImageNet [30]
ResNet-50 [29]	CV	ImageNet
BERT [31]	NLP	CSL [32]
RoBERTa	NLP	SQuAD [33]
DeepFM [34]	Ad.	Criteo
SegNet (MAN) [35]	MTL	NYUv2 [36]

In this section, we study the performance impact of co-locating DL training workloads with DL inference functions. To enhance our comprehension of co-location impacts and guide future design decisions, this study primarily concentrates on the influence of potential resource interference on GPU utilization and workload performance. While previous work has studied characteristics of individual GPU workloads, it lacks profiling for the co-location of different types of workloads. We divide existing works into two types. One is profiling for single-type workloads, which only considers training [8], [21] or inference workloads [22], [23]. The other focuses solely on co-locating single-type workloads, either for training tasks [24] or inference tasks [25]. Compared to existing work, our characterization includes co-location profiling of these two kinds of workloads, making it more comprehensive.

A. Characterization Setup

a) Methodology: Our characterization is divided into the following three steps. We first execute each workload exclusively and then select three training workloads representing distinct utilization levels (low, moderate, and high) to comprehensively analyze co-location effects. Finally, we co-locate different inference functions with these training workloads to observe their impacts on workload performance. We collect relevant data during workload execution for further analysis, including inference function latency, average training workload consumption time, and GPU utilization.

b) Models: For better reproducibility and reliability, we select five common deep learning models with open datasets, as shown in TABLE I. To improve the applicability of the analyzed results, these models cover three common research fields, including computer vision (CV), natural language processing (NLP), advertisement (Ad.), and multi-task learning (MTL).

c) Metrics: We select both system and workload metrics as the basis for our profiling. System metrics include resource utilization, primarily GPU SM utilization. Workload metrics refer to performance indicators related to the workload. We choose the 99th percentile ($P99$) latency for the inference functions and the average time an epoch consumes as a benchmark for training workloads. The performance degradation of both training and inference workloads can be computed from their exclusive and co-location execution.

$$\delta = \frac{P_{colo} - P_{solo}}{P_{solo}} \tag{1}$$

In Equation 1, δ denotes the performance degradation of one workload, while P_{colo} and P_{solo} represent the performance of the workloads in the co-location case and the solo-run case, respectively.

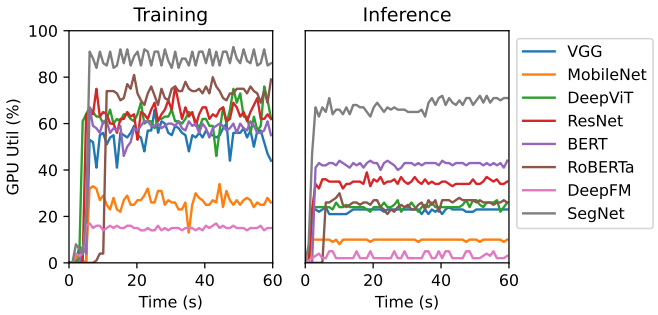


Fig. 1: GPU utilization of different models under solo-run.

d) Environment: All evaluations are implemented and deployed on Ubuntu 20.04 with Nvidia RTX 3090 GPUs. To improve the reliability of experimental results, we run each co-location and solo-run experiment for one minute. Both workloads run directly on bare metal. After that, we record the average $P99$ latency degradation for inference functions and the average execution time degradation for training workloads.

B. Observations

TestCase1: Exclusive execution. To understand the utilization patterns of different workloads, we first examine the GPU SM usage of each model exclusively running on a single GPU, as depicted in Fig. 1. DeepViT, ResNet-50, BERT, and RoBERTa exhibit high computational intensity during training, reaching peak SM usage of over 70%. GPU utilization of DeepFM and MobileNet is relatively low, mostly below 30%. For inference tasks, BERT maintains around 70% SM utilization, while other models are mostly below 40%. The GPU usage patterns of various workloads fluctuate over time, but the fluctuations are generally within 20%.

Observation I

> *DL models exhibit diverse resource requirements, yet SM utilization of the same model remains within a specific range during execution.*

TestCase2: Co-located execution. We employ the $P99$ latency as our performance metric for inference functions and average epoch consumption time for training applications. We select DeepFM (low), VGG (moderate), and RoBERTa (high) to represent distinct utilization levels and investigate the impact of co-location on their performance. We visualize the co-location degradation level in Fig. 2. x-axis label indicates the different inference functions that are co-located with these training applications. Co-locating with these eight inference functions does not result in a significant slowdown (0.4%-18.4%) on average epoch time for the DeepFM training application. Inference workloads such as MobileNet, DeepFM, RoBERTa, and DeepViT a minimal impact (less than 10%) on the training workloads of VGG and RoBERTa. The increase in P99 latency for the co-located functions in approximately half of the cases is also within an acceptable range. Therefore,

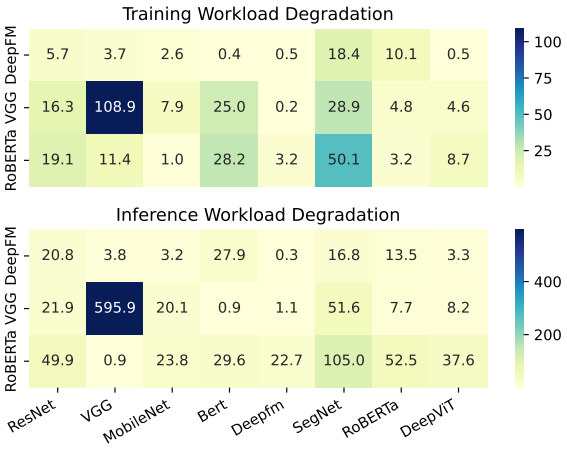


Fig. 2: Performance degradation caused by co-location.

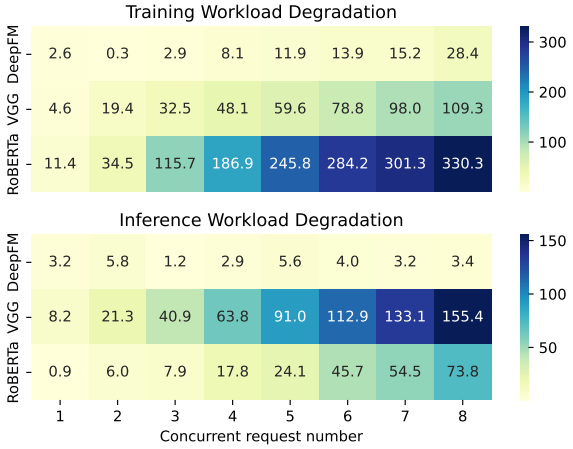


Fig. 3: Co-location evaluation with different request numbers.

TABLE II: Cold start latency of different inference functions.

Model	Warm-start (ms)	Cold-start (s)
VGG-16	3	1.2
MobileNet	9	1.0
DeepViT	11	2.4
ResNet-50	13	1.5
BERT	37	2.6
RoBERTa	10	5.2
DeepFM	8	0.6
SegNet (MAN)	50	1.6

this presents opportunities for co-locating different workloads without significant performance degradation.

Observation II

The extent of performance degradation varies with different inference function co-locations.

TestCase3: Variable request numbers To further investigate co-location, we select DeepFM/MobileNet, VGG/DeepViT, and RoBERTa/VGG combinations for training and inference workloads. We then observe the impact of increasing inference requests on the performance degradation of both workloads. As illustrated in Fig. 3, different combinations exhibit different levels of performance degradation. For the DeepFM/MobileNet combination, training degradation increases steadily, with a surge from 7 to 8 requests, and inference degradation remains acceptably low, staying below 10%. For both the VGG/DeepViT and RoBERTa/VGG combinations, degradation in training and inference workloads increases with the number of concurrent requests. Furthermore, in some instances, the performance degradation exceeds 100%, which is unacceptable in practical scheduling. The upward trend observed in these data suggests the potential for modeling degradation growth using various regression techniques, which can provide a basis for scheduling.

Observation III

There exists the possibility of predicting the performance impact of co-locating multiple inference functions.

TestCase4: Cold start time. To understand the cold start overhead that GPU inference functions incur, we evaluate the initialization time required for several functions from scratch. This includes GPU context initialization, PyTorch framework initialization, and model loading, constituting the cold start time. Additionally, we assess the execution time of these functions after initialization, representing the warm start time. As shown in TABLE II, the cold start time for RoBERTa exceeds 5 seconds, while BERT and DeepViT also exhibit cold start times surpassing 2 seconds. However, the typical execution time for these functions is less than 100 milliseconds. Consequently, the initialization overhead associated with cold starts is substantial. Notably, even though DeepFM has a relatively short cold start time of 0.6 seconds, this is still nearly two orders of magnitude greater than its inference time.

Observation IV

Cold starts significantly increase the execution latency of functions, potentially preventing serverless functions from completing their execution before their deadlines.

C. Implications

To better design a serverless-based GPU cluster utilization optimization scheme, we obtain some inspirations based on the above characterization experiments. **Observation I** and **Observation II** imply our scheme should select reasonable workload co-location combinations to control performance degradation within a certain range. **Observation III** shows when considering the co-location of multiple inference functions, it is possible to estimate their degradation. **Observation IV** indicates our scheme needs to address cold start issues as much as possible to reduce function execution latency. In summary, we progress our design by addressing the following challenges: (1) performance degradation caused by co-location should be controlled; (2) SLOs for serverless functions should be met; (3) considerable overhead from cold starts should be avoided.

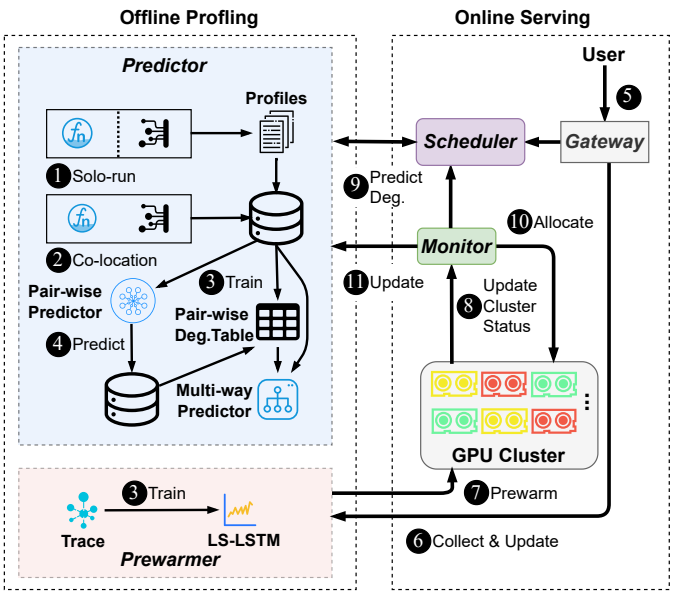


Fig. 4: The overview and workflow of SMORE.

III. DESIGN

Serverless-based optimization requires a clear distinction between serverful and serverless workloads. Because inference functions have short execution times and moderate resource usage, we deploy them as serverless functions, ensuring they meet deadlines and SLO requirements. Conversely, due to the lengthy execution times of training workloads, we classify them as serverful workloads.

Based on this setup, we propose SMORE, a serverless-based framework to optimize the GPU utilization of DL clusters. In response to the first challenge of potential performance degradation under workload co-location, we propose a performance degradation predictor to predict performance degradation levels of co-locating workloads, which can help the scheduler make admission control decisions involving accepting or rejecting submitted function requests. To address the second challenge of meeting SLOs of more serverless functions, we then design a degradation-aware scheduler with hybrid scheduling policies to handle low and high load conditions. The scheduler dynamically selects and admits serverless functions to co-locate with serverful DL training workloads while guaranteeing SLOs of admitted serverless functions and managing performance degradation within an acceptable range. Finally, to tackle the third challenge of cold starts, we apply a prewarmer with LS-LSTM to predict the number of requests in the next period and make decisions regarding preloading or offloading to alleviate cold start issues and reduce resource wastage.

Fig. 4 shows the overall workflow of SMORE. SMORE comprises two phases, an offline profiling phase and an online serving phase. In the offline phase, we collect profiles containing workload performance metrics under exclusive execution condition❶. Then, we randomly select some co-location conditions❷ to generate samples to train❸ an initial model for pair-wise co-location. After obtaining the initial model

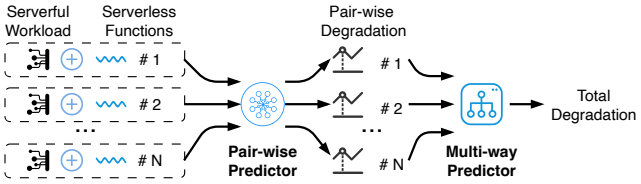


Fig. 5: The workflow of the predictor. The predictor will first predict the degradation generated by pair-wise co-location and then predict the multi-way co-location based on the previous results.

of performance degradation, we predict❹ the performance degradation generated by the remaining co-location combinations to generate a complete degradation table for later quick querying. We also utilize actual traces to train❸ the initial LS-LSTM in the prewarmer module. In the online phase, the user sends❺ a request to the scheduler, and the scheduler judges if the request can be admitted and determines which GPU to co-locate it with. The prewarmer module collects actual requests and periodically updates❻ the LS-LSTM with a certain interval. The prewarmer predicts incoming requests for the next period and preloads models in advance. The scheduler gathers resource usage and task execution status through the monitor and periodically updates the cluster status❽ at fixed time intervals. The scheduler queries the predictor module to obtain the predicted performance degradation result of co-location❾, then performs scheduling and allocates❿ GPU resources to admitted functions based on the obtained information. After the functions are executed in the cluster, the monitor would track the real performance degradation and update the dataset⓫ to retrain the predictor.

IV. CO-LOCATION DEGRADATION PREDICTION

A. Design Overview

We propose a prediction module for co-location performance degradation. The core of the predictor module includes a pair-wise predictor and a multi-way predictor for co-location performance degradation. We combine offline training and online updating. After offline training, we can obtain an initial model developed using a limited dataset. Since the initial model may not be accurate enough, we employ incremental learning to gather online data and continuously enhance the model for improved accuracy during the online phase.

The workflow of the predictor module is shown in Fig. 5. Assume we want to predict the possible performance degradation of one serverful workload co-located with N serverless functions. We combine N serverless functions one by one with the serverful workload to form inputs and obtain N pair-wise co-location performance degradation through the pair-wise predictor. Next, we input these N pair-wise co-location prediction results into the multi-way predictor and get the final output, representing performance degradation of workload caused by co-location of N functions.

B. Pair-wise Co-location Profiling

We first consider performance degradation in pair-wise co-location, where one serverful workload is co-located with

TABLE III: Selected DL model features.

Features	Description
FLOPS	Number of floating-point operations
Params	Number of Parameters
Memory	GPU Memory used by model
Activations	Number of activation functions
Num. Conv	Number of convolution layers
Num. Linear	Number of linear layers
Batch Size	Size of mini-batch
Num. Norm	Number of Normalization layers
Num. ReLU	Number of ReLU layers
Num. Embed	Number of Embedding layers
Num. Pool	Number of Pooling layers
Num. Drop	Number of Dropout layers

one serverless function. While the DL training workload is performed on a GPU, we try to locate different serverless DL inference functions on the same GPU and monitor performance degradation. Since the cold start time of serverless GPU function instances is relatively long, we temporarily disregard its impact on workload co-location. In §VI, we extensively consider the impact of reducing cold start on function execution efficiency.

We consider representative feature selection at both the system and task levels. At the system level, we choose GPU SM utilization and GPU memory utilization as metrics because they best reflect the utilization of GPU resources by the workload. At the task level, we choose the features as shown in TABLE III because they effectively and comprehensively represent the characteristics of a DL model structure. Although FLOPS and Memory can be computed based on the data shape of each layer and other parameters, diverse hardware platforms and libraries might exhibit variations in the calculation methodology of FLOPS for certain operations. It is straightforward to execute one iteration to obtain accurate FLOPS with corresponding tools and GPU memory usage with GPU monitoring tools. The profiling segment can use this method to extract features from different DL models shown in TABLE I. Then, we utilize these features as inputs and the level of performance degradation as the target to construct a prediction model for workload pair-wise co-locations. This is equivalent to a regression problem with an input vector dimension of 24, as the feature dimensions for serverful and serverless workloads are both 12, summing up to 24. We employ classical machine learning models in our study, and among them, the Random Forest model demonstrates the best performance through comparative analysis.

Model evaluation. We evaluate model accuracy using Root Mean Square Log Error (RMSLE). RMSLE is a metric used to assess the performance of regression models and is often used to assess situations where there is a large bias in the prediction results. RMSLE can be seen as an extension of Mean Squared Error (MSE) and can help minimize overfitting. In Equation 2, $\hat{y}_i$ denotes the model predicted value, and y_i denotes the true value.

$$\text{RMSLE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(log(y_i+1) - log(\hat{y}_i+1))^2} \quad (2)$$

C. Multi-way Co-location Prediction

Multi-way co-location represents the situation where one serverful workload is co-located with multiple serverless functions. Multi-way co-location sample space can be much larger than pair-wise co-location, and naive sampling is unfeasible due to the high cost. Multi-way co-location can be divided into multiple pair-wise co-locations. Specifically, co-locating one training workload with N serverless functions can be viewed as a superposition of N sets of training workloads and serverless functions executing one by one, producing performance degradation somehow. Therefore, we design a linear weighted sum of N co-location degradation to predict the performance of multiple workloads co-location. Mathematically, our model has the form for training workload degradation under the co-location of k workloads.

$$Deg_i = \sum_{j \in SL_i} \gamma^k(M_i, M_j) Deg_{ij} \quad (3)$$

In Equation 3, SL_i is the set of co-located serverless functions with serverful workload i, $\gamma^k(M_i, M_j)$ is the weight for the training workload i and serverless function j which is only related to the model name M_i and M_j shown in TABLE I, Deg_{ij} is the degree of performance degradation when workload i is co-located with j only. The degree of degradation of the serverless functions in the multi-way co-location scenario is similar as shown in Equation 4, where α and β represent weights and Deg_{ji} means degradation for the serverless function j under pair-wise co-location with the training workload i.

$$Deg_j = \alpha^k(M_j, M_i) Deg_{ji} + \sum_{w \in SL_i} \beta^k(M_j, M_m) Deg_{jm} \quad (4)$$

The higher the weighting, the greater the performance impact generated by the workload.

Updating the weights of the multi-way predictor. In the beginning, we initialize all weight values to one, and the predictor is equivalent to a linear sum of performance impacts from other workloads. After the system goes live, we record actual execution results and use the error between the actual values and the predicted values, multiplied by a coefficient, to update the weights.

V. Degradation-aware Scheduling

The predictor module provides the foundation for later scheduling, effectively forecasting the performance degradation that occurs during the co-location of serverless functions and serverful workloads. To guarantee SLOs of admitted serverless functions when co-locating with serverful workloads, in this section, we introduce a degradation-aware scheduling strategy. The core target of our degradation-aware scheduling is improving GPU cluster resource utilization by admitting more functions to meet their SLOs and managing performance degradation within an acceptable range.

A. Degradation-aware Function Scheduling

To manage the performance degradation of workloads due to co-location, we use degradation-aware function scheduling

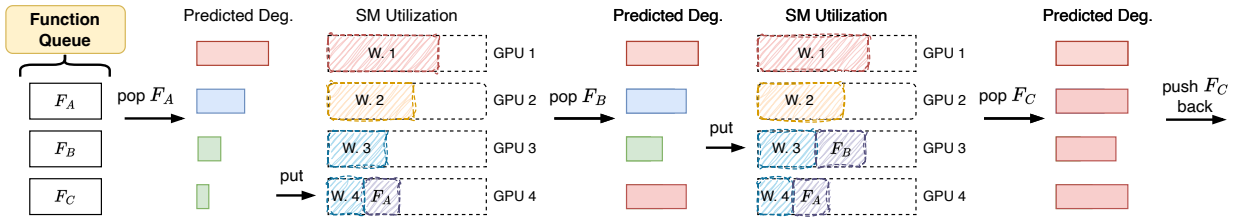


Fig. 6: The example of scheduling. When three functions pop out of the Function Queue in sequence, the system will first predict their degradation and determine whether to execute or return to the queue.

to select the proper placement for admitted serverless functions. We define performance loss boundary judgment as P99 $latency \leq 10\%$ [37].

Priority queue. It is crucial to determine which function to prioritize for resource allocation and scheduling. From the offline profiling results and online execution history, we gather data regarding the co-location scenario for a function and compute its average values to depict a potential increase in GPU usage and performance degradation. Then, we perform a prioritization calculation shown in Equation 5 to reorder the arriving requests.

$$P_{func} = \frac{\text{Potential increase in usage}}{\text{Potential increase in degradation}} \tag{5}$$

After that, we take out the highest priority requests from the queue in order to further determine the most suitable GPUs. In general, requests for functions with a high potential increase in GPU usage and a low potential increase generate more utilization improvements, and we prioritize the satisfaction of these requests.

Hybrid scheduling policies. To deal with high and low loads for request arrival patterns, we will prioritize implementing degradation minimization at low loads and finding a feasible placement position at high loads. Because at low loads, the upper bound on degradation is basically never reached, and there is enough time to find an optimal position for each request. Under high load conditions with numerous requests, our primary focus should be on finding suitable placement positions for a greater number of requests. The complexity of processing all GPUs in the cluster at once is $O(N)$, where N is the number of GPUs in the DL cluster. To reduce the search overhead of finding the most suitable GPUs under high load conditions, we start by randomly selecting d GPUs from a resource pool of N GPUs. This approach can reduce scheduling overhead and enable the processing of a higher volume of functions within a timestamp. In the following part, we present the scheduling algorithm for the low load case as an example, which is similar to the high load case.

Scheduling algorithm. SMORE's GPU resource management logic, which runs when the function request queue is not empty and accepts the highest-priority function from the queue, is shown in Algorithm 1. We randomly select the d most likely GPUs from the GPU resource pool. All memory and computing kernels on a GPU not allocated to DL training workloads are reserved for serverless and referred to as exclusive GPU resources. We need to ensure that the exclusive GPU SM and memory resources are enough for serverless inference

Algorithm 1: Serverless Function Resource Allocation

Input: **Req**: submitted serverless function request;
P: GPU resource pool; **f**: fast processing
Output: Target GPU g_{target}

```
Function ResourceAllocation(req, p, s, f):
    gpu_set ← RandomGPUSelection(p, d)
    degs ← {}
    for g in gpu_set do
        if not GPUResourceSatisfy(g, req) then
            continue
        end
        deg ← ComputeDegradation(g, req)
        if DegradationAcceptable(g, deg) then
            if f then
                return g
            end
            degs.add({g, deg})
        end
    end
    if degs is not empty then
        g_target ← LeastDegradation(degs)
        return g_target
    else
        return null
    end
End Function
```

requests. For a specific workload, its GPU memory footprint is fixed and easily extrapolated, while SM utilization rises in the co-location case compared to the exclusive running case. When SM utilization approaches 100%, continuing to co-locate workloads incurs very large delays. If the remaining resources of a GPU can satisfy the function's demand, we then utilize a predictor to get the performance impact generated by co-locating the function. After processing all the selected GPUs, if none of the GPUs meets the demand for the function, then it is returned to the function queue, awaiting the next round of scheduling. If more than one GPU meets the demand, then the GPU with the least performance impact is selected for function placement.

Fig. 6 shows an example of how to make placement decisions. When functions arrive, we put them into a pending queue and compute their priority. Assuming F_A is the highest priority, we take it out of the queue for processing. From the previous profiling, we can extract certain relevant features

of the function model and use the predictor to forecast the potential degradation that might occur when co-locating with existing workloads of 4 GPUs to be selected. Since GPU 4 generates the least performance impact, F_A will be allocated to GPU 4. Next, we pop out F_B and calculate the degradation that may occur when placing it on four GPUs. Ultimately, we choose GPU 3 to place F_B because it produces the least degradation. Finally, we pop out F_C and find that the degradation it produces exceeds the limit. Therefore, we put it back into the queue and await the next scheduling round.

VI. Prewarmer

A. Managing Cold Starts with Hybrid LSTM

For serverless DL inference functions, cold start includes the time cost of resource allocation and initialization, loading large libraries and frameworks (such as TensorFlow or PyTorch), GPU context initialization, and model loading. Cold starts incur considerable startup overhead and are even much higher than the actual execution time. It is urgent to avoid cold starts to guarantee SLOs for serverless functions.

To alleviate the considerable resource waste and handle the sudden spike condition, we propose a Long-Short LSTM policy to determine when to prewarm the function runtime and when to terminate the function runtime. We track both the long-term and short-term request number per minute and idle time. LSTM can simultaneously consider the long-term and short-term characteristics of data to predict future events and is suitable for the long-term regularity and short-term fluctuations of serverless functions.

Input features of LSTM. We choose the number of requests per minute in the previous n minutes as the input, which is an n-dimensional vector. The LSTM will output the number of requests that will arrive in the next minute. Based on the past trace data, we train the long-term and short-term LSTM models to predict the request number of the next time period. To improve the prediction accuracy, we retrain the model with the newly gained data in a fixed time interval. After getting the predicted request number for the next period, we use the request number per second to prewarm corresponding containers to provide services for more than one concurrent request.

The main purpose of solving the cold start problem is to reduce the cold start rate and resource waste rate. The lower the value of both metrics, the better the optimization effect. The time for resource loading but function not executing is a waste of resource time denoted as T_{waste} and the total time is denoted as T_{total}. We derive the resource waste rate R_{rw} as

$$R_{rw} = \frac{T_{waste}}{T_{total}} \tag{6}$$

The number of functions experiencing cold start is denoted as N_{cs} and the total number of functions is denoted as N_{total}. We derive the cold start rate R_{cs} as

$$R_{cs} = \frac{N_{cs}}{N_{total}} \tag{7}$$

B. Workflow of Prewarmer

Data preprocessing. Our prewarmer module tracks the arrival time, execution time, and end time of functions. Then it calculates the number of function arrivals per minute as input data for LSTM model training.

Long and Short term setting. To accurately capture request arrival patterns, we use the last short (e.g., 1 hour) and long durations (e.g., 1 day), respectively for two LSTM model training. After training two LSTM models, the models will output L_{rqnum} and S_{rqnum} based on the request conditions of the previous n minutes. Here, $rqnum_L$ and $rqnum_S$ respectively denote the prediction results of the long-term model and the short-term model for the number of requests expected to arrive in the next minute. We derive the request number of next minute $rqnum$ using their weighted sum.

$$rqnum = \alpha \cdot rqnum_L + (1 - \alpha) \cdot rqnum_S \tag{8}$$

Where α is a configurable weight between 0 and 1; by default, we set $\alpha = 0.5$.

Control prewarming. At the beginning of a minute, we can get $rqnum$, the number of incoming requests for this minute, through LS-LSTM as depicted in Equation 8. If $rqnum > 0$, we will preload the DL model in advance, ensuring incoming requests can be serviced efficiently. When the number of requests in this minute reaches $rqnum$, the container will be closed. The advantage of our solution is that it can reduce resource wastage during idle times and after the completion of requests.

VII. Implementation

SMore comprises a performance monitor, GPU manager on each node, and a cluster-level scheduler, predictor, and prewarmer. The performance monitor on each node tracks GPU utilization and communicates this information to the scheduler to update cluster status information. The GPU manager on each node handles the actual GPU allocations for functions and communicates with the scheduler. The scheduler makes admission decisions for incoming requests based on collected information and, upon admission, determines the most suitable GPU for their execution. Additionally, the scheduler interacts with both the prewarmer and predictor. The predictor receives potential co-location combinations and returns estimated degradation to the scheduler for decision-making. During serving serverless requests, the prewarmer predicts the next request and preloads the model in advance.

We implement a prototype of SMore in 3000+ LOC of Python. We use the PyTorch framework [38] to implement the corresponding DL workloads and predictors. We use Flask to implement the instances for executing serverless function requests and gRPC to exchange control messages between the scheduler and nodes.

VIII. Evaluation

A. Evaluation Methodology

Experimental setup. We evaluate SMore on a machine of eight GPUs. TABLE IV summarizes the configurations of the

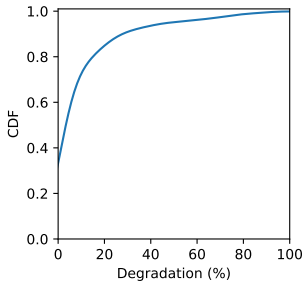


Fig. 7: The degradation distribution of serverless functions under co-location.

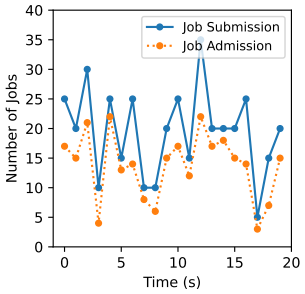


Fig. 8: Number of submitted and admitted tasks by SMORE over time.

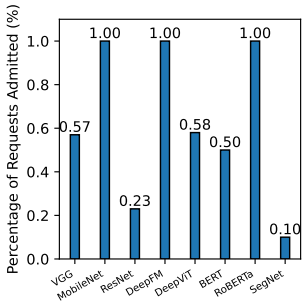


Fig. 9: Admitted ratio of different function types by SMORE over time.

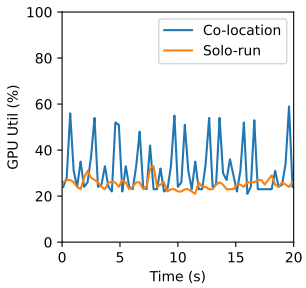


Fig. 10: Resource utilization between exclusive and co-location execution.

TABLE IV: Experimental testbed configuration.

Component	Specification
CPU device	Intel Xeon Gold 6248R
Shared LLC Size	71.5MB
Number of sockets	2
Memory Capacity	512GB
Processor BaseFreq.	3.00 GHz
Operating System	Ubuntu 20.04LTS
Threads	192 (96 physical cores)
SSD Capacity	11TB
GPU device	Nvidia RTX 3090
GPU Memory	24GB DGDDR6
GPU SM cores	10496
Number of GPUs	8

machine. To focus our results on the impact of interference, we co-locate serverful and serverless workloads on these GPUs. The invocations of serverless workloads are hosted on the same machine. We make invocations of functions to the scheduler according to the provided trace, the scheduler will make admission control to decide which GPU to locate the function to minimize performance degradation.

Workloads. The serverful workloads we use for evaluation are listed in TABLE I. For the co-location degradation predictor, only 20% of the task combinations of model name and batch size are used for training. These serverful workloads have a wide variety of GPU utilization ranges, allowing us to comprehensively evaluate our solution under different utilization conditions. The models used by serverless functions are also shown in TABLE I. Their batch sizes are set to ≤ 32. The invocation patterns of these serverless functions are simulated using the production trace from Azure Function [10], which includes 14-day request statistics with different arrival patterns. We classify functions based on their execution time and map them from large to small to the actual functions in Azure Function. Due to the sparse distribution of invocations in Azure Functions, which hinders achieving high requests per second (RPS), we first statistically analyze the distribution of different functions over one-minute intervals. Subsequently, this distribution is converted to a per-second basis and scaled by a corresponding factor to generate test scenarios with the desired RPS. Their latency SLOs are randomly set to values between 1 to 4 times their $P99$ latencies.

B. Real Testbed Experiments with a Single Application

We first consider a co-location with a single serverful DL training application running on one GPU to understand the impact on a single training task under co-location circumstances. We choose MobileNet with relatively uniform utilization as the serverful application. We generated a trace of approximately 390 requests lasting 20 seconds using Azure Functions.

a) Performance degradation: To measure the degradation, we monitor the average iteration time of serverful workloads and the execution latency of serverless functions. Fig. 7 displays the distribution of latency degradation for functions admitted in the trace under co-location conditions. It can be observed that almost all functions experience degradation less than 100%, with over half of the functions experiencing degradation below 10%. This indicates that SMORE effectively manages the degradation of co-located serverless functions.

b) Request admission status: Scheduling algorithm effectiveness can be measured by the ratio of functions admitted that can meet their deadlines. Fig. 8 shows the number of submitted and admitted functions during execution on the testbed. From the figure, it can be observed that in most cases when there are many incoming requests, SMORE successfully handles the majority of requests, with only rare instances where it processes a small portion of them. This indicates that SMORE strives to accept as many tasks as possible, thus maximizing utilization.

c) Request admission preference: To further understand the SMORE's tendency in selecting admitted task types, we conduct an analysis of the proportion of admitted function types, as shown in Fig. 9. From this, it can be observed that the proportion of admitted tasks is high for MobileNet, DeepFM, and RoBERTa, and moderate for VGG, DeepViT and BERT. This indicates that SMORE tends to prioritize functions that exhibit lower performance degradation and potentially lead to higher GPU utilization.

d) GPU utilization: Our objective is to improve GPU utilization while minimizing the performance impact on serverful workloads. Hence, we also prioritize improvements in average utilization as a key metric. Fig. 10 illustrates the changes in GPU utilization over time between exclusive and co-location cases. It can be observed that at each time point when serverless functions arrive, there is a significant increase in

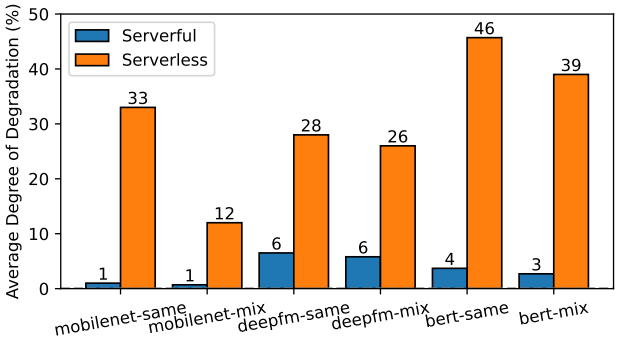


Fig. 11: Comparison of degradation between exclusive and co-location execution cases.

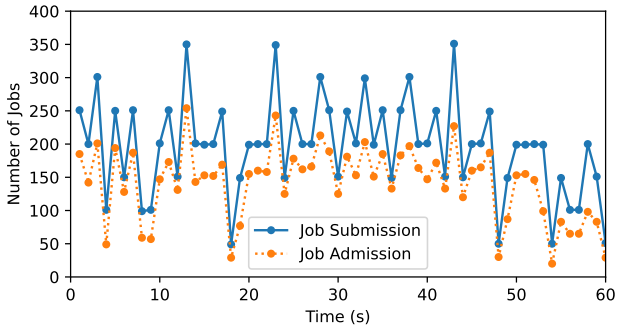


Fig. 12: Number of submitted and admitted functions by SMORE over time.

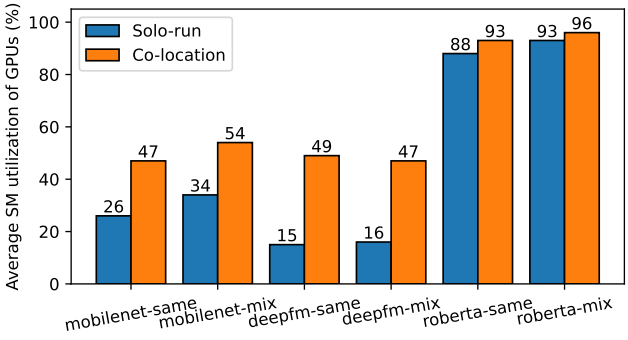


Fig. 13: Comparison of utilization between exclusive and co-location execution cases.

GPU utilization. The magnitude of this increase is dependent on the type of serverless functions and batch size used for co-location. This indicates that adopting co-location indeed increases GPU utilization.

C. Real Testbed Experiments with Multiple Applications

After evaluations on a single application, we now consider the scenario where multiple serverful DL training applications are hosted on one node and expand the evaluation based on eight GPUs. We select three different models with varying utilization rates, ranging from low to high, and configure them with two settings: the same batch size and different batch sizes. This results in a total of six configurations representing different serverful utilization rates. For the same batch size, we opted for a moderate value. We select four varying values from small to large for different batch sizes. We generate traces of approximately 12000 requests per minute scaled from Azure Functions.

a) Performance degradation: Subsequently, we compare the performance degradation of both serverful and serverless workloads under different settings to assess whether SMORE can control degradation within a certain range. Fig. 11 displays the average latency degradation of admitted functions under co-location conditions, as well as the degradation level of the serverful workload. From this, it can be observed that the average degradation of functions is controlled to be below 50%, while the execution performance of the serverful workloads is hardly affected. This indicates that SMORE maintains control over degradation.

b) Request admission status: The reception of requests can provide insights into the performance of the scheduling algorithm. Fig. 12 shows the number of submitted and admitted functions during execution on multiple GPUs co-located with MobileNet. From this, it can be seen that in the case of multiple GPUs, almost all functions in the trace can be admitted and executed. This indicates that co-locating serverless functions in a GPU cluster has advantages.

c) GPU utilization: Similarly, we finally focus on the change in utilization before and after implementing the SMORE scheme. Fig. 13 shows the average GPU utilization under different configurations. From this, it can be observed that under almost all configurations, the average GPU utilization has increased by $3\%-34\%$. For the scenario of RoBERTa-mix, due to its inherently high utilization, resulting in relatively significant degradation of co-located tasks, the proportion of admitted tasks is low, leading to only a minor increase in utilization.

D. Baseline Comparison

To further demonstrate the effectiveness of our system in controlling degradation and enhancing utilization, we implement different allocation method based on the co-execution configuration of TGS [39] as the baseline. TGS is a system designed to provide transparent GPU sharing for deep learning workloads in container clouds.

Baselines. We compare SMORE with three scheduling methods listed as follow.

- Random: All tasks are executed on the GPUs within the current cluster following a First-Come, First-Served (FCFS) principle, with random assignment and without considering potential performance degradation.
- EDF-util: Earliest Deadline First (EDF) is a classic scheduling algorithm that prioritizes incoming serverless function requests based on their deadlines, serving the task with the earliest deadline first. We integrate this approach with utilization-based admission control: an incoming function is admitted if the sum of its utilization

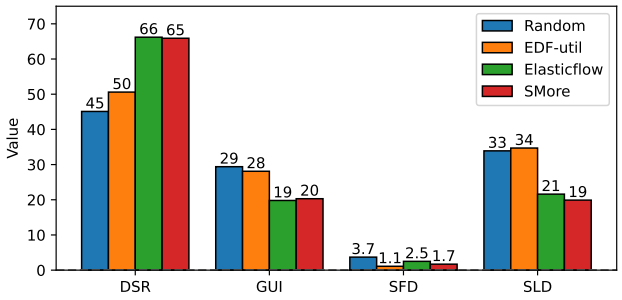


Fig. 14: Comparison between SMORE and other scheduling methods.

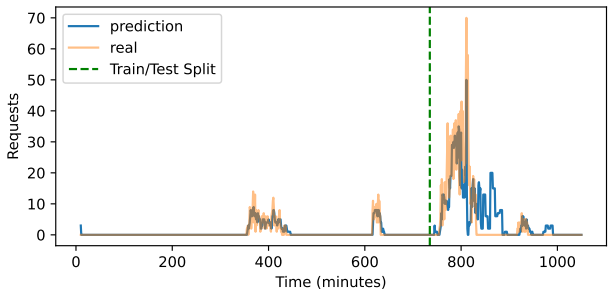


Fig. 15: Prediction results of LS-LSTM on one function trace.

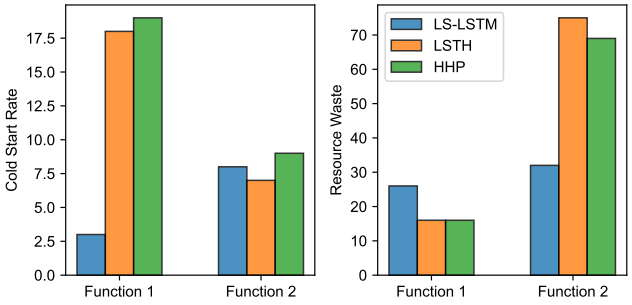


Fig. 16: Comparison results of cold start solutions.

and the current serverful utilization is below a predefined threshold.

- ElasticFlow: ElasticFlow [40], building upon EDF, initially sorts serverless functions by their deadlines. Subsequently, it employs a greedy approach to minimize single-GPU throughput degradation when allocating multiple GPUs to a single function. Integrating ElasticFlow's scheduling with our admission control method allows for a greedy minimization of performance degradation when placing individual functions onto the cluster.

To better demonstrate the differences between SMORE and the baselines, we increase the number of requests per minute to 16000. Fig. 14 illustrates the differences among these methods across various metrics, including deadline satisfactory ratio (DSR), GPU utilization improvement (GUI), serveful workload degradation (SFD), and serverless function degradation (SLD). It is evident that the degradation-aware approach (ElasticFlow and SMORE) significantly improves the deadline satisfaction ratio compared to methods unaware of potential degradation (Random and EDF-util). Furthermore, admitted serverless functions exhibit lower degradation, and the degradation of serverful workloads remains within acceptable limits. Comparing our method with ElasticFlow reveals that our approach achieves a higher GUI, a lower SFD, and a lower SLD. This indicates that our method effectively controls the degree of degradation, thereby enhancing resource utilization while maximizing the satisfaction of a greater number of serverless functions.

E. Component Evaluation

a) Performance degradation predictor evaluation: We set different batch sizes for different models and combine two different workloads, resulting in a total of 1024 samples. The prediction model is trained in an offline mode based on the collected 1024 samples. We select $\alpha \times 1024$ samples to train the model and the left samples are used to test the model accuracy. α ranges from 0 to 1. After trying different values of α, we find that when $\alpha = 0.2$, we can get RMSLE of approximately 0.3. This indicates that we can efficiently train a model with sufficient accuracy using a small sampling space. We compare the results of training and testing different machine learning models, as shown in Table V. Among them, the Random Forest model performs the best, achieving the best performance in both RMSLE and MAE metrics.

TABLE V: RMSLE and MAE for co-location degradation prediction.

Metric	Random Forest	Bagging	Linear
RMSLE	0.305	0.314	0.870
MAE	0.473	0.494	0.721

b) Prewarmer evaluation: We conduct evaluations on the prewarmer module targeting to solve cold starts. For cold start problems, we compare LS-LSTM with the Long-Short Term Histogram (LSTH) [22] and hybrid histogram policy (HHP) [10] to evaluate our method's effectiveness. LSTH collects long-term and short-term interval time (IT) graphs of requests to infer pre-warm and keep-alive values for early container startup. HHP only collects a configurable duration of idle times. The corresponding metrics are the cold start rate of serverless functions and resource waste time when the model is loaded, but no requests arrive. We select actual functions from Azure Functions as training samples.

Fig. 15 shows LS-LSTM prediction result on one function. It can be observed that LS-LSTM can almost accurately predict whether functions will arrive within the minute.

We evaluate the cold start rate and resource waste rate for two functions with varying spike counts, noting that the second function has more spikes. The final results are shown in Fig. 16. We can observe that for Function 1, our approach reduced the cold start rate by 15% while increasing resource wastage by 10%. For Function 2, our approach almost maintained the same cold start rate while reducing resource wastage from 75% to 32%, nearly halving the resource wastage. Compared with LSTH and HHP, our LS-LSTM policy can solve cold start rate problem in more common conditions.

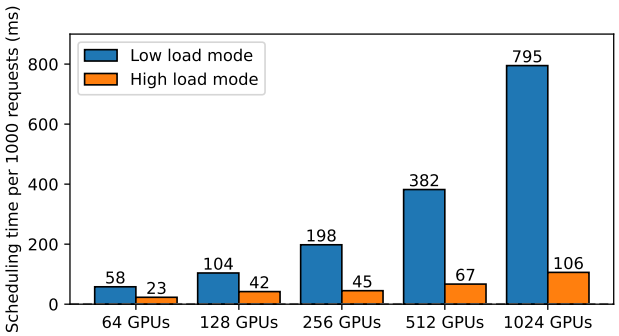


Fig. 17: Scheduling overhead under large-scale simulation experiments.

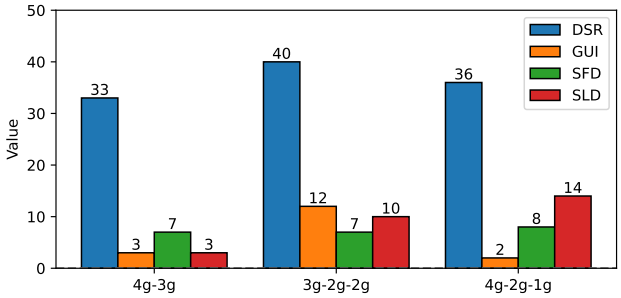


Fig. 18: Comprasion results of different workload groups on MIG-enabled GPUs.

c) Scheduling overheads: For a scheduling space with 8 GPUs, the time overhead required for a single scheduling operation is 0.01 ms. It accounts for nearly less than 1% of the execution time of serverless functions and supports the scalability of the system. To validate the effectiveness of our scheduling system on a larger-scale cluster, we conduct simulation experiments to evaluate scheduling overhead under both high-load and low-load conditions. The core scheduling logic and algorithms in the simulations are consistent with the implementation of the testbed. The results of the simulations are presented in Fig. 17. It is evident that even with a cluster size of 1024 GPUs, the scheduling overhead per request remains below one millisecond. Under high load conditions, our scheduling system allocates functions to the first suitable GPU it encounters. With 1024 GPUs, the scheduling latency per request is approximately 0.1 ms, which is significantly less than the function execution time. This large-scale simulation demonstrates the favorable scalability of our system.

F. Further Evaluation on MIG-enabled GPUs

Multi-Instance GPU (MIG) is a feature offered by NVIDIA's latest data center GPUs that enables the dynamic partitioning of GPU resources. For instance, an A100 GPU with 7 GPCs can be dynamically divided into 5 distinct slice configurations. Building upon MIG, MISO [41] dynamically adjusts the optimal number of GPCs allocated to workloads based on their resource utilization. To validate the applicability of our method to MIG-enabled GPUs, we first employ MISO on an A100 40GB GPU to assign various MIG configurations to different serverful workloads. Subsequently, we generate a request trace with 4500 requests per minute scaled from Azure FUnctions. The evaluation results across different MIG partition combinations are illustrated in Fig. 18. The results indicate that SMORE can accommodate 30% of serverless functions while maintaining the degradation of both serverful and serverless workloads within acceptable limits compared with the original MISO.

G. Discussion and Future Work

GPU types. In production clusters, the majority of devices are relatively low-caliber GPUs. This work can help improve the utilization rate of this type of GPU and our implementation can be easily extended on this type of GPU. For high-calibre GPUs, we consider them as future work considerations.

Other resources. This work mainly focuses on the utilization rate of GPUs. Even without analyzing other computing resources, good results can still be achieved. However, if a more detailed analysis of other resources can be conducted, the interpretability of the method can be better improved. We consider this as future work.

Large models. The increasingly popular large language models (LLMs) demand substantial GPU memory, occasionally exceeding the capacity of a single GPU. As a result, they need to be carefully distributed across multiple GPUs to achieve efficient model parallelism and pipeline [42]. It is possible to consider deploying large model workloads as serverful tasks while concurrently deploying serverless tasks to improve cluster utilization during idle periods. However, deploying large model workloads and conducting experiments with them require significant computational power. We will attempt this in the future if conditions permit. Furthermore, if large language models are to be employed as serverful workloads, the GPU memory resources may constrain the deployment of serverless functions. To address this challenge, more efficient model switching and memory management mechanisms need to be established. We consider this as a topic for future research.

IX. Related Work

a) Workload analysis: Distributed training is essential for large deep learning workloads but faces challenges like reduced average GPU computation throughput [40], idle waiting during group scheduling, and significant communication and I/O overhead [43]. An early description of a machine learning training task in Alibaba's PAI [44] suggested replacing the PS-Worker architecture with Ring AllReduce to utilize the high-speed NVLink between GPUs better. Some deep learning cluster schedulers [45], [46] have investigated training efficiency under different network bandwidths and proposed solutions to mitigate the communication overheads to accelerate training. These works mainly focus on distributed training, while our characterization focuses on general DL workloads.

b) Workload scheduling: Deep learning training workloads in current multi-tenant production clusters are managed by infrastructures such as Kubernetes or YARN, where

the workloads exclusively occupy the allocated GPU resources, leading to a general under-utilization problem [6]. Several related works have made efforts to address the problem. Gandiva [13] has proposed time-slicing, migration, and packing methods to allow GPU sharing. Themis [47] and Gandiva$_{fair}$ [14] focus on addressing fairness among different workloads. AlloX [15] models the scheduling problem as a minimum-cost two-point matching problem to efficiently and fairly schedule deep learning workloads among exchangeable resources. AntMan [8] introduces opportunistic deep learning workloads as low-priority jobs to maximize the utilization of GPU cycles. Orion [48] intercepts GPU kernel launches to design an operator scheduling algorithm to improve GPU utilization. Compared with existing works, SMORE targets workloads of different types and optimizes cluster utilization based on serverless computing.

c) Serverless-based optimization: Some existing work has leveraged serverless computing to optimize cluster resource utilization or improve system throughput. UnFaaSener [18] designs optimal offloading decisions to harvest non-serverless compute resources of serverless users to reduce their bills. Libra [19] takes dynamic profiling resource status to harvest idle resources for accelerating function invocations. Zhang et al. [9] harvests idle resources to create new containers for executing additional functions. ServerMore [20] proposes opportunistic executing serverless functions alongside traditional serverful applications, imposing constraints on CPU, bandwidth, and PCIe to manage performance degradation. Gsight [49] enhances the number of function services and ensures QoS through incremental learning predictions. INFless [22] combines built-in batching and non-uniform scaling mechanisms to improve system throughput. Compared with these works, SMORE leverages serverless computing to colocate different types of workloads in order to enhance cluster GPU utilization.

d) Cold start optimization: Cold start time is a common source of latency in serverless frameworks. SSC [50] and faaShark [51] take a gradient-based algorithm for pre-warming containers but are not suitable for GPU serverless functions. The hybrid histogram policy (HHP) [10] tracks the idle times of a configurable duration and derives two parameters, the pre-warming window and keep-alive window, to control waiting time and keep-alive time after the last execution. Additionally, INFless [22] proposes a Long-Short Term Histogram (LSTH) policy, which tracks the idle time of both long and short duration and draws the histogram to derive the two parameters Further reducing cold start rates and resource waste. Compared to HHP and LSTH, SMORE can better reduce resource wastage and decrease cold start rates. FaaSwap [52] leverages model swapping to reduce the cold start time of model inference, which is orthogonal to SMORE.

X. CONCLUSION

In this paper, we propose SMORE, a framework based on serverless computing to optimize the efficiency of GPU cluster resource utilization for deep learning tasks. SMORE predicts potential degradation for incoming serverless function requests. Based on these predictions, we design a degradation-aware scheduling strategy to increase GPU utilization while controlling degradation to a certain extent. Additionally, SMORE utilizes LS-LSTM to establish prewarmers to address the cold start issue. Through prototype testing, SMORE demonstrates a 34% improvement in GPU utilization compared to exclusive execution condition, without significant degradation.

REFERENCES

[1] Z. Ye, W. Gao, Q. Hu, P. Sun, X. Wang, Y. Luo, T. Zhang, and Y. Wen, "Deep learning workload scheduling in gpu datacenters: A survey," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–38, 2024.

[2] K. Chowdhary and K. Chowdhary, "Natural language processing," *Fundamentals of artificial intelligence*, pp. 603–649, 2020.

[3] M. Malik, M. K. Malik, K. Mehmood, and I. Makhdoom, "Automatic speech recognition: a survey," *Multimedia Tools and Applications*, vol. 80, pp. 9411–9457, 2021.

[4] H. Ko, S. Lee, Y. Park, and A. Choi, "A survey of recommendation systems: recommendation models, techniques, and application fields," *Electronics*, vol. 11, no. 1, p. 141, 2022.

[5] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "Mlaas in the wild: Workload analysis and scheduling in large-scale heterogeneous gpu clusters," in *USENIX NSDI*, 2022.

[6] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant gpu clusters for dnn training workloads." in *USENIX ATC*, 2019.

[7] Y. Gao, Y. He, X. Li, B. Zhao, H. Lin, Y. Liang, J. Zhong, H. Zhang, J. Wang, Y. Zeng *et al.*, "An empirical study on low gpu utilization of deep learning jobs," in *IEEE/ACM ICSE*, 2024.

[8] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia, "Antman: Dynamic scaling on gpu clusters for deep learning." in *USENIX OSDI*, 2020.

[9] Y. Zhang, Í. Goiri, G. I. Chaudhry, R. Fonseca, S. Elnikety, C. Delimitrou, and R. Bianchini, "Faster and cheaper serverless computing on harvested resources," in *ACM SOSP*, 2021.

[10] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *USENIX ATC*, 2020.

[11] H. Zhao, Z. Han, Z. Yang, Q. Zhang, F. Yang, L. Zhou, M. Yang, F. C. Lau, Y. Wang, Y. Xiong *et al.*, "Hived: Sharing a gpu cluster for deep learning with guarantees," in *USENIX OSDI*, 2020.

[12] L. Luo, P. West, A. Krishnamurthy, L. Ceze, and J. Nelson, "Plink: Discovering and exploiting datacenter network locality for efficient cloud-based distributed training," *MLSys*, 2020.

[13] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *USENIX OSDI*, 2018.

[14] S. Chaudhary, R. Ramjee, M. Sivathanu, N. Kwatra, and S. Viswanatha, "Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning," in *ACM EuroSys*, 2020.

[15] T. N. Le, X. Sun, M. Chowdhury, and Z. Liu, "Allox: compute allocation in hybrid clusters," in *ACM EuroSys*, 2020.

[16] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar *et al.*, "Cloud programming simplified: A berkeley view on serverless computing," *arXiv*, 2019.

[17] Y. Li, Y. Lin, Y. Wang, K. Ye, and C. Xu, "Serverless computing: state-of-the-art, challenges and opportunities," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1522–1539, 2022.

[18] G. Sadeghian, M. Elsakhawy, M. Shahrad, J. Hattori, and M. Shahrad, "Unfaasener: Latency and cost aware offloading of functions from serverless platforms," in *USENIX ATC*, 2023.

[19] H. Yu, C. Fontenot, H. Wang, J. Li, X. Yuan, and S.-J. Park, "Libra: Harvesting idle resources safely and timely in serverless clusters," in *ACM HPDC*, 2023.

[20] A. Suresh and A. Gandhi, "Servermore: Opportunistic execution of serverless functions in the cloud," in *ACM SoCC*, 2021.

[21] X. Chen, Z. Cai, R. Buyya *et al.*, "Fasdl: An efficient serverless-based training architecture with communication optimization and resource configuration," *IEEE Transactions on Computers*, 2024.

[22] Y. Yang, L. Zhao, Y. Li, H. Zhang, J. Li, M. Zhao, X. Chen, and K. Li, "Infless: a native serverless system for low-latency, high-throughput inference," in *ACM ASPLOS*, 2022.

[23] P. Jiang, H. Wang, Z. Cai, L. Gao, W. Zhang, R. Ma, and X. Zhou, "Slob: Suboptimal load balancing scheduling in local heterogeneous gpu clusters for large language model inference," *IEEE Transactions on Computational Social Systems*, 2024.

[24] G. Yeung, D. Borowiec, R. Yang, A. Friday, R. Harper, and P. Garraghan, "Horus: Interference-aware and prediction-based scheduling in deep learning systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 88–100, 2021.

[25] W. Cui, H. Zhao, Q. Chen, N. Zheng, J. Leng, J. Zhao, Z. Song, T. Ma, Y. Yang, C. Li *et al.*, "Enable simultaneous dnn services based on deterministic operator overlap and precise latency prediction," in *ACM SC*, 2021.

[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[27] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv*, 2017.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[31] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *NAACL*, J. Burstein, C. Doran, and T. Solorio, Eds., 2019.

[32] Y. Li, Y. Zhang, Z. Zhao, L. Shen, W. Liu, W. Mao, and H. Zhang, "CSL: A large-scale Chinese scientific literature dataset," in *ACM COLING*, 2022.

[33] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for squad," *arXiv preprint arXiv:1806.03822*, 2018.

[34] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: a factorization-machine based neural network for ctr prediction," *arXiv*, 2017.

[35] Z. Xie, J. Chen, Y. Feng, K. Zhang, and Z. Zhou, "End to end multi-task learning with attention for multi-objective fault diagnosis under small sample," *Journal of Manufacturing Systems*, vol. 62, pp. 301–316, 2022.

[36] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[37] Y. Wang, K. Arya, M. Kogias, M. Vanga, A. Bhandari, N. J. Yadwadkar, S. Sen, S. Elnikety, C. Kozyrakis, and R. Bianchini, "Smartharvest: harvesting idle cpus safely and efficiently in the cloud," in *ACM EuroSys*, 2021.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[39] B. Wu, Z. Zhang, Z. Bai, X. Liu, and X. Jin, "Transparent gpu sharing in container clouds for deep learning workloads," in *USENIX NSDI*, 2023.

[40] D. Gu, Y. Zhao, Y. Zhong, Y. Xiong, Z. Han, P. Cheng, F. Yang, G. Huang, X. Jin, and X. Liu, "Elasticflow: An elastic serverless training platform for distributed deep learning," in *ACM ASPLOS*, 2023.

[41] B. Li, T. Patel, S. Samsi, V. Gadepally, and D. Tiwari, "Miso: exploiting multi-instance gpu capability on multi-tenant gpu clusters," in *Proceedings of the 13th Symposium on Cloud Computing*, 2022, pp. 173–189.

[42] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for transformer-based generative models," in *USENIX OSDI*, 2022.

[43] D. G. Murray, J. Simsa, A. Klimovic, and I. Indyk, "tf. data: A machine learning data processing framework," *arXiv*, 2021.

[44] M. Wang, C. Meng, G. Long, C. Wu, J. Yang, W. Lin, and Y. Jia, "Characterizing deep learning training workloads on alibaba-pai," in *IEEE IISWC*, 2019.

[45] Y. Peng, Y. Zhu, Y. Chen, Y. Bao, B. Yi, C. Lan, C. Wu, and C. Guo, "A generic communication scheduler for distributed dnn training acceleration," in *ACM SOSP*, 2019.

[46] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," *arXiv*, 2019.

[47] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla, "Themis: Fair and efficient gpu cluster scheduling," in *USENIX NSDI*, 2020.

[48] F. Strati, X. Ma, and A. Klimovic, "Orion: Interference-aware, fine-grained gpu sharing for ml applications," in *ACM EuroSys*, 2024.

[49] L. Zhao, Y. Yang, Y. Li, X. Zhou, and K. Li, "Understanding, predicting and scheduling serverless workloads under partial interference," in *ACM SC*, 2021.

[50] S. Pan, H. Zhao, Z. Cai, D. Li, R. Ma, and H. Guan, "Sustainable serverless computing with cold-start optimization and automatic workflow resource scheduling," *IEEE Transactions on Sustainable Computing*, 2023.

[51] H. Zhao, S. Pan, Z. Cai, X. Chen, L. Jin, H. Gao, S. Wan, R. Ma, and H. Guan, "faashark: An end-to-end network traffic analysis system atop serverless computing platforms," *IEEE Transactions on Network Science and Engineering*, 2023.

[52] M. Yu, A. Wang, D. Chen, H. Yu, X. Luo, Z. Li, W. Wang, R. Chen, D. Nie, and H. Yang, "Faaswap: Slo-aware, gpu-efficient serverless inference via model swapping," *arXiv*, 2023.

Junhan Liu received the bachelor's degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2023, where he is currently pursuing the master's degree in computer science.

His research interests are focused on resource schedule in cloud computing.

Zinuo Cai received the bachelor's degree in software engineering from Shanghai Jiao Tong University, Shanghai, China, in 2021, where he is currently pursuing the Ph.D. degree in computer science.

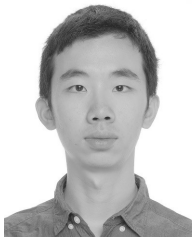
His research interests are focused on resource schedules and system security in cloud computing.

Yumou Liu is currently pursuing the bachelor's degree in Computer Science at Shanghai Jiao Tong University, Shanghai, China.

His research interests are focused on cloud computing.

Hao Li is currently pursuing the bachelor's degree in Computer Science at Shanghai Jiao Tong University, Shanghai, China.

His work interests in algorithms, artificial intelligence, and computer architecture.



Zongpu Zhang received the BS degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2019. He is currently working toward the PhD degree with the Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University, Shanghai, China. His research interests include cloud computing, network system, and hardware acceleration technologies.

Ruhui Ma (Member, IEEE) received the Ph.D. degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2011.

He is currently an Associate Professor with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include cloud computing systems, artificial intelligence (AI) systems, and machine learning.

Rajkumar Buyya (Fellow, IEEE) is a Redmond Barry distinguished professor and director of the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Australia. He has authored more than 625 publications and seven text books including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese and international markets respectively. He is one of the highly cited authors in computer science and software engineering worldwide.