HeROfake: Heterogeneous Resources Orchestration in a Serverless Cloud – An Application to Deepfake Detection

Vincent Lannurien, Laurent d'Orazio, Olivier Barais, Esther Bernard, Olivier Weppe, Laurent Beaulieu, Amine Kacete, Stéphane Paquelet, Jalil Boukhobza

HeROfake: Heterogeneous Resources Orchestration in a Serverless Cloud – An Application to Deepfake Detection

Vincent Lannurien*‡, Laurent d'Orazio*†, Olivier Barais*†,
Esther Bernard*, Olivier Weppe*, Laurent Beaulieu*, Amine Kacete*,
Stéphane Paquelet*, Jalil Boukhobza*‡
*b<>com Institute of Research and Technology, †Univ. Rennes, Inria, CNRS, IRISA.
‡ENSTA Bretagne, Lab-STICC, CNRS, UMR 6285, F-29200 Brest. France
Email: vincent.lannurien@ensta-bretagne.org,
{esther.bernard, olivier.weppe, laurent.beaulieu, amine.kacete, stephane.paquelet}@b-com.com,
{laurent.dorazio, olivier.barais}@irisa.fr, jalil.boukhobza@ensta-bretagne.fr

***Abstract*—Serverless is a trending service model for cloud computing. It shifts a lot of the complexity from customers to service providers. However, current serverless platforms mostly consider the provider's infrastructure as homogeneous, as well as the users' requests. This limits possibilities for the provider to leverage heterogeneity in their infrastructure to improve function response time and reduce energy consumption. We propose a heterogeneity-aware serverless orchestrator for private clouds that consists of two components: the autoscaler allocates heterogeneous hardware resources (CPUs, GPUs, FPGAs) for function replicas, while the scheduler maps function executions to these replicas. Our objective is to guarantee function response time, while enabling the provider to reduce resource usage and energy consumption. This work considers a case study for a deepfake detection application relying on CNN inference. We devised a simulation environment that implements our model and a baseline Knative orchestrator, and evaluated both policies with regard to consolidation of tasks, energy consumption and SLA penalties. Experimental results show that our platform yields substantial gains for all those metrics, with an average of 35% less energy consumed for function executions while consolidating tasks on less than 40% of the infrastructure's nodes, and more than 60% less SLA violations.**

***Index Terms*—deepfake, serverless, allocation, scheduling, SLA, energy consumption, heterogeneous resources, workload characterization, GPU, FPGA**

I. Introduction

Serveless model. Serverless can be understood as both a programming model, called Function as a Service (FaaS), and a deployment model for the cloud. In such a model, developers design their applications as a composition of stateless functions which execution is event-driven [9]. Serverless services free tenants from complex resource reservation as they are designed to handle on-demand scaling requirements.

In the FaaS model, providers only bill customers according to their actual resources usage [10]. They are fully responsible for deploying intelligent resource management and multiplexing at a finer granularity to optimize Quality of Service (QoS) metrics such as response time, energy consumption, etc.

Deepfake detection and serverless. The work presented in this paper was part of a project (at the b<>com research institute [1]) aimed at deploying an energy efficient deepfake detection service in a heterogeneous cloud. Deepfakes are synthetic images, videos or speeches, digitally created to mimic an existing person so as to mislead viewers. Deepfake detection consists in training a neural network (CNNs) to detect patterns of inconsistencies that are introduced in the creation process.

The functions used by our deepfake application satisfy three main characteristics for suitable serverless workloads [11]: their execution can be made parallel (several independent images and videos), they are stateless (pure transformation on input data) and event-driven (launched after data upload).

Hardware heterogeneity in the cloud. Cloud infrastructures are more and more heterogeneous to fit the needs of data-intensive applications such as machine learning model training or big data analytics [12]. However, specialized processors and GPUs are yet to be made available to customers in serverless offerings [13]. Hardware acceleration should be decided by the provider on a per-application or per-request basis.

State-of-the-art work shows that using such hardware in a cloud setting provides substantial gains in execution time and energy consumption [14], [15]. However, reference orchestrators such as Kubernetes with Knative or OpenWhisk lack the support for dynamic allocation of such hardware.

Performance challenge for serverless deployment. Due to the transient nature of unreserved FaaS resources, latency, throughput and continuity of service are hard to guarantee [16], [17]. When applications do not receive incoming requests, function sandboxes are destroyed instead of being kept in an idle state. Then, when a new request arrives, the

This work was supported by the Institute of Research and Technology b<>com, dedicated to digital technologies, funded by the French government through the ANR Investment referenced ANR-A0-AIRT-07.

[1]https://b-com.com/en

TABLE I
STATE OF THE ART WORK ON AUTOSCALING PLATFORMS

	Serverless	Target cloud platform	SLA	Hardware heterogeneity	Resources usage	Energy consumption	Cost-aware
Swayam [1]	✗	Private (Azure, in-house)	✓	✗	✓	✗	✗
Pigeon [2]	✓	Private	✗	✓	✓	✗	✗
MArk [3]	✗	Public (AWS)	✓	✓	✓	✗	✓
ENSURE [4]	✓	Private	✗	✗	✓	✗	✓
Mampage et al. [5]	✓	Private	✓	✗	✓	✗	✓
Atoll [6]	✓	Private	✓	✗	✗	✗	✗
INFless [7]	✓	Private	✓	✗	✓	✗	✓
SMIF [8]	✓	Private	✓	✓	✓	✗	✗
Target solution	✓	Private	✓	✓	✓	✓	✓

provider has to (re)allocate resources and initialize functions to deploy new sandboxes: this is called a cold start. Cold start times are very penalizing for the application performance, they may even dominate the total execution times [18].

Furthermore, in current commercial serverless offerings, Service-Level Agreements (SLAs) are usually limited to automated retries (restarts) on failure, and FaaS providers generally limit the execution time of serverless functions to a few minutes. The absence of QoS guarantees in commercial serverless offerings prevents them from being more widely used [19].

Problem statement – putting it all together. The problem we try to solve in this paper is to determine how to automatically and reactively scale heterogeneous hardware resources in a cloud in adequacy with the application's load and the users' QoS requirements, while keeping the cost in resources and energy as low as possible for the provider. We consider a deepfake application as a case study in our work.

State-of-the-art. Previous studies have explored the need for an autoscaling platform that supports short-running tasks comprised in applications such as Machine Learning as a Service. Table I summarizes how these solutions differ from the target platform we are trying to achieve, and section VI provides further details. While many have established the need for on-demand acceleration as a solution to guaranteeing function response time, none have measured the impact of leveraging heterogeneous resources on dynamic energy consumption. Furthermore, previous studies consider task consolidation as a means to free resources for further computations – we argue that such techniques open possibilities for the service provider to apply power saving policies in private clouds. Finally, as serverless platforms are general purpose and designed to be highly configurable, our target solution should be cost-aware to allow the provider to make configuration choices pertaining to their own objectives.

Our contribution. We argue that opportunistically taking advantage of hardware accelerators (GPUs and FPGAs) to schedule deepfake detection tasks may allow cloud providers to guarantee serverless tasks response time and achieve SLA while reducing resource usage and energy consumption.

In this paper, we propose a full framework to deploy a deepfake detection application on a serverless cloud. This framework comprises an offline and an online phase. The **offline phase** is used to characterize the performance and energy behavior of the deployed heterogeneous hardware platforms. The **online phase** consists of an autoscaling platform and a scheduling strategy that make efficient use of (characterized) heterogeneous hardware resources to achieve per-request SLAs while reducing the energy consumption of the platform.

For this case study, we devised a simulation environment[2] that models the infrastructure for a deepfake detection application, run by the provider as a Software as a Service using a serverless infrastructure.

Some performance figures. With our allocation and scheduling policy, we were able to handle 50000 tasks in the same makespan as Knative with less than 36% QoS penalties. Our framework reduces energy consumption for the execution of tasks by almost 35%, and provides the opportunity for the provider to further reduce static power consumption by consolidating tasks on less than 29% of the available nodes.

The paper is organized as follows: in a first section, we describe the overall platform model for the project. Then, we describe the execution platform and workload characterization phase. In section 3 we describe the challenges of serverless resource orchestration, our task model and the orchestrator's allocation and scheduling policies. Section 4 presents our evaluation methodology and a discussion of the experimental results. Section 5 gives details regarding state-of-the-art work on autoscaling platforms. Finally, we conclude with some perspectives for future work.

II. Deploying Deepfake Detection Tasks in Serverless Cloud

This section introduces our the used serverless platform model and the overall project.

A. Platform model

We consider a deepfake detection system that is deployed as a serverless application consisting of three stateless functions that achieve inference tasks on input images. These images are all RGB and $224 * 224$ pixels in size [3].

Figure 1 introduces the used platform, we differentiate between an *offline* (blue blox in the figure) and an *online* (green box in the figure) phase. During the offline phase, we collect metadata relative to tasks execution on the heterogeneous

[2]The simulator repository will be made publicly available.
[3]Note that videos are not yet considered in our project.

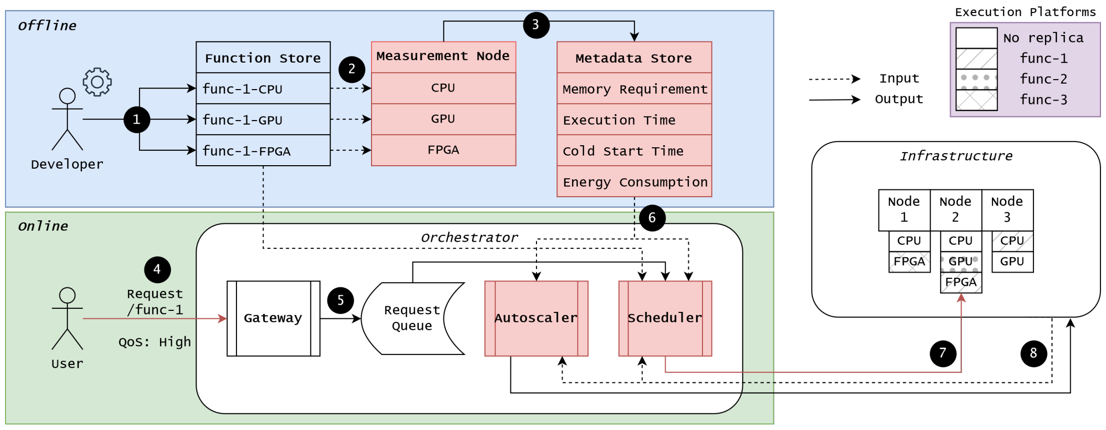


Fig. 1. Serverless deepfake detection platform, system overview

accelerators; during the online phase, we allocate resources and schedule tasks.

Function invocation requests from the users are received on the provider's gateway and handled by the orchestrator. In our model, a function invocation corresponds to a *task*. The user selects one of the three provided models (ResNet50, VGG16 and VGG19, see Section III-B) and uses it to detect a possible deepfake on a picture.

The cloud provider's infrastructure is modeled as a set of heterogeneous *nodes* (section II-A1) comprising various combinations of *platforms* (section II-A2) that can execute incoming *tasks* (section IV-B).

1) Nodes: A node is a server available in the service provider's infrastructure. In this work, we do not consider storage and data locality. Input data are always provided *via* file upload by the user at the time of their request. As such, the only characteristic that defines a node in our infrastructure model is the size of dedicated memory. A node consists of a set of execution platforms defined hereafter.

2) Execution platforms: An execution platform is a hardware processing unit available on a node. Each platform consumes a quantity of energy in the "idle" state expressed in kilowatt-hour (kWh). When it starts executing a task, it consumes additional energy characterized by the task's properties/type: it is then in an "active" state. We differentiate "idle" and "active" time for each platform, so as to measure resources usage. Platforms are characterized by a *platform type* that encompasses the following parameters:

- *Hardware type* – CPU, GPU or FPGA;
- *Price* – the cost of acquisition of such a platform by the cloud provider;
- *Idle energy* – the baseline energy consumption for the platform when it is not running any task.

Task caching and cold start model. We consider a simple task caching mechanism at platform-level, akin to a keep-alive mechanism [20]. In our system, if a platform was previously executing a task of type t, and a new task of the same type t is scheduled on that same platform, then the cold start delay will not be incurred. However, if that same platform were to execute a task of different type tt, then the task will go through a cold start before entering its execution phase. Finally, if the platform was not previously allocated, the task will also experience a cold start delay.

B. Overall project description

The b<>com research institute works on a project that aims to deploy an application of deepfake detection on a private cloud. Users submit a picture to the system and when their request is fulfilled, they obtain a boolean value as a response. The application targets different classes of users – some of them can be media or authorities with high QoS requirements, while others can be casual users tolerating a higher latency.

To differentiate between these classes of users, we propose different levels of per-request SLA. Users with higher requirements will agree to pay a higher per-request price, however if we fail to fulfill their request in the allotted response time, we will consent to a discount – the higher the QoS level, the higher the discount. Hence, there is a strong monetary incentive for the provider to achieve QoS.

Offline phase. In our platform, the lifecycle of the application starts during an offline phase with the developer providing the code of their functions for different hardware architectures ❶. That code is stored in a function repository. Functions are then deployed on a measurement node ❷ where they are run in order to generate metadata relative to the functions: memory requirements, execution time, cold start time and energy consumption for each function are written to a metadata store ❸. The offline phase is is required to run once for a given function on a given platform, it is described in Section III.

Online phase. When a user sends a request to the application ❹, they provide an input picture and specify their desired QoS level. The request is appended to a request queue ❺ at the orchestrator level. When the scheduler pops the request from the queue, the metadata store is queried to retrieve the appropriate function metadata ❻.

The scheduler then proceeds to try to schedule a task (i.e. a function's invocation) to fulfill the request. Tasks are placed on already deployed function *replicas* ❼. Such replicas

TABLE II
EXECUTION PLATFORM CHARACTERIZATION

Platform	Hardware type	Price (MSRP)	Idle energy
Intel Xeon ES-1620 v4	CPU	294	0.067
Nvidia GeForce RTX 2070 Super	GPU	499	0.010
Xilinx Alveo U250	FPGA	7695	0.030

can either be containers or virtual machines, i.e. sandboxed execution environments for the given function. Concurrently, the autoscaler monitors the request queues in all the function replicas ❽. The role of the autoscaler is to rightsize the resources allocation with regard to the fluctuations in load on each function. The designed scheduler and the autoscaler are described in Section IV.

III. OFFLINE PHASE: MEASUREMENT AND METADATA EXTRACTION

A. *Execution platform benchmarks*

As the usage of deep learning inference and energetic impacts grow simultaneously in computing, the power efficiency of the target devices becomes a major concern. The FPGA-based acceleration boards are described as a relevant competitor against the dominant GPU approach. Our study proposes a benchmark, using convolutional neural network (CNN)-based approaches for deepfake detection on CPU, GPU, and FPGA technologies regarding power efficiency during inference time. Our comparison includes power usage, inference speed, and accuracy using traditional CPU and GPU processing against FPGA. Those metrics are crucial for an efficient orchestration on top of heterogeneous platforms.

The used CPU was a Intel Xeon CPU ES-1620 v4 (3.5 GHz) while the GPU was an Nvidia GeForce RTX 2070 Super which can be used with the new versions of AI frameworks. Therefore, both were compatible with TensorFlow, i.e the platform used for inference. with regard to the FPGA, we used the Alveo U250, a cloud computing card from Xilinx, which is compatible with Vitis-AI [21]. The silicon processes used for both devices are similar (12 nm for the GPU and 16 nm for the FPGA), but the GPU may get a slight advantage in this benchmark for its more advanced silicon technology.

In order to carry out the inference on the FPGA, we used Vitis-AI. At the time of this study, the latest available version (v. 2.0) has been used. Vitis-AI proposes two methods for the optimization of the models. The first one is the pruning, which consists in a reduction of the complexity of the model by a compression while removing some non-critical sections of the tree. The second one is the quantization, where we convert the 32 bits floating weights into 8 bits integer. The latter method, which is freely available, is the method we used to optimize our model before the compilation, which converts our model into DPU (Deep Learning Processing Unit) instructions.

B. *Workload characterization*

For the purpose of this study, three popular models have been trained. The first one is based on residual networks (ResNet50), which uses residual blocs and can be efficiently trained [22]. The second one is VGG16 (VGG for Visual Geometry Group), which uses only convolutions as blocks [23] and the third one is VGG19, a variant of VGG16 with three additional layers [24]. Those networks are trained on a GPU, as training is not the subject of this study.

TABLE III
WORKLOAD CHARACTERIZATION

Task	Memory (GB)		Cold start (s)			Execution time (s)			Energy (mWh)		
	CPU	GPU	CPU	GPU	FPGA	CPU	GPU	FPGA	CPU	GPU	FPGA
ResNet50	1.3	3.3	1.232	2.340	9.952	0.124	0.024	0.009	3.11	1.7	0.5
VGG16	1.8	3.3	2.514	4.641	14.528	0.143	0.046	0.010	4.34	3.43	0.55
VGG19	1.9	3.4	2.559	4.641	14.758	0.167	0.048	0.012	5.16	3.58	0.65

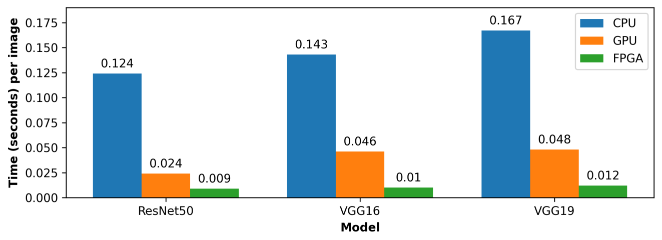


Fig. 2. Inference time for one image with ResNet50, VGG16 and VGG19.

C. *Performance measurements*

As the FPGA acceleration card is intended to be more efficient than a CPU or a GPU [25], making a comparison of the inference time with these three technologies is a first requirement to enable the comparison of the energy cost per image. The performance evaluation in terms of execution time was realized with the same 10,000 images for the three different models. We built a two classes deepfake dataset, the real ones from the CelebA dataset [26], and the fake ones generated using a Generative Adversarial Network (GAN) [27]. The quantization and compilation of the graph was performed with Vitis-AI in order to run it on the FPGA. Only considering the inference time, it turned out that on the three tested models (ResNet50, VGG16 and VGG19), the FPGA is 13.08 to 13.79 times faster than the CPU but also 2.52 to 4.48 times faster than the GPU (see Figure 2).

D. *Energy consumption measurements*

The instant power consumption measured during inference is the overall consumption of the machine (including CPU, memory, mainboard, and power supply) while performing the inference. Measurements have been done using a power distribution unit (PDU) (Raritan PX3-5190R) which is able to monitor instant power and energy consumption of the server (Dell Precision T5810). The results show that inference on CPU yields the lowest instant power consumption. This result is quite expected as the inference on GPU or FPGA also includes power consumption from the CPU.

However, the sole instant power consumption does not reflect the total cost advantage of each platform properly. The execution time needed to process all the images must be considered. The relevant measurement is energy cost per image.

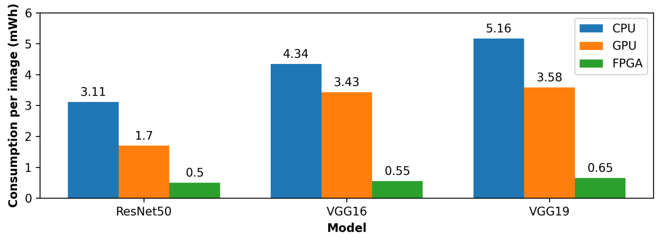


Fig. 3. Energy consumption of inference per image (mWh).

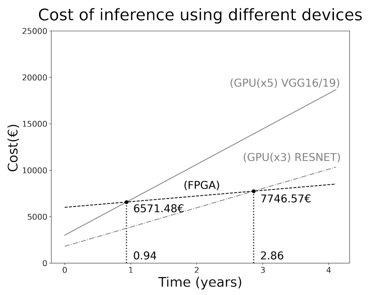


Fig. 4. Total cost of inference on selected devices over time.

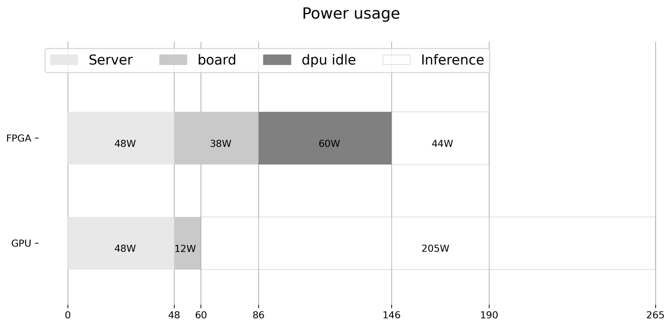


Fig. 5. Power usage breakdown for FPGA and GPU.

The energy consumption has been measured in kilowatt-hour (kWh) for the 10,000 images, then converted into milliwatt-hour (mWh) per image. From that point of view, it is clear that the FPGA is the most energy efficient with regard to the execution time, consuming 6.2 to 6.9 times less than the CPU and 3.3 times to 6.2 times less than the GPU (see Figure 3).

E. Discussion

The results of this benchmark show a clear advantage of the inference on FPGA regarding performance and energy efficiency. The gains in performance are significant, especially with deep learning networks with higher complexity [28]. Computing resources based on servers equipped with FPGA acceleration boards, instead of GPU acceleration boards, would benefit from these advantages.

The raw energy consumption of the inference device does not reflect the total cost of the solution. Indeed, one must also include the cost of the equipment itself. This is a major point in the comparison between GPU and FPGA, because there is a price gap between the two technologies: the GPU (RTX 2070 Super) being used for this benchmark was introduced around 600€, while the FPGA (Alveo U250) is sold around 6000€. The cost of the electric energy to perform the inference is very low (we used the European average of 0.1833€ per kWh as proposed in [29]), compared to the initial cost of the device: the runtime needed to benefit from the cost advantage of the FPGA is in the order of several months of continuous operation. Figure 4 depicts cumulative cost (in euros) of the usage of a server with either GPU or FPGA acceleration versus time (in years). Our cost estimate includes the number of GPU needed with their cost to equalize the FPGA performances and uses a 2x factor [30], to account for the total power consumption of the infrastructure (mainly cooling and networking). The FPGA can become a cost effective solution after a few months for complex CNNs. For networks with lower complexity, the cost advantage of the FPGA is reached after more than two years.

The previous analysis is valid in the situation where the inference is always performed at full load. Indeed when breaking down the power consumption of the GPU between idle power and inference power, it is clear that the GPU is able to dynamically scale its power usage with the intensity of the processing. The FPGA on the other side seems to have very limited power management. Once the DPU design is loaded into the device, its power usage at idle remains very high (see Figure. 5). Adding to the 38W of the FPGA board power, there is indeed a residual 60W power consumption when the DPU is idle. Even though further evolution of the DPU implementation on the FPGA may fix this issue (like reducing clock tree activity when idle), this has an impact on the total cost and must be considered if the device is not always used at full load. With only 12W of idle power, the GPU is a better candidate when full-load device usage cannot be guaranteed.

As the trend towards CNNs with more complexity continues [31], using the most efficient devices will become a major challenge. The FPGA solution offers a new option to perform inference. However, FPGAs are not a drop-in replacement for GPUs yet: the compilation flow remains complex and time-consuming. A trade-off between the flexibility of the GPUs and the efficiency of the FPGAs will have to be made. The next section discusses a first orchestrator that considers the above-mentionned characterization for allocating and scheduling heterogeneous resources.

IV. Online Phase: Autoscaling and Scheduling

In this section, we formulate the problem that our contribution addresses, and give a detailed description of our model. Finally, we present a formal description of our strategy for the autoscaling of resources and scheduling of tasks.

A. Serverless resource orchestration challenges

Scheduling workloads in the serverless paradigm is a two-fold problem: providers have to dynamically handle resource allocation (i.e. managing resources pools when scaling the number of replicas for an application) and job placement (i.e. mapping tasks to existing replicas).

Increasing the replica count introduces a performance challenge: when a new replica is spun up, be it as a container or a virtual machine, the execution sandbox has to go through its initialization phase. This is called a "cold start"

Commercial solutions such as AWS Lambda often avoid the cold start problem by maintaining pools of pre-warmed sandboxes [32]. These virtual machines (VMs) or containers are started in anticipation and paused in a post-initialization state. When activity resumes, incoming requests can be served without suffering a cold start delay, at the expanse of resources multiplexing on the provider side. While this solution allows reducing, or even eliminating cold start delays, it takes a toll on the provider's resources multiplexing capacity [33] and raises the total cost of ownership (TCO).

Furthermore, Machine Learning as a Service (MLaaS) applications exhibit highly fluctuating load [1], thus reinforcing the argument that a reactive resource allocation strategy is necessary to rightsize the infrastructure. However, as the execution time of inference tasks is in the ballpark of hundredths to tenths of a second, while the initialization time of sandboxes can range between hundredths of a second to seconds [34], we need a mechanism to avoid incurring huge latency costs to the execution of functions.

Mission critical tasks require service level guarantees from the provider. SLAs in cloud computing typically consist in agreeing on a resource availability rate over time; if the provider fails to meet this agreement, a discount is offered to the customer. While this may work for reserved resources, we can see that it does not make sense in the serverless paradigm. The ability to guarantee task response time would allow a serverless provider to achieve per-invocation SLAs [3].

A possibility to improve performance-cost ratios is to make use of hardware accelerators. Despite being a costly investment (see Figure 4), these devices can achieve important speedups for parallel tasks (see Figure 2), thus improving function response time, with a decreased cost in energy (see Figure 3).

B. Task model

Applications are composed of functions. A function execution is called a *task*. In this work, there are no dependencies between these tasks: the application is made up of pure, stateless functions. The events that trigger the execution of a task arrive in the system at a random, bounded interval. We formulate the hypothesis that a request always succeeds and leads to the execution of a *task* (an instance of a *function*). When a task has started its execution on its allocated platform, it runs for the totality of its execution time. We do not consider preemption or failures in this contribution: a task always finishes its execution successfully, albeit its response time can exceed its QoS requirements. We do not consider possible interference between workloads on the same node [35].

We consider tasks that can indiscriminately be executed on heterogeneous execution platforms. In the context of our specific case study, implementation of the different functions

TABLE IV
NOTATION DICTIONARY

Notation	Description
$f_{N,P}$	A function f scheduled to run on a platform P available on node N
QP	QoS penalty
QD	QoS deviation
WET	Worst execution time
TT	Task total time
WT	Wait time
CST	Cold start time
ET	Execution time
EC	Energy consumption
HP	Hardware price
TC	Task consolidation
Q	Task queue on a replica
$replicaCount_f$	Size of the replica pool for a function f
$replicaCount_{f,h}$	Size of the replica pool for a function f on hardware type h
$concurrency_f$	Average number of in-flight requests for a function f
$concurrency_{f,h}$	Average number of in-flight requests for a function f on replicas of hardware type h
$x_{f,h}$	Concurrency threshold for a function f on a replica of hardware type h
$scaleCost_{f_{N,P}}$	Cost of creating a new replica for function f on a platform P available on node N
$schedCost_{f_{N,P}}$	Cost of scheduling an execution of function f on a platform P available on node N

has been done by hand for each platform; however, work exists to allow automatic cross-compilation to heterogeneous architectures [36], [37]. The following metadata have been measured for each function, on each execution platform:

- *Memory requirements* – the quantity of memory (in GB) allocated for the task;
- *Cold start duration* – the duration of sandbox initialization when running the task on a platform that does not have the function in cache;
- *Execution time* – the expected duration of effective execution of the task, excluding its initialization phase;
- *Energy consumption* – the difference between idle and active energy incurred by the execution platform when it runs the task.

Equation 1 breaks down the expected response time for the execution of a function f on a platform P on node N.

$$TT_{f_{N,P}} = WT_{f_{N,P}} + CST_{f_{N,P}} + ET_{f_{N,P}} \qquad (1)$$

Where:

- $WT_{f_{N,P}}$ is the duration of the scheduling decision, including the time spent by the user request in the queue;
- $CST_{f_{N,P}}$ is the duration of initialization for the function invocation, including its potential cold start time;
- $ET_{f_{N,P}}$ is the execution time of the function on the platform.

We propose different levels of QoS depending on users' needs in terms of guarantees on execution time. Each level of QoS presents a different *duration deviation* (noted QD in Equation. 3) – a factor by which the worst execution time for a function is multiplied to give an upper bound on the execution time of this function for this QoS level.

Predicted function execution time is always based on the worst execution time (noted WET_f), e.g. a task's execution time when scheduled on the execution platform showing the least level of performance for said function:

$$\forall (N, P),\, WET_f = \max ET_{N,P} \tag{2}$$

After a task is scheduled on an execution platform, it will go through its total time of execution described in equation 1. The task deadline is computed by multiplying the function's worst response time (as expressed in equation 2) by the QoS duration deviation associated to the user's request's QoS level. Equation 3 shows that we set a boolean value $QP_{f_{N,P}}$ for each function invocation if the tasks misses its deadline.

$$QP_{f_{N,P}} = TT_{f_{N,P}} \cdot QD_{f_{N,P}} > WET_f \tag{3}$$

C. Autoscaling strategy

In a serverless platform, the autoscaler has the responsibility to allocate hardware resources for function executions. For any function, an autoscaler can allocate n *replicas*. The number of replicas for a given function at any moment determines the concurrency level.

In Knative, the number of replicas for a given function (equation 4) depends on the moving average load for a function, i.e. the average number of in-flight requests for the function on a 60 second window (in-system concurrency per function). It is bounded by a concurrency threshold per replica, i.e. the maximum number of requests queued in a function's replica at any moment [38].

$$replicaCount_f = \frac{concurrency_f}{x_f} \tag{4}$$

This sizing mechanism allows allocating CPUs under Knative, in reaction to changes in the current state of concurrency in the system. The main contribution of the autoscaler we propose is to upgrade the Knative in order to take into account the heterogeneity of the execution platforms.

Simple Knative mechanism does not hold when the infrastructure consists of a variety of execution platforms. Indeed, such platforms exhibit various levels of performance, energy consumption and cost. This has a consequence on the number of replicas the provider has to deploy on these platforms: for a given level of application load, heterogeneous replicas will be able to handle different numbers of tasks in the same makespan. For our platform to handle heterogeneity in the underlying infrastructure, we propose a per-function and **per-hardware type** replica count as in equation 5.

$$replicaCount_{f_h} = \frac{concurrency_{f_h}}{x_{f_h}} \tag{5}$$

An autoscaling decision can introduce opportunity costs in the system: hardware accelerators are scarcely available as compared to CPUs, and allocating them for a given function at a given time will make them unavailable for further computations. In order for the autoscaler to decide when it is relevant to allocate such accelerators, it has to be **cost-aware**.

In order to determine the concurrency threshold per replica $x_{f,h}$ for a function f on hardware type h (e.g. GPU and FPGA), we fixed the concurrency threshold per replica on CPUs to $x_{f,c} = 100$ as it is the default value in Knative [39]. Then, we used the measurements from the offline phase (table III) to establish a composite ratio (including performance, energy, platform price) as described in equation 6. In our policy, we chose to favor response time by setting $k_{ET} = \frac{2}{3}$, $k_{EC} = \frac{1.5}{6}$ and $k_{HP} = \frac{0.5}{6}$. For example, for the function ResNet50 (described in table IV-B), task queues in replicas are sized to 100 for CPUs, 489 for GPUs and 1292 for FPGAs.

$$x_{f,h} = x_{f,c} \cdot (k_{ET} \cdot \frac{ET_{f_c}}{ET_{f_h}} + k_{EC} \cdot \frac{EC_{f_c}}{EC_{f_h}} + k_{HP} \cdot \frac{HP_{f_c}}{HP_{f_h}}) \tag{6}$$

When the concurrency threshold for a function is exceeded in the queues of replicas on a given hardware type, the autoscaler will proceed to *scale up* the function: a new replica will be spun up to handle further user requests.

Allocation starts with the complete list of nodes available in the infrastructure. First, we build a subset of the available nodes, called *suitable nodes*. Given the memory requirements we measured for each function, we eliminate nodes that currently do not have enough memory available to run a replica for the function. Each replica deployed on a node's execution platform consumes the total quantity of memory required by the function type. If the node is out of memory, its execution platforms cannot be used to deploy any more replica.

In order to select the type of resource to allocate for this replica, the autoscaler minimizes the cost function given in equation 7. In our policy, as for the autoscaling, we chose to favor total task execution time by setting $k_{TT} = \frac{2}{3}$, $k_{EC} = \frac{1.5}{6}$ and $k_{HP} = \frac{0.5}{6}$. Depending on which hardware is available in the pool at scale up time, the autoscaler will favor creating a new function replica on the platform that will execute the task in the lowest total time, including cold start, with the lowest energy consumption and the lowest price.

$$\begin{aligned} scaleCost_{f_{N,P}} = &k_{TT} \cdot TT_{f_{N,P}} \\ &+ k_{EC} \cdot EC_{f_{N,P}} \\ &+ k_{HP} \cdot HP_{f_{N,P}} \end{aligned} \tag{7}$$

On the contrary, when concurrency for a function falls beneath the threshold on a given hardware type, the autoscaler will employ a best effort policy and tries to deallocate any replica with an empty task queue on said hardware type. If a replica does have an empty task queue, it will be released into the available platforms pool, and the memory it was allocated on the node will be freed.

The different weights (k) used in equations 6 and 7 can be modified by the provider to customize the allocation policy according to different priorities.

D. Scheduling strategy

Workload characterization is instrumental to performance prediction, as it can guide scheduling decisions that lead to the fulfillment of QoS requirements [40]. Our scheduling strategy relies on tasks metadata as described in section IV-B. Building knowledge about serverless tasks is achieved during an offline phase on our platform, as code is pushed to the provider's registries ahead of actual execution [41].

In Knative, the scheduler handles incoming tasks in a FIFO fashion. To manage the different levels of QoS requirements, we propose that our scheduler pops tasks from the gateway queue by **earliest deadline first**. We compute the task deadline by using its worst execution time on the platform using equation 2, and multiplying it by the allowed duration deviation set by the QoS level. After the task execution, we will check if we missed its deadline and set the associated penalty accordingly, as described in equation 3.

We iterate on the function's replicas to fetch and predict the following metrics based on task metadata:

- potential penalty: we compute the length of the platform's queue and check whether the task's deadline will be missed, as described in equation 3;
- energy consumption: we retrieve the offline measurements to establish the dynamic energy consumption for this task on the platform;
- task consolidation: we compute the length of the platform's task queue Q by summing the total times of all queued tasks, as described in equations 8 (queue length) and 1 (task total time).

$$len\, Q_{N,P} = \sum TT_{f_{N,P}} \tag{8}$$

These values are normalized to fit in a weighted cost function described in equation 9. We used $k_{QP} = \frac{2}{3}$, $k_{EC} = \frac{0.5}{6}$ and $k_{TC} = \frac{1.5}{6}$ (same as for the autoscaler). The scheduler then minimizes that cost function for all replicas (N, P) (i.e. node and execution platform).

$$\begin{aligned} schedCost_{f_{N,P}} = & k_{QP} \cdot QP_{f_{N,P}} \\ & + k_{EC} \cdot EC_{f_{N,P}} \\ & + k_{TC} \cdot TC_{f_{N,P}} \end{aligned} \tag{9}$$

If the scheduler cannot find an available replica to execute the task, it will be pushed back to the gateway's task queue. This will increase in-system concurrency for the function, nudging the autoscaler into allocating another replica on relevant hardware.

V. Evaluation

A. Experimental setup

We used measurements from the evaluation of three different machine learning models (see table III). These models have been implemented on three different execution platforms (see table II) explained in section III.

These data served as input for a simulator we built using SimPy [42]. The simulator follows the system model described in sections II-A1, II-A2, IV-B.

We measured cold start delays for our case study applications, see table III. It appears that execution times are dominated by cold start delays, making adequate resource allocation a stringent requirement to comply with SLAs.

In the performance evaluation part, we compare two autoscalers:

- HeROfake (HRO) – Our heterogeneity-aware, metadata-based resource allocator;
- Knative (KN) – We modeled the Knative autoscaler behavior to the best of our knowledge.

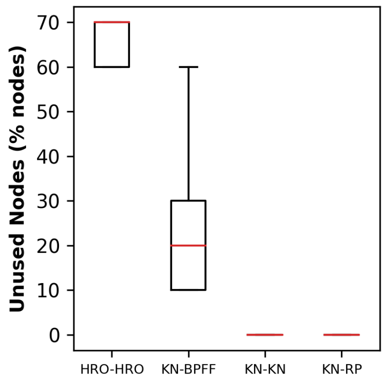
Our evaluation extends to four schedulers:

- HeROfake (HRO) – Our cost-aware scheduler that minimizes SLA violations, energy consumption and resource usage;
- Knative (KN) – Knative selects a platform on the most available node [4]. Execution platforms are sorted by in-flight requests count. The platform with the shortest queue is selected;
- Random Placement (RP) – Tasks are assigned a random execution platform;
- Bin Packing First-Fit (BPFF) – Tasks are consolidated on the minimum number of execution platforms. While an execution platform has enough memory available for incoming tasks, it is systematically chosen until it runs out of memory; then, a new platform is selected. BPFF is likely to be the scheduling policy for AWS Lambda [43]

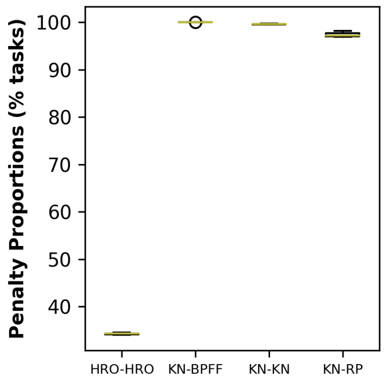
We designed a two-step performance evaluation based on simulations:

- **Comparison against baselines** (figure 6): in this part, we compared our HeROfake combination of autoscaler and scheduler (HRO-HRO) to: (1) full-featured Knative autoscaler and scheduler (KN-KN), (2) Knative autoscaler with BPFF scheduler (KN-BPFF), (3) Knative autoscaler with RP scheduler (KN-RP);
- **Impact of HeROfake components on the overall performance** (figure 7): here we discuss the individual impact of each of the autoscaler and the scheduler. To do so, we devised different strategies: (1) using HeROfake autoscaler with Knative scheduler, and (2) using Knative autoscaler with HeROfake scheduler, and we compared those strategies with full featured HeROfake and Knative.

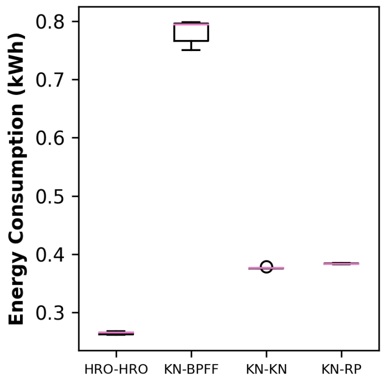
The naming of each scenario in these figures consists of two parts divided by a dash symbol. The first part corresponds to the allocation policy; the second part corresponds to the scheduling policy (for example, HRO-KN means we used



(a) Task consolidation (based on the unused node count)

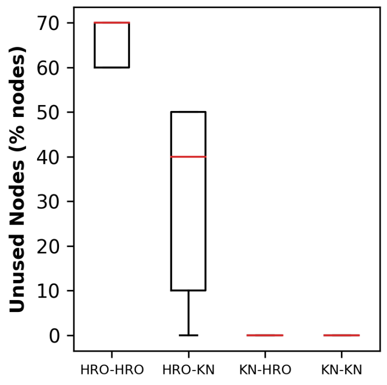


(b) QoS violations (based on tasks with missed deadline)

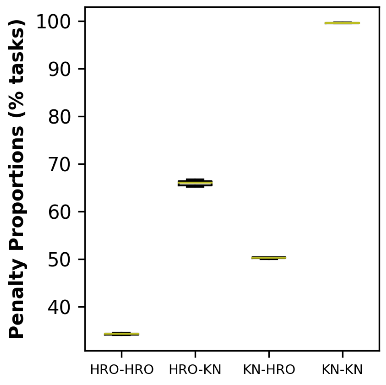


(c) Dynamic energy consumption (in kWh)

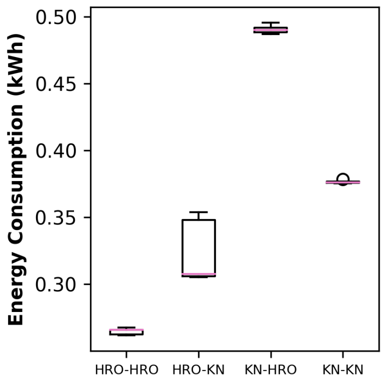
Fig. 6. Evaluation 1 – Comparison against baselines



(a) Task consolidation (based on the unused node count)



(b) QoS violations (based on tasks with missed deadline)



(c) Dynamic energy consumption (in kWh)

Fig. 7. Evaluation 2 – Impact of HeROfake components on the overall performance

the HeROfake autoscaler in conjunction with the Knative scheduler).

For each of the combinations of autoscaler and scheduler policies, we ran the experiment on a synthetic workload scenario consisting of 50000 tasks (user requests). Tasks are assigned a random type (ResNet50, VGG16 or VGG19) and a random QoS level (high, medium, low) following a uniform distribution, with QoS duration deviations respectively set to 2, 3 and 4.

Weights for the concurrency level (equation 6) have been set to $k_{ET} = \frac{2}{3}$, $k_{EC} = \frac{1.5}{6}$ and $k_{HP} = \frac{0.5}{6}$. Weights for the scale up decision (equation 7) have been set to $k_{TT} = \frac{2}{3}$, $k_{EC} = \frac{1.5}{6}$ and $k_{HP} = \frac{0.5}{6}$. Weights for the scheduling decision (equation 9) have been set to $k_{QP} = \frac{2}{3}$, $k_{EC} = \frac{0.5}{6}$ and $k_{TC} = \frac{1.5}{6}$.

B. Experimental results

1) Comparison against baselines: **Tasks consolidation**. Figure 6a shows that our combination of autoscaler and scheduler achieves the highest number of unused nodes. Under Knative's autoscaler, the BPFF scheduler ensures the best consolidation, but that policy still needs more than three times the nodes we need with our policy. We can see that our scheduler comes second best at task consolidation, with almost 70% of nodes left unused – a negligible degradation compared to HRO-BPFF.

Service Level Agreements. Figure 6b shows that HRO-HRO performs the best in terms of QoS violations, with 35% of tasks missing their deadlines. This is a huge improvement with regard to the Knative results, where tasks miss their deadlines more than 99% of the time: the delay introduced by the reactive allocation of resources cannot be compensated in time using only CPUs.

Energy consumption. Figure 6c shows that our policy, with the HRO autoscaler and scheduler working in conjunction, consistently performs the best in terms of dynamic energy consumption. This is obviously because we allocate hardware accelerators; however, during our evaluation, the makespan for our scenario is similar under Knative and HRO policies (around 13.5 minutes). The BPFF scheduling policy also performs the worst in terms of execution time, as it maximizes the task queues in execution platforms, thus yielding the worst results in terms of energy consumption.

2) Impact of each component: **Tasks consolidation**. Figure 7a shows that HRO-HRO performs the best at task consolidation, leaving just under 70% of the available nodes unused, while Knative's scheduler under our autoscaling policy only achieves 40% of unused nodes. This result is expected, as Knative's scheduler employs a Least Connected policy. We see mediocre consolidation results for KN-HRO, but for a different reason: this is because our scheduler tries to minimize QoS violations and spreads the task across all the allocated CPUs.

Service Level Agreements. Figure 7b shows that our scheduler performs the worst when executed in conjunction with Knative's autoscaler. This is because our scheduler tries to minimize penalties: when given only CPUs to work with, it will behave similarly to Knative's scheduler and spread tasks across theses CPUs to limit QoS violations. However, our scheduler under the Knative autoscaler still manages to keep QoS violations at around 50% of tasks, showing that there is room for improvement even when deploying inference tasks on CPUs only. Note that during our evaluation, the Knative autoscaler gave the worst results regarding cold starts frequency (6.5 more under KN-HRO than under HRO-KN).

Energy consumption. Figure 7c shows that energy consumption is always lower when using our autoscaler, which can allocate hardware accelerators. However, our scheduler used with Knative's autoscaler yields the worst results in terms of energy consumption. This is again the result of the scheduler trying to minimize penalties and spreading task across a maximum number of CPUs.

VI. State of the Art

Previous work focused on autoscaling platforms for the deployment of short-lived tasks, comprised in applications exhibiting unpredictable load patterns. Table I summarises how these contributions differ from our target platform.

Some of these contributions attempted to achieve SLA with unreserved resources [1], [3], [5], [6], [44]–[46]. Among these contributions, some focus on the use of additional heterogeneous hardware resources to accelerate workload execution [2], [3], [7]. They often require overcommitting resources to make use of hardware acceleration, e.g. by relying on reserved AWS instances that provide access to GPUs [3], using a pool of pre-warmed containers [2], or even proactively provisioning nodes to meet user-defined function deadlines [6]. These interesting solution however may fall short in terms of resource usage and would incur additional energy consumption in a private cloud.

Furthermore, some authors focus on homogeneous infrastructures [1], [4]–[7]. Those studies could hardly fit the private cloud setting we target, where resources are usually transient and heterogeneous. Also, some of these contributions propose task models that do not cover user-defined, per-request SLA [2], [4]. Finally, some of these contributions are performance-oriented rather than cost-oriented which is crucial in our cloud context [1], [2], [6], [8].

In spite of power being one of the topmost part of the total cost of ownership (TCO) in a datacenter – sometimes exceeding the cost of buying hardware [20] – to the best of our knowledge, none of these contributions cover the impact of dynamic allocation and dynamic placement on energy consumption, nor do they consider energy consumption as a QoS metric. This is a serious limitation, as optimizing for task consolidation opens possibilities for throttling and powering-off policies that can have a major impact on a datacenter's energy efficiency [47].

VII. Conclusion and Future Work

In this paper, we introduced HeROfake, our framework for the deployment of short-lived, interactive deepfake detection tasks on a private, heterogeneous serverless cloud.

We presented the two phases that make up this framework: an offline phase during which we characterize execution platform performances and task requirements; and an online phase during which we dynamically allocate resources and schedule tasks to run on those platforms.

Experimental results show that while total task execution time in HeROfake is similar to vanilla Knative, we achieve more than 60% reduction in QoS penalties; tasks are consolidated on less than 40% of the infrastructure's nodes, with 77% execution platforms left unused; and dynamic energy consumption is reduced by 35% as compared to Knative.

The inclusion of video handling in the framework is an interesting challenge, as it would introduce dependencies between tasks. Function executions would not be *stateless* anymore, resulting in the necessity to tackle the problem of intermediate data storage in a serverless infrastructure.

We also intend to extend the simulator with a parser so as to be able to use real datacenter traces as input scenarios, instead of using synthetic workloads only.

References

[1] 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 *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. Las Vegas Nevada: ACM, Dec. 2017, pp. 109–120. [Online]. Available: https://doi.org/10.1145/3135974.3135993

[2] W. Ling, L. Ma, C. Tian, and Z. Hu, "Pigeon: A dynamic and efficient serverless and FaaS framework for private cloud," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. Las Vegas, NV, USA: IEEE, Dec. 2019, pp. 1416–1421. [Online]. Available: https://doi.org/10.1109/CSCI49370.2019.00265

[3] C. Zhang, M. Yu, W. Wang, and F. Yan, "MArk: Exploiting cloud services for cost-effective, SLO-aware machine learning inference serving," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, Jul. 2019, pp. 1049–1062. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/zhang-chengliang

[4] A. Suresh, G. Somashekar, A. Varadarajan, V. R. Kakarla, H. Upadhyay, and A. Gandhi, "ENSURE: Efficient scheduling and autonomous resource management in serverless environments," in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. Washington, DC, USA: IEEE, Aug. 2020, pp. 1–10. [Online]. Available: https://doi.org/10.1109/ACSOS49614.2020.00020

[5] A. Mampage, S. Karunasekera, and R. Buyya, "Deadline-aware dynamic resource management in serverless computing environments," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. Melbourne, Australia: IEEE, May 2021, pp. 483–492. [Online]. Available: https://doi.org/10.1109/CCGrid51090.2021.00058

[6] 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*. Seattle WA USA: ACM, Nov. 2021, pp. 138–152. [Online]. Available: https://doi.org/10.1145/3472883.3486981

[7] 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*. Lausanne Switzerland: ACM, Feb. 2022, pp. 768–781. [Online]. Available: https://doi.org/10.1145/3503222.3507709

[8] J. Cho, D. Z. Tootaghaj, L. Cao, and P. Sharma, "SLA-driven ml inference framework for clouds with heterogeneous accelerators," in *Proceedings of Machine Learning and Systems*, vol. 4, 2022, pp. 20–32. [Online]. Available: https://proceedings.mlsys.org/paper/2022/file/0777d5c17d4066b82ab86dff8a46af6f-Paper.pdf

[9] 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

[10] E. Jonas, J. Schleier-Smith, V. Sreekanti, C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. J. Yadwadkar, J. E. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud programming simplified: A Berkeley view on serverless computing," *CoRR*, vol. abs/1902.03383, 2019. [Online]. Available: http://arxiv.org/abs/1902.03383

[11] K. Owens, "CNCF WG-Serverless whitepaper v1.0," Cloud Native Computing Foundation, Tech. Rep., 2018.

[12] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*, ser. SoCC '12. New York, NY, USA: Association for Computing Machinery, 2012. [Online]. Available: https://doi.org/10.1145/2391229.2391236

[13] A. Khandelwal, A. Kejariwal, and K. Ramasamy, "Le Taureau: Deconstructing the serverless landscape & a look forward," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Portland OR USA: ACM, Jun. 2020, pp. 2641–2650. [Online]. Available: https://doi.org/10.1145/3318464.3383130

[14] T.-A. Yeh, H.-H. Chen, and J. Chou, "KubeShare: A framework to manage GPUs as first-class and shared resources in container cloud," in *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 173–184. [Online]. Available: https://doi.org/10.1145/3369583.3392679

[15] A. Jahanshahi, H. Z. Sabzi, C. Lau, and D. Wong, "GPU-NEST: Characterizing energy efficiency of multi-GPU inference servers," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 139–142, 2020. [Online]. Available: https://doi.org/10.1109/LCA.2020.3023723

[16] E. van Eyk, A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann, "A SPEC RG cloud group's vision on the performance challenges of FaaS cloud architectures," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. Berlin Germany: ACM, Apr. 2018, pp. 21–24. [Online]. Available: https://doi.org/10.1145/3185768.3186308

[17] J.-E. Dartois, H. B. Ribeiro, J. Boukhobza, and O. Barais, "Cuckoo: Opportunistic MapReduce on ephemeral and heterogeneous cloud resources," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. Milan, Italy: IEEE, Jul. 2019, pp. 396–403. [Online]. Available: https://doi.org/10.1109/CLOUD.2019.00070

[18] I. Müller, R. Marroquín, and G. Alonso, "Lambada: Interactive data analytics on cold data using serverless cloud infrastructure," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Portland OR USA: ACM, Jun. 2020, pp. 115–130. [Online]. Available: https://doi.org/10.1145/3318464.3389758

[19] R. Buyya, S. K. Garg, and R. N. Calheiros, "SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions," in *2011 International Conference on Cloud and Service Computing*. Hong Kong, China: IEEE, Dec. 2011, pp. 1–10. [Online]. Available: https://doi.org/10.1109/CSC.2011.6138522

[20] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732–794, 2016. [Online]. Available: https://doi.org/10.1109/COMST.2015.2481183

[21] Xilinx. (2022) Vitis-AI. [Online]. Available: https://github.com/Xilinx/Vitis-AI

[22] T. Liu, M. Chen, M. Zhou, S. S. Du, E. Zhou, and T. Zhao, "Towards understanding the importance of shortcut connections in residual networks," 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/7716d0fc31636914783865d34f6cdfd5-Paper.pdf

[23] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[24] S. Sukegawa, K. Yoshii, T. Hara, K. Yamashita, K. Nakano, N. Yamamoto, H. Nagatsuka, and Y. Furuki, "Deep neural networks for dental implant system classification," *Biomolecules*, vol. 10, no. 7, 2020. [Online]. Available: https://www.mdpi.com/2218-273X/10/7/984

[25] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," in *2009 International Conference on Field Programmable Logic and Applications*, 2009, pp. 126–131. [Online]. Available: https://doi.org/10.1109/FPL.2009.5272532

[26] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," 2014. [Online]. Available: https://arxiv.org/abs/1411.7766

[27] O. Giudice, L. Guarnera, and S. Battiato, "Fighting deepfakes by detecting GAN DCT anomalies," *Journal of Imaging*, vol. 7, no. 8, 2021. [Online]. Available: https://www.mdpi.com/2313-433X/7/8/128

[28] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels," in *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*, 2019, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ICESS.2019.8782524

[29] Eurostat. (2022) Electricity prices for non-household consumers. [Online]. Available: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Electricity_price_statistics#Electricity_prices_for_non-household_consumers

[30] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner, "United States data center energy usage report," Lawrence Berkeley National Lab, Tech. Rep. LBNL–1005775, 1372902, Jun. 2016.

[31] A. Elhassouny and F. Smarandache, "Trends in deep convolutional neural networks architectures: A review," in *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)*, 2019, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ICCSRE.2019.8807741

[32] P. Vahidinia, B. Farahani, and F. S. Aliee, "Cold start in serverless computing: Current trends and mitigation strategies," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*. Barcelona, Spain: IEEE, Aug. 2020, pp. 1–7. [Online]. Available: https://doi.org/10.1109/COINS49042.2020.9191377

[33] J. M. Hellerstein, J. M. Faleiro, J. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," in *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org, 2019. [Online]. Available: http://cidrdb.org/cidr2019/papers/p119-hellerstein-cidr19.pdf

[34] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici, "My VM is lighter (and safer) than your container," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 218–233. [Online]. Available: https://doi.org/10.1145/3132747.3132763

[35] J.-E. Dartois, J. Boukhobza, A. Knefati, and O. Barais, "Investigating machine learning algorithms for modeling SSD I/O performance for container-based virtualization," *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1103–1116, Jul. 2021.

[36] E. Horta, H.-R. Chuang, N. R. VSathish, C. Philippidis, A. Barbalace, P. Olivier, and B. Ravindran, "Xar-Trek: Run-time execution migration among FPGAs and heterogeneous-ISA CPUs," in *Proceedings of the 22nd International Middleware Conference*. Québec city Canada: ACM, Dec. 2021, pp. 104–118. [Online]. Available: https://doi.org/10.1145/3464298.3493388

[37] Y. Zha and J. Li, "When application-specific ISA meets FPGAs: A multi-layer virtualization framework for heterogeneous cloud FPGAs," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 123–134. [Online]. Available: https://doi.org/10.1145/3445814.3446699

[38] T. Knative Authors. (2022) Knative – Autoscaling. [Online]. Available: https://github.com/knative/serving/tree/main/docs/scaling

[39] ——. (2022) Knative – Configuraing concurrency. [Online]. Available: https://knative.dev/docs/serving/autoscaling/concurrency/#soft-versus-hard-concurrency-limits

[40] A. Mampage, S. Karunasekera, and R. Buyya, "A holistic view on resource management in serverless computing environments: Taxonomy and future directions," *ACM Computing Surveys*, p. 3510412, Jan. 2022. [Online]. Available: https://doi.org/10.1145/3510412

[41] M. Shahrad, R. Fonseca, Í. 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 *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC'20. USA: USENIX Association, 2020, p. 14. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/shahrad

[42] T. SimPy. (2022) SimPy. [Online]. Available: https://simpy.readthedocs.io/

[43] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA: USENIX Association, Jul. 2018, pp. 133–146. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/wang-liang

[44] M. Handaoui, J.-E. Dartois, J. Boukhobza, O. Barais, and L. d'Orazio, "ReLeaSER: A reinforcement learning strategy for optimizing utilization of ephemeral cloud resources," in *2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2020, pp. 1–8.

[45] M. Handaoui, J.-E. Dartois, L. Lemarchand, and J. Boukhobza, "Salamander: A holistic scheduling of MapReduce jobs on ephemeral cloud resources," in *The 20th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2020, pp. 320–329.

[46] S. Yalles, M. Handaoui, J.-E. Dartois, O. Barais, L. d'Orazio, and J. Boukhobza, "RISCLESS: A reinforcement learning strategy to guarantee SLA on cloud ephemeral and stable resources," in *2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2022, pp. 83–87.

[47] N. Chaurasia, M. Kumar, R. Chaudhry, and O. P. Verma, "Comprehensive survey on energy-aware server consolidation techniques in cloud computing," *The Journal of Supercomputing*, vol. 77, no. 10, pp. 11 682–11 737, Oct. 2021. [Online]. Available: https://doi.org/10.1007/s11227-021-03760-1