Voltaire: Precise Energy-Aware Code Offloading Decisions with Machine Learning

Martin Breitbach*, Janick Edinger†, Siim Kaupmees‡, Heiko Trötsch*, Christian Krupitzer§ and Christian Becker*
*University of Mannheim, Germany, {martin.breitbach, heiko.troetsch, christian.becker}@uni-mannheim.de
†University of Hamburg, Germany, edinger@informatik.uni-hamburg.de
‡University of Cambridge, United Kingdom, siim_kaupmees@hotmail.com
§University of Hohenheim, Germany, christian.krupitzer@uni-hohenheim.de

***Abstract*—Code offloading enables resource-constrained devices to leverage idle computing power of remote resources. In addition to performance gains, offloading helps to reduce energy consumption of mobile devices, which is a key challenge in pervasive computing research and industry. In today's distributed computing systems, the decision whether to execute a task locally or remotely for minimal energy usage is non-trivial. Uncertainty about the task complexity and the result data size require a careful offloading decision. In this paper, we present *Voltaire*—a novel scheduler for sophisticated energy-aware code offloading decisions. *Voltaire* applies machine learning methods on crowd-sourced data about past executions to accurately predict the complexity and the result data size of an upcoming task. Combining these predictions with device-specific energy profiles and context knowledge allows *Voltaire* to estimate the energy consumption on the mobile device. Thus, *Voltaire* makes well-informed offloading decisions and carefully selects local or remote execution based on the expected energy consumption. We integrate *Voltaire* into the Tasklet distributed computing system and perform extensive experiments in a real-world testbed. Our results with three real-world applications show that *Voltaire* reduces the energy usage of task executions by 12.5% compared to a baseline scheduler.**

***Index Terms*—energy-aware code offloading, mobile ad-hoc computing, machine learning, Tasklet system**

I. Introduction

The demand for computational power of software in areas such as virtual reality, machine learning, or image processing increases. In addition, battery constraints have become a key challenge in pervasive computing with the advent of mobile computing and smartphones [1]. As a solution, code offloading enables applications to execute computationally intensive parts on remote resource providers in cloud, grid, or edge environments. These providers return the results via the network. In addition to performance gains, code offloading helps to reduce the energy consumption of mobile devices if the cost of transferring a task and receiving the results is lower than the cost of a local execution. Thus, a remote execution is in general beneficial for computationally intensive tasks with small input and result data. Analogously, less complex tasks with large input and result data are to be executed on the mobile device.

In modern edge computing environments, the decision whether to execute a task remotely or locally is non-trivial due to uncertainty. First, in contrast to a traditional batch system, the completion time of a task is not known a priori. Second, similar to the completion time, the size of the execution output might vary for each execution. This results in different transmission costs. Both task completion time and result size may even change for different executions of the same source code, as varying parameters and input data have a considerable influence. Third, important context variables such as connection type or bandwidth change frequently and need to be monitored at runtime [2].

Energy-aware offloading has received much attention in research [3]. Most notably, in *MAUI*, Cuervo *et al.* show the effect of different network interfaces on the energy consumption [4]. In *CloneCloud*, Chun *et al.* apply dynamic profiling to create profile trees that explore the impact of input parameters on the energy consumption of the task [5]. In *ThinkAir*, Kosta *et al.* build upon the two previous approaches and add elasticity and scalability of the remote resources [6]. However, none of the approaches focuses on reducing the uncertainty in the prediction of the energy consumption from the devices's view. Instead, rather straightforward estimators are applied.

In this paper, we propose *Voltaire* — a novel scheduler for sophisticated energy-aware offloading decisions in modern distributed computing systems. *Voltaire* is a centralized scheduler that applies machine learning methods for regression analysis based on crowd-sourced data of past executions of similar tasks. This enables *Voltaire* to accurately predict the complexity and the result size of an upcoming task. In addition, *Voltaire* integrates device-specific energy profiles, which model the influence of CPU and network activity on the energy consumption. With further context knowledge about the input data size of a task and the current bandwidth, *Voltaire* is able to dynamically decide for each specific task whether to offload or execute locally. Thus, *Voltaire* improves energy consumption from the perspective of a mobile device rather than a global system perspective for energy efficiency. We integrate an implementation of *Voltaire* into the *Tasklet System* [7] — a middleware-based code offloading system. We deploy *Voltaire* in the real world to extensively evaluate its effectiveness in realistic settings with three user-facing applications. The experiments show that *Voltaire* is able to predict the complexity and the result size of a task precisely. We observe that *Voltaire* reduces the energy consumption for task execution by 12.5% compared to the existing scheduler of the *Tasklet System*.

In the remainder of this paper, we discuss related work (II), give an overview of the system model and motivate *Voltaire*'s design (III), present *Voltaire* in detail (IV), evaluate its performance (V), and conclude the paper (VI).

II. Related Work

Previous approaches estimate context parameters, profile applications, and monitor network states to find partitioning and offloading strategies that "*make smartphones last longer*" [4, p.1]. A recent comprehensive survey of offloading approaches in edge computing can be found in [3]. Despite all these valuable achievements, several issues remain unresolved and are, therefore, addressed in this paper.

Multiple existing approaches do not consider all relevant context dimensions or make assumptions about them that conflict with real-world applications. Examples can be found in [5], [8], and [9] that consider the size of the result as static, which does not hold true for many real-world applications. Previous works differ in their approaches to predict unknown context dimensions such as the complexity of tasks or the required data transfer. Some approaches perform a static code analysis without taking the effect of input parameters into account [10]. Several strategies include the computation of averages or linear models based on past executions [4], [11], [12]. We argue that machine learning can help to figure out complex relationships between input parameters and the estimates that go beyond simple dependencies [13], [14].

Typically, the offloading decision is made on the mobile device. As this causes additional overhead and thus energy consumption, the decision making process is often kept as lightweight as possible. A remote decision making allows for more complex algorithms [21], [31]. An additional benefit of remote decision making is the number of samples that the scheduler can observe to create execution statistics for an application. A central decision maker can apply crowdsourcing and collect usage data from multiple devices. Only a subset of prior approaches is evaluated in real-world testbeds with actual energy monitoring. This makes the results which are gained through carefully set-up simulations complex to translate to actual energy savings (e.g., [22] and [27]). Most approaches are not integrated into existing offloading frameworks which raises questions about their generalizability and applicability in the physical world [17], [24], [25]. Further, some of the previous works do not run real applications for the evaluations [28], [29]. Table I summarizes existing works.

In this paper, we propose *Voltaire*— a scheduler for energy-aware code offloading that applies machine learning methods on crowdsourced data to make well-informed offloading decisions. In addition, it monitors multiple context dimensions such as device type, bandwidth, and input data size to accurately decide whether to offload a task or to execute it locally. We integrate *Voltaire* into the *Tasklet System* to evaluate its effectiveness with real-world applications in a realistic setting.

TABLE I
Overview of related energy-aware offloading approaches. (Dec. = Decision making, Eval. = Evaluation)
● fulfilled ○ partially fulfilled

		Context				**Prediction**					**Dec.**			**Eval.**		
Author/*System*	**Year**	**Task complexity**	**Data transfer**	**Network**	**Device**	**Static analysis**	**Cost graphs/tables**	**Averages**	**Linear models**	**Machine learning**	**Mobile device**	**Cloud/Broker**	**Crowdsourcing**	**Real-world testbed**	**Offloading framework**	**Real-world applications**
Othman [15]	1998	●		●	●			●			●					●
Rudenko [16]	1998													●		●
Spectra [8]	2002	●	○	●	●				●		●			●	●	●
Xian [17]	2007	●			●				●		●			●		●
MAUI [4]	2010	●	○	●	●			●			●	●		●	●	●
CloneCloud [5]	2011	●									●	●		●	●	●
Giurgiu [18]	2012	●		●	●						●	●		●	●	●
ThinkAir [6]	2012	●	○	●	●						●			●		●
MACS [11]	2012	●	○	○	○	●		●			●			●	●	●
SP Energizer [19]	2013	●	●	●	●					●	●	●		●	●	●
TDM [20]	2013	●	●	●	●		●				●			●		●
Magurawalage [21]	2014	○	○	○	○				●		●	●				
Niu [22]	2014	○				●	●				●					
Ravi [10]	2014				○	●					●			●		●
DREAM [23]	2015	○	○	●	●			●	●		●			●		●
Jade [24]	2015	○	○	●	●		●				●			●		●
Saab [25]	2015		○	●	●						●			●		●
EECOF [26]	2015		○	●							●			●		●
Zhou [12]	2015	●	○	●	●			●			●	●		●		●
Wu [27]	2016	●	○	●			●				●					
EECO [28]	2016	●	●	●	●						●					
EA-OSGi [29]	2017	○	○	○	○			●			●					
Karim [13]	2017		○							●	●					
Crutcher [14]	2017	○	●							●	●					
EMCO [30]	2018	○	○	●	○					●	●	●	●	●	●	●
Jadad [31]	2018	●		●	○		●	●				●				
Hao [32]	2018	●	○	●					●			●				
Lyu [33]	2018	●	○	●	○				●			●				
HetNet [34]	2019	○	○	●					●			●				
Nguyen [35]	2019	○	●	●					●							
Xu [36]	2019	●	●	●		●										
Voltaire		●	●	●	●			●		●		●	●	●	●	●

III. System Model

Voltaire is a scheduler for an environment where a mobile device offloads tasks to remote resource providers to reduce local battery consumption. A mobile device called resource consumer runs several applications which issue tasks. Tasks logically consist of a *type*, i.e., underlying source code, and *parameters*. Additionally, tasks may require *input data*, such as images for face recognition, that have to be transferred to the providers. Consumers and providers use a middleware that orchestrates the offloading process. Providers run (multiple) process-level virtual machines (VMs) that interpret bytecode. We assume that consumers also run such VMs. Thus, a local execution is possible. After the execution of all bytecode instructions, the provider sends the *result* (data) back to the consumer in case of a remote execution. In the system, a central broker performs resource management, i.e., decides whether and whereto offload a task. In practice, several brokers may create a network of brokers, each serving a subset of all

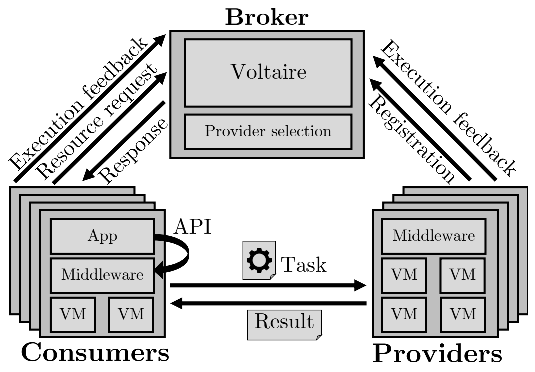


Fig. 1. Schematic overview of an offloading system that runs *Voltaire*.

devices. Providers register at the broker and periodically send heartbeats. Figure 1 illustrates the system model.

Voltaire reduces the energy consumption of the consumer device by deciding whether a task is offloaded or not. In contrast to related work that minimizes the total energy consumption of a whole distributed computing system [28], [37], we focus on improving the energy consumption of a battery-constrained consumer device only, since modern cloud or edge computing environments typically contain offloading targets with constant power supply.

Voltaire suggests to offload a task when the energy required for remote execution is expected to be smaller than the energy required for a local execution, i.e. $E_{offload} < E_{local}$. The energy consumption for a local execution can be approximated by the task complexity measured by the number of bytecode instructions that have to be executed (I_{task}) and the average energy required for a single bytecode instruction ($\overline{E_i}$) in mJ. The energy consumption for a remote execution is determined by the size of the input data in bytes (D_{input}), the result size in bytes (D_{result}), the inbound and outgoing bandwidths β_{in} and β_{out} of the consumer device in $\mathrm{bytes/s}$, and the power for data transmission and data reception in mW ($P_{transmit}$ and $P_{receive}$). Thus, the offloading decision is:

$$\frac{D_{input}}{\beta_{out}} * P_{transmit} + \frac{D_{result}}{\beta_{in}} * P_{receive} < I_{task} * \overline{E_i} \quad (1)$$

Precisely knowing all the variables prior to task execution is not feasible in practice due to three reasons. First, the number of bytecode instructions and the size of the result data need to be predicted for each upcoming task execution. Due to a different parametrization, even tasks of the same type may vary in their number of executed instructions as well as in the sizes of their result data. Second, the energy consumption for sending and receiving data, as well as for executing code is highly device-dependent. Third, pervasive computing systems can be highly mobile, resulting in constantly changing context situations. Even a stationary system might be affected by the dynamic nature of its environment and experience context changes such as a varying bandwidth.

We design *Voltaire* on the basis of Equation 1. *Voltaire*'s goal is to determine accurate values for all variables in Equation 1 (cf. Table II). This results in three tasks:

Predicting the number of bytecode instructions and the result size. Prior to task execution, the application informs *Voltaire* about task type, parameters, and size of the input data D_{input}. Based on crowd-sourced data of past executions of the same task, *Voltaire* predicts the number of bytecode instructions I_{task}, as well as the size of the result data D_{result}. In Section IV-A, we present how *Voltaire* integrates different regression analysis methods from machine learning for this.

Integrating device-dependent energy profiles. The power for sending and receiving data ($P_{transmit}$ and $P_{receive}$) and the average cost for executing a bytecode instruction locally ($\overline{E_i}$) depend on the consumer device that issues the task and the context. In Section IV-B, we show how *Voltaire* uses device-specific energy profiles to retrieve these values.

Estimating inbound and outgoing bandwidth. *Voltaire* estimates inbound bandwidth β_{in} and outgoing bandwidth β_{out} with low overhead by analyzing past data transmissions in the system. Section IV-C describes this in more detail.

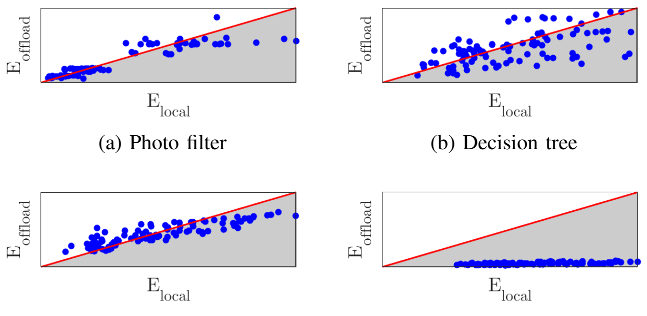
TABLE II
VARIABLES INFLUENCING THE OFFLOADING DECISION (SEE EQ. 1)

Variable	Definition	In
D_{input}	Size of the input data in bytes	known a priori
D_{result}	Size of the result data in bytes	Section IV-A
I_{task}	Number of bytecode instructions of a task	Section IV-A
$\overline{E_i}$	Average energy consumption of 1 bytecode instruction on the consumer in mJ	Section IV-B
$P_{transmit}$	Energy consumption per s data transmission in mW	Section IV-B
$P_{receive}$	Energy consumption per s data reception in mW	Section IV-B
β_{out}	Outgoing bandwidth in $\mathrm{bytes/s}$	Section IV-C
β_{in}	Inbound bandwidth in $\mathrm{bytes/s}$	Section IV-C

IV. VOLTAIRE- AN ENERGY-AWARE SCHEDULER FOR PRECISE OFFLOADING DECISIONS

Voltaire is a centralized scheduler that runs on the broker of a distributed computing system. A centralized approach avoids massive communication overhead and is able to collect and reason on more data. Additionally, running on the broker allows *Voltaire* to apply more complex prediction and decision making mechanisms compared to approaches such as [4] and [19], where the offloading decision is made by the consumer device itself and kept simple to save energy. As a central instance for resource management, the broker is supposed to run on a stable and powerful machine, which is in most cases connected to a constant power supply. *Voltaire*'s computationally intensive parts (e.g., updating the machine learning models) run mostly asynchronous to the offloading process, which reduces the influence on task completion times.

As *Voltaire* uses machine learning techniques that may be computationally-intensive, a scalable deployment is important. Thus, it might be necessary to replicate *Voltaire* on several cloud or edge servers, which is a non-trivial task. Further options are to reduce the number of machine learning models per application, to update the models less frequently, or to



(a) Photo filter (b) Decision tree

(c) Speech detection (d) Image Rendering

Fig. 2. Four real-world applications from our experiments. Each point represents the local energy consumption (E_{local}) and the remote energy consumption ($E_{offload}$) of a task execution. Thus, points in the gray area should be executed remotely and points above the line locally. *Voltaire* is especially attractive for applications (a) to (c) that sometimes benefit from a remote and sometimes from a local execution, based on task and context.

limit the training data sets to a certain size. The realization of scalability is part of future work.

For each task, the mobile device sends a request to the broker. The placement of the broker thus influences task completion times. *Voltaire* analyzes the incoming request and decides whether to execute the task on the consumer device's VMs or remotely. Figure 2 compares the energy consumption on the consumer device of four real-world applications from our experiments when executed locally or remotely. We observe that some applications such as the ray tracing-based image rendering always benefit from a remote (or local) execution. Due to its accurate predictions, *Voltaire* is, however, especially attractive for applications that sometimes benefit from a remote execution and sometimes from a local execution, depending on the task and the context. Here, offloading decisions are particularly complex. *Voltaire*'s offloading decision is not only based on the knowledge of previous task executions from this particular mobile device but from all resource consumers together. *Voltaire* thus exploits crowd-sourced data for the decision. The broker chooses an appropriate provider for remote execution if *Voltaire* suggested a remote execution. The exact choice of the provider is out of scope of this paper and does not affect the consumer's energy consumption. In previous work, we have researched this provider selection extensively [7], [38], [39]. In this section, we show how *Voltaire* performs three basic tasks to determine the required variables for a well-informed offloading decision from Equation 1 and Table II.

A. Predicting number of bytecode instructions and result size

The three variables (i) number of bytecode instructions I_{task}, (ii) input data size D_{input}, (iii) and result size D_{result} differ for every task execution. Whereas the input data size D_{input} is known beforehand and reported from the consumer to *Voltaire*, *Voltaire* needs to predict the number of bytecode instructions I_{task} and the result size D_{result}. We propose to leverage feedback from past executions of the same task type to predict these two variables. Several executions of the same code may still vary considerably in the number of instructions that are executed and the result size due to different parameters and input data. To collect a larger data basis, *Voltaire* crowd-sources information about all previous executions of this task type in the whole system instead of only considering past executions by the current consumer device. The providers report a feedback after task execution to *Voltaire* including the task type, the number of bytecode instructions, and the result size. Crowd-sourced information is helpful for the current decision since number of instructions and result size are device-independent. Other devices may run the same application and, hence, execute tasks of the same type, which is valuable data for prediction. This is especially effective if the behavior of a user barely deviates from the usual usage of an application. Many applications, such as a photo filter or speech recognition, even do not allow much variance in their use in the first place. *Voltaire* integrates three methods for prediction based on the crowd-sourced data. In addition to the (i) average of prior executions and (ii) exponential smoothing, we propose to use regression analysis (iii) as a machine learning method to make precise predictions. In the following, we describe the prediction of bytecode instructions. The prediction of the result size is done analogously.

1) Average: Similar to related work such as *MAUI* [4] or *MACS* [11], *Voltaire* integrates a strategy that predicts the number of bytecode instructions based on the average of past executions of the same task type. This method works well when the number of bytecode instructions is mostly independent from the parameters or the input data.

2) Exponential smoothing: Exponential smoothing [40] is a statistical method where previous executions are weighed less the older they are. It generally performs well in cases with periodic fluctuations in the data [41]. This can be expected for the number of bytecode instructions as consecutive tasks of a similar type often originate from a single application that is likely to start similar tasks. Let S_n be the prediction for instructions executed for the n'th execution and I_n be the corresponding actual number of executions completed. The parameter α with $0 < \alpha < 1$ determines how much weight is assigned to the previous, true value.

$$I_{task} = S_n = \begin{cases} I_0 & \text{for } n = 1 \\ S_{n-1} * \alpha + I_{n-1} * (1 - \alpha) & \text{for } n \neq 1 \end{cases} \quad (2)$$

3) Regression analysis: In experiments with real-world applications, we observe that executions of the same source code still vary considerably in terms of required bytecode instructions. We identify three reasons for these deviations. First, the size of the input data may be different. Classifying a small data set, for instance, requires less instructions than applying the same classifier to a larger data set. Second, the parameters of the particular task execution have a considerable influence. For example, calculating whether a number — which is passed as a parameter — is prime requires on average more instructions the higher this number is. Third, the special characteristics of the input data may affect the number of bytecode instructions. For instance, a photo filter task that converts all colors but red to grayscale, may run considerably shorter if there are only a few red pixels in the image. To learn these complex influences of parameters and input data on the bytecode instructions, we integrate a machine learning module into *Voltaire*. For each task type, *Voltaire* trains three types of regression models based on these three reasons for deviations.

T1: Prediction based on the input data size. This type of regression model uses the input data size as a single feature to predict the number of bytecode instructions. Some tasks perform operations on all elements of the input data, e.g., on all pixels of a bitmap. In these cases, the number of instructions may be well predictable based on the input data size of the upcoming execution and the knowledge from past executions.

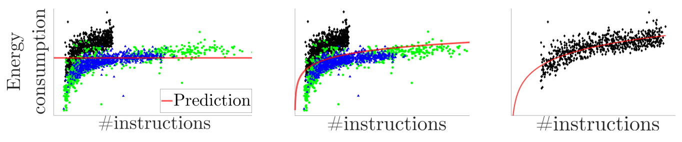
T2: Prediction based on the parameters. In addition to the input data size as a feature, *Voltaire* uses each parameter as a separate feature for this type of model. The input data size and the parameters of earlier executions are known to *Voltaire* anyways, which eliminates additional effort for developers that is required for models of type `T3`.

T3: Prediction based on application-specific features. Experiments with real-world applications revealed that the number of bytecode instructions is highly dependent on the characteristics of the input data in some cases. Accurate predictions are difficult in these cases without any domain knowledge about the task. As an extreme and rather theoretical example, a task may perform a complex operation on every pixel of a bitmap, but only if the first pixel has a certain RGB value. *Voltaire*'s models of type `T3` allow the developer to pass further customized machine learning features for these complex cases. In Section V-A, we show how *Voltaire*'s prototypical implementation offers this interface to application developers. Some features may be easily computed, while more sophisticated ones may require more computation and thus more energy on the mobile device. A calculation on, e.g., only a fraction of the input data is therefore recommended. In our experiments, we use 0.1% of the input data.

Voltaire performs online learning for all three types of models. When new feedback is available, *Voltaire* has a new training sample, consisting of the parameters, the input size, the number of bytecode instructions, the result size, and — if provided by the application developer — additional features. *Voltaire* uses multiple regression models, including decision tree, random forest, gradient boosting, linear regression, adaboost, and bayesian ridge. It periodically performs a 5-fold cross validation for the different regression models on the data and stores the best model for each task type.

B. Integrating device-dependent energy profiles

Voltaire uses device-specific energy profiles to retrieve the average energy consumption of executing one bytecode instruction locally ($\overline{E_i}$) and the energy consumption of sending and receiving data for one second ($P_{transmit}$ and $P_{receive}$) in the offloading process. All three parameters are device-dependent. The energy profile of a smartphone, for instance, differs from the energy profile of a laptop or a Raspberry Pi. In this paper, we perform hardware-based profiling for



(a) Energy profile `P1` (b) Energy profile `P2` (c) Energy profile `P3`

Fig. 3. Three methods to create CPU energy profiles for *Voltaire* based on (a) a constant value for the individual device type, (b) a fitted curve for the individual device type, and (c) a fitted curve for the individual application. The values shown are from exemplary measurements in the evaluation. Tasks from the same application are depicted in the same color.

each device type. For cases where hardware measurements are infeasible such as certain wearables, software-based profiling approaches (e.g., [42]) are viable alternatives. In future work, a combination of hardware- and software-based profiling may further improve accuracy, as, e.g., different operating system versions on devices with the same hardware lead to different energy profiles. *Voltaire* uses two energy profiles per device type: a *CPU energy profile* and a *network energy profile*.

1) CPU energy profile: This profile models the energy consumption while performing a single bytecode instruction on a local VM ($\overline{E_i}$). Our experiments show a tradeoff between accuracy of the profile and the effort for profile creation. We thus propose three approaches to create such profiles.

P1: A constant value for the individual device type. We first assume that each bytecode instruction consumes the same amount of energy. In this case, a CPU energy profile may consist of a single value $\overline{E_i}$ that is device-dependent. Thus, one energy measurement with arbitrary applications is required for each device type. This method (cf. Figure 3a) is rather simple and thus applicable with low overhead in real-world systems.

P2: A fitted curve for the individual device type. During our measurements, we observe that the average energy consumption of a single bytecode instruction depends on the total number of instructions. Shorter tasks with less instructions also require less energy per instruction, whereas longer tasks with a high number of instructions require more energy per instruction. To include such effects, we propose to run multiple applications and measure the energy consumption of a single instruction depending on the overall number of instructions of a task. The value for $\overline{E_i}$ is then calculated as a function of the (predicted) number of instructions of a task as $f_{instr,device}(I_{task})$, where $f_{instr,device}$ is a device-dependent function fitted on the measurement data (cf. Figure 3b).

P3: A fitted curve for the individual application. We also observe that the energy consumption of a fixed number of bytecode instructions varies across applications. Due to a more complex underlying interpretation by the VM, some bytecode instructions run longer and thus consume more energy than others. This has a measurable effect on the energy consumption of different applications as the frequency with which certain instructions are used, varies. Therefore, we propose to perform energy measurements for each application separately as the most accurate method to create a CPU energy profile (cf. Figure 3c). Whereas methods `P1` and `P2` only require one measurement per device type, this method requires $a * d$ measurements in total, where a is the number of applications and d is the number of device types. New devices are introduced comparably rarely to the market, while new applications could be written by any developer at any time. Hence, we believe that in practice, $a >> d$ holds. To eliminate the need to perform energy measurements for each new application, we propose an alternative approach. We observe a linear relation between task completion time and energy consumption in our experiments. Thus, the average energy consumption of a single bytecode instruction of a new application $\overline{E}_{i,new}$ can be approximated by:

$$\overline{E}_{i,new} = \overline{E}_{i,bench\_device} * \frac{r_{instr,new}}{r_{instr,bench\_app}} \quad (3)$$

where $\overline{E}_{i,bench\_device}$ is the average energy consumption per instruction of a benchmark device, $r_{instr,new}$ the number of instructions of the new application executed per second on the benchmark device, and $r_{instr,bench\_app}$ the number of instructions of a benchmark application executed per second on the benchmark device[1]. With this approach, no energy measurements are required for a new application. New applications only have to be executed on a benchmark device with known energy consumption, which is feasible in real-world use cases.

2) Network energy profile: The network energy profile models the energy consumption of a certain device per second while transmitting and receiving data ($P_{transmit}$ and $P_{receive}$). Here, the energy consumption is independent from the type of data that is transmitted or received. Thus, the network energy profile is application-independent. As shown in Figure 4, the network energy profile ideally contains separate energy values for transmitting and receiving under varying bandwidths.

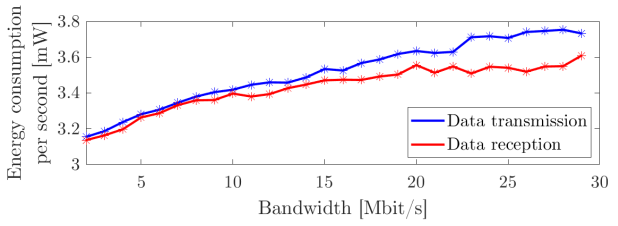


Fig. 4. Network energy profile based on exemplary measurements in the evaluation. Separate models for transmitting and receiving data characterize the respective energy consumption per second dependent on the bandwidth.

C. Estimating inbound and outgoing bandwidth

Inbound and outgoing bandwidth β_{in} and β_{out} have to be measured at runtime as they fluctuate [43]. *Voltaire* uses an approach with low overhead. Consumers estimate the current bandwidth by measuring the duration of transmitting and receiving input and result data of known size. Thus, no additional data transfers are required. The bandwidth estimations are piggybacked to the task request, such that the broker — and, hence, also *Voltaire*— is informed about the recent value.

[1]$\overline{E}_{i,new}$ and $\overline{E}_{i,bench\_device}$ are both functions of the total number of instructions as in `P2`. We omit this in the equation for better readability.

V. Evaluation

In this section, we evaluate *Voltaire* in a real-world testbed. First, we show how we integrate *Voltaire* into the *Tasklet System*. Second, we describe the experimental setup. Then, we perform three experiments that evaluate (i) the effectiveness of regression analysis for the prediction of the number of bytecode instructions and the result size, (ii) the influence of different device energy profiling methods on prediction quality, and (iii) *Voltaire*'s potential to reduce energy consumption.

A. Voltaire as an extension of the Tasklet System

We integrate *Voltaire* into the *Tasklet System* [7] — a middleware-based code offloading system. *Voltaire* is deployed on the broker entity of the *Tasklet System*, which is already used for resource management. Developers may write applications that use the *Tasklet System* in their favorite language. Either via a well-defined API [38] that is available for several popular programming languages or via socket-based interprocess communication, the applications interact with the *Tasklet Middleware*. The *Tasklet System* offers developers to set *Quality of Computation* (QoC) goals to tailor the execution to the requirements of their application. *Voltaire* extends the *Tasklet System* with the *Energy QoC* mechanism. In that way, developers are able to choose between the task scheduler of the *Tasklet System*, which aims for fast task execution and *Voltaire*, which reduces the energy consumption.

```
public int[] calculatePrimes(int lower, int upper) {
        Tasklet t = new Tasklet("primes.cmm");
        t.addInt("lowerBound", lower);
        t.addInt("upperBound", upper);
        t.setQoCReliable(GUARANTEED);
        t.setQoCEnergy();
        TaskletID id = t.start();
        int[] primes = TaskletResults.get(id);
        return primes;}
```

Listing 1. An exemplary Java application that uses the *Tasklet System* to offload a prime number calculation. The highlighted line shows how developers can activate *Voltaire* via the provided API.

Listing 1 shows an exemplary Java application from [44] that uses the *Tasklet System* Java API to offload a prime number calculation. Developers only have to set the Energy QoC goal with the highlighted line of code to activate *Voltaire*. They may conveniently pass parameters to the `setQoCEnergy()` method for prediction approach `T3`. *Voltaire* requires crowd-sourced data about past task executions (cf. Section IV-A). We extend the *Tasklet System* to perform the crowd data collection at the broker, which runs *Voltaire*. In the *Tasklet System*, providers inform the broker after task execution about the *TVM* that has become idle. We piggyback the task type, number of instructions, and result size to these messages to collect data for *Voltaire* with minimal overhead.

B. Experimental setup

We apply *Voltaire* in a real-world distributed computing system that runs the *Tasklet Middleware*. A Raspberry Pi 4B with a 1.5 GHz ARM Quad Core CPU offloads tasks to providers. The Raspberry Pi works battery-powered in many

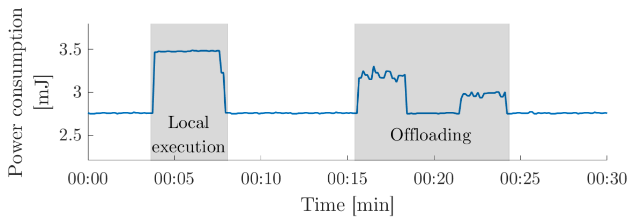


Fig. 5. Exemplary energy consumption of the evaluation device.

use cases and is predestined for code offloading due to its limited computational power. Since we aim to improve the battery consumption of the consumer device, the choice of the provider devices does not affect the measurements. A UM34C digital voltmeter measures the energy consumption of the consumer device. In the following, we isolate and discuss the energy consumption of the applications by subtracting the idle energy consumption from the energy consumption during the local or remote execution. The devices communicate over an IEEE 802.11n Wifi network. The net bit rate achieved at the application layer is $10\,\mathrm{Mbit/s}$. We experimented with different setups (e.g., closer to the access point) and observed similar effects. Figure 5 shows an excerpt of a typical energy measurement during the evaluation.

We run three real-world applications in this setup: (i) a grayscale photo filter, (ii) a decision tree classifier, and (iii) a speech detection. All applications are offloaded to remote devices or executed locally on *Tasklet Virtual Machines*. The grayscale photo filter may be used in photo editing or social media apps. Users choose RGB color ranges (e.g., via a range slider) and the filter application converts all pixels of an image that differ from the specified color range to grayscale. We collect a database of 3,000 smartphone images of different sizes that are edited with the filter application in the experiments. The decision tree classifier is useful in, e.g., pervasive healthcare use cases. For instance, it may perform classification on physiological data collected by wearables or smartphones to determine the health condition of a user. In case of a remote execution, this application offloads a trained decision tree model and a data set to a provider. In the evaluation, we use 3,000 randomly created decision tree models with 2,500 to 5,000 nodes. Additionally, we created 3,000 random data sets with 1,000,000 to 5,000,000 samples and 4 to 25 features each. The speech detection application identifies the periods of an audio file where people talk, similar to, e.g., Matlab's `voiceActivityDetector`. This application may be the basis for many pervasive applications such as speech enhancement for accessibility, speech coding, and speech recognition. We create 3,000 audio files, ranging from 15 to $150\,\mathrm{s}$, from a total of $2.5\,\mathrm{h}$ of audio recordings that we collected in a room with 5 people doing a group work.

C. Predicting number of bytecode instructions and result size

In the first experiment, we evaluate the performance of regression analysis for predicting the number of bytecode instructions I_{task} and the result size D_{result} of an upcoming task. Thus, we isolate *Voltaire*'s machine learning part and predict the two variables with different regression methods. For each method, we evaluate the three model types `T1` to `T3` with 3,000 tasks per application. Table III describes the features that were used. For each application, we perform a 5-fold cross validation with Python's scikit-learn [45] library. Table IV shows the respective R^2 scores for the prediction of the number of bytecode instructions. We omit the values for the prediction of the result size here for reasons of clarity and comprehensibility. In general, the result size is well-predictable by *Voltaire* as it is often directly dependent on the input data size (e.g., for the photo filter).

TABLE III
MACHINE LEARNING FEATURES USED FOR REGRESSION ANALYSIS

Photo filter	
`T1`	Input data size
`T2`	Input data size, photo width, photo height, 9 parameters that describe the RGB range that remains in the original color
`T3`	Features T2, 4 ratios of pixels that need to be converted (according to the R, G, B, and RGB values) from a sample of 0.1 percent of all pixels
Decision tree classifier	
`T1`	Input data size
`T2`	Input data size, tree size, #rows, #features, #tree nodes
`T3`	Features T2, coefficient describing the balance of the tree, tree height, #nodes in the tree/maximum #nodes in a tree of this height
Speech detection	
`T1`	Input data size
`T2`	Input data size, amplitude threshold, #values skipped after speech is detected
`T3`	Features T2, 12 features from a sample of 0.1 percent of the overall audio file (ratio of sample values above threshold, average amplitude, a histogram with 10 groups showing the amplitude distribution of the sample)

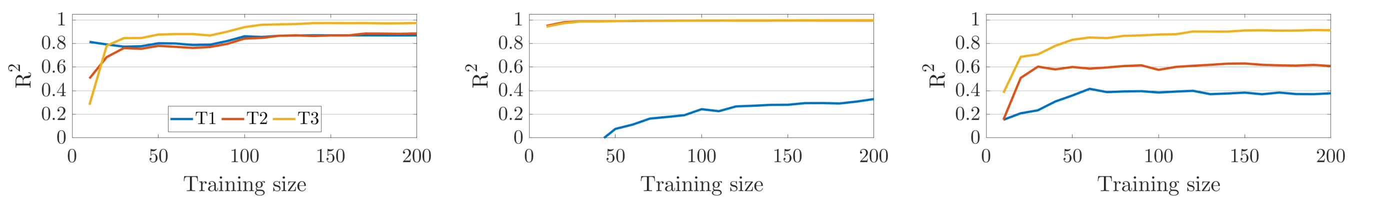
The R^2 scores provide four insights. First, we observe that the quality of the prediction improves for all applications when extending the feature set from `T1` to `T3`. The additional features of `T2` and `T3` help *Voltaire* to better learn the reasons for the behavior of certain tasks.

Second, our experiment reveals that the performance of model type `T1` varies considerably for different applications. The number of bytecode instructions for executing a photo filter task is strongly related to the input data size as the algorithm traverses — and potentially converts — each pixel. Thus, model type `T1` performs comparably well here. Tasks that originate from the decision tree classifier are difficult to predict for model type `T1`. For this application, the input data size can be approximated by $r * f$, where r is the number of rows and f is the number of features in the input data set that has to be classified. The same classifier, however, requires considerably more bytecode instructions to classify a data set with many samples but a few features in comparison to a data set with few samples and many features.

Third, we observe that both photo filter and decision tree classifier are well-predictable with models of type `T2`. Only the speech detection application requires additional, application-specific features. We therefore argue that *Voltaire* is able to predict the number of bytecode instructions for many tasks very precisely, without requiring any additional programming effort by the application developer. *Voltaire*'s models of type `T2` only use input data size and parameters, which are reported to the broker anyways, as features for

TABLE IV
AVERAGE R^2 SCORES FOR DIFFERENT REGRESSION ANALYSIS METHODS IN A 5-FOLD CROSS VALIDATION ON ALL 3,000 TASKS PER APPLICATION

App/Model	**Decision Tree**			**Random Forest**			**Gradient Boosting**			**Linear Regression**			**Adaboost**			**Bayesian Ridge**		
	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3	T1	T2	T3
Photo filter	.838	.834	.967	.857	.916	.985	.860	.919	.986	.875	.901	.945	.872	.905	.968	.875	.875	.875
Decision tree classifier	-.191	.994	.997	.129	.996	.998	.398	.997	.999	.411	.997	.999	.406	.994	.994	.411	.997	.997
Speech detection	.324	.347	.865	.363	.650	.947	.483	.681	.960	.508	.682	.929	.501	.661	.884	.508	.644	.644



(a) Photo filter (b) Decision tree (c) Speech detection

Fig. 6. Effect of training data size on R^2 score for three types of gradient boosting models (T1 = prediction on input data size, T2 = prediction on input data size and parameters, T3 = prediction on input data size, parameters, and application-specific features). Crowdsourcing increases the training size in practice.

an accurate prediction. However, we also acknowledge that *Voltaire* requires additional domain knowledge to deliver high-quality predictions for tasks such as the speech detection tasks. Thus, it is important to offer application developers an easy-to-use API to pass such features to *Voltaire*.

Fourth, this experiment shows that *Voltaire* is able to predict the number of instructions with high confidence when using models of type `T3`, which is an important step towards precise energy-aware offloading decisions. The cross validation reveals that gradient boosting is the most accurate regression method for these applications. In the following experiments, we therefore only show the results for gradient boosting.

Instead of only analyzing past executions of the same task on the same device, *Voltaire* integrates crowd-sourced data from all prior executions of the task in the system. Figure 6 depicts the R^2 score of the gradient boosting models of type `T1` to `T3` as a function of the number of training data samples. The models are tested on data from 1,000 task executions for each application. We observe that the prediction quality improves with an increasing number of samples. Especially an accurate prediction of the speech detection application requires more samples ($R^2 = .909$ for 200 training samples and $R^2 = .960$ from the cross validation on 3,000 samples from Table IV). Thus, we conclude that applying a crowd-sourcing strategy is able to improve *Voltaire*'s prediction quality.

D. Integrating device-dependent energy profiles

Voltaire uses device-specific CPU and network energy profiles. In practice, *Voltaire* estimates the energy for local and remote execution based on the predictions of the regression analysis, i.e., the quality of both regression analysis and energy profiling has an influence on the quality of *Voltaire*'s offloading decisions. In this experiment, we isolate the effect of the energy profiling methods on this process. To achieve this, we calculate the estimated energy of local and remote execution under the assumption that the regression method predicted the correct number of instructions and result size. We compare this estimation with the true energy consumption of local and remote executions of the task in Figure 7.

We observe that the quality of the estimation of a local execution depends on the CPU energy profiling method. Method `P3` performs best for all applications. The profile bases on application-specific measurements and is thus more accurate than the more generic approaches `P1` and `P2`. We therefore recommend to create custom CPU energy profiles for each application with method `P3` to unleash *Voltaire*'s full potential. Whether method `P1` or `P2` performs better, depends on the particular application and its similarity to the applications used to create the profiles. We additionally observe that the energy consumption of a remote execution based on profiled values for data transmission $P_{transmit}$ and data reception $P_{receive}$ is well-predictable (see right part of Figure 7).

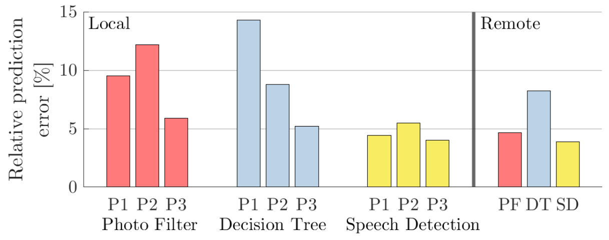


Fig. 7. Average deviation of the estimations for the energy consumption of local and remote executions with different strategies. This experiment assumes that the number of instructions is predicted correctly.

E. Evaluating Voltaire's energy-saving potential

In the final experiment, we evaluate *Voltaire*'s energy-saving potential by applying it to 10 workflows of around 2.5 h each. Each workflow consists of 300 tasks that include 100 photo filter, 100 decision tree classifier, and 100 speech detection tasks. Figure 8 summarizes the average energy consumption per task across all workflows. Table V provides an overview

TABLE V
SUMMARY OF THE EXPERIMENTS (EXP. SM. = EXPONENTIAL SMOOTHING, GB `T3` = GRADIENT BOOSTING WITH MODEL TYPE `T3`)

Appl.	**Energy consumption per task** (mJ)						**Improvement to local execution** (%)						**Correct decisions** (%)					
	Loc.	**Rem.**	**Avg.**	**Exp. Sm.**	**GB `T3`**	**Ideal**	**Loc.**	**Rem.**	**Avg.**	**Exp. Sm.**	**GB `T3`**	**Ideal**	**Loc.**	**Rem.**	**Avg.**	**Exp. Sm.**	**GB `T3`**	**Ideal**
Photo filter	8210	8131	8296	8260	7340	7203	-	1.0	-1.0	-0.6	10.6	12.3	50.5	49.5	45.8	47.0	86.7	100
Decision tree classifier	5828	5942	5181	5224	4885	4876	-	-2.0	11.1	10.3	16.2	16.3	53.6	46.4	71.6	70.5	96.2	100
Speech detection	8196	7770	8087	8087	7225	7144	-	5.2	1.3	1.3	11.8	12.8	43.3	56.7	47.1	47.2	86.6	100
Total	7411	7281	7188	7190	6483	6408	-	1.8	3.0	3.0	12.5	13.5	49.1	50.9	54.8	54.9	89.8	100

of the energy consumption per application, the relative improvement, and the number of correct offloading decisions in comparison to an ideal scheduler. The existing scheduler of the *Tasklet System* offers the two strategies to execute all tasks remotely or to execute all tasks locally. We observe that for the 3,000 tasks in the evaluation, a remote execution is on average preferable to a local execution. Simply offloading all tasks, however, only decreases the energy consumption by 1.8%. We therefore conclude that task-dependent, context-aware, and precise offloading decisions are necessary to realize the full energy-saving potential of code offloading.

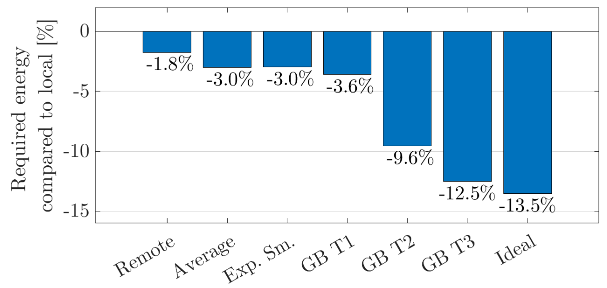


Fig. 8. Average improvement of the energy consumption by *Voltaire* in comparison to the status quo in the *Tasklet System*. Exp. Sm. = exponential smoothing, GB `T1` = gradient boosting with model type `T1`.

A first step is to use the average of past executions as a predictor of upcoming executions, which was proposed in related literature [4], [11] and which is also possible with *Voltaire*. We additionally apply an exponential smoothing method with $\alpha = 0.5$ as a logical extension that exploits the similarity of consecutive executions. Both strategies perform better than the existing scheduler of the *Tasklet System* with energy savings of 3.0%. Section V-C leads to the conclusion that *Voltaire* is able to accurately predict the number of bytecode instructions and the result size of an upcoming task. This final experiment shows that the accurate prediction, together with precise device profiling, has a considerable, positive effect on the energy consumption. *Voltaire*'s regression analysis with gradient boosting of model type `T3` reduces the energy consumption by 12.5% in comparison to the status quo. This is an improvement of 9.0% compared to related work that makes offloading decisions based on the average of past executions. *Voltaire* achieves an energy consumption that is on average only 1.0% worse than a hypothetical, ideal scheduler that always makes the correct decision. Of 3,000 tasks, a small subset of 10.2% of the tasks are incorrectly executed on the local or a remote device. We further observe in Table V that *Voltaire*'s performance is especially beneficial for the photo filter and the decision tree classifier, which confirms the results of Section V-C. Figure 8 underlines that *Voltaire*'s prediction based on parameters and application-specific features is a considerable improvement to a prediction solely on the input data size.

F. Threats to Validity

In this paper, we focus on the energy savings of code offloading from a device-driven perspective. If response times are important, offloading has further advantages that should be taken into account in the decision making such as the choice of powerful providers [7]. A thorough analysis of the interplay of energy awareness and response times is part of future work. We perform measurements in a real-world testbed with the *Tasklet System*. Therefore, measuring errors cannot be ruled out completely. Additionally, future work may include an evaluation with other device types, bandwidths, system load, or connection types such as Bluetooth. The same applies to the choice of machine learning techniques and energy profiling methods, which was extensive but not exhaustive.

VI. CONCLUSION

This paper presents *Voltaire* — a scheduler for sophisticated energy-aware offloading decisions. *Voltaire* decides whether local or remote execution is beneficial for the energy consumption of a mobile device depending on the current context. Based on crowd-sourced data of past executions, *Voltaire* is able to accurately predict the complexity and the result size of an upcoming task with regression methods from machine learning. We show that *Voltaire* reduces the energy consumption for task execution by 12.5% compared to the status quo in an existing code offloading system.

As future work, we plan to investigate the effectiveness of our approach in a large-scale simulation-based study. This simulation will also be used for a thorough investigation of the interplay of energy awareness and response time requirements. In addition, we will enhance *Voltaire* with a module that identifies similar task types and uses those runtime characteristics for further increasing the accuracy of the predictions. Another future research direction is to investigate fairness and incentive mechanisms for a real-world deployment, which may induce new attack vectors such as the manipulation of the crowd-sourced data to obtain more paid tasks.

ACKNOWLEDGMENT

This work was supported by the German Research Foundation (DFG), project number 323412509.

References

[1] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Pers. Commun.*, vol. 8, no. 4, 2001.

[2] C. Becker and G. Schiele, "Middleware and application adaptation requirements and their support in pervasive computing," in *Proc. ICDCSW*. IEEE, 2003, pp. 98–103.

[3] C. Jiang et al., "Energy aware edge computing: A survey," *Comput. Commun.*, vol. 151, 2020.

[4] E. Cuervo et al., "MAUI: Making smartphones last longer with code offload," in *Proc. MobiSys*. ACM, 2010.

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic execution between mobile device and cloud," in *Proc. EuroSys*. ACM, 2011.

[6] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. INFOCOM*. IEEE, 2012.

[7] D. Schäfer, J. Edinger, J. M. Paluska, S. VanSyckel, and C. Becker, "Tasklets : "Better than Best-Effort" Computing," in *Proc. ICCCN*, 2016.

[8] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *Proc. ICDCS*. IEEE, 2002.

[9] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Trans. Wirel. Commun.*, vol. 11, no. 6, 2012.

[10] A. Ravi and S. K. Peddoju, "Handoff Strategy for Improving Energy Efficiency and Cloud Service Availability for Mobile Devices," *Wirel. Pers. Commun.*, vol. 81, 2015.

[11] D. Kovachev and R. Klamma, "Framework for Computation Offloading in Mobile Cloud Computing," *Int. J. Artif. Intell. Interact. Multimed.*, vol. 1, no. 7, 2012.

[12] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, "A Context Sensitive Offloading Scheme for Mobile Cloud Computing Service," in *Proc. CLOUD*. IEEE, 2015.

[13] S. A. Karim and J. J. Prevost, "A machine learning based approach to mobile cloud offloading," in *2017 Computing Conference*. IEEE, 2017.

[14] A. Crutcher, C. Koch, K. Coleman, J. Patman, F. Esposito, and P. Calyam, "Hyperprofile-based computation offloading for mobile edge networks," in *Proc. MASS*. IEEE, 2017.

[15] M. Othman and S. Hailes, "Power conservation strategy for mobile computers using load sharing," *ACM SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 2, no. 1, 1998.

[16] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "Saving portable computer battery power through remote process execution," *ACM SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 2, no. 1, 1998.

[17] C. Xian, Y.-H. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *Proc. ICPADS*. IEEE, 2007.

[18] I. Giurgiu, O. Riva, and G. Alonso, "Dynamic Software Deployment from Clouds to Mobile Devices," in *Proc. Middlew.* ACM/IFIP/USENIX, 2012.

[19] A. Khairy, H. H. Ammar, and R. Bahgat, "Smartphone Energizer: Extending Smartphone's battery life with smart offloading," in *Proc. IWCMC*. IEEE, 2013.

[20] Y. D. Lin, E. T. Chu, Y. C. Lai, and T. J. Huang, "Time-and-energy-aware computation offloading in handheld devices to coprocessors and clouds," *IEEE Syst. J.*, vol. 9, no. 2, 2015.

[21] C. M. Sarathchandra Magurawalage, K. Yang, L. Hu, and J. Zhang, "Energy-efficient and network-aware offloading algorithm for mobile cloud computing," *Comput. Networks*, vol. 74, 2014.

[22] J. Niu, W. Song, and M. Atiquzzaman, "Bandwidth-adaptive partitioning for distributed execution optimization of mobile applications," *J. Netw. Comput. Appl.*, vol. 37, 2014.

[23] J. Kwak, Y. Kim, J. Lee, and S. Chong, "DREAM: Dynamic Resource and Task Allocation for Energy Minimization in Mobile Cloud Systems," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 12, 2015.

[24] H. Qian and D. Andresen, "An energy-saving task scheduler for mobile devices," in *Proc. ICIS*. IEEE, 2015.

[25] S. A. Saab, F. Saab, A. Kayssi, A. Chehab, and I. H. Elhajj, "Partial mobile application offloading to the cloud for energy-efficiency with security measures," *Sustain. Comput. Informatics Syst.*, vol. 8, no. 2015, 2015.

[26] M. Shiraz, A. Gani, A. Shamim, S. Khan, and R. W. Ahmad, "Energy Efficient Computational Offloading Framework for Mobile Cloud Computing," *J. Grid Comput.*, vol. 13, no. 1, 2015.

[27] H. Wu, W. Knottenbelt, K. Wolter, and Y. Sun, "An optimal offloading partitioning algorithm in mobile cloud computing," in *Proc. QEST*. Springer, 2016.

[28] K. Zhang et al., "Energy-Efficient Offloading for Mobile Edge Computing in 5G Heterogeneous Networks," *IEEE Access*, vol. 4, 2016.

[29] S.-J. Lee and X. Lin, "Energy-aware paired sampling-based decision model for dynamic mobile-to-mobile service offloading," *IEEE Access*, vol. 5, 2017.

[30] H. Flores et al., "Evidence-Aware Mobile Computational Offloading," *IEEE Trans. Mob. Comput.*, vol. 17, no. 8, 2018.

[31] H. Jadad, A. Touzene, K. Day, and N. Alzeidir, "A cloud-side decision offloading scheme for mobile cloud computing," *International Journal of Machine Learning and Computing*, vol. 8, no. 4, 2018.

[32] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoneim, "Energy efficient task caching and offloading for mobile edge computing," *IEEE Access*, vol. 6, 2018.

[33] X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Zhang, and R. P. Liu, "Energy-efficient admission of delay-sensitive tasks for mobile edge computing," *IEEE Trans. Commun.*, vol. 66, no. 6, 2018.

[34] S. Li, Y. Tao, X. Qin, L. Liu, Z. Zhang, and P. Zhang, "Energy-aware mobile edge computation offloading for iot over heterogenous networks," *IEEE Access*, vol. 7, 2019.

[35] T. T. Nguyen, L. Le, and Q. Le-Trung, "Computation Offloading in MIMO Based Mobile Edge Computing Systems Under Perfect and Imperfect CSI Estimation," in *IEEE Trans. Serv. Comput.*, 2019.

[36] X. Xu et al., "An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks," *J. Netw. Comput. Appl.*, vol. 133, 2019.

[37] W. Zhang, Y. Wen, and H.-H. Chen, "Toward transcoding as a service: energy-efficient offloading policy for green mobile cloud," *IEEE Network*, vol. 28, no. 6, 2014.

[38] J. Edinger, D. Schäfer, M. Breitbach, and C. Becker, "Developing Distributed Computing Applications with Tasklets," in *Proc. PerCom Workshops*. IEEE, 2017.

[39] J. Edinger, S. VanSyckel, C. Krupitzer, J. M. Paluska, and C. Becker, "Developing a QoS-based Tasklet trading system," in *Proc. PerCom Workshops*. IEEE, 2014.

[40] R. G. Brown and R. F. Meyer, "The Fundamental Theorem of Exponential Smoothing," *Oper. Res.*, vol. 9, no. 5, 1961.

[41] S. Casolari and M. Colajanni, "On the Selection of Models for Runtime Prediction of System Resources," in *Run-time Models for Self-managing Systems and Applications*, 2010.

[42] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "Software-based energy profiling of Android apps: Simple, efficient and reliable?" in *Proc. SANER*. IEEE, 2017.

[43] R. Süselbeck, G. Schiele, P. Komarnicki, and C. Becker, "Efficient Bandwidth Estimation for Peer-to-Peer Systems," in *Proc. P2P*. IEEE, 2011.

[44] D. Schäfer, J. Edinger, C. Becker, and M. Breitbach, "Writing a Distributed Computing Application in 7 Minutes with Tasklets," in *Proc. Middleware Posters and Demos*. ACM, 2016.

[45] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, 2011.