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

Kien Nguyen*, Frank Loh*, Kiem Nguyen†, Chau Nguyen†, Nguyen Huu Thanh†, Tobias Hoßfeld*
*University of Würzburg, Institute of Computer Science, Würzburg, Germany
†Hanoi University of Science and Technology, Hanoi, Vietnam
Email:{firstname.lastname}@uni-wuerzburg.de or thanh.nguyenhuu@hust.edu.vn

***Abstract*—Serverless computing is a promising next-gen deployment paradigm that is resource-efficient and energy-usage friendly. 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 (MEC), becomes imperative. To address this gap, this paper evaluates the suitability of MEC to host serverless functions. We thoroughly assess the performance, resource utilization, and energy consumption of serverless computing through a realistic implementation. Our findings reveal that, although the current serverless design possesses certain limitations, the serverless model holds the promise of enhancing resource and energy efficiency in the context of a MEC environment.**

***Index Terms*—Serverless computing, Multi Access Edge Computing, Internet of Things**

I. 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 increasing drastically, and the IoT finds applicability in various domains, including healthcare, security, sustainability, and digital twins [1]. As a result, IoT applications generate substantial data traffic [2], which contributes to congestion, increasing resource and energy demands, and leads to an impaired Quality of Service (QoS) within conventional cloud computing environments.

To tackle this problem, innovative concepts like distributed cloud, edge, and fog computing, with containerized or virtual machine-based approaches, are bridging the gap between computing resources and end users. In this context, containerization in Multi-Access Edge Computing (MEC) is a promising solution to optimize and deliver customized computing resources closer to end users. MEC integrates computing pools near base stations to minimize latency for mobile and IoT services. Containerization is a lightweight and adaptable virtualization technique to deploy applications on power-constrained devices 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 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 are important to refine serverless performance within the edge-cloud landscape. Second, our examination reveals that while MEC can integrate serverless computing, the existing architecture is vulnerable to latency due to its lack of support for volatile networks. This is valuable to service providers, who need to decide whether and how to incorporate serverless computing into their systems.

Based on our investigation into the serverless behavior in the MEC context, we address the following research questions.

RQ1: Is it feasible to deploy serverless functions on MEC's edge devices with limited computing resources 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 QoS?

RQ3: What are the trade-offs between resource usage, energy consumption, and performance when serverless with dynamic lifecycles is implemented in 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 II, background and related work is summarized. Afterwards, Section III presents our testbed and introduces use case. Then, Section IV evaluates the use cases and Section V concludes the work.

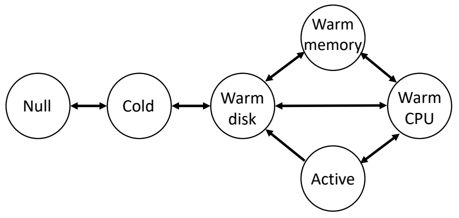


Fig. 1: Lifecycle of a serverless function [6].

II. Background and Related Works

This section provides fundamental background information regarding MEC, serverless computing, and the ongoing efforts to integrate serverless computing into the MEC environment.

A. Multi-Access Edge Computing

Standardized by ETSI [3], MEC establishes a novel service environment to incorporate cloud-computing capabilities seamlessly into the Radio Access Network (RAN), situated in the proximity to mobile subscribers [4]. This integration enables MEC servers to reduce response times for data processing and, consequently, relieve the stress of backbone links. On the other hand, MEC introduces an additional layer of complexity due to device mobility within intricate and variable mobile networks. This complexity poses challenges to adopt advanced platforms like serverless computing [5].

B. Serverless Computing

Serverless computing emerged first in the cloud computing context as a service provided by AWS [7]. 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 like AWS lambda [7] and open-source solutions like Knative [8] or IBM OpenWhisk [9]. Within this context, serverless functions are encapsulated and deployed on demand using ephemeral containers. Consequently, container orchestration frameworks like Kubernetes [10] usually serve as underlying systems for such platforms. The lifecycle of serverless functions consists of several *states* and *processes*. For instance, a model with six distinct *states* has been introduced in our previous study [6]. 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 [6], both remaining in, and transitioning between these states incur costs, including resource utilization, energy consumption, and latency. Understanding these costs helps operators to manage resource consumption effectively and ensure the QoS in serverless deployments.

C. Related Work

While predominantly utilized as cloud services in the commercial domain, serverless computing has gaining attention within distributed environments like the edge–cloud and by MEC. This interest arises from its ability to enhance resource efficiency for unpredictable, event-triggered workloads by automatically deallocating idle resources [11].

Wang et al. [12] demonstrate the effectiveness of serverless computing in the IoT context via a smart-home testbed. The results 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, researchers are developing more resource-efficient and well-provisioned models using serverless computing. Furthermore, serverless-based edge-cloud architectures are proposed to enhance resource allocation and workflow management [13] and extend ML capabilities to the edge using serverless frameworks [14]. Despite advances, studies continue to show difficulties with cold-start latency and uncertain cost implications of serverless edge computing, even when provisioning functions are kept locally. In our previous work [6], we investigate ML workload consumption and latency on embedded computers, analyzing each component of the function lifecycle. We show that intermediate states consume minimal resources, but transitioning between states incurred significant energy costs and impact QoS. However, the study is limited to local edge networks and interactions and trade-offs between performance and consumption in heterogeneous networks and more realistic models such as MEC are not considered.

In the context of the MEC environment, Chaudry et al. [15] assess serverless performance regarding resource consumption and QoS. Cicconetti et al. [16] propose a theoretical model for distributed serverless computing in MEC but simplify crucial constraints such as latency due to different network types and lifecycle state transitions. A comprehensive set of open questions regarding the integration of serverless computing into MEC is outlined in [5], highlighting various challenges like the benefit of serverless flexibility or resource-heterogeneous MEC deployments. To this end, this paper applies an empirical approach to address and resolve these challenges, thereby bridging the gap between theory and practical implementation.

III. Testbed and Use Case

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

A. Testbed Setup

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

TABLE I: Network parameters for emulation. Jitter follows a normal distribution for all cases.

Criteria	Network		
	3G [17]	4G [18]	Wifi [19]
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

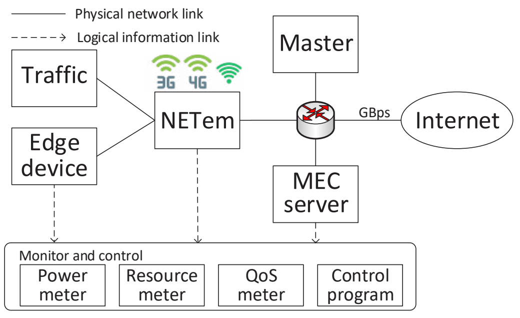


Fig. 2: Implementation of the testbed.

For the serverless framework, we use Knative [8], an open-source solution for serverless applications. 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 the Linux traffic control (TC) tool [20]. The TC tool emulates three distinct network types: 3G, 4G, and WiFi. The outdated 3G network is established to emulate realistic but bad network condition. The emulated network conditions correspond to data traces from literature summarized in Table I. Only the network associated with devices located to the left of NETem is constrained. Other components, such as the Master node and the MEC server, can communicate with each other and access the Internet at the full link speed of 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 traffic source. However, we consciously separate these roles to isolate the impact of non-targeted processes on the computing device in our testbed. The final component is the monitoring and control device, responsible for supervising the measurement procedures and to collect measurement data. Additional details about the testbed instruments are summarized in Table II.

B. Use Cases and Measurement Criteria

In our setup, we deploy the YOLO-v4 [21] object detection function as the smart IoT service. This application takes streaming video as the input and analyzes each frame to provide the object count detected within. Notably, this application exhibits versatility and widespread applicability across various practical scenarios, as surveyed in [22]. Within our study, this service is instantiated as a serverless function. Its instance can be created either at an edge device or an MEC server. When the traffic generator triggers a request to 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.

TABLE II: 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

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 [23], and it has been removed from the newest version of the platform [24]. 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 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.

IV. Use Case Evaluation

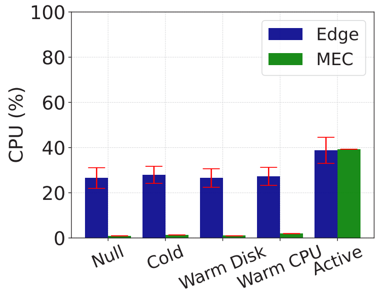
In this section, we provide the measurement outcomes for each state and process throughout the lifecycle of a serverless deployment within the MEC environment.

A. Resource Consumption in Multi-Access Edge Computing

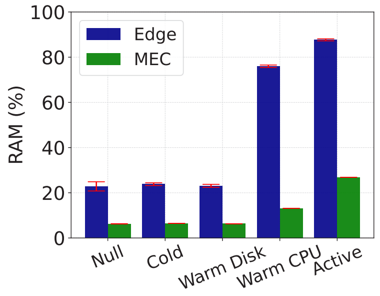
This section investigates serverless resource consumption by analyzing each state and process in the 4G-enabled MEC environment. Firstly, we investigate the different states' resource requirements in the following.

1) State's Resource Consumption: Figure 3a illustrates the CPU usage in different states of a serverless function placed at the edge device and the MEC server. 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 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.

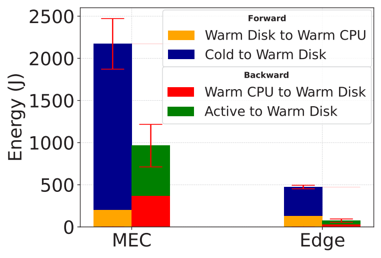
The RAM consumption shown in Fig. 3b follows a similar pattern as the CPU usage, except for the Warm CPU state,



(a) States' CPU usage



(b) States' RAM usage



(c) Processes' energy usage

Fig. 3: Lifecycle's resource usage under maximum load. Error bars are 90 % confidence intervals.

which consumes a notable amount of RAM. This allocation is used to reserve variables and libraries for the active instance, even before 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 addition, WiFi and 3G power consumption vary only marginally, with the most noticeable difference in the *Active* state for 3G due to its lower data rate leading to lower load. Consequently, we omit a detailed analysis of these minor variations in the following.

2) Resource Consumption of Processes: Transiting between different states involves resource allocation and deallocation aligned with the intended state [6]. Given this alignment, we will omit those details and focus on the energy consumption and latency associated with these transitions.

For energy consumption, we highlight only processes that consume considerable amounts. Figure 3c describes the four most energy-consuming processes. As depicted by the left to right arrows in Figure 1, *forward* means the sequence $\{$Null $\rightarrow$ Cold $\rightarrow$ Warm Disk $\rightarrow$ Warm CPU $\rightarrow$ Active$\}$, representing the resource allocation to create a serverless function. Reversely, *Backward* refers to the resource deallocation.

Overall, the forward processes consume considerably more energy compared to the backward processes. In detail, the *Cold to Warm Disk*, involving the download and the extraction of a container's image accounts for the majority of energy consumption. The instance creation, denoted as *Warm Disk to Warm CPU*, also contributes significantly during the forward processes. However, the energy consumption is also not negligible during the backward processes. 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 like the consumption trends of states, the energy consumption of processes remains relatively consistent using WiFi or 3G.

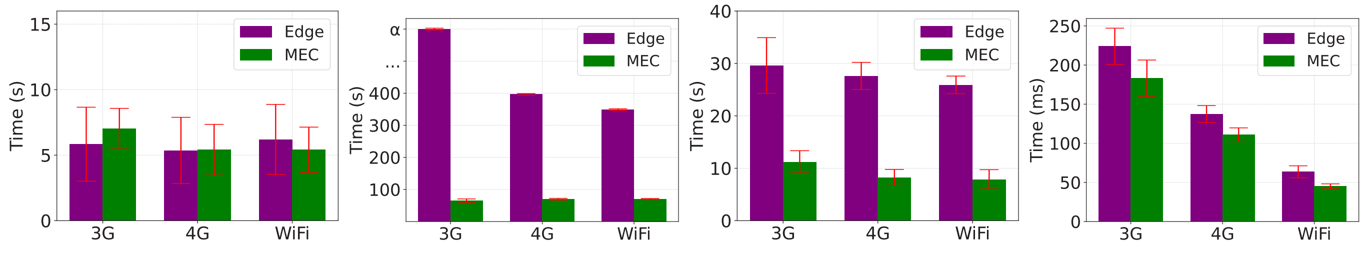
B. Serverless Latency in Multi-Access Edge Computing

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.

1) Latency at Different Locations: We illustrate each process in each separated subfigure in Figure 4. In the following, we first focus on the impact of function placement. The influence of the networks is discussed later in the paper.

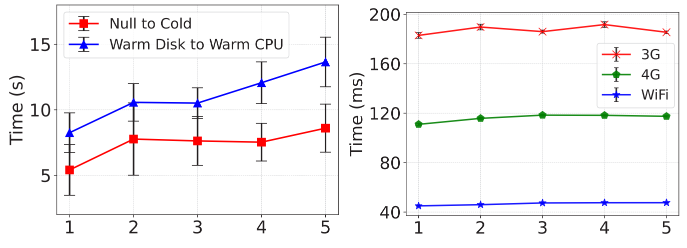
For the *Null to Cold* process in Figure 4a, only the service abstraction is generated at the Master node. Thus, it is not affected by the workers' capability. In Figure 4b and in Figure 4c, the *Cold to Warm Disk* and *Warm Disk to Warm CPU* processes at MEC outperform their counterparts at the edge. This is because the MEC server is not constrained by network types like the edge device, so image downloading takes much less time. Second, MEC is more powerful and enables faster task processing, such as image extraction and resource allocation. The latency when a request triggers a transition from *Warm CPU* to *Active* is shown in Figure 4d. Although this process is influenced only by the latency and not by any device capabilities, MEC outperforms the edge again. This contrasts with the consensus that proximity to the user leads to lower latency. We identified that existing serverless open-source solutions, such as Knative, do not adequately consider distributed environments like MEC. Consequently, networking services like gateways and load balancers are optimally positioned on the more powerful device, which is the MEC server in our testbed. This architectural choice leads to end-user requests always being directed to the MEC server first, regardless of the function's actual location. This challenge persists across our various results.

Combined with the analysis derived in Section IV-A, we can answer our first research question RQ1 as follows. *The MEC environment demonstrates the potential for intelligent deployments such as serverless functions. However, the limitations of less powerful edge devices in general complicate the hosting of serverless frameworks and complex ML tasks. Furthermore, it suffers from high latency caused by the non-edge-oriented design of the existing platform.*



(a) Null to Cold (b) Cold to Warm Disk (c) Warm Disk to Warm CPU (d) Warm CPU to Active

Fig. 4: Latency of each process in the serverless lifecycle placed at different locations. The x-axes indicate the function's location.



(a) Network-free processes (b) Warm CPU to Active

Fig. 5: Latency of lifecycle processes at the MEC server. The x-axes indicate the number of concurrent functions.

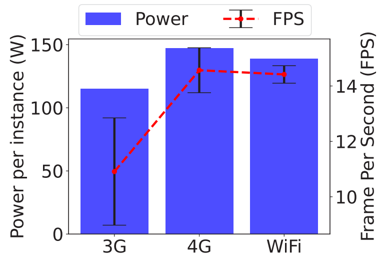


Fig. 6: Average power usage per function versus the output FPS at the edge device for different network types.

2) Latency using Different Networks at Edge Device: Focusing on the network influence illustrated by the different groups of bars in Fig. 4, we can see two groups of processes. The first group is not or only slightly affected by the network. This includes the Null to Cold process, as shown in Figure 4a, which is simply the creation of an abstraction at the Master node. Furthermore, it includes the Warm Disk to Warm CPU process, visualized by Figure 4b, which is slightly affected due to the system communication between the Master and the Worker where the process takes place. The other processes heavily rely on network quality, including *Cold to Warm Disk* and *Warm CPU to Active*. In particular, the image download failed during the *Cold to Warm Disk* process in a 3G network, as shown in Figure 4b, due to exceeding the system timeout. As a result, we could not record the value and mark it as α. Overall, enhanced network quality, characterized by low latency, minimal jitter, and higher bandwidth, consistently leads to lower latency across processes.

3) Latency at the Multi-Access Edge Computing Server: As shown previously, most of the function's processes at the MEC server are not significantly affected by the network type. Therefore, we focus on examining the relationship between load and latency, with load regards to the concurrently spawning number of instances. Figure 5a illustrates the latency of the two network-independent processes, with a varying number 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 5b. This case stands as an exception, being influenced by network characteristics. We notice that the 3G and the 4G network induce more than $100\,\mathrm{ms}$ latency, while WiFi maintains sub-$100\,\mathrm{ms}$. However, increasing the load shows no significant additional result. Given that all functions share the same image, the system downloads it only once, regardless of the number of instances deployed. Thus, the *Cold to Warm Disk* phase remains unaffected by the load.

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

C. Comparison of Power and Performance

To determine the efficiency of deploying serverless functions over MEC, we compare the power consumption per function and the average output FPS as shown in Figure 6 and in Figure 7. 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 6, edge device can accommodate only one instance with very low FPS, of around one, even under good network conditions (WiFi). On

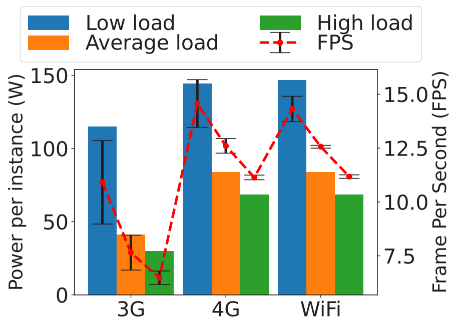


Fig. 7: Average power usage per function versus the output FPS at the MEC server for different network types. Load indicates the number of running instances: one for low, three for average, and five for high load.

the other hand, MEC performance, presented in Figure 7, 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 FPS. Hence, the limiting factor for the edge device is its computational capability rather than network constraints.

In terms of the network type, both 4G and WiFi exhibit comparable performance with the same load. In contrast, 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, scaling up the load leads to a performance loss of up to 40 % in terms of FPS, and the power consumption per function experiences a dramatic reduction of nearly threefold.

To this end, 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.*

V. Conclusion and Future Work

In this work, we comprehensively evaluated the deployment of serverless functions in MEC via a testbed we developed. Our results show that smart serverless deployments can be integrated into MEC, leveraging serverless flexibility and event-driven characteristics to reduce resource consumption and capitalize on cloud capabilities at the edge. However, the current serverless architecture requires significant resources and induces high latency on limited-capability edge devices. Additionally, energy costs and unstable performance due to variable MEC networks are challenges, especially for constrained edge devices. Therefore, future work should consider an edge-oriented serverless platform to eliminate latency issues and manage the lifecycle to exploit serverless benefits.

References

[1] Global Data, "Global IoT Market will Surpass the $1 Trillion Mark by 2024, Says Global Data," 2021, accessed: 2023-08-24. [Online]. Available: https://www.globaldata.com/media/thematic-research/global-iot-market-will-surpass-1-trillion-mark-2024-says-globaldata/

[2] L. S. Vailshery. IoT Connected Devices Worldwide from 2019 to 2023. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[3] M. Patel, Y. Hu, P. Hédé, J. Joubert, C. Thornton, B. Naughton, J. R. Ramos, C. Chan, V. Young, S. J. Tan, D. Lynch, N. Sprecher, T. Musiol, C. Manzanares, and U. Rauschenbach, "Mobile-Edge Computing Introductory Technical White Paper," White Paper, 2014.

[4] ETSI, "Mobile Edge Computing – A Key Technology towards 5G," White Paper, 2015.

[5] K. Nguyen, F. Loh, and T. Hoßfeld, "Challenges of Serverless Deployment in Edge-MEC-Cloud," 2023.

[6] K. Nguyen, F. Loh, T. Nguyen, D. Doan, H. Nguyen, and T. Hoßfeld, "Serverless computing lifecycle model for edge cloud deployments," in *2nd Workshop on Green and Sustainable Networking. IEEE*, 2023.

[7] Amazon. Serverless Computing - AWS Lambda - Amazon Web Services. [Online]. Available: https://aws.amazon.com/lambda/

[8] Knative. Serverless Containers in Kubernetes Environments. [Online]. Available: https://knative.dev/docs/

[9] Apache. Open Source Serverless Cloud Platform. [Online]. Available: https://openwhisk.apache.org/

[10] Kubernetes. Kubernetes: Production-Grade Container Orchestration. [Online]. Available: https://kubernetes.io

[11] R. Xie, Q. Tang, S. Qiao, H. Zhu, F. R. Yu, and T. Huang, "When Serverless Computing Meets Edge Computing: Architecture, Challenges, and Open Issues," *IEEE Wireless Communications*, 2021.

[12] I. Wang, E. Liri, and K. Ramakrishnan, "Supporting IoT Applications with Serverless Edge Clouds," in *Int. Conf. on Cloud Networking*, 2020.

[13] T. Rausch, W. Hummer, V. Muthusamy, A. Rashed, and S. Dustdar, "Towards a Serverless Platform for Edge AI," in *USENIX Workshop on Hot Topics in Edge Computing*, 2019.

[14] A. Glikson, S. Nastic, and S. Dustdar, "Deviceless Edge Computing: Extending Serverless Computing to the Edge of the Network." ACM, 2017.

[15] S. R. Chaudhry, A. Palade, A. Kazmi, and S. Clarke, "Improved QoS at the Edge Using Serverless Computing to Deploy Virtual Network Functions," *IEEE Internet of Things Journal*, 2020.

[16] C. Cicconetti, M. Conti, and A. Passarella, "Uncoordinated Access to Serverless Computing in MEC Systems for IoT," *Comp. Networks*, 2020.

[17] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute Path Bandwidth Traces from 3G Networks: Analysis and Applications," in *ACM Multimedia Systems Conference*, 2013.

[18] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond Throughput: A 4G LTE Dataset with Channel and Context Metrics," in *ACM Multimedia Systems Conference*, 2018.

[19] I. Grigorik, *High Performance Browser Networking*. O'Reilly Media, Inc., 2013.

[20] Linux. TC(8) - Linux Manual Page. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html

[21] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv preprint arXiv:2004.10934*, 2020.

[22] T. Diwan, G. Anirudh, and J. V. Tembhurne, "Object Detection using YOLO: Challenges, Architectural Successors, Datasets and Applications," *Multimedia Tools and Applications*, 2023.

[23] T.-V. Nguyen, N.-N. Dao, V. Dat Tuong, W. Noh, and S. Cho, "User-Aware and Flexible Proactive Caching Using LSTM and Ensemble Learning in IoT-MEC Networks," *IEEE Internet of Things Journal*, 2022.

[24] Knative. v1.10 release Knative. [Online]. Available: https://knative.dev/blog/releases/announcing-knative-v1-10-release/