Investigation of Serverless Consumption and Performance in Multi-Access Edge Computing

First author removed due to double-blind submission
Second author
name-firstauthor@email.com
name-second-author@email.com
Institution
City, state, county

ABSTRACT

The steadily increasing amount of computing resources made accessible through cutting-edge systems like Multi-Access Edge Computing has fundamentally transformed service delivery and quality. Nowadays, the majority of services are reliably available anywhere across the globe. But as the demand for computing resources is still increasing, there arises a necessity for solutions that are both resource-efficient and energy-consumption friendly, supplying the requirements of various computational tasks. In response to this need, serverless computing emerges as a promising next-generation deployment paradigm. Distinguished by its event-driven nature and adaptable lifecycle, serverless computing presents itself as a potential solution. However, due to its current confinement to cloud-based environments, a comprehensive exploration of its applicability within heterogeneous paradigms, such as Multi-Access Edge Computing, becomes imperative.

To address this gap, our paper evaluates the suitability of Multi-Access Edge Computing for hosting serverless functions. We thoroughly assess the performance, resource utilization, and energy consumption of serverless computing through real-world implementations. Our findings reveal that although the current serverless design possesses certain limitations, the serverless model holds the promise of enhancing resource and energy efficiency within the context of a Multi-Access Edge Computing environment.

CCS CONCEPTS

• **Networks** → **Cloud computing**; **Location based services**; *Network measurement*.

KEYWORDS

Serverless computing, Multi Access Edge Computing, IoT

ACM Reference Format:
First author removed due to double-blind submission and Second author. 2018. Investigation of Serverless Consumption and Performance in Multi-Access Edge Computing. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (AINTEC).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

AINTEC, December 12–14, 2023, Hanoi, VN

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The increasing demand for automation and data-driven use cases fosters the development of the Internet of Things (IoT). The number of connected devices is rising drastically, and the IoT finds applicability in various domains, including healthcare, security, sustainability, wearables, and digital twins [11]. As a result, IoT is becoming increasingly relevant in almost every facet of our daily lives, either already or in the near future. Moreover, applications utilizing the IoT generate substantial data traffic [26], which contributes to congestion, increases resources and energy demands, and leads to an impaired Quality of Service (QoS) within conventional cloud computing environments.

One reason for this development is the geographical distance between end-user devices and the cloud servers to which they transmit data. This geographical separation results in a notable increase in latency, negatively affecting users' QoS and consuming unnecessary energy. However, addressing these challenges posed by climate change is essential to reduce resource and energy consumption in the Internet technology sector [9]. Furthermore, from the perspective of network providers, ensuring adequate QoS for end users is crucial to prevent customer churn. Consequently, a range of innovative concepts have emerged to bridge the gap, bringing computing resources closer to end users. These concepts encompass distributed cloud, edge, and fog computing solutions that employ various containerized or virtual machine-based approaches.

To specifically tackle the challenge of optimizing customized computing resources and bring them in closer proximity to the end user, the utilization of containerization within a Multi-Access Edge Computing (MEC) framework has recently emerged as a promising solution. MEC entails the integration of computing pools adjacent to base stations, designed to offer minimal propagation delay for mobile or IoT services. On the other hand, containerization involves the adoption of a lightweight and adaptable virtualization technique for deployments in power-constrained devices situated at the network edge.

To further improve resource efficiency while maintaining a high QoS within the context of extensive IoT applications, MEC can adopt a next-generation deployment paradigm known as *serverless computing*. In serverless computing, a serverless function, which could either be a standalone application or nested within a container, is only "active" when triggered by an event. Otherwise, it can seamlessly transition through various "non-active" states, each with

distinct resource requirements. In theory, this approach has the potential to achieve resource and energy efficiency by strategically managing the lifecycle of serverless functions. Nonetheless, the predominant use of serverless computing in cloud environments, combined with the volatile nature of networks and the distributed characteristics of MEC, might impede serverless adoption from meeting its full potential. Additionally, the extent to which serverless capabilities can effectively secure relevant parameters within an MEC environment remains largely unexplored in existing literature. Consequently, a comprehensive investigation into the performance implications of implementing serverless computing in this context is essential.

For that reason, the contribution of this paper is twofold. Firstly, we undertake a quantitative analysis of the resource and energy consumption, alongside other key performance parameters, encompassing every phase in the lifecycle of serverless functions within a MEC environment. These metrics play an important role in refining serverless performance within the edge-cloud landscape.

Secondly, our examination reveals that while MEC, featuring diverse access technologies, is capable of integrating serverless computing, the existing serverless architecture displays vulnerabilities in terms of latency due to an inherent lack of support for volatile networks as encountered in MEC scenarios. These contributions hold substantial value for service providers as they need to decide whether to incorporate serverless into their systems and how best to use its potential efficiently.

Through our investigation into serverless behavior within the MEC context, we address the following three research questions.

RQ1: Is it feasible to deploy serverless functions on MEC's edge devices having limited computing resources in order to support complex IoT services, such as Machine Learning (ML) tasks?

RQ2: How does the presence of various networks within the MEC environment impact serverless performance, resource requirements, and the expected Quality of Service (QoS)?

RQ3: What trade-offs exist between resource and energy consumption and the performance of services when implementing serverless functions, with their dynamic lifecycles, within a MEC environment?

To answer these questions, we establish a real-world testbed that emulates a MEC environment. Through this setup, we perform comprehensive measurements of resource consumption metrics, including CPU, GPU, RAM, and power consumption for each component within a smart IoT deployment.

The remainder of this paper is structured as follows. Next, in Section 2, background and related work is summarized. Afterwards, Section 3 presents our testbed and introduces use case. Then, Section 4 evaluates the use cases and Section 5 concludes the work.

2 BACKGROUND AND RELATED WORKS

This section provides fundamental background information regarding MEC, serverless computing, and the ongoing efforts in integrating serverless computing into the MEC environment. The section concludes with a comprehensive summary about related literature.

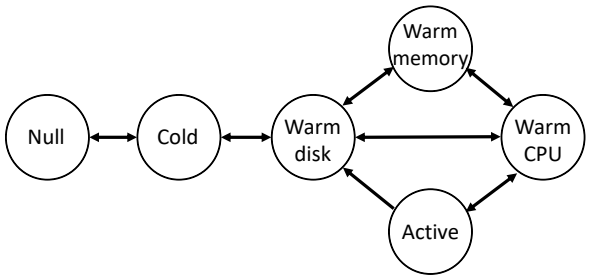

Figure 1: Lifecycle of a serverless function [19].

2.1 Multi-Access Edge Computing

Standardized by ETSI [21], MEC establishes a novel service environment incorporating cloud-computing capabilities seamlessly into the Radio Access Network (RAN), situated in close proximity to mobile subscribers [8]. This integration enables MEC servers to significantly reduce response times for data processing, and as a consequence relieving the stress of backbone links. Yet, MEC has been applied to diverse applications, including efficient Domain Name Systems (DNS) entry caching, content caching, and precision tracking of devices [20].

While MEC encounters the common challenge of managing heterogeneous multidimensional resources seen in edge–cloud or fog computing, it introduces an additional layer of complexity due to device mobility within intricate and variable mobile networks. This complexity poses challenges to the adoption of advanced platforms like serverless computing [18].

2.2 Serverless Computing

Serverless computing emerged first in cloud computing as a service provided by AWS [1]. This innovation streamlines the deployment process for developers by relieving them of tasks such as cluster management and network orchestration. While serverless environments support native functions, container-based approaches have gained prominence across both commercial platforms (e.g., AWS lambda [1]) and open-source solutions (e.g., Knative [13], IBM OpenWhisk [2]). Within this context, serverless functions are encapsulated and deployed on demand using ephemeral containers. Consequently, container orchestration frameworks like Kubernetes [15] usually serve as underlying systems for such serverless platforms.

The lifecycle of serverless functions consists of several *states* and *processes*. For instance, a model with six distinct *states* has been introduced in a previous study [19]. The transitions between the states are called *processes*, as shown in Figure 1. In brief, the *Cold* state stands for the pure existence of an abstraction, while *Warm Disk* denotes the presence of the function's image on the device. *Warm CPU* and *Warm Memory* indicate that a function has been instantiated but is not actively processing data. These states differ based on CPU and memory activation. The *Active* state, on the other hand, is only invoked when a data processing request is triggered. As highlighted in [19], both remaining in and transitioning between these states incur costs, including resource utilization, energy consumption, and latency. Understanding these

costs helps operators manage resource consumption effectively and ensure the QoS in serverless deployments.

2.3 Related Work

While predominantly utilized as cloud services in the commercial domain, serverless computing is gaining attention within distributed environments like edge–cloud and MEC. This interest arises out of its ability to enhance resource efficiency for unpredictable, event-triggered workloads by automatically deallocating idle resources [3].

To demonstrate the effectiveness of serverless computing in the IoT context, Wang et al. [27] establish a testbed and assess the performance of a smart-home use case deployed as serverless functions. The findings reveal that the serverless deployment utilizes between 30 % to 60 % less CPU time compared to standard setups.

As resource-intensive ML tasks gain interest at the edge, numerous researchers aim at developing more resource-efficient and well-provisioned models using serverless computing. Rausch et al. [23] integrate the ML lifecycle into a serverless-based edge–cloud architecture to enhance both resource allocation and workflow management. Glikson et al. [10] devise a serverless model that extends ML capabilities to edge computing, leveraging the OpenWhisk framework. Thorpe et al. [25] introduce a novel serverless-enabled platform named Dorylus to enhance the performance-to-resource ratio of ML training tasks. Despite these advancements, certain challenges persist across the mentioned studies. The issue of "cold-start" latency when initiating functions and the uncertain cost implications of serverless deployment within edge systems continue to show difficulties, even when provisioning functions within local networks.

Nguyen et al. [19] address these challenges by investigating the consumption and latency patterns of ML workloads deployed on an embedded computer. This is achieved by analyzing each component within the function's lifecycle. The study demonstrates that while intermediate states in the lifecycle consume minimal resources, transitioning between these states incurs significant energy costs and significantly impacts QoS. However, Nguyen's environment consists of only one edge device. Any interactions and trade-offs between performance and consumption, influenced by heterogeneous networks and more realistic models such as MEC, have not been thoroughly considered.

In the context of the MEC environment, Chaudry et al. [5] undertake an assessment of serverless performance regarding resource consumption and QoS. Cicconetti et al. [6] focus primarily on the theoretical model for distributed serverless computing within the MEC. However, their model simplifies several crucial constraints, such as latency caused by different network types and transitions among lifecycle states. A comprehensive set of open questions regarding the integration of serverless computing into MEC is outlined in [18], highlighting various challenges like the benefit of serverless flexibility or resource-heterogeneous MEC deployments. For that reason, this paper applies an empirical approach to address and resolve these challenges, thereby bridging the gap between theory and practical implementation.

Table 1: Network parameters for emulation. Jitter follows a normal distribution for all cases.

Criteria	Network		
	3G [24]	4G [22] [17]	Wifi [12]
Bandwidth	11 ~5536 Kbps	1.543 ~108.693 Mbps	150 Mbps
Latency	25 ms	10 ~15 ms	5 ~10 ms
Jitter	10 ms	4 ms	1 ms

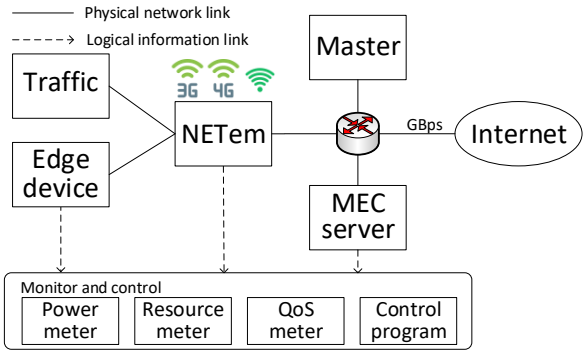

Figure 2: Implementation of the testbed.

3 TESTBED AND USE CASE

To address our defined research questions, we adopt an empirical methodology. Therefore, we establish a MEC-emulated testbed to thoroughly investigate serverless functions and define use cases in this section.

3.1 Testbed Setup

The overview of the testbed is shown in Figure 2. In general, it contains four main parts. Firstly, the serverless cluster comprises two computing nodes and one Master node tasked with management responsibilities. The computing nodes, referred to as Worker nodes, contain both edge devices and MEC servers, each equipped with GPU resources, while the MEC servers have more computational capabilities compared to the edge devices.

For our testbed, we use Knative [13], a well-known open-source framework, as our chosen serverless solution. Notably, Knative operates based on Kubernetes, the widely adopted container orchestration system developed by Google. Consequently, serverless functions are deployed as Kubernetes instances.

The emulation of the MEC network is achieved through the utilization of a network emulator (NETem), which operates on a Raspberry Pi equipped with Linux-TC (traffic control) tool [16]. The TC tool effectively emulates three distinct network types: 3G, 4G, and WiFi. The reason that motivates us to use 3G, a seemingly outdated network, is to emulate a realistic, bad network condition. These emulated network conditions correspond to data traces collected from Table 1. Note that solely the network associated

Table 2: Testbed instruments.

Role	Hardware	Software
Edge device	Jetson Nano: 1.43GHz Cortex-A57, 2GB RAM, GPU NVIDIA Maxwell	Ubuntu 20.04 Kubernetes 1.23.5 Knative 1.8
MEC server	Linux PC: 3.6GHz Ryzen 7 3700X, 32GB RAM, GPU NVIDIA GeForce RTX 2060 6GB VRAM	
Network emulator	Raspberry Pi 4B TP-LINK USB-to-Gigabit adapter	Ubuntu 20.04 Linux TC
Traffic generator	Regular Linux PC	Ubuntu 20.04
Measurement	Tinkerforge Voltage/Current and Energy Bricklets	Prometheus 2.34 Curl 7.68

with devices located to the left of NETem is constrained. Other components, such as the Master node and the MEC server, can communicate with one another and access the Internet at the full link speed, which is set to 1 Gbps within our setup.

The third component is the traffic generator, which functions as the end device, producing data and issuing requests for processing. Typically, the edge device also serves as the source of traffic. However, in our testbed, we consciously separate these roles to isolate the impact of non-targeted processes on the computing device. The final component is the monitoring and control device, responsible for supervising the measurement procedures and collecting measurement data. Additional details about the testbed are summarized in Table 2.

3.2 Use Cases and Measurement Criteria

In our setup, we deploy the YOLO-v4 [4] object detection function as the smart IoT service. This application takes streaming video as the input, subsequently analyzing each frame and providing the count of objects detected within. Notably, this application exhibits versatility and widespread applicability across various practical scenarios, as surveyed in [7]. Within our study, this service is instantiated as a serverless function. Its instance can be initiated either at an edge device or an MEC server. When the traffic generator requests streaming data from the function's address, the system requests the creation of a serverless instance to process the streaming data. Following the task's completion, the instance is automatically removed. Throughout this sequence of events, the monitoring mechanism identifies and quantifies the lifecycle of the serverless instance.

The metrics we measure include the *resource consumption* covering CPU, GPU, RAM, and power for each component within the function's lifecycle, involving states and processes as depicted in Figure 2. However, we exclude *Warm Memory* out of the scope since this state has no valuable use [20], and it has been removed from the newest version of the platform [14]. Additionally, we examine *performance* metrics including latency of each process and the frame per second (FPS) of the function's output in relation to its

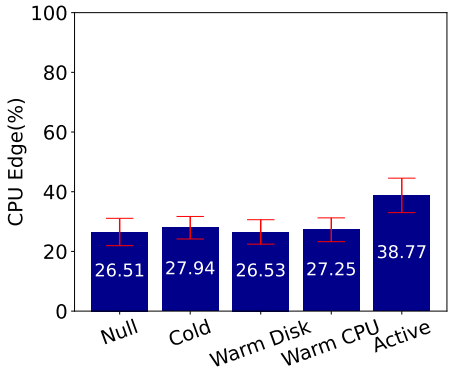

Figure 3: Function states' CPU usage under maximum load at the edge device.

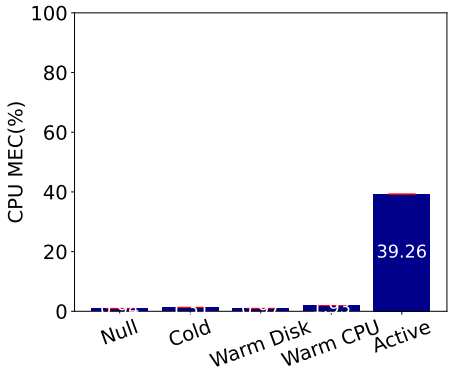

Figure 4: Function states' CPU usage under maximum load at the MEC server.

power consumption. The former determines the major contributing component to the function's total latency, while the latter offers insights into function efficiency across diverse network types and computing resources available within the MEC environment.

4 USE CASE EVALUATION

In this section, we provide the measurement outcomes for each state and process throughout the lifecycle of a serverless deployment within the MEC environment.

4.1 Serverless Resource Consumption in Multi-Access Edge Computing

This section investigates serverless resource consumption by a comprehensive utilization analysis of each state and process in the 4G-enabled MEC environment. Firstly, we investigate the different states' resource requirements in the following.

4.1.1 State's Resource Consumption. Figure 3 and 4 illustrate the CPU usage of different states of a serverless function placed at the edge device and the MEC server, respectively. In both cases, we record the maximum number of instances the device can manage, having a single function at the edge and five functions at the MEC

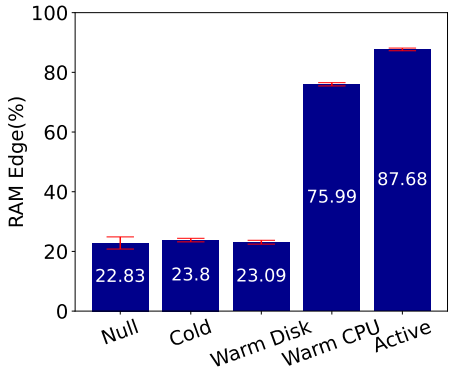

Figure 5: Function states' RAM usage under maximum load at the edge device. Error bars are 90 % confidence intervals (this also applies for the following figures).

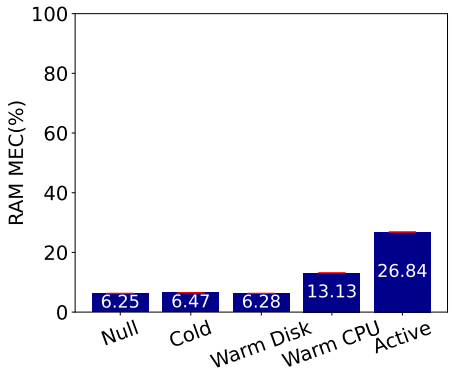

Figure 6: Function states' RAM usage under maximum load at the MEC server.

server. The limiting factor that hinders the system from hosting additional instances stems from the GPU vRAM capacity. Under this maximum load, only the *Active* state consumes significant CPU resources to process arriving requests.

The RAM consumption shown in Fig. 5 and. 6 follows a similar pattern to the CPU usage, with the exception of the Warm CPU state, utilizing a notable amount of RAM. This allocation is used for reserving variables and libraries for the active instance, even prior to the arrival of requests. Additionally, we have measured the power consumption across states, closely mirroring the CPU trend.

When comparing the two devices, it becomes apparent that the edge device struggles to independently manage the serverless framework. This is evident from the fact that close to a quarter of the CPU capacity is utilized during the *Null* state. Additionally, the ML workload utilizes 80 % to 90 % of the device's RAM in *Warm CPU* and *Active*, respectively.

In the case of WiFi, there is only a marginal variation in the state's consumption. Similarly, when considering 3G, differences are mainly noticeable during the *Active* state, which can be attributed to the lower data rate leading to a decreased load. Thus, for the paper's clarity, we omit these analyses in the following.

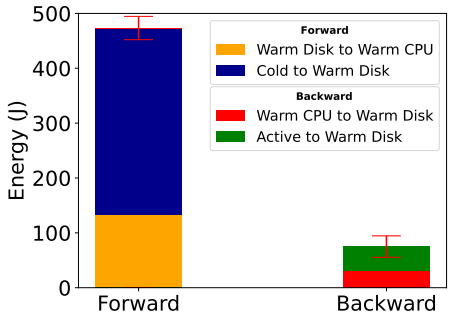

Figure 7: Lifecycle processes' average energy usage at the edge device. x axis indicates the direction of processes.

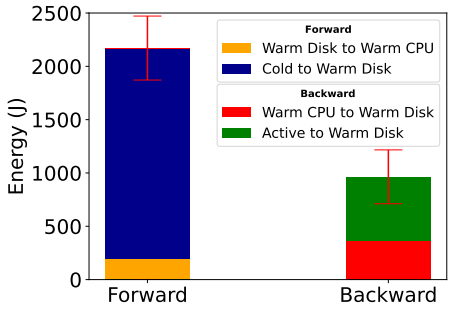

Figure 8: Lifecycle processes' average energy usage at the MEC server. x axis indicates the direction of processes.

4.1.2 Process's Resource Consumption. Transiting between different states involves resource allocation and deallocation aligned with the intended state, as discussed in [19], where we observe a similar trend in our results. Given this alignment, we will omit those details and instead concentrate on the energy consumption and latency associated with these transitions.

Focusing on energy consumption, we highlight only processes that consume considerable amounts. Figure 7 and 8 describe the four most energy-consuming processes. In this context, *forward* means the sequence {Null → Cold → Warm Disk → Warm CPU → Active}, representing the resource allocation for creating a serverless function, as depicted by the left to right arrows in Figure 1. Reversely, *Backward* refers to the resource deallocation.

Overall, the forward direction consumes considerably more energy compared to the backward. In detail, the *Cold to Warm Disk* transition, involving the download and the extraction of a container's image accounts for the majority of energy consumption. The creation of an instance, denoted as *Warm Disk to Warm CPU*, also contributes significantly during the forward processes.

The energy consumption is also not negligible during the backward processes, which should not be ignored. Comparing the energy consumption between the two computing tiers, edge and MEC, it is evident that the MEC server consumes more energy during the same processes due to its more powerful hardware. Conversely, processing on the MEC is generally faster than on an edge device. Much

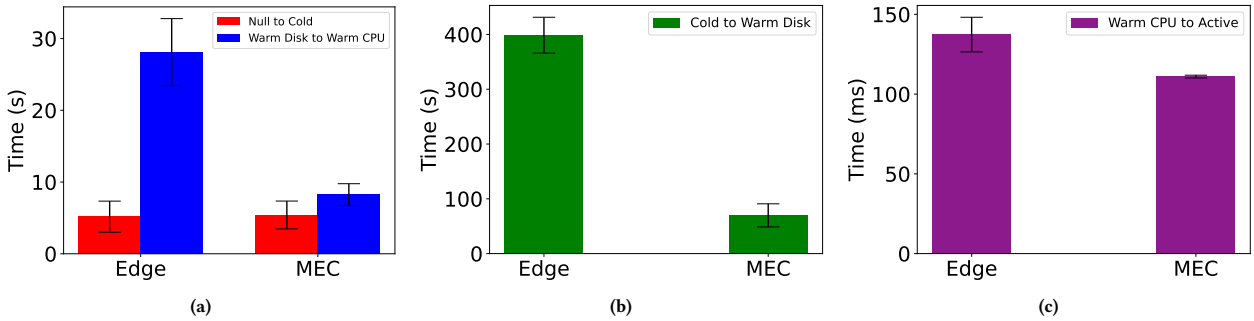

Figure 9: Latency of each process in the serverless lifecycle placed at different locations. x axis indicates the function's location.

like the consumption trends of states, the energy consumption of processes remains relatively consistent using WiFi or 3G.

4.2 Serverless Latency in Multi-Access Edge Computing

This section analyzes the serverless performance with a focus on the latency of processes that have a direct impact on the QoS (forward processes). We explore various instance placements, network configurations, and input loads within the MEC environment.

4.2.1 Latency at Different Placement. Each lifecycle's process has a different duration in the range of milliseconds to hundreds of seconds. For that reason, we illustrate the results in three subfigures in Figure 9. Generally, functions deployed on the MEC consistently exhibit lower latency due to more powerful hardware, enabling faster task processing.

This is reflected in the case of *Warm Disk to Warm CPU*, illustrated in Figure 9–a, where the creation of a container takes place. For the *Null to Cold* process, only the service abstraction is generated at the Master node. Thus, it is not affected by the workers' capability. In Figure 9-b, the *Cold to Warm Disk* process at the MEC repeatedly outperforms its counterpart at the edge. This discrepancy arises because the MEC server's capabilities are not constrained by network limitations, such as the 4G network in the case of edge device.

The latency when a request initiates a transition from *Warm CPU* to *Active* is shown in Figure 9-c. While this process is influenced solely by latency rather than the device's capabilities, the MEC's results again outperform those of the edge. This behavior stands in contrast to the consensus that proximity to the user results in lower perceived latency. We have identified that existing serverless open-source solutions, including Knative, do not adequately consider distributed environments like the MEC. Consequently, networking services like gateways and load balancers are optimally positioned on the more powerful device, which, in our testbed, is the MEC server. This architectural choice leads to end-user requests always being directed to the MEC server first, independent of the function's actual location. This challenge persists across our various results.

Combined with the analysis in Section 4.1, we can answer our first research question RQ1 as follows. *The MEC environment, considering both edge device and MEC server work jointly, demonstrates the potential for intelligent deployment such as serverless functions. However, the inherent limitations of less powerful edge device complicate the hosting of serverless frameworks and ML tasks but also suffers from high latency caused by the non-edge-oriented design of the existing platform.*

4.2.2 Latency with Different Networks for Function at Edge Device. Figure 10 investigates the impact of network conditions on the lifecycle's processes when the function is placed on an edge device. Notably, enhanced network quality, characterized by low latency, minimal jitter, and higher bandwidth, consistently leads to lower latency across all processes.

Processes that heavily rely on network quality, such as the *Warm CPU to Active* transition, show different results among different networks. In particular, during the process *Cold to Warm Disk* in a 3G network, as shown in Figure 10–b, a system error prevented the image from being downloaded within 10 min. Thus, we could not record the value and mark it as α.

However, other processes are less impacted by the network type, like *Warm Disk to Warm CPU*. We attribute this to the potential impact of network latency on the system calls occurring between the Master node and the edge device during the container allocation process. Conversely, the *Null to Cold* process remains unaffected, as it simply involves the creation of an abstraction at the Master node.

4.2.3 Latency at Multi-Access Edge Computing Server. Given the proximity of the MEC server to the Master node within the same network, as depicted in Figure 2, the majority of the function's processes are not affected by the network type. For that reason, we focus on the examination of the relationship between load and latency. Therefore, we increment the load by concurrently spawning varying numbers of instances, as illustrated in Figure 11 and 12. Figure 11 shows the latency of the two network-independent processes with varying numbers of instances that are spawned concurrently. Generally, the latency increases proportional to the load.

The latency behavior of the *Warm CPU to Active* process is depicted in Figure 12. This case stands as an exception, being influenced by network characteristics. We notice that the 3G and the 4G network induce more than 100 ms latency, while WiFi maintains sub-100 ms. However, increasing the load shows no significant additional result. Given that all functions share the same YOLO-v4

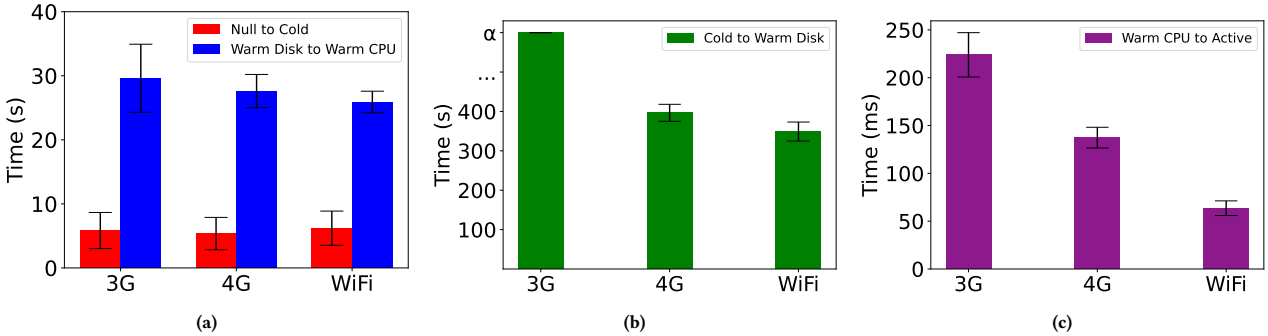

Figure 10: Latency of each process in the serverless lifecycle placed at the edge in different networks. x axis indicates the network type.

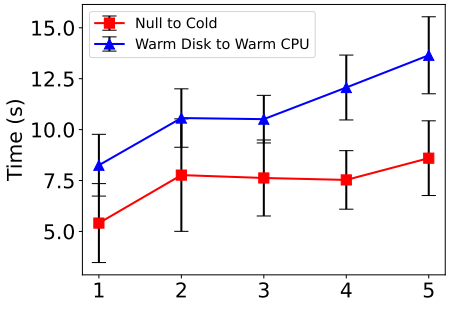

Figure 11: Latency of network-free lifecycle's processes at the MEC server. x axis indicates the number of concurrent functions.

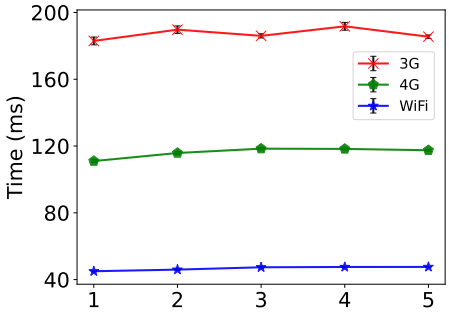

Figure 12: Latency of *Warm CPU to Active* process at the MEC server. x axis indicates the number of concurrent functions.

image, the system downloads it only once, regardless of the number of instances deployed. As a result, the *Cold to Warm Disk* phase remains unaffected by load increases.

Based on these outcomes, the second research question, RQ2 can be answered. *Different network types within the MEC context have a noticeable impact on the function's QoS. This impact is particularly notable in processes such as the bandwidth-intensive* Cold to Warm Disk *and the latency-sensitive* Warm CPU to Active. *Consequently,*

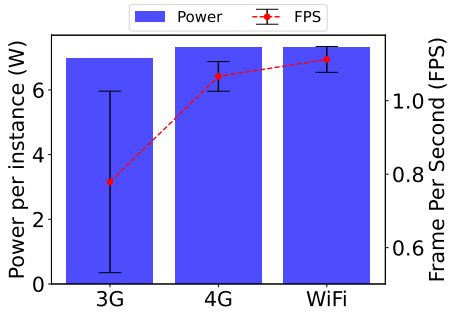

Figure 13: Average power usage per function or instance versus the output FPS at the edge device for different network types.

if enough resources are available, the best practice is that the function immediately progresses through its lifecycle after deployment to reduce avoidable latency. Moreover, a careful consideration of the specific characteristics of an application is essential prior to deploying it within a particular MEC network. This approach ensures that the chosen network environment aligns effectively with the application's unique requirements and performance expectations.

4.3 Comparison of Power and Performance

To determine the efficiency of deploying serverless functions over MEC, conduct a comparison between power consumption per function and the average output FPS as depicted in Figure 13 and 14. In this context, "load" is the number of instances that can effectively transition to the Active state to handle incoming requests. As seen in Figure 13, edge device can accommodate only one instance with very low FPS, of around one, even under optimal network conditions (WiFi). On the other hand, MEC performance significantly outperforms this, maintaining much higher FPS even in challenging network scenarios such as 3G. Even at peak load, MEC manages to deliver approximately six to eight frames per second. Hence, the limiting factor for the edge device is its computational capability rather than network constraints.

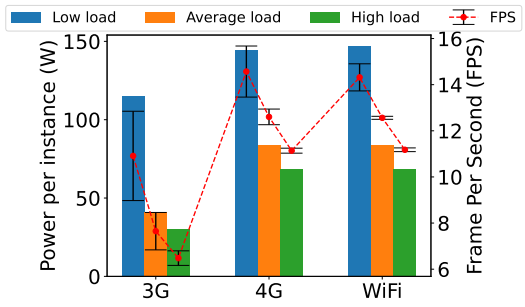

Figure 14: Average power usage per function or instance versus the output FPS at the MEC server for different network types. Load indicates the number of running instances: one for low, three for average, and five for high load.

In terms of network type, with the same load, both 4G and WiFi exhibit comparable performance, whereas 3G displays not only a consistent 40 % to 50 % decline in FPS but also fluctuations due to network instability.

Considering the power consumption, the edge device shows the advantage of consuming minimal power, especially compared to the power-intensive MEC server. Conversely, in the case of MEC, the power per instance ratio inversely decreases as the number of hosted instances rises. Consequently, while scaling up the load results in a performance loss of up to 40 % in terms of FPS, the power consumption per function experiences a dramatic reduction of nearly threefold.

Consequently, we can answer our third and last research question, RQ3, as follows: *A trade-off between environmental factors, like computing power, network quality, and power consumption, is seen regarding the function's output performance. Better networks and more powerful machines yield higher performance but consume more power. Since the power per load decreases with increasing load, power-intense machines like MEC servers should always be utilized with the maximal possible load. Furthermore, a clear trade-off between RAM and latency is seen considering the function's lifecycle. In particular, the closer a state is to* Active*, the lower latency is expected with negligible power and resource demands.*

5 CONCLUSION AND FUTURE WORK

The growing demand for the IoT driven by automation and data-centric requirements demands conventional cloud models, negatively impacting the QoS and energy efficiency. Therefore, bridging the gap between end users and cloud servers is vital, leading to concepts like MEC or edge–cloud. Furthermore, the use of serverless computing within MEC shows promise in optimizing resource efficiency while ensuring the QoS for IoT applications. However, the performance implications of integrating serverless computing into the variable MEC environment require further exploration.

In this work, we have investigated the deployment of serverless functions within the MEC environment through a comprehensive measurement study by our developed testbed. Our results show that serverless functions, specifically smart deployments, have the potential to be integrated into MEC. This synergy leverages the flexible lifecycle and event-driven characteristics of serverless computing to decrease resource consumption while also capitalizing on the cloud's capabilities positioned within the edge network of MEC. However, the current serverless architectural design requires considerable resources and induces high latency at limited-capability edge devices. In addition, costs in terms of energy consumption accumulated over several processes and the unstable performance owing to variable networks in MEC are challenges, particularly concerning edge devices with constrained capabilities. To this end, an edge-oriented platform for serverless to eliminate current latency issues and lifecycle management that can exploit serverless benefits should be considered in the future.

REFERENCES

[1] Amazon. [n. d.]. *Serverless Computing - AWS Lambda - Amazon Web Services.* https://aws.amazon.com/lambda/
[2] Apache. [n. d.]. *Open Source Serverless Cloud Platform.* https://openwhisk.apache.org/
[3] Mohammad S. Aslanpour, Adel N. Toosi, Claudio Cicconetti, Bahman Javadi, Peter Sbarski, Davide Taibi, Marcos Assuncao, Sukhpal Singh Gill, Raj Gaire, and Schahram Dustdar. 2021. Serverless Edge Computing: Vision and Challenges. In *Proceedings of the 2021 Australasian Computer Science Week Multiconference* (Dunedin, New Zealand). Association for Computing Machinery.
[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934* (2020).
[5] Saqib Rasool Chaudhry, Andrei Palade, Aqeel Kazmi, and Siobhán Clarke. 2020. Improved QoS at the Edge Using Serverless Computing to Deploy Virtual Network Functions. *IEEE Internet of Things Journal* (2020).
[6] Claudio Cicconetti, Marco Conti, and Andrea Passarella. 2020. Uncoordinated access to serverless computing in MEC systems for IoT. *Computer Networks* (2020).
[7] Tausif Diwan, G Anirudh, and Jitendra V Tembhurne. 2023. Object detection using YOLO: Challenges, architectural successors, datasets and applications. *multimedia Tools and Applications* (2023).
[8] ETSI. 2015. *Mobile Edge Computing – A Key Technology towards 5G.* White Paper. ETSI MEC ISG, Sophia Antipolis, France.
[9] European Climate, Infrastructure and Environment Executive Agency. 2022. Horizon Europe Work Programme 2023-2024: Climate, Energy and Mobility.
[10] Alex Glikson, Stefan Nastic, and Schahram Dustdar. 2017. Deviceless Edge Computing: Extending Serverless Computing to the Edge of the Network. Association for Computing Machinery.
[11] Global Data. 2021. Global IoT Market will Surpass the $1 Trillion Mark by 2024, Says Global Data. https://www.globaldata.com/media/thematic-research/global-iot-market-will-surpass-1-trillion-mark-2024-says-globaldata/ Accessed: 2023-08-24.
[12] Ilya Grigorik. 2013. *High Performance Browser Networking.* O'Reilly Media, Inc.
[13] Knative. [n. d.]. *Serverless Containers in Kubernetes Environments.* https://knative.dev/docs/
[14] Knative. [n. d.]. *v1.10 release Knative.* https://knative.dev/blog/releases/announcing-knative-v1-10-release/
[15] Kubernetes. [n. d.]. *Kubernetes: Production-Grade Container Orchestration.* https://kubernetes.io
[16] Linux. [n. d.]. *tc(8) - Linux manual page.* https://man7.org/linux/man-pages/man8/tc.8.html
[17] Nikos Makris, Virgilios Passas, Thanasis Korakis, and Leandros Tassiulas. 2018. Employing MEC in the Cloud-RAN: An Experimental Analysis. In *Proceedings of the 2018 on Technologies for the Wireless Edge Workshop* (New Delhi, India). Association for Computing Machinery.
[18] Kien Nguyen, Frank Loh, and Tobias Hoßfeld. 2023. Challenges of Serverless Deployment in Edge-MEC-Cloud. (2023).
[19] Kien Nguyen, Frank Loh, Tung Nguyen, Duong Doan, Nguyen Huu Thanh, and Tobias Hoßfeld. 2023. Serverless Computing Lifecycle Model for Edge Cloud Deployments. (June 2023). Paper presented at the 2ND WORKSHOP ON GREEN AND SUSTAINABLE NETWORKING (GREENNET 2023), part of the IEEE International Conference on Communications.
[20] The-Vi Nguyen, Nhu-Ngoc Dao, Van Dat Tuong, Wonjong Noh, and Sungrae Cho. 2022. User-Aware and Flexible Proactive Caching Using LSTM and Ensemble Learning in IoT-MEC Networks. *IEEE Internet of Things Journal* (2022).
[21] Milan Patel, Yunchao Hu, Patrice Hédé, Jerome Joubert, Chris Thornton, Brian Naughton, Julian Roldan Ramos, Caroline Chan, Valerie Young, Soo Jin Tan, Daniel Lynch, Nurit Sprecher, Torsten Musiol, Carlos Manzanares, and Uwe

Rauschenbach. 2014. *Mobile-edge computing introductory technical white paper*. White Paper. Mobile-Edge Computing (MEC) Industry Initiative, Sophia Antipolis, France.

[22] Darijo Raca, Jason J. Quinlan, Ahmed H. Zahran, and Cormac J. Sreenan. 2018. Beyond Throughput: A 4G LTE Dataset with Channel and Context Metrics. In *Proceedings of the 9th ACM Multimedia Systems Conference* (Amsterdam, Netherlands). Association for Computing Machinery.

[23] Thomas Rausch, Waldemar Hummer, Vinod Muthusamy, Alexander Rashed, and Schahram Dustdar. 2019. Towards a Serverless Platform for Edge AI. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*.

[24] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. 2013. Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications. In *Proceedings of the 4th ACM Multimedia Systems Conference* (Oslo, Norway). Association for Computing Machinery.

[25] John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2021. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association.

[26] Lionel Sujay Vailshery. [n. d.]. *IoT connected devices worldwide from 2019 to 2023*. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[27] I Wang, Elizabeth Liri, and KK Ramakrishnan. 2020. Supporting iot applications with serverless edge clouds. In *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*. IEEE.