Nimbus

Towards Latency-Energy Efficient Task Offloading for AR Services

Cozzolino, Vittorio; Tonetto, Leonardo; Mohan, Nitinder; Ding, Aaron Yi; Ott, Jorg

DOI
10.1109/TCC.2022.3146615

Publication date
2022

Document Version
Accepted author manuscript

Published in
IEEE Transactions on Cloud Computing

Accepted Author Manuscript. Link to published article (IEEE): https://doi.org/10.1109/TCC.2022.3146615

Nimbus: Towards Latency-Energy Efficient Task Offloading for AR Services

Vittorio Cozzolino, Leonardo Tonetto, Nitinder Mohan, Aaron Yi Ding, Jörg Ott

Abstract—Widespread adoption of mobile augmented reality (AR) and virtual reality (VR) applications depends on their smoothness and immersiveness. Modern AR applications applying computationally intensive computer vision algorithms can burden today's mobile devices, and cause high energy consumption and/or poor performance. To tackle this challenge, it is possible to offload part of the computation to nearby devices at the edge. However, this calls for smart task placement strategies in order to efficiently use the resources of the edge infrastructure. In this paper, we introduce Nimbus — a task placement and offloading solution for a multi-tier, edge-cloud infrastructure where deep learning tasks are extracted from the AR application pipeline and offloaded to nearby GPU-powered edge devices. Our aim is to minimize the latency experienced by end-users and the energy costs on mobile devices. Our multifaceted evaluation, based on benchmarked performance of AR tasks, shows the efficacy of our solution. Overall, Nimbus reduces the task latency by $\sim$4$\times$ and the energy consumption by $\sim$77% for real-time object detection in AR applications. We also benchmark three variants of our offloading algorithm, disclosing the trade-off of centralized versus distributed execution.

1 Introduction

Since the advent of consumer mobile devices equipped with multiple sensors and powerful chipsets, multimedia applications have garnered increasing interest amongst smartphone users. A recent study reports that the mobile AR adoption currently stands at 32%, where 54% of the respondents use mobile AR at least once per week and 36% percent several times per week [7]. Despite the increasing popularity of the technology, most current mobile AR applications often offer poor user perceived performance. The reason for this is two-fold. Firstly, object recognition and detection algorithms are a bottleneck for AR [97] as the front-end devices are often insufficiently equipped to execute them with acceptable latencies for the end user [1, 23]. Secondly, extended usage of such applications results in high power consumption, which leads to significant battery drain and overheating [35, 77].

Edge computing allows applications developers to accelerate their services' performance by offloading computationally intensive tasks to nearby powerful machines instead of the distant cloud datacenter. Latency critical applications operating on mobile devices, such as AR/VR, benefit most from the availability of the edge as they can utilize more powerful hardware, in addition to on-board processors, without traversing long paths to the cloud [88, 91]. As shown in previous research, such approaches not only allow smartphones to run multimedia applications and games with better visual quality [17, 21, 30, 34, 41, 80, 81, 93], but also enable older mobile devices (provided they are equipped with the required spatial sensors) to support such applications in the first place.

Unlike other driving applications for edge computing (e.g. smart homes), real-time multimedia applications impose much stricter constraints on offloading computations at edge devices. Since such applications need to incorporate tightly-coupled user interactions, they operate under strict delay thresholds imposed by the human vestibular system – bordering between 75ms for online gaming and 250ms for telemetry [66]. In practice, requirements for seamless interaction between the physical world and the virtual overlay are estimated to be much lower, $\sim$7ms [10, 25]. Currently, a modern smartphone can run object detection in $\sim$200ms per frame using an optimized model [78], which is some orders of magnitude off from the strict requirements of AR applications. Preserving loss of smoothness and excessive delays in applications relying on virtual environment is paramount to prevent phenomena such as motion sickness [66]. Additionally, AR/VR applications are power-hungry and can quickly drain the phone's battery [9]. The growing demand for higher precision deep learning models and increased immersiveness of the augmented experience can cost even more battery power. Chen et al. [28] show that a smartphone can spend significant portion of its battery capacity while running a mobile-optimized object recognition service. Pairing this workload with client-side rendering, network communications, and running specific AR application logic can reduce the expected battery life even further.

Considering the complexities levied by deep learning based real-time applications, it is challenging to exploit a nearby edge infrastructure in a scalable manner. Moreover, while the *cloud* has potentially unlimited resources, the same cannot be assumed for the *edge* computing paradigm. In fact, the latter is by definition distributed across multiple edge networks and hence associated with considerable heterogeneity in bandwidth and compute resources [61]. On the other hand, recent large-scale measurement studies have shown that despite the significant growth in cloud infrastructure, the network latencies from users to nearest cloud datacenters exceed the strict operational boundaries of AR applications

- *V. Cozzolino, L. Tonetto. N. Mohan and J. Ott are with the Technical University of Munich, Germany.*
- *A.Y. Ding is with the Delft University of Technology, Netherlands*

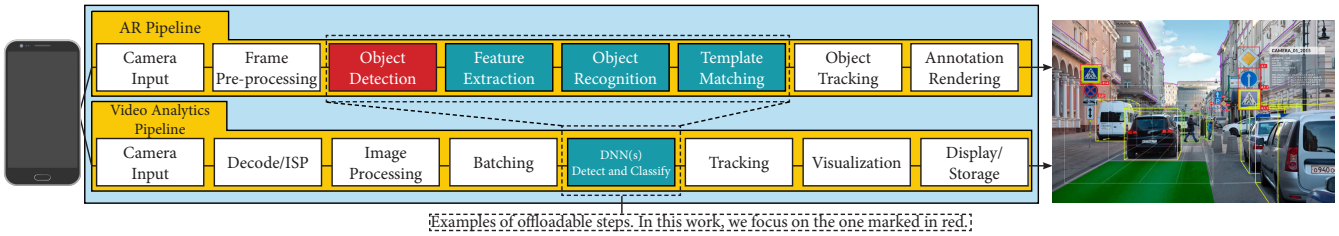

Fig. 1: Mobile applications requiring deep learning steps.

almost globally [31, 32, 36]. As a result, we see *edge-cloud interplay* as key to extend cloud computing reach outside of datacenters, and enhance its services by leveraging an infrastructure closer to the end-users [66, 87]. We believe that effective application offloading is a crucial problem for edge-cloud computing that must be addressed when thinking at scale. For that, selecting an appropriate offloading candidate must be at the core of maximizing user satisfaction, as allocating multiple users to an already overloaded edge node can negatively impact an AR application's performance [28].

To summarize, the **motivation** behind our work is boosting the performance of mobile applications that use DNNs (as shown in Figure 1) by offloading part of their execution pipeline to the edge-cloud infrastructure. The ultimate goal is to improve the quality of experience and enable potentially new classes of applications which have strict latency constraints (such as real-time mobile VR). We approach the problem from a system design perspective and proceed by using an algorithm for resource provisioning to measure the effectiveness of our architecture.

Contributions. In this paper, we present *Nimbus*, a real-time task offloading system designed to determine an optimal task placement strategy. We aim at reducing the *latency gap* afflicting the execution of real-time deep learning models required by AR and similar applications by making use of resources offered at the network edge, at scale. We select and support the execution of mobile-optimized, object detection convolutional neural network (CNN) for AR applications, as shown in Figure 1. This shows also the pipeline for live video analytic applications which programmatically share core components of AR/VR applications and are becoming the solution to many safety and management tasks [95]. The design principles of Nimbus are devised to address three crucial constraints of target applications: (1) *latency* as a primary measure of the application QoS, (2) *battery consumption* which defines the extent of the user's QoE, and (3) *task coordination* as the role of the infrastructure in orchestrating, load balancing and distributing computation based on the users' demands. Nimbus aims at minimizing the overall mobile-to-edge latency while avoiding increasing battery consumption. Additionally, Nimbus's offloading policy ensures a balanced load distribution across the edge nodes participating in the infrastructure. Our contributions in this paper are as follows:

- We benchmark the performance of different classes of edge devices to understand their support towards real-time object detection for mobile AR.
- We devise a multi-tier edge-cloud infrastructure and propose a best-effort resource provisioning algorithm addressing the problem of serving multiple users competing for heterogeneous resources. Overall, our approach reduces task latency by $\sim$4$\times$ and the energy consumption by $\sim$77% for real-time object detection.
- We develop an edge infrastructure simulator[1] to evaluate the performance of Nimbus against other related solutions. From an empirical analysis based on extensive measurements in real testbeds, we extract the parameters of the simulator to closely mimic the realistic operations of edge devices and core network latencies.
- We develop and evaluate several variants of Nimbus reflecting both centralized and distributed execution of the task placement algorithm.

2 Related Work

The intuition of offloading computationally intensive tasks from mobile devices towards powerful servers has been explored vastly in the past decade. Originally, the offloading procedure targeted powerful datacenters in the cloud [39, 42]. With the rise of edge computing, the status quo changed drastically with new possibilities to mitigate the most prominent drawback of cloud offloading: *latency*. In fact, the introduction of cloudlets and edge envisioned a collaborative computational infrastructure where intensive tasks could be offloaded to nearby edge microservers, thus saving on access latency [20, 79]. Moreover, edge computing can also help in reducing energy consumption of mobile devices. For example, with Voltaire [27] it is proposed to perform code offloading to enables resource-constrained devices to leverage idle computing power of remote resources.

Nevertheless, edge nodes have limited computational resources, limiting the number of clients that can be served at the same time. Approaches based on offloading to the nearest edge-cloud can lead to situations where too many clients are allocated to the same node, competing for limited resources. Multiple works have focused on solving a similar problem by using either hierarchical edge-cloud architectures or load balancing among edge-cloud [24, 29, 47, 58, 62, 63, 90, 92]. In particular, MCDNN [45] developed a compiler together with a runtime scheduler to balance between accuracy and resource consumption by reasoning about on-device/cloud execution tradeoffs, while Markov decision processes [46] were used for VMs load management to reduce energy

1. Code and dataset are available here https://github.com/vitcozzolino/nimbus.

consumption in datacenters. A similar approach was proposed by Tan et al. [84] to minimize the expected response time, where tasks uploaded from mobile device are sent to an edge-cloud infrastructure and scheduled by an online job dispatching algorithm. While their method is limited – assuming a server can only process one job at a time – we instead consider parallel execution of multiple jobs. Other approaches have focused on reconfiguration of edge-clouds [51], specifically on how to optimize the placement of cloudlets in a given network. The approach of using a hierarchical edge-cloud infrastructure has been proposed already by Tong et al. [85] to efficiently handle the peak load and satisfy the requirements of remote program execution. Recent work from Braud et al. [26] introduces a task allocation algorithm based on a latency model leveraging multipath computation to offer multiple resources in parallel. The key difference from the these approaches is that our system tackles the problem of *parallel* tasks execution offloaded to the same edge device while they focus on sequentially placed workloads. Additionally, previous solutions focused only on latency (computational and/or communication) without factoring in mobile energy consumption in the offloading decision. Finally, many scheduling algorithms translate task complexity in the number of CPU cycles required for its execution eventually combined with other parameters such RAM, disk, and bandwidth [38, 53, 98]. We instead focus on GPU workloads and their performance variance with overlapping tasks — a parameter which is seldomly explored.

While most of the previous work aimed to minimize mobile task execution time, we focus specifically on AR application offloading [57, 89]. By doing so, we gain a clear understanding of how and where a task should be offloaded since we are aware of the inherent requirements of such applications. Our scheduling algorithm focuses primarily on improving the perceived performance for the mobile user. Similar work has been done for visual applications offloading in the past. LAVEA [96] is a system built on top of an edge computing platform, which offloads computation between clients and edge nodes, to provide low-latency video analytics at places closer to the users. The work closest to ours is [72] – a framework that ties together front-end devices with more powerful backend servers to support complex deep learning tasks. However, unlike our work, the authors do not consider a multi-tier edge infrastructure and scenario where multiple users are competing for the resources offered by the infrastructure.

For mobile-cloud offloading, some work has been conducted in the past for optimizing DNNs. Kang et al. [52] and Xia et al. [94] identified how to optimally slice a model to offload only a part of it to the cloud in order to minimize either latency or energy consumption. DynO [16] is a distributed inference framework addressing several challenges, such as device heterogeneity, varying bandwidth and multi-objective requirements. Key components that enable this are its novel CNN-specific data packing method, which exploits the variability of precision needs in different parts of the CNN when onloading computation, and its novel scheduler that jointly tunes the partition point and transferred data precision at run time to adapt inference to its execution environment. Our work is inspired by those studies and strives to characterize the problem in a multi-tenant environment where resource contention is the primary issue.

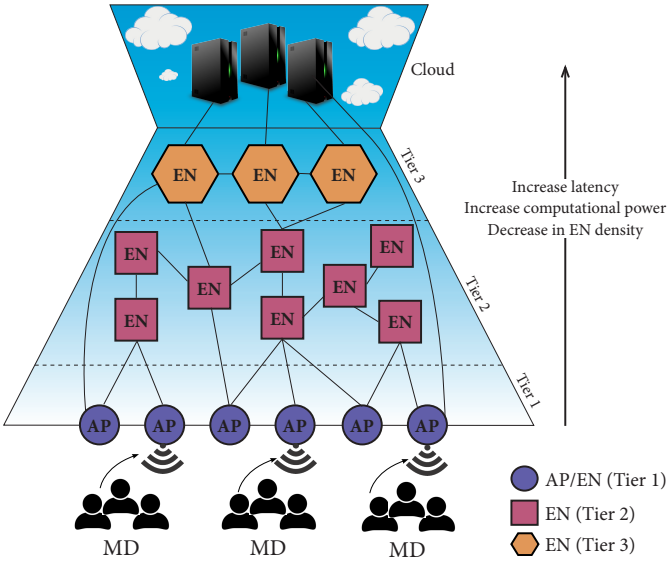

Fig. 2: Multi-tier edge-cloud infrastructure.

Recapping, Nimbus differentiates from the aforementioned research works in many ways. It tackles the problem of parallel task execution instead of makespan optimization (sequential). Our offloading solution revolves around a joint latency and mobile battery optimization procedure with a focus on GPU workloads and their scaling properties. Finally, Nimbus performance is rooted in a set of real measurements gathered from devices which are part of our envisioned edge-cloud infrastructure.

3 System Overview

Figure 2 shows the *entities* in our system – mobile devices (MD) and edge nodes (EN) interacting over the network. While the former interact with the infrastructure as users of AR applications, the latter are responsible for handling tasks offloaded by the MD. In our case, an MD is a battery-powered mobile device that can offload part of its computation to the edge-cloud infrastructure. We assume a hierarchical edge architecture where compute and caching capabilities of EN increase with increasing distance from the MD. Nodes in different (logical) layers of the edge network can be accessed via ad-hoc connections or gateways [33, 56, 60, 65, 67, 83].

The design of our edge computing infrastructure is inspired by networks like Eduroam[2]. The deployment of Eduroam is widespread as it can be found outside academic facilities, e.g., libraries and study centers. While such a network (currently) only offers Internet access to clients, we acknowledge its capabilities to support an edge computing infrastructure due to the presence of multiple connected networked resources capable of running computations on behalf of the connected users. We logically divide the network into three layers – each one offering different capabilities and, as we approach the core of the infrastructure,

2. https://www.eduroam.org

latency and computational capacity of the resources increases. Conceptually, the architecture proposed by Tong et al. [85] and Mohan and Kangasharju [64] come closest to ours and we use them as point of reference in our system design.

Tier One Edge Nodes (T1-EN). The outer-most layer (denoted by blue circles in Figure 2) is a set of augmented access points (AP) or base stations with minimal compute capabilities. We assume these APs to be either equipped with (or directly connected to) an embedded device with low-end GPUs, e.g. NVIDIA Jetson Nano or Intel NCS2. Resources in this layer act as entry points to the network, offering limited computation in addition to standard routing and connectivity functionalities.

Tier Two Edge Nodes (T2-EN). T2-EN (denoted with squares Figure 2) form the second layer of our multi-tier edge cloud infrastructure. Logically these devices can be viewed as backbone routers co-located close to T1-EN. However, unlike T1-ENs, T2-ENs posses more computational power and network bandwidth that allows them to serve multiple users in parallel. An example of T2-EN resources in the real world is a mid-range micro-server equipped with a discrete GPU.

Tier Three Edge Nodes (T3-EN). The core of our architecture comprises of T3-EN (shown as orange hexagons) that are powerful servers equipped with multiple GPUs, offering the most significant computational power of all layers. The capabilities of T3-EN are analogous to traditional cloud datacenters, both in terms of the number of users that can be served in parallel and network bandwidth connecting servers within the layer. However, due to their proximity to the network core, the network latency incurred to access the resources in this layer is the highest amongst edge infrastructure.

We consider a system where a mobile device hosting an AR application can offload component tasks in the pipeline (e.g. those requiring deep learning) to the edge infrastructure. Considering the inherent heterogeneity that exists in the infrastructure — different hardware capabilities, network latency to server, task requirements etc. — an effective task offloading strategy is required ensuring that the application performance meets the required expectations. Additionally, we assume that ENs in our system are managed resources and can communicate/exchange details regarding their current processing load with other ENs. This assumption roughly resembles the current state of resource management in cloud datacenters and it allows our task offloading algorithm (presented in the following section) to have fresh information regarding the edge network state.

4 Task Offloading at the Edge

We consider a system where a controller estimates the feasibility of offloading a task proposed by a mobile device to the edge infrastructure. Since our objective is to showcase the effectiveness of our offloading solution, we start by considering a centralized controller located in the cloud. Later in the paper (§6), we design a distributed and hybrid variant of our offloading mechanism and compare the operational differences of all approaches. Figure 3 shows a high-level concise workflow representing the MD-Controller interaction.

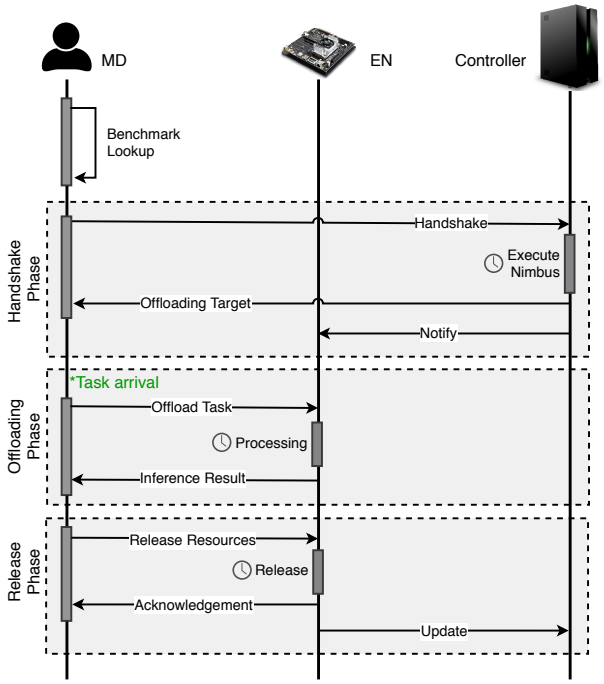

Fig. 3: MD-Controller Workflow.

The infrastructure is composed of **N** interconnected and heterogeneous ENs, which, based on their computing capacity, can serve several concurrent tasks. An MD can offload its task via T1-EN, which act as gateways to the infrastructure.

Before entering the *handshake* phase, the MD performs a one-time procedure called benchmark lookup. Normally, games and other multimedia applications run benchmarks to estimate their runtime performance in order to tune and set configuration parameters. Similarly, there are tools to profile deep learning models on mobile devices [50]. In our model, we assume that benchmarked results for each MD are uploaded to a repository that is looked-up by the system to identify MD's capabilities. Afterwards, the *handshake* procedure begins and the MD exchanges with the infrastructure controller its requirements in terms of deliverable performance (in FPS and battery consumption). In the *offloading* phase, the MD connects to the network to offload and it receives a list of offloading candidates from the controller obtained by running Nimbus (details about the algorithm logic will be provided in §5). Then, the MD will interact with the selected EN until required by the underlying application. Finally, in the *release* phase, the resources booked for the MD on the EN are released, and the controller is notified. The Nimbus offloading decision is based on minimizing deep learning based task latency (*i.e.*, maximizing FPS) as it directly affects the QoE for mobile AR applications.

Offloaded Tasks. As shown previously in Figure 1, AR/VR applications (especially games) can be decomposed into subroutines executed at each rendering step [97]. Some of these steps are not tied to the application logic and are perfect offloading candidates. Let us take the example of tracking-by-detection principle [19, 55, 71] for object tracking. The principle requires that the object is detected in the first and all subsequent frames. The object is tracked simply by

TABLE 1: List of parameters used by the algorithm.

Term	Description	Unit
d_i	Amount of data transferred by the *i-th* MD	KB
BW_{ij}	Bandwidth between *i-th* MD i and *j-th* EN	Mbps
TET_j	Inference time on the *j-th* EN	ms
TEC_i	Local energy execution cost for *i-th* MD	mJ
q_j	Queuing time at *j-th* EN	ms
w	Transmission module power	mJ/ms
ϵ_t	Latency threshold	ms
ϵ_b	Energy budget	J/s
RTT_m	RTT matrix	ms
κ, α, β	Additional coefficients (described in § 5)	—

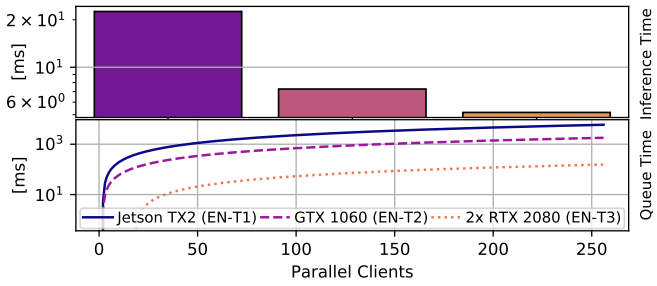

Fig. 4: Inference and queue time for the three EN tiers.

associating detection results to form target trajectories. This is a necessary component in all AR applications where smooth integration with real world is paramount. While tracking requires sophisticated application logic to interpolate objects positions across frames, detection is oblivious to past executions and depends only on the latest frame. Therefore, object detection is a prime candidate for offloading to the edge. In practice, a stream of pictures can be sent by the MD towards the target EN for processing. Even if the EN becomes unresponsive, the MD can switch to executing the task locally so that the underlying offloading process is transparent to the end-user who would experience no interruptions in the service. We consider each task submittable to the infrastructure as *atomic* (i.e., indivisible and uninterruptible). In this work, we focus on stateless tasks that are resilient to the loss of connectivity due to their independence from past transactions. However, our solutions proposed in this paper can be extended to stateful tasks as well with proper synchronization mechanisms. That, we leave them for future work as hereby we concentrate our efforts on the offloading strategy and algorithm formulation.

Our problem formulation assumes that MDs offload tasks to the system in bulks, which translates into a constant, worst-case arrival rate. This allows us to devise a solution that does not assume any prior knowledge about the offloaded task makespan nor use it to optimize the decision making process. It is not realistic for an MD to know in advance for how long the user will run the application (*e.g.*, minutes to hours). The only information available are inference time of the DNN task and its energy cost (estimated during the benchmarking phase shown in Figure 3). Therefore, we optimize the resources allocation in a *maximum concurrency scenario* – where all the MD are concurrently using the infrastructure and all resources, from network bandwidth to compute, must be shared.

In a practical scenario, tasks can have different complexity and requirements. For simplicity, we select a class of tasks for which we provide execution time distributions for the device executing them. We used the NVIDIA Triton [14] suite to benchmark MobileNetv2, a common CNN-model central to image classification tasks, with an increasing number of clients. We run benchmarks on three device types, each representative of the different tiers of our multi-tier edge infrastructure. Figure 4 shows the results we gathered in our experiments and specifically the inference and queue time for different EN tiers. More details will be discussed in §6.

Objective 1: Minimize Latency. The total task latency consists of transmission time Lt_{ij} between the i-th mobile device and the j-th edge node, plus the execution time Le_{ij} of the required tasks at the j-th node. Transmission time depends on the network bandwidth BW_{ij} and on the amount of data d_i sent by a mobile device. Furthermore, this communication delay can be negatively affected by multiple clients interacting with an EN if they share the same access medium (e.g., WiFi). Hence, a fair queuing best-effort communication model is assumed where each client connecting to an EN perceives a connection bandwidth equal to R/N, where R is the total offered data rate and N the number of active users.

Execution time represents the amount of time an MD has to wait in the processing queue before its request can be served, i.e the time to execute a task (TET_j) plus the GPU queuing time (q_j) on the j-th EN. Queue time can grow substantially depending on the EN's capabilities and the number of concurrently served clients. Figure 4 shows inference and queue time for each edge device tier and the number of users concurrently using the device. As expected for T1-EN, the queue time increases with the number of served clients due to the limited hardware capabilities of devices in this tier. On the other hand, powerful discrete GPUs found in expensive workstations can handle many more clients with a minimal queue time penalty. Another key insight from Figure 4 is that unlike inference time, queue time is heavily influenced by the number of parallel users and is a primary variable to model highly concurrent scenarios. Moreover, ENs equipped with powerful GPUs incur a queuing penalty only after concurrently serving many MDs, as shown for the T3-EN in Figure 4. Eventually, this leads to a point where even a powerful EN can not meet the QoE requirements of the MDs.

Objective 2: Reduce battery consumption for the MD. When mobile phones receive or transmit data, they consume energy depending on the network bandwidth and the amount of data to be transferred. Additionally, in real scenarios, wireless mobile devices often experience high variances in link quality [36, 68], directly affecting the data transfer latency and the final energy consumption. When offloading or accessing cloud resources, it is important to take into account the additional delay introduced by the network load pattern as they change throughout the day [59]. Therefore, network conditions for mobile devices experience high variance, and narrowing down to a single energy consumption model for all kinds of mobile devices in all network conditions is very challenging. In this paper, we build on top of previous work from Xia et al. [94] and Kang et al. [52] to express the energy

Device	Name	Inference	
		Energy Cost	Time
OnePlus 5T	Mobile_A	182 mJ	154 ms
OnePlus 3	Mobile_B	318 mJ	116 ms
Redmi Note 4X	Mobile_C	268 mJ	190 ms

TABLE 2: Mobile inference time and energy cost for MobileNetv2.

cost of transferring data B_t as a function of the transmission module power w and the overall transmission time L_t as shown below:

$$L_t = \frac{d}{BW} + RTT \tag{1}$$

$$B_t = L_t \times w \tag{2}$$

where *d* is the amount of transferred data, *BW* the upload bandwidth and *RTT* the network round-trip time.

We follow Xia et al. [94] and define three classes of mobile devices, each one with different hardware resources and power consumption profiles. We benchmark the performance of all three device classes for executing MobileNetv2. For Mobile_A and Mobile_B class, we use a single, CPU core while for Mobile_C we use 8 CPU cores. The energy cost and inference time achieved by all classes is shown in Table 2. In all cases, no model partitioning was applied. Also, the amount of energy spent to execute inference locally on the mobile device allows us to compare the cost of offloading the task against running it locally. While many contributions model both network transfer and mobile inference energy cost [43], we favor the approach described above due to its comprehensiveness and precise results — especially for the object detection task we focus on in our study.

Mathematical formulation. Assume that the *i*-th task is executed by *j*-th EN, the task latency and battery consumption incurred by the device can be formalized as:

$$L_{ij} = (Lt_{ij} + Le_{ij}) = \left[(\frac{d_i}{BW_{ij}}) + RTT_{ij}\right] + (TET_j + q_j) \tag{3}$$

$$B_{ij} = Bt_{ij} = Lt_{ij} \times w \tag{4}$$

We ignore the downlink cost for the energy consumption calculations as we assume it to be negligible when compared to the uplink, especially for object detection applications. While the input can be an image of arbitrary size, the output are bounding boxes of comparatively smaller in size for which the network transmission has a negligible energy cost. Therefore, when a task is offloaded, both its latency and mobile energy consumption are affected by the process of communicating with the edge infrastructure. In other cases, the task is running locally and its execution latency and energy consumption are described in Table 2.

Based on the system described above, we define the task assignment problem as *selecting an EN for assigning a task to minimize latency (L) and battery consumption (B) for the mobile device*. The problem translates into a multi-objective optimization problem with two objective functions in the form of $\min g(L(\vec{x}), B(\vec{x}))$ with $\vec{x} \in X$ and X the space of feasible decision vectors. In our case, we focus on identifying a set of Pareto optimal solutions which, by definition, cannot be improved in any of the objectives without degrading at least one of the others.

To solve for both latency and battery consumption, we make use of an approach called *scalarizing*. Scalarizing is an *a priori* method that allows us to formulate a single-objective optimization problem such that optimal solutions to it are Pareto optimal solutions to the original multi-objective optimization problem [49]. In our case, it would lead to the following reformulation of the problem: $\min g(L(x), B(x), \phi)$ with $x \in X_\phi$ and X_ϕ set depending on the vector ϕ. Of the multiple scalarization techniques, we adopt the ϵ-constraint method [37] to reformulate the multi-objective optimization problem by just keeping one of the objectives and restricting the rest within user-specified values (which fits our scenario). Based on the system described before, the offloading problem demands us to identify the best EN to run a user submitted task to minimize the experienced task latency while not violating the stated constraints. Mathematically, let $x_{ij} \in \{0, 1\}$ denote the case when the *j*-th EN serves the *i*-th device. We express the ϵ-constrained latency minimization problem as follows:

$$\min \sum_{i=1}^{N} \sum_{j=1}^{M} x_{ij} L_{ij}(p) \tag{5}$$

$$\text{subject to} \sum_{i=1}^{N} x_{ij} = 1, \forall j \in M, \tag{6}$$

$$L_{ij} \leq \epsilon_t, \forall j \in M, \tag{7}$$

$$B_{ij} \leq \epsilon_b \equiv TEC_i, \forall j \in M, \tag{8}$$

$$x_{ij} \in \{0, 1\}, \forall i \in N, \forall j \in M \tag{9}$$

where N and M are the set of mobile devices and EN, respectively, and with $\mathbf{p}$= $\langle d_i, BW_{ij}, TET_j, RTT_{ij}, q_j \rangle$ vector containing part of the parameters shown in Table 1. Equation 5 is our objective function. Equation 6 and 9 limit each MD to offload its task to as single EN, at most. Equation 7 and 8 are formalization of the latency and energy consumption constraints limiting the feasible solution space.

Constraints. The ϵ_t represents a predefined latency threshold after which offloading computation does not benefit the mobile device. The value covers both the transmission time and remote execution of the task. Depending on the mobile devices' requirements, ϵ_t can be a different value reflecting the specific user or application needs. Therefore, the threshold value depends on many factors, e.g., FPS requirements of the offloaded task. ϵ_b represents the battery consumption threshold, exceeding which offloading the task becomes too expensive in terms of energy. Fundamentally, ϵ_b depends solely on the cost of running the task locally (TEC_i) on the *i*-th device. This constraints are a function of the MD capabilities. For example, a powerful MD will have a much lower value for ϵ_t and, potentially, ϵ_b as it can complete locally its task quickly and efficiently (in terms of energy cost).

5 Algorithm

Following the ϵ-constrained approach in §4, we are able to re-construct our optimization problem in convex form

Algorithm 1: Nimbus allocation algorithm.

Input : Refer to Table 1.
Output: Best offloading target for the *i-th* MD.

```
// Warmup
E⃗N_r ← FilterAndMinimize(AP, RTT_m, ε_t)
E⃗N ← LookAheadLoad(E⃗N_r, κ)
// Core
for EN_j in E⃗N do
    L_ij = (Lt_ij + Le_ij) =
      [(d_i / BW_ij) + RTT_ij] + (TET_j + q_j)
    B_ij = Bt_ij = (Lt_ij × w)
    if L_ij ≥ ε_t or B_ij ≥ ε_b then
        Drop(EN_j, E⃗N)
    end
end
if E⃗N ≠ ∅ then
    for EN_j in E⃗N do
        return arg min[α * L_ij/ε_t + β * load_j/maxload_j]
    end
else
    // Failover
    cloud ← FindClosest(ε_t)
    if L_cloud ≤ ε_t and B_cloud ≤ ε_b then
        return cloud
    end
end
return ∅
```

Src \ Dst	T1-EN				T2-EN				T3-EN		
	EN_0	...	EN_{n-1}	EN_n	EN_0	...	EN_{n-1}	EN_n	EN_0	...	EN_n
$T1-EN_0$	c	...	...	...	...	...	...	...	...	...	...
...	...	c	...	...	...	...	...	...	...	...	...
$T1-EN_{n-1}$	...	...	c	...	...	...	...	...	...	...	...
$T1-EN_n$	...	...	...	c	...	...	...	...	...	...	...

TABLE 3: RTT matrix structure.

that we solve using a meta-heuristic. The adopted search strategy for our meta-heuristic is inspired by the *hill climbing* algorithm [3] that is widely used due to its effectiveness and simplicity in different convex optimization problems (*e.g.*, artificial intelligence) for which it can provide the optimal solution [74]. Algorithm 1 describes Nimbus task offloading approach.

Nimbus operation is divided into three phases: *Warmup*, *Core*, and *Failover*. The *Warmup* phase identifies a list of ENs that are accessible from the AP the device is connected to and are the best candidates to offload computation. In the *Core* phase, the algorithm calculates the latency and battery cost for offloading to each EN in the list using the formulation described in §4. Afterwards, it selects the best EN based on the balance-ensuring allocator. In the *Failover* phase, if the algorithm failed to find a suitable EN for offloading the task, it looks for a cloud server that best satisfies the latency and energy consumption constraints of the task.

Warmup Phase: To start off, Nimbus identifies a list of EN candidates for offloading the task. The function *FilterAndMinimize* extracts the set of ENs reachable from the AP to which the mobile device is connected. For reducing the search space, Nimbus filters out all ENs for which the network latency or the queue time is already greater (or equal to) the maximum threshold ϵ_t for the *i-th* mobile device. Subsequently, *LookAheadLoad* removes those EN candidates which are already close to their critical mass and serving another MD would violate the ϵ_t constraint. In fact, whenever we offload a task to an EN, the queue time increases for all the other tasks. As we can quantify this *domino effect* (discussed in §4), it is possible to use the MD experiencing the highest network latency as a reference point. If, for such device, we violate the task latency constraint, that EN is excluded from the list of viable offloading targets. Algorithm 2 describes the procedure in details. The parameter κ controls the search space by setting an upper bound to the number of ENs we want to consider. In our evaluation, κ will be used as a tradeoff parameter between convergence time and the solution's goodness. The output of the *Warmup* phase is a list of ENs that are passed to the next phase of the algorithm.

Algorithm 2: LookAheadLoad procedure.

Input : $\vec{EN}$, exploration coefficient κ, ϵ_t.
Output: List of compatible EN.

```
compatibleEN = ∅
for EN_j in E⃗N do
    if devicesList_j ≠ ∅ then
        mnl = arg max deviceNetworkLatency_j
        if size(devicesList) ≪
          maxServableDevices_j(ε_t − mnl) then
            compatibleEN ← EN_j
        end
    else
        compatibleEN ← EN_j
    end
end
if kappa ≡ 0 then
    return compatibleEN
else
    return randomSet(κ, compatibleEN)
end
```

Core Phase: As the name suggests, this phase is the core of the algorithm as it identifies the best offloading target by solving the minimization problem defined in §4. For each of the candidate ENs collected by the *Warmup* phase, Nimbus calculates the execution latency and battery consumption for offloading the task. We then use these estimates in the optimization step to identify which ENs respect the latency and battery constraints and avoid overloading the EN. This step is necessary for an effective task offloading at the edge as any new mobile device allocated to an EN impacts the QoS of all the other device being served by that EN. We assume that MDs do not change their requirements after being offloaded. If, in case they do, the device needs to resubmit the updated requirements triggering a new schedule by the algorithm. The α and β coefficients strike a balance between minimizing the latency for the MDs and avoiding infrastructure overload. If latency optimization is the only objective for the infrastructure's orchestrator, it can easily

achieve it by setting β to zero. In our evaluation, we set α to 0.7 and β to 0.3 to strongly favor latency optimization rather than balancing the infrastructure load. Other combinations can be used depending on the specific optimization goals and on the infrastructure capacity. Additionally, $load_j$ and $maxload_j$ represent the current and maximum load in terms of devices for the *j-th* EN, respectively. We calculate the latter using an inverse formula of the queue time growth, which we omit for brevity.

Failover Phase: The final phase of Nimbus is optional as it is only reached if the algorithm is unable to find any suitable candidate in the edge infrastructure that can meet the mobile device requirements. In this phase, *FindClosest* identifies the best datacenter (in terms of network RTT) for offloading the task. We do not assume any prior knowledge of the compute and hardware capabilities of the target datacenter. Instead, we assume constant execution latency for the cloud, making network RTT the main discriminating factor.

Finally, if the *Failover* step fails, the mobile device fallbacks to local execution and exits the scheduling algorithm.

6 Measurement and Evaluation Setup

To evaluate Nimbus's performance in realistic settings, we conduct several experiments and measurements to collect data concerning multiple variables of the algorithm. In this section, we explore and analyze all facets of our algorithm, namely *network latency*, *inference and queuing time*, and *specifications of MD and T1-EN*. Note, however, that we do not simulate or model network flows. From the network perspective, we elevate our point of view so that all the consequences of routing queues, path selection, and network connection fluctuations are reflected solely by the network RTT. We delve deeper into the consequences of our choice in § 8.

Network latency. As mentioned in § 3, we target an academic network infrastructure like Eduroam. At the time of writing, no network latency datasets were available for such a network. Nevertheless, to provide a meaningful distribution of the network latency across different layers of the infrastructure, we followed two approaches. The first approach focused on measuring network RTTs targeting some of our devices connected to the Eduroam infrastructure. We performed measurements from three vantage locations: overseas (connecting USA to Europe), from a different city (∼20 miles away), and directly connected in the same subnet. By analyzing these data, we generated three probability distributions, one for each EN tier.

We utilized two publicly available RTT datasets from two p2p-based networks: Seattle [11] and PlanetLab [8]. The dataset is publicly available at [4]. In order to assign network RTT to each EN, we identified three latency classes through k-nn clustering and subsequently generated the respective distributions, shown in Figure 5. The distributions were then used to generate relative RTT matrices (Table 3) that we feed to our solver. The row and column of the matrix represent an AP and EN in the network respectively. The values of the matrix represent the network RTT to reach any of the EN from an AP. As the probe's data are anonymized, we do not have information about the relative distance of the nodes or their location.

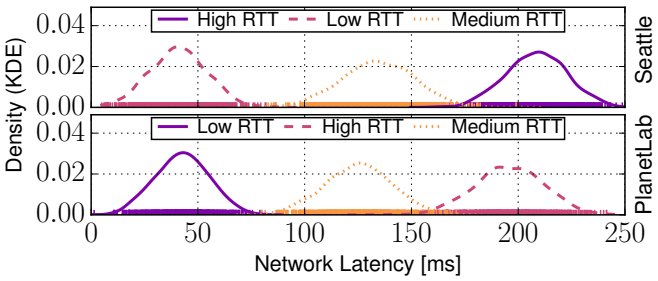

Fig. 5: Latency distributions for selected datasets.

Since our results from Seattle and PlanetLab datasets were almost similar, we only estimate latencies within our edge infrastructure using numbers from the Seattle dataset in § 7. As described in the *Failover* phase, MDs are allowed to connect to a cloud server if the performance offered by the edge network is not satisfactory. To estimate user latency to the cloud, we utilize our large-scale `ping` measurements from 3200+ RIPE Atlas probes [82] to 101 datacenters operated by seven major cloud providers globally. Our measurements over five months resulted in ≈3.2M datapoints spanning several GBs [31]. We make our dataset publicly-available at [40].

Inference and queuing time. To measure the computational cost of the task, we selected three different devices: an NVIDIA Jetson TX2, a laptop with an NVIDIA 1060 GTX, a micro-server with 2x NVIDIA 2080 RTX. We used the NVIDIA Triton [14] suite to benchmark MobileNetv2 with an increasing number of clients. Finally, as shown in Figure 4, we extracted the inference and queuing time. While the former remains constant regardless of the number of users, the latter instead, grows quasi-linearly with the number of clients. Note that this also depends on the amount of model instances loaded into the memory, as GPUs with more available VRAM can host more models in parallel, effectively boosting the overall performance by being able to concurrently serve more clients in parallel. T3-EN nodes have plenty of VRAM but this is not the case for T1-EN which might only be able to load concurrently a handful of models.

MD and T1-EN setting. The MDs are assigned hardware specs based on § 4. For simplicity, we uniformly distribute the total mobile devices across the three available hardware specs. The ratio of APs that are also T1-EN nodes is variable and depends on the experiment we run. However, for each AP, the maximum nominal Wi-Fi bandwidth is set to 300 Mbps. We assume that all devices connected to an AP experience the same connection quality apart from the effective bandwidth. Additionally, we do not account for any transmission-related issues that could negatively affect the signal.

We extrapolate data from the publicly accessible Leibniz-Rechenzentrum (LRZ) dataset [5, 6] to assign a location to each AP in the edge-cloud network plus their respective loads in terms of connected MDs. We extracted nine months' worth of network association data of public buildings and networks from the LRZ dataset. This contains over 4500 access points scattered across ∼450 buildings. The data are aggregated in 15 minutes slices, which we use as MD-batches

in our system (see § 4). We partitioned the dataset in different approaches, described further in the following section. We also compare the performance of several variants of our algorithm in the evaluation and discuss the tradeoff between convergence speed and efficiency of Nimbus.

7 Results

The results presented in this section cover two parts: (*i*) performance gain on MD and (*ii*) algorithm capability. We ran multiple experiments in different conditions (summarized in Table 4), highlighting different characteristics of our algorithm. Due to space constraints, we select a set of scenarios to showcase out system capabilities.

(A) Scalability & Performance. We first analyze the effective task latency and energy benefits[3] of Nimbus for processing tasks offloaded by MDs. We select four combinations of edge infrastructure and MDs, plus we set the required FPS threshold to 15 (frame interval ∼66.6 ms). We select four configurations where the number of connected MDs are 500, 1000, 2000, and 4000. Figure 6 depicts distributions of total task execution time and saved energy (per 1 second, or 15 frames) for 100 simulation iterations.

Even in the worst case (left panel of Figure 6), the expected task latency achieved by Nimbus is ∼**2**× lower than running it locally on the fastest MD in our dataset (see Table 2 in § 4). From a performance standpoint, this offloading strategy can boost deep learning based applications and increase the quality of experience for its end-users. As the number of MDs increases, the performance proportionally decreases. With more congestion and tasks offloaded, the delivered performance drops, as multiple MDs use the same EN and influence each other's execution time by increasing the overall queuing time. This saturation behavior is mirrored by the MDs allocation ratio. Figure 7 shows the percentage of mobile devices served by the edge infrastructure, for four different configurations of ENs shown in Table 4-(A). With an increasing number of users, the edge resources tend to saturate more quickly, forcing most of the mobile devices to run their computation locally or utilize the cloud. We find that, with the largest infrastructure used in our experiments (constituting 4000 MDs), roughly 75% can offload to the edge. Conversely, only 25% utilized the edge in our smallest infrastructure configuration.

Task offloading also allows MDs to save energy (right panel of Figure 6), reducing the power consumption in all cases. These results are significant as battery consumption is hugely relevant for high user satisfaction and retention [99]. Offloading tasks from more modern phones will lead to lesser energy savings due to their more efficient hardware components, decreasing the battery cost for running deep learning tasks. However, our results show a non-trivial margin of gain in offloading using Wi-Fi to the edge infrastructure. Even if we consider the most power-hungry smartphone in our dataset and the average energy saving in the worst-case, Nimbus still consumes ∼**77%** less battery. Note that using a mobile connection (*e.g.*, 4G) alongside task offloading leads to different results, which we discuss in the next experiment.

3. We calculate energy benefits by comparing the energy cost for offloading the task to running it locally at the MD.

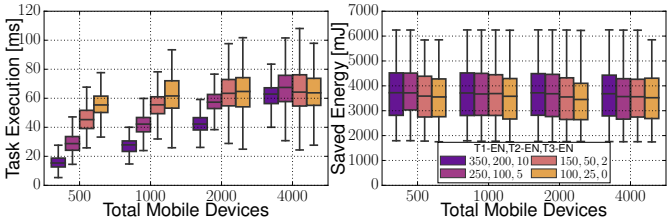

Fig. 6: Task latency and energy saving in various setups.

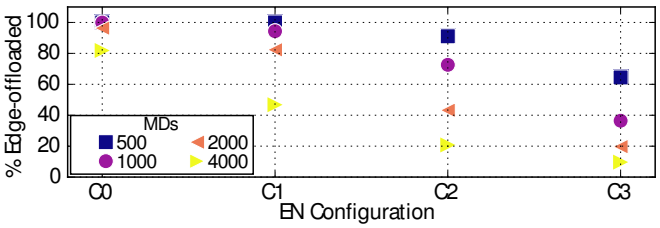

Fig. 7: Fraction of task offloaded to the edge (see Table 4).

Takeaway 1. The offloading strategy of Nimbus can boost deep learning based applications and increase the perceived performance for its end-users. The expected task latency achieved by Nimbus is ∼**2**× lower compared to the fastest MD in our dataset. Additionally, MDs consume up to ∼**77%** less battery when offloading with Nimbus.

(B) Full dataset. For this test, we run our algorithm on the entire nine-month LRZ dataset but limited to the top five most-populated buildings. Additionally, we set a minimum threshold of 30 MDs to simulate a reasonable load on the infrastructure. We fix the other parameters to values shown in Table 4. This experiment provides a broader view of the algorithm performance over an extended period with a fixed-sized edge infrastructure.

The time-series in Figure 8 shows the task execution latency (top) and MD density (bottom) for the entire nine-month period. To obtain these results, we progressively feed

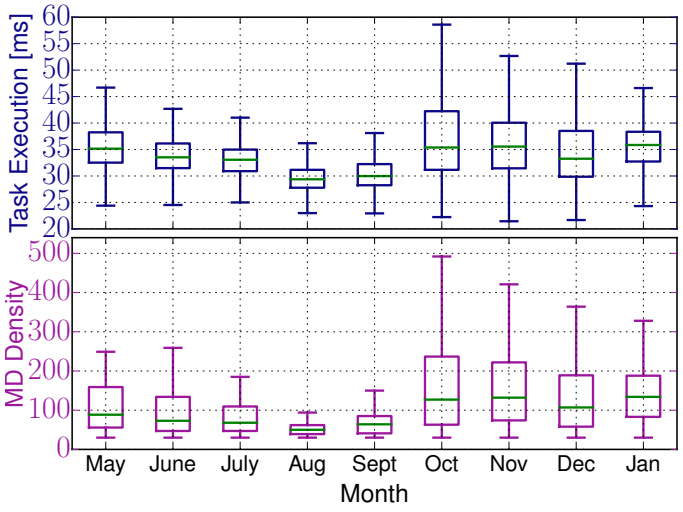

Fig. 8: Mobile devices and task latency for the LRZ dataset [6].

Configuration	Edge-cloud Infrastructure					Mobile Devices				FPS	RTT Dataset
	C#	AP	T1-EN	T2-EN	T3-EN	Density	Low-End	Mid	High-End		
(A) Scalability & Performance	C0 C1 C2 C3	4371 4371 4371 4371	350 250 150 100	200 100 50 25	10 5 2 0	(500, 1000, 2000, 4000)	33%	33%	33%	15	Seattle
(B) Full Dataset	—	182	182	30	3	≥ 30	33%	33%	33%	15	Seattle
(C) Nimbus Baseline	—	4371	100	50	2	1000	33%	33%	33%	10	Seattle
(D) Nimbus Variants	—	4371	100	50	2	1000	33%	33%	33%	10	Seattle

TABLE 4: Evaluation settings.

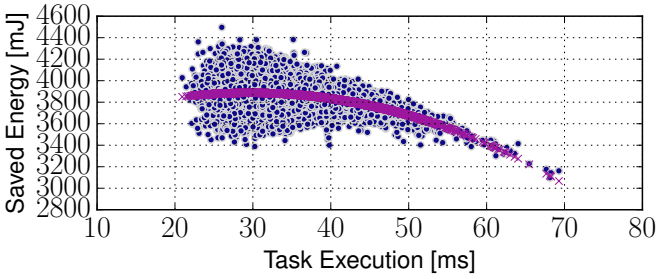

Fig. 9: Energy saving in relation to task execution time.

our algorithm with 15-minutes snapshots of MD densities from the LRZ dataset for the selected set of buildings. We further group the results by months for ease of readability. The selected buildings are part of a university campus, therefore, they exhibit a lower concentration of MDs during summer holiday period (July-September). Consequently, between October and January, the higher delivered task latency grows with the concentration of users connected to the network. However, due to the fairly low number of devices (between 30 and 500) and the generous size of the edge infrastructure, the median task latency is low. For example, the fastest MD in our dataset has a local inference time of 116 ms, which is almost **4**$\times$ higher than the average task latency that our edge infrastructure can deliver.

Figure 9 shows the relationship between the amount of saved energy for the MD and task latency, giving additional insights compared to Figure 6. For this plot, we calculated the energy consumption when the algorithm allocates all the MDs. The trend line demonstrates that with a higher task latency, we tend to save less energy. The leading cause can be a longer transmission time due to lower available uplink bandwidth. As an additional observation, we note having a static edge-cloud infrastructure might not always be the best option as the MD density changes at different times of the year. We hypothesize the possibility of a dynamic edge-cloud infrastructure where EN can be added dynamically in response to an increased density and demand of MDs. This would be similar to cloud computing, where resources are managed on-demand.

Finally, we analyzed the overall MD allocation ratio for the slice of data extracted from the dataset. Notice how 28.3% and 51.6% of total the MDs are allocated to T1- and T2-EN, respectively. The reasons for this can be manifold. Firstly, the number of T1-EN exceeds other tiers in our infrastructure and offers the lowest network RTT which compensates for the longer execution time. However, due to their limited resources, they can only serve a handful of MDs. T2-ENs, on the other hand, are more powerful and strike a good balance between scalability and network latency. Only 18% of the MD were offloaded to T3-EN as they offer low computation time but at the expense of *longer* network RTT. We remind that the MD population is small for this experiment. In fact, increasing the MD density pushes the algorithm to allocate more on T3-EN, as it is the only class of edge nodes capable of scaling efficiently without hindering performance. Finally, 1.1% of the MD ran the task locally, and the remaining 1.1% used the cloud. In the next section, we investigate how infrastructure size and the amount of MDs affect these ratios.

(C) Nimbus Baseline. For the baseline comparison, we evaluate our algorithm against a greedy version for 100 repetitions. Additionally, we also compare against a scenario where only cloud datacenters are available as offloading candidates, and MDs access them via either WiFi or 4G. We do not compare directly against other related algorithms (discussed in § 2) as our task allocation is fundamentally different from these approaches. Unlike related approaches, we do not rearrange and serialize the tasks to minimize the makespan but allow them to execute in parallel. For a fair comparison, we set side by side our approach with variants of Nimbus, which closely mimic the core ideology of related task offloading algorithms.

The *greedy* variant of Nimbus is fundamentally selfish: it selects the most profitable offloading candidate regardless of the possible performance degradation for the other MDs. While the standard version of the algorithm will use an unlimited search space, the greedy one will instead favor a quicker, local solution that minimizes network latency. This approach is typical of greedy algorithms that make the locally optimal choice at each stage [22]. We exploit the exploration parameter (κ) to limit greedy Nimbus's search space. The parameter also allows us to force the algorithm to produce the best offloading target from the network latency perspective and ignore the current load on edge nodes. Additionally, in the greedy version, the *LookAheadLoad* procedure is deactivated, and the weights α and β are set to 1 and 0, respectively.

Figure 10 illustrates the results of our multifaceted analysis. From a latency standpoint, the greedy algorithm is able to find good offloading candidates for mobile devices. As a matter of fact, the difference in terms of median latency achieved by greedy compared to the standard version of Nimbus is minimal. However, the standard deviation is much more noticeable due to the increasing number of non-offloaded MDs. Nimbus offloads $\sim$30% more MDs than the greedy version and, specifically, minimizes MDs that

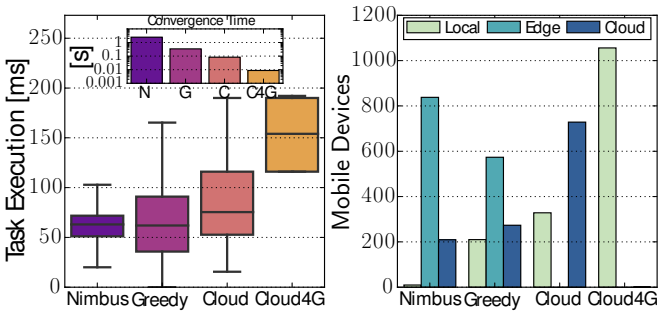

Fig. 10: Baseline comparison.

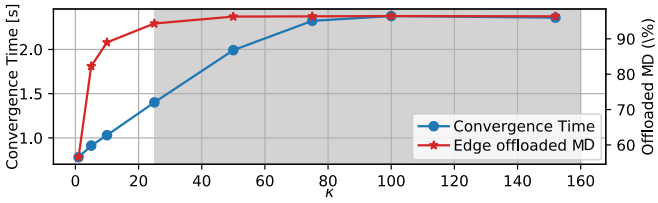

Fig. 11: Exploration Tradeoff (152 ENs, ∼1000 MDs).

resort to using local resources for task execution. Note that these results are drawn over a largely homogeneous edge infrastructure, with only three classes of participating ENs. In a highly heterogeneous environment, the limited search scope used in the greedy configuration could lead to unstable results, since there is an increased chance of missing good offloading targets in the search procedure. We extract the cloud network RTTs from the RIPE Atlas dataset discussed in § 6). We obtained the network RTTs using probes pinging datacenters co-located in the same region. Additionally, we set the inference latency in the cloud to 5 ms (comparable to a T3-EN) regardless of the served devices (*e.g.*, no queue time). The *cloud-only* approach (labeled *Cloud* in Figure 10) produces acceptable results, but at the cost of slight higher median task latency and greater variance compared to Nimbus. Additionally, the approach is unable to offload tasks from many MDs, forcing them to run locally. Finally, the cloud-only variant with *mobile access network* (labeled *Cloud4G*) delivers the worst performance – with close to 100% MDs unable to offload their computation. The primary reasons are significantly expensive transmission and energy costs, and higher network RTT to the processing server. Our result is in line with previous research, which shows that mobile connections require significantly more energy per bit in transmission compared to Wi-Fi [43].

Figure 11 shows the relationship between percentage of edge-offloaded MD, convergence time of the algorithm, and value of κ for an edge-cloud infrastructure of 152 ENs and 1000 MDs. Regarding the algorithm convergence time, the greedy version performs one order of magnitude faster compared to the standard one (inset plot in Figure 10). It should be noted that the Nimbus cloud-only variants converge much faster due to their simplified solver logic. When in need to allocate high densities of MDs, properly tuning the exploration parameter κ allows us to find a convenient tradeoff between offloaded MDs ratio and algorithm convergence time. Selecting a value of 10 for κ allows to already offload ∼91% of the MD while keeping a sub-second convergence time. During our experiments with different infrastructure and ENs configurations, we noticed that setting κ between 10-20% of the total EN in the network strikes a good balance between MDs allocation percentage and convergence time. However, this cutoff point might also be affected by the rather strong homogeneity of our infrastructure, since we only consider three classes of ENs. We hypothesize that with a more heterogeneous network, the cutoff point would be higher which translates into a greater range of exploration and an increased cost in terms of convergence time.

Note that the convergence time in Figure 11 represents the time required to allocate all the MDs in the batch. The allocation operation does not run for every offloaded frame, but only once when the mobile devices initiate an offload request to the infrastructure. Additionally, the system is designed in such a way that, while the MD waits be offloaded, the end-user will not experience any service interruptions as the task will keep running locally until the allocation on the edge-cloud infrastructure is completed. In this case, we assume that the MD is capable of executing the task locally. Finally, it is valid to assume that the amount of time the user will spend using the infrastructure offsets greatly the allocation waiting time similarly to start-up latency in video streaming.

Takeaway 2. The greedy algorithm is able to find good offloading candidates for MDs faster than Nimbus at the cost of sub-optimal utilization of the edge-cloud resources (e.g., skipping good offloading targets in the search procedure). The cloud-only variant is effective but provides higher median task latency, increased energy consumption for the MD, and greater variance compared to Nimbus.

(D) Nimbus Variants. We developed three versions of our solver. The one used in the previous benchmarks was single-threaded (ST), meaning that the decision process was handled by a single controller node which had complete knowledge of the edge infrastructure. From a practical viewpoint, such a solver offers limited scalability, especially when both the size of the edge infrastructure and density of participating MDs increases. For such cases, the convergence time of the single-threaded variant becomes prohibitive. Therefore, we developed a multi-threaded (MT) variant of Nimbus, termed *MT Nimbus*, that makes it deployable in a distributed fashion. We applied a partitioning procedure to the edge-cloud infrastructure. For a simple-yet-effective solution, we adopted a naive approach where we created non-overlapping sets of ENs so that every thread (or, equivalently, the entity managing a network slice) is independent of the others. We are aware that the procedure followed to split the edge-cloud network resources is not optimal, but, in this context, it suffices the evaluation purpose. Transforming an algorithm from centralized to distributed entails additional costs as synchronizing different entities increases communication overheads. Our goal is to demonstrate the possibility of transforming our algorithm into a distributed form and characterize its performance. In this work, we do not delve into communication and cross-node synchronization

challenges of a distributed system and leave it for future work.

As the number of ENs for each tier can be non-proportional to the number of threads, the network slices created can be unbalanced. For example, with ten solver threads and three T3-EN, the first three threads would have in their network slice at most one T3-EN. While the principal benefit for our distributed algorithm is decreased convergence time, we sacrifice in *quality* of the solution as the algorithm is now less capable of fully exploiting the available edge-cloud infrastructure resources. Figure 12 shows the convergence time and task execution latency with an increasing number of threads. It can be observed that the more we slice the network, the fewer MDs are offloaded because each slice becomes *shallower*, thus reducing the degrees of exploration for the algorithm. However, the convergence time per-thread reduces by up to $\sim$**15**$\times$ when Nimbus uses four threads instead of one.

To mitigate the inefficient use of the edge-cloud infrastructure, we developed a *two-stage* solver version of Nimbus. In this variant, all the MDs not offloaded in the first distributed stage are scheduled for a second allocation pass. The second stage executes centrally and is modified so that it attempts to allocate the remaining MDs on the entire edge-cloud infrastructure (updated with the current load). This final variant is called *2PMT Nimbus* and the results obtained are shown in Figure 13.

While there is an additional cost in terms of convergence time due to the presence of a final aggregation step, the amount of non-offloaded MDs reduces drastically, especially with an increasing number of threads. The effective ratio of offloaded users also increases compared to MT-Nimbus as 2PMT-Nimbus tends to fit more MDs into the edge-cloud infrastructure. Overall, 2PMT-Nimbus does not violate any of the inherent constraints and is able to deliver the required quality of experience (e.g., FPS) to all the offloaded users. With only two threads, 2PMT-Nimbus achieves similar MD allocation ratios as the single-threaded version while almost halving the convergence time. With eight threads, 2PMT-Nimbus converges almost 3$\times$ faster than two-threads and offloads the majority of the users. While the convergence time achieved by 2PMT-Nimbus is much slower than MT-Nimbus, the former is able to allocate many more MDs at the edge-cloud infrastructure.

Note the anomaly in convergence time trend of 2PMT-Nimbus – where the convergence time increases despite an increased degree of parallelism. We explain the exception as follows. By assigning more threads, the generated network slices become shallower and fewer EN candidates are available to allocate MDs. The fewer users are allocated, the more effort is required by the centralized solver to complete the final reallocation step. This entails that the law of diminishing returns applies to the threads parallelism. In fact, with ten threads, the multi-threaded convergence time decreases, but the single-threaded increases. However, the overall performance in terms of allocation ratio looks better with increasing thread count. Consequently, if we would progressively increase the assigned threads boundlessly, we would circle back to the single-threaded performance, both for allocation and convergence time.

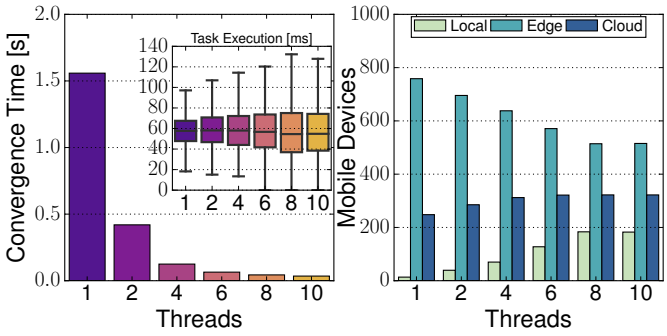

Fig. 12: Performance of MT-Nimbus.

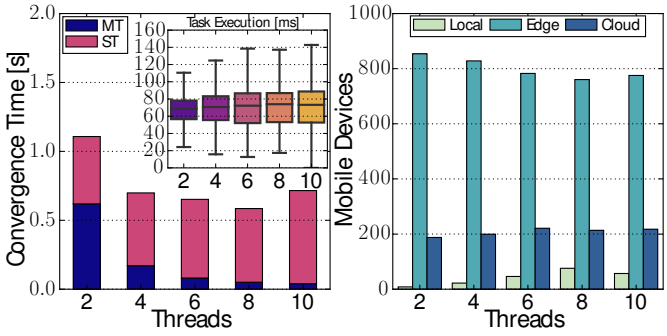

Fig. 13: Performance of 2PMT-Nimbus.

Takeaway 3. The ST version of Nimbus scales poorly as the size of the edge infrastructure and density of participating MDs increases. The MT variant is much faster but cannot fully makes use of the edge-cloud infrastructure. Finally, 2PMT-Nimbus provides the best performance in terms of ratio of offloaded MDs.

8 Limitations and Outlook

Edge computing will play a significant role in reshaping the future of cloud networks infrastructure. New applications and services will leverage information and processing capabilities offered at the network edge for varying purposes – including but not limited to data aggregation and analysis, multimedia content delivery, machine learning and AI. In this section, we explore orthogonal problems affecting edge computing putting our findings into a broader perspective.

Application & Network. Immersive applications, such as AR/VR, necessitate the deployment of edge servers in the network due to the strict latency constraints they impose. Such applications are guided by the human vestibular system which requires sensory inputs and interactions to be in complete sync; failure of which results in motion sickness and dizziness. As QoS of network communication technologies (*e.g.*, 5G and millimeter waves) improve (*i.e.*, shorter network delay and higher throughput [73, 76, 86]), optimizations in compute capabilities and task allocation mechanisms at edge become paramount to support multimedia QoE requirements.

However, end-to-end application latency still accounts for the most significant fraction of the perceived user experience, as discussed in § 7. Therefore, in this work, we focused on

task execution time and network latency while ignoring the non-marginal overhead introduced by other components. These additional delays may have multiple sources, including the operating system, bloated network queues, network fluctuations (retransmissions, packet loss), to name a few. We ignore these variables in this work to keep the problem tractable since added delay caused by some of the above is predictable only to a certain extent. Consequently, our results should be considered as an optimistic estimate on top of which application logic and context overhead must be added. The Nimbus system presented in this paper manages the interaction between edge infrastructure and MDs and offers a device-independent framework to offload tasks to the edge. In our future work, we plan to extend the platform to calculate the additional application overheads, as discussed above.

Smartphone evolution. The symbiosis between edge computing and mobile-based applications is complicated. Factors like ever-increasing computational capacities of smartphones [12, 13], and more general-purpose utility of edge computing begs re-thinking the applicability of edge for mobile clients. For example, high-end smartphones equipped with powerful mobile GPUs benefit more from running computations locally than offloading, due to higher efficiency (in energy consumption and inference time) offered by their processor architectures and algorithms [15]. On the other hand, essential operations utilizing local GPU may become throttled as number of applications competing for the shared GPU cycles increases. We feel that edge resources can be used to further enhance (or enable) what can be achieved by a smartphone. An example could be executing more sophisticated and accurate neural networks – which are often prohibitive for smartphones as they require considerably more RAM and computational power. Until mobile devices are battery-powered, there will always be a trade-off in performance versus battery consumption. One can also envision smartphones becoming part of the edge infrastructure [48], which poses new and exciting challenges for managing transient, mobility capable compute nodes.

Security Implications. We purposefully avoid delving into possible security vulnerabilities of Nimbus since we consider it out-of-scope. Here, we explore possible security holes in our system and provide hints on how to mitigate them. In our approach, we do not restrict an MD to the maximum time for which they can utilize the edge-cloud infrastructure. This can lead to numerous problems: a malicious MD might decide to offload tasks forever and to multiple servers to leech resources from the infrastructure, which may lead to starvation. One solution could be to use a credit or reputation system [54], where an MD can only utilize services offered by the edge-cloud by spending some virtual currency. Other possible approaches could be introducing a fixed time limit after which the MD is forcefully rescheduled. However, all these solutions require MD to be registered so that system can keep track of their credit or the amount of time spent using the service. Distributed ledgers and blockchain might be useful in this scenario to help keep track of the user credit and enable point-to-point payments [44, 70].

In § 7, we discussed the possibility of an *elastic* infrastructure composed of *consumer* ENs offering compute resources similarly to [18] to respond to network overloads. There are several issues associated with such an infrastructure, including reduced control over ENs, intermittent resource availability, reliability, inconsistent execution and queue time predictions, and security and privacy concerns. Additionally, *trust* can be a problem for such an infrastructure as malicious ENs might extract sensitive information while computing a task or deliberately modify the outcome to disrupt the service. Possible resolutions could be employing Trust Execution Environments (TEE) [69, 75] to secure the compute steps at the cost of operational complexity.

Deployment Challenges. When discussing changes advocated by edge computing, it is essential to keep in mind its *adoption* cost. Depending on the type of deployed ENs, the Capital Expenditures (CapEx) [2] and Operational Expenditures (OpEx) cost demand careful planning of the infrastructure as function of the QoS to be delivered over a period of time. Similar to cloud and ISP services, edge-cloud could employ a subscription-based operation model. End-users could choose from different subscription plans that best cater to desired QoE of targeted applications, e.g. gaming, healthcare, video analytics, etc.

9 Conclusion

In this paper we presented Nimbus, a multi-objective task allocation solution that can minimize the latency of mobile real-time object detection models by offloading them to an edge-cloud infrastructure. Based on an extensive set of real data and measurements, our multifaceted evaluation benchmarks three ever-improving variants of Nimbus addressing, especially, the problem of scalability from the infrastructure and end-users point of view. We verify the effectiveness of Nimbus through trace-driven simulations. Based on an extensive set of real data and measurements, we show the potential of Nimbus in boosting the performance of AR applications when offloaded from mobile devices to an edge-cloud infrastructure. Additionally, our multifaceted evaluation presents three ever-improving variants of Nimbus addressing, especially, scalability issues of edge-cloud infrastructure. Finally, in light of our algorithm and approach, we discuss several crucial open questions concerning edge computing and highlights future research directions.

References

[1] Benchmarking Hardware for CNN Inference in 2018. https://towardsdatascience.com/benchmarking-hardware-for-cnn-inference-in-2018-1d58268de12a.

[2] Capital Expenditure. https://en.wikipedia.org/wiki/Capital_expenditure. Accessed: 2020-04-27.

[3] Hill Climbing. https://en.wikipedia.org/wiki/Hill_climbing. Accessed: 2020-04-27.

[4] Network Latency Datasets. https://github.com/uofa-rzhu3/NetLatency-Data. Accessed: 2020-06-30.

[5] Leibniz-Rechenzentrum. https://www.lrz.de, . Accessed: 2020-05-22.

[6] WLAN-Verbindungen im MWN (Statistik). http://wlan.lrz.de/apstat/search, . Accessed: 2020-05-22.

[7] How is Mobile AR Landing with Consumers? https://virtualrealitypop.com/how-is-mobile-ar-

landing-with-consumers-cbc4b14e5957. Accessed: 2020-04-27.
[8] PlanetLab. https://www.planet-lab.org. Accessed: 2020-06-30.
[9] A First Inside Look at Pokémon GO Battery Drain. http://mobileenerlytics.com/a-first-inside-look-at-pokemon-go-battery-drain-you-wont-catch-many-if-your-battery-dies-so-quickly/. Accessed: 2020-04-02.
[10] Latency – the sine qua non of AR and VR. http://blogs.valvesoftware.com/abrash/latency-the-sine-qua-non-of-ar-and-vr/. Accessed: 2020-04-27.
[11] Seattle. https://seattle.poly.edu. Accessed: 2020-06-30.
[12] Your Phone Is Now More Powerful Than Your PC. https://insights.samsung.com/2018/08/09/your-phone-is-now-more-powerful-than-your-pc/, . Accessed: 2020-05-22.
[13] How the computing power in a smartphone compares to supercomputers past and present. https://www.businessinsider.com/infographic-how-computing-power-has-changed-over-time-2017-11?r=DE&IR=T, . Accessed: 2020-05-22.
[14] NVIDIA Triton Inference Server. https://github.com/NVIDIA/triton-inference-server. Accessed: 2020-05-22.
[15] UbiSpark project. https://ubispark.cs.helsinki.fi/. Accessed: 2020-06-30.
[16] Mario Almeida, Stefanos Laskaridis, Stylianos I Venieris, Ilias Leontiadis, and Nicholas D Lane. Dyno: Dynamic onloading of deep neural networks from cloud to device. *arXiv preprint arXiv:2104.09949*, 2021.
[17] Brandon Amos, Bartosz Ludwiczuk, Mahadev Satyanarayanan, et al. Openface: A general-purpose face recognition library with mobile applications. *CMU School of Computer Science*, 6:2, 2016.
[18] David P Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.
[19] Mykhaylo Andriluka, Stefan Roth, and Bernt Schiele. People-tracking-by-detection and people-detection-by-tracking. In *2008 IEEE Conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2008.
[20] Mohammad Babar, Muhammad Sohail Khan, Farman Ali, Muhammad Imran, and Muhammad Shoaib. Cloudlet computing: Recent advances, taxonomy, and challenges. *IEEE Access*, 9:29609–29622, 2021.
[21] Rajesh Krishna Balan, Darren Gergle, Mahadev Satyanarayanan, and James Herbsleb. Simplifying cyber foraging for mobile devices. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 272–285, 2007.
[22] Paul E Black. Greedy algorithm. *Dictionary of Algorithms and Data Structures*, 2:62, 2005.
[23] Michaela Blott, Lisa Halder, Miriam Leeser, and Linda Doyle. Qutibench: Benchmarking neural networks on heterogeneous hardware. *J. Emerg. Technol. Comput. Syst.*, 15(4), dec 2019. ISSN 1550-4832. doi: 10.1145/3358700. URL https://doi.org/10.1145/3358700.
[24] Mathieu Bouet and Vania Conan. Mobile edge computing resources optimization: A geo-clustering approach. *IEEE Transactions on Network and Service Management*, 15 (2):787–796, 2018.
[25] Tristan Braud, Farshid Hassani Bijarbooneh, Dimitris Chatzopoulos, and Pan Hui. Future networking challenges: The case of mobile augmented reality. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1796–1807. IEEE, 2017.
[26] Tristan Braud, Pengyuan Zhou, Jussi Kangasharju, and Pan Hui. Multipath computation offloading for mobile augmented reality. In *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–10. IEEE, 2020.
[27] Martin Breitbach, Janick Edinger, Siim Kaupmees, Heiko Trötsch, Christian Krupitzer, and Christian Becker. Voltaire: Precise energy-aware code offloading decisions with machine learning. In *2021 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–10. IEEE, 2021.
[28] Kaifei Chen, Tong Li, Hyung-Sin Kim, David E. Culler, and Randy H. Katz. Marvel: Enabling mobile augmented reality with low energy and low latency. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, SenSys '18, page 292–304, New York, NY, USA, 2018. Association for Computing Machinery. doi: 10.1145/3274783.3274834.
[29] Ying Chen, Ning Zhang, Yongchao Zhang, and Xin Chen. Dynamic computation offloading in edge computing for internet of things. *IEEE Internet of Things Journal*, 6(3):4242–4251, 2018.
[30] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314, 2011.
[31] Lorenzo Corneo, Maximilian Eder, Nitinder Mohan, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, Per Gunningberg, Jussi Kangasharju, and Jörg Ott. Surrounded by the Clouds: A Comprehensive Cloud Reachability Study. In *Proceedings of The Web Conference 2021*, WWW '21, New York, NY, USA, 2021. Association for Computing Machinery. doi: 10.1145/3442381.3449854. URL https://doi.org/10.1145/3442381.3449854.
[32] Lorenzo Corneo, Nitinder Mohan, Aleksandr Zavodovski, Walter Wong, Christian Rohner, Per Gunningberg, and Jussi Kangasharju. (how much) can edge computing change network latency? In *2021 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2021.
[33] V. Cozzolino, J. Ott, A. Y. Ding, and R. Mortier. Ecco: Edge-cloud chaining and orchestration framework for road context assessment. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 223–230, 2020.
[34] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62, 2010.
[35] Eduardo Cuervo, Alec Wolman, Landon P Cox, Kiron Lebeck, Ali Razeen, Stefan Saroiu, and Madanlal Musuvathi. Kahawai: High-quality mobile gaming using gpu

offload. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 121–135, 2015.

[36] The Khang Dang, Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Jörg Ott, and Jussi Kangasharju. Cloudy with a Chance of Short RTTs: Analyzing Cloud Connectivity in the Internet. In *Proceedings of Internet Measurement Conference*, IMC '21, New York, NY, USA, 2021. Association for Computing Machinery. doi: 10.1145/3487552.3487854. URL https://doi.org/10.1145/3487552.3487854.

[37] Kalyanmoy Deb. Multi-objective optimization. In *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pages 403–449. Springer, 2014.

[38] Thinh Quang Dinh, Jianhua Tang, Quang Duy La, and Tony QS Quek. Offloading in mobile edge computing: Task allocation and computational frequency scaling. *IEEE Transactions on Communications*, 65(8):3571–3584, 2017.

[39] Utsav Drolia, Rolando Martins, Jiaqi Tan, Ankit Chheda, Monil Sanghavi, Rajeev Gandhi, and Priya Narasimhan. The case for mobile edge-clouds. In *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*, pages 209–215. IEEE, 2013.

[40] Maximilian Eder, Lorenzo Corneo, Nitinder Mohan, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, Per Gunningberg, Jussi Kangasharju, and Jörg Ott. Surrounded by the clouds, 2021. URL https://mediatum.ub.tum.de/1593899.

[41] Jason Flinn and Z Morley Mao. Can deterministic replay be an enabling tool for mobile computing? In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, pages 84–89, 2011.

[42] Mark S Gordon, D Anoushe Jamshidi, Scott Mahlke, Z Morley Mao, and Xu Chen. {COMET}: Code offload by migrating execution transparently. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pages 93–106, 2012.

[43] Tian Guo. Cloud-based or on-device: An empirical study of mobile deep inference. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 184–190. IEEE, 2018.

[44] Ye Guo and Chen Liang. Blockchain application and outlook in the banking industry. *Financial Innovation*, 2 (1):24, 2016.

[45] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 123–136. ACM, 2016.

[46] Zhenhua Han, Haisheng Tan, Guihai Chen, Rui Wang, Yifan Chen, and Francis CM Lau. Dynamic virtual machine management via approximate markov decision process. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[47] Ting He, Hana Khamfroush, Shiqiang Wang, Tom La Porta, and Sebastian Stein. It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 365–375. IEEE, 2018.

[48] Junyan Hu, Kenli Li, Chubo Liu, and Keqin Li. Game-based task offloading of multiple mobile devices with qos in mobile edge computing systems of limited computation capacity. *ACM Trans. Embed. Comput. Syst.*, 19(4), July 2020. ISSN 1539-9087. doi: 10.1145/3398038. URL https://doi.org/10.1145/3398038.

[49] C-L Hwang and Abu Syed Md Masud. *Multiple objective decision making—methods and applications: a state-of-the-art survey*, volume 164. Springer Science & Business Media, 2012.

[50] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European conference on computer vision (ECCV)*, pages 0–0, 2018.

[51] Mike Jia, Jiannong Cao, and Weifa Liang. Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. *IEEE Transactions on Cloud Computing*, 5(4):725–737, 2015.

[52] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017.

[53] Kuljeet Kaur, Tanya Dhand, Neeraj Kumar, and Sherali Zeadally. Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers. *IEEE wireless communications*, 24(3): 48–56, 2017.

[54] Michael Kinateder and Kurt Rothermel. Architecture and algorithms for a distributed reputation system. In *International Conference on Trust Management*, pages 1–16. Springer, 2003.

[55] Chenge Li, Gregory Dobler, Xin Feng, and Yao Wang. Tracknet: Simultaneous object detection and tracking and its application in traffic video analysis. *arXiv preprint arXiv:1902.01466*, 2019.

[56] Fangming Liu, Peng Shu, Hai Jin, Linjie Ding, Jie Yu, Di Niu, and Bo Li. Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. *IEEE Wireless communications*, 20(3):14–22, 2013.

[57] Luyang Liu, Hongyu Li, and Marco Gruteser. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[58] Qiang Liu, Siqi Huang, Johnson Opadere, and Tao Han. An edge network orchestrator for mobile augmented reality. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 756–764. IEEE, 2018.

[59] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads.

In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 301–312. IEEE, 2014.

[60] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[61] Sumit Maheshwari, Dipankar Raychaudhuri, Ivan Seskar, and Francesco Bronzino. Scalability and performance evaluation of edge cloud systems for latency constrained applications. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 286–299. IEEE, 2018.

[62] Jiaying Meng, Wenbin Shi, Haisheng Tan, and Xiangyang Li. Cloudlet placement and minimum-delay routing in cloudlet computing. In *2017 3rd International Conference on Big Data Computing and Communications (BIGCOM)*, pages 297–304. IEEE, 2017.

[63] Jiaying Meng, Haisheng Tan, Chao Xu, Wanli Cao, Liuyan Liu, and Bojie Li. Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2287–2295. IEEE, 2019.

[64] Nitinder Mohan and Jussi Kangasharju. Edge-fog cloud: A distributed cloud for internet of things computations. In *2016 Cloudification of the Internet of Things (CIoT)*, pages 1–6. IEEE, 2016.

[65] Nitinder Mohan and Jussi Kangasharju. Placing it right!: optimizing energy, processing, and transport in edge-fog clouds. *Annals of Telecommunications*, 73(7-8):463–474, 2018.

[66] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. Pruning edge research with latency shears. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, HotNets '20, page 182–189, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381451. doi: 10.1145/3422604.3425943. URL https://doi.org/10.1145/3422604.3425943.

[67] Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar, and Jorg Ott. Consolidate iot edge computing with lightweight virtualization. *IEEE Network*, 32(1):102–111, 2018.

[68] Ashkan Nikravesh, David R Choffnes, Ethan Katz-Bassett, Z Morley Mao, and Matt Welsh. Mobile network performance from user devices: A longitudinal, multidimensional analysis. In *International Conference on Passive and Active Network Measurement*, pages 12–22. Springer, 2014.

[69] Zhenyu Ning, Jinghui Liao, Fengwei Zhang, and Weisong Shi. Preliminary study of trusted execution environments on heterogeneous edge platforms. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 421–426. IEEE, 2018.

[70] Jianli Pan, Jianyu Wang, Austin Hester, Ismail Alqerm, Yuanni Liu, and Ying Zhao. Edgechain: An edge-iot framework and prototype based on blockchain and smart contracts. *IEEE Internet of Things Journal*, 6(3): 4719–4732, 2018.

[71] Horst Possegger, Thomas Mauthner, Peter M Roth, and Horst Bischof. Occlusion geodesics for online multi-object tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1306–1313, 2014.

[72] Xukan Ran, Haolianz Chen, Xiaodan Zhu, Zhenming Liu, and Jiasi Chen. Deepdecision: A mobile deep learning framework for edge video analytics. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1421–1429. IEEE, 2018.

[73] Theodore S Rappaport, Shu Sun, Rimma Mayzus, Hang Zhao, Yaniv Azar, Kevin Wang, George N Wong, Jocelyn K Schulz, Mathew Samimi, and Felix Gutierrez. Millimeter wave mobile communications for 5g cellular: It will work! *IEEE access*, 1:335–349, 2013.

[74] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[75] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64. IEEE, 2015.

[76] J. Sachs, G. Wikstrom, T. Dudda, R. Baldemair, and K. Kittichokechai. 5g radio network design for ultra-reliable low-latency communication. *IEEE Network*, 32 (2):24–31, 2018.

[77] Onur Sahin and Ayse K Coskun. Providing sustainable performance in thermally constrained mobile devices. In *Proceedings of the 14th ACM/IEEE Symposium on Embedded Systems for Real-Time Multimedia*, pages 72–77, 2016.

[78] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[79] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4): 14–23, 2009.

[80] Ryan Shea, Di Fu, and Jiangchuan Liu. Rhizome: Utilizing the public cloud to provide 3d gaming infrastructure. In *Proceedings of the 6th ACM Multimedia Systems Conference*, pages 97–100, 2015.

[81] Shu Shi and Cheng-Hsin Hsu. A survey of interactive remote rendering systems. *ACM Computing Surveys (CSUR)*, 47(4):1–29, 2015.

[82] RIPE NCC Staff. Ripe atlas: A global internet measurement network. *Internet Protocol Journal*, 18(3), 2015.

[83] Xiang Sun and Nirwan Ansari. Edgeiot: Mobile edge computing for the internet of things. *IEEE Communications Magazine*, 54(12):22–29, 2016.

[84] Haisheng Tan, Zhenhua Han, Xiang-Yang Li, and Francis CM Lau. Online job dispatching and scheduling in edge-clouds. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[85] Liang Tong, Yong Li, and Wei Gao. A hierarchical edge cloud architecture for mobile computing. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[86] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and D. Soldani. Towards zero latency software defined 5g networks. In *2015 IEEE International Conference on Communication Workshop (ICCW)*, pages 2566–2571, 2015.

[87] Blesson Varghese, Eyal De Lara, Aaron Yi Ding, Cheol-Ho Hong, Flavio Bonomi, Schahram Dustdar, Paul Harvey, Peter Hewkin, Weisong Shi, Mark Thiele, et al. Revisiting the arguments for edge computing research. *IEEE Internet Computing*, 25(5):36–42, 2021.

[88] Massimo Villari, Maria Fazio, Schahram Dustdar, Omer Rana, and Rajiv Ranjan. Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6):76–83, 2016.

[89] Fangxin Wang, Miao Zhang, Xiangxiang Wang, Xiaoqiang Ma, and Jiangchuan Liu. Deep learning for edge computing applications: A state-of-the-art survey. *IEEE Access*, 8:58322–58336, 2020.

[90] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. When edge meets learning: Adaptive control for resource-constrained distributed machine learning. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 63–71. IEEE, 2018.

[91] X. Wang, L. T. Yang, X. Xie, J. Jin, and M. J. Deen. A cloud-edge computing framework for cyber-physical-social services. *IEEE Communications Magazine*, 55(11): 80–85, 2017.

[92] Yue Wang, Xiaofeng Tao, Xuefei Zhang, Ping Zhang, and Y Thomas Hou. Cooperative task offloading in three-tier mobile computing networks: An admm framework. *IEEE Transactions on Vehicular Technology*, 68(3):2763–2776, 2019.

[93] Jiyan Wu, Chau Yuen, Ngai-Man Cheung, Junliang Chen, and Chang Wen Chen. Enabling adaptive high-frame-rate video streaming in mobile cloud gaming applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(12):1988–2001, 2015.

[94] Chunwei Xia, Jiacheng Zhao, Huimin Cui, Xiaobing Feng, and Jingling Xue. Dnntune: Automatic benchmarking dnn models for mobile-cloud computing. *ACM Trans. Archit. Code Optim.*, 16(4), December 2019. ISSN 1544-3566. doi: 10.1145/3368305. URL https://doi.org/10.1145/3368305.

[95] Zhujun Xiao, Zhengxu Xia, Haitao Zheng, Ben Y Zhao, and Junchen Jiang. Towards performance clarity of edge video analytics. *arXiv preprint arXiv:2105.08694*, 2021.

[96] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 15. ACM, 2017.

[97] Wenxiao Zhang, Bo Han, and Pan Hui. On the networking challenges of mobile augmented reality. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*, pages 24–29, 2017.

[98] Junlong Zhou, Tian Wang, Peijin Cong, Pingping Lu, Tongquan Wei, and Mingsong Chen. Cost and makespan-aware workflow scheduling in hybrid clouds. *Journal of Systems Architecture*, 100:101631, 2019.

[99] Agustin Zuniga, Huber Flores, Eemil Lagerspetz, Petteri Nurmi, Sasu Tarkoma, Pan Hui, and Jukka Manner. Tortoise or hare? quantifying the effects of performance on mobile app retention. In *The World Wide Web Conference*, pages 2517–2528, 2019.

Vittorio Cozzolino is a PhD candidate in the Chair of Connected Mobility at the Technical University of Munich, and his research interests are in distributed systems, edge and cloud computing, and resource management and virtualization.

Leonardo Tonetto is a PhD candidate in the Chair of Connected Mobility at the Technical University of Munich, and his research interests are in mobile user behavioral modeling, human mobility and complex networks.

Dr. Nitinder Mohan is a Postdoctoral researcher in the Chair of Connected Mobility at Technical University of Munich, Germany. He received his Ph.D. (as Marie Curie ITN fellow) from the Department of Computer Science at the University of Helsinki in Finland and M.Tech. degree (honors) from Indraprastha Institute of Information Technology Delhi (IIIT-D), India. He has been awarded "Outstanding Ph.D. Dissertation Award" by IEEE Technical Committee on Scalable Computing (TCSC). He has worked as Visting Researcher in NEC Labs Europe, Germany and at University of Göttingen, Germany, and as a Project Scientist in Indian Institute of Technology Delhi (IIT-D), India.

Aaron Yi Ding is the Head of Cyber-Physical Intelligence (CPI) Lab, Asst Professor at TU Delft and Adjunct Professor (Dosentti) in Computer Science at University of Helsinki. He has worked at TU Munich, Columbia University, and University of Cambridge. He received the M.Sc. and Ph.D. degrees from the Department of Computer Science (Birthplace of Linux), University of Helsinki, supervised by Prof. Sasu Tarkoma and Prof. Jon Crowcroft. Funded by Nokia Foundation, part of his Ph.D. programme was completed at the University of Cambridge, U.K., and Columbia University, USA. He has 50+ peer reviewed publications and his work was awarded the best paper of ACM EdgeSys, ACM SIGCOMM Best of CCR, and Nokia Foundation Scholarships.

Jörg Ott holds the Chair of Connected Mobility in the Department of Informatics at TUM. He is also Adjunct Professor for Networking Technology at Aalto University. He received his diploma and doctoral (Dr.-Ing.) degree in computer science from TU Berlin (1991 and 1997, respectively), and his diploma in industrial engineering from TFH Berlin (1995). His research interests are in network architecture, (Internet) protocol design, and networked systems, with a focus on (mobile) decentralized services.