Energy Budget Distribution in Mobile Edge Computing Applications

1st Khadija Akherfi
School of Computation, Information and Technology
Technical University of Munich
Munich, Germany
k.akherfi@tum.de

2nd Michael Gerndt
School of Computation, Information and Technology
Technical University of Munich
Munich, Germany
gerndt@in.tum.de

***Abstract*—Although the performance of mobile devices (MDs) has been steadily improving, advanced applications might suffer long latency for complex functions as well as from draining the battery when continuously used. Function offloading to cloud servers is a proposed technique to solve these challenges. Recently edge computing enables low latency access to resources and, in combination with Function-as-a-service (FaaS), is an obvious target for offloading from MDs.**

In this paper we introduce a framework for FaaS applications on MDs and function offloading to a FaaS Edge implementation. The scheduling of function invocations to local or remote resources is determined by function-specific offloading policies. The policies are selected periodically according to historic data and function specific performance and energy models. The periodic planning considers the distribution of the application's energy budget across a number of periods and adapts the function-specific policies for a period. This enables more time consuming planning and fast decision making for individual function invocations. Our results validate the efficiency of the proposed approach, showing about 33% reduction in energy consumption.

***Index Terms*—Edge Computing, Energy, Budget, Mobile Cloud Computing.**

I. Introduction

With the advancements in cellular technologies and mobile applications, our daily activities, such as smart healthcare and smart driving, are increasingly reliant on mobile devices (MDs). These applications and others including augmented reality (AR), natural language processing, face recognition, and interactive gaming, are not only computation-intensive and energy-demanding but also require significant storage and rapid execution [1]. Consequently, running such applications entirely on a MD can significantly drain the battery. To address this, one solution is to offload tasks to an edge computer. We can envisage an overall Mobile Edge Computing platform (MEC) (Figure1) consisting of the MDs and edge servers located at GSM towers. Offloading can be done to the nearest edge server or to an edge server at another GSM tower in the proximity.

Serverless computing or Function-as-a-Service (FaaS) is increasingly used for event-driven applications in the cloud. In cloud computing, the term "serverless" refers to architectures where the clients do not have to manage own servers. It is offered as Function-as-a-Service (FaaS), where function invocations are triggered by events and are executed on a FaaS platform. The application owner provides the functions and the application logic and does not have to manage the servers where the function invocations are executed. All big cloud providers offer FaaS platforms and also open source platforms such as OpenFaaS, OpenWisk and Knative are available. Recently, light weight implementation such as Knative and OpenFaaS are increasingly used to offer FaaS on the edge as well.

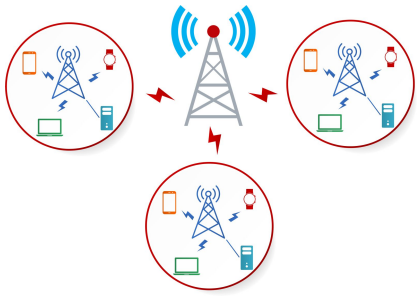

Figure 1: Mobile Edge Computing Architecture

This paper focuses on the mobile side in the offloading scheme. Our mobile applications are written in the FaaS paradigm implemented in Java. The execution of the application on the MD thus leads to function invocations triggered by events such as a button click. For an invocation the decision has to be taken, whether to offload the invocation or to run it local. Obviously, decision taking has to be fast. For that we use function specific *offloading policies*. The used policies are periodically generated by a set of *decision strategies* that are optimized for selecting the best policy for the functions. The decision strategies are applied rarely and thus can run complex and time consuming algorithms. The decision strategy selects for example the local execution policy, if it is required to meet a deadline and energy is available on the mobile device. Otherwise, it will select, for example, the offloading policy if for energy saving purposes, the function should be executed remotely. The policy will be applied until the strategy will revisit the decision.

We assume that an overall energy budget for an application is given, e.g., for executing a game in the next hour an energy budget of 50% of the battery capacity is allocated.

979-8-3503-9281-4/25/$31.00 ©2025 IEEE

This budget is then broken down into a sequence of periods and the decision strategy selects an offloading policy for each function such that the period budget is not exceeded. Before each new period, the selected offloading policies are revisited.

In this work, we consider offloading the function invocations to an edge-cloud FaaS platform which manages the deployment and allocation of resources for the functions. We assume here, that the function will be executed on the edge server. In case of a shortage in resources in the edge server, the function will be executed in the cloud server.

The main contributions of this paper are summarized as follows:

- We extend the FaaS paradigm to the mobile device.
- We then introduce a decision strategy that determines the offloading policy for each function such that the energy budget is met.
- We perform experiments to assess the performance our proposed technique.

The rest of this paper is organized as follows: Section II reviews already existing offload approaches in related works. Section III presents our system design by discussing the planning phase including energy budget distribution and policy selection. Section IV presents decision strategies used on the mobile device. Section V illustrates experimental results. Finally, Section VI concludes the work done in this paper.

II. Related Work

A substantial body of research has explored the integration of cloud and edge computing to enhance the speed of computing and storage services delivered to endpoint devices [2]–[5]. This collaboration harnesses the advantages of both technologies by allocating computational tasks to edge devices while relying on cloud capabilities for storage and processing needs requiring more substantial resources. For example, a particular study on the Internet of Vehicles (IoV) addressed the challenge of optimal edge server placement, focusing on goals such as reducing transmission delays and improving energy efficiency [6]. In the context of the Internet of Things (IoT), a comprehensive review of IoT offloading strategies and platforms, including edge and fog computing tailored to or integrated with IoT, has been presented in [7]. Moreover, investigations have been made into the combined optimization of content caching and simultaneous task offloading [3]. Current research on the offloading decision-making problem in MEC utilizes traditional optimization methods. Lyapunov optimization algorithm solves the optimization problem while ensuring stability of task queues [4], [8], [9]. These methods show good results for various environments. The research paper [2] discusses the advantages of mobile edge computing, which allows user equipment (UE) to offload computationally intensive tasks to a nearby server, thereby enhancing their processing capabilities. The authors highlights the challenge of selecting the optimal components to offload while considering the user equipment performance. They propose a method called EPED (Energy and Performance Efficient Deep Learning-based Offloading Algorithm) for selecting a combination of components to offload, aiming to minimize execution time and energy consumption. This proposed method uses a deep learning-based algorithm designed to optimize energy efficiency for offloading tasks. The presented algorithm uses a cost function to model and train the deep learning network to determine the optimal schemes based on performance metrics. EPED works by designing a cost function for each task, while considering energy consumption, execution time, and server resource usage. Yang et al. [10] describe computational offloading in Mobile Edge Computing as an optimization challenge and explore strategies for selecting the optimal Mobile Edge Server (MES) for offloading. They use Markov decision process (MDP) framework which takes into account both the mobility of users and the diversity of MESs. To determine the optimal offloading time, the reinforcement learning (RL) with value iteration is applied to solve the MDP. In recent years, research has focused on machine learning-based intelligent algorithms. [11] introduces Serverledge, a Function-as-a-Service (FaaS) platform designed to integrate decentralized control with geographically distributed infrastructures. It leverages the capability to offload computation, capitalizing on the extensive resources available in the Cloud. Li et al. [5] proposes a model-free deep reinforcement learning (DRL) algorithm called Energy-aware Task Offloading with Deadline Constraint (DRL-E2D) for multi-eNB mobile edge computing environment. The introduced algorithm aims to maximize rewards while adhering to task deadlines. It employs an actor-critic framework and use k-nearest neighbor (K-NN)-based action representation to handle the large discrete action space. In mobile edge computing, the available studies focused on different algorithms used to offload tasks while minimizing energy or execution time. [12] presents a combination of arithmetic optimization with quantum computing to handle the NP-hard problem. It uses a Markov model to improve decision-making in offloading. [13] proposes a task offloading and resource allocation model using proximal policy optimization to manage heterogenous user demands. In our presented work, we concentrate on mobile applications where functions are written in event based programming and propose an approach to distribute energy budgets for functions based on their requirements. This ensures efficient resource allocation within the mobile device and support better offloading decision-making on invocation basis.

III. Design And System Architecture

Our system has two main parts: the mobile device and the edge server. The mobile device processes a FaaS application for a specific period of time with the constraint of staying within a given application energy budget. Edge servers run a FaaS platform to which functions can be offloaded. Due to the limited capacity of edge servers we also consider overflow to cloud FaaS platforms. The primary objective is to develop a budget-based offloading scheme that selects for each function on the mobile device the best resource under the constraint of

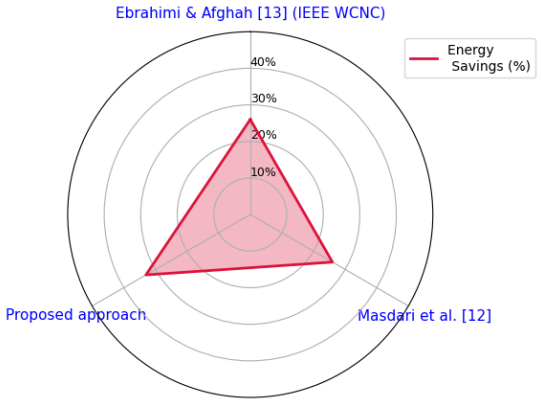

Figure 2: Average Energy Savings of Offloading Methods.

staying within the given energy budget for an application run on the mobile device.

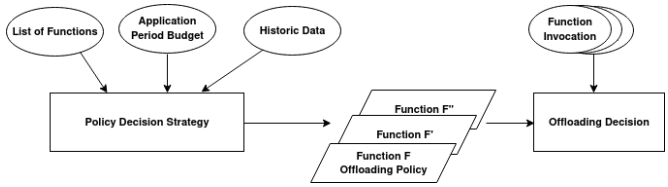

Figure 3: Overview of budget-based offloading decision making.

Figure 3 introduces the flow of the main concepts in our approach. The entire application execution period is broken into a sequence of periods, e.g., five minutes. The overall budget is distributed over all periods and the period energy budget, the list of used functions, and historic data are the basis for the decision strategy. It will select at the start of a period the offloading policy that will be applied to function invocations in that period. The selection of the policy is based on a performance and energy model of the function. For that we distinguish three different classes leading to different models:

- ***Input independent function:*** the input does not have an impact on the execution time of the function.
- ***Input value dependent function:*** the input value of invoked function influences the execution time and it is dependent on the value of the input parameters.
- ***Input size dependent function:*** e.g the size of the array or the size in bytes determine the computation time.

As an example, let's consider a mobile application that is composed of a set of Java functions. A user can specify the time during which the application should run (usage time) and also can specify the *energy budget* for that usage time. The energy budget will then be distributed over budget periods. At the beginning of a period, we run a decision strategy that determines the period energy budget and the offloading policy for each function. The decision strategy might use a function priority to distribute the period budget over the functions. It then selects the policy to meet the function's budget.

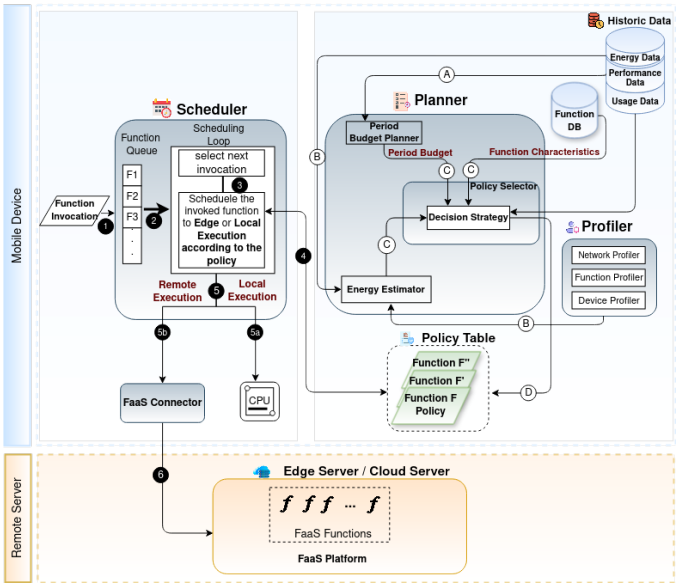

Figure 4: Overall System Architecture: The scheduler gets the function invocation and decides where to run it based on the function's active policy. The FaaS Connector takes care of connecting the MD to the edge server.

Policies are divided into three types:

- Local Policy: for both input value dependent and input independent functions.
- Local with threshold policy: A threshold is given as a parameter to this policy. Function invocation with arguments above the threshold run remotely the others locally. This affects the computation of remaining period budget. Since the input value and input size are not constant, the decision-making will be affected.
- Offload Policy: for both input value dependent and input independent functions.

The decisions strategies are based on the predicted execution time and the call frequency of functions in the upcoming period. They extrapolate the information from previous periods, i.e., historic data. Furthermore, they consider function priorities given by the user and energy consumption if run locally on the mobile device.

Figure 4 provides an overview of the system architecture components. The two main components are the scheduler and the planner.

- *Scheduler*: receives the function invocation (step ❶), adds the invoked function to the queue (step ❷). Next, it selects the function from the queue (step ❸) and schedules where the function should be executed based on the function's active policy (step ❹). After that, the function is executed(step 5) either locally (step ❺a) or is offloaded to the edge server(step 6) via the FaaS Connector (step ❺b).
- *Planner*: is an active component that periodically determines the offloading policy for each function.

- *Energy Estimator*: determines the energy required to run a given function based on the call frequency and the energy per local invocation.
- *Period Budget Planner*: It computes the budget for the next period. It estimates the changes in the energy usage, so the period budget may adjust to reflect the actual period budget used.
- *Policy Selector*: It receives the function characteristics (its type, name...), period budget, and estimated energy as inputs to the Decision Strategy. Decision strategy selects the suitable policy based on these inputs.
- *Policy Table*: It stores for each function the selected policy.
- *Profiler*: Profiling involves gathering the necessary information during runtime to make informed offloading decisions. It includes the function profiler, network profiler, and device profiler. The function profiler collects wall clock and CPU time for each invocation. The network profiler monitors the available bandwidth and latency to the edge server. The device profiler captures the memory usage and the battery level.
- *Historical Data*: This database holds the historical information related to the execution of a given function. It stores per invocation a timestamp, the execution time, the values of scalar inputs as well as the size of other inputs and of the results in number of bytes.
- *Function DB*: This database saves function name, type, and signature.

IV. Decision Strategies

As mentioned before, a decision strategy selects the policy for each function during the next period. The goal is to have a fast decision making for each invocation, while more time consuming operations, like predicting the mix of function invocations in the next period and their impact on period's energy budget, is infrequently done by the decision strategy. We defined several strategies:

- Priority-based Strategy: The user defines the function priority reflecting its importance to run locally.
- Latency-based Strategy: It considers short running function first for local execution.
- Execution Time-based Strategy: It selects for each function the fastest execution (local or remote) as long as remaining period budget is sufficient. If the period's energy budget is all used, it selects the always offload policy.
- Service Level Objective (SLO) Strategy: It aims to reduce energy while meeting the function deadline.

Listing 1 presents the Priority-based Strategy and is based on the acronyms shown in Table I.

The algorithm receives two inputs which are: RPB and FL. Line 1 sorts the functions based on pre-defined priorities. The priority expresses the need to run the function locally. Each function must be assigned to one of the priority levels. If the priority of a given function is zero, it has to run locally (lower numbers - higher priority). Lines 3-4 initialize the mean local execution time and the mean remote execution time. Lines 5-10 handle the situation where no historic data are available. Line 11 determines the function type. If it is an independent function, the energy estimation of the invoked function is computed in line 13. Line 15 checks if the function priority is equal to zero. In that case the function is executed locally and the estimated energy is subtracted from the RPB. Otherwise, if the remaining period budget is greater than the estimated energy and the local execution time is faster than the remote one, the function will be executed locally. If not, the offload policy is chosen in line 23. It may be noted that, based on historical data, the budget should be expanded than the current one used. Without this adjustment, resources may fall short of

TABLE I: Variables and their definitions used in the algorithms

Variable	Interpretations
RB	Period Budget
RPB	Remaining Period Budget
FL	Application function List
FT	Function Type
f	Function Name
MeanRT	Mean Remote Execution Time
MeanLT	Mean Local Execution Time
E_f	Estimated Energy of function f

Algorithm 1 Priority-based Decision Strategy

Input: RPB, FL
Output: PolicyTable Generation

```
1:  SortByPriority(FL)
2:  for f in FL do
3:      MeanRT = HistoricDB.getMeanRT(f)
4:      MeanLT = HistoricDB.getMeanLT(f)
5:      if (MeanRT == 0) then
6:          PerformanceModel.getMeanRT(f)
7:      end if
8:      if (MeanLT == 0) then
9:          PerformanceModel.getMeanLT(f)
10:     end if
11:     FT = Function.getFunctionType(f)
12:     Priority = Function.getFunctionPriority(f)
13:     E_f = Compute_energy(f)
14:     if (FT == "input_independent") then
15:         if (Priority == 0) then
16:             policy = local_policy
17:             RPB ← RPB − E_f
18:         else
19:             if (RPB > E_f and MeanLT < MeanRT) then
20:                 policy = local_policy
21:                 RPB ← RPB − E_f
22:             else
23:                 policy = offload_policy
24:             end if
25:         end if
26:     end if
27: end for
```

meeting future demands effectively.

V. Benchmark and Evaluation

To validate our framework, we developed an Android application called ServerlessApp which combines a set of Java functions to cover a diverse range of testing scenarios. The application is composed of Device_NetworkInfo, which is an input independent function, and the following input dependent functions prime, LinearSystemEquation, ParseFile, and LoadFile. Additionally, we selected a predefined set of inputs to rigorously test the framework under various conditions, creating a benchmark for evaluation. This chosen workload includes a mix of input parameters that lead to executions of varying durations. The application will invoke its functions in different orders. The application is composed of five functions, with the following distribution: Device_NetworkInfo 30%, Prime 10%, LinearSystemEquation 20%, LoadFile 5%, and ParseFile 35%. The input requirements are as follows: Device_NetworkInfo requires no input, Prime function takes an integer value ranging from 1 to 2000 and computes all prime numbers from 1 up to the given input, LinearSystemEquation takes an integer between 1 and 100, and ParseFile takes the file size as an input. The Device_NetworkInfo returns the device and network information. In each iteration, a different function is invoked. What is important here is the distribution (the percentage of invoked functions in the generated distribution), not their sequence. This allows for different usage patterns. At the beginning, we assign the same energy budget to each period which lasts five minutes and then we start updating it based on the historical data retrieved from previous periods. Between periods, offloading decision strategies are revisited so that the policies are updated based on the current circumstances (energy budget, function type, network conditions). This iterative process ensures continuous improvement.

TABLE II: FaaS Functions Used for Evaluation

Categorie	Number	Name
Web-based	*f1*	Device_NetworkInfo
CPU-based	*f2*	Prime
CPU-based	*f3*	Linear System Equations
Network-based	*f4*	LoadFile
Memory-based	*f5*	ParseFile

Our testbed consists of a mobile phone and an edge server. For the mobile device, we use the Samsung Galaxy (CPU: 2.7 GHz, Ram: 6 GB, Storage: 128 GB). For the edge server, we installed OpenFaaS on top of Kubernetes. The stacked bar chart in Figure 5 highlights the different decision strategies in a given period per function invocation. The Execution Time-base strategy, Priority-based strategy, and SLO strategy were evaluated under two distinct budget scenarios: small and large. The X-axis represents the three strategies, while the Y-axis reflects the used energy budget and the Z-axis illustrates the number of invoked function in this period.

The Execution Time-base strategy with large budget indicates that running the invoked functions locally is more

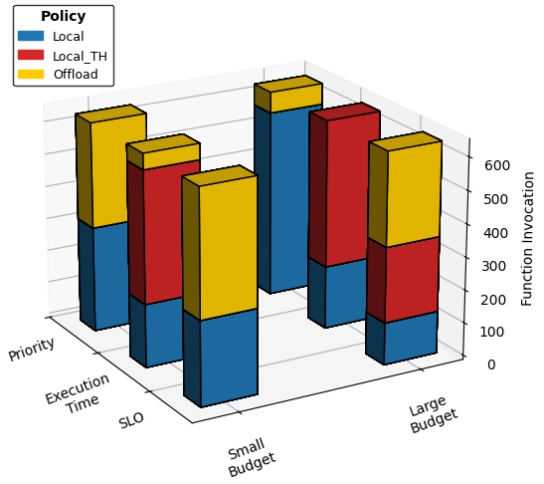

Figure 5: Decision Strategies Per Function Invocations

efficient due to reduced communication delays and better resource availability since the energy budget is sufficient.

The Execution Time-base strategy with small energy budget offloads 7.5% of the invoked functions while large budget, the strategy rely on local and local_TH. In other words, for input dependent functions, the threshold value might be set for the function such that all function invocations will be run locally. The outcome for input dependent functions is the policy with threshold and the threshold varies for the different used strategies because it depends on the energy and execution time.

The priority strategy with small energy budget processes the highest-priority functions locally and offloads the remaining functions to edge server. However, when the available energy budget is large the strategy determines to handle 90% of the invoked functions internally; 10% offloaded when the energy budget becomes limited. Finally, SLO with small energy budget offloads 60% of invoked functions because the strategy should meet the deadlines of the functions while saving energy. When the energy budget is sufficient, 55% of the invoked functions are executed locally, encompassing both function types(input dependent and input independent).

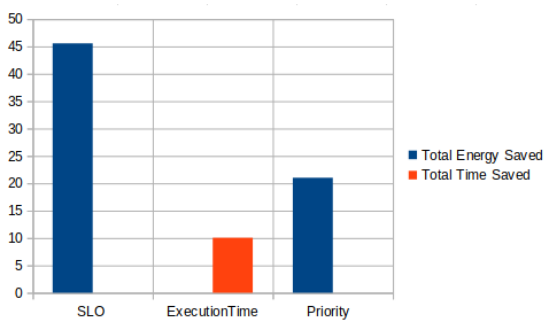

Figure 6: Saved Energy and Time.

Figure 6 shows the execution time saved. The processing time considers all aspects such as function execution, send-

ing inputs and getting output from remote server. The SLO decision strategy optimizes energy consumption by offloading as long as the function deadline is met. It shows significant savings in energy (45.56 Joules). On the other hand, the Execution time strategy achieves significant reductions in execution time (10.051 seconds) because the strategy is based on selecting the fastest platform where the function should be executed. The Priority strategy saves 20% of the energy by outsourcing low-priority functions to the edge server.

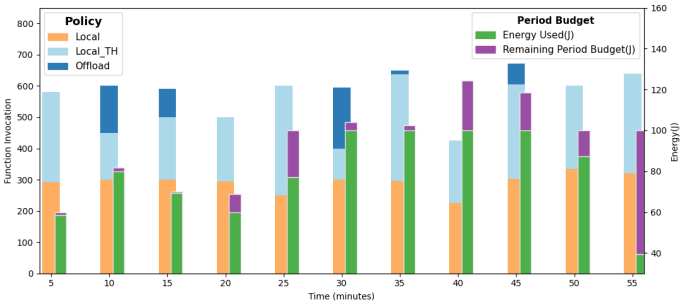

Figure 7: Energy Budget Distribution Over Running Periods.

Figure 7 presents a stacked bar chart comparing two aspects which are: Decision policy and energy budget. The X-axis represents time intervals of 5 minutes (period), while the left Y-axis reflects the invoked functions and the right Y-axis presents the used energy. The first bar shows the used policies during that period. Each segment represents a different policy. The second bar illustrates the energy budget status, divided into two parts: the portion of used energy and the remaining budget. Based on the historical data and the application duration of 55 minutes, the budget energy required over the period is assigned based on the requirements of the invoked functions in the period. In the sixth period at 30 minutes, an increase in energy consumption triggered a revision of the policy. It is changed from local to offload. As a result, 30% of invoked functions were offloaded to the edge server.

VI. Conclusion

Our presented work aims to show the idea of planning energy consumption and selecting the suitable policies in order to make the appropriate offloading decision. The practical application results reveal that our decision strategies yield energy savings ranging from 20% to 46% depending on the function type, complexity and network conditions. Finally, We are working on extending for other types or more flexible functions to have more optimal results.

References

[1] A. Shakarami, M. Ghobaei-Arani, and A. Shahidinejad, "A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective," *Computer Networks*, vol. 182, p. 107 496, 2020.

[2] Y. Gong, C. Lv, S. Cao, L. Yan, and H. Wang, "Deep learning-based computation offloading with energy and performance optimization," *EURASIP Journal on Wireless Communications and Networking*, vol. 2020, pp. 1–8, 2020.

[3] A. Heidari, M. A. Jabraeil Jamali, N. Jafari Navimipour, and S. Akbarpour, "Internet of things offloading: Ongoing issues, opportunities, and future challenges," *International Journal of Communication Systems*, vol. 33, no. 14, e4474, 2020.

[4] Y. Li, S. Xia, M. Zheng, B. Cao, and Q. Liu, "Lyapunov optimization-based trade-off policy for mobile cloud offloading in heterogeneous wireless networks," *IEEE Transactions on Cloud Computing*, vol. 10, no. 1, pp. 491–505, 2019.

[5] Z. Li, V. Chang, J. Ge, L. Pan, H. Hu, and B. Huang, "Energy-aware task offloading with deadline constraint in mobile edge computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, pp. 1–24, 2021.

[6] B. Cao, S. Fan, J. Zhao, *et al.*, "Large-scale many-objective deployment optimization of edge servers," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 6, pp. 3841–3849, 2021.

[7] B. Cao, Z. Sun, J. Zhang, and Y. Gu, "Resource allocation in 5g iov architecture based on sdn and fog-cloud computing," *IEEE transactions on intelligent transportation systems*, vol. 22, no. 6, pp. 3832–3840, 2021.

[8] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, "Eedto: An energy-efficient dynamic task offloading algorithm for blockchain-enabled iot-edge-cloud orchestrated computing," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2163–2176, 2020.

[9] H. Wu, W. J. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 7, pp. 1464–1480, 2019.

[10] G. Yang, L. Hou, X. He, D. He, S. Chan, and M. Guizani, "Offloading time optimization via markov decision process in mobile-edge computing," *IEEE internet of things journal*, vol. 8, no. 4, pp. 2483–2493, 2020.

[11] G. R. Russo, V. Cardellini, and F. L. Presti, "A framework for offloading and migration of serverless functions in the edge–cloud continuum," *Pervasive and Mobile Computing*, vol. 100, p. 101 915, 2024.

[12] M. Masdari, K. Majidzadeh, E. Doustsadigh, A. Babazadeh, and R. Asemi, "Energy-aware computation offloading in mobile edge computing using quantum-based arithmetic optimization algorithm," 2022.

[13] A. Ebrahimi and F. Afghah, "Intelligent task offloading: Advanced mec task offloading and resource management in 5g networks," in *2025 IEEE Wireless Communications and Networking Conference (WCNC)*, IEEE, 2025, pp. 1–6.