10

Fair AI Experimentation on Edge Device Clusters via Distributed Orchestration in dAIEdge-VLab

Baptiste Dupertuis, Maïck Huguenin, Dorvan Favre,
Grégoire Rebstein, Margaux Divernois, and Nuria Pazos

HES-SO, Switzerland

Abstract

Efficient orchestration and scheduling are essential for managing hardware resources in clusters of edge devices, especially when benchmarking their ability to run AI models, perform on-device training, and engage in federated learning. Coordinating benchmarking requests must be responsive, consistent, and resource efficient.

The distributed nature of such clusters adds complexity, particularly in the absence of a central server for resource allocation. Each node must communicate directly with others, increasing overhead—especially when devices are dispersed across different networks.

Open-source orchestration frameworks such as Kubernetes and Nomad (with Consul) offer robust foundations, yet their distributed deployment and effective integration remain challenging.

This work presents the design of a distributed orchestrator and scheduler for fair and efficient execution of AI experiments on remote edge device clusters. The proposed approach combines priority awareness, resource sensitivity, completion awareness, and a FIFO mechanism to optimize task execution. Integration into dAIEdge-VLab—a distributed virtual laboratory for AI experimentation—will enable coordinated benchmarking and workload distribution across heterogeneous edge devices.

An implementation using Nomad and Consul demonstrates the system's ability to meet performance and fairness requirements, validating its effectiveness for scalable AI experimentation in decentralized, resource-constrained environments.

Keywords: Edge AI, Benchmarking AI Model, Distributed Cluster, Orchestration, Scheduling, Nomad.

10.1 Introduction

The dAIEdge-VLab is a distributed framework for fair, efficient, and reproducible benchmarking of AI workloads on heterogeneous edge devices. Built on Nomad and Consul, it orchestrates resources across a decentralized cluster and provides a flexible foundation for testing new scheduling algorithms focused on fairness, efficiency, and resource management in realistic edge environments.

10.1.1 Motivations

The rise of edge computing has created a need to evaluate diverse hardware platforms, from microcontrollers to GPU-enabled boards. The dAIEdge-VLab was developed to meet this need by providing a shared benchmarking environment where researchers can test models and training procedures directly on representative edge devices.

Its goal is to make AI benchmarking on the edge fair, efficient, and reproducible. The platform operates within a decentralized cluster architecture, where each host participates as part of the cluster. This structure improves scalability and robustness but also introduces coordination and scheduling challenges. To address them, the framework implements dedicated scheduling mechanisms that fairly allocate devices, balance workloads, and ensure consistent, comparable results across users and experiments.

10.1.2 Challenges in fairness, efficiency, and resource sensitivity

Designing a scheduler for the dAIEdge-VLab requires balancing three core challenges in a shared, heterogeneous, and decentralized environment:

Fairness: Ensure equitable access to limited devices, prevent monopolization, and maintain reproducible conditions regardless of cluster load.

Efficiency: Maximize utilization and minimize waiting time or wasted computation while keeping global coordination consistent despite decentralization.

Resource sensitivity: Account for diverse benchmark types and device capabilities by enforcing device affinity, one-job-per-device isolation, and adapting to real-time health states.

Together, these factors define the core scheduling problem: achieving fair, efficient, and resource-aware benchmarking across a decentralized, heterogeneous cluster.

10.1.3 Contributions of the paper

The dAIEdge-VLab orchestrator is designed for AI experimentation, embedding priority-, resource-, and completion-aware scheduling to ensure fair and reproducible benchmarking across heterogeneous edge clusters. Built on Nomad and Consul, it adopts a lightweight, semi-distributed model that balances coordination and decentralized execution.

10.1.4 Structure of the paper

This paper presents the dAIEdge-VLab, a distributed framework for fair, efficient, and reproducible benchmarking of AI workloads on heterogeneous edge devices. It outlines the motivation, key challenges, and main contributions of the work, then reviews existing orchestration systems such as Kubernetes, K3s, and Nomad, highlighting their limitations for heterogeneous edge environments.

Building on this analysis, the paper introduces the architecture developed for the dAIEdge-VLab, along with its scheduling framework and algorithms for equitable resource allocation. Finally, it presents experimental results demonstrating fairness and reproducibility and discusses future directions.

10.2 Background and related Work

This section reviews existing orchestration and scheduling approaches in cloud and edge computing.

10.2.1 Overview and research gaps

Distributed orchestration and scheduling manage workloads across computing resources. In clouds, Kubernetes [1] and Apache Mesos [2] run on large,

mostly homogeneous clusters optimized for throughput, scalability, and efficiency. By contrast, edge computing spans heterogeneous, resource-limited devices, from microcontrollers to GPU boards, with constrained compute, memory, energy, and often intermittent connectivity [3, 4], so scheduling must account for device capabilities, availability, and workload diversity.

While mature frameworks like Kubernetes and Nomad offer robust orchestration, they are not designed for launching benchmarking experiments on clusters of heterogeneous edge devices: Kubernetes (and its lightweight and edge-oriented variants such as KubeEdge [5] and OpenYurt [6]) assumes cluster-native, containerized resources, while Nomad supports containerized and standalone workloads via a scalable client-server architecture; both assume stable, continuously connected clusters and lack native mechanisms to handle wide device diversity [7].

Recent research addresses these edge-specific challenges with multi-resource-aware and load-balanced scheduling strategies. Liu et al. [8] propose dynamic multi-resource allocation in heterogeneous edge clusters, improving task throughput while reducing latency by considering device heterogeneity and load balancing. Similarly, the LR2Scheduler framework [9] introduces layer-aware, resource-balanced, and request-adaptive container scheduling, optimizing resource usage and startup time by adapting allocation based on workload demands and container image layer dependencies. Furthermore, multi-cluster layer-sharing scheduling strategies [10] exploit shared image layers across cloud-edge clusters, significantly reducing container startup latency and improving load balancing through algorithms like Probabilistic Relaxation Container Scheduling and Greedy Layer Sorting.

Despite these advances, current frameworks and research still fall short in supporting fair benchmarking in edge scenarios. Key missing features include integration of peripheral devices managed through host nodes, strict one-job-per-device isolation, reproducibility under variable load, and fair access control, all critical for evaluating scheduling policies across clusters of heterogeneous edge devices.

10.3 Architecture

The dAIEdge-VLab addresses the limitations of existing orchestration frameworks through a decentralized, scalable architecture built on Nomad, featuring dedicated scheduling mechanisms that ensure fair, isolated, and reproducible benchmarking across heterogeneous edge platforms.

It integrates distributed orchestration, service discovery, and custom scheduling to provide scalability, robustness, and transparency.

10.3.1 Assumptions

The architecture of the dAIEdge-VLab is designed using the following assumptions: each edge devices are managed by a host, a host can leave the network at any time, an edge device can become unresponsive or failing, and a benchmark running on an edge device can estimate and advertise its completion and state.

10.3.2 Topology

The dAIEdge-VLab uses a distributed topology across hosts [12] that improves scalability and reliability by removing single points of failure. These advantages come at the cost of greater system complexity and increased communication between nodes.

Figure 10.1 shows the overall topology, composed of hosts, edge devices, and users. Users can access the cluster through any reachable host, which

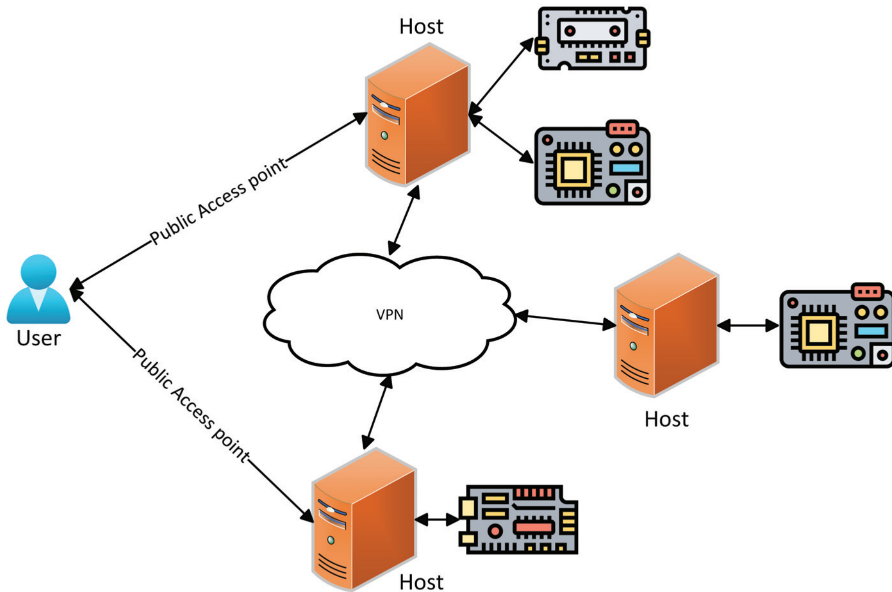

Figure 10.1 dAIEdge-VLab Topology.

serves as their entry point. Hosts are connected through a mesh VPN layer, ensuring secure and resilient communication. Each host can manage multiple edge devices, but every edge device is assigned to a single host, ensuring clear ownership.

10.3.3 Tasks

The dAIEdge-VLab supports three benchmarking services: synthetic model evaluation (TYPE 1), data-driven model evaluation (TYPE 2), and on-device training benchmarking (TYPE 3). Each task is submitted by remote users and scheduled across a shared pool of heterogeneous edge devices.

Users specify the device family (e.g., Raspberry Pi 5) on which their benchmark should run. The scheduler then assigns the job to a compatible, available device of that family.

Execution time depends on factors such as model complexity, dataset size, and device capability, but an empirical duration pattern can be observed: synthetic benchmarks typically complete fastest, followed by data-driven inference, while on-device training is the most time-consuming.

Benchmarks are non-preemptive; if a job is stopped, it must restart from the beginning to preserve a consistent device state.

10.3.4 Host architecture

Each host acts as a bridge between its local edge devices and the rest of the cluster, advertising their capabilities and current state to the network. Every host runs a VPN layer ensuring secure and persistent connectivity among all hosts. On top of this layer, Consul and Nomad instances are deployed, as illustrated in Figure 10.2.

Consul provides service discovery and cluster integrity management, ensuring that distributed services such as the *Benchmark API*, and web interface server remain reachable across the network.

Nomad handles orchestration, maintaining service health, deploying jobs, and tracking their status throughout the cluster. A custom *EdgeDevices* plugin extends Nomad to recognize and manage attached edge devices, enabling precise resource utilization and controlled benchmark deployment.

Each host runs the core services required for cluster operation and may also host application-level services such as the Web Interface or the *Benchmark API*.

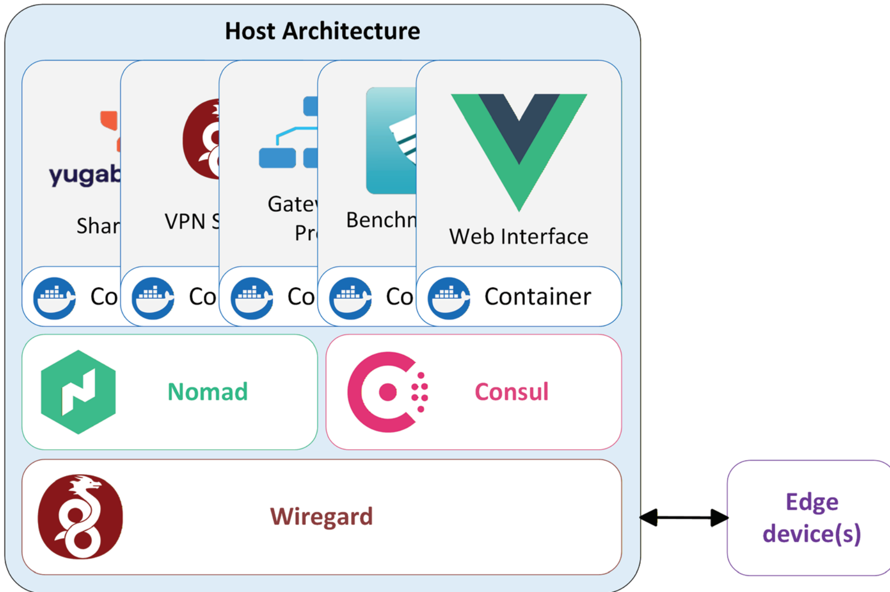

Figure 10.2 Host architecture.

10.3.5 Orchestrator and Scheduler

Scheduling decisions are handled by a dedicated scheduler integrated within the *Benchmark API*. Both components are co-located in the same service: the API acts as the entry point for user interactions, while the scheduler processes validated requests to determine job placement and execution timing. Figure 10.3 illustrates the conceptual interactions between the cluster components.

Once a job is ready for execution, the scheduler submits it to the Nomad cluster, which coordinates deployment across available hosts. Scheduling decisions rely on the aggregated cluster and edge device state (the *resource pool*) advertised by Nomad, ensuring placement only on compatible and available devices.

Before scheduling, the *Benchmark API* validates requests against available resources and user permissions. Infeasible or unauthorized submissions are rejected early.

In summary, users interact solely with the *Benchmark API*: it validates and forwards requests to the scheduler, which plans execution, while Nomad manages deployment and lifecycle operations on the target hardware.

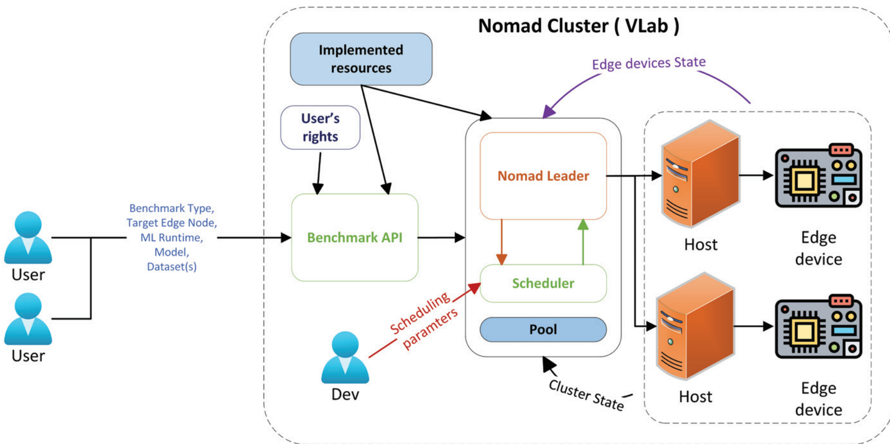

Figure 10.3 Conceptual interaction diagram.

10.4 Scheduling

The dAIEdge-VLab requires efficient scheduling of diverse benchmark workloads on its Nomad cluster. The scheduling design must therefore reconcile heterogeneity, fairness, and efficiency, while maintaining the flexibility to evolve with future workload types.

10.4.1 Requirements

The dAIEdge-VLab scheduling framework is built around functional and non-functional requirements derived from the core challenges of fairness, efficiency, and resource sensitivity, ensuring reproducible and consistent benchmarking across heterogeneous edge devices.

10.4.1.1 Functional requirements

The system must provide the following capabilities:

- **Job submission:** Users shall be able to submit one-time benchmark jobs to the cluster.
- **Job management:** Users shall be able to cancel submitted jobs and monitor their execution status.
- **Non-preemptive execution:** Running benchmarks shall not be preempted once started, to preserve consistency of results.

- **Job eviction**: Jobs may be stopped and rescheduled only if they have not yet been completed. This can be used to serve high priority job by evicting lower running ones.
- **Hardware dispatching:** The scheduler shall dispatch jobs only to hosts that provide the hardware explicitly requested by the benchmark.
- **Exclusive execution:** At most one benchmark shall run on an edge device at any given time to ensure the best performance and consistency.
- **Priority handling:** The scheduler shall consider job priorities when allocating scarce edge resources.
- **Advanced scheduling:** The system may support more advanced scheduling features, such as dynamic adaptation of priorities or user quota management.

10.4.1.2 Non-Functional requirements

In addition, the system must satisfy the following considerations:

- **High availability:** The system shall be fault-tolerant and free of single points of failure. If a host fails, jobs and services shall be rescheduled to another compatible host if available.
- **Low latency:** Jobs shall be scheduled and dispatched with minimal delay to avoid long idle times on available devices.
- **Consistency:** Jobs shall be executed only once, and benchmark results shall remain consistent and reproducible across repeated runs.

10.4.2 Objectives

The objectives of the scheduling framework can be summarized as follows:

- **Efficiency:** Maximize utilization of cluster resources without unnecessary idle times or avoiding wasting too much computation time.
- **Fairness:** Ensure equitable access for multiple users and competing benchmark submissions.
- **Scalability:** Support a growing number of benchmarks and diverse resource types as the platform evolves.
- **Flexibility:** Accommodate different scheduling policies (e.g., priority-based, resource-sensitive) depending on workload characteristics.
- **Reproducibility:** Enable consistent allocation strategies and environments, so benchmark results are comparable across runs and independent of cluster load variations.

10.4.3 Metrics

The evaluation of scheduling performance in the dAIEdge-VLab relies on a set of metrics that quantify both resource usage and user satisfaction.

- **Resource Utilization**: The ratio of total computation time spent executing jobs (including restarts) to the total available device time.

$$RU = \frac{total\ active\ compute\ time}{total\ available\ device\ time}$$

- **Resource usefulness:** The ratio of useful computation time that directly contributed to completing the benchmark to the total computation time (i.e. restarted jobs reduce this ratio, especially if the job was stopped after a long period of time).

$$RUF = \frac{Useful\ computation\ time}{Total\ computation\ time}$$

- **Latency / Wait Time**: The elapsed time between benchmark submission and the start of the execution that produced the result. This reflects user-perceived responsiveness, especially important for short-running tasks.
- **Stress ratio**: The stress ratio defines a dynamic waiting allowance for each job, assuming users tolerate longer waits for longer tasks. It compares actual waiting time W_j to an allowed delay proportional to the job's execution time T_j:

$$S_j = \frac{W_j}{\alpha_k \cdot T_j}$$

  Where α_k is a type-specific scaling factor (e.g. 3 for TYPE 1) it adapts the waiting allowance time for each type. When S_j is $< = 1$, this means that the waiting time of the job was within its waiting allowance. For $S_j > 1$, this means that the job waited longer than its allowance.
- **Compliance rate**: This metric simply counts the percentage of jobs that waited a time within their allowance.
- **Average service score for a user and benchmark type:** The service score quantifies how well a job was served relative to its allowed waiting time. It is derived from the stress ratio S_j and bounded between 0 and 1.

$$Q_j = \min\left(1, \frac{1}{S_j}\right)$$

  A value of $Q_j = 1$ indicates the job was executed within its allowance (good service), while lower values reflect increasing delay beyond the acceptable threshold.

The Average service score for a user and benchmark type is simply the average service score that a user experienced for a given type of benchmark.

$$\overline{Q}_u = \frac{1}{n_u} \sum_{j \in user\ u,\quad type\ t} Q_j$$

Where n_u is number of benchmarks type that the user has in the considered experiment.

- **Fairness Index:** A Jain's index [11] is then computed from per-user average service scores for a benchmark type to assess equality of treatment across users:

$$J\left(\{\overline{Q}_u\}\right) = \frac{\left(\sum_u \overline{Q}_u\right)^2}{n \cdot \sum_u \overline{Q}_u{}^2}$$

Where Q_u is the mean service score of user u for the considered benchmark type and *n* the number of users that has at least one benchmark of the considered type.
This index measures fairness, not absolute performance. Values close to 1 indicate that users were treated evenly.
- **Job Completion Success Rate**: The percentage of benchmarks that complete successfully without eviction or restart due to resource contention.

These metrics provide both a system-level view (efficiency and resource use) and a user-level view (responsiveness and fairness) of scheduling performance within the dAIEdge-VLab.

10.4.4 Techniques

The dAIEdge-VLab ensures efficient, fair, and hardware-aware scheduling across heterogeneous edge devices through three key principles:

- Resource sensitivity: The scheduler tracks cluster and device states in real time, considering availability, health, and progress. Only one benchmark runs per device to prevent interference and ensure reproducibility.
- Priority awareness: Jobs are prioritized by urgency and duration, with priority aging to prevent starvation and maintain fair access.
- Heterogeneity handling: Each device is managed by a host that abstracts hardware differences and reports capabilities, enabling consistent benchmarking across diverse platforms.

10.5 Scheduling Algorithms

The scheduler receives a job description from the Benchmark API containing key information such as the edge device family, runtime, and benchmark type. It then uses this data together with the current state of the edge nodes, including their capabilities, unique IDs, and dynamic attributes like job progress, to compute a scheduling plan. Once the plan is determined, the scheduler submits or removes jobs through Nomad using its deployment interface.

10.5.1 Baseline algorithm

The baseline scheduler uses a FIFO queue per edge device family within the dAIEdge-VLab *Benchmark API*, with Nomad as the backend executor.

Submitted benchmarks are queued by device family. The dispatcher selects the oldest compatible job when a suitable device becomes available, converts it into a Nomad job specification, and submits it for execution.

The scheduler subscribes to the Nomad event stream, a key component that provides real-time updates on job completions and node availability. Running in a separate thread from the dispatcher, this mechanism keeps the scheduler continuously aware of the cluster state without requiring redundant polling.

10.5.2 Explored Algorithms

The next sections will describe the different algorithms tested for the scheduler logic.

10.5.2.1 Algo 1 - FIFO with Type-Based Fixed Priority (No Job eviction)

This algorithm extends the baseline by introducing fixed priorities per benchmark type while preserving FIFO order within each category.

Priority classes (static; higher value = higher priority):

- Type 1 - synthetic model benchmarking: 80
- Type 2 - data-driven model benchmarking: 60
- Type 3 - on-device training benchmarking: 50

The dispatcher always selects the highest non-empty priority class and, within that class, the oldest job that matches an available edge device. There is no interruption or restart of running jobs; once a job starts, it runs to completion.

10.5.2.2 Algo 2 - FIFO with Type-Based Fixed Priority and Job eviction

This algorithm extends Algo 1 by introducing job eviction. When no capacity is available, a running job may be evicted only if a waiting task has higher priority within the same device family. The newest eligible lower-priority job is selected to minimize wasted computation, while jobs of equal priority are never evicted. Displaced jobs are requeued in FIFO order.

This approach improves responsiveness for short, high-priority tasks but may lead to starvation under sustained load due to the absence of priority aging.

10.5.2.3 Algo 3 FIFO with Type-Based Fixed Priority, Bounded eviction and Aging

This algorithm builds on Algo 2. When all devices are busy, a bounded eviction policy applies: a running job can be evicted only if a waiting job has higher priority and the running job's progress is below a set threshold (e.g., progress $<$ X%). Progress is reported via the *EdgeDevices* Nomad plugin. Among eligible candidates, the newest lower-priority job is selected to minimize wasted work.

To prevent starvation, the scheduler adds priority aging, allowing long-waiting or repeatedly evicted jobs to gradually gain priority up to a defined limit.

10.5.3 Algorithm evaluation

A heterogeneous cluster including three Raspberry Pi 5 devices was used to evaluate the scheduling algorithms. For simplicity, only the Raspberry Pi 5 devices were used as benchmark targets. Since users select the resource type, the results remain representative of overall algorithm performance.

A test bench ensured repeatable and fair evaluation by replaying a fixed sequence of job requests from multiple users and benchmark types under identical conditions. Four jobs were defined per benchmark type, each with a unique Experiment ID. The test bench tracked job states, generated execution timelines, and computed performance metrics.

Two evaluation scenarios were considered:

- Realistic workload: assessing waiting times, useful resource utilization, and user experience.

- High contention: evaluating fairness, waiting times, completion rate, and resource utilization with all devices fully loaded.

All metrics are defined in Section 1.4.4.

10.5.3.1 Base test-bench

The first test bench is summarized in Table 10.1. Three users submit different types of benchmark requests, making this setup representative of a typical dAIEdge-VLab use case. The parameter α, used to compute each job's waiting allowance, was set to 1 for all benchmark types.

10.5.3.2 Contention test-bench

The Table 10.2 describes a test bench designed to create contention within the cluster. Three users simultaneously submit identical sets of benchmarks, mixing different types in quantities exceeding the available resources.

Job waiting allowances were computed using scaling factors of $\alpha = 3$, 4, and 5 for TYPE1, TYPE2, and TYPE3 benchmarks, respectively. These values were tuned to produce meaningful results and prevent all jobs from exceeding their waiting allowance.

10.5.4 Algorithms performances

Each figure shows the evolution of job states over time, including four main statuses:

- Queued: waiting in the scheduler's internal queue until selected for submission to Nomad.
- Pending: accepted by Nomad but not yet placed on a suitable resource.

Table 10.1 Base test-bench definition

User	Time	Type	Edge device	Runtime	Experiment ID
1	T+0	TYPE3	Rpi5	tflite	3-2
2	T+2	TYPE3	Rpi5	tflite	3-3
3	T+2	TYPE3	Rpi5	tflite	3-3
1	T+1	TYPE1	Rpi5	tflite	1-1
2	T+8	TYPE1	Rpi5	tflite	1-2
3	T+8	TYPE1	Rpi5	onnx	1-0
1	T+76	TYPE1	Rpi5	tflite	1-1
2	T+83	TYPE1	Rpi5	tflite	1-2
3	T+83	TYPE1	Rpi5	onnx	1-0
1	T+12	TYPE2	Rpi5	onnx	2-0
2	T+90	TYPE2	Rpi5	onnx	2-0

Table 10.2 Contention test-bench definition

User	Time	Type	Edge device	Runtime	Experiment ID
1-2-3	T+0	TYPE1	Rpi5	tflite	1-1
1-2-3	T+0	TYPE1	Rpi5	tflite	1-1
1-2-3	T+0	TYPE2	Rpi5	onnx	2-0
1-2-3	T+0	TYPE3	Rpi5	tflite	3-3
1-2-3	T+15	TYPE1	Rpi5	tflite	1-1
1-2-3	T+15	TYPE1	Rpi5	tflite	1-1
1-2-3	T+15	TYPE2	Rpi5	onnx	2-0
1-2-3	T+15	TYPE3	Rpi5	tflite	3-3
1-2-3	T+30	TYPE1	Rpi5	tflite	1-1
1-2-3	T+30	TYPE1	Rpi5	tflite	1-1
1-2-3	T+30	TYPE2	Rpi5	onnx	2-0
1-2-3	T+30	TYPE3	Rpi5	tflite	3-3

- Running: actively executing on an assigned device.
- Evicted: interrupted to free a resource for a higher-priority job and rescheduled later.

Timestamps follow the test-bench configuration, showing when users initiated requests, while time zero marks their arrival at the API. Small offsets stem from upload time, latency, or polling. Figures therefore reflect actual API arrival times, showing how network and upload delays affect order.

10.5.4.1 Base algorithm

The Figure 10.4 illustrates the timeline generated by the baseline algorithm using the Base test-bench. The behaviour is straightforward: jobs are executed strictly in their order of arrival.

The median waiting time for TYPE1 benchmarks is 3 seconds, with a standard deviation of 14.33 seconds. This suggests that while most jobs start quickly, some experience significantly longer delays, indicating uneven scheduling among jobs of the same type and a potential impact on user experience. If TYPE3 benchmarks were hour-long experiments, shorter jobs could be delayed for an hour before execution.

The usefulness of performed computations and the clean success rate both reach 100%, as this algorithm does not support job restarts.

10.5.4.2 Algo 1

Figure 10.5 shows the timeline produced by the Algo 1, which prioritizes short-running jobs in the queue. Overall, it resembles the baseline algorithm's

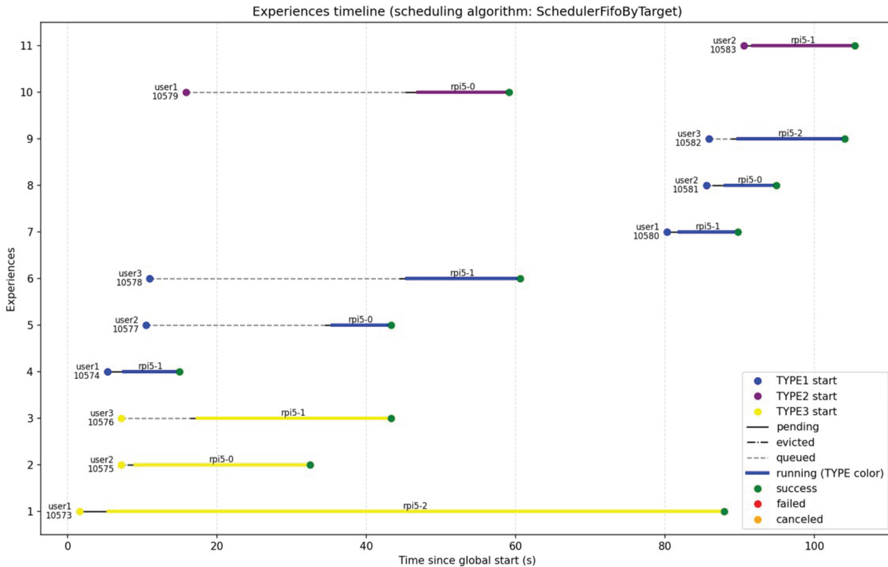

Figure 10.4 Base test-bench timeline base algorithm.

Table 10.3 Metrics base test-bench – Base Algo

Metrics	**Base**		
TYPE	**1**	**2**	**3**
Wait time mean [s]	11.39	15.93	5.06
Wait time median [s]	3.00	15.93	3.60
Wait time standard deviation [s]	14.33	21.17	4.37
Average stress ratio	1.05	1.28	0.16
Average service score	0.80	0.70	1
Compliance rate [%]	66.7	50.0	100
Jain (user service)	0.97	0.85	1
Clean success rate [%]	100	100	100
Resource usefulness [%]	100		

timeline, except that the three first short jobs are executed before the single long job submitted by User 3.

The median waiting time for TYPE1 benchmarks is 2.68 seconds, with a standard deviation of 6.45 seconds, indicating fairer waiting times compared to the baseline algorithm, as also reflected in the Jain's fairness index. However, if all resources were occupied with TYPE3 benchmarks, the standard deviation would increase significantly. This prioritization improves responsiveness but is still limited by the absence of an eviction mechanism.

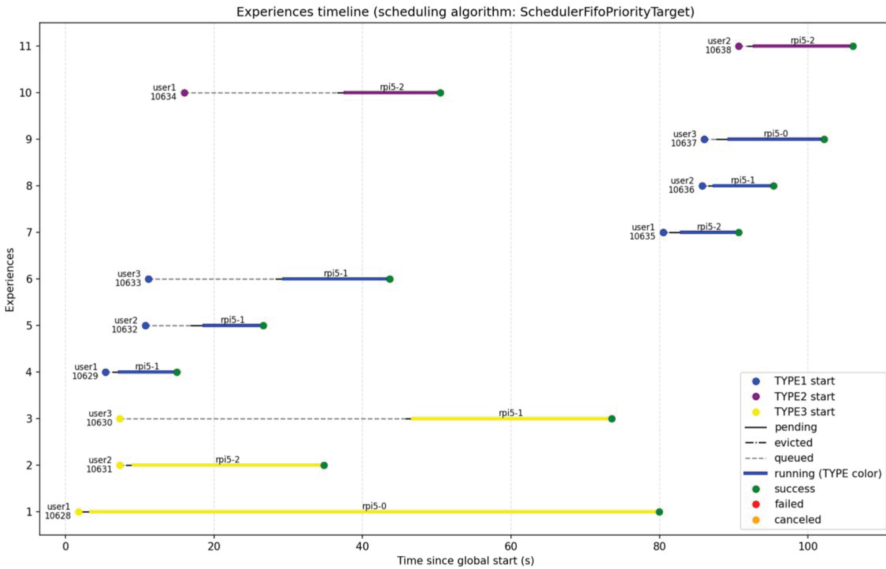

Figure 10.5 Base test-bench timeline Algo 1.

Table 10.4 Metrics base test-bench – Algo 1

Metrics	**Algo 1**		
TYPE	**1**	**2**	**3**
Wait time mean [s]	5.71	11.70	14.11
Wait time median [s]	2.68	11.70	1.64
Wait time standard deviation [s]	6.45	13.86	21.74
Average stress ratio	0.52	0.90	0.51
Average service score	0.98	0.80	0.87
Compliance rate [%]	83.3	50.0	66.7
Jain (user service)	1	0.94	0.97
Clean success rate [%]	100	100	100
Resource usefulness [%]	100		

10.5.4.3 Algo 2

This algorithm introduces job eviction capability as illustrated in Figure 10.6. Thus, the short running jobs are always placed as soon as they arrive.

In this case, the median waiting time for TYPE1 jobs is 2.12 seconds, with a standard deviation of 0.67 seconds, indicating consistent scheduling.

However, the clean success rate drops to 81.8%, as two long-running jobs were stopped and restarted. This also reduces the useful computation ratio to 77.7%, meaning 22.3% of the total computation was wasted.

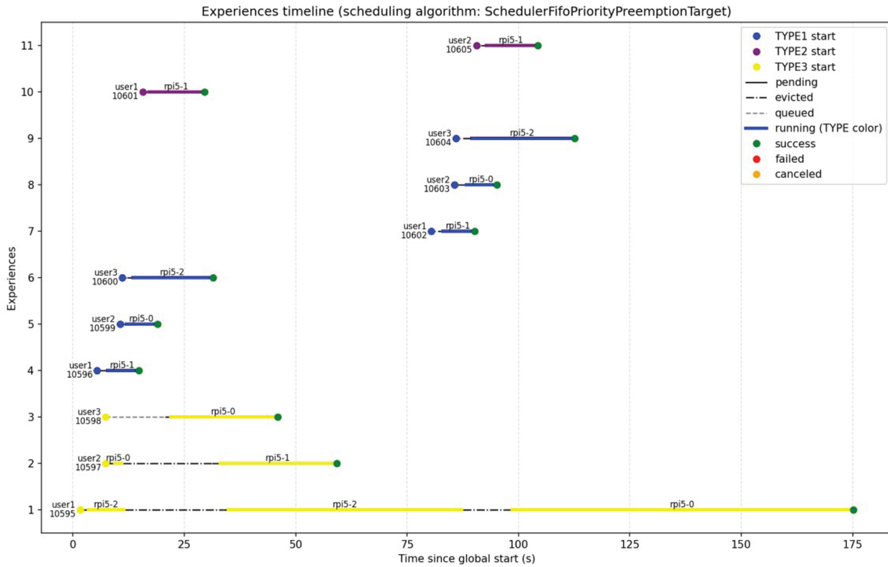

Figure 10.6 Base test-bench timeline Algo 2.

Table 10.5 Metrics base test-bench – Algo 2.

Metrics	Algo 2		
TYPE	**1**	**2**	**3**
Wait time mean [s]	2.14	1.45	45.40
Wait time median [s]	2.12	1.45	25.37
Wait time standard deviation [s]	0.67	0.52	44.70
Average stress ratio	0.22	0.12	0.93
Average service score	1	1	0.93
Compliance rate [%]	100.0	100.0	66.7
Jain (user service)	1	1	0.99
Clean success rate [%]	100	100	33.3
Resource usefulness [%]	77.7		

10.5.4.4 Algo 3

Finally, the last algorithm considers job progress before eviction, preventing near-completion jobs from being stopped and thus reducing computational waste. As shown in Figure 10.7, the first long-running job is not evicted by later short-running requests, with only a slight delay of a few seconds for one of them. Overall, the test bench completes noticeably faster than with Algo 2.

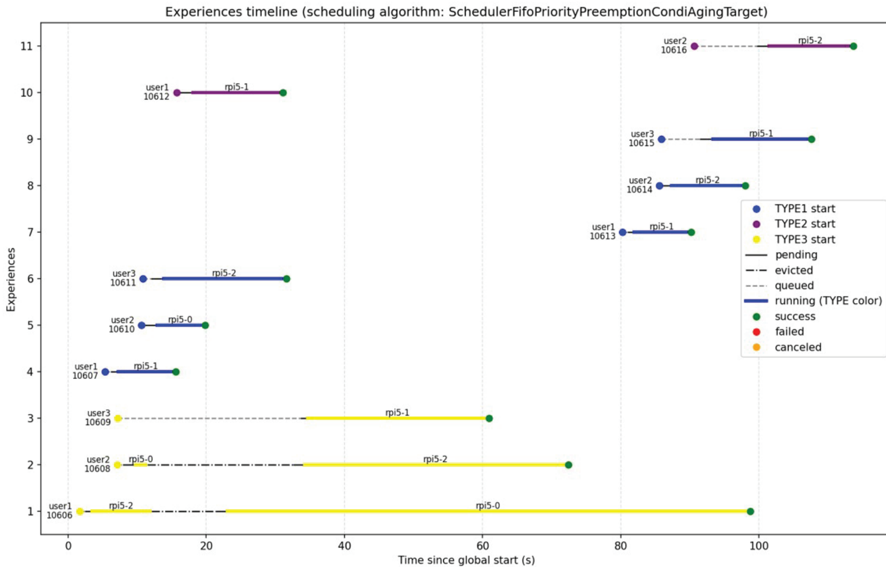

Figure 10.7 Base test-bench timeline Algo 3.

Table 10.6 Metrics base test-bench – Algo 3

Metrics	**Algo 3**		
TYPE	**1**	**2**	**3**
Wait time mean [s]	2.78	6.40	25.10
Wait time median [s]	1.82	6.40	26.85
Wait time standard deviation [s]	2.25	6.02	3.39
Average stress ratio	0.24	0.51	0.67
Average service score	1	1	0.99
Compliance rate [%]	100.0	100.0	66.7
Jain (user service)	1	1	1
Clean success rate [%]	100	100	33.3
Resource usefulness [%]	95.6		

For this final algorithm, the median waiting time for short-running jobs is 1.82 seconds, with a slightly higher standard deviation of 2.25 seconds.

While the clean success rate remains at 81.8%, the amount of useful computation increases significantly to 95.6%, meaning only 4.4% of the total computation was wasted due to the eviction mechanism.

Table 10.7 Metrics contention test-bench - 1

Metrics	**Base**			**Algo 1**		
TYPE	**1**	**2**	**3**	**1**	**2**	**3**
Wait time mean [s]	59.91	71.80	84.78	11.99	63.32	119.88
Wait time median [s]	57.84	74.97	87.81	12.48	63.16	120.37
Wait time standard deviation [s]	40.60	44.07	44.62	6.91	1.09	11.28
Average stress ratio	2.65	1.46	0.66	0.52	1.30	0.96
Average service score	0.50	0.69	0.98	0.99	0.78	0.98
Compliance rate [%]	27.8	33.3	66.7	94.4	0.0	66.7
Jain (user service)	0.99	1	1	1	0.99	1
Clean success rate [%]	100	100	100	100	100	100
Resource Utilization [%]	82.4			83.0		
Resource usefulness [%]	100			100		

Table 10.8 Metrics contention test-bench - 2

Metrics	**Algo 2**			**Algo 3**		
TYPE	**1**	**2**	**3**	**1**	**2**	**3**
Wait time mean [s]	11.32	62.24	119.10	40.86	67.94	96.3
Wait time median [s]	11.6	62.58	119.19	14.41	43.32	84.93
Wait time standard deviation [s]	7.03	1.36	10.83	47.06	39.15	28.62
Average stress ratio	0.54	1.29	0.95	1.76	1.40	0.77
Average service score	0.98	0.78	0.99	0.74	0.78	0.98
Compliance rate [%]	83.3	0.0	55.6	66.7	44.4	66.7
Jain (user service)	1	0.99	1	1	1	1
Clean success rate [%]	100	100	100	100	100	100
Resource Utilization [%]	81.2			83.7		
Resource usefulness [%]	100			100		

10.5.5 Algorithms performances under heavy starvation

The following tables present the results of all algorithms under the Contention test-bench, designed to evaluate their behaviour under starvation conditions. No figures are presented due to the large number of jobs.

Under contention, all algorithms maintain fairness across users, but their compliance rates differ. Algos 1 and 2 favour short benchmarks due to the absence of priority aging, causing longer TYPE2 and TYPE3 jobs to risk starvation over time.

The base algorithm avoids starvation by serving jobs strictly in arrival order, but this leads to higher waiting times and lower compliance for short jobs. Algo 3 provides the best balance by raising the priority of long-waiting jobs, improving fairness. Its median waiting time for TYPE1 jobs is about

14.4 s, with the highest compliance across all types, though with higher variance.

All schedulers achieve roughly 83 % utilization within the test window. Remaining inefficiencies arise from network latency and short synchronization delays, which could be mitigated by more frequent scheduler updates or event-driven triggers.

10.5.6 Results analysis

Algo 3 proves to be the most adaptable and effective across diverse conditions. Under normal circumstances, it enhances user experience by prioritizing short-running jobs while avoiding wasted computation through the non-eviction of jobs nearing completion. Its priority-aging mechanism prevents starvation of lower-priority tasks, and by selecting the oldest job among those with equal priority, it reduces uneven treatment across users.

Under severe resource scarcity, Algo 3 behaves similarly to a FIFO strategy, maintaining fairness but at the cost of increased waiting times for all users. This results in a uniformly degraded experience rather than one favouring specific users or workloads, demonstrating the algorithm's balanced approach to fairness and performance.

10.5.7 Limitations of the current implementation

The scheduling logic remains simple, focusing on priorities, device availability, and fairness through priority aging. This design ensures robustness and transparency but lacks user-quota mechanisms to balance long-term resource access and prevent unfairness under heavily unbalanced user requests.

10.6 Conclusion

This work introduced the dAIEdge-VLab, a distributed framework for fair, efficient, and reproducible benchmarking across heterogeneous edge devices. By combining Nomad's orchestration capabilities with a custom scheduler and Nomad plugin, it enables device-aware, priority-based execution while ensuring balanced resource allocation and consistent results.

Among the evaluated strategies, Algo 3 achieved the best balance between responsiveness, fairness, and resource utilization through its priority-aging mechanism and adaptive behaviour under varying load conditions. The resulting architecture is scalable, resilient, and flexible, uniting the robustness

of cloud orchestration with the adaptability required for decentralized edge environments.

10.6.1 Future Work

Future work on the dAIEdge-VLab will aim to enhance scheduling intelligence, fairness, and scalability. Planned developments include a predictive scheduler using historical data for runtime estimation, and support for federated learning requiring coordinated multi-device execution. Further research will address dynamic user quotas and broader hardware support to improve responsiveness and sustainability.

Acknowledgements

This work has been funded by the European Union's Horizon Europe project dAIEdge under grant agreement No. 101120726.

References

[1] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes,"Borg, Omega, and Kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 50–57, May 2016, doi: 10.1145/2890784.

[2] B. Hindman *et al.*, "Mesos: a platform for fine-grained resource sharing in the data center," in *NSDI'11: Proc. 8th USENIX Conf. Networked Systems Design and Implementation*, Boston, MA, USA, Mar. 2011, pp. 295–308, doi:10.5555/1972457.1972488.

[3] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 2017, doi: 10.1109/COMST.2017.2745201.

[5] KubeEdge, "kubeedge/kubeedge," *GitHub repository*. [Online]. Available: https://github.com/kubeedge/kubeedge. Accessed: Dec. 10, 2025.

[6] OpenYurt, "OpenYurt: extending your native Kubernetes to edge," *GitHub repository* openyurtio/openyurt. [Online]. Available: https://github.com/openyurtio/openyurt. Accessed: Dec. 10, 2025.

[7] HashiCorp, “Nomad: A Simple and Flexible Scheduler and Orchestrator for Cloud and Edge Workloads,” [Online]. Available: https://www.nomadproject.io Accessed: Oct. 17, 2025.

[8] X. Li, E. Zhu, J. Qin, W. Xu, and Y. Wu, “A Multi-Resource-Aware and Load-Balanced Scheduling Strategy for Heterogeneous Edge Clusters,” *J. Grid Comput.*, vol. 23, art. no. 32, 2025. Available: https://doi.org/10.1007/s10723-025-09817-2

[9] W. Peng, Z. Tang, J. Guo, J. Lou, T. Wang, and W. Jia,“LR^{2}Scheduler: layer-aware, resource-balanced, and request-adaptive container scheduling for edge computing,”*CCF Trans. Pervasive Comput. Interact.*, Online First, 2025.Available: https://doi.org/10.1007/s42486-025-00186-z

[10] Y. Cao, S. Hu, Z. Qu, L. Hao, and B. Ye, “Multi-cluster layer-sharing container scheduling in cloud-edge collaboration,” in *Advanced Intelligent Computing Technology and Applications*, D.-S. Huang, C. Zhang, Q. Zhang, and Y. Pan, Eds., *Lecture Notes in Computer Science*, vol. 15856, Singapore: Springer, 2025, pp. 16–27, doi: 10.1007/978-981-96-9914-8_2

[11] R. Jain, D.-M. Chiu, and W. Hawe, “A quantitative measure of fairness and discrimination for resource allocation in shared computer systems,” *arXiv preprint* arXiv:cs/9809099, 1998. Available: https://arxiv.org/abs/cs/9809099.

[12] B. G. S. Costa, J. Bachiega Jr., L. R. de Carvalho, and A. P. F. Araujo,“Orchestration in fog computing: A comprehensive survey,” *ACM Comput. Surv.*, vol. 55, no. 2, Art. 29, pp. 1–34, Jan. 2022, doi:10.1145/3486221.