Efficient Task Orchestration Including Mixed Reality Applications in a Combined Cloud-Edge Infrastructure

Luciano de S. Fraga[1], **Leizer de L. Pinto**[1], **Kleber V. Cardoso**[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)

lucianofraga@inf.ufg.br;{leizer,kleber}@ufg.br

Abstract. *Mixed Reality (MR) is establishing itself as one of the most prominent immersive applications, enabled by advanced computing and communication infrastructures that must deliver high throughput, low latency, and high reliability simultaneously. To meet the stringent requirements of MR applications, while balancing competition with less demanding workloads and varying operational costs, cloud and edge computing resources must be efficiently orchestrated. Despite extensive research on task offloading, key aspects such as accurately modeling MR demand and incorporating adequate infrastructure cost remain underexplored. In this work, we formalize the resource allocation problem and introduce LOTOS (Local Optimal Task Orchestration Solution), a novel approach that efficiently orchestrates multiple workloads, including MR applications. Through simulations using EdgeCloudSim, a load generator based on real MR application traces, and the real-world cloud and edge platform costs, we demonstrate the effectiveness of LOTOS. Compared to a widely cited approach in the literature, LOTOS achieves over 15% more successfully completed tasks while reducing costs by up to 8 times.*

1. Introduction

The future of wireless access networks introduces significant advancements that enable the deployment of services with highly demanding requirements in terms of resource consumption and quality of service (QoS). Among these, Mixed Reality (MR) applications stand out as a compelling use case, exemplifying the stringent demands of such services due to their high requirements for low latency and computational resources [Nowak et al. 2021, Wang et al. 2023, Toczé et al. 2020]. Beyond the inherent challenges posed by the application class itself, the increasing density of connected devices in wireless networks, driven by the proliferation of ubiquitous Internet of Things (IoT) devices, further complicates the challenge of maintaining the required quality parameters. Each mobile device presents unique characteristics and is constrained by limited processing power, memory, energy, and storage, making it impractical for many workloads to be processed locally on user devices. To address these limitations, the adoption of edge computing emerges as a key enabler for immersive applications powered by wireless technologies such as WiFi 6E/7/8. Edge computing brings computational resources closer to end users, reducing latency and alleviating the burden on resource-constrained devices, thus playing a critical role in meeting the stringent demands of MR and other immersive services.

The adoption of edge computing enables workloads to be executed on servers located at the edge of the network, offering an alternative to traditional cloud computing, where services rely on centralized servers in distant data centers. By bringing computational resources closer to the user, edge computing effectively meets the stringent requirements of services demanding low latency and high computational capacity. In

addition to ensuring low latency, edge computing also alleviates congestion in the access network, improving overall service quality. Furthermore, it helps conserve scarce resources on mobile devices, such as battery life, memory, and storage, enhancing efficiency and usability.

Despite the benefits of deploying services on edge servers, the combination of high workloads and the dynamic nature of these demands can lead to resource scarcity at the edge. In this context, it becomes crucial to prioritize services with more stringent QoS requirements during resource allocation. Additionally, when edge resources are insufficient, it is essential to identify tasks that can be offloaded to the cloud. Resource constraints and network congestion are not uncommon and can occur during peak hours of regular service operation. They are further exacerbated during social events that concentrate large populations in a short timeframe, such as sporting events or music festivals. These scenarios highlight the need for advanced resource allocation strategies that ensure QoS requirements are met while optimizing the use of the limited resources available on distributed edge servers.

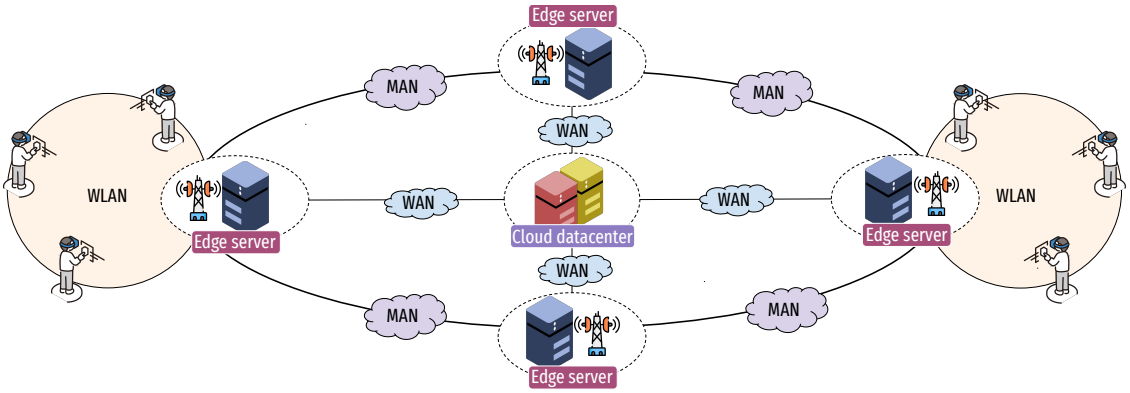


Figure 1. Edge/Cloud computing infrastructure.

In this context, we propose **LOTOS**, a framework designed to address the resource allocation problem by orchestrating a set of tasks (an abstraction to users workload) offloading the processing demand into virtual machines (VM) on the edge or cloud infrastructure (Figure 1). **LOTOS** employs a locally optimal approach, solving a portion of the problem iteratively to achieve efficient solutions. It is worth noting that, while the proposed solution leverages WLAN as the access interface to the edge infrastructure, it can be easily adapted for telecommunications networks, such as 5G. In this work we intend to: 1) verify the benefits of preemptively solving resource allocation for multiple tasks; 2) present a framework that considers not only users QoS requirements but also the cost of consuming infrastructure resources; 3) analyze the improvements achieved by **LOTOS** in comparison to existing solutions in the literature.

The remainder of this work is organized as follows: Section 2 discusses related works and highlights the comparative aspects between those solutions and **LOTOS**. In Section 3, we detail our solution, structured into three stages. Section 4 provides a comparative analysis of our approach, presenting a detailed description of the experimental scenario. Finally, Section 5 concludes the work and outlines directions for future research.

2. Related Works

In this section, we compare our solution with existing works in the literature that address resource allocation and task orchestration in cloud and edge infrastructures. Table 1 summarizes the main comparative aspects of these works. The first column lists the referenced papers, the second column describes the type of solution employed, where ILP stands for Integer Linear Programming problem, and the last four columns outline the objectives pursued by the respective studies. The acronyms **MF**, **MQ**, **OR**, and **MC** denote Minimize Failures, Maximize Quality of Service (QoS), Optimize Resource Utilization, and Minimize Cost, respectively.

Table 1. Related works characteristics.

Article	Solution approach	MF	MQ	OR	MC
[Sonmez et al. 2019]	Heuristic	✓	-	✓	-
[Toczé and Nadjm-Tehrani 2019]	Heuristic	✓	-	-	-
[Jamil et al. 2023]	Heuristic	✓	-	✓	-
[Peixoto and Azim 2023]	Machine Learning	✓	-	-	-
[Almutairi and Aldossary 2021]	Heuristic	✓	-	-	-
[Sonmez et al. 2021]	Machine Learning	✓	-	-	-
[Abdah et al. 2021]	Heuristic	✓	-	-	-
[Zheng et al. 2022]	Machine Learning	✓	-	-	-
[Theodoropoulos et al. 2023]	Heuristic	✓	-	-	-
LOTOS	ILP	✓	✓	✓	✓

The authors in [Sonmez et al. 2019] propose the use of fuzzy logic as a decision-making mechanism for orchestrating workloads between edge and cloud servers. The problem is addressed in two phases: first, a decision is made to select the candidate edge node to receive the workload. In the second phase, a comparison is conducted between the selected edge node and the central cloud node to determine where the workload will be processed. The proposed solution aims to minimize failures caused by insufficient network or processing resources, as well as issues related to mobility, such as asynchrony between user association and access points during *upload* and *download*. Additionally, the work analyzes the service time of tasks and the utilization of processing resources. The proposed model considers a three-tier infrastructure composed of multiple edge servers and a single cloud server.

The authors in [Toczé and Nadjm-Tehrani 2019] propose a solution named ORCH to address the task orchestration problem in scenarios involving non-stationary edge servers. This approach not only determines task placement but also optimizes the positioning of the servers themselves, ensuring that all requests are handled with minimal failures.

In [Jamil et al. 2023], the Belief Rule Base (BRB) technique is applied to address the high complexity and inherent uncertainties of the orchestration problem. This approach relies on pre-established decision rules to determine the orchestration of tasks across servers within a three-tier architecture. Similar to the methodology in [Sonmez et al. 2019], the problem is addressed in two phases, maintaining the same decision scope. The primary objective is to optimize the utilization of computational resources, minimize failures, and ensure compliance with latency requirements.

In [Sonmez et al. 2021], the authors propose a machine learning-based task orchestration model, structured in two phases. In the first phase, a classification model is employed to predict whether a task should be allocated to the edge or the cloud. In the second phase, a regression model is utilized to predict the service time for tasks allocated to the selected architecture, as determined in the first phase. The work considers Vehicular Edge Computing (VEC), where tasks can be allocated to either the edge or the cloud via Road Communication Units (RSUs) or through cellular networks, such as 5G.

Similar to the approach in [Sonmez et al. 2021], the authors in [Peixoto and Azim 2023] also explore a VEC scenario and tackle the orchestration problem using machine learning algorithms. The authors analyze and compare several regression models, including random forest, linear regression, and Support Vector Machine (SVM). However, unlike the work in [Sonmez et al. 2021], their model consists of a single phase. In this case, the proposed solution, which utilizes only a regression model to predict the service time of tasks allocated either at the edge or in the cloud, demonstrates more efficient results compared to its counterpart.

The authors in [Almutairi and Aldossary 2021] and [Abdah et al. 2021] employ fuzzy logic to make task allocation decisions, taking into account various application characteristics such as processing demand, network requirements, and latency constraints, as well as the level of resource utilization at both the edge and the cloud. In [Abdah et al. 2021], the authors propose a solution consisting of three modules. The identification module determines the most suitable edge server for task allocation. In the classification module, the task is categorized as either low-demand or high-demand. Finally, in the decision module, the parameters derived from the previous modules are fed into the fuzzy logic model, which decides whether the task should be allocated to the cloud server or to the edge server identified in the first module.

In [Zheng et al. 2022], Deep Reinforcement Learning (DRL) is applied as an orchestration solution aimed at minimizing the service time experienced by the user. The decision space includes the set of edge and cloud servers available for task allocation, with the reward or penalty function based on task service time to guide the agent's decisions. While the paper presents a model formalizing the addressed problem, it does not include an evaluation with an exact solution.

Almost all of the works presented (except for [Theodoropoulos et al. 2023]) utilize the EdgeCloudSim [Sonmez et al. 2017] tool to simulate the implementation of the proposed strategies. Among these, only [Zheng et al. 2022] presents a mathematical model formalizing the addressed problem, although the authors do not provide an evaluation based on this model. The majority of the strategies rely on fuzzy logic or machine learning models as orchestration solutions, meaning all the approaches offer non-exact solutions to the problem. Additionally, none of these works consider the infrastructure cost in the decision-making process. A further limitation across all the cited works is the decision being made for each task individually, without considering the overall system.

3. Model and Problem Statement

Subsection 3.1 introduces the system model for the task orchestration problem in an edge and cloud computing infrastructure. Subsection 3.2 provides a detailed explanation of the three-stage strategy designed to address this problem.

3.1. System Model

We consider that a user $u \in \mathcal{U}$ is connected to an access point $a \in \mathcal{A}$ in an infrastructure similar to the one shown in Figure 1. Each user generates a set of tasks, which are an abstract concept representing the workload unit composed of processing instructions, demand of memory and CPU, and the amount of data generated by an application. We define a VM as $v \in \mathcal{E} \cup C \cup \mathbb{0}$, where $\mathbb{0}$ is a machine located close to the access point to which user u is connected, $\mathcal{E}$ is a neighboring edge VM, and C is a VM in the cloud. The overall objective of the model is to allocate users tasks to VMs in order to maximize the number of tasks successfully executed in accordance with network, delay, processing, and memory constraints, while minimizing the cost of using the infrastructure.

3.2. Problem Formulation

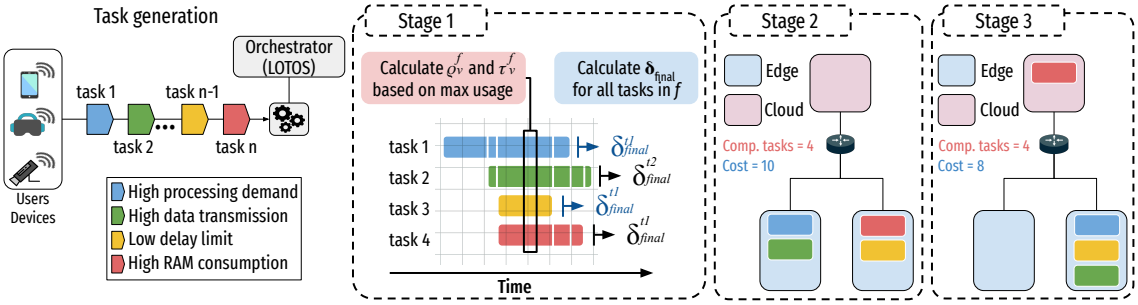


Figure 2. Local Optimal Task Orchestration Solution (LOTOS).

The proposed solution consists of three stages. In the first stage, we calculate the amount of resources consumed by a set of tasks allocated to a VM. Simultaneously, given the available resources, we determine the processing time of tasks sharing resources within the VM. In the second stage having as input the output of **Stage 1**, the objective is to maximize the number of tasks successfully executed by deciding where each task will be allocated. Finally, in the third stage, the goal is to minimize the monetary cost of processing the maximum number of tasks identified in **Stage 2**. Figure 2 provides an overview of the three-stage solution proposed in this work. The following subsections provide a detailed explanation of each of the three stages.

3.2.1. Stage 1 - Calculate configuration demand

In the first stage of **LOTOS**, we need to calculate the resource consumption and the time required to process each task in a given virtual machine (VM). To achieve this, we implement Algorithms 1 and 2, which perform the calculations for a specific set of tasks f allocated to a particular VM v. We define f as an element of the power set $\mathcal{F} = \mathbb{P}(\mathcal{T})$, representing a specific combination of tasks, where the resources available in VM v are shared among them. The outputs of these algorithms for all $f \in \mathcal{F}$ and $v \in \mathcal{V}$ will be used in the execution of stages 2 and 3.

The Algorithm 1 outlines the calculation of the processing time of all tasks $t \in f$ when assigned to a virtual machine v. In addition to the set f and VM v, the algorithm takes as input the CPU speed in Million Instructions Per Second (MIPS) and the number of cores of the VM (Ω^v, Φ^v), the demands of each task t (ω_u^t, ϕ_u^t) in Million Instructions (MI) and number of cores respectively, and the time at which each task completed the *upload*

Algorithm 1 ConfigurationFeatures

Require: $v, f, \Phi^v, \Omega^v, \omega_u^t, \phi_u^t, \delta_{initial}^{t,u} \quad \forall t \in f$
Ensure: Update $\delta_{final}^{t,u}$
1: $\mathcal{T}_{ord}^v \leftarrow \text{Sort}(\delta_{initial}^{t,u} \quad \forall t \in f)$.
2: $\delta_{current} \leftarrow \text{Min}(\delta_{initial}^{t,u} \quad \forall t \in f)$.
3: $k \leftarrow 1 \quad i \leftarrow \mathcal{T}_{ord}^v[k] \quad j \leftarrow \mathcal{T}_{ord}^v[k+1] \quad \mathcal{T}_{con}^v \leftarrow i$.
4: **while** $(k \neq |\mathcal{T}_{ord}^v|)$ **do**
5: $\quad \delta_{dif} \leftarrow \delta_{initial}^j - \delta_{initial}^i$.
6: $\quad$ UpdateWorkload.
7: $\quad \mathcal{T}_{con}^v \leftarrow \mathcal{T}_{con}^v \cup j \qquad k \leftarrow k+1 \qquad \delta_{current} \leftarrow \delta_{current} + \delta_{dif}$.
8: $\quad \rho_v^f \leftarrow \text{Max}(\rho_v^f, \text{CPU}(\mathcal{T}_{con}^v)) \quad , \quad \tau_v^f \leftarrow \text{Max}(\tau_v^f, \text{RAM}(\mathcal{T}_{con}^v))$
9: $\quad i \leftarrow \mathcal{T}_{ord}^v[k] \qquad j \leftarrow \mathcal{T}_{ord}^v[k+1]$.
10: **end while**
11: **while** $(\mathcal{T}_{con}^v \neq \emptyset)$ **do**
12: $\quad i \leftarrow \text{Min}(\omega_u^t \times \phi_u^t \quad \forall t \in \mathcal{T}_{con}^v)$.
13: $\quad cpt \leftarrow \frac{\Phi^v}{|\mathcal{T}_{con}^v| + |\mathcal{T}_{legacy}^v|} \qquad cap_u^i \leftarrow \frac{cpt}{\text{Max}(\phi_u^i, cpt)} \times \Omega^v \qquad \delta_{dif} \leftarrow \frac{\omega_u^i}{cap_u^i}$.
14: $\quad$ UpdateWorkload.
15: $\quad \delta_{current} \leftarrow \delta_{current} + \delta_{dif}$.
16: **end while**

($\delta_{initial}^{t,u}$). The algorithm output provides the time at which the tasks in f finish executing their entire workload. Initially, the tasks in f are sorted by the $\delta_{initial}^{t,u}$ values (line 1). The variable $\delta_{current}$ receives the most recent time among all tasks in f (line 2). The set $\mathcal{T}_{con}^v$ stores all tasks with residual workloads that compete for processing resources. Initially, $\mathcal{T}_{con}^v$ is populated with the most recently generated task(s) (line 3).

In the first loop (lines 4-9), we iterate through the list $\mathcal{T}_{ord}^v$ while updating $\mathcal{T}_{con}^v$ with the tasks that begin to demand computational resources. The variable δ_{dif} represents the time period during which the tasks in $\mathcal{T}_{con}^v$ compete for resources. This value is used to decrease the remaining number of instructions for each task, as well as to update the time and determine when a task is completed (Algorithm 2). Additionally, in line 8, we track the maximum CPU and RAM usage by comparing the most recent usage record with the amount consumed by the tasks in $\mathcal{T}_{con}^v$.

The Algorithm 2 only updates the remaining workload for each task t. cap_u^t stores the amount of processing capacity in MIPS available for task t generated by user u when the tasks in the set $\mathcal{T}_{con}^v$ are being processed simultaneously. At the end of the first loop, $\mathcal{T}_{con}^v$ contains the tasks that have not yet finished processing their workload. In the second loop (lines 11-15), we calculate the completion time of the task with the smallest remaining workload while simultaneously updating the residual workload of the other tasks. $\mathcal{T}_{legacy}^v$ represents the legacy tasks that are still being processed in the VM. Once $\mathcal{T}_{con}^v = \emptyset$, we will have completed the final processing time calculation for all tasks in f in the VM v.

3.2.2. Stage 2 - Maximize the number of tasks executed successfully

Equations 1-9 represent the problem to be solved in **Stage 2** of **LOTOS**. The objective, defined in equation 1, aims to maximize the number of tasks executed successfully by allocating them to any available VM within the infrastructure. The binary decision

Algorithm 2 UpdateWorkload

Require: $\mathcal{T}^v_{con}, \delta_{dif}, \delta_{current}, cap, \phi^t_u, \omega^t_u \ \ \forall t \in \mathcal{T}^v_{con}$
Ensure: Update $\mathcal{T}^v_{con}$, $\delta^{t,u}_{final}$ e ω^t_u

1: **for** $t \in \mathcal{T}^v_{con}$ **do**
2: $\quad cpt \leftarrow \frac{\Phi^v}{|\mathcal{T}^v_{con}|+|\mathcal{T}^v_{legacy}|} \qquad cap^t_u \leftarrow \frac{cpt}{\text{Max}(\phi^t_u, cpt)} \times \Omega^v.$
3: $\quad$ **if** $\delta_{dif} \times cap^t_u \geq \omega^t_u$ **then**
4: $\quad\quad \mathcal{T}^v_{con} \leftarrow \mathcal{T}^v_{con} - t.$
5: $\quad\quad \delta^{t,u}_{final} \leftarrow \delta_{current} + \frac{\omega^t_u}{cap^t_u}.$
6: $\quad$ **else**
7: $\quad\quad \omega^t_u \leftarrow \omega^t_u - (\delta_{dif} \times cap^t_u).$
8: $\quad$ **end if**
9: **end for**

variable x^f_v indicates the allocation of the task set f to the VM v, where $x^f_v = 1$ if the set f is allocated to VM v, and $x^f_v = 0$ otherwise. The function $A(x^f_v)$ returns the total numbers of tasks in the set f.

Each virtual machine, whether located at the edge or in the cloud, has processing capabilities defined by P_v. Additionally, the computational resource consumption when a set of tasks f is executed on a VM v is represented by ρ^f_v. Equation 2 establishes that the computational resources consumed by a task allocation to a virtual machine must not exceed the machine's capacity. Similarly, constraint 3 ensures that the amount of RAM memory demanded, denoted as τ^f_v, does not exceed the VM capacity T_v. The CPU and RAM consumption, represented by ρ^f_v and τ^f_v, are calculated in Algorithm 1, which records the maximum consumption of resources during task workload execution.

Equations 4-6 establish the capacity constraints for the WLAN, WAN, and MAN links, respectively. The WLAN link is utilized when a task is allocated to any VM within the edge infrastructure. The WAN link is employed when the task is assigned to a cloud-based VM, while the MAN link is activated for tasks allocated to neighboring edge VMs that are not co-located with the same access point. Each access point features dedicated WLAN and WAN communication links: the WLAN link connects users within its coverage area to the access point, and the WAN link connects the access point to the cloud infrastructure.

The parameters ζ^{wlan}_a and ζ^{wan}_a define the maximum number of users that can be supported by the WLAN and WAN links, respectively, and are provided as input values. The functions $W^{wlan}_{dev}(x^f_v, a)$ and $W^{wan}_{dev}(x^f_v, a)$ calculate the number of users transmitting data through the links at access point a, based on the decision variable x^f_v. Equation 6 establishes the usage limit for the MAN link, which is shared among all access points. The parameters ζ^{man}_{ul} and ζ^{man}_{dl} specify the maximum number of tasks that can be transmitted on the MAN link in *upload* and *download*, respectively. The function $W_{tasks}(x^f_v)$ represents the number of tasks utilizing the MAN link, according to the decision variable x^f_v.

Constraint 7 ensures that the delay experienced by each task remains below a predefined input threshold. The total delay comprises two components: the communication delay, which accounts for the time required to transmit a task to a VM and return the results to the user's device, and the processing delay, which reflects the time needed to complete the task's workload. The function $M(x^f_v, t, u) = 1$ if task t, generated by user u,

is included in the set f, and $M(x_v^f, t, u) = 0$ otherwise. The terms $\Gamma_{com}^{t,u,v}$, $\Gamma_{proc}^{t,u,v}$ and $\Gamma_{wait}^{t,u,v}$ represent the communication delay, the processing delay, and the waiting time before a task is send to the orchestrator respectively.

$$\text{maximize} \quad \sum_{f\in\mathcal{F}}\sum_{v\in\mathcal{V}} x_v^f \times A(x_v^f) \tag{1}$$

$$\text{subject to:} \quad \sum_{f\in\mathcal{F}}\sum_{v\in\mathcal{V}} (x_v^f \times \rho_v^f) \leq P_v, \tag{2}$$

$$\sum_{f\in\mathcal{F}}\sum_{v\in\mathcal{V}} (x_v^f \times \tau_v^f) \leq T_v, \tag{3}$$

$$\sum_{a\in\mathcal{A}}\sum_{f\in\mathcal{F}}\sum_{v\in\mathcal{E}} (x_v^f \times W_{dev}^{wlan}(x_v^f, a)) \leq \zeta_a^{wlan}, \tag{4}$$

$$\sum_{a\in\mathcal{A}}\sum_{f\in\mathcal{F}}\sum_{v\in\mathcal{C}} (x_v^f \times W_{dev}^{wan}(x_v^f, a)) \leq \zeta_a^{wan}, \tag{5}$$

$$\sum_{f\in\mathcal{F}}\sum_{v\in\mathcal{E}} (x_v^f \times W_{tasks}(x_v^f)) \leq Min(\zeta_{ul}^{man},\ \zeta_{dl}^{man}), \tag{6}$$

$$(\Gamma_{com}^{t,u,v} + \Gamma_{proc}^{t,u,v} + \Gamma_{wait}^{t,u,v}) \times x_v^f \times M(x_v^f, t, u) \leq \Delta_u^t, \quad \forall f \in \mathcal{F}, t \in \mathcal{T}, u \in \mathcal{U}, v \in \mathcal{V}, \tag{7}$$

$$\sum_{f\in\mathcal{F}}\sum_{v\in\mathcal{V}} x_v^f \times M(x_v^f, t, u) \leq 1, \forall t \in \mathcal{T}, u \in \mathcal{U}, \tag{8}$$

$$\sum_{f\in\mathcal{F}} x_v^f \leq 1, \forall v \in \mathcal{V}. \tag{9}$$

Finally, Equation 8 ensures that the model produces a single solution for each task t generated by user u, while Equation 9 guarantees that at most one set of tasks f is allocated to a VM v.

The communication time is calculated using the following equations:

$$\Gamma_{com}^{t,u,v} = \frac{\alpha_u^t + \beta_u^t}{R_{wlan}} \quad , \quad \frac{\alpha_u^t + \beta_u^t}{R_{wan}} + \Gamma_{prop}^{wan} \quad , \quad \frac{1}{\zeta_{ul}^{man} - \lambda} + \frac{1}{\zeta_{dl}^{man} - \lambda} + \Gamma_{prop}^{man}, \tag{10}$$

each representing the communication delay for the WLAN, WAN and MAN links, respectively. Here, α_u^t and β_u^t denote the task size during *upload* and *download*, respectively, while R_{wlan} and R_{wan} represent the per user bandwidth of the WLAN and WAN links. The term Γ_{prop}^{wan} corresponds to the propagation delay of the WLAN link. The third equation computes the communication delay for the MAN link, modeled as an M/M/1 queueing system. In this context, $\lambda = \sum_{f\in\mathcal{F}}\sum_{v\in\mathcal{E}}(x_v^f \times W_{tasks}(x_v^f))$ represents the total number of tasks transmitted through the MAN link, based on the allocation of tasks f to VM v. The term Γ_{prop}^{man} accounts for the propagation delay of the MAN link.

The processing time is determined using the following equation:

$$\Gamma_{proc}^{t,u,v} = \delta_{final}^{t,u} - \delta_{initial}^{t,u}, \tag{11}$$

where $\delta_{initial}^{t,u}$ is given as input, and $\delta_{final}^{t,u}$ is computed in Stage 1 using Algorithms 1 and 2.

3.2.3. Stage 3 - Minimize architecture cost

The solution presented in Stage 2 determines the destination for each task, aiming to minimize failures and optimize the utilization of storage, processing, and communication resources, while ensuring compliance with the quality requirements specific to the application. In **Stage 3**, the goal is to minimize the infrastructure implementation cost, without overlooking the benefits achieved by the solution developed in **Stage 2**.

Equation 12 defines the objective function for minimizing the total cost of using virtual machines for task processing. The cost c^v represents the monetary value per time unit that a VM incurs for processing a set of tasks. The function $T(x_v^f)$ denotes the total time the VM is utilized to process the complete workload of the tasks in f. Equation 13 ensures that the number of tasks executed without failure remains consistent with the result achieved in **Stage 2**. In this context, x_v^{f*} represents the decision variables corresponding to the optimal solution obtained in **Stage 2**.

$$\text{minimize} \quad \sum_{f \in \mathcal{F}} \sum_{v \in \mathcal{V}} x_v^f \times T(x_v^f) \times c^v \tag{12}$$

$$\text{subject to} \quad \sum_{f \in \mathcal{F}} \sum_{v \in \mathcal{V}} x_v^f \times A(x_v^f) = \sum_{f \in \mathcal{F}} \sum_{v \in \mathcal{V}} x_v^{f*} \times A(x_v^{f*}), \tag{13}$$

$$\text{Equations} \quad 2\text{-}9 \tag{14}$$

Complexity Analysis
In the worst-case scenario, the complexity of Stage 1 of LOTOS (S1) is given by $O(|\mathcal{V}||\mathcal{F}|O(A1))$, where $\mathcal{F}$ represents the power set with $|\mathcal{F}| = 2^{|batch\ size|}$, and $O(A1)$ denotes the complexity of Algorithm 1. A1 has a complexity of $O(|f|O(A2)+|f|O(A2))$, where f is an element of $\mathcal{F}$ and can be at most the batch size. The complexity of Algorithm 2, $O(A2)$, can be expressed as $O(|f|)$. Stages 2 and 3 of LOTOS are formulated as ILP problems, which are known to be NP-Complete [Karp 2010].

4. Simulation Results and Analysis

In this work, we perform an extensive evaluation of the performance of **LOTOS**, comparing it against three alternative approaches: a method that prioritizes the least loaded VMs (UTIL.), a solution that selects the VM with the lowest cost (MIN. COST), and a fuzzy logic-based approach (Fuzzy) proposed in [Sonmez et al. 2019]. For this evaluation, we use EdgeCloudSim, a network simulator built on CloudSim [Buyya et al. 2009] that enables the modeling of wireless communication links, characterization of diverse applications, creation of user mobility models, and customization of task orchestration strategies in a flexible manner. To represent MR applications, we employ the load generator presented by the characterization of an application called MR-Leo [Toczé et al. 2019]. The implementation was carried out using Java 21.0.5 and Python 3.10. In addition to the EdgeCloudSim framework, we utilize the optimization tool CPLEX (version 22.1.0) and docplex 2.23.221 to develop the optimization model. The complete code of this work is available in a github repository [1].

Unlike the solution presented in [Sonmez et al. 2019], we incorporate RAM demands and delay constraints as key factors in the decision-making process. Additionally,

[1] https://github.com/LABORA-INF-UFG/paper-LLK-2025

we account for the infrastructure cost when allocating resources in Stage 3. Our approach also considers resource allocation decisions for multiple tasks simultaneously. We define a batch size and a batch timestamp to represents the maximum number of tasks to be allocated together and the maximum time difference between the last and the first task in a batch respectively. This guarantees that tasks are not grouped into a batch with significantly different generation times, preventing any impact on the solution quality.

Table 2 presents the parameters used to characterize four distinct services [Toczé et al. 2020, Sonmez et al. 2019]. **MR 1** and **MR 2** represent the characterization of two services available in MR-Leo. In **MR 1**, the MR-Leo application captures a dynamic scene and generates a point cloud to represent the surface of the scene. In **MR 2**, a virtual element is overlaid on the captured scene and shown as output to the user. **APP 3** (health application), **APP 4** (computation-intensive application), and **APP 5** (infotainment application) represent other services that share resources with MR-Leo.

Table 2. Application characteristics.

	MR 1	**MR 2**	**APP 3**	**APP 4**	**APP 5**
Tasks arrival	33ms	$\lambda = 5(min)$	$\lambda = 3s$	$\lambda = 20s$	$\lambda = 7s$
Upload task size α^t (kB)	41	0.06	20	2500	20
Download task size β^t (kB)	41	0.06	1250	25	1000
Task delay limit Δ^t (ms)	33	66	100	500	1000
Task generator rates (%)	30		20	20	30
Task length ω_u^t (MI)	$\mathcal{N}(47\%)$ and $\mathcal{N}(53\%)$		3000	45000	15000
VM util. edge/cloud ρ_v^f (%)	6/0.6		2/0.2	30/3	10/1
CPU cores required ϕ_u^t (#)	3		1		
Task RAM usage τ_u^t (GB)	2.4 + 1.3		0 + 1.3		

Line 1 in Table 2 represents the time interval between each task generated by a user. For **MR 1** we have a constant inter-arrival of 33ms between tasks, in **MR 2**, the intervals follow a *Poisson* distribution, with the mean value of 5 tasks generated per minute. For **APP 3**, **APP 4** and **APP 5** the inter-arrival times follow an exponential distribution with the mean values of 3, 20, and 7 seconds between the tasks. Line 5 shows the task generation percentage for each service. In line 6, the number of instructions required to process each task is specified. For **MR 1** and **MR 2**, the task size is defined by a normal distribution. For 47% of the MR tasks, the mean is 155.62 MI with a standard deviation of 14.10, while for the remaining 53% of the tasks, the size follows a normal distribution with a mean of 322.38 MI and a standard deviation of 71.18. Line 9 defines the RAM usage, which is set at 1.3 GB to run the services without any connected devices, plus an additional 2.4 GB for each user device connected and using the **MR 1** or **MR 2** services.

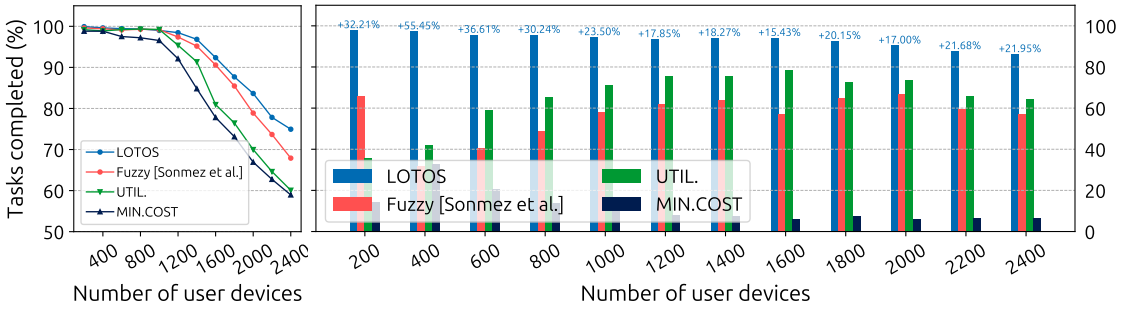
Table 3 provides additional parameters that define the experimental setup. The first two columns describe the characteristics of the virtual machines (VMs), including the number of CPU cores, RAM capacity, CPU speed, and hourly cost, based on values available on [AWS 2025]. The last two columns indicate the size of the instances considered in the experiments. Each experiment has a virtual duration of 10 seconds, with the number of user devices per instance varying from 200 to 2400. The batch timestamp specifies the maximum time interval between tasks within the same batch, while the batch size defines the maximum number of tasks grouped in a single batch.

Table 3. Experiments parameters.

Parameters	Value	Parameters	Value
Φ^v edge and cloud (#)	4/4/8/16	Experiment time (s)	10
T_v edge and cloud (GB)	16/32/64/128	VMs edge/cloud (#)	8/4
Ω^v edge/cloud (MIPS)	10000/100000	batch timestamp (ms)	5
c^v cloud ($/hour)	0.224/0.389/0.68/1.555	batch size (#)	6
c^v edge ($/hour)	0.134/0.262/0.403/1.048	User devices (#)	200-2400

Figure 3 illustrates the percentage of successfully completed tasks that did not encounter errors across two distinct scenarios. In the original scenario, RAM usage and delay limits for each task are not considered, and all edge VMs are configured with the same number of CPU cores. This setup was chosen to ensure a consistent evaluation environment, matching the one presented in [Sonmez et al. 2019]. As shown in Figure 3(a), our solution outperforms all other approaches under these conditions.

The second scenario represents an environment with varying VM capacities, reflecting the specifications of VMs in an AWS server (Table 3). All subsequent results in this work are evaluated based on this scenario. As shown in Figure 3(b), our solution achieves a significant improvement in successful task execution, with increases ranging from 15.43% to 55.45% across all instances. The Fuzzy approach, on the other hand, does not incorporate latency or RAM utilization into its decision-making process, which hinders its ability to allocate resources efficiently and avoid performance bottlenecks. In contrast, our solution accounts for these critical factors, enabling more effective resource allocation. Furthermore, our approach's ability to evaluate multiple tasks simultaneously during the decision-making process allows for a more accurate and proactive allocation strategy compared to the immediate, reactive nature of the Fuzzy method.



(a) Original scenario. (b) Second scenario.

Figure 3. Rate of tasks completed.

Figure 4 compares the types of errors observed during the simulation across all instances. In the Fuzzy approach, 24.41% of the errors were caused by latency limit violations, while 16% resulted from RAM memory overload. As previously mentioned, this approach did not consider these factors during decision-making, leading to a higher overall error rate. The UTIL. model performed better in this scenario, achieving 69.4% of completed tasks. In contrast, the MIN. COST model showed the worst performance, with only 8.8% of tasks completed. This inferior result confirms that selecting the cheapest machine is not the best strategy, as evidenced by the 73.6% of errors caused by RAM memory overload in this approach. The **LOTOS** model outperformed all other approaches,

achieving 91.5% of completed tasks. Errors due to delay limits were minimal, at just 5.4%. This is comprehensible, as **LOTOS** employs a locally optimal solution.

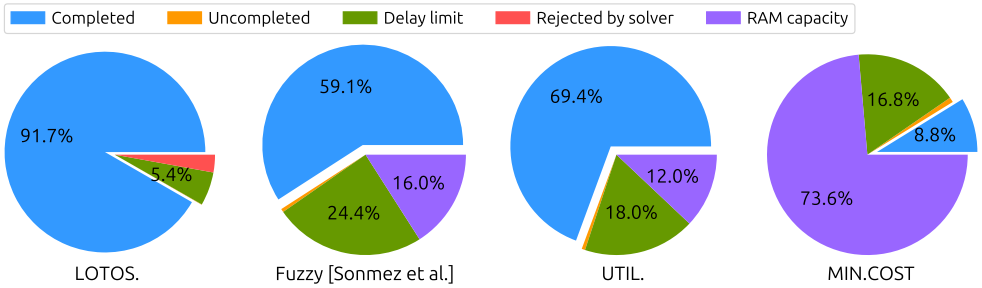
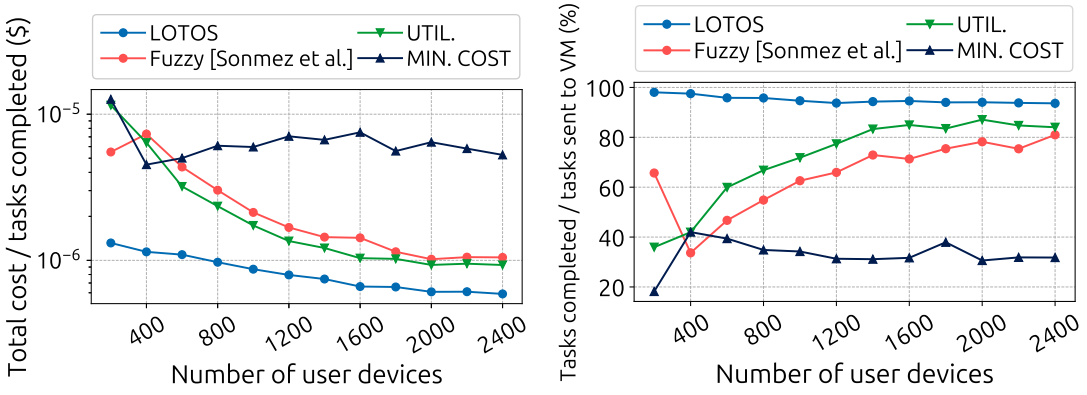


Figure 4. Rate of task errors.

We observe that 2.8% of all tasks in our approach were not completed due to being rejected during the optimization process. This indicates that, during Stages 2 and 3 of the **LOTOS** solution, the solver identified the impossibility of executing these tasks within the given constraints and opted not to allocate resources to them. While the other solutions reports errors during the simulation execution, our solution proactively identifies tasks that cannot be executed before the simulation begins. This preemptive decision-making ensures a more efficient utilization of resources, as they are not allocated to tasks that would fail to execute or whose results would not be received by users. Moreover, by incorporating infrastructure cost considerations, an aspect further detailed in the upcoming results, this efficient resource allocation plays a crucial role in reducing overall infrastructure expenses.

Figure 5(a) presents the cost per completed task, demonstrating that our solution achieves greater efficiency in resource allocation while completing a higher number of tasks without errors. Notably, our approach reduces costs by up to approximately 8 times compared to the UTIL. model, even with superior task completion efficiency. Figure 5(b) illustrates resource utilization efficiency, measured as the percentage of tasks that consume resources and are successfully executed. This efficiency is calculated by dividing the number of accepted tasks by the total number of tasks sent to a VM. With **LOTOS**, at least 90% of the tasks sent to a VM are executed successfully, whereas all other solutions exhibit efficiency levels below 90% across all instances.

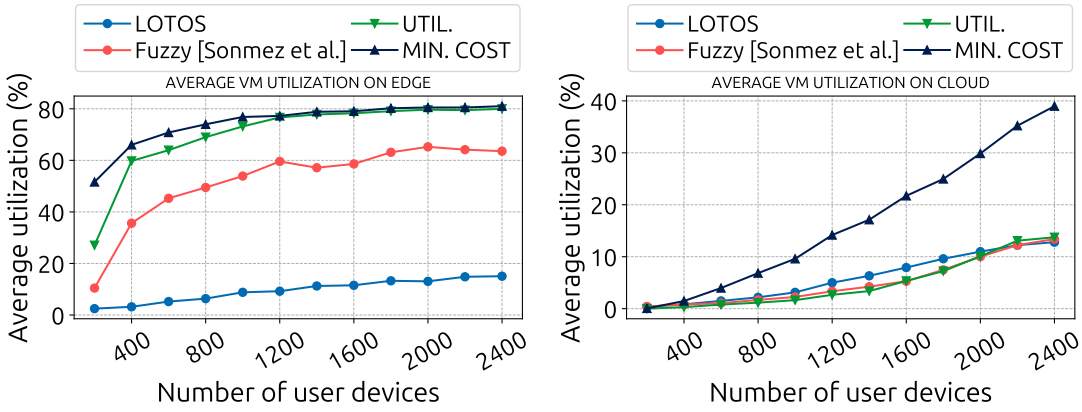


(a) Cost per task completed.

(b) Task execution efficiency.

Figure 5. Infrastructure cost and model efficiency.

Finally, the results presented in Figure 6 show the average resource utilization on the edge and cloud. As observed, our solution exhibits significantly lower utilization on the edge while remaining a strong competitor in cloud usage. This contributes to minimizing infrastructure costs, despite our solution achieving the highest number of completed tasks.



(a) Cost per task completed.

(b) Task execution efficiency.

Figure 6. Average VM utilization on edge and cloud.

5. Conclusion

In this work, we introduced **LOTOS**, a three-stage framework designed to address the task orchestration problem by maximizing the successful execution of tasks while minimizing infrastructure costs. The solution employs a locally optimal approach that considers the simultaneous allocation of resources for a batch of tasks. Our evaluation demonstrated that **LOTOS** significantly improves the rate of successfully completed tasks and achieves cost efficiency up to 8 times greater than the alternative approaches. Additionally, our solution ensures that at least 90% of tasks allocated to a VM are executed without errors, in contrast to the maximum efficiency of approximately 90% achieved by the comparative works. Furthermore, **LOTOS** optimizes infrastructure resource utilization and cost by identifying potential errors before the simulation phase, thereby preventing the allocation of tasks to VMs when their successful execution is not feasible.

In future work, we plan to address some of the time efficiency issues that were not fully explored in this study. We aim to propose heuristics or meta-heuristics that can decrease solution times while maintaining the overall efficiency of the solution. Additionally, we believe it would be beneficial to integrate Graph Neural Networks (GNN) into the model to predict tasks that will be generated in the future and solve them based on the output of the learning model.

Acknowledgments

This work was supported by CAPES, MCTIC/CGI.br/São Paulo Research Foundation (FAPESP) through the Smart 5G Core And MUltiRAn Integration (SAMURAI) project under Grant 2020/05127-2 and RNP/MCTI through the Brasil 6G project under Grant 01245.020548/2021-07.

References

Abdah, H. et al. (2021). Three-Tier Fuzzy-based Orchestration in MEC. In *IEEE Global Communications Conference (GLOBECOM)*, pages 1–6.

Almutairi, J. and Aldossary, M. (2021). A novel approach for IoT tasks offloading in edge-cloud environments. *Journal of Cloud Computing*, 10(1):28.

AWS (2025). AWS Calculator. https://calculator.aws/#/createCalculator, 2025-01-08.

Buyya, R. et al. (2009). Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In *2009 International Conference on High Performance Computing & Simulation*, pages 1–11.

Jamil, M. N. et al. (2023). Workload Orchestration in Multi-Access Edge Computing Using Belief Rule-Based Approach. *IEEE Access*, 11:118002–118023.

Karp, R. M. (2010). *Reducibility Among Combinatorial Problems*, pages 219–241. Springer Berlin Heidelberg, Berlin, Heidelberg.

Nowak, T. W. et al. (2021). Verticals in 5G MEC-Use Cases and Security Challenges. *IEEE Access*, 9:87251–87298.

Peixoto, M. J. P. and Azim, A. (2023). Design and Development of a Machine Learning-Based Task Orchestrator for Intelligent Systems on Edge Networks. *IEEE Access*, 11:33049–33060.

Sonmez, C. et al. (2017). EdgeCloudSim: An environment for performance evaluation of Edge Computing systems. In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 39–44.

Sonmez, C. et al. (2019). Fuzzy Workload Orchestration for Edge Computing. *IEEE Transactions on Network and Service Management*, 16(2):769–782.

Sonmez, C. et al. (2021). Machine Learning-Based Workload Orchestrator for Vehicular Edge Computing. *IEEE Transactions on Intelligent Transportation Systems*, 22(4):2239–2251.

Theodoropoulos, T. et al. (2023). Graph neural networks for representing multivariate resource usage: A multiplayer mobile gaming case-study. *International Journal of Information Management Data Insights*, 3(1):100158.

Toczé, K. et al. (2019). Performance Study of Mixed Reality for Edge Computing. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, UCC'19, page 285–294, New York, NY, USA. Association for Computing Machinery.

Toczé, K. et al. (2020). Characterization and modeling of an edge computing mixed reality workload. *Journal of Cloud Computing*, 9(1):46.

Toczé, K. and Nadjm-Tehrani, S. (2019). ORCH: Distributed Orchestration Framework using Mobile Edge Devices. In *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–10.

Wang, H. et al. (2023). A Survey on the Metaverse: The State-of-the-Art, Technologies, Applications, and Challenges. *IEEE Internet of Things Journal*, 10(16):14671–14688.

Zheng, T. et al. (2022). Deep Reinforcement Learning-Based Workload Scheduling for Edge Computing. *Journal of Cloud Computing*, 11(1):3.