



Delft University of Technology

## 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](https://doi.org/10.1109/TCC.2022.3146615)

#### Publication date

2022

#### Document Version

Final published version

#### Published in

IEEE Transactions on Cloud Computing

#### Citation (APA)

Cozzolino, V., Tonetto, L., Mohan, N., Ding, A. Y., & Ott, J. (2022). Nimbus: Towards Latency-Energy Efficient Task Offloading for AR Services. *IEEE Transactions on Cloud Computing*, 11 (2023)(2), 1530-1545. <https://doi.org/10.1109/TCC.2022.3146615>

#### Important note

To cite this publication, please use the final published version (if applicable).  
Please check the document version above.

#### Copyright

Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

#### Takedown policy

Please contact us and provide details if you believe this document breaches copyrights.  
We will remove access to the work immediately and investigate your claim.



***Green Open Access added to TU Delft Institutional Repository***

***'You share, we take care!' - Taverne project***

***<https://www.openaccess.nl/en/you-share-we-take-care>***

Otherwise as indicated in the copyright section: the publisher is the copyright holder of this work and the author uses the Dutch legislation to make this work public.



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

Vittorio Cozzolino<sup>ID</sup>, Leonardo Tonetto, Nitinder Mohan<sup>ID</sup>, Aaron Yi Ding<sup>ID</sup>, and Jörg Ott<sup>ID</sup>

**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.

**Index Terms**—Edge computing, augmented reality, optimization, resource management, cloud computing

## 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. First, 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]. Second, 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,  $\sim 7\text{ms}$  [10], [25]. Currently, a modern smartphone can run object detection in  $\sim 200\text{ms}$  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

- Vittorio Cozzolino, Leonardo Tonetto, Nitinder Mohan, and Jörg Ott are with the Technical University of Munich, 80333, München, Germany. E-mail: {cozzolin, tonetto, mohan, ott}@in.tum.de.
- Aaron Yi Ding is with the Delft University of Technology, 2628, CD, Delft, The Netherlands. E-mail: Aaron.Ding@tudelft.nl.

Manuscript received 16 Sept. 2020; revised 13 Jan. 2022; accepted 19 Jan. 2022. Date of publication 27 Jan. 2022; date of current version 7 June 2023.

(Corresponding author: Vittorio Cozzolino.)

Recommended for acceptance by F. Ergun.

Digital Object Identifier no. 10.1109/TCC.2022.3146615



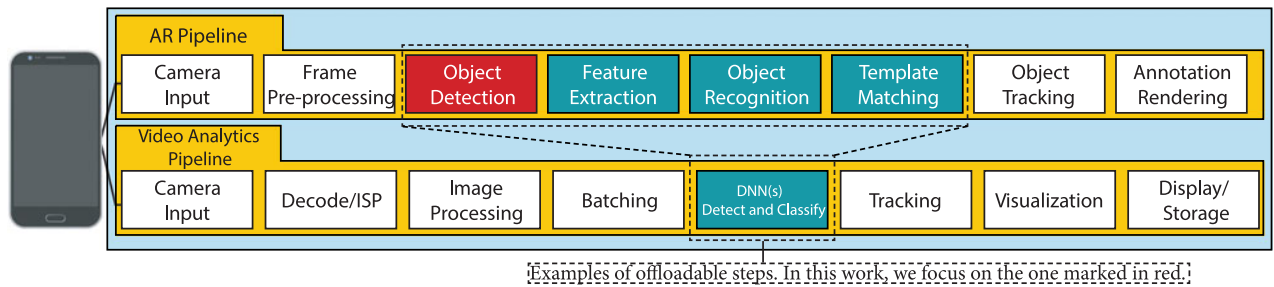


Fig. 1. Mobile applications requiring deep learning steps.

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 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 Fig. 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 Fig. 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:

- 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<sup>1</sup> 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

1) *latency* as a primary measure of the application QoS,

Authorized licensed use limited to: TU Delft Library. Downloaded on August 21, 2024 at 11:23:04 UTC from IEEE Xplore. Restrictions apply.

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



reducing energy consumption of mobile devices. For example, with Voltaire [27] it is proposed to perform code offloading to enable 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 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 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

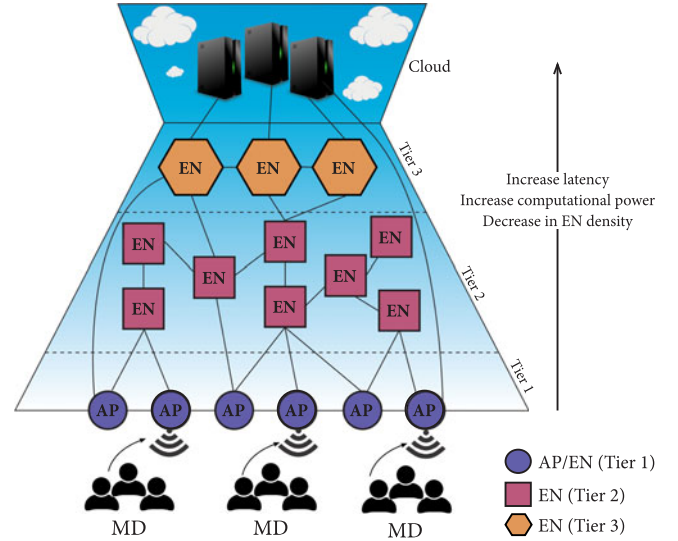


Fig. 2. Multi-tier edge-cloud infrastructure.

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.

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

Fig. 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<sup>2</sup>. 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, 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 Fig. 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 in Fig. 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 possess 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.

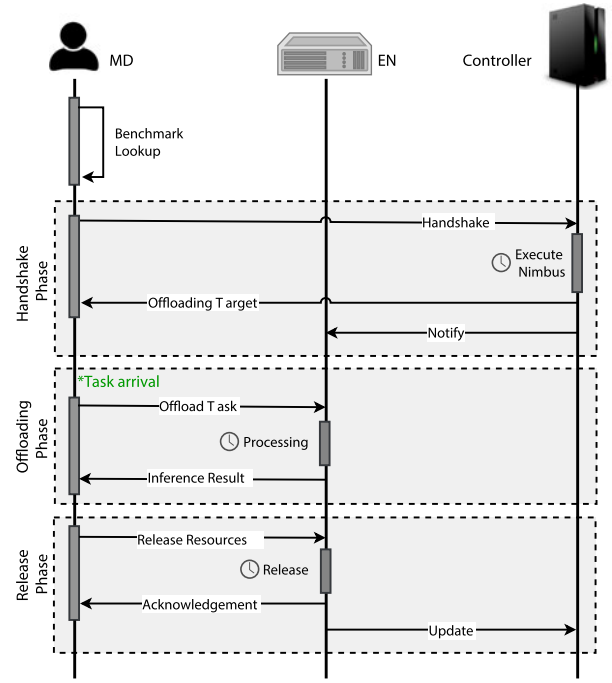


Fig. 3. MD-Controller Workflow.

#### 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. Fig. 3 shows a high-level concise workflow representing the MD-Controller interaction. 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.

2. <https://www.eduroam.org>



**Offloaded Tasks.** As shown previously in Fig. 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 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 uninteruptible). 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 Fig. 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. Fig. 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  $L_{t_{ij}}$  between the  $i$ th mobile device and the  $j$ th edge node, plus the execution time  $L_{e_{ij}}$  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

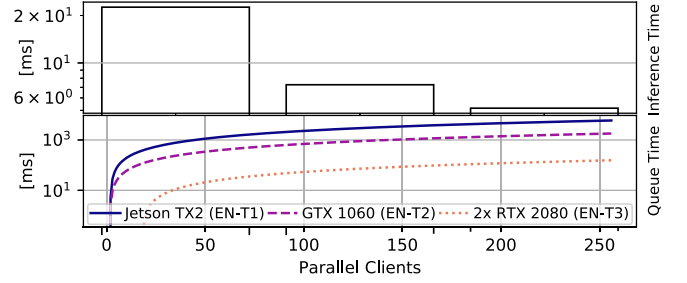


Fig. 4. Inference and queue time for the three EN tiers.

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. Fig. 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 Fig. 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 Fig. 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 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

$$L_t = \frac{d}{BW} + RTT \quad (1)$$

$$B_t = L_t \times w \quad (2)$$

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

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

$$B_{ij} = Bt_{ij} = Lt_{ij} \times w \quad (4)$$

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  $\min(L(\vec{x}), B(\vec{x}))$  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:  $\min(L(x), B(x), \phi)$  with  $x \in X_\phi$  and  $X_\phi$  set depending on the vector  $\phi$ . Of the multiple scalarization techniques, we adopt the  $\epsilon$ -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

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
$\epsilon_t$	Latency threshold	ms
$\epsilon_b$	Energy budget	J/s
$RTT_m$	RTT matrix	ms
$\kappa, \alpha, \beta$	Additional coefficients (described in § 5)	—

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  $\epsilon$ -constrained latency minimization problem as follows:

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

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

$$L_{ij} \leq \epsilon_t, \forall j \in M, \quad (7)$$

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

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

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. Equations (6) and (9) limit each MD to offload its task to as single EN, at most. Equations (7) and (8) are formalization of the latency and energy consumption constraints limiting the feasible solution space.

**Constraints.** The  $\epsilon_t$  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,  $\epsilon_t$  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.  $\epsilon_b$  represents the battery consumption threshold, exceeding which offloading the task becomes too expensive in terms of energy. Fundamentally,  $\epsilon_b$  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  $\epsilon_t$  and, potentially,  $\epsilon_b$  as it can complete locally its task quickly and efficiently (in terms of energy cost).

## 5 ALGORITHM

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



TABLE 2  
Mobile Inference Time and Energy Cost for MobileNetv2

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

artificial intelligence) for which it can provide the optimal solution [74]. Algorithm 1 describes Nimbus task offloading approach.

#### Algorithm 1. Nimbus Allocation Algorithm

**Input:** Refer to Table 1.  
**Output:** Best offloading target for the  $i$ -th MD.  
 // Warmup  
 1:  $\vec{EN}_r \leftarrow \text{FilterAndMinimize}(AP, RTT_m, \epsilon_t)$   
 2:  $\vec{EN} \leftarrow \text{LookAheadLoad}(\vec{EN}_r, \kappa)$   
 // Core  
 3: **for**  $EN_j$  in  $\vec{EN}$  **do**  
 4:  $L_{ij} = (Lt_{ij} + Le_{ij}) = \left[ \left( \frac{d_i}{BW_{ij}} \right) + RTT_{ij} \right] + (TET_j + q_j)$   
 5:  $B_{ij} = Bt_{ij} = (Lt_{ij} \times w)$   
 6: **if**  $L_{ij} \geq \epsilon_t$  **or**  $B_{ij} \geq \epsilon_b$  **then**  
 7:    $\text{Drop}(EN_j, \vec{EN})$   
 8: **end**  
 9: **end**  
 10: **if**  $\vec{EN} \neq \emptyset$  **then**  
 11:   **for**  $EN_j$  in  $\vec{EN}$  **do**  
 12:     **return**  $\arg \min[\alpha * \frac{L_{ij}}{\epsilon_t} + \beta * \frac{\text{load}_j}{\text{maxload}_j}]$   
 13:   **end**  
 14: **else**  
 // Failover  
 15:  $\text{cloud} \leftarrow \text{FindClosest}(\epsilon_t)$   
 16: **if**  $L_{\text{cloud}} \leq \epsilon_t$  **and**  $B_{\text{cloud}} \leq \epsilon_b$  **then**  
 17:   **return**  $\text{cloud}$   
 18: **end**  
 19: **end**  
 20: **return**  $\emptyset$

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  $\epsilon_t$  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  $\epsilon_t$  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  $\kappa$  controls the search space by setting an upper bound to the number of ENs we want to consider. In our evaluation,  $\kappa$  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  $\kappa, \epsilon_t$ .  
**Output:** List of compatible EN.  
 1:  $\text{compatibleEN} = \emptyset$   
 2: **for**  $EN_j$  in  $\vec{EN}$  **do**  
 3:   **if**  $\text{devicesList}_j \neq \emptyset$  **then**  
 4:      $\text{mnl} = \arg \max \text{deviceNetworkLatency}_j$   
 5:     **if**  $\text{size}(\text{devicesList}) \ll \text{maxServingDevices}_j(\epsilon_t - \text{mnl})$  **then**  
 6:        $\text{compatibleEN} \leftarrow EN_j$   
 7:     **end**  
 8:   **else**  
 9:      $\text{compatibleEN} \leftarrow EN_j$   
 10:   **end**  
 11: **end**  
 12: **if**  $\kappa \equiv 0$  **then**  
 13:   **return**  $\text{compatibleEN}$   
 14: **else**  
 15:   **return**  $\text{random.Set}(\kappa, \text{compatibleEN})$   
 16: **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  $\alpha$  and  $\beta$  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  $\beta$  to zero. In our evaluation, we set  $\alpha$  to 0.7 and  $\beta$  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,  $\text{load}_j$  and  $\text{maxload}_j$  represent the current and maximum load in terms of devices for the  $j$ -th EN, respectively. We calculate the latter using an



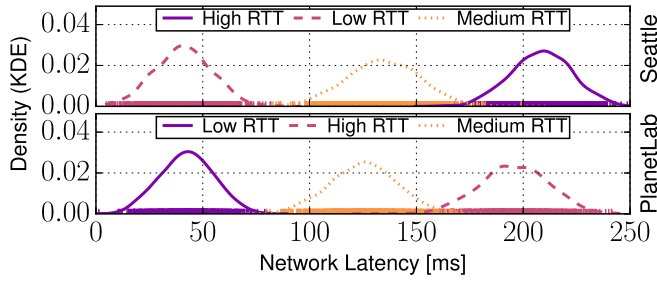


Fig. 5. Latency distributions for selected datasets.

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 Fig. 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.

TABLE 3  
RTT Matrix Structure

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

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.

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  $\approx 3.2M$  data-points 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 Fig. 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  $\sim 450$  buildings. The data are aggregated in 15 minutes slices, which we use as MD-batches



TABLE 4  
Evaluation Settings

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

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<sup>3</sup> 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  $\sim 66.6$  ms). We select four configurations where the number of connected MDs are 500, 1000, 2000, and 4000. Fig. 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 Fig. 6), the expected task latency achieved by Nimbus is  $\sim 2\times$  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. Fig. 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.

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

Task offloading also allows MDs to save energy (right panel of Fig. 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  $\sim 77\%$  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.

*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  $\sim 2\times$  lower compared to the fastest MD in our dataset. Additionally, MDs consume up to  $\sim 77\%$  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 Fig. 8 shows the task execution latency (top) and MD density (bottom) for the entire nine-month period. To obtain these results, we progressively feed our algorithm with 15-minutes snapshots of MD densities from

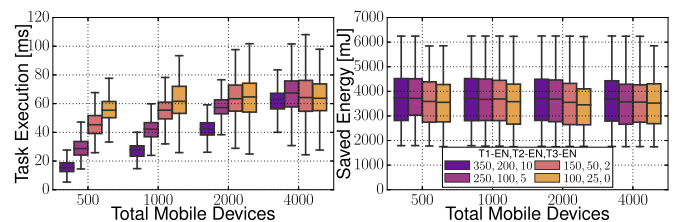


Fig. 6. Task latency and energy saving in various setups.



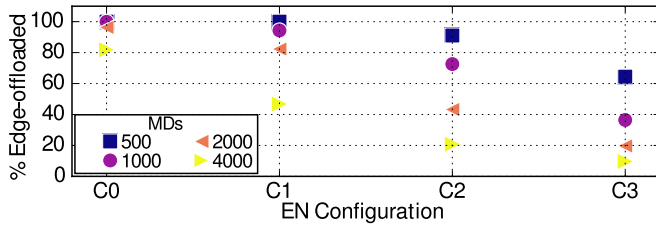


Fig. 7. Fraction of task offloaded to the edge (see Table 4).

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.

Fig. 9 shows the relationship between the amount of saved energy for the MD and task latency, giving additional insights compared to Fig. 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.

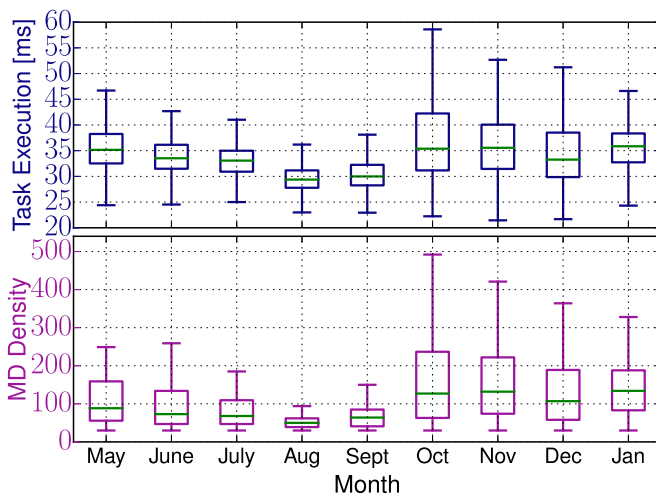


Fig. 8. Mobile devices and task latency for the LRZ dataset [6].

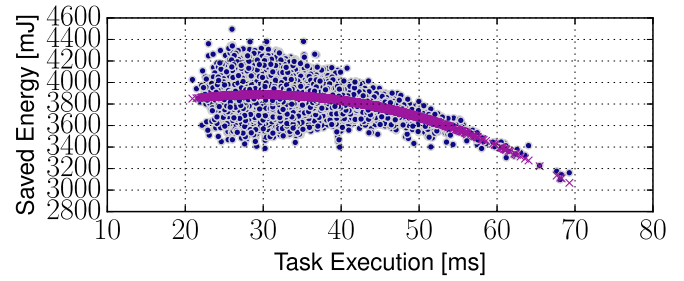


Fig. 9. Energy saving in relation to task execution time.

First, 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 ( $\kappa$ ) 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 *LookAhead-Load* procedure is deactivated, and the weights  $\alpha$  and  $\beta$  are set to 1 and 0, respectively.

Fig. 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



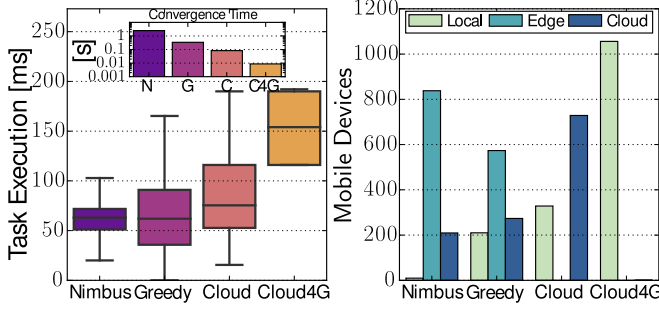
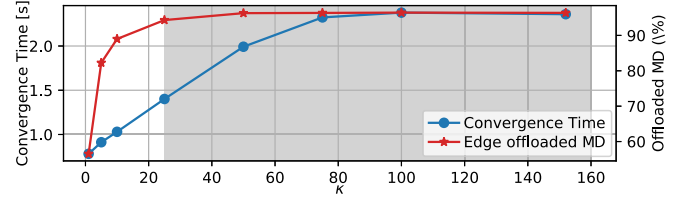


Fig. 10. Baseline comparison.

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 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 Fig. 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].

Fig. 11 shows the relationship between percentage of edge-offloaded MD, convergence time of the algorithm, and value of  $\kappa$  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 Fig. 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  $\kappa$  allows us to find a convenient tradeoff between offloaded MDs ratio and algorithm convergence time. Selecting a value of 10 for  $\kappa$  allows to already offload  $\sim 91\%$  of the MD while keeping a sub-second convergence time. During our experiments with different infrastructure and ENs configurations, we noticed that setting  $\kappa$  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

Fig. 11. Exploration Tradeoff (152 ENs,  $\sim 1000$  MDs).

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 Fig. 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 to 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



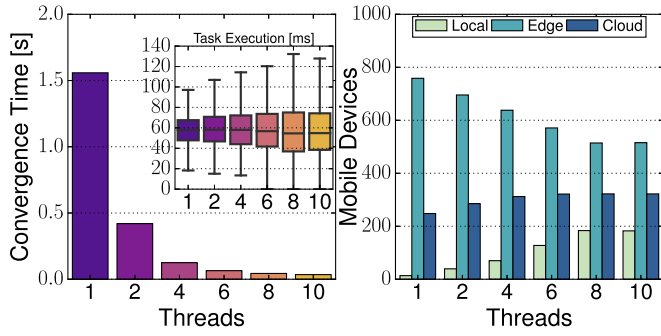


Fig. 12. Performance of MT-Nimbus.

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. Fig. 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 Fig. 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

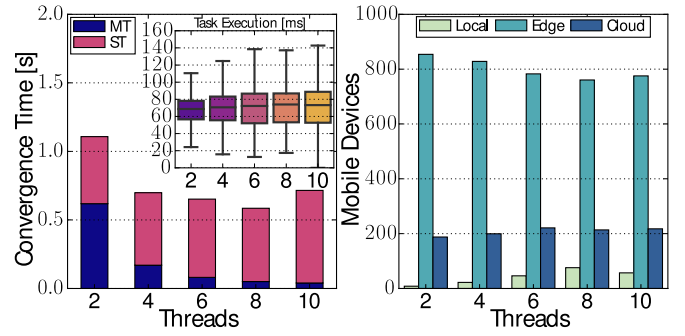


Fig. 13. Performance of 2PMT-Nimbus.

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.

**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. [Online]. Available: <https://towardsdatascience.com/benchmarking-hardware-for-cnn-inference-in-2018-1d58268de12a>
- [2] Capital expenditure. Accessed: Apr. 27, 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Capital\\_expenditure](https://en.wikipedia.org/wiki/Capital_expenditure)
- [3] Hill climbing. Accessed: Apr. 27, 2020. [Online]. Available: [https://en.wikipedia.org/wiki/Hill\\_climbing](https://en.wikipedia.org/wiki/Hill_climbing)
- [4] Network latency datasets. Accessed: Jun. 30, 2020. [Online]. Available: <https://github.com/uofa-rzhu3/NetLatency-Data>
- [5] Leibniz-Rechenzentrum. Accessed: May 22, 2020. [Online]. Available: <https://www.lrz.de>
- [6] WLAN-Verbindungen im MWN (Statistik). Accessed: May 22, 2020. [Online]. Available: <http://wlan.lrz.de/apstat/search>
- [7] How is mobile AR landing with consumers?, Accessed: Apr. 27, 2020. [Online]. Available: <https://virtualrealitypop.com/how-is-mobile-ar-landing-with-consumers-cbc4b14e5957>
- [8] PlanetLab. Accessed: Jun. 30, 2020. [Online]. Available: <https://www.planet-lab.org>
- [9] A first inside look at Pokémon GO battery drain. Accessed: Apr. 2, 2020. [Online]. Available: <http://mobileenergylytics.com/a-first-inside-look-at-pokemon-go-battery-drain-you-wont-catch-many-if-your-battery-dies-so-quickly/>



- [10] Latency – The Sine Qua non of AR and VR. Accessed: Apr. 27, 2020. [Online]. Available: <http://blogs.valvesoftware.com/abrash/latency-the-sine-qua-non-of-ar-and-vr/>
- [11] Seattle. Accessed: Jun. 30, 2020. [Online]. Available: <https://seattle.poly.edu>
- [12] Your phone is now more powerful than your PC. Accessed: May 22, 2020. [Online]. Available: <https://insights.samsung.com/2018/08/09/your-phone-is-now-more-powerful-than-your-pc/>
- [13] How the computing power in a smartphone compares to supercomputers past and present. Accessed: May 22, 2020. [Online]. Available: <https://www.businessinsider.com/infographic-how-computing-power-has-changed-over-time-2017-11?r=DE&IR=T>
- [14] NVIDIA Triton Inference Server. Accessed: May 22, 2020. [Online]. Available: <https://github.com/NVIDIA/triton-inference-server>
- [15] UbiSpark project. Accessed: Jun. 30, 2020. [Online]. Available: <https://ubispark.cs.helsinki.fi/>
- [16] M. Almeida, S. Laskaridis, S. I. Venieris, I. Leontiadis, and N. D. Lane, "Dyno: Dynamic onloading of deep neural networks from cloud to device," 2021, *arXiv:2104.09949*.
- [17] B. Amos et al., "Openface: A general-purpose face recognition library with mobile applications," *CMU Sch. Comput. Sci.*, vol. 6, no. 2, 2016.
- [18] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@home: An experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, 2002.
- [19] M. Andriluka, S. Roth, and B. Schiele, "People-tracking-by-detection and people-detection-by-tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2008, pp. 1–8.
- [20] M. Babar, M. S. Khan, F. Ali, M. Imran, and M. Shoaib, "Cloudlet computing: Recent advances, taxonomy, and challenges," *IEEE Access*, vol. 9, pp. 29 609–29 622, 2021.
- [21] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb, "Simplifying cyber foraging for mobile devices," in *Proc. 5th Int. Conf. Mobile Syst. Appl. Serv.*, 2007, pp. 272–285.
- [22] P. E. Black, "Greedy algorithm," *Dictionary Algorithms Data Struct.*, vol. 2, no. 62, 2005.
- [23] M. Blott, L. Halder, M. Leeser, and L. Doyle, "Qutibench: Benchmarking neural networks on heterogeneous hardware," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 4, pp. 1–38, 2019.
- [24] M. Bouet and V. Conan, "Mobile edge computing resources optimization: A geo-clustering approach," *IEEE Trans. Netw. Serv. Manage.*, vol. 15, no. 2, pp. 787–796, Jun. 2018.
- [25] T. Braud, F. H. Bijarbooneh, D. S. Chatzopoulos, and P. Hui, "Future networking challenges: The case of mobile augmented reality," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 1796–1807.
- [26] T. Braud, P. Zhou, J. Kangasharju, and P. Hui, "Multipath computation offloading for mobile augmented reality," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2020, pp. 1–10.
- [27] M. Breitbach, J. Edinger, S. Kaupmees, H. Trötsch, C. Krupitzer, and C. Becker, "Voltaire: Precise energy-aware code offloading decisions with machine learning," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2021, pp. 1–10.
- [28] K. Chen, T. Li, H.-S. Kim, D. E. Culler, and R. H. Katz, "Marvel: Enabling mobile augmented reality with low energy and low latency," in *Proc. 16th ACM Conf. Embedded Netw. Sensor Syst.*, 2018, pp. 292–304.
- [29] Y. Chen, N. Zhang, Y. Zhang, and X. Chen, "Dynamic computation offloading in edge computing for Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4242–4251, Jun. 2019.
- [30] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proc. 6th Conf. Comput. Syst.*, 2011, pp. 301–314.
- [31] L. Corneo et al., "Surrounded by the clouds: A comprehensive cloud reachability study," in *Proc. Web Conf.*, 2021, pp. 295–304.
- [32] L. Corneo et al., "(How much) Can edge computing change network latency?," in *Proc. IFIP Netw. Conf.*, 2021, pp. 1–9.
- [33] V. Cozzolino, J. Ott, A. Y. Ding, and R. Mortier, "Ecco: Edge-cloud chaining and orchestration framework for road context assessment," in *Proc. IEEE/ACM 5th Int. Conf. Internet Things Des. Implementation*, 2020, pp. 223–230.
- [34] E. Cuervo et al., "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst. Appl. Serv.*, 2010, pp. 49–62.
- [35] E. Cuervo et al., "Kahawai: High-quality mobile gaming using GPU offload," in *Proc. 13th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2015, pp. 121–135.
- [36] K. Dang, N. Mohan, L. Corneo, A. Zavodovski, J. Ott, and J. Kangasharju, "Cloudy with a chance of short RTTs: Analyzing cloud connectivity in the Internet," in *Proc. Internet Meas. Conf.*, 2021, pp. 62–79.
- [37] K. Deb, "Multi-objective optimization," in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. New York, NY, USA: Springer, 2014, pp. 403–449.
- [38] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.
- [39] U. Drolia et al., "The case for mobile edge-clouds," in *Proc. IEEE 10th Int. Conf. Ubiquitous Intell. Comput. 10th Int. Conf. Auton. Trusted Comput.*, 2013, pp. 209–215.
- [40] M. Eder et al., "Surrounded by the clouds, 2021. [Online]. Available: <https://mediatum.ub.tum.de/1593899>
- [41] J. Flinn and Z. M. Mao, "Can deterministic replay be an enabling tool for mobile computing?," in *Proc. 12th Workshop Mobile Comput. Syst. Appl.*, 2011, pp. 84–89.
- [42] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "[COMET]: Code offload by migrating execution transparently," in *Proc. 10th USENIX Symp. Operating Syst. Des. Implementation*, 2012, pp. 93–106.
- [43] T. Guo, "Cloud-based or on-device: An empirical study of mobile deep inference," in *Proc. IEEE Int. Conf. Cloud Eng.*, 2018, pp. 184–190.
- [44] Y. Guo and C. Liang, "Blockchain application and outlook in the banking industry," *Financial Innov.*, vol. 2, no. 1, 2016, Art. no. 24.
- [45] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "McDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proc. 14th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2016, pp. 123–136.
- [46] Z. Han, H. Tan, G. Chen, R. Wang, Y. Chen, and F. C. M. Lau, "Dynamic virtual machine management via approximate Markov decision process," in *Proc. IEEE INFOCOM 35th Annu. Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
- [47] T. He, H. Khamfroush, S. Wang, T. L. Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 365–375.
- [48] J. Hu, K. Li, C. Liu, and K. Li, "Game-based task offloading of multiple mobile devices with QoS in mobile edge computing systems of limited computation capacity," *ACM Trans. Embedded Comput. Syst.*, vol. 19, no. 4, pp. 1–21, Jul. 2020.
- [49] C.-L. Hwang and A. S. M. Masud, *Multiple Objective Decision Making—Methods and Applications: A State-of-the-Art Survey*, Berlin, Germany: Springer, 2012.
- [50] A. Ignatov et al., "Ai benchmark: Running deep neural networks on android smartphones," in *Proc. Eur. Conf. Comput. Vis.*, 2018.
- [51] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Trans. Cloud Comput.*, vol. 5, no. 4, pp. 725–737, Oct.–Dec. 2017.
- [52] Y. Kang et al., "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, 2017.
- [53] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 48–56, Jun. 2017.
- [54] M. Kinatader and K. Rothermel, "Architecture and algorithms for a distributed reputation system," in *Proc. Int. Conf. Trust Manage.*, 2003, pp. 1–16.
- [55] C. Li, G. Dobler, X. Feng, and Y. Wang, "Tracknet: Simultaneous object detection and tracking and its application in traffic video analysis," 2019, *arXiv:1902.01466*.
- [56] F. Liu et al., "Gearing resource-poor mobile devices with powerful clouds: Architectures, challenges, and applications," *IEEE Wireless Commun.*, vol. 20, no. 3, pp. 14–22, Jun. 2013.
- [57] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, 2019, pp. 1–16.
- [58] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2018, pp. 756–764.
- [59] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit.*, 2014, pp. 301–312.



- [60] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surv. Tut.*, vol. 19, no. 3, pp. 1628–1656, Oct.–Dec. 2017.
- [61] S. Maheshwari, D. Raychaudhuri, I. Seskar, and F. Bronzino, "Scalability and performance evaluation of edge cloud systems for latency constrained applications," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2018, pp. 286–299.
- [62] J. Meng, W. Shi, H. Tan, and X. Li, "Cloudlet placement and minimum-delay routing in cloudlet computing," in *Proc. 3rd Int. Conf. Big Data Comput. Commun.*, 2017, pp. 297–304.
- [63] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2019, pp. 2287–2295.
- [64] N. Mohan and J. Kangasharju, "Edge-fog cloud: A distributed cloud for internet of things computations," in *Proc. Cloudification Internet Things*, 2016, pp. 1–6.
- [65] N. Mohan and J. Kangasharju, "Placing it right!: Optimizing energy, processing, and transport in edge-fog clouds," *Ann. Telecommun.*, vol. 73, no. 7–8, pp. 463–474, 2018.
- [66] N. Mohan, L. Corneo, A. Zavodovski, S. Bayhan, W. Wong, and J. Kangasharju, "Pruning edge research with latency shears," in *Proc. 19th ACM Workshop Hot Top. Netw.*, 2020, pp. 182–189.
- [67] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate IoT edge computing with lightweight virtualization," *IEEE Netw.*, vol. 32, no. 1, pp. 102–111, Jan./Feb. 2018.
- [68] A. Nikraves, D. R. Choffnes, E. K.-Bassett, Z. M. Y. Mao, and M. Welsh, "Mobile network performance from user devices: A longitudinal, multidimensional analysis," in *Proc. Int. Conf. Passive Act. Netw. Meas.*, 2014, pp. 12–22.
- [69] Z. Ning, J. Liao, F. Zhang, and W. Shi, "Preliminary study of trusted execution environments on heterogeneous edge platforms," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2018, pp. 421–426.
- [70] J. Pan, J. Wang, A. Hester, I. Alqerm, Y. Liu, and Y. Zhao, "Edgechain: An edge-IoT framework and prototype based on blockchain and smart contracts," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4719–4732, Jun. 2019.
- [71] H. Possegger, T. Mauthner, P. M. Roth, and H. Bischof, "Occlusion geodesics for online multi-object tracking," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 1306–1313.
- [72] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2018, pp. 1421–1429.
- [73] T. S. Rappaport *et al.*, "Millimeter wave mobile communications for 5G cellular: It will work!," *IEEE Access*, vol. 1, pp. 335–349, 2013.
- [74] S. Russell and P. Norvig, "Artificial intelligence: A modern approach," 2002.
- [75] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *Proc. IEEE Trust-com/BigDataSE/ISPA*, 2015, pp. 57–64.
- [76] J. Sachs, G. Wikstrom, T. Dudda, R. Baldemair, and K. Kittichokechai, "5G radio network design for ultra-reliable low-latency communication," *IEEE Netw.*, vol. 32, no. 2, pp. 24–31, Mar./Apr. 2018.
- [77] O. Sahin and A. K. Coskun, "Providing sustainable performance in thermally constrained mobile devices," in *Proc. 14th ACM/IEEE Symp. Embedded Syst. Real-Time Multimedia*, 2016, pp. 72–77.
- [78] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
- [79] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct.–Dec. 2009.
- [80] R. Shea, D. Fu, and J. Liu, "Rhizome: Utilizing the public cloud to provide 3D gaming infrastructure," in *Proc. 6th ACM Multimedia Syst. Conf.*, 2015, pp. 97–100.
- [81] S. Shi and C.-H. Hsu, "A survey of interactive remote rendering systems," *ACM Comput. Surv.*, vol. 47, no. 4, pp. 1–29, 2015.
- [82] RIPE NCC Staf, "Ripe atlas: A global internet measurement network," *Internet Protoc. J.*, vol. 18, no. 3, 2015.
- [83] X. Sun and N. Ansari, "EdgeIoT: Mobile edge computing for the Internet of Things," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 22–29, Dec. 2016.
- [84] H. Tan, Z. Han, X.-Y. Li, and F. C. M. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2014, pp. 1–9.
- [85] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. IEEE INFOCOM 35th Annu. Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
- [86] R. Trivisonno, R. Guerzoni, I. Vaishnavi, and D. Soldani, "Towards zero latency software defined 5G networks," in *Proc. IEEE Int. Conf. Commun. Workshop*, 2015, pp. 2566–2571.
- [87] B. Varghese *et al.*, "Revisiting the arguments for edge computing research," *IEEE Internet Comput.*, vol. 25, no. 5, pp. 36–42, Sep./Oct. 2021.
- [88] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Comput.*, vol. 3, no. 6, pp. 76–83, Nov./Dec. 2016.
- [89] F. Wang, M. Zhang, X. Wang, X. Ma, and J. Liu, "Deep learning for edge computing applications: A state-of-the-art survey," *IEEE Access*, vol. 8, pp. 58 322–58 336, 2020.
- [90] S. Wang *et al.*, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, 2018, pp. 63–71.
- [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 Commun. Mag.*, vol. 55, no. 11, pp. 80–85, Nov. 2017.
- [92] Y. Wang, X. Tao, X. Zhang, P. Zhang, and Y. T. Hou, "Cooperative task offloading in three-tier mobile computing networks: An ADMM framework," *IEEE Trans. Veh. Technol.*, vol. 68, no. 3, pp. 2763–2776, Mar. 2019.
- [93] J. Wu, C. Yuen, N.-M. Cheung, J. Chen, and C. W. Chen, "Enabling adaptive high-frame-rate video streaming in mobile cloud gaming applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 12, pp. 1988–2001, Dec. 2015.
- [94] C. Xia, J. Zhao, H. Cui, X. Feng, and J. Xue, "DNNtune: Automatic benchmarking DNN models for mobile-cloud computing," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 4, pp. 1–26, Dec. 2019.
- [95] Z. Xiao, Z. Xia, H. Zheng, B. Y. Zhao, and J. Jiang, "Towards performance clarity of edge video analytics," 2021, *arXiv:2105.08694*.
- [96] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "LAVEA: Latency-aware video analytics on edge computing platform," in *Proc. 2nd ACM/IEEE Symp. Edge Comput.*, 2017, pp. 1–13.
- [97] W. Zhang, B. Han, and P. Hui, "On the networking challenges of mobile augmented reality," in *Proc. Workshop Virtual Reality Augmented Reality Netw.*, 2017, pp. 24–29.
- [98] J. Zhou, T. Wang, P. Cong, P. Lu, T. Wei, and M. Chen, "Cost and makespan-aware workflow scheduling in hybrid clouds," *J. Syst. Archit.*, vol. 100, 2019, Art. no. 101631.
- [99] A. Zuniga *et al.*, "Tortoise or hare? Quantifying the effects of performance on mobile app retention," in *Proc. World Wide Web Conf.*, 2019, pp. 2517–2528.



**Vittorio Cozzolino** is currently working toward the PhD degree from the Chair of Connected Mobility, the Technical University of Munich. His research interests include distributed systems, edge and cloud computing, resource management, and virtualization.



**Leonardo Tonetto** is currently working toward the PhD degree from the Chair of Connected Mobility, Technical University of Munich. His research interests include mobile user behavioral modeling, human mobility, and complex networks.



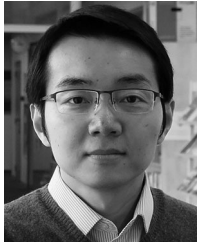


**Dr. Nitinder Mohan** received the MTech degree (Hons.) from the Indraprastha Institute of Information Technology Delhi (IIIT-D), India and the PhD degree (Marie Curie ITN fellow) from the Department of Computer Science, University of Helsinki, Finland. He is currently a postdoctoral researcher with the Chair of Connected Mobility, Technical University of Munich, Germany. He was a visiting researcher with NEC Labs Europe, Germany and with the University of Göttingen, Germany and a project scientist with the Indian Institute of Technology Delhi (IIT-D), India. He was the recipient of the “Outstanding Ph.D. Dissertation Award” by IEEE Technical Committee on Scalable Computing (TCSC).



**Jörg Ott** received the diploma degree in computer science from TU Berlin in 1991, the diploma degree in industrial engineering from TFH Berlin in 1995 and the PhD degree in 1997. He explores edge and in-network computing as well as decentralized services, with a particular interest in privacy. His research interests include network architectures, protocols, and algorithms for connecting mobile nodes to the Internet and to each other. Since August 2015, he has been with the Chair of Connected Mobility, Faculty of Informatics, Technische Universität München.

► **For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/csdl](http://www.computer.org/csdl).**



**Aaron Yi Ding** received the MSc and PhD degrees from the Department of Computer Science, University of Helsinki. He is currently a tenured faculty leads with Cyber-Physical Intelligence Lab, TU Delft and an associate professor (docent) in computer science with the University of Helsinki. With more than 14 years of R&D experience across EU, U.K., and USA, he and Joerg Ott were with TU Munich, University of Cambridge with Jon Crowcroft, and Columbia University with Henning Schulzrinne. He has authored or coauthored more

than 60 peer reviewed publications. His research focuses on Edge AI solutions for cyber-physical systems in smart health, mobility, and energy domains. He is an associate editor for *ACM Transactions on Internet of Things*, *IEEE Open Journal of Intelligent Transportation Systems*, and *Multimodal Technologies and Interaction*. He was the recipient of best paper awards and recognition from ACM SIGCOMM, ACM EdgeSys, ACM SenSys CCIoT, IEEE INFOCOM, and the esteemed Nokia Foundation Scholarships.