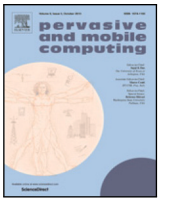
Pervasive and Mobile Computing 108 (2025) 102028

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc



A novel middleware for adaptive and efficient split computing for real-time object detection

Matteo Mendula [a,*], Paolo Bellavista [a], Marco Levorato [b], Sharon Ladron de Guevara Contreras [b]

[a] Department of Computer Science and Engineering, University of Bologna, Italy
[b] Donald Bren School of Information and Computer Sciences, University of California Irvine, United States

ARTICLE INFO

Keywords:
Edge computing
Split computing
Machine learning
Object detection
Image compression
Middleware
Energy efficiency

ABSTRACT

Real-world applications requiring real-time responsiveness frequently rely on energy-intensive and compute-heavy neural network algorithms. Strategies include deploying distributed and optimized Deep Neural Networks on mobile devices, which can lead to considerable energy consumption and degraded performance, or offloading larger models to edge servers, which requires low-latency wireless channels. Here we present Furcifer, a novel middleware that autonomously adjusts the computing strategy (i.e., local computing, edge computing, or split computing) based on context conditions. Utilizing container-based services and low-complexity predictors that generalize across environments, Furcifer supports supervised compression as a viable alternative to pure local or remote processing in real-time environments. An extensive set of experiments coversdiverse scenarios, including both stable and highly dynamic channel environments with unpredictable changes in connection quality and load. In moderate-varying scenarios, Furcifer demonstrates significant benefits: achieving a 2x reduction in energy consumption, a 30% higher mean Average Precision score compared to local computing, and a three-fold FPS increase over static offloading. In highly dynamic environments with unreliable connectivity and rapid increases in concurrent clients, Furcifer's predictive capabilities preserves up to 30% energy, achieving a 16% higher accuracy rate, and completing 80% more frame inferences compared to pure local computing and approaches without trend forecasting, respectively.

1. Introduction

The increasing adoption of Machine Learning (ML) solutions in a broad range of real-world application scenarios has highlighted the need for runtime system architectures with high flexibility and adaptability. However, the usability of ML algorithms in real-world settings is often hampered by computing and communication limitations that are often not considered at algorithm design and evaluation stages. Challenges include the constrained computing capabilities and energy budget of mobile devices, as well as limited and volatile communication channel capacity. With reference to Fig. 1, Local Computing (LC) and Edge Computing (EC) stand as the primary strategies for tackling the broad and heterogeneous tasks centered on the execution of complex data analysis and decision making algorithms. On the one hand, LC, that is, the execution of the ML algorithm onboard a mobile device, aims at the optimal interplay between software applications and hardware components to efficiently harness available onboard resources. This near-deterministic computing strategy comes at the price of reduced lifetime and limited computing capabilities. On the other

* Corresponding author.
E-mail addresses: matteo.mendula@unibo.it (M. Mendula), paolo.bellavista@unibo.it (P. Bellavista), levorato@uci.edu (M. Levorato), sladrond@uci.edu (S.L.d.G. Contreras).

https://doi.org/10.1016/j.pmcj.2025.102028
Received 4 August 2024; Received in revised form 13 January 2025; Accepted 13 February 2025
Available online 22 February 2025
1574-1192/© 2025 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

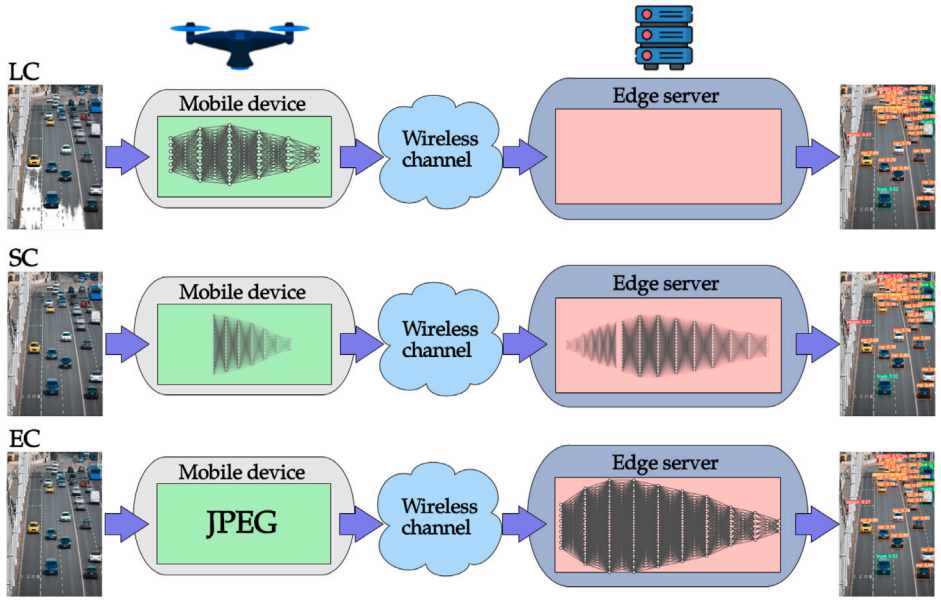


Fig. 1. Representation of computation distribution between Mobile Device (MD) and Edge Server (ES) for Local, Split, and Edge computing scenarios.

hand, EC, where the ML tasks are offloaded to a compute-capable device positioned at the network edge, leverages high-performance communication and computing technologies to effectively support real-time applications. While EC promises higher computational capabilities and larger energy budget for pervasive mobile systems, it tends to mandate a stable wireless link, which is typically not available consistently.

Despite the good progress made recently in all the above computing paradigms, none of them can be deemed as a universal solution suitable for every scenario. We contend that the volatility and time-varying nature of a multitude of relevant parameters and state variables (*e.g.*, channel quality or network/server load) make online adaptation not just desirable to achieve optimal performance, but also necessary in several practical deployments. However, adaptation of the computing configuration is explored only in academic and theory-driven investigations, which often overlook critical characteristics and issues emerging in real-world systems and deployments [1–3]. Moreover, despite its potential to become a critical computing configuration in many operational settings and system states, SC has yet to be evaluated beyond purely academic frameworks, making its ability to contend with EC and LC solutions unclear.

In response to these open technical challenges and gaps, in this paper we propose *Furcifer*, an innovative middleware specifically designed to provide seamless adaptation of the employed computing modality in realistic application environments. Furcifer transparently monitors the state of the underlying resources employed at runtime, by evaluating the current feasibility of EC, LC, and SC configurations and operating an automated switching between them when appropriate. At its core, Furcifer proposes a novel containerized approach that effectively support the dynamic transition between EC, LC and SC. Operatively, all three computing strategies are instantiated on startup time to be selected with no additional overhead according to QoS requirements. Experimental results demonstrated the capabilities of the middleware to reduce context switch latency between different computing modalities – less than 45 ± 2 ms – by using storage as an additional and inexpensive resource.

In particular, in this article we evaluate Furcifer's applicability to Object Detection (OD), a representative Computer Vision task performed collaboratively between edge devices and mobile systems. In addition, the paper describes our extensive assessment work that has involved a measurement campaign over more than 250 indoor and outdoor experiments, featuring wireless technologies, such as the IEEE 802.11n and 802.11ac Wi-Fi protocols. In our tests we evaluate our middleware capabilities on both moderate and highly dynamic scenarios. With a gradual fluctuation of connection quality and concurrent connected clients, Furcifer achieves an impressive **2x** reduction in energy consumption and an additional **30%** mean Average Precision score compared to LC, while also providing a remarkable **threefold** frame per second rate increase compared to EC. In highly dynamic scenarios, where channel quality and the number of concurrent connected clients rapidly change, a simplistic reactive approach would excessively change computing strategies, potentially reducing the actual number of frames per second (FPS) for object detection tasks. To prevent this, Furcifer employs recent data to forecast future trends and then proactively react to system changes. Our experiments in highly dynamic real-world environments demonstrate Furcifer's effectiveness, achieving up to 30% energy savings and a 16% increase in accuracy compared to static local computing. Furcifer also achieves an additional 80% successful frame detections compared to traditional reactive computing strategy selection methods. The primary novel contributions of this paper are as follows:

- An innovative middleware for distributed Object Detection tasks capable of switching between EC, SC and LC.

- A low-overhead container-enabled adaptation solution that allows model switching with minimal latency and negligible additional bandwidth usage.
- The first SC encoder–decoder model that competes with highly optimized state-of-the-art OD models in practical applications.
- A comprehensive dataset[1] containing over 250 indoor and outdoor experiments across diverse scenarios with mobile devices, various channel conditions, wireless technologies, and system load levels.
- A low-complexity policy management module, trained on the aforementioned datasets and experiments, which optimizes computing configurations based on target power consumption, OD performance, and overall latency.
- A time series prediction component capable of attenuating context adaptation frequency, thereby reducing unsustainable context switch costs.

2. Related work

Furcifer builds on prior advancements in multiple areas, including context adaptation in edge computing, image compression for object detection, optimization techniques such as quantization for LC, micro-service orchestration for mobile systems, and energy consumption on embedded systems (Sections 2.1–2.5).

2.1. Real-world context adaptation in edge computing

Dynamic offloading of computational load to edge servers is essential to support resource-intensive applications on constrained mobile devices while meeting demanding QoS requirements [4]. Despite substantial advances in self-adaptive offloading strategies, most state-of-the-art solutions have mainly tackled this optimization problem from a theoretical perspective [5,6]. Recent studies have investigated the benefits of context-aware adaptation for specific tasks such as 4k mobile Augmented Reality (AR) [7] and mobile video streaming [8] providing an in depth overview of the set of optimization operations required to effectively deploy self-adaptive policy managers in real-world field experiments.

Progressive adaptation to different OD contexts using synthetic data [9] or uncertainty-aware domain adaptation networks [10] proved to be a promising direction when implementing an ML-based solution in real-world scenarios. Furthermore, employing strategies such as the random exploration of optimal scaling factors [11] can help alleviate the negative effects of source domain bias.

However, these approaches often fail to consider critical system metrics such as energy consumption and network occupancy. In stark contrast with the current state of the art, Furcifer takes a comprehensive approach by tackling both system related and Computer Vision challenges, demonstrating the practical deployability of a low-complexity policy managers in constrained scenarios.

2.2. Image compression and object detection

Many neural models for Computer Vision, and OD in particular, are commonly trained and evaluated using state-of-the-art datasets such as COCO2017 [12] and Pascal VOC [13]. However, these evaluations often overlook the significant performance degradation caused by image compression [14,15], which is inevitable in practical EC systems. In particular, widespread image compression techniques are designed for human perception rather than for image analysis. As a consequence, high performance requires the transfer of large volumes of data over capacity-constrained channels. To address this issue, SC (also known as supervised compression in some contexts) [16,17] has recently emerged as a promising alternative to achieve state-of-the-art performance in computer vision tasks while effectively reducing bandwidth usage. The idea is to incorporate encoder/decoder-like structures within the ML models themselves, and use specialized training techniques to train task-oriented compressed representations [18,19]. Knowledge Distillation [20,21] is one of the tools used to maximize the effectiveness of SC frameworks.

Although the academic literature has demonstrated the potential of SC in popular datasets, an evaluation considering a complete computer vision pipeline is missing. Our work addresses this gap by illustrating how – often overlooked – components such as preprocessing, acquisition, and timing characteristics significantly influence the overall performance of a computing configuration.

Additionally, we emphasize the importance of proper model optimization, noting that SC is frequently compared to non-quantized models rarely used in practical deployments. Our work aims to evaluate the resilience of SC frameworks to quantization and to evaluate their performance compared to optimally designed models for embedded computers.

2.3. Quantization: Performance vs complexity trade-off

In the past, 32-bit single-precision floating-point format was prevalent as the standard numerical format for Deep Learning applications. However, in recent times, researchers and developers have proposed alternative formats whose goal is to reduce the computational load and time associated with the execution of a neural model while preserving as much task performance as possible. These alternatives explore more effective representations of tensors and parameters from the point of view of the complexity vs task performance point of view, while making the execution of models for difficult computer vision tasks possible on embedded devices. Compared with the canonical float32 (FP32) data representation, float16 (FP16) achieves a twofold reduction in bandwidth and an

[1] https://github.com/MatteoMendula/FurciferDataset

impressive eightfold decrease in computational throughput for math-intensive operations [22]. Incremental quantization to INT4 and INT3 achieves an astounding 128-fold speedup in computation and a 32-fold improvement in bandwidth usage. The specific extent of these improvements depends significantly on the system architecture. Notably, the benefits of integer quantization are even more pronounced on ARM CPUs [23].

Quantization can be approached in different ways, depending on the quantization mapping strategy and the level of eagerness or stationarity chosen in the process. Affine quantization [24] and scale quantization [25] are the two main techniques used in quantization and determine whether the quantization will be asymmetric or symmetric, respectively. Affine quantization involves scaling the original floating-point values by a scaling factor, while scale quantization ensures that the range of quantized values is symmetric around zero. Another crucial aspect of quantization is the calibration process, which can be either offline or online. Typically, offline calibration is performed to assess the quality of a quantized version of the model, but in real-world scenarios, this may not be feasible since the data distribution may not be known at deployment time. Thus, in practical applications, online calibration is commonly adopted, where the zero point and scaling factor are calculated on-the-fly for each processed image, allowing for dynamic adaptation based on the data encountered at runtime. This online calibration approach makes quantization more versatile and suitable for real-world applications, as it enables adaptive quantization in dynamic environments.

Indeed, the literature provides a wide range of solutions for quantizing smaller models, such as MobileNet, AlexNet, EfficientNet [26–28], and YOLO family models [29–31]. However, as mentioned earlier, a notable gap in current state of the art is the lack of evaluation of model quantization applied to SC. We remark how quantized models are a crucial and widespread component of real-world applications, and proving SC frameworks effective when quantized and competitive against LC quantized models would establish this class of algorithms and computing paradigm as another feasible operating point in the set available to mobile applications. In this paper, we demonstrate that quantized SC models can achieve inference latencies comparable to their LC counterparts even in mission-critical contexts where network quality cannot be considered a reliable resource.

2.4. Container orchestration

Service-oriented architecture (SOA) defines the essence of modern software applications. The demand for rapid scalability and streamlined maintenance operations has sculpted existing cloud solutions into modular and flexible structures [32]. In this context, microservices can be perceived as the logical evolution of software segregation principles, addressing the necessity for a lightweight approach to achieve application portability, which stands in contrast to the use of virtual machines (VMs). In fact, microservice containers proved to be a lower overhead solution compared with VMs in a broad set of computational tasks [33–36]. Hence, researchers and developers have evaluated the viability of container technologies for resource-constrained devices [37,38].

Contemporary container orchestration systems embrace hardware acceleration capabilities [39], enabling enhanced inference speed on GPU-enabled devices [40]. Among the array of vendor choices, NVIDIA Jetson devices strike a balance between monetary cost and critical factors like size, weight, and power (SWaP) [41]. For these reasons, the incorporation of GPU-enabled devices stands as a pivotal and debated frontier in the realm of Computer Vision tasks across Internet of Things (IoT) devices [42–44].

Furcifer takes a distinctive stance as the pioneering middleware that delves into the practicality of GPU-containers through a context-aware approach. This framework accentuates the heightened portability of containerized applications, emphasizing the capacity for seamless adaptation to diverse contexts. In scenarios where the ongoing application fails to meet Quality of Service (QoS) demands, the system can swiftly deploy a new container to the mobile device, facilitating a smooth transition to an alternative configuration with minimal latency. Furcifer capitalizes on the layered structure of container images to curtail the startup duration of containerized applications. By executing the pulling package operation during initialization, the framework we propose effectively diminishes the total container instantiation time by 76% [45].

2.5. Energy consumption on embedded systems

While much of the current ML-centered research is primarily focused on achieving the best task performance in the absence of resource restrictions, it is imperative to acknowledge energy consumption as a pivotal metric when considering mobile deployments. In recent years, initiatives such as The Low Power Image Recognition Challenge (LPIRC) [46] and a burgeoning energy-conscious perspective [47–49] have emerged, underscoring a deliberate shift towards evaluating energy consumption. This evolving approach aligns with a sustainable trajectory aimed at achieving Green AI [50], standing in stark contrast to the opposite trend of Red AI.

In the domain of real-time computer vision, energy consumption is not solely determined by the number of Floating Point Operations (FLOPs) or Multiply-Accumulate (MAC) operations indicative of the model's complexity. Indeed, energy consumption is also proportional to the number of frame per seconds (FPS) processed by the system [51–53] and the resolution of the input image [54,55]. Although in-depth studies address energy optimization from an embedded system perspective [56–58], current state of the art falls short of evaluating this aspect from a holistic cloud continuum [59] perspective.

Furcifer aims to strike a balance between resource efficiency and predictive precision, spanning from the edge to the cloud, and catering to the comprehensive energy optimization needs of modern mobile computing. By minimizing the energy consumption of mobile devices based on the desired mean Average Precision (mAP) score FPS rate, our solution represents a leap forward in realizing practical ubiquitous computer vision applications.

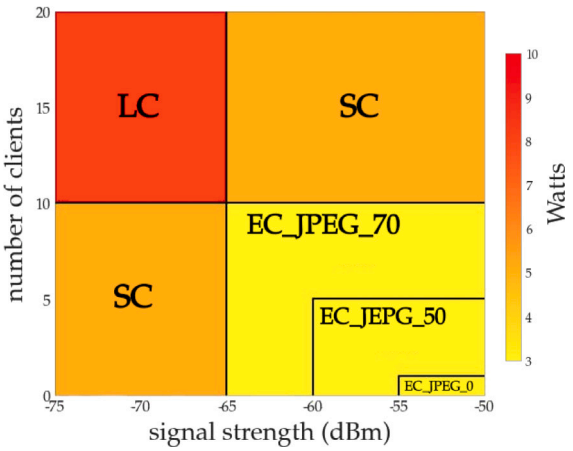


Fig. 2. Best performing computing modality and associated MD's power consumption as a function of signal strength and number of connected users.

3. Problem statement and preliminaries

First, we discuss the application setting we target in this work, where a cluster of mobile devices (MD) and an edge server (ES) collaboratively perform OD on the streams of images captured at MD's side. Here, we conceptually describe the computing strategies that can be used in terms of energy consumption, OD performance, frame rate, and bandwidth usage.

- **Edge Computing:** In this case, the entire model execution is performed on the edge server. This allows the usage of very precise and powerful models (*e.g.*, large non-quantized models). However, the limited computing capabilities of ESs compared to cloud servers means that the server may struggle to serve a large number of task streams. Moreover, the need to transfer the input data to ES means that robust and high-capacity wireless channels are needed. While image [60] and video [61] compression techniques significantly lower data transmission costs, they also diminish the model's ability to extract meaningful information from processed images once a certain compression threshold is exceeded [62].
- **Local Computing:** In settings where the task complexity is low enough to match the capabilities of the mobile device, then local execution of the model is a viable option. A trade-off is struck between task performance, power consumption and frame rate. Importantly, LC performance is not dependent on the state of the wireless channel connecting the MD to the ES, or the network and server load. In this context, quantization plays a crucial role in reducing both execution time and energy consumption, making it possible to deploy high-performance models that would otherwise be impractical on resource-constrained devices with limited computational power. However, this benefit comes with the trade-off of reduced model accuracy. In addition, while being completely resilient to signal strength fluctuation, LC implies high energy usage, which leads to a reduced battery lifespan.
- **Split Computing:** In SC, a subset of operations that would be executed on the ES is allocated to the mobile device. This subset often includes pre-processing operations, such as JPEG encoding and partial model inference altered to embed neural supervised encoding [63,64]. The objective is to decrease the amount of data to be transported over the wireless channel while minimizing the involvement of MD and possibly also decreasing the server load. This computing modality proves advantageous in settings where the communication channel's reliability is compromised, bandwidth demands exceed channel capacity or computing demands exceed server capacity. SC is specifically designed to address this scenario by mitigating both channel usage and computation burden on the ES. Unfortunately, existing SC solutions are not competitive with LC and EC, as the inference time on the mobile device often exceeds real-time requirements, or the size of the output from the first portion of the model exceeds that of the image itself, making SC less convenient compared to LC (faster) and EC (more bandwidth-efficient). In contrast, our proposed SC encoder achieves an inference time comparable to LC while delivering a superior compression ratio compared to EC, positioning SC as a practical and efficient intermediary between the two approaches.

Computing strategies comparison: The most popular metric used to evaluate OD is mean Average Precision (mAP), which combines precision and recall values based on Intersection over Union (IoU) scores across various levels of confidence thresholds. Typically, mAP scores are obtained by testing the algorithm on benchmark datasets such as COCO2017. However, when deploying an OD engine in a real-world setting, additional factors such as camera resolution and scaling factor alterations operated at preprocessing time come into play to determine the final performance at application level. With these in mind, we conduct an extensive evaluation of EC and LC, as well as Furcifer's SC engine. As MD, we select a Jetson Nano Dev Kit device, connected to a Jetson AGX Orin Dev Kit that acts as ES.

At the ES, we deploy a modified version of Faster R-CNN [65] with Res50 backbone as a feature extractor testing various JPEG compression rates: 0%, 50%, and 70%. We also explore high-frame-rate alternatives for LC. Specifically, we select a quantized FP16

Table 1
Mean Average Precision and FPS statistics depending on OD computing strategies. The best and second-best values are highlighted in green and light green, respectively.

Computing Strategy	mAP	FPS± std	FPS_{min}	FPS_{max}
EC JPEG_0	**37.051**	3.29 ± 1.264	1.2	5.52
EC JPEG_50	**31.797**	6.46 ± 3.007	1.89	10.01
EC JPEG_70	29.476	6.66 ± 3.058	2.01	10.32
SC FURCIFER (FP_{16})	25.964	**8.38 ± 1.992**	**4.96**	**11.72**
LC YOLOv5 (FP_{16})	23.403	**13.46 ± 0.261**	**11.89**	**13.89**

version of YOLOv5 [66]. In our study of SC, we propose a specialized encoder–decoder architecture trained through supervised compression, with Faster R-CNN serving as the teacher model. Building upon the framework introduced in [62], we enhanced the original model by quantizing the encoder to FP16 and crafting an optimized inference engine for improved efficiency. Furthermore, we fine-tuned the student model to align the camera resolution with the feature extractor's upscaling factor then reducing the size of the input tensor fed to the feature extraction portion of the encoder. Additional details regarding the custom distillation process are discussed in Section 4.4.

By looking at Fig. 2, in terms of power consumption LC (top left) is by far the most demanding computing strategy. Being the whole computation performed on the mobile device, this results in a peak of 10 Watts, drastically reducing the lifetime of MD. However, the complete autonomy of LC from the edge server makes it the only viable option when poor signal strength prevents any communication with the edge server or when the total number of concurrent clients does not allow the server to process incoming messages while guaranteeing QoS requirements. On the opposite end of the spectrum (bottom right), EC emerges as the most efficient strategy for preserving MD battery life. This is because only minimal, computationally negligible pre-processing operations are performed on the mobile device. However, this approach comes with drawbacks: the number of clients that can be simultaneously supported is limited, and it requires strong signal strength to transmit the captured images to the edge server effectively. As an intermediary solution, SC splits the computation between MDs and the edge server while reducing the amount of data transferred over the network. Given a smaller bandwidth usage compared with EC, SC emerges a viable option in those cases when: (i) the MD remains reachable, but the limited signal quality prevents the transmission of compressed images, or (ii) the edge server is operating at full capacity, and additional concurrent clients require further tasks to be processed.

The data reported Table 1 shows a comparison of the frame per second (FPS) and mAP obtained by each computing modality and model. The results indicate that the best mAP performance is obtained using EC without JPEG compression — that is, the largest model running without image compression. Conversely, the maximum frame rate is achieved by a quantized version of the original model deployed on the MD.

From the fourfold comparison of power consumption, FPS rate, mAP, and bandwidth usage, (i) EC stands out as the most favorable strategy under optimal signal conditions. It preserves MD battery life while achieving higher-quality object detections. (ii) SC is the preferred choice when the MD remains connected but the wireless channel cannot support even compressed image transmission. This approach reduces the computational load on the edge server and achieves a higher mAP compared to LC. Conversely, (iii) LC is the least desirable option, as it significantly reduces MD battery life and delivers the lowest object detection quality.

4. Furcifer: system design and implementation

The quantitative indicators presented in the previous section emphasize how there is no absolute winner among EC, SC, and LC even when considering a specific task, computing platforms, and communication technology. Instead, the – time varying – state of the system, which is influenced by mobility and load dynamics, determines the best computing strategies. This variability arises because the most efficient strategy, EC, heavily relies on the quality of wireless connectivity, which in turn depends on the mobility of the MD over the time. Furthermore, stochastic surges in concurrent client connections could compromise the feasibility of this approach, making SC a more suitable alternative when some connectivity is still available. In the worst-case scenario, where the MD is no longer reachable, the system defaults to LC as the only viable option. However, changing the computing modality in real-world deployments is technically non-trivial. Furcifer addresses this challenge by encapsulating each computing strategy within a container. This containerization approach packages the code along with its dependencies, ensuring that the ML application operates efficiently and reliably across varied computing environments. It also enables seamless transitions between ML models based on the specific characteristics of the context. The resulting highly efficient containerized models are then dynamically activated based on two key factors: (i) the current signal quality and (ii) the number of concurrent users connected to the edge server. Further details about the policy manager are provided in Section 4.5.

Within this section, we provide an in-depth discussion of the main features of each individual component while graphically representing the overall architecture and component units in Figs. 4 and 5.

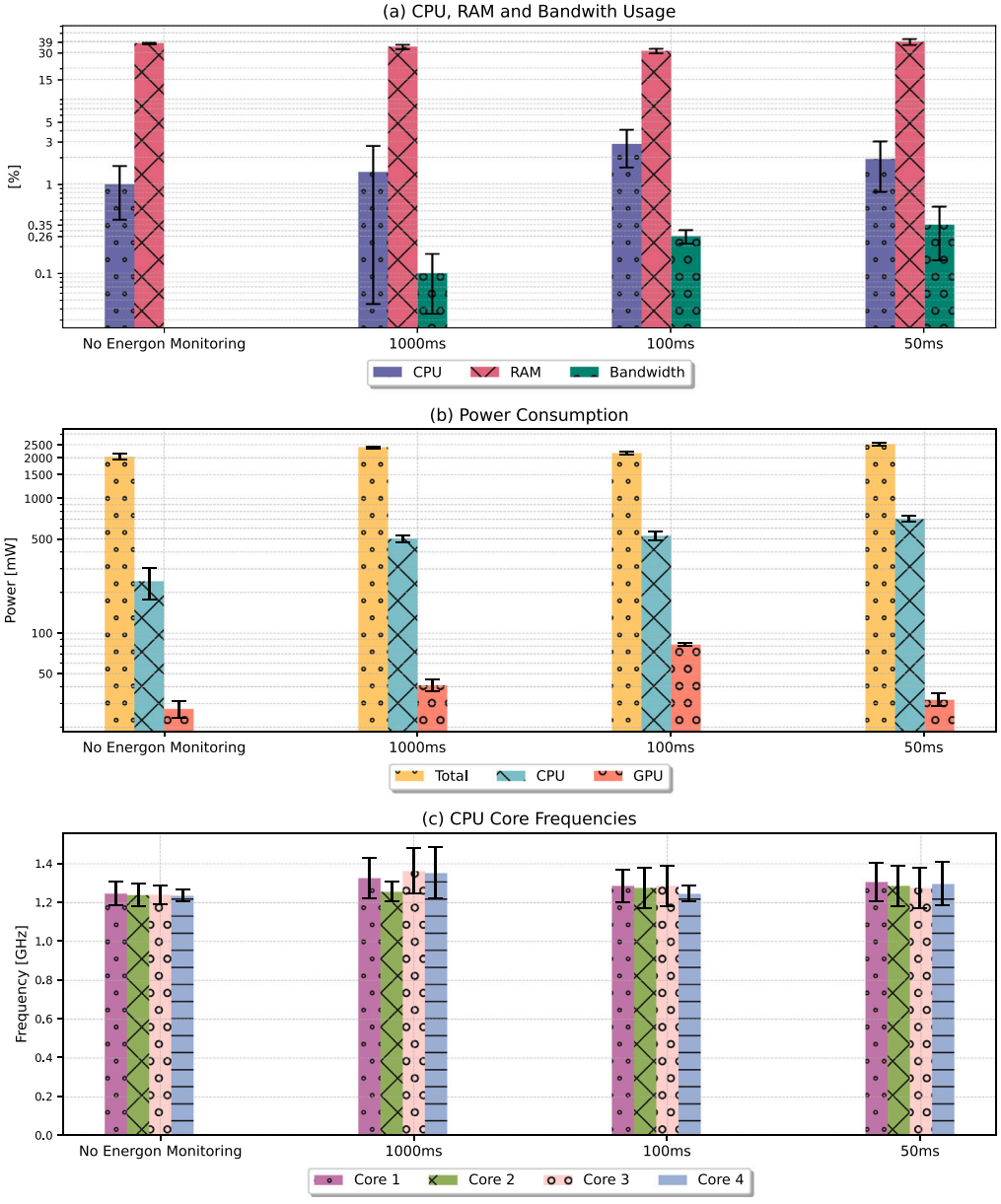


Fig. 3. Energon's overhead as a function of monitoring frequency. The results are presented in three parts: (a) the percentage utilization of CPU, RAM, and bandwidth; (b) a comparison of total power consumption between CPU and GPU; and (c) the individual core frequency dynamics.

4.1. Energon: a transparent energy monitoring module

We first describe the system monitoring module - a mandatory component as it informs Furcifer's decision-making process about the current state of the system. Energon is developed from scratch in order to be compliant with Prometeus [67], a well-established real time metric standard. Energon collects system metrics from connected *MDs* by unobtrusively extracting data from the registries of the targeted operating system with minimal overhead to the monitored device and without the need to implement modifications to the application's underlying logic.

Energon focuses primarily on energy consumption and resource utilization in *MDs*, while also providing insights into additional metrics, including network quality, packet transmission and drop rates, CPU usage for individual cores, storage utilization, GPU usage percentage, and temperature measurements from various regions of the board.

To evaluate the feasibility of our approach, we conducted a series of experiments to quantify the overhead introduced by Energon on mobile devices. Over a 120-second interval, we collected system metrics under four scenarios: (i) without Energon, (ii) with

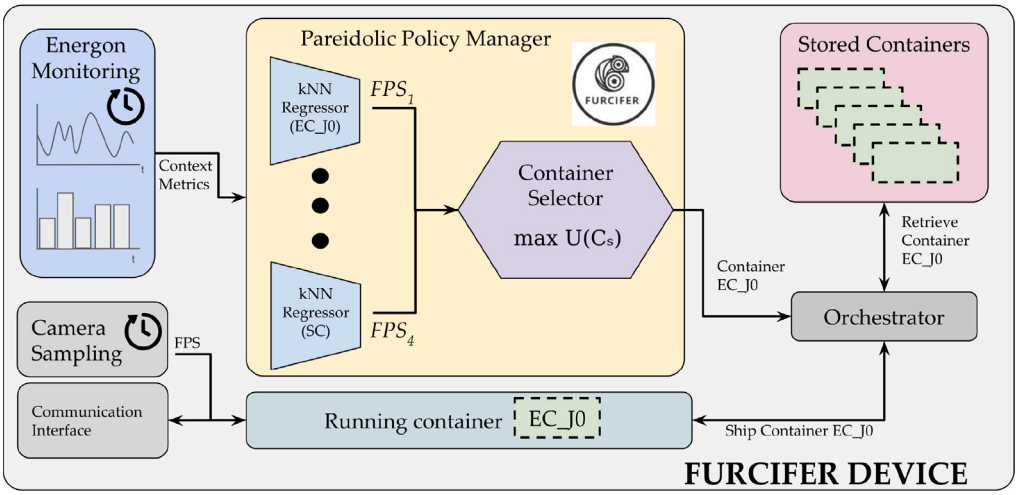


Fig. 4. Operational main components cooperation on Furcifer's *M D*.

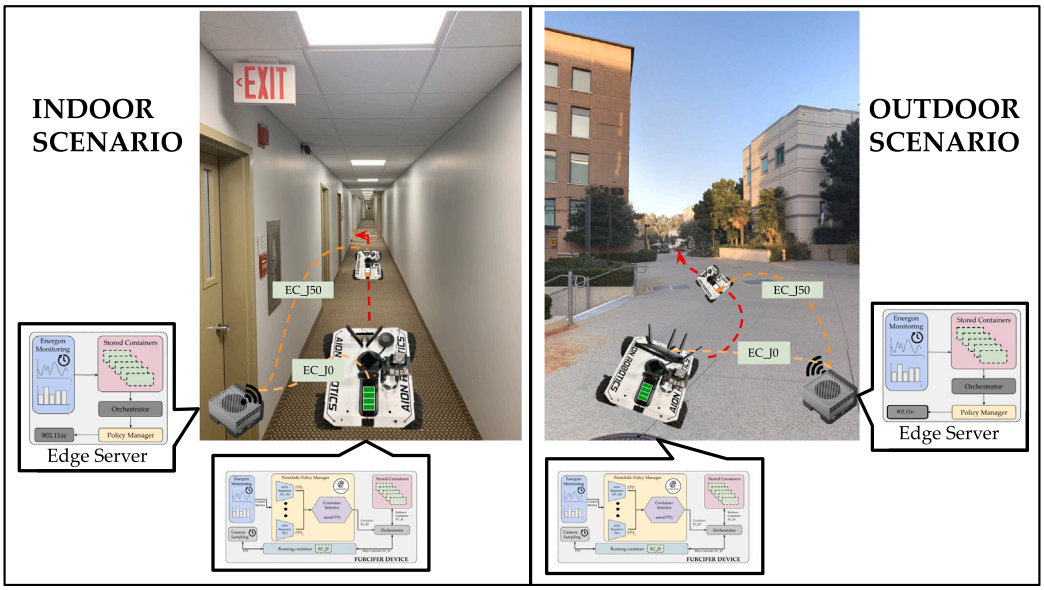


Fig. 5. Architecture and Dataset Collection Settings at Donald Bren Hall department and its adjacent outdoor space at the University of California, Irvine.

Energon at a monitoring frequency of 1000 ms, (iii) at 100 ms, and (iv) at 50 ms. For baseline system metrics without Energon, we utilized Tegrastats, a robust and widely adopted tool provided by NVIDIA for monitoring Jetpack[2] based devices. The monitoring frequencies for Energon were restricted to 1000, 100, 50 ms, as 50 ms represents the experimental maximum frequency supported by the Prometheus standard on the Jetson Nano. Fig. 3 illustrates Energon's overhead across various system components as a function of monitoring frequency. Specifically, Fig. 3a reveals that higher monitoring frequencies have a negligible impact on CPU and RAM usage. Bandwidth usage, as expected, increases slightly with higher frequencies but remains below 0.35%. It is important to note that the values on the y-axis are scaled logarithmically to better highlight smaller variations.

In terms of power consumption, Fig. 3b demonstrates that Energon does not introduce significant overhead in total, CPU, or GPU power metrics. Similarly, Fig. 3c highlights that the core frequencies of the mobile device remain comparable across all scenarios, with or without Energon, regardless of monitoring frequency. Based on our assessment, we determined that a monitoring frequency of 50 ms is optimal for our system, as Energon introduces negligible overhead across all tested frequencies. The collected metrics are subsequently made available via an HTTP endpoint, which can be queried on demand by the policy manager at desired rate.

[2] https://docs.nvidia.com/jetson/archives/r34.1/DeveloperGuide

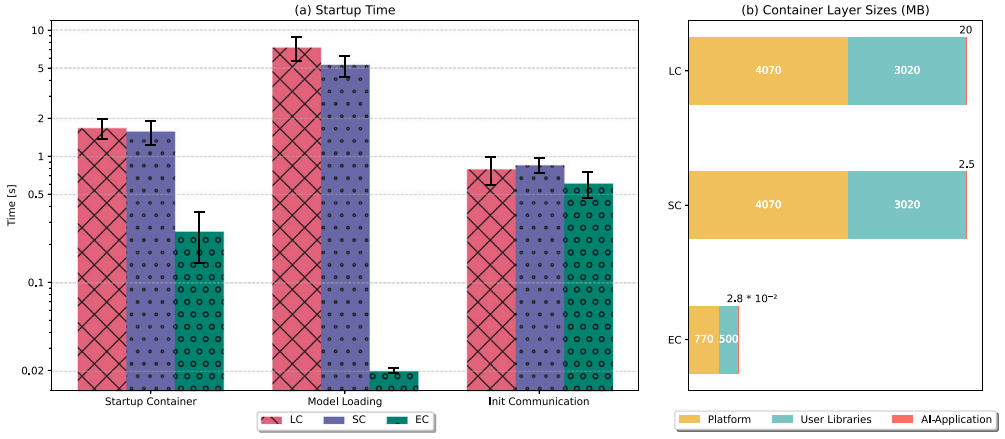


Fig. 6. Furcifer's GPU-enabled containers: (a) Startup time breakdown into container initialization, model loading, and communication instantiation for the *LC*, *SC*, and *EC* computing configurations; (b) Resource footprint comparison for the same configurations.

4.2. From the cloud the edge: On-demand image pulling

Our middleware integrates GPU-enabled capabilities offered by the original Docker runtime in a lightweight version of the renowned container framework where unused modules were removed. Furcifer includes a specialized container registry specifically designed for micro-service image logging and storage. This registry stores well-tailored images optimized for each compatible device, which are cached for future use based on the specific task the device is assigned. For each device type, a subset of images shares identical interfaces with the operating system hypervisor. In fact, these images only differ at the application layer and user library level, adapting to the specific task to be executed and the corresponding dependencies that are necessary.

We choose to apply containerization as a practical way to guarantee flexibility and fast reactiveness of the framework to future environment states. This shift of paradigm from the cloud to the edge empowers proactive mechanisms that enable seamless adjustments in response to evolving context requirements. Our evaluation of the size of resulting container images reveals that less than 1% of the image comprises application-level files. This efficient design enables the download of only the last layer of the image, which has the same footprint size of the model itself. This approach minimizes network usage, since the model would anyway need to be transferred no matter whether a containerized approach is used or not.

In Fig. 6a, the plot illustrates the startup time all available computing strategies across three key stages: Startup Container, Model Loading, and Init Communication. The results show that the time required to initialize the container is highest for *LC*, followed by *SC*, with *EC* demonstrating the lowest and most efficient startup time. Model loading is the longest stage for all strategies, with *LC* and *SC* taking considerably more time compared to *EC*, which achieves the fastest loading time. For Init Communication, all computing configurations exhibit similar times. Fig. 6b shows the memory footprint of Furcifer container images in the *LC*, *SC* and *EC* configurations. Notably, the memory requirements for hardware acceleration dependencies at both the platform and user library levels are about four times more extensive for *LC* and *SC* compared to *EC*. Remarkably, the contribution of the application layer remains minimal compared to the other components, reinforcing the viability of switching between containers on demand. This is achieved by pulling only the necessary components, specifically the last layer of the container image, thereby reducing unnecessary network usage and resource consumption.

While startup times are notable, this delay does not pose a significant issue during operation. In fact, all three containers are launched simultaneously during device startup, thanks to optimizations that ensure all modes fit within the mobile device without the need for unmounting at runtime. As a result, switching between computing strategies incurs virtually no overhead, with a minimal delay of just 45 ± 2 ms, which is solely attributed to the unavoidable GPU warm-up.

4.3. Communication interface and protocol

Deployed containerized models corresponds to a distinct and uniquely identified end point on the MD. Those microservices interact with the central orchestrator through a REST API, facilitating seamless communication and interaction while capitalizing on the advantages of minimal communication overhead.

The orchestrator continuously monitors potential new connections and, in the event of a connection loss, takes informed countermeasures to address the situation by switching to a local computing strategy. Furcifer's REST APIs operate on a request–response TCP-based model, enabling the framework to discern round-trip packet latencies. This functionality allows the introduction of well-defined rules for filtering out requests characterized by excessive communication delays. This mechanism ensures that the framework remains responsive and efficient, even in scenarios where the network conditions might fluctuate. Exchanged messages are defined as follows:

- **keep_alive**: This message is sent periodically using a polling mechanism to ascertain the presence of mobile devices within the same network.
- **start/stop_OD**: This message instructs the Mobile Device MD to initiate or terminate an Object Detection task. When initiating a task, the message also specifies the preferred computing strategy among LC, SC, and EC.
- **set_target_frame_rate**: This command sets the desired FPS rate for camera sampling based on dynamic requirements defined at the application level. Recognizing the direct correlation between higher FPS rates and increased power consumption, Furcifer intelligently conserves energy and network resources when higher frequency camera sampling is unnecessary, *e.g.*, Vehicle-to-Vehicle (V2V) cameras in low-traffic environments [68].
- **set_compression_rate**: If an EC configuration is used, the MD can opt to compress captured images before transmitting them to the ES for final detection. This message specifies the desired compression rate, controlling the balance between reduced compression for improved mAP score.

4.4. Furcifer's SC engine

In this section, we point out the main advancements made by Furcifer in the SotA, by comparing our crafted encoder against other solutions. Leveraging Faster R-CNN as the teacher model, we use a modified version of the knowledge distillation process adopted in SC2 Benchmark [62] to design a compact encoder optimized for constrained devices. This encoder serves a dual purpose: minimizing channel occupancy and effectively distributing computation load between mobile devices and the edge server. In stark contrast to [62], we replace the traditional matrix operations with cuDNN tensor operations [69], leveraging GPU parallel execution to its fullest potential. Additionally, we execute the model using a TensorRT engine [69], enabling high-performance inference optimization. For the fine-tuning process, we utilize the COCO2017 dataset to ensure comparability with other proposed solutions. As an additional adjustment to the original encoder proposed in [62], we adopted a different pre-processing technique. The original teacher model, Faster R-CNN, requires input images to be scaled to a minimum height of 800 pixels, which is significantly larger than the original 640 × 480 pixels resolution of the COCO2017 dataset. As a result, the enlarged images reach a resolution of 1066 × 800 pixels, which exceeds the onboard computational capabilities of the mobile device, preventing real-time object detection within constraints. In [62], the same enlargement at the preprocessing stage is preserved to maximize the mean Average Precision (mAP) score, despite the increased inference time. To prioritize real-time performance, we adopted a smaller image upscaling transformation, limiting the height to 400 pixels.

The optimized encoder heavily relies on quantization and channel compression to reduce execution time and transmitted data size as much as possible. To enhance data compression, we strategically place a one-channel bottleneck in the initial layers of the feature extraction segment of the network. This choice leads to further data reduction, increasing the efficiency of the whole process. Additionally, we incorporate INT8 quantization at the end of the encoder. This quantization approach optimizes the representation of the data, contributing to both improved data compression and streamlined computation. The scaling factor and zero point are calculated on a per-image basis as they are processed and then communicated to the decoder located at the ES, along with the resulting INT8 tensor from the encoder inference process.

Capitalizing on the online INT8 quantization of the latent space between the encoder and the decoder our encoder is novelly quantized from FP32 (as in [62]) to FP16. The negligible impact on the mAP score of Furcifer's SC engine on the COCO2017 dataset has to be attributed to the already strong approximation performed on the latent space to INT8. As a result, the transition from FP32 to FP16 yields a minimal difference, further highlighting the convenience of the quantization process. Thus, the adoption of an FP16 quantized encoder on the mobile device delivers a nearly 20% increase in processing speed compared to its FP32 counterpart, while preserving about the same mAP score. This finding underscores the additional advantage of applying quantization to SC encoders, which, unlike their LC counterparts, are already quantizing the final encoding result to minimize channel occupancy. As a result, they are less susceptible to mAP score reduction due to quantization. We remark how this optimization makes SC a competitive option against optimized models for embedded devices.

In addition to the comparison with the original encoder proposed in [62], we extend the comparison to CutEdge [70], a well-known solution for model partitioning, and Ladon [71], a recent advancement of [62] for multitask image encoding. It is important to note that in CutEdge, not only is the detection model different, but no distillation techniques are used to reduce the size of the latent space. Instead, the authors dynamically partitioned YOLOv4-tiny based on context metrics such as edge server workload and mobile computing capabilities. Unsurprisingly, all comparison metrics highlight Furcifer's SC encoder as the best option, as its accuracy and resulting FPS rate outperform CutEdge. Nonetheless, we selected CutEdge for benchmarking to emphasize the crucial role of knowledge distillation in making SC comparable to the performance of LC and EC. We also included Ladon in our benchmark, the multitask evolution of SC2 proposed in [62]. In this case, the same teacher model is used in the distillation process, and both ResNet50 and ResNet269 backbones are tested to ensure a fair evaluation.

In Table 2, we present a comparative evaluation of various split models and architectures based on three key metrics: latent space size, inference time, and mean Average Precision (mAP) at thresholds 0.5:0.05:0.95. The table includes results for CutEdge partition models at different route layers (37, 85, and 199) alongside more advanced architectures such as SC2 ResNet50, Ladon ResNet50, Ladon ResNet269, and two versions of Furcifer (FP32 and FP16). As the input image, we selected the standard resolution of 640 × 480 pixels to ensure compatibility with the evaluation conducted on the COCO2017 dataset. This choice results in a original input image size of 3.69 MB once converted from image to tensor. As a comparison, its JPEG encoding with no compression would result in a file of 0.1 MB. The results reported in Table 2 highlight the trade-offs between output size and inference time

Table 2
Performance Comparison between SotA encoders in terms of (i) latent space dimension, (ii) inference time, and (iii) mAP score-with. The best and second-best values are highlighted in green and light green, respectively.

Split model	Output size [MB]	Inference [ms]	mAP (0.5:0.05:0.95)
Cut Edge partition at			
Route layer 37	37.5000	202.35 ± 23.94	22.660
Route layer 85	9.3750	256.52 ± 34.18	22.660
Route layer 199	4.6875	366.73 ± 19.52	22.660
SC2 ResNet50	0.4458	599.63 ± 1.70	24.157
Ladon (β=0.32) ResNet50	**0.1683**	**92.00 ± 0.72**	25.243
Ladon (β=0.32) ResNet269	**0.1449**	118.82 ± 4.27	**27.784**
Furcifer FP32	**0.0194**	**104.95 ± 1.94**	**25.966**
Furcifer FP16	**0.0194**	**83.52 ± 1.54**	**25.964**

while maintaining competitive mAP. Specifically, the Furcifer FP16 model achieves the fastest inference time of 83.52 ms with an output size of 0.0194 MB, the smallest among all evaluated models. Similarly, the Ladon ResNet50 and Ladon ResNet269 models also demonstrate competitive performance, with output sizes of 0.1683 MB and 0.1449 MB, and inference times of 92.00 ms and 118.82 ms, respectively. The Ladon ResNet269 achieves the highest mAP of 27.784, showcasing its superior accuracy. In contrast, CutEdge, as an example of model partitioning, exhibits significantly larger output sizes and slower inference times, ranging from 202.35 ms at Route layer 37 to 366.73 ms at Route layer 199, while maintaining a consistent mAP of 22.660. This underscores the inefficiency of earlier split points in balancing output size and computational performance. Overall, the Furcifer FP16 and Ladon ResNet269 models emerge as the best-performing options, offering an optimal balance between inference speed, model compression, and accuracy.

When comparing Furcifer FP16 with Ladon ResNet269, we prioritize Furcifer FP16 due to its shorter inference time, making it more suitable for real-time applications. As a consequence, in our exploration of LC, SC, and EC, we selected our custom FP16 encoder as the SC option. This choice achieves the highest data compression with only a minimal reduction in mAP.

4.5. Pareidolia: Low-complexity similarity-based context adaptation

Pareidolia, a concept rooted in human perception, reflects the inclination to perceive distinct, often meaningful shapes or images within random or ambiguous visual patterns. It manifests as a natural cognitive process, wherein the brain attempts to link novel ideas with existing concepts. Leveraging already solved tasks, Furcifer leverages "pareidolia" as a context adaptation approach. The Pareidolic Policy Manager (PPM), as defined, performs low-complexity trend forecasting [72–74] based on the current system state.

Specifically, the policy manager takes as input the increasing number of concurrent clients and the variability of network conditions to forecast the expected number of FPS that the mobile device will achieve when employing each considered computing configuration. By considering potential failures when signal strength is minimal or the number of concurrent clients exceeds the edge server's capabilities, this predictive functionality allows PPM to anticipate the impact of each strategy choice on FPS and adjust the decision accordingly. PPM forecasting capability comes from a set of predictors, one for each computing strategy, which based on the current metrics collected by Energon (as described in Section 4.1) estimate the overall FPS rate the framework will achieve as a consequence of choosing a specific computing configuration.

In practice, when the target FPS rate is low and both networking conditions and the number of concurrent clients allow for the use of EC, this is always the preferred choice in terms of extending mobile device battery life and maintaining mAP detection quality. When the number of concurrent tasks remains stable but signal quality degrades, JPEG compression applied as a pre-processing operation by the mobile device reduces bandwidth usage, enabling EC to remain the preferred choice. In cases where signal strength is severely limited and/or the number of clients connected to the same edge server is excessive, SC becomes preferable to EC because it offloads part of the computation from the server, while reducing bandwidth usage. Finally, under extreme conditions where the edge server is unreachable due to wireless channel limitations or excessive computational load, LC must be selected, even though this choice reduces the mobile device's battery life and result in lower detection quality.

By gradually offloading computation from the edge server when adverse conditions, such as poor signal quality or high load on the edge server, arise, PPM periodically selects the best computing strategy. This is achieved by predicting the FPS for each strategy and then maximizing a simple "utility" [75] U metric for each computing strategy C_s.

$$U(C_s) = \begin{cases} 0 & \text{if } C_s \text{ does } \textbf{not} \text{ reach the target } FPS \text{ rate} \\ \frac{Acc(C_s)}{E(C_s)} & \text{if } C_s \text{ reaches the target } FPS \text{ rate} \end{cases} \quad (1)$$

where $Acc(C_s)$ and $E(C_s)$ are the mAP accuracy score and the energy consumption of C_s, respectively.

From this, it derives:

$$U(EC) > U(SC) > U(LC) \quad (2)$$

The problem as formulated becomes straightforward to solve by introducing our version of SC as an intermediary solution to preserve both detection quality and mobile device battery life.

5. Experimental evaluation

To validate and evaluate the Furcifer performance under a wide set of differentiated deployment environments and situations, we have performed an extensive set of experiments, by considering both outdoor and indoor execution environments covered with IEEE 802.11n and 802.11ac connectivity. The main objective was to assess Furcifer ability to dynamically adapt its adaptive split computing decisions for the cloud continuum against system state dynamics. We have collected an original dataset consisting of over 250 distinct combinations of channel conditions (expressed as signal strength) and the number of concurrent client connections by moving the mobile device along the corridor of our laboratory and in the open space in front of Donald Bren Hall at the University of California, Irvine. Although indoor and outdoor experiments exhibit similar trends, indoor scenarios tend to be characterized by a higher degree of unpredictability, which is primarily due to the presence of obstacles that complicate signal propagation. As a result, the overall channel quality is adversely affected, leading to more variable and less consistent performance outcomes.

5.1. IEEE 802.11n experiments

First, we focus on the widely used Wi-Fi 801.11n standard. Our experimental setup features a Jetson Nano DevKit as the mobile device equipped with a 640 × 480 pixels USB webcam. In indoor scenarios, we orchestrated the movement of the device along a designated path spanning approximately 20 m. In outdoor scenarios, the path extends over a distance of 50 m. This deliberate variation allowed us to replicate a spectrum of signal strengths and network dynamics, capturing the intricacies of both indoor environments and outdoor settings. For each experimental run, we meticulously examine the system scalability across different user scenarios. Specifically, we investigate the performance of the system, as the number of clients varies between 1 and 20.

Figs. 7a and 7b depict the FPS and error percentage metrics for SC and EC with JPEG compression gain 0, 50 and 70% as a function of distance with a single connected user. Within the indoor experiment settings (Fig. 7a), there is a striking similarity in the average FPS achieved by EC and SC, except in the scenario without image compression. Conversely, in the outdoor environment (Fig. 7b), SC takes advantage of the improved channel conditions compared to the indoor setting and achieves up to 2 additional FPS. Importantly, it is worth noting that SC not only excels in FPS but also achieves a higher mAP score compared to LC, showcasing its suitability in terms of both task performance and frame processing speed.

Figs. 7c and 7d show performance metrics as the number of clients connected to ES varies. Several notable trends emerge from this analysis. First, we observe an interesting pattern regarding the impact of image compression techniques as the number of concurrent clients increases. The failure rates associated with JPEG are dramatically larger compared to SC, reaching up to 100% failure rate in some measurements. This result underscores the perils of relying solely on image compression when dealing with a larger number of clients, highlighting the potential limitations of this approach in dynamic and demanding network/server conditions. On the contrary, SC achieves almost a steady 0% failure rate due to the small amount of network and server resources used by this configuration. The resulting improved resilience of Furcifer underlines the effectiveness of its context-aware approach, which enables SC to consistently outperform EC in the range of tested network conditions.

5.2. IEEE 802.11ac experiments

The data rate provided by a wireless communication channel depends on the technology, thereby influencing the trade-off between computing and wireless communication inherent in the SC and EC conditions. We extend the evaluation of Furcifer performance to include IEEE 802.11ac. In these experiments, we replicated the same path for the MD, while concurrently running increasing parallel connections of up to thirty clients. We use a Wi-Fi network interface that supports the IEEE 802.11ac 5 GHz protocol and establish a connection between the ES and MD over an 80 MHz band. Our Wi-Fi 5 GHz antennas also support MU-MIMO technology which additionally improves the overall communication performance.

Fig. 8 shows FPS rate as a function of the number of concurrent clients, ranging from 10 to 30. All EC compression strategies benefit from improved connection capabilities, outperforming SC when the number of clients is smaller than 10. Instead, when the number of clients increases to 20 and 30, the additional load on ES penalizes EC over SC of about 40% in terms of the average FPS successfully processed over time. As shown in the analysis that follows, the superior performance compared to ES when the system is under pressure is due both to the decrease in channel usage and the effort of the server granted by SC. On the contrary, when the full capacity of the network and server are available, SC is penalized by the computing effort allocated to MD.

To better appreciate the impact of communications on the computing mode, we conduct a comparative analysis between the IEEE 802.11n and 802.11ac Wi-Fi protocols. The results shown in Fig. 9 depict the FPS achieved as the number of connected clients increases. We note how the improved data rates offered by the IEEE 802.11ac means that EC is the winning solution up to a certain load level, whereas in IEEE 802.11n experiments, the superior compression granted by SC results in the latter being the winning solution. However, this comparison is made with the same signal strength variability for EC and SC, moving the MD along the same spatial trajectory.

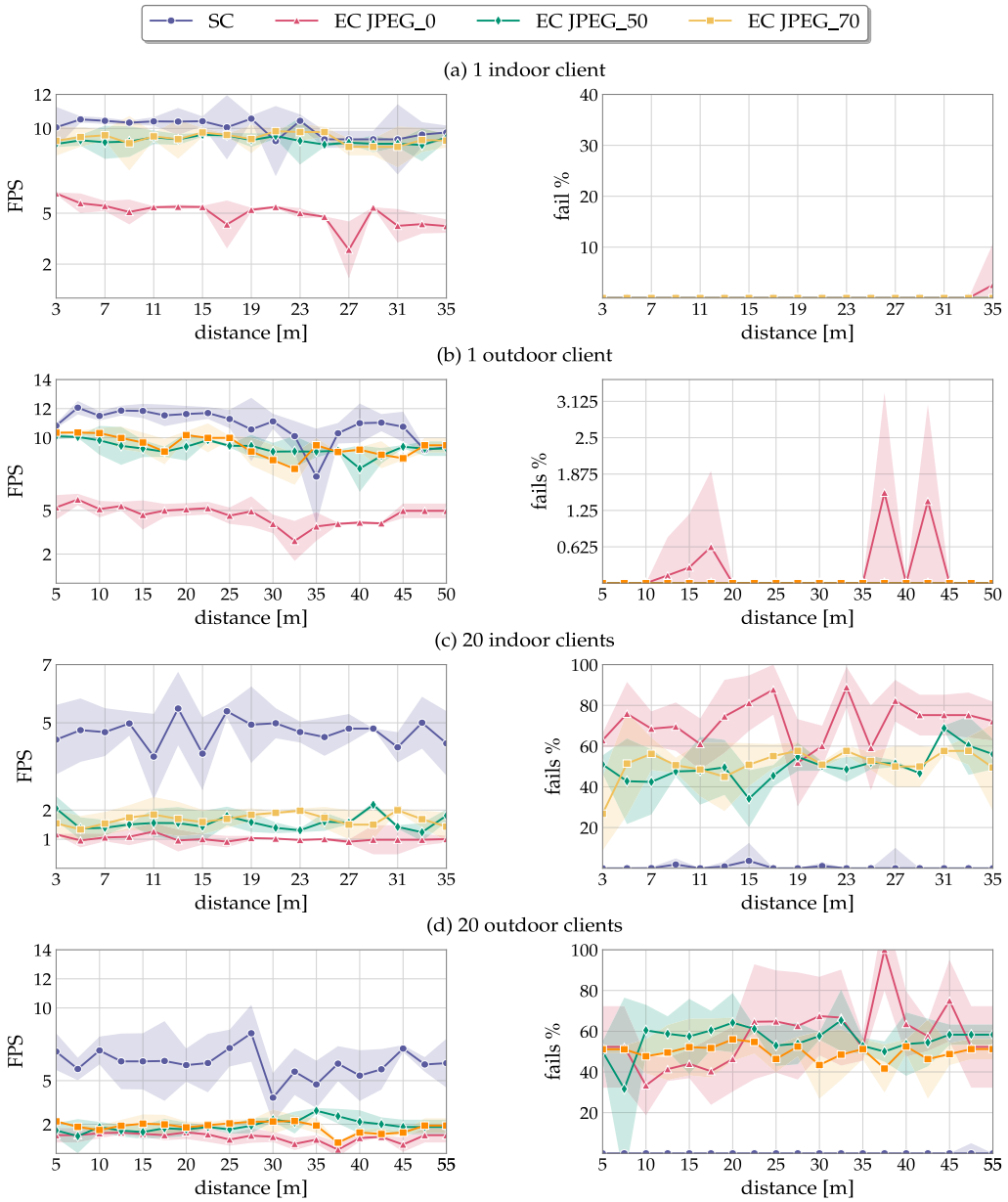


Fig. 7. IEEE 802.11n experiments: with one connected client (a) indoors and (b) outdoors, and twenty connected clients (c) indoors and (d) outdoors. Performance metrics – FPS, fail percentage – as a function of distance for SC and EC with 0%, 50%, and 70% JPEG compression gains. The lines in the plots correspond to average metrics, while the shading represents variance.

5.3. Edge server workload evaluation

In this section, we focus on the resource usage and the power consumption measured on the ES side as a function of load. ESs are predominantly modeled as devices connected to the power grid, and thus their power consumption can be considered as a performance metric. In this set of experiments, we limited our assessment to EC and SC, as in LC the ES is not involved in the computing pipeline. We select EC_JPEG_50 for EC as it obtains higher FPS rate compared with EC without compression and better mAP score compared with EC_JPEG_70. Fig. 10 shows the distribution of computational effort among RAM, CPU and GPU of EC_JPEG_50 (a) and Furcifer's SC engine (b). While the percentage of RAM and CPU utilization does not increases as higher number of clients is connected to the ES, EC shows a steady increase in terms of GPU utilization. On the other hand, SC GPU utilization shows a slower degree of growth and a larger variance. Specifically, SC experiences about 25% less GPU utilization on average, accompanied by a 3x increase in variance.

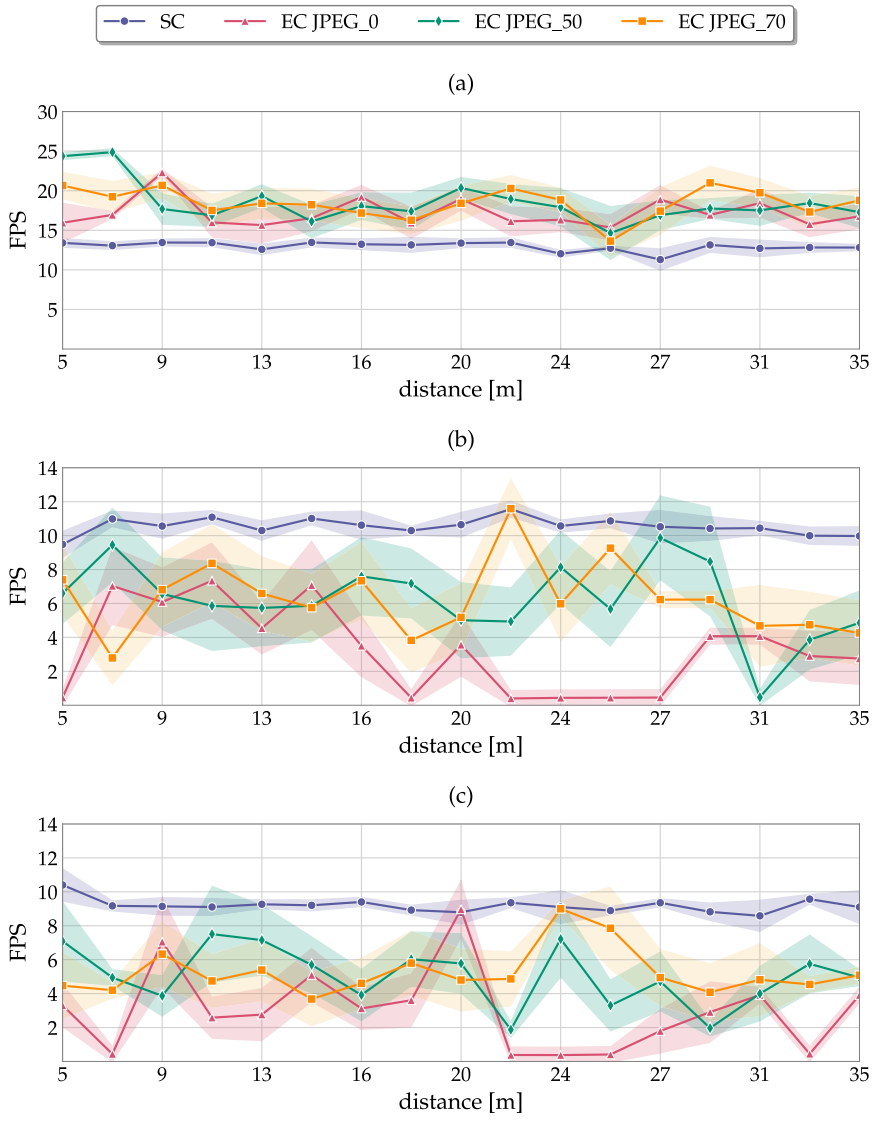


Fig. 8. IEEE 802.11ac FPS rate metric: ten (a), twenty (b) and thirty (c) connected clients — as a function of distance for SC and EC with JPEG compression gain 0, 50 and 70%. The lines in the plots correspond to average metrics, while the shade is the variance.

In Fig. 11 we report the corresponding power consumption of the ES broken down across the same components. As expected, the reduced power consumption of Furcifer's SC in comparison to EC_JPEG_50 reflects the lower resource utilization of the former compared to the latter. This results in approximately 20% less power consumption on the edge server side for SC, albeit with a 2x increase in variance.

5.4. Moderately dynamic scenario: PPM reactive adaptation

In the previous sections, we demonstrated the need for the dynamic adaptation of the computing configuration. We now evaluate the ability of Furcifer – and specifically its policy management module – to provide adaptation capabilities without imposing a significant overhead. In terms of energy consumption and optimal task performance, computing configurations can be clearly ranked based on the MD perspective. In fact, EC does not impose any computing load to the MD and achieves the best mAP thanks to the use of larger models. SC allocates minimal computing effort to the MD and has the second best mAP. Finally, LC results in the largest energy intake and worst performance. Thus, the decision engine has to evaluate the ability of the individual computing configurations to achieve the desired FPS rate given the currently perceived system state, and then select them in the order dictated by energy and mAP.

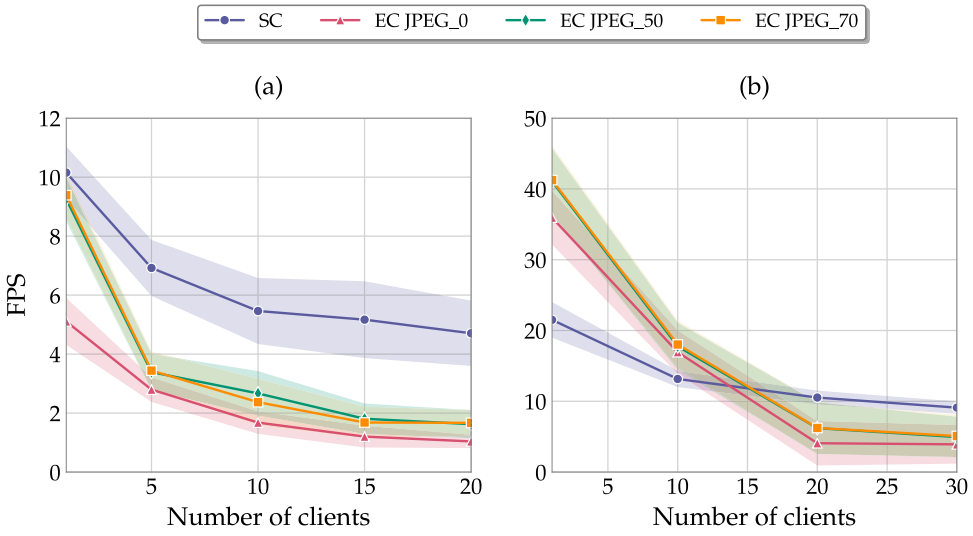


Fig. 9. Comparison between the 802.11n (a) and 802.11ac (b) Wi-Fi protocols, depicting the FPS_fails score with an increasing number of concurrent clients.

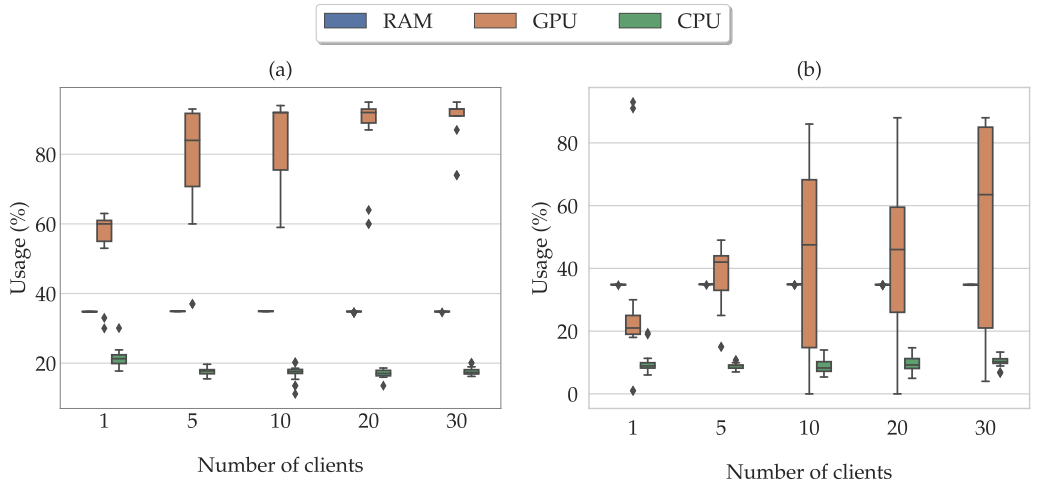


Fig. 10. Comparison between EC (a) and SC (b) resource utilization on the Edge Server with an increasing number of concurrent clients.

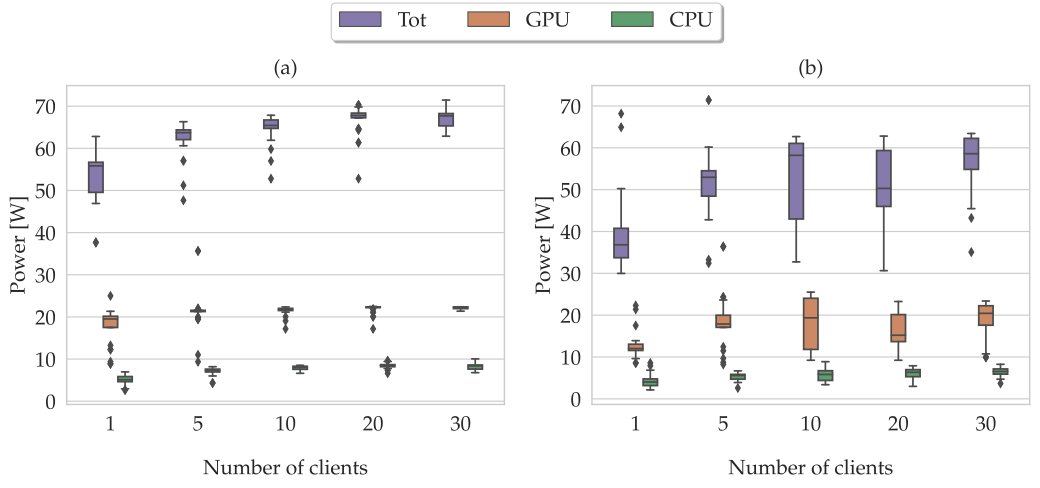


Fig. 11. Comparison between EC (a) and SC (b) power consumption on the Edge Server with an increasing number of concurrent clients.

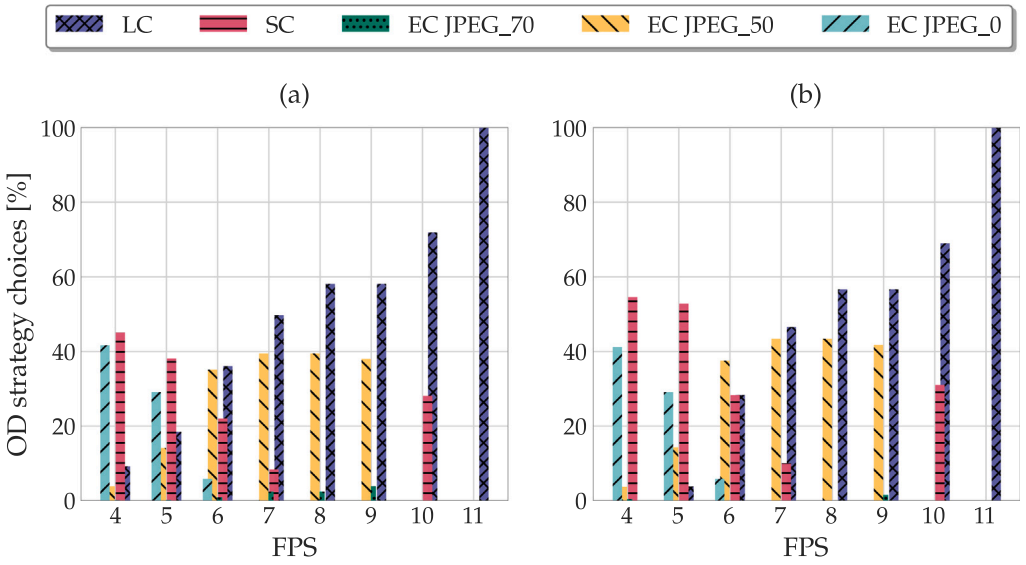


Fig. 12. Choices distribution of considered computing configurations depending on target FPS rate for the Ground truth oracle (a) and our Pareidolic Policy manager (b).

To support such decision process, we then build simple set of KNN regressors that take as input application context metrics such as: the inference time on the ES, the average round trip time, the communication channel quality, and the current resource usage. This produces as output the predicted FPS rate for each considered computing configuration, allowing PPM to anticipate the resulting Quality of Service (QoS) of a particular action before executing it.

We evaluate the regressors when training and applying them in specific environments (*e.g.*, indoor or outdoor), as well as on their ability to generalize. The resulting loss metrics are reported in Table 3. It can be observed that when these regressors are trained in the same context where they are applied, the error is minimal (below 11% $MAPE$ – Mean Absolute Percentage Error) across all computing configurations. In the case of transfer learning, where models trained indoors are deployed outdoors, the maximum loss value $MAPE$ increases to 20%. We then structure the decision making process as a simple set of nested conditions based on simple reactive predictors. Algorithm 1 shows the pseudo code embedded in Furcifer's decision engine.

Algorithm 1: Low complexity pareidolic algorithm

Data: $T \in [1, 30]$
Data: $N_{LC}, N_{SC}, N_{EC0}, N_{EC50}, N_{EC70}$
while *True* **do**
  if $pred_{E0}(context) \geq T$ **then**
    $N_{EC0} \leftarrow N_{EC0} + 1$;
  end
  if $pred_{E50}(context) \geq T$ **then**
    $N_{EC50} \leftarrow N_{EC50} + 1$;
  end
  if $pred_{E70}(context) \geq T$ **then**
    $N_{EC70} \leftarrow N_{EC70} + 1$;
  end
  if $pred_{SC}(context) \geq T$ **then**
    $N_{SC} \leftarrow N_{SC} + 1$;
  end
  $N_{LC} \leftarrow N_{LC} + 1$;
end

Adaptation Results: We compare Furcifer performance with a static LC solution, which represents the only viable option when the connection quality or system load cannot support the desired FPS rate. Fig. 12 shows the distribution of decisions made by Furcifer policy manager across the spectrum of available cloud continuum strategies. It is important to note that our policy manager, despite its low complexity, demonstrates the ability to match the configuration that an oracle controller would implement. Table 4 reports the Root Mean Squared Error (RMSE) between the percentage of choices made by Furcifer PPM low complexity policy manager and a baseline DRL agent. The striking alignment between the decisions made by the Furcifer pareidolic policy manager and the

Table 3
Low complexity frame rate predictors.

	Indoor		Outdoor		Indoor → outdoor	
	RMSE	MAPE	RMSE	MAPE	RMSE	MAPE
EC_J0	0.27	7.30	0.37	10.69	0.58	20.37
EC_J50	0.50	7.86	0.59	9.25	0.71	15.37
EC_J70	0.47	7.48	0.53	7.68	0.81	15.98
SC	0.70	7.46	0.97	9.48	1.23	15.63

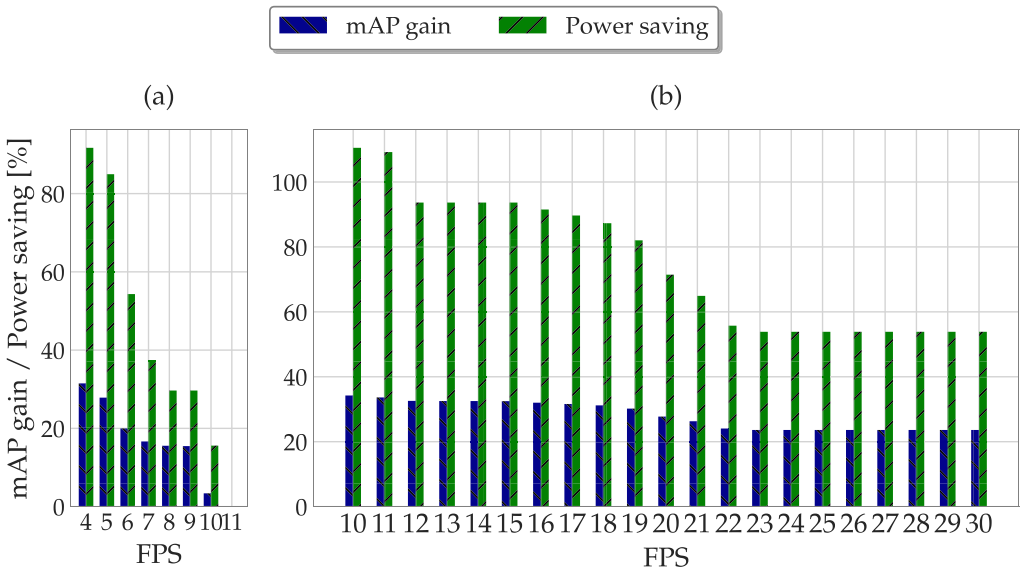


Fig. 13. *mAP* gain and Energy saving with IEEE (a) 802.11n and (b) 802.11ac Wi-Fi protocols.

Table 4
RMSE between percentage of OD computing configurations.

Target FPS	PPM	DRL
4	7.848	87.210
5	2.287	104.979
6	21.875	123.547
7	32.552	101.824
8	36.892	75.419
9	7.465	78.375
10	24.045	69.928
11	58.593	71.310
Average RMSE	23.944	89.074

ground truth highlights the feasibility of deploying a low complexity predictor deployed at MD, where more complex controllers may fail to train properly or adequately generalize.

In the IEEE 802.11n configuration, Furcifer reduces the energy intake by approximately 80% while achieving an average mAP score increase of over 20% in comparison to LC. The relevance of these outcomes is further amplified when using the IEEE 802.11ac protocol. In this scenario, energy savings exceed 100%, and mAP consistently maintains a level above 20% for all defined targets FPS. As Fig. 13 shows, Furcifer can easily generalize, accurately predicting the frame per second even when trained on an indoor environment and then tested outdoor.

5.5. Highly dynamic scenario: PPM proactive adaptation

To demonstrate the robustness of PPM against real-world challenges such as unreliable channel connections and sudden fluctuations in the number of concurrent clients, we simulated a scenario with unstable communication channels and high-frequency

Table 5
Performance metrics at different prediction window duration.

Window duration	Accuracy	F1 score	MFLOPS
500 ms	0.9247	0.9244	4.968
800 ms	0.9164	0.9186	7.9488
1 s	0.9050	0.9072	9.936
2 s	0.9038	0.9050	19.872
3 s	0.9072	0.9094	29.808
5 s	0.9157	0.9169	49.68
10 s	0.9182	0.9192	99.36
15 s	0.9280	0.9296	149.04
20 s	0.9061	0.9106	198.72
25 s	0.9015	0.9028	248.4

changes in user numbers. In particular, the MD was moved to a different distance from the ES every 25 s. Each new position was chosen to maximize the difference between connection qualities, while also accommodating a sudden increase/decrease in the number of users. Given that switching from one computing strategy to another takes 42 $\pm$ 14 ms, reacting instantaneously without trend awareness would result in more than 30% of frames being lost. In real-world scenarios caching [76,77] cannot be considered a viable solution. For this reason we extended the capabilities of PPM by adding a smart and low complexity time series forecasting module. The model architecture consists of 2 layers of LSTM (Long Short Time Memory) [78] with 50 neurons each. The sequence to sequence predictive models takes as an input the two most relevant metrics for best class identification which are the channel quality and the number of concurrent connected clients. We explored various sequence lengths to assess the balance between longer forecasting capabilities and the requirement to swiftly adapt to unpredictable context changes. In fact, on one hand a longer forecasting horizon allows for fewer changes in computing strategy, while on the other hand, a shorter window duration provides greater flexibility for the system to adjust to unexpected state changes. Additionally, it is important to note that to ensure the desired frame rate per second (FPS), only the most conservative computing strategy from each predicted sequence can be considered. More accurate offloading strategies such as SC or EC would typically necessitate additional computation and networking resources that may not be readily available during the considered time frame. To validate this, if a computing strategy is chosen for the upcoming interval Δt but cannot be supported by the edge server due to connection issues or resource unavailability (already allocated to other users), the target FPS rate cannot be guaranteed. For our experiment, we set a minimum target of 5 FPS as the QoS requirement to be maintained regardless of external influencing factors. We chose 5 FPS because, as illustrated in Fig. 12, it presents the most critical and heterogeneous distribution for the optimal computing strategy.

Fig. 14 illustrates the aggregated comparison between the best computing strategies and the predicted ones. Each strategy is ranked in descending order of priority: EC_0 is the most accurate and least energy-consuming for MD, thus it has the highest priority (rank 4), followed by EC_50 (rank 3) and EC_70 (rank 2). Finally, SC (rank 1) and LC (rank 0) complete the ranking where static LC is the only one who does not depend on edge server support and thus is not affected by fluctuating channel quality and concurrent user's connections. Clearly, the objective is to offload as much computation as possible to the edge server by utilizing EC or SC, depending on the opportunistic availability of channel and computational resources. As it can be observed, for each considered prediction window. the PPM module takes always the most conservative computing strategy choice even when the prediction does not perfectly match the ground truth. It should be noted that as the prediction window duration increases, the percentage of LC as the most conservative computing strategy also increases. This is due to the increased likelihood of encountering contexts where the channel is either unavailable or saturated with multiple user connections.

Table 5 reports the resulting accuracy and F1 score for the sequence to sequence classification task after 150 epochs. For each predicted window duration the corresponding amount of MFLOPS (Mega Floating Point Operations per Second) is also reported. As can be observed, even a very small network architecture can easily map the dynamic trend once a conservative approach is adopted.

After evaluating the PPM's ability to forecast sub-optimal computing strategies, we assessed the resulting benefits in terms of reduced power consumption, improved accuracy, and minimized loss in FPS (frames per second) rate. Fig. 15 shows the energy saving and the average accuracy gain in terms of mAP score of PPM against static LC as a function of different sequence lengths. Here, the percentage of preserved FPS against instantaneous reacting policy is illustrated, too. As illustrated, a longer prediction window duration results in smaller percentages of energy savings and accuracy gains. Conversely, the percentage of additional processed frames increases due to fewer switches in computational strategies.

6. Conclusions and directions of future work

In this paper, we presented the Furcifer [79] innovative middleware for the dynamic seamless adjustments of adaptive split computing decisions on the cloud continuum, in particular targeted to real-time object detection in pervasive and mobile execution environments. By leveraging container-based and low-complexity predictors, Furcifer offers an extremely low-overhead solution that proves effective across various deployment environments, system states, and wireless settings. In addition to edge computing (EC) and local computing (LC), we developed a highly optimized Split Computing model that performs supervised compression, establishing it as an effective third adaptive split computing configuration. Our experiments cover diverse scenarios, including both stable and highly dynamic conditions with gradual and unpredictable changes in connection quality and concurrent clients. Under

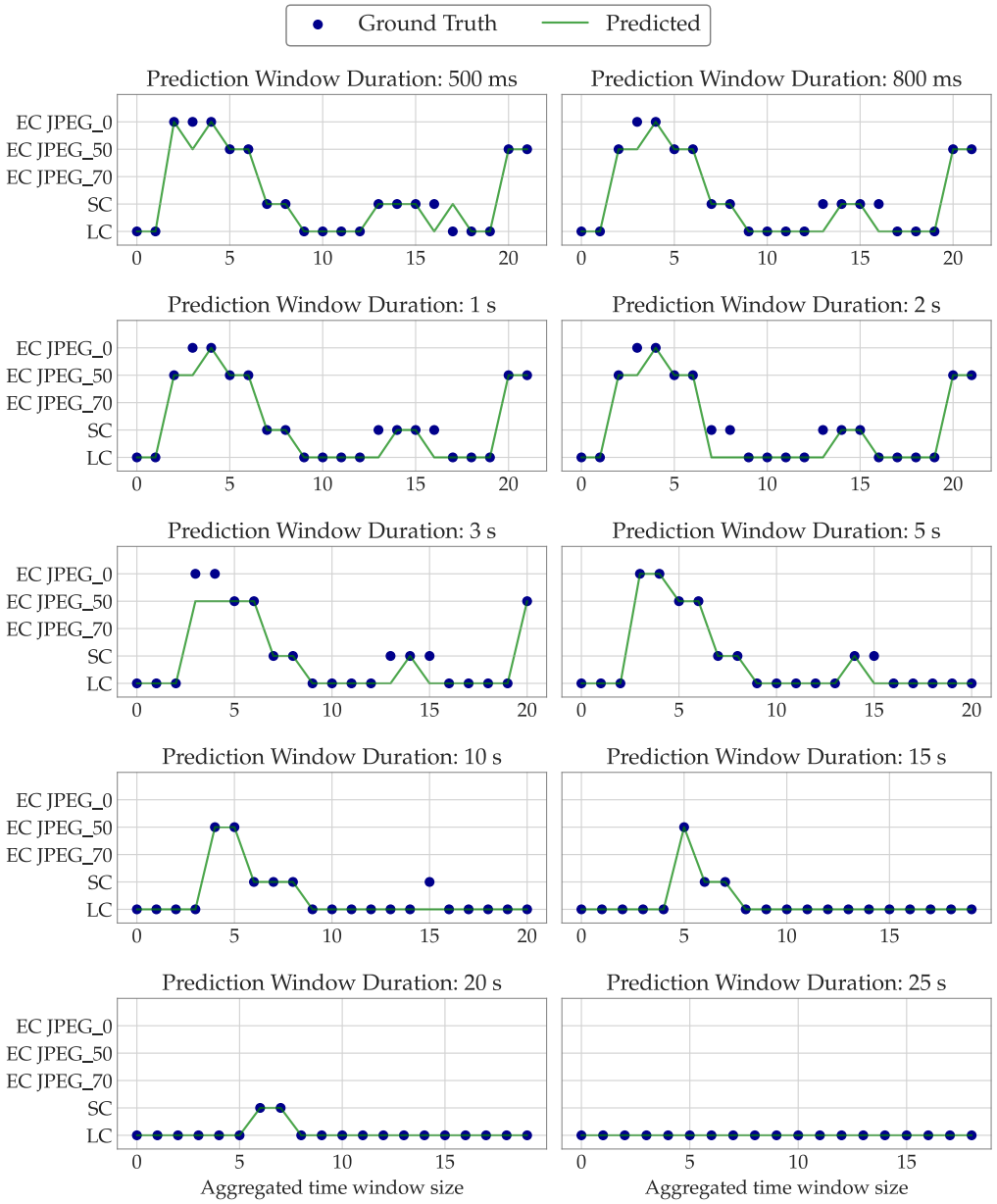


Fig. 14. Best computing strategy ground truth vs predicted for each forecasting window size in {500 ms, 800 ms, 1 s, 2 s, 3 s, 5 s, 10 s, 15 s, 20 s, 25 s}.

stable conditions, Furcifer demonstrates significant benefits: achieving a **2x** reduction in energy consumption, a **30%** higher mean Average Precision (mAP) score compared to local computing, and a **three-fold** FPS increase over static offloading. In highly dynamic environments with unreliable connectivity and rapid increases in concurrent clients, Furcifer's policy management preserves up to **30%** energy, achieving a **16%** higher accuracy rate, and completing **80%** more frame inferences compared to both local computing and approaches without trend forecasting, respectively. On the edge server side, the overall load is reduced by up to **25%** compared to traditional edge computing, while power consumption is approximately **20%** lower. Looking ahead, we plan to extend Furcifer's applicability to other computationally demanding tasks, such as Semantic Segmentation and LiDAR-based 3D mapping. Additionally, we aim to release an open-source version of Furcifer and explore the benefits of our policy-based adaptation across a wider range of mobile devices, further validating its versatility and effectiveness in real-world scenarios.

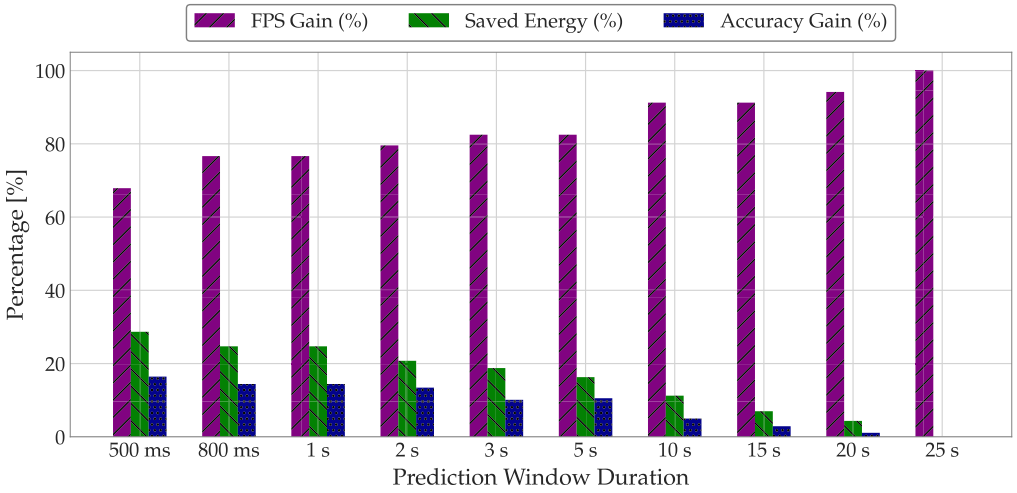


Fig. 15. Energy saving with FPS and Accuracy gains of PPM against static LC.

CRediT authorship contribution statement

Matteo Mendula: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Data curation, Conceptualization. **Paolo Bellavista:** Writing – review & editing, Writing – original draft, Supervision, Conceptualization. **Marco Levorato:** Writing – review & editing, Writing – original draft, Supervision, Funding acquisition, Conceptualization. **Sharon Ladron de Guevara Contreras:** Writing – review & editing, Visualization, Data curation.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Marco Levorato reports financial support was provided by National Science Foundation. Matteo Mendula is one of th invertor of Furcifer: A Context Adaptive Middleware for Real-world Object Detection Exploiting Local, Edge, and Split Computing in the Cloud Continuum pending to University of California Irvine. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was partially supported by National Science Foundation under grants CCF 2140154 and CNS 2134567.

Data availability

The dataset link is provided in the article.

References

[1] Eoin Martino Grua, Ivano Malavolta, Patricia Lago, Self-adaptation in mobile apps: a systematic literature study, in: 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, 2019.
[2] Jinsung Byun, Sehyun Park, Development of a self-adapting intelligent system for building energy saving and context-aware smart services, IEEE Trans. Consum. Electron. 57 (1) (2011) 90–98.
[3] Gabriel Orsini, Dirk Bade, Winfried Lamersdorf, CloudAware: Empowering context-aware self-adaptation for mobile applications, Trans. Emerg. Telecommun. Technol. (2017) e3210.
[4] Karthik Kumar, Jibang Liu, Yung-Hsiang Lu, Bharat Bhargava, A survey of computation offloading for mobile systems, Mob. Netw. Appl. (2012).
[5] Wenhan Zhan, Chunbo Luo, Geyong Min, Chao Wang, Qingxin Zhu, Hancong Duan, Mobility-aware multi-user offloading optimization for mobile edge computing, IEEE Trans. Veh. Technol. (2020).
[6] Kuanishbay Sadatdiynov, Laizhong Cui, Lei Zhang, Joshua Zhexue Huang, Salman Salloum, Mohammad Sultan Mahmud, A review of optimization methods for computation offloading in edge computing networks, Digit. Commun. Netw. (2023).
[7] Jie Ren, Ling Gao, Xiaoming Wang, Miao Ma, Guoyong Qiu, Hai Wang, Jie Zheng, Zheng Wang, Adaptive computation offloading for mobile augmented reality, in: Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 2021, pp. 1–30.
[8] Waqas Ur Rahman, Choong Seon Hong, Eui-Nam Huh, Edge computing assisted joint quality adaptation for mobile video streaming, IEEE Access (2019).
[9] Han-Kai Hsu, Chun-Han Yao, Yi-Hsuan Tsai, Wei-Chih Hung, Hung-Yu Tseng, Maneesh Singh, Ming-Hsuan Yang, Progressive domain adaptation for object detection, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, WACV, 2020.

[10] Dayan Guan, Jiaxing Huang, Aoran Xiao, Shijian Lu, Yanpeng Cao, Uncertainty-aware unsupervised domain adaptation in object detection, IEEE Trans. Multimed. (2022).
[11] Jihan Yang, Shaoshuai Shi, Zhe Wang, Hongsheng Li, Xiaojuan Qi, ST3D: Self-training for unsupervised domain adaptation on 3D object detection, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2021.
[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár, Microsoft COCO: Common objects in context, 2014.
[13] Mark Everingham, S.M. Ali Eslami, Luc Van Gool, Christopher K.I. Williams, John Winn, Andrew Zisserman, The pascal visual object classes challenge: A retrospective, Int. J. Comput. Vis. 111 (1) (2014) 98–136.
[14] Leonardo Galteri, Marco Bertini, Lorenzo Seidenari, Alberto Del Bimbo, Video compression for object detection algorithms, in: 2018 24th International Conference on Pattern Recognition, ICPR, 2018.
[15] Hyomin Choi, Ivan V. Bajić, Deep feature compression for collaborative object detection, in: 2018 25th IEEE International Conference on Image Processing, ICIP, 2018.
[16] Xiaofei He, Ming Ji, Hujun Bao, A unified active and semi-supervised learning framework for image compression, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009.
[17] Ewa Nowara, Daniel McDuff, Combating the impact of video compression on non-contact vital sign measurement using supervised learning, in: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops, 2019.
[18] David Tellez, Geert Litjens, Jeroen van der Laak, Francesco Ciompi, Neural image compression for gigapixel histopathology image analysis, IEEE Trans. Pattern Anal. Mach. Intell. (2021).
[19] David Tellez, Diederik Hoppener, Cornelis Verhoef, Dirk Grunhagen, Pieter Nierop, Michal Drozdzal, Jeroen van der Laak, Francesco Ciompi, Extending unsupervised neural image compression with supervised multitask learning, 2020, arXiv preprint arXiv:2004.07041.
[20] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, Sameer Singh, Distilled split deep neural networks for edge-assisted real-time systems, in: Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges, 2019.
[21] Yoshitomo Matsubara, Davide Callegaro, Sameer Singh, Marco Levorato, Francesco Restuccia, BottleFit: Learning compressed representations in deep neural networks for effective and efficient split computing, in: 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM, 2022.
[22] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, Paulius Micikevicius, Integer quantization for deep learning inference: Principles and empirical evaluation, 2020, arXiv preprint arXiv:2004.09602.
[23] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, Dmitry Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2018.
[24] John R. Klauder, The benefits of affine quantization, 2019, arXiv preprint arXiv:1912.08047.
[25] Jian Li, Raziel Alvarez, On the quantization of recurrent neural networks, 2021, arXiv preprint arXiv:2101.05453.
[26] Miloš Nikolić, Boukli Hacene, Bannon Ghouthi, Delmas Lascorz Ciaran, Courbariaux Alberto, Bengio Matthieu, Gripon Yoshua, Vincent, Andreas Moshovos, BitPruning: Learning bitlengths for aggressive and accurate quantization, 2020, arXiv preprint arXiv:2002.03090.
[27] Yuxuan Xie, Bing Liu, Lei Feng, Xipeng Li, Danyin Zou, A FPGA-oriented quantization scheme for MobileNet-SSD, in: Advances in Intelligent Information Hiding and Multimedia Signal Processing, 2019.
[28] Raghuraman Krishnamoorthi, Quantizing deep convolutional networks for efficient inference: A whitepaper, 2018, arXiv preprint arXiv:1806.08342.
[29] Caiwen Ding, Shuo Wang, Ning Liu, Kaidi Xu, Yanzhi Wang, Yun Liang, REQ-YOLO, in: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2019.
[30] Benedetta Liberatori, Ciro Antonio Mami, Giovanni Santacatterina, Marco Zullich, Felice Andrea Pellegrino, YOLO-based face mask detection on low-end devices using pruning and quantization, in: 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology, MIPRO, 2022.
[31] Shaobin Chen, Wei Lin, Embedded system real-time vehicle detection based on improved YOLO network, in: 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference, IMCEC, 2019.
[32] Christof Ebert, Gorka Gallardo, Josune Hernantes, Nicolas Serrano, DevOps, IEEE Softw. (2016).
[33] Sogand Shirinbab, Lars Lundberg, Emiliano Casalicchio, Performance evaluation of container and virtual machine running cassandra workload, in: 2017 3rd International Conference of Cloud Computing Technologies and Applications, CloudTech, 2017.
[34] Amit M.G. Potdar, D. Narayan, Shivaraj Kengond, Mohammed M. Moin Mulla, Performance evaluation of docker container and virtual machine, Procedia Comput. Sci. (2020).
[35] Cristian Constantin Spoiala, Alin Calinciuc, Corneliu Octavian Turcu, Constantin Filote, Performance comparison of a WebRTC server on Docker versus virtual machine, in: 2016 International Conference on Development and Application Systems, DAS, 2016.
[36] Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio, An updated performance comparison of virtual machines and Linux containers, in: 2015 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, 2015.
[37] Duo Lu, Dijiang Huang, Andrew Walenstein, Deep Medhi, A secure microservice framework for IoT, in: 2017 IEEE Symposium on Service-Oriented System Engineering, SOSE, 2017.
[38] Samodha Pallewatta, Vassilis Kostakos, Rajkumar Buyya, Placement of microservices-based IoT applications in fog computing: A taxonomy and future directions, ACM Comput. Surv. (2023).
[39] Jaewook Kim, Tae Joon Jun, Daeyoun Kang, Dohyeun Kim, Daeyoung Kim, GPU enabled serverless computing framework, in: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP, 2018.
[40] Diana M. Naranjo, Sebastián Risco, Carlos de Alfonso, Alfonso Pérez, Ignacio Blanquer, Germán Moltó, Accelerated serverless computing based on GPU virtualization, J. Parallel Distrib. Comput. (2020).
[41] Tanya Amert, Nathan Otterness, Ming Yang, James H. Anderson, F. Donelson Smith, GPU scheduling on the NVIDIA TX2: Hidden details revealed, in: 2017 IEEE Real-Time Systems Symposium, RTSS, 2017.
[42] Nathan Otterness, Ming Yang, Sarah Rust, Eunbyung Park, James H. Anderson, F. Donelson Smith, Alex Berg, Shige Wang, An evaluation of the NVIDIA TX1 for supporting real-time computer-vision workloads, in: 2017 IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS, 2017.
[43] Tagel Aboneh, Abebe Rorissa, Ramasamy Srinivasagan, Ashenafi Gemechu, Computer vision framework for wheat disease identification and classification using Jetson GPU infrastructure, Technologies (2021).
[44] Sergei Alyamkin, Matthew Ardi, Alexander C. Berg, Achille Brighton, Bo Chen, Yiran Chen, Hsin-Pai Cheng, Zichen Fan, Chen Feng, Bo Fu, et al., Low-power computer vision: Status, challenges, and opportunities, IEEE J. Emerg. Sel. Top. Circuits Syst. (2019).
[45] Tyler Harter, Brandon Salmon, Rose Liu, Andrea C. Arpaci-Dusseau, Remzi H. Slacker Arpaci-Dusseau, Slacker: Fast distribution with lazy docker containers, in: 14th USENIX Conference on File and Storage Technologies, FAST 16, 2016.
[46] Kent Gauen, Rohit Rangan, Anup Mohan, Yung-Hsiang Lu, Wei Liu, Alexander C. Berg, Low-power image recognition challenge, in: 2017 22nd Asia and South Pacific Design Automation Conference, ASP-DAC, 2017.

[47] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, Diana Marculescu, NeuralPower: Predict and deploy energy-efficient convolutional neural networks, in: Min-Ling Zhang, Yung-Kyun Noh (Eds.), Proceedings of the Ninth Asian Conference on Machine Learning, 2017.

[48] Jiajun Li, Ahmed Louri, Avinash Karanth, Razvan Bunescu, GCNAX: A flexible and energy-efficient accelerator for graph convolutional neural networks, in: 2021 IEEE International Symposium on High-Performance Computer Architecture, HPCA, 2021.

[49] Ji-Yan Wu, Vithurson Subasharan, Tuan Tran, Archan Misra, MRIM: Enabling mixed-resolution imaging for low-power pervasive vision tasks, in: 2022 IEEE International Conference on Pervasive Computing and Communications, PerCom, 2022.

[50] Roy Schwartz, Jesse Dodge, Noah A. Smith, Oren Etzioni, Green AI, Commun. ACM (2020).

[51] Nils Tijtgat, Wiebe Van Ranst, Toon Goedeme, Bruno Volckaert, Filip De Turck, Embedded real-time object detection for a UAV warning system, in: Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops, 2017.

[52] Amr Suleiman, Vivienne Sze, An energy-efficient hardware implementation of HOG-based object detection at 1080HD 60 fps with multi-scale support, J. Signal Process. Syst. (2015).

[53] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, SSD: Single shot multibox detector, in: Computer Vision – ECCV 2016, 2016.

[54] Micaela Verucchi, Gianluca Brilli, Davide Sapienza, Mattia Verasani, Marco Arena, Francesco Gatti, Alessandro Capotondi, Roberto Cavicchioli, Marko Bertogna, Marco Solieri, A systematic assessment of embedded neural networks for object detection, in: 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, 2020.

[55] Z. Tang, X. Liu, H. Chen, J. Hupy, B. Yang, Deep learning based wildfire event object detection from 4K aerial images acquired by UAS, AI 1 (2) (2020) 166–179, MDPI.

[56] Jianing Deng, Zhiguo Shi, Cheng Zhuo, Energy-efficient real-time UAV object detection on embedded platforms, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (2020).

[57] Huizi Mao, Song Yao, Tianqi Tang, Boxun Li, Jun Yao, Yu Wang, Towards real-time object detection on embedded systems, IEEE Trans. Emerg. Top. Comput. (2018).

[58] Alex Omid-Zohoor, Christopher Young, David Ta, Boris Murmann, Toward always-on mobile object detection: Energy versus performance tradeoffs for embedded HOG feature extraction, IEEE Trans. Circuits Syst. Video Technol. (2018).

[59] Sergio Moreschini, Fabiano Pecorelli, Xiaozhou Li, Sonia Naz, David Hästbacka, Davide Taibi, Cloud continuum: The definition, IEEE Access (2022).

[60] M. Rabbani, R. Joshi, An overview of the JPEG 2000 still image compression standard, Signal Process., Image Commun. 17 (1) (2002) 3–48, Elsevier.

[61] T. Ebrahimi, C. Horne, MPEG-4 natural video coding–An overview, Signal Process., Image Commun. 15 (4–5) (2000) 365–385, Elsevier.

[62] Yoshitomo Matsubara, Ruihan Yang, Marco Levorato, Stephan Mandt, SC2 benchmark: Supervised compression for split computing, Trans. Mach. Learn. Res. (2023).

[63] Yoshitomo Matsubara, Marco Levorato, Francesco Restuccia, Split computing and early exiting for deep learning applications: Survey and research challenges, ACM Comput. Surv. (2022).

[64] Yuzhu Ji, Haijun Zhang, Zhao Zhang, Ming Liu, CNN-based encoder–decoder networks for salient object detection: A comprehensive review and recent advances, Inform. Sci. (2021).

[65] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, Faster R-CNN: Towards real-time object detection with region proposal networks, in: Advances in Neural Information Processing Systems, 2016.

[66] Glenn Jocher, Alex Stoken, Jirka Borovec, Liu Changyu, Adam Hogan, Laurentiu Diaconu, Jake Poznanski, Lijun Yu, Prashant Rai, Russ Ferriday, et al., Ultralytics/Yolov5: V7.0 - YOLOv5 SOTA realtime instance segmentation, 2022, Zenodo.

[67] Nitin Sukhija, Elizabeth Bautista, Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus, in: 2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation, SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI, 2019.

[68] Isamu Takai, Tomohisa Harada, Michinori Andoh, Keita Yasutomi, Keiichiro Kagawa, Shoji Kawahito, Optical vehicle-to-vehicle communication system using LED transmitter and camera receiver, IEEE Photonics J. (2014).

[69] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, Evan Shelhamer, cuDNN: Efficient primitives for deep learning, 2014, arXiv preprint arXiv:1410.0759.

[70] J.-A. Lim, J. Lee, J. Kwak, Y. Kim, Cutting-edge inference: Dynamic DNN model partitioning and resource scaling for mobile AI, IEEE Trans. Serv. Comput. 17 (6) (2024) 3300–3316, http://dx.doi.org/10.1109/TSC.2024.3466848.

[71] Y. Matsubara, M. Mendula, M. Levorato, A multi-task supervised compression model for split computing, 2025, arXiv preprint arXiv:2501.01420.

[72] Yuyi Mao, Jun Zhang, Khaled B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices, IEEE J. Sel. Areas Commun. (2016).

[73] Dong Huang, Ping Wang, Dusit Niyato, A dynamic offloading algorithm for mobile computing, IEEE Trans. Wirel. Commun. (2012).

[74] Haiyan Wang, Wei Li, Jiayu Sun, Li Zhao, Xingmei Wang, Hongwu Lv, Guangsheng Feng, Low-complexity and efficient dependent subtask offloading strategy in IoT integrated with multi-access edge computing, IEEE Trans. Netw. Serv. Manag. (2023).

[75] F. Lai, X. Zhu, H.V. Madhyastha, M. Chowdhury, Oort: Efficient federated learning via guided participant selection, in: 15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, USENIX Association, 2021, pp. 19–35.

[76] Jonathan Shade, Dani Lischinski, David H. Salesin, Tony DeRose, John Snyder, Hierarchical image caching for accelerated walkthroughs of complex environments, in: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, 1996.

[77] S. Sen, J. Rexford, D. Towsley, Proxy prefix caching for multimedia streams, in: IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. the Future Is now, Cat. No. 99CH36320, 1999.

[78] Yong Yu, Xiaosheng Si, Changhua Hu, Jianxun Zhang, A review of recurrent neural networks: LSTM cells and network architectures, Neural Comput. (2019).

[79] Matteo Mendula, Paolo Bellavista, Marco Levorato, Sharon Ladron de Guevara Contreras, Furcifer: A context adaptive middleware for real-world object detection exploiting local, edge, and split computing in the cloud continuum, in: 2024 IEEE International Conference on Pervasive Computing and Communications, PerCom, 2024.

Matteo Mendula received his Ph.D. in Computer Science Engineering from the University of Bologna, Italy, in 2024. His research focuses on energy optimization for artificial intelligence, brain-inspired machine learning technologies, and edge computing. He earned both his Master's (2018) and Bachelor's (2020) degrees from the University of Bologna. During his Ph.D., he completed a visiting research period at the University of California, Irvine, where he further expanded his expertise in cutting-edge AI applications. Matteo has extensive experience in applied deep learning, with a strong background spanning computer vision, natural language processing, and transformer-based models. His work aims to enhance the efficiency and scalability of AI systems through innovative computational strategies.

Paolo Bellavista received the Ph.D. degree in computer science engineering from the University of Bologna, Italy, in 2001. He is currently a Full Professor with the University of Bologna. His research interests include middleware for mobile computing, QoS management in the cloud continuum, infrastructures for big data processing in industrial environments, and performance optimisation in wide-scale and latency-sensitive deployment environments. He serves on the Editorial Boards of IEEE Communications Surveys and Tutorials, IEEE Transactions on Network and Service Management, IEEE Transactions on Service Computing, ACM CSUR, ACM TIOT, and PMC (Elsevier). He is the Scientific Coordinator of the H2020 IoTwins Project (https://www.iotwins.eu).

Marco Levorato joined the Computer Science department at UCIrvine in August 2013. Between 2010 and 2012, He was a post-doctoral researcher with a joint affiliation at Stanford and the University of Southern California working with prof. Andrea Goldsmith and prof. Urbashi Mitra. From January to August 2013, he was an Access post-doctoral affiliate at the Access center, Royal Institute of Technology, Stockholm. He is a member of the ACM, IEEE and IEEE Comsoc society. His research interests are focused on next-generation wireless networks, signal processing, cyber-physical systems, smart city and smart energy systems. He has co-authored over 75 technical articles on these topics, including the paper that has received the best paper award at IEEE GLOBECOM (2012). He completed the PhD in Electrical Engineering at the University of Padova, Italy, in 2009. He obtained the B.S. and M.S. in Electrical Engineering summa cum laude at the University of Ferrara, Italy in 2005 and 2003, respectively. In 2016, he received the UC Hellman Foundation Award for his research on Smart City IoT infrastructures.

Sharon L.G. Contreras is a Ph.D. student at the Donald Bren Hall Computer Science Department at the University of California, Irvine. Her research focuses on edge computing, embedded systems, and autonomous driving. She holds a Master of Science in Electrical Engineering from the University of Southern California (2020) and dual Bachelor of Science degrees in Telematics Engineering and Computer Engineering from the Instituto Tecnológico Autónomo de México (2017, 2016). With a strong background in advanced computing and intelligent systems, her work aims to develop innovative solutions for real-time data processing and autonomous technologies.