information

MDPI

Article

AI-Driven QoS-Aware Scheduling for Serverless Video Analytics at the Edge

Dimitrios Giagkos, Achilleas Tzenetopoulos *, Dimosthenis Masouros, Sotirios Xydis, Francky Catthoor and Dimitrios Soudris

School of Electrical and Computer Engineering, National Technical University of Athens, 15780 Athens, Greece
* Correspondence: atzenetopoulos@microlab.ntua.gr

Abstract: Today, video analytics are becoming extremely popular due to the increasing need for extracting valuable information from videos available in public sharing services through camera-driven streams in IoT environments. To avoid data communication overheads, a common practice is to have computation close to the data source rather than Cloud offloading. Typically, video analytics are organized as separate tasks, each with different resource requirements (e.g., computational- vs. memory-intensive tasks). The serverless computing paradigm forms a promising approach for mapping such types of applications, enabling fine-grained deployment and management in a per-function, and per-device manner. However, there is a tradeoff between QoS adherence and resource efficiency. Performance variability due to function co-location and prevalent resource heterogeneity make maintaining QoS challenging. At the same time, resource efficiency is essential to avoid waste, such as unnecessary power consumption and CPU reservation. In this paper, we present Darly, a QoS-, interference- and heterogeneity-aware Deep Reinforcement Learning-based Scheduler for serverless video analytics deployments on top of distributed Edge nodes. The proposed framework incorporates a DRL agent that exploits performance counters to identify the levels of interference and the degree of heterogeneity in the underlying Edge infrastructure. It combines this information along with user-defined QoS requirements to improve resource allocations by deciding the placement, migration, or horizontal scaling of serverless functions. We evaluate Darly on a typical Edge cluster with a real-world workflow composed of commonly used serverless video analytics functions and show that our approach achieves efficient scheduling of the deployed functions by satisfying multiple QoS requirements for up to 91.6% (Profile-based) of the total requests under dynamic conditions.

Keywords: edge computing; serverless computing; deep reinforcement learning; IoT video processing; resource management

check for updates

Citation: Giagkos, D.; Tzenetopoulos, A.; Masouros, D.; Xydis, S.; Catthoor, F.; Soudris, D. AI-Driven QoS-Aware Scheduling for Serverless Video Analytics at the Edge. *Information* **2024**, *15*, 480. https://doi.org/10.3390/info15080480

Academic Editors: Vasco N. G. J. Soares, João M. L. P. Caldeira, Bruno Bogaz Zarpelão and Jaime Galán-Jiménez

Received: 12 July 2024
Revised: 7 August 2024
Accepted: 7 August 2024
Published: 13 August 2024

1. Introduction

The continuous advancements in data transmission speeds and network bandwidth are pivotal in enabling the Internet of Things (IoT). Increasingly, societies are leveraging IoT to address everyday needs effectively, enabling intelligence at the network's edge [1]. The IoT paradigm facilitates computation closer to the data source, resulting in cost reduction, real-time response times, and improved energy efficiency, while ensuring greater data privacy. Video analytics stands out as a significant IoT application, wherein various domains utilize camera data to derive valuable insights. These insights can be used for event-driven responses [2] or for aggregating knowledge to support long-term strategic planning [3]. From the infrastructure operator's point of view, the optimization goal for such applications varies depending on the domain and the criticality of their requirements. Applications like video analytics, mobile augmented reality (MAR) [4], autonomous vehicles, and smart manufacturing [5], require rapid incident response and thus prioritize low-latency and high-throughput. Conversely, less critical, latency-tolerant applications may prioritize lower costs over immediate responsiveness.

On the other hand, serverless computing [6] is an emerging paradigm based on the Function-as-a-Service (FaaS) model. In a serverless architecture, decomposed services, i.e., functions, are deployed only when requested, such as in response to events. With serverless being initially designed for Cloud, several works [7,8] leverage its primitives to implement video analytics frameworks that exploit the massive scale of distributed functions for embarrassingly parallel computation. However, serverless at the Edge of the network has recently gained traction in both industry [9] and academia [10–13]. Additionally, UAV-assisted MEC Systems [14] leverage serverless computing due to its alignment with the intermittent activity and fine-grained parallelism of edge tasks, eliminating the need for long-term resource reservations, while automating task scheduling and stragglers mitigation.

In particular, video analytics services, which are a major use case for serverless computing, are characterized by various types of constraints such as frame rate (throughput), latency, reliability (uptime), and max power usage (for heat mitigation). In a serverless architecture, the infrastructure provider manages resource provisioning, thus assuming responsibility for meeting these requirements, while achieving efficient resource utilization for cost reduction. Focusing on latency constraints, one of the main challenges in achieving these constraints is the high degree of performance variability [15]. This variability can be attributed to several factors: (i) The heterogeneity of hardware resources [16]; (ii) Interference phenomena in multi-tenant edge environments [17] where resource sharing and contention can cause the execution of one workload to interfere with another; (iii) Unawareness of workload characteristics both in terms of individual functions and overall workflow [8].

These challenges are compounded by the current management strategies for serverless workloads. Today, serverless workloads are managed either by open-source runtimes such as OpenFaaS [18] and OpenWhisk [19], which leverage container orchestrators like Kubernetes [20] for scheduling and deployment at the application level, or by managed services such as AWS Lambda [21]. While these workload orchestrators offer transparent management of virtual nodes and the overall infrastructure, the resource management decisions made by such platforms typically follow a traditional one-off approach. This approach, also employed by the native Kubernetes scheduler, means that workload placement is performed only once at the beginning of each job, neglecting future system states or changes in quality of service (QoS) targets that fail to be preserved. Nonetheless, precisely satisfying the strict requirements of latency-critical applications in the presence of heterogeneity and interference remains a significant challenge.

A common practice for meeting the expected QoS is resource overprovisioning, which often leads to significant resource waste [22], a situation that can be particularly problematic for resource-constrained edge platforms. State-of-the-art serverless frameworks apply function placement and reactive scale-out, relying mostly on virtual resources, i.e., vCPUs and memory allocation percentage, neglecting the variable impact of interference and heterogeneity on end-to-end latency [23–26]. In addition, they are unaware of application composition to apply heuristics for reducing intolerable delays. Artificial intelligence (AI), and machine learning (ML)-based approaches, which are attracting much attention lately in computer systems [27], can automatically discover complex patterns and relationships, which renders them a good fit for environments that experience a high degree of variability, uncertainty, and dynamicity. More specifically, deep reinforcement learning (DRL) forms a very effective solution in modeling environmental variability [24,28] and can be leveraged for integrating multi-layered awareness with a scheduling algorithm to derive orchestration decisions for online serverless workflow management.

In this paper, we present *Darly*, a QoS-precise, DRL-based scheduling framework for managing video analytics pipelines in serverless infrastructures, which in principle can also be applied by design to serverless workflows from other domains. Darly addresses the critical problem of regulating end-to-end latency while minimizing resource overprovisioning on heterogeneous hardware platforms. It effectively considers resource interference and dynamically changing QoS requirements, accommodating the diverse SLAs defined by different users (e.g., enterprise clients may have stricter latency requirements compared to other users). The framework is designed to handle the complexities of serverless workflows, including latency propagation effects and varying function behaviors, which further complicate scheduling decisions.

Specifically, *Darly* exploits low-level performance system monitoring events to identify interference on the underlying machines and combines this information with user-defined QoS requirements to regulate end-to-end latency of video analytics pipelines, through horizontal scaling and migration of functions. Our solution can orchestrate the deployed functions dynamically under various runtime conditions, i.e., resource stress fluctuations due to interference and/or dynamically changing QoS targets.

The key contributions of *Darly* are the following:

(i) We characterize a serverless video analytics workflow by demonstrating the impact of interference and heterogeneity on its performance. This characterization involves analyzing how resource interference and the presence of heterogeneous hardware configurations affect the latency of serverless video analytics tasks.

(ii) We have designed a DRL agent capable of addressing different QoS requirements under varying levels of resource interference on a distributed, multi-tenant, heterogeneous cluster of virtual machines (VMs). The DRL agent dynamically adjusts the placement, migration, and scaling of serverless functions to regulate the workflow's end-to-end latency, keeping it as close as possible to the user-defined QoS targets. By doing so, the agent minimizes resource waste, ensuring efficient utilization of computational resources while maintaining the desired performance levels.

(iii) We have integrated the designed DRL agent into four distinct scheduler implementations to assess the efficacy of different levels of dependence on the DRL model. These implementations include varying degrees of reliance on the DRL agent's decisions, from full control over function placement and migration to partial integration with existing scheduling mechanisms like Kubernetes and OpenFaaS. This comparative analysis helps in understanding the benefits and tradeoffs of using a DRL-based approach vs. traditional scheduling methods.

(iv) Through extensive experimental evaluation, we demonstrate that *Darly* significantly improves the management of serverless video analytics workloads. Specifically, *Darly* achieves up to 11 times fewer QoS violations compared to the Kubernetes scheduler.

The rest of the paper is organized as follows. In Section 2, we present a comprehensive overview of related works and identify the key differentiation points. In Section 3, we analyze our video analytics workflow and explore the impact of heterogeneity and interference on its performance. Finally, in Section 4, we present our proposed framework design, while in Section 5, we compare the Kubernetes scheduling with four different exploitation approaches based on *Darly*.

2. Related Work

The problems of application placement and runtime resource management have been extensively studied in the literature. In this section, we categorize and present related works that focus on the following: (i) Performance enhancement of serverless workflows; (ii) Scheduling techniques for application placement; (iii) Runtime resource tuning frameworks for QoS-aware serverless computing. *Darly* was developed with the consideration of all these aspects, while primarily contributing to the latter.

(i) The criticality of enhancing the performance of serverless workflows has been discussed in various research works [29–34], which succeed in addressing the user-defined latency requirements for a specific workload by decreasing the function's intercommunication; this accounts for a major performance bottleneck for naturally stateless serverless functions [35]. Faastlane [29] executes functions of a workflow instance on separate threads of a process to minimize function interaction latency. However, heterogeneity or resource interference that may cause unpredictable performance variability is not considered. Respectively, in Sonic [30], it is thoroughly studied in which ways inter-function data exchange could be implemented in terms of storage technologies to save execution time and costs. Also, Pocket [32] focuses on efficient data sharing, but it does not significantly consider the application's computational profile, a factor that can introduce stochasticity and performance variability.

(ii) Much research has been conducted regarding the placement of applications [16,17,26,36,37]. In [17,36], the authors design an interference-aware scheduler, focusing on batch workloads. Paragon [16] uses collaborative filtering to classify and co-locate applications targeting interference minimization. Cometes [26] targets energy consumption reduction on Edge devices, following a static approach that does not consider the dynamicity of the runtime state. Cirrus [37] improves the performance of ML training serverless workflows (time-to-accuracy) by employing several techniques to extend AWS Lambda offerings at infrastructure-level, i.e., data-prefetching, data-streaming, as well as in application-level, i.e., training algorithms redesign. Therefore, while it achieves significant performance improvement, both developer effort (for custom algorithm design) and domain-specific tuning knobs make it difficult to be generalized for serverless workflows.

(iii) In [23], reinforcement learning (RL) is employed for defining the concurrency level, i.e., the per-function concurrent request allowance before auto-scaling out. The paper focuses on homogeneous cloud servers and targets single-function applications, considering only horizontal scaling as a viable action. Additionally, it neglects interference due to co-location. Also, in [24], a reinforcement learning solution is introduced to address the cold start problem with function auto-scaling. However, their work neither considers dynamically changing QoS requirements, nor accounts for resource heterogeneity and interference. DVFaaS [38] and SequenceClock [39] employ proportional–derivative–integral (PID) control for dynamic resource allocation. DVFaaS utilizes dynamic voltage and frequency scaling (DVFS), focusing on power minimization [40], while SequenceClock employs CPU quota scaling. However, neither of these works considers function migration, nor hardware resource heterogeneity.

This paper combines DRL-based dynamic scheduling and scaling of functions for serverless video analytics workflows to meet end-to-end latency constraints. We differentiate from the most closely related prior art (iii) in the following key points: (a) We utilize low-level system metrics monitoring to capture resource interference, which was not considered in [23]; (b) We consider workflow composition and node heterogeneity as model parameters, a case not studied in [23,24,29]; (c) Compared to [30,34,36], we also adapt the decisions both to *fluctuating system-level resource pressure*, as well as to *dynamically changing QoS requirements*; (d) We extend beyond vertical scaling [39,40] and horizontal scaling [23], considering a broader range of actions, including stateless service migration. This is a key feature, necessary for straggler mitigation on edge environments.

3. Target Serverless Infrastructure and Video Analytics Pipeline Characterization

In this section, we describe the experimental infrastructure utilized for the needs of this work, and afterward, we describe and characterize the video analytics pipeline in order to explore its performance variation due to interference applied to the underlying heterogeneous cluster.

3.1. Target Serverless Infrastructure

To emulate heterogeneous Edge infrastructures, all of our experiments have been performed on a cluster of four VMs deployed on top of an on-premise, heterogeneous setup, as outlined in Table 1. Each VM included in our setup is provisioned with diverse virtual resource capacity, and is deployed on heterogeneous x86 machines that are typically used both for serverless [41], and at the Edge [42]. This setup exhibits several flavors of heterogeneity across the environment. For example, as depicted in Table 1, `w01` is composed of half the CPU cores (but they are more powerful) compared to `w03` (8 to 16), while `w01` has at its disposal double the size of L3 cache memory.

Relative heterogeneity characterizes the rest of the VMs as well, allowing us to study its impact from various angles, i.e., number of CPU cores, CPU generation, RAM size, and L3 cache size. To monitor low-level performance counters, we utilize Intel's Performance Counter Monitoring tool (PCM) [43]. We perform OS-level pinning of specific cores of the physical machines to the VMs to be able to obtain core-level system metrics from the hardware counters. We focus on specific performance counters that are discussed in detail in Section 4. All the individual functions of our video analytics workflow (see Section 3.1) have been containerized utilizing the Docker technology. To simulate a cloud environment, we utilize Kubernetes for container orchestration and OpenFaaS as the serverless runtime, where we set the number of Queue-Worker replicas to four, a practice that allows for concurrent asynchronous requests to our target workflow functions [23].

Table 1. Technical characteristics of heterogeneous nodes.

VM	vCPUs	Memory	Underlying CPU (Intel® Xeon®)	L3(MB)
worker-01 (w01)	8	15.6 GB	Gold 5218R @ 2.10 GHz	28
worker-02 (w02)	8	15.6 GB	Gold 6138 @ 2.00 GHz	28
worker-03 (w03)	16	31.4 GB	Silver 4210 @ 2.00 GHz	14
worker-04 (w04)	4	15.6 GB	E5-2658A @ 2.20 GHz	30

3.2. Target Video Analytics Pipeline

Video analytics are applications that automatically recognize temporal and spatial events in videos, which can be used to identify persons, cars, or other objects in a video stream. Video analytics are shifting from conventional "if/then" statements to more intelligent approaches, where machine learning techniques are applied to detect or classify objects in the frames of the video [44]. For this paper, we develop a video analytics workflow, similar to [45], that performs computer vision inference in human faces or objects detected in a video. The workflow highly reflects real-world pipelines of serverless functions operating on video streams, commonly used in the cloud [35,45–49].

We break down the video analytics into individual functions in a per-task manner, i.e., each function of the pipeline performs a distinct job of the overall workflow. The investigation of a decomposed version of the workflow rather than a monolithic one boosts the benefits of partial and on-demand execution of a workload that outweighs the intercommunication overhead among the functions. The pipeline by design contains both parallel (i.e., functions that support horizontal scaling) and sequential parts (i.e., functions that support only migration) for generating a more concrete and general artifact. Specifically, our pipeline is represented as a directed acyclic graph (DAG), as shown in Figure 1, and consists of five separate components:

1. **Framer:** The Framer parses the input `mp4` video file and extracts a user-defined number of frames (n). All frames are sequentially extracted from the video; thus, this function does not support horizontal scaling. After the extraction of the last frame, all collected frames are saved to MinIO remote storage [50].
2. **Face-detector:** The Face-detector uses Haar feature-based cascade classifier [51] to perform an object-detection task in the extracted frames, i.e., examine if a frame contains a human face. If yes, it forwards the frame to the *Face-analyzer*, otherwise, it forwards it to the *Object-recognition* function.
3. **Face-analyzer:** The Face-analyzer utilizes a pre-trained ResNet50 DNN model [52] and performs emotion recognition on the faces identified by the Face-detector.
4. **Object-recognition:** Using the Mobilenet convolutional neural network (CNN) [53], Object-recognition classifies objects detected in the given image into ImageNet [54] classes.

5. **Uploader:** Last, the Uploader aggregates the inference results of (3) and (4) and uploads them to remote storage.

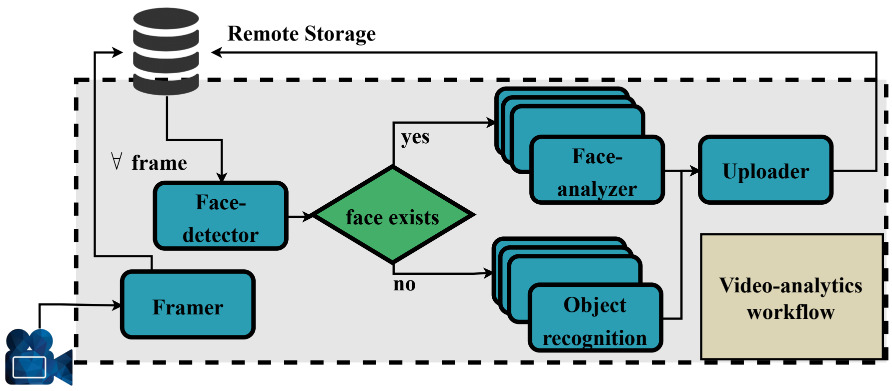


Figure 1. Video analytics workflow.

3.3. *Performance Characterization of Video Analytics Pipeline*

Next, we present a performance analysis of our video analytics pipeline under different deployment scenarios. Specifically, we analyze the execution profile of our video pipeline, as well as quantify the impact of hardware heterogeneity and resource interference on the performance of our workflow. To evaluate performance, we invoke the workflow instance and measure the time elapsed for the *Framer* and the aggregated time for the ML-models, i.e., *Face-detector, Face-analyzer,* and *Object-recognition*, with one instance each. We deactivate the scale-to-zero functionality of OpenFaaS reserving zero cold starts at all iterations.

Pipeline's Execution Profile: Figure 2 shows the performance of our pipeline for a different number of frames extracted (n) and resource contention scenarios when deployed on *w01*. First, we examine the performance of the pipeline under isolation (0% load). We see that for a lesser amount of extracted frames, the total execution latency of the pipeline is dominated by the execution time of the Framer function, which accounts for almost 85% of the overall time. However, as the number of frames increases, the execution time of the pipeline becomes balanced between the Framer and the rest of the functions due to the increased workload performed by the ML models.

Impact of interference: We further examine the sensitivity of the functions to resource interference. To do so, we spawn different amounts of cpu micro-benchmarks from the iBench suite [55], which have been validated to increase the computational load of the underlying VM almost linearly with their intensity. We break down CPU interference applied to w01 into four levels: 0%, 25%, 50%, and 75% of the total available cores, as portrayed in Figure 2. The pipeline exhibits significant performance variability, non-linearly impacted by CPU interference. This interference leads to performance degradation of up to 57.6% for the 16-frames input in the *Framer* case and up to 47.2% for the 32-frames in the ML-models functions case. This is the worst performance compared to isolated execution. Moreover, the imposed degradation does not present a linear relationship with the interference load, with CPU pressure levels below 50% imposing minimal performance degradation to all the functions.

Impact of heterogeneity: Figure 3 shows the performance variation of the examined functions with respect to resource heterogeneity. Resource heterogeneity in our setup includes combinations of varying core numbers, memory sizes, and diverse CPU architectures, as described in Table 1. Functions allocate all available VM's CPU cores at runtime, meaning that a VM, (e.g., w04) with fewer cores provides less multi-threading capacity to the hosted function, which results in poorer latency. For the *Framer* function, we find deltas with a maximum value of 23% and a minimum of 10% performance variation in the 16-frames and 65-frames deployments, respectively. Moreover, for the ML-models functions, the measured deltas have a maximum value of 34% and a minimum of 5% variation, respectively. Overall, we observe that the impact of resource heterogeneity becomes more perceptible as the number of frames increases, due to the accumulated computational burden of less powerful hardware resources. Also, despite the variation in the available resources (vCPUs, memory) per VM, *w01* and *w02* provide the overall best performance, due to the lack of vertical scaling mechanisms within the functions.

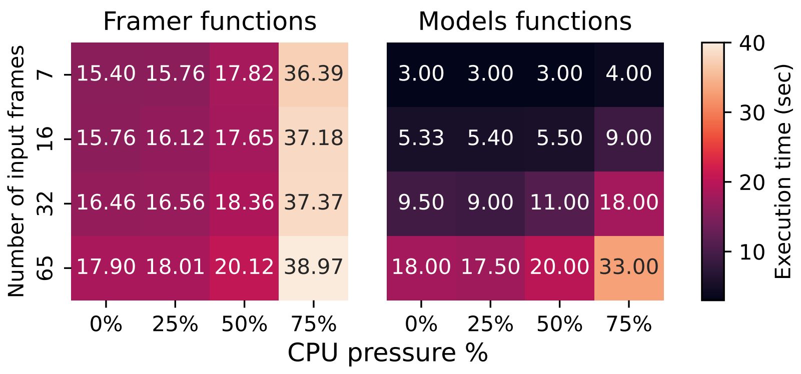


Figure 2. Impact of interference on serverless functions.

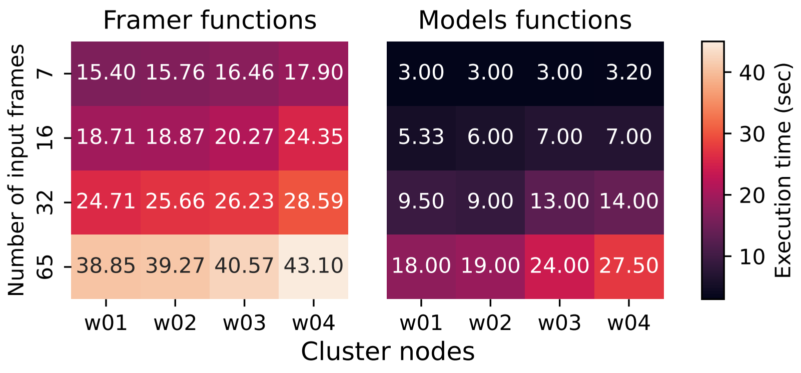


Figure 3. Impact of heterogeneity on serverless functions.

4. *Darly*: A Dynamic DRL-Based Scheduler

In modern cloud environments, the design of a dynamic scheduler that is aware of node heterogeneity, resource interference from third-party applications, and is resistant to fluctuations caused by unpredictable user demand forms a very challenging and complex problem. Based on the observations made in Section 3.3, we leverage DRL to design *Darly*, a Dynamic DRL-based Scheduler for one of the most widely adopted open-source serverless platforms, OpenFaaS. We implement a scheduling strategy that adequately interprets the state of a heterogeneous cluster while being aware of unpredictable interference applied by co-existing third-party workloads. Our scheduler receives a request with a user-defined QoS and after scanning the cluster state, orchestrates the workflow functions to the appropriate nodes to optimize resource utilization and end-to-end latency while serving the request successfully. The proposed framework, shown in Figure 4, consists of four components: a `System Monitor` which monitors and collects metrics representing the system's state, a `DRL-based agent` which reads the system metrics and calculates the next action to be performed on the deployed functions regarding the specified QoS, a `Runtime Engine` that, given the functions' current placement, is responsible for orchestrating the execution of a workflow instance and a function `Mapper` which maps a function to a node according to the scheduler's latest decision.

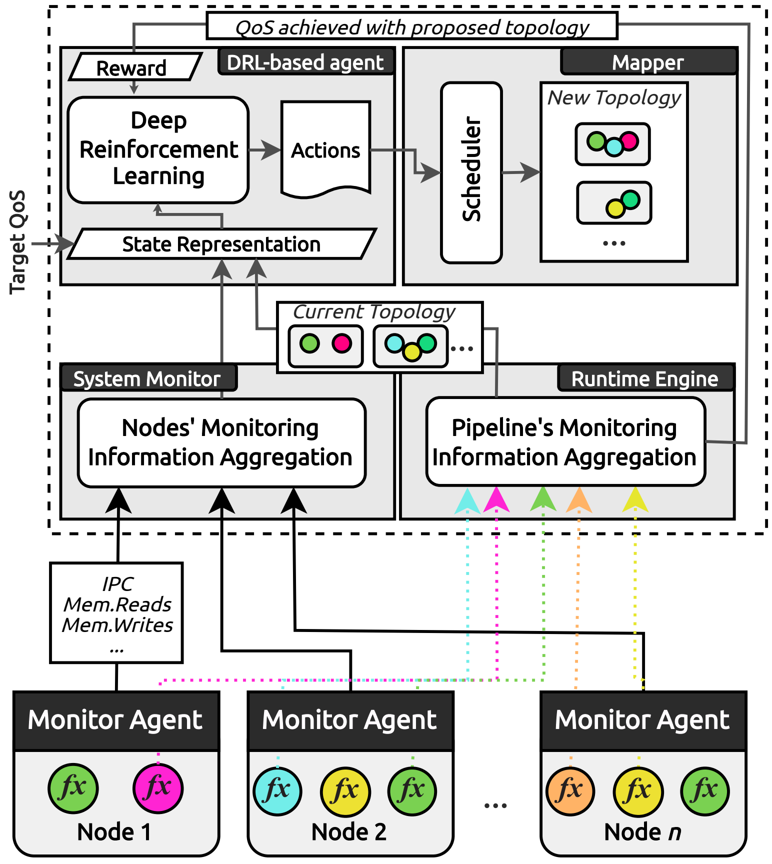


Figure 4. Framework overview.

4.1. System Monitor

The System Monitor is responsible for gathering the required data for system-related state representation, e.g., low-level performance hardware events. It aggregates system-, socket-, and core-level information from distributed PCM [43] monitoring agents (MA) through a high-performance time series database (InfluxDB) [56]; hence, the current cluster state is formulated. For the rest of this paper, we employ the following five performance counters: *Instructions Per Cycle (IPC)*, which is an approximate indicator of the performance of the processor, providing valuable insight into the efficiency of the deployed functions. In prior works, IPC has been used as a metric of interest to depict performance-related behaviors [17,57]. *Memory Reads/Writes* that depict the access patterns from/to the DRAM memory, which is considered a major bottleneck in modern server systems. The amount of memory reads and writes performed in a time period could be a highly accurate indicator of a system's load. *L3 Cache Misses* lead to increased memory reads/writes; thus, it is critically representative of performance. *C-States (C0, C1)*: For energy-saving reasons during the CPU's idle state, the CPU could be forced to enter a low-power mode. Each core has three scaled idle states: C0, C1, and C6. C0 is the normal CPU operating mode, where the CPU is 100% active. The higher the C index is, the less activated the CPU is, which differentiates the utilization ratio.

4.2. Runtime Engine

The purpose of the Runtime Engine (RE) is to initiate asynchronous function requests, monitor the activity and application-level performance by polling logs from OpenFaaS components, and manage functions' intercommunication. We develop a custom RE that extends OpenFaaS and supports both synchronous and asynchronous function invocations through the Queue-Worker modality. The Queue-Worker is used by OpenFaaS for processing asynchronous requests to enable parallel execution of functions [58]. In a functions' topology agnostic fashion, upon request, RE administers the execution of a workflow instance and monitors the end-to-end latency. RE utilizes any number of available OpenFaaS Queue-Workers transparently, with no need for extra configurations.

4.3. DRL-Based Agent

DRL is employed for pattern discovery in scenarios that are hard to model by typical techniques. The problem we investigate falls within this spectrum, as it involves high dimensionality by considering low-level system metrics, multiple QoS levels, interference levels, and various constraints. We employ Deep Q-Network (DQN), to solve the described problem. Following the RL narrative, we consider the presence of an *agent* inside an *environment* with which it interacts at discrete time steps t. Given a state $S_t \in S$, where S is the set of all possible *states* of the environment, the agent can choose an action A_t among a discrete set of available actions A and force the transition of the environment to a new state S_{t+1} while receiving a *reward* $r \in R \subset \mathbb{R}$. Our scheduler, i.e., the DRL-based agent, aims to maximize the cumulative reward received by its interactions with the environment over time. We consider as *policy* π, a function that maps an action to be taken by the agent to a probability distribution of available actions at a given state.

$$\pi : A \times S \rightarrow [0,1] \tag{1}$$

4.3.1. Policy Optimization

For the purposes of our scheduler, we use Q-learning, a widely adopted value function-based RL algorithm that learns the value of the agent being present in a state, and based on this state, selects a specific action, $Q(s,a)$. A Q-function (Q) is used for learning the transition probabilities across all available actions and states and can be represented as a table of states and actions, with each entry $Q(s,a)$ representing the estimated reward for choosing action a in state s. In the case of DRL, the Q-function is modeled by a deep neural network (explained later). Essentially, the DRL-based agent in each decision-making step has to cope with a discrete set of actions, $A \subset \mathbb{N}$. This is often modeled in literature as a Markov decision process [59] (MDP). In this work, we model our MDP with tuples $(S, A, R, \mathcal{T}, \gamma)$ that represent states that can be generated in our environment. Rewards R and transitions probabilities $\mathcal{T}$ are unknown; thus, for solving the problem, the agent has to calculate $\mathcal{T}(s'|s,a)$ and $R(s'|s,a)$ of taking action $A_t = a$ in the state $S_t = s$ and moving to state $S_{t+1} = s'$. The objective is to discover an optimal policy π^* that maximizes the expected return $G(t)$ of rewards over time. $G(t)$ is defined as the sum of weighted rewards received within a time period T:

$$G(t) = \sum_{\tau=0}^{T} \gamma^{\tau} r_{\tau}, \gamma \in [0,1], \tag{2}$$

where $r_\tau \backsim R(s'|s,a)$ is the reward at time step t for taking action a and making a transition to state s′ and γ is a weighted discount factor often used for the convergence of the Bellman Equation [60], as it gradually reduces the importance of future rewards [61].

Bellman proved that if an agent's decisions include the highest Q-values, then its policy is optimal and leads to the maximum $G(t)$ as well [60]. Once the Q-values are estimated, the optimal policy could be extracted: $\pi^* = argmax_\pi Q_\pi(s,a)$, for all $s \in S$ and $a \in A$. However, since our target environment is complex, i.e., continuous state space, dynamic programming algorithms do not scale with enough efficacy for large combinations of Q-values, $Q(s,a)$; thus, we approximate them by utilizing a DQN, $Q(s,a,\phi) \simeq Q^*(s,a)$, where ϕ stands for the parameters learned by the neural network (NN).

4.3.2. State Encoding

Our state representation $S : \langle IPC, L3m, C0, C6, L_a \rangle$ is modeled in a continuous fashion and includes (i) *low-level metrics* that contribute to the agent's interpretation of the system's current conditions and (ii) *end-to-end execution latency* L_a of the target workflow as performed in the previous chain invocation that indicates the computing capacity of the topology in the previous state S_{t-1} when deciding to modify it or not for state S_t.

4.3.3. Action Set

Darly considers that actions are applied in a per-function basis. The set of available actions A include the following: (i) function *horizontal scaling* by tuning the number of function replicas; (ii) function *migration* to different nodes; (iii) inactivity, i.e., preserving the function topology as is. Notably, we avoid any service disruption during migration by implementing a rolling-update strategy. Additionally, any latency overhead induced due to function migration or horizontal scaling is reflected in the reward, allowing the agent to consider it in its learning process. Recent studies [62] have shown that in production-level environments, 95% of the serverless applications have at most 10 functions, which in practice eliminates any scalability issues of *Darly*'s per-function action space encoding. After performing the selected action, the system transitions to a new state S_t, according to a probability distribution function. After each step, the agent receives a reward R based on the reward function that is described in Equation (3).

4.3.4. Rewarding Strategy

The incentive behind the construction of the reward function (Equation (3)) is the regulation of the execution latency by striving not to violate the latency threshold L_t set by the user (QoS), similarly to [24]. The rewarding function can also be while we attempt to minimize both the number of utilized servers sp (maximum of N) and the replica count r for each function (maximum of Rs), parameters that depict the cost of resource reservations. We consider a server utilized if at least one function is placed there. Moreover, it is possible to deploy more than one function replica to accelerate the workflow execution, a policy that inevitably allocates more cluster resources, thus increasing billing costs [63]. The higher (lesser, respectively) the resource reservation for successfully serving a user's request ($L_a \leq L_t$), the poorer (richer, respectively) the reward offered to the agent. Also, $k1$, $k2$, and $k3$ are parameters that can be tuned accordingly, based on the acceptable tradeoff between violation penalties and resource usage, depending on the situation.

$$R = \begin{cases} \frac{N}{sp} + \frac{Rs}{r} + \frac{L_a}{L_t} \times k_1, & \text{if } L_a \leq L_t \\ \max(-k_2, -k_3 - \frac{L_a}{L_t}), & \text{otherwise} \end{cases} \tag{3}$$

The DRL-based agent forms the decision-making component of *Darly* (Figure 4). It accounts for (i) the fluctuations in system-state, caused by resource interference through the System Monitor, (ii) the actual end-to-end latency achieved through the Runtime Engine, and (iii) the dynamically changing user-specified QoS requirements. Finally, the DRL-based agent selects the required action, e.g., move $function_i$ from $node_j$ to $node_{j+1}$, and forwards it to the Mapper for reconfiguring, if necessary, the function topology among the cluster.

4.4. Mapper

Mapper is the component that carries out the decision of the DRL-based agent. It is aware of the current topology of the workflow, and conducts the required set of operations by interacting with the OpenFaaS API. As we will discuss in Section 5, the Mapper component is capable of implementing any kind of scheduling mechanism that effectuates the DRL-based agent's decision.

4.5. Technical Implementation

We utilize the OpenAI Gym's API [64] and Stable-Baselines-3 framework [65], leveraging the Deep-Q-Network reinforcement learning algorithm [66]. The pseudocode of Darly is described in Algorithm 1. The NN parameters of the DRL agent are tuned by training enough agents that converge in various ways to the desired result. The *state* is a 35-dimensional vector while the *action* vector ranges from 4 to 15 dimensions, depending on the scheduling policy deployed on each scenario, as described in Section 5. The employed

NN consists of three hidden layers with 256, 128, and 64 features each. We set the *minibatch size* to 32 and the *target update interval* to 60 for the target network. We use Adam as our optimizer, with a learning rate α of 0.0025. Also, for keeping the agent's horizon short in our approach, we tune the discount factor *gamma* to 0.99 since the problem setup is quite stochastic for investing more in long-term rewards. Last, but not least, ReLu is employed as the activation function, along with a replay buffer of size 10^6 for simulating the agent's experience at a certain moment. Finally, we tune the reward hyper-parameters $k1, k2$, and $k3$ of Equation (3) to 3, 6, and 4, respectively.

Algorithm 1 Darly Algorithm

```
 1: Initialize replay memory $\mathcal{D}$ to capacity $N$
 2: Initialize target update interval $T$
 3: Initialize learning rate $\alpha$
 4: Initialize discount factor $\gamma$
 5: Initialize exploration rate $\epsilon$
 6: Initialize action-value function $\mathcal{Q}$ with random weights
 7: for episode = 1, $M$ do
 8:    Initialize sequence $s_1 = x_1$ and pre-processed sequence $\phi_1 = \phi(s_1)$
 9:    for t = 1, $T$ do
10:       With probability $\epsilon$ select a random action $a_t$
          otherwise select $a_t = max_a Q^*(\phi(s_t), a; \theta)$
11:       Execute action $a_t$ in emulator and observe reward $r_t$ and state $s_{t+1}$
12:       Set $s_{t+1}$, $a$ and pre-process $\phi_{t+1} = \phi(s_{t+1})$
13:       Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
14:       Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
15:       if $\phi_{j+1}$ is terminal then
16:          $y_j = r_j$
17:       else
18:          $y_j = r_j + \gamma * max_{a'} Q(\phi_{j+1}, a'; \theta)$
19:       end if
20:    end for
21:    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
22: end for
23: Return result
```

5. Results

Evaluation Criteria: We evaluate our proposed framework with respect to its efficiency to identify the appropriate set of actions (function migration and/or horizontal scaling) to satisfy the pre-defined latency constraint while allocating the least amount of cluster resources. Our desired metric of interest is the QoS *quotient*, which is the quotient of the execution latency achieved divided by the target, user-specified latency requirement. A quotient less than or equal to 1 implies a successful QoS serving, while a quotient greater than 1 suggests a QoS violation. Furthermore, the closer to 1 a successful QoS serving is, the more regulated the end-to-end execution latency is considered, i.e., the QoS_{target} is achieved without over-allocating resources; thus, utilization optimization is fulfilled.

Experimental Conditions: We set two QoS levels, relying on the performance characterization presented in Section 3.3, i.e., 35 and 26 s, which correspond to looser and stricter constraints, respectively. Training on discrete levels of QoS is essential for exposing the DRL-based agent to a wide enough spectrum of states to boost its ability to identify patterns among (*state*, *action*) pairs. During the experiments, we dynamically change the underlying interference on the cluster by randomly altering the number of cpu micro-benchmarks per VM. Last, each pipeline invocation is set with a different amount of frames to be extracted (n) from the video, a process that modifies the input data size and thus highly impacts the workflow end-to-end latency.

Examined Schedulers: We examine four different scheduling policies as part of the Mapper component (Section 4.4) to determine the inter-relationship between the DRL agent's proposed actions and the employed scheduling mechanism. With this approach, we aim to quantify the impact of (i) the scheduling granularity when migrating functions and (ii) heterogeneity- and interference-awareness with our proposed framework. Specifically, we developed four distinct schedulers with varying action spaces to assess the efficacy of different levels of dependence on the DRL model. These include full dependence (*Fullmap-based*), expert knowledge (*Custom-based*), zero dependence (*Kubernetes-based*), and accumulated knowledge (*Profile-based*).

- *Fullmap-based:* Decides both the migration and the destination of a function (i.e., move $function_i$ from $node_j$ to $node_{j+1}$), while having the freedom to relocate any function to any node.
- *Custom-based:* Decides both the migration and the destination of the *Framer* and *Face-detector* functions, but only the migration (if chosen to be performed) for the *Face-analyzer* and *Object-recognition* functions since their destination node will always be the least loaded node. In this way, we investigate whether giving the agent partial or full freedom upon the landing node makes any difference to convergence speed and quality.
- *Kubernetes-based:* While Kubernetes does not support migration, we consider a Kubernetes -based policy that decides the migration of a function (i.e., move $function_i$) and afterward, the native Kubernetes scheduler is employed for determining the migrating node based on its own scheduling policy. In this way, we examine the functionality enhancement of the kube-scheduler that, by default, is unaware of the individual performance characteristics of functions.
- *Profile-based:* Again, decides just the migration of a function but the destination node is chosen by leveraging knowledge extracted from offline profiling that was performed in Section 3.3, where each function's performance is analyzed under various circumstances; therefore, we can make an accurate enough estimation of its latency before deciding the landing node.

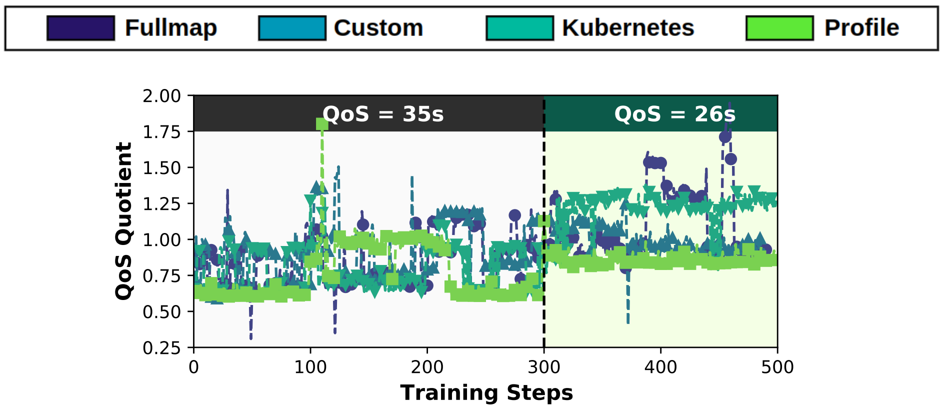
5.1. Comparative Evaluation of Schedulers during Training

First, we examine the ability of the DRL agent to learn the appropriate actions to effectively adapt to dynamic interference conditions and QoS requirements during training. For the first 300 training steps, we set a loose QoS value of 35 s, and for the other 200, a stricter QoS of 26 s is set. We evaluate the four examined schedulers through three different aspects: (a) QoS quotient; (b) Cumulative Reward; (c) QoS Violation Ratio. Figure 5 shows the respective results.

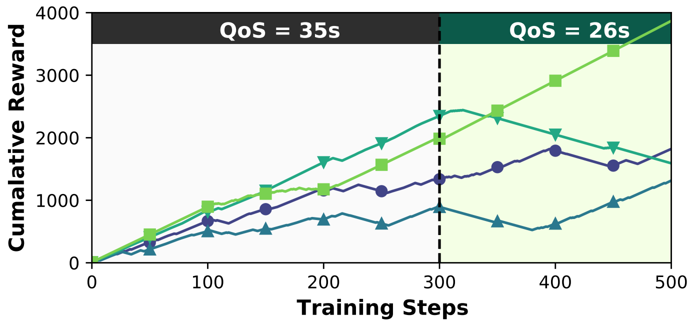
QoS quotient: As depicted in Figure 5a, both the *Profile-based* and *Custom-based* approaches demonstrate valuable stability in response to changes in resource stress levels and the transition to a stricter QoS. They adapt relatively quickly while maintaining a value close to 1, indicating precise adherence to QoS.

The stability of the *Profile-based* approach can be attributed to its use of historical data from previous executions for action selection. In contrast, the *Fullmap-based* scheduler shows similar but less stable behavior, and does not remain as close to 1. This is partly due to its larger action space compared to the *Custom-based* approach, which has a reduced action space by omitting some placement combinations, guided by observations from our motivational analysis (Section 3.3). Last, the *Kubernetes-based* approach, even though it seems to be expectantly adaptive at first, during the strict QoS period, it fails to adjust its decisions to the occurring conditions. This is attributed to the best-fit heuristic that Kubernetes scheduler uses, neglecting heterogeneity and interference.

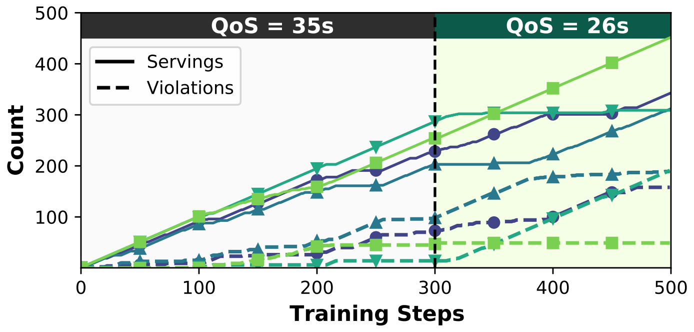
Cumulative Reward: The reward achieved per scheduler reveals its effectiveness in identifying a more efficient function topology in terms of resource utilization since our proposed reward function is designed to maximize the reward with respect to resource efficiency (Section 4.3). Figure 5b shows the received rewards per agent over time. Again, as expected, the *Profile-based* approach is the most dominant, as it converges within 10 training steps in any change of conditions and QoS. Hence, it achieves the goals formulated in the Rewards function (Equation (3)). The *Fullmap-based* scheduler seems to secure a greater amount of reward than the *Custom-based*, but the *Custom-based* approach presents a steeper slope at the latest stages of training, which hints a better knowledge of the given task that leads to better results in the future. Finally, the *Kubernetes-based* scheduler as highlighted above because of its inability to adjust to the QoS change, achieved no convergence in the stricter QoS level and thus failed to receive positive rewards in the second half of the training period.



(**a**) QoS quotient



(**b**) Cumulative reward



(**c**) QoS violations

Figure 5. Comparative evaluation of different schedulers during training of the DRL.

QoS Violation Ratio: Figure 5c shows the successful QoS servings and QoS violations per scheduler. *Fullmap-based* and *Custom-based* prove that DRL has the ability to tackle the dynamic scheduling problem since QoS successful servings outnumber the QoS violations. The *Profile-based* agent serves most of the requests with great ease, sealing the ability of DRL to collaborate with workload performance features, showing the criticality of a highly efficient scheduler. Last, but not least, the *Kubernetes-based* scheduler, as mentioned above, seems to be the worst performer among all approaches.

5.2. Decision-Making Analysis of the DRL Agent

As shown previously, the final reward claimed by the DRL agent highly depends on the integrated scheduling logic. So a question that arises is as follows: *"Does the DRL agent alter its decision based on the integrated scheduler?"* To evaluate the "intelligence" of the DRL for different schedulers, we freeze the parameters of the DQN and examine the action space per scheduler during deployment. We explore different phases, where each one is characterized by different interference and QoS levels. Figure 6 shows the respective results, where the top figure reveals the interference pressure per VM and the middle and bottom ones revealthe QoS Quotient and the actions made per scheduler, respectively. This figure reveals three major insights: (i) The first phase is characterized by high diversity in the action space, since none of the schedulers is able to satisfy the target QoS. (ii) In the second phase, the Profile-based agent is the most capable of meeting the target QoS. Compared to the rest of the schedulers that mostly migrate the Framer function, the Profile-based agent satisfies the QoS by migrating the Face-Detector, even though the Framer accounts for the greater part of the workflow's latency (Section 3.3). (iii) Lastly, even in cases with minimal interference (Phase 4) or with similar decision patterns with the Profile-based (Phase 5), the Kubernetes-based agent is unable to satisfy the target QoS due to its unawareness both regarding the interference of the underlying infrastructure and the functions' performance variability due to heterogeneity.

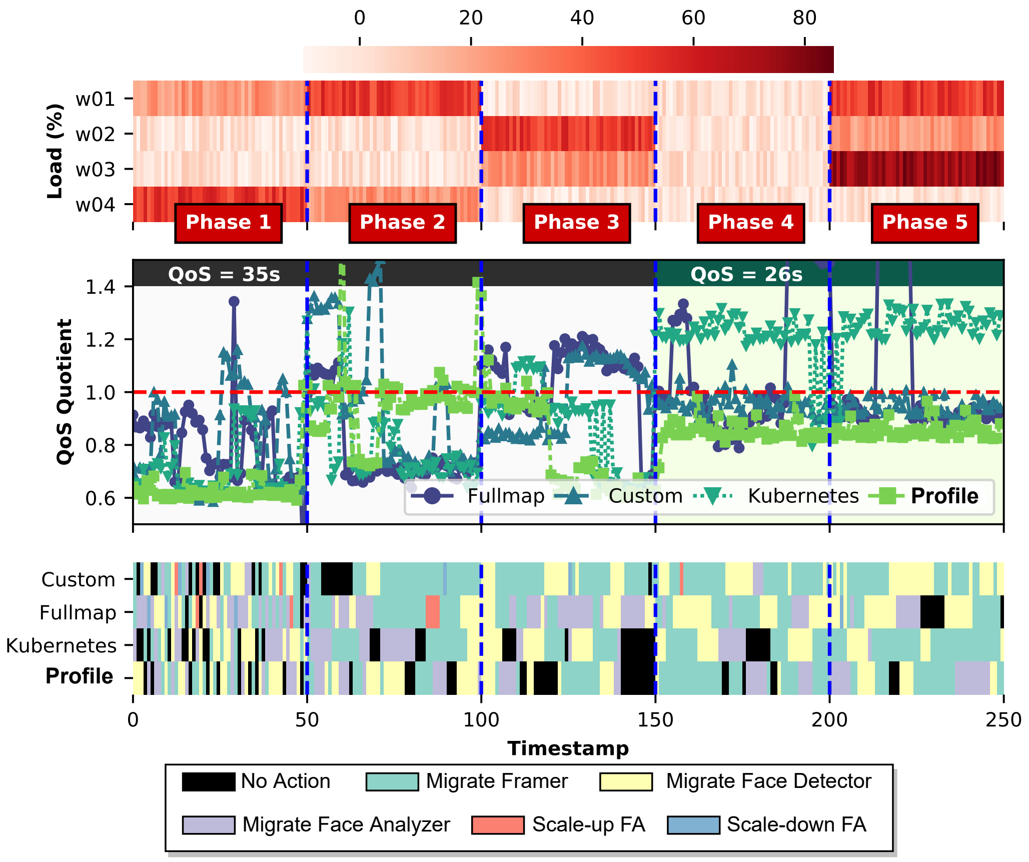


Figure 6. Interference level, QoS Quotient, and decision-making of the DRL agent under different scheduling policies.

Retraining Overhead: We note that the retraining overhead for any of the DRL-based schedulers is also considered because of the need to broaden the knowledge repertoire of the agent for serving additional QoS levels. More precisely, a dedicated but sharp tuning effort is required for this task. The user-defined QoS for a workflow request is one out of the thirty-five features that form the input state vector. Thus, the agent, which has already encoded performance tendencies and correlations for (*state*, *action*) pairs, needs relatively little calibration for adding a new QoS level in the state vector.

Serving different workflows: Each scheduling logic integrated into all four scheduler types presents a different approach to the granularity that characterizes the decision-making contribution of RL in our problem. The Profile-based scheduler had the most efficient convergence to the defined requests, but to obtain the offline knowledge for a different workflow, one needs to profile the target application to be orchestrated offline, a constraint that demands extra development effort. On the opposite side, the other schedulers do not require extra work for serving different applications while their performance falls not too far behind the Profile-based. Most notably, the Custom-based achieves up to 78.8% compliance to system fluctuations (only 12.8% less than Profile-based) while featuring an almost application-agnostic architecture.

5.3. DRL-Based vs. Native Kubernetes Scheduling

Lastly, we compare the four DRL-based schedulers with a straight-forward orchestration approach (typical in current deployments), where containers are orchestrated solely by Kubernetes without any interaction with the DRL component in attempting to highlight the need for a scheduling framework if serving user-constrained requests is the goal. We deploy the video pipeline as separate containers sequentially with one replica per container, and we run 500 iterations of the workflow with different QoS constraints and resource interference. For the loose QoS constraint, Kubernetes manages to satisfy the target QoS only 34% of the time, whereas for strict QoS, it fails to satisfy the constraint 100% of the time. In contrast, the DRL achieves the target almost 95% of the time for loose and 75% for strict QoS on average, respectively.

5.4. Darly's Performance Overhead

Darly monitors low-level resource metrics, performs inference for decision-making, and orchestrates the execution of a workflow instance, thus inevitably imposing an overhead on the underlying system's performance. Firstly, all framework components (i.e., System Monitor, DRL-based agent, Runtime Engine, Mapper) are deployed to the master node of the cluster and hereby do not contribute any interference or resource contention to the worker nodes, where the workflow computations are performed. For calculating the execution overhead, we invoked the workflow for a fixed number of requests with and without *Darly* and investigated the average time needed for producing the desired output. Therefore, *Darly*'s overhead concerns solely the added latency for the end-to-end execution, i.e., System Monitor's delay for gathering PCM metrics from worker nodes, DRL-based agent's inference time for decision-making, and Mapper's delay for performing the action and their intercommunication costs. After 100 requests in both scenarios, we observed an average performance overhead of 8.6% by *Darly* when workflow executions were carried out in the same worker nodes at all times, with no third-party interference applied to the cluster.

6. Benefits and Challenges of Integrating Darly into IoT/MEC Infrastructures

Notably, Darly was designed, trained, and evaluated on real hardware, ensuring that both resource interference and noise were present during the evaluation experiments. It was also deployed on widely used frameworks in the Cloud-Edge continuum, such as Kubernetes for container orchestration, OpenFaaS for serverless function management, and Docker for containerization. These frameworks facilitate interoperability across heterogeneous IoT/MEC infrastructures. Darly can potentially benefit MEC systems, e.g., UAC-assisted, by optimizing resource allocation in resource-limited environments, mitigating stragglers through task migration, and ensuring adherence to QoS requirements for diverse clients with varying SLAs. However, several considerations must be addressed for successful deployment in real-world scenarios. The retraining and adaptation of the DRL agent to specific environments, as well as challenges regarding scalability, such as supporting many UAVs, need to be taken into account. Nonetheless, Darly's potential to enhance MEC system efficiency and performance makes it a promising addition to real-world applications.

7. Conclusions

The management of latency-critical serverless workloads serving IoT applications at the Edge can be significantly affected by resource interference and device heterogeneity. Consequently, performance variability can undermine the QoS offered by infrastructure providers. To address this challenge, we propose *Darly*, a DRL-based scheduling framework designed for dynamic function scaling and scheduling of serverless video analytics. Our solution employs an AI-driven solution, utilizing low-level system monitoring as part of the RL state representation to capture interference phenomena. It adapts to resource pressure fluctuations and considers hardware heterogeneity, achieving up to 91.6% adherence to changing QoS targets, compared to just 34% with the native Kubernetes scheduler.

Author Contributions: Conceptualization, D.G., A.T., D.M., S.X., F.C. and D.S.; formal analysis, D.G., A.T., D.M. and S.X.; funding acquisition, F.C. and D.S.; investigation, D.G., A.T., D.M. and S.X.; methodology, D.G., A.T., D.M. and S.X.; project administration, D.S.; software, D.G. and A.T.; supervision, A.T., D.M., S.X., F.C. and D.S.; validation, A.T.; visualization, D.G.; writing—original draft preparation, D.G., A.T. and D.M.; writing—review and editing, A.T., D.M., S.X., F.C. and D.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the HFRI (Fellowship Number: 5349).

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Asim, M.; Wang, Y.; Wang, K.; Huang, P.Q. A review on computational intelligence techniques in cloud and edge computing. *IEEE Trans. Emerg. Top. Comput. Intell.* **2020**, *4*, 742–763. [CrossRef]
2. Lu, Y.; Chowdhery, A.; Kandula, S. Optasia: A relational platform for efficient large-scale video analytics. In Proceedings of the 7th ACM Symposium on Cloud Computing, Santa Clara, CA, USA, 5–7 October 2016; pp. 57–70.
3. Musalem, A.; Olivares, M.; Schilkrut, A. Retail in high definition: Monitoring customer assistance through video analytics. *Manuf. Serv. Oper. Manag.* **2021**, *23*, 1025–1042. [CrossRef]
4. Baresi, L.; Filgueira Mendonça, D.; Garriga, M. Empowering low-latency applications through a serverless edge computing architecture. In *Proceedings of the Service-Oriented and Cloud Computing: 6th IFIP WG 2.14 European Conference, ESOCC 2017, Oslo, Norway, 27–29 September 2017*; Proceedings 6; Springer: Berlin/Heidelberg, Germany, 2017; pp. 196–210.
5. Lyu, X.; Cherkasova, L.; Aitken, R.; Parmer, G.; Wood, T. Towards efficient processing of latency-sensitive serverless dags at the edge. In Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking, Rennes, France, 5–8 April 2022; pp. 49–54.
6. Jonas, E.; Schleier-Smith, J.; Sreekanti, V.; Tsai, C.C.; Khandelwal, A.; Pu, Q.; Shankar, V.; Carreira, J.; Krauth, K.; Yadwadkar, N.; et al. Cloud programming simplified: A berkeley view on serverless computing. *arXiv* **2019**, arXiv:1902.03383.
7. Fouladi, S.; Wahby, R.S.; Shacklett, B.; Balasubramaniam, K.V.; Zeng, W.; Bhalerao, R.; Sivaraman, A.; Porter, G.; Winstein, K. Encoding, Fast and Slow:{Low-Latency} Video Processing Using Thousands of Tiny Threads. In Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), Boston, MA, USA, 27–29 March 2017; pp. 363–376.
8. Romero, F.; Zhao, M.; Yadwadkar, N.J.; Kozyrakis, C. Llama: A heterogeneous & serverless framework for auto-tuning video analytics pipelines. In Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, 1–4 November 2021; pp. 1–17.
9. AWS Lambda @ Edge. Available online: https://aws.amazon.com/lambda/edge/ (accessed on 10 May 2024).
10. Tzenetopoulos, A.E.A. FaaS and Curious: Performance Implications of Serverless Functions on Edge Computing Platforms. In Proceedings of the International Conference on High Performance Computing, Virtual Event, 24 June–2 July 2021; pp. 428–438.
11. Rausch, T.; Hummer, W.; Muthusamy, V.; Rashed, A.; Dustdar, S. Towards a serverless platform for edge {AI}. In Proceedings of the 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19), Renton, WA, USA, 10–12 July 2019.

12. Pfandzelter, T.; Bermbach, D. tinyfaas: A lightweight faas platform for edge environments. In Proceedings of the 2020 IEEE International Conference on Fog Computing (ICFC), Sydney, Australia, 21–24 April 2020; pp. 17–24.
13. Russo, G.R.; Cardellini, V.; Presti, F.L. A framework for offloading and migration of serverless functions in the Edge–Cloud Continuum. *Pervasive Mob. Comput.* **2024**, *100*, 101915. [CrossRef]
14. Patterson, L.; Pigorovsky, D.; Dempsey, B.; Lazarev, N.; Shah, A.; Steinhoff, C.; Bruno, A.; Hu, J.; Delimitrou, C. HiveMind: A hardware-software system stack for serverless edge swarms. In Proceedings of the 49th Annual International Symposium on Computer Architecture, New York, NY, USA, 18–22 June 2022; pp. 800–816.
15. Ginzburg, S.; Freedman, M.J. Serverless isn't server-less: Measuring and exploiting resource variability on cloud faas platforms. In Proceedings of the 2020 Sixth International Workshop on Serverless Computing, Delft, The Netherlands, 7–11 December 2020; pp. 43–48.
16. Delimitrou, C.; Kozyrakis, C. Paragon: QoS-aware scheduling for heterogeneous datacenters. *ACM SIGPLAN Not.* **2013**, *48*, 77–88. [CrossRef]
17. Tzenetopoulos, A.; Masouros, D.; Xydis, S.; Soudris, D. Orchestration Extensions for Interference-and Heterogeneity-Aware Placement for Data-Analytics. *Int. J. Parallel Program.* **2024**, *52*, 298–323. [CrossRef]
18. OpenFaas. Available online: https://www.openfaas.com/ (accessed on 10 May 2024).
19. Baldini, I.; Castro, P.; Cheng, P.; Fink, S.; Ishakian, V.; Mitchell, N.; Muthusamy, V.; Rabbah, R.; Suter, P. Cloud-native, event-based programming for mobile applications. In Proceedings of the International Conference on Mobile Software Engineering and Systems, Austin, TX, USA, 16–17 May 2016; pp. 287–288.
20. Kubernetes. Available online: https://kubernetes.io/ (accessed on 1 February 2024).
21. AWS Lambda. Available online: https://aws.amazon.com/lambda/ (accessed on 22 January 2024).
22. Song, B.; Paolieri, M.; Golubchik, L. Performance and Revenue Analysis of Hybrid Cloud Federations with QoS Requirements. In Proceedings of the 2022 IEEE 15th International Conference on Cloud Computing (CLOUD), Barcelona, Spain, 10–16 July 2022; pp. 321–330. [CrossRef]
23. Schuler, L.; Jamil, S.; Kühl, N. AI-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments. In Proceedings of the 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Melbourne, Australia, 10–13 May 2021; pp. 804–811.
24. Agarwal, S.; Rodriguez, M.A.; Buyya, R. A Reinforcement Learning Approach to Reduce Serverless Function Cold Start Frequency. In Proceedings of the 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Melbourne, Australia, 10–13 May 2021; pp. 797–803. [CrossRef]
25. Wang, B.; Ali-Eldin, A.; Shenoy, P. Lass: Running latency sensitive serverless computations at the edge. In Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing, Stockholm, Sweden, 21–25 June 2021; pp. 239–251.
26. Marantos, C.; Tzenetopoulos, A.; Xydis, S.; Soudris, D. Cometes: Cross-device mapping for energy and time aware deployment on edge infrastructures. *IEEE Embed. Syst. Lett.* **2023**, *16*, 98–101. [CrossRef]
27. Wu, N.; Xie, Y. A survey of machine learning for computer architecture and systems. *ACM Comput. Surv. (CSUR)* **2022**, *55*, 1–39. [CrossRef]
28. Giagkos, D.; Tzenetopoulos, A.; Masouros, D.; Soudris, D.; Xydis, S. Darly: Deep Reinforcement Learning for QoS-aware scheduling under resource heterogeneity Optimizing serverless video analytics. In Proceedings of the 2023 IEEE 16th International Conference on Cloud Computing (CLOUD), Chicago, IL, USA, 2–8 July 2023; pp. 1–3.
29. Kotni, S.; Nayak, A.; Ganapathy, V.; Basu, A. Faastlane: Accelerating {Function-as-a-Service} Workflows. In Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC 21), Online, 14–16 July 2021; pp. 805–820.
30. Mahgoub, A.; Wang, L.; Shankar, K.; Zhang, Y.; Tian, H.; Mitra, S.; Peng, Y.; Wang, H.; Klimovic, A.; Yang, H.; et al. {SONIC}: Application-aware data passing for chained serverless applications. In Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC 21), Online, 14–16 July 2021; pp. 285–301.
31. Sreekanti, V.; Wu, C.; Lin, X.C.; Schleier-Smith, J.; Faleiro, J.M.; Gonzalez, J.E.; Hellerstein, J.M.; Tumanov, A. Cloudburst: Stateful functions-as-a-service. *arXiv* **2020**, arXiv:2001.04592.
32. Zhang, T.; Xie, D.; Li, F.; Stutsman, R. Narrowing the gap between serverless and its state with storage functions. In Proceedings of the ACM Symposium on Cloud Computing, Santa Cruz, CA, USA, 20–23 November 2019; pp. 1–12.
33. Shillaker, S.; Pietzuch, P. Faasm: Lightweight isolation for efficient stateful serverless computing. In Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 20), Online, 15–17 July 2020; pp. 419–433.
34. Klimovic, A.; Wang, Y.; Stuedi, P.; Trivedi, A.; Pfefferle, J.; Kozyrakis, C. Pocket: Elastic ephemeral storage for serverless analytics. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), Carlsbad, CA, USA, 8–10 October 2018; pp. 427–444.
35. Klimovic, A.; Wang, Y.; Kozyrakis, C.; Stuedi, P.; Pfefferle, J.; Trivedi, A. Understanding ephemeral storage for serverless analytics. In Proceedings of the 2018 USENIX annual technical conference (USENIX ATC 18), Boston, MA, USA, 11–13 July 2018; pp. 789–794.
36. Tzenetopoulos, A.; Masouros, D.; Xydis, S.; Soudris, D. Interference-aware orchestration in kubernetes. In Proceedings of the International Conference on High Performance Computing, Frankfurt/Main, Germany, 22–25 June 2020; pp. 321–330.

37. Carreira, J.; Fonseca, P.; Tumanov, A.; Zhang, A.; Katz, R. Cirrus: A serverless framework for end-to-end ml workflows. In Proceedings of the ACM Symposium on Cloud Computing, Santa Cruz, CA, USA, 20–23 November 2019; pp. 13–24.
38. Tzenetopoulos, A.; Masouros, D.; Soudris, D.; Xydis, S. DVFaaS: Leveraging DVFS for FaaS Workflows. *IEEE Comput. Archit. Lett.* **2023**, *22*, 85–88. [CrossRef]
39. Fakinos, I.; Tzenetopoulos, A.; Masouros, D.; Xydis, S.; Soudris, D. Sequence Clock: A Dynamic Resource Orchestrator for Serverless Architectures. In Proceedings of the 2022 IEEE 15th International Conference on Cloud Computing (CLOUD), Barcelona, Spain, 10–16 July 2022; pp. 81–90.
40. Tzenetopoulos, A.; Masouros, D.; Xydis, S.; Soudris, D. Leveraging Core and Uncore Frequency Scaling for Power-Efficient Serverless Workflows. *arXiv* **2024**, arXiv:2407.18386.
41. Wang, L.; Li, M.; Zhang, Y.; Ristenpart, T.; Swift, M. Peeking behind the curtains of serverless platforms. In Proceedings of the 2018 USENIX Annual Technical Conference (USENIX ATC 18), Boston, MA, USA, 11–13 July 2018; pp. 133–146.
42. Tzenetopoulos, A.; Lentaris, G.; Leftheriotis, A.; Chrysomeris, P.; Palomares, J.; Coronado, E.; Kazhamiakin, R.; Soudris, D. Seamless HW-accelerated AI serving in heterogeneous MEC Systems with AI@EDGE. In Proceedings of the 33rd International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC 2024), Pisa, Italy, 3–7 July 2024; Forthcoming.
43. Intel® Performance Counter Monitor—A Better Way to Measure CPU Utilization. 2012. Available online: https://www.intel.com/content/www/us/en/developer/articles/tool/performance-counter-monitor.html (accessed on 10 March 2024).
44. Ananthanarayanan, G.; Bahl, P.; BodÃk, P.; Chintalapudi, K.; Philipose, M.; Ravindranath, L.; Sinha, S. Real-time video analytics: The killer app for edge computing. *Computer* **2017**, *50*, 58–67. [CrossRef]
45. Ao, L.; Izhikevich, L.; Voelker, G.M.; Porter, G. Sprocket: A serverless video processing framework. In Proceedings of the ACM Symposium on Cloud Computing, Carlsbad, CA, USA, 11–13 October 2018; pp. 263–274.
46. Netflix and AWS Lambda Case Study. Available online: https://aws.amazon.com/solutions/case-studies/netflix-and-aws-lambda/ (accessed on 15 December 2023).
47. Zhang, M.; Wang, F.; Zhu, Y.; Liu, J.; Wang, Z. Towards cloud-edge collaborative online video analytics with fine-grained serverless pipelines. In Proceedings of the 12th ACM Multimedia Systems Conference, Istanbul, Turkey, 28 September–1 October 2021; pp. 80–93.
48. Zhang, H.; Shen, M.; Huang, Y.; Wen, Y.; Luo, Y.; Gao, G.; Guan, K. A serverless cloud-fog platform for dnn-based video analytics with incremental learning. *arXiv* **2021**, arXiv:2102.03012.
49. Rohan, M.; Ahmed, S.; Kaleem, M.; Nazir, S. Serverless Video Analysis Pipeline for Autonomous Remote Monitoring System. In Proceedings of the 2022 International Conference on Emerging Technologies in Electronics, Computing and Communication (ICETECC), Jamshoro, Pakistan, 7–9 December 2022; pp. 1–6.
50. MinIO. Available online: https://min.io/ (accessed on 18 November 2023).
51. Viola, P.; Jones, M. Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), Kauai, HI, USA, 8–14 December 2001; Volume 1, p. I.
52. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
53. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
54. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009; pp. 248–255. [CrossRef]
55. Delimitrou, C.; Kozyrakis, C. ibench: Quantifying interference for datacenter applications. In Proceedings of the 2013 IEEE International Symposium on Workload Characterization (IISWC), Portland, OR, USA, 22–24 September 2013; pp. 23–33.
56. InfluxDB. Available online: https://www.influxdata.com/ (accessed on 10 March 2024).
57. Masouros, D.; Xydis, S.; Soudris, D. Rusty: Runtime interference-aware predictive monitoring for modern multi-tenant systems. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *32*, 184–198. [CrossRef]
58. Queue-Worker. Available online: https://docs.openfaas.com/reference/async/ (accessed on 10 May 2023).
59. Puterman, M.L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed.; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1994.
60. Bellman, R. A Markovian decision process. *J. Math. Mech.* **1957**, *6*, 679–684. [CrossRef]
61. Kim, M.; Kim, J.S.; Choi, M.S.; Park, J.H. Adaptive discount factor for deep reinforcement learning in continuing tasks with uncertainty. *Sensors* **2022**, *22*, 7266. [CrossRef]
62. Shahrad, M.; Fonseca, R.; Goiri, Í.; Chaudhry, G.; Batum, P.; Cooke, J.; Laureano, E.; Tresness, C.; Russinovich, M.; Bianchini, R. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 20), Online, 15–17 July 2020; pp. 205–218.
63. AWS Lambda Pricing. Available online: https://aws.amazon.com/lambda/pricing/ (accessed on 10 June 2024).
64. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. Openai gym. *arXiv* **2016**, arXiv:1606.01540.

65. Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *J. Mach. Learn. Res.* **2021**, *22*, 1–8.
66. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.