Towards SLO-Compliant and Cost-Effective Serverless Computing on Emerging GPU Architectures

Vivek M. Bhasi
The Pennsylvania State University
vmb5204@psu.edu

Aakash Sharma
The Pennsylvania State University
abs5688@psu.edu

Rishabh Jain
The Pennsylvania State University
rishabh@psu.edu

Jashwant Raj Gunasekaran
Adobe Research
jgunasekaran@adobe.com

Ashutosh Pattnaik
Arm
ashutosh13@gmail.com

Mahmut Taylan Kandemir
The Pennsylvania State University
mtk2@psu.edu

Chita Das
The Pennsylvania State University
cxd12@psu.edu

Abstract

Serverless platforms are supporting an increasing variety of applications (apps). Among these, apps such as Machine Learning (ML) inference serving can benefit significantly from leveraging accelerators like GPUs. Yet, major serverless providers, despite having GPU-equipped servers, do not offer GPU support for their serverless functions. While recent works have attempted to bridge this gap, they are agnostic to the capabilities of new-generation GPUs, thereby, overlooking several performance optimization opportunities.

To address this, we leverage unique features of newer NVIDIA GPU architectures (specifically, their Multi-Instance GPU (MIG) and Multi-Process Service (MPS) capabilities) to devise a serverless framework, Protean, that can guarantee a higher degree of Service Level Objective (SLO) compliance than that offered by state-of-the-art works. Moreover, Protean also proposes to host its components on a combination of both on-demand (reliable) VMs and heavily discounted VMs to reduce costs to the end consumer, while offering high service availability. We extensively evaluate Protean using 22 ML inference workloads with real-world traces on an 8×A100 GPU cluster. Our results show that Protean significantly outperforms state-of-the-art works in terms of SLO compliance (up to ~93% more) and tail latency (up to 82% less), while reducing cost by up to 70%. We also maintain reasonable tail latencies (< 200 ms) for best effort requests.

CCS Concepts

• **Computer systems organization → Cloud computing**.

Keywords

serverless, heterogeneous, GPU, scheduling, spot instances, resource-management

MIDDLEWARE '24, December 2–6, 2024, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-0623-3/24/12…$15.00
https://doi.org/10.1145/3652892.3700760

ACM Reference Format:
Vivek M. Bhasi, Aakash Sharma, Rishabh Jain, Jashwant Raj Gunasekaran, Ashutosh Pattnaik, Mahmut Taylan Kandemir, and Chita Das. 2024. Towards SLO-Compliant and Cost-Effective Serverless Computing on Emerging GPU Architectures. In *25th International Middleware Conference (MIDDLEWARE '24), December 2–6, 2024, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3652892.3700760

1 Introduction

Over the years, serverless platforms have risen in popularity, with an increasing variety of apps being deployed on them [5, 51, 52, 87]. The adoption of serverless computing in apps such as Amazon Alexa skills [30], Facebook Messenger Bots [31], and Optical Character Recognition (OCR) [32], are not only testament to this, but are also indicative of the rising number of serverless use cases that are based on ML inference. Such ML inference apps are typically deployed in user-facing settings and hence, are administered under strict SLOs in terms of response time deadlines [56, 66, 94, 100]. Achieving a low tail latency is also critical for these apps [34, 36, 45, 46].

It is well known that such inference apps can benefit, performance-wise, by leveraging accelerators like GPUs, that are becoming increasingly prevalent in cloud datacenters [39, 88, 89, 101]. Despite this fact, major serverless providers do *not* currently offer GPU support for their serverless functions. This is likely due to the challenges of effective GPU sharing between fine-grained serverless tasks and the high demand for GPU-equipped instances. Recent works have attempted to address this through GPU-accelerated serverless functions [48, 53, 75, 81, 82, 86, 100]. However, as we will demonstrate in this paper, these frameworks fail to remain performant when deployed on the recent MIG-enabled NVIDIA GPU architectures (Ampere and Hopper [20, 23]) since:

• The majority of the aforementioned works are not designed to be SLO compliant. This includes frameworks like *Molecule* [48], which, currently, does not use any GPU spatial sharing technologies such as NVIDIA MPS [25] and hence, can suffer from queuing delays when serving workload batches, one after the other, on the GPU(s) via time sharing.

• Even works like *INFless* [100] and *Llama* [82], that use MPS to spatially share GPUs (and possibly improve SLO compliance), deteriorate in performance, since they *consolidate excessive* workload batches on individual GPUs, which leads to high job interference.

This is also driven by the fact that *INFless* and *Llama* are agnostic to the GPU's MIG capabilities that allow it to be physically partitioned into different GPU instances (GPU slices or simply, slices) across which workloads can be scheduled with performance isolation[1].

Note that addressing these challenges using both MPS and MIG is *non-trivial*: (i) GPUs must be *dynamically configured* in advance according to incoming requests, and (ii) requests should be scheduled across GPU slices to *effectively trade off* job interference arising due to co-location and performance degradation due to the slices having lesser resources (henceforth, known as the *resource deficiency effect*).

The above performance challenges faced by these works can also potentially impact customer costs [87]. In fact, given that serverless costs are tied to the cost of host VMs (offered by an IaaS provider) [103], these frameworks do not utilize heavily-discounted VM tiers (such as spot VMs [7, 8, 17]), which could make the offering more affordable. Such cost savings can be critical, given that GPU host VMs are typically more expensive than their CPU counterparts. However, since spot VMs are susceptible to evictions, hosting a serverless framework on them is challenging because such revocations can adversely affect performance.

To address the above concerns, we propose Protean, a highly SLO-compliant, cost-effective and GPU-enabled serverless framework that can be deployed on recent NVIDIA GPUs with minimal effort. To our knowledge, Protean is the *first* serverless framework that improves performance (especially SLO compliance) by leveraging *both* the MPS and MIG capabilities of GPUs. Protean uses these *architectural capabilities* to *intelligently configure GPUs* and schedule workloads in a *performance-aware* fashion across the slices so as to *trade off between job interference and resource deficiency*. Additionally, Protean also employs request batching and reordering to prioritize the service of requests with strict SLO targets while utilizing the GPU effectively. This is coupled with conservative container provisioning and keep-alive policies to minimize the effects of cold starts on performance.

As a cost saving measure, Protean proposes to leverage the short-running nature of GPU serverless workloads (< 1s [100]) and the buffer time afforded by spot eviction notifications (30-120s [7, 8, 17]) to employ spot VM offerings to host its components, as and when they are available. To prevent performance degradation due to revocations, Protean uses on-demand (reliable) VMs during spot VM unavailability. While similar cost optimizations have been proposed in traditional (CPU-based) serverless [103], Protean is the first GPU-enabled serverless framework to do so. Table 1 compares Protean against related works.

We perform an extensive evaluation of Protean on an 8×A100 GPU cluster with real world traces using 22 ML inference workloads in the domains of vision and language. Our results show that Protean significantly outperforms state-of-the-art works in terms of SLO compliance (up to ~93% more) and tail latency (up to 82% less), while reducing cost by up to 70%. We achieve these results while also maintaining reasonable tail latencies (< 200 ms) for requests with no explicit latency targets also.

[1]ElasticFlow [53], while designed for ML training, also only uses MPS, and would behave similarly to these schemes, if repurposed for inference.

Features	Prophet [40]	Atoll [95]	Harvest VM FaaS [103]	Kraken [36]	Llama [82]	Molecule [48]	INFless [100]	Cypress [35]	MISO [71]	GPUlet [42]	Aquatope [104]	KRISP [43]	ElasticFlow [53]	Protean
Serverless framework	✗	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✗	✓	✓
GPU-enabled	✓	✗	✗	✗	✓	✓	✓	✗	✓	✓	✗	✓	✓	✓
Overall SLO Compliance	✓	✓	✓	✓	✓	✗	✓	✓	✗	✓	✓	✗	✓	✓
Cost-effective	✗	✗	✓	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗	✓
Space-sharing via MPS	✓	✗	✗	✗	✓	✗	✓	✗	✗	✓	✗	✓	✗	✓
Space-sharing via MIG	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓
Intelligent GPU reconfiguration	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✓	✗	✓
Performance-aware GPU job scheduling	✓	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✓
High GPU resource utilization	✓	✗	✗	✗	✓	✗	✓	✗	✓	✓	✗	✓	✓	✓
Satisfactory tail latency	✓	✓	✓	✓	✗	✗	✓	✓	✗	✗	✓	✓	✗	✓

Table 1: Comparison of Protean against related works. The features listed here are *only promised* by the works (excluding Protean) and may not reflect in the results shown in our evaluation (Section 6).

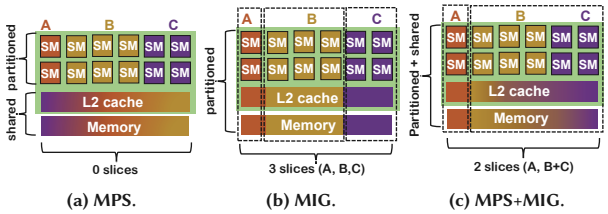
2 Context and Opportunities

This section sets the context for our work by examining the current state-of-the-art in GPU-enabled serverless frameworks and other relevant technologies. Through this, we arrive at the opportunities that underpin our work.

2.1 Serverless Computing on GPUs

In serverless computing, developers upload their code (composed of function(s)) to the serverless provider and have their functions invoked by events (such as user requests) to run them in sandbox environments (henceforth, containers will be the default) inside Virtual Machines (VMs) [38, 99]. Here, function execution may be preceded by a container bootup latency (called the cold start), which can take up to tens of seconds [3, 4]. Typically, the developers are billed only for the resources consumed during function execution. Serverless computing also mitigates resource management overheads for developers, while offering instantaneous scalability. These factors have led to serverless becoming a prime candidate for deploying latency-critical, user-facing apps on the cloud.

Why GPU serverless? Many latency-critical apps (especially ML inference apps) can achieve greater performance using specialized accelerators [49, 93] as well as datacenter GPUs [39, 89, 90]. With enough users, we envision a GPU serverless platform to receive enough requests to: (a) be performant, as a larger number of jobs can more effectively be served on GPUs versus CPUs [18] and (b) utilize GPUs more effectively, especially using new GPU spatial sharing features such as MPS and MIG. We believe this is realistic as existing inference serving frameworks such as the Azure ML Inference router [9] report receiving request rates up to 5k requests per second (rps) from users. Moreover, improving interconnect technologies continue to lower data movement costs of GPUs, thus, facilitating inference serving on them [22, 27]. Despite these reasons, major commercial serverless platforms including AWS Lambda [6], Microsoft Azure Functions [19] and Google Cloud Functions [16] do not offer GPU support for their functions. Recent research works [37, 48, 50, 53, 67, 75, 81, 82, 86, 100], however, have attempted to remedy this by proposing various GPU-accelerated serverless frameworks. Note that, while there are also GPU-enabled serverless startups, such as *Banana* [10], *Beam* [12], *Replicate* [28], *Runpod* [29], *Cerebrium* [13], and *trainML* [33], their job scheduling techniques are not public, thus, preventing us from drawing comparisons against them.



(a) MPS. (b) MIG. (c) MPS+MIG.

Figure 1: GPU spatial sharing methods between apps A, B, C.

SLO compliance in GPU serverless: GPU-enabled inference systems have been employed in industrial settings with SLO (latency) targets [56, 64, 66, 80]. However, only one of the GPU serverless startups mentioned earlier, *Banana*, offers any SLO guarantees – though it is at least 13× worse than Protean [11]. Note that meeting SLO targets can be critical to the reliability and operational stability of user-facing apps and can make the serverless offering more appealing to the end user [1, 36, 65]. The majority of GPU serverless research works are also not designed to be SLO compliant. Even works like *INFless* [100] and *Llama* [82], which are designed for this objective, fail to meet SLOs when employed on newer GPU architectures, as we will demonstrate in this paper. This is primarily because such works fail to exploit the unique features afforded by new-generation GPUs, which are described in the next subsection.

2.2 GPU Resource Sharing

Although multiple apps can share one GPU via time sharing, the GPU can be severely under-utilized, especially when executing typical light-weight serverless tasks [71]. NVIDIA introduced the below two key capabilities to address this: **Multi-Process Service (MPS):** MPS [25], introduced from the Kepler-based NVIDIA GPUs, enables co-operative multi-process CUDA apps (such as A, B, and C in Figure 1) to be processed concurrently on the GPU via software-based spatial sharing. As shown in Figure 1a, MPS partitions the GPU compute units, Streaming Multiprocessors (SMs), into multiple partitions such that each partition is dedicated to a user app. However, the resulting co-location of apps is not interference-free because the memory bandwidth, caches and capacity are all shared between the concurrent MPS processes [14]. Note that *INFless* [100] and *Llama* [82] rely solely on this technique to enable concurrent task execution on GPUs.

Multi-Instance GPU (MIG): MIG [24], introduced in the recent NVIDIA Ampere architecture, allows GPUs to be partitioned (via hardware + software support) into a maximum of 7 GPU slices. MIG provides *performance isolation between apps* running on separate GPU slices (Figure 1b). This is because unlike MPS, which only partitions GPU SMs between apps, MIG also partitions GPU caches and memory, while isolating memory bandwidth per slice [24]. MIG can even be used with MPS to allow multiple apps to share each slice, further improving resource utilization. For example, apps B and C share a GPU slice with MPS, thus, getting exclusive SMs, but shared memory resources (Figure 1c).

However, MIG does have its limitations:

(1) As shown in Table 2, each MIG partition (called a *profile*) can have only a fraction of the total SMs and memory, thus, potentially degrading the performance of a task running on it (versus running it on the whole GPU (*7g*)). We refer to this as the *resource deficiency effect*. Note that any configuration with a combination of profiles is referred to as a *geometry*.

Slice	Compute fraction	Memory	Cache fraction	Max Count
7g.40gb ('7g')	Full	40 GB	Full	1
4g.20gb ('4g')	4/7	20 GB	4/8	1
3g.20gb ('3g')	3/7	20 GB	4/8	2
2g.10gb ('2g')	2/7	10 GB	2/8	3
1g.5gb ('1g')	1/7	5 GB	1/8	7

Table 2: Possible MIG instance profiles on an A100 GPU.

(2) Re-configuring the current MIG geometry requires all slices to be idle without any running processes. However, the resource partitioning among processes are adjusted dynamically in MPS without requiring an idle GPU.

Quantifying the tradeoffs: To appreciate the above discussion, we conduct an experiment comparing various schemes subjected to constant request traces of ML inference workloads using: (i) *Simplified DLA* [102] (500 rps, batch size: 128), and (ii) *AlBERT* [69] (6 rps, batch size: 4). The workloads run on a single A100 GPU. Here, 50% of requests (strict requests) have a strict SLO deadline of 3× the batch execution latency on *7g*, while the other 50% are Best Effort (BE) requests with no deadline. Henceforth, SLO compliance will refer to the percentage of strict requests meeting their SLO targets. The schemes evaluated are described below.

- ***No MPS or MIG*** executes each workload batch on the entire GPU using *only* time-sharing (similar to *Molecule* [48]).
- ***MPS Only*** spatially shares the entire GPU among all the workloads using MPS only (as in *INFless/Llama* [82, 100]).
- ***MIG Only*** uses MIG to create GPU slices and time-shares them among requests equally scheduled across them.
- ***MPS+MIG*** uses MIG GPU slices and spatially shares them (via MPS) among requests equally scheduled across them.
- ***'Smart' MPS+MIG*** is similar to *MPS+MIG*, except that it schedules strict and BE requests on separate slices, while also ensuring strict requests get the largest slice.

For this experiment, we select the (*4g, 3g*) geometry for all MIG-enabled schemes as they have the most resources and hence, yield the best possible performance for the requests. For instance, choosing a *1g* to schedule requests on instead of *3g* can lead to a slowdown of up to 4×.

While analyzing these results, note that the performance degradation due to interference is closely tied to the number and characteristics of all co-located tasks [40]. From Figure 2, we observe that *'Smart' MPS+MIG* has the highest SLO compliance (up to 98% more) and the least tail latency (up to 72% less). This is because it isolates the strict requests from the other (low priority) BE requests, thus, minimizing the job interference from them (by up to 83%).

For *Simplified DLA* (Figure 2a), all schemes except *No MPS or MIG*, and *MIG Only* are highly (99% or higher) SLO-compliant. This is because both *No MPS or MIG*, and *MIG Only* time share either the entire GPU or GPU slices, respectively, instead of spatially sharing them using MPS. Thus, they incur queuing overheads of up to 80%. Here, *MPS Only* suffers from higher (up to 6× as much) interference than the MIG-enabled schemes as the strict requests are co-located with all the BE requests. *MPS+MIG*, despite not fully isolating strict and BE requests, reduces the overall interference experienced by strict requests by evenly splitting the request batches among the available slices.

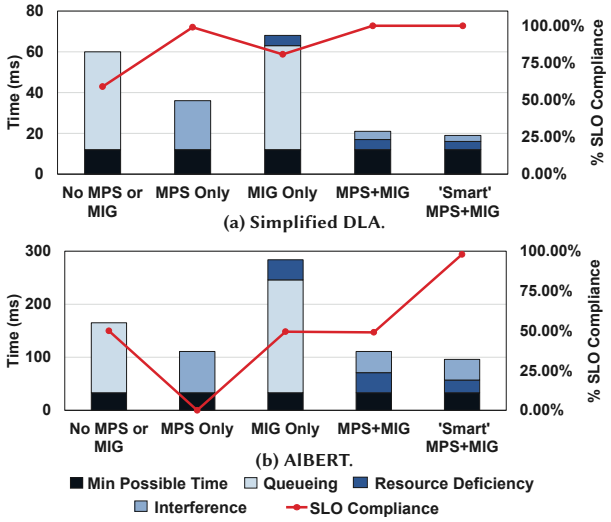


Figure 2: Breakdown of tail (P99) latencies vs. SLO compliance for the considered schemes. Here, 'Min possible time' refers to the execution time of a workload batch when run on *7g*.

For *AlBERT* (Figure 2b), most schemes except *'Smart' MPS+MIG* have worse performance than earlier. As we will see in Section 3, this is mainly due to the resource-intensive nature of *AlBERT*. In particular, *MPS Only* is the worst-affected by the interference due to co-locating all requests on the single GPU (taking up to 70% of the latency), thus, not satisfying the SLO for even a single request. While *MPS+MIG* has a similar P99 latency to that of *MPS Only*, it only suffers SLO violations for the strict requests that it schedules on the smaller slice. *No MPS or MIG*, despite its high tail latency, satisfies the SLO for strict requests that are earlier in the queue, since they execute on the whole GPU without interference from other requests. *MIG Only*, similarly, also does not suffer from interference as it time shares the GPU slices. However, *AlBERT* exacerbates resource deficiency effects, thus, increasing its batch execution time by 2.15× which, in turn, adversely affects its queuing overhead. *'Smart' MPS+MIG* performs the best (at both metrics) as it isolates strict requests on the largest slice, thus, minimizing the interference from BE requests (up to 30% versus others), while also reducing resource deficiency effects (up to 9%). *Note that 'Smart' MPS+MIG is only a straw man version of the framework we wish to implement.* This is because for the framework to remain performant for varying request rates and mixed workloads, (i) the requests will have to be scheduled with more thought, balancing the effects of resource deficiency and interference in a dynamic scenario, and (ii) the geometry will have to be changed dynamically, in advance, as well.

> ***Opportunity 1:*** *Intelligently scheduling requests using both MPS and MIG in a GPU-enabled serverless framework can significantly improve SLO compliance.*

2.3 The GPU 'Spot' Market

In addition to performance guarantees, affordable pricing can make a serverless platform more compelling to potential customers. Since serverless providers are required to provision, manage and pay its *IaaS provider* for the VMs that host its platform, the cost of the serverless platform is heavily influenced by that of the underlying VMs, as claimed in [103]. Fortunately, most IaaS providers offer their surplus resources as VMs at heavy discounts. One class of such VMs, which we will refer to as *Spot Instances/VMs* (from [7, 8, 17]), are those that trade off their lower cost for relaxed availability guarantees, i.e., they can be preempted/evicted at any time. From Table 3, we see that cost savings attainable from Spot VMs can go up to ~71% versus on-demand VMs.

IaaS Provider	On-Demand Price	Spot Price	Cost Savings
AWS	32.7726	9.8318	69.99%
Microsoft Azure	32.7700	18.0235	45.01%
Google Cloud	30.0846	8.8147	70.70%

Table 3: On-demand and spot hourly pricing (in dollars per hour) for an 8×A100 GPU instance for the three main IaaS providers (by marketshare [63]) averaged across the US-east and west regions (at the time of writing).

The challenge in leveraging Spot VMs is that they can be revoked at any time by the provider, thus potentially reducing the number of effective worker nodes available to service the incoming request load. This can adversely affect the system SLO compliance and tail latency. However, providers do notify the Spot VM users just before revocation (ranging from 30s to 120s in advance [7, 8, 17]). Since GPU serverless workloads typically run for less than a second [100], on receiving the revocation notification, we can potentially wait for all running requests on the Spot VM (in danger of eviction) to finish as a new VM (either on-demand or spot, depending on spot VM availability) is spun up to replace the soon-to-be-evicted VM. In this paper, we explore the scope of possible cost savings that can be achieved by employing such a system while it is subjected to spot VM evictions.

> ***Opportunity 2:*** *Employing both On-Demand and Spot VMs smartly can reduce costs significantly while ensuring high service availability and performance.*

3 Modeling Job Slowdown for Smart Scheduling

As seen in the previous section, isolating strict and BE requests can be integral to SLO-compliant request scheduling. However, it is still unclear as to what geometry to choose and where to place what requests, especially for dynamic scenarios. As the first step towards addressing this, we attempt to model job slowdowns (here 'job' is a generic term that can also refer to request batches) by considering two main causal factors: *job interference* and *resource deficiency*.

We *re-purpose* the interference model for co-located MPS jobs from *Prophet* [40] so as to model job slowdowns for a hybrid MPS-MIG scenario. Note that, unlike *Prophet*[40], which focuses on MPS interference modeling, our framework focuses on scheduling requests and dynamically reconfiguring the GPU(s) to maintain high performance using *both* MPS and MIG, while leveraging our repurposed slowdown model. In *Prophet's* interference model, concurrent MPS jobs are found to be slowed down mainly due to memory bandwidth contention. The execution time, T_k, of a job, J_k, co-located with other jobs, J_1, J_2,..J_n is given by:

$$T_k = Solo_k \times max\left\{bw_k \times sm_k + \sum_{i=1}^{n}\left(bw_i \times sm_i\right), 1\right\}. \quad (1)$$

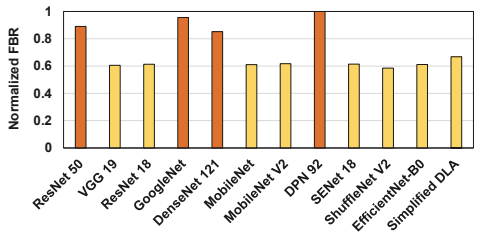


Figure 3: Normalized FBRs of inference-based workloads. Yellow and Orange bars correspond to LI and HI models, respectively.

Here, $Solo_k$ is the execution time of J_k when run in isolation on the concerned GPU, $(bw_j \times sm_j)$ is the Fractional Bandwidth Requirement (FBR) of job J_j, where bw_j represents its FBR per SM and sm_j is the number of SMs used by it. For example, an FBR of 0.2 indicates that the job requires 20% of the global memory bandwidth. We observe that $Solo_k$ is slowed down chiefly in proportion to $\sum_{i=1}^{n} (bw_i \times sm_i)$ for typical request rates and workloads [9, 82]. Note that the above discussion provides the rationale for isolating strict requests from BE ones on separate slices (Section 2.2): *strict requests will be slowed down in proportion to the overall contention for bandwidth on the slice.*

Next, the slices for request scheduling have to be chosen. This decision is influenced by *both* resource deficiency and job interference effects. From our experience, larger slices, owing to having larger fractions of total resources, tend to have the least job slowdowns due to resource deficiency, whereas smaller slices suffer the most slowdowns due to having lesser resources. This is also corroborated by prior work [71]. Since BE requests do not have a latency target, co-locating as many BE requests on as few, small slices as possible will not impact the overall SLO compliance despite the slowdown suffered by these requests. This can leave the larger slices in the selected geometry to be occupied by the strict requests. Therefore, the geometry should be chosen with this in mind.

> ***Guideline 1:*** *BE requests may be 'packed' onto the fewest, smallest slices with no effect on SLO compliance, thus, leaving larger slices for strict requests.*

While scheduling all strict requests on the largest slice may seem to be the ideal decision to minimize SLO violations, *there is a tradeoff between picking the largest slice, and load balancing jobs across slices to reduce interference.* Below, we formulate our equation that takes *both* these factors into account while determining job slowdown.

With knowledge of $Solo_j$ and $(bw_j \times sm_j)$ for the possible jobs, J_j, on all viable slices (Equation 1), we can estimate the job execution times on those slices. For the current job, J_k, consider the ratio of $Solo_k$ on the current slice to that on $7g$ to be the *Resource Deficiency Factor (RDF)*, i.e., $RDF = \frac{Solo_k^{current\_slice}}{Solo_k^{7g}}$. Therefore, we should ideally schedule a strict job on the slice with the minimum *slowdown factor*, η, where:

$$\eta = RDF \times max\left\{bw_k \times sm_k + \sum_{i=1}^{n} (bw_i \times sm_i), 1\right\}. \quad (2)$$

For example, an RDF value of 1.3 indicates that the 'solo time' of the job increases by 30% when run on the concerned slice. RDF can be calculated by finding the required ratio of execution times on the concerned slice. $(bw_j \times sm_j)$ (FBR) for any job J_j can also be estimated by averaging the values obtained from solving the linear equations derived from Equation 1 for multiple co-locations of J_j. Figure 3 depicts the normalized FBR values of some of our workloads. For the purposes of our later experiments, we categorize the workloads' models into *Low Interference* (LI) and *High Interference* (HI) models, based on the values in Figure 3.

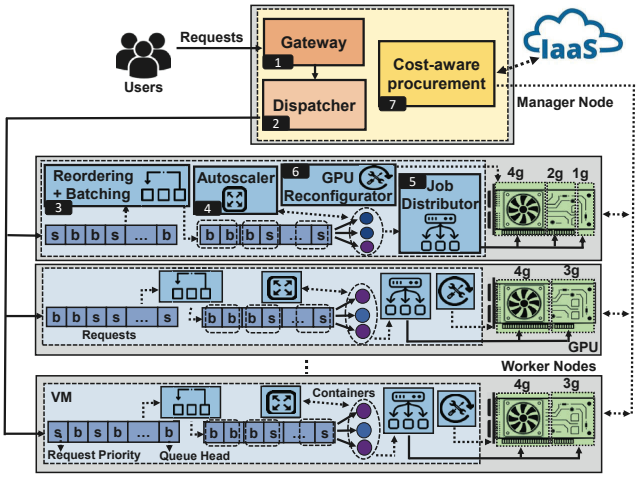


Figure 4: A schematic depicting PROTEAN's design.

> ***Guideline 2:*** *Strict requests should be scheduled on the appropriate (large) slice by accounting for both FBRs of all potentially co-located jobs as well as the RDF of the incoming job for the considered slice.*

The above insights lay the foundation for the scheduling policies that are incorporated into our design.

4 Overall Design of PROTEAN

Figure 4 outlines the overall design of PROTEAN. The Gateway ❶ provides a point of access to user requests to our serverless framework. These requests are load-balanced across the worker nodes by the Dispatcher ❷, thereby routing them to the appropriate nodes. Before invoking the respective workloads, the requests are first reordered ❸ by *prioritizing* strict requests to improve SLO compliance. After this, requests are serviced as batches ❸ by GPU-accelerated containers. The Autoscaler ❹ appropriately scales containers up or down such that there is at least one container to service each incoming request batch. The containers use the Job Distribution logic ❺ to schedule jobs on different slices in the current GPU geometry such that BE requests are packed into the fewest, smallest possible slices while distributing strict requests among the large slices to balance the tradeoff between resource deficiency and job interference (Equation 2). The GPU Reconfigurator ❻ selects the appropriate geometry to facilitate the above scheduling by configuring the GPU on-the-fly. Here, the worker nodes are acquired from the IaaS provider by the Cost-aware Procurement Module ❼ to provision Spot instances whenever possible, thereby reducing costs. We detail the key design features of PROTEAN below.

4.1 Request Reordering and Batching

To enable high throughput and GPU resource utilization, we enable requests to be batch-served ❸ [91, 100] by GPU-accelerated

containers. Provided there are multiple batches, the end-to-end latency of a request batch, t, is given by $t = t_{cold} + t_{queue} + t_{exec}$, where t_{cold} is the cold start time, t_{queue} is the queuing delay incurred by a batch waiting to be served behind other batches at a container, and t_{exec} represents the batch execution time. Assuming that we are able to avoid cold starts using 'warm' containers (which will be discussed in the next subsection), the end-to-end latency, t, can be minimized by reducing the queuing delay, t_{queue}.

To improve SLO compliance, we prioritize strict requests by reordering requests ❸ accordingly before batch-serving them with containers (total overhead < 1ms). This allows strict requests to be served *earlier*, thereby reducing the queuing delay and, hence, potentially improving SLO compliance. Note that provisioning a sufficient number of containers (one per batch) without reordering requests can also potentially eliminate queuing. However, unpredictable request surges can find the system underprovisioned with containers, thus, potentially reintroducing queuing delays. Our request reordering policy can potentially negate this as well.

4.2 Autoscaling

Protean has an auto-scaling module ❹ to scale containers to service incoming requests. Here, we *re-purpose* existing techniques to better suit inference apps to prevent SLO violations due to container under-provisioning:

Reactive scale-up – Containers are scaled up such that there is one container spawned for each batch of requests received (versus many traditional (CPU-based) autoscaling policies that spawn a container per request [6, 95]). This helps negate queuing effects (seen in Section 4.1). Thus, the number of containers spawned, n_c, is given by $n_c = \sum_{i=1}^{n_m} \left\lceil \frac{n_r(m_i)}{batch\_size(m_i)} \right\rceil$, where n_m is the total number of models invoked, $n_r(m_i)$ is the total number of incoming requests for a particular model, m_i, and $batch\_size(m_i)$ is the model batch size. Note that, because of the relatively high batch sizes used, we can tolerate most request load variations without spawning more containers (and incurring cold starts).

Delayed termination – For increased SLO compliance, we terminate active (warm) containers only after an extended (tunable) period of time (~10 minutes) elapses throughout which some containers (which will eventually be shut down) are consistently deemed to be 'surplus' by the reactive scale-up policy (similar to existing keep-alive policies [99]). This, combined with request batching, *reduces the number of cold starts by up to 98%* versus scaling down containers immediately in response to temporary request load reduction.

4.3 Job Distribution

The containers spawned by the autoscaling module employ the Job Distribution logic ❺ to effectively schedule jobs across the available GPU slices. This is done with the scheduling criteria discussed in Section 3 in mind: whenever possible, all BE requests are isolated from the strict requests on the fewest, smallest slices whereas strict requests are scheduled on the larger slices such that they suffer minimal slowdown due to both job interference and resource deficiency (Equation 2). To this effect, the Job Distributor implements Algorithm 1.

Here, slices (in ascending order of resources) are associated with a *tag_value* ⓐ, indicating what fraction of its memory will be occupied by BE requests (default *tag_value* = 0). Then, the *Distribute_Jobs* procedure schedules jobs on the slices chosen by helper methods ⓑ, ⓒ based on job latency targets (or lack thereof). *choose_strict_slice( )* ⓑ chooses slices which will not be completely occupied by BE requests (*tag_value* < 1), and will yield the least slowdown factor, η, (Equation 2) when the job is scheduled on it. This also considers potential slowdown due to occupancy by BE requests using *tag_value*s. In contrast, *choose_best_effort_slice( )* ⓒ packs BE requests in as few, small slices as possible via *First Fit Bin Packing* [68]. Here, prerequisites, such as FBRs, are estimated through profiling.

Algorithm 1 Scheduling Jobs using Job Distribution logic

```
1: from request_reordering_module get BE_mem
2: for slice in geometry.sorted(ascending) do
3:     if BE_mem > 0 then
4:         tag_value ← min(1, BE_mem / slice.available_mem) (a)
5:         BE_mem ← max(0, BE_mem - slice.available_mem)
6:         tag(slice, tag_value)
7:     else
8:         break
9: Distribute_Jobs(∀ batch ∈ sorted_batch_queue)
10: procedure Distribute_Jobs(batch)
11:     if batch is strict then
12:         chosen_slice ← choose_strict_slice(batch) (b)
13:     else
14:         chosen_slice ← choose_best_effort_slice(batch) (c)
15:     schedule(batch, chosen_slice)
```

4.4 GPU Reconfigurator

As per Algorithm 2, the GPU Reconfigurator configures the GPU geometry to enable arriving requests to be served using the heuristics discussed previously. Using the number of BE requests predicted (via the light-weight EWMA model [95]) ⓐ, their total memory requirement is calculated ⓑ. This is done so that the GPU is configured *in advance* to serve requests arriving after its reconfiguration time (~2s) has elapsed.

Once the future 'BE memory footprint' is estimated, the final geometry is decided by finding the set of slices with the minimum total memory that can accommodate it ⓒ. As a consequence, the remaining larger slices are also selected. We schedule (most) strict requests on them as seen in Section 4.3. To avoid using a sub-optimal geometry in corner cases (very few or high BE requests), we identify 'threshold values' for BE request occupancy for the smaller slices (using profiling information, predicted occupancy, etc.). Here, T_{low} ⓓ determines the number of BE requests below which consolidating both strict and BE requests on a *3g* is preferred since the performance offered by the *3g* outweighs any interference caused by BE requests in this case. Similarly, T_{high} ⓔ is the number of BE requests above which the slowdown (due to resource deficiency and interference) in (*2g,1g*) would be too high whenever strict and BE requests get co-located. If the chosen slices are out of the threshold bounds or if they cannot fit all BE requests, we use the (*4g, 3g*) geometry ⓕ, which, in our experience, is the most effective in such scenarios. Finally, our choice of geometry is compared against the current one and the GPU is reconfigured if there is a mismatch and similar mismatches have happened repeatedly (3 times at least) ⓖ (indicating a trend in the request trace variation, thus justifying the (brief) downtime for GPU reconfiguration). Note that only ~30% of GPUs (on average) are allowed to be reconfigured simultaneously to keep overall GPU downtime low.

Algorithm 2 GPU Reconfigurator

```
1: for Every Monitor_Interval = W do
2:     from curr_request_queue get curr_queue_info
3:     GPU_Reconfigurator(current_queue_info)
4: procedure GPU_Reconfigurator(curr_queue_info)
5:     init: final_geometry : null
6:           small_slice_set : [ [ 1g, 2g], [ 3g] ]
7:           found : False
8:     pred_BE_num ← predict_num_BE( curr_queue_info, history_info) (a)
9:     pred_BE_mem ← mem(BE_model, pred_BE_num) (b)
10:    for slice_set in small_slice_set do
11:        if sum_max_mem(slice_set) >= pred_BE_mem then (c)
12:            chosenSS ← slice_set
13:            update_pot_occupancy(chosenSS, pred_BE_num, BE_model)
14:            T_low ← calc_Lthresh(chosenSS, pred_BE_num, BE_model) (d)
15:            T_high ← calc_Hthresh(chosenSS, pred_BE_num, BE_model) (e)
16:            final_geometry.append(chosenSS.members)
17:            found ← True
18:            break
19:    if found is False or pot_occupancy[chosenSS] < T_low or
       pot_occupancy[chosenSS] > T_high then (f)
20:        final_geometry ← [ 4g, 3g]
21:    else
22:        if found is True then
23:            final_geometry.append(4g)
24:    if curr_geometry is not final_geometry then
25:        if wait_ctr >= wait_limit then (g)
26:            reconfigure_GPU(final_geometry)
27:        else
28:            wait_ctr ← wait_ctr + 1
29:    if curr_geometry is final_geometry then
30:        wait_ctr ← 0
```

4.5 Cost-aware Procurement

With serverless costs being closely linked to VM costs of the hosting IaaS provider(s) [103], we propose to acquire inexpensive spot VMs to host key components of our framework whenever possible. However, as relying solely on spot VMs can inevitably lead to server downtimes resulting from spot VM evictions, we employ reliable on-demand VMs as a backup hosting platform. On receipt of the eviction notification, we retry for another spot VM and only issue an on-demand VM request if the spot request fails. Once the new VM (spot/on-demand) is acquired, we redirect the incoming traffic to it. Two key factors enable this optimization: (i) The eviction notification arrives at least 30s prior [7, 8, 17], thus, affording us enough time to recover. (ii) As GPU serverless workloads are typically short-running ($< 1s$ [100]), the remaining requests on the soon-to-be-evicted VM can finish before the eviction. Note that our system can also provision *only* on-demand instances (with higher availability, but higher prices), if the user so desires.

5 Implementation and Experimental Setup

Protean's *real-system implementation* has modules (as shown in Figure 4) which are primarily implemented as daemon processes (in Python), achieving their functionalities (described in Section 4) by using MPS [25], MIG [24], GPU-accelerated containers [21] (with PyTorch v1.1 [77]), etc. We use Docker (v20) swarm [15] to manage the cluster. Here, we emulate *only* the spot/on-demand VM worker aspect (the pricing and revocations) by projecting the total cost (including scheduling/profiling costs) based on running time (including keep-alive and execution times) using average AWS spot and on-demand pricing. There is one spot/on-demand VM per node in the cluster, each VM image being 512GB. We generate revocation notifications at each worker node at fixed time intervals based on revocation probability (P_{rev}) values derived from [76]: (i) High spot VM availability ($P_{rev} = 0$); (ii) Moderate spot VM availability ($P_{rev} = 0.354$); and (iii) Low spot VM availability ($P_{rev} = 0.708$).

Below, we detail some aspects of the experimental setup.

Hardware: Protean is set up on a 9 node (multi-GPU) cluster (including a dedicated manager node). Each of the 8 worker nodes is equipped with an NVIDIA A100 40GB GPU, an AMD EPYC Rome CPU (48 cores), 128 GB of RAM, and 1 TB of storage and is connected via a 100 Gigabit Infiniband connection. We measure GPU utilization (percentage non-idle time) and GPU memory usage using NVIDIA-smi [26].

Workloads: We use 22 *diverse* ML inference workloads from the vision and language domains with varying memory footprints (from ~2 to 14 GB per batch) and resource requirements. For our primary experiments, we use image classification workloads based on the following models: *ResNet 50* [57], *GoogleNet* [96], *DenseNet 121* [61], *DPN 92* [41], *VGG 19* [92], *ResNet 18* [57], *MobileNet* [59], *MobileNet V2* [84], *SENet 18* [60], *ShuffleNet V2* [73], *EfficientNet-B0* [97], and *Simplified DLA* [102]. We use a batch size of 128 and the ImageNet 1k [83] dataset. As a part our sensitivity study, we use sequence classification workloads that are based on Large Language Models (LLMs) (including modern generative LLMs) with very high FBRs, namely, *AlBERT* [69], *BERT* [47], *DeBERTa* [58], *DistilBERT* [85], *FlauBERT* [70], *Funnel-Transformer* [44], *RoBERTa* [72], *Squeeze-BERT* [62], *OpenAI GPT-1 and GPT-2*. Here, batch size = 4 and the dataset is the Large Movie Review Dataset [74]. 'Strictness' of all requests is assumed to be user-annotated. We set all workload SLOs (for strict requests) to be 3× higher than its execution time [54] on *7g*. Also, our batch sizes are selected so that their execution latency on *7g* is between ~50-200ms, similar to the inference requests in China's largest local life service website [100].

Request Traces: We use real-world traces from Wikipedia [98] as they resemble the diurnal request arrivals of ML inference workloads [55]. We also consider the Twitter trace [2] which is erratic, and has a large peak-to-mean ratio (4561:2969) versus the Wiki trace (316:303). We also scale the request rates of the Wiki and Twitter traces so that their mean and peak rates, respectively, are ~5000 requests per second (rps) (as seen in [9]), for the vision models. For the language models, we use a much lower rate of 128 rps. We use a 50-50 mix of strict and BE requests for all experiments, unless specified. Here, the strict requests correspond to either a (fixed) LI or HI model, and the BE request varies randomly (every ~20s), to correspond to models from the opposite category (HI/LI), where applicable.

Evaluated schemes: We compare Protean against schemes which employ the request serving policies of state-of-the-art GPU-enabled serverless frameworks, namely, *INFless* [100], *Llama* [82], and *Molecule* [48]. Despite being different frameworks, *INFless* and *Llama* both employ MPS to schedule multiple request batches onto the available GPU while being agnostic of its MIG capabilities. We refer to the corresponding scheme as *INFless/Llama*. *Molecule (beta)* is the scheme representative of *Molecule*, which offers minimal GPU support without MPS to consolidate requests on GPU(s). Instead, it executes workload batches on the GPU(s) via *time sharing*. Finally, we introduce a scheme, *Naïve Slicing*, which spatially shares (via MPS) static MIG slices among requests, load-balanced according

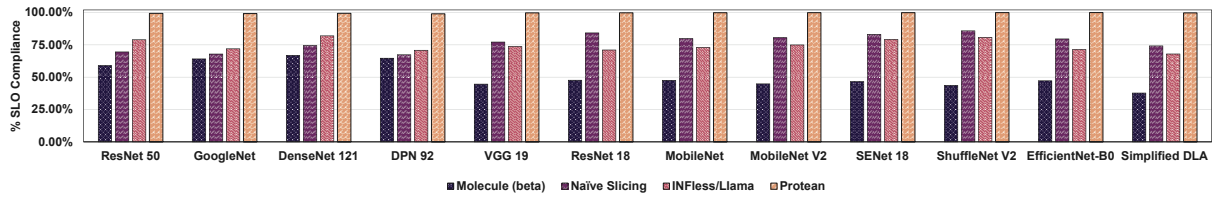


Figure 5: Comparison of SLO compliance of all schemes for all vision models.

to slice memory, without any of the intelligence of Protean. Note that we omit a 'MIG Only' scheme from our experiments for brevity since *MIG Only* performs worse than our proposed scheme for the considered models (Section 2).

6 Evaluation

This section presents a thorough evaluation of Protean. Unless mentioned otherwise, the plots presented here pertain to the Wiki trace with vision workloads. For individual examples, similar results are seen for other workloads as well.

6.1 Primary Results

6.1.1 ***SLO compliance and Tail Latency*** Figures 5 and 6 depict the percentage of strict requests satisfying their SLO target and the P99 latency of strict requests, respectively, for the specified vision models. As can be observed, Protean outperforms all other schemes with up to 62% more of its requests being SLO-compliant (Figure 5). Similarly, we observe Protean to also have the least tail latency (up to 73% lower than others) (Figure 6). These results can be attributed primarily to Protean leveraging the MPS and MIG capabilities of underlying GPUs dynamically and scheduling requests intelligently so as to ensure strict requests are scheduled by considering the tradeoff between interference and resource deficiency. Now, let us delve into the specific reasons for each scheme's performance, by considering the breakdown of tail (P99) latencies plotted in Figure 6.

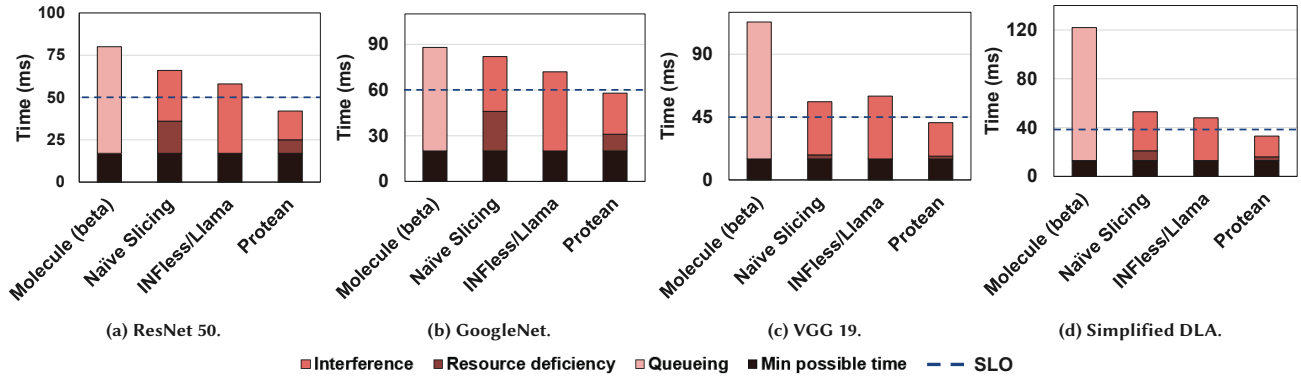
INFless/Llama consolidates all requests (to a node) on the same individual GPU using MPS, regardless of their latency targets (Section 5). This, as shown in Figure 6, can lead to increased job interference between the co-located requests, thus degrading their collective performance (particularly that of strict requests, as discussed in Section 3). For instance, for *VGG 19* (Figure 6c), 75% of the tail latency of *INFless/Llama* is due to the cumulative job interference of all co-located requests, including the BE requests. Consequently, for the same model, these schemes have an SLO compliance of only 73.76% (Figure 5). In comparison, Protean has 99.74% SLO compliance here. This is partly due to the reduced interference (47% lesser) suffered by strict requests (Figure 6c), due to a) being scheduled intelligently on the available larger slice(s), and b) being isolated from the BE requests.

In comparison to *Naïve Slicing*, which apportions requests among GPU slices without prioritizing strict requests in any form, Protean has better SLO compliance (up to 32% more) and tail latency (up to 36% less) (Figures 5, 6). This is due to Protean ensuring that strict requests are scheduled on the largest slice with the least slowdown factor (Equation 2). In contrast, *Naïve Slicing* may schedule strict requests without considering the impact of resource deficiency and/or job interference.

As mentioned in Section 5, *Molecule (beta)* does not spatially share GPU(s), but instead, relies on time sharing to service workload batches. Thus, despite *Molecule (beta)* not suffering from job interference or resource deficiency like the other schemes, a significant proportion of the requests served by it (many of which are strict requests) are plagued by queuing delays. Protean, on the other hand, strategically leverages both the MPS and MIG capabilities of the underlying GPU (as discussed previously) and thereby, outperforms *Molecule (beta)* in terms of both SLO compliance (up to 62% more) (Figure 5) and tail latency (up to 73% less) (Figure 6).
Demonstration: To illustrate the effects of Protean's dynamic GPU reconfiguration, we show a snippet of its execution in Figure 7. Compared to other schemes, Protean remains well within the SLO target by appropriately scheduling requests across the slices (*4g* and *2g*) and via request reordering. As per Algorithm 2, Protean's initial geometries of all GPUs are set to (*4g,2g,1g*). At ❶ in Figure 7, when the BE model changes to *DPN 92* (with up to a 2.74× larger memory footprint than that of previous models), the latency rises, since (*2g, 1g*) cannot collectively hold all DPN 92 requests, leading to their 'spillage' to *4g*, in turn, causing interference to its resident strict requests also. Now, Protean detects that changing 2 GPUs' geometries to (*4g, 3g*) can remedy this situation according to Algorithm 2. Once Protean's waiting limit elapses (ⓖ in Algorithm 2), it initiates the geometry change at ❷ in Figure 7, thereby, lowering the latency.

6.1.2 ***End-to-End Latency Distribution*** An analysis of the end-to-end latency distribution can shed more light on the observed performance of each of the schemes (Figure 8). Protean has a relatively flat distribution curve which remains within the SLO for the entire range (till P99). The low variation in latency values can be attributed to the isolation of BE and strict requests (largely) enforced by Protean. Here, both *INFless/Llama* as well as *Naïve Slicing* exceed the SLO at not only the tail (P99), but around P80 as well, due to the reasons discussed in Section 6.1.1. The CDF curve for *Molecule (beta)* rises progressively, due to queueing effects faced by strict requests.

6.1.3 ***Cost Savings*** Let us consider Figure 9. Here, we have an additional scheme, *Spot Only*, which is a variant of our scheme that attempts to aggressively reduce costs by employing only spot VMs. Protean, by only using spot VMs when available, and switching to on-demand VMs during periods of spot VM unavailability, achieves cost savings (up to 70% more) compared to the other schemes (which use only on-demand VMs) while also having better SLO compliance (up to 99% more). While *Spot Only* has lower costs than Protean (35% on average, with up to 71% savings), its SLO compliance is adversely affected under scenarios with limited spot VM availability due to sufficient number of workers being unavailable to service the arriving requests, thereby increasing the load on the few available



(a) ResNet 50. (b) GoogleNet. (c) VGG 19. (d) Simplified DLA.

Figure 6: Breakdown of job tail (P99) latencies for all schemes for a subset of the vision models.

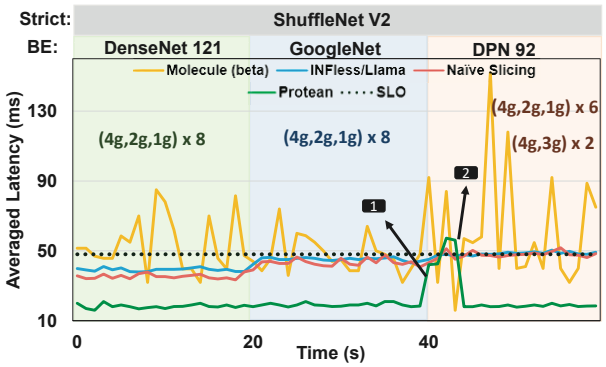


Figure 7: Snapshot showing the effects of Protean's dynamic geometry selection for the *ShuffleNet V2* model. Protean's geometry for all GPUs for each BE model is indicated in the body of the graph. The other schemes are shown for reference.

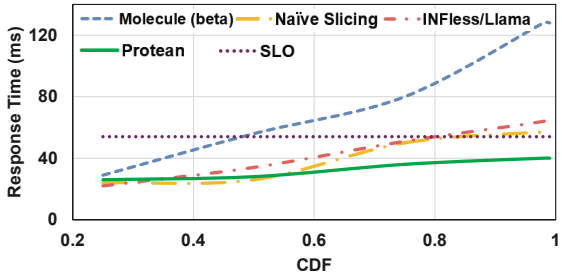
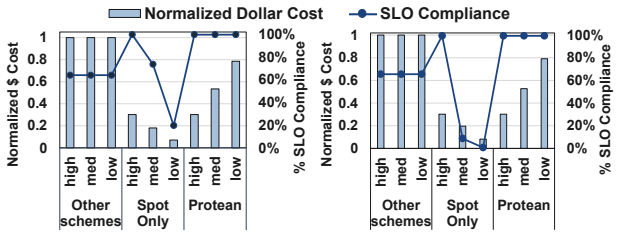


Figure 8: Cumulative Distribution Function (CDF) of the end-to-end job latencies for all schemes for the *SENet 18* model.

workers. This can degrade the performance of *Spot Only*, especially for HI models (Section 3). For instance, for *ResNet 50*, *Spot Only* has only 8.76% and 0.68% SLO compliance under medium and low spot VM availability compared to Protean's 99.35% (Figure 9b). Given these results, we believe our strategy of relying on both spot and on-demand VMs strikes the appropriate balance between cost savings and SLO compliance.

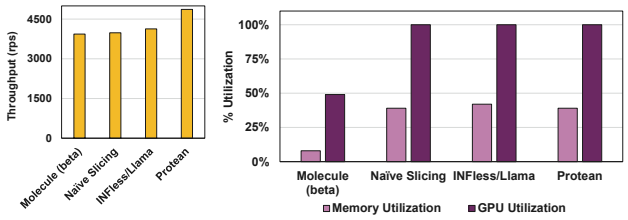
6.1.4 ***Analysis of other Key Benefits*** Here, we evaluate Protean with respect to other metrics.

Throughput Here, throughput refers to the total number of strict requests served on average per GPU per second. We observe that Protean has up to 24% higher throughput than other schemes (Figure 10a). Given that the average number of strict requests and



(a) VGG 19. (b) ResNet 50.

Figure 9: Normalized Dollar Cost vs. SLO compliance for other schemes (averaged across them), *Spot Only* and Protean for high, medium, and low spot VM availability.



(a) Throughput. (b) GPU Utilization.

Figure 10: Protean's other key benefits: Throughput (for *DenseNet 121*), and Resource Utilization (for *EfficientNet-B0*).

batch sizes used are the same across the schemes, throughput is determined by the batch execution latency of strict requests. Thus, the higher throughput of Protean is due to its policies that prioritize strict requests. Note that Protean also achieves the highest combined throughput (strict and BE) here as well (up to ~2.5× more), closely followed by *INFless/Llama* (2% lesser)[2].

GPU Utilization As per Figure 10b, Protean, similar to *INFless/Llama* and *Naïve Slicing*, maximizes GPU utilization (i.e., minimizes GPU idle time), owing to the collective compute requirements of co-located request batches. Protean achieves 39% memory utilization, which is very similar to that of both *Naïve Slicing* (39%) and *INFless/Llama* (42%). The slightly lower memory utilization of Protean here is due to an observed decrease in workload memory footprint when scheduled on smaller slices. In comparison to these schemes, *Molecule (beta)* achieves only 49% GPU utilization and 8% memory utilization due to executing one request batch at a time on the whole GPU, without spatially sharing it.

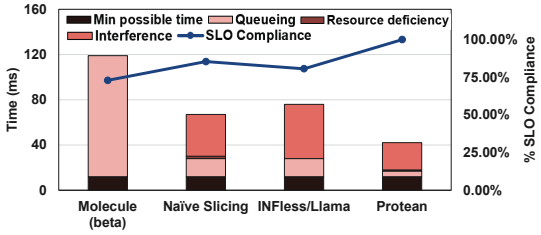


Figure 11: Tail latency breakdown vs. SLO compliance for all schemes for *MobileNet* when subjected to an *erratic* trace (Twitter trace).

Tail Latency of BE Requests Since our scheduling and GPU reconfiguration policies are designed to minimize tail latency for strict requests, we observe that the other policies can achieve up to 42% lower tail latency than Protean for BE requests[2]. Despite this, Protean successfully keeps their performance degradation in check, with the tail latency of all BE requests remaining under 200ms [2] (within the expected latency for user-facing inference tasks [36, 100]). This is likely due to Protean conservatively provisioning containers, thus, being able to serve BE requests with minimal queuing.

6.2 Sensitivity Study

Now, we evaluate Protean under varied settings from those of the previous experiments.

Erratic trace Consider Figure 11 that depicts the performance of all schemes for the Twitter trace. Note that since we scale the request rates of Twitter to have a peak of ~5000 rps (as per Section 5), we obtain a resultant mean rate of ~3000 rps, which is 35% lower than that used previously for the Wiki trace. Despite this, *INFless/Llama* and *Naïve Slicing* are adversely affected by the sudden request surges here as this can find these schemes under-provisioned with containers, thus causing them queuing overheads (up to 24% of their P99 latency). While Protean is also affected by similar circumstances, it minimizes queuing effects (by ~69% versus *INFless/Llama* and *Naïve Slicing*) by prioritizing strict requests through request reordering (Section 4.1). Combining this with our other policies, Protean achieves much higher SLO compliance (99.90%) versus other schemes.

Very High Interference (VHI) models Here, we consider large language models with high FBRs (59% higher on average versus the vision models). We refer to them as Very High Interference (VHI) models. The schemes which use MPS to co-locate the VHI models (all except *Molecule (beta)*) suffer more from job interference due to the high FBRs of the workloads (Figure 12). Protean, despite using MPS, almost always outperforms all others in terms of SLO compliance (by as much as ~93%) by virtue of all its policies that prioritize strict requests through intelligent request scheduling and GPU reconfiguration. *INFless/Llama*, in particular, is the worst-affected, with an average SLO compliance of only 5.92% for the VHI models due to the high interference from 'MPS-only' co-location. Generally, *Molecule (beta)* has lower SLO compliance than Protean due to time sharing GPUs (resulting in queueing), except for *FlauBERT*, where the relative queueing delays are lower.

[2]Not shown in Figures/Tables due to space constraints

Performance of Modern Generative LLMs: As part of the VHI model analysis, we also study the performance of generative LLMs (*OpenAI GPT-1*, [78] and *GPT-2* [79]) (Figure 13). Here, the strict requests correspond to a *GPT* model, whereas the BE requests correspond to the previously-seen LLMs (varied at random). Due to the especially high FBRs of the *GPT* models (up to 42% versus the previous LLMs), the SLO compliance is further reduced for all MPS-based schemes, especially for *INFless/Llama*, which fails to satisfy the SLOs for any request due to the higher interference resulting from MPS-based co-location of all requests. Protean's performance degradation, on the other hand, is limited primarily because it co-locates both BE and strict on the smaller GPU slice to reduce the total interference experienced by the majority of strict requests on the larger slice(s). Doing so helps it achieve (the highest) SLO compliance of 90% on average across *GPT-1* and *GPT-2*. This reiterates the fact that Protean does not always merely isolate strict and BE requests, but instead, *carefully determines* the ideal request co-location levels for the available slices. *Molecule (beta)*, in this case, achieves better SLO compliance for *GPT-2* (~79%) than *GPT-1* (61.45%), due to the lower queueing delays for *GPT-2* relative to its (high) execution latency.

Varied Strictness Ratios Here, we study the performance of all schemes under two scenarios with varied 'strictness' ratios: (i) *Strict skewed*: 75% strict and 25% BE requests, (ii) *BE skewed*: 25% strict and 75% BE requests.

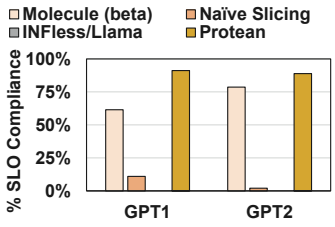


Figure 13: SLO Compliance for modern generative LLMs.

In the *Strict skewed* case, Protean achieves an SLO compliance of 99.99% and 93.78% for the *ShuffleNet V2* and *DPN 92* models, respectively, thereby, outperforming all other schemes (Figure 14a). For *ShuffleNet V2*, we observe that all MPS-based schemes (including Protean) achieve at least ~89% SLO compliance. This is due to the lower FBRs (and low interference) of *ShuffleNet V2*, which composes the majority of the request trace. Conversely, for *DPN 92*, since its requests form the majority, the job interference experienced in MPS-based schemes is high (as *DPN 92* is an HI model).

For *BE skewed*, Protean, yet again, achieves the highest SLO compliance (at least 99.96%) for both models (Figure 14b). For *ShuffleNet V2*, *Naïve Slicing* achieves a high SLO compliance (97.86%) because of its request scheduling across slices and since *ShuffleNet V2* is barely affected (< 2%) by resource deficiency effects for those slices. For *DPN 92*, all schemes perform well, with at least 98.56% SLO compliance, since BE requests (corresponding to LI models) form the majority of requests, causing minimal interference/queueing.

Extreme Skewed Cases: As part of the above study, we also analyze cases where the requests are either 100% strict or 100% BE. Table 4 shows the SLO compliance for all schemes for a 100% strict case with the *ResNet 50* model. Note that this (no BE requests) is the 'default' scenario for which works like *INFless/Llama* were designed for. Since *all* the requests here correspond to an HI model (versus previous experiments with both HI and LI models), the resultant

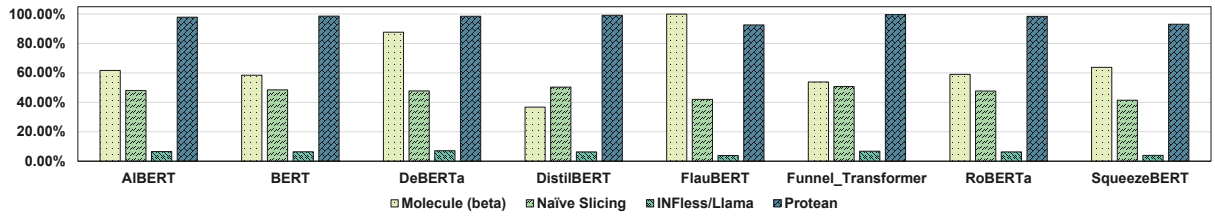
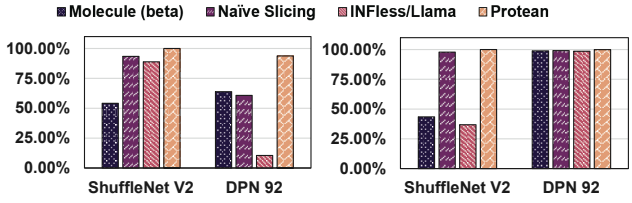


Figure 12: Comparison of SLO compliance of all schemes for the shown large language (VHI) models.



(a) Strict skewed. (b) BE skewed.

Figure 14: SLO compliance for skewed strictness ratios.

interference due to MPS is noticeably higher, thus, degrading the performance of all MPS-based schemes, including Protean. In particular, *INFless/Llama* is adversely affected by this, achieving <1% SLO compliance. However, Protean minimizes interference due to co-location amongst the strict requests with its intelligent policies, and achieves the highest SLO compliance (94.19%).

Molecule (beta)	*Naïve Slicing*	INFless/Llama	Protean
60.12%	54.31%	0.42%	94.19%

Table 4: SLO Compliance for 100% strict case.

For the 100% BE case (Table 5), the corresponding models are varied at random from the pool of HI models. Since SLO compliance is not a valid metric for BE requests, we compare the median (P50) and tail (P99) latency values of the schemes in this case. We observe that, while Protean achieves the best median latency for BE requests (up to 48.5% lesser), other schemes outperform it in terms of P99 latency (by up to 28%). This is because Protean gives BE requests lower priority (unlike other schemes which are agnostic to request strictness) and hence, schedules many of them on smaller GPU slices, in addition to putting them at the back of request queues, thus degrading tail latency. However, optimizing request scheduling for both P50 and P99 latency for such corner cases (100% BE) is worth looking into as part of future work.

Molecule (beta)	*Naïve Slicing*	INFless/Llama	Protean
(68, 165)	(50, 99)	(57, 130)	(35, 138)

Table 5: (P50, P99) latency (in ms) for 100% BE case.

Tight SLO Target This experiment studies the performance of all schemes under a tighter SLO target (2× the minimum execution latency). Due to this, the SLO compliance of the other schemes degrade considerably (overall, by up to 22%) versus Protean (which degrades up to ~5%) (Figure 15). Here, for

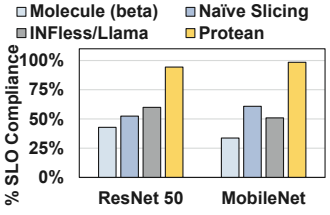


Figure 15: SLO Compliance when SLO target is tightened.

ResNet 50, Protean drops as low as 94.38%, likely due to it being an HI model, coupled with the enforcement of the tight SLO bounds. Despite this, these results demonstrate the resilience of Protean (especially at the tail of the response time distribution) that is a consequence of its intelligent policies.

Comparison against strategic MPS-only usage Protean uses the *default MPS configuration*, which allows the co-running workloads to decide their fraction of SMs. However, to compare against a scheme with carefully-allocated SM partitions via MPS (*GPUlet* [42]), we use an MPS execution mode, which sets an upper bound on the fraction of SMs used by a workload. Here, the limits on SM fractions are set according to the SLO of the model and the point of diminishing returns of performance. Thus, we set a ~60-65% upper bound on the SM usage for strict requests, with the remaining used by the BE requests. From Figure 16, Protean is seen to have up to ~16% more SLO compliance versus *GPUlet*. *GPUlet*, despite prioritizing strict requests by giving them a larger potential fraction of SMs, suffers from job interference (up to ~2× overhead[2]), due to the cache and memory bandwidth still being shared (Section 2.2). Protean, in comparison, maintains an average SLO compliance of 99.65% here, as a result of the much lower overheads (~88% lesser[2]), due to its policies that intelligently leverage *both* MPS and MIG.

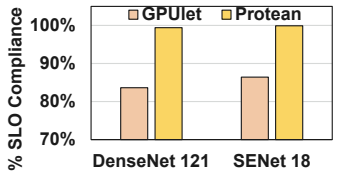


Figure 16: SLO Compliance of Protean versus *GPUlet* (Strategic MPS-only scheme).

Comparison versus Oracle Finally, we compare Protean with *Oracle*, a scheme with all of Protean's policies, but with knowledge of the ideal GPU configurations and job scheduling on slices to minimize strict request slowdowns (due to being done offline). As shown in Figure 17, *Oracle* only outperforms Protean by up to 0.42%, and in terms of tail latency by up to 17%. This difference in performance is largely because a) *Oracle* performs multiple offline configuration/scheduling sweeps b) *Oracle* does not suffer from GPU reconfiguration overheads. Note that Protean remains competitive despite not knowing the workloads and request rates beforehand.

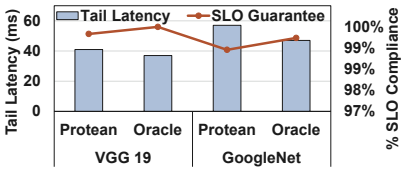


Figure 17: P99 latency and SLO compliance versus *Oracle*.

7 Discussion

Below, we provide additional insights and clarifications regarding key aspects of Protean, informed by the context established throughout the paper.

Generalizability: Protean can work for equivalent GPU sharing mechanisms of other vendors also since it does not depend on NVIDIA-specific features. Rather, it models common system overheads: (i) the job interference of co-located requests due to contention for shared resources, (in an MPS-like scenario) and (ii) resource deficiency due to partitioning a resource. (in a MIG-like scenario).

Revisiting Novelty: As demonstrated in the above Evaluation Section, we improve upon key performance and cost metrics compared to existing GPU-enabled serverless frameworks by being the *first to intelligently leverage both MPS and MIG*. As mentioned already, the problem we solve is *not* one of only isolating strict requests on larger slices. In fact, we use our slowdown models (Section 3) to *prudently trade off interference versus resource deficiency* (overcrowding larger slice(s) versus taking a lesser-occupied, yet smaller slice) when scheduling requests. Moreover, the GPUs have to be dynamically 'sliced' ahead of time to facilitate request scheduling by predicting the request mix that will arrive later (*no prior work does the above*). Finally, our work is the *first* to show the applicability of using spot instances intelligently (in the context of GPU-enabled serverless), to yield cost savings without compromising on performance.

Statistical Significance: For all experiments, for both the vision and NLP models/LLMs considered, we observed the following:

Confidence Intervals: Narrow ranges were seen (<0.1%), indicating that the true mean performance metric for each scheme likely falls very close to the shown estimates.

P-Values: All p-values were ~0.0, indicating that the differences between the compared scheduling schemes are statistically significant at the 0.05 level.

Effect Sizes (Cohen's d): Large to very large Cohen's d values were observed (7.80 to 304.37), indicating substantial differences in performance between the scheduling schemes. Protean significantly outperforms the other schemes, with a particularly large effect when compared to *Molecule (beta)* (for vision models), and *INFless/Llama* (for NLP models/LLMs).

8 Concluding Remarks

The unique features of new-generation hardware and major cloud provider offerings must be leveraged to develop an SLO compliant and cost-effective GPU-based serverless framework. To this end, we design, implement, and evaluate Protean, a GPU-enabled serverless framework which employs policies that intelligently leverage the MPS and MIG capabilities of recent NVIDIA GPU architectures (to improve SLO compliance), while using a hybrid spot/on-demand VM setup to host its key components (to reduce costs). Our evaluation results show that Protean significantly outperforms state-of-the-art works, in terms of SLO compliance (up to ~93% more) and tail latency (up to 82% less), while reducing the costs by up to 70%.

9 Acknowledgement

This research was partially supported by NSF grants #1931531, #2149389, and #2122155. All product names used here are for identification purposes only and may be trademarks of their respective companies. We are grateful to our anonymous reviewers and our shepherd, Djob Mvondo, for their insightful comments and guidance, which significantly improved this work. Finally, the lead author would like to express his deepest gratitude to his mother, Jerly P.V., for her invaluable support, from staying with him during most of the paper's development (and helping out in countless ways) to offering continuous moral encouragement.

References

[1] 2020. Establishing Effective SLOs. https://www.datadoghq.com/blog/establishing-service-level-objectives/.
[2] 2020. Twitter Stream traces. https://archive.org/details/twitterstream. Accessed: 2020-05-07.
[3] 2021. AWS Lambda Cold Starts. https://mikhail.io/serverless/coldstarts/aws/.
[4] 2021. Azure Functions Cold Starts. https://mikhail.io/serverless/coldstarts/azure/.
[5] 2022. The State of Serverless. https://www.datadoghq.com/state-of-serverless/.
[6] 2024. AWS Lambda. https://aws.amazon.com/lambda/.
[7] 2024. AWS Spot Instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html.
[8] 2024. Azure Spot Virtual Machines. https://learn.microsoft.com/en-us/azure/virtual-machines/spot-vms.
[9] 2024. AzureML Inference Router. https://learn.microsoft.com/en-us/azure/machine-learning/how-to-kubernetes-inference-routing-azureml-fe.
[10] 2024. Banana. https://docs.banana.dev/banana-docs/.
[11] 2024. Banana Latency Guarantees. https://www.banana.dev/.
[12] 2024. Beam. https://www.beam.cloud/.
[13] 2024. Cerebrium. https://docs.cerebrium.ai/introduction.
[14] 2024. CUDA Concurrency Mechanisms. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/#cuda-concurrency.
[15] 2024. Docker Swarm. https://docs.docker.com/engine/swarm/.
[16] 2024. Google Cloud Functions. https://cloud.google.com/functions.
[17] 2024. Google Spot VMs. https://cloud.google.com/compute/docs/instances/preemptible.
[18] 2024. GPUs vs CPUs for deployment of deep learning models. https://azure.microsoft.com/en-us/blog/gpus-vs-cpus-for-deployment-of-deep-learning-models/.
[19] 2024. Microsoft Azure Serverless Functions. https://azure.microsoft.com/en-us/services/functions/.
[20] 2024. NVIDIA A100 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/a100/.
[21] 2024. NVIDIA-Docker. https://github.com/NVIDIA/nvidia-docker.
[22] 2024. NVIDIA Grace Hopper Superchip Architecture In-Depth. https://developer.nvidia.com/blog/nvidia-grace-hopper-superchip-architecture-in-depth/.
[23] 2024. NVIDIA H100 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/technologies/hopper-architecture/.
[24] 2024. NVIDIA Multi-Instance GPU. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html.
[25] 2024. NVIDIA Multi-Process Service. https://docs.nvidia.com/deploy/mps/index.html.
[26] 2024. NVIDIA-smi. https://developer.nvidia.com/nvidia-system-management-interface.
[27] 2024. PCIe Special Interest Group PCIe 6 Specification. https://pcisig.com/pci-express-6.0-specification.
[28] 2024. Replicate. https://replicate.com/docs.
[29] 2024. Runpod. https://www.runpod.io/serverless-gpu.
[30] 2024. Serverless Application Lens: Alexa Skills. https://docs.aws.amazon.com/wellarchitected/latest/serverless-applications-lens/alexa-skills.html.
[31] 2024. Serverless Facebook Messenger Bot. https://github.com/pmuens/serverless-facebook-messenger-bot.
[32] 2024. Serverless Optical Character Recognition (OCR) Tutorial. https://cloud.google.com/functions/docs/tutorials/ocr.
[33] 2024. trainML. https://docs.trainml.ai/.
[34] Daniel S. Berger, Benjamin Berg, Timothy Zhu, Siddhartha Sen, and Mor Harchol-Balter. 2018. RobinHood: Tail Latency Aware Caching – Dynamic Reallocation from Cache-Rich to Cache-Poor. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 195–212. https://www.usenix.org/conference/osdi18/presentation/berger
[35] Vivek M. Bhasi, Jashwant Raj Gunasekaran, Aakash Sharma, Mahmut Taylan Kandemir, and Chita Das. 2022. Cypress: Input Size-Sensitive Container Provisioning and Request Scheduling for Serverless Platforms. In *Proceedings of the 13th Symposium on Cloud Computing* (San Francisco, California) *(SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 257–272. https://doi.org/10.1145/3542929.3563464

[36] Vivek M. Bhasi, Jashwant Raj Gunasekaran, Prashanth Thinakaran, Cyan Subhra Mishra, Mahmut Taylan Kandemir, and Chita Das. 2021. Kraken: Adaptive Container Provisioning for Deploying Dynamic DAGs in Serverless Platforms. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) *(SoCC '21)*. Association for Computing Machinery, New York, NY, USA, 153–167. https://doi.org/10.1145/3472883.3486992

[37] Vivek M. Bhasi, Aakash Sharma, Shruti Mohanty, Mahmut Taylan Kandemir, and Chita R. Das. 2024. Paldia: Enabling SLO-Compliant and Cost-Effective Serverless Computing on Heterogeneous Hardware. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 100–113. https://doi.org/10.1109/IPDPS57955.2024.00018

[38] Marc Brooker, Andreea Florescu, Diana-Maria Popa, Rolf Neugebauer, Alexandru Agache, Alexandra Iordache, Anthony Liguori, and Phil Piwonka. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *NSDI*.

[39] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, et al. 2016. A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE, 1–13.

[40] Quan Chen, Hailong Yang, Minyi Guo, Ram Srivatsa Kannan, Jason Mars, and Lingjia Tang. 2017. Prophet: Precise QoS Prediction on Non-Preemptive Accelerators to Improve Utilization in Warehouse-Scale Computers. *SIGARCH Comput. Archit. News* 45, 1 (apr 2017), 17–32. https://doi.org/10.1145/3093337.3037700

[41] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. 2017. Dual Path Networks. https://doi.org/10.48550/ARXIV.1707.01629

[42] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2022. Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 199–216. https://www.usenix.org/conference/atc22/presentation/choi-seungbeom

[43] Marcus Chow, Ali Jahanshahi, and Daniel Wong. 2023. KRISP: Enabling Kernel-wise RIght-sizing for Spatial Partitioned GPU Inference Servers. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 624–637. https://doi.org/10.1109/HPCA56546.2023.10071121

[44] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V. Le. 2020. Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing. https://doi.org/10.48550/ARXIV.2006.03236

[45] Jeffrey Dean and Luiz André Barroso. 2013. The Tail at Scale. *Commun. ACM* 56, 2 (feb 2013), 74–80. https://doi.org/10.1145/2408776.2408794

[46] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's Highly Available Key-Value Store. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles* (Stevenson, Washington, USA) *(SOSP '07)*. Association for Computing Machinery, New York, NY, USA, 205–220. https://doi.org/10.1145/1294261.1294281

[47] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://doi.org/10.48550/ARXIV.1810.04805

[48] Dong Du, Qingyuan Liu, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. 2022. Serverless Computing on Heterogeneous Computers. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 797–813. https://doi.org/10.1145/3503222.3507732

[49] Murali Emani, Zhen Xie, Siddhisanket Raskar, Varuni Sastry, William Arnold, Bruce Wilson, Rajeev Thakur, Venkatram Vishwanath, Zhengchun Liu, Michael E. Papka, Cindy Orozco Bohorquez, Rick Weisner, Karen Li, Yongning Sheng, Yun Du, Jian Zhang, Alexander Tsyplikhin, Gurdaman Khaira, Jeremy Fowers, Ramakrishnan Sivakumar, Victoria Godsoe, Adrian Macias, Chetan Tekur, and Matthew Boyd. 2022. A Comprehensive Evaluation of Novel AI Accelerators for Deep Learning Workloads. In *2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 13–25. https://doi.org/10.1109/PMBS56514.2022.00007

[50] Henrique Fingler, Zhiting Zhu, Esther Yoon, Zhipeng Jia, Emmett Witchel, and Christopher J. Rossbach. 2022. DGSF: Disaggregated GPUs for Serverless Functions. In *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 739–750. https://doi.org/10.1109/IPDPS53621.2022.00077

[51] Sadjad Fouladi, Francisco Romero, Dan Iter, Qian Li, Shuvo Chatterjee, Christos Kozyrakis, Matei Zaharia, and Keith Winstein. 2019. From Laptop to Lambda: Outsourcing Everyday Jobs to Thousands of Transient Functional Containers. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 475–488. http://www.usenix.org/conference/atc19/presentation/fouladi

[52] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 363–376. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi

[53] Diandian Gu, Yihao Zhao, Yinmin Zhong, Yifan Xiong, Zhenhua Han, Peng Cheng, Fan Yang, Gang Huang, Xin Jin, and Xuanzhe Liu. 2023. ElasticFlow: An Elastic Serverless Training Platform for Distributed Deep Learning. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 266–280. https://doi.org/10.1145/3575693.3575721

[54] Arpan Gujarati, Sameh Elnikety, Yuxiong He, Kathryn S. McKinley, and Björn B. Brandenburg. 2017. Swayam: Distributed Autoscaling to Meet SLAs of Machine Learning Inference Services with Resource Efficiency. In *USENIX Middleware Conference*.

[55] Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R. Das. 2022. Cocktail: A Multidimensional Optimization for Model Serving in Cloud. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 1041–1057. https://www.usenix.org/conference/nsdi22/presentation/gunasekaran

[56] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2020. DeepRecSys: A System for Optimizing End-to-End at-Scale Neural Recommendation Inference. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture* (Virtual Event) *(ISCA '20)*. IEEE Press, 982–995. https://doi.org/10.1109/ISCA45697.2020.00084

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. https://doi.org/10.48550/ARXIV.1512.03385

[58] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. DeBERTa: Decoding-enhanced BERT with Disentangled Attention. https://doi.org/10.48550/ARXIV.2006.03654

[59] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. https://doi.org/10.48550/ARXIV.1704.04861

[60] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. 2017. Squeeze-and-Excitation Networks. https://doi.org/10.48550/ARXIV.1709.01507

[61] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2016. Densely Connected Convolutional Networks. https://doi.org/10.48550/ARXIV.1608.06993

[62] Forrest N. Iandola, Albert E. Shaw, Ravi Krishna, and Kurt W. Keutzer. 2020. SqueezeBERT: What can computer vision teach NLP about efficient neural networks? https://doi.org/10.48550/ARXIV.2006.11316

[63] Syed M. Iqbal, Haley Li, Shane Bergsma, Ivan Beschastnikh, and Alan J. Hu. 2022. CoSpot: A Cooperative VM Allocation Framework for Increased Revenue from Spot Instances. In *Proceedings of the 13th Symposium on Cloud Computing* (San Francisco, California) *(SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 540–556. https://doi.org/10.1145/3542929.3563499

[64] Rishabh Jain, Scott Cheng, Vishwas Kalagi, Vrushabh Sanghavi, Samvit Kaul, Meena Arunachalam, Kiwan Maeng, Adwait Jog, Anand Sivasubramaniam, Mahmut Taylan Kandemir, and Chita R. Das. 2023. Optimizing CPU Performance for Recommendation Systems At-Scale. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (Orlando, FL, USA) *(ISCA '23)*. Association for Computing Machinery, New York, NY, USA, Article 77, 15 pages. https://doi.org/10.1145/3579371.3589112

[65] Ram Srivatsa Kannan, Lavanya Subramanian, Ashwin Raju, Jeongseob Ahn, Jason Mars, and Lingjia Tang. 2019. GrandSLAm: Guaranteeing SLAs for Jobs in Microservices Execution Frameworks. In *EuroSys*.

[66] Liu Ke, Udit Gupta, Mark Hempstead, Carole-Jean Wu, Hsien-Hsin S. Lee, and Xuan Zhang. 2022. Hercules: Heterogeneity-Aware Inference Serving for At-Scale Personalized Recommendation. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 141–154. https://doi.org/10.1109/HPCA53966.2022.00019

[67] Jaewook Kim, Tae Joon Jun, Daeyoun Kang, Dohyeun Kim, and Daeyoung Kim. 2018. GPU Enabled Serverless Computing Framework. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. 533–540. https://doi.org/10.1109/PDP2018.2018.00090

[68] Bernhard Korte and Jens Vygen. 2018. Bin-Packing. In *Combinatorial Optimization*. Springer, 489–507.

[69] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. https://doi.org/10.48550/ARXIV.1909.11942

[70] Hang Le, Loïc Vial, Jibril Frej, Vincent Segonne, Maximin Coavoux, Benjamin Lecouteux, Alexandre Allauzen, Benoît Crabbé, Laurent Besacier, and Didier Schwab. 2019. FlauBERT: Unsupervised Language Model Pre-training for French. https://doi.org/10.48550/ARXIV.1912.05372

[71] Baolin Li, Tirthak Patel, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2022. MISO: Exploiting Multi-Instance GPU Capability on Multi-Tenant GPU Clusters. In *Proceedings of the 13th Symposium on Cloud Computing* (San Francisco, California) *(SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 173–189. https://doi.org/10.1145/3542929.3563510

[72] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. https://doi.org/10.48550/ARXIV.1907.11692

[73] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. https://doi.org/10.48550/ARXIV.1807.11164

[74] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, 142–150. http://www.aclweb.org/anthology/P11-1015

[75] Diana M. Naranjo, Sebastián Risco, Carlos de Alfonso, Alfonso Pérez, Ignacio Blanquer, and Germán Moltó. 2020. Accelerated serverless computing based on GPU virtualization. *J. Parallel and Distrib. Comput.* 139 (2020), 32–42. https://doi.org/10.1016/j.jpdc.2020.01.004

[76] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, , and Matei Zaharia. [n.d.]. Analysis and Exploitation of Dynamic Pricing in the Public Cloud for ML Training. *VLDB DISPA Workshop 2020* ([n. d.]). https://par.nsf.gov/biblio/10213411

[77] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[78] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[79] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[80] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2020. MLPerf Inference Benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 446–459. https://doi.org/10.1109/ISCA45697.2020.00045

[81] Sebastián Risco and Germán Moltó. 2021. GPU-enabled serverless workflows for efficient multimedia processing. *Applied Sciences* 11, 4 (2021), 1438.

[82] Francisco Romero, Mark Zhao, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Llama: A Heterogeneous Serverless Framework for Auto-Tuning Video Analytics Pipelines. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) *(SoCC '21)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3472883.3486972

[83] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[84] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. (2018). https://doi.org/10.48550/ARXIV.1801.04381

[85] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. https://doi.org/10.48550/ARXIV.1910.01108

[86] Klaus Satzke, Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Andre Beck, Paarijaat Aditya, Manohar Vanga, and Volker Hilt. 2021. Efficient GPU Sharing for Serverless Workflows. In *Proceedings of the 1st Workshop on High Performance Serverless Computing* (Virtual Event, Sweden) *(HiPS '21)*. Association for Computing Machinery, New York, NY, USA, 17–24. https://doi.org/10.1145/3452413.3464785

[87] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J. Yadwadkar, Raluca Ada Popa, Joseph E. Gonzalez, Ion Stoica, and David A. Patterson. 2021. What Serverless Computing is and Should Become: The next Phase of Cloud Computing. *Commun. ACM* 64, 5 (apr 2021), 76–84. https://doi.org/10.1145/3406011

[88] Aakash Sharma, Vivek M. Bhasi, Sonali Singh, Rishabh Jain, Jashwant Raj Gunasekaran, Subrata Mitra, Mahmut Taylan Kandemir, George Kesidis, and Chita R. Das. 2022. Analysis of Distributed Deep Learning in the Cloud. arXiv:2208.14344 [cs.LG] https://arxiv.org/abs/2208.14344

[89] Aakash Sharma, Vivek M. Bhasi, Sonali Singh, Rishabh Jain, Jashwant Raj Gunasekaran, Subrata Mitra, Mahmut Taylan Kandemir, George Kesidis, and Chita R. Das. 2023. Stash: A Comprehensive Stall-Centric Characterization of Public Cloud VMs for Distributed Deep Learning. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. 1–12. https://doi.org/10.1109/ICDCS57875.2023.00023

[90] Aakash Sharma, Vivek M. Bhasi, Sonali Singh, George Kesidis, Mahmut T. Kandemir, and Chita R. Das. 2024. GPU Cluster Scheduling for Network-Sensitive Deep Learning. arXiv:2401.16492 [cs.PF] https://arxiv.org/abs/2401.16492

[91] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: A GPU Cluster Engine for Accelerating DNN-Based Video Analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 322–337. https://doi.org/10.1145/3341301.3359658

[92] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. https://doi.org/10.48550/ARXIV.1409.1556

[93] Sonali Singh, Anup Sarma, Nicholas Jao, Ashutosh Pattnaik, Sen Lu, Kezhou Yang, Abhronil Sengupta, Vijaykrishnan Narayanan, and Chita R. Das. 2020. NEBULA: A Neuromorphic Spin-Based Ultra-Low Power Architecture for SNNs and ANNs. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 363–376. https://doi.org/10.1109/ISCA45697.2020.00039

[94] Sonali Singh, Anup Sarma, Sen Lu, Abhronil Sengupta, Vijaykrishnan Narayanan, and Chita R. Das. 2021. Gesture-SNN: Co-optimizing accuracy, latency and energy of SNNs for neuromorphic vision sensors. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 1–6. https://doi.org/10.1109/ISLPED52811.2021.9502506

[95] Arjun Singhvi, Arjun Balasubramanian, Kevin Houck, Mohammed Danish Shaikh, Shivaram Venkataraman, and Aditya Akella. 2021. Atoll: A Scalable Low-Latency Serverless Platform. In *Proceedings of the ACM Symposium on Cloud Computing* (Seattle, WA, USA) *(SoCC '21)*. Association for Computing Machinery, New York, NY, USA, 138–152. https://doi.org/10.1145/3472883.3486981

[96] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. https://doi.org/10.48550/ARXIV.1409.4842

[97] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. (2019). https://doi.org/10.48550/ARXIV.1905.11946

[98] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. 2009. Wikipedia workload analysis for decentralized hosting. *Computer Networks* (2009).

[99] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *ATC*.

[100] Yanan Yang, Laiping Zhao, Yiming Li, Huanyu Zhang, Jie Li, Mingyang Zhao, Xingzhen Chen, and Keqiu Li. 2022. INFless: A Native Serverless System for Low-Latency, High-Throughput Inference. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (Lausanne, Switzerland) *(ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 768–781. https://doi.org/10.1145/3503222.3507709

[101] Fuxun Yu, Di Wang, Longfei Shangguan, Minjia Zhang, Chenchen Liu, and Xiang Chen. 2022. A survey of multi-tenant deep learning inference on GPU. *arXiv preprint arXiv:2203.09040* (2022).

[102] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. 2017. Deep Layer Aggregation. https://doi.org/10.48550/ARXIV.1707.06484

[103] Yanqi Zhang, Íñigo Goiri, Gohar Irfan Chaudhry, Rodrigo Fonseca, Sameh Elnikety, Christina Delimitrou, and Ricardo Bianchini. 2021. Faster and Cheaper Serverless Computing on Harvested Resources. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles* (Virtual Event, Germany) *(SOSP '21)*. Association for Computing Machinery, New York, NY, USA, 724–739. https://doi.org/10.1145/3477132.3483580

[104] Zhuangzhuang Zhou, Yanqi Zhang, and Christina Delimitrou. 2022. AQUATOPE: QoS-and-Uncertainty-Aware Resource Management for Multi-Stage Serverless Workflows. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3567955.3567960