2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 979-8-3503-8711-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/IPDPS57955.2024.00018

PALDIA: Enabling SLO-Compliant and Cost-Effective Serverless Computing on Heterogeneous Hardware

Vivek M. Bhasi, Aakash Sharma, Shruti Mohanty, Mahmut Taylan Kandemir, Chita R. Das
The Pennsylvania State University
Email: {vmb5204, abs5688, sxm1743, mtk2, cxd12}@psu.edu

***Abstract*—Among the variety of applications (apps) being deployed on serverless platforms, apps such as Machine Learning (ML) inference serving can achieve better performance from leveraging accelerators like GPUs. Yet, major serverless providers, despite having GPU-equipped servers, do not offer GPU support for their serverless functions. Given that serverless functions are deployed on various generations of CPUs already, extending this to various (typically more expensive) GPU generations can offer providers a greater range of hardware to serve incoming requests according to the functions and request traffic. Here, providers are faced with the challenge of selecting hardware to reach a well-proportioned trade-off point between cost and performance. While recent works have attempted to address this, they often fail to do so as they overlook optimization opportunities arising from intelligently leveraging existing GPU sharing mechanisms. To address this point, we devise a heterogeneous serverless framework, PALDIA, which uses a prudent Hardware selection policy to acquire capable, cost-effective hardware and perform intelligent request scheduling on it to yield high performance and cost savings. Specifically, our scheduling algorithm employs hybrid spatio-temporal GPU sharing that intelligently trades off job queueing delays and interference to allow the chosen cost-effective hardware to also be highly performant. We extensively evaluate PALDIA using 16 ML inference workloads with real-world traces on a 6 node heterogeneous cluster. Our results show that PALDIA significantly outperforms state-of-the-art works in terms of Service Level Objective (SLO) compliance (up to 13.3% more) and tail latency (up to ∼50% less), with cost savings up to 86%.**

I. INTRODUCTION

Serverless platforms have risen in popularity over the years, with an increasing variety of apps being deployed on them [5], [49], [50], [74]. The adoption of serverless computing in apps such as Facebook Messenger Bots [30], Amazon Alexa skills [29], and Optical Character Recognition (OCR) [31], not only affirms this, but also points to 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 [54], [60], [86]. Ideally, we would want such apps to be highly SLO compliant, by which we mean that a high percentage (at least >99% [1], [16]) of all requests should meet their SLO targets. Achieving a low tail latency is also critical for these apps as it is an integral part of the Quality of Service (QoS) delivered to the end-user [33], [35], [44], [45].

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 [37], [76], [77]. Despite this, major serverless providers do *not* currently offer direct 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 nodes. While it could be possible to host serverless functions on GPU-based VMs/nodes (interchangeably referred to as GPU instances), this generally costs more than doing so on CPU-only nodes/VMs (interchangeably referred to simply as CPU nodes/instances), primarily due to GPU instances being typically more expensive than CPU instances [7].

However, since serverless platforms can experience occasional request surges during, otherwise, relatively stable and sparse request traffic [9], [75], providers can potentially benefit from leveraging *both* CPU and GPU nodes to serve requests. Here, CPU nodes can serve low request traffic and GPU nodes can serve request surges (due to the (typically) higher achievable throughput for inference jobs on GPUs versus CPUs [17]). This makes sense from a cost perspective as well, since it requires multiple CPU nodes to achieve the same throughput for such jobs, thus, also resulting in higher costs.

Recent research works, such as [47], [48], [61], [65], [68], [69], [73], [86], [89], have attempted to offer heterogeneous serverless computing by also enabling GPU-accelerated serverless functions. However, as we will demonstrate in this paper, these frameworks fail to remain highly SLO compliant while also being cost-effective. A large part of this can be attributed to the inability of these schemes to share GPUs efficiently to service higher request rates since this is usually critical in determining overall performance (as will become evident in Section VI). The specific reasons for the observed performance of these works include the following:

- The majority of the aforementioned works are not designed to be SLO compliant. This includes works like *Molecule* [47], which currently does not support GPU spatial sharing techniques (such as NVIDIA MPS [22]) and hence, can suffer performance degradation due to queuing delays arising from serving workloads (typically as batches), one after the other, via time sharing. This is especially true when *Molecule* is deployed on more cost-effective (typically wimpier) GPUs.
- Even works like *INFless* [86] and *Llama* [69], which use GPU spatial sharing via MPS (aiming to possibly improve

1530-2075/24/$31.00 ©2024 IEEE
DOI 10.1109/IPDPS57955.2024.00018

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

Features	Atoll [80]	Llama [69]	Molecule [47]	INFless [86]	GPUlet [40]	KRISP [41]	ElasticFlow [51]	GPU-enabled FaaS [89]	CE-scaling [85]	PALDIA
Serverless framework	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓
Heterogeneous Hardware support	✗	✓	✓	✓	✗	✗	✓	✓	✗	✓
Overall SLO Compliance	✓	✓	✗	✓	✓	✗	✓	✓	✗	✓
Cost-effective	✗	✓	✗	✓	✗	✗	✗	✗	✓	✓
GPU time-sharing	✗	✗	✓	✗	✓	✗	✓	✓	✗	✓
GPU spatial-sharing	✗	✓	✗	✓	✓	✓	✗	✗	✗	✓
High CPU/GPU node utilization	✗	✓	✗	✓	✓	✓	✓	✓	✗	✓
Satisfactory tail latency	✓	✗	✗	✓	✗	✓	✗	✗	✗	✓

TABLE I: Comparison of PALDIA against a few related works. The features listed here are *only promised* by the works (excluding PALDIA) and may not reflect in the results shown in our evaluation (Section VI).

SLO compliance), suffer from performance degradation due to being agnostic to job interference. Specifically, while serving requests using a GPU, these works only consider whether a batch of requests can be executed *in isolation* on it within the SLO target and hence, *consolidate excessive* workload batches on the GPU, leading to high job interference. This is especially detrimental to performance when using cheaper (typically wimpier) GPUs. Additionally, their interference-agnostic nature may prevent them from leveraging the appropriately-powerful GPU to serve all incoming requests[1].

From the above discussion, we can infer that an ideal heterogeneous framework should (i) select the appropriate hardware to service requests according to the workloads and/or request rates, and (ii) be capable of serving high request rates with cost-effective GPUs efficiently without suffering excessive overheads due to job interference and/or queueing.

To this point, we propose PALDIA, a *heterogeneous serverless framework*, that uses (i) a prudent Hardware Selection module to select cost-effective hardware (CPU and/or GPU nodes) that can serve requests according to the workload and request rates, and (ii) a hybrid time/spatial GPU sharing mechanism that minimizes total job overhead by intelligently trading off job interference and queueing delays, thus, yielding high performance (especially SLO compliance), even when using cost-effective GPUs. To our knowledge, PALDIA is the *first* serverless framework that improves performance by intelligently leveraging *both* the MPS and time sharing capabilities of GPUs. PALDIA also employs other essential features, such as request batching, conservative autoscaling mechanisms and keep-alive policies, to keep the framework performant. Table I compares PALDIA against related works.

We perform an extensive evaluation of PALDIA on a 6× heterogeneous CPU/GPU node cluster with real world traces using 16 ML inference workloads in the domains of vision and language. Our results show that PALDIA significantly outperforms state-of-the-art works in terms of SLO compliance (up to ∼13.3% more) and tail latency (up to ∼50% less), while reducing cost by up to 86%.

[1]ElasticFlow [51], while designed for ML training, also only uses MPS, and would behave similarly to these schemes, if repurposed for user-facing apps.

II. BACKGROUND AND MOTIVATION

This section discusses the state of the art of heterogeneous serverless frameworks and other relevant technologies. Through this, we arrive at the insights that underpin our work.

A. Heterogeneous Serverless Computing

In serverless computing, developers upload their application 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) [36], [84]. Here, function execution may be preceded by a container bootup latency (called the cold start), which can take up to multiple seconds [3], [4]. Typically, the VMs here are hosted on hardware chosen from a range of CPU nodes in the datacenter so as to allow providers flexibility in scheduling functions and improve datacenter utilization [36], [84]. Serverless computing also mitigates resource management overheads for developers, while offering instantaneous scalability and fine-grained billing. These factors have led to serverless becoming a prime candidate for deploying latency-critical, user-facing apps on the cloud.

Why use CPUs *and* GPUs for serverless? Many latency-critical apps (especially ML inference apps) can achieve greater performance using datacenter GPUs due to their architecture [17], [37]. Moreover, improving interconnect technologies continue to lower data movement costs of GPUs, thus, facilitating inference serving on them [21], [25]. However, hosting serverless functions on GPU-based VMs can typically cost more than (traditional) CPU-based hosting since (a) serverless costs are tied to host VM/instance costs [88], (b) in a one-to-one comparison, GPU instances are typically more expensive than CPU instances [7]. Nevertheless, given that serverless platforms can experience occasional bursts of requests during, otherwise, relatively stable and sparse request traffic [9], [75], providers can potentially use both CPUs and GPUs to serve requests, wherein CPUs can be used during low request traffic and GPUs can be used to serve request bursts (since a larger number of jobs can more effectively be served on GPUs versus CPUs [17]). This typically makes monetary sense as well, since serving the same, large number of requests using multiple CPU instances will cost more compared to using a single GPU instance to achieve comparable throughput. For example, in AWS EC2, we observe that serving a *ResNet 50* model at a throughput of ∼750 requests per second (rps) requires at least seven *m4.xlarge* (CPU) instances, costing 86% more versus using a single *g3s.xlarge* (GPU) instance.

> ***Insight 1:*** *Both datacenter CPU and GPU nodes can be used to cost-efficiently service serverless requests according to the dynamic workloads and request rates.*

Despite this potential, major commercial serverless platforms including AWS Lambda [8], Microsoft Azure Functions [19] and Google Cloud Functions [15] continue to offer *only* CPU support for their serverless functions. Recent research works [47], [48], [61], [65], [68], [69], [73], [86], [89], however, have attempted to extend serverless

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

platforms to being heterogeneous by also enabling GPU-accelerated serverless functions. Note that, while there are also GPU-enabled serverless startups, such as *Banana* [10], *Beam* [12], *Pipeline* [26], *Replicate* [28], *Cerebrium* [13] and *trainML* [32], given that their job scheduling techniques are not public, and that some of them only execute on GPUs, we do not draw comparisons against them.

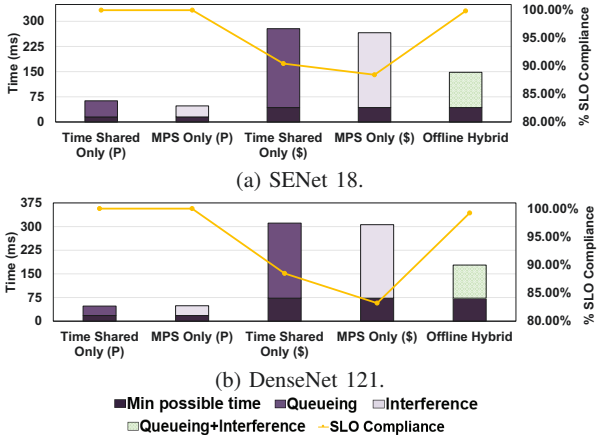
SLO compliance in heterogeneous serverless: Both CPU-enabled [52] and GPU-enabled inference systems [54], [60], [67] have been employed in industrial settings with SLO (latency) targets. However, in the serverless landscape, none of the major commercial providers offer SLOs with respect to latency. Moreover, only one of the GPU serverless startups mentioned earlier, *Banana*, offers any SLO guarantees – though it is at least 13× worse than PALDIA [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], [16], [35], [59]. The majority of heterogeneous serverless research works are also not designed to be highly SLO compliant. This includes works such as *Molecule* [47], which share GPUs via time sharing only and hence, suffer from SLO violations caused due to request queueing. Even works like *INFless* [86] and *Llama* [69], which are designed for SLO compliance, fail to meet SLOs when attempting to also be cost efficient, as we will demonstrate in this paper. This is primarily because these works are agnostic to the job interference arising from spatially sharing GPUs (with MPS) among too many requests. To elucidate the above points, we investigate common GPU sharing mechanisms and their implications next.

B. GPU Resource Sharing

Below, we present two common GPU sharing mechanisms:

Time Sharing: GPU time (temporal) sharing emerged around the mid to late 2000s with the rising need for improved GPU usage in datacenters. Here, jobs are interleaved such that each one gets a time slice to execute on the GPU. While time sharing can ensure that each job does not have to contend with other jobs for GPU resources while executing, it has two major drawbacks: (i) time-sharing GPUs can leave them underutilized, especially when running light-weight tasks common to serverless platforms, and (ii) some jobs can suffer from high queueing delays due to being scheduled later on the GPUs and this may degrade the overall SLO compliance.

Spatial Sharing via Multi-Process Service (MPS): MPS [22], introduced from the Kepler-based NVIDIA GPUs, enables co-operative multi-process CUDA apps to be processed concurrently on the GPU via software-based spatial sharing. MPS divides the GPU compute units, Streaming Multiprocessors (SMs), into multiple partitions such that each partition is dedicated to a user app (process). Thus, multiple apps can execute on the GPU at once, thereby, utilizing it more effectively. However, each app is not isolated from the interference from other apps when using MPS. This interference arises due to the contention for memory bandwidth, caches and capacity, all of which are shared between the concurrent MPS processes. This can potentially cause SLO violations as well.



(a) SENet 18.

(b) DenseNet 121.

Fig. 1: Breakdown of tail (P99) latencies vs. SLO compliance for the considered schemes. Here, 'Min possible time' is the execution time independent of queueing and/or interference.

Quantification of tradeoffs: To appreciate the above discussion, we conduct an experiment comparing various schemes subjected to ML inference workloads using the (relatively stable) Wiki request trace [83]: (i) *SENet 18* [57] ($\mu \approx 575$ rps, batch size: 128), and (ii) *DenseNet 121* [58] ($\mu \approx 160$rps, batch size: 64). Depending on the scheme, the workloads are executed either on an NVIDIA M60 or V100 GPU. We set an SLO of 200ms for all requests for both workloads [86]. The schemes considered are described below.

- *Time Shared Only (P)* executes each workload batch on the most performant GPU (V100) in a time-shared manner.
- *MPS Only (P)* spatially shares the most performant GPU (V100) among all the workloads using MPS only.
- *Time Shared Only ($)* executes workloads on the most cost-effective GPU (M60) in a time-shared manner.
- *MPS Only ($)* spatially shares the most cost-effective GPU (V100) among all the workloads using MPS only.
- *Offline Hybrid* uses both time and spatial sharing (via MPS) to execute only a limited number of batches (determined offline) with spatial sharing and queue the other batches for execution. Here, we use the cost-effective M60 GPU to run the workloads.

Note that the time-shared schemes are variants of *Molecule*'s GPU sharing strategy [47], whereas the spatial-shared schemes represent variations of *INFless/Llama*'s schemes [69], [86]. For the *Offline Hybrid* scheme, we perform a sweep of numerous possible combinations of workload occupancy on the GPU *beforehand*. From these combinations, we pick the number of time/spatial sharing batches which yields the highest overall SLO compliance. The cost values for the compute nodes here are obtained from the pricing of the corresponding AWS EC2 instances [7], with the M60 and V100 nodes priced at \$0.75/hour and \$3.06/hour, respectively.

From Figure 1, we observe that *Offline Hybrid* achieves high SLO compliance ($> 99\%$) while *also* remaining cost-effective. This is because it prudently trades off queueing delays and job interference effects by determining the appropriate number of

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

requests to perform spatial and time sharing, respectively.

For both *SENet 18* and *DenseNet 121* (Figures 1a, and 1b), *MPS Only ($)* achieves up to 16% less SLO compliance compared to *Offline Hybrid*. Upon analysis of their tail latency breakdowns, we infer that this is due to *MPS Only ($)* suffering from up to 2.2x more overhead (due to job interference) versus *Offline Hybrid*. This is a result of *MPS Only ($)* consolidating too many jobs on the M60 GPU. Similarly, *Time Shared Only ($)* also has up to $\sim$11% lower SLO compliance than *Offline Hybrid* due to high queueing delays incurred as a result of time sharing the GPU with too many request batches. *Offline Hybrid*, on the other hand, finds the appropriate tradeoff point (in terms of number of requests) where the combined overhead from interference (due to spatial sharing) and queueing (due to time sharing) is minimal. While *MPS Only (P)* and *Time Shared Only (P)* have up to 0.78% higher SLO compliance than *Offline Hybrid*, they accomplish this by executing on the most performant (and costly) GPU (V100) compared to that of *Offline Hybrid* (M60). As a consequence, both these schemes incur more than 4$\times$ the cost of that of *Offline Hybrid* [7].

> ***Insight 2:*** *Queuing delays and job interference can possibly be traded off to improve SLO compliance on GPUs by choosing the appropriate number of requests to perform spatial and time sharing. This can allow requests to be served effectively on cheaper GPUs.*

Here, Offline Hybrid is infeasible for real-time request serving because: (i) *Offline Hybrid* sweeps through numerous combinations, which is impractical in real-time request serving as the same workload should not be repeatedly executed for every request: there must be a mechanism to predict the ideal number of requests to temporally/spatially share the GPU, and (ii) Since there will be multiple generations of CPUs and GPUs to potentially execute workloads in production datacenters, choosing the appropriate hardware can be time consuming: it would be beneficial to prune the hardware search space. We address these two issues through profiling workloads/hardware and by modeling interference and queueing tradeoffs. This is discussed in detail in the next section.

III. Modeling Interference and Queueing Overheads

Towards predicting the ideal number of requests to perform temporal and spatial GPU sharing, we model the overheads due to queueing and job interference, respectively, that result from this, and find the appropriate *balance* between them to ensure high SLO compliance. For this, we *modify* the job interference model for co-located MPS jobs from *Prophet* [38], to use the number of requests and batch size as input parameters. We also *introduce* the execution time of queued requests as an additional term, which is approximated (with $< 4\%$ error) as the proportionate fraction of the batch execution time. Here, we want the execution time of the requests executed last, T_{max}, to remain within the SLO, where T_{max} is:

$$Solo_M \cdot \frac{y}{BS_M} + Solo_M \cdot \left\{ \frac{(N_M - y)}{BS_M} \cdot FBR_M \right\} < SLO. \quad (1)$$

Here, $Solo_M$ is the execution time of the model M when run in isolation on the concerned GPU, BS_M is the model's batch size, N_M is the number of requests corresponding to M at that given time, FBR_M is the Fractional Bandwidth Requirement (FBR) of the model. For example, an FBR of 0.2 indicates that the job requires 20% of the global memory bandwidth. Here, y requests are queued and $(N_M - y)$ requests are concurrently executed on the GPU via spatial sharing.

As shown in Equation (1), we wish to find a suitable y value so as to keep the execution times of *all* requests within the SLO target. The term $Solo_M \cdot \frac{y}{BS_M}$ (an approximation that we introduce) represents the execution time of y queued requests and $Solo_M \cdot \left\{ \frac{(N_M - y)}{BS_M} \cdot FBR_M \right\}$ (modified from *Prophet*'s model) is the execution time of the other requests that are executed concurrently with spatial GPU sharing (via MPS).

Here, all terms except y in Equation (1) are either known beforehand for the arrived requests (BS_M, SLO, and N_M), or can be obtained through profiling the workloads over time on the GPU ($Solo_M$, and FBR_M). This profiling can also aid in significantly *reducing the hardware search space* when servicing requests as we can identify capable hardware candidates that can serve the given workloads and request rates. Note here that there are additional *constraints* that y must satisfy: (i) $N_M > y$ – this simply implies that y cannot exceed the total number of current requests, and (ii) $\left(\frac{N_M - y}{BS_M} \right) \cdot FBR_M > 1$ – this is to ensure that there are enough requests that are co-located via MPS so that the job interference portion of Equation (1) is valid [38]. Now, given Equation (1) and these constraints (with known values for all other terms), we can obtain *a suitable range of values (which we refer to as the 'optimal range') to find y so as to minimize T_{max}*. In practice, we are able to obtain the best y value with minimal overhead (< 3 ms) through multi-threading by comparing possible T_{max} values obtained by substituting various y values from the optimal range.

For cases where a suitable y value does not exist due to an invalid optimal range, we reattempt the same procedure for the next more performant (and typically more expensive) GPU available which will more likely be capable of serving the same requests within the SLO using our spatial/temporal hybrid scheduling technique. We prefer this to techniques like rate limiting (i.e., reducing N_M in Equation (1), which can cause many requests to violate the SLO (due to throttling) in order to serve the other requests with the current GPU. The above-discussed modeling techniques lay the foundation for the scheduling policies in our design.

IV. Overall Design of Paldia

Figure 2 outlines the overall design of Paldia. The Gateway ❶ provides a point of access to user requests to our serverless framework. These requests are routed to the appropriate worker nodes (chosen by the Hardware Selection module ❷) by the Dispatcher ❸. Depending on the workload and request rates, a small pool of capable hardware candidates (determined through profiling) are explored by the Hardware Selection module to find the cheapest (yet, performant) hardware config-

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

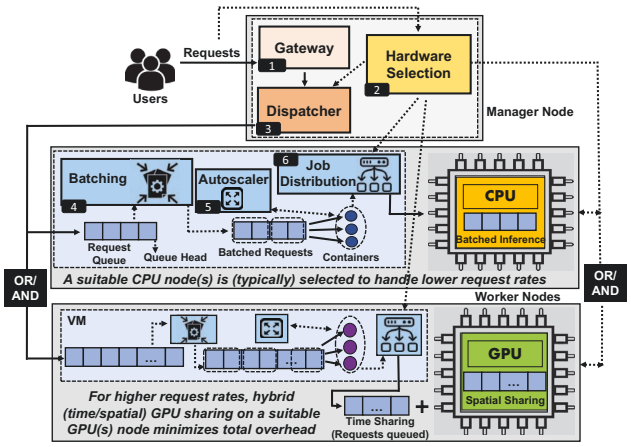


Fig. 2: A schematic depicting PALDIA's design.

uration for upcoming requests. At the selected worker node(s), the requests are first batched ❹ for improved throughput while inference serving. The Autoscaler ❺ appropriately scales containers up or down as required to service each incoming request batch. The containers use the Job Distribution logic ❻ to schedule jobs on the compute hardware of the worker node. In particular, when using GPUs, we use a hybrid time/spatial sharing mechanism where a certain number of requests are serviced concurrently using MPS on the GPU and the remaining requests are queued. We use the techniques described in Section III to determine the appropriate number of requests for time and spatial sharing in order to effectively trade off job interference and queueing delays (Equation (1)). We detail the key design features of PALDIA below.

A. Hardware Selection

Algorithm 1 shows the working of the Hardware Selection module ❷. As mentioned in Section III, from the provider's perspective, we can profile workloads by observing their execution latency values (and other relevant metrics) when executing on various available hardware configurations in order to gain an understanding of the best possible candidate workers for specific workloads and request rate ranges. This pool of candidate hardware configurations is chosen such that each configuration is able to serve requests arriving in the immediate future (∼4s ahead) within the target latency (SLO) ❷. The number of future requests can be estimated using a lightweight statistical model (such as EWMA [80]) which relies on current and history request information (regarding the workload, request rate etc.) ⓐ. We choose the appropriate hardware pool for future requests so as to allow enough time to acquire the hardware to service those requests. Typically, we use CPU nodes to handle lower request rates (up to ∼25 rps for workloads with high FBRs) by selecting the cheapest one that can serve the requests capably ⓒ. For higher request rates, the suitable hardware candidates are usually GPU nodes. From this pool of GPU nodes, the Hardware Selection module first estimates the worst-case execution time (T_{max} from Equation 1) that each GPU will yield based on the workload, future request rate, the hardware itself, and the

Algorithm 1 Hardware Selection.

```
for Every Monitor_Interval= W do
    from curr_request_queue get curr_queue_info
    Hardware_Selection(current_req_info)
procedure HARDWARE_SELECTION(curr_req_info)
    init: HW_dict : null
          wait_ctr : 0
    future_req_info ← predict_req(curr_req_info, history_info) (a)
    HW_pool ← get_HW_pool(future_req_info) (b)
    HW_pool.sort_by_cost_ascending()
    par_for HW in HW_pool do
        cost ← get_cost(HW)
        least_T_max ← ∞
        best_y ← null
        if HW.type is CPU then (c)
            least_T_max ← approx_T_max(HW, future_req_info)
            HW_dict[HW] ← [least_T_max, best_y, cost]
            break
        else
            par_for y in optimal_range(HW, future_req_info) do
                T_max ← calc_T_max(y, HW, future_req_info) (d)
                if T_max < least_T_max then
                    least_T_max ← T_max
                    best_y ← y
            end par_for
        HW_dict[HW] ← [least_T_max, best_y, cost]
    end par_for
    chosen_HW ← choose_best_HW(HW_dict) (e)
    if chosen_HW is not current_HW then (f)
        if wait_ctr >= wait_limit then
            reconfigure_HW(chosen_HW) (g)
        else
            wait_ctr ← wait_ctr + 1
    if curr_HW is chosen_HW then
        wait_ctr ← 0
```

y value ❹ (from Equation (1)). To get the lowest possible T_{max}, we probe multiple possible y values in parallel, and pick the one that yields the lowest T_{max} for each GPU. Given the best achievable execution latency values for the GPU nodes considered, the *choose_best_HW( )* procedure selects the cheapest node that has latency within ∼50ms of that of the most performant one ⓔ (from our experience, this strikes a good balance between cost savings and performance).

Finally, our choice of hardware is compared against the current one ⓕ and we procure our new hardware choice if there is a mismatch between our choice and the current hardware and similar mismatches have happened repeatedly (3 times at least) ⓖ. We do this since multiple mismatches can reveal a trend in variations in the request trace and thus, indicates that our current hardware is likely not capable and/or cost-effective to handle future request rates. To acquire the hardware, we launch a VM on our chosen node and spawn the appropriate number of containers on it using the autoscaling techniques discussed in the next subsection. This is done in the background (so as to tolerate hardware procurement overheads) as the current hardware is used to serve the current requests. Once the containers are spawned on the newly procured hardware, we reroute requests to them and relinquish the current hardware. It is the responsibility of the *reconfigure_HW( )* procedure in Algorithm 1 to perform these actions.

B. Request Batching

To enable high throughput and resource utilization, especially for GPUs, we enable requests to be batch-served ❹ [78],

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

[86] by containers. In order to keep the maximum execution time of a batch within the SLO, we keep an upper bound for the batch sizes we use and this is dependent on the hardware and workloads being considered. We allow flexible batch sizes to be used on-the-fly, especially to facilitate the hybrid time/spatial sharing on GPUs, which could require different, specific number of requests to be queued and/or concurrently executed (via MPS) on the GPU (something which uniform batching would hinder).

C. Autoscaling

PALDIA has an auto-scaling module ❺ which is *re-purposes* to better suit inference apps and especially prevent SLO violations due to container under-provisioning:

Reactive scale-up – Containers are scaled up such that there is one container spawned per batch of requests that will be spatially shared, as determined by the Hardware Selection module. This allows each container to launch jobs in a parallel fashion on the GPU via spatial sharing (MPS). Having fewer containers than batches would lead to the same container serving multiple batches, resulting in request batches being queued, which, in turn, can deteriorate the overall SLO compliance. For time sharing, we only require one container, since each request batch is executed one at a time on the GPU. For this, we can reuse a warm container spawned for the spatially-shared requests. Thus, the total number of containers spawned for all incoming requests, n_c, is given by $n_c = \left\lceil \frac{n_{spatial}(m)}{batch\_size(m)} \right\rceil$, where $n_{spatial}(m)$ is the number of incoming requests that will be spatially shared for the model m, and *batch_size*(m) is the current model batch size.

Predictive scale-up – While reactive scale-up spawns containers that can aid in servicing an unexpected number of requests, the newly-spawned containers only come into play after incurring cold start delays. During this time, existing containers are used to serve the increased load, which can lead to queuing, and thus, result in SLO violations. To avoid complete reliance on reactive scale-up, we pre-warm containers so as to service future request loads that we predict using a lightweight, pluggable model (EWMA in our case) (as in [34], [80]). In effect, predictions about the number of future incoming requests for a model is made every $\sim 10s$ using history information collected in that time window. If more containers are deemed necessary than the current number of containers (using similar criteria as in Reactive scale-up), containers are scaled up as required.

Delayed termination – For increased SLO compliance, we terminate active (warm) containers only after an extended (tunable) period of time ($\sim$10 minutes) elapses throughout which some containers (which will eventually be shut down) are consistently deemed to be 'surplus' by the other scaling policies (similar to existing keep-alive policies [84]). 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.

D. Job Distribution

The containers spawned by the autoscaling module employ the Job Distribution logic ❻ to effectively schedule jobs on the selected hardware. For lower request rates, we service the incoming requests using the native batched CPU execution mode of the ML framework that we use. When scheduling on GPUs (for high request rates), the Job Distributor uses the best y value calculated already by the Hardware Selection module for the selected GPU (which minimizes the maximum end-to-end latency) (Section IV-A) to determine the number of requests that should perform spatial and temporal GPU sharing, respectively (using Equation (1)). The Job Distributor then leverages both the GPU's time and spatial (MPS) sharing mechanisms to schedule the respective number of jobs estimated for each mechanism. The Job Distributor also automatically adjusts the request batch size to enable this.

V. Implementation and Experimental Setup

PALDIA is implemented primarily in Python with its modules being daemon processes that perform their respective tasks, as described in Section IV. Specifically, when using GPUs to service requests, we leverage the default GPU time sharing mechanism as well as the spatial sharing MPS [22] feature (wherever applicable), to spawn GPU-accelerated containers [20] (which use PyTorch v1.1 [66]) on the appropriate node. When using CPU nodes, we use Pytorch's default CPU batched mode execution for servicing requests. We use Docker (v20) swarm [14] to manage the cluster.

Below, we detail some aspects of the experimental setup.

Hardware: PALDIA is set up on a 7 node cluster (including a dedicated manager node). The 6 worker nodes are from AWS EC2 [6] and are equipped with a range of compute hardware, the details of which are listed in Table II.

Name	Primary Compute Hardware	CPU/GPU Memory	Cost
p3.2xlarge	NVIDIA V100 GPU	16 GB	\$3.06/h
p2.xlarge	NVIDIA K80 GPU	12 GB	\$0.9/h
g3s.xlarge	NVIDIA M60 GPU	8 GB	\$0.75/h
c6i.4xlarge	Intel IceLake CPU, 16 vCPUs	32 GB	\$0.68/h
c6i.2xlarge	Intel IceLake CPU, 8 vCPUs	16 GB	\$0.34/h
m4.xlarge	Intel Broadwell CPU, 2 vCPUs	8 GB	\$0.2/h

TABLE II: Worker node details from AWS EC2. Here, 'Primary Compute Hardware' refers to the compute unit used by the node to serve the inference requests,

For our experiments, the cost values reported are primarily based on the total weighted cost of each scheme calculated according to the time spent using each type of compute node (since the experiments are run for the same time with approximately the same number of containers for all schemes). We measure GPU and CPU node utilization using NVIDIA-smi [23], and htop [18], respectively. Here, 'utilization' refers to percentage GPU/CPU non-idle time. For GPU power measurements, we use the *nvtop* tool [24]. Since our public cloud CPU nodes have no Running Average Power Limit (RAPL) interface support, we resort to projecting power measurements from the *powerstat* tool [27] run on the same hardware in a private cluster for the same experiments.

Workloads: We use workloads based on 16 ML inference models in the vision and language domains. For our primary experiments, we use image classification workloads based on the following models: *ResNet 50* [55], *GoogleNet* [81],

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

DenseNet 121 [58], *DPN 92* [39], *VGG 19* [79], *ResNet 18* [55], *MobileNet* [56], *MobileNet V2* [71], *SENet 18* [57], *ShuffleNet V2* [63], *EfficientNet-B0* [82], and *Simplified DLA* [87]. We use a maximum batch size of 128 and the ImageNet 1k [70] dataset. As a part our sensitivity study, we use sequence classification workloads that are based on large language models with very high FBRs, namely, *AlBERT* [62], *BERT* [46], *DistilBERT* [72], and *Funnel-Transformer* [43]. The maximum batch size used here is 8 and the dataset is the Large Movie Review Dataset [64]. We set all workload SLOs to be 200 ms, as reported for inference requests in [86]. Similarly, our batch sizes are selected so that the batch execution latency remains between ∼50-200ms [86].

Request Traces: For the majority of our experiments, we use a sample trace from the Azure serverless traces [9], [75] which represents request arrival behavior in an actual serverless setting. Our chosen trace has a large peak-to-mean ratio (∼673:55), thus, also capturing potential request surges that the serverless platform can be subjected to. The trace duration here is ∼25 minutes, and each experiment conducted using the trace is repeated for 5 times. These ensure that the results are minimally impacted by randomness. We also scale the request rates of the trace according to the workload used, with high-FBR (Section III) vision models (such as *GoogleNet*, *DPN92* etc.) subjected to a peak rate of 225 rps and the other vision models subjected to double the peak (∼450 rps). For the language models, we use a much lighter request trace, with a peak of only 8 rps, owing to their much higher FBRs compared to vision models.

Evaluated schemes: We compare PALDIA against schemes which employ the request serving policies of state-of-the-art GPU-enabled serverless frameworks, namely, *INFless* [86], *Llama* [69], and *Molecule* [47]. Despite being different frameworks, *INFless* and *Llama* both employ MPS to *spatially share* the available GPU(s) by scheduling multiple request batches onto it. They are, however, agnostic of the performance degradation that can occur due to the resultant job interference. *Molecule*, currently, offers minimal GPU support and thus, executes workloads on the GPU(s) via *time sharing* only. For our experiments, we use *cost-effective ($)* and *performance-focused (P)* variants of these schemes: (i) *INFless/Llama ($)* represents a version of *INFless/Llama* where its hardware selection policy chooses the most cost-effective hardware that can serve one batch of requests (for the current request rate) within the SLO, (ii) *INFless/Llama (P)* is a version of those schemes which uses the most performant GPU (V100) to serve requests regardless of the request rate, (iii) Since *Molecule* does not have an explicit hardware selection policy, we have *Molecule (beta) ($)* which represents its request serving mechanism but with the same hardware selection as *INFless/Llama ($)*, and (iv) *Molecule (beta) (P)*, which is a variant that uses the same hardware as *INFless/Llama (P)*.

VI. EVALUATION

This section presents a thorough evaluation of PALDIA. Unless mentioned otherwise, the plots presented pertain to the average values of the concerned metric (with outliers of more than 2.5× the standard deviation from the mean ignored) when using the Azure serverless trace with vision workloads. For isolated examples, similar results are seen for other workloads also.

A. Primary Results

1) ***SLO compliance and Tail Latency*** *:* Figure 3 depicts the percentage of all requests satisfying their SLO target for all vision models and Figure 4 shows the P99 latency values of two characteristically different models. We observe that PALDIA is among the most performant schemes with up to 13.3% more of its requests being SLO-compliant than its competitors (Figure 3). Similarly, we observe PALDIA to also have tail latency that is well within the SLO target (Figure 4). Note that *Molecule (beta) (P)* and *INFless/Llama (P)* only outperform PALDIA (marginally, in terms of SLO compliance, by up to ∼0.8%) due to always using the most performant hardware (the V100 GPU, in our case) and thus, incurring much higher costs (up to 6.9× more than PALDIA, as we will see in the next subsection). PALDIA's high SLO compliance (and low cost) can be attributed primarily to it efficiently leveraging cost-effective hardware (which it procures using its Hardware Selection module) by intelligently using hybrid (time/spatial) GPU sharing as discussed in Sections III and IV. Now, let us delve into the specific reasons for each scheme's performance by considering the breakdown of tail (P99) latency values plotted in Figure 4. When discussing the following results, note that the major differences in performance between the schemes arise when serving high request rates using GPUs, as all schemes are (typically) able to serve low request rates with their selected hardware (CPU and/or GPU nodes, depending on the scheme) in a performant fashion.

When serving high request rates, *INFless/Llama ($)* spatially shares the selected (cheap) GPU among all incoming requests using MPS (Section V). This often results in the co-location of too many requests on the GPU, leading to increased job interference between them, in turn, degrading their collective performance (Figure 4). For instance, for *ResNet 50* (Figure 4a), 76% of the tail latency of *INFless/Llama ($)* is due to the job interference of all co-located requests. Consequently, for the same model, these schemes have an SLO compliance of only 89.43% (Figure 3). On the other hand, PALDIA has 99.55% SLO compliance in this case. From Figure 4a, we can observe that this is mainly due to the reduced total overhead incurred by PALDIA due to both job interference (spatial sharing) and queueing delays (time sharing). PALDIA achieves this by (i) intelligently scheduling the appropriate number of requests to perform spatial and time sharing of the GPU, as discussed in Section III, and (ii) occasionally using more powerful GPU nodes to handle sudden request surges, since PALDIA's Hardware Selection module, unlike that of *INFless/Llama ($)* (and even *Molecule (beta) ($)*), can detect when the job interference can cause SLO violations.

Since *Molecule (beta) ($)* time shares its selected (cost-effective) GPUs to serve requests, it suffers from high queueing overheads, especially at P99 (Figure 4). For example,

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

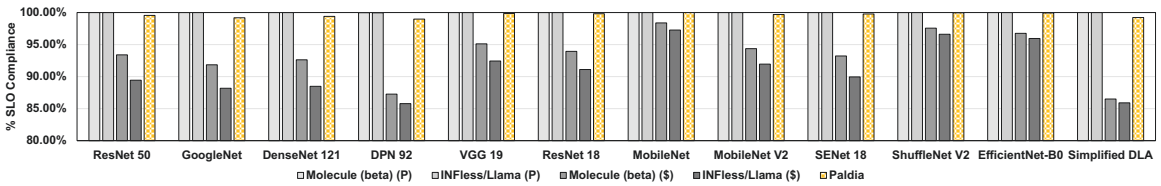
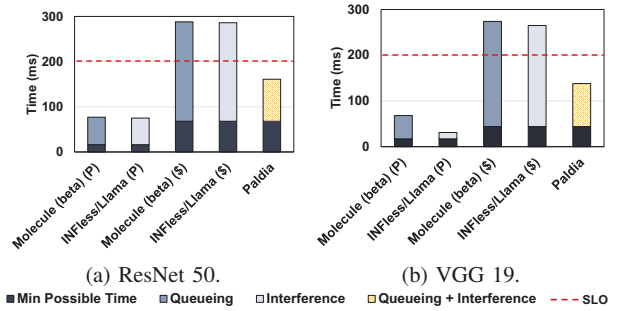
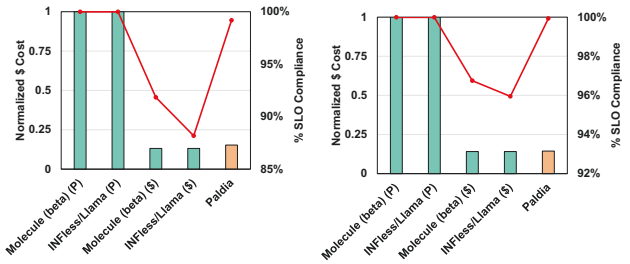


Fig. 3: Comparison of SLO compliance of all schemes for all vision models.



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

Fig. 4: Breakdown of tail (P99) latencies for the considered schemes. Here, 'Min possible time' refers to the execution time of a workload batch (free of interference and queueing) on the selected hardware.



(a) GoogleNet. (b) EfficientNet-B0.

Fig. 5: Comparison of Normalized Cost (in $) vs. SLO compliance across all schemes for the shown models.

for VGG 19, *Molecule (beta) ($)* suffers from up to 84% queueing overhead and as a result, only achieves 95.11% SLO compliance here (Figure 4b). PALDIA, in comparison, achieves higher SLO compliance (99.85%) and lower tail latency (∼50%) than *Molecule (beta) ($)* due to suffering 59% lower total overhead than it (due to its hardware selection and hybrid GPU sharing strategies).

As mentioned earlier, since both *INFless/Llama (P)* and *Molecule (beta) (P)* always use the most performant GPU to serve all requests, they have the highest SLO compliance (99.99% on average) and least tail latency values (< 100ms on average). This is a consequence of the minimal interference/queueing overheads they suffer from as a result of the brawnier GPU they use compared to the other schemes. Despite using cheaper (and less performant) GPUs, PALDIA remains within 0.38% of the SLO compliance of these schemes (on average) by intelligently employing its hybrid scheduling on its selected GPUs. In the next subsection, we detail the cost benefits that PALDIA yields due to this.

*2) **Cost Savings** :* PALDIA, through its Hardware Selection module (described in Section IV-A), acquires cost-efficient hardware to serve requests according to the workload and request rate, without compromising on performance. Due to this, we observe that PALDIA yields cost savings of 85% (on average) versus *INFless/Llama (P)*, and *Molecule (beta) (P)*, which rely solely on the most expensive GPU to serve requests (Figure 5). As can be observed, PALDIA has comparable SLO compliance to these schemes, despite using cheaper (wimpier) GPUs due to its hybrid scheduling, as previously discussed.

Note that the other cost-efficient schemes, *INFless/Llama ($)* and *Molecule (beta) ($)*, select the cheapest available hardware that can serve a batch of requests within the SLO. However, they do not consider the effects of job interference or queueing that results from serving multiple batches of requests at the same time. As a result, although these schemes have the highest cost savings (marginally, by ∼1-3%), they have low SLO compliance (Figure 5). In comparison, at nearly the same cost, PALDIA achieves up to ∼11% more SLO compliance than these schemes. The slightly higher cost of PALDIA (up to ∼ 3%) is due to the fact that its hardware selection module occasionally selects more expensive GPU/CPUs to serve requests (especially during extremely high request rates for workloads with high FBRs (Section III)) to avoid compromising on performance. For instance, in Figure 5a, PALDIA costs 2.4% more than the other cost-effective schemes, since it uses more expensive hardware during ∼3% of the trace duration. Note however that, for models with lower FBRs, such as *EfficientNet B0*, the cost difference between the cost-efficient schemes and PALDIA is minimal (0.3%, in this case, from Figure 5b), since PALDIA can efficiently service requests mostly with the same hardware as that of the others and only switching to more expensive hardware for ∼1% of the time. Given the above results, we believe that PALDIA strikes the right balance between cost savings and SLO compliance.

*3) **Analysis of other Key Benefits** :* Here, we evaluate PALDIA with respect to other metrics.

End-to-End Latency Distribution An analysis of the end-to-end latency distribution can elucidate the observed performance of each of the schemes. Consider Figure 6. PALDIA remains within the SLO for the entirety of the measured range (until P99) owing to its intelligent hybrid time/spatial scheduling and hardware selection policies that allow it to effectively use cost-efficient hardware to serve incoming requests, even during peak request traffic (as discussed previously). As we can observe, *INFless/Llama ($)* and *Molecule (beta) ($)* exceed the SLO at not only the tail (P99), but around P80 as well since they suffer from high job interference and queueing

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

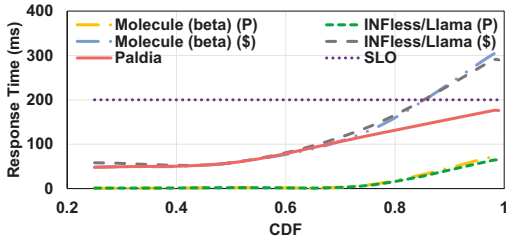


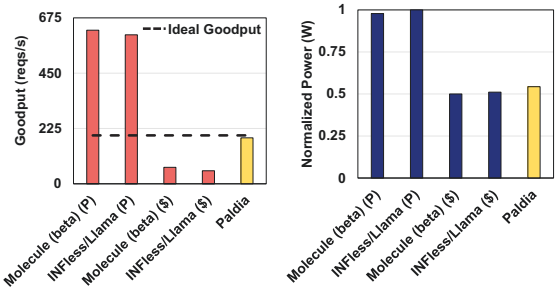
Fig. 6: Cumulative Distribution Function (CDF) of the end-to-end job latencies for all schemes for the *SENet 18* model.

overheads, respectively, during periods of high request traffic. *INFless/Llama (P)*, and *Molecule (beta) (P)*, both have latency curves that are well within the SLO target, even at P99. This is as expected, as they use the most performant GPU to serve all requests. However, as discussed in subsection VI-A2, they incur 6.9 × higher cost on average, in comparison to the other (cost-efficient) schemes, including PALDIA. Also note that PALDIA uses the appropriate cost-efficient hardware and hybrid scheduling that leverages the 'slack' in latency afforded by the latency target, something which *INFless/Llama (P)*, and *Molecule (beta) (P)* fail to do.

Tolerance to Request Surges We observe that the difference in performance between the cost-efficient schemes, including PALDIA, is heavily dependent on their performance during request surges/spikes, since they (typically) are able to serve low request traffic within the SLO. Consider Figure 7a, which compares the *'goodput'* of each scheme during the periods of highest request traffic (up to 225 rps) for *DenseNet 121*. By *goodput*, we refer to the average number of requests that *can* be served within the SLO target. We also depict the average request rate during this period with the dotted line shown.

Ideally, a scheme should be able to achieve a *goodput* that matches the incoming request rate during high traffic. As we can observe, the other cost-efficient schemes, *INFless/Llama ($)*, and *Molecule (beta) ($)*, can only serve 27% and 34% of the incoming (high) request rate within the SLO, respectively. This is because of the high interference/queueing overheads that they suffer from due to relying solely on either spatial/time sharing, and their choice of hardware. In comparison, PALDIA is within 5% of the ideal *goodput*, owing to (i) its intelligent hybrid GPU sharing policy, and (ii) selecting the appropriate hardware (which may be more expensive, occasionally) to service the high request rate. In contrast, *INFless/Llama (P)* and *Molecule (beta) (P)*, over-commit resources to serving the request rate, effectively only bettering PALDIA by ∼0.4% SLO compliance while incurring much higher costs (up to 6.9 ×).

Power Consumption Here, we report the average (normalized) power consumption of each scheme. Generally, we observe the average power consumption of each scheme to be primarily dependent on its selection of hardware to serve requests. From Figure 7b, we observe that PALDIA is among the highly power-efficient schemes, as it consumes 45% lesser power (on average) than *INFless/Llama (P)*, and *Molecule (beta) (P)*. This is because PALDIA can rely on cost/power-efficient hardware to effectively serve requests with very high



(a) Goodput. (b) Power Consumption.

Fig. 7: PALDIA's other benefits: *Goodput* (for *DenseNet 121*), and Power Consumption (for *Simplified DLA*).

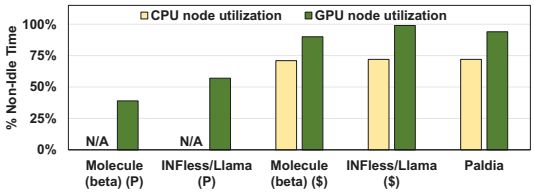


Fig. 8: Compute Node Utilization (non-idle time) comparison of all schemes for the *VGG 19* model.

SLO compliance due to its intelligent time/spatial GPU sharing strategy. Compared to *INFless/Llama ($)*, and *Molecule (beta) ($)*, although PALDIA does consume up to 4% more power (due to occasionally switching to more powerful hardware), the higher performance it affords versus these schemes (as testified by the previous subsections), in our opinion, more than compensates for the marginally higher power consumption.

Resource Utilization Here, we define CPU/GPU node utilization as the non-idle time of the CPU/GPU nodes that are used to service the workload. As per Figure 8, PALDIA, similar to *INFless/Llama ($)*, and *Molecule (beta) ($)*, achieves similar (high) values of CPU node utilization (∼72%) due to using the batched CPU inference mode (supported by the ML framework) to serve low request traffic. Since *INFless/Llama (P)* and *Molecule (beta) (P)* only use the V100 GPU-equipped nodes, this comparison is not applicable to them.

For GPU nodes, we observe that *INFless/Llama ($)* has the highest GPU node utilization (99%) due to two reasons: (i) it uses less powerful (and cheaper) GPUs, thereby, resulting in higher utilization of its (relatively) limited resources, (ii) it spatially shares the GPU among all incoming requests, thus, maximizing its utilization. In comparison, *Molecule (beta) ($)*, despite using the same GPU nodes as *INFless/Llama ($)*, has lower utilization (∼90%) since it executes only one request batch at a time on the GPU, without spatially sharing it. PALDIA, owing to using a hybrid time/spatial GPU sharing mechanism, has a GPU utilization of 94%, which is between that of exclusively using time/spatial sharing. All these schemes have higher GPU node utilization (up to 60%) than both *INFless/Llama (P)*, and *Molecule (beta) (P)* since these '(P) schemes' use nodes with more powerful GPUs that get under-utilized due to not receiving an amount of requests

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

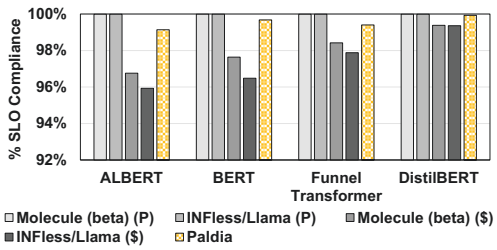


Fig. 9: Comparison of SLO compliance of all schemes for large language models.

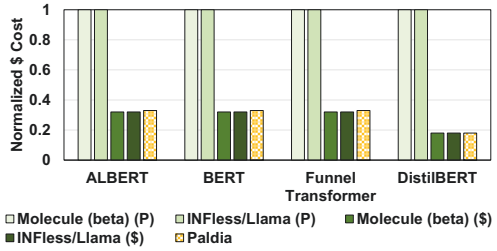


Fig. 10: Cost of all schemes for large language models.

to serve that is commensurate with their compute capabilities.

B. Sensitivity Studies

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

Large Language Models Here, we consider large language models which have significantly higher execution times, memory footprints, and FBRs than those of the vision models previously seen. We observe that all the cost-effective schemes, including PALDIA, select more powerful (and expensive) hardware, in general, to service the requests corresponding to these models, resulting in an average increase in cost by 86% compared to the cost values for the vision models (Figure 10).

Nevertheless, PALDIA and the other cost-effective schemes achieve ∼72% cost savings (on average) versus *INFless/Llama (P)*, and *Molecule (beta) (P)*, as they are able to serve most requests with cheaper hardware. However, since PALDIA leverages its intelligent time/spatial GPU sharing and occasionally switches to more powerful hardware to serve peak request rates (incurring a ∼2% average cost overhead), it achieves an average of 99.54% SLO compliance compared to the 97.73% of *INFless/Llama ($)*, and *Molecule (beta) ($)* (Figure 9). In fact, due to the reasons mentioned in the above discussion, PALDIA remains within 0.45% (on average) of the SLO compliance of the most performant schemes, *INFless/Llama (P)*, and *Molecule (beta) (P)* (Figure 9), by incurring a fraction (29%, on average) of their cost (Figure 10).

Comparison versus Oracle: In this subsection, we compare PALDIA against *Oracle*, an offline, *clairvoyant* scheme with all of PALDIA's policies, but with knowledge of the ideal hardware to use for the request trace (which is also known beforehand), and the best ratio of requests to perform hybrid spatial/temporal scheduling (which can be obtained through multiple offline configuration sweeps). From Figure 11, we observe that PALDIA, despite servicing requests in real time with its non-clairvoyant request arrival and performance prediction models, remains within ∼0.8% of the SLO compliance of *Oracle*, and sometimes nearly matches *Oracle's* performance (with only a 0.1% difference). Note also that *Oracle* has slightly lower cost than PALDIA due to: (i) the additional cost incurred by PALDIA during hardware transition periods and (ii) the performance difference between the two schemes. However, the difference in cost between the schemes is minimal (<1%).

Fig. 11: Cost and SLO compliance: PALDIA versus *Oracle*.

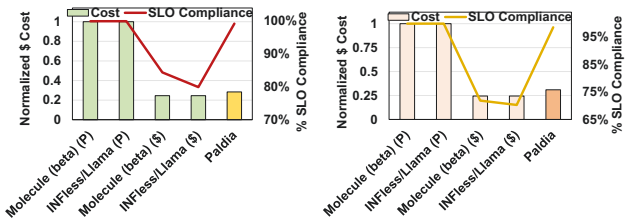
Additional Real-World Traces: We analyze the performance of all schemes when other real-world traces are used.

Realistic Inference Request Arrival Pattern: We use a 5-day-long trace (peak rate scaled to ∼170 rps) from Wikipedia [83] as they capture the diurnal request arrival patterns of ML inference workloads [53]. From Figure 12a, we observe that the other cost-effective schemes, *Molecule (beta) ($)* and *INFless/Llama ($)*, suffer more SLO violations compared to when the serverless trace was used (Figure 3), achieving 84.39%, and 79.93% SLO compliance, respectively. As seen previously, this is primarily because these cost-effective schemes fail to efficiently spatially/temporally share the selected GPU node during peak traffic. This is exacerbated here due to the sustained period of high traffic (∼16 hours per day) of the Wikipedia trace. With only a 4% higher cost than these schemes (due to occasionally using more expensive hardware), PALDIA achieves much higher SLO compliance (99.25%), primarily due to using its hybrid GPU sharing capabilities, as seen earlier. PALDIA is also within ∼0.7% of the SLO compliance of the most performant schemes (which always use the most expensive GPU node), while costing 72% less.

Erratic and Dense Request Arrival Pattern: We use a 90 minute sample from the erratic Twitter trace [2] with an average request rate that is 5× higher than that of the Serverless trace used for previous experiments. As with the Wikipedia trace, the cost-effective schemes, *Molecule (beta) ($)*, and *INFless/Llama ($)*, achieve much lower SLO Compliance values (71.86% and 70.28%, respectively) than PALDIA (with 98.48%) due to being incapable of coping with the high request rates throughout the trace (Figure 12b). This is exacerbated by the erratic nature of the trace. PALDIA remains relatively resilient to these adverse conditions (albeit costing ∼7% more than the other cost-effective schemes) by virtue of its intelligent hardware selection and hybrid GPU sharing mechanisms and policies. PALDIA achieves comparable SLO compliance to that of the most performant (P) schemes (99.82%) as well, despite costing 69% less overall.

Resource Exhaustion Scenario: This study compares the

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

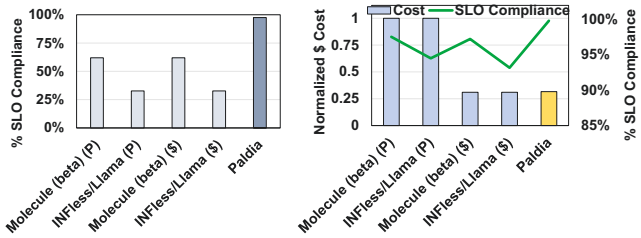


(a) Wikipedia, ResNet 50. (b) Twitter, DPN 92.

Fig. 12: Cost vs. SLO compliance of all schemes for the above realistic traces and workloads.

performance of all schemes during a scenario of Resource Exhaustion (R. Exhaustion), where the request rate is high enough such that even the most powerful GPU cannot serve all incoming requests concurrently within the target SLO. For this, we use the *GoogleNet* workload with a synthetic Poisson arrival trace (mean is ∼700 rps), in an attempt to overwhelm even our most capable GPU (V100). Consequently, all schemes resort to using the V100 GPU to serve requests (thereby, costing the same) since using less capable hardware results in much worse SLO compliance. From Figure 13a, we observe that both schemes which solely spatially share the GPU among all incoming requests, *INFless/Llama ($)/(P)*, only achieve ∼33% SLO compliance due to the high job interference of the concurrent requests for the majority of the trace duration. The schemes that time-share the GPU, *Molecule (beta) ($)/(P)*, while more performant than the aforementioned spatial sharing schemes due to limiting the number of concurrent jobs running on the GPU, still only achieve ∼62% SLO compliance due to the excessive queueing delays faced by requests at the back of the queue. PALDIA, by virtue of its hybrid GPU sharing policy, appropriately manages GPU occupancy so as to prudently trade off job interference and queueing delays, thus, yielding the best SLO compliance among all schemes (97.55%).

Node Failure Scenario: Here, we analyze the performance of all schemes during a scenario where the node being used by a scheme is made unavailable through an induced failure at every minute, and stays unavailable for an entire minute. To cope with node failures, we modify all schemes to switch to the more performant hardware with the least cost so as to not compromise on performance. Note that if a scheme is using the most performant hardware when the failure occurs, it switches to the next best GPU. From Figure 13b, we observe that the cost-effective schemes, including PALDIA, achieve higher SLO compliance than before (as in Figure 3). In fact, PALDIA achieves the highest SLO compliance (99.82%) here. This is because, despite minor SLO violations due to temporary node unavailability, switching to more performant hardware during failures improves the SLO compliance overall for these schemes. However, *INFless/Llama (P)* and *Molecule (P)* achieve worse SLO compliance (at most only 97.55%), since they are forced to use less performant hardware during node failures. Note that, here PALDIA, while being more performant than these schemes, also costs ∼70% lesser.



(a) R. Exhaustion, GoogleNet. (b) Node failures, DenseNet 121.

Fig. 13: Performance and/or cost of all schemes under the above adverse scenarios for the shown workloads.

Mixed Workloads: This study analyzes the effect of co-location of 'regular' CPU-bound serverless workloads (namely, *file compression*, *dynamic HTML generation*, and *image thumbnailing*, from the *SeBS* Serverless Benchmark Suite [42]) with the inference workloads. Here, we observe that the SLO compliance of all cost-effective schemes, including PALDIA, deteriorate (up to ∼10%) due to interference from these 'regular' workloads running concurrently on the host CPU of each node (Table III). The interference effects are especially pronounced when the cost-effective schemes use CPU-only nodes, since there is direct contention for the host CPU's resources from the regular workloads. Despite this, PALDIA achieves ∼95% SLO compliance due to its choice of hardware for certain parts of the trace. *INFless/Llama (P)* and *Molecule (P)* suffer the least performance deterioration since they exclusively use the most performant GPU (but cost 6.9× versus PALDIA). PALDIA's performance can likely be improved by incorporating the interference effects of co-resident CPU-bound workloads into our existing performance model (which currently only accounts for GPU workload interference). We leave this for future work.

Molecule(beta)(P)	INFless/Llama(P)	Molecule(beta)($)	INFless/Llama($)	PALDIA
99.99%	99.99%	76.44%	75.83%	94.78%

TABLE III: Resultant SLO compliance due to interference from 'regular' serverless workloads.

VII. CONCLUDING REMARKS

The CPU-only and GPU nodes that are prevalent in cloud datacenters can be leveraged for SLO compliant and cost-effective serverless computing. To this end, we introduce PALDIA, a heterogeneous serverless framework, which employs: (i) cost-effective hardware acquisition policies that reduce costs by choosing the appropriately-capable hardware to serve dynamic workloads and/or request rates, and (ii) an intelligent hybrid spatio-temporal GPU sharing technique that keeps the framework highly SLO compliant by prudently trading off job interference and queueing overheads. Our evaluation results show that PALDIA outperforms state-of-the-art works in terms of SLO compliance (up to 13.3% more) and tail latency (up to ∼50% less), while reducing costs by up to 86%.

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] "Establishing Effective SLOs." 2020, https://www.datadoghq.com/blog/establishing-service-level-objectives/.

[2] "Twitter stream traces," https://archive.org/details/twitterstream, 2020, accessed: 2020-05-07.

[3] "AWS Lambda Cold Starts." 2021, https://mikhail.io/serverless/coldstarts/aws/.

[4] "Azure Functions Cold Starts." 2021, https://mikhail.io/serverless/coldstarts/azure/.

[5] "The State of Serverless." 2022, https://www.datadoghq.com/state-of-serverless/.

[6] "AWS EC2 Instance Types." 2023, https://aws.amazon.com/ec2/instance-types/.

[7] "AWS EC2 Pricing." 2023, https://aws.amazon.com/ec2/pricing/on-demand/.

[8] "AWS Lambda," 2023, https://aws.amazon.com/lambda/.

[9] "Azure Public Dataset." 2023, https://github.com/Azure/AzurePublicDataset.

[10] "Banana." 2023, https://docs.banana.dev/banana-docs/.

[11] "Banana Latency Guarantees." 2023, https://www.banana.dev/banana-vs-modal-comparison.

[12] "Beam." 2023, https://www.beam.cloud/.

[13] "Cerebrium." 2023, https://docs.cerebrium.ai/introduction.

[14] "Docker Swarm." 2023, https://docs.docker.com/engine/swarm/.

[15] "Google Cloud Functions," 2023, https://cloud.google.com/functions.

[16] "Google SRE Book: Implementing SLOs." 2023, https://sre.google/workbook/implementing-slos/.

[17] "GPUs vs CPUs for deployment of deep learning models." 2023, https://azure.microsoft.com/en-us/blog/gpus-vs-cpus-for-deployment-of-deep-learning-models/.

[18] "htop: an interactive process viewer." 2023, https://htop.dev/.

[19] "Microsoft Azure Serverless Functions," 2023, https://azure.microsoft.com/en-us/services/functions/.

[20] "NVIDIA-Docker." 2023, https://github.com/NVIDIA/nvidia-docker.

[21] "NVIDIA Grace Hopper Superchip Architecture In-Depth." 2023, https://developer.nvidia.com/blog/nvidia-grace-hopper-superchip-architecture-in-depth/.

[22] "NVIDIA Multi-Process Service." 2023, https://docs.nvidia.com/deploy/mps/index.html.

[23] "NVIDIA-smi." 2023, https://developer.nvidia.com/nvidia-system-management-interface.

[24] "NVTOP." 2023, https://github.com/Syllo/nvtop.

[25] "PCIe Special Interest Group PCIe 6 Specification." 2023, https://pcisig.com/pci-express-6.0-specification.

[26] "Pipeline." 2023, https://docs.pipeline.ai/docs.

[27] "Powerstat tool." 2023, https://manpages.ubuntu.com/manpages/focal/man8/powerstat.8.html.

[28] "Replicate." 2023, https://replicate.com/docs.

[29] "Serverless Application Lens: Alexa Skills." 2023, https://docs.aws.amazon.com/wellarchitected/latest/serverless-applications-lens/alexa-skills.html.

[30] "Serverless Facebook Messenger Bot." 2023, https://github.com/pmuens/serverless-facebook-messenger-bot.

[31] "Serverless Optical Character Recognition (OCR) Tutorial." 2023, https://cloud.google.com/functions/docs/tutorials/ocr.

[32] "trainML." 2023, https://docs.trainml.ai/.

[33] D. S. Berger, B. Berg, T. Zhu, S. Sen, and M. Harchol-Balter, "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)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 195–212. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/berger

[34] V. M. Bhasi, J. R. Gunasekaran, A. Sharma, M. T. Kandemir, and C. Das, "Cypress: Input size-sensitive container provisioning and request scheduling for serverless platforms," in *Proceedings of the 13th Symposium on Cloud Computing*, ser. SoCC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 257–272. [Online]. Available: https://doi.org/10.1145/3542929.3563464

[35] V. M. Bhasi, J. R. Gunasekaran, P. Thinakaran, C. S. Mishra, M. T. Kandemir, and C. Das, "Kraken: Adaptive container provisioning for deploying dynamic dags in serverless platforms," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 153–167. [Online]. Available: https://doi.org/10.1145/3472883.3486992

[36] M. Brooker, A. Florescu, D.-M. Popa, R. Neugebauer, A. Agache, A. Iordache, A. Liguori, and P. Piwonka, "Firecracker: Lightweight Virtualization for Serverless Applications," in *NSDI*, 2020.

[37] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim *et al.*, "A cloud-scale acceleration architecture," in *2016 49th Annual IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.

[38] Q. Chen, H. Yang, M. Guo, R. S. Kannan, J. Mars, and L. Tang, "Prophet: Precise qos prediction on non-preemptive accelerators to improve utilization in warehouse-scale computers," *SIGARCH Comput. Archit. News*, vol. 45, no. 1, p. 17–32, apr 2017. [Online]. Available: https://doi.org/10.1145/3093337.3037700

[39] Y. Chen, J. Li, H. Xiao, X. Jin, S. Yan, and J. Feng, "Dual path networks," 2017. [Online]. Available: https://arxiv.org/abs/1707.01629

[40] S. Choi, S. Lee, Y. Kim, J. Park, Y. Kwon, and J. Huh, "Serving heterogeneous machine learning models on Multi-GPU servers with Spatio-Temporal sharing," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 199–216. [Online]. Available: https://www.usenix.org/conference/atc22/presentation/choi-seungbeom

[41] M. Chow, A. Jahanshahi, and D. Wong, "Krisp: Enabling kernel-wise right-sizing for spatial partitioned gpu inference servers," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 624–637.

[42] M. Copik, G. Kwasniewski, M. Besta, M. Podstawski, and T. Hoefler, "Sebs: A serverless benchmark suite for function-as-a-service computing," in *Proceedings of the 22nd International Middleware Conference*, ser. Middleware '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 64–78. [Online]. Available: https://doi.org/10.1145/3464298.3476133

[43] Z. Dai, G. Lai, Y. Yang, and Q. V. Le, "Funnel-transformer: Filtering out sequential redundancy for efficient language processing," 2020. [Online]. Available: https://arxiv.org/abs/2006.03236

[44] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, p. 74–80, feb 2013. [Online]. Available: https://doi.org/10.1145/2408776.2408794

[45] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 205–220. [Online]. Available: https://doi.org/10.1145/1294261.1294281

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

[47] D. Du, Q. Liu, X. Jiang, Y. Xia, B. Zang, and H. Chen, "Serverless computing on heterogeneous computers," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 797–813. [Online]. Available: https://doi.org/10.1145/3503222.3507732

[48] H. Fingler, Z. Zhu, E. Yoon, Z. Jia, E. Witchel, and C. J. Rossbach, "Dgsf: Disaggregated gpus for serverless functions," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 739–750.

[49] S. Fouladi, F. Romero, D. Iter, Q. Li, S. Chatterjee, C. Kozyrakis, M. Zaharia, and K. Winstein, "From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, Jul. 2019, pp. 475–488. [Online]. Available: http://www.usenix.org/conference/atc19/presentation/fouladi

[50] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein, "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)*. Boston, MA: USENIX Association, Mar. 2017, pp. 363–376. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi

[51] D. Gu, Y. Zhao, Y. Zhong, Y. Xiong, Z. Han, P. Cheng, F. Yang, G. Huang, X. Jin, and X. Liu, "Elasticflow: An elastic serverless training

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

platform for distributed deep learning," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 266–280. [Online]. Available: https://doi.org/10.1145/3575693.3575721

[52] A. Gujarati, S. Elnikety, Y. He, K. S. McKinley, and B. B. Brandenburg, "Swayam: Distributed Autoscaling to Meet SLAs of Machine Learning Inference Services with Resource Efficiency," in *USENIX Middleware Conference*, 2017.

[53] J. R. Gunasekaran, C. S. Mishra, P. Thinakaran, B. Sharma, M. T. Kandemir, and C. R. Das, "Cocktail: A multidimensional optimization for model serving in cloud," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 1041–1057. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/gunasekaran

[54] U. Gupta, S. Hsia, V. Saraph, X. Wang, B. Reagen, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "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*, ser. ISCA '20. IEEE Press, 2020, p. 982–995. [Online]. Available: https://doi.org/10.1109/ISCA45697.2020.00084

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

[56] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: https://arxiv.org/abs/1704.04861

[57] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-excitation networks," 2017. [Online]. Available: https://arxiv.org/abs/1709.01507

[58] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2016. [Online]. Available: https://arxiv.org/abs/1608.06993

[59] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, "Grandslam: Guaranteeing slas for jobs in microservices execution frameworks," in *EuroSys*, 2019.

[60] L. Ke, U. Gupta, M. Hempstead, C.-J. Wu, H.-H. S. Lee, and X. Zhang, "Hercules: Heterogeneity-aware inference serving for at-scale personalized recommendation," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 141–154.

[61] J. Kim, T. J. Jun, D. Kang, D. Kim, and D. Kim, "Gpu enabled serverless computing framework," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2018, pp. 533–540.

[62] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," 2019. [Online]. Available: https://arxiv.org/abs/1909.11942

[63] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," 2018. [Online]. Available: https://arxiv.org/abs/1807.11164

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

[65] D. M. Naranjo, S. Risco, C. de Alfonso, A. Pérez, I. Blanquer, and G. Moltó, "Accelerated serverless computing based on gpu virtualization," *Journal of Parallel and Distributed Computing*, vol. 139, pp. 32–42, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731519303533

[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "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., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[67] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 446–459.

[68] S. Risco and G. Moltó, "Gpu-enabled serverless workflows for efficient multimedia processing," *Applied Sciences*, vol. 11, no. 4, p. 1438, 2021.

[69] F. Romero, M. Zhao, N. J. Yadwadkar, and C. Kozyrakis, "Llama: A heterogeneous serverless framework for auto-tuning video analytics pipelines," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1–17. [Online]. Available: https://doi.org/10.1145/3472883.3486972

[70] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[71] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2018. [Online]. Available: https://arxiv.org/abs/1801.04381

[72] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2019. [Online]. Available: https://arxiv.org/abs/1910.01108

[73] K. Satzke, I. E. Akkus, R. Chen, I. Rimac, M. Stein, A. Beck, P. Aditya, M. Vanga, and V. Hilt, "Efficient gpu sharing for serverless workflows," in *Proceedings of the 1st Workshop on High Performance Serverless Computing*, ser. HiPS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 17–24. [Online]. Available: https://doi.org/10.1145/3452413.3464785

[74] J. Schleier-Smith, V. Sreekanti, A. Khandelwal, J. Carreira, N. J. Yadwadkar, R. A. Popa, J. E. Gonzalez, I. Stoica, and D. A. Patterson, "What serverless computing is and should become: The next phase of cloud computing," *Commun. ACM*, vol. 64, no. 5, p. 76–84, apr 2021. [Online]. Available: https://doi.org/10.1145/3406011

[75] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 2020, pp. 205–218. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/shahrad

[76] A. Sharma, V. M. Bhasi, S. Singh, R. Jain, J. R. Gunasekaran, S. Mitra, M. T. Kandemir, G. Kesidis, and C. R. Das, "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)*, 2023, pp. 1–12.

[77] A. Sharma, V. M. Bhasi, S. Singh, G. Kesidis, M. T. Kandemir, and C. R. Das, "Gpu cluster scheduling for network-sensitive deep learning," *arXiv preprint arXiv:2401.16492*, 2024.

[78] H. Shen, L. Chen, Y. Jin, L. Zhao, B. Kong, M. Philipose, A. Krishnamurthy, and R. Sundaram, "Nexus: A gpu cluster engine for accelerating dnn-based video analysis," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 322–337. [Online]. Available: https://doi.org/10.1145/3341301.3359658

[79] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: https://arxiv.org/abs/1409.1556

[80] A. Singhvi, A. Balasubramanian, K. Houck, M. D. Shaikh, S. Venkataraman, and A. Akella, "Atoll: A scalable low-latency serverless platform," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SoCC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 138–152. [Online]. Available: https://doi.org/10.1145/3472883.3486981

[81] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014. [Online]. Available: https://arxiv.org/abs/1409.4842

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

[83] G. Urdaneta, G. Pierre, and M. Van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, 2009.

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.

[84] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *ATC*, 2018.
[85] H. Wu, J. Deng, H. Fan, S. Ibrahim, S. Wu, and H. Jin, "Qos-aware and cost-efficient dynamic resource allocation for serverless ml workflows," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2023, pp. 886–896.
[86] Y. Yang, L. Zhao, Y. Li, H. Zhang, J. Li, M. Zhao, X. Chen, and K. Li, "Infless: A native serverless system for low-latency, high-throughput inference," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 768–781. [Online]. Available: https://doi.org/10.1145/3503222.3507709
[87] F. Yu, D. Wang, E. Shelhamer, and T. Darrell, "Deep layer aggregation," 2017. [Online]. Available: https://arxiv.org/abs/1707.06484
[88] Y. Zhang, I. Goiri, G. I. Chaudhry, R. Fonseca, S. Elnikety, C. Delimitrou, and R. Bianchini, "Faster and cheaper serverless computing on harvested resources," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, ser. SOSP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 724–739. [Online]. Available: https://doi.org/10.1145/3477132.3483580
[89] M. Zhao, K. Jha, and S. Hong, "Gpu-enabled function-as-a-service for machine learning inference," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 918–928. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IPDPS54959.2023.00096

Authorized licensed use limited to: Penn State University. Downloaded on October 29,2024 at 19:14:59 UTC from IEEE Xplore. Restrictions apply.