Job Recommendation Service for GPU Sharing in Kubernetes

Aritra Ray[1], Kyle Lafata[1], Zhaobo Zhang[2], Ying Xiong[2], and Krishnendu Chakrabarty[3]

[1]Duke University, *NC, USA*
[2]Futurewei Technologies, *CA, USA*
[3]Arizona State University, *AZ, USA*
[1]{*aritra.ray, kyle.lafata*}*@duke.edu*
[2]{*zzhang1, yxiong*}*@futurewei.com*
[3]{*krishnendu.chakrabarty*}*@asu.edu*

2023 IEEE Cloud Summit | 979-8-3503-2217-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/CloudSummit57601.2023.00008

***Abstract*—Cloud infrastructures encourage the multi-tenancy of hardware resources. User-defined Machine Learning (ML) training jobs are offloaded to the cloud for efficient training. State-of-the-art resource schedulers do not preserve user privacy by accessing sensitive meta-data of the user-defined training workload. We present the design of a fine-grain, online, privacy-preserving job scheduler built on top of the Kubernetes platform in combination with Argo workflow. We categorize ML training workloads on standard benchmark architectures and datasets over sixty-six different features, cluster them based on exploratory data analysis, and perform inter- and intra-cluster task interference. We assume black-box access to the user-defined ML training jobs and refrain from accessing sensitive meta-data. We define three scheduler-level objectives to maximize gains from users' and cloud providers' perspectives. Our scheduler promotes multi-tenancy by intelligently selecting competitor jobs for concurrent execution in every pod while abiding by scheduler-level objectives.**

***Index Terms*—GPU sharing, job scheduler, privacy-preserving, Kubernetes**

I. Introduction

Machine Learning (ML) is used in several application domains, ranging from computer vision [1], natural language processing [2], recommendation systems [3], and robotics [4], to name a few. Training of ML networks requires accessibility to hardware accelerators like GPUs. Users offloading their ML network training to compute clusters in the cloud has emerged as a popular choice.

Cloud infrastructures designed for ML training encourage multi-tenancy of the compute resources. Efforts to facilitate the multi-tenancy of hardware accelerators concerning GPUs have led to service-level orchestration [5], runtime-level scheduling [6], and resource-level management [7], [8]. Studies [9], [10] aim to improve the Job Completion Time (JCT) of user-defined tasks, improve kernel performance [11], reduce latency [6], while maintaining the Service Level Agreements (SLA) [7], [8]. Efforts involving service-level orchestration focus on designing query schedulers having heuristic-based preemption schedulers [10]. Resource level management mostly involves multi-process based resource scheduling [7], adaptive batching [7], kernel tuning [11]. However, state-of-the-art GPU schedulers [12], [13] refrain from considering preserving user privacy for scheduling user-defined ML training jobs.

A popular framework for handling auto-deployment and managing containerized applications alongside scaling is Kubernetes (K8s) [14]. It facilitates production-oriented container orchestration. The kube-scheduler, a component of K8s, which facilities valid placement of pods to the nodes while accounting for all system-level constraints, is coarse-grained. User-defined ML training jobs are placed on pods and executed on a container runtime, as placed by the kube-scheduler.

On top of K8s, we propose the design of a fine-grain, online, privacy-preserving job scheduler combining Argo workflow. We design a job profiler on worker nodes to extract features from user-defined ML training jobs. We categorize the ML workloads and perform inter- and intra-cluster task interference. In online mode, we extract features of the user-defined ML workload, cluster them based on the extracted features, and make informed scheduling decisions to meet the scheduler-level objectives. We assume black-box access to the user-defined ML training jobs throughout the process. We refrain from accessing any job-specific sensitive hyper-parameters like model architecture, batch size, datasets, and the number of epochs. The key contributions of our work include the following:

- We propose the design of a fine-grain, online, privacy-preserving job scheduler, built on top of Kubernetes, harnessing Argo workflow that promotes multi-tenancy through intelligently selecting competitor jobs for concurrent execution in every pod, abiding to scheduler-level objectives.
- We propose the design of three schedulers in combination with Argo workflow, which improves multi-tenancy in GPUs by 10.82% from service providers' perspectives, and reduces job slowdown by 5.08% from users' perspectives, with respect to our defined metrics, in comparison to kube-scheduler.

The rest of the paper is organized as follows: Section II discusses our proposed methodology, followed by the evaluation

979-8-3503-2217-0/23/$31.00 ©2023 IEEE
DOI 10.1109/CloudSummit57601.2023.00008

results in Section III. Section IV concludes the paper.

II. Design Methodology

In the traditional design of the Kubernetes architecture, the kube-scheduler assigns pods to available nodes according to their resource requirements, preserving system constraints. Kube-scheduler, coarse-grain in nature, helps decide which nodes would be the valid placement for every pod while accounting for all system-level constraints. User-defined ML training workloads are placed in the pods for execution in the assigned node with the available computing resources. In our paper, we propose the design of a fine-grain, online, privacy-preserving job scheduler that promotes multi-tenancy through intelligently selecting competitor jobs for concurrent execution in every pod.

We introduce two components in the control plane, namely, *Knowledge Base* (*KB*), and *Job Recommendation Service* (*JRS*), alongside a *Job Profiler* (*JP*) in the worker node, as outlined in Fig. 1 of our proposed design. We assume the availability of n user-defined ML training workloads $j_1, j_2, ..., j_n$ in the job queue. With a black-box access, *JP* profiles $\forall\ j$ to extract features for every job while preserving user privacy. The *KB* component keeps a tab on the profiling results obtained from *JP* and clusters the jobs based on the profiled features. It also assigns an uncertainty score depending on the distance of the job from its assigned cluster center. Relative rankings of inter- and intra-cluster task interference based on mean values of our defined metrics are analyzed in an online manner in the *JRS* component. Taking into account ⓐ the available number of jobs to be scheduled, ⓑ the task interference analysis, and ⓒ the multi-tenancy objective prioritizing the user, the cloud provider, or a combination of both; the *JRS* recommends two or more competitor jobs to the kube-scheduler for concurrent scheduling in the same pod. The kube-scheduler, guided by the *JRS*, schedules the competitor jobs in the same pod in conjunction with Argo workflow for concurrent execution. The execution results are returned to the *JRS* for dynamically updating the task interference analysis results.

The following sub-sections introduce workload characterization and task interference analysis in further detail. It is followed by an elaborate discussion of our *JRS* component that recommends competitor jobs to the kube-scheduler to promote multi-tenancy while abiding by scheduler-level objectives.

A. Workload Characterization and Task Interference Analysis

We analyze ML training workloads on standard benchmark architectures and create a dataset with privacy-preserving features. We perform exploratory data analysis on the collected dataset and generate a synthetic workload to bolster the dataset size. Based on unsupervised clustering techniques, we cluster the ML workloads, all while preserving user privacy.

- **Characterizing ML Training Workloads:** Using our job profiler, *JP*, we characterize user-defined ML training workloads over several attributes, ranging from network parameters, GPU and CPU utilization metrics, GPU and disk memory utilization metrics, interrupts, system calls, runtime environment, context switching, and several other. We profile jobs on standard benchmark architectures over publicly available datasets from categories of image classification, image segmentation, reinforcement learning, generative adversarial networks, and natural language processing.
- **Feature Selection:** To maintain user privacy, we discard the sensitive hyper-parameters like job ID, batch size, epochs, the dataset used, and model architecture. For our two target variables, maximum GPU utilization percent ($Max.GPU_{util}$) and maximum GPU memory allocation percent ($Max.GPU_{memAlloc}$), we perform exploratory data analysis to find the correlation to other features through Pearson's correlation coefficient. Additionally, we analyze the inter-dependence of the other features for our target variables through random forests, ridge regression, and lasso regression.
- **Synthetic Data Generation:** We generate synthetic ML training workload for our target and predictor variables to generate more insights while clustering. We generate synthetic workloads for varying batch sizes from the collected ground truth (GT_{real}) through interpolation measures of barycentric, krogh, pchip, spline, and linear. To evaluate the quality of the generated synthetic samples, we blind the collected ground truth GT_{real}, and assuming the synthetic workload as pseudo-ground truth; we re-generate the collected ground truth ($ReGen_{GT}$). We assess the quality of our synthetic data generation process by estimating the mean absolute difference between $ReGen_{GT}$ and GT_{real} and choose the best interpolation technique for every workload.
- **Clustering:** We cluster GT_{real} and synthetic workload

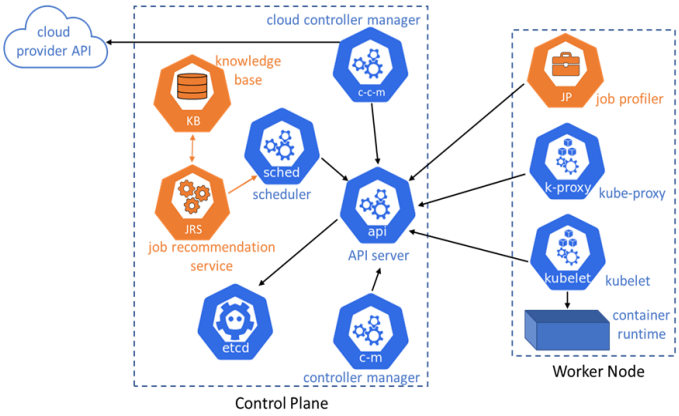

Fig. 1. Our proposed design of a fine-grain online privacy-preserving job scheduler, built on top of the Kubernetes platform. The components outlined in blue are the traditional components of the Kubernetes cluster. In the control plane, we introduce two components, *KB* (Knowledge Base) and *JRS* (Job Recommendation Service), and a *JP* (Job Profiler) in the worker node, all deployed as pods. For scheduling every user-defined ML training workload in the compute cluster, *KB* logs the job profile, obtained from *JB*, while preserving user-privacy. Through an online inter- and intra-cluster task interference analysis, *JRS* recommends the kube-scheduler for the most suited competitor job that meets the desired scheduler-level objective of multi-tenancy.

based on our target and predictor variables through the k-means algorithm and assign every ML training job into a cluster. We extensively schedule inter- and intra-cluster training workloads on the same GPU for task interference analysis. We also record the relative task interference ranking based on mean values. To make online scheduling decisions, *JRS* recommends jobs relying on the relative ranking of inter- and intra-cluster task interference to meet scheduling level objectives. The relative ranking gets updated after every execution.

B. Job Recommendation Service:

We define two metrics, namely, Individual Slowdown (Ind_{SD}), and Packing Saving (Pac_{Sav}), to analyze the task interference. Ind_{SD} refers to the percent difference in JCT for a job to complete execution when it runs in the hardware accelerator without any competitor job to the presence of a competitor job. Equation 1 expresses the Ind_{SD} for a job, j, where J_c is the job completion time when the job (j) is executed in the presence of one or more competitor job(s), and J_a is the job completion time when the job (j) in executed alone in the same GPU, keeping all other computing resources alike. Packing Saving (Pac_{Sav}) refers to the percent difference in JCT when multiple jobs are packed together into the same GPU for concurrent execution, compared to each executed sequentially. Equation 2 expresses the PAC_{Sav} for n jobs, $j_1, ..., j_n$, executed concurrently on the same GPU, where M is maximum function.

$$Ind_{SD}(j)\% = \frac{J_c(j) - J_a(j)}{J_a(j)} \tag{1}$$

$$Pac_{Sav}(j_1, .., j_n)\% = \frac{\sum_{x=j_1}^{j_n} J_a(x) - M[J_c(j_1), ..., J_c(j_n)]}{\sum_{x=j_1}^{j_n} J_a(x)} \tag{2}$$

When a new user-defined job is made available in the job queue, the *JP* profiles the job for the target and predictor variables, and the *KB* component assigns it to a pre-existing cluster. The *JRS*, based on the prior knowledge of inter- and intra-cluster task interference results, recommends the best-suited competitor job from the job queue that would meet the desired scheduler-level objectives. It also considers the uncertainty score of jobs in the cluster by providing selection precedence to jobs with a lower uncertainty score. The scheduler-level objectives vary based on the prioritization of the user and cloud provider demands. Ideally speaking, the user aims to achieve the least Ind_{SD}, while the cloud provider aims to achieve the maximum Pac_{Sav}. To meet the two conflicting objectives while encouraging multi-tenancy for GPU sharing in compute clusters, we propose three algorithms, ⓐ one that balances Ind_{SD} and Pac_{Sav}, ⓑ one that minimizes Ind_{SD}, and finally, ⓒ one that maximizes Pac_{Sav}. *JRS* provides scheduling recommendations to the kube-scheduler for the concurrent execution of two competitor jobs that would satisfy the scheduler-level objectives. The recommended jobs are scheduled through the K8s' Argo workflow framework. The scheduler-level objectives are defined as follows:

- **Balancing between $Ind_{SD}(j)$ and Pac_{Sav}:** Based on the dynamic task interference analysis, *JRS* is equipped with the relative rank of inter- and intra-cluster task interference for $Ind_{SD}(j)$ and Pac_{Sav}, which is updated post every concurrent execution. From the user's perspective, minimizing $Ind_{SD}(j)$ for training user-defined ML job, j, remains the primary objective. Alongside, from the cloud provider perspective, scheduling several jobs concurrently on the same GPU would promote multi-tenancy and provide room for optimal use of available computing resources. Maximization of Pac_{Sav}, through concurrent training of n user-defined ML workloads, while minimizing $\forall j\ Ind_{SD}$ is the primary motivation of the proposed algorithm. To meet the conflicting interests, based on the relative ranking of inter- and intra-cluster task interference analysis for $Ind_{SD}(j)$ and Pac_{Sav}, the *JRS* recommends two competitor jobs that balance both parties.
- **Emphasizing $Ind_{SD}(j)$:** Ideally speaking, scheduling every user-defined ML training job on a single GPU would bring down $Ind_{SD}(j)$ to zero. However, that would incur significant computing costs for the cloud providers. Multi-tenancy encourages concurrently scheduling multiple training jobs on the same computing resources. Thus, selecting the competitor job is essential as the Ind_{SD} of the user-defined job j primarily depends on it. Through the relative ranking of inter- and intra-cluster task interference analysis, *JRS* intelligently chooses competitor jobs that would minimize Ind_{SD} for user-defined job j and recommend it to the kube-scheduler to meet this scheduler-level objective.
- **Emphasizing Pac_{Sav}:** Ideally speaking from a cloud provider's perspective, concurrently scheduling several user-defined ML training jobs would help maximize Pac_{Sav}. However, the relative order for inter- and intra-cluster task interference for packing saving has a high variance. We propose scheduling jobs based on a bin-packing strategy to negate high variance in Pac_{Sav} analysis. For every user-defined ML training job j, the *JP* extracts $Max.GPU_{util}$ and $Max.GPU_{memAlloc}$. We propose to pack jobs from the job queue based on $Max.GPU_{util}$, or $Max.GPU_{memAlloc}$ for a given GPU, till its full capacity. The rest of the jobs in the job queue can be scheduled based on the relative inter- and intra-cluster ranking of Pac_{Sav} metric.

For each of the three scheduler-level objectives, to preserve user privacy, we avoid taking into account any sensitive hyper-parameters of the user-defined ML training task, j, like batch size, epochs, the dataset used, and model architecture. These are considered influential in analyzing the computing resource requirements, thereby promoting a selection of the competitor job to address the scheduling objective. Our design also provides selection precedence for jobs with lower uncertainty

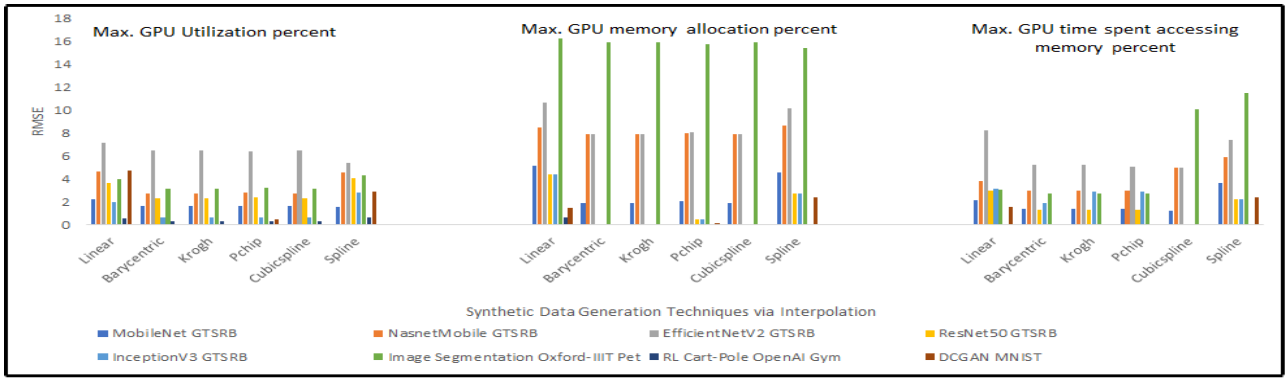

Fig. 2. Quality of synthetic data generation for every ML workload, for varying batch size, for all three selected features, with respect to the interpolation techniques. Technique that yielded the lowest RMSE was selected for synthetic data generation.

scores within a cluster. Moreover, we introduce controlled randomized intra-cluster exploration by introducing an exploration factor ϵ. ϵ represents the probability of negating the precedence given to uncertainty score for scheduling decisions. Our experimental evaluations show improvement in results upon introducing ϵ.

III. Evaluations

A. ML Workload Profiling

We profiled five ML training workloads from categories of image classification, image segmentation, Reinforcement Learning (RL), Generative Adversarial Networks (GAN), and Natural Language Processing (NLP) domain. ML training workloads on architectures of MobileNet [15], EfficientNetV2 [16], ResNet-50 [17], InceptionV3 [18], and NasnetMobile [19] on German Traffic Sign Recognition Benchmark (GTSRB) dataset [20] were profiled for image classification. We profiled ML training workload for image segmentation on Oxford-IIIT Pet dataset [21] using MobileNet architecture [15]. For generative models, we profiled Deep Convolutional Generative Adversarial Network (DCGAN) [22] training script on MNIST dataset [23]. For RL jobs, we trained a deep Q-network model on OpenAI's gym environment for balancing the pole on a cart with all motions restricted to one

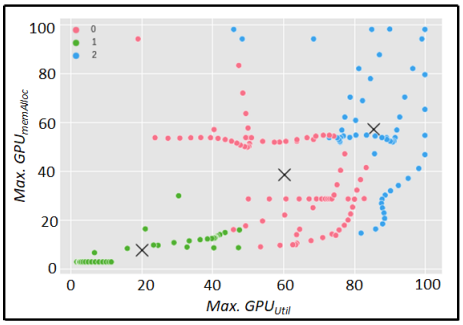

Fig. 3. Clustering of all ML workloads with respect to $Max.GPU_{util}$, $Max.GPU_{memAlloc}$, and $Max.GPU_{timeAccMemPer}$. The black cross represents the cluster center. Further a workload is from its assigned cluster center, higher its associated uncertainty score.

dimension. For NLP models, we fine-tuned deep bidirectional transformer BERT [24] models on the large IMDb movie review dataset [25] for sentiment analysis.

Profiling was done on Google Pro platform [26], using Weights and Biases [27], an MLOps tool, on Tesla T4 GPUs. We collected sixty-six features ranging from model hyperparameters, CPU and GPU utilization, memory utilization, disk usage, network configurations, kernel logs, and several others. The comprehensive dataset, with forty-nine ML training workloads, and the training scripts are released publicly on our GitHub repository [28].

B. Feature Selection

The chief limiting factor in allocating multiple ML training jobs on the same GPU within the compute clusters is the maximum GPU utilization ($Max.GPU_{util}$) and maximum GPU memory allocation percent ($Max.GPU_{memAlloc}$). Therefore, we performed feature selection on our target variables through Pearson's feature correlation metric and feature extraction through random forests, ridge regression, and lasso regression. The most significant predictor out of the sixty-six features was maximum GPU time spent accessing memory percent ($Max.GPU_{timeAccMemPer}$). The feature selection scripts are available publicly on our GitHub repository [28]. A comprehensive view of the feature ranking for $Max.GPU_{memoryAlloc}$ over the feature selection methods is available in our GitHub repository [28]. To meet the scheduling level objectives, the *JP* profiles for $Max.GPU_{util}$, $Max.GPU_{memAlloc}$, and $Max.GPU_{timeAccMemPer}$ for every user-defined ML training workload, ensuring a black-box access to the same, which is forwarded to the *KB* for further processing (assigning the job to a pre-existing cluster).

C. Generating Synthetic Workloads and Clustering

Synthetic Workload Generation: We amplify our dataset five times with respect to the three selected features, namely, $Max.GPU_{util}$, $Max.GPU_{memAlloc}$, and $Max.GPU_{timeAccMemPer}$. Based on the proposed methodology of synthetic workload generation as presented in Section II, we generate synthetic data from the ground truth

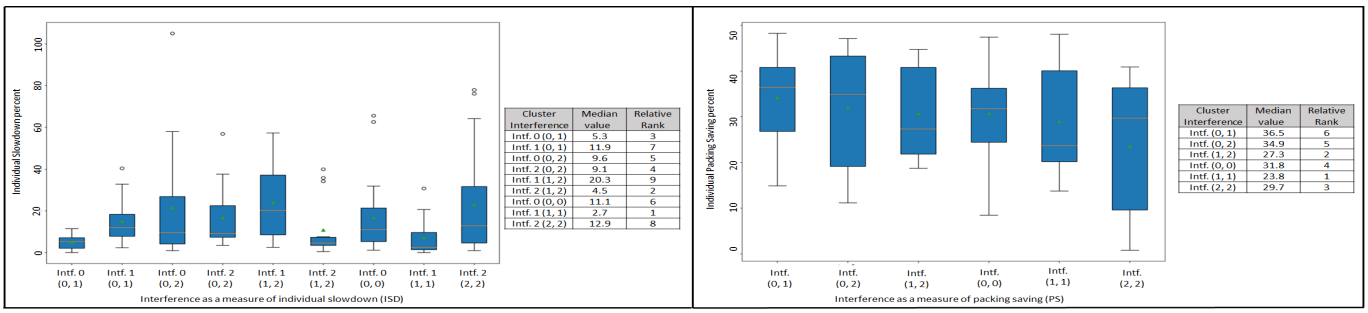

Fig. 4. The relative ranking of inter- and intra-cluster task interference for $Ind_{SD}(j)$ and Pac_{Sav} metrics. On the left, Intf.0(0,1) represents the interference of a job from cluster 0 when concurrently scheduled with a job from cluster 1, on the same GPU. On the right, Intf.(0,1) represents the interference when jobs from cluster 0 and 1 are concurrently scheduled on the same GPU.

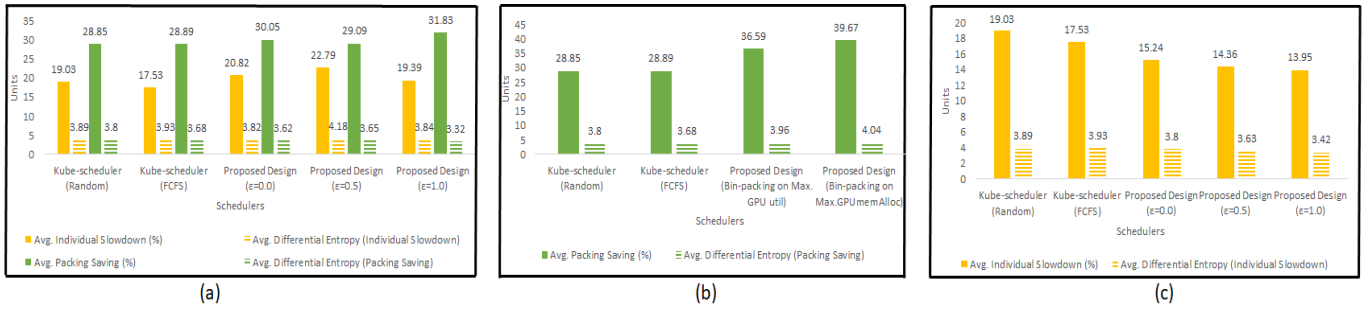

Fig. 5. Results show the improvement in scheduler-level objectives for our three proposed designs, in comparison to the kube-scheduler making random or First Come First Serve (FCFS) decision to promote multi-tenancy. Differential entropy reflects the uncertainty in the outcomes.

GT_{real}. To evaluate the quality of the synthetic samples, we blind the ground truth. We attempt to re-generate the ground truth using the generated synthetic data as the pseudo-ground truth. The Root Mean Square Error (RMSE) between the re-generated ground truth and ground truth helps us evaluate the regeneration quality. Fig. 2 graphically represents the RMSE for all the synthetic data generation techniques.

Clustering: We cluster the ML training workloads, GT_{real}, and the synthetic samples through the unsupervised k-means approach. Based on the elbow function and distortion score of the k-means algorithm, featuring $Max.GPU_{util}$, $Max.GPU_{memAlloc}$, and $Max.GPU_{timeAccMemPer}$, we cluster the jobs into groups. Depending on the distance of the job from its assigned cluster center, an uncertainty score is assigned to the same. Fig. 3 shows the clustering results. The *KB* contains a privacy-preserving record of all user-defined ML training workloads. In online mode, based on the profiling results from *JP*, a job is assigned to a pre-existing cluster in *KB* along with an uncertainty score.

D. Task Interference Analysis

We ran extensive experiments to analyze the intra- and inter-cluster task interference for various workloads on V100 GPU, at *Futurewei Technologies*. We randomly select an array of ML training workloads (from the list of workloads presented in Section III A), profile them using *JP*, and cluster them with an uncertainty score in the *KB*. Next, we schedule two ML jobs concurrently on the same GPU via Argo workflow to analyze the inter- and intra-cluster ML task interference on V100 GPU. Table II accounts for the inter-cluster task interference results, while Table III accounts for the intra-cluster task interference results. We analyze interference with regards to our defined metrics of $Ind_{SD}(j)$ and Pac_{Sav}. The relative ranking of inter- and intra-cluster task interference is retained in the *JRS* component. The relative ranking is updated dynamically whenever a set of workloads completes concurrent execution in the GPU. To meet scheduler-level objectives, the *JRS* component recommends competitor jobs from the job queue to the kube-scheduler for concurrent execution in the hardware accelerators, scheduled via Argo workflow. Fig. 4 shows the relative ranking of the clusters with respect to our defined task interference metrics at a given time instance.

E. Recommendation Service for Scheduling ML Training Jobs

The *JRS*, which maintains the relative ranking for task interference, recommends the kube-scheduler for competitor jobs from the user-defined jobs in queue to meet scheduler-level objectives. We randomly select 24 ML training workloads, along with introducing autoencoder models. Table I shows the 15 unique workloads, which are varied with respect to their hyper-parameters to simulate the user-defined job queue at a given instance of time. The *JP* profiles the job in the worker node. The *KB* assigns these jobs to pre-existing cluster along with an uncertainty score for each of them. The *JRS*, depending on the scheduler-level objectives, recommends competitor jobs to the kube-scheduler. The kube-scheduler places them in nodes, and the concurrent scheduling is facilitated by Argo workflow.

- **Balancing between $Ind_{SD}(j)$ and Pac_{Sav}:** In comparison to the kube-scheduler taking a random or first-

come-first-serve approach to promote multi-tenancy, decisions influenced by our method are superior in terms of $Ind_{SD}(j)$ and Pac_{Sav}, as reflected in Fig. 5(a). Table VII shows the concurrent workloads used for performance testing of our proposed design. Balancing out the conflicting goals in our proposed design is further boosted by introducing ϵ.

- **Emphasizing $Ind_{SD}(j)$:** In comparison to the kube-scheduler taking a random or first-come-first-serve approach to promote multi-tenancy, decisions influenced by our method minimize $Ind_{SD}(j)$ by **3.79%** and **2.29%** respectively, in terms of $Ind_{SD}(j)$, as reflected in Fig. 5(c). Table IV shows the concurrent workloads used for performance testing of our proposed design. Our proposed design further minimizes $Ind_{SD}(j)$ by **5.08%** by introducing ϵ.
- **Maximizing Pac_{Sav}:** In comparison to the kube-scheduler taking a random or first-come-first-serve approach to promote multi-tenancy, decisions influenced by our method is **7.74%** and **7.70%** superior in terms of Pac_{Sav} respectively, for bin-packing via $Max.GPU_{util}$, as reflected in Fig. 5(b). Tables V and VI show the concurrent workloads used for performance testing of our proposed design. Our proposed design is further boosted to an improvement of **10.82%** for bin-packing via $Max.GPU_{memAlloc}$.

We also present the associated differential entropy to account for certainty in the outcomes' certainty. A lower differential entropy is favourable.

IV. Conclusion

We design and test the performance of a fine-grain, online, privacy-preserving job scheduler, built on top of Kubernetes, harnessing Argo workflow that promotes multi-tenancy through intelligently selecting competitor jobs for concurrent execution in every pod. The three scheduler-level objectives caters to the interests of both the user and cloud providers, all while preserving user privacy. In future work, we aim to work upon exploring designs of bin packing algorithms to maximize Pac_{Sav} while further minimizing resource fragmentation on hardware accelerators.

TABLE I
Machine Learning Workloads for performance testing

Category	Task ID	ML Training Job
Image Classification	J1	MobileNet on GTSRB
	J2	EfficientNetV2 on GTSRB
	J3	ResNet50 on GTSRB
	J4	InceptionV3 on GTSRB
	J5	NasneMobile on GTSRB
Image Segmentation	J6	MobileNet on Oxford-IIIT Pet Dataset
Generative Adversarial Networks	J7	DCGAN on MNIST
Autoencoders	J8	De-noising Autoencoder on CIFAR-10
	J9	De-noising Autoencoder on MNIST
	J10	De-noising Autoencoder on F-MNIST
Natural Language Processing	J11	Fine-tuning pre-trained Small BERT L2-H256-A4 model on IMDb movie review dataset
	J12	Fine-tuning pre-trained Small BERT L2-H128-A2 model on IMDb movie review dataset
	J13	Fine-tuning pre-trained Small BERT L2-H768-A12 model on IMDb movie review dataset
	J14	Fine-tuning pre-trained Small BERT L4-H256-A4 model on IMDb movie review dataset
Reinforcement Learning	J15	Cart-pole

TABLE II
Inter Cluster Task Interference for ML Training Workloads

Task	Cluster ID	Job Completion Time (Secs)	Individual Slow Down	Packing Saving
MobileNet on GTSRB (*BS*=1024, *Epoch*=5) + DCGAN on MNIST (*BS*=230, *Epoch*=20)	0, 1	260.71, 292.14	7.5%, 12.3%	41.8%
NasnetMobile on GTSRB (*BS*=409, *Epoch*=5) + DCGAN on MNIST (*BS*=90, *Epoch*=20)	0, 1	268.26, 574.80	0.04%, 6.7%	28.6%
InceptionV3 on GTSRB (*BS*=20, *Epoch*=5) + RL (*BS*=128, *Epoch*=5000)	0, 1	317.16, 107.16	1.5%, 30.2%	19.6%
NasnetMobile on GTSRB (*BS*=154, *Epoch*=5) + Small BERT L2-H128-A2 (*BS*=256, *Epoch*=1)	0, 1	279.10, 270.72	1.1%, 2.2%	48.3%
NasnetMobile on GTSRB (*BS*=230, *Epoch*=5) + DCGAN on MNIST (*BS*=180, *Epoch*=20)	0, 1	276.87, 383.06	4.8%, 12.4%	36.6%
MobileNet on GTSRB (*BS*=869, *Epoch*=5) + DCGAN on MNIST (*BS*=154, *Epoch*=20)	0, 1	251.08, 392.68	5.8%, 3.1%	36.4%
Image Segmentation (*BS*=32, *Epoch*=20) + RL (*BS*=205, *Epoch*=5000)	0, 1	92.40, 104.88	11.4%, 14.6%	39.8%
Image Segmentation (*BS*=8, *Epoch*=20) + RL (*BS*=230, *Epoch*=5000)	0, 1	163.06, 114.38	2.1%, 32.8%	33.6%
MobileNet on GTSRB (*BS*=562, *Epoch*=5) + RL (*BS*=180, *Epoch*=5000)	0, 1	258.84, 93.50	7.1%, 8.2%	21.1%
Image Segmentation (*BS*=70, *Epoch*=20) + RL (*BS*=51, *Epoch*=5000)	0, 1	92.18, 109.56	10.6%, 40.4%	32.0%
MobileNet on GTSRB (*BS*=1024, *Epoch*=5) + ResNet50 on GTSRB (*BS*=1024, *Epoch*=5)	0, 2	253.99, 258.79	4.8%, 9.1%	43.3%
NasnetMobile on GTSRB (*BS*=409, *Epoch*=5) + InceptionV3 (*BS*=230, *Epoch*=5)	0, 2	269.87, 275.58	1.0%, 8.4%	47.1%
NasnetMobile on GTSRB (*BS*=154, *Epoch*=5) + Small BERT L4-H512-A8 (*BS*=256, *Epoch*=1)	0, 2	285.18, 318.65	3.3%, 5.2%	44.9%
NasnetMobile on GTSRB (*BS*=230, *Epoch*=5) + EfficientNetV2 on GTSRB (*BS*=256, *Epoch*=5)	0, 2	287.42, 3766.11	8.8%, 37.6%	29.9%
Image Segmentation (*BS*=32, *Epoch*=20) + InceptionV3 on GTSRB (*BS*=409, *Epoch*=5)	0, 2	118.16, 273.76	42.4%, 7.1%	19.1%
InceptionV3 on GTSRB (*BS*=20, *Epoch*=5) + Small BERT L2-H768-A12 (*BS*=256, *Epoch*=1)	0, 2	348.13, 319.64	11.4%, 5.7%	43.3%
MobileNet on GTSRB (*BS*=562, *Epoch*=5) + EfficientNetV2 on GTSRB (*BS*=562, *Epoch*=5)	0, 2	258.61, 318.69	6.9%, 15.7%	38.3%
Image Segmentation(*BS*=8, *Epoch*=20) + MobileNet on GTSRB (*BS*=869, *Epoch*=5)	0, 2	175.99, 258.75	10.2%, 9.1%	34.7%
EfficientNetV2 on GTSRB (*BS*=109, *Epoch*=5) + ResNet50 on GTSRB (*BS*=154, *Epoch*=5)	0, 2	472.88, 410.05	58.1%, 56.8%	15.6%
Image Segmentation (*BS*=70, *Epoch*=20) + ResNet-50 on GTSRB (*BS*=180, *Epoch*=5)	0, 2	170.98, 291.21	105.1%, 12.7%	14.7%
DCGAN on MNIST (*BS*=230, *Epoch*=20) + EfficientNetV2 on GTSRB (*BS*=109, *Epoch*=5)	1, 2	372.22, 406.37	43.1%, 35.8%	27.3%
DCGAN on MNIST (*BS*=230, *Epoch*=20) + EfficientNetV2 on GTSRB (*BS*=109, *Epoch*=5)	1, 2	372.22, 406.37	43.1%, 35.8%	27.3%
DCGAN on MNIST (*BS*=90, *Epoch*=20) + Small BERT L4-H512-A8 (*BS*=256, *Epoch*=1)	1, 2	552.29, 306.56	2.6%, 1.2%	34.3%
DCGAN on MNIST (*BS*=180, *Epoch*=20) + EfficientNetV2 on GTSRB (*BS*=256, *Epoch*=5)	1, 2	446.90, 382.49	%31.1, 39.9%	27.2%
Small BERT L2-H128-A2 (*BS*=256, *Epoch*=1) + ResNet-50 on GTSRB (*BS*=154, *Epoch*=5)	1, 2	296.76, 281.19	12.1%, 7.5%	43.5%
DCGAN on MNIST (*BS*=154, *Epoch*=20) + InceptionV3 on GTSRB (*BS*=230, *Epoch*=5)	1, 2	425.46, 270.28	11.6%, 6.3%	33.0%
RL (*BS*=180, *Epoch*=5000) + Small BERT L2-H768-A12 (*BS*=256, *Epoch*=1)	1, 2	91.94, 303.74	6.4%, 0.5%	21.8%
RL (*BS*=128, *Epoch*=5000) + EfficientNetV2 on GTSRB (*BS*=562, *Epoch*=5)	1, 2	129.54, 287.80	57.3%, 4.5%	19.4%
RL (*BS*=230, *Epoch*=5000) + InceptionV3 on GTSRB (*BS*=409, *Epoch*=5)	1, 2	130.56, 266.97	51.6%, 4.4%	21.8%
RL (*BS*=205, *Epoch*=5000) + Resnet-50 on GTSRB (*BS*=1024, *Epoch*=5)	1, 2	136.20, 280.88	48.8%, 7.2%	20.5%
RL (*BS*=51, *Epoch*=5000) + ResNet-50 on GTRSB (*BS*=180, *Epoch*=5000)	1, 2	100.15, 273.06	28.3%, 5.7%	18.7%

TABLE III
INTRA CLUSTER TASK INTERFERENCE FOR ML TRAINING WORKLOADS

Task	Cluster ID	Job Completion Time (Secs)	Individual Slow Down	Packing Saving
MobileNet on GTSRB (*BS*=1024, *Epoch*=5) + MobileNet on GTSRB (*BS*=869, *Epoch*=20)	0, 0	253.81, 253.05	4.7%, 6.7%	47.1%
MobileNet on GTSRB (*BS*=562, *Epoch*=5) + Image Segmentation (*BS*=8, *Epoch*=20)	0, 0	254.39, 182.70	5.2%, 14.5%	36.6%
NasnetMobile on GTSRB (*BS*=409, *Epoch*=5) + InceptionV3 on GTSRB (*BS*=20, *Epoch*=5)	0, 0	270.65, 394.90	1.2%, 26.3%	31.8%
NasnetMobile on GTSRB (*BS*=230, *Epoch*=5) + Image Segmentation (*BS*=32, *Epoch*=20)	0, 0	278.34, 99.59	5.4%, 20.0%	19.7%
NasnetMobile on GTSRB (*BS*=154, *Epoch*=5) + Image Segmentation (*BS*=109, *Epoch*=20)	0, 0	322.51, 101.12	16.8%, 31.8%	8.5%
Image Segmentation (*BS*=70, *Epoch*=20) + NasnetMobile on GTSRB (*BS*=230, *Epoch*=5)	0, 0	100.05, 287.07	20.1%, 7.6%	18.2%
DCGAN on MNIST (*BS*=230, *Epoch*=20) + DCGAN on MNIST (*BS*=90, *Epoch*=20)	1, 1	296.67,588.42	14.1%, 9.3%	26.2%
RL (*BS*=230, *Epoch*=5000) + DCGAN on MNIST (*BS*=180, *Epoch*=20)	1, 1	112.62, 340.66	30.8%, 0.2%	19.9%
RL (*BS*=205, *Epoch*=5000) + RL (*BS*=51, *Epoch*=5000)	1, 1	92.61, 80.29	1.2%, 2.9%	45.3%
DCGAN on MNIST (*BS*=154, *Epoch*=20) + Small BERT L2-H128-A2 (*BS*=256, *Epoch*=1)	1, 1	372.88, 290.24	0.3%, 9.6%	40.8%
RL (*BS*=180, *Epoch*=5000) + RL (*BS*=128, *Epoch*=5000)	1, 1	84.26, 87.6	1.4%, 0.5%	48.1%
EfficientNetV2 on GTSRB (*BS*=562, *Epoch*=5) + ResNet-50 on GTSRB (*BS*=1024, *Epoch*=5)	2, 2	452.29, 424.10	64.3%, 61.9%	15.7%
EfficientNetV2 on GTSRB (*BS*=256, *Epoch*=5) + EfficientNetV2 on GTSRB (*BS*=109, *Epoch*=5)	2, 2	486.44, 526.50	77.9%, 76.1%	0.8%
InceptionV3 on GTSRB (*BS*=409, *Epoch*=5) + ResNet-50 on GTSRB (*BS*=180, *Epoch*=5)	2, 2	286.54, 360.76	12.1%, 39.7%	29.7%
ResNet-50 on GTSRB (*BS*=154, *Epoch*=5) + Small BERT L4-H512-A8 (*BS*=256, *Epoch*=1)	2, 2	304.51, 357.79	16.4%, 18.1%	36.6%
InceptionV3 on GTSRB (*BS*=230, *Epoch*=5) + Small BERT L2-H768-A12 (*BS*=256, *Epoch*=1)	2, 2	279.61, 328.48	9.9%, 8.7%	40.9%

TABLE IV
PERFORMANCE TESTING: MINIMIZING INDIVIDUAL SLOWDOWN, $\epsilon = 0.0$

Task	Cluster ID	Dist. from cluster center	JCT (Secs)	ISD (%)	PS (%)
J11 (*BS*=256, *Epoch*=1) + J7 (*BS*=205, *Epoch*=20)	1, 1	133.07. 133.11	282.12, 292.51	0.2%, 3.9%	47.9%
J7 (*BS*=109, *Epoch*=20) + J7 (*BS*=180, *Epoch*=20)	1, 1	135.99, 134.28	533.61, 383.31	14.3%, 19.0%	32.3%
J12 (*BS*=256, *Epoch*=1) + J15 (*BS*=32, *Epoch*=5000)	1, 1	150.17, 198.35	261.29, 80.41	0.4%, 8.7%	21.7%
J15 (*BS*=562, *Epoch*=5000) + J15 (*BS*=716, *Epoch*=5000)	1, 1	208.97, 209.15	158.81, 188.69	3.1%, 0.7%	44.6%
J1 (*BS*=205, *Epoch*=5) + J4 (*BS*=409, *Epoch*=55)	0, 2	114.8, 122.85	287.27, 275.05	12.3%, 5.9%	44.2%
J6 (*BS*=230, *Epoch*=20) + J3 (*BS*=230, *Epoch*=5)	0, 2	116.2, 140.82	135.11, 313.33	70.7%, 20.6%	7.5%
J5 (*BS*=562, *Epoch*=5) + J6 (*BS*=716, *Epoch*=20)	0, 2	123.82, 169.33	304.29, 92.45	15.7%, 16.7%	11.0%
J8 (*BS*=256, *Epoch*=50) + J14 (*BS*=256, *Epoch*=1)	0, 2	124.15, 207.51	136.90, 283.88	2.6%, 5.5%	29.4%
J7 (*BS*=1024, *Epoch*=20) + J2 (*BS*=716, *Epoch*=5)	0, 2	125.03, 212.72	166.99, 354.50	41.4%, 28.6%	9.9%
J8 (*BS*=1024, *Epoch*=50) + J4 (*BS*=869, *Epoch*=5)	0, 2	131.74, 234.35	120.77, 305.94	20.4%, 19.1%	14.3%
J10 (*BS*=256, *Epoch*=50) + J13 (*BS*=256, *Epoch*=1)	0, 2	134.81, 237.52	98.08, 291.03	5.2%, 0.2%	24.1%
J9 (*BS*=256, *Epoch*=50) + J3 (*BS*=1024, *Epoch*=5)	0, 2	135.31, 245.53	128.12, 313.52	32.2%, 18.2%	13.3%

TABLE V
PERFORMANCE TESTING: MAXIMIZING PACKING SAVING. BIN PACKING VIA PROPORTIONAL RESOURCE DISTRIBUTION ON $Max.GPU_{util}$, FOLLOWED BY *JRS* RECOMMENDATIONS BASED ON CLUSTERING.

Task	Cluster ID	Distance from cluster center	Job Completion Time (Secs)	Individual Slow Down	Packing Saving
J15(*BS*=716, *Epoch*=5000) + J15(*BS*=562, *Epoch*=5000) + J15(*BS*=32, *Epoch*=5000) + J12(*BS*=256, *Epoch*=1) + J11(*BS*=256, *Epoch*=1)	NA	NA	189, 167, 91, 281, 285	1.5%, 9.1%, 24.6%, 8.1%, 1.2%	70.1%
J7(*BS*=205, *Epoch*=20) + J7(*BS*=180, *Epoch*=20)	1, 1	133.11, 134.28	331.24, 329.24	17.7%, 2.5%	45.1%
J7(*BS*=109, *Epoch*=20) + J4(*BS*=409, *Epoch*=5)	1, 2	135.99, 122.85	520.04, 265.64	11.3%, 2.3%	28.4%
J3(*BS*=230, *Epoch*=5) + J6(*BS*=716, *Epoch*=20)	2, 2	140.82, 169.33	267.21, 82.14	2.9%, 3.7%	21.1%
J14(*BS*=256, *Epoch*=1) + J2(*BS*=716, *Epoch*=5)	2, 2	207.51, 212.72	270.29, 287.93	0.4%, 4.4%	47.1%
J4(*BS*=869, *Epoch*=5) + J13(*BS*=256, *Epoch*=1)	2, 2	234.35, 237.52	271.93, 313.96	5.8%, 8.1%	42.6%
J1(*BS*=205, *Epoch*=5) + J6(*BS*=230, *Epoch*=20)	0, 0	114.8, 116.2	270.24, 99.97	5.6%, 26.3%	19.2%
J5(*BS*=562, *Epoch*=5) + J8(*BS*=256, *Epoch*=50)	0, 0	123.82, 124.15	273.01, 167.45	3.8%, 25.5%	31.1%
J7(*BS*=1024, *Epoch*=50) + J8(*BS*=1024, *Epoch*=50)	0, 0	125.03, 131.74	160.54, 150.94	35.9%, 50.5%	26.4%
J10(*BS*=256, *Epoch*=50) + J9(*BS*=256, *Epoch*=50)	0, 0	134.81, 135.31	122.71, 123.64	26.7%, 32.7%	34.9%

TABLE VI
PERFORMANCE TESTING: MAXIMIZING PACKING SAVING. BIN PACKING VIA PROPORTIONAL RESOURCE DISTRIBUTION ON $Max.GPU_{memAlloc}$, FOLLOWED BY *JRS* RECOMMENDATIONS BASED ON CLUSTERING.

Task	Cluster ID	Distance from cluster center	Job Completion Time (Secs)	Individual Slow Down	Packing Saving
J15 (*BS*=562, *Epoch*=5000) + J15 (*BS*=32, *Epoch*=5000) + J15 (*BS*=716, *Epoch*=5000) + J9 (*BS*=256, *Epoch*=50) + J10 (*BS*=256, *Epoch*=50) + J7 (*BS*=109, *Epoch*=20) + J7 (*BS*=180, *Epoch*=20) + J7 (*BS*=205, *Epoch*=20) + J8 (*BS*=256, *Epoch*=50)	NA	NA	248, 214, 257, 175, 167, 556, 365, 334, 187,	62.0%, 193.1%, 37.4%, 80.4%, 79.5%, 19.3%, 13.7%, 18.8%, 40.6%	69.2%
J11 (*BS*=256, *Epoch*=1) + J12 (*BS*=256, *Epoch*=1)	1, 1	13.07, 150.17	262.92, 238.19	3.5%, 1.7%	46.1%
J4 (*BS*=409, *Epoch*=5) + J3 (*BS*=230, *Epoch*=5)	2, 2	122.85, 140.33	264.50, 309.80	1.8%, 19.3%	40.3%
J6 (*BS*=716, *Epoch*=20) + J14 (*BS*=256, *Epoch*=1)	2, 2	169.33, 207.51	81.51, 272.01	2.9%, 1.1%	21.8%
J2 (*BS*=716, *Epoch*=5) + J4 (*BS*=869, *Epoch*=5)	2, 2	212.72, 234.35	278.90, 274.01	1.2%, 6.6%	47.6%
J13 (*BS*=256, *Epoch*=1) + J3 (*BS*=1024, *Epoch*=5)	2, 2	237.52, 245.53	295.13, 316.90	1.6%, 19.5%	42.9%
J1 (*BS*=205, *Epoch*=5) + J6 (*BS*=230, *Epoch*=20)	0, 0	114.8, 116.2	264.85, 82.37	3.5%, 4.1%	20.1%
J5 (*BS*=562, *Epoch*=5) + J7 (*BS*=1024, *Epoch*=20)	0, 0	123.82, 125.03	268.72, 121.72	2.2%, 3.1%	29.4%

TABLE VII
PERFORMANCE TESTING: BALANCING $Ind_{SD}(j)$ AND Pac_{Sav}, $\epsilon = 0.0$

Task	Cluster ID	Distance from cluster center	Job Completion Time (Secs)	Individual Slow Down	Packing Saving
J11 (*BS*=256, *Epoch*=1) + J4 (*BS*=409, *Epoch*=5)	1, 2	207.51, 122.85	288.91, 272.13	2.7%, 4.8%	46.5%
J7 (*BS*=205, *Epoch*=20) + J3 (*BS*=230, *Epoch*=5)	1, 2	133.11, 140.82	394.69, 342.09	40.3%, 31.7%	27.03%
J7 (*BS*=180, *Epoch*=20) + J6 (*BS*=716, *Epoch*=20)	1, 2	134.28, 169.33	361.85, 116.39	12.4%, 46.9%	9.7%
J7 (*BS*=109, *Epoch*=20) + J14 (*BS*=256, *Epoch*=1)	1, 2	135.99, 207.51	482.97, 289.89	3.4%, 7.7%	34.3%
J12(*BS*=256, *Epoch*=1) + J2 (*BS*=716, *Epoch*=5)	1, 2	150.17, 212.72	320.60, 316.84	23.2%, 14.9%	40.1%
J15 (*BS*=32, *Epoch*=5000) + J4 (*BS*=869, *Epoch*=5)	1, 2	198.65, 234.25	107.81, 265.76	45.8%, 3.4%	19.6%
J15 (*BS*=562, *Epoch*=5000) + J13 (*BS*=256, *Epoch*=1)	1, 2	208.97, 237.52	172.15, 293.24	11.8%, 0.96%	34.01%
J15 (*BS*=716, *Epoch*=5000) + J3 (*BS*=1024, *Epoch*=5)	1, 2	209.15, 245.53	256.16, 281.45	36.8%, 6.1%	37.7%
J1 (*BS*=205, *Epoch*=5) + J6 (*BS*=230, *Epoch*=20)	0, 0	114.8, 116.2	270.24, 99.97	5.6%, 26.3%	19.2%
J5 (*BS*=562, *Epoch*=5) + J8 (*BS*=256, *Epoch*=50)	0, 0	123.82, 124.15	273.01, 167.45	3.8%, 25.5%	31.1%
J7 (*BS*=1024, *Epoch*=20) + J8 (*BS*=1024, *Epoch*=50)	0, 0	125.03, 131.74	160.54, 150.94	35.9%, 50.5%	26.4%
J7 (*BS*=256, *Epoch*=50) + J10 (*BS*=256, *Epoch*=50)	0, 0	135.31, 134.81	122.71, 123.64	26.7%, 32.7%	34.9%

REFERENCES

[1] S. Jamil, M. J. Piran, and O.-J. Kwon, "A comprehensive survey of transformers for computer vision," 2022. [Online]. Available: https://arxiv.org/abs/2211.06004

[2] M. Treviso, T. Ji, J.-U. Lee, B. van Aken, Q. Cao, M. R. Ciosici, M. Hassid, K. Heafield, S. Hooker, P. H. Martins, A. F. T. Martins, P. Milder, C. Raffel, E. Simpson, N. Slonim, N. Balasubramanian, L. Derczynski, and R. Schwartz, "Efficient methods for natural language processing: A survey," 2022. [Online]. Available: https://arxiv.org/abs/2209.00099

[3] Y. Peng, "A survey on modern recommendation system based on big data," 2022. [Online]. Available: https://arxiv.org/abs/2206.02631

[4] T. Sakai and T. Nagai, "Explainable autonomous robots: A survey and perspective," 2021. [Online]. Available: https://arxiv.org/abs/2105.02658

[5] D. Mendoza, F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "Interference-aware scheduling for inference serving." New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3437984.3458837

[6] F. Yu, S. Bray, D. Wang, L. Shangguan, X. Tang, C. Liu, and X. Chen, "Automated runtime-aware scheduling for multi-tenant dnn inference on gpu," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[7] A. Dhakal, S. G. Kulkarni, and K. Ramakrishnan, "Gslice: controlled spatial sharing of gpus for a scalable inference platform," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 492–506.

[8] C. Tan, Z. Li, J. Zhang, Y. Cao, S. Qi, Z. Liu, Y. Zhu, and C. Guo, "Serving dnn models with multi-instance gpus: A case of the reconfigurable machine scheduling problem," *arXiv preprint arXiv:2109.11067*, 2021.

[9] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 220–233.

[10] X. Wu, H. Xu, and Y. Wang, "Irina: Accelerating dnn inference with efficient online scheduling," in *4th Asia-Pacific Workshop on Networking*, 2020, pp. 36–43.

[11] A. Dhakal, J. Cho, S. G. Kulkarni, K. Ramakrishnan, and P. Sharma, "Spatial sharing of gpu for autotuning dnn models," *arXiv preprint arXiv:2008.03602*, 2020.

[12] 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 *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI) 18)*, 2018, pp. 595–610.

[13] T. N. Le, X. Sun, M. Chowdhury, and Z. Liu, "Allox: compute allocation in hybrid clusters," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[14] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–37, 2022.

[15] 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," 2017. [Online]. Available: https://arxiv.org/abs/1704.04861

[16] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2019. [Online]. Available: https://arxiv.org/abs/1905.11946

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[18] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015. [Online]. Available: https://arxiv.org/abs/1512.00567

[19] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," 2017. [Online]. Available: https://arxiv.org/abs/1707.07012

[20] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural Networks*, vol. 32, pp. 323–332, 2012, selected Papers from IJCNN 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608012000457

[21] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, "Cats and dogs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[22] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.06434

[23] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805

[25] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, Jun. 2011, pp. 142–150. [Online]. Available: https://aclanthology.org/P11-1015

[26] "Google colaboratory," https://colab.research.google.com/, accessed: 2023-02-28.

[27] "Weights and biases," https://wandb.ai/site, accessed: 2023-02-28.

[28] "Alnair github," https://github.com/CentaurusInfra/alnair/tree/main/open-data/gpu-sharing, accessed: 2023-02-28.