Informatics

A Benchmark Suite for AI workloads in Serverless Edge Computing

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Paul Prüller, BSc
Matrikelnummer 01326451

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Schahram Dustdar
Mitwirkung: Univ.Ass. Dipl.-Ing. Philipp Raith, BSc

Wien, 7. April 2024

Paul Prüller	Schahram Dustdar

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Informatics

A Benchmark Suite for AI workloads in Serverless Edge Computing

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Paul Prüller, BSc
Registration Number 01326451

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Schahram Dustdar
Assistance: Univ.Ass. Dipl.-Ing. Philipp Raith, BSc

Vienna, 7th April, 2024

Paul Prüller	Schahram Dustdar

Erklärung zur Verfassung der Arbeit

Paul Prüller, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 7. April 2024

Paul Prüller

Danksagung

Die letzten Jahre waren eine außergewöhnliche Zeit und es ist nicht selbstverständlich durch diese Phase mit so einer Unterstützung gehen zu dürfen. Ich möchte daher diese Möglichkeit nutzen, um mich bei meinen Eltern, FreundInnen und WegbegleiterInnen zu bedanken, die mir immer wieder die Zeit, aber auch den gewissen Schubser gegeben haben, um dieses Studium und diese Arbeit mit der nötigen Motivation und Kraft abschließen zu können. Dieser stetige Rückhalt hat dazu beigetragen, dass ich mit Stolz und Freude dieses Studium abschließen konnte. Einen großen Dank auch an meinen Diplomarbeitsbetreuer Philipp Raith, der mich mit Geduld und unglaublichen Fachwissen unterstützt hat. Weiters möchte ich der Technischen Universität Wien danken und die vielseitigen und interessanten Studien- und Forschungsmöglichkeiten hervorheben.
Vielleicht sieht man sich ja wieder!

Acknowledgements

The last few years have been an extraordinary time and it is not a matter of course to be able to go through this phase with such support. I would therefore like to take this opportunity to thank my parents, friends, and companions, who have always given me the time, but also the certain push to be able to complete this study and this work with the necessary motivation and strength. This continuous support has contributed to the fact that I was able to complete my studies with pride and joy. A big thank you also to my thesis supervisor Philipp Raith, who supported me with patience and incredible expertise. I would also like to thank the Technical University of Vienna and emphasize the diverse and interesting study and research opportunities.
Maybe we'll see each other again!

Kurzfassung

Die Durchführung von Benchmarking für Anwendungen im Bereich Virtual Reality (VR) und artifizielle Intelligenz (AI) in Edge Computing Umgebungen gestaltet sich aufgrund begrenzter verfügbarer Hardwareressourcen als herausfordernd. Insbesondere bei der Analyse von Smart-City Umgebungen ist es oft nicht möglich, eine realistische Hardwareinfrastruktur zu implementieren. Beim Benchmarking serverloser Anwendungen gibt es zwei gängige Methoden. Auf der einen Seite gibt es maßgeschneiderte Testbed-Setups, bei denen echte Hardware zum Einsatz kommt, die realistische Bedingungen ermöglichen, auf der anderen Seite existieren Simulationstools, bei denen ausschließlich Algorithmen und Modelle zur Nachbildung des Verhaltens eines Systems verwendet werden. Im Gegensatz zu groß angelegten realitätsnahen Experimenten, die mit der Container Orchestrierungsplattform *Kubernetes* durchgeführt werden, besteht die entwickelte Testumgebung aus nur wenigen verschiedenen Hardwarekomponenten. Diese ermöglichen es, realistischere Versuchsergebnisse in einem kleineren Rahmen zu sammeln, als es in einer simplen Simulation möglich wäre. Der optimale Weg hängt vom Benchmarking Prozess und den spezifischen Anforderungen bezüglich der Steuerung und Reproduktion von Experimenten ab. Die Begrenzung der verfügbaren Hardwareressourcen kann jedoch mithilfe von Simulation-Frameworks überwunden werden. Diese können auf einem einzelnen Rechner ausgeführt werden und helfen dabei, die benötigten Ressourcen erheblich zu reduzieren. Derartige Frameworks bieten eine breite Palette von Parametern an und erfordern eine Standardisierung hinsichtlich der verwendeten Workloads, Netzwerkkomponenten und anderer Teile derartiger Setups. Es weist Einschränkungen hinsichtlich der Genauigkeit von Benchmarking Ergebnissen auf. Dabei müssen viele Parameter berücksichtigt werden, um den Grad der Realitätsnähe zu erhöhen. Obwohl die Simulationen die Realität nicht exakt abbilden, dienen sie als ausgezeichneter Ausgangspunkt für die Entscheidungsfindung bei der Entwicklung und Bewertung von Arbeitslasten und Topologien. Zusätzlich können die Ergebnisse dieser Evaluierungen auch die Ressourcenplanung von Edge Computing Hardware-Setups unterstützen, speziell, wenn es um die Integration von Edge Computing Hardwarekomponenten geht. Diese Arbeit erweitert das Simulationsframework *faassim* um verschiedene AI-basierte Arbeitslasten, Workload Profilen und geodistribuierte Topologien. Ein spezielles Setup ermöglicht es, Anfragemuster basierend auf benutzerdefinierten Szenarien zu generieren, indem individuelle Topologien und Ereignisdatensätze verwendet werden. Durch eine systematische Literaturrecherche werden auch bestehende Frameworks verglichen, um eine solide Grundlage für zukünftige Designentscheidungen zu schaffen. Das Framework *faas-sim* ermöglicht die Durchführung umfangreicher Experimente durch Simulationen. In dieser Studie werden die Ergebnisse verschiedener Szenario- und Profiling-Experimente verglichen, die auf einer eigens erstellten Testumgebung (Testbed) durchgeführt wurden. Die entwickelte Suite umfasst sechs verschiedene *Openfaas*-basierte Funktionen, sowie eine Inferenzpipeline für das Benchmarking mehrerer verbundener serverloser Funktionen. Zusätzlich beinhaltet sie Skripte zur Erstellung von Topologien aus der Open-Source-Datenbank *OpenCellid*[1]. Der verwendete NYC Taxi-Datensatz dient als Grundlage für die Erstellung von Anfragemustern und Workload Profilen. Dieser Datensatz kann durch jeden beliebigen Ereignisdatensatz, der aus einzelnen Einträgen mit einfachen Zeitstempeln besteht, ersetzt werden. Die Benchmark Suite wurde in der Hinsicht entwickelt, sodass das Testbed und die Simulation mit denselben Daten arbeiten können. Die Ergebnisse zeigen, dass die Metriken der Simulation die experimentellen Messwerte nicht genau widerspiegeln, in beiden Fällen jedoch erkennbare Unterschiede zwischen den Zonen erkennbar sind. Simulationskonfigurationen von Geräten und Topologien sind daher wichtig, um realistische Simulationsszenarien zu gewährleisten.

[1]https://opencellid.org/

Abstract

Edge Intelligence applications combine resources in the edge-cloud continuum to provide new AI applications such as Mobile Augmented Reality. Serverless Edge Computing can facilitate the deployment of these applications, but current offerings are not yet suitable. Benchmarking in the field of VR and AI is limited due to the capacity of the available hardware, and the creation of realistic large-scale Smart City infrastructure for testing purposes is impractical and very expensive. When benchmarking serverless applications, there are two common ways to do that. On the one side, we have customized testbed setups, where real hardware is involved to allocate realistic conditions and allow small scale experiments, on the other side simulation tools where only algorithms and models are used to replicate the behavior of a system. They run on a single machine and reduce the needed amount of resources tremendously. It has limitations like a lower accuracy of benchmarking and must consider a lot of parameters to increase the degree of realism. They do not reproduce the real world but can be seen as a point of reference for further decision. Both ways need standardization concerning the workload inputs and network components. The portability and reproducibility between real world and simulation is therefore an aggravating factor. In this thesis, the framework *faas-sim* is going to be extended by a suite of AI-based workloads, request patterns, topologies, and a custom request pattern generator. By using a systematic literature review, other existing frameworks are compared and can be used for further design decisions. The suite offers six different open-source based serverless AI functions and an inference pipeline for benchmarking multiple connected serverless functions. It also includes functionality to create topologies out of the box for simulation and experiments. The used *NYC Taxi Dataset* is the basis for creating united request pattern and workload profile generation. This can be exchanged by any event dataset. The evolved suite[2] will be incorporated into the study, ensuring that both the testbed and simulation process the same input data. The *faas-sim* framework results were then compared with the testbed experiments results. It shows, that the simulation results do not precisely mirror the experimental metrics, but recognizable variations among zones are evident in both cases, which allows to start more detailed analysis regarding further design decisions. Hence, the simulation configuration of devices and topologies is important to guarantee realistic simulation scenarios.

[2]https://github.com/pruellerpaul/benchmark_suite

Contents

CHAPTER 1

Introduction

Edge Intelligence offers a widespread application area. In the case of this thesis, the areas of user-centered applications, Augmented Reality, and Artificial Intelligence are viewed as an opportunity for taking a close look at how effective and performative these applications and services work in Edge Computing networks and how service orchestration is done in different scenarios. *Kubernetes*, which facilitates container virtualization and deployment, plays a central role in this thesis by enabling the independent rollout of applications across different devices, independently of the operating system. Applications in these areas behave differently when looking at the resources needed. Some of them require intensive computing power, others memory-intensive work. The complexity of managing these applications is high, but can be handled by serverless computing and implementing specific scheduling strategies. To obtain the optimal scheduling strategy, it is necessary to know about the resource needs of a function or application. The performance, costs and effectiveness of Edge Computing platforms can be influenced by many factors and must therefore be analyzed using benchmarking methods. Two ways e.g. to achieve this are benchmarking experiments on real devices that provide very accurate results, and simulation tools that provide approximations and allow representing upscaled scenarios.

The *faas-sim* project [RRFD23], developed at the Distributed Systems Group at TU Wien, is concerned with the simulation of container-based FaaS (Function-as-a-Service) platforms and the development of strategies for scheduling, auto-scaling or load balancing in different scenarios like urban sensing edge systems, hybrid Industry 4.0 settings or cloud regions. Object Classification or Speech-to-text, which are application areas in AR and AI, are already implemented in this framework as the first examples for such simulations [RRD21]. Edge Intelligence requires a high-performance architecture for computing. Therefore, Serverless Edge Computing is playing an important role in this area. Functions that are going to be uploaded by users are deployed and managed autonomously by the Edge Computing platforms, which gets more complex when handling heterogeneous environments, like the cloud-edge continuum, where a wide range of different hardware resources and network participants appear. The efficient execution and the compatibility among the competitors are challenging for this reason [ZCL+19]. It shows, that the data management, performance, infrastructure engineering, and AI support are still in need of improvement [RND23].

Serverless Edge Computing makes use of a hierarchical architecture approach. It is usually designed to orchestrate and manage systems, applications, or services that are distributed over multiple cloud providers and network domains. In the case of Edge Computing, which is a very dynamic environment with many devices and network participants, the orchestration of heterogeneous services proves difficult due to the e.g. geographical distances between network nodes and the resource (CPU, RAM, etc.) limitations of edge devices. In [LLR+21], the placement of server and edge devices are discussed and display a comprehensible use case of this problem area. Let's think about a smart city environment, where the need for an even distribution of Edge Computing units is important, such that the resources and edge devices are placed efficiently regarding the distribution of the potential clients over the network. The results and methods of this thesis are a motivational use case regarding server placement and resource planning and can be part of a prospective way to solve such planning problems. Because of the expensive and complex computing and data storage infrastructure, it is not easy to evaluate real-world Edge Computing systems against benchmarks and performance indicators, without simulation and emulation programs. The problem is, that it is not possible to scale up local settings to a real-world edge system with thousands of machines without an enormous amount of financial resources. There are several simulation tools, that offer repeatable evaluation of application traces in virtual large-scale network topologies. For the emulation of edge infrastructure, some tools can help to test real applications in an experimental environment. A possible way to emulate and test the behavior of a specific deployment and orchestration strategy is to set up a testbed in a local environment, which represents an abstract edge system, and to evaluate the performance for this setting. To come along with both contexts (testbed emulation and simulation) it is essential to have a baseline to operate together. The generated traces of the testbed emulation are the input parameters of the simulator, which then simulates the behavior in a big-scaled environment.

Currently, in our *faas-sim* context, the tools are limited and not suitable to cover a wide area of AI-based workloads, typical infrastructure topologies, geo-distributed usage patterns and to ensure sufficient evaluation. Also, the offered benchmark indicators are limited in the context of Quality of Service (QoS). An extendable and complete suite of geo-distributed request patterns, different workloads, and AI-centered benchmarks, considering Quality of Service and realistic topologies would close the gap and establish a verifiable way to test the behavior and performance of AI application deployments on edge devices.

1.1 Aim of the work

First of all, we want to give a brief overview of the techniques, and theoretical information regarding this thesis, analyze common Cloud/Edge Computing simulation & emulation frameworks, and asses benchmarking tools & metrics in the research area of AI and AR. The simulation and emulation frameworks are intended to provide a reference and motivation as to which aspects should be taken into account when working with such tools, what the existing challenges and obstacles are and what further developments are possible. The first main aim is to set up an extendable suite of AI-based user-focused workloads that work with *faas-sim* and can also be adapted on an existing small-scaled testbed, which represents a low-scaled edge Kubernetes cluster and works with container-based orchestration. Because of the large scale of AI applications and workloads, this thesis is going to limit the scope to Machine learning inference. ML consists of long-running lived tasks like preprocessing and training and short-running lived tasks like inference, so we decided to set the focus on the latter to narrow the scope of the work. The accurate workload structure for the subsequent experiments has to be specified and a collection of realistic infrastructure topologies and geo-distributed usage patterns should be developed by using real-world datasets. After this, we are going to define and run benchmark experiments. In the evaluation phase, we start to line out the differences between simulation and testbed experiment runs regarding the benchmarks and metrics, which we gathered before in the requirement engineering phase. Our work enables as future work to find constellations to implement common scaling and scheduling strategies with the most outcomes and offer a suite that allows one to find the optimal edge device and server placement more efficiently.

1.2 Research Questions

RQ1 **Which applications or functions, benchmarking tools, and metrics are used in the research area of Edge Computing and Artificial Intelligence? What are the necessary aspects and components of a state-of-the-art AI benchmarking suite?** In the context of AI and AR, considering an Edge Computing environment, it is necessary to know which application fields and serverless functions are already trialed and which tools and metrics are used in established works to generate an optimal benchmarking setup. This will be done by literature review, related work research, and requirement engineering.

RQ2 **How can realistic infrastructure topologies, request patterns, workloads, and recent serverless AI or AR functions be developed for simultaneous use on a testbed infrastructure and a simulation framework, by only using open data? How do the developed functions behave in the real-world on heterogeneous devices in terms of resource usage?** When looking at common benchmarking and performance tests, realistic data is missing, or not used for benchmarking. For this work, open data is used to develop infrastructure topologies, workloads, and request patterns, that can be used for simulation and testbed experiments at once. Also, simple AI-based serverless functions will be applied in the benchmarking phase. Their profiling results are baseline for the simulation execution.

RQ3 **When comparing the results of the testbed experiments and the simulation, what differences can be examined in terms of the metrics and benchmarks that are important for our use cases defined in *RQ1*?** To envelope a reasonable simulation framework like *faas-sim*, it is necessary to check and compare its outcome with real-world trace-driven testbed data. For this reason, the serverless functions from **RQ2** will be implemented in the *faas-sim* simulation. The simulation and the testbed experiments will get tested with the same request pattern and topologies we gathered in *RQ2*. Metrics like round-trip time, latency, or memory consumption time are useful examples for the benchmarking process. Different *faas-sim* reconcile intervals can show how close the simulation is to the real-world data.

1.3 Structure of Thesis

This thesis starts with the Chapter 2, which presents the fundamental background of Edge Computing, Edge Intelligence, and Serverless Computing and shows how Kubernetes and *faas-sim* work. In Chapter 4, the several methodical stages are described. It is split into the literature review, related work research, and requirement engineering subsection, the infrastructure topologies, request pattern, and workload development subsection, followed by the testbed experiments, simulation runs, and analysis results subsections. In Chapter 3 the found simulation, emulation, and hybrid frameworks are listed. Also, a selection of serverless benchmarking tools and suites is shown in this chapter. In Chapter 5 and Chapter 6 the detailed approach of the methods and experiments are presented. In the experiment chapter, the testbed setup, the profiling, and the scenario experiment implementation are described in detail. Chapter 7, the evaluation chapter, shows the results of the related work research, topology and cell extraction, request pattern generation, and the testbed scenario and profiling experiments. The Chapter 8 shows the conclusion and elaborates on the limitations of this thesis and future work.

CHAPTER 2

Background

The background chapter elucidates information and knowledge about technologies and concepts that are important to this thesis. It starts in Section 2.1 with an overview of Edge Intelligence. In Section 2.2 the Edge Computing paradigm is introduced, and in Section 2.3 the concept of Serverless Computing gets outlined with a special focus on the Function-as-a-Service paradigm. Finally the platform systems *Kubernetes* and *faas-sim* are presented.

2.1 Edge Intelligence

In the past years, strong growth of performative computing devices, Cloud Computing servers, or edge devices with hardware accelerators could be observed. This is also a consequence of the Edge Computing paradigm, presented in Section 2.2. Simultaneously, the amount of applications, which evolved a heavy need of computing resources, also increased. In [RD19], the current cyber-human transition is discussed, where augmented human cognition becomes more and more in our daily lives. According to their definition, EI can be divided into different categories, namely public, private, predictive maintenance and intersecting. Due to the successful progression of AI applications and their use cases, such as image recognition, speech recognition, recommendation systems, the Industrial Internet of Things or e.g. audio & video surveillance [ZCL$^+$19], these new algorithms and applications were pushed into the market of systems that have high demand for computing and network resources. The best examples in which this phenomenon can be observed are *smart home devices* and their applications, such as self-assisted systems *Alexa* and *Siri*. These are only two well-known examples of Artificial Intelligence (AI) systems that use edge devices for computing and allow an insight into how big AI is already distributed in our daily life, the edge environment. This fact concludes that the amount of data generated on the edge of the network has also increased. The need for the knowledge of what data is needed and where it should be processed is also a significant key point. Therefore, the paradigm of *Edge Intelligence* (EI) is introduced, where AI and Edge Computing find a connection. In this section, we are going to focus on EI, the included components, and the occurring challenges, advantages & limitations.

Challenges

The data generated at the edge side increases, so the AI algorithms in the cloud data centers are going to process more and more data in the near future. Hence, a significant challenge is to reduce the consumption of bandwidth resources by, e.g. improving the performance of DL algorithms. Therefore, e.g. low-latency data processing is needed to reduce the needed computing capabilities of the cloud data centers [ZCL+19] [DZF+20]. It is also necessary to know which scenarios are applicable to edge devices and which are not, which model is the right one for specific AI tasks and what coordination mechanism betweeen heterogeneous edge devices is used [DZF+20]. In Section 2.1.2 a possible classification is shown. The further development of hardware edge devices, the miscellaneous amount of data produced at the edge, and the needed memory size as a result of it will also be challenging. In [RD19], some unique challenges are listed. For example the increase in sensing and modular AI capabilities, or the requirement of edge devices to handle multi-purpose applications. Also, edge coordination mechanisms are, like in Cloud Computing, a tremendous factor in optimizing workload scheduling. The privacy factor must not be ignored and is indispensable for applications in both private and corporate predictive maintenance [RD19].

Advantages

EI is capable of unlocking the full potential of the data generated at the edge of the network, making fast decisions, and handling a large amount of different data efficiently. Edge Computing is also able to advance AI with more prolifically data and application scenarios than Cloud Computing. AI is closer to people, data sources, and devices when it is used in the Edge Computing environment, which is also more affordable and accessible than the Cloud Computing environment [ZCL+19].

Limitations

When looking at the resource limitations of edge devices, it seems clear that not all AI tasks can be moved to the edge. It would require high-end processors to train DNN models locally. This increases the costs and would not be compatible with existing edge end devices [ZCL+19] [RD19]. In [RD21], *Edge Intelligence as a Service* is presented as a theoretical approach, highlighting many different points of discussion and limitations. It shows, that not every decentralization, for example, the transition from load balancers to the edge of the network, is expedient, as security problems can arise here [RD21].

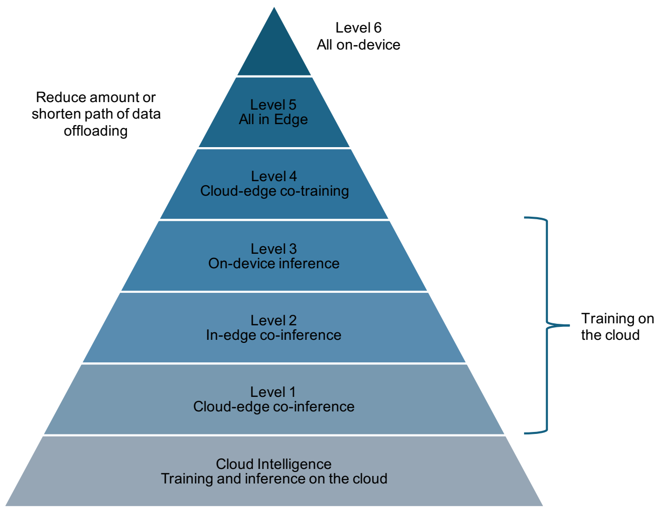


Figure 2.1: Level rating adapted from [ZCL+19]

2.1.1 Deep Learning & Deep Neural Networks

Nowadays, self-learning and machine-learning capabilities are necessary for contributing AI-based applications. The Deep Learning paradigm is inspired by the human Neuron mechanism, where synapses, neurons, axons, and impulses (activations) play a necessary role in self-learning and solution-finding. In [ZCL+19] and [SCYE17] the mechanism is explained in more detail. The main reason, why this human-inspired approach was chosen is, that the human brain shows perfect preconditions for evolving an ML-based algorithm [SCYE17]. In this context, different layers process some real-world input data, combined with weights, and create a nonlinear output, which is then forwarded to the next layer. These weights will be optimized during the model training phase, which consists of backpropagation and feedforward processes [ZCL+19], to generate a more precise model. This model is used in the inference phase to determine predictions and classifications for different use cases like the already mentioned image recognition. Such Deep Learning Neural Networks (DNN) can consist of multiple layers. In [ZCL+19], three popular networks are presented, namely the multilayer perceptron (MLP), which is the most common form of DNNs, the convolution neural network (CNN), and the recurrent neural network (RNN). Each of these networks can be used for different use cases and is the most popular in their application field. CNNs are used e.g. in computer vision areas and RNNs in speech recognition and natural language processing areas where no fixed input length is given. Popular CNNs are, e.g. *AlexNet*, *GoogleNet*, *VGG* and *ResNet*. In RNNs, the backpropagation through time (BPTT) is used for model training where the input is based on the previous sample data. Long-short-term memory (LSTM), which is an extension of an RNN, allows the network to keep or forget information using memory cells [ZCL+19]. Then there are also so-called Generative Adversarial Networks (GANs), where a generator network creates new data based on real data and a discriminator network has to classify the two data sets. This is used in e.g. image generation or image transformation tasks [ZCL+19]. DNNs are also used together with Reinforcement Learning (RL) to contribute Deep Reinforcement Learning (DRL), where long-term tasks are tackled by continuous learning. It is, e.g. applied in gaming applications for solving scheduling or decision problems [ZCL+19]. Existing heterogeneous AI networks have different computational needs. It is therefore important to plan ahead and benchmark such networks.

2.1.2 Levels of EI

In [ZCL$^+$19] an excellent classification of EI levels was made. To come along with different resource requirements, application scenarios, and end-user requirements the different AI tasks have to be distributed over edge devices or servers and cloud centers, according to their requirements. Hence, seven levels were introduced, which are also shown in Figure 2.1.
Cloud intelligence allows full data offloading for training and inference, **cloud–edge co-inference** partial offloading of inference and complete training in the cloud, and **in-edge co-inference** facilitate *in-edge* inference and complete training in the cloud. **On-device inference** enables no data-offloading and full training in the cloud, **cloud–edge co-training** partial offloading of inference and training, **all in-edge** inference and inference training only *in-edge*, and **all on-device** does not allow data offloading in training and inference.

In-edge:

> „In-edge means that the model inference is carried out within the network edge, which can be realized by fully or partially offloading the data to the edge nodes or nearby devices (via D2D communication).“ [ZCL$^+$19]

2.2 Edge Computing

Edge Computing extended the paradigm of Cloud Computing by placing the computing and data storage nodes closer to the requesting services and network participants. A summary of Edge Computing devices is shown in the next sub-section. Fog Computing closes the gap between edge and cloud by using the advantages of both sides. In Figure 2.2 the placement of Edge and Fog devices is displayed. Because of the computational power and the data storage which are geographically close to the edge devices, the latency is much lower than for Cloud Computing devices. The resources that are used in these edge devices are in the majority of cases based on Linux technologies and virtualized by using modern virtualization methods, like container virtualization which we discuss in Section 2.3.1. This is done to optimize overall performance, memory usage, and reliability. It can be said that Edge Computing enables EI, but there is currently no clear definition, which clearly shows the division into the areas of Mobile Cloud Computing and Mobile Edge Computing, which define the different computing methods of workload offloading.

2.2.1 Mobile Cloud Computing

Mobile Cloud Computing (MCC) is part of the Edge Computing paradigm and takes advantage of the computational offloading architecture. It uses remote data centers for computing. This leads to higher latency and bandwidth limitations [LSHG16].

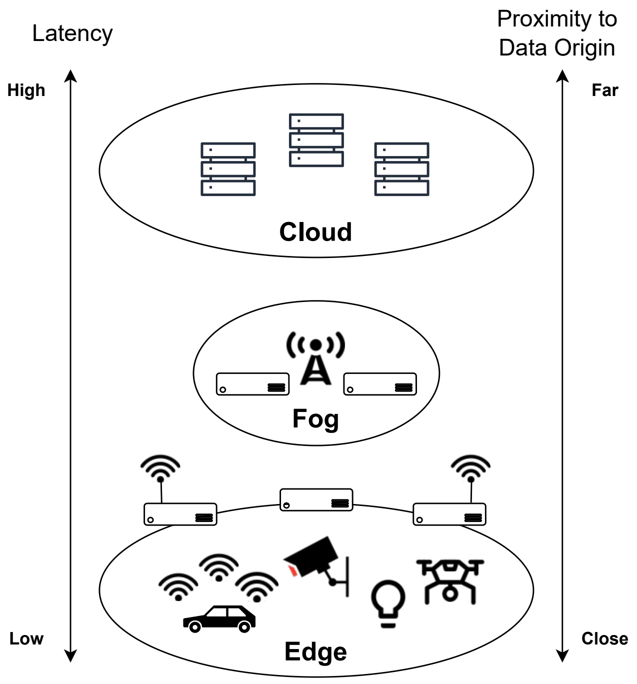


Figure 2.2: Edge Computing overview. (Adapted from [CDPLR19])

2.2.2 Mobile Edge Computing

Mobile Edge Computing (MEC) is also part of the Edge Computing paradigm and is a more recent form of Edge Computing concerning mobile devices. It takes advantage of mobile base stations and the Radio Access Network (RAN) and allows to extend Cloud Computing services to the edge of the mobile network. The so-called cloudlets, which are small-scale servers at the location of mobile base stations, allow one to perform computational tasks close to the actual location of mobile devices. MEC uses the nearest mobile station to reduce latency and provide a large geographic coverage [LSHG16].

2.2.3 Edge Devices

Sensors are devices that output and generate data, e.g. airflow sensors, thermometers, or cameras. The only task of these devices is to produce raw data. They are not configured to do computational work or aggregation. **Devices with computational power**, like servers, have the purpose of processing generated data and performing the hard computation tasks required by other services. **End-user** devices, like smartphones, tablets, etc., are used to receive and display the computed or aggregated data. Such **Edge devices** can be divided into constrained devices, single-board computers, or mobile devices. They rarely support virtualization and have, by comparison with e.g. cloud servers a much smaller memory and computing power. **Edge/Fog Server**, e.g. for special use cases such as automotive, or common cloud, and customized platform devices are used in both levels. They support virtualization and relay on CPUs with one or more GPU co-processors [CDPLR19].

2.2.4 Applications in Edge Computing

Edge Computing has many areas of deployment in real-world environments, and the integration of Edge Intelligence enhances its capabilities further. Some of the most applied areas are listed in this section.

Cloud Offloading outsources the computational work to the centralized cloud server(s), which enables much more latency than compute workloads on edge devices. In some cases, e.g. shopping cart modification in e-commerce online applications, the latency can be reduced by pushing the computational work to the edge nodes near the users. Hence, the data in the edge devices has to be synchronized to offer the current availability information. In addition, aggregation, filtering, vision aid entertainment games, augmented reality, and connected health workloads can benefit from the Edge Computing paradigm [SCZ+16] and *Edge Intelligence as a Service* [RD21].

Video Analytic can benefit from Edge Computing and EI, due to the wide distribution of edge devices with camera and video recording capabilities. Computing work can be performed on each device, so the time to get results, e.g. to search for someone in a video or image, could be decreased tremendously [SCZ+16].

Smart Home devices produce an enormous amount of data, so these data must be considered in privacy considerations regarding processing data on central Cloud Computing devices. Edge Computing and EI can prevent such privacy difficulties by allowing only the processing of the data in the home environment [SCZ+16].

Because of the large data quantity, e.g. heavy traffic workloads, or applications that require a low network latency, the Edge Computing paradigm with help of EI can improve **Smart City** computing structures. Regarding applications where the geographical location of the data used is more important, computation can also be performed on the edge instead of in the cloud [SCZ+16].

Collaborative Edge connects multiple physically distributed factions and allows data sharing and collaboration with these data. In [SCZ+16] an example of collaborative Edge Computing is shown in relation to healthcare cooperation.

[RD21] shows applications in different domains and their classification into different EI levels. In this thesis, we will focus on **low latency applications**. These are applications or functions where users only send one single request and demand on a low network latency towards end-user satisfaction. Examples of such applications are microbenchmarks such as floating point arithmetic operations, solving linear equations, solving linear equations [KL19], or ML steps such as data pre-processing, model training, and model serving which are used in [RRD21]. More detailed workloads are mentioned in Section 2.3.2

2.3 Serverless Computing

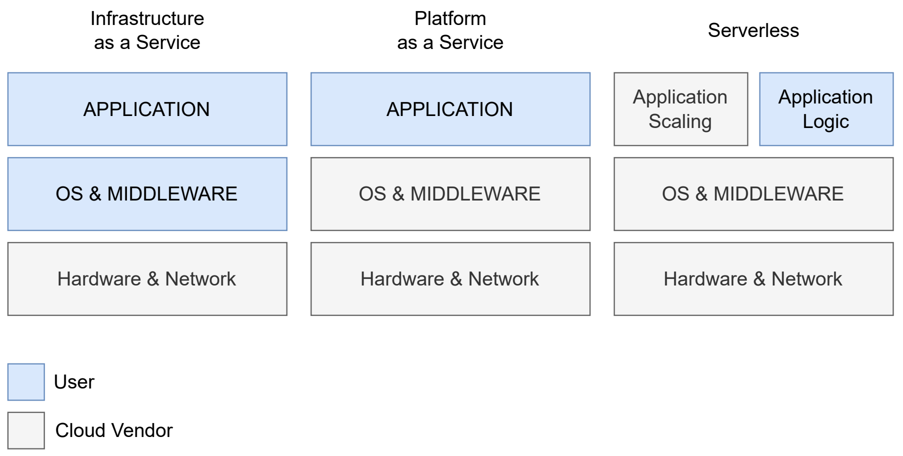


Figure 2.3: Differences between IaaS, PaaS and Serverless Computing

This paradigm abstracts the infrastructure components, operating system, middleware, runtime, and other parts that are needed to run an application away from the specific application logic. As distinct from *Infrastructure as a Service* (IaaS) and *Platform as a Service* (PaaS), *Serverless computing* allows the developer to focus on the application logic and allows to specify scaling and resource configuration, depending on the service provider. In Figure 2.3 the differences are briefly shown. Section 2.3.1 introduces containerization, which allows the provider of the serverless computing platform to deploy functions or applications on demand using a stateless container.

Function as a Service

In this thesis, we focus on *Function as a Service*, where only single functions are going to get deployed, instead of a whole application. These functions get invoked by events like e.g. timer events or normal user requests and have only one single specific task to fulfill. In Section 2.2.4 some of these functions are already listed (low latency applications). Because of this modern form of software development and deployment, the paradigm allows the developer to back away from the time-consuming monolith development and focus on single code components. This enables the platform to autonomously adapt faster deployments, reducing service provider costs, because only function usage has to be paid, loss of environment control, and higher degree of testing trouble, due to the difficult debugging strategy with regard to multiple function deployments [SS18]. Scheduling, scaling and load balancing are strategies that go hand in hand with *FaaS* platforms and must be handled differently depending on the scenario, available resources and other influences such as user behavior, number of function requests, or, for example, the technical coverage in Edge Computing networks. In Section 2.3.1 an open-source framework for serverless computing is introduced and in Section 2.3.2 typical workloads and application types are presented.

2.3.1 Kubernetes

Container Virtualization

Beyond virtual machines and unikernel virtualization, container virtualization also plays an enormous role in modern service and application deployment. In contrast to the hypervisor-based virtualization methods mentioned above, containers only share the same kernel and are constructed to isolate the software from the current environment. This enables the software, which is supposed to run in containers, to get deployed on different kinds of devices, independent from the operating system. In a single container, the necessary components like libraries, source code, environment variables, and other dependencies are packed together. One of the most widely used container-based frameworks is *Docker* [1], which requires only a single text file (*Dockerfile*[2]) to build an application image for the container. It is supported by the majority of cloud provider platforms and is open source. Docker images, which represent an executable, static, and complete form of a function or application, are used in these containers. The contained application or function is available during the runtime of the container. Kubernetes[3]

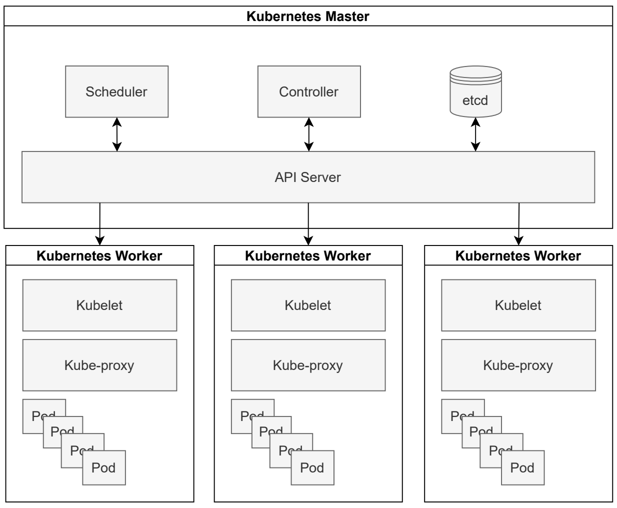


Figure 2.4: Kubernetes architecture

is a container orchestration platform that can be deployed on physical hardware, like a normal laptop, clouds, or VMs. It consists of the following elements:

- The **control pane node** is responsible for the orchestration of the worker nodes and all administrative tasks e.g. scheduling. There can be multiple control pane nodes in the cluster, but only one of them can orchestrate the cluster at the same time. Communication with this node is done via CLI, API Server (RESTful), or

[1]https://www.docker.com/
[2]https://docs.docker.com/reference/dockerfile/
[3]https://kubernetes.io/docs/concepts/overview/components/

graphical dashboard. It consists of a scheduler, a (cloud) controller manager, an API server, and the etcd key-value store.

- **Worker nodes** can be seen as devices that can run Linux containers. Users who want to use a service of the cluster can only communicate with the worker nodes. These nodes consist of a network proxy that listens to the API Server, a container runtime like *Docker*, *rtklet*, or *containerd*, and an agent named *Kubelet* who is responsible for communicating with the control pane node and receiving the pod definition through the API server. The worker nodes are based on the Kubernetes Container Runtime Interface (CRI). [4]

- A **Pod** contains multiple running containers and is used by the scheduler to provide redundant services on multiple nodes. The container healthiness of the pods is monitored by the Kubelet agent.

- **Etcd:** This is open-source software that contributes a distributed key-value store that saves the state of the cluster. It can be deployed by default on the control pane node or on other nodes, has a very small memory usage, and provides a redundant, fast-accessible, and resilient data store for the updatable node configuration files.

2.3.2 Workloads

In this section, common workloads and usage patterns are presented. This is done for the sake of clarity and comprehensibility and because of the later executed experiments. To select the appropriate workloads for evaluation and experiments, a general overview is necessary. Regarding [CMT16] the following workloads are defined:

Web Workloads

These workloads consist mainly of common HTTP requests and can be divided into some subsections like conventional web workloads where users request pages in a *periodic usage behavior*, shopping service workloads, online auctions, web robot traffic, or workloads concerning the web content which can be highly dynamic nowadays. Also, the page & traffic properties, the access patterns, and the user behavior play an important role in this workload type [CMT16].

Online Social Network Workloads

Online Social Network workloads can be divided into *General-Purpose Services*, such as Facebook where user profiles, user activities, and social interactions influence the workload and user behavior. E.g. the small-world phenomenon, which means that users usually form groups that are closer connected and have more latent interactions between users than direct ones. *Microblogging Services* like Twitter, which focus on content

[4]https://github.com/kubernetes/community/blob/master/contributors/devel/sig-node/container-runtime-interface.md

propagation and social interaction, *Visual Content Sharing Services* like Flickr which enables the upload, organization, dissemination, and rating of content and *Location-Based Services* like Instagram where the location to user relation, network structure, and mobility pattern has a big influence on the user behavior and workload [CMT16].

Video Service Workloads

These workloads are governed by the media properties, the traffic properties, the social sharing properties, and the user behavior itself. *Media Content Services* like Netflix generate workloads that have to consider the request arrival process, video popularity, content access patterns, and interactive user behavior. Workloads in *Video Sharing Platforms* like Youtube are concerned with traffic patterns, uploading and usage patterns, popularity evolution, and social interactions [CMT16].

Cloud & FaaS Workloads

Because of the high variation of cloud application types and the scaleable ecosystem of cloud providers, it is necessary to have an exact look at the workloads that appear in these systems. To have a deeper understanding usage patterns, arrivals processes, and cloud workload patterns have to be analyzed [CMT16]. In [SFG$^+$20] the *Azure Public dataset*[5] got analyzed and it shows that the trigger types in Cloud Computing are mainly HTTP requests (35.9%), followed by queue (33.5%) and common trigger events (24.7%). Timer, storage & orchestration events are between 0.7% and 2% with respect to the total number of invocations. It also emerged that 54% of all applications deployed only have one function. This indicates that the *Function as a Service* paradigm is a widespread approach [SFG$^+$20].

Concerning the invocation pattern, it can be outlined that 50% of the invocations do not show variation. The average interval between invocations is at most once per hour in 45% of the applications and at most once per minute in 81% of the applications. This concludes that most applications are invoked very infrequently. The warm-keeping costs of application and function containers are going to be higher regarding the total execution time. 18.6% of the most popular functions, those that are invoked on average at least once per minute, represent 99.6% of all function invocations. According to [SFG$^+$20] the average execution time of applications is between 200ms and 2 seconds at the median. Memory usage is present in 90% of the applications never over 400MB and in 50% of the applications at most by 170MB. Memory is an important factor because of the warm-up, allocation, and keep-alive decisions that have to be made in FaaS orchestration. These parameters will also help to construct realistic workloads for later evaluation and experiments. The *FaasProfiler* [SBW19] is built for generating workloads and testing FaaS platforms with Apache OpenWhisk[6]. It offers several benchmark applications

[5]https://github.com/Azure/AzurePublicDataset
[6]https://openwhisk.apache.org/

(functions), like sentiment analysis, string auto-completion, or e.g. image resizing, and allows us to specify the workload distribution type, test duration, and other parameters.

Mobile Device Workloads

Nowadays mobile devices dominate the situation of connected things on the Internet. These workloads are the majority in wireless networks and can be characterized by their typical task offloading and the varying traffic load. Because of the limited resources in mobile devices, the need for computation power in the current radius of the device, e.g. offered by cloudlets, is high. In Section 2.2.2 the related Mobile Edge Computing paradigm is mentioned. In [SAA21] augmented reality, speech recognition, language translation, and navigation applications were mentioned and analyzed as examples for mobile applications and mobile workloads by using CloudSim. In [YQZ+15] usage of mobile data, mobility patterns, and application usage of different user groups were analyzed toward a metropolitan data set in China. They conclude that heavy users, which are approximately 1% of all users, are the main drivers of mobile traffic. Approximately 80% of the mobile traffic generated was generated from social networks, e-commerce, advertisements, and search requests, and half of the users use more than five different application categories per day. The number of cells where users log in while changing their location is about ten cells per day. The most time is spent on social networks and e-commerce and less time on email services. Periodic usage patterns are mainly discovered in email and online gaming applications, social media and e-commerce applications do not show a high variation in the usage pattern. The analyzed video application usage shows that users only watch video if the network bandwidth is good enough because of the large traffic volume of videos. News browsing is also done only in short time intervals, indicating that users do not stay long on news sites. On the other hand, users visit social media and search for services for longer periods [YQZ+15].

2.4 Faas-sim

This Python-based discrete event simulation tool was developed at the Distributed System research department[7] at TU Wien. In the next sections, the foundational concept and components of *faas-sim* are presented.

2.4.1 Concept

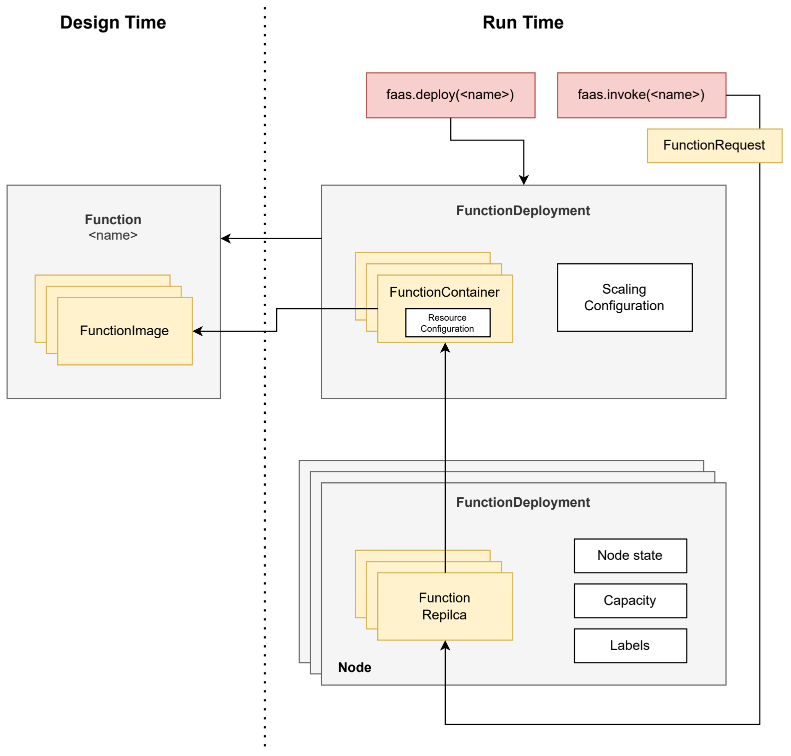


Figure 2.5: Function and Deployment concept in faas-sim

Faas-sim receives different input parameters to calculate and simulate serverless Function-as-a-Service platforms. It takes a network & cluster topology configuration and a benchmark setup which represents a distinct simulation experiment. The conceptual model of *faas-sim* consists of the following components and is shown in Figure 2.4.1:

- **Function:** This component is an abstraction at the design time of a single functionality with respect to the FaaS paradigm. Section 2.3 describes this paradigm. It is recognizable by a unique name and the function can be invoked by a *FunctionRequest*.
- A **Function Image** is a specific type of *Function* implementation, which can consider different deployment platforms to configure e.g. TPU, GPU, or CPU-based

[7]https://dsg.tuwien.ac.at/

implementations. With this possibility of variant images, the resource scheduler is enabled to make the decision on which image is used.

- A **FunctionDeployment** is a definite function instance with a specific allocation and scaling configuration and includes several **FunctionContainer** which represents a *FunctionImage* in runtime. The container includes the resource usage (VRAM, CPU, etc.) configuration.
- The **FunctionReplica** is the running instance of a *FunctionContainer* in a cluster **Node** and can be conceptually seen as a typical Docker container.

Ether

The Python-based tool is used to generate the edge infrastructure topologies in *faas-sim* and allows to evaluate strategies for resource allocation and capacity planning. It implements different cloud region scenarios like industrial IoT scenarios or urban sensing scenarios which can be used for further simulation.

Simpy

SimPy is a Python-based framework and is used to handle events and asynchronous (background) processes in *faas-sim* by offering a usable concurrent environment. It allows to define shared resources and capacity limitations.

Skippy

Faas-sim takes use of the Skippy scheduler presented in [RRD21]. Skippy is basically a container scheduling system that extends serverless frameworks like Kubernetes to work with edge functions. In *faas-sim* only a certain code is used which is needed for scheduling. The Kubernetes API is not triggered.

Request patterns

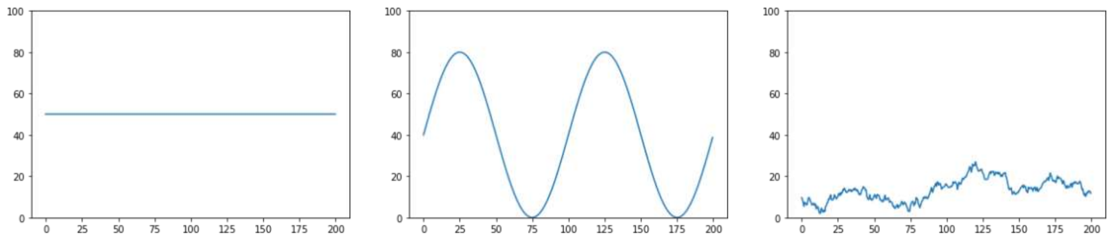


Figure 2.6: Constant, sine wave and random walk pattern

Faas-sim provide different workload request patterns per default. In principle, it allows one to configure the following request pattern profiles and mutate them with arrival profiles to get more realistic profiles that represent the request value per timestamp.:

- *Constant*: In this pattern, a variation of the instructions (invocations) per second is not possible. It fires e.g. ten requests every one second.
- *Sine*: In this pattern, the requests are addicted to the sine wave function. It fires requests concerning a given maximal value and period (time between peaks) value.
- *Random Walk*: For this pattern standard deviation and a maximal and start value have to be configured. The spikes (higher values) will increase with the request time.

In Figure 2.6 the three pattern plots are shown. The arrival profiles then readjust the above patterns to handle zero values like in the sine wave pattern. In *faas-sim* it is possible to save and load these profiles.

In *faas-sim* the implementation of a function can be configured by customizing the different states of a serverless function in a so-called *FunctionSimulator* which is influenced by *OpenFaas*[8]. It enables the modification of the *deploy*, *startup*, *setup*, *invoke*, and *teardown* behavior of a serverless function. Every time a request is sent, the implementation of the *invoke* method simulates the behavior of the function, blocks resources, and consumes time in the *Simpy* environment. Similarly in the other states, the environment can consume time and resources of the replica which is associated with a node. *Faas-sim* also provides two *Watchdog* modes, namely *Forking* and *HTTP*. Watchdog starts and monitors functions, works like a reverse proxy for running functions, and is inspired by the official OpenFaas implementation[9]. *Faas-sim* offers following different examples for *FunctionSimulator* implementations:

- **ForkingWatchdog** enables a mechanism for queuing and simulating works. If, i.e. a worker becomes available, it will claim the resources after it receives a token.
- **HTTPWatchdog** claims the resources and executes the function after each request immediately. Therefore, it does not generate a delay.
- **TrainingFunctionSim** uses the ForkingWatchdog implementation to simulate the training of an ML model. It claims the resources per request and simulates the download, training, and upload of the model per request. It also simulates a docker pull command for deploying the function and the start-up and tear-down phase.
- **InferenceFunctionSim** simulates the model download the resources needed to cache it and the inference per execution. It uses, in contrast to the TrainingFunctionSim, the HTTPWatchdog implementation. During the setup phase, the basic CPU usage and the basic memory consumption are going to be claimed.

[8]https://www.openfaas.com/
[9]https://github.com/openfaas/of-watchdog#modes

- **PythonHTTPSimulator** enables a simple implementation of function invocation for sending concurrent requests.
- **AIPythonHTTPSimulator** shows how a combination of AI-specific preprocessing, training, and inference could look together in a single implementation. It estimates the additional time needed if concurrent requests happen and also simulates the model down and uploads.
- **InterferenceAwarePythonHttpSimulator** extends the functionality of *AIPython-HTTPSimulator* and adds degradation for inference.

Methods

Faas-sim provides a basic interface consisting of the following methods:

method name	**description**
deploy	It takes a FunctionDeployment and deploys multiple FunctionReplicas concerning the ScalingConfiguration.
invoke	When the function is deployed it is able to get invoked. With the invoke method a user is able to send a *FunctionRequest*. The load balancer then selects a replica and fires the invoke method of the function simulator associated with the function.
remove	It removes all replicas fires the teardown method and frees all resource allocations in the environment.
get_deployments	If instances of *FunctionDeployments* are available, this method returns a list of them.
get_function_index	This method returns a dictionary of all available *FunctionContainer* regarding their *Function* names.
get_replicas	It takes a function name and state and returns a list of corresponding *FunctionReplicas*. If the *state == null* the method returns all replicas. Possible states are *CONCEIVED*, *STARTING*, *RUNNING*, or *SUSPENDED*.
scale_down	This method takes a *Function* name and the number of replicas which then get removed from the cluster. The method removes the most recently deployed replicas.
scale_up	This method takes a *Function* name and the number of replicas which then get added to the cluster if the maximum
discover	By forwarding the *Function* name this method returns a list of the associated *FunctionReplicas*
suspend	Tears down all running replicas by forwarding the specific *Function* name
poll_available_replica	Continuous polling of available function replicas by forwarding the specific *Function* name and an interval

Table 2.1: Methods overview of *faas-sim*

CHAPTER 3

Related Work

In this chapter, a selection of cloud, fog, and edge-based simulation and emulation frameworks is presented in Section 3.1 and Section 3.2. Five simulation and emulation frameworks and two hybrid frameworks were investigated. In addition, a collection of current serverless benchmarking tools is listed in Section 3.3. The results of this related work research are going to answer the first research question shown in Section 1.2. The gathered related work will give an overview of the existing frameworks and benchmark suites, that are currently offered in this research area. The selected works are summarized, and their competencies are listed in detail in the evaluation Chapter 7.

3.1 Simulation Frameworks

In this thesis, several simulation frameworks were investigated. The event-driven and layer-based simulator *CloudSim* allows to model of large-scale topologies and manage e.g. the memory, storage, and bandwidth control. It is very flexible and customizable regarding its own workload request allocations or performance tests [CRB$^+$11]. *IFogSim* is also a layer-based simulation framework. It is Java-based and extends the CloudSim framework. *IFogSim* allows to simulate of edge and fog networks and consists of different components like sensors, actuators, fog devices, etc... It has also a monitoring service for investigating resource usage statistics [PVCM20]. Based on *IFogSim* an extension named *IFogSim2* exists . It enables new components like Microservices, Clustering, or Mobility [MPGB21]. The *EdgeCloudSim* simulator is also a CloudSim extension. Based on five different expandable modules, the framework enables new configurable parameters like the definition of mobile device places according to a mobility motion model [SOE17].

3.2 Emulation Frameworks

Beyond the simulation, also emulation frameworks play a necessary role in this field. Tools like *EmuFog*, which is based on *MaxiNet*, allow to generate fog computing infrastructure topologies and enable different parameter configurations for the emulation setup [MGG$^+$17]. When combining simulation and emulation frameworks the area of hybrid frameworks is showing up. *EMUSIM* and *EmuEdge* are two of this frameworks. They are connected with local or external infrastructure and extend also a simulation environment. *EMUSIM* can produce configuration files for deploying the locally tested setup into production infrastructure, where *EmuEdge* can also run *MiniNet* containers [CNRB13] [ZCS19].

3.3 Benchmarking Suites

Seven benchmarking suites were selected during the research. They differ from each other in particular points. Some of them, like *FunctionBench*, can only offer micro-benchmark workloads, which are e.g. simple mathematical operations [KL19]. Other frameworks, like *Faasdom* or *iBench*, have already implemented machine learning training or inference models in their workload proposition [MFKS20] [BBS$^+$20]. The potentially usable metrics reach from End-to-end latency, processing time, round trip time, and function execution time to model accuracy or e.g. the power consumption. The detailed review and evaluation of all suites are shown in Section 7.2.

CHAPTER 4

Methods

This section describes the approaches and methods that are used and applied in this thesis. To gain a better understanding, the methods are segmented in the order in which they are used and applied in the subsequent implementation phase. In Chapter 4 the methodical approach can be seen and which research question, shown in Section 1.2, is going to be answered by which stage of the methods. It starts with a literature review and related work research. The results of them will then be used for the requirement engineering and to answer **RQ1**. From this point, the topology generation and workload development are the subsequent steps, which will be applicable to the already mentioned testbed and simulation framework *faas-sim*. Furthermore, several scenarios for the experiments are going to be developed, based on the previously developed topologies, request patterns, and workloads. After that, all important and fundamental elements are gathered for answering **RQ2** and starting the testbed profiling and scenario experiments. The results of the profiling experiments, topology generation, and request pattern generation are then used in the simulation runs by *faas-sim*. In the evaluation and result presentation phase, the experiment and simulation results will be compared and illustrated. This will answer **RQ3**.

4.1 Literature Review & Related Work Research & Requirement Engineering

This thesis uses systematic literature research to gather articles and related work. With the help of a well-defined requirement engineering phase, the potential metrics for the benchmark will be explored. The search is carried out only with online research, where common digital libraries, such as IEEE Xplore[1], are used to search and access journal articles, conference journals, technical standards, and related work materials.

[1]https://ieeexplore.ieee.org/

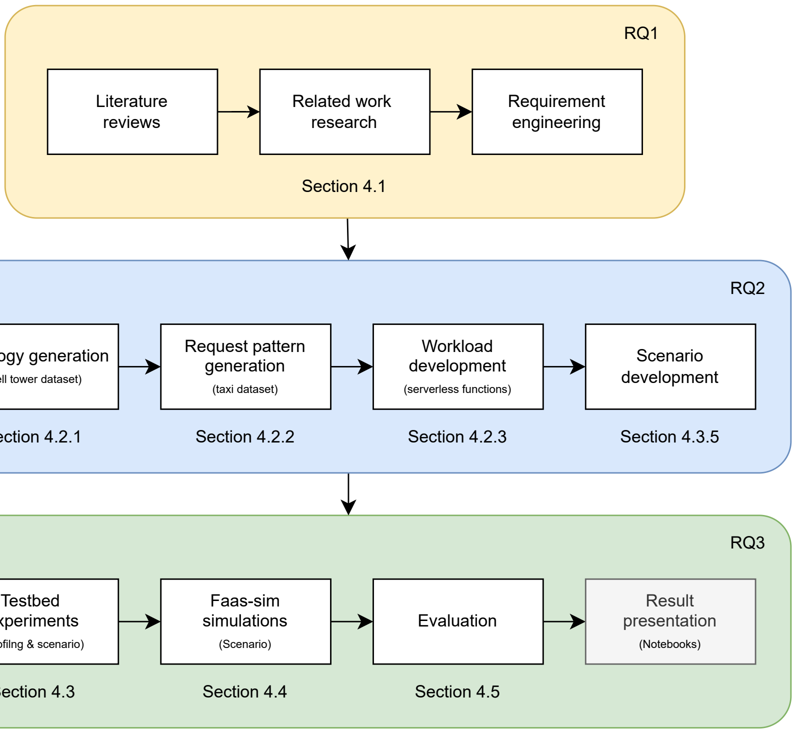


Figure 4.1: Methodical approach

To limit the amount of results while searching and increase the quality of the gathered literature and papers, criteria like the publication date, language, type, and number of citations are applied as search constraints. The results are then analyzed by quality check to see if the criteria are met and summarized. The aim of this step is to also evaluate the essential aspects in text form, to present them in a comparative manner, and to present and evaluate the essential findings of the included studies to answer the research questions. From the information obtained after reviews of the literature and related research, the requirement engineering method defines how to select metrics for benchmarking.

4.2 Infrastructure topologies, request pattern & workload development

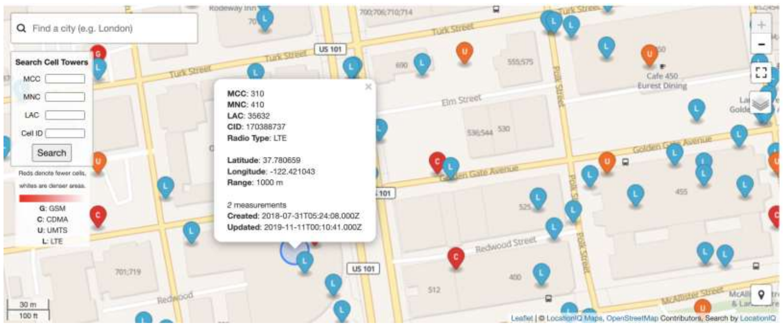
The creation of realistic infrastructure topologies, request patterns and workloads is necessary for the course of this work. Two open source datasets are the starting points for the development of the topologies and query patterns. LTE cells are filtered from the *OpenCellid* dataset and linked to the *NYC Taxi* event dataset. A corresponding cell must be mapped for each event so that the events can be divided into their occurrence areas.

4.2.1 Infrastructure topologies

When developing realistic infrastructure topologies, *OpenCellid* dataset is used as a baseline. To match various scenarios, this dataset is processed to extract different constellations such as metropolis topologies, average cities, or countryside locations.

OpenCellid Dataset

This dataset is open source licensed under the *Creative Commons Attribution-ShareAlike 4.0 International* license and includes worldwide data of geo-distributed cell towers. The corresponding database is being updated daily and can be downloaded as a *CSV* file for free. The database entries consist of several parameters like cellular network technology, mobile country code, Longitude, Latitude, unique identification number, etc. The complete list of parameters is listed in [Lab]. The cellular network technologies are GSM (Global System for Mobile Communications), CDMA (Code Division Multiple Access), UMTS (Universal Mobile Telecommunications System), and LTE (Long Term Evolution). For the purpose of this thesis, only the data entries with LTE network technology are used to determine the position of the specific network nodes. In Figure 4.2 an example of multiple network cells is shown, rendered in a map view.


MCC: 310
MNC: 410
LAC: 35632
CID: 170388737
Radio Type: LTE

Latitude: 37.780659
Longitude: -122.421043
Range: 1000 m

2 measurements
Created: 2018-07-31T05:24:08.000Z
Updated: 2019-11-11T00:10:41.000Z

Reds denote fewer cells, whites are denser areas.

Figure 4.2: Example screenshot from https://www.opencellid.org/

Topology extraction

To find an appropriate topology for a related scenario, the dataset will be pre-processed and filtered by a Python script, which can handle the following input parameters:

- topology name (e.g. new_york_2x2) as *string*
- city name (e.g. New York) as *string*
- latitude (e.g. 40.754380) as *float* number
- longitude (e.g. -73.984986) as *float* number
- width in kilometers of the area (e.g. 2) as *float* number

- height in kilometers of the area (e.g. 2) as *float* number
- cloudlet area width in km (e.g. 1) as *float* number
- cloudlet area height in km (e.g. 1) as *float* number

The *topology name* specifies the distinct topology area for later reuse and clarity. When entering the parameter *city name*, the script uses the central point of the city. It is also possible to specify custom *latitude* and *longitude* values. This location is the center point of the filtered area and is limited by the parameters *width* and *height*. The script then only considers cells in this specific area. The rectangle area is the size of the width and height entered. The *cloudlet area width* and *height* define the sub-areas in kilometers. All cells are going to be associated with a cloudlet area. These cloudlets will handle all requests that are sent to a specific cell in this area and will be used for the request pattern generation later. In Section 4.2.2 an example area of filtered cell towers is shown schematically. The correlating scripts are published on a public repository[2]. The implementation is explained in detail at Section 5.2.

Faas-sim topology creation

For the simulation in *faas-sim* a own topology of the testbed is already implemented in the *FaaS Sim Evaluation*[3] repository. It uses the preexisting topology implementation of *Ether* and consists of three different scenario parts, namely the Cloud Scenario, IoTBox Scenario, and Cloudlet Scenario, which are all combined into the Testbed Scenario. All nodes are figured with the parameters, like CPU, architecture, and memory size, of the real testbed. Also, links between the nodes are created in the simulation topology. Section 4.4 describes, how the latency of the real testbed nodes is measured for these links.

4.2.2 Request pattern

The *edgebench* project contains already implemented profiles & patterns which can be used for later experiments and simulations. In addition, Section 4.2.1 shows how the request pattern generation method takes advantage of the *OpenCelliD* dataset and extracted topologies. In combination with the *NYC Taxi* dataset, where pickup and drop-off events are recorded, the request pattern will be created. The generation method can be used with other databases similar to the *NYC Taxi Dataset*. The detailed procedure is described in the following section.

[2]https://github.com/edgerun/faas-topologies
[3]https://github.com/edgerun/faas-sim-evaluation/blob/main/evaluation/simulation/topology/testbed.py

NYC Taxi Dataset

This dataset[4] from the year 2013 includes taxi pickup and drop-off location data, date time information, and other parameters, which were recorded in New York City, United States. The 28,85 GB database is split into multiple CSV files, where one file contains the data for a whole month. An entry in the CSV has many different columns. For the purpose of this thesis, the important and required columns are the pickup longitude and latitude, the passenger count, and the pickup date & time.

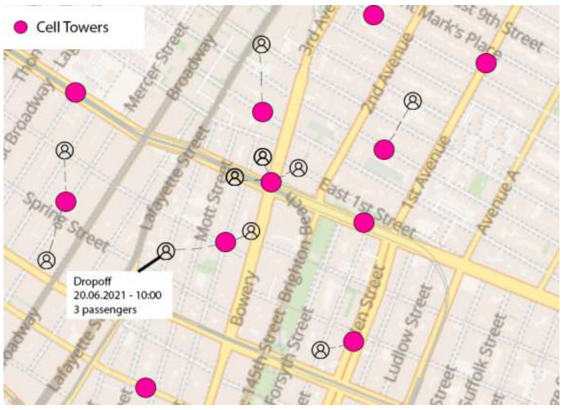


Figure 4.3: Example cell tower map with *NYC Taxi Dataset* events

Prepare Trip Dataset

Before the trip dataset is used to generate the request pattern, it must be prepared by removing the unnecessary columns and limiting the entries by the pickup location (latitude & longitude). This is done by a script, which takes a trip database file and an extracted topology from Section 4.2.1 as input and calculates the maximum and minimum boundaries of the topology. After that, it limits the *NYC Taxi Dataset* by these boundaries, so the prepared dataset contains only entries in the area of the topology. In Section 4.2.2 the pickup events are schematically shown in the area of the cell tower topology.

Generate Request Pattern

To generate a realistic request pattern, pickup events must be associated with a cell tower. We define a request pattern as an list of time delta entries, which allows to create various pattern with different intervals between the events. The time deltas are defined in seconds. The association of events and cell towers is done by searching for the nearest cell tower. The range in which cell towers can receive requests is variable and depends on the infrastructure environment and the obstructed technologies in the cell. Due to this missing information, the communication range of an LTE cell tower is

[4]https://github.com/andresmh/nyctaxitrips

defined, in this case, as *500 meters*, based on previous research and data on 5G (26Ghz) connectivity collected from Samsung in 2018[5]. If no cell is in range of the pick-up event, the event is ignored. When all events are associated with a cell tower, the script creates the request pattern by calculating the timestamp delta of all pick-up events in the distinct cloudlet area. Because every cell is linked to a cloudlet, the script generates a request pattern file for each cloudlet in the topology. If a trip event includes more than one passenger, the script counts these events as multiple events and adds a time difference of 0.1 seconds between the single events. If e.g. a trip event includes three passengers, then three requests are sent to the cell tower with a gap of 0.1 seconds. This gap is necessary for the later use of the request pattern files in the experiments and should simulate concurrent request activities. In Section 4.2.2 the cloudlet areas are shown and indicated by the rectangles numbered 1 to 4. The advantage of this method is, that

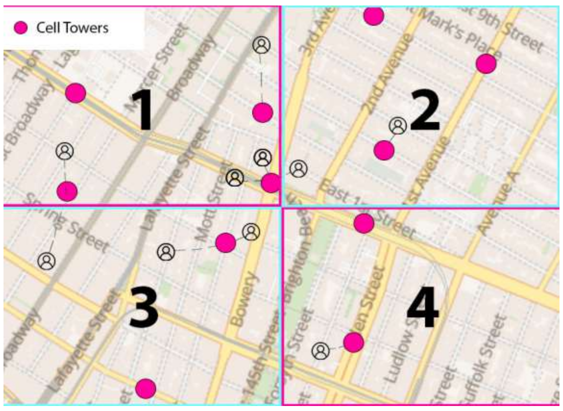


Figure 4.4: Example cell tower map with cloudlet areas

it is possible to generate patterns for different time ranges and special periods within a whole year. With the generated data, it is possible to analyze it and search for "Low" (minimal number of requests), "Normal" (average number of requests), or "Intensive" usage patterns (maximum number of requests) per cloudlet. This will become important to cover different use cases in the experiment phase.

4.2.3 Workloads

The development of various AI-based workloads is necessary to cover the need for a balanced suite of serverless AI-related functions. Therefore, this section describes how workloads will be developed, which workload types are excluded, and what structure for the developed functions is chosen. Due to the already implemented setup of *OpenFaas* function deployment in the context of the existing testbed, provided by the *Distributed System Group at TU Wien*, this implementation method is mandatory. The implementation language will be adapted from the existing implementation; hence *Python* is chosen

[5]https://news.samsung.com/global/samsung-and-verizon-announce-first-5g-customer-trials-set-to-begin-in-q2-2017

for the development of the distinct functions. In addition to the developed function, a *Dockerfile* per function must also be created. The workloads can have various attributes such that different resource requirements are covered. Therefore, the workloads can be seen as *data intensive*, *latency sensitive*, and *accuracy critical* workloads, or a combination of them. When looking at *data intensive* workloads, these functions usually process a large amount of data, and *latency sensitive* functions are functions that depend on a minimal response time. The *accuracy critical* functions have to meet a high inference model accuracy. Due to the limited resources regarding this thesis, we focus on *inference only* tasks. Therefore, learning and training tasks related to ML are dismissed in this context. The ML models used in this thesis are pre-trained.

A single function may include the following steps: pre-processing; model loading; prediction and post-processing. Therefore, every function has to measure the time spent according to the different steps and to output the times in the corresponding *JSON* formatted output. The development will be distinguished between the so-called *One-Step* functions and inference pipelines. The pipelines offer a more realistic way to emulate real-world serverless function scenarios as One-Step functions.

One-Step Functions

These functions are common serverless functions with a unique purpose. They are not dependent on other functions and can be developed usually by implementing only a single method. Section 4.2.3 this concept is illustrated.

Inference Pipelines

Regarding this thesis, an inference pipeline, illustrated in Section 4.2.3, is declared as a serverless function that is based on other different serverless functions. Such inference pipelines can call other functions in a successive or concurrent way so that the return value of the pipeline is dependent on the return values of the other functions. The called functions can also be invoked as One-Step functions, from outside the pipeline.

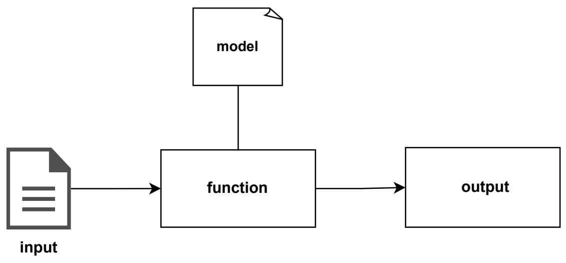


Figure 4.5: Conceptual illustration of an One-Step function

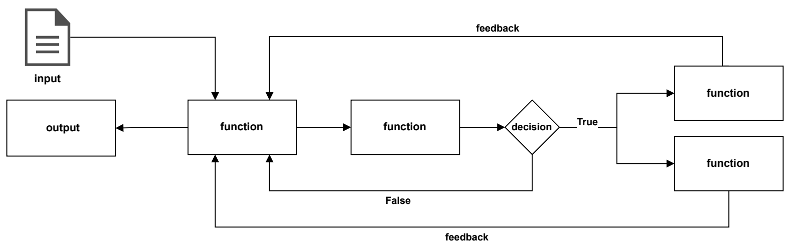


Figure 4.6: Conceptual illustration of an Inference Pipeline

Build & release:

The distinct functions will be built using the **faas-cli** client. To satisfy the distinguish architecture needs of the testbed hardware devices, the functions are built in different operating system architecture variants, namely *linux/arm32v7*, *linux/arm64v8* and *linux/amd64*. The functions are built and provided on the public docker registry *https://hub.docker.com/u/edgerun* and can be pulled, e.g. with the command *docker pull edgerun/function_name*. Related prediction models are uploaded to a private file server and can be fetched using the provided script.

4.3 Testbed Experiments

The empirical experiments for gathering the traces of the simulation are going to be divided up into single *profiling runs* and *workload scenarios*. The profiling runs perform experiments on only one serverless function at the same time, and the scenarios are going to cover multiple serverless functions concurrently. The serverless functions developed from Section 2.3.2 are going to be used as workloads for the experiments. The request pattern collected from Section 4.2.2 is used as input for request generation and association with the individual zones of the testbed.

4.3.1 Testbed setup

The provided testbed is a combination of multiple hardware devices that cover different responsibilities. It is divided into three zones (A, B, and C), where each of them includes a list of distinct nodes that own a unique IP address and can be called by SSH via the TU Wien VPN. Devices are, e.g., *Raspberry Pi*, *NVIDIA Jetson*, or *Xeon CPU* nodes that represent the worker and controller nodes and, e.g., an *Intel NUC* (Next Unit of Computing) node that will execute client requests.

4.3.2 Emulated network latency

To emulate the issues of WAN latency in the testbed setup, we take advantage of the Linux kernel component called *netem*[6]. It allows one to add and remove simulated

[6]https://wiki.linuxfoundation.org/networking/netem

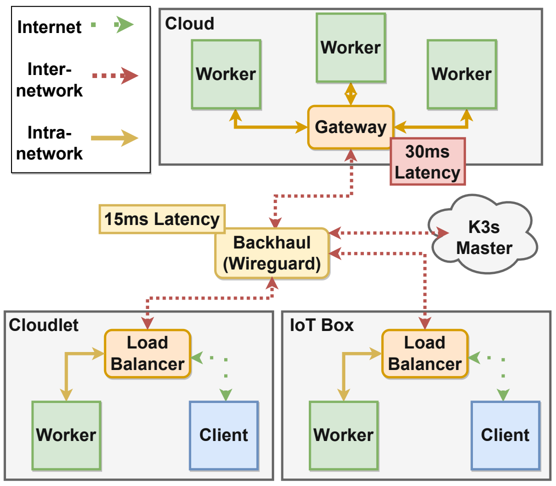


Figure 4.7: Testbed illustration with the cloudlet, IoT Box, and cloud zones

network latency rules to the Linux server. The following command allows, for example, to add 100 ms network delay to the ethernet interface of the server: *tc qdisc add dev eth0 root netem delay 100ms*. In [BZKH20] directly connected, edge and cloud setups were measured according to Mobile Augmented Reality application offloading. For the purpose of this thesis, the values for the LTE round-trip time in edge environments are used to add realistic network delay for the testbed nodes. For the edge zones a and b, a delay of 19.9 ms is chosen. [BHQT22] reported that the average network delay from 260 locations to the closest Amazon EC2 zone is approximately 74ms. This delay is used for the cloud cluster zone c.

4.3.3 Galileo

The experiments will be executed with the Galileo experiment repository [7] that is already implemented on top of *galileo* and the Galileo shell and is a framework for distributed load testing. It allows the recording of HTTP traces, and telemetry data, and implements a container orchestration functionality. To adapt its own workloads and request pattern, the repository has to be extended by itself.

4.3.4 Profiling

Before the simulation can be started, the applications must be profiled, to get a realistic distribution of the function execution time and the used resources. For this, the single functions will get deployed on the same testbed node as the scenarios were executed, with the difference, that the workload profiles are fixed to 100 requests, where every two seconds a single request is sent to the deployed function. With the help of the running telemd[8] daemon, the system data of the node will be gathered. In Section 6.3,

[7]https://github.com/edgerun/galileo-experiments
[8]https://github.com/edgerun/telemd

the profiling experiments are going to be described in detail.

4.3.5 Experiment Scenarios

In this section, the different scenarios for the experiments are defined. The scenarios attempt to cover a realistic scope of usage patterns and topology combinations. They will be divided into around 12 different setups according to the limitation of the testbed and the scope of the thesis. Because the testbed only offers an emulation of three cloudlet areas, the experiments will be fitted to this requirement. Therefore, the request pattern generation will be suited to a maximum of three cloudlet areas, where the generated pattern will then be associated with the testbed zones. When, for example, four different patterns were generated, pattern 1 and pattern 2 are going to be associated with zone A and zone B and pattern 3 with zone C. In our case, the last pattern will be ignored. Because the script is flexible, it would hypothetically be possible to scale the scenarios to multiple cloudlet areas. In the next section, the exact breakdown of the scenarios is shown.

Scenario 1, 2 & 3

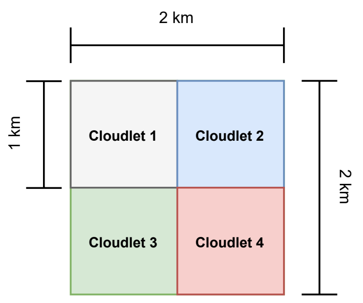


Figure 4.8: Cloudlet setup for scenario 1, 2 & 3

These scenarios are going to cover a wider city area with 2 km × 2 km in combination with large-scale cloudlet areas that have an area of 1 km × 1 km. This topology has four expected cloudlet areas, which are shown in Figure 4.8. But in contrast to this assumption, the cloudlet area 3 and 4 are going to act as areas, where requests are sent to a cloud provider and not to a nearby cloudlet. The latency will therefore increase in these areas, so the function execution times are not as good as in the cloudlet areas.

- Topology size: 2 km × 2 km
- Cloudlet size: 1 km × 1 km
- Timespan: 10 min of max., avg. & min. requests
- Expected cloudlet areas: 2
- Expected cloud areas: 2

Scenario 4, 5 & 6

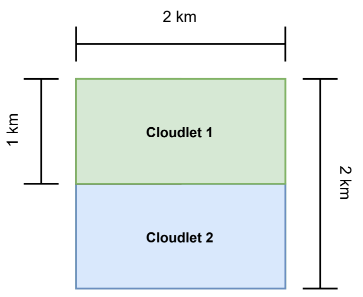


Figure 4.9: Cloudlet setup for scenario 4, 5 & 6

The experiment setting for these scenarios is similar to the previous scenarios with the difference being that there are only two areas, the cloudlet area is much larger than the previous one and none of the cloudlets will act as a cloud provider. It shows a setup where the cloudlet coverage is not as good as in highly advanced smart city environments. Therefore, this can be seen as a worst-case scenario. The resource usage in these areas will be an interesting object of investigation.

- Topology size: 2 km × 2 km
- Cloudlet size: 2 km × 1 km
- Timespan: 10 min of max., avg. & min. re-

quests

- Expected cloudlet areas: 2

Scenario 7, 8 & 9

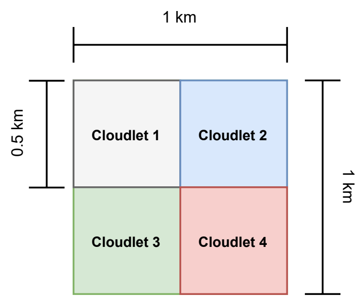


Figure 4.10: Cloudlet setup for scenario 7, 8 & 9

Scenario 7 - 9 will cover only a topology size of 1 km × 1 km and will allow examination of a cloudlet area of only 0.5 km × 0.5 km. This should simulate an average smart city setting. Like in Section 4.3.5, also the areas 3 and 4 will act as a cloud provider. It will show if the area reduction has a significant effect on the benchmarks or not.

- Topology size: 1 km × 1 km
- Cloudlet size: 0.5 km × 0.5 km
- Timespan: 10 min of max., avg. & min. requests
- Expected cloudlet: 2
- Expected cloud areas: 2

Scenario 10, 11 & 12

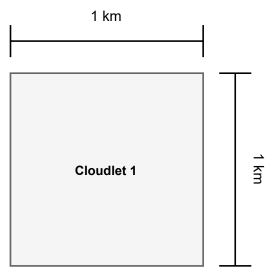


Figure 4.11: Cloudlet setup for scenario 10, 11 & 12

Like in scenarios 7, 8 & 9, this will also cover the same topology size of 1 km × 1 km, but it will handle only a single cloudlet area by setting the cloudlet area to the same size as the topology area. It is also not an optimal setting for smart cities. This case should show if such a setting has a bad impact on resource usage and function execution times.

- Topology size: 1 km × 1 km
- Cloudlet size: 1 km × 1 km
- Timespan: 10 min of max., avg. & min. requests
- Expected cloudlet areas: 1

Section 4.3.6 describes how these scenarios will cover the three different request pattern edge cases.

4.3.6 Edge Case Extraction

To also consider worst cases and not just regular scenarios, this part of the scenario and pattern creation phase enables the extraction of various edge cases. This expands the evaluation. First of all, the dataset allows us to extract three different edge cases to allow realistic experiment setups and to cover foreseeable scenarios: Time ranges where the most trips are registered, the average amount of events, and times when a minor amount of events are detectable. Due to the limitation of the experimental hardware resources, the time range is set to a maximum of ten minutes. First, the *NYC Taxi Dataset* will be iterated over all entries that occur in the corresponding topology. Because the dataset stretches over one year, the resulting amount of intervals would be around 50.000, therefore, the script will only consider a time range of three months (May, June, July), so the calculation effort is reduced. In order to avoid statistical outliers when extracting the ten-minute intervals, the method uses the fifth percentile of the result set as the minimum value and the 95th percentile as the maximum value. The average interval is closest to the mean value of all intervals.

4.4 Simulation runs

When finishing the profiling runs and the experiments of the scenarios, the simulation of the scenarios is going to be executed in *faas-sim*. It takes the resource usage results and execution times of the profiling runs, the generated arrival profiles, and the size of the single container images as parameters. For this evaluation, an own repository[9] was set up. It already implements the *ether* topology of the testbed and allows to create simulation scenarios by adapting the existing implementation of an example scenario, load balancers, and an inference function simulation. To prepare the simulation for the desired scenarios, the deployment of the clients must be adapted, the latency must be added for each device and the inference function has to be modified, so the profiling results can be used for each function. To get realistic latency for the simulation, the log-normal distribution of 100 ping requests to each of the devices will be calculated. Before executing the ping requests, the emulated latency must be added like in Section 4.3.2. The distribution calculation is required, because in real-world environments the latency fluctuates over time.

4.5 Benchmarking result analysis

After the simulation of the scenarios, the results of the simulation and the results of the experiments are going to be compared. The metrics CPU, FET, and RAM will be compared. The metrics are limited to the implementation of the *faas-sim* repository but can be extended in further research. The traces also need to be examined. The goal is to determine whether there are significant differences between the testbed experiments and the simulation runs.

[9]https://github.com/edgerun/faas-sim-evaluation

CHAPTER 5 Approach

The approach chapter presents a concrete way for gathering the related work, reviewing the literature, and finding the optimal requirements. It shows the structure and concrete implementation of the developed software parts and the adaptations that were made to existing software. It shows the coding languages, frameworks, libraries, and third-party software that were used to generate the workloads, topologies, and request patterns. In Section 5.2 the implementation of the generation of infrastructure topologies is shown, in Section 5.3 the generation of request patterns is presented, and in Section 5.4 the different workloads and their implementations of serverless functions are listed. Furthermore, Section 5.4.2 describes the implementation of an inference pipeline.

5.1 Literature Review, Related Work Research & Requirement Engineering

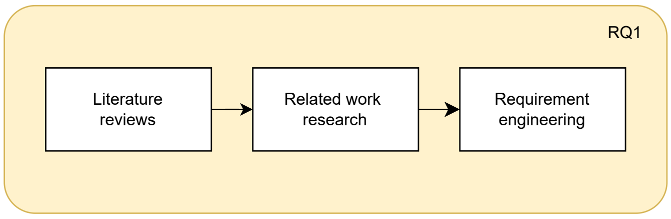


Figure 5.1: Approach towards RQ1

The following constraints and main categories, which then influence the search terms, are defined to meet the research area of this thesis:

- Edge & Serverless Computing
- Artificial & Edge Intelligence

- Simulation & Emulation frameworks
- Serverless Benchmarking tools
- Serverless, Geo-distributed, and AI workloads & applications

The following criteria are applied to narrow the potential search results:

- Because Edge Computing is more up-to-date than Cloud Computing, only *recent* (last five years) publications are considered when searching for Edge Computing related publications. Generally speaking, no publication older than about ten years should be considered due to the actuality of the topic.
- When searching for papers, articles, etc. only publications in English are accepted.
- The search will focus on work published in renowned peer-reviewed scientific journals and not on e.g. blog posts, company-based websites, or similar sources. It will not consider articles published in journals with a very low impact factor but in at least peer-reviewed conferences.
- Papers or articles with more citations in other papers are typically more trustworthy, therefore, this type of work is going to be examined with more priority than those without any citation.
- When a trusted source is found, the citations in these papers will also be examined, because of the high possibility of finding similar work related to the concrete topic.

The requirement engineering will focus on metrics of state-of-the-art Cloud Computing and Edge Computing benchmarking tools and will also take care of AI-based benchmarks. It will select the most common metrics that are examined in related work and Quality of Service (QoS) metrics specific to AI. Due to the limited scope of this thesis, not all metrics will be taken into account. The key metrics will be pre-selected by analyzing the summarized related work. This set of benchmarks is going to be filtered by the following questions:

- Is the metric necessary for the proposed evaluation and helps answer the research questions?
- Is it possible to implement this metric in the given experiment setup?
- How great is the benefit if metrics are selected for evaluation?

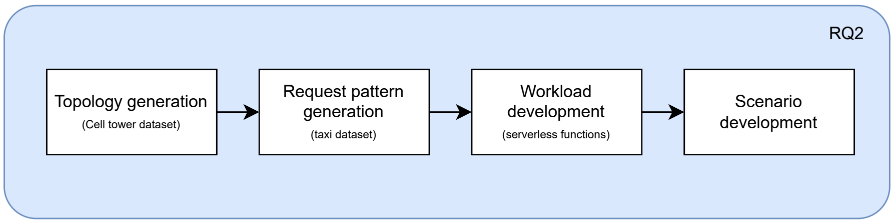


Figure 5.2: Approach towards RQ2

5.2 Infrastructure Topology Generation

When looking at Figure 5.2, four important stages of the approach towards research question two are shown. It starts with the topology generation. The infrastructure topology generation is implemented in Python. It is split into four steps: preparing and filtering the OpenCelliD dataset, combining the cells with a cloudlet area, and creating the Ether-based topology on top of it. In the first step (prepare), a simple script removes unnecessary columns from the dataset reduces the data by the given *radio type* parameter, and saves them as a new file for the next step. This is done using pandas, a Python-based data analysis library.[1] In Algorithm 5.1 the preparation steps are shown as pseudo-code. After that, it is possible to start the filter algorithm, where the user

Algorithm 5.1: Prepare dataset

```
1 dataframe = pd.readCSV(file);
2 dataframe.drop(['mcc', ..., 'created', 'updated']);
3 for row in dataframe do
4     if row['radio'] != "LTE" then
5         dataframe.remove(row);
6     end
7 end
```

can set specific parameters for generating location-based datasets. In Section 4.2.1 the available parameters are already listed. When using the parameter —-*city*, the script searches for coordinates using the geocoder geopy library[2], where it is possible to request location data regarding the name of a city. If no city is specified, the script takes the longitude and latitude parameters for filtering. Parameters *width* and *height* then narrow the already prepared cell data to a rectangle area with the corresponding height and width. In Algorithm 5.2 the pseudo-code is shown to filter the data set.

[1]https://pandas.pydata.org/
[2]https://pypi.org/project/geopy/

Algorithm 5.2: Filter topology dataset

```
if name != null then
    centerPoint = getLatLonFromCityName(name);
else
    centerPoint = (lat, lon);
end
data = pd.readCSV(file);
[maxLat, minLat, maxLon, minLon] = getBounds(centerPoint, width, height);
filteredData = data.between(minLat, maxLat, minLon, maxLon);
```

When the dataset was filtered, the *Create cloudlet membership* algorithm, shown in Algorithm 5.3, takes the parameters *width* and *height* and the maximum boundaries of the filtered topology dataset to determine the specific cloudlet areas for the corresponding cells. This is done by iterating over the longitude and latitude and adding the *height* to the longitude and the *width* to the latitude at every step of the loop. To do that, the kilometer values have to be transformed into degrees. If some cells are located between the new boundaries, they will be associated with the current cloudlet (iteration index i).

Algorithm 5.3: Create cloudlet membership

```
topology = pd.readCSV(topologyFile);
minLat, maxLat, minLon, maxLon = getMaxBounds(topology);
i = 0;
newLon = minLon;
oldLon = newLon;
newLat = minLat;
while newLon <= maxLon do
    newLat = minLat;
    oldLon = newLon;
    newLon = addKmToLon(oldLon, newLat, height);
    while newLat <= maxLat do
        oldLat = newLat;
        newLat = addKmToLat(newLat, w);
        toplogy.between(oldLon, newLon, oldLat, newLat)['cloudlet'] = i;
        i = i + 1;
    end
end
```

5.3 Request Pattern Generation

The generation of the request pattern is divided into three steps, namely the dataset preparation, the edge case extraction, and the pattern generation. First, the trip dataset will be narrowed by the max bounds of the topology file, which was created in Section 5.2. In addition, unnecessary columns, such as travel time or car medallion, are removed using the *Pandas* library. This is done to reduce the size of the file and increase performance.

After that, the resulting trip dataset will be used in the *Extract edge cases* algorithm, shown in Algorithm 5.5. The script reads the dataset takes the first and the last pick-up date-time value of the set and iterates these values by adding five minutes to the date time object in every iteration. In each step, the algorithm calculates the sum of measured passengers and adds the interval (trips in this time range) and the sum to a dictionary. When the loop is finished, the fifth percentile, the average, and the 95th percentile are calculated. After the calculation, the script saves the three intervals, which are closest to the values extracted before, as single *SVG* files.

From this point on, it is now possible to use the edge-case files to generate the specific pattern. The *Generate pattern* script, shown in Algorithm 5.6, loads the topology file and the edge-case trip file and iterates all trip entries. In every iteration, the *getNearestCell* method searches for the cell closest to the topology according to the longitude and latitude of the pickup event of the trip. The corresponding cloud area number is then added to the trip event. In the next steps, the trip events that hold more than one request will be split up into single trip events. The gathered trips are then grouped by the cloudlet area. For every cloudlet area, the specific set of trips is iterated, the difference between the timestamp then gets added to an array, and trips with the same timestamp will be adjusted to have a difference of 0.1 seconds. The first event will start with a difference of 0.5 seconds. The generated pattern will have the form of a list of time deltas and is going to be saved in a single CSV file for every cloudlet area.

Algorithm 5.4: Prepare dataset

```
1 trips = pd.readCSV(tripFile);
2 trips.drop(['medallion', ..., 'trip_time_in_secs', 'trip_distance']);
3 topology = pd.readCSV(topologyFile);
4 maxBounds = getMaxBoundsOfTopology(topology);
5 trips = trips.between(maxBounds)
```

5.4 Workload Implementations

This section describes the specific workload implementations. The workloads are grouped into so-called One-Step functions, where the purpose of the single function is straightforward, and inference pipelines, where different One-Step functions are used concurrently or sequentially. The developed inference pipeline (*Taxi Driver Safety App*) is inspired by

Algorithm 5.5: Extract edge cases

```
intervals = dict();
sumIntervals = dict();
trips = pd.readCSV(tripFile);
firstDate = min(trips['pickup_datetime']);
lastDate = min(trips['pickup_datetime']);
date = firstDate;
i = 0;
while date <= lastDate do
    start = date;
    end = date + '10min';
    date = date + '5min';
    temp = trips.between(start, end);
    sumPickups = sum(temp['passenger_count'];
    sumIntervals[i] = sumPickups;
    intervals[i] = temp;
    i = i + 1
end
p5 = np.percentile(sumIntervals, 5);
avg = sum(sumIntervals) / sumIntervals.length;
p95 = np.percentile(sumIntervals, 95);
trips_p5 = getClosestInterval(p5);
trips_avg = getClosestInterval(avg);
trips_p95 = getClosestInterval(p95);
```

the *NYC Taxi Dataset* and the One-Step functions are based on state-of-the-art literature and research. The implementation language is *Python* and the functions are implemented in the *Openfaas*[3] *Docker* environment and can be built and pushed to a repository as common *Docker* images. This approach allows us to extend the workloads practically. The functions are provided and listed in their own repository at Git Hub.[4]

Openfaas function templates

The *Openfaas* templates[5] allow one to specify options for creating serverless-based functions in a *yaml* file. Different coding languages, like e.g. Python, Go, Java, or Dockerfile-based templates, can be chosen. The following example shows how a simple *Python3*-based function with the name *example* is specified.

```
version: 1.0
provider:
```

[3]https://www.openfaas.com/
[4]https://github.com/edgerun/galileo-experiments-functions
[5]https://github.com/openfaas/templates

```
  name: openfaas
  gateway: http://127.0.0.1:8080
functions:
  example:
    lang: python3
    handler: ./example
    image: registry/example:latest
```

Listing 5.1: Openfaas function template

The gateway and related image must also be set. The handler specifies the callable name of the function in the openfaas context. These templates are predefined, in the context of this thesis, the used templates are going to be modified, because of the distinct needs regarding different AI-based *Python* libraries and testbed architectures. The *Python* file, which handles the single request for the specific function, has the following structure, where only the single method *handle* must be specified:

```
def handle(req):
    return "HI, you entered: " + req
```

Listing 5.2: Python file template

5.4.1 One-Step Functions

Object Detection

This function allows deducing objects from a committed image file. It ranks the objects found by *score*, which defines object detection accuracy and assigns them to the correlating *category name*. This AI-based function rests on a pre-trained *TensorFlow Lite*[6] model, which gets loaded by the function to predict the objects in the image. In Figure 5.4.1 the abstract workflow is shown.

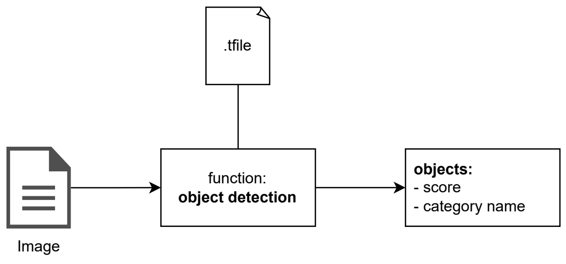


Figure 5.3: Object detection workflow

- **input format:** byte array

[6]https://www.tensorflow.org/lite/guide

- **function name:** *objectdetection*
- **model:** TensorFlow Lite (.tflite)
- **attributes:** accuracy critical & data intensive
- **output:**

```
model load time: float ,
pre-process time: float ,
prediction time: float ,
post-process time: float ,
results: [(float , string), ...]
```

Human Detection

The *Human Detection* function offers to determine if one or more persons are present in an image file. The model used is a pre-trained HOG (Histogram of Oriented Gradients) & linear SVM (Support Vector Machine) model, provided by the *OpenCV AI*[7] library. The function outputs, after prediction, whether humans were found in the image or not by returning a simple Boolean expression. Figure 5.4.1 shows the simplified workflow of this method.

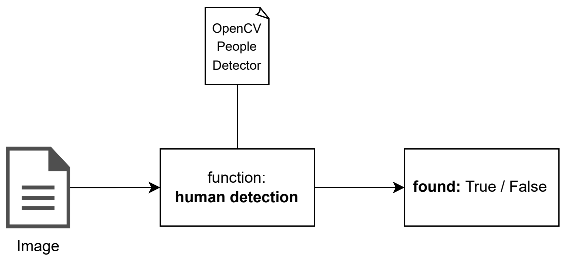


Figure 5.4: Human detection workflow

- **input format:** byte array
- **function name:** *humandetection*
- **model:** Standard OpenCV People Detector
- **attributes:** latency sensitive
- **output:**

[7]https://opencv.org/

```
model load time: float ,
pre−process time: float ,
prediction time: float ,
post−process time: float ,
found: boolean
```

Mask Detection

Like in Section 5.4.1 this function uses a pre-trained *TensorFlow* model to predict worn masks on faces from a committed image and returns the found mask types (homemade, surgical, n95, or bare). It also includes some pre-process tasks, like image resizing or converting into a floating model, which is needed for prediction.

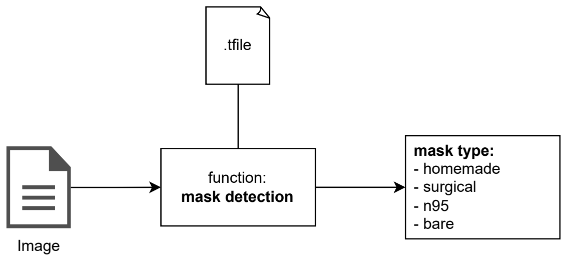


Figure 5.5: Mask detection workflow

- **input format:** byte array
- **function name:** *maskdetection*
- **model:** TensorFlow Lite (.tflite)
- **attributes:** latency sensitive & accuracy critical
- **output:**

```
model load time: float ,
pre−process time: float ,
prediction time: float ,
post−process time: float ,
masks: [string , ...]
```

Sleep Detection

When trying to determine whether a person is sleeping or awake, based on an image, this function calculates the ratio of eye and mouth closeness in a list of facial landmarks. If the ratios violet a distinct threshold, the likelihood of a sleeping person is given. This prediction method demands the *dlib*[8] ML algorithm library and uses a pre-trained shape prediction model for the detection of landmarks of the face. The function also takes advantage of the pre-existing *Frontal Face Detector* of *dlib*. The *Sleep Detection* includes some pre and post-processing steps such as image resizing, image converting, or calculation of eye and mouth closeness ratios, where also the *OpenCV AI* library is used.

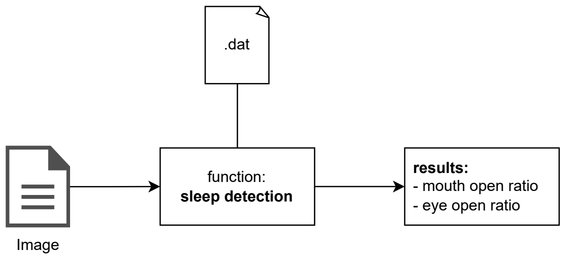


Figure 5.6: Sleep detection workflow

- **input format:** byte array
- **function name:** *sleepdetection*
- **model:** *dlib* shape prediction model (.dat),

[8]http://dlib.net/

- **attributes:** data-intensive, accuracy critical & latency sensitive
- **output:**

```
model load time: float,
pre-process time: float,
prediction time: float,
post-process time: float,
results: [(float, float), ...]
```

Pose Estimation

This function has the goal if a person is located in a committed image, to return the points regarding the pose and body parts pairs of the person. To do this, it utilizes the DNN from the *OpenCV AI* library and loads a pre-trained network model, which is provided in the ".caffemodel" format.

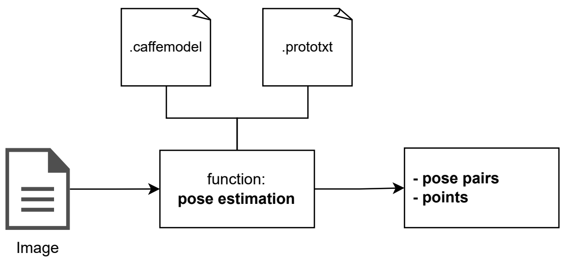


Figure 5.7: Pose estimation workflow

- **input format:** byte array
- **function name:** *poseestimation*
- **model:** *DNN OpenCV* network model (.caffemodel),
- **attributes:** data-intensive, accuracy critical
- **output:**

```
model load time: float,
pre-process time: float,
prediction time: float,
post-process time: float,
pose_pairs: [(float, float), ...],
points: [((float, float), ...]
```

Gun Detection

The function is created to detect guns in images. The algorithm is based on the implementation in [Gee]. It uses the already known *OpenCV* library and takes advantage of their Cascade Classifier, which loads a *xml* Haarcascade file with positive and negative examples. For this thesis, the predefined Haarcascade file from [Gee] is used, for simplicity. Returns True or False if a gun was detected or not.

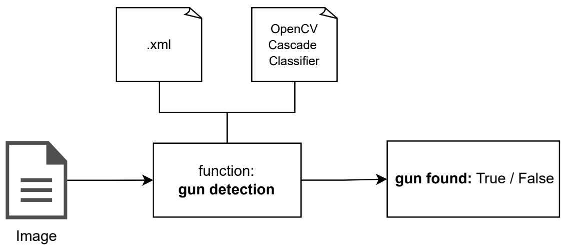


Figure 5.8: Gun detection workflow

- **input format:** byte array
- **function name:** *gundetection*
- **model:** *Haarcascade* file (.xml),
- **attributes:** accuracy critical & latency sensitive
- **output:**

```
model load time: float ,
pre-process time: float ,
prediction time: float ,
post-process time: float ,
gunExist: boolean ,
```

5.4.2 Inference Pipelines

Taxi Driver Safety App

The purpose of this inference pipeline is to assess whether a taxi driver is safe by recognizing people who want to get into a taxi and to detect if they are unarmed and wear a mask. This is done by looking for a human in a taken picture calling the human detection function presented in Section 5.4.1. If this proves to be true, the pipeline calls the functions *the gun detection* and *mask detection*, shown in Section 5.4.1 and Section 5.4.1, simultaneously. With the return values of these calls, the pipeline can assess if the driver is safe. It only has to check if the response of the gun detection request is *false* and the mask detection response does not include the mask type *bare*.

Since the two calls are concurrently made, the pipeline can return as soon as one of the responses leads to a negative result. In Figure 5.4.2 the workflow is shown schematically. To show how the implementation was performed, the following pseudo-code illustrates

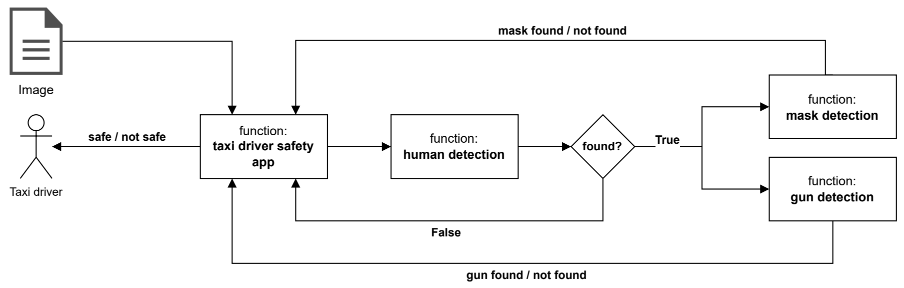


Figure 5.9: Taxi Driver Safety App workflow

the inference pipeline function, where the two function calls (mask & gun detection) are concurrently:

```
def handle(req):

    # 1. call human detector
    response_human = call_request(req, "humandetection")

    # 2. no human found
    if response_human == False:
        return "driver is safe"
    # 3. human found
    else:
        # 3.1 Call mask detection
        start call "mask detection":
            response_mask = call_request(req, "maskdetection")
            if "bare" in response_mask['masks']:
                return "driver is not safe"
                break;

        # 3.2 Call gun detection
        start call "gun detection":
            response_obj = call_request(req, "gundetection")
            if response_obj == True:
                return "driver is not safe";
                break;

    return "driver is safe"
```

Listing 5.3: *Taxi Driver Safety App* pseudo code

Algorithm 5.6: Generate pattern

```
topology = pd.readCSV(topologyFile);
trips = pd.readCSV(tripFile);
for row in trips do
    cell = getNearestCell(row, topology);
    row['cloudlet'] = cell['cloudlet'];
end
tripsTemp = [];
for row in trips do
    r = row['requests'];
    if $r > 1$ then
        for 1 ... r do
            row['requests'] = 1;
            tripsTemp.append(row);
        end
    else
        tripsTemp.append(row);
    end
end
cloudletDataframes = tripsTemp.groupBy('cloudlet');
for df in cloudletDataframes do
    pattern = [];
    sort(df, 'timestamp');
    lastTimestamp = 0;
    for row in df do
        if lastTimestamp == 0 then
            pattern.append(0.5);
        else
            diff = row['timestamp'] - lastTimestamp;
            if diff == 0 then
                pattern.append(0.1);
            else
                pattern.append(diff);
            end
        end
        lastTimestamp = row['timestamp'];
    end
    pattern.save(path + '/' + df['cloudlet'] + '.csv');
end
```

CHAPTER 6

Experiments

This chapter shows the setup of the experiment, the setup of the cluster machines, the testbed configurations used and the profiling experiments. Section 4.3.5 introduces the implementation of the scenarios.

6.1 Testbed setup

The testbed is split into three zones, zone A, B, and C, a master node that allows the starting of a *tmux* session, and a storage node that runs Redis, MYSQL, and Influx databases for saving the experiment results. Every single zone consists of a controller node and several different worker nodes. The zones are already shown in Figure 4.3.1. The controller nodes are also configured as worker nodes. Zones A and B represent two different IoT and cloudlet clusters in a smart city network environment. Zone C illustrates a cloud cluster and consists of four virtual machines. There are also two Intel NUC devices for zones A and B that emulate the user requests. For this thesis, the following devices, shown in Table 6.1, will be important and considered when executing the experiments:

Table 6.1: Testbed, table from [RRP+22]

Device	**Arch**	**CPU**	**Memory**	**Cluster**
1x AsRock	x86	8x Ryzen @ 2 GHz	33GB	IoT Box
1x Xeon	x86	4x Xeon @ 4.6GHz	16 GB	Cloudlet
4x VM	x86	4x vCPU @ 2Ghz	8 GB	Cloud
2x NUC	x86	4x i5 @ 2.2 GHz	16 GB	*Clients*

6.2 Edgerun galileo experiments framework

To start experiments on the testbed, the Galileo Experiments Edgebench repository, which relies on the Galileo Experiments repository [1] offers Kubernetes related deployment of applications and clients, telemetry data collection and controller handling. It uses the galileo shell to start the experiments. The experiments will get triggered remotely by using a *Tmux* session.

6.2.1 Tmux

Tmux is a Linux-based terminal multiplexer. It allows one to open terminal sessions, leave and reconnect to different sessions of applications running on a terminal window. Considering the experiments, *tmux* is useful to start them when the single experiments will get executed remotely on the testbed.

6.3 Profiling experiments

To get the resource usage and the execution time data of the single investigated functions on the different devices, profiling experiments have to be executed on the testbed. This is done, because the simulation in *faas-sim* requires the profiling data and the resource usage of the single functions and devices, such that the simulation and experiments can be compared with the same database. With the help of the *Edgerun Galileo Experiments* framework, the human detection, gun detection, mask detection, and object detection serverless functions presented in Section 5.4 will get profiled. The following parameters must be set when starting a profiling experiment:

- creator name
- hostname (i.e., *eb-a-controller*)
- container image of the application (i.e. *edgerun/maskdetection:1.1.0*)
- zone (i.e., *zone-a*)
- master node name (i.e. *eb-k3s-master*)
- picture URL
- number of pods on the host
- number of requests & the inter-arrival time
- number of clients

[1] https://github.com/edgerun/galileo-experiments

For this thesis the number of pods is set to 1, the number of requests to 100, and the inter-arrival time to two seconds. To get realistic and authentic resource usage and execution time data, a two-second pause between the single requests is necessary. After completion of the experiment, the mean execution time and the mean resource usage for each device will be calculated by using the *K3SGateway* implementation and *Jupyter* analysis notebooks. Also, the log-normal distribution of the execution time will be calculated for the later executed simulation runs. The concrete implementation of an profiling experiment (e.g. maskdetection) can be investigated in the *Galileo Experiments Extension* repository[2].

6.4 Scenario experiments

The scenario experiments are executed, to have comparable data regarding the simulation and to show how server and edge device placement is important for the infrastructure and resource planning of such Edge Computing settings. For the scenarios, the same repositories will be used, with some parameter changes. Also, the additional inference pipeline, shown in Section 5.4.2, is going to be executed as an experiment. As distinct from the profiling experiments, these experiments are going to use the request pattern generated in Section 5.3. Before the experiments can be started, the emulated network delay must be applied to the different network nodes. Section 4.3.2 shows the corresponding method. Each scenario will last about ten minutes. Because of the scope of this thesis, not every single one of the twelve scenarios, shown in Section 4.3.5, is going to be executed for each application/function, but every scenario will be covered by at least one of the applications. The *Galileo Experiments Extension* repository also allows to start of scenario experiments, therefore following parameters are needed for a single experiment run:

- creator name
- container image of the application (i.e. *edgerun/maskdetection:1.1.0*)
- number of zones (from 1 - 3)
- master node name (i.e. *eb-k3s-master*)
- path to request pattern files
- picture URL

When executing an experiment in one zone, the framework will start it only in *zone A*, when the zone number is set to 2, the framework will also execute it in *zone B*, and when the zone number is set to 3, all three clusters (IoT Box, Cloudlet, Cloud) are going to be used. This should emulate the concurrent execution of the gathered requests and provide

[2]https://github.com/edgerun/galileo-experiments-extensions/tree/pruellerpaul/experiments/

realistic results for later evaluation. In the scenario implementation[3] of the framework, the following mappings are used to indicate the correct zone, node, and image:

- application mapping (maps the unique name of an application to the function image, e.g. *maskdetection-zone-a* will get mapped to *edgerun/maskdetection:1.1.0*)
- Zone mapping (maps the node, e.g. *eb-a-controller*, to a distinct zone, e.g. *zone A*)
- Service mapping (maps the node, e.g. eb-b-controller, to a function and determines the number of available services, e.g. *maskdetection-zone-a : 1*)
- Profiling application mapping (maps the distinct application name to the concrete profiling implementation regarding the used framework)
- Arrival profile mapping (e.g. client of *zone A* uses a different arrival profile than the client for *zone B*)

A special case is the inference pipeline. This experiment needs four images in each zone, namely the main function (taxi driver safety app) which makes use of three other deployed functions (humandetection, maskdetection, gundetection). The workflow of the pipeline is shown in Figure 5.4.2. Therefore the mapping needs to consider more images.

[3]https://github.com/edgerun/galileo-experiments-extensions/tree/pruellerpaul/experiments/

CHAPTER 7

Evaluation

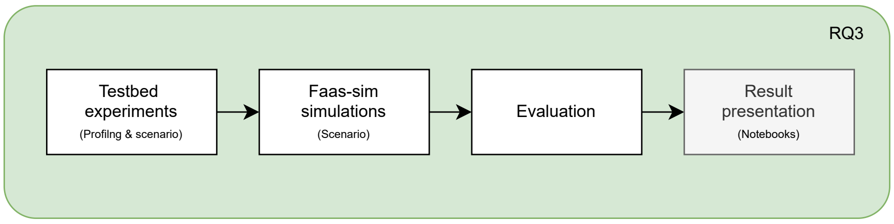


Figure 7.1: Approach towards RQ3

This chapter shows the gathered related work, describes the experiments and scenarios examined, and the results that could be gathered from it. It begins with the found simulation and emulation frameworks in Section 7.1.1 and Section 7.1.2, the serverless benchmark suites in Section 7.2, followed by the preparation and filtering of the cell database set in Section 7.3.1, topology creation, edge case extraction, and request pattern generation. After that, the results of the experiment and the simulation runs are inspected. At the end of this chapter, the metrics collected from the experiments will be analyzed and compared against the collected benchmarks. Section 4.1 introduces the method for this part.

7.1 Simulation & Emulation Frameworks

In this section, we examine frameworks by their configuration possibilities and features, if they are pure simulation or emulation tools or also have hybrid features for both approaches. The configurable parameters are listed and the buildup of the frameworks is shown. Afterward, a conclusion paragraph lines out the most important facts about the simulation and emulation tools.

7.1.1 Simulation Frameworks

Simulation Frameworks, in the context of cloud and Edge Computing, are predominantly open-source tools for repeatable evaluation of applications in large-scale network topologies. Because of the financial limits in common software development, setting up a real-world environment for testing applications is a well-known problem. These tools can offer multiple platform support, generate communication, energy, and cost models of the submitted application & network settings, and provide sometimes graphical user interfaces for end users. There is no need for an expensive IT infrastructure to work with such simulators [FKK17].

CloudSim

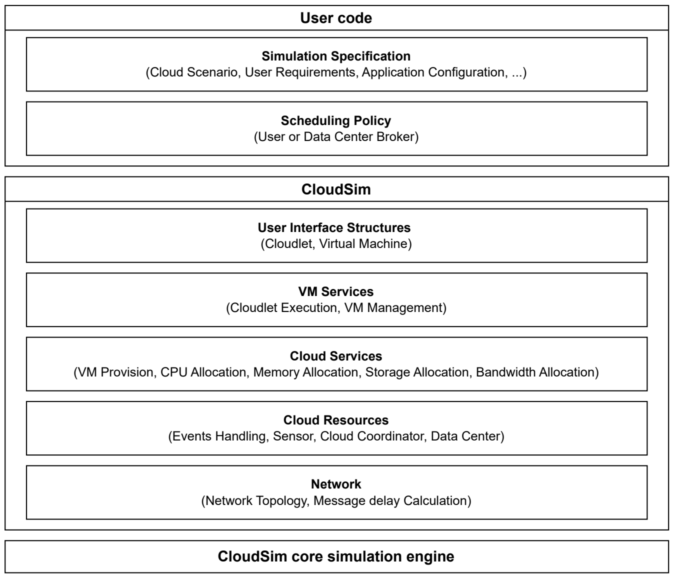

User code
Simulation Specification
(Cloud Scenario, User Requirements, Application Configuration, ...)
Scheduling Policy
(User or Data Center Broker)
CloudSim
User Interface Structures
(Cloudlet, Virtual Machine)
VM Services
(Cloudlet Execution, VM Management)
Cloud Services
(VM Provision, CPU Allocation, Memory Allocation, Storage Allocation, Bandwidth Allocation)
Cloud Resources
(Events Handling, Sensor, Cloud Coordinator, Data Center)
Network
(Network Topology, Message delay Calculation)
CloudSim core simulation engine

Figure 7.2: The CloudSim layer architecture (Adapted from [CRB+11])

CloudSim is an extensible, event-driven simulator for testing the performance of Cloud Computing related services. It enables the modeling of large-scale topologies with distinct network infrastructure, service brokers, virtual machines (VM), and other cloud components like data centers, and allows one to define allocation policies, all on a single machine. It allows one to model public, private, hybrid, or multi-cloud environments. Components maintain a message queue by sending messages to other network participants along the queue. The data centers are made up of storage servers and physical host machines, which host the so-called cloudlets. Cloudlets perform specific tasks from workloads that are assigned by a simulation [CRB+11] [FKK17]. Regarding virtualized services, it is possible to switch between space-shared and time-shared processing core allocation. This time effectiveness, applicability, and flexibility are the main advantages of using CloudSim to simulate application services in cloud environments [CRB+11]. In Figure 7.2 the CloudSim multi-layer architecture is displayed. The CloudSim simulation layer provides management interfaces for virtual machines and handles the following topics:

- Memory, storage, and bandwidth control
- Hosts to VMs provisioning
- Application execution management
- (Dynamic) system state monitoring

The User Code layer allows a developer to generate workload request allocations, custom application provisioning methods, cloud availability scenarios, and perform tests regarding system robustness based on the following extendable basic entities [CRB$^+$11]: *Number of machines*; *Number of tasks*; *Virtual machines*; *Number of users*; *Application types*; and *Scheduling policies*.

iFogSim

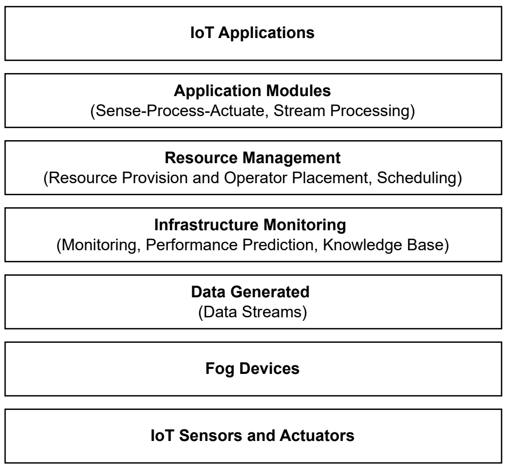

IoT Applications
Application Modules
(Sense-Process-Actuate, Stream Processing)
Resource Management
(Resource Provision and Operator Placement, Scheduling)
Infrastructure Monitoring
(Monitoring, Performance Prediction, Knowledge Base)
Data Generated
(Data Streams)
Fog Devices
IoT Sensors and Actuators

Figure 7.3: The iFogSim layer architecture (Adapted from [GVGB17])

This open-source, Java-based framework enables the simulation of edge and fog networks and provides application scheduling policies through edge and cloud resources. It also offers a resource management tool and extends on the CloudSim simulator presented in Section 7.1.1. IFogSim offers a graphical user interface (GUI) where physical topologies can be created and exported in JSON format. It is also possible to create the topologies in JAVA programmatically [PVCM20].

In iFogSim, the architecture consists of seven different layers shown in Figure 7.3. The first layer, at the bottom of this architecture, contains the Internet of Things (IoT) devices, such as sensors, cameras, etc. which generate data and react to changes in the environment. In the next layer, so-called fog devices, such as cloud resources, gateways, or application modules, are hierarchically created. They can only communicate between parent and child devices, so device-to-device communication is not possible, and only tree topologies are supported. Data streams are sequences of value tuples that are issued by e.g. sensor devices, application modules, or fog devices. Furthermore, these data streams can be used by the next layer, the infrastructure & monitor layer which monitors the resources, power usage, sensors, and other fog devices. The resource management layer makes use of this information. The main task of resource management is, with the help of scheduler and placement components, to minimize resource consumption by listening to the information of the previous layer and distributing resources with respect to the desired application modules. Generally, it is possible to change the implementation of resource management and have a distributed, hybrid, or static way of resource allocation. The distributed data flow (DDF) model is the basement for developing applications that are deployed in the fog. DDF means that the data flow is deployed to multiple devices rather than one device. Therefore, the data input and output flow can be drawn as a graph between modules [GVGB17].

The following table shows the components of the iFogSim framework [GVGB17] and their main attributes that the user can define:

Fog Devices	- Accessible memory, processor, storage size, uplink, and downlink bandwidths - Custom policies: changeable methods to handle resource scheduling and modules deployment
Sensors	- Connected gateway reference, gateway connection latency, output characteristics of the sensor - Tuple inter-transmissions distribution (tuple arrival rate)
Actuators	- Connected gateway reference, gateway connection latency - Changeable actuator methods
Tuples	- Type, source and destination application modules - Processing requirements in million instructions (MI), length of data
Application	- Number of output tuples per input tuple, periodic and event-based application edges - Process-control loops to measure end-to-end latency
Monitoring service	- Resource usage statistics for each device - Power consumption at given CPU use
Resource management service	- Changeable placement policy - Changeable application scheduling policy - Default scheduling policy: uniformly distributed

Table 7.1: Components of the iFogSim framework

iFogSim2

This framework is an extension of *iFogSim* presented in Section 7.1.1. It extends the existing core functionality with three new components, namely *Mobility*, *Microservices*, and *Clustering*. With the Mobility component, it is possible to choose between different mobility models, e.g. the random, or the directional pattern. Different parameters such as location, speed, stop time, communication range, etc. are set in these models to generate the movement data of the IoT devices. The *Microservices* component adds the possibility to simulate microservice architecture orchestration, in contrast to the standard monolithic approach. The new *Clustering* component enables the coordination and communication among distributed nodes in the simulation network [MPGB21].

EdgeCloudSim

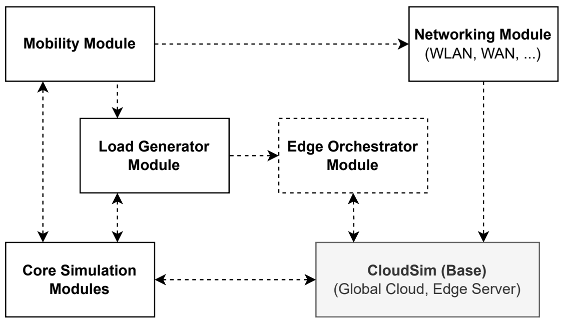


Figure 7.4: EdgeCloudSim modules (Adapted from [SOE17])

These CloudSim extensions enable the user to specify designated computational and networking resources before running Edge Computing scenarios. It consists of five expandable modules, shown in Figure 7.4. EdgeCloudSim[1] offers additional configurable parameters. It allows us to define the places where mobile devices are simulated with respect to a distinct mobility motion model, change the WAN and WLAN parameters, or enter the number of edge servers per place [SOE17]. The complete list is shown below.

Conclusion

CloudSim is a free, powerful, and detailed documented tool to create models and simulations of large-scale Cloud Computing data centers or virtualized server hosts. It is the base layer of many other simulation frameworks. With its great customizability and user-defined policies, it is very flexible and can be adapted to the developer's needs. The *CloudSim* based frameworks *iFogSim* and *iFogSim2* extend the functionality of *CloudSim* by a graphical interface and many components like a monitoring service, resource management service, clustering, a mobility component and customizable fog devices and sensors. The GUI enables a user-friendly way to use such a framework and create e.g. topologies for the simulation. Also the *EdgeCloudSim* extension offers the developer new opportunities, like configurable WAN and WLAN parameters or the already mentioned mobility motion model. These extensions of *CloudSim* can be seen as an advancement, if further special needs are recommended when planning simulations.

[1]https://github.com/CagataySonmez/EdgeCloudSim

Parameters
Poisson Interarrival Time of Tasks (second)
Simulation Time (hours)
Number of repetitions
User Mobility Model Nomadic M. M.
Number of Mobile Devices
Number of Place Type (Attractiveness Level)
Probability of selecting a place type equal
Number of places
Dwell time of place (minute)
Active/Idle period of the user (second)
Number of Edge Server per Place
Number of VMs per Edge Server/Cloud
CPU Utilization
VM Processor Speed (MIPS) per Edge Server/Cloud
Probability of Offloading to Cloud
Average Data Size for Upload/Download (KB)
Average Task Size (MI)
WAN/WLAN Bandwidth (Mbps)
WAN Propagation Delay (ms)

Table 7.2: EdgeCloudSim Parameters (Adapted from [SOE17]

7.1.2 Emulation Frameworks

Emulation tools allow one to run assessable and repeatable experiments with respect to real-world conditions and benefit the detection of e.g. bottlenecks before production deployment. These frameworks are able to emulate real code, in contrast to simulation tools, which only consider assumptions and set application parameters. Simulation tools and their configuration possibilities, such as the one presented in Section 7.1.1, are not always representative of real-world applications, so we also have to keep in mind emulation frameworks to test Edge Computing applications.

EmuFog

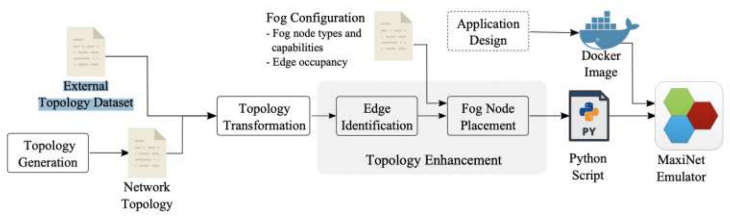


Figure 7.5: EmuFog emulation workflow (Adapted from [MGG+17])

EmuFog[2] is an open-source and extensible emulation framework that enables the design of fog computing infrastructures and the emulation of docker-based real-world applications. End users can choose the topology that is consistent with the required application case. Implement scalability for large-scale topologies by allowing the creation of topologies with network topology generators or by loading real-world datasets with a single file [MGG+17]. The user can configure the following parameters before starting the emulation. The

Parameters
Topology generation or external topology datasets
Fog and device node types
Maximum connections (fog nodes)
Costs, memory limit, and CPU share (fog nodes)
Scaling factor, average device count (device nodes)
Memory limit and CPU share (device nodes)
Maximal number of fog nodes
The cost function's threshold (like latency)
Host device latency and bandwidth
Computational capabilities (high level specification)
Expected client numbers

Table 7.3: EmuFog Parameters

defined settings are stored in a single configuration file. EmuFog allows one to export the defined software to execute it in the MaxiNet[3] emulator. This is a distributed network emulator based on the MiniNet[4] emulator. MaxiNet adds docker-based virtual hosts to the basic implementation of MiniNet, which allows one to run the emulation only on a single physical machine. This provides a more realistic approach than the above-described simulation attempt. It is also cheaper, more sustainable, and more efficient than real deployments.

7.1.3 Hybrid Frameworks

EMUSIM

EMUSIM is a framework that uses both techniques, namely simulation, and emulation, to investigate the application's behavior and software-as-a-service cloud deployment scenarios. It enables the user to generate a more precise deployment model for the service and allows one to measure costs and performance in the cloud environment. EMUSIM aggregates a configuration for deployment in production infrastructure. It applies the emulator on a local environment to extract the application profile, which is one of the inputs of the CloudSim-based simulator, and to gather the necessary external

[2]https://github.com/emufog/emufog
[3]https://maxinet.github.io/
[4]https://github.com/mininet/mininet

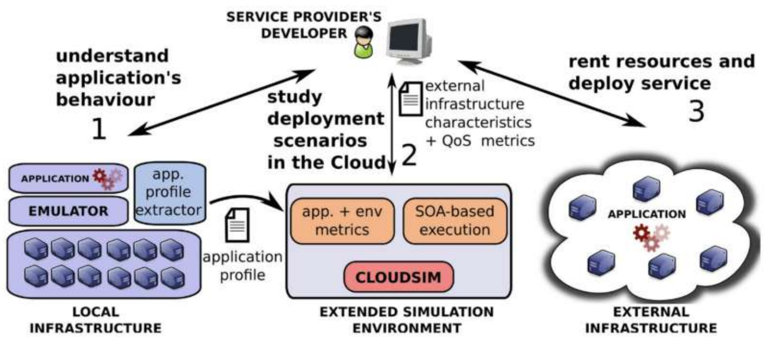

SERVICE PROVIDER'S DEVELOPER
understand application's behaviour
1
study deployment scenarios in the Cloud
2
external infrastructure characteristics + QoS metrics
rent resources and deploy service
3
APPLICATION
EMULATOR
app. profile extractor
application profile
app. + env metrics
SOA-based execution
CLOUDSIM
APPLICATION
LOCAL INFRASTRUCTURE
EXTENDED SIMULATION ENVIRONMENT
EXTERNAL INFRASTRUCTURE

Figure 7.6: EMUSIM organization overview (Adapted from [CNRB13])

infrastructure characteristics and QoS (Quality of Service) metrics. The framework needs the following four configuration files to run emulation and simulation of the application [CNRB13]:

1. Physical environment configuration (XML file)
2. Emulation environment configuration (minimum & maximum number of VMs)
3. Application configuration
4. Simulation configuration (e.g. number of users, request arrival pattern, data center capacity, number of virtual machines, and policies for provisioning)

EmuEdge

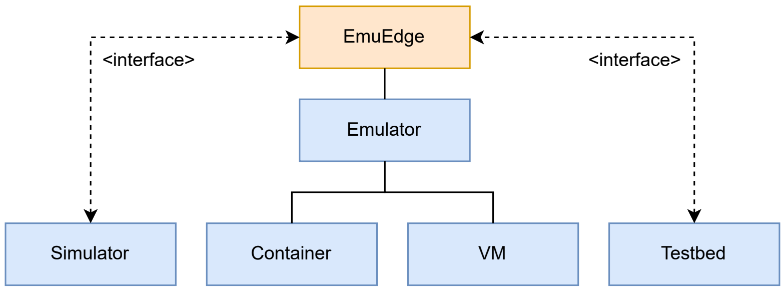


Figure 7.7: EmuEdge structure (Adapted from [ZCS19])

EmuEdge [ZCS19] is an open-source hybrid emulator that enables interfaces to connect to simulation frameworks and real-world testbeds. It is built on an emulator that can run applications in similar Mininet containers or enables full system virtualization in VMs and allows the physical deployment nodes to be connected to virtual hosts. The coarse structure is shown in Figure 7.7. EmuEdge offers its own reproduction framework, which allows setting the following inputs:

- Emulation parameters (CPU core, memory, or disk allocation)
- Network topologies (routers, switches, devices, physical devices, network links)
- Network traces (gathered from experiment logs)
- Synthetic traces (generated from simulations)

Conclusion

EmuFog is a pure emulation framework, that allows the generation and import of topologies and configures multiple parameters like scaling factors, node types, computational capabilities, and many more. As distinct from the two hybrid emulation frameworks *EMUSIM* and *EmuEdge*, it does not have features for deploying or exporting configurations for productive infrastructure, but can be connected to a simulation framework via its own interface.

Framework	Features	Simulation only?	Emulation only?	Parameters
CloudSim	Public, private, hybrid, or multi-cloud environment models; Space-shared and time-shared processing core allocation; Hosts to VMs provisioning; Application execution management; System state monitoring	Yes	No	Number of machines; Number of tasks; Number of users; Application types; Scheduling policies; Virtual machines
IFogSim	All features of CloudSim; Simulation of edge and fog networks; Resource management, GUI for topology creation; Fog device customization;	Yes	No	All parameters of CloudSim; Accessible memory, processor, storage size, uplink, and downlink bandwidths; Custom policies (scheduling, deployment); More parameters are listed in Table 7.1
IFogSim2	All features of IFogSim; Mobility models; Microservice orchestration; Clustering	Yes	No	Location, speed, stop time, communication range of mobility models
EdgeCloudSim	All features of CloudSim; Mobility motion model; Edge Orchestration;	Yes	No	WAN & WLAN parameters; Mobile device places; Number of places; Number of Edge Server per Place; More parameters are listed in Table 7.2
EmuFog	Docker-based application emulation; Network topology generator or real-world data import; Export for MaxiNet emulation	No	Yes	Fog and device node types; Fog node maximum connections; Memory limit and CPU share; Expected client numbers; More parameters are listed in Table 7.3
EMUSIM	Generate precise deployment models; Cost and performance measurement; Configuration aggregation for production deployment; CloudSim simulation input; Gather infrastructure characteristics and QoS metrics	No	No	Physical environment configuration; Minimum & maximum number of VMs; Number of users; Request arrival pattern; Data center capacity; Provisioning policies; Application configuration
EmuEdge	Interface for simulation; Interface for testbed; MiniNet emulator; Full system virtualization	No	No	CPU core; Memory; Disk allocation; Network topologies; Network traces; Synthetic traces

Table 7.4: Summary of the investigated emulation and simulation frameworks

In the next section, seven different benchmarking suites and tools are summarized. The workloads, applications and benchmarks of each tool are consolidated. In the end, a small conclusion lines out the features and hard facts about the single tool.

7.2 Serverless Benchmarking Suites

Seven benchmarking tools were examined. For example, FunctionBench, which applies in the area of Microbenchmarks, Data Processing, Applications, and ML Model Training & Serving, offers a widespread set of serverless FaaS workloads to measure the latency in cloud and edge environments. EdgeBench is a cloud and edge benchmarking suite for workloads in the area of Smart Home & Autonomous Vehicles, that enables end-to-end latency, bandwidth utilization, local resource utilization, and infrastructure costs benchmarking. The IoTBench, DeepEdgeBench, and EdgeDroid suites have to be emphasized because they are suites that only work in edge environments, while the iBench and FaaSdom suites are only for use in cloud benchmarking.

7.2.1 FunctionBench

FunctionBench is a suite of serverless FaaS workloads that can be deployed on common cloud services, namely AWS Lambda Google[5], Cloud Functions[6] and Azure Functions[7]. It offers micro benchmarking and application workloads which will be lined out precisely in this section. Micro benchmarks allow one to evaluate different resource usage self-contained. Such workloads are not common in modern FaaS applications, because real-world applications or functions require multiple concurrent resources for executing the program code. This is why FunctionBench extends the benchmarks with different applications, ML Model Training, and ML Model Serving workloads to evaluate more concurrent resource-addicted workloads.

Workloads & Benchmarks

First of all, the workloads in FunctionBench have four different characteristics; they can be distinguished into *CPU*, *memory*, *Disk I/O*, and *network* required loads.

Micro benchmarks:
The suite offers **float point operations**, such as square root or sine and cosine, linear equation (**Linpack**), and **matrix multiplication** computation. These are mainly CPU and memory-intensive workloads. It also allows performing a disk I/O-based workload, namely a function where the **dd system command** is executed. To measure network performance, FunctionBench has a **cloud storage** workload in which an object is down and uploaded. The **iperf3** workload is also a network-intensive workload where a direct connection between sender and receiver is required.

Applications:
FunctionBench has an application workload that executes an **Image Processing** computation. It has medium CPU and memory usage for image processing and low disk I/O

[5]https://aws.amazon.com/de/lambda/
[6]https://cloud.google.com/functions/
[7]https://azure.microsoft.com/en-us/services/functions/

and network usage for down- and uploading the image. Additionally, the suite offers a **Video Processing** workload. This load has high CPU and memory usage and a medium disk I/O and network overhead.

ML Model Training & Serving:
To evaluate ML workloads, the suite offers a preprocessing **featurization** workload to prepare raw data for the next ML step. It executes TF-IDF vector transformation on a text dataset. The **logistic regression** workload then builds a model based on the featured data. These two loads require high memory, CPU, and network resources due to the large size of the datasets that must be accessed. For the ML serving phase, FunctionBench offers a **face detection**, a logistic regression, a deep learning-based **image classification**, and a **word generation** inference workload. All these loads do not require much resource usage like the training workloads.

Benchmarks:
FunctionBench offers the investigation of **latency** after performing the workloads. The main goal of this suite is to evaluate the performance of the different functions of different service providers.

Conclusion

The corresponding paper [KL19] does not show how they set up the request pattern and user behavior. Also, the orchestration performance is not examined in FunctionBench, but it offers a great balanced selection of workloads to build on.

7.2.2 EdgeBench

EdgeBench offers a suite of different benchmark applications to measure the performance of common cloud providers. Like in FunctionBench, presented in Section 7.2.1, the EdgeBench suite also offers different application types to benchmark serverless platforms. The difference here is that it processes the input data of the application on an edge device and then sends it to two selectable cloud providers (AWS Greengrass[8] and Microsoft Azure IoT Edge[9]) which is nearer on the thread this thesis wants to examine [DPW18].

Workloads & Benchmarks

The suite allows one to use three different types of application, namely an **image recognizing** application, a **speech to text** application, and a **scalar value generator** that emulates a sensor and is used to measure performance with a lightweight workload when the resources on the edge are sparse. The speech-to-text area is relevant since Smart Home devices like Amazon Echo or Google Home are readily available for all households and users all over the world. Image processing is used in areas such as smart cameras or autonomous vehicles, to name only two examples [DPW18].

Benchmarks:
The developers of this suite implemented two pipelines to benchmark the three application types in combination with the selected cloud provider. They offer a cloud-only pipeline to measure the performance of the cloud provider like in FunctionBench and an edge pipeline to measure the edge-based performance benchmarks. The Python-based suite generates metrics like Compute time, Time-in-flight, Payload size, CPU and memory utilization, and end-to-end latency. In [DPW18] the contributors set up an experiment with a *Raspberry Pi 3B* as an edge device and investigated **end-to-end latency**, **bandwidth utilization** and **local resource utilization**. Subsequently, they presented **Infrastructure Costs** and compared the two pipelines according to all the benchmarks.

Conclusion

This is a good approach to manually investigate the providers, but, like FunctionBench, it is limited to measuring orchestration performance and managing user behavior. It offers a good selection of metrics and benchmarks that can be used for a more precise examination.

[8]https://aws.amazon.com/de/greengrass/
[9]https://azure.microsoft.com/de-de/services/iot-edge/

7.2.3 DeepEdgeBench

In [BJCG21] the contributors investigated the model inference performance on five different edge devices regarding four AI-based Deep Neural Network frameworks (Tensorflow, TensorRT, Tensor-flow Lite, and RKNN-Toolkit). Some of the devices conclude an Artificial Intelligence (AI) unit for which performance will also be examined.

Application & workload:
DeepEdgeBench evaluates the performance through **image classification** workloads by specifying the DNN model and the number of images used from the *ImageNet* dataset.

Benchmarks:
They developed a method to measure **power consumption**, the time required for inference (**inference speed**) and **model accuracy**. For model accuracy, the evaluation investigates the claimed accuracy and compares it with the measured accuracy of the model by looking at **TOP-5** and **TOP-1 accuracy**. They examined different models like MobileNetV2, MobileNetV2 Lite, MobileNetV2 Quant. Lite, or MobileNetV1 Quant. Lite.

Conclusion

The focus in this setup is on comparing hardware devices and not orchestration techniques. The paper shows how to benchmark different edge hardware devices in a comparable way. The investigation of the model accuracy by excluding and including AI units is a very insightful evaluation part of DeepEdgeBench. Also, the way in which the power consumption is done is a good input for further work.

7.2.4 FaaSdom

FaaSdom is an open-source automated test suite to evaluate the performance of different providers of serverless computing platforms (Amazon Web Services, Microsoft Azure, Google Cloud, and IBM Cloud) and programming languages. The suite allows the deployment, execution, and clean-up of associated tests in an automated way and also offers continuous monitoring of the benchmark tests.

Applications:
The suite offers several HTTP trigger-based functions: A **CPU bounded** function that does **integer factorization** and **matrices multiplications**; a **network bounded** function that allows one to measure the round-trip time of geographically distributed deployments; a **IO Disk bounded** function to measure **disk read and write performance**; and the possibility of defining a custom function [MFKS20].

Workloads:

The workloads are invoked by the **wrk2**[10] framework, which allows a constant throughput load to be injected through the HTTP trigger. This tool also shows the average latency of the requests.

Benchmarks:
In [MFKS20] the **call latency** (round trip), the **cold start latency**, the **execution time** regarding different memory setups, and the **successful requests per second** (throughput) by handling CPU intensive workloads. In addition, **active instances** are measured during the load test. The tool offers a **pricing calculation** to compare each platform and configuration.

Conclusion

The suite offers a user interface to work with. It allows one to set up their own functions by providing their own function template and [MFKS20] shows how to provide a pricing calculation that gives the developer a good opportunity to estimate the future costs of serverless functions on different platform providers. FaasDom allows one to investigate the cold start latency and round-trip time, which are also indicators of how good scheduling and auto-scaling are done on the different platforms.

7.2.5 EdgeDroid

This tool is developed to benchmark human-in-the-loop applications in the context of Edge Computing. Human-in-the-loop applications are, e.g. wearable cognitive assistance (WCA) or augmented reality (AR) mobile applications that have a tight affinity with the end users themselves. EdgeDroid applies recorded user interaction traces and a so-called *user model* to imitate user feedback in the subsequent benchmarking process. Python-based *control backend* and application instances are deployed in a cloudlet, client emulators, and *user model* run on one or more Android devices. A detailed description of the EdgeDroid approach is shown in [OMnWSG19].

Applications:
The main application types considered by the contributors in [OMnWSG19] are **human-in-the-loop apps**. They got pulled and deployed in the cloudlet by a simple Docker image and are communicating over TCP with the clients. In the associated paper, the contributors set up an experiment that looks at a **WCA** based application called *gabriel-lego*[11].

Workloads:
As described above, the workloads are generated by the **recorded traces** and the evolved *user model* which are able to consider, for example, fatigue, annoyance, and other user reactions to the feedback of the application. The experiment shown in [OMnWSG19]

[10]https://github.com/giltene/wrk2
[11]https://github.com/cmusatyalab/gabriel-lego

also considered **single** and **multiple users**.

Benchmarks:
It includes several **latency** benchmarks, namely **processing time**, **uplink** and **downlink transmission**. The comparison of the metrics shows the distribution between the components and allows one to make application and architecture decisions based on it. They also compared the impact of well-connected devices and clients in an impaired WiFi environment and looked at **RTT** regarding **input-feedback cycles** that allows one to analyze the latency of different steps in the application.

Conclusion

EdgeDroid is a practical tool for developing solutions to prevent bottlenecks that appear after the use of AR and WCA applications. With this method, developers can decide whether they need to improve scalability, WiFi connection, or processing power in the back-end itself and allow one to determine whether the weakness is affected by software or hardware [OMnWSG19].

7.2.6 iBench

IBench is a benchmark suite for determining the inference performance of distributed Edge Computing systems at the system level and also for measuring the performance of the AI accelerators used. It consists of two main components, namely a data simulator engine and an inference server. IBench allows measuring more components of a High-Performance Computing (HPC) system than a chip-level-based performance measuring like in MLPerf[12] [BBS$^+$20].

Application & workloads:
The focus lies on benchmarking **AI based inference** server on HPC systems. Workloads are generated in the *Source module* of iBench. This is a simulation engine that generates **image** and **document data** and allows one to specify **data transfer rates** (Velocity, Volume and Variety).

Benchmarks:
In iBench the *Post-processing module* stores the results and allows one to generate visualizations and offers a search function for filtering results. The main benchmarks are **throughput**, **latency**, **ingest rate / bandwidth**, **pre-processing time**, and **GPU efficiency**. Compared to other suites for ML benchmarking, iBench offers a more extensive selection of metrics [BBS$^+$20].

[12]https://www.nvidia.com/en-us/data-center/resources/mlperf-benchmarks/

Conclusion

The suite allows one to have a more precise insight into the performance of distributed HPC architectures. By looking at the additional metrics, the developer has the opportunity to take the appropriate action, e.g., add additional GPUs or improve network speed and CPU performance.

7.2.7 IoTBench

This edge processing benchmark suite covers a selection of Internet of Things (IoT) related applications. Due to the limited resources in IoT devices, this suite should enable more precise architecture decisions for IoT device platforms. The contributors of IoTBench show in [LLYC19] a benchmark evaluation on a Raspberry Pi3 device.

Applications:
The suite offers three different application areas. **Computer Vision** offers technologies such as self-driving cars or video surveillance, **Speech Recognition** is widely spread in smart home devices and **Physiological Signal Processing** applications can run on wearable personal health devices.

Workloads:
IoTBench includes imposing workloads in all of the three areas mentioned above. The workload domains in the Vision area are **Video Summarization**, **Stereo Image Matching**, **Image Recognition**, and **Scan Matching**. In the area of *Speech Recognition*, it offers a **Voice Feature Extraction** workload and **Signals Enhancement**. For *Physiological Signal Processing*, IoTBench allows one to run a **Data Compression** workload. Workloads are executed by the user.

Benchmarks:
It offers benchmark metrics regarding computational demand, efficiency, and energy consumption. For measuring computational demand, metrics like **frames per second** or **execution time** for inference deep learning in *Computer Vision* workloads. **Processing rate** is used to measure the demand in *Speech Recognition* and *Physiological Signal Processing*. The evaluation of performance efficiency in [LLYC19] shows an **instruction breakdown** of difference characteristics such as **CPU instructions**, **cycles**, **cache**, **memory**, **floating point**, **branch** or **SIMD instructions** in different workloads. Also, the **Cache sensitivity** is lined out. The power and energy breakdown is separated into **computation**, **on-chip memory**, **off-chip memory** and **storage consumption**.

Conclusion

IoTBench is a compact and balanced tool to investigate the performance of the IoT device and energy consumption by offering different areas of common applications and workloads. It allows one to make a decision regarding the hardware design architecture in

IoT edge devices. [LLYC19] shows a practical example of how to benchmark IoT devices on a Raspberry Pi3, which is widely used in an Edge Computing environment. It is only used for edge processing (front-end processing) and does not include the opportunity to test computational offloading.

In Table 7.5, all benchmarking suites/tools previously investigated are listed. The used metrics, application areas & workloads are shown. Also, the Cloud and Edge compatibility is present.

Benchmark Suite	**Areas**	**Workloads**	**Cloud only?**	**Edge Only?**	**Metrics**
FunctionBench	Microbenchmarks, Data Processing Applications, ML Model Training & Serving	Float point, Matrix multiplication, Dd system, Cloud storage, Iperf3, Image processing, Video processing, Featurization, Logistic regression, Face detection, Image classification, Word generation	No	No	Latency
EdgeBench	Smart Home & Autonomous Vehicles	Image recognizing, Speech to text, Scalar value generator	No	No	End-to-end latency, Bandwidth utilization, Local resource utilization, Infrastructure costs
DeepEdgeBench	ML Model Inference	Image classifiction	No	Yes	Power consumption, Inference speed, Model accuracy, TOP-5 and TOP-1 accuracy
FaaSdom	HTTP trigger-based functions	Wrk2 framework: Integer factorization, Matrix multiplication, Network bounded function, Disk I/O	Yes	No	Call latency, Cold start latency, Execution time, Successful Requests per second, Active instances, Pricing calculation
EdgeDroid	Human-in-the-Loop Applications	Gabriel-lego	No	Yes	Latency, Processing time, Uplink & downlink transmission, RTT, Input-feedback cycles
iBench	ML Model Inference	Allow to specify: Image and document data, Data transfer rates	Yes	No	Throughput, Latency, Ingest rate/bandwidth, Preprocessing time, GPU efficiency
IoTBench	Computer Vision, Speech Recognition, Physiological Signal Processing	Video summarizing, Stereo image matching, Image recognition, Scan matching, Voice feature extraction, Signals enhancement, Data compression	No	Yes	Execution time, Processing rate, CPU instructions, Cycles, Cache, Memory, Floating point, Branch, SIMD, Cache sensitivity, On-chip & off-chip memory, Storage consumption

Table 7.5: Summary of the investigated benchmarking tools

Conclusion

Towards **RQ1** in Section 1.2, it can be concluded, that there are many multifaceted and adaptable benchmarking tools, that cover application areas like Microbenchmarks, Data Processing applications, ML Model Training, Serving & Inference, Smart Home & Autonomous Vehicles, Human-in-the-Loop applications, Computer Vision, speech recognition and physiological signal processing. The examined tools offer metrics like latency, bandwidth & local resource utilization, infrastructure costs, power consumption, inference speed, model accuracy, and many more in greater detail, e.g. Input-feedback cycles, Cache sensitivity, or On-chip & off-chip memory. Some frameworks presented in Section 7.1 allow custom workload & topology generation, generate application profiles and enable the creation of mobility models, which is a notable baseline to perform VR and AI-based benchmarking. Some frameworks offer monitoring of e.g. end-to-end latency and allow defining and changing application types. Furthermore, the hybrid frameworks are a huge opportunity to combine the emulation metrics with the one of the simulation.

7.3 Workload Definition

This section shows the evaluation of the workload definition phase, which includes the cell extraction and request pattern generation results.

7.3.1 Cells extraction

The original *OpenCelliD* dataset file has a size of 3.95 GB[13] and will be prepared with the help of the script provided in the *faas-topologies*[14] repository. Before starting the following command, the original file is placed in the *datacells_data* folder.

```
python prepare_dataset.py --radio LTE
```

After preparation, the new file, saved as *datacells_data/cell_towers_prepared.csv*, now has a size of around 720 MB. From this point on, it is easier to work with the cell data file. The next step is the filtering. With the following command, the dataset will be filtered against the latitude and longitude values 40.754380 and 73.984986, which are in the center of New York City, and the width and height of 1 *km* respectively 2 *km*.

```
python filter_dataset.py --name new_york_1x1
--lat 40.754380 --lon -73.984986 --width 1 --height 1
```

The script saves the files with the given *name* parameter in the *topologies* folder. The *new_york_1x1* has a size of 109 KB and includes 2839 cells. The *new_york_2x2* is 320 KB in size and includes 8313 cells. Now, the cells are ready to be allocated with the different cloudlet scenarios mentioned in Section 4.3.6. With the following command, the *create_cloudlet_membership.py* script creates a *CSV* file with the new cloudlet membership entries.

```
python create_cloudlet_membership.py
--path topologies/new_york_1x1.csv  --name new_york_1x1_1x1
--width 1 --height 1
```

Now, every cell in the files has a cloudlet number assigned. Figure 7.8 shows the cloudlet assignment, where the x-axis represents the longitude and the y-axis the latitude. The value points in the figures represent the cells in the area. The different colors show

[13] date of download: 30.12.2022
[14] insert url

the cloudlet belonging. The Figure 7.8b consists of 2839 cells, which is the size of the total number of cells in the file. In Figure 7.8a, the area was divided into four cloudlet areas, where the single cloudlet areas have 807, 825, 525, and 682 cells. When analyzing Figure 7.8c, it shows that the cloudlet areas are split into 2319, 1900, 1867, and 2227 cells. The largest amount of cells in a single cloudlet area is to be located in Figure 7.8d, where the first area includes 4219 and the second 4094 cells.

7.3.2 Request pattern generation

When the topology and cloudlet data are created. Algorithm 5.5 describes the script, which allows one to extract the three edge cases from the trip data. In this evaluation, the minimum, average, and maximum ten-minute intervals of the $1km \times 1km$ and $2km \times 2km$ areas were calculated by looking at a time range of three months (May, June, July). With the following command, the Python script starts searching for the edge cases.

```
python get_edge_cases.py --name new_york_2x2
--trips data/trips/new_york_2x2/trips.csv
```

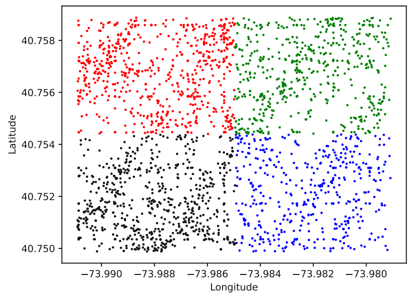
After completion, the intervals, illustrated in Figure 7.9, were created and show that the smaller area has only twenty events occurring in ten minutes as the minimum interval, 164 events as the average, and 265 for the maximum. For the bigger area, the minimum number (5th percentile) of events in a ten-minute interval is 119, for an average of 710 and the maximum is 1128. In Table 7.6 an overall overview of the total number of trip events in the different intervals and areas is shown. Because of the used dataset, the intervals shown in Figure 7.9 have accumulations on the exact minutes between the interval start and end. That goes back up to the measuring method of the *NYC Taxi Dataset* and should be resolved in future work by using more detailed datasets.

With these interval trip data, the baseline is set for the request pattern creation. Before starting the experiments, the patterns have to be generated using the script described in Algorithm 5.6. With the following command, the patterns for every cloudlet area are generated.

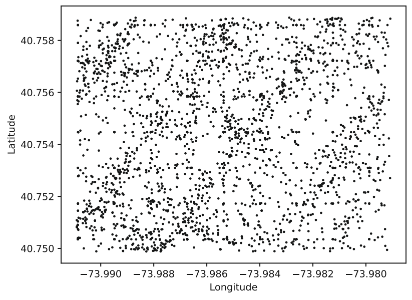
```
python generate_request_pattern.py --name new_york_1x1_1x1_min
--topology data/topologies/new_york_1x1_1x1.csv
--trips data/trips/new_york_1x1/minInterval.csv
```

area	minimum	average	maximum
$1km \times 1km$	20	164	265
$2km \times 2km$	119	710	1128

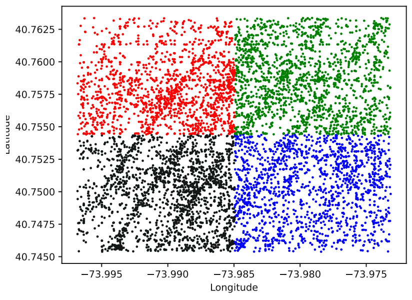
Table 7.6: Total number of trip events in all extracted intervals.



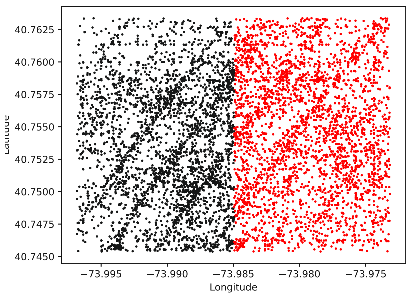
(a) New York 1 km × 1 km with a cloudlet area size of 0.5 km × 0.5 km



(b) New York 1 km × 1 km with a cloudlet area size of 1 km × 1 km

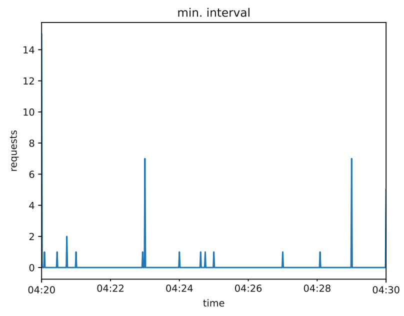


(c) New York 2 km × 2 km with a cloudlet area size of 1 km × 1 km

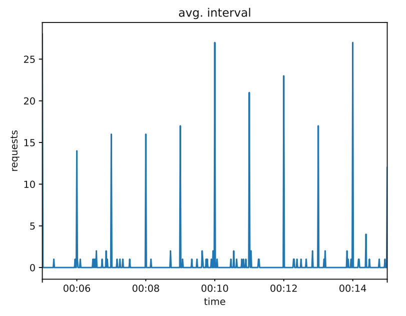


(d) New York 2 km × 2 km with a cloudlet area size of 2 km × 1 km

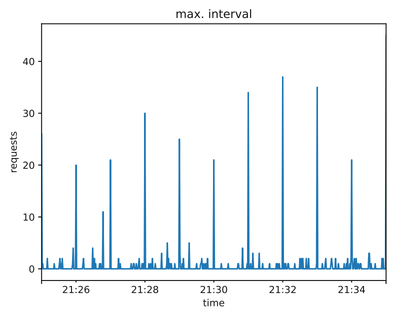
Figure 7.8: Cells extracted from different-sized areas in New York City. The different colors of the data points illustrate the cloudlet areas.



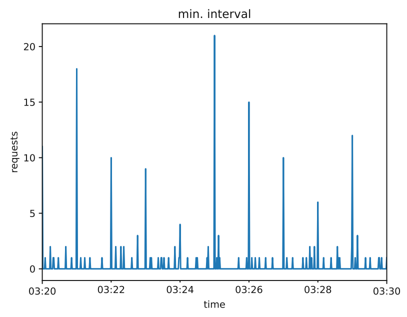
(a) Minimum interval of trip events in a 1 km × 1 km area



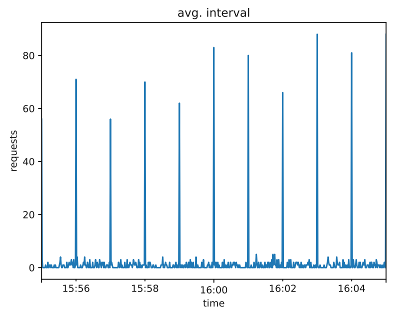
(b) Average interval of trip event in a 1 km × 1 km area



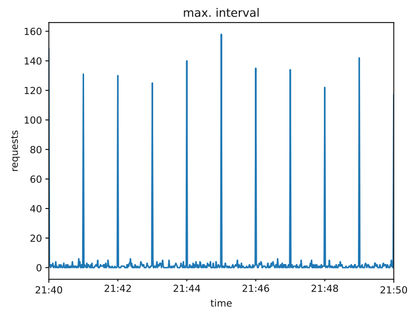
(c) Maximum interval of trip events in a 1 km × 1 km area



(d) Minimum interval of trip events in a 2 km × 2 km area



(e) Average interval of trip event in a 2 km × 2 km area



(f) Maximum interval of trip events in 2 km × 2 km area

Figure 7.9: Trip event intervals in New York City, with an area size of 1 km × 1 km and 2 km × 2 km. The interval time range is 10 minutes.

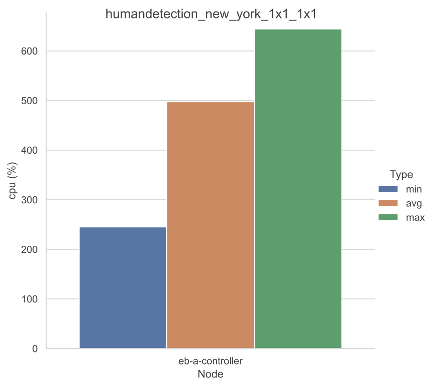
7.4 Simulation and Testbed Evaluation

7.4.1 Experiments evaluation

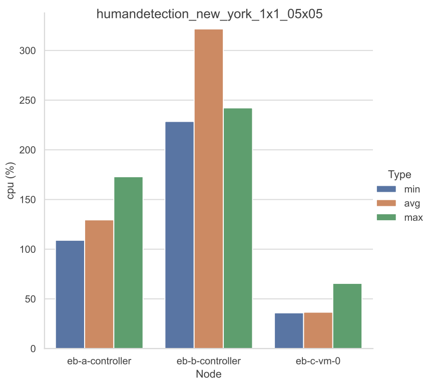
This section shows the results of the profiling and scenario experiments. The experiments deliver the baseline for the comparison between experiment and simulation and give an insight of how the developed functions and workloads perform on the testbed. For this purpose, the function execution time, CPU usage, RAM usage, and network usage are going to be investigated. The results will then be examined to find noticeable abnormalities in the data. The profiling data is then also used for the simulation runs. The profiling delivers a concrete performance metric for each function in a sheltered environment. The function were executed in a fix interval with no other external influence, such that this metrics can be used in the simulation of *faas-sim*.

Profiling

The figures in Figure 7.11 display the results of all executed profiling experiments. It shows that the network usage is consistently similar across all devices. Gun detection and human detection are allocating roughly 390 KB, the object and mask detection around 195 KB. The function execution time varies depending on the device. While gun, object, and human detection have a higher execution (42 ms, 185 ms, and 76ms) on the cloud cluster *eb-c-vm-0* than on the other two devices, the mask detection occurs to be a little bit faster on the *eb-b-controller* (54 ms). When looking at the CPU usage in Figure 7.11b, the cloud cluster uses the least amount of computational power. Human detection requires the most amount of computational power on *eb-a-controller* and *eb-b-controller* (557 % and 544 %). The *eb-b-controller* is using the maximum amount of memory considering the gun, object, and human detection functions. The object detection function is using the maximum amount of RAM (245 MB) on the *eb-b-controller*. The mask detection needs roughly the same memory on each device. In Table 7.7 the whole results are listed as a table.

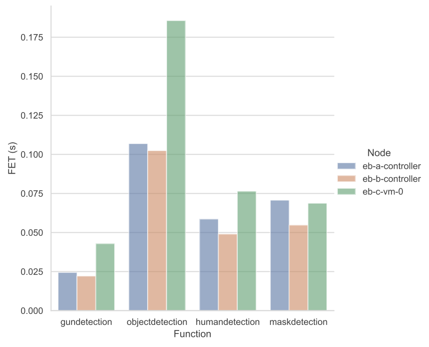


(a) Experiment with 1 cloudlet

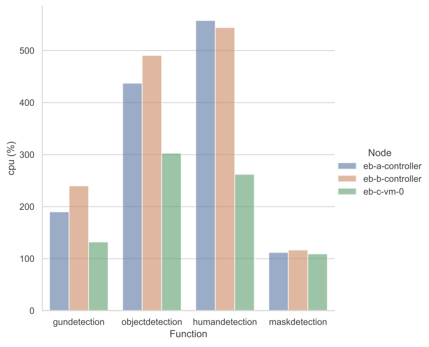


(b) Experiment with 3 cloudlets

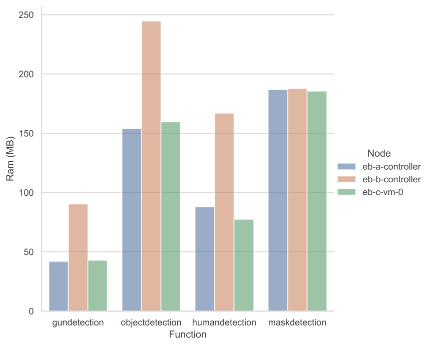
Figure 7.10: CPU consumption in human detection scenario



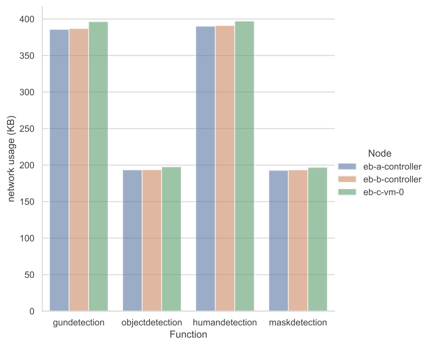
(a) Function Execution Time



(b) CPU usage



(c) RAM usage



(d) Network usage

Figure 7.11: Resource and execution time profiling results.

Scenarios

Because of the 10s interval of the *telemd* setup on the testbed, the scenario results are calculated in a other way than the profiling experiments. The ten-second interval forces us to get three different time stamps to calculate the mean values of every metric. They are the last measurement before the single request, the *telemd* value before the request, and in distinct cases the value in between the start and end timestamp of the request. When looking at the results in the table below, some interesting investigations can be outlined. It shows, that in all scenarios the function execution times in zone C are greatly higher than in the other zones. The most CPU-intensive scenario was the human detection scenario in zone A with a cloudlet size of one square kilometer and the maximum workload size. The needed CPU consumption was around 644 %, which means that six cores were needed in the mean. The thriftiest scenario was the gun detection scenario in zone C, with minimum workload size and a small cloudlet size of only 1 times 0.5 kilometers, where only around 10 % CPU consumption was recorded. It is remarkable, that cloud zone C requires the lowest CPU consumption. It also shows, that when increasing the cloudlet size, the request amount gets greater and in further succession, the CPU utilization increases. E.g., when looking at the human detection scenarios in zone A, with an average workload size, the CPU consumption in a cloudlet area of 1 times 0.5 kilometers was at 321,67 %. In contrast, the mean CPU consumption in a one square kilometer cloudlet was 397,59 %. This phenomenon could be observed in all experiment scenarios and shows clearly, that the CPU consumption is directly connected to the cloudlet size and request amount. Figure 7.10 shows the CPU consumption of the human detection experiments in a single cloudlet area and in three different areas. In [LLR+21] the problem of edge server placement is discussed and solutions were presented. It shows, that in cases where different workload profiles and cloudlet sizes are present, the knowledge in which area how many Edge Computing units should be placed is important for providing the most efficient coverage. This suite can provide support for such problems.

Service	Node	FET (s)	CPU (%)	Network (kB)	RAM (MB)
objectdetection	eb-a-controller	0.106965	436.892487	193.328495	153.941067
objectdetection	eb-b-controller	0.102486	490.657676	193.673717	244.592371
objectdetection	eb-c-vm-0	0.185652	302.506663	197.588980	159.656854
humandetection	eb-a-controller	0.058615	557.584905	390.010495	87.977795
humandetection	eb-b-controller	0.049020	544.140779	391.005737	166.844933
humandetection	eb-c-vm-0	0.076519	261.859020	397.133354	77.331822
maskdetection	eb-a-controller	0.070769	111.692226	192.681798	186.889371
maskdetection	eb-b-controller	0.054737	116.302763	193.364535	187.779072
maskdetection	eb-c-vm-0	0.068802	109.009730	196.817283	185.441870
gundetection	eb-a-controller	0.024452	189.764076	385.866730	41.775906
gundetection	eb-b-controller	0.022127	239.732796	387.007330	90.363576
gundetection	eb-c-vm-0	0.042908	131.736536	396.268414	42.851447

Table 7.7: Profiling results of all functions on every device of the testbed.

7.4.2 Simulation evaluation

The data of the profiling function execution time and resource usage experiments are taken as input for the function resource characterization and parameterized distribution of the function execution time in the *faas-sim* simulation setup. Also, the ping results of each device, shown in Table 6.1, were included in the latency distribution of each part of the simulated testbed scenario. The latency of the cloud cluster was around 75 ms, and the latency of the other two cloudlets and IoT Box cluster was around 21 ms. The request profiles for each scenario and zone were provided as *Pickle* files like in the testbed experiments. The image properties, like size and architecture, were also defined based on the *Docker* images of the different workload functions. Before starting the simulations for the scenarios, the size of the image, that has to be sent in the request, has also to be set. When all of these parameters are defined, the simulation owns the complementary setting of the testbed scenario. In Table 7.10 and Table 7.13 the generated simulation results for the interval of $0.1s$ are shown. The tables show the size of the measured area and cloud(let) area, request pattern type, service name, zone, function execution time, CPU usage, RAM, and round trip time. The mask and gun detection scenarios are a lot less CPU intensive than the human and object detection scenarios. While they only require between 39 % and 52 % in the mean, human and object detection are more CPU intensive and more widely distributed, with mean usages of 83 % to 280 %. The lowest mean function execution time, 0.0186s, and the lowest round trip time, 0.0353s, produce the gun detection scenario in zone B with the average request pattern. Remarkably, the object detection scenario in zone C with the minimum request pattern has the highest round trip and function execution time (0.2184s, 0.1896s) and not the one with the maximum pattern. Zone B needs in the human detection, gun detection, and object detection scenarios more RAM than in the other zones. Only the mask detection scenarios are nearly equally distributed when looking at the RAM usage. The most RAM usage appears in the object detection scenario of zone B with 244.58 MB usage.

Size, Cloudlet Size	Type	Service	Zone	FET (s)	CPU (%)	RAM (mb)	RTT (s)
1 km^2, 1 km^2	max	humandetection	a	0.1176	644.30	90.21	0.1674
1 km^2, 1 km^2	avg	humandetection	a	0.1173	497.59	91.26	0.1674
1 km^2, 1 km^2	min	humandetection	a	0.1166	245.17	81.45	0.1690
1 km^2, 0.5 km^2	max	humandetection	a	0.1187	172.91	89.54	0.1694
1 km^2, 0.5 km^2	max	humandetection	b	0.1243	242.19	87.62	0.1748
1 km^2, 0.5 km^2	max	humandetection	c	0.3238	65.56	74.63	0.4951
1 km^2, 0.5 km^2	avg	humandetection	a	0.1188	129.49	80.25	0.1701
1 km^2, 0.5 km^2	avg	humandetection	b	0.1271	321.67	82.19	0.1773
1 km^2, 0.5 km^2	avg	humandetection	c	0.3556	36.57	70.97	0.5383
1 km^2, 0.5 km^2	min	humandetection	a	0.1162	108.98	69.04	0.1672
1 km^2, 0.5 km^2	min	humandetection	b	0.1274	228.56	70.54	0.1777
1 km^2, 0.5 km^2	min	humandetection	c	0.3568	35.91	59.37	0.5998
1 km^2, 1 km^2	max	gundetection	a	0.0850	138.45	43.06	0.1324
1 km^2, 1 km^2	avg	gundetection	a	0.0849	109.37	42.39	0.1324
1 km^2, 1 km^2	min	gundetection	a	0.0848	53.53	39.71	0.1324
1 km^2, 0.5 km^2	max	gundetection	a	0.0853	37.92	40.81	0.1328
1 km^2, 0.5 km^2	max	gundetection	b	0.0868	67.71	39.06	0.1348
1 km^2, 0.5 km^2	max	gundetection	c	0.2524	25.12	42.88	0.4202
1 km^2, 0.5 km^2	avg	gundetection	a	0.0861	28.48	39.90	0.1338
1 km^2, 0.5 km^2	avg	gundetection	b	0.0874	94.06	39.04	0.1354
1 km^2, 0.5 km^2	avg	gundetection	c	0.2922	13.03	41.55	0.4711
1 km^2, 0.5 km^2	min	gundetection	a	0.0851	23.85	38.07	0.1332
1 km^2, 0.5 km^2	min	gundetection	b	0.0865	66.68	36.71	0.1347
1 km^2, 0.5 km^2	min	gundetection	c	0.3070	10.06	39.59	0.5447

Table 7.8: Experiment results of human and gun detection scenarios. The color gradation allows to distinguish between the minimal, average and maximum request pattern.

Size, Cloudlet Size	Type	Service	Zone	FET (s)	CPU (%)	RAM (mb)	RTT (s)
2 km^2, 1 km^2	max	maskdetection	a	0.1094	195.66	170.37	0.1569
2 km^2, 1 km^2	max	maskdetection	b	0.1025	208.47	181.88	0.1500
2 km^2, 1 km^2	max	maskdetection	c	0.1677	142.95	183.94	0.3212
2 km^2, 1 km^2	avg	maskdetection	a	0.1110	107.20	171.77	0.1583
2 km^2, 1 km^2	avg	maskdetection	b	0.1014	135.93	183.76	0.1491
2 km^2, 1 km^2	avg	maskdetection	c	0.1861	87.58	176.63	0.3410
2 km^2, 1 km^2	min	maskdetection	a	0.1167	37.33	169.22	0.1645
2 km^2, 1 km^2	min	maskdetection	b	0.0993	35.90	181.20	0.1471
2 km^2, 1 km^2	min	maskdetection	c	0.2267	22.56	170.88	0.4002
2 km^2, 2 x 1 km	max	maskdetection	a	0.1097	301.51	173.70	0.1569
2 km^2, 2 x 1 km	max	maskdetection	b	0.1023	374.68	183.46	0.1499
2 km^2, 2 x 1 km	avg	maskdetection	a	0.1117	199.75	170.39	0.1591
2 km^2, 2 x 1 km	avg	maskdetection	b	0.1020	188.65	181.20	0.1495
2 km^2, 2 x 1 km	min	maskdetection	a	0.1118	50.84	173.22	0.1596
2 km^2, 2 x 1 km	min	maskdetection	b	0.0999	50.97	174.73	0.1477
1 km^2, 1 km^2	max	objectdetection	a	0.1577	607.44	164.25	0.2051
1 km^2, 1 km^2	avg	objectdetection	a	0.1658	468.47	160.32	0.2133
1 km^2, 1 km^2	min	objectdetection	a	0.1532	226.97	136.82	0.2010
1 km^2, 0.5 km^2	max	objectdetection	a	0.2097	127.75	151.01	0.2575
1 km^2, 0.5 km^2	max	objectdetection	b	0.1712	223.99	150.96	0.2197
1 km^2, 0.5 km^2	max	objectdetection	c	0.8169	52.99	160.13	1.0169
1 km^2, 0.5 km^2	avg	objectdetection	a	0.1841	106.41	145.06	0.2322
1 km^2, 0.5 km^2	avg	objectdetection	b	0.1681	302.90	149.20	0.2169
1 km^2, 0.5 km^2	avg	objectdetection	c	0.6287	45.78	155.50	0.8240
1 km^2, 0.5 km^2	min	objectdetection	a	0.2211	72.67	112.70	0.2684
1 km^2, 0.5 km^2	min	objectdetection	b	0.1627	230.42	128.19	0.2123
1 km^2, 0.5 km^2	min	objectdetection	c	0.6817	43.48	104.12	0.9335

Table 7.9: Experiment results of the mask and object detection scenarios. The color gradation allows to distinguish between the minimal, average and maximum request pattern.

7.4.3 Comparison of Experiments & Simulation

The CPU calculation method, which was used to calculate the experiment CPU usage, did not apply to the simulation results, because the simulation monitors the usage in a different way than the experiment *telemd* monitor. It handles a customizable monitoring interval (reconcile interval), while the *telemd* only monitors the resource usage every 10 seconds. Therefore, the direct comparison of the values would be not well substantiated, so the results must be examined by its noticeable distinctions and common tendencies. For the purpose of **RQ3**, three different reconcile intervals are applied to investigate what differences appear when setting the monitoring interval to $0.1s$, $1s$, and $10s$. For each of the reconcile interval, an average error value is calculated. For this, we subtract the result of each scenario experiment by the one of the simulation and calculate the average for the interval over all scenarios. In the evaluation repository[15] the calculation is accessible. The results for each scenario are listed in Table 7.8 to Table 7.15.

Size, Cloudlet Size	Type	Service	Zone	FET (s)	CPU (%)	RAM (mb)	RTT (s)
1 km^2, 1 km^2	max	humandetection	a	0.0584	217.89	87.97	0.0765
1 km^2, 1 km^2	avg	humandetection	a	0.0584	229.61	87.96	0.0769
1 km^2, 1 km^2	min	humandetection	a	0.0584	163.73	87.96	0.0790
1 km^2, 0.5 km^2	max	humandetection	a	0.0586	261.81	87.96	0.0745
1 km^2, 0.5 km^2	max	humandetection	b	0.0487	199.19	166.82	0.0642
1 km^2, 0.5 km^2	max	humandetection	c	0.0768	112.46	77.32	0.0946
1 km^2, 0.5 km^2	avg	humandetection	a	0.0585	225.87	87.96	0.0755
1 km^2, 0.5 km^2	avg	humandetection	b	0.0489	204.71	166.82	0.0661
1 km^2, 0.5 km^2	avg	humandetection	c	0.0759	83.10	77.32	0.1005
1 km^2, 0.5 km^2	min	humandetection	a	0.0574	277.99	87.96	0.0754
1 km^2, 0.5 km^2	min	humandetection	b	0.0476	220.77	166.82	0.0632
1 km^2, 0.5 km^2	min	humandetection	c	0.0771	201.86	77.32	0.1036
1 km^2, 1 km^2	max	gundetection	a	0.0245	40.67	41.77	0.0417
1 km^2, 1 km^2	avg	gundetection	a	0.0244	40.26	41.77	0.0421
1 km^2, 1 km^2	min	gundetection	a	0.0246	39.16	41.77	0.0439
1 km^2, 0.5 km^2	max	gundetection	a	0.0242	39.37	41.77	0.0402
1 km^2, 0.5 km^2	max	gundetection	b	0.0191	41.86	90.35	0.0358
1 km^2, 0.5 km^2	max	gundetection	c	0.0424	44.38	42.85	0.0561
1 km^2, 0.5 km^2	avg	gundetection	a	0.0237	43.00	41.77	0.0421
1 km^2, 0.5 km^2	avg	gundetection	b	0.0186	39.60	90.35	0.0353
1 km^2, 0.5 km^2	avg	gundetection	c	0.0419	43.16	42.85	0.0649
1 km^2, 0.5 km^2	min	gundetection	a	0.0234	40.10	41.77	0.0456
1 km^2, 0.5 km^2	min	gundetection	b	0.0198	44.24	90.35	0.0372
1 km^2, 0.5 km^2	min	gundetection	c	0.0402	41.36	42.85	0.0548

Table 7.10: Simulation results of human and gun detection scenarios in a reconcile interval of 0.1s. The color gradation allows to distinguish between the minimal, average and maximum request pattern.

RAM

The simulation and experiment RAM usage have a similar range of values in zone A and C, while zone B needs significantly more RAM in the simulation than in the experiments, except in the mask detection scenarios. E.g. the average object detection scenario allocates only 149.20 MB memory in zone B, while the simulation requires 244.58 MB memory. These results indicate a possible discrepancy in the allocation of memory resources in zone B during the simulation and the testbed runs. The error values for the intervals were −20.71 MB, −20.59 MB, and −19.59 MB, which means that the simulation calculates around 20 MB more RAM on average than was measured in the experiment. The differences between the simulation intervals are minimal and show no conspicuous features.

[15]https://github.com/edgerun/faas-sim-evaluation/blob/feature/pprueller/evaluation

Size, Cloudlet Size	Type	Service	Zone	FET (s)	CPU (%)	RAM (mb)	RTT (s)
1 km^2, 1 km^2	max	humandetection	a	0.0585	79.44	87.87	0.0786
1 km^2, 1 km^2	avg	humandetection	a	0.0585	79.58	87.86	0.0781
1 km^2, 1 km^2	min	humandetection	a	0.0589	73.20	87.85	0.0776
1 km^2, 0.5 km^2	max	humandetection	a	0.0583	75.43	87.86	0.0781
1 km^2, 0.5 km^2	max	humandetection	b	0.0482	55.18	166.59	0.0684
1 km^2, 0.5 km^2	max	humandetection	c	0.0763	32.90	77.22	0.1757
1 km^2, 0.5 km^2	avg	humandetection	a	0.0585	57.69	87.85	0.0784
1 km^2, 0.5 km^2	avg	humandetection	b	0.0483	67.24	166.58	0.0683
1 km^2, 0.5 km^2	avg	humandetection	c	0.0761	31.81	77.21	0.1708
1 km^2, 0.5 km^2	min	humandetection	a	0.0577	64.34	87.85	0.0709
1 km^2, 0.5 km^2	min	humandetection	b	0.0478	54.74	166.57	0.0689
1 km^2, 0.5 km^2	min	humandetection	c	0.0787	32.47	77.21	0.1481
1 km^2, 1 km^2	max	gundetection	a	0.0244	12.55	41.72	0.0441
1 km^2, 1 km^2	avg	gundetection	a	0.0244	12.28	41.72	0.0462
1 km^2, 1 km^2	min	gundetection	a	0.0250	10.96	41.71	0.0466
1 km^2, 0.5 km^2	max	gundetection	a	0.0242	11.37	41.72	0.0450
1 km^2, 0.5 km^2	max	gundetection	b	0.0193	10.54	90.35	0.0407
1 km^2, 0.5 km^2	max	gundetection	c	0.0421	10.32	42.85	0.1396
1 km^2, 0.5 km^2	avg	gundetection	a	0.0245	9.25	41.72	0.0434
1 km^2, 0.5 km^2	avg	gundetection	b	0.0192	12.60	90.35	0.0388
1 km^2, 0.5 km^2	avg	gundetection	c	0.0425	8.50	42.85	0.1359
1 km^2, 0.5 km^2	min	gundetection	a	0.0230	11.64	41.71	0.0371
1 km^2, 0.5 km^2	min	gundetection	b	0.0179	9.38	90.35	0.0383
1 km^2, 0.5 km^2	min	gundetection	c	0.0436	9.19	42.85	0.1442

Table 7.11: Simulation results of human and gun detection scenarios in a reconcile interval of 1s. The color gradation allows to distinguish between the minimal, average and maximum request pattern.

FET

The function execution time of the experiments in zone C is significantly higher than the one in the simulation. Generally, all experiment execution times are eminent higher than in the simulation. The highest execution time in the 0.1s interval simulation is 0.1896s, while the experiments have several execution times above 0.300s and up to 0.8169s. This remarkable difference indicates possible deviations between the real environment (zone C) and the simulated environment. The error values for the intervals were 0.114429s, 0.114405s, and 0.114616s, which means that the simulation calculates approximately 0.114s less on average than was measured in the experiment. The function execution time is almost the same between the three simulation intervals, which indicates that the FET calculation does not directly depend on the reconcile interval.

Size, Cloudlet Size	Type	Service	Zone	FET (s)	CPU (%)	RAM (mb)	RTT (s)
1 km^2, 1 km^2	max	humandetection	a	0.0585	25.70	87.12	0.0792
1 km^2, 1 km^2	avg	humandetection	a	0.0583	20.14	87.07	0.0780
1 km^2, 1 km^2	min	humandetection	a	0.0580	9.24	87.01	0.0815
1 km^2, 0.5 km^2	max	humandetection	a	0.0582	12.76	87.07	0.0790
1 km^2, 0.5 km^2	max	humandetection	b	0.0481	10.22	166.08	0.0691
1 km^2, 0.5 km^2	max	humandetection	c	0.0764	6.99	76.98	0.1786
1 km^2, 0.5 km^2	avg	humandetection	a	0.0584	8.98	87.03	0.0780
1 km^2, 0.5 km^2	avg	humandetection	b	0.0487	12.75	166.04	0.0699
1 km^2, 0.5 km^2	avg	humandetection	c	0.0756	4.57	76.96	0.1774
1 km^2, 0.5 km^2	min	humandetection	a	0.0593	8.81	87.00	0.0772
1 km^2, 0.5 km^2	min	humandetection	b	0.0485	7.97	166.01	0.0685
1 km^2, 0.5 km^2	min	humandetection	c	0.0758	5.29	76.95	0.1625
1 km^2, 1 km^2	max	gundetection	a	0.0245	3.63	41.36	0.0458
1 km^2, 1 km^2	avg	gundetection	a	0.0246	2.83	41.34	0.0446
1 km^2, 1 km^2	min	gundetection	a	0.0241	1.43	41.32	0.0466
1 km^2, 0.5 km^2	max	gundetection	a	0.0245	1.84	41.34	0.0465
1 km^2, 0.5 km^2	max	gundetection	b	0.0187	1.80	89.94	0.0401
1 km^2, 0.5 km^2	max	gundetection	c	0.0432	2.21	42.65	0.1438
1 km^2, 0.5 km^2	avg	gundetection	a	0.0244	1.40	41.32	0.0474
1 km^2, 0.5 km^2	avg	gundetection	b	0.0196	2.25	89.92	0.0401
1 km^2, 0.5 km^2	avg	gundetection	c	0.0431	1.37	42.64	0.1467
1 km^2, 0.5 km^2	min	gundetection	a	0.0246	1.25	41.31	0.0401
1 km^2, 0.5 km^2	min	gundetection	b	0.0167	1.18	89.90	0.0386
1 km^2, 0.5 km^2	min	gundetection	c	0.0411	1.44	42.63	0.1350

Table 7.12: Simulation results of human and gun detection scenarios in a reconcile interval of $10s$. The color gradation allows to distinguish between the minimal, average and maximum request pattern.

RTT

Like the function execution time, also the round trip time is in all cases distinguished higher in the experiment scenarios. The maximum round trip time of the experiments is $1.0169s$, while the $0.1s$ interval simulation only has one with $0.2184s$. The zone C round trip time is higher in all scenarios of simulation and emulation, because of the added latency for this zone, but it shows that the experiments have a significantly higher round trip time in zone C than in the simulation. The error values for the intervals were $0.17883s$, $0.15978s$, and $0.15806s$, which means that the simulation calculates approximately $0.1580s$ to $0.1788s$ less on average than was measured in the experiment. When looking at the $10s$ interval results in Table 7.12 and Table 7.15, it shows that the round trip time is significantly higher in zone C than in the $0.1s$ interval. The other metrics do not show any higher deviations.

Size, Cloudlet Size	Type	Service	Zone	FET (s)	CPU (%)	RAM (mb)	RTT (s)
2 km^2, 2 x 1 km	max	maskdetection	a	0.0675	49.22	186.88	0.0841
2 km^2, 2 x 1 km	max	maskdetection	b	0.0538	43.48	187.78	0.0700
2 km^2, 2 x 1 km	avg	maskdetection	a	0.0675	48.92	186.88	0.0845
2 km^2, 2 x 1 km	avg	maskdetection	b	0.0536	45.95	187.78	0.0705
2 km^2, 2 x 1 km	min	maskdetection	a	0.0677	48.51	186.88	0.0845
2 km^2, 2 x 1 km	min	maskdetection	b	0.0540	52.58	187.78	0.0683
2 km^2, 1 km^2	max	maskdetection	a	0.0674	51.53	186.88	0.0830
2 km^2, 1 km^2	max	maskdetection	b	0.0538	45.67	187.78	0.0700
2 km^2, 1 km^2	max	maskdetection	c	0.0683	47.38	185.44	0.0860
2 km^2, 1 km^2	avg	maskdetection	a	0.0674	47.69	186.88	0.0829
2 km^2, 1 km^2	avg	maskdetection	b	0.0537	44.54	187.78	0.0703
2 km^2, 1 km^2	avg	maskdetection	c	0.0686	49.35	185.44	0.0918
2 km^2, 1 km^2	min	maskdetection	a	0.0679	49.44	186.88	0.0823
2 km^2, 1 km^2	min	maskdetection	b	0.0540	42.59	187.78	0.0706
2 km^2, 1 km^2	min	maskdetection	c	0.0687	46.99	185.44	0.0877
1 km^2, 1 km^2	max	objectdetection	a	0.1055	234.86	153.92	0.1234
1 km^2, 1 km^2	avg	objectdetection	a	0.1062	233.76	153.92	0.1236
1 km^2, 1 km^2	min	objectdetection	a	0.1068	237.65	153.92	0.1251
1 km^2, 0.5 km^2	max	objectdetection	a	0.1065	225.27	153.92	0.1228
1 km^2, 0.5 km^2	max	objectdetection	b	0.1015	245.60	244.58	0.1178
1 km^2, 0.5 km^2	max	objectdetection	c	0.1854	220.49	159.65	0.2095
1 km^2, 0.5 km^2	avg	objectdetection	a	0.1080	236.02	153.92	0.1234
1 km^2, 0.5 km^2	avg	objectdetection	b	0.1021	263.31	244.58	0.1191
1 km^2, 0.5 km^2	avg	objectdetection	c	0.1822	223.55	159.65	0.1989
1 km^2, 0.5 km^2	min	objectdetection	a	0.1019	194.50	153.92	0.1247
1 km^2, 0.5 km^2	min	objectdetection	b	0.1058	259.47	244.57	0.1199
1 km^2, 0.5 km^2	min	objectdetection	c	0.1896	222.08	159.65	0.2184

Table 7.13: Simulation results of object and mask detection scenarios in a reconcile interval of 0.1s. The color gradation allows to distinguish between the minimal, average and maximum request pattern.

CPU

Contrary to the experiments, where an increase in CPU usage is noticeable between the usage patterns, the CPU usage of the simulation scenarios seems to stay at the same level when comparing the results of the min., avg., and max. usage pattern. In some cases, also the minimum or average pattern creates higher mean CPU usage values than the maximum pattern. The error values for the intervals were 32.21%, 116.45%, and 149.15%, which means that in all intervals the simulation calculates less CPU and the higher the reconciliation interval, the less CPU is calculated. In the two intervals, 1s and 10s, the CPU usage metric is significantly lower than in the 0.1s interval. For example, the mask detection scenario of zone B with a minimum request pattern shown in Table 7.15 and Table 7.14 has a CPU utilization of only around 1% and 10%, while the same scenario in the interval of 0.1s has a CPU utilization of 52.58%.

Size, Cloudlet Size	Type	Service	Zone	FET (s)	CPU (%)	RAM (mb)	RTT (s)
2 km^2, 2 x 1 km	max	maskdetection	a	0.0676	20.81	186.86	0.0866
2 km^2, 2 x 1 km	max	maskdetection	b	0.0536	17.04	187.59	0.0731
2 km^2, 2 x 1 km	avg	maskdetection	a	0.0674	18.08	186.85	0.0869
2 km^2, 2 x 1 km	avg	maskdetection	b	0.0537	14.15	187.56	0.0737
2 km^2, 2 x 1 km	min	maskdetection	a	0.0678	13.42	186.85	0.0873
2 km^2, 2 x 1 km	min	maskdetection	b	0.0531	10.84	187.53	0.0722
2 km^2, 1 km^2	max	maskdetection	a	0.0674	19.31	186.85	0.0869
2 km^2, 1 km^2	max	maskdetection	b	0.0534	16.76	187.58	0.0733
2 km^2, 1 km^2	max	maskdetection	c	0.0688	13.47	185.25	0.1581
2 km^2, 1 km^2	avg	maskdetection	a	0.0676	16.53	186.85	0.0868
2 km^2, 1 km^2	avg	maskdetection	b	0.0536	15.28	187.57	0.0727
2 km^2, 1 km^2	avg	maskdetection	c	0.0683	12.16	185.24	0.1599
2 km^2, 1 km^2	min	maskdetection	a	0.0676	13.48	186.85	0.0839
2 km^2, 1 km^2	min	maskdetection	b	0.0541	10.90	187.53	0.0746
2 km^2, 1 km^2	min	maskdetection	c	0.0689	10.40	185.19	0.1646
1 km^2, 1 km^2	max	objectdetection	a	0.1056	101.47	153.73	0.1244
1 km^2, 1 km^2	avg	objectdetection	a	0.1054	100.93	153.72	0.1247
1 km^2, 1 km^2	min	objectdetection	a	0.1067	89.10	153.70	0.1264
1 km^2, 0.5 km^2	max	objectdetection	a	0.1062	93.30	153.72	0.1257
1 km^2, 0.5 km^2	max	objectdetection	b	0.1011	92.72	244.36	0.1202
1 km^2, 0.5 km^2	max	objectdetection	c	0.1875	80.55	159.50	0.2845
1 km^2, 0.5 km^2	avg	objectdetection	a	0.1071	77.30	153.70	0.1264
1 km^2, 0.5 km^2	avg	objectdetection	b	0.1005	107.26	244.34	0.1204
1 km^2, 0.5 km^2	avg	objectdetection	c	0.1874	71.31	159.49	0.2768
1 km^2, 0.5 km^2	min	objectdetection	a	0.1103	85.45	153.70	0.1315
1 km^2, 0.5 km^2	min	objectdetection	b	0.1026	96.99	244.33	0.1220
1 km^2, 0.5 km^2	min	objectdetection	c	0.1789	58.94	159.49	0.2890

Table 7.14: Simulation results of object and mask detection scenarios in a reconcile interval of 1s. The color gradation allows to distinguish between the minimal, average and maximum request pattern.

Size, Cloudlet Size	Type	Service	Zone	FET (s)	CPU (%)	RAM (mb)	RTT (s)
2 km^2, 2 x 1 km	max	maskdetection	a	0.0675	9.49	185.24	0.0876
2 km^2, 2 x 1 km	max	maskdetection	b	0.0533	8.17	186.87	0.0734
2 km^2, 2 x 1 km	avg	maskdetection	a	0.0675	6.20	185.06	0.0865
2 km^2, 2 x 1 km	avg	maskdetection	b	0.0540	5.52	186.77	0.0741
2 km^2, 2 x 1 km	min	maskdetection	a	0.0672	2.27	184.77	0.0861
2 km^2, 2 x 1 km	min	maskdetection	b	0.0526	1.54	186.61	0.0732
2 km^2, 1 km^2	max	maskdetection	a	0.0674	6.01	185.20	0.0866
2 km^2, 1 km^2	max	maskdetection	b	0.0535	5.14	186.85	0.0732
2 km^2, 1 km^2	max	maskdetection	c	0.0677	4.51	184.52	0.1655
2 km^2, 1 km^2	avg	maskdetection	a	0.0676	4.55	185.11	0.0877
2 km^2, 1 km^2	avg	maskdetection	b	0.0533	3.08	186.80	0.0734
2 km^2, 1 km^2	avg	maskdetection	c	0.0682	3.83	184.47	0.1668
2 km^2, 1 km^2	min	maskdetection	a	0.0670	1.83	184.74	0.0854
2 km^2, 1 km^2	min	maskdetection	b	0.0535	1.44	186.59	0.0717
2 km^2, 1 km^2	min	maskdetection	c	0.0682	1.62	184.26	0.1549
1 km^2, 1 km^2	max	objectdetection	a	0.1061	33.29	152.26	0.1262
1 km^2, 1 km^2	avg	objectdetection	a	0.1059	27.77	152.16	0.1261
1 km^2, 1 km^2	min	objectdetection	a	0.1068	14.72	151.99	0.1276
1 km^2, 0.5 km^2	max	objectdetection	a	0.1046	16.20	152.16	0.1225
1 km^2, 0.5 km^2	max	objectdetection	b	0.1021	19.97	242.17	0.1227
1 km^2, 0.5 km^2	max	objectdetection	c	0.1858	20.04	158.07	0.2912
1 km^2, 0.5 km^2	avg	objectdetection	a	0.1045	11.66	152.05	0.1243
1 km^2, 0.5 km^2	avg	objectdetection	b	0.1011	23.07	242.01	0.1211
1 km^2, 0.5 km^2	avg	objectdetection	c	0.1831	14.38	157.97	0.2895
1 km^2, 0.5 km^2	min	objectdetection	a	0.1055	12.30	151.99	0.1227
1 km^2, 0.5 km^2	min	objectdetection	b	0.0979	12.80	241.93	0.1209
1 km^2, 0.5 km^2	min	objectdetection	c	0.1926	15.53	157.92	0.3069

Table 7.15: Simulation results of object and mask detection scenarios in a reconcile interval of $10s$. The color gradation allows to distinguish between the minimal, average and maximum request pattern.

CHAPTER 8

Conclusion

Edge Intelligence aims to provide AI applications between the edge and the cloud. This is enabled by the targeted, timely provision of application logic and data sources close to the network participants. Applications that require a high level of cloud offloading, e.g. in the areas of Smart City, Smart Home, and Collaborative Edge, benefit from Edge Intelligence. Moving these from the cloud to the edge demonstrably reduces latency between the network participants and the requested edge devices. The high complexity of managing and deploying applications in edge environments requires a performative provision solution. The Function As A Service paradigm autonomously manages application deployments, thus, is a promising solution for managing EI applications. However, appropriate FaaS platforms for EI have yet to be developed and are challenging to evaluate in real-world environments. Therefore, simulations and testbeds are used to test solutions. A benchmark suite that can be used for simulation and testbeds is therefore beneficial, but only a few works address this. In this work, a method is used that first analyzes the performance of the desired application on a testbed and provides profiling and individual scenario data for the simulation, whereby scenarios, such as a Smart City setup, can be reproduced in the simulation and on the testbed. Benchmarking applications in edge environments prove to be complex and depends on many factors, such as the resource consumption of each serverless function, the specific implementation of these functions, the available hardware edge devices, the placement of these devices in the network, and the used scheduling strategy. This thesis provides an overview of various existing benchmarking and simulation tools, all of which have different approaches in terms of measurement techniques and metrics. This large number of gathered simulation frameworks and benchmarking tools, and their different areas of applications, show a high level of interest in functioning and capable solutions in this field.

To this end, we introduce a benchmark suite of AI-based workloads, which are used for benchmarking and profiling applications on a real-world testbed and in the simulation.

We extend the existing *faas-sim* simulation framework by an approach to automate the benchmark process by using a suite of AI-based workloads, realistic request patterns, an inference pipeline, and geo-distributed topologies. We used two open-source datasets, that can be easily replaced by custom datasets which enable new use cases, similar to the *NYC Taxi* and the *OpenCellid* dataset. It employs the applications, benchmarking tools, and metrics that are already used in the research area of Edge Computing and Artificial Intelligence. Trace-based data is used in the *faas-sim* framework, where the simulation creates a compact overview of four metrics (RAM, CPU, FET, RTT), regarding the different examined scenarios. We present various scenarios that are intended to reflect reality and cover all areas such as the minimum, average, and maximum utilization of edge devices. The results of the simulation and real-world experiments are compared and allow conclusions to be drawn as to whether the simulation delivers realistic results in such scenarios. The evaluation results are then also available for further design decisions that affect server placement in the current environment, changes to the application logic and scheduling strategy, or the improvement of individual network device computing resources.

A contribution of this work is the developed Taxi Driver Safety App inference pipeline in Section 5.4.2, which allows several serverless functions to be called one after the other or in parallel and introduces a way more complex scenario in Edge Computing. Also, the presented technique of request pattern generation in Section 5.3 is a contribution and a more realistic way of workload development. The merging of two open-source datasets is a standalone method in this generation process.

With regard to **RQ1**, the benchmarking tools and frameworks, evaluated in Table 7.5 and Table 7.4, show how diverse the context of benchmarking, simulation, and emulation can be. It enables an extensive baseline to build upon. The investigated benchmark suites offer a wide spectrum of metrics, from standard metrics like latency, CPU usage, and memory usage, to much more complex metrics like model accuracy, infrastructure costs, or power consumption, and have already working benchmarking tools in the area of model inference, smart home or Human-in-the-Loop applications. It shows that AI and Edge Computing are already an integral part of benchmarking, but an overall solution for EI benchmarking still needs to be improved. Our benchmark suite covers the most important aspects of these findings and shows how an approach for an overall solution can look conceptually and also in part practically.

For answering **RQ2**, we introduced methods in Section 4.2 to establish realistic infrastructure topologies, request patterns, and AI-related workloads. Based on these methods we build a suite[1], which is practicable, extendable, and creates realistic usage patterns in a semi-automatic way by using a bundle of tools. The developed serverless functions are representative of the area of AI and AR applications and can be seen as examples and a baseline for further work. They include function-specific metrics of model load, preprocess, prediction, and post-process time, which are available during

[1]https://github.com/pruellerpaul/benchmark_suite

testbed experiments. The functions are also used in the developed inference pipeline, which is also part of the suite but is not yet implemented in the simulation process. This inference pipeline example, and the developed benchmarking approach of this thesis, allow measuring more complex serverless functions and enable the benchmarking of serverless functions, distributed over multiple nodes in an edge environment. The developed suite offers a connection point for such evaluations and can be used in simulation and real-world experiments. The profiling results in Table 7.7 can be used as a reference for further optimization. Like the four investigated applications, all other serverless-based functions can be easily profiled with our benchmarking method.

With regard to **RQ3**, we can state some main differences between testbed experiments and the simulation. Also, some decisive differences can be observed when evaluating the different simulation intervals. The configured devices in the simulation do not always behave the same as in the testbed experiments, this can be derived from a different topology and device configuration towards the simulation and real-world devices. The simulation does not exactly replicate the experiment metrics, but the differences between the functions and zones are visible. The average error values of RTT, FET and CPU all show a deviation in the sense that the simulation calculates less. Only regarding the RAM the simulation calculates more. When looking at the CPU error values 32.21%, 116.45%, and 149.15%, we have a much greater differ than in then other metrics. The contrasting function execution time and round trip time measured for all scenarios in zone C strongly suggest that further simulation adjustments are required. The simulations with the reconcile interval of $10s$ and $1s$ can be used to analyze usage trends, but it is very inaccurate in terms of CPU and RTT measurement. These differ by far from the experiment results. It is noticeable that the CPU usage suffers enormously from a too high reconcile interval. The RTT in the cloud zone C also differs extremely between the narrow interval of $0.1s$ and the other ones. It can be stated that the CPU simulation results with lower intervals approach those of the experiments. Therefore, decreasing the interval is an important aspect for generating realistic results in *faas-sim*. It is relatively easy to test different configurations and topologies. The simulation provides a good basis for this. The frameworks offer more metrics like network usage, or also node utilization metrics, such that the fields of benchmarking are also extendable in this direction. This suite has a lot of scope and possibilities for expansion and improvement, but already offers a very simple way to benchmark AI-based applications in local environments with Edge Computing units.

8.1 Future work & limitations

The procedure to create all necessary parts for the benchmarking is slightly inflexible yet, therefore, an automated pipeline that creates the pattern and topologies and runs experiments and simulations one after the other would be much more efficient. Also, the collecting of experiment metric data must be improved, by decreasing the *telemd* measurement interval, such that the comparison of the results can be much more precise.

In this thesis, only four function utilization parameters, namely the function execution time, round trip time, RAM, and CPU usage were measured. The investigated metrics are limited in *faas-sim* and the used testbed, therefore a wider spectrum of metrics, like e.g. the power consumption, cold start latency, pricing calculations, or infrastructure costs, would create a more consummate suite of benchmarks. Some of the presented suites of the related work in Section 7.2 already implement these metrics, so an adaption would be possible. The function-specific metrics, like the model loading time, can be used in the future to simulate the QoS metrics. It would be progressive to enable the customization of parameters such as the latest, common, or outdated technology of edge devices and network components by an interface like in Section 7.1.1. Also, the possibility of working with such testbeds and hardware is restricted, therefore publicly accessible and affordable infrastructure (IaaS) would improve the value of the benefit for the suite. The used dataset in this thesis does not provide the exact distance between the event and the cell. Hence, if a dataset can be found that contains this data, the reduction of network speed based on the distance between the event and cell would be a great possibility to increase accuracy. The used *NYC Taxi Dataset* does not include the exact timestamp of the pickup event, so many events were logged at the exact minute, without including the exact seconds. This could distort the results and should be fixed by using a dataset with more exact timestamps. Increasing the amount of distributed event datasets, that can be used in the framework, would enable a wider perspective and could allow a more effective optimization of the framework. An enormous relief would be to implement a feature for this suite, where placement strategies are compiled by searching for the optimal edge and cloud hardware location by considering the key indicators like the minimal round trip time, CPU usage, and function execution time. An important feature would be the implementation of the inference pipeline in the simulation. The more complex the interactions between edge functions and various devices are, the more important the evaluation of benchmarks and the placement of edge devices and servers is.

List of Figures

List of Tables

List of Algorithms

Bibliography

[BBS$^+$20] Wesley Brewer, Greg Behm, Alan Scheinine, Ben Parsons, Wesley Emeneker, and Robert P. Trevino. ibench: a distributed inference simulation and benchmark suite. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, Sep. 2020.

[BHQT22] L. Baresi, D. Hu, G. Quattrocchi, and L. Terracciano. Neptune: Network- and gpu-aware management of serverless functions at the edge. In *2022 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 144–155, Los Alamitos, CA, USA, may 2022. IEEE Computer Society.

[BJCG21] Stephan Patrick Baller, Anshul Jindal, Mohak Chadha, and Michael Gerndt. Deepedgebench: Benchmarking deep neural networks on edge devices. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 20–30, Oct 2021.

[BZKH20] Tristan BRAUD, Pengyuan ZHOU, Jussi KANGASHARJU, and Pan HUI. Multipath computation offloading for mobile augmented reality. In *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–10, March 2020.

[CDPLR19] Maurantonio Caprolu, Roberto Di Pietro, Flavio Lombardi, and Simone Raponi. Edge computing perspectives: Architectures, technologies, and open security issues. In *2019 IEEE International Conference on Edge Computing (EDGE)*, pages 116–123, July 2019.

[CMT16] Maria Carla Calzarossa, Luisa Massari, and Daniele Tessera. Workload characterization: A survey revisited. *ACM Comput. Surv.*, 48(3), feb 2016.

[CNRB13] Rodrigo Neves Calheiros, Marco Aurélio Stelmar Netto, César A. F. De Rose, and Rajkumar Buyya. Emusim: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications. *Software: Practice and Experience*, 43, 2013.

[CRB+11] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, jan 2011.

[DPW18] Anirban Das, Stacy Patterson, and Mike Wittie. Edgebench: Benchmarking edge computing platforms. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 175–180, Dec 2018.

[DZF+20] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y. Zomaya. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal*, 7(8):7457–7469, 2020.

[FKK17] Fairouz Fakhfakh, Hatem Hadj Kacem, and Ahmed Hadj Kacem. Simulation tools for cloud computing: A survey and comparative study. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, pages 221–226, 2017.

[Gee] GeeksforGeeks. Gun detection using python-opencv. `https://www.geeksforgeeks.org/gun-detection-using-python-opencv/`. Accessed: 2022-07-11.

[GVGB17] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.

[KL19] Jeongchul Kim and Kyungyong Lee. Functionbench: A suite of workloads for serverless cloud function service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 502–504, July 2019.

[Lab] Unwired Labs. Opencellid database. `http://wiki.opencellid.org/wiki/Menu_map_view#database:~:text=https%3A//opencellid.org/-,Columns%20present%20in%20database%3A,-Parameter`. Accessed: 2022-07-12.

[LLR+21] Tero Lähderanta, Teemu Leppänen, Leena Ruha, Lauri Lovén, Erkki Harjula, Mika Ylianttila, Jukka Riekki, and Mikko J. Sillanpää. Edge computing server placement with capacitated location allocation. *Journal of Parallel and Distributed Computing*, 153:130–149, 2021.

[LLYC19] Chien-I Lee, Meng-Yao Lin, Chia-Lin Yang, and Yen-Kuang Chen. Iotbench: A benchmark suite for intelligent internet of things edge devices. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 170–174, Sep. 2019.

[LSHG16] Hongxing Li, Guochu Shou, Yihong Hu, and Zhigang Guo. Mobile edge computing: Progress and challenges. In *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 83–84, March 2016.

[MFKS20] Pascal Maissen, Pascal Felber, Peter Kropf, and Valerio Schiavoni. Faasdom: A benchmark suite for serverless computing. In *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems*, DEBS '20, page 73–84, New York, NY, USA, 2020. Association for Computing Machinery.

[MGG+17] Ruben Mayer, Leon Graser, Harshit Gupta, Enrique Saurez, and Umakishore Ramachandran. Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures. In *2017 IEEE Fog World Congress (FWC)*, pages 1–6, Oct 2017.

[MPGB21] Redowan Mahmud, Samodha Pallewatta, Mohammad Goudarzi, and Rajkumar Buyya. Ifogsim2: An extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments. *arXiv preprint arXiv:2109.05636*, 2021.

[OMnWSG19] Manuel Osvaldo J. Olguín Muñoz, Junjue Wang, Mahadev Satyanarayanan, and James Gross. Edgedroid: An experimental approach to benchmarking human-in-the-loop applications. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, HotMobile '19, page 93–98, New York, NY, USA, 2019. Association for Computing Machinery.

[PVCM20] David Perez Abreu, Karima Velasquez, Marilia Curado, and Edmundo Monteiro. A comparative analysis of simulators for the cloud to fog continuum. *Simulation Modelling Practice and Theory*, 101:102029, 2020. Modeling and Simulation of Fog Computing.

[RD19] Thomas Rausch and Schahram Dustdar. Edge intelligence: The convergence of humans, things, and ai. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pages 86–96, June 2019.

[RD21] Philipp Raith and Schahram Dustdar. Edge intelligence as a service. In *2021 IEEE International Conference on Services Computing (SCC)*, pages 252–262, 2021.

[RND23] Philipp Raith, Stefan Nastic, and Schahram Dustdar. Serverless edge computing—where we are and what lies ahead. *IEEE Internet Computing*, 27(3):50–64, 2023.

[RRD21] Thomas Rausch, Alexander Rashed, and Schahram Dustdar. Optimized container scheduling for data-intensive serverless edge computing. *Future Generation Computer Systems*, 114:259–271, 2021.

[RRFD23] Philipp Raith, Thomas Rausch, Alireza Furutanpey, and Schahram Dustdar. faas-sim: A trace-driven simulation framework for serverless edge computing platforms. *Software: Practice and Experience*, 53(12):2327–2361, 2023.

[RRP+22] Philipp Raith, Thomas Rausch, Paul Prüller, Alireza Furutanpey, and Schahram Dustdar. An end-to-end framework for benchmarking edge-cloud cluster management techniques. In *2022 IEEE International Conference on Cloud Engineering (IC2E)*, pages 22–28, Sep. 2022.

[SAA21] Banele G. Simelane, Isaiah O. Adebayo, and Matthew O. Adigun. Evaluating the effect of mobile applications on cloudlet placement. In *2021 3rd International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*, pages 1–5, Nov 2021.

[SBW19] Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff. Architectural implications of function-as-a-service computing. In *Architectural Implications of Function-as-a-Service Computing*, 10 2019.

[SCYE17] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, Dec 2017.

[SCZ+16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016.

[SFG+20] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 205–218. USENIX Association, July 2020.

[SOE17] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. Edgecloudsim: An environment for performance evaluation of edge computing systems. In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 39–44, May 2017.

[SS18] Mohit Sewak and Sachchidanand Singh. Winning in the era of serverless computing and function as a service. In *2018 3rd International Conference for Convergence in Technology (I2CT)*, pages 1–5, April 2018.

[YQZ+15] Jie Yang, Yuanyuan Qiao, Xinyu Zhang, Haiyang He, Fang Liu, and Gang Cheng. Characterizing user behavior in mobile internet. *IEEE Transactions on Emerging Topics in Computing*, 3(1):95–106, March 2015.

[ZCL+19] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, Aug 2019.

[ZCS19] Yukun Zeng, Mengyuan Chao, and Radu Stoleru. Emuedge: A hybrid emulator for reproducible and realistic edge computing experiments. In *2019 IEEE International Conference on Fog Computing (ICFC)*, pages 153–164, June 2019.